SOLUTIONS MANUAL

DIGITAL DESIGN

WITH AN INTRODUCTION TO THE VERILOG HDL
Fifth Edition

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

MICHAEL D. CILETTI
Professor Emeritus

University of Colorado, Colorado Springs

rev 02/14/2012

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

CHAPTER 1

1.1

Base-10:	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Octal:	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-12	14	15	16	17	18	19	1A	1B	20	21	22	23	24	25	26	27	28

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

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

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

$(435)_8 = 4 * 8^2 + 3 * 8^1 + 5 * 8^0 = 285_{10}$

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

1.4 16-bit binary: 1111_1111_1111_1111
Decimal equivalent: $2^{16} -1 = 65{,}535_{10}$
Hexadecimal equivalent: $FFFF_{16}$

1.5 Let b = base

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

(b) 54/4 = (5*b + 4)/4 = b + 3, so 5 * b = 52 – 4, and b = 8

(c) (2 *b + 4) + (b + 7) = 4b, so b = 11

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

Therefore: 6 + 3 = b + 1m, 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)
Answer: $1111_1010_2 = FA_{16}$

(b) Results of repeated division by 16:

431_{10} = 26(15); 1(10) (Faster)
Answer: FA = 1111_1010

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

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

(c) $26.24_8 = 2 * 8 + 6 + 2/8 + 4/64 = 22.3125$

(d) $DADA.B_{16} = 14*16^3 + 10*16^2 + 14*16 + 10 + 11/16 = 60{,}138.6875$

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

(e) $1010.1101_2 = 8 + 2 + .5 + .25 + .0625 = 10.8125$

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

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

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

1.11

```
      1011.11
101 | 111011.0000
      101
      01001
       101
        1001
         101
          1000
           101
            0110
```

The quotient is carried to two decimal places, giving 1011.11
Checking: $111011_2 / 101_2 = 59_{10} / 5_{10} \cong 1011.11_2 = 58.75_{10}$

1.12 **(a)** 10000 and 110111

```
  1011          1011
 +101          x101
 10000 = 16₁₀   1011
               1011
               110111 = 55₁₀
```

(b) 62_h and 958_h

```
2E_h   0010_1110          2E_h
+34_h  0011_0100         x34_h
62_h   0110_0010 = 98₁₀   B^38
                         8^2A
                         9 5 8_h = 2392₁₀
```

1.13 **(a)** Convert 27.315 to binary:

	Integer Quotient		Remainder	Coefficient
27/2 =	13	+	½	$a_0 = 1$
13/2	6	+	½	$a_1 = 1$
6/2	3	+	0	$a_2 = 0$
3/2	1	+	½	$a_3 = 1$
½	0	+	½	$a_4 = 1$

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

$27_{10} = 11011_2$

		Integer		Fraction	Coefficient
.315 x 2	=	0	+	.630	$a_{-1} = 0$
.630 x 2	=	1	+	.26	$a_{-2} = 1$
.26 x 2	=	0	+	.52	$a_{-3} = 0$
.52 x 2	=	1	+	.04	$a_{-4} = 1$

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

$27.315 \cong 11011.0101_2$

(b) 2/3 ≅ .6666666667

		Integer		Fraction	Coefficient
.6666_6666_67 x 2	=	1	+	.3333_3333_34	$a_{-1} = 1$
.3333333334 x 2	=	0	+	.6666666668	$a_{-2} = 0$
.6666666668 x 2	=	1	+	.3333333336	$a_{-3} = 1$
.3333333336 x 2	=	0	+	.6666666672	$a_{-4} = 0$
.6666666672 x 2	=	1	+	.3333333344	$a_{-5} = 1$
.3333333344 x 2	=	0	+	.6666666688	$a_{-6} = 0$
.6666666688 x 2	=	1	+	.3333333376	$a_{-7} = 1$
.3333333376 x 2	=	0	+	.6666666752	$a_{-8} = 0$

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

$.101010102 = .1010_1010_2 = .AA_{16} = 10/16 + 10/256 = .6641_{10}$ (Same as (b)).

1.14

(a)	0001_0000	**(b)**	0000_0000	**(c)**	1101_1010
1s comp:	1110_1111	1s comp:	1111_1111	1s comp:	0010_0101
2s comp:	1111_0000	2s comp:	0000_0000	2s comp:	0010_0110
(d)	1010_1010	**(e)**	1000_0101	**(f)**	1111_1111
1s comp:	0101_0101	1s comp:	0111_1010	1s comp:	0000_0000
2s comp:	0101_0110	2s comp:	0111_1011	2s comp:	0000_0001

`

1.15

(a)	25,478,036	**(b)**	63,325,600
9s comp:	74,521,963	9s comp:	36,674,399
10s comp:	74,521,964	10s comp:	36,674,400
(c)	25,000,000	**(d)**	00000000
9s comp:	74,999,999	9s comp:	99999999
10s comp:	75,000,000	10s comp:	100000000

1.16

	C3DF	C3DF:	1100_0011_1101_1111
15s comp:	3C20	1s comp:	0011_1100_0010_0000
16s comp:	3C21	2s comp:	0011_1100_0010_0001 = 3C21

1.17 **(a)** 2,579 → 02,579 →97,420 (9s comp) → 97,421 (10s comp)
4637 – 2,579 = 2,579 + 97,421 = 2058_{10}

(b) 1800 → 01800 → 98199 (9s comp) → 98200 (10 comp)
125 – 1800 = 00125 + 98200 = 98325 (negative)
Magnitude: 1675
Result: 125 – 1800 = 1675

All rights reserved.

(c) 4,361 → 04361 → 95638 (9s comp) → 95639 (10s comp)
2043 – 4361 = 02043 + 95639 = 97682 (Negative)
Magnitude: 2318
Result: 2043 – 6152 = -2318

(d) 745 → 00745 → 99254 (9s comp) → 99255 (10s comp)
1631 -745 = 01631 + 99255 = 0886 (Positive)
Result: 1631 – 745 = 886

1.18 Note: Consider sign extension with 2s complement arithmetic.

(a)
0_10010
1s comp: 1_01101
2s comp: 1_01110
0_10011
Diff: 0_00001 (Positive)
Check:19-18 = +1

(b)
0_100110
1s comp: 1_011001 with sign extension
2s comp: 1_011010
0_100010
1_111100 sign bit indicates that the result is negative
0_000011 1s complement
0_000100 2s complement
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)
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 is positive
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

All rights reserved.

(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 = 009102
Result: (+9742) + (-641) = 9102

(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

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)** **(b)**

6 3 1 1	Decimal	6 4 2 1	Decimal
0 0 0 0	0	0 0 0 0	0
0 0 0 1	1	0 0 0 1	1
0 0 1 0	2	0 0 1 0	2
0 1 0 0	3	0 0 1 1	3
0 1 1 0	4 (*or* 0101)	0 1 0 0	4
0 1 1 1	5	0 1 0 1	5
1 0 0 0	6	1 0 0 0	6 (*or* 0110)
1 0 1 0	7 (*or* 1001)	1 0 0 1	7
1 0 1 1	8	1 0 1 0	8
1 1 0 0	9	1 0 1 1	9

1.25

(a)	$6{,}248_{10}$	BCD:	0110_0010_0100_1000
(b)		Excess-3:	1001_0101_0111_1011
(c)		2421:	0110_0010_0100_1110
(d)		6311:	1000_0010_0110_1011

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

1.26	6,248	9s Comp:	3,751
		2421 code:	0011_0111_0101_0001
		1s comp c:	1001_1101_1011_0001 (2421 code alternative #1)
		$6{,}248_{2421}$	0110_0010_0100_1110 (2421 code alternative #2)
		1s comp c	1001_1101_1011_0001 Match

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

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

1.31 62 + 32 = 94 printing characters

1.32 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				

1.35 **(a)**

1.36

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

CHAPTER 2

2.1 **(a)**

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

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

(b)

x y z	$x + yz$	$(x + y)$	$(x + z)$	$(x + y)(x + z)$
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) $(A + B)'(A' + B')' = (A'B')(A\ B) = (A'B')(BA) = A'(B'B)A = 0$

(e) $(a + b + c')(a'b' + c) = aa'b' + ac + ba'b' + bc + c'a'b' + c'c = ac + bc + a'b'c'$

(f) $a'bc + abc' + abc + a'bc' = a'b(c + c') + ab(c + c') = a'b + ab = (a' + a)b = b$

2.3 **(a)** $ABC + A'B + ABC' = AB + A'B = B$

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

(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) $(BC' + A'D)(AB' + CD') = BC'AB' + BC'CD' + A'DAB' + A'DCD' = 0$

(f) $(a' + c')(a + b' + c') = a'a + a'b' + a'c' + c'a + c'b' + c'c' = a'b' + a'c' + ac' + b'c' = c' + b'(a' + c')$
$= c' + b'c' + a'b' = c' + a'b'$

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

(b) $(x'y' + z)' + z + xy + wz = (x'y')'z' + z + xy + wz = [(x + y)z' + z] + xy + wz =$
$= (z + z')(z + x + y) + xy + wz = z + wz + x + xy + y = z(1 + w) + x(1 + y) + y = x + y + z$

(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)

All rights reserved.

(d)

(e)

(f)

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

2.6 **(a)**

(b)

(c)

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

(d)

(e)

(f)

2.7 **(a)**

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

(b)

(c)

(d)

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

(e)

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

$FF' = wx(w' + x')(y' + z') + yz(w' + x')(y' + z') = 0$
$F + F' = wx + yz + (wx + yz)' = A + A' = 1$ 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) $F1\ F2 = \Sigma\, m_i\, \Sigma m_j$ 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) $F(a, b, c) = \Sigma(0, 2, 3, 7)$

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

x y z	F
0 0 0	0
0 0 1	1
0 1 0	0
0 1 1	0
1 0 0	1
1 0 1	1
1 1 0	1
1 1 1	1

$F = bc + a'c'$

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

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

(a) *A AND B = 1010_0000*
(b) *A OR B = 1011_1101*
(c) *A XOR B = 0001_1101*
(d) *NOT A = 0100_1110*
(e) *NOT B = 0101_0011*

2.13 **(a)**

(b)

(c)

(d)

(e)

(f)

All rights reserved.

2.14 **(a)**

(b)

(c)

(d)

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

(e)

F = xy + x'y' + y'z

= (x′ + y′)' + (x + y)' + (y + z′)'

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

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

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

$A'B'$ $A'C'$

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

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

AC' AC

BC

$T_2 = AC' + BC + AC = A + BC$

2.16 **(a)** $F(A, B, C) = A'B'C' + A'B'C + A'BC' + A'BC + AB'C' + AB'C + ABC' + ABC$
$= A'(B'C' + B'C + BC' + BC) + A((B'C' + B'C + BC' + BC)$
$= (A' + A)(B'C' + B'C + BC' + BC) = B'C' + B'C + BC' + BC$
$= B'(C' + C) + B(C' + C) = B' + B = 1$

(b) $F(x_1, x_2, x_3, \ldots, x_n) = \Sigma m_i$ has $2^n/2$ minterms with x_1 and $2^n/2$ 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 and 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 $F = x_nG + x'_nG$ with $G = 1$. So $F = (x_n + x'_n)G = 1$.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

2.17 **(a)** $F = (b + cd)(c + bd)$ $bc + bd + cd + bcd = \Sigma(3, 5, 6, 7, 11, 14, 15)$

$F' = \Sigma(0, 1, 2, 4, 8, 9, 10, 12, 13)$

$F = \Pi(0, 1, 2, 4, 8, 9, 10, 12, 13)$

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	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) $(c' + d)(b + c')$ $= bc' + c' + bd + c'd = (c' + bd)$

$= \Sigma\,(0, 1, 4, 5, 7, 8, 12, 13, 15)$

$F = \Pi\,(2, 3, 6, 9, 10, 11, 14)$

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

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

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

$F = \Pi\,(02, 3, 6, 7, 8, 12, 13, 15)$

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

2.18 (a)

wx y z	F
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

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

$F = \Sigma(1, 5, 6, 7, 9, 10\ 11, 13, 14, 15)$

(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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

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

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

(b) $F(x, y, z) = \Pi(3, 5, 7)$
$F' = \Sigma(3, 5, 7)$

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

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

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)** $B'C + AB + ACD$

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

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

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

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

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)** $x \mid y = xy' \neq y \mid x = x'y$ Not commutative

$(x \mid y) \mid z = xy'z' \neq x \mid (y \mid z) = x(yz')' = xy' + xz$ Not associative

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

(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 $f_1 = a'b'c' + a'bc' + a'bc + ab'c' + abc = a'c' + bc + a'bc' + ab'c'$

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

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	**1 0001**	1
0 0010	0	1 0010	0
0 0011	0	**1 0011**	1
0 0100	0	1 0100	0
0 0101	0	**1 0101**	1
0 0110	0	1 0110	0
0 0111	0	**1 0111**	1
	0		0
0 1000	0	1 1000	0
0 1001	0	**1 1001**	1
0 1010	0	1 1010	0
0 1011	0	**1 1011**	1
0 1100	0	1 1100	0
0 1101	0	**1 1101**	1
0 1110	0	1 1110	0
0 1111	0	1 1111	0

(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	
011001 = 25	010101 = 21	010011 = 19	
011010 = 26	010110 = 22	010110 = 22	
011011 = 27	010111 = 23	010111 = 23	
011100 = 28	011100 = 28	011010 = 26	
011101 = 29	011101 = 29	011001 = 27	
011110 = 30	011110 = 30	011110 = 30	
011111 = 31	011111 = 31	011111 = 31	

-b' c--f	-b' -d-f	-b' --ef
001001 = 9	001001 = 9	000011 = 3
001011 = 11	001011 = 11	000111 = 7
001101 = 13	001101 = 13	001011 = 11
001111 = 15	001111 = 15	001111 = 15
101001 = 41	101001 = 41	100011 = 35
101011 = 43	101011 = 43	100111 = 39
101101 = 45	101101 = 45	101011 = 51
101111 = 47	101111 = 47	101111 = 55

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

$y_1 = \Sigma$ (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)

ab cdef	y_1 y_2	*ab cdef*	y_1 y_2	*ab cdef*	y_1 y_2	*ab cdef*	y_1 y_2
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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

Chapter 3

3.1

$F = xy' + x'z'$

$F = z' + xy'$

$F = x' + y'z$

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

3.2

(a) $F = x'y' + xz$

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

$F = xy' + x'y$

(c)

$F = y + z$

(d)

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

$F = z'$

(e)

$F = x + y\,z$

(f)

3.3

(a) $F = xy + x'y'z' + x'yz'$
$F = xy + x'z'$

(b) $F = x'y' + yz + x'yz'$
$F = x' + yz$

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

(c)

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

(d)

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

3.4

(a) $F = y$

(b) $F = BCD + A'BD'$

(c) $F = CD + ABD + ABC$

(d) $F = w'x'y + wx$

$F = wx + wyz$

(e)

$F = wz' + xy'w$

(f)

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

3.5

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

(b) $F = AC' + ABC' + ABD'$

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

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

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

3.7

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

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

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

(d) $F = xw' + xz + xy$

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

3.8

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

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

y (11, 10); x (1); z (01, 11)

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

$AB \backslash CD$	00	01	11	10
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}

C (11, 10); B (01, 11, 10); A (11, 10); D (01, 11)

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

$wx \backslash yz$	00	01	11	10
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}

y (11, 10); x (01, 11, 10); w (11, 10); z (01, 11)

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

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

3.9

(a)

(b)

Essential: *xz, x'z'*
Non-essential: *w'x, w'z'*
F = xz + x'z' + (w'x or w'z')

Essential: *B'D', AC, A'BD*
Non-essential: *CD, B'C*
F = B'D' + AC + A'BD + (CD OR B'C)

(c)

(d)

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

Essential: *BC′, AC, A′B′D*
Non-Essential: *A′B*
F = BC′ + AC + A′B′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 = Σ(0, 2, 5, 7, 8, 10, 12, 13, 14, 15)
Essential: *xz, wx, x′z′*
F = xz + wx + x′z′

(a)

F = Σ(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)

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

F = Σ(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*
F = AC + BC′ + A′B′D

(c)

F = Σ(0, 1, 4, 5, 6, 7, 9, 11, 14, 15)
Essential: *w′y′, xy, wx′z*
Non-essential: *wx, x′y′z, w′wz, w′x′z*
F = *w′y′ + xy + wx′z*

(d)

F(A, B, C, D) = S(0, 1, 3, 7 8, 9, 10,13,15)
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, 10, 15)
Essential: *w′y′, w′z′, xyz, x′yz′*
Non-Essential: *w′x*
F = w′y′ + w′z′ + xyz + x′yz′

(f)

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

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

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

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

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

$F' = yz + xy + xz' + wx'z$

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

3.12 **(a)**

$F = \Pi(1, 3, 5, 7, 13, 15)$
$F' = A'D + B'D$
$F = (A + D')(B' + D')$
$F = C'D' + AB' + CD'$

(b)

$F = \Pi(1, 3, 6, 9, 11, 12, 14)$
$F' = B'D + BCD' + ABD'$
$F = (B + D')(B' + C' + D)(A' + B' + D)$
$F = BD + B'D' + A'C'D'$

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

3.13 **(a)** $F = x'z' + y'z' + yz' + xy = x'z' + z' + xy = z' + xy$

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

y

z

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

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

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

C

B

A

D

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

(c)

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

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

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

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

AB \ CD	00	01	11	10
00	m_0 1	m_1 0	m_3 0	m_2 1
01	m_4 1	m_5 0	m_7 1	m_6 1
11	m_{12} 0	m_{13} 0	m_{15} 0	m_{14} 0
10	m_8 1	m_9 1	m_{11} 1	m_{10} 1

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

(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: $F' = AC' + A'C + A'C'D' + BCD$
$F = (A' + C)(A + C')(A + C + D)(B' + C' + D')$

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

3.15

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

$F = A'D' + B'D' + BCD' + ABC'D$
$F = \Sigma(0, 2, 4, 6, 8, 10, 13, 14)$

$F = BC + CD + ABD' + A'BD$
$F = \Sigma(3, 5, 6, 7, 11, 12, 14, 15)$

$F = B'D' + C'D' + A'BC$
$F =$ $F = \Sigma(0, 2, 4, 6, 7, 8, 10, 12)$

3.16 **(a)**

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

(b)

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

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

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

(d)

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

3.17

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

3.18 $F = (A \oplus B)'(C \oplus D) = (AB' + A'B)'(CD' + C'D)$
$= (AB + A'B')(CD' + C'D) = ABCD' + ABC'D + A'B'CD' + A'B'C'D$
$F' = (AB + A'B')' + (CD' + C'D)'$
$F' = (\ (A' + B')' + (A + B)'\ )' + (\ (C' + D)' + (C + D')'\ )'$

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

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

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

(b)

$F = \Sigma(0, 3, 12, 15)$
$F' = y'z + yz' + w'x + wx' = [(y + z')(w + x')(w + x')(w' + x)]'$
$F = (y + z')' + (y' + z)' + (w + x')' + (w' + x)'$

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

3.20 Multi-level NOR:

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

Multi-level NAND:

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

3.21 $F = w(x + y + z) + xyz$
$F' = [w(x + y + z)]'[xyz]' = [w(x'y'z')')]'(xyz)'$

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

3.22

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

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:

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

(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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

NOR-AND → NOR NAND-OR → NAND

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

3.26

$f = abc' + c'd + a'cd' + b'cd'$

ab \ cd	00	01	11	10
00	m_0	m_1 1	m_3	m_2 1
01	m_4	m_5 1	m_7	m_6 1
11	m_{12} 1	m_{13} 1	m_{15}	m_{14}
10	m_8	m_9 1	m_{11}	m_{10} 1

$g = (a + b + c' + d')(b' + c' + d)(a' + c + d')$

$g' = a'b'cd + bcd' + ac'd$

ab \ cd	00	01	11	10
00	m_0 1	m_1 1	m_3 0	m_2 1
01	m_4 1	m_5 1	m_7 1	m_6 0
11	m_{12} 1	m_{13} 0	m_{15} 1	m_{14} 0
10	m_8 1	m_9 0	m_{11} 1	m_{10} 1

$fg = ac'd + abc'd + b'cd'$

3.27 $x \oplus y = x'y + xy'$; Dual $= (x' + y)(x + y') = (x \oplus y)'$

3.28

(a) 3-bit odd parity generator *(b) 4-bit odd parity generator*

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

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.

(a)
```
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
```

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

(c)
```
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
```

(d)
```
module Fig_3_21b_gates (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, w3, w4, F_bar;
  nand    (w1, A, B_bar);
  nand    (w2, A_bar, B);
  not     (w1_bar, w1);
  not     (w2_bar, w2);
  or      (w3, w1_bar, w2_bar);
  or      (w4, w5, w6);
  not     (w5, C_bar);
  not     (w6, D);
  nand    (F_bar, w3, w4);
  not     (F, F_bar);
endmodule
```

(e)
```
module Fig_3_24_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, w5, w6, w7, w8, w7_bar, w8_bar;
  not     (w1_bar, w1);
  not     (w2_bar, w2);
  not     (w3, E_bar);
  nor     (w1, A, B);
  nor     (W2, C, D);
  and     (F, w1_bar, w2_bar, w3);
endmodule
```

(f)
```
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, w1_bar, w2, w2_bar;
  wire    w3, w4, w5, w6, w7, w8;
  not     (w1, A_bar);
  not     (w2, B);
  not     (w3, A);
  not     (w4, B_bar);
  and     (w5, w1_bar, w2_bar));
  and     (w6, w3, w4);
  nor     (w7, w5, w6);
  nor     (w8, c, d_bar);
  and     (F, w7, w8);
endmodule
```


All rights reserved.

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 sclara result (true or false).

(a)
```
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 = !w3;
  assign   w5 = !(B && C_bar);
  assign   w5_bar = !w5;
  assign   F = w4 || w5_bar);
endmodule
```

(b)
```
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
```

(c)
```
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
```

(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
```


All rights reserved.

(e)
```
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
```

(f)
```
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
```

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.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

(b)

```
`timescale 1ns/1ps

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

and #6 (w3, x, w1);
not #3 (w1, x);
and #6 (w4, y, w1);
not #3 (w2, y);
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;
    #20 y = 1;
  join
endmodule
```

(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 = 1 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.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

3.34

```
module Prob_3_34 (Out_1, Out_2, Out_3, A, B, C, D);
  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 mispelling 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). Termiante 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).

All rights reserved.

3.36 **(a)**

(b)

(c)

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

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
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
// 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
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

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
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

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

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

$F_2 = A'B + D'$

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

4.2

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

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

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

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

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

(b) 1024 rows and 14 columns

4.4 **(a)**

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

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

(b)

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

$F = z$

4.5

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

$A = x'y + yz$

$B = x'y' + y'z + xyz'$

$C = x'z + xz'$

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

4.6

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

$F = xz + yz + xy$

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

4.7 **(a)**

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

$w = A$

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

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

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

(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
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

4.9

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

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

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

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

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

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

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

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

4.10

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

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

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

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

$z = D$

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

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

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

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

4.13

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

4.14 xor AND OR XOR

10 + 5 + 5 + 10 = 30 ns

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

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

4.16 **(a)**

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

(b)

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

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

$(P_iG'_i)\oplus C_i = (A_i + B_i)(A_iB_i)'\oplus C_i = (A_i + B_i)(A'_i + B'_i)\oplus C_i$
$= (A'_iB_i + A_iB'_i)\oplus C_i = A_i\oplus B_i\oplus C_i = S_i$
(b)
Output of NOR gate = $(A_0 + B_0)' = P'_0$
Output of NAND gate = $(A_0B_0)' = G'_0$

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

4.18

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

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

4.19

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

$w = AB + ACD$

$x = B'C' + B'D' + BCD$
$y = C'D + CD'$
$z = D'$

4.23

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

4.24

Inputs: A, B, C, D

$D_0 = A'B'C'D'$
$D_1 = A'B'C'D$
$D_2 = B'CD'$
$D_3 = B'CD$
$D_4 = BC'D'$

Outputs: $D_0, D_1, \ldots D_9$
$D_5 = BC'D$
$D_6 = BCD'$
$D_7 = BCD$
$D_8 = AD'$
$D_9 = AD$

AB \ CD	00	01	11	10
00	m_0 D_0	m_1 D_1	m_3 D_3	x 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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

4.25

4.26

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

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)

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

4.29

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

$V = D_0 + D_1 + D_2 + D_3$

$x = D_1'D_0'$

$y = D_0'D_2' + D_1D_0'$

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

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

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

(b)

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

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

4.33

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

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

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

4.34 **(a)**

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

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

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

(b)

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

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

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

4.35 **(a)**

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

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

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

4.36

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

Note: See Problem 4.45 for testbench)

4.37

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

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


All rights reserved.

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

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

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

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

4.38

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

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

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

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


All rights reserved.

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

With three-state output:

4.39 // Verilog 1995

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

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

// Verilog 2001, 2005

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


All rights reserved.

4.40

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

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

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

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

4.41

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

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

endmodule


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

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


All rights reserved.

4.42 **(a)**

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

(b)

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

(c)

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

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

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

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

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

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

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

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

4.44

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


All rights reserved.

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

endmodule

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

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

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

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

4.45

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


All rights reserved.

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

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

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

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

4.46 **(a)**

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

See code below.

(b) From prob 4.32:

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

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

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

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


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

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

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

4.47

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

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

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

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

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

4.48

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


All rights reserved.

```
endmodule

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

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

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

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

4.49

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

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

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

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


All rights reserved.

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

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

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

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

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

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

// Verilog 2001, 2005

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

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

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

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

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

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


All rights reserved.

```
endmodule
```

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

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

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

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

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

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

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

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

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

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


All rights reserved.

```
  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
```


All rights reserved.

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
```


All rights reserved.

```
    #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
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

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

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

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

  Problem_4_52a_Data_Flow  M0 (sum, carry, A);

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

(b) Decrementer for unsigned 4-bit numbers

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

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

  Problem_4_52b_Data_Flow  M0 (diff, borrow, A);

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

4.53

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

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

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

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

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

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

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

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

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

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

  Nines_Complementer M0 (Word_9s_Comp, Word_BCD);

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

(b) 9s complement of Gray Code

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

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

  Nines_Complementer M0 (Word_9s_Comp, Word_Gray);

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


All rights reserved.

4.55 From Problem 4.19:

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

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

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


All rights reserved.

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

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

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

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

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

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

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

  initial #1000 $finish;

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

4.56

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

4.57

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

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

// Verilog 2001, 2005

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

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


All rights reserved.

4.58 **(a)**

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

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


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

  reg [31: 0] sig_in;

  //shift_right_by_3_V2001 M0 (sig_out_V2001, sig_in);

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

(b)

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

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


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

  reg [31: 0] sig_in;

  shift_left_by_3_V2001 M0 (sig_out_V2001, sig_in);

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

4.59

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

4.60

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

See Problem 4.62 for testbench and waveforms.

4.61

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

4.62

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

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

4.63

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

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

4.64

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

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

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

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

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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

4.65

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

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

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

module Mux_16x1 (
  output y_out,
```


All rights reserved.

```
  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
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

CHAPTER 5

5.1 **(a)**

(b)

(c)

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

5.2

5.3 $Q'(t + 1) = (JQ' + K'Q)' = (J' + Q)(K + Q') = J'Q' + KQ$

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

$Q(t+1) = PQ' + NQ$

(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.

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.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

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

(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

$S = x \oplus y \oplus Q$
$Q(t + 1) = xy + xQ + yQ$

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

5.8 A counter with a repeated sequence of 00, 01, 10.

Present state A	Present state B	Next state A	Next state B	FF Inputs T_A	FF Inputs T_B
0	0	0	0	0	1
0	1	1	0	1	1
1	0	0	0	1	0
1	1	0	0	1	1

$T_A = A + B$
$T_B = A' + B$

Repeated sequence:

$00 \rightarrow 01 \rightarrow 10 \rightarrow$

5.9

$J_A = x$ $\quad K_A = B$
$J_B = x$ $\quad K_B = A'$

$A(t+1) = J_A A' + K_A' A = xA' + B'A$
$B(t+1) = J_B B' + K_B' B = xB' + AB$

x	A	B	$xA' + B'A$	$xB' + AB$
0	0	0	0	0
0	0	1	0	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	1

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

5.10 **(a)** $J_A = Bx + B'y'$ $\quad J_B = A'x$

$K_A = B'xy'$ $\quad K_B = A + xy'$ $\quad z = Axy + Bx'y'$

(b)

Present state		Inputs		Next state		Output	FF Inputs			
A	B	x	y	A	B	z	J_A	K_A	J_A	J_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)

$A(t+1) = Ax' + Bx + Ay + A'B'y'$

$B(t+1) = A'B'x + A'B'(x' + y)$

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

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 = (!input) && (state == b)

5.12

Present state	**Next state 0**	**Next state 1**	**Output 0**	**Output 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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

5.13 **(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	

5.14

	Present state ABCDE	Next state x=0	Next state x=1	Output x=1	Output 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 Q	*Inputs* J	*Inputs* K	*Next state* 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

$Q(t+1) = D_Q + Q'J + QK'$

5.16 **(a)** $D_A = Ax' + Bx$

$D_B = A'x + Bx'$

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

Present state A	Present state B	Input x	Next state A	Next state 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

$D_A = Ax' + Bx$

$D_B = A'x + Bx'$

(b) $D_A = A'x + Ax'$

$D_B = AB + Bx'$

Present state A	Present state B	Input x	Next state A	Next state 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

$D_A = A'x + Ax'$

$D_B = AB + Bx'$

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'.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

Present state	*Input*	*Next state*	*Output*
A	*x*	*A*	*y*
0	*0*	*0*	*0*
0	*1*	*1*	*1*
1	*0*	*1*	*1*
1	*1*	*1*	*0*

$D_A = A + x$
$y = Ax' + A'x$

5.18 Binary up-down counter with enable *E*.

Present state *A B*	*Input* *x*	*Next state* *A B*	*Flip-flop inputs* J_A	K_A	J_B	K_B
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

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

$J_A = (Bx + B'x')E$ $K_A = (Bx + B'x')E$

$J_B = E$ $K_B = E$

5.19 **(a)** Unused states (see Fig. P5.19): 101, 110, 111.

Present state ABC	*Input x*	*Next state ABC*	*Output 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

$d(A, B, C, x) = \Sigma\,(10, 11, 12, 13, 14, 15)$

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

$D_A = A'B'x$

$D_B = A + C'x' + BCx$

$D_C = Cx' + Ax + A'B'x'$

$y = A'x$

The machine is self-correcting, i.e., the unused states transition to known states.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

(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

$J_A = B'x$ $\quad K_A = 1$
$J_B = A + C'x'$ $\quad K_B = C'x + Cx'$
$J_C = Ax + A'B'x'$ $\quad K_C = x$
$y = A'x$
The machine is self-correcting because $K_A = 1$.

5.20 From state table 5.4: $T_A(A, B, x) = \Sigma(2, 3, 6)$, $T_B(A, B, x) = \Sigma(0, 3, 4, 6)$.

$T_A = A'B + Bx'$

$T_B = B'x' + A'x + A'Bx$

5.21 The statements associated with an **initial** keyword execute once, in sequence, with the activity expiring after the last statment competes execution; the statements assocated with the **always** keyword execute repeatedly, subject to timing control (e.g, #10).

5.22

5.23 **(a)** *RegA* = 125, *RegB* = 125
(b) *RegA* = 125, *RegB* = 50 Note: Text has error, with RegB = 30 at page 526).

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

5.24 **(a)**

```
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)

```
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
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

5.25

```
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
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

5.26 (a)

$Q(t + 1) = JQ' + K'Q$

When $Q = 0$, $Q(t + 1) = J$
When $Q = 1$, $Q(t + 1) = K'$

```
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
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

(b)

```
module JK_Behavior_b (output reg Q, input J, K, CLK, reset_b);
  always @ (posedge CLK, negedge 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
```

5.27

```
// 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;
```


All rights reserved.

```
  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.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

5.28 **(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
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

(b)

```
 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

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
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

(c) See results of (b) and (c).

```
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
```

5.29

```
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 ();
```


All rights reserved.

```
  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
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

5.30

5.31

```
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
```

Note: The statements must be written in an order than produces the effect of concurrent assignments.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

5.32

```
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
```

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.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

5.34 Note: Problem statement should refer to Problem 5.2 instead of Fig 5.5.

```
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
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

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

(c)

```
module Prob_5_35 (output out_z, input in_x, in, 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)
    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 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;
    endcase
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
module t_Prob_5_35 ();
  wire out_z;
  reg in_x, in, in_y, clk, reset_b;

  Prob_5_35 M0 (out_z, in_x, in, 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'b00
    #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
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

5.36 Note: See Problem 5.8 (counter with repeated sequence: (A, B) = 00, 01, 10, 00 ....

```
// See Fig. P5.8
module Problem_5_36 (output A, B, input Clock, reset_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
```

5.37

```
module Problem_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
```


All rights reserved.

```
      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 Problem_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;
```


All rights reserved.

```
  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
```


All rights reserved.

5.38 (a)

```
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
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
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
```

(b)

```
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
endmodule
```


All rights reserved.

```
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
```

5.39

```
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 ();
```


All rights reserved.

```
  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
```

5.40

```
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
```


All rights reserved.

```
    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_segment = 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
   #4 reset_b = 1;
   #2 E = 0;
   #20 begin E = 1; F = 1; end
   #60 E = 0;
   #80 E = 1;
   #90 E = 0;
   #110 E = 1;
   #120 E = 0;
   #140 E = 1;
   #150 E = 0;
   #170 E= 1;
   #170 F = 0;
  join
endmodule
```

5.41

```
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;
```


All rights reserved.

```
  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
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

5.42

```
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;
    #4reset_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
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

5.43

```
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
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

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.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

5.44

```
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
```

5.45

```
module Seq_Detector_Prob_5_45 (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
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

```
module t_Seq_Detector_Prob_5_45 ();
  wire detect;
  reg bit_in, clk, reset_b;

  Seq_Detector_Prob_5_45 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;
    #4reset_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
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

5.46 Pending simulation results

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.

```
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;
```


All rights reserved.

```
    #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
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

5.47

Assume synchronous active-low reset.

```
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'b000;
    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
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

5.48

Assume "a" is the reset state.

```
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…
  Transitons to b; hold x_in = 1 and get loop bdacd…
  Transition to d; hold x_in = 0 and get loop dcbc…
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…
```


All rights reserved.

```
    #300 reset_b = 0;
    #310 reset_b = 1;
    #310 x_in = 1;
    #330 x_in = 0;          // loop acdcbcbc….
join
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

5.49

Assume "a" is the reset state.

```
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…
    // Transitons to b; hold x_in = 1 and get loop bdacd…
    // Transition to d; hold x_in = 0 and get loop dcbc…
// 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 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…
```


All rights reserved.

```
    #300 reset_b = 0;
    #310 reset_b = 1;
    #310 x_in = 1;
    #330 x_in = 0;          // loop acdcbcbc….
  join
  endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

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.

```
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;
    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'b0; Set_flag = 1'b0; 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
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
// 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
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

5.51

Assumption: Synchronous active-low reset
Moore machine, links for reset on-the-fly are implicit and not shown

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

5.54

Assumption: Synchronous active-low reset
Moore machine, links for reset on-the-fly are implicit and not shown

5.55

Assumption: Synchronous active-low reset
Mealy machine, links for reset on-the-fly are implicit and not shown

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

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') || (x_in' D2 D1' D0') || (x_in D2 D1' D0') || (x_in D2 D1 D0')
nD1 = (x_in D2' D1' D0') || (x_in' D2' D1 D0') || (x_in D2 D1' D0') || (x_in' D2 D1 D0')

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

$nD2 = D2\ D1'D0' + x_in\ D1\ D0'$

$nD1 = x_inD1'\ D0' + x_in'\ D1\ D0'$

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

5.57

Assume synchronous active-low reset. Assume that the counter is controlled by assertion of *Run*.

```
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

// 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
```

All rights reserved.

5.58

```
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: if (x_in == 1'b0) next_state = s0; else if (x_in = 1'b1) begin  next_state = s3; y_out = 1; 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
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

5.59

```
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;
      3'b010:     count <= 3'b100;
      3'b100:     count <= 3'b110;
      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
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

5.60

Assume synchronous active-low reset. Assume that counting is controlled by Run.

```
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'b000;
    else if (Run && (y_out < 4'b1001)) y_out <= y_out + 4'b0001;
    else if (Run && (y_out == 4'b1001)) y_out <= 4'b0000;500
    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
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

CHAPTER 6

6.1 The structure shown below gates the clock through a nand gate. In practice, the circuit can exhibit two problems if the load signal is asynchronous: (1) the gated clock arrives in the setup interval of the clock of the flip-flop, causing metastability, and (2) the load signal truncates the width of the clock pulse. Additionally, the propagation delay through the nand gate might compromise the synchronicity of the overall circuit.

6.2 Modify Fig. 6.2, with each stage replicating the first stage shown below:

Load	Clear	D	Operation
0	0	A_0	No change
0	1	0	Clear to 0
1	x	I_0	Load input

Note: In this design, *clear* has priority over *load*.

6.3 Serial data is transferred one bit at a time, in sequence. Parallel data is transferred n bits at a time (n > 1), concurrently.

A shift register can convert serial data into parallel data by first shifting one bit a time into the register and then taking the parallel data from the register outputs.

A shift register with parallel load can convert parallel data to a serial format by first loading the data in parallel and then shifting the bits one at a time.

6.4 0110 => 0011, 0001, 1000, 1100, 1110, 0111, 1011

6.5 **(a)** See Fig. 11.19: IC 74194

(b) See Fig. 11.20. Connect two 74194 ICs to form an 8-bit register.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

6.6 First stage of register:

6.7 First stage of register:

6.8 A = 0010, 0001, 1000, 1100. Carry = 1, 1, 1, 0

6.9 **(a)** In Fig. 6.5, complement the serial output of shift register B (with an inverter), and set the initial value of the carry to 1.

(b)

Present state Q	Inputs x	y	Next state Q	Output D	FF inputs J_Q	K_Q
0	0	0	0	0	0	x
0	0	0	1	1	1	x
0	0	1	0	1	0	x
0	0	1	0	0	0	x
1	1	0	1	1	x	0
1	1	0	1	0	x	0
1	1	1	0	0	x	1
1	1	1	1	1	x	0

$J_Q = x'y$

$K_Q = xy' \oplus \quad \oplus$

$D = Q \oplus x \oplus y$

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

6.10 See solution to Problem 5.7.
Note that y = x if Q = 0, and y = x' if Q = 1. Q is set on the first 1 from x.
Note that x ⊕ 0 = x, and x ⊕ 1 = x'.

6.11 **(a)** A count down counter.

(b) A count up counter.

6.12 Similar to diagram of Fig. 6.8.

(a) With the bubbles in *C* removed (positive-edge).

(b) With complemented flip-flops connected to *C*.

6.13

6.14 **(a)** 10_0110_0111 **->** 10_011**_1000** 4;

(b) 11_1100_0111 -> 11_1100**_1000** 4;

(c) 00_0000_1111 -> 00_000**1_0000** 5

6.15 The worst case is when all 10 flip-flops are complemented. The maximum delay is 10 x 3ns = 30 ns.

The maximum frequency is $10^9/30$ = 33.3 MHz

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

6.16

Q8 Q4 Q2 Q1 :	1010	1100	1110	Self correcting
Next state:	1011	1101	1111	
Next state:	0100	0100	0000	

1010 → 1011 → 0100
1100 → 1101 → 0100
1110 → 1111 → 0000

6.17 With E denoting the count enable in Fig. 6.12 and D-flip-flops replacing the J-K flip-flops, the toggling action of the bits of the counter is determined by: $T_0 = E$, $T_1 = A_0E$, $T_2 = A_0A_1E$, $T_3 = A_0A_1A_2E$. Since $D_A = A \oplus T_A$ the inputs of the flip-flops of the counter are determined by: $D_{A0} = A_0 \oplus E$; $D_{A1} = A_1 \oplus (A_0E)$; $D_{A2} = A_2 \oplus (A_0A_1E)$; $D_{A3} = A_3 \oplus (A_0A_1A_2E)$.

6.18 When *up* = *down* = 1 the circuit counts up.

up	*down*	*x*	*y*	Operation
0	0	0	0	No change
0	1	0	0	Count down
1	0	1	0	Count up
1	1	0	0	No change

x = up (down)'
y = (up)'down

6.19 **(b)** From the state table in Table 6.5:

$D_{Q1} = Q'_1$
$D_{Q2} = \sum (1, 2, 5, 6)$
$D_{Q4} = \sum (3, 4, 5, 6)$
$D_{Q8} = \sum (7, 8)$
Don't care: d = $\sum (10, 11, 12, 13, 14, 15)$

Simplifying with maps:
$D_{Q2} = Q_2Q'_1 + Q'_8Q'_2Q_1$
$D_{Q4} = Q_4Q'_1 + Q4Q'2 + Q'_4Q_2Q_1$
$D_{Q8} = Q_8Q'_1 + Q_4Q_2Q_1$

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

(a)

Present state	*Next state*	*Flip-flop inputs*			
$A_8\,A_4\,A_2\,A_1$	$A_8\,A_4\,A_2\,A_1$	$J_{A8}\,K_{A8}$	$J_{A4}\,K_{A4}$	$J_{A2}\,K_{A2}$	$J_{A1}\,K_{A1}$
0 0 0 0	0 0 0 1	0 x	0 x	0 x	1 x
0 0 0 1	0 0 1 0	0 x	0 x	1 x	x 1
0 0 1 0	0 0 1 1	0 x	0 x	x 0	1 x
0 0 1 1	0 1 0 0	0 x	1 x	x 1	x 1
0 1 0 0	0 1 0 1	0 x	x 0	0 x	1 x
0 1 0 1	0 1 1 0	0 x	x 0	1 x	x 1
0 1 1 0	0 1 1 1	0 x	x 0	x 0	1 x
0 1 1 1	1 0 0 0	1 x	x 1	x 1	x 1
1 0 0 0	1 0 0 1	x 0	0 x	0 x	1 x
1 0 0 1	0 0 0 0	x 1	0 x	0 x	x 1

$J_{A1} = 1$
$K_{A1} = 1$
$J_{A2} = A_1A'_8$
$K_{A2} = A_1$
$J_{A4} = A_1A_2$
$K_{A4} = A_1A_2$
$J_{A8} = A_1A_2A_4$
$K_{A8} = A_1$

$d(A_8, A_4, A_2, A_1) = \Sigma\,(10, 11, 12, 13, 14, 15)$

6.20 **(a)**

Block diagram of 4-bit circuit:

16-bit counter needs 4 circuits with output carry connected to the count input of the next stage.

(b) Need 2 units to count to 127. Counter is re-loaded with 0s when count reaches 128.
An alternative version would AND output bits 0 through 6 and assert Load while the count is 127.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

6.21 **(a)**

$J_{A0} = LI_0 + L'C \qquad KA_0 = LI'_0 + L'C$

(b)

$J = [L(LI)']'(L + C) = (L' + LI)(L + C)$

$LI + L'C + LIC = LI + L'C$ *(use a map)*

$K = (LI)' (L + C) = (L' + I')(L + C) = LI' + L'C$

6.22

Count sequence: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Count sequence: 6, 7, 8, 9, 10, 11, 12 13, 14, 15

Count sequence: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

6.23 Use a 4-bit counter and a flip-flop (initially at 0). A start signal sets the flip-flop, which in turn enables the counter. On the count of 11 (binary 1011) reset the flip-flop to 0 to disable the count (with the value of 0000).

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

6.24

Present state ABC	*Next state ABC*	*Flip-flop inputs* T_A	T_B	T_C
000	001	0	0	1
001	011	0	1	0
010	xxx	x	x	x
011	111	1	1	0
100	000	1	1	0
101	xxx	x	x	x
110	100	0	1	0
111	110	0	0	1

6.25 **(a)** Use a 6-bit ring counter.

(b)

6.26 The clock generator has a period of 12.5 ns. Use a 2-bit counter to count four pulses.

80/4 = 20 MHz; cycle time = 1000 x 10^{-9} /20 = 50 ns.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

6.27

Present state ABC	*Next state ABC*	*Flip-flop inputs* J_A	K_A	J_B	K_B	J_C	K_C
000	001	0	x	0	x	1	x
001	010	0	x	1	x	x	1
010	011	0	x	x	0	1	x
011	100	1	x	x	1	x	1
100	100	x	x	0	0	1	x
101	110	x	x	1	x	x	1
110	000	x	x	x	1	0	x
111	xxx	x	x	x	x	x	x

Self-correcting

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

6.28

Present state ABC	*Next state ABC*
000	001
001	010
010	100
011	xxx
100	110
101	xxx
110	000
111	xxx

A \ BC	00	01	11	10
0	m_0	m_1	m_3 x	m_2 1
1	m_4 1	m_5 x	m_7 x	m_6

$D_A = A \oplus B$

A \ BC	00	01	11	10
0	m_0 1	m_1 x	m_3 x	m_2 1
1	m_4 1	m_5 x	m_7 x	m_6

$D_B = AB' + C$

A \ BC	00	01	11	10
0	m_0 1	m_1	m_3 x	m_2
1	m_4	m_5 x	m_7 x	m_6

$D_C = A'B'C'$

Self-correcting

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

6.29 **(a)** The 8 valid states are listed in Fig. 8.18(b), with the sequence: 0, 8, 12, 14, 15, 7, 3, 1, 0, ...

The 8 unused states and their next states are shown below:

State *ABCE*	*Next state* *ABCE*	*All invalid states*
0000	1001	9
0100	1010	10
0101	0010	2
0110	1011	11
1001	0100	4
1010	1101	13
1011	0101	5
1101	0110	6

(b) Modification: $D_C = (A + C)B$.

The valid states are the same as in (a). The unused states have the following sequences: 2→ 9→ 4→ 8 and 10→ 13→ 6→11→ 5→ 0. The final states, 0 and 8, are valid.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

6.30

The 5-bit Johnson counter has the following state sequence:

6.31

```
module Reg_4_bit_beh (output reg A3, A2, A1, A0, input I3, I2, I1, I0, Clock, Clear);
  always @ (posedge Clock, negedge Clear)
    if (Clear == 0) {A3, A2, A1, A0} <= 4'b0;
    else {A3, A2, A1, A0} <= {I3, I2, I1, I0};
endmodule

module Reg_4_bit_Str (output A3, A2, A1, A0, input I3, I2, I1, I0, Clock, Clear);
  DFF M3DFF (A3, I3, Clock, Clear);
  DFF M2DFF (A2, I2, Clock, Clear);
  DFF M1DFF (A1, I1, Clock, Clear);
  DFF M0DFF (A0, I0, Clock, Clear);
endmodule

module DFF(output reg Q, input D, clk, clear);
  always @ (posedge clk, posedge clear)
    if (clear == 0) Q <= 0; else Q <= D;
endmodule

module t_Reg_4_bit ();
  wire A3_beh, A2_beh, A1_beh, A0_beh;
  wire A3_str, A2_str, A1_str, A0_str;
  reg I3, I2, I1, I0, Clock, Clear;
  wire [3: 0] I_data = {I3, I2, I1, I0};
  wire [3: 0] A_beh = {A3_beh, A2_beh, A1_beh, A0_beh};
  wire [3: 0] A_str = {A3_str, A2_str, A1_str, A0_str};

  Reg_4_bit_beh M_beh (A3_beh, A2_beh, A1_beh, A0_beh, I3, I2, I1, I0, Clock, Clear);
  Reg_4_bit_Str M_str (A3_str, A2_str, A1_str, A0_str, I3, I2, I1, I0, Clock, Clear);

  initial #100 $finish;
  initial begin Clock = 0; forever #5 Clock = ~Clock; end
  initial begin Clear = 0; #2 Clear = 1; end
  integer K;
  initial begin
    for (K = 0; K < 16; K = K + 1) begin {I3, I2, I1, I0} = K; #10 ; end
  end
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

6.32 **(a)**

```
module Reg_4_bit_Load (output reg A3, A2, A1, A0, input I3, I2, I1, I0, Load, Clock, Clear);
  always @ (posedge Clock, negedge Clear)
    if (Clear == 0) {A3, A2, A1, A0} <= 4'b0;
    else if (Load) {A3, A2, A1, A0} <= {I3, I2, I1, I0};
endmodule

module t_Reg_4_Load ();
  wire A3_beh, A2_beh, A1_beh, A0_beh;
   reg I3, I2, I1, I0, Load, Clock, Clear;
  wire [3: 0] I_data = {I3, I2, I1, I0};
  wire [3: 0] A_beh = {A3_beh, A2_beh, A1_beh, A0_beh};

  Reg_4_bit_Load M0 (A3_beh, A2_beh, A1_beh, A0_beh, I3, I2, I1, I0, Load, Clock, Clear);

  initial #100 $finish;
  initial begin Clock = 0; forever #5 Clock = ~Clock; end
  initial begin Clear = 0; #2 Clear = 1; end
  integer K;
  initial fork
    #20 Load = 1;
    #30 Load = 0;
    #50 Load = 1;
  join
  initial begin
    for (K = 0; K < 16; K = K + 1) begin {I3, I2, I1, I0} = K; #10 ; end
  end
endmodule
```


All rights reserved.

(b)

```
module Reg_4_bit_Load_str (output A3, A2, A1, A0, input I3, I2, I1, I0, Load, Clock, Clear);
  wire y3, y2, y1, y0;
  mux_2 M3 (y3, A3, I3, Load);
  mux_2 M2 (y2, A2, I2, Load);
  mux_2 M1 (y1, A1, I1, Load);
  mux_2 M0 (y0, A0, I0, Load);

  DFF M3DFF (A3, y3, Clock, Clear);
  DFF M2DFF (A2, y2, Clock, Clear);
  DFF M1DFF (A1, y1, Clock, Clear);
  DFF M0DFF (A0, y0, Clock, Clear);
endmodule

module DFF(output reg Q, input D, clk, clear);
  always @ (posedge clk, posedge clear)
    if (clear == 0) Q <= 0; else Q <= D;
endmodule

module mux_2 (output y, input a, b, sel);
  assign y = sel ? a: b;
endmodule

module t_Reg_4_Load_str ();
  wire A3, A2, A1, A0;
  reg I3, I2, I1, I0, Load, Clock, Clear;
  wire [3: 0] I_data = {I3, I2, I1, I0};
  wire [3: 0] A = {A3, A2, A1, A0};

  Reg_4_bit_Load_str M0 (A3, A2, A1, A0, I3, I2, I1, I0, Load, Clock, Clear);
```


All rights reserved.

```
  initial #100 $finish;
  initial begin Clock = 0; forever #5 Clock = ~Clock; end
  initial begin Clear = 0; #2 Clear = 1; end
  integer K;
  initial fork
    #20 Load = 1;
    #30 Load = 0;
    #50 Load = 1;
    #80 Load = 0;
  join
  initial begin
    for (K = 0; K < 16; K = K + 1) begin {I3, I2, I1, I0} = K; #10 ; end
  end
endmodule
```

(c)

```
module Reg_4_bit_Load_beh (output reg A3, A2, A1, A0, input I3, I2, I1, I0, Load, Clock, Clear);
  always @ (posedge Clock, negedge Clear)
    if (Clear == 0) {A3, A2, A1, A0} <= 4'b0;
    else if (Load) {A3, A2, A1, A0} <= {I3, I2, I1, I0};
endmodule

module Reg_4_bit_Load_str (output A3, A2, A1, A0, input I3, I2, I1, I0, Load, Clock, Clear);
  wire y3, y2, y1, y0;
  mux_2 M3 (y3, A3, I3, Load);
  mux_2 M2 (y2, A2, I2, Load);
  mux_2 M1 (y1, A1, I1, Load);
  mux_2 M0 (y0, A0, I0, Load);

  DFF M3DFF (A3, y3, Clock, Clear);
  DFF M2DFF (A2, y2, Clock, Clear);
  DFF M1DFF (A1, y1, Clock, Clear);
  DFF M0DFF (A0, y0, Clock, Clear);
endmodule

module DFF(output reg Q, input D, clk, clear);
  always @ (posedge clk, posedge clear)
    if (clear == 0) Q <= 0; else Q <= D;
endmodule

module mux_2 (output y, input a, b, sel);
  assign y = sel ? a: b;
endmodule
```


All rights reserved.

```
module t_Reg_4_Load_str ();
  wire A3_beh, A2_beh, A1_beh, A0_beh;
  wire A3_str, A2_str, A1_str, A0_str;
  reg I3, I2, I1, I0, Load, Clock, Clear;
  wire [3: 0] I_data, A_beh, A_str;
  assign  I_data = {I3, I2, I1, I0};
  assign  A_beh = {A3_beh, A2_beh, A1_beh, A0_beh};
  assign A_str = {A3_str, A2_str, A1_str, A0_str};

  Reg_4_bit_Load_str M0 (A3_beh, A2_beh, A1_beh, A0_beh, I3, I2, I1, I0, Load, Clock, Clear);
  Reg_4_bit_Load_str M1 (A3_str, A2_str, A1_str, A0_str, I3, I2, I1, I0, Load, Clock, Clear);

  initial #100 $finish;
  initial begin Clock = 0; forever #5 Clock = ~Clock; end
  initial begin Clear = 0; #2 Clear = 1; end
  integer K;
  initial fork
    #20 Load = 1;
    #30 Load = 0;
    #50 Load = 1;
    #80 Load = 0;
  join
  initial begin
    for (K = 0; K < 16; K = K + 1) begin {I3, I2, I1, I0} = K; #10 ; end
  end
endmodule
```

6.33

```
// Stimulus for testing the binary counter of Example 6-3

module testcounter;
  reg Count, Load, CLK, Clr;
  reg [3: 0] IN;
  wire C0;
  wire [3: 0] A;
  Binary_Counter_4_Par_Load M0 (
  A,      // Data output
  C0,         // Output carry
  IN,         // Data input
  Count,          // Active high to count
  Load,           // Active high to load
  CLK,        // Positive edge sensitive
  Clr         // Active low
);
```


All rights reserved.

```
 always
   #5 CLK = ~CLK;
  initial
   begin
     Clr = 0;                    // Clear de-asserted
     CLK = 1;                    // Clock initialized high
     Load = 0; Count = 1;                // Enable count
     #5 Clr = 1;                         // Clears count, then counts for five cycles
     #50 Load = 1; IN = 4'b1100;         // Count is set to 4'b1100 (12₁₀)
     #10 Load = 0;
     #70 Count = 0;              // Count is deasserted at t = 135
     #20 $finish;                // Terminate simulation
   end
endmodule
// Four-bit binary counter with parallel load
// See Figure 6-14 and Table 6-6
module Binary_Counter_4_Par_Load (
  output reg [3:0] A_count,   // Data output
  output           C_out, // Output carry
  input [3:0]      Data_in,   // Data input
  input            Count,     // Active high to count
             Load,  // Active high to load
             CLK,   // Positive edge sensitive
             Clear  // Active low
);


  assign C_out = Count & (~Load) & (A_count == 4'b1111);
  always @ (posedge CLK, negedge Clear)
  if (~Clear)        A_count <= 4'b0000;
  else if (Load)     A_count <= Data_in;
  else if (Count)    A_count <= A_count + 1'b1;
  else               A_count <= A_count; // redundant statement
endmodule

// Note: a preferred description if the clock is given by:
// initial begin CLK = 0; forever #5 CLK = ~CLK; end
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

6.34

```
module Shiftreg (SI, SO, CLK);
  input      SI, CLK;
  output     SO;
  reg    [3: 0]    Q;
  assign     SO = Q[0];
  always @ (posedge CLK)
    Q = {SI, Q[3: 1]};
endmodule

// Test plan
//
// Verify that data shift through the register
// Set SI =1 for 4 clock cycles
// Hold SI =1 for 4 clock cycles
// Set SI = 0 for 4 clock cycles
// Verify that data shifts out of the register correctly

module t_Shiftreg;
  reg    SI, CLK;
  wire   SO;

  Shiftreg M0 (SI, SO, CLK);

  initial #130 $finish;
  initial begin CLK = 0; forever #5 CLK = ~CLK; end
  initial fork
    SI = 1'b1;
    #80 SI = 0;
  join
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

6.35 **(a)** Note that *Load* has priority over *Clear*.

```
module Prob_6_35a (output [3: 0] A, input [3:0] I, input Load,  Clock, Clear);
  Register_Cell R0 (A[0], I[0], Load, Clock, Clear);
  Register_Cell R1 (A[1], I[1], Load, Clock, Clear);
  Register_Cell R2 (A[2], I[2], Load, Clock, Clear);
  Register_Cell R3 (A[3], I[3], Load, Clock, Clear);
endmodule

module Register_Cell (output A, input I, Load, Clock, Clear);
  DFF M0 (A, D, Clock);
  not (Load_b, Load);
  not (w1, Load_b);
  not (Clear_b, Clear);
  and (w2, I, w1);
  and (w3, A, Load_b, Clear_b);
  or (D, w2, w3);
endmodule

module DFF (output reg Q, input D, clk);
always @ (posedge clk) Q <= D;
endmodule

module t_Prob_6_35a ( );

  wire [3: 0] A;
  reg [3: 0] I;
  reg Clock, Clear, Load;

  Prob_6_35a M0 ( A, I, Load, Clock, Clear);
  initial #150 $finish;
  initial begin Clock = 0; forever #5 Clock = ~Clock; end
  initial fork
  I = 4'b1010;Clear = 1;
  #40 Clear = 0;
  Load = 0;
  #20 Load = 1;
  #40 Load = 0;
  join
endmodule
```

(b) Note: The solution below replaces the solution given on the preliminary CD.

```
module Prob_6_35b (output reg [3: 0] A, input [3:0] I, input Load, Clock, Clear);
  always @ (posedge Clock)
   if (Load) A <= I;
  else if (Clear) A <= 4'b0;
  //else A <= A;              // redundant statement
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

```
endmodule

module t_Prob_6_35b ( );

  wire [3: 0] A;
  reg [3: 0] I;
  reg Clock, Clear, Load;

  Prob_6_35b M0 ( A, I, Load, Clock, Clear);
  initial #150 $finish;
  initial begin Clock = 0; forever #5 Clock = ~Clock; end
  initial fork
  I = 4'b1010; Clear = 1;
  #60 Clear = 0;
  Load = 0;
  #20 Load = 1;
  #40 Load = 0;
   join
endmodule
```

(c)

```
module Prob_6_35c (output [3: 0] A, input [3:0] I, input Shift, Load,  Clock);
  Register_Cell R0 (A[0], I[0], A[1], Shift, Load, Clock);
  Register_Cell R1 (A[1], I[1], A[2], Shift, Load, Clock);
  Register_Cell R2 (A[2], I[2], A[3], Shift, Load, Clock);
  Register_Cell R3 (A[3], I[3], A[0], Shift, Load, Clock);
endmodule

module Register_Cell (output A, input I, Serial_in, Shift, Load, Clock);
  DFF M0 (A, D, Clock);
  not (Shift_b, Shift);
  not (Load_b, Load);
  and (w1, Shift, Serial_in);
  and (w2, Shift_b, Load, I);

  and (w3, A, Shift_b, Load_b);
  or (D, w1, w2, w3);
endmodule

module DFF (output reg Q, input D, clk);
always @ (posedge clk) Q <= D;
endmodule

module t_Prob_6_35c ( );

  wire [3: 0] A;
  reg [3: 0] I;
```


All rights reserved.

```
    reg Clock, Shift, Load;

    Prob_6_35c M0 (A, I, Shift, Load, Clock);
    initial #150 $finish;
    initial begin Clock = 0; forever #5 Clock = ~Clock; end
    initial fork
    I = 4'b1010;
    Load = 0; Shift = 0;
    #20 Load = 1;
    #40 Load = 0;
    #50 Shift = 1;
     join
   endmodule
```

(d)

```
module Prob_6_35d (output reg [3: 0] A, input [3:0] I, input Shift, Load, Clock, Clear);
  always @ (posedge Clock)
   if (Shift) A <= {A[0], A[3:1]};
   else if (Load) A <= I;
   else if (Clear) A <= 4'b0;
   //else A <= A;              // redundant statement
endmodule

module t_Prob_6_35d ( );

  wire [3: 0] A;
  reg [3: 0] I;
  reg Clock, Clear, Shift, Load;

   Prob_6_35d M0 ( A, I, Shift, Load, Clock, Clear);
  initial #150 $finish;
  initial begin Clock = 0; forever #5 Clock = ~Clock; end
  initial fork
   I = 4'b1010; Clear = 1;
   #100 Clear = 0;
   Load = 0;
   #20 Load = 1;
   #40 Load = 0;
   #30 Shift = 1;
   #90 Shift = 0;
  join
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

(e)

```
module Shift_Register
  (output [3: 0] A_par, input [3: 0] I_par, input MSB_in, LSB_in, s1, s0, CLK, Clear);
  wire y3, y2, y1, y0;
  DFF D3 (A_par[3],  y3, CLK, Clear);
  DFF D2 (A_par[2],  y2, CLK, Clear);
  DFF D1 (A_par[1],  y1, CLK, Clear);
  DFF D0 (A_par[0],  y0, CLK, Clear);

  MUX_4x1 M3 (y3, I_par[3], A_par[2], MSB_in, A_par[3], s1, s0);
  MUX_4x1 M2 (y2, I_par[2], A_par[1], A_par[3], A_par[2], s1, s0);
  MUX_4x1 M1 (y1, I_par[1], A_par[0], A_par[2], A_par[1], s1, s0);
  MUX_4x1 M0 (y0, I_par[0], LSB_in, A_par[1], A_par[0], s1, s0);
endmodule

module MUX_4x1 (output reg y, input I3, I2, I1, I0, s1, s0);
  always @ (I3, I2, I1, I0, s1, s0)
    case ({s1, s0})
      2'b11:  y = I3;
      2'b10:  y = I2;
      2'b01:  y = I1;
      2'b00:  y = I0;
    endcase
endmodule

module DFF (output reg Q,  input D, clk, reset_b);
  always @ (posedge clk, negedge reset_b) if (reset_b == 0) Q <= 0; else Q <= D;
endmodule

module t_Shift_Register ( );
  wire   [3: 0] A_par;
  reg    [3: 0] I_par;
  reg MSB_in, LSB_in, s1, s0, CLK, Clear;

  Shift_Register M_SR( A_par, I_par, MSB_in, LSB_in, s1, s0, CLK, Clear);
  initial #300 $finish;
  initial begin CLK = 0; forever #5 CLK = ~CLK; end
   initial fork
    MSB_in = 0; LSB_in = 0;
    Clear = 0;              // Active-low reset
    s1  = 0; s0 = 0;            // No change
    #10 Clear = 1;
    #10 I_par = 4'hA;
    #30 begin s1 = 1; s0 = 1; end // 00: load I_par into A_par
    #50 s1 = 0;            // 01: shift right (1010  to 0101 to 0010 to 0001 to 0000)
    #90 begin s1 = 1; s0 = 1; end // 11: reload A with 1010
    #100 s0 = 0;               // 10: shift left (1010 to 0100 to 1000 to 000)
    #140 begin s1 = 1; s0 = 1; MSB_in = 1;  LSB_in = 1; end    // Repeat with MSB and LSB
    #150 s1 = 0;
    #190 begin s1 = 1; s0 = 1;  end     // reload with A = 1010
    #200 s0 = 0;               // Shift left
    #220 s1 = 0;               // Pause
      #240  s1 = 1;            // Shift left
   join
endmodule
```


All rights reserved.

(f)

```
module Shift_Register_BEH
  (output [3: 0] A_par, input [3: 0] I_par, input MSB_in, LSB_in, s1, s0, CLK, Clear);

  always @ (posedge CLK, negedge Clear) if (Clear == 0) A_par <= 4'b0;
    else case ({s1, s0})
      2'b11:    A_par <= I_par;
      2'b01:    A_par <= {MSB_in, A_par[3: 1]};
      2'b10:    A_par <= {A_par[2: 0], LSB_in};
      2'b00:    A_par <=A_par;
    endcase
endmodule

module t_Shift_Register ( );
  wire [3: 0] A_par;
  reg   [3: 0] I_par;
  reg MSB_in, LSB_in, s1, s0, CLK, Clear;

  Shift_Register_BEH M_SR( A_par, I_par, MSB_in, LSB_in, s1, s0, CLK, Clear);
  initial #300 $finish;
  initial begin CLK = 0; forever #5 CLK = ~CLK; end
   initial fork
    MSB_in = 0; LSB_in = 0;
    Clear = 0;              // Active-low reset
    s1  = 0; s0 = 0;            // No change
    #10 Clear = 1;
    #10 I_par = 4'hA;
    #30 begin s1 = 1; s0 = 1; end // 00: load I_par into A_par
    #50 s1 = 0;                   // 01: shift right (1010  to 0101 to 0010 to 0001 to 0000)
    #90 begin s1 = 1; s0 = 1; end // 11: reload A with 1010
    #100 s0 = 0;                  // 10: shift left (1010 to 0100 to 1000 to 000)
    #140 begin s1 = 1; s0 = 1; MSB_in = 1;  LSB_in = 1; end    // Repeat with MSB and LSB
    #150 s1 = 0;
    #190 begin s1 = 1; s0 = 1;  end     // reload with A = 1010
    #200 s0 = 0;                // Shift left
    #220 s1 = 0;                // Pause
      #240  s1 = 1;             // Shift left
   join
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

(g)

```
module Ripple_Counter_4bit_a (output [3: 0] A,  input Count, reset_b);
  reg A0, A1, A2, A3;
  assign A = {A3, A2, A1, A0};
  always @ (negedge Count, negedge reset_b)
    if (reset_b == 0) A0 <= 0; else if (T) A0 <= ~A0;

  always @ (negedge A0, negedge reset_b)
    if (reset_b == 0) A1 <= 0; else if (T) A1 <= ~A1;

  always @ (negedge A1, negedge reset_b)
    if (reset_b == 0) A2 <= 0; else if (T) A2 <= ~A2;

  always @ (negedge A2, negedge reset_b)
    if (reset_b == 0) A3 <= 0; else if (T) A3 <= ~A3;
endmodule

module t_Ripple_Counter_4bit ();
  wire [3: 0] A;
  reg Count, reset_b;

  Ripple_Counter_4bit_a M0 (A,  Count, reset_b);

  initial #300 $finish;
   initial fork
       reset_b  = 0;            // Active-low reset
    #60 reset_b = 1;

    Count = 1;
    #15 Count = 0;
    #30 Count = 1;
    #85 begin  Count = 0;  forever #10 Count = ~Count; end
  join
endmodule
```


All rights reserved.

```
module Ripple_Counter_4bit_b (output [3: 0] A,  input Count, reset_b);
  reg A0, A1, A2, A3;
  assign A = {A3, A2, A1, A0};
  always @ (negedge Count, negedge reset_b)
    if (reset_b == 0) A0 <= 0; else A0 <= ~A0;

  always @ (negedge A0, negedge reset_b)
    if (reset_b == 0) A1 <= 0; else A1 <= ~A1;

  always @ (negedge A1, negedge reset_b)
    if (reset_b == 0) A2 <= 0; else A2 <= ~A2;

  always @ (negedge A2, negedge reset_b)
    if (reset_b == 0) A3 <= 0; else A3 <= ~A3;
endmodule

module t_Ripple_Counter_4bit ();
  wire [3: 0] A;
  reg Count, reset_b;

  Ripple_Counter_4bit_b M0 (A,  Count, reset_b);

  initial #300 $finish;
   initial fork
       reset_b  = 0;            // Active-low reset
    #60 reset_b = 1;

    Count = 1;
    #15 Count = 0;
    #30 Count = 1;
    #85 begin  Count = 0;  forever #10 Count = ~Count; end
  join
endmodule
```

(h) Note: This version of the solution situates the data shift registers in the test bench.

```
module Serial_Subtractor (output SO, input SI_A, SI_B, shift_control, clock, reset_b);
// See Fig. 6.5 and Problem 6.9a (2s complement serial subtractor)
  reg [1: 0] sum;
  wire mem = sum[1];
  assign SO = sum[0];

   always @ (posedge clock, negedge reset_b)
    if (reset_b == 0) begin
      sum <= 2'b10;
    end
    else if (shift_control) begin
      sum <= SI_A + (!SI_B) + sum[1];
```


All rights reserved.

```
    end
endmodule

module t_Serial_Subtractor ();
  wire SI_A, SI_B;
  reg shift_control, clock, reset_b;

  Serial_Subtractor M0 (SO, SI_A, SI_B, shift_control, clock, reset_b);

  initial #250 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial fork
    shift_control = 0;
    #10 reset_b = 0;
    #20 reset_b = 1;
    #22 shift_control = 1;
    #105 shift_control = 0;
    #112 reset_b = 0;
    #114 reset_b = 1;
    #122 shift_control = 1;
    #205 shift_control = 0;
  join
  reg [7: 0] A, B, SO_reg;
  wire s7;
  assign s7 = SO_reg[7];
  assign SI_A = A[0];
  assign SI_B = B[0];
  wire SI_B_bar = ~SI_B;
  initial fork
    A = 8'h5A;
    B = 8'h0A;
    #122 A = 8'h0A;
    #122 B = 8'h5A;
  join

  always @ (negedge clock, negedge reset_b)
    if (reset_b == 0) SO_reg <= 0;
    else if (shift_control == 1) begin
      SO_reg <= {SO, SO_reg[7: 1]};
      A <= A >> 1;
      B <= B >> 1;
    end
  wire negative = !M0.sum[1];
  wire [7: 0] magnitude = (!negative)? SO_reg: 1'b1 + ~SO_reg;
endmodule
```

Simulation results are shown for 5Ah – 0Ah = 50h = 80 d and 0Ah – 5Ah = -80. The magnitude of the result is also shown.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

(i) See Prob. 6.35h.

(j)

```
module Serial_Twos_Comp (output y, input [7: 0] data, input load, shift_control, Clock, reset_b);
  reg [7: 0] SReg;
  reg Q;
  wire SO = SReg [0];
  assign y = SO ^ Q;
  always @ (posedge Clock, negedge reset_b)
  if (reset_b == 0) begin
    SReg <= 0;
    Q <= 0;
  end
  else begin
    if (load) SReg = data;
    else if (shift_control) begin
      Q <= Q | SO;
      SReg <= {y, SReg[7: 1]};
    end
  end
endmodule

module t_Serial_Twos_Comp ();
  wire y;
  reg [7: 0] data;
  reg load, shift_control, Clock, reset_b;

  Serial_Twos_Comp M0 (y, data, load, shift_control, Clock, reset_b);

  reg [7: 0] twos_comp;

  always @ (posedge Clock, negedge reset_b)
  if (reset_b == 0) twos_comp <= 0;
  else if (shift_control && !load) twos_comp <= {y, twos_comp[7: 1]};

  initial #200 $finish;
  initial begin Clock = 0; forever #5 Clock = ~Clock; end
  initial begin #2 reset_b = 0; #4 reset_b = 1; end

  initial fork
    data = 8'h5A;
    #20 load = 1;
    #30 load = 0;
    #50 shift_control = 1;
    #50 begin repeat (9) @ (posedge Clock) ;
      shift_control = 0;
    end
  join
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

(k) From the solution to Problem 6.13:

```
module Prob_6_35k_BCD_Counter (output A1, A2, A3, A4, input clk, reset_b);
  wire {A1, A2, A3, A4} = A;
  nand (Clear, A2, A4);
   Ripple_Counter_4bit M0 (A, Clear, reset_b);
endmodule

module Ripple_Counter_4bit (output [3: 0] A,  input Count, reset_b);
  reg A0, A1, A2, A3;
  assign A = {A3, A2, A1, A0};
  always @ (negedge Count, negedge reset_b)
    if (reset_b == 0) A0 <= 0; else A0 <= ~A0;
   always @ (negedge A0, negedge reset_b)
    if (reset_b == 0) A1 <= 0; else A1 <= ~A1;
   always @ (negedge A1, negedge reset_b)
    if (reset_b == 0) A2 <= 0; else A2 <= ~A2;
  always @ (negedge A2, negedge reset_b)
    if (reset_b == 0) A3 <= 0; else A3 <= ~A3;
endmodule
module t_ Prob_6_35k_BCD_Counter ();
  wire [3: 0] A;
  reg Count, reset_b;

  Prob_6_35k_BCD_Counter M0 (A1, A2, A3, A4, reset_b);
  initial #300 $finish;
   initial fork
       reset_b  = 0;            // Active-low reset
    #60 reset_b = 1;
/*
    Count = 1;
    #15 Count = 0;
    #30 Count = 1;
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
   #85 begin  Count = 0;  forever #10 Count = ~Count; end*/
  join
endmodule
```

(l)

```
module Prob_6_35l_Up_Dwn_Beh (output reg [3: 0] A, input CLK, Up, Down, reset_b);

  always @ (posedge CLK, negedge reset_b)
    if (reset_b ==0) A <= 4'b0000;
    else case ({Up, Down})
      2'b10:  A <= A + 4'b0001;    // Up
      2'b01:  A <= A - 4'b0001;     // Down
      default:    A <= A;     // Suspend (Redundant statement)
    endcase
 endmodule

module t_Prob_6_35l_Up_Dwn_Beh ();
  wire [3: 0] A;
  reg CLK, Up, Down, reset_b;

  Prob_6_35l_Up_Dwn_Beh M0 (A, CLK, Up, Down, reset_b);

  initial #300 $finish;
  initial begin CLK = 0; forever #5 CLK = ~CLK; end
  initial fork
      Down = 0; Up= 0;
    #10 reset_b = 0;
    #20 reset_b = 1;
    #40 Up = 1;
    #150 Down = 1;
    #220 Up = 0;
    #280 Down = 0;
  join
endmodule
```

6.36 (a)

```
// See Fig. 6.13., 4-bit Up-Down Binary Counter
module Prob_6_36_Up_Dwn_Beh (output reg [3: 0] A, input CLK, Up, Down, reset_b);

  always @ (posedge CLK, negedge reset_b)
    if (reset_b ==0) A <= 4'b0000;
    else if (Up) A <= A + 4'b0001;
    else if (Down) A <= A - 4'b0001;
 endmodule

module t_Prob_6_36_Up_Dwn_Beh ();
  wire [3: 0] A;
  reg CLK, Up, Down, reset_b;
```


All rights reserved.

```
  Prob_6_36_Up_Dwn_Beh M0 (A, CLK, Up, Down, reset_b);

  initial #300 $finish;
  initial begin CLK = 0; forever #5 CLK = ~CLK; end
  initial fork
      Down = 0; Up= 0;
    #10 reset_b = 0;
    #20 reset_b = 1;
    #40 Up = 1;
    #150 Down = 1;
    #220 Up = 0;
    #280 Down = 0;
  join
endmodule
```

(b)

```
module Prob_6_36_Up_Dwn_Str (output [3: 0] A, input CLK, Up, Down, reset_b);
  wire Down_3, Up_3, Down_2, Up_2, Down_1, Up_1;
  wire A_0b, A_1b, A_2b, A_3b;


  stage_register SR3 (A[3], A_3b, Down_3, Up_3, Down_2, Up_2, A[2], A_2b, CLK, reset_b);
  stage_register SR2 (A[2], A_2b, Down_2, Up_2, Down_1, Up_1, A[1], A_1b, CLK, reset_b);
  stage_register SR1 (A[1], A_1b, Down_1, Up_1, Down_not_Up, Up, A[0], A_0b, CLK, reset_b);
  not (Up_b, Up);
  and (Down_not_Up, Down, Up_b);
  or (T, Up, Down_not_Up);
  Toggle_flop TF0 (A[0], A_0b, T, CLK, reset_b);
endmodule

module stage_register (output A, A_b, Down_not_Up_out, Up_out, input Down_not_Up, Up, A_in,
A_in_b, CLK, reset_b);

  Toggle_flop T0 (A, A_b, T, CLK, reset_b);
  or (T, Down_not_Up_out, Up_out);
  and (Down_not_Up_out, Down_not_Up, A_in_b);
  and (Up_out, Up, A_in);
endmodule

module Toggle_flop (output reg Q, output Q_b, input T, CLK, reset_b);
  always @ (posedge CLK, negedge reset_b) if (reset_b == 0) Q <= 0; else Q <= Q ^ T;
  assign Q_b = ~Q;

endmodule

module t_Prob_6_36_Up_Dwn_Str ();
  wire [3: 0] A;
  reg CLK, Up, Down, reset_b;
  wire T3 = M0.SR3.T;
  wire T2 = M0.SR2.T;
```


All rights reserved.

```
    wire T1 = M0.SR1.T;
    wire T0 = M0.T;

    Prob_6_36_Up_Dwn_Str M0 (A, CLK, Up, Down, reset_b);

    initial #150 $finish;
    initial begin CLK = 0; forever #5 CLK = ~CLK; end
    initial fork
      Down = 0; Up= 0;
      #10 reset_b = 0;
      #20 reset_b = 1;
      #50 Up = 1;
      #140 Down = 1;
      #120 Up = 0;
      #140 Down = 0;
    join
  endmodule
```

6.37

```
module Counter_if (output reg [3: 0] Count, input clock, reset);
  always @ (posedge clock , posedge reset)
    if (reset)Count <= 0;
    else if (Count == 0) Count <= 1;
    else if (Count == 1) Count <= 3;     // Default interpretation is decimal
    else if (Count == 3) Count <= 7;
    else if (Count == 4) Count <= 0;
    else if (Count == 6) Count <= 4;
    else if (Count == 7) Count <= 6;
    else Count <= 0;
endmodule

module Counter_case (output reg [3: 0] Count, input clock, reset);
  always @ (posedge clock , posedge reset)
    if (reset)Count <= 0;
    else begin
      Count <= 0;
      case (Count)
        0:        Count <= 1;
        1:        Count <= 3;
        3:        Count <= 7;
        4:        Count <= 0;
        6:        Count <= 4;
        7:        Count <= 6;
        default:     Count <= 0;
      endcase
  end
endmodule
```


All rights reserved.

```
module Counter_FSM (output reg [3: 0] Count, input clock, reset);
  reg [2: 0] state, next_state;
  parameter s0 = 0, s1 = 1, s2 = 2, s3 = 3, s4 = 4, s5 = 5, s6 = 6, s7 = 7;

  always @ (posedge clock ,  posedge reset)
    if (reset) state <= s0; else state <= next_state;

  always @ (state) begin
    Count = 0;
    case (state)
       s0:      begin next_state = s1; Count = 0; end
       s1:      begin next_state = s2; Count = 1; end
       s2:      begin next_state = s3; Count = 3; end
       s3:      begin next_state = s4; Count = 7; end
       s4:      begin next_state = s5; Count = 6; end
       s5:      begin next_state = s6; Count = 4; end
       default:    begin next_state = s0; Count = 0; end
     endcase
  end
endmodule
```

6.38 **(a)**

```
module Prob_6_38a_Updown (OUT, Up, Down, Load, IN, CLK);   // Verilog 1995
  output [3: 0]   OUT;
  input   [3: 0]   IN;
  input            Up, Down, Load, CLK;
  reg [3:0]    OUT;

  always @ (posedge CLK)
  if (Load) OUT <= IN;
  else if (Up)      OUT <= OUT + 4'b0001;
  else if (Down) OUT <= OUT -  4'b0001;
  else          OUT <= OUT;
  endmodule

module updown (                        // Verilog 2001, 2005
  output reg [3: 0]   OUT,
  input          [3: 0]   IN,
  input              Up, Down, Load, CLK
);
```

(b)

```
module Prob_6_38b_Updown (output reg [3: 0] OUT, input [3: 0] IN, input s1, s0, CLK);

  always @ (posedge CLK)
```


All rights reserved.

```
 case ({s1, s0})
   2'b00:  OUT <= OUT + 4'b0001;
   2'b01:  OUT <= OUT - 4'b0001;
   2'b10:  OUT <= IN;
   2'b11:  OUT <= OUT;
 endcase
 endmodule

module t_Prob_6_38b_Updown ();
 wire [3: 0] OUT;
 reg [3: 0] IN;
 reg s1, s0, CLK;
 Prob_6_38b_Updown M0 (OUT, IN, s1, s0, CLK);

 initial #150 $finish;
 initial begin CLK = 0; forever #5 CLK = ~CLK; end
  initial fork
  IN = 4'b1010;

  #10   begin s1 = 1; s0 = 0; end      // Load IN
  #20   begin s1 = 1; s0 = 1; end      // no change
  #40   begin s1 = 0; s0 = 0; end      // UP;
  #80   begin s1 = 0; s0 = 1; end      // DOWN
  #120 begin s1 = 1; s0 = 1; end
 join
endmodule
```

6.39

```
module Prob_6_39_Counter_BEH (output reg [2: 0] Count, input Clock, reset_b);
 always @ (posedge Clock, negedge reset_b)  if (reset_b == 0) Count <= 0;
   else case (Count)
     0:  Count <= 1;
     1:  Count <= 2;
     2:  Count <= 4;
     4:  Count <= 5;
     5:  Count <= 6;
     6:  Count <= 0;
   endcase
endmodule

module Prob_6_39_Counter_STR (output [2: 0] Count, input Clock, reset_b);
 supply1 PWR;
 wire Count_1_b = ~Count[1];

 JK_FF M2 (Count[2],    Count[1],   Count[1],   Clock, reset_b);
 JK_FF M1 (Count[1],    Count[0],   PWR,        Clock, reset_b);
 JK_FF M0 (Count[0],    Count_1_b,          PWR,        Clock, reset_b);
endmodule
```


All rights reserved.

```
module JK_FF (output reg Q, input J, K, clk, reset_b);
  always @ (posedge clk, negedge 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_6_39_Counter ();
  wire [2: 0] Count_BEH, Count_STR;
  reg Clock, reset_b;

  Prob_6_39_Counter_BEH M0_BEH (Count_STR, Clock, reset_b);
  Prob_6_39_Counter_STR M0_STR (Count_BEH, Clock, reset_b);

  initial #250 $finish;
  initial fork #1 reset_b = 0; #7 reset_b = 1; join
  initial begin  Clock = 1; forever #5 Clock = ~Clock; end
endmodule
```

6.40

```
module Prob_6_40 (output reg [0: 7] timer, input clk, reset_b);

  always @ (negedge clk, negedge reset_b)
    if (reset_b  == 0) timer <= 8'b1000_0000; else
    case (timer)

      8'b1000_0000:     timer <= 8'b0100_0000;
      8'b0100_0000:     timer <= 8'b0010_0000;
      8'b0010_0000:     timer <= 8'b0001_0000;
      8'b0001_0000:     timer <= 8'b0000_1000;
      8'b0000_1000:     timer <= 8'b0000_0100;
      8'b0000_0100:     timer <= 8'b0000_0010;
      8'b0000_0010:     timer <= 8'b0000_0001;
      8'b0000_0001:     timer <= 8'b1000_0000;
      default:          timer <= 8'b1000_0000;
    endcase
endmodule

module t_Prob_6_40 ();
  wire [0: 7] timer;
  reg clk, reset_b;

  Prob_6_40 M0 (timer, clk, reset_b);

  initial #250 $finish;
  initial fork #1 reset_b = 0; #7 reset_b = 1; join
  initial begin  clk = 1; forever #5 clk = ~clk; end
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
endmodule
```

6.41

```
module Prob_6_41_Switched_Tail_Johnson_Counter (output [0: 3] Count, input CLK, reset_b);
  wire Q_0b, Q_1b, Q_2b, Q_3b;

  DFF M3 (Count[3], Q_3b, Count[2], CLK, reset_b);
  DFF M2 (Count[2], Q_2b, Count[1], CLK, reset_b);
  DFF M1 (Count[1], Q_1b, Count[0], CLK, reset_b);
  DFF M0 (Count[0], Q_0b, Q_3b, CLK, reset_b);
endmodule

module DFF (output reg Q, output Q_b,  input D, clk, reset_b);
  assign Q_b = ~Q;
  always @ (posedge clk, negedge reset_b) if (reset_b ==0) Q <= 0; else Q <= D;
endmodule

module t_Prob_6_41_Switched_Tail_Johnson_Counter ();
  wire [3: 0] Count;
  reg CLK, reset_b;
  wire s0 = ~ M0.Count[0] && ~M0.Count[3];
  wire s1 =    M0.Count[0] && ~M0.Count[1];
  wire s2 =    M0.Count[1] && ~M0.Count[2];
  wire s3 =    M0.Count[2] && ~M0.Count[3];
  wire s4 =    M0.Count[0] &&   M0.Count[3];
  wire s5 = ~ M0.Count[0] &&   M0.Count[1];
  wire s6 = ~ M0.Count[1] &&   M0.Count[2];
  wire s7 = ~ M0.Count[2] &&  M0.Count[3];

  Prob_6_41_Switched_Tail_Johnson_Counter M0 (Count, CLK, reset_b);
  initial #150 $finish;
  initial fork #1 reset_b = 0; #7 reset_b = 1; join
  initial begin  CLK = 1; forever #5 CLK = ~CLK; end
endmodule
```


All rights reserved.

6.42 Because A is a register variable, it retains whatever value has been assigned to it until a new value is assigned. Therefore, the statement $A <= A$ has the same effect as if the statement was omitted.

6.43

```
module Prob_6_43_Str (output SO, input [7: 0] data, input load, Shift_control, Clock, reset_b);
  supply0 gnd;
  wire SO_A;

  Shift_with_Load M_A (SO_A, SO_A, data, load, Shift_control, Clock, reset_b);
  Shift_with_Load M_B (SO,  SO_A, data, gnd, Shift_control, Clock, reset_b);

endmodule


module Shift_with_Load (output SO, input D_in, input [7: 0] data, input load, select, Clock, reset_b);
  wire [7: 0] Q;
  assign SO = Q[0];
  SR_cell M7 (Q[7], D_in, data[7], load, select, Clock, reset_b);
  SR_cell M6 (Q[6], Q[7], data[6], load, select, Clock, reset_b);
  SR_cell M5 (Q[5], Q[6], data[5], load, select, Clock, reset_b);
  SR_cell M4 (Q[4], Q[5], data[4], load, select, Clock, reset_b);
  SR_cell M3 (Q[3], Q[4], data[3], load, select, Clock, reset_b);
  SR_cell M2 (Q[2], Q[3], data[2], load, select, Clock, reset_b);
  SR_cell M1 (Q[1], Q[2], data[1], load, select, Clock, reset_b);
  SR_cell M0 (Q[0], Q[1], data[0], load, select, Clock, reset_b);
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
module SR_cell (output Q, input D, data, load, select, Clock, reset_b);
  wire y;
  DFF_with_load M0 (Q, y, data, load, Clock, reset_b);
  Mux_2 M1 (y, Q,  D, select);
endmodule
module DFF_with_load (output reg Q, input D, data, load, Clock, reset_b);
  always @ (posedge Clock, negedge reset_b)
    if (reset_b == 0) Q <= 0; else if (load) Q <= data; else Q <= D;
endmodule

module Mux_2 (output reg y, input a, b, sel);
  always @ (a, b, sel) if (sel ==1) y = b; else y = a;
endmodule

module t_Fig_6_4_Str ();
  wire SO;
  reg load, Shift_control, Clock, reset_b;
  reg [7: 0] data, Serial_Data;

  Prob_6_43_Str M0 (SO, data, load, Shift_control, Clock, reset_b);

  always @ (posedge Clock, negedge reset_b)
  if (reset_b == 0) Serial_Data <= 0;
  else if (Shift_control ) Serial_Data <= {M0.SO_A, Serial_Data [7: 1]};

  initial #200 $finish;
  initial begin Clock = 0; forever #5 Clock = ~Clock; end
  initial begin #2 reset_b = 0; #4 reset_b = 1; end

  initial fork
    data = 8'h5A;
    #20 load = 1;
    #30 load = 0;
    #50 Shift_control = 1;
    #50 begin repeat (9) @ (posedge Clock) ;
      Shift_control = 0;
    end
  join
endmodule
```

Alternative: a behavioral model for synthesis is given below. The behavioral description implies the need for a mux at the input to a D-type flip-flop.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
module Fig_6_4_Beh (output SO, input [7: 0] data, input load, Shift_control, Clock, reset_b);
  reg [7: 0] Shift_Reg_A, Shift_Reg_B;
  assign SO = Shift_Reg_B[0];
  always @ (posedge Clock, negedge reset_b)
    if (reset_b == 0) begin
      Shift_Reg_A <= 0;
      Shift_Reg_B <= 0;
    end
    else if (load) Shift_Reg_A <= data;
    else if (Shift_control) begin
      Shift_Reg_A <= { Shift_Reg_A[0], Shift_Reg_A[7: 1]};
      Shift_Reg_B <= {Shift_Reg_A[0], Shift_Reg_B[7: 1]};
    end

endmodule
module t_Fig_6_4_Beh ();
  wire SO;
  reg load, Shift_control, Clock, reset_b;
  reg [7: 0] data, Serial_Data;

  Fig_6_4_Beh M0 (SO, data, load, Shift_control, Clock, reset_b);

  always @ (posedge Clock, negedge reset_b)
  if (reset_b == 0) Serial_Data <= 0;
  else if (Shift_control ) Serial_Data <= {M0.Shift_Reg_A[0], Serial_Data [7: 1]};

  initial #200 $finish;
  initial begin Clock = 0; forever #5 Clock = ~Clock; end
  initial begin #2 reset_b = 0; #4 reset_b = 1; end

  initial fork
    data = 8'h5A;
    #20 load = 1;
    #30 load = 0;
    #50 Shift_control = 1;
    #50 begin repeat (9) @ (posedge Clock) ;
      Shift_control = 0;
    end
  join
endmodule
```

6.44

```
// See Figure 6.5
// Note: Sum is stored in shift register A; carry is stored in Q
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
// Note: Clear is active-low.

module Prob_6_44_Str (output SO, input [7: 0] data_A, data_B, input S_in, load, Shift_control, CLK,
        Clear);
  supply0 gnd;
  wire sum, carry;
  assign SO = sum;
  wire SO_A, SO_B;

  Shift_Reg_gated_clock M_A (SO_A, sum, data_A, load, Shift_control, CLK, Clear);
  Shift_Reg_gated_clock M_B (SO_B,  S_in, data_B, load, Shift_control, CLK, Clear);
  FA M_FA (carry, sum, SO_A, SO_B, Q);
  DFF_gated M_FF (Q, carry, Shift_control, CLK, Clear);

endmodule

module Shift_Reg_gated_clock (output SO, input S_in, input [7: 0] data, input load, Shift_control,
        Clock, reset_b);
  reg [7: 0] SReg;
  assign SO = SReg[0];

  always @ (posedge Clock, negedge reset_b)
    if (reset_b == 0) SReg <= 0;
    else if (load) SReg <= data;
    else if (Shift_control) SReg <= {S_in, SReg[7: 1]};
endmodule

module DFF_gated (output Q, input D, Shift_control, Clock, reset_b);
  DFF M_DFF (Q,   D_internal, Clock, reset_b);
  Mux_2 M_Mux (D_internal, Q, D, Shift_control);
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

module Mux_2 (output reg y, input a, b, sel);
  always @ (a, b, sel) if (sel ==1) y = b; else y = a;
 endmodule

module FA (output reg carry, sum, input a, b, C_in);
  always @ (a, b, C_in) {carry, sum} = a + b + C_in;
endmodule

module t_Prob_6_44_Str ();
  wire SO;
  reg SI, load, Shift_control, Clock, Clear;
  reg [7: 0] data_A, data_B;

  Prob_6_44_Str M0 (SO, data_A, data_B, SI, load, Shift_control, Clock, Clear);

  initial #200 $finish;
  initial begin Clock = 0; forever #5 Clock = ~Clock; end
  initial begin #2 Clear = 0; #4 Clear = 1; end

  initial fork
    data_A =  8'hAA;    //8'h ff;
    data_B = 8'h55;     //8'h01;
    SI = 0;
    #20 load = 1;
    #30 load = 0;
```


All rights reserved.

```
    #50 Shift_control = 1;
    #50 begin repeat (8) @ (posedge Clock) ;
      #5 Shift_control = 0;
    end
  join
endmodule
```

6.45

```
module Prob_6_45 (output reg y_out, input start, clock, reset_bar);
 parameter      s0 = 4'b0000,
      s1 = 4'b0001,
      s2 = 4'b0010,
      s3 = 4'b0011,
      s4 = 4'b0100,
      s5 = 4'b0101,
      s6 = 4'b0110,
      s7 = 4'b0111,
      s8 = 4'b1000;
 reg [3: 0]   state, next_state;

 always @ (posedge clock, negedge reset_bar)
```


All rights reserved.

```
    if (!reset_bar) state <= s0; else state <= next_state;

  always @ (state, start) begin
    y_out = 1'b0;
    case (state)
      s0:   if (start) next_state = s1; else next_state = s0;
      s1:   begin next_state = s2; y_out = 1; end
      s2:   begin next_state = s3; y_out = 1; end
      s3:   begin next_state = s4; y_out = 1; end
      s4:   begin next_state = s5; y_out = 1; end
      s5:   begin next_state = s6; y_out = 1; end
      s6:   begin next_state = s7; y_out = 1; end
      s7:   begin next_state = s8; y_out = 1; end
      s8:   begin next_state = s0; y_out = 1; end
      default: next_state = s0;
    endcase
  end
endmodule


// Test plan

// Verify the following:
// Power-up reset
// Response to start in initial state
// Reset on-the-fly
// Response to re-assertion of start after reset on-the-fly
// 8-cycle counting sequence
// Ignore start during counting sequence
// Return to initial state after 8 cycles and await start
// Remain in initial state for one clock if start is asserted when the state is entered

module t_Prob_6_45;
  wire     y_out;
  reg start, clock, reset_bar;

  Prob_6_45 M0 (y_out, start, clock, reset_bar);

  initial #300 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial fork
    reset_bar = 0;
    #2 reset_bar = 1;
    #10 start = 1;
    #20 start = 0;
    #30 reset_bar = 0;
    #50 reset_bar = 1;
    #80 start = 1;
    #90 start = 0;
    #130 start = 1;
    #140 start = 0;
    #180 start = 1;
  join
endmodule
```


All rights reserved.

6.46

```
module Prob_6_46 (output reg [0: 3] timer, input clk, reset_b);

  always @ (negedge clk, negedge reset_b)
    if (reset_b  == 0) timer <= 4'b1000; else
    case (timer)
      4'b1000:   timer <= 4'b0100;
      4'b0100:   timer <= 4'b0010;
      4'b0010:   timer <= 4'b0001;
      4'b0001:   timer <= 4'b1000;
      default:   timer <= 4'b1000;
    endcase
endmodule

module t_Prob_6_46 ();
  wire [0: 3] timer;
  reg clk, reset_b;

  Prob_6_46 M0 (timer, clk, reset_b);

  initial #150 $finish;
  initial fork #1 reset_b = 0; #7 reset_b = 1; join
  initial begin  clk = 1; forever #5 clk = ~clk; end
endmodule
```

6.47

```
module Prob_6_47 (
  output reg P_odd,
  input D_in, CLK, reset
);
  wire D;

  assign D = D_in ^ P_odd;
  always @ (posedge CLK, posedge reset)
```


All rights reserved.

```
    if (reset)     P_odd <= 0;
    else      P_odd <= D;
endmodule

 module t_Prob_6_47 ();
  wire P_odd;
  reg D_in, CLK, reset;

  Prob_6_47 M0 (P_odd,  D_in, CLK, reset);

  initial #150 $finish;
  initial fork #1 reset = 1; #7 reset = 0; join
  initial begin  CLK = 0; forever #5 CLK = ~CLK; end
  initial begin D_in = 1; forever #20 D_in = ~D_in; end

endmodule
```

6.48 **(a)**

```
module Prob_6_48a (output reg [7: 0] count, input clk, reset_b);
  reg [3: 0] state;
  always @ (posedge clk, negedge reset_b)
    if (reset_b == 0) state <= 0; else state <= state + 1;
  always @ (state)
    case (state)
      0, 2, 4, 6, 8, 10, 12:   count = 8'b0000_0001;
      1:              count = 8'b0000_0010;
      3:              count = 8'b0000_0100;
      5:              count = 8'b0000_1000;
      7:              count = 8'b0001_0000;
      9:              count = 8'b0010_0000;
      11:             count = 8'b0100_0000;
      13:             count = 8'b1000_0000;
      default:        count = 8'b0000_0000;
    endcase
endmodule

module t_Prob_6_48a ();
  wire [7: 0] count;
  reg clk, reset_b;

  Prob_6_48a M0 (count, clk, reset_b);

  initial #200 $finish;
  initial begin clk = 0; forever #5 clk = ~clk; end
  initial begin reset_b = 0; #2 reset_b = 1; end
endmodule
```


All rights reserved.

(b)

```
module Prob_6_48b (output reg [7: 0] count, input clk, reset_b);
  reg [3: 0] state;
  always @ (posedge clk, negedge reset_b)
    if (reset_b == 0) state <= 0; else state <= state + 1;
  always @ (state)
    case (state)
      0, 2, 4, 6, 8, 10, 12:   count = 8'b1000_0000;
      1:              count = 8'b0100_0000;
      3:              count = 8'b0010_0000;
      5:              count = 8'b0001_0000;
      7:              count = 8'b0000_1000;
      9:              count = 8'b0000_0100;
      11:             count = 8'b0000_0010;
      13:             count = 8'b0000_0001;
      default:        count = 8'b0000_0000;
    endcase
endmodule

module t_Prob_6_48b ();
  wire [7: 0] count;
  reg clk, reset_b;

  Prob_6_48b M0 (count, clk, reset_b);

  initial #180 $finish;
  initial begin clk = 0; forever #5 clk = ~clk; end
  initial begin reset_b = 0; #2 reset_b = 1; end
endmodule
```


All rights reserved.

6.49

```
// Behavioral description of a 4-bit universal shift register
// Fig. 6.7 and Table 6.3
module Shift_Register_4_beh (          // V2001, 2005
  output reg   [3: 0]   A_par,          // Register output
  input        [3: 0]   I_par,          // Parallel input
  input                 s1, s0,         // Select inputs
               MSB_in, LSB_in,  // Serial inputs
               CLK, Clear          // Clock and Clear
);
  always @ (posedge CLK, negedge Clear)    // V2001, 2005
    if (~Clear) A_par <= 4'b0000;
    else
      case ({s1, s0})
        2'b00: A_par <= A_par;          // No change
        2'b01: A_par <= {MSB_in, A_par[3: 1]};          // Shift right
        2'b10: A_par <= {A_par[2: 0], LSB_in};          // Shift left
        2'b11: A_par <= I_par;          // Parallel load of input
      endcase
endmodule
```


All rights reserved.

```
// Test plan:
// test reset action load
// test parallel load
// test shift right
// test shift left
// test circulation of data
// test reset on the fly

module t_Shift_Register_4_beh ();
  reg    s1, s0,            // Select inputs
     MSB_in, LSB_in,       // Serial inputs
     clk, reset_b;         // Clock and Clear
  reg    [3: 0]   I_par;       // Parallel input
  wire   [3: 0]   A_par;       // Register output

  Shift_Register_4_beh M0 (A_par, I_par,s1, s0, MSB_in, LSB_in, clk, reset_b);

  initial #200 $finish;
  initial begin clk = 0; forever #5 clk = ~clk; end

  initial fork
    // test reset action load
    #3 reset_b = 1;
    #4 reset_b = 0;
    #9 reset_b = 1;

    // test parallel load
    #10 I_par = 4'hA;
    #10 {s1, s0} = 2'b11;

    // test shift right
    #30 MSB_in = 1'b0;
    #30 {s1, s0} = 2'b01;

    // test shift left
    #80 LSB_in = 1'b1;
    #80 {s1, s0} = 2'b10;

    // test circulation of data
   #130 {s1, s0} = 2'b11;
    #140 {s1, s0} = 2'b00;

   // test reset on the fly

    #150 reset_b = 1'b0;
    #160 reset_b = 1'b1;
    #160 {s1, s0} = 2'b11;

  join
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

6.50 **(a)** See problem 6.27.

```
module Prob_8_50a (output reg [2: 0] count, input clk, reset_b);
  always @ (posedge clk, negedge reset_b)
    if (!reset_b) count <= 0;
    else case (count)
      3'd0:    count <= 3'd1;
      3'd1:    count <= 3'd2;
      3'd2:    count <= 3'd3;
      3'd3:    count <= 3'd4;
      3'd4:    count <= 3'd5;
      3'd5:    count <= 3'd6;
      3'd4:    count <= 3'd6;
      3'd6:    count <= 3'd0;
      default:    count <= 3'd0;
    endcase
endmodule

module t_Prob_8_50a;
  wire [2: 0] count;
  reg clock, reset_b  ;

  Prob_8_50a M0 (count, clock, reset_b);

  initial #130 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial fork
       reset_b = 0;
    #2 reset_b = 1;
    #40 reset_b = 0;
    #42 reset_b = 1;
  join
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

(b) See problem 6.28.

```
module Prob_8_50b (output reg [2: 0] count, input clk, reset_b);
  always @ (posedge clk, negedge reset_b)
    if (!reset_b) count <= 0;
    else case (count)
      3'd0:    count <= 3'd1;
      3'd1:    count <= 3'd2;
      3'd2:    count <= 3'd4;
      3'd4:    count <= 3'd6;
      3'd6:    count <= 3'd0;
      default:    count <= 3'd0;
    endcase
endmodule

module t_Prob_8_50b;
  wire [2: 0] count;
  reg clock, reset_b  ;

  Prob_8_50b M0 (count, clock, reset_b);

  initial #100 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial fork
        reset_b = 0;
    #2 reset_b = 1;
    #40 reset_b = 0;
    #42 reset_b = 1;
  join
endmodule
```

6.51

```
module Seq_Detector_Prob_5_51 (output detect, input bit_in, clk, reset_b);
  reg [2: 0] sample_reg;
  assign detect = (sample_reg == 3'b111);
  always @ (posedge clk, negedge reset_b) if (reset_b ==0) sample_reg <= 0;
    else sample_reg <= {bit_in, sample_reg [2: 1]};
endmodule

module Seq_Detector_Prob_5_45 (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;
```


All rights reserved.

```
      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_Seq_Detector_Prob_6_51 ();
  wire detect_45, detect_51;
  reg bit_in, clk, reset_b;

  Seq_Detector_Prob_5_51 M0 (detect_51, bit_in, clk, reset_b);
  Seq_Detector_Prob_5_45 M1 (detect_45, bit_in, clk, reset_b);

  initial #350$finish;
  initial begin clk = 0; forever #5 clk = ~clk; end
  initial fork

        reset_b = 0;
    #4  reset_b = 1;
    #10 bit_in = 1;
    #20 bit_in = 0;
    #30 bit_in = 1;
    #50 bit_in = 0;
    #60 bit_in = 1;
    #100 bit_in = 0;
  join
endmodule
```

The circuit using a shift register uses less hardware.

6.52 Universal Shift Register

```
module Prob_6_52 (
  output [3:0] A_par,
  input  [3: 0] In_par,
  input MSB_in, LSB_in,
  input [1: 0] s1, s0,
  input CLK, Clear_b
);
  wire y0, y1, y2, y3;
  Mux_4x1 M0 (y0, In_par[0],  LSB_in, A_par[1], A_par[0], s1, s0);
  Mux_4x1 M1 (y1, In_par[1], A_par[0], A_par[2], A_par[1], s1, s0);
  Mux_4x1 M2 (y2, In_par[2], A_par[1], A_par[3], A_par[2], s1, s0);
  Mux_4x1 M3 (y3, In_par[3], A_par[2], MSB_in, A_par[3], s1, s0);
  DFF D0 (A_par[0], y0, CLK, Clear_b);
  DFF D1 (A_par[1], y1, CLK, Clear_b);
  DFF D2 (A_par[2], y2, CLK, Clear_b);
  DFF D3 (A_par[3], y3, CLK, Clear_b);
endmodule
```


All rights reserved.

```
module Mux_4x1 (output reg y, input in3, in2, in1, in0, s1, s0);
  always @ (in3, in2, in1, in0, s1, s0)
    case ({s1, s0})
      2'b00:  y = in0;
      2'b01:  y = in1;
      2'b10:  y = in2;
      2'b11:  y = in3;
    endcase
endmodule

module DFF (output reg q, input d, clk, clr_b);
  always @ (posedge clk, negedge clr_b)
    if (clr_b == 1'b0) q <= 0; else q <= d;
endmodule
```

Features to be tested:

- Action of Clear_b
  - Power-up initialization
  - On-the-fly
- Action of mode controls

s1	s0	
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load

```
module t_Problem_6_52 ();
  wire [3:0]     A_par;
  reg  [3: 0]    In_par;
  reg            MSB_in, LSB_in;
  reg            s1, s0;
  reg            CLK, Clear_b;
  reg [3:0]      In_par;

  Prob_6_52 M0 (A_par, In_par, MSB_in, LSB_in, s1, s0, CLK, Clear_b);
  initial #300 $finish;
  initial begin CLK = 0, forever #5 CLK = ~CLK; end
  initial fork
    Clear_b = 0;                        // Power-up initialization
    #20  Clear_b = 1;                   // Running

    In_par = 4'b1010;                   // Word for parallel load
    MSB_in = 1'b1;                      // Bit for serial load
    LSB_in = 1'b0;                      // Bit for serial load
    s1 = 0; s0 = 0;                     // Initial action to no change
    #40  begin s1 = 1; s0 = 1; end          // parallel load
    #60 Clear_b = 1'b0;                 // Reset on-the-fly
    #80 Clear_b = 1'b1;                 // Resume action with parallel load at next clock edge
    #90 begin s1 = 0; s0 = 0; end       // No action – register holds 4'b1010

    #120 Clear_b = 1'b0;                // Clear register
    #130 Clear_b = 1'b1;
    #140 begin s1 = 1'b0; s0 = 1'b1; end      // Shifting to right (from MSB)
    #170 begin s1 = 1'b0; s0 = 1'b0; end      // Register should hold 4'b1111

    #190 begin Clear_b = 1'b0; s1 = 1'b0; s0 = 1'b0; end
    #200 begin Clear_b = 1'b1; s1 = 1'b1; s0 = 1'b0; end         // Resume action – shift left
    #230 begin s1 = 1'b0; s0 = 1'b0; end
  join
endmodule
```


All rights reserved.

6.53

```
module Prob_6_53 (output reg [3:0] SR_A, input Shift_control, SI, CLK, Clear_b);
  reg [3: 0] SR_B;
  wire Sum, Carry;
  wire SO_A = SR_A[3];
  wire SO_B = SR_B[3];
  wire SI_A = Sum;
  wire SI_B = SI;
  wire Q;

  always @ (posedge CLK)
    if (Clear_b == 1'b0) SR_A<= 4'b0; else if (Shift_control) SR_A <= {Sum, SR_A[3:1]};

  always @ (posedge CLK)
    if (Clear_b == 1'b0) SR_B <= 4'b0; else if (Shift_control) SR_B <= {SI, SR_B[3:1]};

  FA M0 (Sum, Carry, SO_A, SO_B, Q);
  and (clk_to_DFF, CLK, Shift_control);      // Caution: gated clock
  DFF M1 (Q, Carry, clk_to_DFF, Clear_b);
endmodule

module FA (output S, C, input x, y, z);
  assign {C, S} = x + y + z;
endmodule

module DFF (output reg Q, input D, C, Clear_b);
  always @ (posedge C) if (Clear_b == 1'b0) Q <= 1'b0; else Q <= D;
endmodule


module t_Prob_6_53 ();
  wire [3:0] SR_A;
  reg Shift_control, SI, CLK, Clear_b;
```


All rights reserved.

```
Prob_6_53 M0 (SR_A, Shift_control, SI, CLK, Clear_b);
initial #300 $finish;
initial begin CLK = 0; forever #5 CLK = ~CLK; end
initial fork
  Clear_b = 0;
 #20 Clear_b = 1;

 #40 Shift_control = 1;
 SI = 0;
// Sequence of 1s
/*
 #60 SI = 1;
 #70 SI = 0;
 #100 SI = 1;
 #110 SI = 0;
 #140 SI = 1;
 #150 SI = 0;
 #180 SI = 1;
 #190 SI = 0;
*/

// Sequence of threes
 #60 SI = 1;
 #80 SI = 0;
 #100 SI = 1;
 #120 SI = 0;
 #140 SI = 1;
 #160 SI = 0;
 #180 SI = 1;
 #200 SI = 0;
join
endmodule
```

Simulation results for accumulating a sequence of four 1s.

Simulation results for accumulating a sequence of four 3s.

All rights reserved.

Additional test patterns are left to the student.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

6.54

```
module Prob_6_54 (output reg [3:0] SR_A, input Shift_control, SI, CLK, Clear_b);
  reg [3:0] SR_B;
  wire S;
  wire Q;
  wire SI_A = S;
  wire SO_A = SR_A[0];
  wire SO_B = SR_B[0];
  wire SI_B = SI;
  and (J_in, SO_A, SO_B);
  nor (K_in, SO_A, SO_B);

  xor (S, SO_A, SO_B, Q);
  and (clk_to_JKFF, Shift_control, CLK);

  always @ (posedge CLK)
    if (Clear_b == 1'b0) SR_A<= 4'b0; else if (Shift_control) SR_A <= {SI_A, SR_A[3:1]};

  always @ (posedge CLK)
    if (Clear_b == 1'b0) SR_B <= 4'b0; else if (Shift_control) SR_B <= {SI_B, SR_B[3:1]};

  and (clk_to_JKFF, CLK, Shift_control);          // Caution: gated clock

  JKFF M1 (Q, J_in, K_in, clk_to_JKFF, Clear_b);
endmodule

module FA (output S, C, input x, y, z);
  assign {C, S} = x + y + z;
endmodule

module JKFF (output reg Q, input J_in, K_in, C, Clear_b);
  always @ (posedge C) if (Clear_b == 1'b0) Q <= 1'b0; else
    case ({J_in, K_in})
      2'b00:      Q <= Q;
      2'b01:      Q <= 1'b0;
      2'b10:      Q <= 1'b1;
      2'b11:      Q <= ~Q;
    endcase
endmodule

module t_Prob_6_54 ();
  wire [3:0] SR_A;
  reg Shift_control, SI, CLK, Clear_b;

  Prob_6_54 M0 (SR_A, Shift_control, SI, CLK, Clear_b);
  //initial #200 $finish;          // sequence of 1s
  initial #400 $finish;            // sequence of 3s
  initial begin CLK = 0; forever #5 CLK = ~CLK; end
  initial fork
     Clear_b = 0;
    #20 Clear_b = 1;

    #40 Shift_control = 1;
    SI = 0;
  // Sequence of 1s
  /*
    #60 SI = 1;
    #70 SI = 0;
    #100 SI = 1;
    #110 SI = 0;
    #140 SI = 1;
    #150 SI = 0;
```


All rights reserved.

```
    #180 SI = 1;
    #190 SI = 0;
  */

  // Sequence of threes
    #60 SI = 1;
    #80 SI = 0;
    #100 SI = 1;
    #120 SI = 0;
    #140 SI = 1;
    #160 SI = 0;
    #180 SI = 1;
    #200 SI = 0;
  join
endmodule
```

Simulation results: Accumulation of a sequence of four 1s.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

Accumulation of a sequence of 3s:

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

6.55

```
module Prob_6_55 (output Q8, Q4, Q2, Q1, input Count, Clear_b);
  supply1 Pwr;
  not (Q8_bar, Q8);
  and (J_in_M8, Q2, Q4);
  JKFF M1 (Q1, Pwr, Pwr, Count, Clear_b);
  JKFF M2 (Q2, Q8_bar, Pwr, Q1, Clear_b);
  JKFF M4 (Q4, Pwr, Pwr, Q2_ Clear_b);
  JKFF M8 (Q8, J_in_M8, Pwr, Q1_ Clear_b);
endmodule

module JKFF (output reg Q, input J_in, K_in, C, Clear_b);
  always @ (negedge C) if (Clear_b== 1'b0) Q <= 1'b0; else
    case ({J_in, K_in})
      2'b00:      Q <= Q;
      2'b01:      Q <= 1'b0;
      2'b10:      Q <= 1'b1;
      2'b11:      Q <= ~Q;
    endcase
endmodule

module t_Prob_6_55 ();
  wire Q8, Q4, Q2, Q1;
  reg Count , Clear_b;
  wire [3:0] value = {Q8, Q4, Q2, Q1};          // Display counter
  Prob_6_55 M0 (Q8, Q4, Q2, Q1, Count, Clear_b);

  initial #200 $finish;
  initial begin Count = 0; forever #5 Count = ~Count ; end
  initial fork
    Clear_b = 0;
    #20 Clear_b = 1;
  join
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

6.56 Clear_b

```
module Prob_6_56 (output A3, A2, A1, A0, Next_stage, input Count_enable, CLK, Clear_b);
  assign Next_stage = A3 && A2 && A1 && A0;
  and (JK_in_M1, Count_enable, A0);
  and (JK_in_M2, JK_in_M1, A1);
  and (JK_in_M3, JK_in_M2, A2);
  and (Next_stage, JK_in_M3, A3);
  JKFF M0 (A0, Count_enable, Count_enable, CLK, Clear_b);
  JKFF M1 (A1, JK_in_M1, J_in_M1, CLK, Clear_b);
  JKFF M2 (A2, JK_in_M2, JK_in_M2, CLK, Clear_b);
  JKFF M3 (A3, JK_in_M3, JK_in_M3, A3, CLK, Clear_b);
endmodule

module JKFF (output reg Q, input J_in, K_in, C, Clear_b);
  always @ (posedge C) if (Clear_b == 1'b0) Q <= 0; else
    case ({J_in, K_in})
      2'b00:      Q <= Q;
      2'b01:      Q <= 1'b0;
      2'b10:      Q <= 1'b1;
      2'b11:      Q <= ~Q;
    endcase
endmodule

module t_Prob_6_56 ();
  wire A3, A2, A1, A0;
  wire Next_stage;
  reg Count_enable;
  reg CLK, Clear_b

  wire [3:0] value = {A3, A2, A1, A0};

  Prob_6_56 M0 (A3, A2, A1, A0, Next_stage, Count_enable, CLK, Clear_b);

  initial #400 $finish;
  initial begin CLK = 0; forever #5 CLK = ~CLK; end
  initial fork
    Clear_b = 0;
    #10 Clear_b = 1;
    #100 Clear_b = 0;       // Reset on the fly
    #120 Clear_b = 1;

    Count_enable = 0;
    #20 Count_enable = 1;
    #50 Count_enable = 0;
    #80 Count_enable = 1;
  join
endmodule
```


All rights reserved.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

6.57

```
module Prob_6_57 (output A3, A2, A1, A0, input Up, Down, CLK, Clear_b);
  not (Up_bar, Up);
  not (A0_bar, A0);
  not (A1_bar, A1);
  not (A2_bar, A2);
  and (w1, Up_bar, Down);
  and (w2, w1, A0_bar);
  and (w3, Up, A0);
  and (w4, w2, A1_bar);
  and (w5, w3, A1);
  and (w6, w4, A2_bar);
  and (w7, w5, A2);
  or (T0, w1, Up);
  or (T1, w2, w3);
  or (T2, w4, w5);
  or (T3, w6, w7);
  TFF M0 (A0, A0_bar, T0, CLK, Clear_b);
  TFF M1 (A1, A1_bar, T1, CLK, Clear_b);
  TFF M2 (A2, A2_bar, T2, CLK, Clear_b);
  TFF M3 (A3, A3_bar, T3, CLK, Clear_b);
endmodule

module TFF (output reg Q, output Q_bar, input T, Clear_b, C, Clear_b);   // Active low reset is needed
  assign Q_bar = ~Q;
  always @ (posedge C) if (Clear_b == 1'b0) Q <= 0; else if (T) Q <= ~Q;
endmodule

module t_Prob_6_57 ();
  wire A3, A2, A1, A0;
  reg Up, Down, CLK, Clear_b;
  wire [3:0] value = {A3, A2, A1, A0};      // Display count

  Prob_6_57 M0(A3, A2, A1, A0, Up, Down, CLK, Clear_b);

  initial #250 $finish;
  initial begin CLK = 0; forever #5 CLK = ~CLK; end
  initial fork
    Clear_b = 1'b0;
    #20 Clear_b = 1'b1;
    #60 Clear_b = 0;    // Reset on the fly
    #80 Clear_b = 1;
    Up = 1'b0;
    Down = 1'b0;

    #50 Up = 1'b1;
    #80 Down = 1'b1;         // Up has priority
    #160 Up = 1'b0;
    #200 Down = 1'b0;
  join
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

6.58

```
module Problem_6_58 (output A3, A2, A1, A0, C_out, input I3, I2, I1, I0, Count, Load, CLK, Clear_b);
  not (Load_bar, Load);
  not (I0_bar, I0);
  not (I1_bar, I1);
  not (I2_bar, I2);
  not (I3_bar, I3);
  and (w0, Count, Load_bar);
  and (w1, Load, I0);
  and (w2, Load, I0_bar);
  and (w3, Load, I1);
  and (w4, Load, I1_bar);
  and (w5, Load, I2);
  and (w6, Load, I2_bar);
  and (w7, Load, I3);
  and (w8, Load, I3_bar);
  or ( w9, w1, w0);
  or ( w10, w2, w0);
  or ( w11, w3, w17);
  or ( w12, w4, w17);
  or ( w13, w5, w18);
  or ( w14, w6, w18);
  or ( w15, w7, w19);
  or ( w16, w8, w19);
  and (w17, w0, A0);
  and (w18, w0, A0, A1);
  and (w19, w0, A0, A1, A2);
  and (C_out, w0, A0, A1, A2, A3);

  JKFF M0 (A0, w9, w10, CLK, Clear_b);
  JKFF M1 (A1, w11, w12, CLK, Clear_b);
  JKFF M2 (A2, w13, w14, CLK, Clear_b);
  JKFF M3 (A3, w15, w16, CLK, Clear_b);
endmodule

module JKFF (output reg Q, input J_in, K_in, C, Clear_b);
  always @ (posedge C) if (Clear_b == 1'b0) Q <= 0; else
    case ({J_in, K_in})
      2'b00:      Q <= Q;
      2'b01:      Q <= 1'b0;
      2'b10:      Q <= 1'b1;
      2'b11:      Q <= ~Q;
    endcase
endmodule

module t_Problem_6_58 ();
  wire A3, A2, A1, A0, C_out;
  reg I3, I2, I1, I0, Count, Load, CLK, Clear_b;
  wire [3:0] value = {A3, A2, A1, A0};
  wire [3:0] Par_word = {I3, I2, I1, I0};

  Problem_6_58 M0 (A3, A2, A1, A0, C_out, I3, I2, I1, I0, Count, Load, CLK, Clear_b);


  initial #400 $finish;
  initial begin CLK = 0; forever #5 CLK = ~CLK; end
  initial fork
    {I3, I2, I1, I0} = 4'b0101;    // Data for parallel load
    Clear_b = 0;
    #20 Clear_b = 1;
```


All rights reserved.

```
    Count = 0;
    #50 Count = 1;      // Counting
    #150 Count = 0;     // Pause
    #200 Count = 1;     // Resume counting
    Load = 0;
    #250 Load = 1; // Parallel load
    #260 Load = 0;
  join
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

6.59

```
module Problem_6_59 (output reg A3, A2, A1, A0, output C_out, input I3, I2, I1, I0, Count, Load, CLK, Clear_b);
  always @ (posedge CLK) if (Clear_b == 1'b0) {A3, A2, A1, A0} <= 4'b0; else
    if (Load) {A3, A2, A1, A0} <= {I3, I2, I1, I0};
    else if (Count) {A3, A2, A1, A0} <= {A3, A2, A1, A0} + 4'b0001;

  assign C_out = A3 && A2 && A1 && A0 && Count && (!Load);
endmodule

module t_Problem_6_59 ();
  wire A3, A2, A1, A0, C_out;
  reg I3, I2, I1, I0, Count, Load, CLK, Clear_b;
  wire [3:0] value = {A3, A2, A1, A0};
  wire [3:0] Par_word = {I3, I2, I1, I0};

  Problem_6_59 M0 (A3, A2, A1, A0, C_out, I3, I2, I1, I0, Count, Load, CLK, Clear_b);


  initial #400 $finish;
  initial begin CLK = 0; forever #5 CLK = ~CLK; end
  initial fork
    {I3, I2, I1, I0} = 4'b0101;    // Data for parallel load
    Clear_b = 0;
    #20 Clear_b = 1;

    Count = 0;
    #50 Count = 1;       // Counting
    #150 Count = 0;      // Pause
    #200 Count = 1;      // Resume counting
    Load = 0;
    #250 Load = 1; // Parallel load
    #260 Load = 0;
  join
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

Chapter 7

mdc 1/19/07 11:02 AM
Comment [1]: Spell check

7.1 **(a)** 8 K x 32 = 2^{13} x 16 A = 13 D = 16

(b) 2 G x 8 = 2^{31} x 8 A = 31 D = 8

(c) 16 M x 32 = 2^{24} x 32 A = 24 D = 32

(d) 256 K x 64 = 2^{18} x 64 A = 18 D = 64

(e)

7.2 **(a)** 2^{13} **(b)** 2^{31} **(c)** 2^{26} **(d)** 2^{21}

7.3 Address: 563_{10} = 10_0011_0011_2

Data word: $1{,}212_{10}$ = 0000_0100_1011_1100_2

7.4 f_{CPU} = 150 MHz, $T_{CPU} = 1/f_{CPU} = 6.67^{-9}$ Hz^{-1}

7.5

Pending

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

7.6

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

7.7 **(a)** 16 K = 2^{14} = 2^7 x 2^7 = 128 x 128
Each decoder is 7 × 128
Decoders require 256 AND gates, each with 7 inputs

(b) 6,000 = 0101110_1110000
x = 46 y = 112

7.8 **(a)** 256 K / 32 K = 8 chips

(b) 256 K = 2^{18} (18 address lines for memory); 32 K = 215 (15 address pins / chip)

(c) 18 – 15 = 3 lines ; must decode with 3 × 8 decoder

7.9 13 + 12 = 25 address lines. Memory capacity = 2^{25} words.

7.10 01011011 = 1 2 3 4 5 6 7 8 9 10 11 12 13
P_1 P_2 0 P_4 1 0 1 P_8 1 0 1 1 P_{13}

P_1 = Xor of bits (3, 5, 7, 9, 11) = 0, 1, 1, 1, 1 = 0 (Note: even # of 0s)
P_2 = Xor of bits (3, 6, 7, 10, 11) = 0, 0, 1, 0, 1 = 0
P_4 = Xor of bits (5, 6, 7, 12) = 1, 0, 1, 1 = 1 (Note: odd # of 0s)
P_8= Xor of bits (9, 10, 11, 12) = 1, 0, 1, 1, = 1

Composite 13-bit code word: 0001 1011 1011 1

7.11 11001001010 = 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
P_1 P_2 1 P_4 1 0 0 P_8 1 0 0 1 0 1 0

P_1 = Xor of bits (3, 5, 7, 9, 11, 13, 15) = 1, 1, 0, 1, 0, 0, 0 = 1 (Note: odd # of 0s)
P_2 = Xor of bits (3, 6, 7, 10, 11, 14, 15) = 1, 0, 0, 0, 0, 1, 0 = 0 (Note: even # of 0s)
P_4 = Xor of bits (5, 6, 7, 12, 13, 14, 15) = 1, 0, 0, 1, 0, 1, 0 = 1
P_8= Xor of bits (9, 10, 11, 12, 13, 14, 15) = 1, 0, 0, 1, 0, 1, 0 = 1

Composite 15-bit code word: 101 110 011 001 010

7.12 **(a)** 1 2 3 4 5 6 7 8 9 10 11 12
0 0 0 0 1 1 1 0 1 0 1 0

C_1 (1, 3, 5, 7, 9, 11) = 0, 0, 1, 1, 1, 1 = 0
C_2 (2, 3, 6, 7, 10, 11) = 0, 0, 1, 1, 0, 1 = 1
C_4 (4, 5, 6, 7, 12) = 0, 1, 1, 1, 0 = 1
C_8 (8, 9, 10, 11, 12) = 0, 1, 0, 1, 0 = 0

C = 0110
Error in bit 6.
Correct data: 0101 1010

(b) 1 2 3 4 5 6 7 8 9 10 11 12
1 0 1 1 1 0 0 0 0 1 1 0

C_1 (1, 3, 5, 7, 9, 11) = 1, 1, 1, 0, 0, 1 = 0
C_2 (2, 3, 6, 7, 10, 11) = 0, 1, 0, 0, 1, 1 = 1
C_4 (4, 5, 6, 7, 12) = 1, 1, 0, 0, 0 = 0
C_8 (8, 9, 10, 11, 12) = 0, 0, 1, 1, 0 = 0

C = 0010

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

Error in bit 2 = Parity bit P_2.

	3	5	6	7	9	10	11	12
Correct 8-bit data:	1	1	0	0	0	1	1	0

(c)

1	2	3	4	5	6	7	8	9	10	11	12
1	0	1	1	1	1	1	1	0	1	0	0

C = 0000)No errors)
C_1 (1, 3, 5, 7, 9, 11) = 1, 1, 1, 0, 0, 1 = 0
C_2 (2, 3, 6, 7, 10, 11) = 0, 1, 0, 0, 1, 1 = 1
C_4 (4, 5, 6, 7, 12) = 1, 1, 0, 0, 0 = 0
C_8 (8, 9, 10, 11, 12) = 0, 0, 1, 1, 0 = 0

	3	5	6	7	9	10	11	12
Correct 8-bit data:	1	1	1	1	0	1	0	0

7.13 (**a**) 16-bit data (From Table 7.2): 5 Check bits
1 bit
---
6 parity bits

(**b**) 32-bit data (From Table 7.2): 6 Check bits
1 bit
---
7 parity bits

(6) 16-bit data (From Table 7.2): 5 Check bits
1 bit
---
6 parity bits

7.14 **(a)**

1	2	3	4	5	6	7
P_1	P_2	0	P4	0	1	0

P_1 = Xor (3, 5, 7) = 0, 0, 0 = 1
P_2 = Xor (3, 6, 7) = 0, 1, 0 = 0
P_4 = Xor (5, 6, 7) = 0, 1, 0 = 1

7-bit word: 0101010

(b) No error:

C_1 = Xor (1, 3, 5, 7) = 0, 0, 0, 0 = 0
C_2 = Xor (2, 3, 6, 7) = 1, 0, 1, 0 = 0
C_4 = Xor (4, 5, 6, 7) = 1, 0, 1, 0 = 0

(c) Error in bit 5:

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

C_1 = Xor (0, 0, 1, 0) = 1
C_2 = Xor (1, 0, 1, 0) = 0
C_4 = Xor (1, 1, 1, 0) = 1
Error in bit 5: C = 101

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

(d) 8-bit word 1 2 3 4 5 6 7 8
0 1 0 1 0 1 0 1
Error in bits 2 and 5: 0 0 0 1 1 1 0 1
C_1 = Xor (0, 0, 1, 0) = 1
C_2 = Xor (0, 0, 1, 0) = 1
C4 = Xor (1, 1, 1, 0) = 1
P = 0
C =(1, 1, 1) ≠ 0 and P = 0 indicates double error.

7.15

7.16

Note: $4096 = 2^{12}$

16 inputs + 8 outputs requires a 24-pin IC.

7.18 **(a)** 256 × 8 **(b)** 512 × 5 **(c)** 1024 × 4 **(d)** 32 × 7

7.17

All rights reserved.

Input Address	Output of ROM			
$I_5 I_4 I_3 I_2 I_1$	$D_6 D_5 D_4$	$D_3 D_2 D_1$	D_0 (2^0)	Decimal
0 0 0 0 0	0 0 0	0 0 0	0, 1	0, 1
0 0 0 0 1	0 0 0	0 0 1	0, 1	2, 3
…	…	…	…	…
…	…	…	…	…
0 1 0 0 0	0 0 1	0 1 1	0, 1	16, 17
0 1 0 0 1	0 0 1	1 0 0	0, 1	18, 19
…	…	…	…	…
…	…	…	…	…
1 1 1 1 0	1 1 0	0 0 0	0, 1	60, 61
1 1 1 1 1	1 1 0	0 0 1	0, 1	62, 63

7.18 **(a)** 8 inputs 8 outputs 2^8 x 8 256 x 8 ROM

(b) 9 inputs 5 outputs 2^9 x 5 512 x 5 ROM

(c) 10 inputs 4 outputs 2^{10} x 4 1024 x 4 ROM

(d) 5 inputs 7 outputs 2^5 x 7 32 x 7 ROM

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

7.19

$A = y'z + x'z + xyz'$
$A' = y'z' + x'z' + xyz$

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

$C = x'yz + xy'z$
$C' = z' + x'y' + xy$

$D = xy' + xz + y'z + x'yz'$
$D' = x'y'z' + x'yz + xyz'$

	Product term	Inputs $x\ y\ z$	Outputs A	B	C	D
$y'z$	1	- 0 1	1	-	-	1
$x'z$	2	0 - 1	1	-	-	-
xyz'	3	1 1 0	1	-	-	-
xy	4	1 1 -	-	1	-	-
$x'y'$	5	0 0 -	-	1	-	-
$x'yz$	6	0 1 1	-	-	1	-
$xy'z$	7	1 0 1	-	-	1	-
xy'	8	1 0 -	-	-	-	1
Xz	9	1 – 1	-	-	-	1
$y'z$	10	- 0 1	-	-	-	1
$x'yz'$	11	0 1 0	-	-	-	1

7.20

Inputs $x\ y\ z$	Outputs A, B, C, D	
0 0 0	1 1 0 1	
0 0 1	0 1 1 1	$M[001] = 0111$
0 1 0	0 0 0 0	
0 1 1	1 0 0 1	
1 0 0	1 1 0 0	$M[100] = 1100$
1 0 1	0 0 1 1	
1 1 0	1 0 0 0	
1 1 1	0 1 0 1	

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

7.21 Note: See truth table in Fig. 7.12(b).

A_2 \ A_1A_0	00	01	11	10
0	m_0 0	m_1 0	m_3 0	m_2 0
1	m_4 0	m_5 0	m_7 1	m_6 1

$F_1 = A_2A_1$
$F'_1 = A'_2 + A'_1$

A_2 \ A_1A_0	00	01	11	10
0	m_0 0	m_1 0	m_3 0	m_2 0
1	m_4 1	m_5 1	m_7 1	m_6 0

$F_2 = A_2A'_1 + A_2A_0$
$F_2' = A'_2 + A_1A'_0$

A_2 \ A_1A_0	00	01	11	10
0	m_0 0	m_1 0	m_3 1	m_2 0
1	m_4 0	m_5 1	m_7 0	m_6 0

$F_3 = A'_2A_1A_0 + A_2A'_1A_0$
$F_3' = A'_0 + A'_2A'_1 + A_2A_1$

A_2 \ A_1A_0	00	01	11	10
0	m_0 0	m_1 0	m_3 0	m_2 1
1	m_4 0	m_5 0	m_7 0	m_6 1

$F_4 = A_1A'_0$
$F'_4 = A'_1 + A_0$

	Product term	Inputs $A_2A_1A_0$	Outputs F_1	F_2	F_3	F_4
A_2A_1	1	1 1 -	1	-	-	-
A'_2	2	0 - -	-	1	-	-
$A_1A'_0$	3	- 1 0	-	1	-	1
$A'_2A_1A_0$	4	- 1 1	-	-	1	-
$A_2A'_1$	5	1 0 1	-	-	1	-
			T	*C*	*T*	*T*

Alternative: F'_1, F'_2, F_3, F_4
(5 terms)

7.22

Decimal		w	x	y	z	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	1	0	0	0	0	0	0	0	1
2	4	0	0	1	0	0	0	0	0	0	1	0	0
3	9	0	0	1	1	0	0	0	0	1	0	0	1
4	16	0	1	0	0	0	0	0	1	0	0	0	0
5	25	0	1	0	1	0	0	0	1	1	0	0	1
6	36	0	1	1	0	0	0	1	0	0	1	0	0
7	49	0	1	1	1	0	0	1	1	0	0	0	1
8	64	1	0	0	0	0	1	0	0	0	0	0	0
9	81	1	0	0	1	0	1	0	1	0	0	0	1
10	100	1	0	1	0	0	1	1	0	0	1	0	0
11	121	1	0	1	1	0	1	1	1	1	0	0	1
12	144	1	1	0	0	1	0	0	1	0	0	0	0
13	169	1	1	0	1	1	0	1	0	1	0	0	1
14	196	1	1	1	0	1	1	0	0	0	1	0	0
15	225	1	1	1	1	1	1	1	0	0	0	0	1

Note: $b_0 = z$, and $b_1 = 0$. ROM would have 4 inputs and 6 outputs. A 4 x 8 ROM would waste two outputs.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

$b_2 = yx'$

$b_3 = xy'z + x'yz$

$b_4 = w'xz + xy'z' + wx'z$

$b_5 = w'xy + wxz + wx'y$

$b_6 = wy + wx'$

$b_7 = wx$

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

7.23

From Fig. 4-3:
$w = A + BC + BD$
$w' = A'B' + A'C'D'$
$x = B'C + B'D + BC'D'$
$x' = B'C'D' + BC\ \ BD$
$y = CD + C'D'$
$y' = C'D + CD'$
$z = D'$
$z' = D$
Use w, x′, y, z (7 terms)

	Product term	Inputs A	B	C	D	Outputs F_1	F_2	F_3	F_4
A	1	1	-	-	-	1	-	-	-
BC	2	-	1	1	-	1	1	-	-
BD	3	-	1	-	1	1	1	-	-
$B'C'D'$	4	-	0	0	0	-	1	-	-
CD	5	-	-	1	1	-	-	1	-
$C'D'$	6	-	-	0	0	-	-	1	-
D'	7	-	-	-	0	-	-	-	1
						T	C	T	T

7.24

Product term	AND Inputs A	B	C	D	Outputs
1	1	-	-	-	
2	-	1	1	-	$w = A + BC + BD$
3	-	1	-	1	
4	-	0	1	-	
5	-	0	-	1	$x = B'C + B'D + BC'D'$
6	-	1	0	0	
7	-	-	1	1	
8	-	-	0	0	$y = CD + C'D'$
9	-	-	-	-	
10	-	-	-	0	
11	-	-	-	-	$z = D'$
12	-	-	-	-	

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

7.25

$A = yz' + xz' + x'y'z$

$B = y'z' + x'y' + yz$

$C = A + xyz$

$D = z + x'y$

Product term	AND Inputs x y z A	Outputs
1	- 1 0 -	A = yz' + xz' + x'y'z
2	1 - 0 -	
3	0 0 1 -	
4	- 0 0 -	B = y'z' + x'y' + yz
5	0 0 - -	
6	0 1 1 -	
7	0 - - 1	C = A + xyz
8	1 1 1 -	
9	0 - - -	
10	0 - 1 -	D = z + x'y
11	0 1 - -	
12	- - - -	

$A = yz' + xz' + x'y'z$
$B = y'z' + x'y' + yz$
$C = A + xyz$
$D = z + x'y$

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

7.26

7.27

The results of Prob. 6.17 can be used to develop the equations for a three-bit binary counter with D-type flip-flops.

$DA_0 = A'_0$
$DA_1 = A'_1A_0 + A_1A'_0$
$DA_2 = A'_2\,A_1A_0 + A_2A'_1 + A_2A'_0$

$C_{out} = A_2A_1A_0$

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

7.28

7.29

	Product term	Inputs $x\ y\ A$	Output D_A
$x'y'A$	1	0 0 1	1
$x'yA'$	2	0 1 0	1
$xy'A'$	3	1 0 0	1
xyA	4	1 1 1	1

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

CHAPTER 8

8.1 **(a)** The transfer and increment occur concurrently, i.e., at the same clock edge. After the transfer, *R2* holds the contents that were in *R1* before the clock edge, and *R2* holds its previous value incremented by 1.

(b) Decrement the content of *R3* by one.

(c) If (S_1 = 1), transfer content of *R1* to *R0*. If (S_1 = 0 and S_2 = 1), transfer content of *R2* to *R0*.

8.2

8.3

(a) (b) (c)

8.4

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

8.5 The operations specified in a flowchart are executed sequentially, one at a time. The operations specified in an ASM chart are executed concurrently for each ASM block. Thus, the operations listed within a state box, the operations specified by a conditional box, and the transfer to the next state in each ASM block are executed at the same clock edge. For example, in Fig. 8.5 with *Start* = 1 and Flag = 1, signal *Flush_R* is asserted. At the clock edge the state moves to *S_2*, and register *R* is flushed.

An ASM chart describes the state transitions and output signals generated by a finite state machine in response to its input signals. An ASMD chart is an ASM chart that has been annotated to indicate the register operations that are executed by the machine in response to the control signals (outpus) generated by the state machine.

8.6

Note: In practice, the asynchronous inputs x and y should be synchronized to the clock to avoid metastable conditons in the flip-flops..

Note: To avoid counting a person more than once, the machine waits until x or y is de-asserted before incrementing or decrementing the counter. The machine also accounts for persons entering and leaving simultaneously.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

8.7 RTL notation:

S0: Initial state: if (start = 1) then (*RA* ← data_*A*, *RB* ← data_*B*, go to *S1*).
S1: {Carry, *RA}* ← *RA* + (2's complement of *RB*), go to *S2*.
S2: If (borrow = 0) go to *S0*. If (borrow = 1) then *RA* ← (2's complement of *RA*), go to *S0*.

Block diagram and ASMD chart:

```
module Subtractor_P8_7
  (output done, output [7:0] result, input [7: 0] data_A, data_B, input  start, clock, reset_b);

  Controller_P8_7 M0 (Load_A_B, Subtract, Convert, done, start, borrow, clock, reset_b);
  Datapath_P8_7 M1 (result, borrow, data_A, data_B, Load_A_B, Subtract, Convert, clock, reset_b);
endmodule

module Controller_P8_7 (output reg Load_A_B, Subtract, output reg Convert, output done,
  input start, borrow, clock, reset_b);
  parameter  S0 = 2'b00, S1 = 2'b01, S2 = 2'b10;
  reg [1: 0] state, next_state;
  assign done = (state == S0);

  always @ (posedge clock, negedge reset_b)
    if (!reset_b) state <= S0; else state <= next_state;

  always @ (state, start, borrow) begin
    Load_A_B = 0;
    Subtract = 0;
    Convert = 0;

    case (state)
      S0:       if (start) begin Load_A_B = 1; next_state = S1; end
      S1:       begin  Subtract = 1; next_state = S2; end
      S2:        begin next_state = S0; if (borrow) Convert = 1; end
      default: next_state = S0;
    endcase
  end
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
module Datapath_P8_7 (output [7: 0] result, output borrow, input [7: 0] data_A, data_B,
  input Load_A_B, Subtract, Convert, clock, reset_b);
  reg      carry;
reg [8:0] diff;
  reg [7: 0]    RA, RB;
  assign       borrow = carry;
  assign result = RA;

  always @ (posedge clock, negedge reset_b)
    if (!reset_b) begin carry <= 1'b0; RA <= 8'b0000_0000; RB <= 8'b0000_0000; end
    else begin
      if (Load_A_B) begin RA <= data_A; RB <= data_B; end
      else if (Subtract) {carry, RA} <= RA +  ~RB + 1;

      // In the statement above, the math of the LHS is done to the wordlength of the LHS
      // The statement below is more explicit about how the math for subtraction is done:
      // else if (Subtract) {carry, RA} <= {1'b0, RA} + {1'b1, ~RB } + 9'b0000_0001;
      // If the 9-th bit is not considered, the 2s complement operation will generate a carry bit,
      //  and borrow must be formed as borrow = ~carry.

      else if (Convert) RA <= ~RA + 8'b0000_0001;
    end
endmodule

// Test plan – Verify;
// Power-up reset
// Subtraction with data_A > data_B
// Subtraction with data_A < data_B
// Subtraction with data_A = data_B
// Reset on-the-fly: left as an exercise

module t_Subtractor_P8_7;
  wire                done;
  wire     [7:0]      result;
  reg [7: 0]    data_A, data_B;
  reg                 start, clock, reset_b;

  Subtractor_P8_7 M0 (done, result, data_A, data_B, start, clock, reset_b);

  initial #200 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial fork
     reset_b = 0;
      #2 reset_b = 1;
    #90 reset_b = 1;
    #92 reset_b = 1;
  join

  initial fork
    #20 start = 1;
    #30 start = 0;
    #70 start = 1;
    #110 start = 1;
  join

  initial fork
    data_A = 8'd50;
    data_B = 8'd20;

    #50 data_A = 8'd20;
    #50 data_B = 8'd50;
```


All rights reserved.

```
    #100 data_A = 8'd50;
    #100 data_B = 8'd50;
  join
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

8.8 RTL notation:
S0: if (start = 1) *AR* ← input data, *BR* ← input data, go to *S1*.
S1: if (AR [15]) = 1 (sign bit negative) then *CR* ← *AR*(shifted right, sign extension).
else if (positive non-zero) then (Overflow ← *BR*([15] ⊕ [14]), *CR* ← *BR*(shifted left)
else if (*AR* = 0) then (CR ← 0).

```
module Prob_8_8 (output done, input [15: 0] data_AR, data_BR, input  start, clock, reset_b);

 Controller_P8_8 M0 (
   Ld_AR_BR, Div_AR_x2_CR, Mul_BR_x2_CR, Clr_CR, done,
   start, AR_lt_0, AR_gt_0, AR_eq_0, clock, reset_b
 );
 Datapath_P8_8 M1 (
   Overflow, AR_lt_0, AR_gt_0, AR_eq_0, data_AR, data_BR,
   Ld_AR_BR, Div_AR_x2_CR, Mul_BR_x2_CR, Clr_CR, clock, reset_b
 );
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
module Controller_P8_8 (
  output reg Ld_AR_BR, Div_AR_x2_CR, Mul_BR_x2_CR, Clr_CR,
  output done, input start, AR_lt_0, AR_gt_0, AR_eq_0, clock, reset_b
);
  parameter  S0 = 1'b0, S1 = 1'b1;
  reg state, next_state;
  assign done = (state == S0);

  always @ (posedge clock, negedge reset_b)
    if (!reset_b) state <= S0; else state <= next_state;

  always @ (state, start, AR_lt_0, AR_gt_0, AR_eq_0) begin
    Ld_AR_BR = 0;
    Div_AR_x2_CR = 0;
    Mul_BR_x2_CR = 0;
    Clr_CR = 0;

    case (state)
      S0:        if (start) begin Ld_AR_BR = 1; next_state = S1; end
      S1:        begin
                   next_state = S0;
                    if (AR_lt_0) Div_AR_x2_CR = 1;
                   else if (AR_gt_0) Mul_BR_x2_CR = 1;
                   else if (AR_eq_0) Clr_CR = 1;
                 end
      default: next_state = S0;
    endcase
  end
endmodule

module Datapath_P8_8 (
  output reg Overflow, output AR_lt_0, AR_gt_0, AR_eq_0, input [15: 0] data_AR, data_BR,
  input Ld_AR_BR, Div_AR_x2_CR, Mul_BR_x2_CR, Clr_CR, clock, reset_b
);
  reg [15: 0]  AR, BR, CR;
  assign       AR_lt_0 = AR[15];
  assign       AR_gt_0 = (!AR[15]) && (| AR[14:0]);          // Reduction-OR
  assign       AR_eq_0 = (AR == 16'b0);


  always @ (posedge clock, negedge reset_b)
    if (!reset_b) begin AR <= 8'b0; BR <= 8'b0; CR <= 16'b0; end
    else begin
      if (Ld_AR_BR) begin AR <= data_AR; BR <= data_BR; end
      else if (Div_AR_x2_CR) CR <= {AR[15], AR[15:1]};   // For compiler without arithmetic right shift
      else if (Mul_BR_x2_CR) {Overflow, CR} <= (BR << 1);
      else if (Clr_CR) CR <= 16'b0;
    end
 endmodule

// Test plan – Verify;
// Power-up reset
// If AR < 0 divide AR by 2 and transfer to CR
// If AR > 0 multiply AR by 2 and transfer to CR
// If AR = 0 clear CR
// Reset on-the-fly
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
module t_Prob_P8_8;
  wire        done;
  reg [15: 0] data_AR, data_BR;
  reg         start, clock, reset_b;
  reg [15: 0] AR_mag, BR_mag, CR_mag;    // To illustrate 2s complement math

// Probes for displaying magnitude of numbers
  always @ (M0.M1.AR)              // Hierarchical dereferencing
    if (M0.M1.AR[15]) AR_mag = ~M0.M1.AR+ 16'd1; else AR_mag = M0.M1.AR;
  always @ (M0.M1.BR )
    if (M0.M1.BR[15]) BR_mag = ~M0.M1.BR+ 16'd1; else BR_mag = M0.M1.BR;
  always @ (M0.M1.CR)
    if (M0.M1.CR[15]) CR_mag = ~M0.M1.CR + 16'd1; else CR_mag = M0.M1.CR;

  Prob_8_8 M0 (done, data_AR, data_BR, start, clock, reset_b);

  initial #250 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial fork
        reset_b = 0;       // Power-up reset
     #2 reset_b = 1;
   #50 reset_b = 0;  // Reset on-the-fly
   #52 reset_b = 1;
   #90 reset_b = 1;
   #92 reset_b = 1;
  join

  initial fork
   #20 start = 1;
   #30 start = 0;
   #70 start = 1;
   #110 start = 1;
  join

  initial fork
   data_AR = 16'd50;         // AR > 0
   data_BR = 16'd20;         // Result should be 40

   #50 data_AR = 16'd20;
   #50 data_BR = 16'd50;  // Result should be 100

   #100 data_AR = 16'd50;
   #100 data_BR = 16'd50;

   #130 data_AR = 16'd0;  // AR = 0, result should clear CR

   #160 data_AR = -16'd20;     // AR < 0, Verilog stores 16-bit 2s complement
   #160 data_BR = 16'd50;// Result should have magnitude10

   #190 data_AR = 16'd20;// AR < 0, Verilog stores 16-bit 2s complement
   #190 data_BR = 16'hffff;// Result should have overflow
  join
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

8.9

Design equations:
D_{S_idle} = S_2 + S_idle Start'
D_{S_1} = S_idle Start + S_1 (A2 A3)'
D_{S_2} = A2 A3 S_1

HDL description:

```
module Prob_8_9 (output E, F, output [3: 0] A, output A2, A3,  input  Start, clock, reset_b);

 Controller_Prob_8_9 M0 (set_E, clr_E, set_F, clr_A_F, incr_A, Start, A2, A3, clock, reset_b);
 Datapath_Prob_8_9 M1 (E, F,  A,  A2, A3, set_E, clr_E, set_F, clr_A_F, incr_A, clock, reset_b);

endmodule

// Structural version of the controller (one-hot)
// Note that the flip-flop for S_idle must have a set input and reset_b is wire to the set
// Simulation results match Fig. 8-13

module Controller_Prob_8_9 (
  output    set_E, clr_E, set_F, clr_A_F, incr_A,
  input     Start, A2, A3, clock, reset_b
);

  wire      D_S_idle, D_S_1, D_S_2;
  wire      q_S_idle, q_S_1, q_S_2;
  wire      w0, w1, w2, w3;
  wire [2:0]     state = {q_S_2, q_S_1, q_S_idle};

  // Next-State Logic
  or (D_S_idle, q_S_2, w0);        // input to D-type flip-flop for q_S_idle
  and (w0, q_S_idle, Start_b);
  not (Start_b, Start);

  or (D_S_1, w1, w2, w3);          // input to D-type flip-flop for q_S_1
  and (w1, q_S_idle, Start);
  and (w2, q_S_1, A2_b);
  not (A2_b, A2);
  and (w3, q_S_1, A2, A3_b);
  not (A3_b, A3);

  and (D_S_2, A2, A3, q_S_1);        // input to D-type flip-flop for q_S_2

  D_flop_S M0 (q_S_idle, D_S_idle, clock, reset_b);
  D_flop M1 (q_S_1, D_S_1, clock, reset_b);
  D_flop M2 (q_S_2, D_S_2, clock, reset_b);

  // Output Logic
  and (set_E, q_S_1, A2);
  and (clr_E,  q_S_1, A2_b);
  buf (set_F, q_S_2);
  and (clr_A_F, q_S_idle, Start);
  buf (incr_A, q_S_1);
endmodule

module D_flop (output reg q, input data, clock, reset_b);
  always @ (posedge clock, negedge reset_b)
    if (!reset_b) q <= 1'b0; else q <= data;
endmodule

module D_flop_S (output reg q, input data, clock, set_b);
```


All rights reserved.

```
  always @ (posedge clock, negedge set_b)
    if (!set_b) q <= 1'b1; else q <= data;
endmodule

/*
// RTL Version of the controller
// Simulation results match Fig. 8-13

module Controller_Prob_8_9 (
  output reg   set_E, clr_E, set_F, clr_A_F, incr_A,
  input        Start, A2, A3, clock, reset_b
);
  parameter   S_idle = 3'b001, S_1 = 3'b010, S_2 = 3'b100;        // One-hot
  reg [2: 0] state, next_state;

  always @ (posedge clock, negedge reset_b)
    if (!reset_b) state <= S_idle; else state <= next_state;

  always @ (state, Start, A2, A3) begin
    set_E    = 1'b0;
    clr_E    = 1'b0;
    set_F    = 1'b0;
    clr_A_F      = 1'b0;
    incr_A   = 1'b0;
    case (state)
      S_idle:      if (Start) begin next_state = S_1; clr_A_F = 1; end
                     else next_state = S_idle;

      S_1:    begin
                        incr_A = 1;
                        if (!A2) begin next_state = S_1; clr_E = 1; end
                        else begin
                          set_E = 1;
                          if (A3) next_state = S_2; else next_state = S_1;
                        end
                      end
      S_2:    begin next_state = S_idle; set_F = 1; end
      default:    next_state = S_idle;
    endcase
  end
endmodule
*/
module Datapath_Prob_8_9 (
  output reg E, F, output reg [3: 0] A, output A2, A3,
  input set_E, clr_E, set_F, clr_A_F, incr_A, clock, reset_b
);
  assign A2 = A[2];
  assign A3 = A[3];

  always @ (posedge clock, negedge reset_b) begin
    if (!reset_b) begin E <= 0; F <= 0; A <= 0; end
    else begin
      if (set_E)  E <= 1;
      if (clr_E)   E <= 0;
      if (set_F)  F <= 1;
      if (clr_A_F)     begin A <= 0; F <= 0; end
      if (incr_A) A <= A + 1;
    end
  end
endmodule
```


All rights reserved.

```
// Test Plan - Verify: (1) Power-up reset, (2) match ASMD chart in Fig. 8-9 (d),
// (3) recover from reset on-the-fly

module t_Prob_8_9;
  wire E, F;
  wire [3: 0] A;
  wire A2, A3;
  reg Start, clock, reset_b;

  Prob_8_9 M0 (E, F, A, A2, A3, Start, clock, reset_b);

  initial #500 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial begin reset_b = 0; #2 reset_b = 1; end
  initial fork
    #20 Start = 1;
    #40 reset_b = 0;
    #62 reset_b = 1;
  join
endmodule
```

8.10

```
module Prob_8_10 (input x, y, clock, reset_b);
  reg [ 1: 0]     state, next_state;
  parameter    s0 = 2'b00, s1 = 2'b01, s2 = 2'b10, s3 = 2'b11;
  always @ (posedge clock, negedge reset_b)
    if (reset_b == 0) state <= s0; else state <= next_state;

  always @ (state, x, y) begin
    next_state = s0;
    case (state)
      s0: if (x == 0) next_state = s0; else next_state = s1;
      s1: if (y == 0) next_state = s2; else next_state = s3;
      s2: if (x == 0) next_state = s0; else if (y == 0) next_state = s2; else next_state = s3;
      s3: if (x == 0) next_state = s0; else if (y == 0) next_state = s2; else next_state = s3;
    endcase
  end
endmodule

module t_Prob_8_10 ();
  reg x, y, clock, reset_b;
```


All rights reserved.

```
  Prob_8_10 M0 (x, y, clock, reset_b);

  initial #150 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial fork
    reset_b = 0;
    #12 reset_b = 1;
     x = 0; y = 0;          // Remain in s0
    #10 y = 1;              // Remain in s0
    #20 x = 1;              // Go to s1 to s3
    #40 reset_b = 0;        // Go to s0
    #42 reset_b = 1;        // Go to s2 to s3
    #60 y = 0;              // Go to s2
    #80 y = 1;              // Go to s3
    #90  x = 0;             // Go to s0
    #100 x = 1;             // Go to s1
    #110 y = 0;             // Go to s2
    #130      x = 0;        // Go to s0
  join
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

8.11 $D_A = A'B + Ax$
$D_B = A'B'x + A'By + xy$

state	inputs	next state
0 0	0 0	0 0
0 0	0 1	0 0
0 0	1 0	0 1
0 0	1 1	0 1
0 1	0 0	1 0
0 1	0 1	1 1
0 1	1 0	1 0
0 1	1 1	1 1
1 0	0 0	0 0
1 0	0 1	0 0
1 0	1 0	1 0
1 0	1 1	1 1
1 1	0 0	0 0
1 1	0 1	0 0
1 1	1 0	1 0
1 1	1 1	1 1

$D_A = A'B + Ax$

$D_B = A'B'x + A'By + xy$

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

8.12 For the 4-bit synchronous counter, modify the counter in Fig. 6.12 to add a signal, Clear, to clear the counter synchronously, as shown in the circuit diagram below.

```
module Counter_4bit_Synch_Clr (output [3: 0]  A, output next_stage, input Count_enable, Clear, CLK);
  wire A0, A1, A2, A3;
  assign A[3: 0] = {A3, A2, A1, A0};
  JK_FF M0 (A0, J0, K0, CLK);
  JK_FF M1 (A1, J1, K1, CLK);
  JK_FF M2 (A2, J2, K2, CLK);
  JK_FF M3 (A3, J3, K3, CLK);

  not (Clear_b, Clear);
  and (J0, Count_enable, Clear_b);
  and (J1, J0, A0);
  and (J2, J1, A1);
  and (J3, J2, A2);

  or (K0, Clear, J0);
  or (K1, Clear, J1);
  or (K2, Clear, J2);
  or (K3, Clear, J3);

  and (next_stage, A3, J3);
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
module JK_FF (output reg Q, input J, K, clock);
  always @ (posedge clock)
    case ({J,K})
      2'b00:  Q <= Q;
      2'b01:  Q <= 0;
      2'b10:  Q <= 1;
      2'b11:  Q <= ~Q;
    endcase
endmodule

module t_Counter_4bit_Synch_Clr ();
  wire [3: 0]  A;
  wire next_stage;
   reg Count_enable, Clear, clock;

  Counter_4bit_Synch_Clr M0 (A, next_stage, Count_enable, Clear, clock);

  initial #300 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial fork
    Clear = 1;
    Count_enable = 0;
    #12 Clear = 0;
    #20 Count_enable = 1;
    #180 Clear = 1;
    #190 Clear = 0;
    #230 Count_enable = 0;
  join
endmodule
```

Simulation results: synchronous clear.

(b)

```
module Counter_4bit_Asynch_Clr_b (
  output [3: 0]  A, output next_stage, input Count_enable, Clk, Clear_b
);
  wire A0, A1, A2, A3;
  assign A[3: 0] = {A3, A2, A1, A0};
```


All rights reserved.

```
  wire J0, K0, J1, K1, J2, K2, J3, K3;
  assign K0 = J0;
  assign K1 = J1;
  assign K2 = J2;
  assign K3 = J3;

  JK_FF M0 (A0, J0, K0, Clk, Clear_b);
  JK_FF M1 (A1, J1, K1, Clk, Clear_b);
  JK_FF M2 (A2, J2, K2, Clk, Clear_b);
  JK_FF M3 (A3, J3, K3, Clk, Clear_b);

  and (J0, Count_enable);
  and (J1, J0, A0);
  and (J2, J1, A1);
  and (J3, J2, A2);

  and (next_stage, A3, J3);
endmodule

module JK_FF (output reg Q, input J, K, clock, Clear_b);
  always @ (posedge clock, negedge Clear_b)
    if (Clear_b == 1'b0) 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_Counter_4bit_Asynch_Clr_b ();
  wire [3: 0]  A;
  wire next_stage;
  reg Count_enable, Clk, Clear_b;

  Counter_4bit_Asynch_Clr_b M0 (A, next_stage, Count_enable, Clk, Clear_b);

  initial #200 $finish;
  initial begin Clk = 0; forever #5 Clk = ~Clk; end
  initial fork
    Count_enable = 0;
    Clear_b = 0;
    #30 Count_enable = 1;
    #50 Clear_b = 1;
    #90 Count_enable = 0;
    #110 Count_enable= 1;
    #150 Clear_b = 0;
    #170 Clear_b = 1;
  join
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

8.13

```
// Structural description of design example (Fig. 8-10, 8-12)
module Design_Example_STR

  ( output  [3:0]    A,
    output           E, F,
    input            Start, clock, reset_b
  );

  Controller_STR M0 (clr_A_F, set_E, clr_E, set_F, incr_A, Start, A[2], A[3], clock, reset_b );
  Datapath_STR M1 (A, E, F, clr_A_F, set_E, clr_E, set_F, incr_A, clock);
endmodule

module Controller_STR
( output clr_A_F, set_E, clr_E, set_F, incr_A,
   input Start, A2, A3, clock, reset_b
);

  wire        G0, G1;
  parameter    S_idle = 2'b00, S_1 = 2'b01, S_2 = 2'b11;
  wire        w1, w2, w3;

  not (G0_b, G0);
  not (G1_b, G1);
  buf (incr_A, w2);
  buf (set_F, G1);
  not (A2_b, A2);
  or (D_G0, w1, w2);
  and (w1, Start, G0_b);
  and (clr_A_F, G0_b, Start);
  and (w2, G0, G1_b);
  and (set_E, w2, A2);
  and (clr_E, w2, A2_b);
  and (D_G1, w3, w2);
  and (w3, A2, A3);
  D_flip_flop_AR M0 (G0, D_G0, clock, reset_b);
  D_flip_flop_AR M1 (G1, D_G1, clock, reset_b);
endmodule

// datapath unit

module Datapath_STR
( output [3: 0] A,
  output    E, F,
  input      clr_A_F, set_E, clr_E, set_F, incr_A, clock
);

  JK_flip_flop_2 M0 (E, E_b, set_E, clr_E, clock);
  JK_flip_flop_2 M1 (F, F_b, set_F, clr_A_F, clock);
  Counter_4 M2 (A, incr_A, clr_A_F, clock);

endmodule

module Counter_4 (output reg [3: 0] A, input incr, clear, clock);
  always @ (posedge clock)
    if (clear)      A <= 0; else if (incr) A <= A + 1;
endmodule

module D_flip_flop_AR (Q, D, CLK, RST);
  output    Q;
  input D, CLK, RST;
  reg    Q;
```


All rights reserved.

```
  always @ (posedge CLK, negedge RST)
    if (RST == 0) Q <= 1'b0;
    else Q <= D;
endmodule

module JK_flip_flop_2 (Q, Q_not, J, K, CLK);
  output    Q, Q_not;
  input J, K, CLK;
  reg    Q;


  assign    Q_not = ~Q
;
  always @ (posedge CLK)
    case ({J, K})
      2'b00:  Q <= Q;
      2'b01:  Q <= 1'b0;
      2'b10:  Q <= 1'b1;
      2'b11:  Q <= ~Q;
    endcase
 endmodule

module t_Design_Example_STR;
  reg         Start, clock, reset_b;
  wire [3: 0]     A;
  wire        E, F;
  wire [1:0] state_STR = {M0.M0.G1, M0.M0.G0};


  Design_Example_STR M0 (A, E, F, Start, clock, reset_b);

  initial #500 $finish;
  initial
    begin
      reset_b = 0;
      Start = 0;
      clock = 0;
      #5 reset_b = 1; Start = 1;
      repeat (32)
        begin
          #5 clock = ~ clock;
        end
    end
  initial
  $monitor ("A = %b E = %b F = %b time = %0d", A, E, F, $time);
endmodule
```

The simulation results shown below match Fig. 8.13.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

8.14 The state code 2'b10 is unused. If the machine enters an unused state, the controller is written with default assignment to *next_state*. The default assignment forces the state to *S_idle*, so the machine recovers from the condition.

8.15 Modify the test bench to insert a reset event and extend the clock.

```
// RTL description of design example (see Fig.8-11)

module Design_Example_RTL (A, E, F, Start, clock, reset_b);

  // Specify ports of the top-level module of the design
  // See block diagram Fig. 8-10

  output [3: 0] A;
  output        E, F;
  input         Start, clock, reset_b;

  // Instantiate controller and datapath units

  Controller_RTL M0 (set_E, clr_E, set_F, clr_A_F, incr_A, A[2], A[3], Start, clock, reset_b );
  Datapath_RTL M1 (A, E, F, set_E, clr_E, set_F, clr_A_F, incr_A, clock);

endmodule

module Controller_RTL (set_E, clr_E, set_F, clr_A_F, incr_A, A2, A3, Start, clock, reset_b);
  output reg   set_E, clr_E, set_F, clr_A_F, incr_A;
  input      Start, A2, A3, clock, reset_b;
  reg [1:0]  state, next_state;
  parameter    S_idle = 2'b00, S_1 = 2'b01, S_2 = 2'b11;   // State codes

  always @ (posedge clock or negedge reset_b)       // State transitions (edge-sensitive)
    if (reset_b == 0) state <= S_idle;
    else state <= next_state;

  // Code next state logic directly from ASMD chart (Fig. 8-9d)

  always @ (state, Start, A2, A3 ) begin      // Next state logic (level-sensitive)
    next_state = S_idle;
    case (state)
```


All rights reserved.

```
      S_idle:    if (Start) next_state = S_1; else next_state = S_idle;
      S_1:       if (A2 & A3) next_state = S_2; else next_state = S_1;
      S_2:       next_state = S_idle;
      default:   next_state = S_idle;
    endcase
  end

  // Code output logic directly from ASMD chart (Fig. 8-9d)

  always @ (state, Start, A2) begin
    set_E = 0;        // default assignments; assign by exception
    clr_E = 0;
    set_F = 0;
    clr_A_F = 0;
    incr_A = 0;
    case (state)
      S_idle:          if (Start) clr_A_F = 1;
      S_1:        begin incr_A = 1; if (A2) set_E = 1; else clr_E = 1; end
      S_2:        set_F = 1;
    endcase
  end
endmodule

module  Datapath_RTL  (A, E, F, set_E, clr_E, set_F, clr_A_F, incr_A, clock);
  output reg [3: 0]     A;            // register for counter
  output reg        E, F;             // flags
  input             set_E, clr_E, set_F, clr_A_F, incr_A, clock;

  // Code register transfer operations directly from ASMD chart (Fig. 8-9d)

  always @ (posedge clock) begin
    if (set_E)                E <= 1;
    if (clr_E)                E <= 0;
    if (set_F)                F <= 1;
    if (clr_A_F)              begin A <= 0; F <= 0; end
    if (incr_A)               A <= A + 1;
  end
endmodule

module t_Design_Example_RTL;
  reg           Start, clock, reset_b;
  wire [3: 0]   A;
  wire          E, F;

  // Instantiate design example

  Design_Example_RTL M0 (A, E, F, Start, clock, reset_b);

  // Describe stimulus waveforms

  initial #500 $finish;      // Stopwatch
  initial fork
     #25 reset_b = 0;        // Test for recovery from reset on-the-fly.
     #27 reset_b = 1;
  join
  initial
    begin
     reset_b = 0;
     Start = 0;
     clock = 0;
     #5 reset_b = 1; Start = 1;
     //repeat (32)
```


All rights reserved.

```
    repeat (38)                 // Modify for test of reset_b on-the-fly
      begin
        #5 clock = ~ clock;     // Clock generator
      end
  end
 initial
   $monitor ("A = %b E = %b F = %b time = %0d", A, E, F, $time);
endmodule
```

8.16 RTL notation:

s0: (initial state) If *start* = 0 go back to state *s0*, If (*start* = 1) then *BR* ← *multiplicand*, *AR* ← *multiplier*, *PR* ← 0, go to *s1*.

s1: (check *AR* for Zero) *Zero* = 1 if *AR* = 0, if (*Zero* = 1) then go back to *s0* (*done*) If (*Zero* = 0) then go to *s1*, *PR* ← *PR* + *BR*, *AR* ← *AR* – 1.

The internal architecture of the datapath consists of a double-width register to hold the product (*PR*), a register to hold the multiplier (*AR*), a register to hold the multiplicand (*BR*), a double-width parallel adder, and single-width parallel adder. The single-width adder is used to implement the operation of decrementing the multiplier unit. Adding a word consisting entirely of 1s to the multiplier accomplishes the 2's complement subtraction of 1 from the multiplier. Figure 8.16 (a) below shows the ASMD chart, block diagram, and controller of the circuit. Figure 8.16 (b) shows the internal architecture of the datapath. Figure 8.16 (c) shows the results of simulating the circuit.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

(a) ASMD chart, block diagram, and controller

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

(b) Datapath

(c) Simulation results

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
module Prob_8_16_STR (
output [15: 0] PR, output done,
input [7: 0] data_AR, data_BR, input  start, clock, reset_b
);

Controller_P8_16 M0 (done, Ld_regs, Add_decr, start, zero, clock, reset_b);

 Datapath_P8_16 M1 (PR, zero, data_AR, data_BR, Ld_regs, Add_decr, clock, reset_b);
endmodule

module Controller_P8_16 (output done, output reg Ld_regs, Add_decr, input start, zero, clock, reset_b);
parameter  s0 = 1'b0, s1 = 1'b1;
reg state, next_state;
assign done = (state == s0);

always @ (posedge clock, negedge reset_b)
if (!reset_b) state <= s0; else state <= next_state;

always @ (state, start, zero) begin
Ld_regs = 0;
Add_decr = 0;
case (state)
s0:      if (start) begin Ld_regs = 1; next_state = s1; end
s1:      if (zero) next_state = s0; else  begin next_state = s1; Add_decr = 1; end
default:    next_state = s0;
endcase
end
endmodule

module Register_32 (output [31: 0] data_out, input [31: 0] data_in, input clock, reset_b);
Register_8 M3 (data_out [31: 24] , data_in  [31: 24], clock, reset_b);
Register_8 M2 (data_out [23: 16] , data_in  [23: 16], clock, reset_b);
Register_8 M1 (data_out [15: 8] , data_in  [15: 8], clock, reset_b);
Register_8 M0 (data_out [7: 0] , data_in  [7: 0], clock, reset_b);
endmodule

module Register_16 (output [15: 0] data_out, input [15: 0] data_in, input clock, reset_b);
Register_8 M1 (data_out [15: 8] , data_in  [15: 8], clock, reset_b);
Register_8 M0 (data_out [7: 0] , data_in  [7: 0], clock, reset_b);
endmodule

module Register_8 (output [7: 0] data_out, input [7: 0] data_in, input clock, reset_b);
D_flop M7 (data_out[7] data_in[7], clock, reset_b);
D_flop M6 (data_out[6] data_in[6], clock, reset_b);
D_flop M5 (data_out[5] data_in[5], clock, reset_b);
D_flop M4 (data_out[4] data_in[4], clock, reset_b);
D_flop M3 (data_out[3] data_in[3], clock, reset_b);
D_flop M2 (data_out[2] data_in[2], clock, reset_b);
D_flop M1 (data_out[1] data_in[1], clock, reset_b);
D_flop M0 (data_out[0] data_in[0], clock, reset_b);
endmodule

module Adder_32 (output c_out, output [31: 0]  sum, input [31: 0] a, b);
assign {c_out, sum} = a + b;
endmodule

module Adder_16 (output c_out, output [15: 0] sum, input [15: 0] a, b);
assign {c_out, sum} = a + b;
endmodule
```


All rights reserved.

```
module D_flop (output q, input data, clock, reset_b);
always @ (posedge clock, negedge reset_b)
if (!reset_b) q <= 0; else q <= data;
endmodule

module  Datapath_P8_16 (
output reg [15: 0] PR, output zero,
input [7: 0] data_AR, data_BR, input Ld_regs, Add_decr, clock, reset_b
);

reg [7: 0]   AR, BR;
assign      zero = ~( | AR);

always @ (posedge clock, negedge reset_b)
if (!reset_b) begin AR <= 8'b0; BR <= 8'b0; PR <= 16'b0; end
else begin
if (Ld_regs) begin AR <= data_AR; BR <= data_BR; PR <= 0; end
else if (Add_decr) begin PR <= PR + BR; AR <= AR -1; end
end
endmodule


// Test plan – Verify;
// Power-up reset
// Data is loaded correctly
// Control signals assert correctly
// Status signals assert correctly
// start is ignored while multiplying
// Multiplication is correct
// Recovery from reset on-the-fly

module t_Prob_P8_16;
wire         done;
wire [15: 0]     PR;
reg [7: 0]   data_AR, data_BR;
reg          start, clock, reset_b;

Prob_8_16_STR M0 (PR, done, data_AR, data_BR, start, clock, reset_b);

initial #500 $finish;
initial begin clock = 0; forever #5 clock = ~clock; end
initial fork
reset_b = 0;
#12 reset_b = 1;
#40 reset_b = 0;
#42 reset_b = 1;
#90 reset_b = 1;
#92 reset_b = 1;
join

initial fork
#20 start = 1;
#30 start = 0;
#40 start = 1;
#50 start = 0;
#120 start = 1;
#120 start = 0;
join
```


All rights reserved.

```
initial fork
data_AR = 8'd5;        // AR > 0
data_BR = 8'd20;

#80 data_AR = 8'd3;
#80 data_BR = 8'd9;

#100 data_AR = 8'd4;
#100 data_BR = 8'd9;
join
endmodule
```

8.17 $(2^n - 1)(2^n - 1) < (2^{2n} - 1)$ for $n \geq 1$

8.18 **(a)** The maximum product size is 32 bits available in registers *A* and *Q*.
(b) *P* counter must have 5 bits to load 16 (binary 10000) initially.
(c) *Z* (zero) detection is generated with a 5-input NOR gate.

8.19

Multiplicand $B = 11011_2 = 27_{10}$
Multiplier $Q = 10111_2 = 23_{10}$
Product: $CAQ = 621_{10}$

	C	***A***	***Q***	***P***
Multiplier in *Q*	0	00000	10111	101
*Q*0 = 1; add *B*		11011		
First partial product	0	11011	10111	100
Shift right *CAQ*	0	01101	11011	
*Q*0 = 1; add *B*		11011		
Second partial product	1	01000	11011	011
Shift right *CAQ*	0	10100	01101	
*Q*0 = 1; add *B*		11011		
Third partial product	1	01111	01101	010
Shift right *CAQ*	0	10111	10110	
Shift right *CAQ*	0	01011	11011	
Fourth partial product	0	01011	11011	001
*Q*0 = 1; add *B*		11011		
Fifth partial product	1	00110	11011	000
Shift right *CAQ*	0	10011	01101	
Final product in *AQ*:				
AQ = 10011_01101 = 621_{10}				

8.20 *S_idle* = 1t ns
The loop between *S_add* and *S_shift* takes 2nt ns)
Total time to multiply: $(2n + 1)$t

8.21

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

State codes:	G_1	G_0
S_idle	0	0
S_add	0	1
*S_shift*1	0	
unused	0	0

8.22 Note that the machine described by Fig. P8.22 requires four states, but the machine described byFig. 8.15 (b) requires only three. Also, observe that the sample simulation results show a case where the carry bit regsiter, C, is needed to support the addition operation. The datapath is 8 bits wide.

```
module Prob_8_22 # (parameter m_size = 9)
(
  output [2*m_size -1: 0] Product,
  output Ready,
  input [m_size -1: 0] Multiplicand, Multiplier,
  input Start, clock, reset_b
);
  wire [m_size -1: 0] A, Q;

  assign Product = {A, Q};
  wire Q0, Zero, Load_regs, Decr_P, Add_regs, Shift_regs;

Datapath_Unit M0 (A, Q, Q0, Zero, Multiplicand, Multiplier, Load_regs, Decr_P, Add_regs, Shift_regs,
clock, reset_b);
Control_Unit M1 (Ready, Decr_P, Load_regs, Add_regs, Shift_regs, Start, Q0, Zero, clock, reset_b);
endmodule
```


All rights reserved.

```
module Datapath_Unit # (parameter m_size = 9,  BC_size = 4)
(
  output reg [m_size -1: 0] A, Q,
  output Q0, Zero,
  input [m_size -1: 0] Multiplicand, Multiplier,
  input Load_regs, Decr_P,  Add_regs, Shift_regs, clock, reset_b
);
  reg C;
  reg [BC_size -1: 0] P;
  reg [m_size -1: 0] B;

  assign Q0 = Q[0];
  assign Zero = (P == 0);

  always @ (posedge clock, negedge reset_b)
    if (reset_b == 0) begin
    B <= 0;C <= 0;
    A <= 0;
    Q <= 0;
    P <= m_size;
  end
  else begin
    if (Load_regs) begin
      A <= 0;
      C <= 0;
      Q <= Multiplier;
      B <= Multiplicand;
      P <= m_size;
    end
    if (Decr_P) P <= P -1;
    if (Add_regs) {C, A} <= A + B;
    if (Shift_regs) {C, A, Q} <= {C, A, Q} >> 1;
  end
endmodule

module Control_Unit (
  output Ready, Decr_P, output reg Load_regs, Add_regs, Shift_regs, input Start, Q0, Zero, clock,
reset_b
);
  reg [ 1: 0]     state, next_state;
  parameter    S_idle = 2'b00, S_loaded = 2'b01, S_sum = 2'b10, S_shifted = 2'b11;
  assign Ready = (state == S_idle);
  assign Decr_P = (state == S_loaded);

  always @ (posedge clock, negedge reset_b)
    if (reset_b == 0) state <= S_idle; else state <= next_state;

  always @ (state, Start, Q0, Zero) begin
    next_state = S_idle;
     Load_regs = 0;
    Add_regs = 0;
    Shift_regs = 0;
    case (state)
      S_idle: if (Start == 0) next_state = S_idle; else begin next_state = S_loaded; Load_regs = 1; end
      S_loaded: if (Q0) begin next_state = S_sum; Add_regs = 1; end
      else begin next_state = S_shifted; Shift_regs = 1; end
      S_sum:     begin next_state = S_shifted; Shift_regs = 1; end
      S_shifted:  if (Zero) next_state = S_idle; else next_state = S_loaded;
    endcase
  end
endmodule
```


All rights reserved.

```
module t_Prob_8_22 ();
  parameter                 m_size = 9;          // Width of datapath
  wire   [2 * m_size - 1: 0]  Product;
  wire                      Ready;
  reg    [m_size - 1: 0]    Multiplicand, Multiplier;
  reg                       Start, clock, reset_b;
  integer                   Exp_Value;
  reg                       Error;

   Prob_8_22 M0 (Product, Ready, Multiplicand, Multiplier, Start, clock, reset_b);


  initial #140000 $finish;
  initial begin clock = 0; #5 forever #5 clock = ~clock; end
  initial fork
    reset_b = 1;

    #2 reset_b = 0;
    #3 reset_b = 1;
  join
  initial begin #5 Start = 1; end
    always @ (posedge Ready) begin
    Exp_Value = Multiplier * Multiplicand;
    //Exp_Value = Multiplier * Multiplicand +1;    // Inject error to confirm detection
  end
  always @ (negedge Ready) begin
    Error = (Exp_Value ^ Product) ;
  end

  initial begin
    #5 Multiplicand = 0;
    Multiplier = 0;
    repeat (64) #10 begin Multiplier = Multiplier + 1;
      repeat (64) @ (posedge M0.Ready) #5 Multiplicand = Multiplicand + 1;
    end
  end
endmodule
```


All rights reserved.

8.23 As shown in Fig. P8.23 the machine asserts *Load_regs* in state *S_load*. This will cause the machine to operate incorrectly. Once *Load_regs* is removed from *S_load* the machine operates correctly. The state *S_load* is a wasted state. Its removal leads to the same machine as dhown in Fig. P8.15b.

```
module Prob_8_23 # (parameter m_size = 9)
(
  output [2*m_size -1: 0] Product,
  output Ready,
  input [m_size -1: 0] Multiplicand, Multiplier,
  input Start, clock, reset_b
);
  wire [m_size -1: 0] A, Q;

  assign Product = {A, Q};
  wire Q0, Zero, Load_regs, Decr_P, Add_regs, Shift_regs;

Datapath_Unit M0 (A, Q, Q0, Zero, Multiplicand, Multiplier, Load_regs, Decr_P, Add_regs, Shift_regs,
clock, reset_b);
Control_Unit M1 (Ready, Decr_P, Shift_regs, Add_regs, Load_regs, Start, Q0, Zero, clock, reset_b);
endmodule

module Datapath_Unit # (parameter m_size = 9,  BC_size = 4)
(
  output reg [m_size -1: 0] A, Q,
  output Q0, Zero,
  input [m_size -1: 0] Multiplicand, Multiplier,
  input Load_regs, Decr_P,  Add_regs, Shift_regs, clock, reset_b
);
  reg C;
  reg [BC_size -1: 0] P;
  reg [m_size -1: 0] B;

  assign Q0 = Q[0];
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
  assign Zero = (P == 0);

  always @ (posedge clock, negedge reset_b)
    if (reset_b == 0) begin
    A <= 0;
    C <= 0;
    Q <= 0;
    B <= 0;
    P <= m_size;
  end
  else begin
    if (Load_regs) begin
      A <= 0;
      C <= 0;
      Q <= Multiplier;
      B <= Multiplicand;
      P <= m_size;
    end
    if (Decr_P) P <= P -1;
    if (Add_regs) {C, A} <= A + B;
    if (Shift_regs) {C, A, Q} <= {C, A, Q} >> 1;
  end
endmodule

module Control_Unit (
  output Ready, Decr_P, Shift_regs, output reg Add_regs, Load_regs, input Start, Q0, Zero, clock,
reset_b
);
  reg [ 1: 0]     state, next_state;
  parameter    S_idle = 2'b00, S_load = 2'b01, S_decr = 2'b10, S_shift = 2'b11;

  assign Ready = (state == S_idle);
  assign Shift_regs = (state == S_shift);
  assign Decr_P = (state == S_decr);

  always @ (posedge clock, negedge reset_b)
    if (reset_b == 0) state <= S_idle; else state <= next_state;

  always @ (state, Start, Q0, Zero) begin
    next_state = S_idle;
    Load_regs = 0;
    Add_regs = 0;
    case (state)
      S_idle: if (Start == 0) next_state = S_idle; else begin next_state = S_load; Load_regs = 1; end
      S_load:     begin next_state = S_decr; end
      S_decr:     begin next_state = S_shift; if (Q0) Add_regs = 1; end
      S_shift:    if (Zero) next_state = S_idle; else next_state = S_load;
    endcase
  end
endmodule

module t_Prob_8_23 ();
  parameter            m_size = 9;              // Width of datapath
  wire   [2 * m_size - 1: 0]      Product;
  wire                 Ready;
  reg    [m_size - 1: 0]          Multiplicand, Multiplier;
  reg                  Start, clock, reset_b;
  integer                   Exp_Value;
  reg                  Error;

   Prob_8_23 M0 (Product, Ready, Multiplicand, Multiplier, Start, clock, reset_b);
```


All rights reserved.

```
 initial #140000 $finish;
 initial begin clock = 0; #5 forever #5 clock = ~clock; end
 initial fork
   reset_b = 1;

   #2 reset_b = 0;
   #3 reset_b = 1;
 join
 initial begin #5 Start = 1; end
 always @ (posedge Ready) begin
   Exp_Value = Multiplier * Multiplicand;
   //Exp_Value = Multiplier * Multiplicand +1;     // Inject error to confirm detection
 end
 always @ (negedge Ready) begin
   Error = (Exp_Value ^ Product) ;
 end

 initial begin
   #5 Multiplicand = 0;
   Multiplier = 0;
   repeat (64) #10 begin Multiplier = Multiplier + 1;
     repeat (64) @ (posedge M0.Ready) #5 Multiplicand = Multiplicand + 1;
   end
 end
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

8.24

```
module Prob_8_24 # (parameter dp_width = 5)
(
  output    [2*dp_width - 1: 0]     Product,
  output                            Ready,
  input     [dp_width - 1: 0]       Multiplicand, Multiplier,
  input                             Start, clock, reset_b
);
  wire Load_regs, Decr_P, Add_regs, Shift_regs, Zero, Q0;

  Controller M0 (
    Ready, Load_regs, Decr_P, Add_regs, Shift_regs, Start, Zero, Q0,
    clock, reset_b
);

Datapath M1(Product, Q0, Zero,Multiplicand, Multiplier,
  Start, Load_regs, Decr_P, Add_regs, Shift_regs, clock, reset_b);
endmodule

module Controller (
  output Ready,
  output reg Load_regs, Decr_P, Add_regs, Shift_regs,
  input Start, Zero, Q0, clock, reset_b
);

  parameter       S_idle =    3'b001,          // one-hot code
                  S_add =     3'b010,
                  S_shift =   3'b100;
  reg    [2: 0]   state, next_state;       // sized for one-hot
  assign          Ready = (state == S_idle);

  always @ (posedge clock, negedge reset_b)
    if (~reset_b) state <= S_idle; else state <= next_state;

  always @ (state, Start, Q0, Zero) begin
    next_state = S_idle;
    Load_regs = 0;
    Decr_P = 0;
    Add_regs = 0;
    Shift_regs = 0;
    case (state)
      S_idle: if (Start) begin next_state = S_add; Load_regs = 1; end
      S_add:begin next_state = S_shift; Decr_P = 1; if (Q0) Add_regs = 1; end
      S_shift:     begin
        Shift_regs = 1;
        if (Zero) next_state = S_idle;
        else next_state = S_add;
                    end
     default:    next_state = S_idle;
    endcase
  end
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
module Datapath #(parameter dp_width = 5, BC_size = 3) (
  output [2*dp_width - 1: 0] Product, output Q0, output Zero,
  input [dp_width - 1: 0] Multiplicand, Multiplier,
  input Start, Load_regs, Decr_P, Add_regs, Shift_regs, clock, reset_b
);
// Default configuration: 5-bit datapath
  reg    [dp_width - 1: 0]        A, B, Q;                    // Sized for datapath
  reg                     C;
  reg    [BC_size - 1: 0]         P;                  // Bit counter

  assign Q0 = Q[0];
  assign    Zero = (P == 0);              // Counter is zero
  assign Product = {C, A, Q};
  always @ (posedge clock, negedge reset_b)
    if (reset_b == 0) begin           // Added to this solution, but
      P <= dp_width;                        // not really necessary since Load_regs
      B <= 0;                               // initializes the datapath
      C <= 0;
      A <= 0;
      Q <= 0;
    end
    else begin
    if (Load_regs) begin
      P <= dp_width;
      A <= 0;
      C <= 0;
      B <= Multiplicand;
      Q <= Multiplier;
    end
    if (Add_regs) {C, A} <= A + B;
    if (Shift_regs) {C, A, Q} <= {C, A, Q} >> 1;
    if (Decr_P) P <= P -1;
  end
endmodule

module t_Prob_8_24;
  parameter                       dp_width = 5;             // Width of datapath
  wire   [2 * dp_width - 1: 0]    Product;
  wire                            Ready;
  reg    [dp_width - 1: 0]        Multiplicand, Multiplier;
  reg                             Start, clock, reset_b;
  integer                         Exp_Value;
  reg                             Error;

  Prob_8_24 M0(Product, Ready, Multiplicand, Multiplier, Start, clock, reset_b);

  initial #115000 $finish;
  initial begin clock = 0; #5 forever #5 clock = ~clock; end
  initial fork
    reset_b = 1;
    #2 reset_b = 0;
    #3 reset_b = 1;
  join
  always @ (negedge Start) begin
    Exp_Value = Multiplier * Multiplicand;
    //Exp_Value = Multiplier * Multiplicand +1;     // Inject error to confirm detection
  end
  always @ (posedge Ready) begin
    # 1 Error <= (Exp_Value ^ Product) ;
  end

  initial begin
```


All rights reserved.

```
  #5 Multiplicand = 0;
  Multiplier = 0;

  repeat (32) #10 begin
    Start = 1;
     #10 Start = 0;
     repeat (32) begin
       Start = 1;
       #10 Start = 0;
       #100 Multiplicand = Multiplicand + 1;
     end
    Multiplier = Multiplier + 1;
  end
 end
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

8.25 **(a)**

(b)

```
// The multiplier of Fig. 8.15 is modified to detect whether the multiplier or multiplicand are initially zero,
// and to detect whether the multiplier becomes zero before the entire multiplier has been applied
// to the multiplicand. Signal empty is generated by the datapath unit and used by the
// controller. Note that the bits of the product must be selected according to the stage at which
// termination occurs.  The test for the condition of an empty multiplier is hardwired here for
// dp_width = 5 because the range bounds of a vector must be defined by integer constants.
// This prevents development of a fully parameterized model.
// Note: the test bench has been modified.

module Prob_8_25 #(parameter dp_width = 5)
(
  output   [2*dp_width - 1: 0]     Product,
  output                           Ready,
  input    [dp_width - 1: 0]       Multiplicand, Multiplier,
  input                            Start, clock, reset_b
);
  wire Load_regs, Decr_P, Add_regs, Shift_regs, Empty, Zero, Q0;
  Controller M0 (
    Ready, Load_regs, Decr_P, Add_regs, Shift_regs, Start, Empty, Zero, Q0,
    clock, reset_b
);

Datapath M1(Product, Q0, Empty, Zero,Multiplicand, Multiplier,
  Start, Load_regs, Decr_P, Add_regs, Shift_regs, clock, reset_b);

endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
module Controller (
  output Ready,
  output reg Load_regs, Decr_P, Add_regs, Shift_regs,
  input Start, Empty, Zero, Q0, clock, reset_b
);

  parameter        BC_size =      3;        // Size of bit counter
  parameter        S_idle =    3'b001,          // one-hot code
                   S_add =     3'b010,
                   S_shift =   3'b100;
  reg    [2: 0]    state, next_state;       // sized for one-hot
  assign           Ready = (state == S_idle);

  always @ (posedge clock, negedge reset_b)
    if (~reset_b) state <= S_idle; else state <= next_state;

  always @ (state, Start, Q0, Empty, Zero) begin
    next_state = S_idle;
    Load_regs = 0;
    Decr_P = 0;
    Add_regs = 0;
    Shift_regs = 0;
    case (state)
      S_idle:     if (Start) begin next_state = S_add; Load_regs = 1; end
      S_add:      begin next_state = S_shift; Decr_P = 1; if (Q0) Add_regs = 1; end
      S_shift:    begin
                    Shift_regs = 1;
                    if (Zero) next_state = S_idle;
                    else if (Empty) next_state = S_idle;
                    else next_state = S_add;
                  end
      default:    next_state = S_idle;
    endcase
  end
endmodule

module Datapath #(parameter dp_width = 5, BC_size = 3) (
  output reg [2*dp_width - 1: 0] Product, output Q0, output Empty, output Zero,
  input [dp_width - 1: 0] Multiplicand, Multiplier,
  input Start, Load_regs, Decr_P, Add_regs, Shift_regs, clock, reset_b
);
// Default configuration: 5-bit datapath
  parameter        S_idle =    3'b001,          // one-hot code
                   S_add =     3'b010,
                   S_shift =   3'b100;
  reg    [dp_width - 1: 0]     A, B, Q;                    // Sized for datapath
  reg                          C;
  reg    [BC_size - 1: 0]      P;                  // Bit counter
  wire   [2*dp_width -1: 0]    Internal_Product = {C, A, Q};

  assign         Q0 = Q[0];
  assign         Zero = (P == 0);                  // Bit counter is zero

  always @ (posedge clock, negedge reset_b)
    if (reset_b == 0) begin              // Added to this solution, but
      P <= dp_width;                     // not really necessary since Load_regs
      B <= 0;                            // initializes the datapath
      C <= 0;
      A <= 0;
      Q <= 0;
    end
    else begin
```


All rights reserved.

```
    if (Load_regs) begin
      P <= dp_width;
      A <= 0;
      C <= 0;
      B <= Multiplicand;
      Q <= Multiplier;
    end
    if (Add_regs) {C, A} <= A + B;
    if (Shift_regs) {C, A, Q} <= {C, A, Q} >> 1;
    if (Decr_P) P <= P -1;
  end
  // Status signals
  reg    Empty_multiplier;
  wire   Empty_multiplicand = (Multiplicand == 0);
  assign Empty = Empty_multiplicand || Empty_multiplier;

  always @ (P, Internal_Product) begin// Note: hardwired for dp_width  5
    Product = 0;
    case (P)          // Examine multiplier bits
      0:  Product  = Internal_Product;
      1:  Product = Internal_Product [2*dp_width -1: 1];
      2:  Product = Internal_Product [2*dp_width -1: 2];
      3:  Product = Internal_Product [2*dp_width -1: 3];
      4:  Product = Internal_Product [2*dp_width -1: 4];
      5:  Product = 0;
    endcase
  end

  always @ (P, Q) begin                // Note: hardwired for dp_width  5
    Empty_multiplier = 0;
    case (P)
      0:  Empty_multiplier = 1;
      1:  if (Q[1] == 0) Empty_multiplier = 1;
      2:  if (Q[2: 1] == 0) Empty_multiplier = 1;
      3:   if (Q[3: 1] == 0) Empty_multiplier = 1;
      4:  if (Q[4: 1] == 0) Empty_multiplier = 1;
      5:   if (Q[5: 1] == 0) Empty_multiplier = 1;
      default: Empty_multiplier = 1'bx;
    endcase
  end
endmodule

module t_Prob_8_25;
  parameter          dp_width = 5;          // Width of datapath
  wire  [2 * dp_width - 1: 0]   Product;
  wire               Ready;
  reg   [dp_width - 1: 0]       Multiplicand, Multiplier;
  reg                Start, clock, reset_b;
  integer                Exp_Value;
  reg                Error;

  Prob_8_25 M0(Product, Ready, Multiplicand, Multiplier, Start, clock, reset_b);

  initial #115000 $finish;
  initial begin clock = 0; #5 forever #5 clock = ~clock; end
  initial fork
    reset_b = 1;
    #2 reset_b = 0;
    #3 reset_b = 1;
  join
  always @ (negedge Start) begin
    Exp_Value = Multiplier * Multiplicand;
```


All rights reserved.

```
  //Exp_Value = Multiplier * Multiplicand +1;    // Inject error to confirm detection
 end
 always @ (posedge Ready) begin
  # 1 Error <= (Exp_Value ^ Product) ;
 end

 initial begin
  #5 Multiplicand = 0;
  Multiplier = 0;

  repeat (32) #10 begin
    Start = 1;
    #10 Start = 0;
    repeat (32) begin
      Start = 1;
      #10 Start = 0;
      #100 Multiplicand = Multiplicand + 1;
    end
   Multiplier = Multiplier + 1;
  end
 end
endmodule
```

(c) Test plan: Exhaustively test all combinations of multiplier and multiplicand, using automatic error checking. Verify that early termination is implemented. Sample of simulation results is shown below.

8.26

All rights reserved.

```
module Prob_8_26 (Product, Ready, Multiplicand, Multiplier, Start, clock, reset_b);
// Default configuration: 5-bit datapath
  parameter                dp_width = 5;      // Set to width of datapath
  output   [2*dp_width - 1: 0]   Product;
  output                         Ready;
  input        [dp_width - 1: 0]  Multiplicand, Multiplier;
  input                          Start, clock, reset_b;
  parameter                      BC_size =     3;       // Size of bit counter
  parameter                      S_idle =   2'b01,      // one-hot code
                                 S_add_shift =     2'b10;

  reg   [2: 0]                   state, next_state;
  reg   [dp_width - 1: 0]        A, B, Q;                // Sized for datapath
  reg                            C;
  reg   [BC_size -1: 0]          P;
  reg              Load_regs, Decr_P, Add_shift, Shift;
  assign           Product = {C, A, Q};
  wire             Zero = (P == 0);          // counter is zero
  wire             Ready = (state == S_idle);   // controller status

// control unit
  always @ (posedge clock, negedge reset_b)
    if (~reset_b) state <= S_idle; else state <= next_state;

  always @ (state, Start, Q[0], Zero) begin
    next_state = S_idle;
    Load_regs = 0;
    Decr_P = 0;
    Add_shift = 0;
    Shift = 0;
    case (state)
      S_idle:            begin if (Start) next_state = S_add_shift; Load_regs = 1; end
      S_add_shift:       begin
                           Decr_P = 1;
                           if (Zero) next_state = S_idle;
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
                                else begin
                                  next_state = S_add_shift;
                                  if (Q[0]) Add_shift = 1; else Shift = 1;
                                end
                              end
      default:                next_state = S_idle;
    endcase
  end

// datapath unit
  always @ (posedge clock) begin
    if (Load_regs) begin
      P <= dp_width;
      A <= 0;
      C <= 0;
      B <= Multiplicand;
     Q <= Multiplier;
    end
    if (Decr_P) P <= P -1;
    if (Add_shift) {C, A, Q} <= {C, A+B, Q} >> 1;
    if (Shift) {C, A, Q} <= {C, A, Q} >> 1;
  end
endmodule

module t_Prob_8_26;
  parameter          dp_width = 5;          // Width of datapath
  wire   [2 * dp_width - 1: 0]   Product;
  wire                           Ready;
  reg    [dp_width - 1: 0]       Multiplicand, Multiplier;
  reg                            Start, clock, reset_b;
  integer                        Exp_Value;
  wire                           Error;

   Prob_8_26 M0 (Product, Ready, Multiplicand, Multiplier, Start, clock, reset_b);

  initial #70000 $finish;
  initial begin clock = 0; #5 forever #5 clock = ~clock; end
  initial fork
    reset_b = 1;
    #2 reset_b = 0;
    #3 reset_b = 1;
  join
  initial begin #5 Start = 1; end
  always @ (posedge Ready) begin
    Exp_Value = Multiplier * Multiplicand;
  end
  assign Error = Ready & (Exp_Value ^ Product);
  initial begin
    #5 Multiplicand = 0;
    Multiplier = 0;
    repeat (32) #10 begin Multiplier = Multiplier + 1;
      repeat (32) @ (posedge M0.Ready) #5 Multiplicand = Multiplicand + 1;
    end
  end
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

Sample of simulation results.

8.27 **(a)**

```
// Test bench for exhaustive simulation
module t_Sequential_Binary_Multiplier;
  parameter                 dp_width = 5;          // Width of datapath
  wire  [2 * dp_width - 1: 0]  Product;
  wire                      Ready;
  reg   [dp_width - 1: 0]   Multiplicand, Multiplier;
  reg                       Start, clock, reset_b;

   Sequential_Binary_Multiplier M0 (Product, Ready, Multiplicand, Multiplier, Start, clock, reset_b);

  initial #109200 $finish;
  initial begin clock = 0; #5 forever #5 clock = ~clock; end
  initial fork
    reset_b = 1;
    #2 reset_b = 0;
    #3 reset_b = 1;
  join
  initial begin #5 Start = 1; end
  initial begin
    #5 Multiplicand = 0;
    Multiplier = 0;
      repeat (31) #10 begin Multiplier = Multiplier + 1;
        repeat (32) @ (posedge M0.Ready) #5 Multiplicand = Multiplicand + 1;
      end
    Start = 0;
  end

// Error Checker
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
  reg Error;
  reg [2*dp_width -1: 0] Exp_Value;
  always @ (posedge Ready) begin
    Exp_Value = Multiplier * Multiplicand;
    //Exp_Value = Multiplier * Multiplicand + 1;        // Inject error to verify detection
    Error = (Exp_Value ^ Product);
  end
endmodule

 module Sequential_Binary_Multiplier (Product, Ready, Multiplicand, Multiplier, Start, clock, reset_b);
// Default configuration: 5-bit datapath
  parameter                    dp_width = 5;       // Set to width of datapath
  output    [2*dp_width - 1: 0]  Product;
  output                         Ready;
  input     [dp_width - 1: 0]    Multiplicand, Multiplier;
  input                          Start, clock, reset_b;

  parameter        BC_size =     3;   // Size of bit counter
  parameter        S_idle =      3'b001,     // one-hot code
                   S_add =       3'b010,
                   S_shift =     3'b100;

  reg    [2: 0]                  state, next_state;
  reg    [dp_width - 1: 0]       A, B, Q;                // Sized for datapath
  reg                            C;
  reg    [BC_size - 1: 0]        P;
  reg                            Load_regs, Decr_P, Add_regs, Shift_regs;

// Miscellaneous combinational logic

  assign           Product = {C, A, Q};
  wire             Zero = (P == 0);            // counter is zero
  wire             Ready = (state == S_idle);          // controller status

// control unit

  always @ (posedge clock, negedge reset_b)
    if (~reset_b) state <= S_idle; else state <= next_state;

  always @ (state, Start, Q[0], Zero) begin
    next_state = S_idle;
    Load_regs = 0;
    Decr_P = 0;
    Add_regs = 0;
    Shift_regs = 0;
    case (state)
      S_idle: begin if (Start) next_state = S_add; Load_regs = 1; end
      S_add:begin next_state = S_shift; Decr_P = 1; if (Q[0]) Add_regs = 1; end
      S_shift:     begin Shift_regs = 1; if (Zero) next_state = S_idle;
      else next_state = S_add; end
      default:    next_state = S_idle;
    endcase
  end
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
// datapath unit

  always @ (posedge clock) begin
    if (Load_regs) begin
      P <= dp_width;
      A <= 0;
      C <= 0;
      B <= Multiplicand;
      Q <= Multiplier;
    end
    if (Add_regs) {C, A} <= A + B;
    if (Shift_regs) {C, A, Q} <= {C, A, Q} >> 1;
    if (Decr_P) P <= P -1;
  end
endmodule
```

Sample of simulation results:

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

(b) In this part the controller is described by Fig. 8.18. The test bench includes probes to display the state of the controller.

```
// Test bench for exhaustive simulation
module t_Sequential_Binary_Multiplier;
  parameter                 dp_width = 5;          // Width of datapath
  wire  [2 * dp_width - 1: 0]  Product;
  wire                      Ready;
  reg   [dp_width - 1: 0]   Multiplicand, Multiplier;
  reg                       Start, clock, reset_b;

   Sequential_Binary_Multiplier M0 (Product, Ready, Multiplicand, Multiplier, Start, clock, reset_b);

  initial #109200 $finish;
  initial begin clock = 0; #5 forever #5 clock = ~clock; end
  initial fork
    reset_b = 1;
    #2 reset_b = 0;
    #3 reset_b = 1;
  join
  initial begin #5 Start = 1; end
  initial begin
    #5 Multiplicand = 0;
    Multiplier = 0;
    repeat (31) #10 begin Multiplier = Multiplier + 1;
      repeat (32) @ (posedge M0.Ready) #5 Multiplicand = Multiplicand + 1;
    end
    Start = 0;
  end

// Error Checker
  reg Error;
  reg [2*dp_width -1: 0] Exp_Value;
  always @ (posedge Ready) begin
    Exp_Value = Multiplier * Multiplicand;
    //Exp_Value = Multiplier * Multiplicand + 1;      // Inject error to verify detection
    Error = (Exp_Value ^ Product);
  end

  wire [2: 0] state = {M0.G2, M0.G1, M0.G0};
endmodule

 module Sequential_Binary_Multiplier (Product, Ready, Multiplicand, Multiplier, Start, clock, reset_b);
// Default configuration: 5-bit datapath
  parameter                 dp_width =     5;      // Set to width of datapath
  output    [2*dp_width - 1: 0]  Product;
  output                    Ready;
  input     [dp_width - 1: 0]  Multiplicand, Multiplier;
  input                     Start, clock, reset_b;

  parameter                 BC_size =      3;  // Size of bit counter
  reg   [dp_width - 1: 0]   A, B, Q;            // Sized for datapath
  reg                       C;
  reg   [BC_size - 1: 0]    P;
  wire                      Load_regs, Decr_P, Add_regs, Shift_regs;
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
// Status signals

  assign        Product = {C, A, Q};
  wire          Zero = (P == 0);          // counter is zero
  wire          Q0 = Q[0];

// One-Hot Control unit (See Fig. 8.18)
  DFF_S M0 (G0, D0, clock, Set);
  DFF M1 (G1, D1, clock, reset_b);
  DFF M2 (G2, G1, clock, reset_b);
  or (D0, w1, w2);
  and (w1, G0, Start_b);
  and (w2, Zero, G2);
  not (Start_b, Start);
  not (Zero_b, Zero);
  or (D1, w3, w4);
  and (w3, Start, G0);
  and (w4, Zero_b, G2);

  and (Load_regs, G0, Start);
  and (Add_regs, Q0, G1);
  assign Ready = G0;
  assign Decr_P = G1;
  assign Shift_regs = G2;
  not (Set, reset_b);

// datapath unit

  always @ (posedge clock) begin
    if (Load_regs) begin
      P <= dp_width;
      A <= 0;
      C <= 0;
      B <= Multiplicand;
      Q <= Multiplier;
    end
    if (Add_regs) {C, A} <= A + B;
    if (Shift_regs) {C, A, Q} <= {C, A, Q} >> 1;
    if (Decr_P) P <= P -1;
  end
endmodule

module DFF_S (output reg Q, input data, clock, Set);
  always @ ( posedge clock, posedge Set)
    if (Set) Q <= 1'b1; else Q<= data;
endmodule
module DFF (output reg Q, input data, clock, reset_b);
  always @ ( posedge clock, negedge reset_b)
    if (reset_b == 0) Q <= 1'b0; else Q<= data;
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

Sample of simulation results:

8.28

```
// Test bench for exhaustive simulation
module t_Sequential_Binary_Multiplier;
  parameter                 dp_width = 5;          // Width of datapath
  wire  [2 * dp_width - 1: 0]  Product;
  wire                      Ready;
  reg   [dp_width - 1: 0]   Multiplicand, Multiplier;
  reg                       Start, clock, reset_b;

   Sequential_Binary_Multiplier M0 (Product, Ready, Multiplicand, Multiplier, Start, clock, reset_b);

  initial #109200 $finish;
  initial begin clock = 0; #5 forever #5 clock = ~clock; end
  initial fork
   reset_b = 1;
   #2 reset_b = 0;
   #3 reset_b = 1;
  join
  initial begin #5 Start = 1; end
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
  initial begin
    #5 Multiplicand = 0;
    Multiplier = 0;
    repeat (31) #10 begin Multiplier = Multiplier + 1;
      repeat (32) @ (posedge M0.Ready) #5 Multiplicand = Multiplicand + 1;
    end
    Start = 0;
  end

// Error Checker
  reg Error;
  reg [2*dp_width -1: 0] Exp_Value;
  always @ (posedge Ready) begin
    Exp_Value = Multiplier * Multiplicand;
    //Exp_Value = Multiplier * Multiplicand + 1;        // Inject error to verify detection
    Error = (Exp_Value ^ Product);
  end
  wire [2: 0] state = {M0.M0.G2, M0.M0.G1, M0.M0.G0};   // Watch state
endmodule

 module Sequential_Binary_Multiplier
   #(parameter dp_width =    5)
(
  output [2*dp_width -1: 0]   Product,
  output                      Ready,
  input [dp_width -1: 0]      Multiplicand, Multiplier,
  input                       Start, clock, reset_b
);
   wire Load_regs, Decr_P, Add_regs, Shift_regs, Zero, Q0;

  Controller M0 (Ready, Load_regs, Decr_P, Add_regs, Shift_regs, Start, Zero, Q0, clock, reset_b);
  Datapath M1(Product, Q0, Zero,Multiplicand, Multiplier, Start, Load_regs, Decr_P, Add_regs,
    Shift_regs, clock, reset_b);
endmodule

module Controller (
  output Ready,
  output Load_regs, Decr_P, Add_regs, Shift_regs,
  input Start, Zero, Q0, clock, reset_b
);
// One-Hot Control unit (See Fig. 8.18)
  DFF_S M0 (G0, D0, clock, Set);
  DFF M1 (G1, D1, clock, reset_b);
  DFF M2 (G2, G1, clock, reset_b);
  or (D0, w1, w2);
  and (w1, G0, Start_b);
  and (w2, Zero, G2);
  not (Start_b, Start);
  not (Zero_b, Zero);
  or (D1, w3, w4);
  and (w3, Start, G0);
  and (w4, Zero_b, G2);

  and (Load_regs, G0, Start);
  and (Add_regs, Q0, G1);
  assign Ready = G0;
  assign Decr_P = G1;
  assign Shift_regs = G2;
  not (Set, reset_b);
endmodule
```


All rights reserved.

```
module Datapath #(parameter dp_width = 5, BC_size = 3) (
  output [2*dp_width - 1: 0] Product, output Q0, output Zero,
  input [dp_width - 1: 0] Multiplicand, Multiplier,
  input Start, Load_regs, Decr_P, Add_regs, Shift_regs, clock, reset_b
);
  reg    [dp_width - 1: 0]        A, B, Q;              // Sized for datapath
  reg                    C;
  reg    [BC_size - 1: 0]         P;
  assign                   Product = {C, A, Q};
  // Status signals
  assign                   Zero = (P == 0);        // counter is zero
  assign                   Q0 = Q[0];

  always @ (posedge clock) begin
    if (Load_regs) begin
      P <= dp_width;
      A <= 0;
      C <= 0;
      B <= Multiplicand;
      Q <= Multiplier;
    end
    if (Add_regs) {C, A} <= A + B;
    if (Shift_regs) {C, A, Q} <= {C, A, Q} >> 1;
    if (Decr_P) P <= P -1;
  end
endmodule

module DFF_S (output reg Q, input data, clock, Set);
  always @ ( posedge clock, posedge Set)
    if (Set) Q <= 1'b1; else Q<= data;
endmodule
module DFF (output reg Q, input data, clock, reset_b);
  always @ ( posedge clock, negedge reset_b)
    if (reset_b == 0) Q <= 1'b0; else Q<= data;
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

8.29 **(a)**

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

(b)

$DS_0 = x'y'S_0 + S_3 + S_5 + S_7$
$DS_1 = xS_0$
$DS_2 = x'yS_0 + S_1$
$DS_3 = FS_2$
$DS_4 = F'S_2$
$DS_5 = E'S_5$
$DS_6 = E'S_4$
$DS_7 = S_6$

(c)

Output	Present state $G_1\,G_2\,G_3$	Inputs $x\ y\ E\ F$	Next state $G_1\,G_2\,G_3$
$S0$	0 0 0	0 0 x x	0 0 0
$S0$	0 0 0	1 x x x	0 0 1
$S0$	0 0 0	0 1 x x	0 1 0
$S1$	0 0 1	x x x x	0 1 0
$S2$	0 1 0	x x 0 x	1 0 0
$S2$	0 1 0	x x 1 x	0 1 1
$S3$	0 1 1	x x x x	0 0 0
$S4$	1 0 0	x x x 0	1 1 0
S4	1 0 0	x x x 1	1 0 1
$S5$	1 0 1	x x x x	0 0 0
$S6$	1 1 0	x x x x	1 1 0
$S7$	1 1 1	x x x x	0 0 0

(d)

$DG_1 = F'S_2 + S_4 + S_6$
$DG_2 = x'yS_0 + S_1 + FS_2 + E'S_4 + S_6$
$DG_3 = xS_0 + FS_2 + ES_4 + S_6$

(e)

All rights reserved.

Present state $G_1\,G_2\,G_3$	Next state $G_1\,G_2\,G_3$	Input conditions	Mux1	Mux2	Mux3
0 0 0 0 0 0 0 0 0	0 0 0 0 0 1 0 1 0	$x'y'$ x $x'y$	0	$x'y$	x
0 0 1	0 1 0	None	0	1	0
0 1 0 0 1 0	1 0 0 0 1 1	F' F'	F'	F	F
0 1 1	0 0 0	None	0	0	0
1 0 0 1 0 0	1 1 0 1 0 1	E' E'	1	E'	E
1 0 1	0 0 0	None	0	0	0
1 1 0	1 1 0	None	1	1	1
1 1 1	0 0 0	None	0	0	0

(f)

(g)

```
module Controller_8_29g (input  x, y, E, F, clock, reset_b);
  supply0 GND;
  supply1 VCC;

  mux_8x1 M3 (m3, GND, GND, F_bar, GND, VCC, GND, VCC, GND, G3, G2, G1);
  mux_8x1 M2 (m2, w1, VCC, F, GND, E_bar, GND, VCC, GND, G3, G2, G1);
  mux_8x1 M1 (m1, x, GND, F, GND, E, GND, VCC, GND, G3, G2, G1);
  DFF_8_28g DM3 (G3, m3, clock, reset_b);
  DFF_8_28g DM2 (G2, m2, clock, reset_b);
  DFF_8_28g DM1 (G1, m1, clock, reset_b);
  decoder_3x8 M0_D (y0, y1, y2, y3, y4, y5, y6, y7, G3, G2, G1);
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
  and (w1, x_bar, y);
  not (F_bar, F);
  not (E_bar, E);
  not (x_bar, x);
endmodule

// Test plan: Exercise all paths of the ASM chart

module t_Controller_8_29g ();
  reg  x, y, E, F, clock, reset_b;
  Controller_8_29g M0 (x, y, E, F, clock, reset_b);
  wire [2: 0] state = {M0.G3, M0.G2, M0.G1};

  initial #500 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial begin end
  initial fork
    reset_b = 0; #2 reset_b = 1;
    #0 begin x = 1; y = 1; E = 1; F = 1; end   // Path: S_0, S_1, S_2, S_34
    #80 reset_b = 0; #92 reset_b = 1;
    #90 begin x = 1; y = 1; E = 1; F = 0; end
    #150 reset_b = 0;
    #152 reset_b = 1;
    #150 begin x = 1; y = 1; E = 0; F = 0; end    // Path: S_0, S_1, S_2, S_4, S_5
    #200 reset_b = 0;
    #202 reset_b = 1;
    #190 begin x = 1; y = 1; E = 0; F = 0; end    // Path: S_0, S_1, S_2, S_4, S_6, S_7
    #250 reset_b = 0;
    #252 reset_b = 1;
    #240 begin x = 0; y = 0; E = 0; F = 0; end    // Path: S_0
    #290 reset_b = 0;
    #292 reset_b = 1;
    #280 begin x = 0; y = 1; E = 0; F = 0; end    // Path: S_0, S_2, S_4, S_6, S_7
    #360 reset_b = 0;
    #362 reset_b = 1;
    #350 begin x = 0; y = 1; E = 1; F = 0; end    // Path: S_0, S_2, S_4, S_5
    #420 reset_b = 0;
    #422 reset_b = 1;
    #410 begin x = 0; y = 1; E = 0; F = 1; end    // Path: S_0, S_2, S_3
  join
endmodule

module  mux_8x1 (output reg y, input x0, x1, x2, x3, x4, x5, x6, x7, s2, s1, s0);
  always @ (x0, x1, x2, x3, x4, x5, x6, x7, s0, s1, s2)
    case ({s2, s1, s0})
      3'b000: y = x0;
      3'b001: y = x1;
      3'b010: y = x2;
      3'b011: y = x3;
      3'b100: y = x4;
      3'b101: y = x5;
      3'b110: y = x6;
      3'b111: y = x7;
    endcase
endmodule

module DFF_8_28g (output reg q, input data, clock, reset_b);
  always @ (posedge clock, negedge reset_b)
    if (!reset_b) q <= 1'b0; else q <= data;
endmodule

module  decoder_3x8 (output reg y0, y1, y2, y3, y4, y5, y6, y7, input x2, x1, x0);
```


All rights reserved.

```
  always @ (x0, x1, x2) begin
    {y7, y6, y5, y4, y3, y2, y1, y0} = 8'b0;
    case ({x2, x1, x0})
      3'b000: y0= 1'b1;
      3'b001: y1= 1'b1;
      3'b010: y2= 1'b1;
      3'b011: y3= 1'b1;
      3'b100: y4= 1'b1;
      3'b101: y5= 1'b1;
      3'b110: y6= 1'b1;
      3'b111: y7= 1'b1;
    endcase
  end
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

Path: S_0, S_1, S_2, S_3 and Path: S_0, S_1, S_2, S_4, S_5

Path: S_0, S_1, S_2, S_4, S_6, S_7

Path: S_0 and Path , S_0, S_2, S_4, S_6, S_7

Path: S_0, S_2, S_4, S_5 and path S_0, S_2, S_3

(h)

```
module Controller_8_29h (input  x, y, E, F, clock, reset_b);
 parameter  S_0 = 3'b000, S_1 = 3'b001, S_2 = 3'b010,
    S_3 = 3'b011, S_4 = 3'b100, S_5 = 3'b101, S_6 = 3'b110, S_7 = 3'b111;
 reg [2: 0 ] state, next_state;

 always @ (posedge clock, negedge reset_b)
  if (!reset_b) state <= S_0; else state <= next_state;
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
 always @ (state, x, y, E, F) begin
   case (state)
     S_0:                 if (x) next_state = S_1;
                          else next_state =  y ? S_2: S_0;
     S_1:                 next_state = S_2;
     S_2:                 if (F) next_state = S_3; else next_state = S_4;
     S_3, S_5, S_7:       next_state = S_0;
     S_4:                 if (E) next_state = S_5; else next_state = S_6;
     S_6:                 next_state = S_7;
     default:             next_state = S_0;
   endcase
  end
endmodule

// Test plan: Exercise all paths of the ASM chart

module t_Controller_8_29h ();
  reg  x, y, E, F, clock, reset_b;

  Controller_8_29h M0 (x, y, E, F, clock, reset_b);

  initial #500 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial begin end
  initial fork
    reset_b = 0; #2 reset_b = 1;
    #20 begin x = 1; y = 1; E = 1; F = 1; end // Path: S_0, S_1, S_2, S_34
    #80 reset_b = 0; #92 reset_b = 1;
    #90 begin x = 1; y = 1; E = 1; F = 0; end
    #150 reset_b = 0;
    #152 reset_b = 1;
    #150 begin x = 1; y = 1; E = 0; F = 0; end    // Path: S_0, S_1, S_2, S_4, S_5
    #200 reset_b = 0;
    #202 reset_b = 1;
    #190 begin x = 1; y = 1; E = 0; F = 0; end    // Path: S_0, S_1, S_2, S_4, S_6, S_7
    #250 reset_b = 0;
    #252 reset_b = 1;
    #240 begin x = 0; y = 0; E = 0; F = 0; end    // Path: S_0
    #290 reset_b = 0;
    #292 reset_b = 1;
    #280 begin x = 0; y = 1; E = 0; F = 0; end    // Path: S_0, S_2, S_4, S_6, S_7
    #360 reset_b = 0;
    #362 reset_b = 1;
    #350 begin x = 0; y = 1; E = 1; F = 0; end    // Path: S_0, S_2, S_4, S_5
    #420 reset_b = 0;
    #422 reset_b = 1;
    #410 begin x = 0; y = 1; E = 0; F = 1; end    // Path: S_0, S_2, S_3
  join
endmodule
```

Note: Simulation results match those for 8.39g.

8.30 **(a)** $E = 1$ **(b)** $E = 0$

8.31 A = 0110, B = 0010, C = 0000.

A * B = 1100	A | B = 0110	A && C = 0
A + B = 1000	A ∧ B = 0100	| A = 1
A – B = 0100	&A = 0	A < B = 0
~ C = 1111	~|C = 1	A > B = 1
A & B = 0010	A || B = 1	A != B = 1

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

8.32

8.33

Assume that the states are encoded one-hot as T_0, T_1, T_2, T_3. The select lines of the mux are generated as:

$s_1 = T_2 + T_3$
$s_0 = T_1 + T_3$

The signal to load R_4 can be generated by the host processor or by:

$load = T_0 + T_1 + T_2 + T_3$.

8.34 (a)

```
module Datapath_BEH
#(parameter dp_width = 8, R2_width = 4)
(
  output [R2_width -1: 0] count, output reg E, output Zero, input [dp_width -1: 0] data,
input Load_regs, Shift_left, Incr_R2, clock, reset_b);
reg [dp_width -1: 0] R1;
reg [R2_width -1: 0] R2;
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

```
  assign count = R2;
  assign Zero = ~(| R1);
  always @ (posedge clock) begin
    E <= R1[dp_width -1] & Shift_left;
    if (Load_regs) begin R1 <= data; R2 <= {R2_width{1'b1}}; end
    if (Shift_left) {E, R1} <= {E, R1} << 1;
    if (Incr_R2) R2 <= R2 + 1;
  end
endmodule

// Test Plan for Datapath Unit:
// Demonstrate action of Load_regs
//      R1 gets data, R2 gets all ones
// Demonstrate action of Incr_R2
// Demonstrate action of Shift_left and detect E

// Test bench for datapath

module t_Datapath_Unit
#(parameter dp_width = 8, R2_width = 4)
( );
  wire [R2_width -1: 0]  count;
  wire                          E, Zero;
  reg [dp_width -1: 0]    data;
  reg                           Load_regs, Shift_left, Incr_R2, clock, reset_b;

  Datapath_BEH M0 (count, E, Zero, data, Load_regs, Shift_left, Incr_R2, clock, reset_b);


  initial #250 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial begin reset_b = 0; #2 reset_b = 1; end
  initial fork
    data = 8'haa;
    Load_regs = 0;
    Incr_R2 = 0;
    Shift_left = 0;
    #10 Load_regs = 1;
    #20 Load_regs = 0;
    #50 Incr_R2 = 1;
    #120 Incr_R2 = 0;
    #90 Shift_left = 1;
    #200 Shift_left = 0;
  join
endmodule
```

Note: The simulation results show tests of the operations of the datapath independent of the control unit, so *count* does not represent the number of ones in the *data*.

All rights reserved.

(b) // Control Unit

```
module Controller_BEH (
  output       Ready,
  output reg   Load_regs,
  output       Incr_R2, Shift_left,
  input        Start, Zero, E, clock, reset_b
);
  parameter S_idle = 0, S_1 = 1, S_2 = 2, S_3 = 3;
  reg [1:0] state, next_state;

  assign Ready = (state == S_idle);
  assign Incr_R2 = (state == S_1);
  assign Shift_left = (state == S_2);

  always @ (posedge clock, negedge reset_b)
    if (reset_b == 0) state <= S_idle;
    else state <= next_state;

  always @ (state, Start, Zero, E) begin
    Load_regs = 0;
    case (state)
      S_idle:    if (Start) begin Load_regs = 1; next_state = S_1; end
                 else next_state = S_idle;
      S_1:       if (Zero) next_state = S_idle; else next_state = S_2;

      S_2:       next_state = S_3;
      S_3:       if (E) next_state = S_1; else next_state = S_2;
    endcase
  end
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
// Test plan for Control Unit
// Verify that state enters S_idle with reset_b asserted.
// With reset_b de-asserted, verify that state enters S_1 and asserts Load_Regs when
// Start is asserted.
// Verify that Incr_R2 is asserted in S_1.
// Verify that state returns to S_idle from S_1 if Zero is asserted.
// Verify that state goes to S_2 if Zero is not asserted.
// Verify that Shift_left is asserted in S_2.
// Verify that state goes to S_3 from S_2 unconditionally.
//  Verify that state returns to S_2 from S_3 id E is not asserted.
// Verify that state goes to S_1 from S_3 if E is asserted.

// Test bench for Control Unit

module t_Control_Unit ();
  wire Ready, Load_regs, Incr_R2, Shift_left;
  reg Start, Zero, E, clock, reset_b;

  Controller_BEH M0 (Ready, Load_regs, Incr_R2, Shift_left, Start, Zero, E, clock, reset_b);

  initial #250 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial begin reset_b = 0; #2 reset_b = 1; end
  initial fork
    Zero = 1;
    E = 0;
    Start = 0;
    #20 Start = 1;   // Cycle from S_idle to S_1
    #80 Start = 0;
    #70 Zero = 0;   // S_idle to S_1 to S_2 to S_3 and cycle to S_2.
    #130 E = 1; // Cycle to S_3 to S_1 to S_2 to S_3
    #150 Zero = 1; // Return to S_idle
  join
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

(c)

```
// Integrated system
module Count_Ones_BEH_BEH
# (parameter dp_width = 8, R2_width = 4)
(
  output [R2_width -1: 0]       count,
  input [dp_width -1: 0]    data,
  input                 Start, clock, reset_b
);
  wire  Load_regs, Incr_R2, Shift_left, Zero, E;
  Controller_BEH M0 (Ready, Load_regs, Incr_R2, Shift_left, Start, Zero, E, clock, reset_b);
  Datapath_BEH M1 (count, E, Zero, data, Load_regs, Shift_left, Incr_R2, clock, reset_b);
endmodule

// Test plan for integrated system
// Test for data values of 8'haa, 8'h00, 8'hff.

// Test bench for integrated system

module t_count_Ones_BEH_BEH ();
  parameter dp_width = 8, R2_width = 4;
  wire [R2_width -1: 0] count;
  reg [dp_width -1: 0] data;
  reg Start, clock, reset_b;

  Count_Ones_BEH_BEH M0 (count, data, Start, clock, reset_b);
  initial #700 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial begin reset_b = 0; #2 reset_b = 1; end
  initial fork
    data = 8'haa;          // Expect count = 4
    Start = 0;
    #20 Start = 1;
    #30 Start = 0;
    #40 data = 8'b00;   // Expect count = 0
    #250 Start = 1;
    #260 Start = 0;
    #280 data = 8'hff;
    #280 Start = 1;
    #290 Start = 0;
  join
endmodule
```


All rights reserved.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

(d)

```
// One-Hot Control unit

module Controller_BEH_1Hot
(
  output       Ready,
  output reg   Load_regs,
  output       Incr_R2, Shift_left,
  input        Start, Zero, E, clock, reset_b
);
  parameter S_idle = 4'b001, S_1 = 4'b0010, S_2 = 4'b0100, S_3 = 4'b1000;
  reg [3:0] state, next_state;

  assign Ready = (state == S_idle);
  assign Incr_R2 = (state == S_1);
  assign Shift_left = (state == S_2);

  always @ (posedge clock, negedge reset_b)
    if (reset_b == 0) state <= S_idle;
    else state <= next_state;

  always @ (state, Start, Zero, E) begin
    Load_regs = 0;
    case (state)
      S_idle: if (Start) begin Load_regs = 1; next_state = S_1; end
              else next_state = S_idle;
      S_1:    if (Zero) next_state = S_idle; else next_state = S_2;

      S_2:    next_state = S_3;
      S_3:    if (E) next_state = S_1; else next_state = S_2;
    endcase
  end
endmodule
```

Note: Test plan, test bench and simulation results are same as (b), but with states numbered with one-hot codes.

(e)

```
// Integrated system with one-hot controller

module Count_Ones_BEH_1Hot
```


All rights reserved.

```
# (parameter dp_width = 8, R2_width = 4)
(
  output [R2_width -1: 0]      count,
  input [dp_width -1: 0]       data,
  input                        Start, clock, reset_b
);
  wire  Load_regs, Incr_R2, Shift_left, Zero, E;
  Controller_BEH_1Hot M0 (Ready, Load_regs, Incr_R2, Shift_left, Start, Zero, E, clock, reset_b);
  Datapath_BEH M1 (count, E, Zero, data, Load_regs, Shift_left, Incr_R2, clock, reset_b);
endmodule
```

Note: Test plan, test bench and simulation results are same as (c), but with states numbered with one-hot codes.

8.35 Note: Signal *Start* is initialized to 0 when the simulation begins. Otherwise, the state of the structural model will become X at the first clock after the reset condition is deasserted, with Start and Load_Regs having unknown values. In this condition the structural model cannot operate correctly.

```
module Count_Ones_STR_STR (count, Ready, data, Start, clock, reset_b);
// Mux – decoder implementation of control logic
// controller is structural
// datapath is structural

  parameter   R1_size = 8, R2_size = 4;
  output      [R2_size -1: 0]    count;
  output                         Ready;
  input       [R1_size -1: 0]    data;
  input                          Start, clock, reset_b;
  wire                           Load_regs, Shift_left, Incr_R2, Zero, E;

 Controller_STR    M0 (Ready, Load_regs, Shift_left, Incr_R2, Start, E, Zero, clock, reset_b);
 Datapath_STR    M1 (count, E, Zero, data, Load_regs, Shift_left, Incr_R2, clock);

endmodule

module Controller_STR (Ready, Load_regs, Shift_left, Incr_R2, Start, E, Zero, clock, reset_b);
```


All rights reserved.

```
  output        Ready;
  output        Load_regs, Shift_left, Incr_R2;
  input         Start;
  input         E, Zero;
  input         clock, reset_b;
  supply0       GND;
  supply1       PWR;
  parameter     S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 2'b11;  // Binary code
  wire          Load_regs, Shift_left, Incr_R2;
  wire          G0, G0_b, D_in0, D_in1, G1, G1_b;
  wire          Zero_b = ~Zero;

  wire          E_b = ~E;
  wire [1:0]select = {G1, G0};
  wire [0:3]Decoder_out;

  assign            Ready = ~Decoder_out[0];
  assign            Incr_R2 = ~Decoder_out[1];
  assign            Shift_left = ~Decoder_out[2];
  and               (Load_regs, Ready, Start);
  mux_4x1_beh       Mux_1 (D_in1, GND, Zero_b, PWR, E_b, select);
  mux_4x1_beh       Mux_0     (D_in0, Start, GND, PWR, E, select);
  D_flip_flop_AR_b  M1      (G1, G1_b, D_in1, clock, reset_b);
  D_flip_flop_AR_b  M0      (G0, G0_b, D_in0, clock, reset_b);
  decoder_2x4_df    M2      (Decoder_out, G1, G0, GND);

endmodule

module Datapath_STR (count, E, Zero, data, Load_regs, Shift_left, Incr_R2, clock);
  parameter       R1_size = 8, R2_size = 4;
  output  [R2_size -1: 0]  count;
  output                   E, Zero;
  input [R1_size -1: 0]    data;
  input                    Load_regs, Shift_left, Incr_R2, clock;
  wire [R1_size -1: 0] R1;
  supply0                  Gnd;
  supply1                  Pwr;
  assign                   Zero = (R1 == 0);

 Shift_Reg          M1     (R1, data, Gnd, Shift_left, Load_regs, clock, Pwr);
 Counter            M2     (count, Load_regs, Incr_R2, clock, Pwr);
  D_flip_flop_AR    M3     (E, w1, clock, Pwr);
  and          (           w1, R1[R1_size -1], Shift_left);
endmodule

module Shift_Reg (R1, data, SI_0, Shift_left, Load_regs, clock, reset_b);
  parameter                R1_size = 8;
  output [R1_size -1: 0]   R1;
  input [R1_size -1: 0]    data;
  input                    SI_0, Shift_left, Load_regs;
  input                    clock, reset_b;
  reg [R1_size -1: 0]      R1;

  always @ (posedge clock, negedge reset_b)
    if (reset_b == 0) R1 <= 0;
    else begin
      if (Load_regs) R1 <= data; else
        if (Shift_left) R1 <= {R1[R1_size -2:0], SI_0}; end
endmodule

module Counter (R2, Load_regs, Incr_R2, clock, reset_b);
  parameter                R2_size = 4;
```


All rights reserved.

```
  output [R2_size -1: 0]   R2;
  input                    Load_regs, Incr_R2;
  input                    clock, reset_b;
  reg [R2_size -1: 0]      R2;

  always @ (posedge clock, negedge reset_b)
    if (reset_b == 0) R2 <= 0;
    else if (Load_regs) R2 <= {R2_size {1'b1}};  // Fill with 1
      else if (Incr_R2 == 1) R2 <= R2 + 1;
endmodule

module D_flip_flop_AR (Q, D, CLK, RST);
  output    Q;
  input     D, CLK, RST;
  reg       Q;

  always @ (posedge CLK, negedge RST)
    if (RST == 0) Q <= 1'b0;
    else Q <= D;
endmodule

module D_flip_flop_AR_b (Q, Q_b, D, CLK, RST);
  output    Q, Q_b;
  input     D, CLK, RST;
  reg       Q;
  assign        Q_b = ~Q;
  always @ (posedge CLK, negedge RST)
    if (RST == 0) Q <= 1'b0;
    else Q <= D;
endmodule

// Behavioral description of 4-to-1 line multiplexer
// Verilog 2005 port syntax

module mux_4x1_beh
( output reg   m_out,
  input        in_0, in_1, in_2, in_3,
  input [1: 0]   select
);

  always @ (in_0, in_1, in_2, in_3, select)   // Verilog 2005 syntax
    case (select)
      2'b00: m_out = in_0;
      2'b01: m_out = in_1;
      2'b10: m_out = in_2;
      2'b11: m_out = in_3;
    endcase
  endmodule
// Dataflow description of 2-to-4-line decoder
// See Fig. 4.19. Note: The figure uses symbol E, but the
// Verilog model uses enable to clearly indicate functionality.

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

  input               A, B;
  input               enable;


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


All rights reserved.

```
        D[3] = ~(A & B & ~enable);
endmodule

module t_Count_Ones;
  parameter R1_size = 8, R2_size = 4;
  wire  [R2_size -1: 0]     R2;

  wire  [R2_size -1: 0]     count;
  wire                      Ready;
  reg   [R1_size -1: 0]     data;
  reg                       Start, clock, reset_b;
  wire [1: 0] state;      // Use only for debug
  assign state = {M0.M0.G1, M0.M0.G0};

  Count_Ones_STR_STR M0 (count, Ready, data, Start, clock, reset_b);

  initial #4000 $finish;
  initial begin clock = 0; #5 forever #5 clock = ~clock; end
  initial fork
    Start = 0;

    #1 reset_b = 1;
    #3 reset_b = 0;
    #4 reset_b = 1;
     data = 8'Hff;
    # 25 Start = 1;
    # 35  Start = 0;
    #310 data = 8'h0f;
    #310 Start = 1;
    #320 Start = 0;
    #610 data = 8'hf0;
    #610 Start = 1;
    #620 Start = 0;
    #910 data = 8'h00;
    #910 Start = 1;
    #920 Start = 0;
    #1210 data = 8'haa;
    #1210 Start = 1;
    #1220 Start = 0;
    #1510 data = 8'h0a;
    #1510 Start = 1;
    #1520 Start = 0;
    #1810 data = 8'ha0;
    #1810 Start = 1;
    #1820 Start = 0;
    #2110 data = 8'h55;
    #2110 Start = 1;
    #2120 Start = 0;
    #2410 data = 8'h05;
    #2410 Start = 1;
    #2420 Start = 0;
    #2710 data = 8'h50;
    #2710 Start = 1;
    #2720 Start = 0;
    #3010 data = 8'ha5;
    #3010 Start = 1;
    #3020 Start = 0;
    #3310 data = 8'h5a;
    #3310 Start = 1;
    #3320 Start = 0;
  join
endmodule
```


All rights reserved.

8.36 Note: See Prob. 8.35 for a behavioral model of the datapath unit, Prob. 8.36d for a one-hot control unit.

(a) T_0, T_1, T_2, T_3 be asserted when the state is in *S_idle*, *S*_1, *S*_2, and *S*_3, respectively. Let *D*0, *D*1, *D*2, and *D*3 denote the inputs to the one-hot flip-flops.

$D_0 = T_0\, Start' + T_1\, Zero$
$D_1 = T_0\, Start + T_3\, E$
$D_2 = T_1\, Zero' + T_3\, E'$
$D_3 = T_2$

(b) Gate-level one-hot controller

```
module Controller_Gates_1Hot
(
 output    Ready,
 output    Load_regs, Incr_R2, Shift_left,
 input     Start, Zero, E, clock, reset_b
);
 wire w1, w2, w3, w4, w5, w6;
 wire T0, T1, T2, T3;
 wire set;
 assign Ready = T0;
 assign Incr_R2 = T1;
 assign Shift_left = T2;
 and (Load_regs, T0, Start);
 not (set, reset_b);
 DFF_S M0 (T0, D0, clock, set);      // Note: reset action must initialize S_idle = 4'b0001
 DFF M1 (T1, D1, clock, reset_b);
 DFF M2 (T2, D2, clock, reset_b);
 DFF M3 (T3, D3, clock, reset_b);

 not (Start_b, Start);
 and (w1, T0, Start_b);
 and (w2, T1, Zero);
 or (D0, w1, w2);
```


All rights reserved.

```
  and (w3, T0, Start);
  and (w4, T3, E);
  or (D1, w3, w4);

  not (Zero_b, Zero);
  not (E_b, E);
  and (w5, T1, Zero_b);
  and (w6, T3, E_b);
  or (D2, w5, w6);

  buf (D3, T2);
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
module DFF_S (output reg Q, input D, clock, set);
  always @ (posedge clock, posedge set)
    if (set == 1) Q <= 1;
    else Q <= D;
endmodule
```

(c)

```
// Test plan for Control Unit
// Verify that state enters S_idle with reset_b asserted.
// With reset_b de-asserted, verify that state enters S_1 and asserts Load_Regs when
// Start is asserted.
// Verify that Incr_R2 is asserted in S_1.
// Verify that state returns to S_idle from S_1 if Zero is asserted.
// Verify that state goes to S_2 if Zero is not asserted.
// Verify that Shift_left is asserted in S_2.
// Verify that state goes to S_3 from S_2 unconditionally.
//  Verify that state returns to S_2 from S_3 id E is not asserted.
// Verify that state goes to S_1 from S_3 if E is asserted.

// Test bench for One-Hot Control Unit

module t_Control_Unit ();
  wire Ready, Load_regs, Incr_R2, Shift_left;
  reg Start, Zero, E, clock, reset_b;
  wire [3: 0] state = {M0.T3, M0.T2, M0.T1, M0.T0};        // Observe one-hot state bits
  Controller_Gates_1Hot M0 (Ready, Load_regs, Incr_R2, Shift_left, Start, Zero, E, clock, reset_b);

  initial #250 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial begin reset_b = 0; #2 reset_b = 1; end
  initial fork
    Zero = 1;
    E = 0;
    Start = 0;
    #20 Start = 1;// Cycle from S_idle to S_1
    #80 Start = 0;
    #70 Zero = 0;      // S_idle to S_1 to S_2 to S_3 and cycle to S_2.
    #130 E = 1;        // Cycle to S_3 to S_1 to S_2 to S_3
    #150 Zero = 1;     // Return to S_idle
  join
endmodule
```

Note: simulation results match those for Prob. 8.34(d). See Prob. 8.34(c) for annotations.

All rights reserved.

(d) Datapath unit detail:

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
//  Datapath unit – structural model
module Datapath_STR
#(parameter dp_width = 8, R2_width = 4)
(
  output [R2_width -1: 0] count, output E, output Zero, input [dp_width -1: 0] data,
  input Load_regs, Shift_left, Incr_R2, clock, reset_b);
  supply1 pwr;
  supply0 gnd;
  wire [dp_width -1: 0] R1_Dbus, R1;
  wire [R2_width -1: 0] R2_Dbus;
  wire DR1_0, DR1_1, DR1_2, DR1_3, DR1_4, DR1_5, DR1_6, DR1_7;
  wire R1_0, R1_1, R1_2, R1_3, R1_4, R1_5, R1_6, R1_7;
  wire R2_0, R2_1, R2_2, R2_3;
  wire [R2_width -1: 0] R2 = {R2_3, R2_2, R2_1, R2_0};
  assign count = {R2_3, R2_2, R2_1, R2_0};
  assign R1 = { R1_7, R1_6, R1_5, R1_4, R1_3, R1_2, R1_1, R1_0};
  assign DR1_0 = R1_Dbus[0];
  assign DR1_1 = R1_Dbus[1];
  assign DR1_2 = R1_Dbus[2];
  assign DR1_3 = R1_Dbus[3];
  assign DR1_4 = R1_Dbus[4];
  assign DR1_5 = R1_Dbus[5];
  assign DR1_6 = R1_Dbus[6];
  assign DR1_7 = R1_Dbus[7];

  nor (Zero, R1_0, R1_1, R1_2, R1_3, R1_4, R1_5, R1_6, R1_7);
  DFF D_E (E, R1_7, clock, pwr);

  DFF DF_0 (R1_0, DR1_0, clock, pwr);      // Disable reset
  DFF DF_1 (R1_1, DR1_1, clock, pwr);
  DFF DF_2 (R1_2, DR1_2, clock, pwr);
  DFF DF_3 (R1_3, DR1_3, clock, pwr);
  DFF DF_4 (R1_4, DR1_4, clock, pwr);
  DFF DF_5 (R1_5, DR1_5, clock, pwr);
  DFF DF_6 (R1_6, DR1_6, clock, pwr);
  DFF DF_7 (R1_7, DR1_7, clock, pwr);

  DFF_S DR_0 (R2_0, DR2_0, clock, Load_regs);  // Load_regs (set) drives R2 to all ones
  DFF_S DR_1 (R2_1, DR2_1, clock, Load_regs);
  DFF_S DR_2 (R2_2, DR2_2, clock, Load_regs);
  DFF_S DR_3 (R2_3, DR2_3, clock, Load_regs);

  assign DR2_0 = R2_Dbus[0];
  assign DR2_1 = R2_Dbus[1];
  assign DR2_2 = R2_Dbus[2];
  assign DR2_3 = R2_Dbus[3];

  wire [1: 0] sel = {Shift_left, Load_regs};
  wire [dp_width -1: 0] R1_shifted = {R1_6, R1_5, R1_4, R1_3, R1_2, R1_1, R1_0, 1'b0};
  wire [R2_width -1: 0] sum = R2 + 4'b0001;

  Mux8_4_x_1 M0 (R1_Dbus, R1, data, R1_shifted, R1, sel);
  Mux4_2_x_1 M1 (R2_Dbus, R2, sum, Incr_R2);
endmodule
```


All rights reserved.

```
  module Mux8_4_x_1 #(parameter dp_width = 8) (output reg [dp_width -1: 0] mux_out,
    input [dp_width -1: 0] in0, in1, in2, in3, input [1: 0] sel);
    always @ (in0, in1, in2, in3, sel)
      case (sel)
        2'b00:  mux_out = in0;
        2'b01:  mux_out = in1;
        2'b10:  mux_out = in2;
        2'b11:  mux_out = in3;
      endcase
  endmodule

  module Mux4_2_x_1 #(parameter dp_width = 4) (output [dp_width -1: 0] mux_out,
    input [dp_width -1: 0] in0, in1, input  sel);
    assign mux_out = sel ? in1: in0;
endmodule

// Test Plan for Datapath Unit:
// Demonstrate action of Load_regs
//      R1 gets data, R2 gets all ones
// Demonstrate action of Incr_R2
// Demonstrate action of Shift_left and detect E

// Test bench for datapath
module t_Datapath_Unit
#(parameter dp_width = 8, R2_width = 4)
( );
  wire [R2_width -1: 0]  count;
  wire                          E, Zero;
  reg [dp_width -1: 0]    data;
  reg                           Load_regs, Shift_left, Incr_R2, clock, reset_b;
  Datapath_STR M0 (count, E, Zero, data, Load_regs, Shift_left, Incr_R2, clock, reset_b);

  initial #250 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial begin reset_b = 0; #2 reset_b = 1; end
  initial fork
    data = 8'haa;
    Load_regs = 0;
    Incr_R2 = 0;
    Shift_left = 0;
    #10 Load_regs = 1;
    #20 Load_regs = 0;
    #50 Incr_R2 = 1;
    #120 Incr_R2 = 0;
    #90 Shift_left = 1;
    #200 Shift_left = 0;
  join
endmodule

// Integrated system
module Count_Ones_Gates_1_Hot_STR
# (parameter dp_width = 8, R2_width = 4)
(
  output [R2_width -1: 0]    count,
  input [dp_width -1: 0] data,
  input              Start, clock, reset_b
);
  wire     Load_regs, Incr_R2, Shift_left, Zero, E;
  Controller_Gates_1Hot M0 (Ready, Load_regs, Incr_R2, Shift_left, Start, Zero, E, clock, reset_b);
  Datapath_STR M1 (count, E, Zero, data, Load_regs, Shift_left, Incr_R2, clock, reset_b);
endmodule
```


All rights reserved.

```
// Test plan for integrated system
// Test for data values of 8'haa, 8'h00, 8'hff.

// Test bench for integrated system

module t_count_Ones_Gates_1_Hot_STR ();
  parameter dp_width = 8, R2_width = 4;
  wire [R2_width -1: 0] count;
  reg [dp_width -1: 0] data;
  reg Start, clock, reset_b;
  wire [3: 0] state = {M0.M0.T3, M0.M0.T2, M0.M0.T1, M0.M0.T0};

  Count_Ones_Gates_1_Hot_STR M0 (count, data, Start, clock, reset_b);
  initial #700 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial begin reset_b = 0; #2 reset_b = 1; end
  initial fork
    data = 8'haa;      // Expect count = 4
    Start = 0;
    #20 Start = 1;
    #30 Start = 0;
    #40 data = 8'b00;     // Expect count = 0
    #250 Start = 1;
    #260 Start = 0;
    #280 data = 8'hff;
    #280 Start = 1;
    #290 Start = 0;
  join
endmodule
```

Note: The simulation results show tests of the operations of the datapath independent of the control unit, so count does not represent the number of ones in the data.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

Simulations results for the integrated system match those shown in Prob. 8.34(e). See those results for additional annotation.

8.37 **(a)** ASMD chart:

(b) RTL model:

```
module Datapath_Unit_2_Beh #(parameter dp_width = 8, R2_width = 4)
(
  output [R2_width -1: 0]      count,
  output                       Zero,
  input [dp_width -1: 0]    data,
  input                        Load_regs, Add_shift, clock, reset_b
);
  reg [dp_width -1: 0]       R1;
  reg [R2_width -1: 0]          R2;
  assign count = R2;
  assign Zero = ~|R1;
  always @ (posedge clock, negedge reset_b)
    begin
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
    if (reset_b == 0) begin R1 <= 0; R2 <= 0; end else begin
      if (Load_regs) begin R1 <= data; R2 <= 0; end
      if (Add_shift) begin R1 <= R1 >> 1; R2 <= R2 + R1[0]; end // concurrent operations
    end
  end
endmodule

// Test plan for datapath unit
// Verify active-low reset action
// Test for action of Add_shift
// Test for action of Load_regs

module t_Datapath_Unit_2_Beh();
  parameter R1_size = 8, R2_size = 4;
  wire  [R2_size -1: 0]     count;
  wire                      Zero;
  reg   [R1_size -1: 0]     data;
  reg                       Load_regs, Add_shift, clock, reset_b;

  Datapath_Unit_2_Beh M0 (count, Zero, data, Load_regs, Add_shift, clock, reset_b);

  initial #1000 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial fork
    #1 reset_b = 1;
    #3 reset_b = 0;
    #4 reset_b = 1;
  join
  initial fork
    data = 8'haa;
    Load_regs = 0;
    Add_shift = 0;
    #10 Load_regs = 1;
    #20 Load_regs = 0;
    #50 Add_shift = 1;
    #150 Add_shift  = 0;
  join
endmodule
```

Note that the operations of the datapath unit are tested independent of the controller, so the actions of *Load_regs* and *add_shift* and the value of *count* do not correspond to *data*.

```
module Controller_2_Beh (
  output        Ready,
```


All rights reserved.

```
  output reg   Load_regs,
               Add_shift,
  input        Start, Zero, clock, reset_b
);
  parameter    S_idle = 0, S_running = 1;
  reg          state, next_state;
  assign       Ready = (state == S_idle);

  always @ (posedge clock, negedge reset_b)
    if (reset_b == 0) state <= S_idle;
    else state <= next_state;

  always @ (state, Start, Zero) begin
    next_state = S_idle;
    Load_regs = 0;
    Add_shift = 0;

    case (state)
      S_idle:       if (Start) begin Load_regs = 1; next_state = S_running; end
      S_running:    if (Zero) next_state = S_idle;
                    else begin Add_shift = 1; next_state = S_running; end
    endcase
  end
endmodule

module t_Controller_2_Beh ();
  wire Ready, Load_regs, Add_shift;
  reg Start, Zero, clock, reset_b;

  Controller_2_Beh M0 (Ready, Load_regs, Add_shift, Start, Zero, clock, reset_b);

  initial #250 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial begin reset_b = 0; #2 reset_b = 1; end
  initial fork
    Zero = 1;
    Start = 0;
    #20 Start = 1;   // Cycle from S_idle to S_1
    #80 Start = 0;
    #70 Zero = 0;    // S_idle to S_1 to S_idle
    #90 Zero = 1;    // Return to S_idle
  join
endmodule
```

Note: The state transitions and outputs of the controller match the ASMD chart.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
module Count_of_Ones_2_Beh #(parameter dp_width = 8, R2_width = 4)
(
  output [R2_width -1: 0] count,
  output Ready,
  input [dp_width -1: 0] data,
  input Start, clock, reset_b
);
  wire Load_regs, Add_shift, Zero;

  Controller_2_Beh M0 (Ready, Load_regs, Add_shift, Start, Zero, clock, reset_b);
  Datapath_Unit_2_Beh M1 (count, Zero, data, Load_regs, Add_shift, clock, reset_b);
endmodule

// Test plan for integrated system
// Test for data values of 8'haa, 8'h00, 8'hff.

// Test bench for integrated system

module t_Count_Ones_2_Beh ();
  parameter dp_width = 8, R2_width = 4;
  wire [R2_width -1: 0] count;
  reg [dp_width -1: 0] data;
  reg Start, clock, reset_b;

  Count_of_Ones_2_Beh M0 (count, Ready, data, Start, clock, reset_b);

  initial #700 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial begin reset_b = 0; #2 reset_b = 1; end
  initial fork
    data = 8'haa;          // Expect count = 4
    Start = 0;
    #20 Start = 1;
    #30 Start = 0;
    #40 data = 8'b00;   // Expect count = 0
    #120 Start = 1;
    #130 Start = 0;
    #140 data = 8'hff;
    #160 Start = 1;
    #170 Start = 0;

  join
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

(c) T_0, T_1 are to be asserted when the state is in *S_idle*, *S_running*, respectively. Let $D0$, $D1$ denote the inputs to the one-hot flip-flops.

$D_0 = T_0\ Start' + T_1\ Zero$
$D_1 = T_0\ Start + T_1\ E'$

(d) Gate-level one-hot controller

```
module Controller_2_Gates_1Hot
(
  output      Ready, Load_regs, Add_shift,
  input       Start, Zero, clock, reset_b
);
  wire w1, w2, w3, w4;
  wire T0, T1;
  wire set;
  assign Ready = T0;
  assign Add_shift = T1;
  and (Load_regs, T0, Start);
  not (set, reset_b);
  DFF_S M0 (T0, D0, clock, set);     // Note: reset action must initialize S_idle = 2'b01
  DFF M1 (T1, D1, clock, reset_b);

  not (Start_b, Start);
  not (Zero_b, Zero);
  and (w1, T0, Start_b);
  and (w2, T1, Zero);
  or (D0, w1, w2);

  and (w3, T0, Start);
  and (w4, T1, Zero_b);
  or (D1, w3, w4);
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
module DFF_S (output reg Q, input D, clock, set);
```


All rights reserved.

```
  always @ (posedge clock, posedge set)
    if (set == 1) Q <= 1;
    else Q <= D;
endmodule

// Test plan for Control Unit
// Verify that state enters S_idle with reset_b asserted.
// With reset_b de-asserted, verify that state enters S_running and asserts Load_Regs when
// Start is asserted.
// Verify that state returns to S_idle from S_running if Zero is asserted.
// Verify that state goes to S_running if Zero is not asserted.

// Test bench for One-Hot Control Unit

module t_Control_Unit ();
  wire        Ready, Load_regs, Add_shift;
  reg         Start, Zero, clock, reset_b;
  wire [3: 0] state = {M0.T1, M0.T0};      // Observe one-hot state bits

  Controller_2_Gates_1Hot M0 (Ready, Load_regs, Add_shift, Start, Zero, clock, reset_b);

  initial #250 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial begin reset_b = 0; #2 reset_b = 1; end
  initial fork
    Zero = 1;
    Start = 0;
    #20 Start = 1;   // Cycle from S_idle to S_1
    #80 Start = 0;
    #70 Zero = 0;   // S_idle to S_1 to S_idle
    #90 Zero = 1;   // Return to S_idle
  join
endmodule
```

Simulation results show that the controller matches the ASMD chart.

```
//  Datapath unit – structural model

module Datapath_2_STR
#(parameter dp_width = 8, R2_width = 4)
```


All rights reserved.

```
(
 output [R2_width -1: 0]      count,
 output                       Zero,
 input [dp_width -1: 0]       data,
 input                        Load_regs, Add_shift, clock, reset_b);
 supply1                      pwr;
 supply0                      gnd;
 wire [dp_width -1: 0] R1_Dbus, R1;
 wire [R2_width -1: 0] R2_Dbus;
 wire DR1_0, DR1_1, DR1_2, DR1_3, DR1_4, DR1_5, DR1_6, DR1_7;
 wire R1_0, R1_1, R1_2, R1_3, R1_4, R1_5, R1_6, R1_7;
 wire R2_0, R2_1, R2_2, R2_3;
 wire [R2_width -1: 0] R2 = {R2_3, R2_2, R2_1, R2_0};
 assign count = {R2_3, R2_2, R2_1, R2_0};
 assign R1 = { R1_7, R1_6, R1_5, R1_4, R1_3, R1_2, R1_1, R1_0};
 assign DR1_0 = R1_Dbus[0];
 assign DR1_1 = R1_Dbus[1];
 assign DR1_2 = R1_Dbus[2];
 assign DR1_3 = R1_Dbus[3];
 assign DR1_4 = R1_Dbus[4];
 assign DR1_5 = R1_Dbus[5];
 assign DR1_6 = R1_Dbus[6];
 assign DR1_7 = R1_Dbus[7];

 nor (Zero, R1_0, R1_1, R1_2, R1_3, R1_4, R1_5, R1_6, R1_7);
 not (Load_regs_b, Load_regs);

 DFF DF_0 (R1_0, DR1_0, clock, pwr);      // Disable reset
 DFF DF_1 (R1_1, DR1_1, clock, pwr);
 DFF DF_2 (R1_2, DR1_2, clock, pwr);
 DFF DF_3 (R1_3, DR1_3, clock, pwr);
 DFF DF_4 (R1_4, DR1_4, clock, pwr);
 DFF DF_5 (R1_5, DR1_5, clock, pwr);
 DFF DF_6 (R1_6, DR1_6, clock, pwr);
 DFF DF_7 (R1_7, DR1_7, clock, pwr);

 DFF DR_0 (R2_0, DR2_0, clock, Load_regs_b);    // Load_regs (set) drives R2 to all ones
 DFF DR_1 (R2_1, DR2_1, clock, Load_regs_b);
 DFF DR_2 (R2_2, DR2_2, clock, Load_regs_b);
 DFF DR_3 (R2_3, DR2_3, clock, Load_regs_b);

 assign DR2_0 = R2_Dbus[0];
 assign DR2_1 = R2_Dbus[1];
 assign DR2_2 = R2_Dbus[2];
 assign DR2_3 = R2_Dbus[3];

 wire [1: 0]                  sel = {Add_shift, Load_regs};
 wire [dp_width -1: 0]        R1_shifted = {1'b0, R1_7, R1_6, R1_5, R1_4, R1_3, R1_2, R1_1};
 wire [R2_width -1: 0]        sum = R2 + {3'b000, R1[0]};

 Mux8_4_x_1 M0 (R1_Dbus, R1, data, R1_shifted, R1, sel);
 Mux4_2_x_1 M1 (R2_Dbus, R2, sum, Add_shift);
endmodule
```


All rights reserved.

```
  module Mux8_4_x_1 #(parameter dp_width = 8) (output reg [dp_width -1: 0] mux_out,
    input [dp_width -1: 0] in0, in1, in2, in3, input [1: 0] sel);
    always @ (in0, in1, in2, in3, sel)
      case (sel)
        2'b00:  mux_out = in0;
        2'b01:  mux_out = in1;
        2'b10:  mux_out = in2;
        2'b11:  mux_out = in3;
      endcase
  endmodule

  module Mux4_2_x_1 #(parameter dp_width = 4) (output [dp_width -1: 0] mux_out,
    input [dp_width -1: 0] in0, in1, input  sel);
    assign mux_out = sel ? in1: in0;
endmodule

// Test Plan for Datapath Unit:
// Demonstrate action of Load_regs
// R1 gets data, R2 gets all ones
// Demonstrate action of Incr_R2
// Demonstrate action of Add_shift and detect Zero

// Test bench for datapath

module t_Datapath_Unit
#(parameter dp_width = 8, R2_width = 4)
( );
  wire [R2_width -1: 0]  count;
  wire                   Zero;
  reg [dp_width -1: 0]   data;
    reg                  Load_regs, Add_shift, clock, reset_b;

  Datapath_2_STR M0 (count,  Zero, data, Load_regs, Add_shift, clock, reset_b);

  initial #250 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial begin reset_b = 0; #2 reset_b = 1; end
  initial fork
    data = 8'haa;
    Load_regs = 0;
    Add_shift = 0;
    #10 Load_regs = 1;
    #20 Load_regs = 0;
    #50 Add_shift = 1;
    #140 Add_shift = 0;
  join
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
// Integrated system

module Count_Ones_2_Gates_1Hot_STR
# (parameter dp_width = 8, R2_width = 4)
(
  output [R2_width -1: 0]    count,
  input [dp_width -1: 0] data,
  input              Start, clock, reset_b
);
  wire     Load_regs, Add_shift, Zero;
  Controller_2_Gates_1Hot M0 (Ready, Load_regs, Add_shift, Start, Zero, clock, reset_b);
  Datapath_2_STR M1 (count,  Zero, data, Load_regs, Add_shift, clock, reset_b);
endmodule

// Test plan for integrated system
// Test for data values of 8'haa, 8'h00, 8'hff.

// Test bench for integrated system

module t_Count_Ones_2_Gates_1Hot_STR ();
  parameter                dp_width = 8, R2_width = 4;
  wire [R2_width -1: 0]  count;
  reg [dp_width -1: 0]    data;
  reg                      Start, clock, reset_b;
  wire [1: 0]                  state = {M0.M0.T1, M0.M0.T0};

 Count_Ones_2_Gates_1Hot_STR  M0 (count, data, Start, clock, reset_b);

  initial #700 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial begin reset_b = 0; #2 reset_b = 1; end
  initial fork
    data = 8'haa;       // Expect count = 4
    Start = 0;
    #20 Start = 1;
    #30 Start = 0;
    #40 data = 8'b00;      // Expect count = 0
    #120 Start = 1;
    #130 Start = 0;
    #150 data = 8'hff;      // Expect count = 8
    #200 Start = 1;
    #210 Start = 0;
  join
endmodule
```


All rights reserved.

8.38

```
module Prob_8_38 (
 output reg [7: 0]     Sum,
 output reg            Car_Bor,
 input [7: 0]          Data_A, Data_B);
 reg [7: 0]            Reg_A, Reg_B;

 always @ (Data_A, Data_B)
    case ({Data_A[7], Data_B[7]})
      2'b00, 2'b11:          begin                                  // ++, --
                               {Car_Bor, Sum[6: 0]} = Data_A[6: 0] + Data_B[6: 0];
                              Sum[7] = Data_A[7];
                             end

      default:               if (Data_A[6: 0] >= Data_B[6: 0]) begin              // +-, -+
                                {Car_Bor, Sum[6: 0]} = Data_A[6: 0] - Data_B[6: 0];
                                Sum[7] = Data_A[7];
                              end
                              else begin
                                {Car_Bor, Sum[6: 0]} = Data_B[6: 0] - Data_A[6: 0];
                                Sum[7] = Data_B[7];
                              end
    endcase
endmodule

module t_Prob_8_38 ();
 wire [7: 0]    Sum;
 wire           Car_Bor;
 reg [7: 0]     Data_A, Data_B;
 wire [6: 0]    Mag_A, Mag_B;
 assign         Mag_A = M0.Data_A[6: 0];          // Hierarchical dereferencing
 assign         Mag_B = M0.Data_B[6: 0];
 wire           Sign_A = M0.Data_A[7];
 wire           Sign_B = M0.Data_B[7];
 wire           Sign = Sum[7];
 wire [7: 0]    Mag = Sum[6: 0];

 Prob_8_38 M0 (Sum, Car_Bor,  Data_A, Data_B);

 initial #650 $finish;
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
initial fork

  // Addition                                  //  A        B
   #0 begin Data_A = {1'b0, 7'd25}; Data_B = {1'b0, 7'd10}; end          //+25, +10
  #40 begin Data_A = {1'b1, 7'd25}; Data_B = {1'b1, 7'd10}; end          // -25, -10
  #80 begin Data_A = {1'b1, 7'd25}; Data_B = {1'b0, 7'd10}; end          // -25, +10
  #120 begin Data_A = {1'b0, 7'd25}; Data_B = {1'b1, 7'd10}; end         // 25, -10
                                    //  B        A
  #160 begin Data_B = {1'b0, 7'd25}; Data_A = {1'b0, 7'd10}; end         //+25, +10
  #200 begin Data_B = {1'b1, 7'd25}; Data_A = {1'b1, 7'd10}; end         // -25, -10

  #240 begin Data_B = {1'b1, 7'd25}; Data_A = {1'b0, 7'd10}; end         // -25, +10
  #280 begin Data_B = {1'b0, 7'd25}; Data_A = {1'b1, 7'd10}; end         // +25, -10
  // Addition of matching numbers

  #320 begin Data_A = {1'b1,7'd0}; Data_B = {1'b1,7'd0}; end             // -0, -0
  #360 begin Data_A = {1'b0,7'd0}; Data_B = {1'b0,7'd0}; end             // +0, +0
  #400 begin Data_A = {1'b0,7'd0}; Data_B = {1'b1,7'd0}; end             // +0, -0
  #440 begin Data_A = {1'b1,7'd0}; Data_B = {1'b0,7'd0}; end             // -0, +0

  #480 begin Data_B = {1'b0, 7'd25}; Data_A = {1'b0, 7'd25}; end         // matching +
  #520 begin Data_B = {1'b1, 7'd25}; Data_A = {1'b1, 7'd25}; end         // matching –

  // Test of carry (negative numbers)
  #560 begin Data_A = 8'hf0; Data_B = 8'hf0; end                         // carry - -
  // Test of carry (positive numbers)
  #600 begin Data_A = 8'h70; Data_B = 8'h70; end                         // carry ++
 join
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

8.39 Block diagram and ASMD chart:

```
module Prob_8_39 (
  output [15: 0] PR, output done,
  input        [7: 0]   data_AR, data_BR, input  start, clock, reset_b
);

  Controller_P8_39 M0 (done, Ld_regs, Add_decr, start, zero, clock, reset_b);

 Datapath_P8_39 M1 (PR, zero, data_AR, data_BR, Ld_regs, Add_decr, clock, reset_b);
endmodule

module Controller_P8_16 (output done, output reg Ld_regs, Add_decr, input start, zero, clock, reset_b);
  parameter      s0 = 1'b0, s1 = 1'b1;
  reg      state, next_state;
  assign done = (state == s0);
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
  always @ (posedge clock, negedge reset_b)
    if (!reset_b) state <= s0; else state <= next_state;

  always @ (state, start, zero) begin
    Ld_regs = 0;
    Add_decr = 0;
    case (state)
      s0:        if (start) begin Ld_regs = 1; next_state = s1; end
      s1:        if (zero) next_state = s0; else  begin next_state = s1; Add_decr = 1; end
      default: next_state = s0;
    endcase
  end
endmodule

module  Datapath_P8_16 (
  output reg      [15: 0]  PR, output zero,
  input         [7: 0]    data_AR, data_BR, input Ld_regs, Add_decr, clock, reset_b
);

  reg       [7: 0]    AR, BR;
  assign       zero = ~( | AR);

  always @ (posedge clock, negedge reset_b)
    if (!reset_b) begin AR <= 8'b0; BR <= 8'b0; PR <= 16'b0; end
    else begin
      if (Ld_regs) begin AR <= data_AR; BR <= data_BR; PR <= 0; end
      else if (Add_decr) begin PR <= PR + BR; AR <= AR -1; end
    end
 endmodule

// Test plan – Verify;
// Power-up reset
// Data is loaded correctly
// Control signals assert correctly
// Status signals assert correctly
// start is ignored while multiplying
// Multiplication is correct
// Recovery from reset on-the-fly

module t_Prob_P8_16;
  wire          done;
  wire      [15: 0]  PR;
  reg [7: 0]    data_AR, data_BR;
  reg           start, clock, reset_b;

  Prob_8_16 M0 (PR, done, data_AR, data_BR, start, clock, reset_b);

  initial #500 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial fork
     reset_b = 0;
      #12 reset_b = 1;
    #40 reset_b = 0;
    #42 reset_b = 1;
    #90 reset_b = 1;
    #92 reset_b = 1;
  join
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
 initial fork
   #20 start = 1;
   #30 start = 0;
   #40 start = 1;
   #50 start = 0;
   #120 start = 1;
   #120 start = 0;
 join

 initial fork
   data_AR = 8'd5;       // AR > 0
   data_BR = 8'd20;

   #80 data_AR = 8'd3;
   #80 data_BR = 8'd9;

   #100 data_AR = 8'd4;
   #100 data_BR = 8'd9;
 join
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

8.40

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

The bytes of data will be read sequentially. Registers *Q* and *B* are organized to act as byte-wide parallel shift registers, taking 8 clock cycles to fill the pipe. The least significant byte of the multiplicand enters the most significant byte of *Q* and then moves through the bytes of Q to enter *B*, then proceed to occupy successive bytes of *B* until it occupies the least significant byte of *B*, and so forth until both *B* and *Q* are filled. Wait states are used to wait for *Run* and *Send_Data*.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
module Prob_8_40 (
  output    [7: 0]        Data_out,
  output                  Ready, Got_Data, Done_Product,
  input     [7: 0]        Data_in,
  input                   Start, Run, Send_Data, clock, reset_b
);

Controller M0 (
    Ready, Shift_in, Got_Data, Done_Product, Decr_P, Add_regs, Shift_regs, Shift_out,
    Start, Run, Send_Data, Zero, Q0, clock, reset_b
);
Datapath M1(Data_out, Q0, Zero, Data_in,
  Start, Shift_in, Decr_P, Add_regs, Shift_regs, Shift_out, clock
);
endmodule

module Controller (
    output reg Ready, Shift_in, Got_Data, Done_Product, Decr_P, Add_regs,
      Shift_regs, Shift_out,
    input Start, Run, Send_Data, Zero, Q0, clock, reset_b
);

  parameter   S_idle =     5'd20,
              S_Ld_0 =  5'd0,
              S_Ld_1 =  5'd1,
              S_Ld_2 =  5'd2,
              S_Ld_3 =  5'd3,
              S_Ld_4 =  5'd4,
              S_Ld_5 =  5'd5,
              S_Ld_6 =  5'd6,
              S_Ld_7 =  5'd7,
              S_wait_1 = 5'd8,  // Wait state
              S_add =    5'd9,
              S_Shift =  5'd10,
              S_product =    5'd11,
              S_wait_2 =     5'd12,  // Wait state
              S_Send_0 =   5'd13,
              S_Send_1 =   5'd14,
              S_Send_2 =   5'd15,
              S_Send_3 =   5'd16,
              S_Send_4 =   5'd17,
              S_Send_5 =   5'd18,
              S_Send_6 =   5'd19;

  reg    [4: 0]          state, next_state;

  always @ (posedge clock, negedge reset_b)
    if (~reset_b) state <= S_idle; else state <= next_state;

  always @ (state, Start, Run, Q0, Zero, Send_Data) begin
    next_state = S_idle;                  // Prevent accidental synthesis of latches
    Ready = 0;
    Shift_in = 0;
    Shift_regs = 0;
    Add_regs = 0;
    Decr_P = 0;
    Shift_out = 0;
    Got_Data = 0;
    Done_Product = 0;
```


All rights reserved.

```
   case (state)      // Assign by exception to default values
    S_idle:          begin
                       Ready = 1;
                       if (Start) begin next_state = S_Ld_0; Shift_in = 1; end
                     end
    S_Ld_0:          begin next_state = S_Ld_1; Shift_in = 1; end
    S_Ld_1:          begin next_state = S_Ld_2; Shift_in = 1; end
    S_Ld_2:          begin next_state = S_Ld_3; Shift_in = 1; end
    S_Ld_3:          begin next_state = S_Ld_4; Shift_in = 1; end
    S_Ld_4:          begin next_state = S_Ld_5; Shift_in = 1; end
    S_Ld_5:          begin next_state = S_Ld_6; Shift_in = 1; end
    S_Ld_6:          begin next_state = S_Ld_7; Shift_in = 1; end
    S_Ld_7:          begin Got_Data = 1;
                       if (Run) next_state = S_add;
                       else next_state = S_wait_1;
                     end
    S_wait_1:        if (Run) next_state = S_add; else next_state = S_wait_1;
    S_add:           begin next_state = S_Shift; Decr_P = 1; if (Q0) Add_regs = 1; end
    S_Shift:         begin Shift_regs = 1; if (Zero) next_state = S_product;
                     else next_state = S_add; end
    S_product:       begin
                       Done_Product = 1;
                       if (Send_Data) begin  next_state = S_Send_0; Shift_out = 1; end
                       else next_state = S_wait_2; end
    S_wait_2:        if (Send_Data) begin next_state =S_Send_0; Shift_out = 1; end
                     else next_state = S_wait_2;
    S_Send_0:        begin next_state = S_Send_1; Shift_out = 1; end
    S_Send_1:        begin next_state = S_Send_2; Shift_out = 1; end
    S_Send_2:        begin next_state = S_Send_3; Shift_out = 1; end
    S_Send_3:        begin next_state = S_Send_4; Shift_out = 1; end
    S_Send_4:        begin next_state = S_Send_5; Shift_out = 1; end
    S_Send_5:        begin next_state = S_Send_6; Shift_out = 1; end
    S_Send_6:        begin next_state = S_idle; Shift_out = 1; end
    default:         next_state = S_idle;
   endcase
  end
endmodule

module Datapath #(parameter dp_width = 32, P_width = 6) (
  output   [7: 0]        Data_out,
  output                 Q0, Zero,
  input    [7: 0]        Data_in,
  input                  Start, Shift_in, Decr_P, Add_regs, Shift_regs, Shift_out, clock
);
  reg   [dp_width - 1: 0]      A, B, Q;                  // Sized for datapath
  reg                          C;
  reg   [P_width - 1: 0]       P;
  assign                       Q0 = Q[0];
  assign                       Zero = (P == 0);          // counter is zero
  assign                       Data_out = {C, A, Q};

  always @ (posedge clock) begin
    if (Shift_in) begin
      P <= dp_width;
      A <= 0;
      C <= 0;
      B[7: 0]      <= B[15: 8];        // Treat B and Q registers as a pipeline to load data bytes
      B[15: 8]     <= B[ 23: 16];
      B[23: 16]    <= B[31: 24];
      B[31: 24]    <= Q[7: 0];
      Q[7: 0]      <= Q[15: 8];
      Q[15: 8]     <= Q[ 23: 16];
```


All rights reserved.

```
      Q[23: 16]  <= Q[31: 24];
      Q[31: 24]  <= Data_in;
    end
    if (Add_regs) {C, A} <= A + B;
    if (Shift_regs) {C, A, Q} <= {C, A, Q} >> 1;
    if (Decr_P) P <= P -1;
    if (Shift_out) begin {C, A, Q} <= {C, A, Q} >> 8; end
  end
endmodule

module t_Prob_8_40;
  parameter          dp_width = 32;          // Width of datapath
  wire  [7: 0]       Data_out;
  wire               Ready, Got_Data, Done_Product;
  reg                Start, Run, Send_Data, clock, reset_b;

  integer            Exp_Value;
  reg                Error;
  wire  [7: 0]       Data_in;
  reg [dp_width -1: 0]        Multiplicand, Multiplier;
  reg [2*dp_width -1: 0]      Data_register;        // For test patterns
  assign                      Data_in = Data_register [7:0];
  wire [2*dp_width -1: 0]     product;
  assign                      product = {M0.M1.C, M0.M1.A, M0.M1.Q};
  Prob_8_40  M0 (
    Data_out,  Ready, Got_Data, Done_Product, Data_in, Start, Run, Send_Data, clock, reset_b
);

  initial #2000 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial fork
    reset_b = 1;
    #2 reset_b = 0;
    #3 reset_b = 1;
  join
  initial fork
    Start =0;
    Run = 0;
    Send_Data = 0;
    #10 Start = 1;
    #20 Start = 0;


    #50 Run= 1;        // Ignored by controller
    #60 Run = 0;
    #120 Run = 1;
    #130 Run = 0;

    #830 Send_Data = 1;
    #840 Send_Data = 0;
  join
// Test patterns for multiplication

  initial begin
    Multiplicand = 32'h0f_00_00_aa;
    Multiplier = 32'h0a_00_00_ff;
    Data_register = {Multiplier, Multiplicand};
  end

 initial begin          // Synchronize input data bytes
  @ (posedge Start)
     repeat (15) begin
```


All rights reserved.

```
      @ (negedge clock)
        Data_register <= Data_register >> 8;
    end
  end
endmodule
```

Simulation results: Loading multiplicand ($0f0000aa_H$) and multiplier ($0a0000ff_H$), 4 bytes each, in sequence, beginning with the least significant byte of the multiplicand.

Note: *Product* is not valid until *Done_Product* asserts. The value of *Product* shown here (255_{10}) reflects the contents of *{C, A, Q}* after the multiplier has been loaded, prior to multiplication.

Note: The machine ignores a premature assertion of *Run*.

Note: *Got_Data* asserts at the 8th clock after Start asserts, i.e., 8 clocks to load the data.

Note: *Product*, *Multiplier*, and *Multiplicand* are formed in the test bench.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

Note: Product (64 bits) is formed correctly

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

8.41 (a)

(b) HDL model, test bench and simulation results for datapath unit.

```
module Datapath_unit
(
  output reg [15: 0]    R0, input [7: 0] Data, input Clr_P1_P0, Ld_P1_P0, Ld_R0, clock, rst);
   reg [7: 0]      P1, P0;

  always @ (posedge clock) begin
    if (Clr_P1_P0) begin P1 <= 0; P0 <= 0; end
    if (Ld_P1_P0) begin P1 <= Data; P0 <= P1; end
    if (Ld_R0) R0 <= {P1, P0};
  end
endmodule
```


All rights reserved.

```
// Test bench for datapath
module t_Datapath_unit ();
  wire [15: 0]   R0;
  reg [7: 0]     Data;
  reg            Clr_P1_P0,   Ld_P1_P0,   Ld_R0,   clock,   rst;

   Datapath_unit M0 (R0, Data, Clr_P1_P0, Ld_P1_P0, Ld_R0, clock, rst);

  initial #100 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial begin rst = 0; #2 rst = 1; end
  initial fork
    #20 Clr_P1_P0 = 0;
    #20 Ld_P1_P0 = 0;
    #20 Ld_R0 = 0;
    #20 Data = 8'ha5;
    #40 Ld_P1_P0 = 1;
   #50 Data = 8'hff;
    #60 Ld_P1_P0 = 0;
    #70 Ld_R0 = 1;
    #80 Ld_R0 = 0;
  join
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

(c) HDL model, test bench, and simulation results for the control unit.

```
module Control_unit (output reg Clr_P1_P0, Ld_P1_P0, Ld_R0, input En, Ld, clock, rst);
  parameter S_idle = 4'b0001, S_1 = 4'b0010, S_full = 4'b0100, S_wait = 4'b1000;
  reg [3: 0] state, next_state;

  always @ (posedge clock)
    if (rst) state <= S_idle;
    else state <= next_state;

  always @ (state, Ld, En) begin
    Clr_P1_P0 = 0;              // Assign by exception
    Ld_P1_P0 = 0;
    Ld_R0 = 0;
    next_state = S_idle;
    case (state)
      S_idle:     if (En) begin Ld_P1_P0 = 1; next_state = S_1; end
                  else next_state = S_idle;

      S_1:        begin Ld_P1_P0 = 1; next_state = S_full; end

      S_full:     if (!Ld) next_state = S_wait;
                  else begin
                    Ld_R0 = 1;
                    if (En) begin Ld_P1_P0 = 1; next_state = S_1;  end
                    else begin Clr_P1_P0 = 1; next_state = S_idle; end
                  end

      S_wait:     if (!Ld) next_state = S_wait;
                  else begin
                    Ld_R0 = 1;
                    if (En) begin Ld_P1_P0 = 1; next_state = S_1; end
                    else begin Clr_P1_P0 = 1; next_state = S_idle;  end
                  end
      default:    next_state = S_idle;
    endcase
  end
endmodule

// Test bench for control unit
module t_Control_unit ();
  wire  Clr_P1_P0, Ld_P1_P0, Ld_R0;
  reg   En, Ld, clock, rst;

  Control_unit M0 (Clr_P1_P0, Ld_P1_P0, Ld_R0, En, Ld, clock, rst);

  initial #200 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial begin rst = 0; #2 rst = 1; #12 rst = 0; end
  initial fork
    #20 Ld = 0;
    #20 En = 0;
    #30 En = 1; // Drive to S_wait
    #70 Ld = 1; // Return to S_1 to S_full tp S_wait
    #80 Ld = 0;
     #100 Ld = 1;    // Drive to S_idle
    #100 En = 0;
    #110 En = 0;
    #120 Ld = 0;
  join
endmodule
```


All rights reserved.

(c) Integrated system Note that the test bench for the integrated system uses the input stimuli from the test bench for the control unit and displays the waveforms produced by the test bench for the datapath unit.:

```
module Prob_8_41 (output [15: 0] R0, input [7: 0] Data, input En, Ld, clock, rst);
 wire Clr_P1_P0, Ld_P1_P0, Ld_R0;

 Control_unit M0 (Clr_P1_P0, Ld_P1_P0, Ld_R0, En, Ld, clock, rst);
 Datapath_unit M1 (R0, Data, Clr_P1_P0, Ld_P1_P0, Ld_R0, clock);

endmodule

module Control_unit (output reg Clr_P1_P0, Ld_P1_P0, Ld_R0, input En, Ld, clock, rst);
  parameter S_idle = 4'b0001, S_1 = 4'b0010, S_full = 4'b0100, S_wait = 4'b1000;
  reg [3: 0] state, next_state;

  always @ (posedge clock)
    if (rst) state <= S_idle;
    else state <= next_state;

  always @ (state, Ld, En) begin
    Clr_P1_P0 = 0;              // Assign by exception
    Ld_P1_P0 = 0;
    Ld_R0 = 0;
    next_state = S_idle;
    case (state)
      S_idle:     if (En) begin Ld_P1_P0 = 1; next_state = S_1; end
                  else next_state = S_idle;

      S_1:        begin Ld_P1_P0 = 1; next_state = S_full; end

      S_full:     if (!Ld) next_state = S_wait;
                  else begin
                    Ld_R0 = 1;
                    if (En) begin Ld_P1_P0 = 1; next_state = S_1; end
                    else begin Clr_P1_P0 = 1; next_state = S_idle; end
                  end

      S_wait:     if (!Ld) next_state = S_wait;
                  else begin
                    Ld_R0 = 1;
                    if (En) begin Ld_P1_P0 = 1; next_state = S_1; end
```


All rights reserved.

```
                  else begin Clr_P1_P0 = 1; next_state = S_idle; end
                end
      default:    next_state = S_idle;
    endcase
  end
endmodule

module Datapath_unit
(
  output reg [15: 0]   R0,
  input [7: 0]         Data,
  input                Clr_P1_P0,
                       Ld_P1_P0,
                       Ld_R0,
                       clock);
   reg [7: 0]     P1, P0;

  always @ (posedge clock) begin
    if (Clr_P1_P0) begin P1 <= 0; P0 <= 0; end
    if (Ld_P1_P0) begin P1 <= Data; P0 <= P1; end
    if (Ld_R0) R0 <= {P1, P0};
  end
endmodule


// Test bench for integrated system
module t_Prob_8_41 ();
  wire [15: 0]   R0;
  reg [7: 0]     Data;
  reg            En, Ld, clock, rst;

  Prob_8_41 M0 (R0, Data, En, Ld, clock, rst);

  initial #200 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial begin rst = 0; #10 rst = 1; #20 rst = 0; end
  initial fork

    #20 Data = 8'ha5;
    #50 Data = 8'hff;

    #20 Ld = 0;
    #20 En = 0;
    #30 En = 1; // Drive to S_wait
    #70 Ld = 1; // Return to S_1 to S_full tp S_wait
    #80 Ld = 0;
     #100 Ld = 1;    // Drive to S_idle
    #100 En = 0;
    #110 En = 0;
    #120 Ld = 0;
  join
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

8.42

```
module Datapath_BEH
  #(parameter dp_width = 8, R2_width = 4)
(
  output [R2_width -1: 0] count, output E, //output reg E,
  output Zero, input [dp_width -1: 0] data,
  input Load_regs, Shift_left, Incr_R2, clock, reset_b);
  reg [dp_width -1: 0] R1;
  reg [R2_width -1: 0] R2;


  assign E = R1[dp_width -1];


  assign count = R2;
  assign Zero = ~(| R1);
  always @ (posedge clock) begin
   // E <= R1[dp_width -1] & Shift_left;
   // if (Load_regs) begin R1 <= data; R2 <= {R2_width{1'b1}}; end
    if (Load_regs) begin R1 <= data; R2 <= {R2_width{1'b0}}; end
    if (Shift_left) R1 <=  R1 << 1;
    //if (Shift_left) {E, R1} <= {E, R1} << 1;
    if (Incr_R2) R2 <= R2 + 1;
  end
endmodule

module Controller_BEH (
  output        Ready,
  output reg    Load_regs,
  output        Incr_R2, Shift_left,
  input         Start, Zero, E, clock, reset_b
);
  parameter S_idle = 0, S_1 = 1, S_2 = 2, S_3 = 3;
  reg [1:0] state, next_state;

  assign Ready = (state == S_idle);
  assign Incr_R2 = (state == S_1);
  assign Shift_left = (state == S_2);

  always @ (posedge clock, negedge reset_b)
    if (reset_b == 0) state <= S_idle;
    else state <= next_state;

  always @ (state, Start, Zero, E) begin
    Load_regs = 0;
    case (state)
      S_idle: if (Start) begin Load_regs = 1; next_state = S_1; end
                  else next_state = S_idle;
      S_1:        if (Zero) next_state = S_idle; else next_state = S_2;

      S_2:        next_state = S_3;
      //S_3:      if (E) next_state = S_1; else next_state = S_2;
      S_3:        if (E) next_state = S_1; else if (Zero) next_state = S_idle; else next_state = S_2;

    endcase
  end
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
// Integrated system
module Count_Ones_BEH_BEH
# (parameter dp_width = 8, R2_width = 4)
(
  output [R2_width -1: 0]      count,
  input [dp_width -1: 0]       data,
  input                        Start, clock, reset_b
);
  wire  Load_regs, Incr_R2, Shift_left, Zero, E;
  Controller_BEH M0 (Ready, Load_regs, Incr_R2, Shift_left, Start, Zero, E, clock, reset_b);
  Datapath_BEH M1 (count, E, Zero, data, Load_regs, Shift_left, Incr_R2, clock, reset_b);
endmodule

// Test plan for integrated system
// Test for data values of 8'haa, 8'h00, 8'hff.

// Test bench for integrated system

module t_count_Ones_BEH_BEH ();
  parameter dp_width = 8, R2_width = 4;
  wire [R2_width -1: 0] count;
  reg [dp_width -1: 0] data;
  reg Start, clock, reset_b;

  Count_Ones_BEH_BEH M0 (count, data, Start, clock, reset_b);
  initial #700 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial bebgin reset_b = 0; #2 reset_b = 1; enbd
  initial fork
    data = 8'haa;         // Expect count = 4
    Start = 0;
    #20 Start = 1;
    #30 Start = 0;
    #40 data = 8'b00;   // Expect count = 0
    #250 Start = 1;
    #260 Start = 0;
    #280 data = 8'hff;
    #280 Start = 1;
    #290 Start = 0;
  join
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

CHAPTER 9

9.1 Oscilloscope display:

BCD count: Oscilloscope displays from 0000 to 1001

Output pattern:

QA = alternate 1's and 0s
QB = Two 1's, two 0's, two 1's, four 0's
QC = Four 1's, six 0's
QD = Two 1's, eight 0's.

Other counts:

(a) 0101 must reset at 0110 – connect QB to R1, QC to R2
(b) 0111 must reset at 1000 – connect QD to both R1 and R2
(c) 1011 must reset at 1100 – connect QC to R1, QD to R2

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

9.2 Truth table:

Inputs A	Inputs B	NAND	NOR	NOT(A)	AND	OR	XOR
0	0	1	0	1	0	0	0
0	1	1	1	0	0	1	1
1	0	1	1	1	0	1	1
1	1	0	1	0	1	1	0

Waveforms:

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

9.3

Logic Diagram

$F = xy' + yz$

Boolean Functions:

$F_1 = C' + AB'D'$

Boolean Functions:

$F_2 = BD + CD + AB'D'$

2 ICs: 7400, 7410

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

Complement:

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

9.4

Design Example:

Majority Logic

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

Decoder Implementation

$F_1 = xz + x'y'z' = \Sigma(0, 5, 7)$

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

$F_3 = xy + x'y'z = \Sigma(1, 6, 7)$

9.5 Gray code to Binary – See solution to Prob. 4.7.

9's complementer – See solution to Prob. 4.18.

w = A'B'C'
x = BC' + B'C
y = C
z = D'
E = AB + AC

3 ICs: 7400, 7404, 7410

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

9.6

$A = \sum (0, 2, 3, 6, 7, 8, 9, 12, 13)$
$B = \sum (0, 2, 3, 4, 512, 13, 14)$
$C = \sum (0, 1, 3, 5, 6, 9, 10, 13, 14)$
$D = \sum (0, 7, 11)$

Mux A

D0	D1	D2	D3	D4	D5	D6	D7
1	w	w′	w′	w	w	w′	w′

Mux B

D0	D1	D2	D3	D4	D5	D6	D7
w′	0	w′	w′	1	1	w	0

Mux C

D0	D1	D2	D3	D4	D5	D6	D7
w′	1	w	w′	0	1	1	0

Mux D

D0	D1	D2	D3	D4	D5	D6	D7
w′	0	0	w	0	0	0	w′

9.7

Half - Adder

Full- Adder

Parallel adder - See circuit of Fig. 9.10.
Adder-subtractor – See circuit of Fig. 9.11.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

Operation	Inputs M	A	B	C_0	Outputs S	C_4	
9 + 5 = 14	0	1001	0101	0	1110	0	sum < 15
9 + 9 = 19 = 16 + 2	0	1001	1001	0	0010	1	sum > 15
9 + 15 = 24 = 16 + 8	0	1001	1111	0	1000	1	sum > 15
9 - 5 = 4	1	1001	0101	1	0100	1	A > B
9 - 9 = 0	1	1001	1001	1	0000	1	A = B
9 - 15 = -6	1	1001	1111	1	1010	0	A < B

9.8 **SR Latch:** See Fig. 5.4.

D Latch:

Let $CP = C$, x = output of gate 4.

$x = [(DC)'C]' = (D'C)'$

Master-Slave D Flip-Flop: The circuit is as in Fig. 5.9.

The oscilloscope display:

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

Edge-Triggered D Flip-Flop: Circuit is shown in Fig. 5.10.

IC Flip-Flops:

Connect all inputs to toggle switches, the clock to a pulser, and the outputs to indicator lamps.

9.9 Up-Down Counter with Enable:

$J_B = K_B = E$ (Complement B when E = 1)

$J_A = K_A = E\ (Bx + B'x')$
Complement A when E = 1 and:
B = 1 when x = 1 (Count up)
B = 0 when x = 0 (Count down)

State Diagram:

$J_A = B$ $\quad J_B = Ax + A'x' = (A \oplus x)'$ $\quad Y = A \oplus B \oplus x$
$K_A = B'$ $\quad K_B = Ax + A'x' = (A \oplus x)'$

Design of Counter: ABCD

$J_A = K_A = B(CD)$ $\quad$ 0000 → 0101 → 090
$J_B = K_B = CD$ $\quad$ 1000 → 1001 → 1010
$J_C = D$ $\quad K_C = AD$
$J_D = K_D = 1$

All rights reserved.

9.10 Ripple counter: See Fig. 6.8
Down counter: Either take outputs from Q' outputs or connect complement Q' to next clock input.

Synchronous counter: See Fig. 6.12.

BCD counter: See solution to Prob. 6.19.
Unused states:

Binary counter wth parallel load:

Connect QA and QD through a NAND gate to the load. See Fig. 6.15.

9.11 Ring counter:

See Fig. 6.17(a).
States of register:

QA	*QB*	*QC*	*QD*
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

Switch-tail ring counter: See Fig. 6.18(a). Connect (QD)' at pin 12 to the serial input at pin 4. State sequence as in Fig. 6.18(b).

Feedback shift register: Serial input = QC ⊕ QD (Use 7486).

Sequence of states:

QA	*QB*	*QC*	*QD*								
1	0	0	0	0	1	1	0	1	1	1	0
0	1	0	0	1	0	1	1	1	1	1	1
0	0	1	0	0	1	0	1	0	1	1	1
1	0	0	1	1	0	1	0	0	0	1	1
1	1	0	0	1	1	0	1	0	0	0	1

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

Bidirectional shift register with parallel load:

Function table:

74195			74157		
Clear	Clock	SH/LD	STROBE	SELECT	Function
0	x	x	x	x	Async clear
1	↑	1	x	x	Shift right ($QA \rightarrow QB$)
1	↑	0	0	1	Shift left (Select B)*
1	↑	0	0	0	Parallel Load (Select A)
1	↑	0	1	x	Synchronous clear

* B inputs come from QA-QD shifted by one position.

9.12

9.13 Testing the RAM:

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

Memory Expansion:

9.14 Circuit Analysis – Answers to questions:

1) Resets to 0 the two 74194 ICs, the two D flip-flops, and the start SR latch. This makes *S1S0* = 11 (parallel load).
2) The start switch sets the SR latch to 1. The clock pulses load 0000_0001 into the 8-bit register. If the start switch stays on, the register never clears to all 0s when *S1S0* = 11 (right-most *QD* stays on).
3) Pressing the pulser makes *S1S0* = 10 and the light shifts left. When *QC* becomes 1, the start SR latch is cleared to 0. When *QA* of the left 74194 becomes 1, it changes *S1* to 0 (through the *PR* input) and *S0* to 1 (through the *CLR* input. with *S1S0* = 01, the single light shifts right.
4) If the pulser is pressed while the light is moving to the left or the right, *S1S0* becomes 11 and all 0s are loaded into the register in parallel. The light goes off.
5) When the right-most *QD* becomes a 1, *S1S0* changes from 01 (shift right) to 11 (parallel load). If the pulser is pressed before the next clock pulse, *S1S0* goes to 10 (shift left). If not pressed, an all 0s value is loaded into the register in parallel. (Provided the start switch is in the logic 1 position.)

Lamp Ping-Pong

Add a left pulser. Three wire changes to the D flip-flop on the left:
1) Connect the clock input of the flip-flop to the pulser.
2) Connect the D input to the *QA* of the left 74197
3) Connect the input of the inverter (that goes to *PR*) to ground.

Counting the Losses

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

9.15 Clock Pulse Generator

$t_L = 0.693\ R_B C = 10^{-6}$
$R_B = 10^{-6} /(0.693 \times 0.001 \times 10^{-6}) = 10^3 / 0.693 = 1.44\ K\Omega$ (Use $R_B = 1.5\ K\Omega$)

$t_H/t_L = 0.693\ (R_A + R_B)C /(0.693\ R_B\ C) = (R_A + R_B) / R_B = 9/ 1 = 9$
$9\ R_B = R_A + R_B \quad R_A = 8\ R_B = 8 \times 1.5\ K\Omega = 12\ K\Omega$

Oscilloscope Waveforms (Actual results may be off by $\pm$ 20 %.)

Variable Frequency Pulse Generator:

20 KHz: $10^{-3} / 20 = 0.05 \times 10^{-3} = 50\ \mu s$
100 KHz: $10^{-3} / 100 = 10^{-5} = 10\ \mu s$

$t_H = 49\ \mu s$: $(R_A + R_P + R_B) / R_B = 49/ 1 = 49$
$R_P = 48\ R_B - R_A = 48 \times 1.5 - 11 = 60\ K\Omega$

9.16 Control of Register

SW_1	SW_2	
0	0	*No change*
0	1	*shift right*
1	1	*Load*

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

Checking the Circuit:

	Carry	Register
Initial	0	0 0 0 0
+ 0110	0	0110
+ 1110	1	0100
+ 1101	1	0001
+ 0101	0	0110
+ 0011	0	1001

Circuit Operation:

Address	Carry	RAM	
0	0	0110	RAM Value
1	0	0110	RAM + Register
2	0	0011	Shfit Register
3		1110	RAM Value
4	1	0001	RAM + Register
5	1	1000	SHIFT
6		1101	RAM Value
7	1	0101	RAM + Register
8	1	1010	SHIFT
9		0101	RAM Value
10	0	1111	RAM + Register
11	0	0111	SHIFT
12		0011	RAM Value
13	0	1010	RAM + REgiser
14	0	0101	SHIFT

9.17 Multiplication Example (11 x 15 = 165)

Multiplicand B = 1111

		C	*A*	*Q*	*P*
Initial:	$T_1 = 1$	0	0000	1011	0000
$T_2 = 1$	Add B; P <= P+1		1111		
		0	1111	1011	0001
$T_3 = 1$	Shift CAQ	0	0111	1101	0001
$T_2 = 1$	Add B; P <= P+1		1111		
		1	0110	1101	0010
$T_3 = 1$	Shift CAQ	0	1011	0110	0010
$T_2 = 1$	P <= P+1	0	1011	0110	0011
$T_3 = 1$	Shift CAQ	0	0101	1011	0011
$T_2 = 1$	Add B; P <= P+1		1111		
		1	0100	1011	0100
$T_3 = 1$	Shift CAQ	0	1010	0101	0100
$T_0 = 1$	(Because $P_C = 1$)		1010	0101	= Product

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

Data Processor Design

Load Q T_1	Load A T_2Q_1	Shift AQ T_3	Register Q $S_1\ S_0$	Register A $S_1\ S_0$
0	0	0	0 0	0 0
1	0	0	1 1	0 0
0	1	0	0 0	1 1
0	0	1	0 1	0 1

$S_1(Q) = T_1$ $\qquad$ $S_1(A) = T_2Q_1$

$S_4(Q) = T_1 + T_3$ $\qquad$ $S_0(A) = T_2Q_1 + T_3$

Design of Control: See Section 8.8.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012, All rights reserved.

SOLUTIONS FOR SECTION 9.19

Supplement to Experiment 2:

(a)

Initially, with *xy* = 00, *w1* = w2 = 1, *w3* = *w4* = 0 and *F* = 0. *w1* should change to 0 10ns after *xy* changes to 01. *w4* should change to 1 20 ns after xy changes to 01. *F* should change from 0 to 1 30 ns after *w4* changes from 0 to 1, i.e., 50 ns after *xy* changes from 00 to 01. w3 should remain unchanged because x = 0 for the entire simulation.

(b)

```
`timescale 1ns/1ps

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

and #20 (w3, x, w1);
not #10 (w1, x);
and #20 (w4, y, w1);
not #10 (w2, y);
or  #30 (F, w3, w4);        A

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;
    #100 y = 1;
  join
endmodule
```

(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 = 1 at t = 100 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. The waveforms show the response to xy = 01 applied at t = 100 ns.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

Supplement to Experiment 4:

(a)

```
// Gate-level description of circuit in Fig. 4-2
module Circuit_of_Fig_4_2 (
  output   F1, F2,
  input    A, B, C);
  wire  T1, T2, T3, F2_not, E1, E2, E3;
  orG1 (T1, A, B, C);
  and   G2 (T2, A, B, C);
  and   G3 (E1, A, B);
  and   G4 (E2, A, C);
  and   G5 (E3, B, C);
  orG6 (F2, E1, E2, E3);
  not   G7 (F2_not, F2);
  and   G8 (T3, T1, F2_not);
  orG9 (F1, T2, T3);
endmodule

module t_Circuit_of_Fig_4_2;
  reg   [2: 0] D;
  wire  F1, F2;
  parameter stop_time = 100;

  Circuit_of_Fig_4_2 M1 (F1, F2, D[2], D[1], D[0]);

  initial # stop_time $finish;
  initial begin                    // Stimulus generator
     D = 3'b000;
   repeat (7)
     #10 D = D + 1'b1;
  end

  initial begin
   $display ("A     B     C     F1   F2");
   $monitor ("%b     %b     %b     %b    %b", D[2], D[1], D[0], F1, F2);
  end
endmodule
```


All rights reserved.

```
/*
A   B   C   F1  F2
0   0   0   0   0
0   0   1   1   0
0   1   0   1   0
0   1   1   0   1
1   0   0   1   0
1   0   1   0   1
1   1   0   0   1
1   1   1   1   1

*/
```

The simulation results demonstrate the behavior of a full adder, with F1 = sum, and F2 – carry.

(b)

```
// 3-INPUT MAJORITY DETECTOR CIRCUIT.
// Circuit implements F = xy + xz +yz.
module Majority_Detector (output F, input x, y, z);
  wire wl, w2, w3;
  nand     nl(wl, x, y),
           n2(w2, x, z),
           n3(w3, y, z),
           n4(F, wl, w2, w3) ;
endmodule

// Test bench
//Treating inputs to majority detector as a vector, reg [2:0]D; //D[2] = x, D[l] = y, D[0] = z. wire F;
module t_Majority_Detector ();
  wire F;
 reg [2: 0] D;
 wire x = D[2];
 wire y = D[1];
 wire z = D[0];

Majority_Detector M0 (F, x, y, z);

 initial #100 $finish;
 initial $monitor ($time,, "xyz = %b F = %b", D, F);
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
 initial begin
   D = 0;
   repeat (7)
     #10 D = D + 1;
  end
endmodule
```

Simulation results:

```
  0 xyz = 000 F = 0
 10 xyz = 001 F = 0
 20 xyz = 010 F = 0
 30 xyz = 011 F = 1
 40 xyz = 100 F = 0
 50 xyz = 101 F = 1
 60 xyz = 110 F = 1
 70 xyz = 111 F = 1
```

Supplement to Experiment 5: See the solution to Prob. 4.42.

Supplement to Experiment 7:

(a)

```
//BEHAVIORAL DESCRIPTION OF 7483 4-BIT ADDER,
module Adder_7483 (
  output S4, S3, S2, S1, C4,
  input A4, A3, A2, A1, B4, B3, B2, B1, C0, VCC, GND
);
// Note: connect VCC and GND to supply1 and supply0 in the test bench
  wire [4: 1] sum;
  wire [4: 1] A = {A4, A3, A2, A1};
  wire [4: 1] B = {B4, B3, B2, B1};
  assign S4 = sum[4];
  assign S3 = sum[3];
  assign S2 = sum[2];
  assign S1 = sum[1];

  assign {C4, sum} = A + B + C0;
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
module t_Adder_7483 ();
  wire S4, S3, S2, S1, C4;
  wire A4, A3, A2, A1, B4, B3, B2, B1;
  reg C0;
  supply1 VCC;
  supply0 GND;
  reg [4:1] A, B;
  assign A4 = A[4];
  assign A3 = A[3];
  assign A2 = A[2];
  assign A1 = A[1];
  assign B4 = B[4];
  assign B3 = B[3];
  assign B2 = B[2];
  assign B1 = B[1];

  Adder_7483 M0 (S4, S3, S2, S1, C4, A4, A3, A2, A1, B4, B3, B2, B1, C0, VCC, GND);

  initial #2600 $finish;
  initial begin
    A = 0; B = 0; C0 = 0;
    repeat (256) #5 {A, B} = {A, B} + 1;
    A = 0; B = 0; C0 = 1;
    repeat (256) #5 {A, B} = {A, B} + 1;
  end
endmodule
```

(b)

```
module Supp_9_17b (output [4: 1] S, output carry, input [4: 1] A, B, input M, VCC, GND);
  wire B4, B3, B2, B1;
  xor (B4, M, B[4]);
  xor (B3, M, B[3]);
  xor (B2, M, B[2]);
  xor (B1, M, B[1]);
  Adder_7483 M0 (S[4], S[3], S[2], S[1], carry, A[4], A[3], A[2], A[1], B4, B3, B2, B1, M, VCC, GND);
endmodule

module Adder_7483 (
  output S4, S3, S2, S1, C4,
  input A4, A3, A2, A1, B4, B3, B2, B1, C0, VCC, GND
);
// Note: connect VCC and GND to supply1 and supply0 in the test bench
  wire [4: 1] sum;
  wire [4: 1] A = {A4, A3, A2, A1};
  wire [4: 1] B = {B4, B3, B2, B1};
  assign S4 = sum[4];
  assign S3 = sum[3];
  assign S2 = sum[2];
  assign S1 = sum[1];
  assign {C4, sum} = A + B + C0;
endmodule
```


All rights reserved.

```
module t_Supp_9_17b ();
  wire [4: 1] S;
  wire carry;
  reg C0;
  reg [4: 1] A, B;
  reg M;
  supply1 VCC;
  supply0 GND;

  Supp_9_17b M0 (S, carry, A, B, M, VCC, GND);
  initial #2600 $finish;
  initial begin
    A = 0; B = 0; M = 0;
    repeat (256) #5 {A, B} = {A, B} + 1;
    A = 0; B = 0; M = 1;
    repeat (256) #5 {A, B} = {A, B} + 1;
  end
endmodule
```

(c), (d)

```
module supp_9_7c (output S3, S2, S1, S0, C, V, input A3, A2, A1, A0, B3, B2, B1, B0, M);
  wire [3: 0] Sum, B;
  assign S3 = Sum[3];
  assign S2 = Sum[2];
  assign S1 = Sum[1];
  assign S0 = Sum[0];
  wire [3:0]  A = {A3, A2, A1, A0};
  xor(B[3], B3, M);
  xor(B[2], B2, M);
  xor(B[1], B1, M);
  xor(B[0], B0, M);
  xor (V, C, C3);
  ripple_carry_4_bit_adder M0 (Sum, C, C3, A, B, M);
endmodule

module t_supp_9_7c ();
  wire S3, S2, S1, S0, C, V;
  reg A3, A2, A1, A0, B3, B2, B1, B0, M;
  wire [3: 0] sum = {S3, S2, S1, S0};
  wire [3: 0] A = {A3, A2, A1, A0};
  wire [3: 0] B = {B3, B2, B1, B0};

  supp_9_7c M0 (S3, S2, S1, S0, C, V, A3, A2, A1, A0, B3, B2, B1, B0, M);

  initial #2600 $finish;
  initial begin
    {A3, A2, A1, A0, B3, B2, B1, B0} = 0; M = 0;
    repeat (256) #5 {A3, A2, A1, A0, B3, B2, B1, B0} = {A3, A2, A1, A0, B3, B2, B1, B0} + 1;
    {A3, A2, A1, A0, B3, B2, B1, B0} = 0; M = 1;
    repeat (256) #5 {A3, A2, A1, A0, B3, B2, B1, B0} = {A3, A2, A1, A0, B3, B2, B1, B0} + 1;
  end
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
module half_adder (output S, C, input x, y);        // Verilog 2001, 2005 syntax
 // Instantiate primitive gates
  xor (S, x, y);
  and (C, x, y);
endmodule

module full_adder (output S, C, input x, y, z);
  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

// Modify for C3 output
module ripple_carry_4_bit_adder ( output [3: 0] Sum, output C4, C3, input [3:0]  A, B, input C0);
  wire       C1, C2;     // Intermediate carries
// Instantiate chain of full adders
  full_adder     FA0 (Sum[0], C1, A[0], B[0], C0),
                 FA1 (Sum[1], C2, A[1], B[1], C1),
                 FA2 (Sum[2], C3, A[2], B[2], C2),
                 FA3 (Sum[3], C4, A[3], B[3], C3);
endmodule
```

Addition:

Subtraction:

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

Supplement to Experiment 8:

(a)

```
module Flip_flop_7474 (output reg Q, input D, CLK, preset, clear);
  always @ (posedge CLK, negedge preset , negedge clear)
  if (!preset)              Q <= 1'b1;
  else if (!clear)          Q <= 1'b0;
  else                      Q <= D;
endmodule

module t_Flip_flop_7474 ();
  wire   Q;
  reg    D, CLK, preset, clear;

  Flip_flop_7474 M0 (Q, D, CLK, preset, clear);

  initial #150 $finish;
  initial begin CLK = 0; forever #5 CLK = ~CLK; end

  initial fork
    preset = 0; clear = 0;
    #20 preset = 1;
    #40 clear = 1;
  join

  initial begin D = 0; #60 forever #20 D = ~D; end
endmodule
```

(b)

```
//Solution to supplement Experiment 8(b)
//Behavioral description of a 7474 D flip-flop with Q_not
module Flip_Flop_7474_with_Q_not (output reg Q, Q_not, input D, CLK, Preset, Clear);

  always @ (posedge CLK, negedge Preset, negedge Clear)
  /* case ({Preset, Clear})
     2'b00:  begin Q <= 1; Q_not <= 1; end
     2'b01:  begin Q <= 1; Q_not <= 0; end
     2'b10:  begin Q <= 0; Q_not <= 1; end
     2'b11:  begin Q <= D; Q_not <= ~D; end

     // NOTE: Q_not <= ~Q will produce a pipeline effect and delay Q_not by one clock
    endcase*/
if (Preset == 0) begin  Q <= 1; if (Clear == 0) Q_not <= 1; else Q_not <= 0; end
else if (Clear == 0) begin Q <= 0; Q_not <= 1; end
else begin Q <= D; Q_not <= ~D; end
```


All rights reserved.

```
endmodule

 // Note: this model will not work if Preset and Clear are // both brought low and then high again.
 // A case statement for both Q and Q_not is also OK.
module t_Flip_Flop_7474_with_Q_not ();
  wire   Q, Q_not;
  reg    D, CLK, Preset, Clear;

  Flip_Flop_7474_with_Q_not M0 (Q, Q_not, D, CLK, Preset, Clear);

  initial #250 $finish;
  initial begin CLK = 0; forever #5 CLK = ~CLK; end

  initial fork
    Preset = 1; Clear = 1;
    #50 Preset = 0;
    #80 Clear =
```

Supplement to Experiment #9:

(a)

```
module Figure_9_9a (output reg y, input x, clock, reset_b);
  reg [1: 0] state, next_state;
  parameter S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 2'b11;
  always @ (posedge clock, negedge reset_b) if (reset_b == 0) state <= S0; else state <= next_state;
  always @ (state, x) begin
    y = 0;
    case (state)
      S0:if (x) begin next_state = S0; y = 1; end else begin next_state = S1; y = 0; end
      S1:if (x) begin next_state = S3; y = 0; end else begin next_state = S2; y = 1; end
      S2:if (x) begin next_state = S1; y = 0; end else begin next_state = S0; y = 1; end
      S3:if (x) begin next_state = S2; y = 1; end else begin next_state = S3; y = 0; end
    endcase
  end
endmodule

module t_Figure_9_9a ();
  wire y;
  reg x, clock, reset_b;

  Figure_9_9a M0 (y, x, clock, reset_b);

  initial #200 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial fork
    reset_b = 0;
```


All rights reserved.

```
    x = 0;              // S0. S1, S2 after release of reset_b
    #10 reset_b = 1;
    #40 x = 1;          // Stay in S0
    #60 x= 0;           // S1, S2
    #80 x = 1;          //  s1, S3,
    #100  x = 0;// S3
    #130 x = 1;         // S2, S1, S3 cycle
  join
endmodule
```

(b) The solution depends on the particular design.

(c, d)
Note: The HDL description of the state diagram produces outputs *T0*, *T1*, and *T2*. Additional logic must form the signals that control the datapath unit (*Load_regs*, *Incr_P*, *Add_regs*, and *Shift_regs)*. An alternative controller that generates the control signals, rather than the states, as the outputs is given below too. It produces identical simulation results.

```
module Supp_9_9cd # (parameter dp_width = 5)
(
  output   [2*dp_width - 1: 0]    Product,
  output                          Ready,
  input    [dp_width - 1: 0]      Multiplicand, Multiplier,
  input                           Start, clock, reset_b
);
  wire Load_regs, Incr_P, Add_regs, Shift_regs, Done, Q0;

  Controller M0 (
    Ready, Load_regs, Incr_P, Add_regs, Shift_regs, Start, Done, Q0,
    clock, reset_b
);

Datapath M1(Product, Q0, Done, Multiplicand, Multiplier,
  Start, Load_regs, Incr_P, Add_regs, Shift_regs, clock, reset_b);
endmodule

/* // This alternative controller directly  produces the signals needed to control the datapath.
module Controller (
  output Ready,
  output reg Load_regs, Incr_P, Add_regs, Shift_regs,
  input Start, Done, Q0, clock, reset_b
);

  parameter   S_idle =        3'b001,          // one-hot code
              S_add =    3'b010,
              S_shift =   3'b100;
  reg    [2: 0]       state, next_state;      // sized for one-hot
  assign      Ready = (state == S_idle);
```


All rights reserved.

```
  always @ (posedge clock, negedge reset_b)
    if (~reset_b) state <= S_idle; else state <= next_state;

  always @ (state, Start, Q0, Done) begin
    next_state = S_idle;
    Load_regs = 0;
    Incr_P = 0;
    Add_regs = 0;
    Shift_regs = 0;
    case (state)
      S_idle:     if (Start) begin next_state = S_add; Load_regs = 1; end
      S_add:      begin next_state = S_shift; Incr_P = 1; if (Q0) Add_regs = 1; end
      S_shift:    begin Shift_regs = 1;
                    if (Done) next_state = S_idle;
                    else next_state = S_add;
                  end
      default:    next_state = S_idle;
    endcase
  end
endmodule
*/


// This controller has an embedded unit to generate T0, T1, and T2 and additional logic to form // // the
signals needed to control the datapath.

module Controller (
  output Ready, Load_regs, Incr_P, Add_regs, Shift_regs,
  input Start, Done, Q0, clock, reset_b
);

  State_Generator M0 (T0, T1, T2, Start, Done, Q0, clock, reset_b);
  assign Ready = T0;
  assign Load_regs = T0 && Start;
  assign Incr_P = T1;
  assign Add_regs = T1 && Q0;
  assign Shift_regs = T2;
endmodule

module State_Generator (output T0,T1, T2, input Start, Done, Q0, clock, reset_b);
  parameter        S_idle =    3'b001,          // one-hot code
                   S_add =    3'b010,
                   S_shift =   3'b100;
  reg    [2: 0]    state, next_state;       // sized for one-hot
  assign    T0 = (state == S_idle);
  assign    T1 = (state == S_add);
  assign    T2 = (state == S_shift);

  always @ (posedge clock, negedge reset_b)
    if (~reset_b) state <= S_idle; else state <= next_state;
```


All rights reserved.

```
  always @ (state, Start, Q0, Done) begin
    next_state = S_idle;
    case (state)
      S_idle:     if (Start) next_state = S_add;
      S_add:      next_state = S_shift;
      S_shift:    if (Done) next_state = S_idle; else next_state = S_add;
      default:    next_state = S_idle;
    endcase
  end
endmodule

module Datapath #(parameter dp_width = 5, BC_size = 3) (
  output [2*dp_width - 1: 0] Product, output Q0, output Done,
  input [dp_width - 1: 0] Multiplicand, Multiplier,
  input Start, Load_regs, Incr_P, Add_regs, Shift_regs, clock, reset_b
);
// Default configuration: 5-bit datapath
  reg    [dp_width - 1: 0]        A, B, Q;                    // Sized for datapath
  reg                             C;
  reg    [BC_size - 1: 0]         P;                  // Bit counter

  assign Q0 = Q[0];
  assign    Done = (P == dp_width );                          // Multiplier is exhausted
  assign Product = {C, A, Q};
  always @ (posedge clock, negedge reset_b)
    if (reset_b == 0) begin                 // Added to this solution, but
      P <= 0;                               // not really necessary since Load_regs
      B <= 0;                               // initializes the datapath
      C <= 0;
      A <= 0;
      Q <= 0;
    end
    else begin
    if (Load_regs) begin
      P <= 0;
      A <= 0;
      C <= 0;
      B <= Multiplicand;
      Q <= Multiplier;
    end
    if (Add_regs) {C, A} <= A + B;
    if (Shift_regs) {C, A, Q} <= {C, A, Q} >> 1;
    if (Incr_P) P <= P+1 ;
  end
endmodule

module t_Supp_9_9cd;
  parameter                       dp_width = 5;             // Width of datapath
  wire  [2 * dp_width - 1: 0]     Product;
  wire                            Ready;
  reg    [dp_width - 1: 0]        Multiplicand, Multiplier;
  reg                             Start, clock, reset_b;
  integer                         Exp_Value;
  reg                             Error;

  Supp_9_9cd M0(Product, Ready, Multiplicand, Multiplier, Start, clock, reset_b);

  initial #115000 $finish;
  initial begin clock = 0; #5 forever #5 clock = ~clock; end
```


All rights reserved.

```
initial fork
  reset_b = 1;
  #2 reset_b = 0;
  #3 reset_b = 1;
 join
 always @ (negedge Start) begin
  Exp_Value = Multiplier * Multiplicand;
  //Exp_Value = Multiplier * Multiplicand +1;    // Inject error to confirm detection
 end
 always @ (posedge Ready) begin
  # 1 Error <= (Exp_Value ^ Product) ;
 end

 initial begin
  #5 Multiplicand = 0;
  Multiplier = 0;

  repeat (32) #10 begin
    Start = 1;
     #10 Start = 0;
     repeat (32) begin
       Start = 1;
       #10 Start = 0;
       #100 Multiplicand = Multiplicand + 1;
     end
    Multiplier = Multiplier + 1;
  end
 end
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

Supplement to Experiment #10:

```
module Counter_74161 (
  output      QD, QC, QB, QA,     // Data output
  output      COUT,               // Output carry
  input       D, C, B, A,         // Data input
  input       P, T,               // Active high to count
              L,                  // Active low to load
              CK,                 // Positive edge sensitive
              CLR                 // Active low to clear
);

  reg [3: 0] A_count;
  assign QD = A_count[3];
  assign QC = A_count[2];
  assign QB = A_count[1];
  assign QA = A_count[0];

  assign COUT = ((P == 1) && (T == 1) && (L == 1) && (A_count == 4'b1111));

  always @ (posedge CK, negedge CLR)
    if (CLR == 0)          A_count <= 4'b0000;
    else if (L == 0)       A_count <= {D, C, B, A};
    else if ((P == 1) && (T == 1))  A_count <= A_count + 1'b1;
    else A_count <= A_count; // redundant statement
endmodule

module t_Counter_74161 ();
  wire        QD, QC, QB, QA;
  wire [3: 0] Data_outputs = {QD, QC, QB, QA};
  wire        Carry_out;        // Output carry
  reg [3:0]   Data_inputs;      // Data input
  reg         Count,            // Active high to count
              Load,             // Active low to load
              Clock,            // Positive edge sensitive
              Clear;            // Active low to clear


Counter_74161 M0 (QD, QC, QB, QA, Carry_out,
  Data_inputs[3], Data_inputs[2], Data_inputs[1], Data_inputs[0], Count, Count, Load, Clock, Clear);

  initial #200 $finish;
  initial begin Clock = 0; forever #5 Clock = ~Clock; end

  initial fork
    Clear = 0;
    Load = 1;
    Count = 0;
    #20 Clear = 1;
    #40 Load = 0;
    #50 Load = 1;
    #80 Count = 1;
    #180 Count = 0;
    Data_inputs = 4'ha;        // 10
  join
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

Supplement to Experiment #11.

(a)

```
// Note: J and K_bar are assumed to be connected together.
module SReg_74195 (
  output reg QA, QB, QC, QD,
  output QD_bar,
  input A, B, C, D, SH_LD, J, K_bar, CLR_bar, CK
);
  assign QD_bar = ~QD;

  always @ (posedge CK, negedge CLR_bar)
    if (!CLR_bar) {QA, QB, QC, QD} <= 4'b0;
    else if (!SH_LD) {QA, QB, QC, QD} <= {A, B, C, D};
    else case ({J, K_bar})
      2'b00:  {QA, QB, QC, QD} <= {1'b0, QA, QB, QC};
      2'b11:  {QA, QB, QC, QD} <= {1'b1, QA, QB, QC};
      2'b01:  {QA, QB, QC, QD} <= {QA, QA, QB, QC};   // unused
      2'b10:  {QA, QB, QC, QD} <= {~QA, QA, QB, QC}; // unused
    endcase
endmodule

module t_SReg_74195 ();
  wire QA, QB, QC, QD;
  wire QD_bar;
  reg A, B, C, D, SH_LD, CLR_bar, CK;
  reg    Serial_Input;
  wire J = Serial_Input;
  wire K_bar = Serial_Input;
  wire [3: 0] Data_inputs = {A, B, C, D};
  wire [3: 0] Data_outputs = {QA, QB, QC, QD};

  SReg_74195 M0 (QA, QB, QC, QD, QD_bar, A, B, C, D, SH_LD, J, K_bar, CLR_bar, CK);

  initial #200 $finish;
  initial begin CK = 0; forever #5 CK = ~CK; end
  initial fork
    {A, B, C, D} = 4'ha;
    CLR_bar = 0;
    Serial_Input = 0;
    SH_LD = 0;
    #30 CLR_bar = 1;
    #60 SH_LD = 1;
    #120 Serial_Input = 1;
  join
endmodule
```


All rights reserved.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

(b)

```
module Mux_74157 (
  output reg Y1, Y2, Y3, Y4,
  input A1, A2, A3, A4, B1, B2, B3, B4, SEL, STB
);
  wire [4: 1] In_A = {A1, A2, A3, A4};
  wire [4: 1] In_B = {B1, B2, B3, B4};


  always @ (In_A, In_B, SEL, STB)
    if (STB) {Y1, Y2, Y3, Y4} = 4'b0;
    else if (SEL) {Y1, Y2, Y3, Y4} = In_B;
    else {Y1, Y2, Y3, Y4} = In_A;
endmodule


module t_Mux_74157 ();
  wire Y1, Y2, Y3, Y4;
  reg A1, A2, A3, A4, B1, B2, B3, B4, SEL, STB;
  wire [4: 1] In_A = {A1, A2, A3, A4};
  wire [4: 1] In_B = {B1, B2, B3, B4};
  wire [4: 1] Y = {Y1, Y2, Y3, Y4};

  Mux_74157 M0 (Y1, Y2, Y3, Y4, A1, A2, A3, A4, B1, B2, B3, B4, SEL, STB);

  initial #200 $finish;
  initial fork
    {A1, A2, A3, A4} = 4'ha;
    {B1, B2, B3, B4} = 4'hb;
    STB = 1;
    SEL = 1;
    #50 STB = 0;
    #100 SEL = 0;
    #150 STB = 1;
  join
endmodule
```

(c)

```
module Bi_Dir_Shift_Reg (output [1: 4] D_out, input [1: 4] D_in, input SEL, STB, SH_LD, clock,
CLR_bar);
  wire          QD_bar;
  wire [1: 4]   Y;
  SReg_74195 M0 (D_out[1], D_out[2], D_out[3], D_out[4], QD_bar, Y[1], Y[2], Y[3], Y[4],
    SH_LD, D_out[4], D_out[4], CLR_bar, clock
  );
  Mux_74157 M1 (Y[1], Y[2], Y[3], Y[4], D_in[1], D_in[2], D_in[3], D_in[4],
```


All rights reserved.

```
    D_out[2], D_out[3], D_out[4], D_out[1], SEL, STB
  );
endmodule

module SReg_74195 (
  output reg   QA, QB, QC, QD,
  output       QD_bar,
  input        A, B, C, D, SH_LD, J, K_bar, CLR_bar, CK
);
  assign       QD_bar = ~QD;
  always @ (posedge CK, negedge CLR_bar)
    if (!CLR_bar) {QA, QB, QC, QD} <= 4'b0;
    else if (!SH_LD) {QA, QB, QC, QD} <= {A, B, C, D};
    else case ({J, K_bar})
      2'b00:  {QA, QB, QC, QD} <= {1'b0, QA, QB, QC};
      2'b11:  {QA, QB, QC, QD} <= {1'b1, QA, QB, QC};
      2'b01:  {QA, QB, QC, QD} <= {QA, QA, QB, QC};   // unused
      2'b10:  {QA, QB, QC, QD} <= {~QA, QA, QB, QC}; // unused
    endcase
endmodule

module Mux_74157 (
  output reg Y1, Y2, Y3, Y4,
  input A1, A2, A3, A4, B1, B2, B3, B4, SEL, STB
);
  wire [4: 1] In_A = {A1, A2, A3, A4};
  wire [4: 1] In_B = {B1, B2, B3, B4};

  always @ (In_A, In_B, SEL, STB)
    if (STB) {Y1, Y2, Y3, Y4} = 4'b0;
    else if (SEL) {Y1, Y2, Y3, Y4} = In_B;      // SEL = 1
    else {Y1, Y2, Y3, Y4} = In_A;               // SEL = 0
endmodule

module t_Bi_Dir_Shift_Reg ();
  wire [1: 4] D_out;
  reg [1: 4] D_in;
  reg SEL, STB, SH_LD, clock, CLR_bar;
  Bi_Dir_Shift_Reg M0 (D_out, D_in, SEL, STB, SH_LD, clock, CLR_bar);
  initial #200 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial fork
    D_in = 4'h8;          // Data for walking 1 to right
    CLR_bar = 0;
    STB = 0;
    SEL = 0;              // Selects D_in
    SH_LD = 0;            // load D_in
    #10 CLR_bar = 1;
    #20 STB = 1;
    #40 STB = 0;
    #30 SH_LD = 1;
    #50 SH_LD = 0;     // Interrupt count to load
    #60 SH_LD = 1;
    #80 SEL = 1;
    #100 STB = 1;
    #130 STB = 0;
    #140 SH_LD = 0;
    //#150 SH_LD = 1;
  join
endmodule
```


All rights reserved.

The behavioral model is listed below. The two models have matching simulation results.

```
module Bi_Dir_Shift_Reg_beh (output reg [1: 4] D_out, input [1: 4] D_in, input SEL, STB, SH_LD, clock,
CLR_bar);
  always @ (posedge clock, negedge CLR_bar)
    if (!CLR_bar) D_out <= 4'b0;
    else if (SH_LD ) D_out <= {D_out[4], D_out[1], D_out[2], D_out[3]};
   else if (!STB) D_out <= SEL ? {D_out[2: 4], D_out[1]}: D_in;
   else D_out <= 4'b0;
 endmodule

module t_Bi_Dir_Shift_Reg_beh ();
  wire [1: 4] D_out;
  reg [1: 4] D_in;
  reg SEL, STB, SH_LD, clock, CLR_bar;

  Bi_Dir_Shift_Reg_beh M0 (D_out, D_in, SEL, STB, SH_LD, clock, CLR_bar);

  initial #200 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial fork
    D_in = 4'h8;          // Data for walking 1 to right
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
    CLR_bar = 0;
    STB = 0;
    SEL = 0;          // Selects D_in
    SH_LD = 0;      // load D_in
    #10 CLR_bar = 1;
    #20 STB = 1;
    #40 STB = 0;
    #30 SH_LD = 1;
    #50 SH_LD = 0;     // Interrupt count to load
    #60 SH_LD = 1;
    #80 SEL = 1;
    #100 STB = 1;
    #130 STB = 0;
    #140 SH_LD = 0;
    //#150 SH_LD = 1;
  join
endmodule
```

Supplement to Experiment #13.

```
module RAM_74189 (output S4, S3, S2, S1, input D4, D3, D2, D1, A3, A2, A1, A0, CS, WE);
// Note: active-low CS and WE
  wire [3: 0]                address = {A3, A2, A1, A0};
  reg [3: 0]                 RAM [0: 15];              // 16 x 4 memory
  wire [4: 1]                Data_in = { D4, D3, D2, D1}; // Input word
  tri [4: 1]                 Data;                     // Output data word, three-state output
  assign S1 = Data[1];               // Output bits
  assign S2 = Data[2];
  assign S3 = Data[3];
  assign S4 = Data[4];

  always @ (Data_in, address, CS, WE) if (~CS && ~WE) RAM[address] = Data_in;
  assign Data = (~CS && WE) ? ~RAM[address] : 4'bz;
endmodule

module t_RAM_74189 ();
  reg [4: 1] Data_in;
  reg [3: 0] address;
  reg CS, WE;
  wire S1, S2, S3, S4;
  wire D1, D2, D3, D4;
  wire A0, A1, A2, A3;
  wire [4: 1] Data_out = {S4, S3, S2, S1};
  assign  D1 = Data_in [1];
  assign  D2 = Data_in [2];
  assign  D3 = Data_in [3];
  assign  D4 = Data_in [4];
  assign A0 = address[0];
  assign A1 = address[1];
  assign A2 = address[2];
  assign A3 = address[3];

  wire [3: 0] RAM_0 = M0.RAM[0];
  wire [3: 0] RAM_1 = M0.RAM[1];
  wire [3: 0] RAM_2 = M0.RAM[2];
  wire [3: 0] RAM_3 = M0.RAM[3];
  wire [3: 0] RAM_4 = M0.RAM[4];
  wire [3: 0] RAM_5 = M0.RAM[5];
  wire [3: 0] RAM_6 = M0.RAM[6];
  wire [3: 0] RAM_7 = M0.RAM[7];
  wire [3: 0] RAM_8 = M0.RAM[8];
  wire [3: 0] RAM_9 = M0.RAM[9];
```


All rights reserved.

```
  wire [3: 0] RAM_10 = M0.RAM[10];
  wire [3: 0] RAM_11 = M0.RAM[11];
  wire [3: 0] RAM_12= M0.RAM[12];
  wire [3: 0] RAM_13 = M0.RAM[13];
  wire [3: 0] RAM_14 = M0.RAM[14];
  wire [3: 0] RAM_15 = M0.RAM[15];
  wire [4: 1] word = ~Data_out;

  RAM_74189 M0 (S4, S3, S2, S1, D4, D3, D2, D1, A3, A2, A1, A0, CS, WE);

  initial #110 $finish;
  initial fork
    WE = 1;
    CS = 1;
    address = 0;
    Data_in = 3;
    #10 CS = 0;
    #15 WE = 0;
    #20 WE = 1;
    #25 address = 14;
    #25 Data_in = 1;
    #30 WE = 0;
    #35 WE = 1;
    #40 CS = 1;
    #50 address = 0;
    #60 CS = 0;
    #70 CS = 1;
    #80 address = 14;
    #90 CS = 0;
  join
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

Note: Data_out is the complement of the stored value

Supplement to Experiment #14.

```
module Bi_Dir_Shift_Reg_74194 (
  output reg   QA, QB, QC, QD,
  input        A, B, C, D, SIR, SIL, s1, s0, CK, CLR
);
  always @ (posedge CK, negedge CLR)
    if (!CLR) {QA, QB, QC, QD} <= 4'b0;
    else case ({s1, s0})
      2'b00:  {QA, QB, QC, QD} <= {QA, QB, QC, QD};
      2'b01:  {QA, QB, QC, QD} <= {SIR, QA, QB, QC};
      2'b10:  {QA, QB, QC, QD} <= {QB, QC, QD, SIL};
      2'b11:  {QA, QB, QC, QD} <= {A, B, C, D};
    endcase
endmodule

module t_Bi_Dir_Shift_Reg_74194 ();
  wire QA, QB, QC, QD;
  reg A, B, C, D, SIR, SIL, s1, s0, clock, CLR;

  Bi_Dir_Shift_Reg_74194 M0 (QA, QB, QC, QD, A, B, C, D, SIR, SIL, s1, s0, clock, CLR);

  initial #250 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
```


All rights reserved.

```
 initial fork
  CLR = 0;
  {A, B, C, D} = 4'hf;
  s1 = 0;
  s0 = 0;
  SIL = 0;
  SIR = 0;
  #10 CLR = 1;
  #30 begin s1 = 1; s0 = 1; end// load
  #40   s1 = 0;     // shift right
  #100   s1 = 1;   // load
  #110 begin s1 = 0; s0 = 0; end
  #140 s1 = 1;     // shift left
  #160 s1 = 0;     // pause
  #180 s1 = 1;     // resume
 join
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

Supplement to Experiment #16.

The HDL behavioral descriptions of the components in the block diagram of Fig. 9.23 are described in the solutions of previous experiments, along with their test benches and simulations results: 74189 is described in Experiment 13(a); 74157 in Experiment 11(b); 74161 in Experiment 10; 7483 in Experiment 7(a); 74194 in Experiment 14; and 7474 in Experiment 8(a). The structural description of the parallel adder instantiates these components to show how they are interconnected (see the solution to the supplement for Experiment 17 for a similar procedure). A test bench and simulation results for the integrated unit are given below.

```
// LOAD condition for 74194: s1 = 1, s0 = 1
// SHIFT condition: s1 = 0, s0 = 1
// NO CHANGE condition: s1 = 0, s0 = 0

module Supp_9_16 (
  output [3: 0] accum_sum,
  output carry,
  input [3: 0] Data_in, Addr_in,
  input SIR, SIL, CS, WE, s1, s0, count, Load, select, STB, clock, preset, clear, VCC, GND
);
  wire B4 = Data_in[3];    // Data world to memory
  wire B3 = Data_in[2];
  wire B2 = Data_in[1];
  wire B1 = Data_in[0];
  wire S4, S3, S2, S1;
  wire D4, D3, D2, D1;
  wire S4b = ~S4; // Inverters
  wire S3b = ~S3;
  wire S2b = ~S2;
  wire S1b = ~S1;
  wire D = Addr_in[3];// For parallel load of address counter
  wire C = Addr_in[2];
  wire B = Addr_in[1];
  wire A = Addr_in[0];
  wire Ocar, Y1, Y2, Y3, Y4, QA, QB, QC, QD, A3, A2, A1, A0;
  assign accum_sum = {D4, D3, D2, D1};

  Flip_flop_7474   M0 (Ocar, carry, clock, preset, clear);
  Adder_7483 M1 (D4, D3, D2, D1, carry, S4b, S3b, S2b, S1b, QD, QC, QB, QA, Ocar, VCC, GND);
  Mux_74157  M2 (Y4, Y3, Y2, Y1, QD, QC, QB, QA, B4, B3, B2, B1, select, STB);
  Counter_74161  M3 (A3, A2, A1, A0, COUT, D, C, B, A, count, count, Load, clock, clear);
  RAM_74189 M4 (S4, S3, S2, S1, Y4, Y3, Y2, Y1, A3, A2, A1, A0, CS, WE);
  Reg_74194   M5 (QD, QC, QB, QA, D4, D3, D2, D1, Ocar, SIL, s1, s0, clock, clear);
endmodule

module t_Supp_9_16 ();
  wire [3: 0] sum;
  wire carry;
  reg [3: 0] Data_in, Addr_in;
  reg SIR, SIL, CS, WE, s1, s0, count, Load, select, STB, clock, preset, clear;
  supply1 VCC;
  supply0 GND;
  wire [3: 0] RAM_0 = M0.M4.RAM[0];
  wire [3: 0] RAM_1 = M0.M4.RAM[1];
  wire [3: 0] RAM_2 = M0.M4.RAM[2];
  wire [3: 0] RAM_3 = M0.M4.RAM[3];
  wire [3: 0] RAM_4 = M0.M4.RAM[4];
  wire [3: 0] RAM_5 = M0.M4.RAM[5];
  wire [3: 0] RAM_6 = M0.M4.RAM[6];
  wire [3: 0] RAM_7 = M0.M4.RAM[7];
  wire [3: 0] RAM_8 = M0.M4.RAM[8];
```


All rights reserved.

```
 wire [3: 0] RAM_9 = M0.M4.RAM[9];
 wire [3: 0] RAM_10 = M0.M4.RAM[10];
 wire [3: 0] RAM_11 = M0.M4.RAM[11];
 wire [3: 0] RAM_12= M0.M4.RAM[12];
 wire [3: 0] RAM_13 = M0.M4.RAM[13];
 wire [3: 0] RAM_14 = M0.M4.RAM[14];
 wire [3: 0] RAM_15 = M0.M4.RAM[15];

 wire [4: 1] word = {M0.S4b, M0.S3b,M0.S2b, M0.S1b};
wire [4: 1] mux_out = { M0.Y4, M0.Y3, M0.Y2, M0.Y1};
wire [4: 1] Reg_Output = {M0.QD, M0.QC, M0.QB, M0.QA};

Supp_9_16 M0 (sum, carry, Data_in, Addr_in, SIR, SIL, CS, WE, s1, s0, count, Load,
 select, STB,   clock, preset, clear, VCC, GND);

 integer k;
 initial #600 $finish;
 initial begin clock = 0; forever #5 clock = ~clock; end
 initial fork
  #10 begin preset = 1; clear = 0; s1 = 0; s0 = 0; Load = 1; count = 0; CS = 1; WE = 1; STB = 0;  end
  // initialize memory
  #10 begin k = 0; repeat (16) begin M0.M4.RAM[k] = 4'hf; k = k + 1; end end
  #20 begin Data_in = 4'hf; Addr_in  = 0; select = 1; end
  #30 begin clear = 1; WE = 0; end
    // load memory
  #40 begin
    count = 1;
    CS = 0;
    begin
      repeat (16) @ (negedge clock) Data_in = Data_in + 1;
      count = 0;
      @ (negedge clock) CS = 1;
    end
  end
  #200 count = 1;                    // Establish address
  #240 count = 0;
  #250 WE = 1;
  #260 CS = 0;                       // Read from memory
  #280 clear = 0;
  #290 clear = 1;
  #300 count = 1;                    // Establish address
  #340 begin s1 = 1; s0 = 1; count = 0; end
  #390 CS = 0;
  #400 clear = 0;                 // Clear the registers
  #410 clear = 1;
  #420 begin count = 1; CS = 0; end        // Accumulate values
  #490 begin count = 0; CS = 1; end
 join
endmodule

module Flip_flop_7474 (output reg Q, input D, CLK, preset, clear);
  always @ (posedge CLK, negedge preset , negedge clear)
  if (!preset)                    Q <= 1'b1;
  else if (!clear)                Q <= 1'b0;
  else                            Q <= D;
endmodule

module Adder_7483 (
  output S4, S3, S2, S1, C4,
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
  input A4, A3, A2, A1, B4, B3, B2, B1, C0, VCC, GND
);
// Note: connect VCC and GND to supply1 and supply0 in the test bench
  wire [4: 1] sum;
  wire [4: 1] A = {A4, A3, A2, A1};
  wire [4: 1] B = {B4, B3, B2, B1};
  assign S4 = sum[4];
  assign S3 = sum[3];
  assign S2 = sum[2];
  assign S1 = sum[1];
  assign {C4, sum} = A + B + C0;
endmodule

module Mux_74157 (
  output reg Y1, Y2, Y3, Y4,
  input A1, A2, A3, A4, B1, B2, B3, B4, SEL, STB
);
  wire [4: 1] In_A = {A1, A2, A3, A4};
  wire [4: 1] In_B = {B1, B2, B3, B4};

  always @ (In_A, In_B, SEL, STB)
    if (STB) {Y1, Y2, Y3, Y4} = 4'b0;
    else if (SEL) {Y1, Y2, Y3, Y4} = In_B;
    else {Y1, Y2, Y3, Y4} = In_A;
endmodule

module Counter_74161 (
  output   QD, QC, QB, QA,      // Data output
  output   COUT,            // Output carry
  input    D, C, B, A,      // Data input
  input    P, T,            // Active high to count
           L,            // Active low to load
           CK,              // Positive edge sensitive
           CLR              // Active low to clear
);
reg [3: 0] A_count;
assign QD = A_count[3];
assign QC = A_count[2];
assign QB = A_count[1];
assign QA = A_count[0];
assign COUT = ((P == 1) && (T == 1) && (L == 1) && (A_count == 4'b1111));
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
always @ (posedge CK, negedge CLR)
if (CLR == 0)                    A_count <= 4'b0000;
else if (L == 0)                 A_count <= {D, C, B, A};
else if ((P == 1) && (T == 1))   A_count <= A_count + 1'b1;
else                             A_count <= A_count; // redundant statement
endmodule

module RAM_74189 (output S4, S3, S2, S1, input D4, D3, D2, D1, A3, A2, A1, A0, CS, WE);
// Note: active-low CS and WE
  wire [3: 0]    address = {A3, A2, A1, A0};
  reg [3: 0]     RAM [0: 15];          // 16 x 4 memory
  wire [4: 1]    Data_in = { D4, D3, D2, D1}; // Input word
  tri [4: 1]     Data;                 // Output data word, three-state output
  assign S1 = Data[1];                 // Output bits
  assign S2 = Data[2];
  assign S3 = Data[3];
  assign S4 = Data[4];

  always @ (Data_in, address, CS, WE) if (~CS && ~WE) RAM[address] = Data_in;
  assign Data = (~CS && WE) ? ~RAM[address] : 4'bz;    // Note complement of data word
endmodule

module Reg_74194 (
  output reg   QA, QB, QC, QD,
  input        A, B, C, D, SIR, SIL, s1, s0, CK, CLR
);
  always @ (posedge CK, negedge CLR)
    if (!CLR) {QA, QB, QC, QD} <= 4'b0;
    else case ({s1, s0})
      2'b00: {QA, QB, QC, QD} <= {QA, QB, QC, QD};
      2'b01: {QA, QB, QC, QD} <= {SIR, QA, QB, QC};
      2'b10: {QA, QB, QC, QD} <= {QB, QC, QD, SIL};
      2'b11: {QA, QB, QC, QD} <= {A, B, C, D};
    endcase
endmodule
```

Simulation results: initializing memory to 4'hf, then writing to memory. Note: the values of the inputs are ambiguous until the clear signal is asserted. Signals Ocar and carry are ambiguous because the output of memory is high-z until memory is read is read.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

Supplement to Experiment #17.

The HDL behavioral descriptions of the components in the block diagram of Fig. 9.23 are described in the solutions of previous experiments, along with their test benches and simulations results: 74161 in Experiment 10; 7483 in Experiment 7(a); 74194 in Experiment 14; and 7474 in Experiment 8(a). The structural description of the parallel adder instantiates these components to show how they are interconnected (see the solution to the supplement for Experiment 17 for a similar procedure). A test bench and simulation results for the integrated unit are given below.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

```
// Control unit is obtained by modifying the solution to Prob. 8.24.
// Datapath is implemented with a structural  HDL model and IC components.
// LOAD condition for 74194: s1 = 1, s0 = 1
// SHIFT condition: s1 = 0, s0 = 1
// NO CHANGE condition: s1 = 0, s0 = 0

module Supp_9_17_Par_Mult # (parameter dp_width = 4)
(
  output    [2*dp_width - 1: 0]     Product,
  output                            Ready,
  input     [dp_width - 1: 0]           Multiplicand, Multiplier,
  input                                 Start, clock, reset_b, VCC, GND
);
  wire Load_regs, Incr_P, Add_regs, Shift_regs, Done, Q0;

  Controller M0 (
    Ready, Load_regs, Incr_P, Add_regs, Shift_regs, Start, Done, Q0,
    clock, reset_b);

  Datapath M1(Product, Q0, Done, Multiplicand, Multiplier,
  Start, Load_regs, Incr_P, Add_regs, Shift_regs, clock, reset_b, VCC, GND);
endmodule

module Controller (
  output Ready,
  output reg Load_regs, Incr_P, Add_regs, Shift_regs,
  input Start, Done, Q0, clock, reset_b
);

  parameter   S_idle =    3'b001,          // one-hot code
              S_add = 3'b010,
              S_shift = 3'b100;
  reg    [2: 0]   state, next_state;      // sized for one-hot
  assign          Ready = (state == S_idle);

  always @ (posedge clock, negedge reset_b)
    if (~reset_b) state <= S_idle; else state <= next_state;
  always @ (state, Start, Q0, Done) begin
    next_state = S_idle;
    Load_regs = 0;
    Incr_P = 0;
    Add_regs = 0;
    Shift_regs = 0;
    case (state)
      S_idle: if (Start) begin next_state = S_add; Load_regs = 1; end
      S_add:     begin next_state = S_shift; Incr_P = 1; if (Q0) Add_regs = 1; end
      S_shift:   begin
                   Shift_regs = 1;
                   if (Done) next_state = S_idle;
                   else next_state = S_add;
                 end
      default:   next_state = S_idle;
    endcase
  end
endmodule

module Datapath #(parameter dp_width = 4, BC_size = 3) (
  output [2*dp_width - 1: 0] Product, output Q0, output Done,
  input [dp_width - 1: 0] Multiplicand, Multiplier,
  input Start, Load_regs, Incr_P, Add_regs, Shift_regs, clock, clear, VCC, GND
);
  wire C;
```


All rights reserved.

```
  wire Cout, Sum3, Sum2, Sum1, Sum0, P3, P2, P1, P0, A3, A2, A1, A0;
  wire Q3, Q2, Q1;
  wire [dp_width -1: 0] A = {A3, A2, A1, A0};
  wire [dp_width -1: 0] Q = {Q3, Q2, Q1, Q0};
  assign Product = {C, A, Q};
   wire [ BC_size -1: 0] P = {P3, P2, P1, P0};

// Registers must be controlled separately to execute add and shift operations correctly.
// LOAD condition for 74194: s1 = 1, s0 = 1
// SHIFT condition: s1 = 0, s0 = 1
// NO CHANGE condition: s1 = 0, s0 = 0

  wire B3 = Multiplicand[3];        // Data word to adder
  wire B2 = Multiplicand[2];
  wire B1 = Multiplicand[1];
  wire B0 = Multiplicand[0];
  wire Q3_in = Multiplier[3];
  wire Q2_in = Multiplier[2];
  wire Q1_in = Multiplier[1];
  wire Q0_in = Multiplier[0];
  assign Done = ({P3, P2, P1, P0} == dp_width);        // Counts bits of multiplier
  wire s1A = Load_regs || Add_regs;                // Controls for A register
  wire s0A = Load_regs || Add_regs || Shift_regs;
  wire s0Q = Load_regs || Shift_regs;              // Controls for Q register
  wire s1Q = Load_regs;
   wire Pout;                                      // Unused
  wire clr_P = clear && ~Load_regs;
  Flip_flop_7474   M0_C (C, Cout, clock, VCC, clr_P);
  Adder_7483 M1 (Sum3, Sum2, Sum1, Sum0, Cout, A3, A2, A1, A0, B3, B2, B1, B0, GND, VCC, GND);
  Counter_74161  M3_P (P3, P2, P1, P0, Pout, GND, GND, GND, GND, Incr_P, Incr_P, VCC, clock,
clr_P);
  Reg_74194   M4_A (A3, A2, A1, A0, Sum3, Sum2, Sum1, Sum0, C, GND, s1A, s0A, clock, clr_P);
  Reg_74194   M5_Q (Q3, Q2, Q1, Q0, Q3_in, Q2_in, Q1_in, Q0_in, A0, GND, s1Q, s0Q, clock, clear);

endmodule

module t_Supp_9_17_Par_Mult;
  parameter          dp_width = 4;          // Width of datapath
  wire  [2 * dp_width - 1: 0]   Product;
  wire               Ready;
  reg   [dp_width - 1: 0]       Multiplicand, Multiplier;
  reg                Start, clock, reset_b;
  integer                Exp_Value;
  reg                Error;
  supply0            GND;
  supply1            VCC;
  Supp_11_17_Par_Mult M0 (Product, Ready, Multiplicand, Multiplier, Start, clock, reset_b, VCC, GND);
  wire [dp_width -1: 0] sum = {M0.M1.Sum3, M0.M1.Sum2, M0.M1.Sum1, M0.M1.Sum0};
  initial #115000 $finish;
  initial begin clock = 0; #5 forever #5 clock = ~clock; end
  initial fork
    reset_b = 1;
    #2 reset_b = 0;
    #3 reset_b = 1;
  join
  always @ (negedge Start) begin
    Exp_Value = Multiplier * Multiplicand;
    //Exp_Value = Multiplier * Multiplicand +1;    // Inject error to confirm detection
  end
```


All rights reserved.

```
always @ (posedge Ready) begin
   # 1 Error <= (Exp_Value ^ Product) ;
  end
  initial begin
   #5 Multiplicand = 0;
   Multiplier = 0;
   repeat (32) #10 begin
      Start = 1;
       #10 Start = 0;
       repeat (32) begin
         Start = 1;
         #10 Start = 0;
         #100 Multiplicand = Multiplicand + 1;
       end
     Multiplier = Multiplier + 1;
   end
  end
endmodule

module Flip_flop_7474 (output reg Q, input D, CLK, preset, clear);
  always @ (posedge CLK, negedge preset , negedge clear)
  if (!preset)                Q <= 1'b1;
  else if (!clear)            Q <= 1'b0;
  else                        Q <= D;
endmodule

module Adder_7483 (
  output S4, S3, S2, S1, C4,
  input A4, A3, A2, A1, B4, B3, B2, B1, C0, VCC, GND
);
// Note: connect VCC and GND to supply1 and supply0 in the test bench
  wire [4: 1] sum;
  wire [4: 1] A = {A4, A3, A2, A1};
  wire [4: 1] B = {B4, B3, B2, B1};
  assign S4 = sum[4];
  assign S3 = sum[3];
  assign S2 = sum[2];
  assign S1 = sum[1];
  assign {C4, sum} = A + B + C0;
endmodule

module Counter_74161 (
  output    QD, QC, QB, QA,      // Data output
  output        COUT,            // Output carry
  input     D, C, B, A,      // Data input
  input         P, T,                // Active high to count
         L,              // Active low to load
         CK,                 // Positive edge sensitive
         CLR                 // Active low to clear
);
reg [3: 0] A_count;
assign QD = A_count[3];
assign QC = A_count[2];
assign QB = A_count[1];
assign QA = A_count[0];
assign COUT = ((P == 1) && (T == 1) && (L == 1) && (A_count == 4'b1111));
always @ (posedge CK, negedge CLR)
```


All rights reserved.

```
if (CLR == 0)            A_count <= 4'b0000;
else if (L == 0)              A_count <= {D, C, B, A};
else if ((P == 1) && (T == 1))     A_count <= A_count + 1'b1;
else                  A_count <= A_count; // redundant statement
endmodule

module Reg_74194 (
  output reg QA, QB, QC, QD,
  input A, B, C, D, SIR, SIL, s1, s0, CK, CLR
);
  always @ (posedge CK, negedge CLR)
    if (!CLR) {QA, QB, QC, QD} <= 4'b0;
    else case ({s1, s0})
      2'b00:  {QA, QB, QC, QD} <= {QA, QB, QC, QD};
      2'b01:  {QA, QB, QC, QD} <= {SIR, QA, QB, QC};
      2'b10:  {QA, QB, QC, QD} <= {QB, QC, QD, SIL};
      2'b11:  {QA, QB, QC, QD} <= {A, B, C, D};
    endcase
endmodule
```

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

CHAPTER 10

10.1

10.2 See textbook.

10.3

10.4 BCD-to-decimal decoder (similar to IC 7442)

10.5 Similar to 7438:

10.6 IC type 74153.

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.

10.7

10.8 The common control block is used when the circuit has one or more inputs that are common to all lower sections.

10.9

10.10 See textbook.

10.11

10.12

Digital Design With An Introduction to the Verilog HDL – Solution Manual. M. Mano. M.D. Ciletti, Copyright 2012,
All rights reserved.