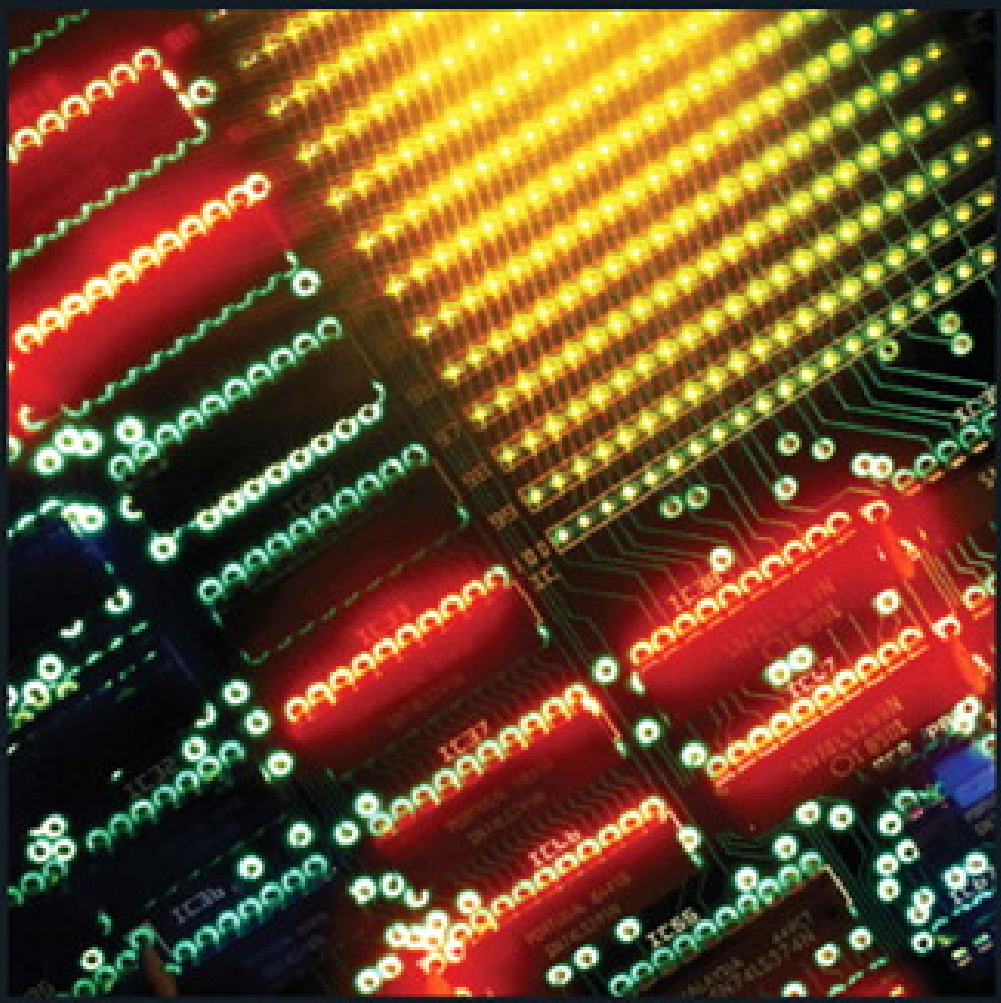
TENTH EDITION

# DIGITAL SYSTEMS

## PRINCIPLES AND APPLICATIONS



**INSTRUCTOR'S SOLUTION MANUAL**

RONALD J. TOCCI
NEAL S. WIDMER
GREGORY L. MOSS

Instructor's Resource Manual
*to accompany*

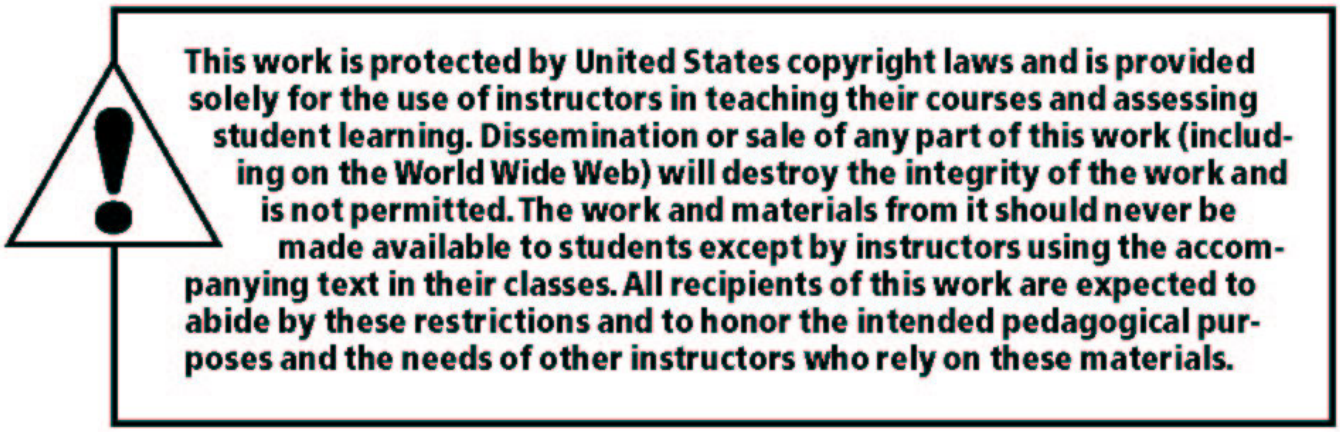
# DIGITAL SYSTEMS
## Principles and Applications

## Tenth Edition

Ronald J. Tocci
Neal S. Widmer
Gregory L. Moss

*Prepared by*
Frank J. Ambrosio
Monroe Community College


Upper Saddle River, New Jersey
Columbus, Ohio






10 9 8 7 6 5 4 3 2 1

PEARSON
Prentice
Hall

ISBN 0-13-172665-X

# TABLE OF CONTENTS

**SOLUTIONS TO:**

**RONALD J. TOCCI/NEAL S. WIDMER/GREGORY L. MOSS**

## DIGITAL SYSTEMS: PRINCIPLES AND APPLICATIONS

**TENTH EDITION**

## CHAPTER ONE - Introductory Concepts

**1.1** (a), (e) are digital.
(b), (c), (d) are analog.

**1.2** (a) Analog
(b) Analog
(c) Digital
(d) Digital
(e) Analog

**1.3** (a) $11001_2 = 16+8+1 = \mathbf{25_{10}}$
(b) $1001.1001_2 = 8+1+0.5+0.0625 = \mathbf{9.5625_{10}}$
(c) $10011011001.10110_2 = 1024+128+64+16+8+1+0.5+0.125+0.0625 = \mathbf{1241.6875_{10}}$

**1.4** (a) $10011_2 = 16+2+1 = \mathbf{19_{10}}$
(b) $1100.0101_2 = 8+4+0.25+0.0625 = \mathbf{12.3125_{10}}$
(c) $10011100100.10010_2 = \mathbf{1252.5625_{10}}$

**1.5** $000_2, 001_2, 010_2, 011_2, 100_2, 101_2, 110_2, 111_2$

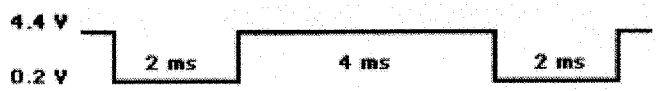
**1.7** $2^N-1 = 2^{10}-1 = \mathbf{1023}$

**1.8** $2^N-1 = 2^{14}-1 = \mathbf{16,383}$

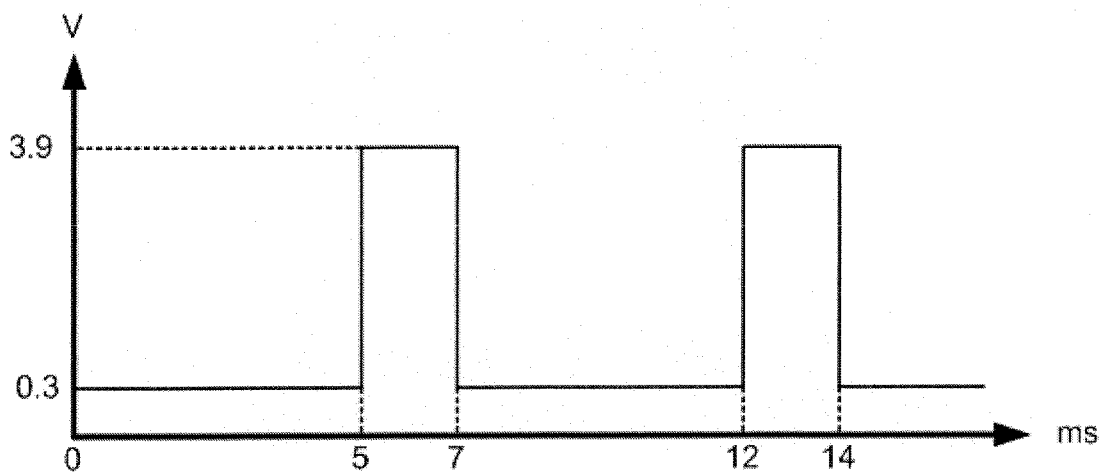
**1.9** $2^8 = 256$ and $2^9 = 512$, therefore ***9 bits are needed***.

**1.10** $2^N-1 = 63$, therefore ***6 bits are needed***.

**1.11**




**1.12**



**1.13** (a) $2^N-1 = 15$
N = 4; Therefore, ***4 lines*** are required for parallel transmission.
(b) Only ***1 line*** is required for serial transmission.

**1.14** A ***microprocessor*** is a CPU on a chip. The CPU contains the control unit and the arithmetic logic unit (ALU). A ***microcomputer*** generally consists of several IC chips including a microprocessor chip, memory chips, and input/output interface chips along with input/output devices.

**1.15** A ***microcontroller*** is a specialized type of microcomputer that is designed to be used as a dedicated or embedded controller. Microcontrollers are generally much smaller than general-purpose microcomputers.

## CHAPTER TWO - Number Systems and Codes

**2.1** (a) $10110_2 = 16+4+2 = \mathbf{22_{10}}$

(b) $10010101_2 = 128+16+4+1 = \mathbf{149_{10}}$

(c) $100100001001_2 = 2048+256+8+1 = \mathbf{2313_{10}}$

(d) $01101011_2 = 64+32+8+2+1 = \mathbf{107_{10}}$

(e) $11111111_2 = 128+64+32+16+8+4+2+1 = \mathbf{255_{10}}$

(f) $01101111_2 = 64+32+8+4+2+1 = \mathbf{111_{10}}$

(g) $1111010111_2 = 512+256+128+64+16+4+2+1 = \mathbf{983_{10}}$

(h) $11011111_2 = 128+64+16+8+4+2+1 = \mathbf{223_{10}}$

**2.2** (a) $37_{10} = 32+4+1 = \mathbf{100101_2}$

(b) $13_{10} = 8+4+1 = \mathbf{1101_2}$

(c) $189_{10} = 128+32+16+8+4+1 = \mathbf{10111101_2}$

(d) $1000_{10} = 512+256+128+64+32+8 = \mathbf{1111101000_2}$

(e) $77_{10} = 64+8+4+1 = \mathbf{1001101_2}$

(f) $390_{10} = 256+128+4+2 = \mathbf{110000110_2}$

(g) $205_{10} = 128+64+8+4+1 = \mathbf{11001101_2}$

(h) $2133_{10} = 2048+64+16+4+1 = \mathbf{100001010101_2}$

(i) $511_{10} = 256+128+64+32+16+8+4+2+1 = \mathbf{111111111_2}$

**2.3** $(2^8-1) = \mathbf{255_{10}}$ ; $(2^{16}-1) = \mathbf{65,535_{10}}$

**2.4** (a) $743_{16} = 7x16^2+4x16^1+3x16^0 = \mathbf{1859_{10}}$

(b) $36_{16} = 3x16^1+6x16^0 = \mathbf{54_{10}}$

(c) $37FD_{16} = 3x16^3+7x16^2+15x16^1+13x16^0 = \mathbf{14333_{10}}$

(d) $2000_{16} = 2x16^3 = \mathbf{8192_{10}}$

(e) $165_{16} = 1x16^2+6x16^1+5x16^0 = \mathbf{357_{10}}$

(f) $ABCD_{16} = 10x16^3+11x16^2+12x16^1+13x16^0 = \mathbf{43981_{10}}$

(g) $7FF_{16} = 7x16^2+15x16^1+15x16^0 = \mathbf{2047_{10}}$

(h) $1204_{16} = 1x16^3+2x16^2+4x16^0 = \mathbf{4612_{10}}$

**2.5** (a)
59/16 = 3 Remainder of 11}
3/16 = 0 Remainder of 3 } $59_{10} = \mathbf{3B_{16}}$

(b)
372/16 = 23 Remainder of 4 }
23/16 = 1 Remainder of 7 }
1/16 = 0 Remainder of 1 } $372_{10} = \mathbf{174_{16}}$

(c)
919/16 = 57 Remainder of 7 }
57/16 = 3 Remainder of 9 }
3/16 = 0 Remainder of 3 } $919_{10} = \mathbf{397_{16}}$

(d) 1024/16 = 64 Remainder of 0 }
64/16 = 4 Remainder of 0 }
4/16 = 0 Remainder of 4 } $1024_{10}$ = ***$400_{16}$***

(e) 771/16 = 48 Remainder of 3 }
48/16 = 3 Remainder of 0 }
3/16 = 0 Remainder of 3 } $771_{10}$ = ***$303_{16}$***

(f) 2313/16 = 144 Remainder of 9 }
144/16 = 9 Remainder of 0 }
9/16 = 0 Remainder of 9 } $2313_{10}$ = ***$909_{16}$***

(g) 65536/16 = 4096 Remainder of 0 }
4096/16 = 256 Remainder of 0 }
256/16 = 16 Remainder of 0 }
16/16 = 1 Remainder of 0 }
1/16 = 0 Remainder of 1 } $65{,}536_{10}$ = ***$10000_{16}$***

(h) 255/16 = 15 Remainder of 15 }
15/16 = 0 Remainder of 15 } $255_{10}$ = ***$FF_{16}$***

**2.6** (a) $743_{16}$ = ***$11101000011_2$*** (b) $36_{16}$ = ***$110110_2$***
(c) $37FD_{16}$ = ***$11011111111101_2$*** (d) $2000_{16}$ = ***$10000000000000_2$***
(e) $165_{16}$ = ***$101100101_2$*** (f) $ABCD_{16}$ = ***$1010101111001101_2$*** (g) $7FF_{16}$ = ***$011111111111_2$***
(h) $1204_{16}$ = ***$1001000000100_2$***

**2.7** (a) $10110_2$ =***$16_{16}$*** (b) $10010101_2$=***$95_{16}$*** (c) $100100001001_2$=***$909_{16}$*** (d) $01101011_2$=***$6B_{16}$***
(e) $11111111_2$=***$FF_{16}$*** (f) $01101111_2$=***$6F_{16}$*** (g) $1111010111_2$=***$3D7_{16}$*** (h) $11011111_2$=***$DF_{16}$***

**2.8** 195, 196, 197, 198, 199, 19A, 19B, 19C, 19D, 19E, 19F, 2A0, 2A1, 2A2,.... 2A9, 2AA, 2AB, 2AC, 2AD, 2AE, 2AF, 2B0.

**2.9** 2133/16 = 133 Remainder of 5 }
133/16 = 8 Remainder of 5 }
8/16 = 0 Remainder of 8 } $2133_{10}$=$855_{16}$=***$100001010101_2$***

**2.10** $16^N \geq 20{,}000$; Therefore, n=4

**2.11** (a) $92_{16} = 9x16^1+2x16^0$ = ***$146_{10}$*** (b) $1A6_{16} = 1x16^2+10x16^1+6x16^0$ = ***$422_{10}$***
(c) $37FD_{16} = 3x16^3+7x16^2+15x16^1+13x16^0$ = ***$14333_{10}$***
(d) $ABCD_{16} = 10x16^3+11x16^2+12x16^1+13x16^0$ = ***$43981_{10}$***
(e) $000F_{16} = 0x16^3+0x16^2+0x16^1+15x16^0$ = ***$15_{10}$***
(f) $55_{16} = 5x16^1+5x16^0$ = ***$85_{10}$*** (g) $2C0_{16} = 2x16^2+12x16^1+0$ = ***$704_{10}$***
(h) $7FF_{16} = 7x16^2+15x16^1+15x16^0$ = ***$2047_{10}$***

**2.12**

| | | | | |
|---|---|---|---|---|
| (a) | 75/16 | = 4 | Remainder of 11 [B] | } |
| | 4/16 | = 0 | Remainder of 4 | } $75_{10}$=***$4B_{16}$*** |
| (b) | 314/16 | = 19 | Remainder of 10 [A] | } |
| | 19/16 | = 1 | Remainder of 3 | } |
| | 1/16 | = 0 | Remainder of 1 | } $314_{10}$=***$13A_{16}$*** |
| (c) | 2048/16 | = 128 | Remainder of 0 | } |
| | 128/16 | = 8 | Remainder of 0 | } |
| | 8/16 | = 0 | Remainder of 8 | } $2048_{10}$=***$800_{16}$*** |
| (d) | 24/16 | = 1 | Remainder of 8 | |
| | 1/16 | = 0 | Remainder of 1 | } $24_{10}$=***$18_{16}$*** |
| (e) | 7245/16 | = 452 | Remainder of 13 [D] | } |
| | 452/16 | = 28 | Remainder of 4 | } |
| | 28/16 | = 1 | Remainder of 12 [C] | } |
| | 1/16 | = 0 | Remainder of 1 | } $7245_{10}$=***$1C4D_{16}$*** |
| (f) | 498/16 | = 31 | Remainder of 2 | } |
| | 31/16 | = 1 | Remainder of 15 [F] | } |
| | 1/16 | = 0 | Remainder of 1 | } $498_{10}$=***$1F2_{16}$*** |
| (g) | 25619/16 | = 1601 | Remainder of 3 | } |
| | 1601/16 | = 100 | Remainder of 1 | } |
| | 100/16 | = 6 | Remainder of 4 | } |
| | 6/16 | = 0 | Remainder of 6 | } $25619_{10}$=***$6413_{16}$*** |
| (h) | 4095/16 | = 255 | Remainder of 15 [F] | } |
| | 255/16 | = 15 | Remainder of 15 [F] | } |
| | 15/16 | = 0 | Remainder of 15 [F] | } $4095_{10}$=***$FFF_{16}$*** |

**2.13** (a) 9 (b) D (c) 8 (d) 0 (e) F (f) 2 (g) A (h) 9 (I) B (j) C (k) 3
(l) 4 (m) 1 (n) 5 (o) 7 (p) 6

**2.14** (a) 0110 (b) 0111 (c) 0101 (d) 0001 (e) 0100 (f) 0011 (g) 1100 (h) 1011
(I) 1001 (j) 1010 (k) 0010 (l) 1111 (m) 0000 (n) 1000 (o) 1101 (p) 1001

**2.15** (a) $00010110_2$=***$16_{16}$*** (b) $10010101_2$=***$95_{16}$*** (c) $100100001001_2$=***$909_{16}$*** (d) $01101011_2$=***$6B_{16}$***
(e) $11111111_2$=***$FF_{16}$*** (f) $01101111_2$=***$6F_{16}$*** (g) $001111010111_2$=***$3D7_{16}$*** (h) $11011111_2$=***$DF_{16}$***

**2.16** (a) $92_{16}$=***$10010010_2$*** (b) $1A6_{16}$=***$000110100110_2$*** (c) $37FD_{16}$=***$0011011111111101_2$***
(d) $ABCD_{16}$=***$1010101111001101_2$*** (e) $000F_{16}$=***$1111_2$*** (f) $55_{16}$=***$01010101_2$***
(d) $2C0_{16}$=***$001011000000_2$*** (e) $7FF_{16}$=***$011111111111_2$***

**2.17** $280_{16}$, $281_{16}$, $282_{16}$,..... $288_{16}$, $289_{16}$, $28A_{16}$, $28B_{16}$, $28C_{16}$, $28D_{16}$, $28E_{16}$, $28F_{16}$, $290_{16}$, $291_{16}$,..... $298_{16}$, $299_{16}$, $29A_{16}$, $29B_{16}$, $29C_{16}$, $29D_{16}$, $29E_{16}$, $29F_{16}$, $2A0_{16}$.

**2.18** With *four* hex digits we can represent a decimal number up to: $FFFF_{16} = (16^4-1) = 65{,}535_{10}$
With *five* hex digits we can represent a decimal number up to: $FFFFF_{16} = (16^5-1) = 1{,}048{,}575_{10}$
Therefore, we need ***five*** hex digits to represent decimal numbers up to 1 million.

**2.19** (a) $47_{10}$=***0100 0111***$_{BCD}$ (b) $962_{10}$=***1001 0110 0010***$_{BCD}$ (c) $187_{10}$=***0001 1000 0111***$_{BCD}$
(d) $6727_{10}$=***0110 0111 0010 0111***$_{BCD}$ (e) $13_{10}$=***0001 0011***$_{BCD}$
(f) $529_{10}$=***0101 0010 1001***$_{BCD}$ (g) $89{,}627_{10}$=***1000 1001 0110 0010 0111***$_{BCD}$
(h) $1024_{10}$=***0001 0000 0010 0100***$_{BCD}$

**2.20** (a) $(2^N-1)=999$. Therefore, N=10. Hence, it requires ***10 bits*** for straight binary.
(b) $999_{10}$ requires ***12 bits*** for BCD (4 bits per digit).

**2.21** (a) 1001 0111 0101 $0010_{BCD}$ = $9752_{10}$ (b) 0001 1000 $0100_{BCD}$ = $184_{10}$
(c) 0110 1001 $0101_{BCD}$ = $695_{10}$ (d) 0111 0111 0111 $0101_{BCD}$ = $7775_{10}$
(e) 0100 1001 $0010_{BCD}$ = $492_{10}$ (f) 0101 0101 $0101_{BCD}$ = $555_{10}$

**2.22** (a) 1 byte = 8 bits. Thus, 8 bytes = ***64 bits***
(b) 4 bytes = 32 bits. A hex digit requires four bits to be represented. Thus, the largest hex number that can be represented in four bytes is $FFFFFFFF_{16}$.
(c) The largest BCD-encoded decimal value that can be represented in three bytes is ***999,999***.

**2.23** (a) 0101 (b) 4 nibbles (c) 3 bytes

**2.24**

| **x = 3*y** | **Hex** | **Bin** | **With odd-parity** |
|---|---|---|---|
| x ----------> | 78 | = 111 1000 | **1**111 1000 = F8 |
| space -------> | 20 | = 010 0000 | **0**010 0000 = 20 |
| = ----------> | 3D | = 011 1101 | **0**011 1101 = 3D |
| space -----> | 20 | = 010 0000 | **0**010 0000 = 20 |
| 3 ----------> | 3 | = 011 0011 | **1**011 0011 = B3 |
| * ----------> | 2A | = 010 1010 | **0**010 1010 = 2A |
| y ----------> | 79 | = 111 1001 | **0**111 1001 = 79 |

***1111 1000*** **0010 0000** ***0011 1101*** **0010 0000** ***1011 0011*** **0010 1010** ***0111 1001***
x space = space 3 * y

**2.25**

| **x = 3*y** | **Hex** | **Bin** | **With even-parity** |
|---|---|---|---|
| x ----------> | 78 | = 111 1000 | **0**111 1000 = 78 |
| space -------> | 20 | = 010 0000 | **1**010 0000 = A0 |
| = ----------> | 3D | = 011 1101 | **1**011 1101 = BD |
| space -----> | 20 | = 010 0000 | **1**010 0000 = A0 |
| 3 ----------> | 3 | = 011 0011 | **0**011 0011 = 33 |
| * ----------> | 2A | = 010 1010 | **1**010 1010 = AA |
| y ----------> | 79 | = 111 1001 | **1**111 1001 = F9 |

***1111 1000*** **0010 0000** ***0011 1101*** **0010 0000** ***1011 0011*** ***0010 1010*** **0111 1001**
x space = space 3 * y

**2.26** (a) 42=***B***; 45=***E***; 4E=***N***; 20=blank; 53=***S***; 4D=***M***; 49=***I***; 54=***T***; 48=***H***.
Thus, the name of the person is ***BEN SMITH***.

(b) 4A=***J***; 6F=***o***; 65=***e***; 20=blank; 47=***G***; 72=***r***; 65=***e***; 65=***e***; 6E=***n***.
Thus, the name of the person is ***Joe Green***.

**2.27** (a) $74_{10} = 01110100_{BCD}$ } **1**01110100 (b) $38_{10} = 00111000_{BCD}$ } **0**00111000
(c) $8884_{10} = 1000100010000100_{BCD}$ } **1**1000100010000100
(d) $275_{10} = 001001110101_{BCD}$ } **0**001001110101
(e) $165_{10} = 000101100101_{BCD}$ } **0**000101100101
(f) $9201_{10} = 1001001000000001_{BCD}$ } **1**1001001000000001

**2.28** (a) 1001 0101 1000 **0** { parity bit
9 5 8
Since the number of 1s is 5, there is ***no single-bit*** error.

(b) 0100 0111 0110 **0**
4 7 6
Since there are six 1s there is a ***single*** error.

c) 0111 1100 0001 **1**
7 12 1

There are seven 1s. However, the second BCD code group has an error since *1100* is an illegal BCD code. Thus, there must be a ***double*** error because there are an odd number of 1s.
(d) 1000 0110 0010 **1**
8 6 2
There are five 1s. Thus, ***no single-bit*** errors.

**2.29** 01001000 } O.K
11000101 } O.K
11001100 } O.K
11001000 } There is a single error.
11001100 } Error can't be detected by the receiver.

**2.30** (a) $10110001001_2$ (b) $11111111_2$ (c) $209_{10}$ (d) $59{,}943_{10}$ (e) $9C1_{16}$ (f) $010100010001_{BCD}$
(g) $565_{10}$ (h) $10DC_{16}$ (i) $1961_{10}$ (j) $15{,}900_{10}$ (k) $640_{16}$ (l) $952B_{16}$
(m) $100001100101_{BCD}$ (n) $947_{10}$ (o) $10001100101_2$ (p) $101100110100_2$
(q) Convert to decimal, then to binary to obtain $1001010_2$
(r) Convert to decimal, then to BCD to obtain $01011000_{BCD}$

**2.31** (a) $100101_2$ (b) $00110111_{BCD}$ (c) $25_{16}$ (d) $01100110110111_{ASCII}$

**2.32** (a) Hex (b) Two (c) digit (d) Gray code (e) parity bit/errors (f) ASCII (g) Hex (h) Byte

**2.33** (a) $1000_2$ (b) $010100_2$ (c) $1100_2$

**2.34** (a) $1011_2$ (b) $100111_2$ (c) $1101_2$

**2.35** (a) $777A_{16}$ (b) $999A_{16}$ (c) $1000_{16}$ (d) $2001_{16}$ (e) $A00_{16}$ (f) $100B_{16}$

**2.36** (a) $7778_{16}$ (b) $9998_{16}$ (c) $0FFE_{16}$ (d) $1FFF_{16}$ (e) $9FE_{16}$ (f) $1009_{16}$

**2.37** (a) A 20-bit address will allow 1,048,576 ($2^{20}$) different memory locations to exist.
(b) Since a hex digit requires 4 bits to represent, it will take 5 hex digits to represent the 20-bit address of a memory location.
(c) $000FF_{16}$

**2.38** (a) $2^6$=64 different voltage values; $2^8$=256 different voltage values; $2^{10}$=1,024 different voltage values.
(b) In 1s there are about 44,000 samples of 10-bits each recorded on the CD surface. Thus, there are about 440,000 bits recorded on the CD disk during 1s of sampling.

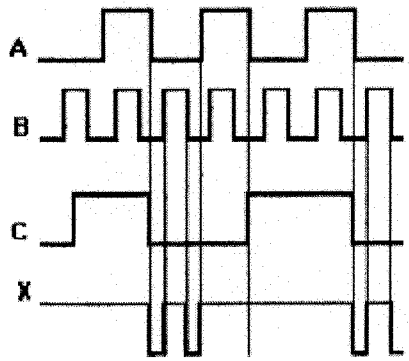
(c) There are about 440,000 bits recorded on the CD disk in 1 second of audio. Therefore, 5 billion bits of audio stored on the CD disk will be equivalent to approximately 11,363.63 seconds ($5x10^9$/440,000).

**2.39** $254=2^x$. Therefore x=7.98~8-bits

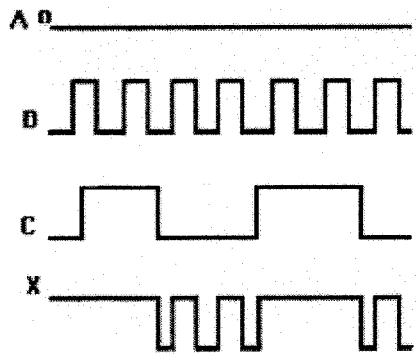
**2.40** Mega = $2^{20}$ =1,048,576
3 Bytes/pixel (1 byte per primary color)
(3 Bytes/pixel) x 3 x 1,048,576 = 9,437,184 Bytes/photo
Memory card capacity = 128 x 1,048,576 = 134,217,728 Bytes/card
Thus, (134,217,728 Bytes/card) / (9,437,184 Bytes/photo) = 14.2 photos/card or ***14 Pictures.***

## CHAPTER THREE - Describing Logic Circuits

**3.1**

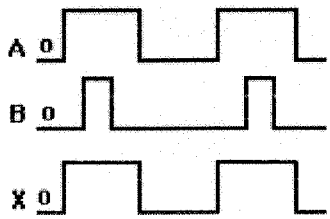



**3.2**

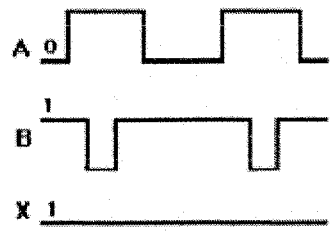



**3.3** With A=1, X will always be 1 since the OR gate output is 1 whenever any input is a 1.

**3.4** (a) Here's one case that refutes this statement.

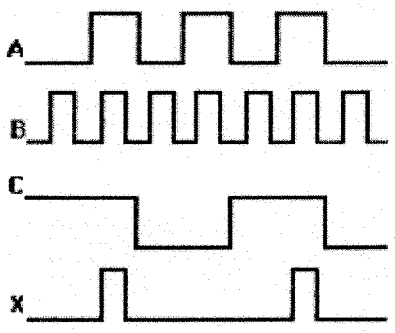



(b) Here's one case that refutes this statement.



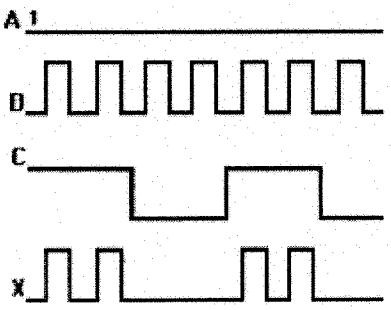
**3.5** There are $2^5$=32 different input conditions. Only one of these (the 00000 condition) produces a LOW output.

**3.6** (a)




(b) X = constant LOW.

(c)



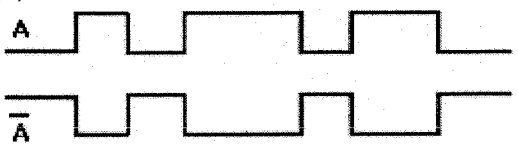

**3.7** Change the OR gate to an AND gate.

**3.8** OUT is always LOW since one or more inputs is always LOW.

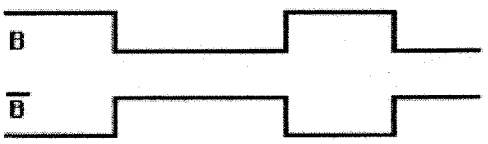
**3.9** A logic HIGH and a logic LOW applied to the inputs of the unknown 2-input gate would tell us what type of gate it is. If the resulting output logic level is HIGH, then the gate is an OR gate. If the resulting output logic level is LOW, then the gate is an AND gate.

**3.10** True. The output of any AND gate will be HIGH only when all of its inputs are HIGH.

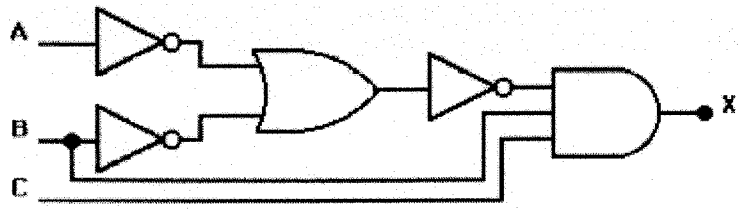
**3.11** (a)




(b)



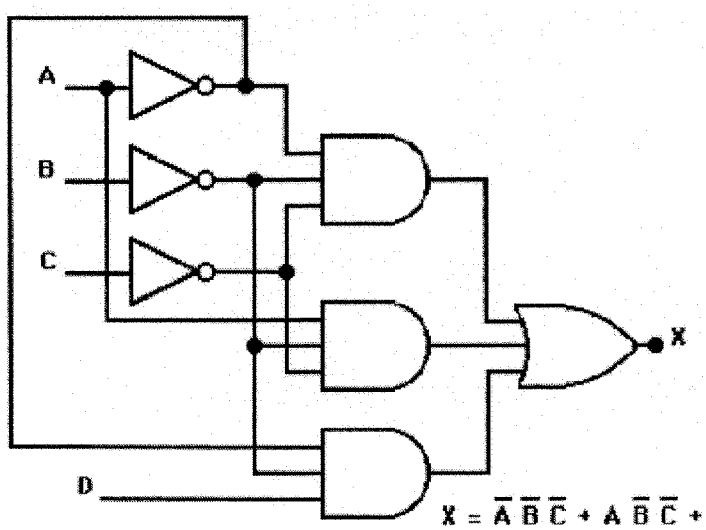
**3.12** (a)




| A | B | C | X |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

$X = [\overline{\overline{A} + \overline{B}}] \cdot BC$

(b)




| A | B | C | D | X |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 |

$X = \overline{A}\,\overline{B}\,\overline{C} + A\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,D$

**3.13**

| *E* | *D* | *C* | *B* | *A* | *(A+B)* | *(A+B)C* | *[(A+B)C]'* | *D+[(A+B)C]'* | *[D+((A+B)C)']E* |
|---|---|---|---|---|---|---|---|---|---|
| *0* | *0* | *0* | *0* | *0* | *0* | *0* | *1* | *1* | *0* |
| *0* | *0* | *0* | *0* | *1* | *1* | *0* | *1* | *1* | *0* |
| *0* | *0* | *0* | *1* | *0* | *1* | *0* | *1* | *1* | *0* |
| *0* | *0* | *0* | *1* | *1* | *1* | *0* | *1* | *1* | *0* |
| *0* | *0* | *1* | *0* | *0* | *0* | *0* | *1* | *1* | *0* |
| *0* | *0* | *1* | *0* | *1* | *1* | *1* | *0* | *0* | *0* |
| *0* | *0* | *1* | *1* | *0* | *1* | *1* | *0* | *0* | *0* |
| *0* | *0* | *1* | *1* | *1* | *1* | *1* | *0* | *0* | *0* |
| *0* | *1* | *0* | *0* | *0* | *0* | *0* | *1* | *1* | *0* |
| *0* | *1* | *0* | *0* | *1* | *1* | *0* | *1* | *1* | *0* |
| *0* | *1* | *0* | *1* | *0* | *1* | *0* | *1* | *1* | *0* |
| *0* | *1* | *0* | *1* | *1* | *1* | *0* | *1* | *1* | *0* |
| *0* | *1* | *1* | *0* | *0* | *0* | *0* | *1* | *1* | *0* |
| *0* | *1* | *1* | *0* | *1* | *1* | *1* | *0* | *1* | *0* |
| *0* | *1* | *1* | *1* | *0* | *1* | *1* | *0* | *1* | *0* |
| *0* | *1* | *1* | *1* | *1* | *1* | *1* | *0* | *1* | *0* |
| *1* | *0* | *0* | *0* | *0* | *0* | *0* | *1* | *1* | *1* |
| *1* | *0* | *0* | *0* | *1* | *1* | *0* | *1* | *1* | *1* |
| *1* | *0* | *0* | *1* | *0* | *1* | *0* | *1* | *1* | *1* |
| *1* | *0* | *0* | *1* | *1* | *1* | *0* | *1* | *1* | *1* |
| *1* | *0* | *1* | *0* | *0* | *0* | *0* | *1* | *1* | *1* |
| *1* | *0* | *1* | *0* | *1* | *1* | *1* | *0* | *0* | *0* |
| *1* | *0* | *1* | *1* | *0* | *1* | *1* | *0* | *0* | *0* |
| *1* | *0* | *1* | *1* | *1* | *1* | *1* | *0* | *0* | *0* |
| *1* | *1* | *0* | *0* | *0* | *0* | *0* | *1* | *1* | *1* |
| *1* | *1* | *0* | *0* | *1* | *1* | *0* | *1* | *1* | *1* |
| *1* | *1* | *0* | *1* | *0* | *1* | *0* | *1* | *1* | *1* |
| *1* | *1* | *0* | *1* | *1* | *1* | *0* | *1* | *1* | *1* |
| *1* | *1* | *1* | *0* | *0* | *0* | *0* | *1* | *1* | *1* |
| *1* | *1* | *1* | *0* | *1* | *1* | *1* | *0* | *1* | *1* |
| *1* | *1* | *1* | *1* | *0* | *1* | *1* | *0* | *1* | *1* |
| *1* | *1* | *1* | *1* | *1* | *1* | *1* | *0* | *1* | *1* |

**3.14**

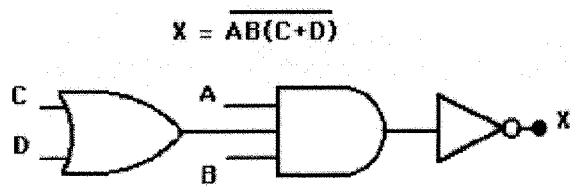
| *E* | *D* | *C* | *B* | *A* | *AB* | *(AB)+C* | *[(AB)+C]'* | *D[(AB)+C]'* | *D[(AB)+C]'+E* |
|---|---|---|---|---|---|---|---|---|---|
| *0* | *0* | *0* | *0* | *0* | *0* | *0* | *1* | *0* | *0* |
| *0* | *0* | *0* | *0* | *1* | *0* | *0* | *1* | *0* | *0* |
| *0* | *0* | *0* | *1* | *0* | *0* | *0* | *1* | *0* | *0* |
| *0* | *0* | *0* | *1* | *1* | *1* | *1* | *0* | *0* | *0* |
| *0* | *0* | *1* | *0* | *0* | *0* | *1* | *0* | *0* | *0* |
| *0* | *0* | *1* | *0* | *1* | *0* | *1* | *0* | *0* | *0* |
| *0* | *0* | *1* | *1* | *0* | *0* | *1* | *0* | *0* | *0* |
| *0* | *0* | *1* | *1* | *1* | *1* | *1* | *0* | *0* | *0* |
| *0* | *1* | *0* | *0* | *0* | *0* | *0* | *1* | *1* | *1* |
| *0* | *1* | *0* | *0* | *1* | *0* | *0* | *1* | *1* | *1* |
| *0* | *1* | *0* | *1* | *0* | *0* | *0* | *1* | *1* | *1* |
| *0* | *1* | *0* | *1* | *1* | *1* | *1* | *0* | *0* | *0* |
| *0* | *1* | *1* | *0* | *0* | *0* | *1* | *0* | *0* | *0* |
| *0* | *1* | *1* | *0* | *1* | *0* | *1* | *0* | *0* | *0* |
| *0* | *1* | *1* | *1* | *0* | *0* | *1* | *0* | *0* | *0* |
| *0* | *1* | *1* | *1* | *1* | *1* | *1* | *0* | *0* | *0* |

| *E* | *D* | *C* | *B* | *A* | *AB* | *(AB)+C* | *[(AB)+C]'* | *D[(AB)+C]'* | *D[(AB)+C]'+E* |
|---|---|---|---|---|---|---|---|---|---|
| *1* | *0* | *0* | *0* | *0* | *0* | *0* | *1* | *0* | *1* |
| *1* | *0* | *0* | *0* | *1* | *0* | *0* | *1* | *0* | *1* |
| *1* | *0* | *0* | *1* | *0* | *0* | *0* | *1* | *0* | *1* |
| *1* | *0* | *0* | *1* | *1* | *1* | *1* | *0* | *0* | *1* |
| *1* | *0* | *1* | *0* | *0* | *0* | *1* | *0* | *0* | *1* |
| *1* | *0* | *1* | *0* | *1* | *0* | *1* | *0* | *0* | *1* |
| *1* | *0* | *1* | *1* | *0* | *0* | *1* | *0* | *0* | *1* |
| *1* | *0* | *1* | *1* | *1* | *1* | *1* | *0* | *0* | *1* |
| *1* | *1* | *0* | *0* | *0* | *0* | *0* | *1* | *1* | *1* |
| *1* | *1* | *0* | *0* | *1* | *0* | *0* | *1* | *1* | *1* |
| *1* | *1* | *0* | *1* | *0* | *0* | *0* | *1* | *1* | *1* |
| *1* | *1* | *0* | *1* | *1* | *1* | *1* | *0* | *0* | *1* |
| *1* | *1* | *1* | *0* | *0* | *0* | *1* | *0* | *0* | *1* |
| *1* | *1* | *1* | *0* | *1* | *0* | *1* | *0* | *0* | *1* |
| *1* | *1* | *1* | *1* | *0* | *0* | *1* | *0* | *0* | *1* |
| *1* | *1* | *1* | *1* | *1* | *1* | *1* | *0* | *0* | *1* |

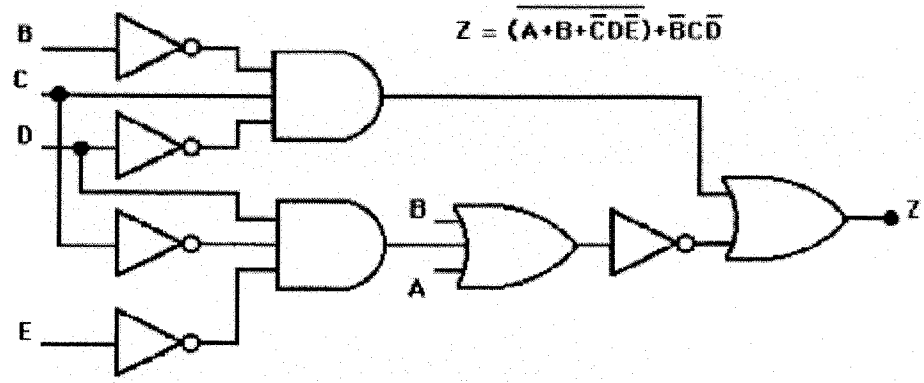
**3.15**

| *A* | *B* | *C* | *D* | *A'BC* | *A+D* | *(A+D)'* | *(A+D)'(A'BC)* |
|---|---|---|---|---|---|---|---|
| *0* | *0* | *0* | *0* | *0* | *0* | *1* | *0* |
| *0* | *0* | *0* | *1* | *0* | *1* | *0* | *0* |
| *0* | *0* | *1* | *0* | *0* | *0* | *1* | *0* |
| *0* | *0* | *1* | *1* | *0* | *1* | *0* | *0* |
| *0* | *1* | *0* | *0* | *0* | *0* | *1* | *0* |
| *0* | *1* | *0* | *1* | *0* | *1* | *0* | *0* |
| *0* | *1* | *1* | *0* | *1* | *0* | *1* | *1* |
| *0* | *1* | *1* | *1* | *1* | *1* | *0* | *0* |
| *1* | *0* | *0* | *0* | *0* | *1* | *0* | *0* |
| *1* | *0* | *0* | *1* | *0* | *1* | *0* | *0* |
| *1* | *0* | *1* | *0* | *0* | *1* | *0* | *0* |
| *1* | *0* | *1* | *1* | *0* | *1* | *0* | *0* |
| *1* | *1* | *0* | *0* | *0* | *1* | *0* | *0* |
| *1* | *1* | *0* | *1* | *0* | *1* | *0* | *0* |
| *1* | *1* | *1* | *0* | *0* | *1* | *0* | *0* |
| *1* | *1* | *1* | *1* | *0* | *1* | *0* | *0* |

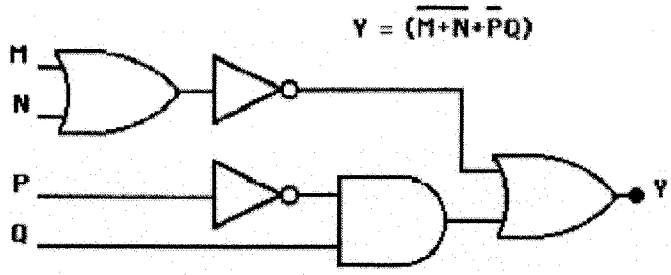
**3.16** (a)



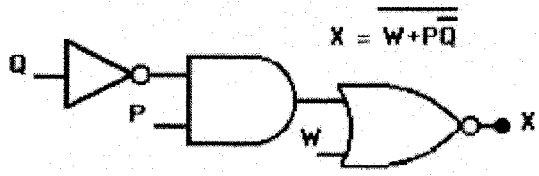
(b)



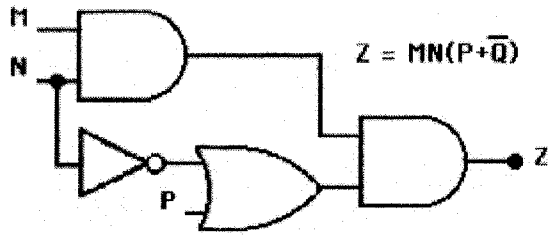

(c)



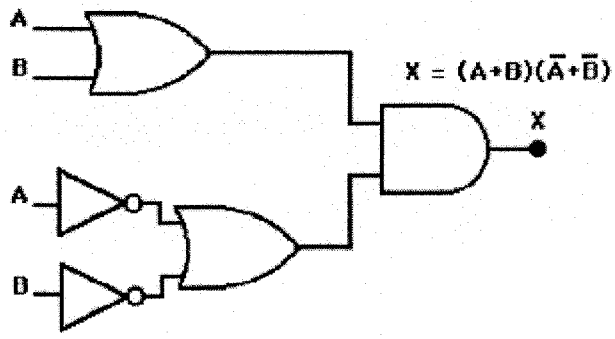

(d)



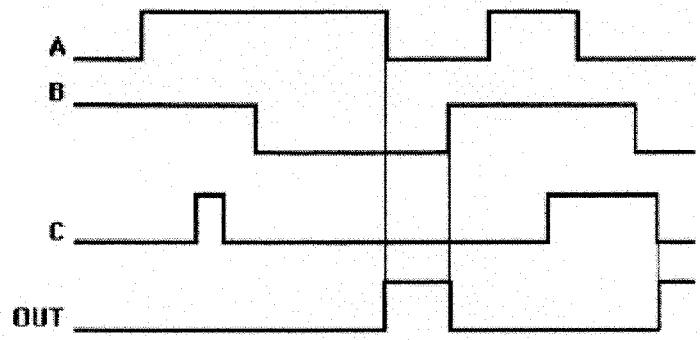

(e)



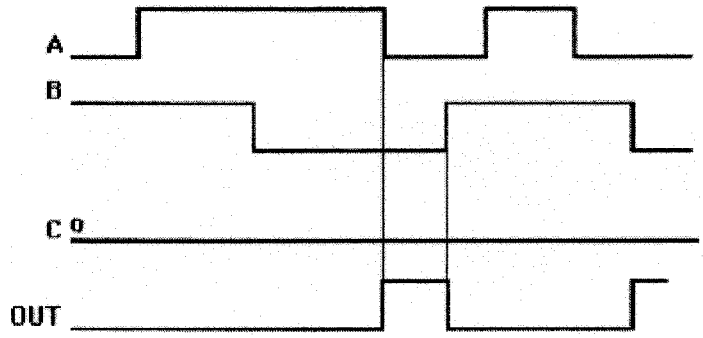
(f)




**3.17** (a) OUT = 1 only when all inputs are = 0.

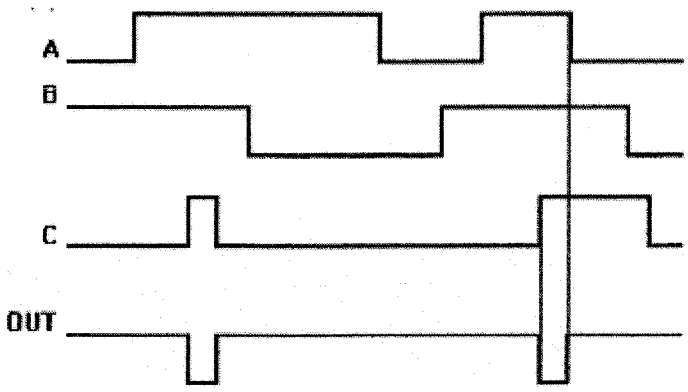



(b) With C = 0

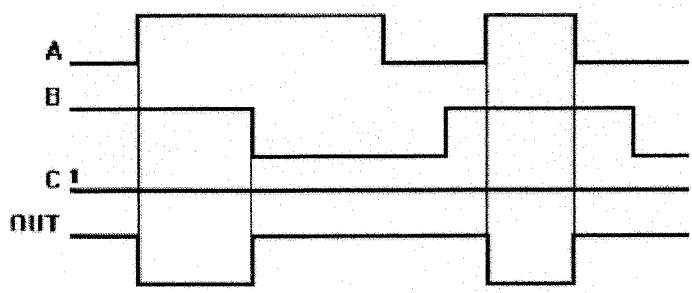



(c) With C = 1, OUT = 0 at all times.

**3.18** (a) OUT = 0 only when all inputs are = 1.



(b) With C = 0, OUT = 1 at all times.
(c) With C = 1

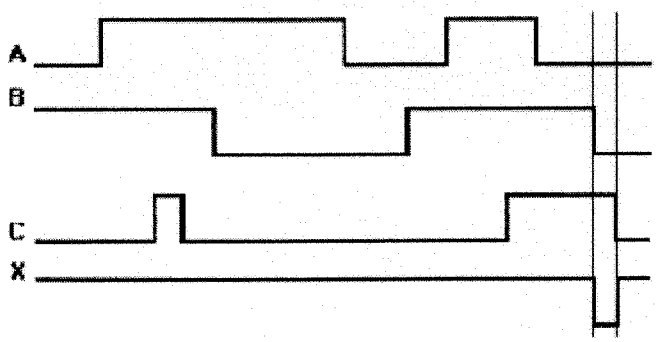



**3.19**

$X = (A+B)(B+\overline{C})$

$X = (A+B)+(B+\overline{C})$

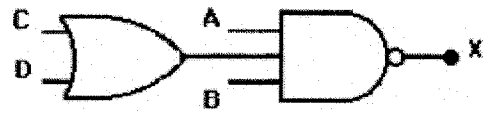
| A | B | C | X |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |



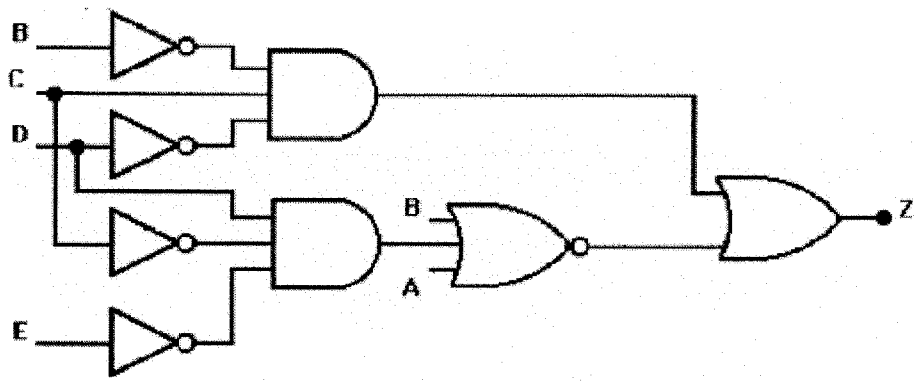

**3.20**

| *A* | *B* | *C* | *D* | *X* |
|---|---|---|---|---|
| *0* | *0* | *0* | *0* | *1* |
| *0* | *0* | *0* | *1* | *1* |
| *0* | *0* | *1* | *0* | *1* |
| *0* | *0* | *1* | *1* | *1* |
| *0* | *1* | *0* | *0* | *1* |
| *0* | *1* | *0* | *1* | *1* |
| *0* | *1* | *1* | *0* | *1* |
| *0* | *1* | *1* | *1* | *1* |
| *1* | *0* | *0* | *0* | *1* |
| *1* | *0* | *0* | *1* | *1* |
| *1* | *0* | *1* | *0* | *1* |
| *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* |

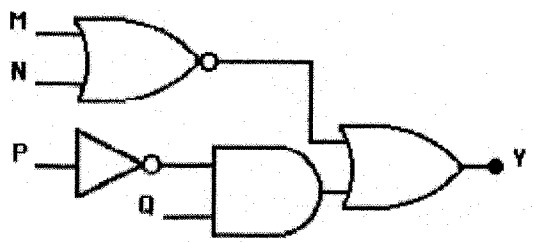
**3.21** (a)




(b)




(c)




**3.22** Proving theorem 15a: $X + \overline{X}Y = X + Y$

| | | | |
|---|---|---|---|
| **X=1** | { 1 + 0 1 = 1 } | **X=0** | { 0 + 1 0 = 0 + 0 } |
| **Y=1** | { 1 + 0 = 1 } | **Y=0** | { 0 + 0 = 0 } |
| | { **1=1** } | | { **0=0** } |
| **X=1** | { 1 + 0 0 = 1 + 0 } | **X=0** | { 0 + 1 1 = 0 + 1} |
| **Y=0** | { 1 + 0 = 1 } | **Y=1** | { 0 + 1 = 1 } |
| | { **1=1** } | | { **1=1** } |

Proving theorem 15b: $\overline{X} + XY = \overline{X} + Y$

| | | | |
|---|---|---|---|
| **X=1** | { 0 + 1 1 = 1 } | **X=0** | { 1 + 0 0 = 1 + 0 } |
| **Y=1** | { 0 + 1 = 1 } | **Y=0** | { 1 + 0 = 1 } |
| | { **1=1** } | | { **1=1** } |
| **X=1** | { 0 + 1 0 = 0 + 0 } | **X=0** | { 1 + 0 1 = 1 + 0} |
| **Y=0** | { 0 + 0 = 0 } | **Y=1** | { 1 + 0 = 1 } |
| | { **0=0** } | | { **1=1** } |

**3.23**

(a) $A+1=1$ (b) $A \cdot A = A$ (c) $B \cdot \overline{B} = 0$ (d) $C+C=C$ (e) $X \cdot 0 = 0$ (f) $D \cdot 1 = D$
(g) $D+0=D$ (h) $C+\overline{C}=1$ (i) $G+GF=G$ (j) $Y+\overline{W}Y=Y$

**3.24**

(a)

$$X = (M+N)(\overline{M}+P)(\overline{N}+\overline{P})$$
$$X = (M\overline{M}+MP+N\overline{M}+NP)(\overline{N}+\overline{P})$$
$$X = (M\overline{M}\,\overline{N}+M\overline{M}\,\overline{P}+MP\overline{N}+MP\overline{P}+N\overline{M}\,\overline{N}+N\overline{M}\,\overline{P}+NP\overline{N}+NP\overline{P})$$
$$X = (0+0+MP\overline{N}+0+0+N\overline{M}\,\overline{P}+0+0)$$
$$X = MP\overline{N}+N\overline{M}\,\overline{P}$$

(b)

$$Z = \overline{A}B\overline{C}+AB\overline{C}+B\overline{C}D$$
$$Z = B\overline{C}(\overline{A}+A+D)$$
$$Z = B\overline{C}(1+D)$$
$$Z = B\overline{C}$$

**3.25** $\overline{A+B} = \overline{A} \cdot \overline{B}$

| | | | |
|---|---|---|---|
| **A=1** | $\overline{1+1} = \overline{1} \cdot 1$ | **A=0** | $\overline{0+0} = \overline{0} = 1$ |
| **B=1** | | **B=0** | |
| **A=0** | $\overline{0+1} = \overline{0} \cdot \overline{1} = 0$ | **A=1** | $\overline{1+0} = \overline{1} \cdot \overline{0} = 0$ |
| **B=1** | | **B=0** | |

$\overline{A \cdot B} = \overline{A} + \overline{B}$

| | | | |
|---|---|---|---|
| **A=1** | $\overline{1 \cdot 1} = \overline{1} + \overline{1} = 0$ | **A=0** | $\overline{0 \cdot 0} = \overline{0} = 1$ |
| **B=1** | | **B=0** | |
| **A=0** | $\overline{0 \cdot 1} = \overline{0} + \overline{1} = 1$ | **A=1** | $\overline{1 \cdot 0} = \overline{1} + \overline{0} = 1$ |
| **B=1** | | **B=0** | |

**3.26**

(a) $\overline{\overline{A}\,\overline{B}\,\overline{C}} = \overline{\overline{A}} + \overline{\overline{B}} + \overline{\overline{C}} = A + \overline{B} + C$
(b) $\overline{\overline{A} + \overline{B}C} = \overline{\overline{A}}(\overline{\overline{B}} + \overline{C}) = A(B + \overline{C})$
(c) $\overline{AB\overline{CD}} = \overline{AB} + \overline{\overline{CD}} = \overline{A} + \overline{B} + CD$
(d) $\overline{A + \overline{B}} = \overline{A}\,\overline{\overline{B}} = \overline{A}B$
(e) $\overline{\overline{A}\,\overline{B}} = \overline{\overline{A}} + \overline{\overline{B}} = A + B$
(f) $\overline{\overline{A} + \overline{C} + \overline{D}} = \overline{\overline{A}\,\overline{\overline{C}}\,\overline{\overline{D}}} = ACD$

(g) $\overline{A(B+\overline{C})D} = \overline{A} + \overline{(B+\overline{C})} + \overline{D} = \overline{A} + B + \overline{C} + \overline{D}$

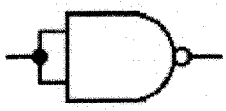
(h) $\overline{(M+\overline{N})(\overline{M}+N)} = \overline{M}N + M\overline{N}$

(i) $\overline{\overline{A}BCD} = \overline{A}\overline{B}C + \overline{D} = (\overline{A}+\overline{B})C + \overline{D}$

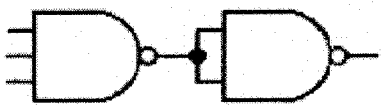
**3.27**

$$X = \overline{\overline{(A+B)}\overline{B}C} = \overline{\overline{A}+\overline{B}+\overline{B}C} = A + B + B + \overline{C} = A + B + \overline{C}$$

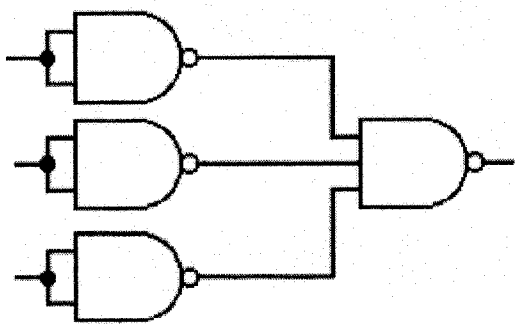
**3.28** Change each inverter to:



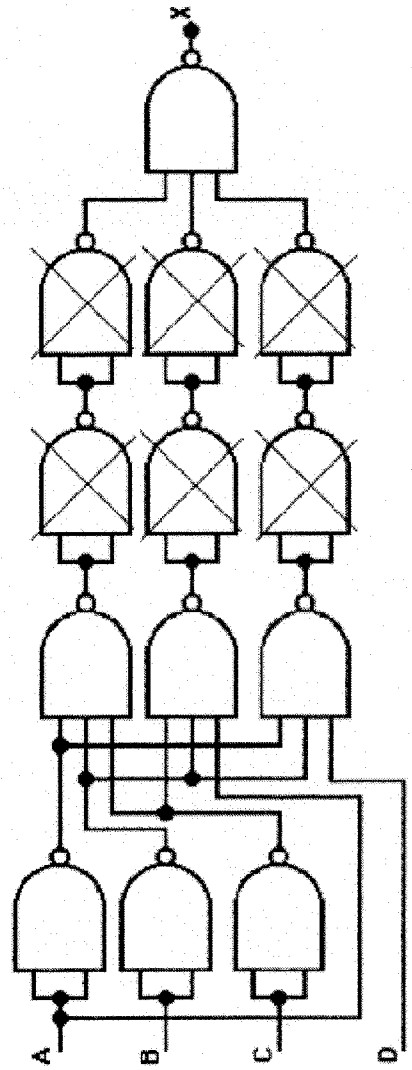
Change each AND to:



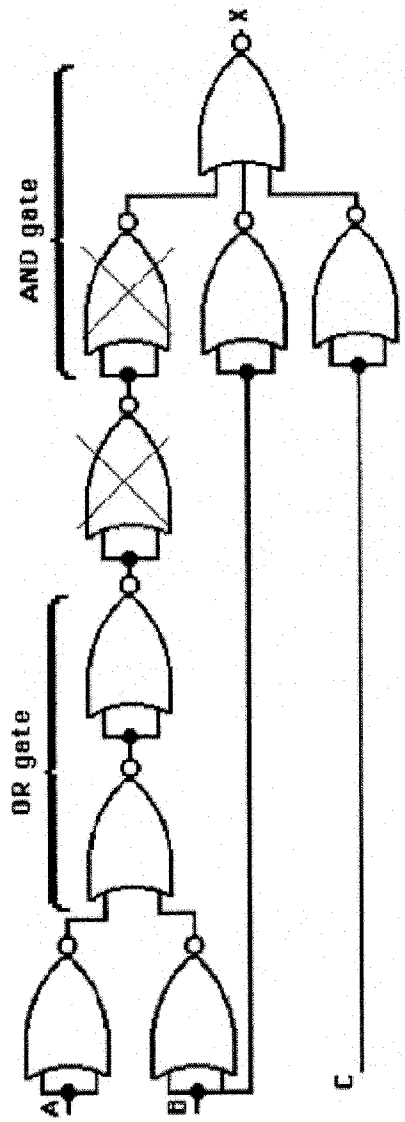
Change each OR to:



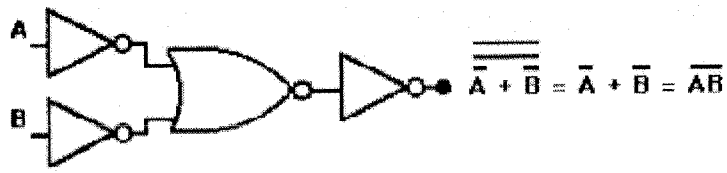
By canceling double INVERTERS, the result is: $X = \overline{A}\,\overline{B}\,\overline{C} + A\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,D$



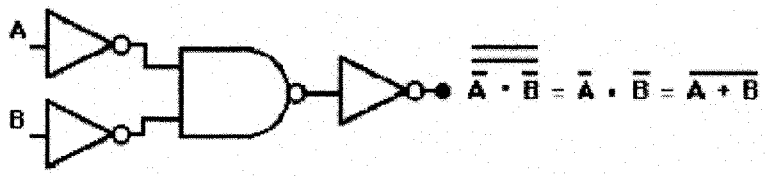
**3.29** X=ABC


X
AND gate
OR gate
A
B
C

**3.30**

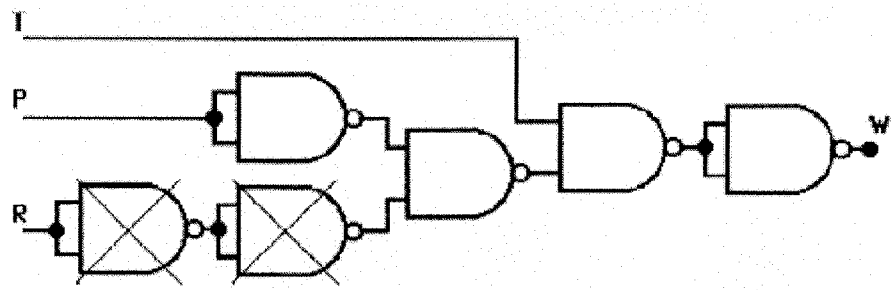

A
B
$\overline{\overline{A} + \overline{B}} = \overline{\overline{A}} \cdot \overline{\overline{B}} = AB$

**3.31**



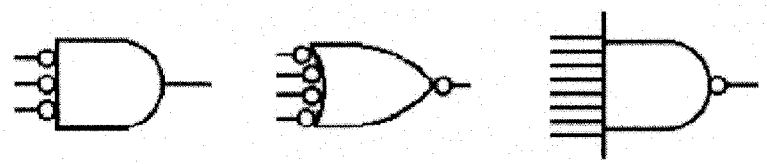

**3.32** (a) The warning light W will be activated, when temperature (T) is >200°F **and** either the pressure (P) is >220 p.s.i., **or** the speed (R) is < 4800 r.p.m. In conclusion, **W=1 when T=1 and either P=1 or R=0.**

(b)

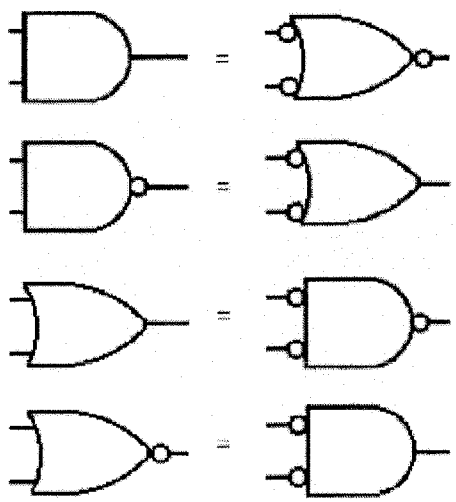



**3.33**

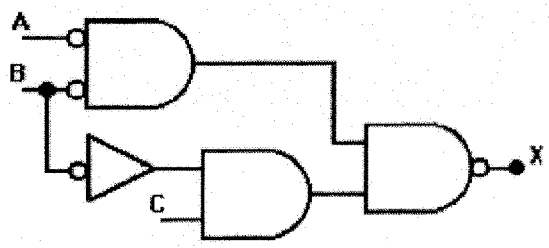
(a) NOR gate (b) AND gate (c) NAND gate



**3.34**



**3.35** (a)

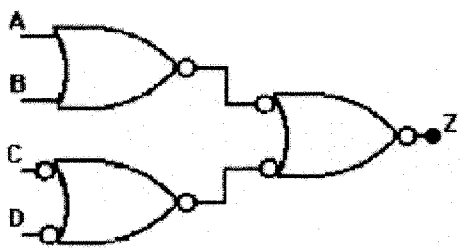



(b)

| *A* | *B* | *C* | *X* |
|---|---|---|---|
| *0* | *0* | *0* | *1* |
| *0* | *0* | *1* | *0* |
| *0* | *1* | *0* | *1* |
| *0* | *1* | *1* | *1* |
| *1* | *0* | *0* | *1* |
| *1* | *0* | *1* | *1* |
| *1* | *1* | *0* | *1* |
| *1* | *1* | *1* | *1* |

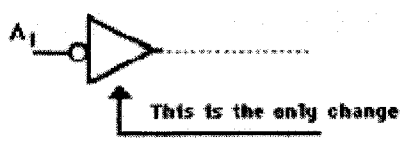
**3.36** (a) Z is HIGH only when A=B=0 and C=D=1.

(b)




Z will be LOW when A or B is HIGH, or when C or D is LOW.

**3.37**




**3.38** X will go HIGH when E=1, or D=0, or C=B=0, or when B=1 and A=0.

**3.39** (a) X is asserted (active) HIGH.
(b) Z is asserted (active) LOW.

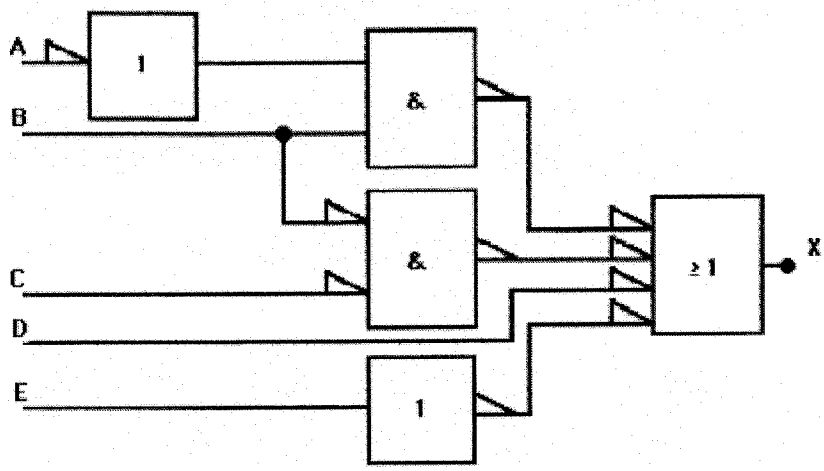
**3.40**

| *E* | *D* | *C* | *B* | *A* | *X* |
|---|---|---|---|---|---|
| *0* | *0* | *0* | *0* | *0* | *1* |
| *0* | *0* | *0* | *0* | *1* | *1* |
| *0* | *0* | *0* | *1* | *0* | *1* |
| *0* | *0* | *0* | *1* | *1* | *1* |
| *0* | *0* | *1* | *0* | *0* | *1* |
| *0* | *0* | *1* | *0* | *1* | *1* |
| *0* | *0* | *1* | *1* | *0* | *1* |
| *0* | *0* | *1* | *1* | *1* | *1* |
| *0* | *1* | *0* | *0* | *0* | *1* |
| *0* | *1* | *0* | *0* | *1* | *1* |
| *0* | *1* | *0* | *1* | *0* | *1* |
| *0* | *1* | *0* | *1* | *1* | *0* |
| *0* | *1* | *1* | *0* | *0* | *0* |
| *0* | *1* | *1* | *0* | *1* | *0* |
| *0* | *1* | *1* | *1* | *0* | *1* |
| *0* | *1* | *1* | *1* | *1* | *1* |
| *1* | *0* | *0* | *0* | *0* | *1* |
| *1* | *0* | *0* | *0* | *1* | *1* |
| *1* | *0* | *0* | *1* | *0* | *1* |
| *1* | *0* | *0* | *1* | *1* | *1* |
| *1* | *0* | *1* | *0* | *0* | *1* |
| *1* | *0* | *1* | *0* | *1* | *1* |
| *1* | *0* | *1* | *1* | *0* | *1* |
| *1* | *0* | *1* | *1* | *1* | *1* |
| *1* | *1* | *0* | *0* | *0* | *1* |
| *1* | *1* | *0* | *0* | *1* | *1* |
| *1* | *1* | *0* | *1* | *0* | *1* |
| *1* | *1* | *0* | *1* | *1* | *1* |
| *1* | *1* | *1* | *0* | *0* | *1* |
| *1* | *1* | *1* | *0* | *1* | *1* |
| *1* | *1* | *1* | *1* | *0* | *1* |
| *1* | *1* | *1* | *1* | *1* | *1* |

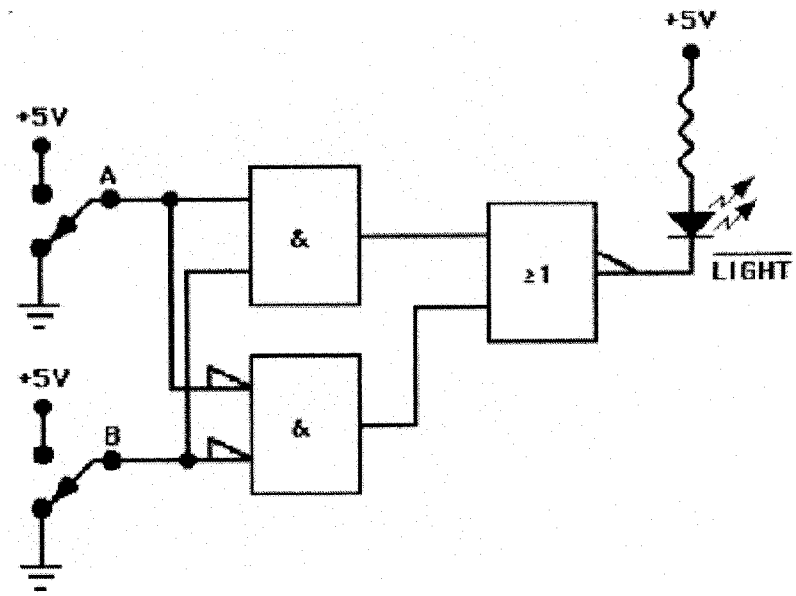
**3.41** $\overline{\text{LIGHT}}$ = LOW when A=B=1, or when A=B=0.

| *B* | *A* | $\overline{\text{LIGHT}}$ |
|---|---|---|
| *0* | *0* | *0* |
| *0* | *1* | *1* |
| *1* | *0* | *1* |
| *1* | *1* | *0* |

**3.42** (a)



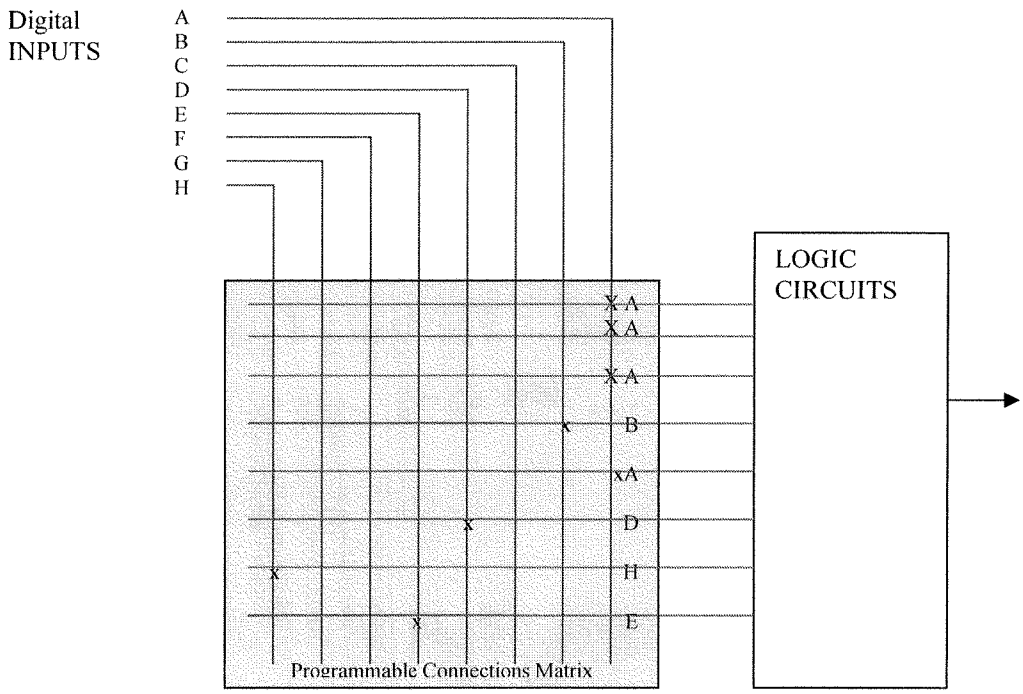

(b)




**3.43**

(a) FALSE (b) TRUE (c) FALSE (d) TRUE (e) FALSE (f) FALSE (g) TRUE
(h) FALSE (I) TRUE (j) TRUE

**3.44**




**3.45**

**AHDL**

```
SUBDESIGN prob3_45
(
                    a,b,c           :INPUT;          --define inputs to block
                    x1,y,z          :OUTPUT;         --define block output
)
                    BEGIN
                    x1 = a # b;
                    y = !(a & b);
                    z = a # b # c;
                    END;
```

**VHDL**

```
ENTITY prob3_45 IS
                    PORT (  a, b, c         :IN bit;         --define inputs to block
                            x, y, z         :OUT bit);       --define block output
END prob3_45  ;

ARCHITECTURE ckt OF prob3_45 IS
                    BEGIN
                            x< = a OR b;                     -- logic descriptions
                            y <= NOT(a AND b);
                            z <= a OR b OR c;
                    END ckt;
```

## 3.46

### (a) AHDL

```
SUBDESIGN prob3_46
(
        rd, rom_a, rom_b, ram           :INPUT;         --define inputs to block
        mem                             :OUTPUT;        --define block output
)
        BEGIN
                mem = !rd & ((!rom_a # !rom_b) # !ram);
        END;
```

### (a) VHDL

```
ENTITY prob3_46 IS
        PORT (rd, rom_a, rom_b, ram     :IN bit;        --define inputs to block
                mem                     :OUT bit);      --define block output
END prob3_46;

ARCHITECTURE ckt OF prob3_46 IS
        BEGIN
                mem <= (NOT rd) AND ((NOT rom_a) OR (NOT rom_b) OR (NOT ram));
        END ckt;
```

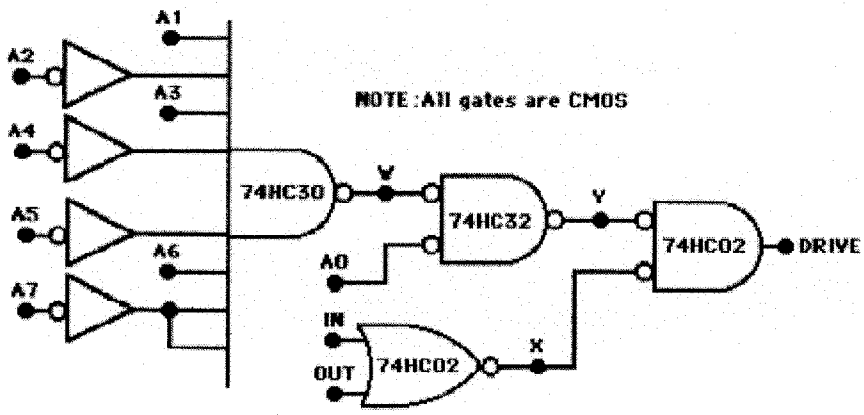
### (b) AHDL

```
SUBDESIGN prob3_46
(
        rd, rom_a, rom_b, ram           :INPUT;         --define inputs to block
        mem                             :OUTPUT;        --define block output
)
VARIABLE
        v,w,x1,y  :NODE;
        BEGIN
                x1 = !rd;
                w = !rom_a # !rom_b;
                v = !ram;
                y = w # v;
                mem = x1 & y;
        END;
```

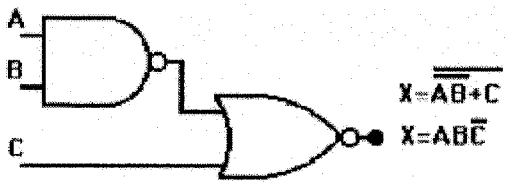
### (b) VHDL

```
ENTITY prob3_46 IS
        PORT (rd, rom_a, rom_b, ram     :IN bit;        --define inputs to block
                mem                     :OUT bit);      --define block output
END prob3_46;

ARCHITECTURE ckt OF prob3_46 IS
SIGNAL v,w,x,y   :BIT;

        BEGIN
                x <= NOT rd;
                w <= (NOT rom_a) OR (NOT rom_b);
                v <= NOT ram;
                y <= w OR v;
                mem = x AND y;
        END ckt;
```

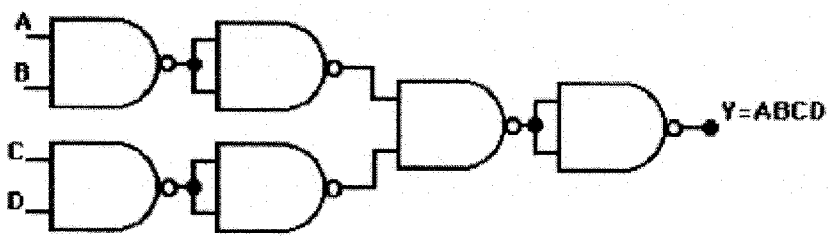
**3.47**


A1
A2
A3
A4
A5
A6
A7
NOTE :All gates are CMOS
74HC30
V
74HC32
Y
74HC02
DRIVE
A0
IN
OUT
74HC02
X

**3.48**


A
B
C
$X=\overline{\overline{AB}+C}$
$X=AB\overline{C}$

**3.49**


A
B
C
D
Y=ABCD

## CHAPTER FOUR - Combinational Logic Circuits

**4.1**

(a) $x = ABC + \overline{A}C = C(AB + \overline{A}) = C(\overline{A} + B)$

(b) $y = (Q + R)(\overline{Q} + \overline{R}) = Q\overline{Q} + Q\overline{R} + R\overline{Q} + R\overline{R} = Q\overline{R} + R\overline{Q}$

(c) $w = ABC + A\overline{B}C + \overline{A} = AC(B + \overline{B}) + \overline{A} = AC + \overline{A} = \overline{A} + C$

(d)

$q = \overline{RST(R + S + T)}$

$q = (\overline{R} + \overline{S} + \overline{T})(\overline{RST})$

$q = \overline{R}RST + \overline{S}RST + \overline{T}RST$

$q = \overline{RST} + \overline{RST} + \overline{RST}$

$q = \overline{RST}$

(e)

$x = \overline{A}\overline{B}C + \overline{A}BC + ABC + A\overline{B}C + A\overline{B}C$

$x = \overline{A}\overline{B}C + BC(A + \overline{A}) + A\overline{B}(C + \overline{C})$

$x = \overline{A}\overline{B}C + BC + A\overline{B}$

$x = BC + \overline{B}(\overline{A}\overline{C} + A) = BC + \overline{B}(A + \overline{C})$

**One possibility:**

(f)

$z = (B + \overline{C})(\overline{B} + C) + \overline{\overline{A} + B + \overline{C}}$

$z = B\overline{B} + BC + \overline{B}C + C\overline{C} + \overline{\overline{A}}\,\overline{B}\,\overline{\overline{C}}$

$z = BC + \overline{B}C + A\overline{B}C$

$z = BC + \overline{B}(\overline{C} + AC)$

$z = BC + \overline{B}(\overline{C} + A)$

$z = BC + \overline{B}C + A\overline{B}$

(g)

$y = \overline{(C + D)} + \overline{A}C\overline{D} + A\overline{B}C + \overline{A}B\overline{C}D + AC\overline{D}$

$y = \overline{C}\overline{D} + \overline{A}C\overline{D} + A\overline{B}C + \overline{A}B\overline{C}D + AC\overline{D}$

$y = \overline{C}\overline{D} + C\overline{D}(\overline{A} + A) + A\overline{B}C + \overline{A}B\overline{C}D$

$y = \overline{C}\overline{D} + C\overline{D} + A\overline{B}C + \overline{A}B\overline{C}D$

$y = \overline{D}(\overline{C} + C) + A\overline{B}C + \overline{A}B\overline{C}D$

$y = \overline{D} + A\overline{B}C + \overline{A}B\overline{C}D$

$y = \overline{D} + A\overline{B}C + \overline{A}BC$

(h)

$$x = AB(\overline{\overline{C}D}) + \overline{A}BD + \overline{BCD}$$

$$x = AB(C + \overline{D}) + \overline{A}BD + \overline{BCD}$$

$$x = ABC + AB\overline{D} + \overline{A}BD + \overline{BCD}$$

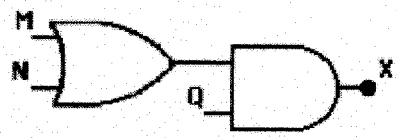
**4.2**

$$X = \overline{\overline{M}\,N\,Q} \cdot \overline{M\,\overline{N}\,Q} \cdot \overline{\overline{M}\,N\,Q}$$

$$X = M\,N\,Q + M\,\overline{N}\,Q + \overline{M}\,N\,Q$$

$$X = MQ + NQ$$

$$X = Q(M + N)$$




**4.3**

$$X = \overline{(M + N + Q)} + \overline{(M + \overline{N} + Q)} + \overline{(\overline{M} + N + Q)}$$

$$X = (M + N + Q)(M + \overline{N} + Q)(\overline{M} + N + Q)$$

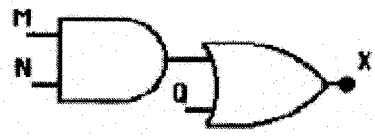
$$X = (MM + M\overline{N} + MQ + NM + N\overline{N} + NQ + QM + Q\overline{N} + QQ)(\overline{M} + N + Q)$$

$$X = (M + M\overline{N} + MQ + NM + NQ + QM + Q\overline{N} + Q)(\overline{M} + N + Q)$$

$$X = (M + Q)(\overline{M} + N + Q)$$

$$X = M\overline{M} + MN + MQ + \overline{M}Q + QN + QQ$$

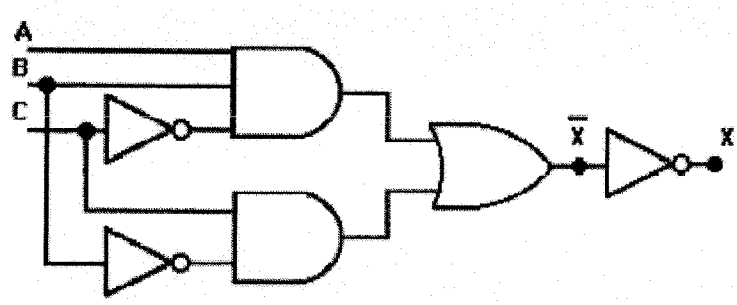
$$X = MN + Q$$




**4.4** Use $\overline{X}$ since this would give only three terms.

$$\overline{X} = \overline{A}\,\overline{B}\,C + A\,\overline{B}\,C + A\,B\,\overline{C}$$

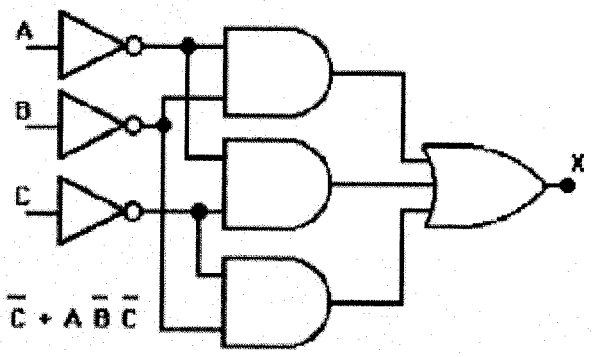
$$\overline{X} = \overline{B}C + AB\overline{C}$$




Alternate solution using S-of-P expression for X would be: $X = \overline{A}B + \overline{B\overline{C}} + BC$

**4.5**

| A | B | C | X |
|---|---|---|---|
| 0 | 0 | 0 | 1 ($\bar{A}\bar{B}\bar{C}$) |
| 0 | 0 | 1 | 1 ($\bar{A}\bar{B}C$) |
| 0 | 1 | 0 | 1 ($\bar{A}B\bar{C}$) |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 ($A\bar{B}\bar{C}$) |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |




$X = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C}$

By adding the term $\bar{A}\bar{B}\bar{C}$ three times and then factoring, the following is obtained:

$$X = \bar{A}\bar{B}(C + \bar{C}) + \bar{A}\bar{C}(B + \bar{B}) + \bar{B}\bar{C}(A + \bar{A})$$

$$X = \bar{A}\bar{B} + \bar{A}\bar{C} + \bar{B}\bar{C}$$

**4.6** ***Make the following assumptions:***

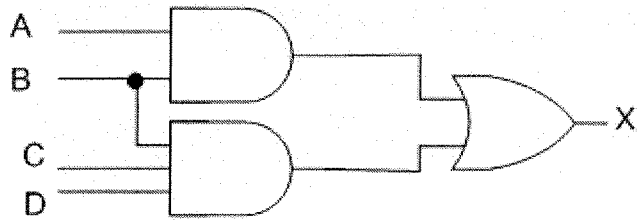
**A** - It's 5:00 or later; **B** - All machines are shut down; **C** - It's Friday
**D** - Production run for the day is complete

*Output Y assumes all variables that are not mentioned in the conditions of the story problem must be zero to blow the horn. Output X assumes that all variables that are not mentioned in the conditions of the story problem can be either 1 or 0 in order to blow the horn.*

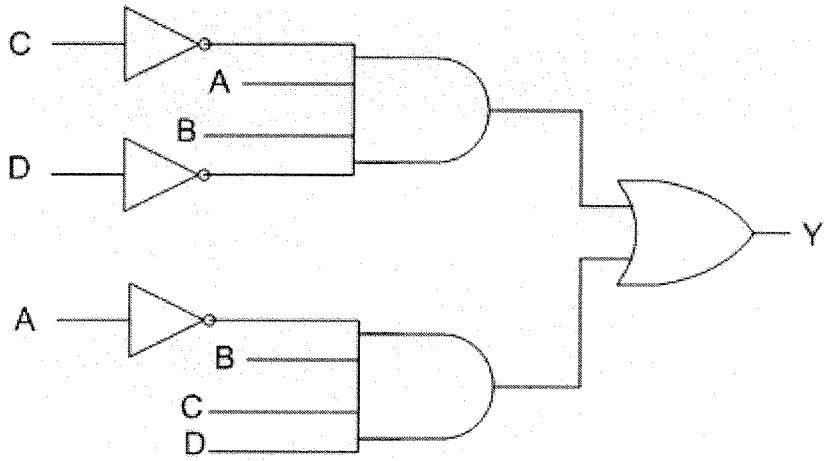
| *D* | *C* | *B* | *A* | *X* | *Y* |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 |

$$X = AB\bar{C}\bar{D} + AB\bar{C}D + AB\bar{C}D + \bar{A}BCD + ABCD$$

$$X = AB + BCD$$




$$Y = AB\overline{C}\,\overline{D} + \overline{A}BCD$$




**4.7**

| *A3* | *A2* | *A1* | *A0* | *X* |
|---|---|---|---|---|
| *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* | *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* | *0* |
| *1* | *1* | *0* | *0* | *0* |
| *1* | *1* | *0* | *1* | *0* |
| *1* | *1* | *1* | *0* | *0* |
| *1* | *1* | *1* | *1* | *0* |

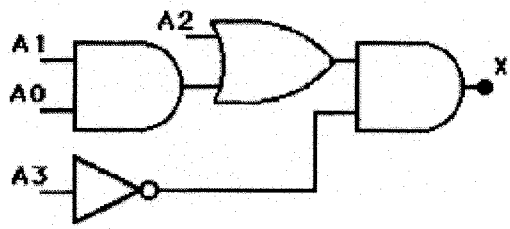
By inspection, X will be 1 whenever A3=0, A2=1; or when A3=A2=0, while A1=A0=1. Thus, we can write:

$$X = \overline{A3}\ A2 + \overline{A3}\ \overline{A2}\ A1\ A0$$

$$X = \overline{A3}(A2 + \overline{A2}\ A1\ A0)$$

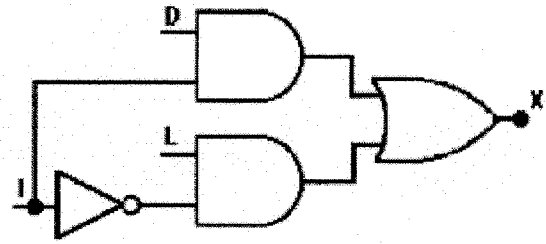
$$X = \overline{A3}(A2 + A1\ A0)$$

The same result can be obtained by writing the S-of-P expression and then simplifying it.




**4.8** **Door = D; Ignition = I; Lights = L**

| L | I | D | X |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

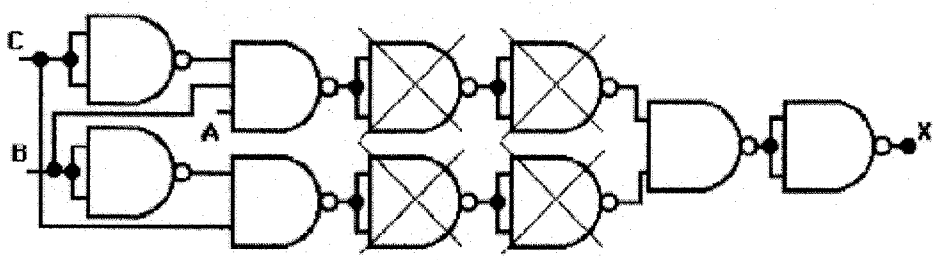



$X = \bar{L}\,I\,D + L\,\bar{I}\,\bar{D} + L\,\bar{I}\,D + L\,I\,D$

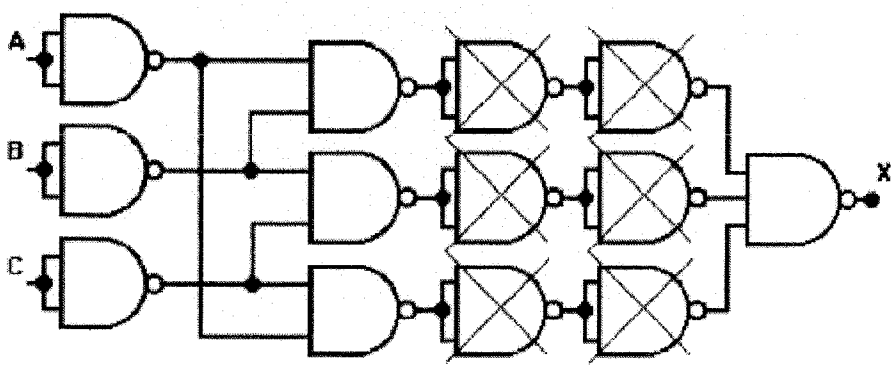
$X = ID(\bar{L} + L) + L\bar{I}(\bar{D} + D)$

$X = ID + L\bar{I}$

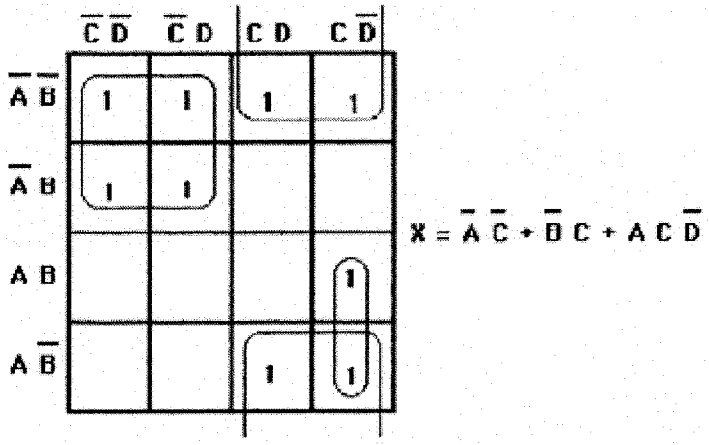
**4.9** Change each gate to its NAND equivalent and then cancel double inversions.



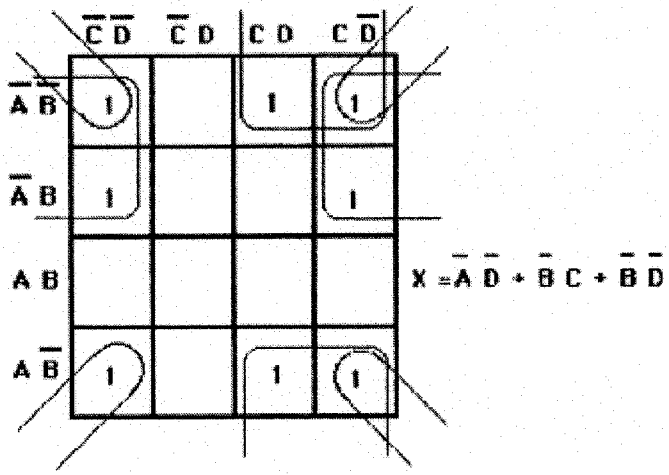

**4.10** Change each gate to its NAND equivalent and then cancel double inversions.



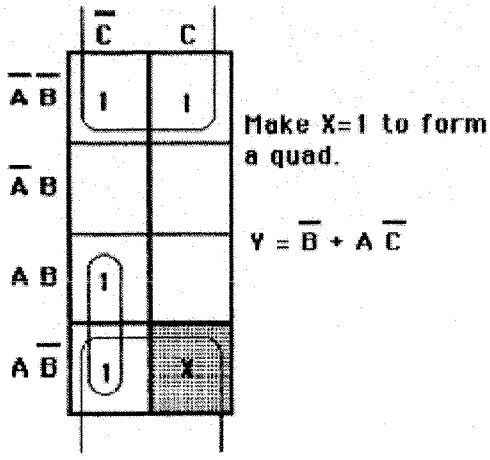
**4.11** (a)


$\overline{C}\,\overline{D}$ $\overline{C}\,D$ $C\,D$ $C\,\overline{D}$
$\overline{A}\,\overline{B}$
$\overline{A}\,B$
$A\,B$
$A\,\overline{B}$
$X = \overline{A}\,\overline{C} + \overline{B}\,C + A\,C\,\overline{D}$

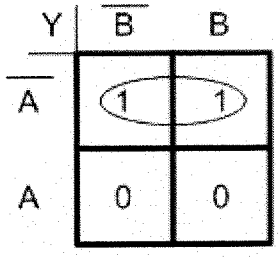
(b)


$\overline{C}\,\overline{D}$ $\overline{C}\,D$ $C\,D$ $C\,\overline{D}$
$\overline{A}\,\overline{B}$
$\overline{A}\,B$
$A\,B$
$A\,\overline{B}$
$X = \overline{A}\,\overline{D} + \overline{B}\,C + \overline{B}\,\overline{D}$

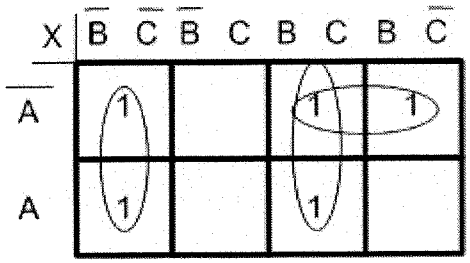
(c)


$\overline{C}$ $C$
$\overline{A}\,\overline{B}$
$\overline{A}\,B$
$A\,B$
$A\,\overline{B}$
X
Make X=1 to form a quad.
$Y = \overline{B} + A\,\overline{C}$

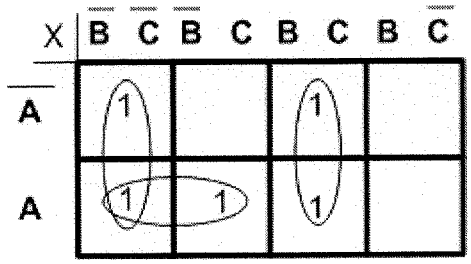
**4.12** $Y = \overline{A}$




**4.13** $X = \overline{B}\overline{C} + BC + \overline{A}B$ **Other solution:** $X = \overline{B}\overline{C} + BC + \overline{A}\overline{C}$

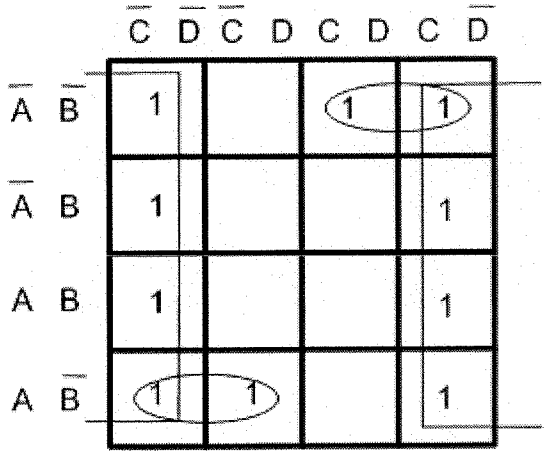



**4.14 (a)** $X = \overline{A}\overline{B}\overline{C} + \overline{A}BC + ABC + A\overline{B}\overline{C} + A\overline{B}C$




$X = \overline{B}\overline{C} + BC + A\overline{B}$ **Other solution:** $X = \overline{B}\overline{C} + BC + AC$

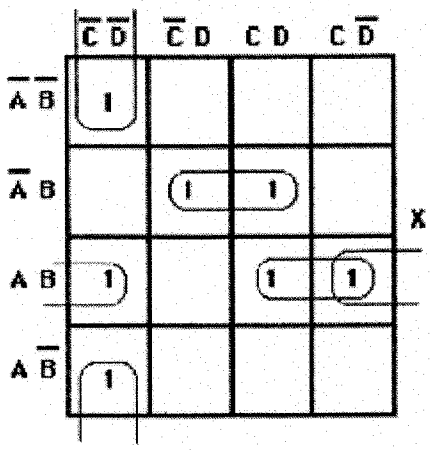
**(b)** $Y = \overline{C}\overline{D} + \overline{A}C\overline{D} + A\overline{B}\overline{C} + \overline{A}\overline{B}CD + AC\overline{D}$




$Y = \overline{D} + A\overline{B}\overline{C} + \overline{A}\overline{B}C$

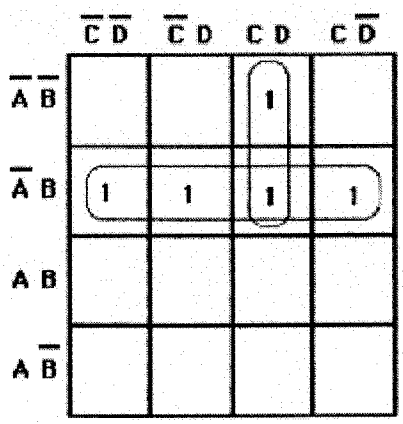
**(c)** One possibility:

$X = A\,B\,C + A\,B\,\overline{D} + \overline{A}\,B\,D + \overline{B}\,\overline{C}\,\overline{D}$




$X = \overline{A}\,B\,D + A\,B\,C + A\,B\,\overline{D} + \overline{C}\,\overline{D}\,\overline{B}$

**4.15** For visual convenience, let A3=A, A2=B, A1=C, A0=D

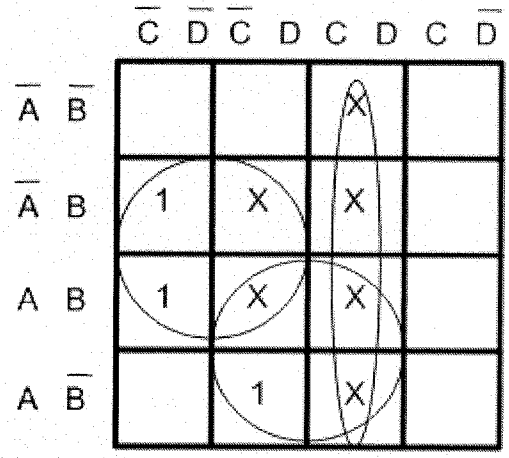



$X = \overline{A}\,B + \overline{A}\,C\,D$

$X = \overline{A3} \cdot A2 + \overline{A3} \cdot A1 \cdot A0$

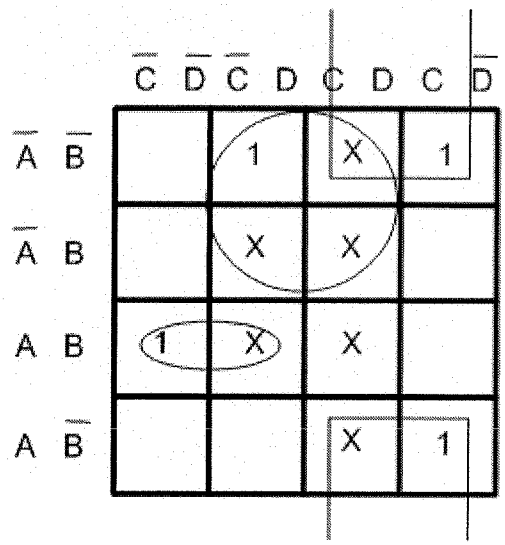
**4.16 (a)** $X = B\overline{C} + AD$

| D | C | B | A | X |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | x |
| 1 | 0 | 1 | 1 | x |
| 1 | 1 | 0 | 0 | x |
| 1 | 1 | 0 | 1 | x |
| 1 | 1 | 1 | 0 | x |
| 1 | 1 | 1 | 1 | x |



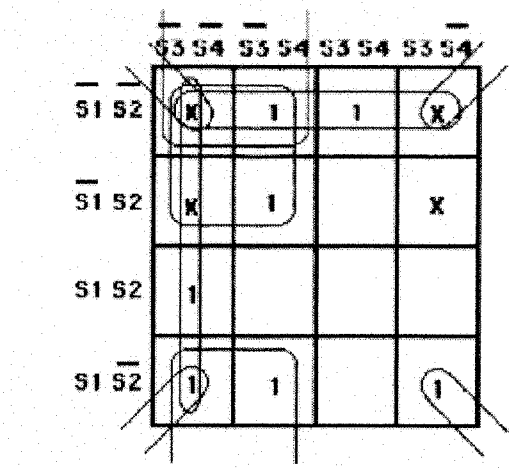

**(b)** $X = \overline{B}C + \overline{A}D + AB\overline{C}$

| D | C | B | A | X |
|---|---|---|---|---|
| 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 | 1 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | x |
| 1 | 0 | 1 | 1 | x |
| 1 | 1 | 0 | 0 | x |
| 1 | 1 | 0 | 1 | x |
| 1 | 1 | 1 | 0 | x |
| 1 | 1 | 1 | 1 | x |



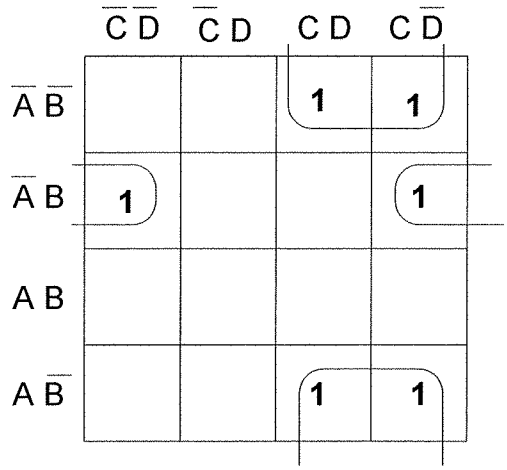
**4.17**

| S4 | S3 | S2 | S1 | X | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | X | |
| 0 | 0 | 0 | 1 | 1 | $\overline{S1}\,\overline{S2}\,\overline{S3}\,\overline{S4}$ |
| 0 | 0 | 1 | 0 | X | |
| 0 | 0 | 1 | 1 | 1 | $\overline{S1}\,\overline{S2}\,\overline{S3}\,S4$ |
| 0 | 1 | 0 | 0 | X | |
| 0 | 1 | 0 | 1 | 1 | $\overline{S1}\,\overline{S2}\,S3\,\overline{S4}$ |
| 0 | 1 | 1 | 0 | X | |
| 0 | 1 | 1 | 1 | 0 | |
| 1 | 0 | 0 | 0 | 1 | $\overline{S1}\,\overline{S2}\,\overline{S3}\,S4$ |
| 1 | 0 | 0 | 1 | 1 | $S1\,\overline{S2}\,\overline{S3}\,S4$ |
| 1 | 0 | 1 | 0 | 1 | $\overline{S1}\,S2\,\overline{S3}\,S4$ |
| 1 | 0 | 1 | 1 | 0 | |
| 1 | 1 | 0 | 0 | 1 | $\overline{S1}\,\overline{S2}\,S3\,S4$ |
| 1 | 1 | 0 | 1 | 0 | |
| 1 | 1 | 1 | 0 | 0 | |
| 1 | 1 | 1 | 1 | 0 | |




$$X = \overline{S1}\,\overline{S2} + \overline{S1}\,\overline{S3} + \overline{S3}\,\overline{S4} + \overline{S2}\,\overline{S3} + \overline{S2}\,\overline{S4}$$

**4.18** $z = \overline{A}B\overline{D} + \overline{B}C$




**4.19** In Example 4.3 of your textbook, after the DeMorgan part is completed, we have:

$$z = \overline{A}\,\overline{B}C + \overline{A}C\overline{D} + \overline{A}BC\overline{D} + A\overline{B}C$$

$$z = \overline{A}\,\overline{B}C + \overline{A}C\overline{D}(B + \overline{B}) + \overline{A}BC\overline{D} + A\overline{B}C$$

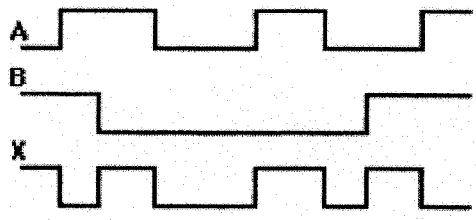
$$z = \overline{A}\,\overline{B}C + \overline{A}BC\overline{D} + \overline{A}\,\overline{B}C\overline{D} + \overline{A}BC\overline{D} + A\overline{B}C$$

$$z = \overline{A}\,\overline{B}C + \overline{A}BC\overline{D} + \overline{A}\,\overline{B}C\overline{D} + \overline{A}BC\overline{D} + A\overline{B}C$$

$$z = \overline{B}C(\overline{A} + A + \overline{A}\,\overline{D}) + \overline{A}B\overline{D}(C + \overline{C})$$

$$z = \overline{B}C + \overline{A}B\overline{D}$$

**4.20** (a) Output X will be HIGH only when A and B are at different levels.



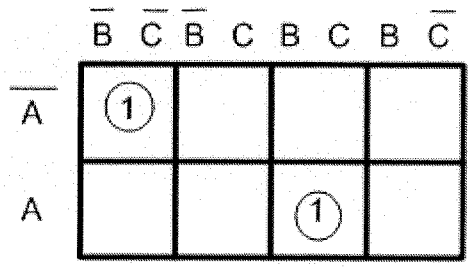

(b) With B held LOW, X=A.


(c) With B held HIGH, $X = \overline{A}$.


**4.21** X will be HIGH when A≠B, B=C, and C=1. Thus, C=1, B=1, A=0 is the only input condition that produces X=1.

**4.22 (a)** $X = ABC + \overline{A}\overline{B}\overline{C}$

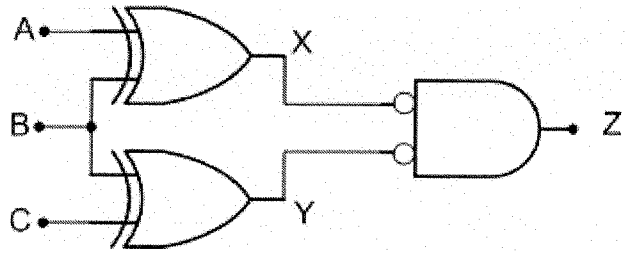
| C | B | A | X |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |



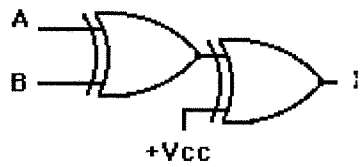

**(b)** To find if A=B=C:

1. $X = A \oplus B$ (X is Low when A=B)
2. $Y = B \oplus C$ (Y is Low when B=C)

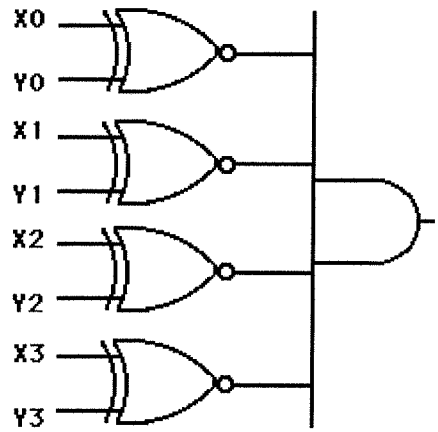
A=B=C when both 1 & 2 are true.



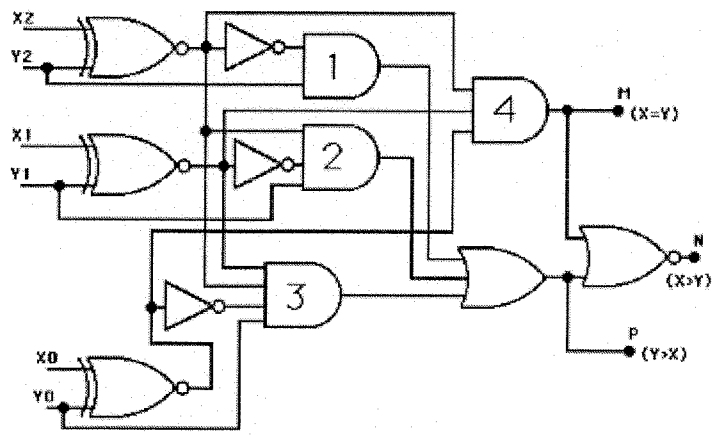

**4.23**




**4.24**




**4.25** One possibility is on the next page. Note the use of the XNOR gates and AND gate 4 to determine when the two numbers are equal; that is, when X2=Y2, X1=Y1 and X0=Y0 simultaneously. AND gates 1,2,3 and the OR gate are used to sense when Y2 Y1 Y0 > X2 X1 X0. The NOR gate simply uses the fact that if neither M nor P is HIGH then it must be true that X2 X1 X0 > Y2 Y1 Y0, and therefore N=1.



**4.26**

| INPUTS | | | | OUTPUTS | | | |
|---|---|---|---|---|---|---|---|
| Y1 | Y0 | X1 | X0 | Z3 | Z2 | Z1 | Z0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |

Four outputs are required since the largest product will be 1001(nine) for the case where Y1Y0=11(three), and X1X0=11(three). Z0 is the LSB of the output.

Since there are four separate outputs, then four separate circuits must be designed; one for each output.

**Output Z3:** Z3 = 1 only for single case in the T-T. Thus, **Z3 = Y1 Y0 X1 X0**

**Output Z2:** Z2 is HIGH for three cases. Thus,

$$Z2 = Y1\overline{Y0}X1\overline{X0} + Y1\overline{Y0}X1X0 + Y1Y0X1\overline{X0}$$

$$Z2 = Y1X1(\overline{Y0} + \overline{X0}) = Y1X1(\overline{Y0X0})$$

**Output Z1:** Z1 is HIGH for six cases. Thus,

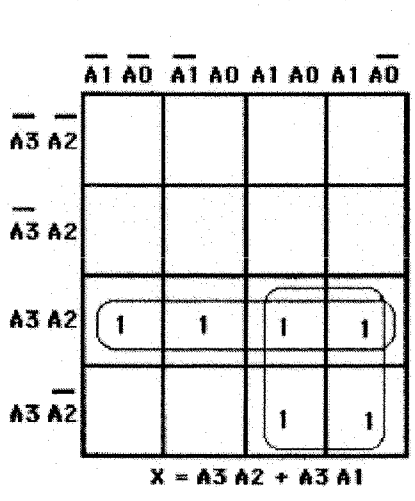
$$Z1 = \overline{Y1}Y0X1\overline{X0} + \overline{Y1}Y0X1X0 + Y1\overline{Y0}\,\overline{X1}X0 + Y1\overline{Y0}X1X0 + Y1Y0\overline{X1}X0 + Y1Y0X1\overline{X0}$$

$$Z1 = Y0X1(\overline{Y1} + \overline{X0}) + Y1X0(\overline{Y0} + \overline{X1})$$

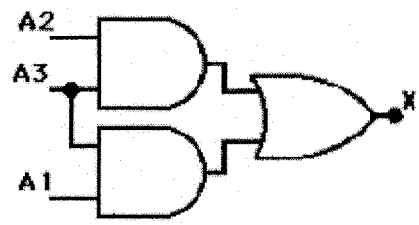
**Output Z0:** Z0 is HIGH for four cases.

$Z0 = \overline{Y1}Y0\overline{X1}X0 + \overline{Y1}Y0X1X0 + Y1Y0\overline{X1}X0 + Y1Y0X1X0$ . Thus, $Z0 = Y0X0$

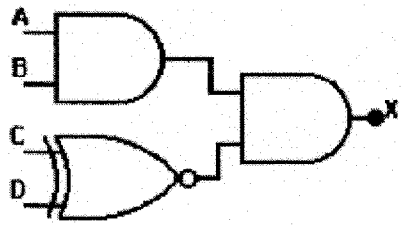
**4.27**



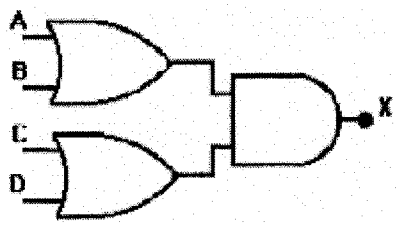

| A3 | A2 | A1 | A0 | X |
|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |
| (all other cases) | | | | 0 |



**4.28**




**4.29**




**4.30**

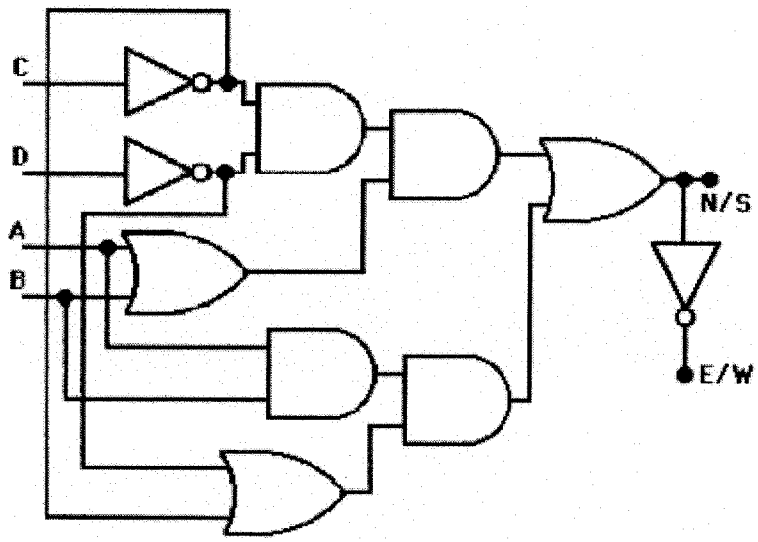
| D | C | B | A | E/W | N/S | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | |
| 0 | 0 | 0 | 1 | 1 | 0 | |
| 0 | 0 | 1 | 0 | 1 | 0 | |
| 0 | 0 | 1 | 1 | 1 | 0 | |
| 0 | 1 | 0 | 0 | 0 | 1 | $\overline{A}B\overline{C}\,\overline{D}$ |
| 0 | 1 | 0 | 1 | 1 | 0 | |
| 0 | 1 | 1 | 0 | 1 | 0 | |
| 0 | 1 | 1 | 1 | 1 | 0 | |
| 1 | 0 | 0 | 0 | 0 | 1 | $A\overline{B}\,\overline{C}\,\overline{D}$ |
| 1 | 0 | 0 | 1 | 1 | 0 | |
| 1 | 0 | 1 | 0 | 1 | 0 | |
| 1 | 0 | 1 | 1 | 1 | 0 | |
| 1 | 1 | 0 | 0 | 0 | 1 | $AB\overline{C}\,\overline{D}$ |
| 1 | 1 | 0 | 1 | 0 | 1 | $AB\overline{C}D$ |
| 1 | 1 | 1 | 0 | 0 | 1 | $ABC\overline{D}$ |
| 1 | 1 | 1 | 1 | 1 | 0 | |

Since there are only five cases when N/S=1, we will design for N/S.

$$N/S = \overline{A}BC\overline{D} + A\overline{B}\,\overline{C}\,\overline{D} + AB\overline{C}\,\overline{D} + AB\overline{C}D + ABC\overline{D}$$

This can be simplified to: $N/S = \overline{C}\,\overline{D}(A + B) + AB(\overline{C} + \overline{D})$

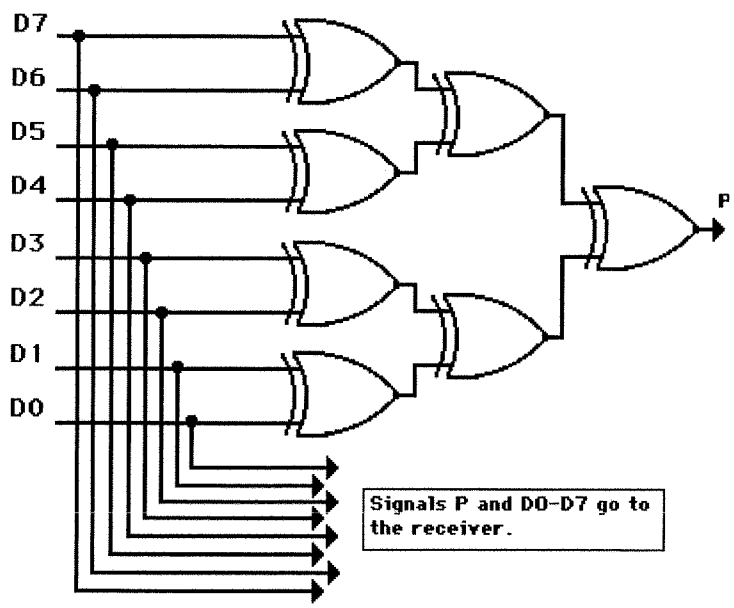
Obviously, $E/W = \overline{N/S}$



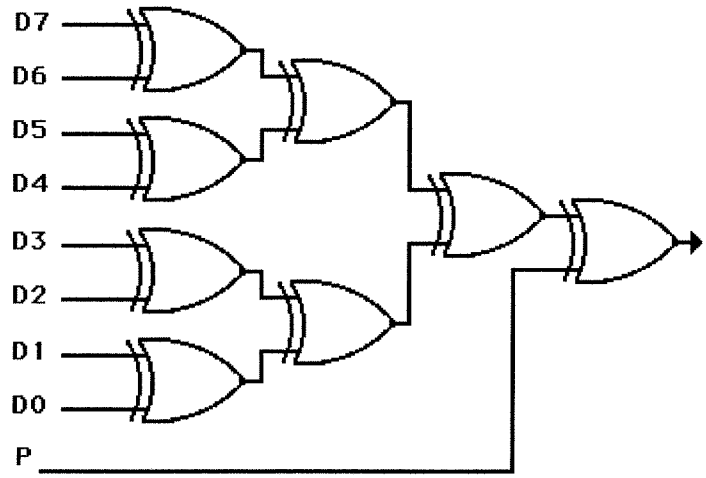

**4.31 (a)** **Parity Generator:** To modify the circuit of figure 4-25 (a) to an "Odd Parity Generator" all that is needed is an inverter at the output.

**Odd Parity Checker:** To modify the circuit of figure 4-25 (b) to an "Odd Parity Checker" the 2-input exclusive-OR gates should be changed to 2-input exclusive-NOR gates.

**(b)** **Even Parity Generator**



**Even Parity Checker**




**4.32** (a) When all of the other inputs to the OR gate are in the LOW state the logic signal will pass through to its output unchanged.

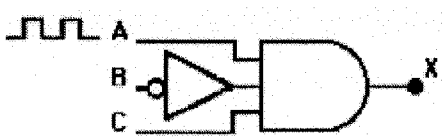
(b) When all of the other inputs to the AND gate are in the HIGH state the logic signal will pass through to its output unchanged.

(c) When all of the other inputs to the NAND gate are in the HIGH state the logic signal will pass through to its output INVERTED.

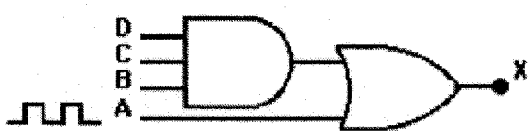
(d) When all of the other inputs to the NOR gate are in the LOW state the logic signal will pass through to its output INVERTED.

**4.33** (a) No. A logic circuit must have two inputs in order to be used as an enable/disable circuit.
(b) No. The control input of an XOR gate can be either HIGH or LOW. If the control input is LOW the signal at the other input reaches the gate's output unaffected. If the control input is HIGH the signal at the other input reaches the gate's output INVERTED.

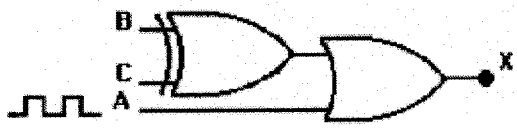
**4.34** Use an AND gate that is enabled when B=0, C=1. X=A only if B=0, C=1



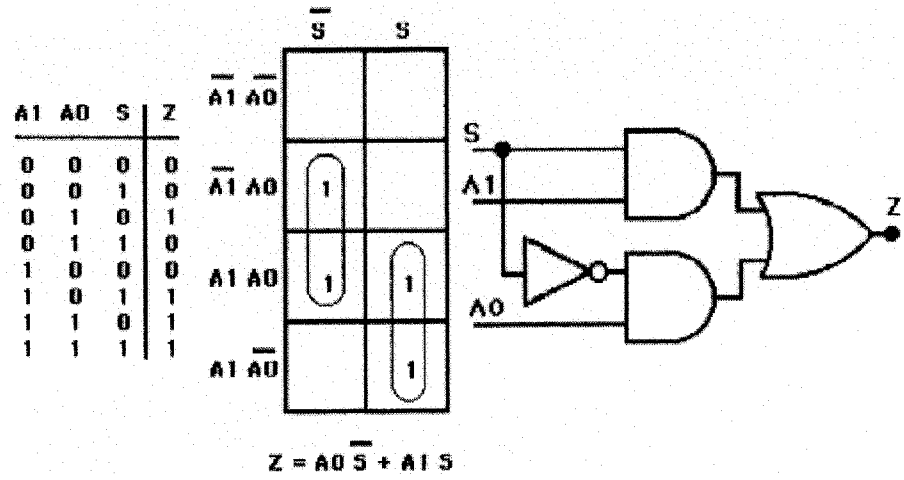

**4.35** Use an OR gate since output is to be HIGH when inhibited. X=A only if BCD ≠ 1. X=1 when BCD = 1



**4.36** X=A when B=C. X=1 when B≠C

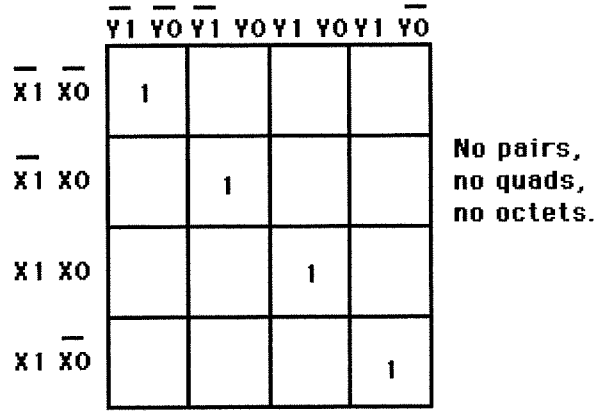



**4.37**




$Z = A0\,\bar{S} + A1\,S$

**4.38**



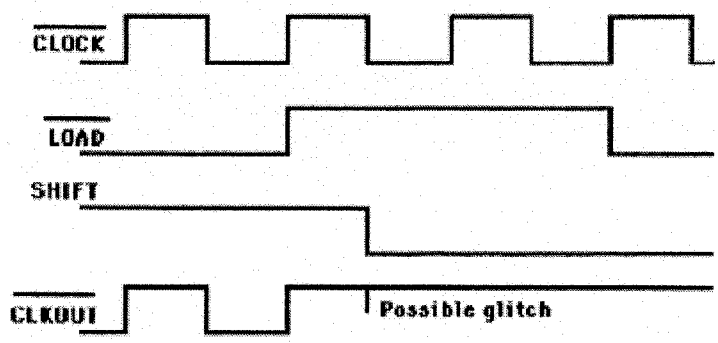

No pairs,
no quads,
no octets.

$$Z = \bar{X}1\,\bar{X}0\,\bar{Y}1\,\bar{Y}0 + \bar{X}1\,X0\,\bar{Y}1\,Y0 + X1\,X0\,Y1\,Y0 + X1\,\bar{X}0\,Y1\,\bar{Y}$$

**4.39** (a) 1. The output of the inverter is internally grounded.
2. The output of the inverter is externally grounded.
3. The input being driven by the output of the inverter is internally grounded.

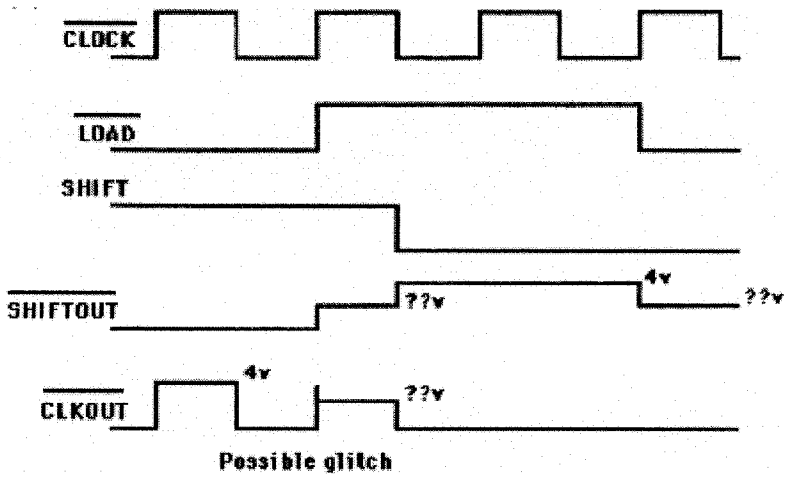
(b) The output of the inverter is shorted to the output of another logic circuit.

**4.40** (a) Since Z1-4 is essentially floating, the Logic Probe will show an indeterminate logic level.
(b) There will be 1.4V-1.8V at the output.




Terminal Z2-9 will be floating (HIGH in TTL) since Z1-4 is opened internally. Thus, the signal at Z2-8 is the opposite of the signal at Z2-10.

(d)




**4.41** IC Z2-2 will be floating and therefore its voltage will fluctuate as it picks up noise. Thus, Z2-3 level will be unpredictable. IC-Z2 may also become overheated and eventually destroy itself.

**4.42** 1) First isolate Z1-4 from Z2-1 by using one of the following methods:

(a) cutting the trace from Z1-4 to Z2-1.
(b) clipping pin 4 of Z1.
(c) clipping pin 1 of Z2.

2) Check to see if Z1-4 is pulsing. If it is, then one can be sure that the inverter Z1 is working properly. If it's always LOW (internally shorted to ground) then inverter Z1 must be replaced.

3) If step 2 above proves IC Z1 to be working properly then the problem must be with NAND gate Z2 (internally shorted to ground). By using a logic probe, check the logic level at Z2-1. Chances are that it will have a permanent logic LOW which kept Z1-4 LOW and Z2-3 HIGH. Replace Z2.

**4.43** 1) Faulty IC bias (Vcc and/or Ground).
2) Z2-2 is internally open (floating).
3) Z2-1 is internally open (floating).
4) Z2-3 is internally open (floating).
Procedure: With a VOM or logic probe, check Vcc and Ground to the IC. If the Vcc and Ground measurements are correct, disconnect Z2-3 from any load it may be driving. If problem persists, replace Z2.

**4.44** Yes. (c), (e), (f).

(a) No. This would've kept point X at a logic LOW permanently and the first case (A=1, B=0) wouldn't have worked.

(b) No. An open at Z2-13 has the same effect as a logic HIGH (only in TTL). Thus, in the second case (A=0,B=1,C=1) Z2-11 would've been LOW and Z2-8 HIGH.

(d) No. This would've cause IC Z2 to be unbiased and prevent the circuit from working properly for the first case.

(g) No. This would've caused Z2-10 to be always LOW and Z2-8 HIGH for all cases.

**4.45** 1) Make A=0 (Z1-1), B=1 (Z1-2) and C=1 (Z2-12). This is the case that causes the circuit to malfunction. Note that the other three possible combinations of A and B do not cause a problem. We know that IC Z1 is working from the results of the first case.

2) The logic levels at Z2-13 and Z2-12 should be HIGH.

(a) Check to see if Z2-11 has a logic LOW.

(b) If Z2-11 is LOW and Z2-9 isn't turn off the power to the circuit.

(c) Use a VOM to make a continuity check between Z2-11 and Z2-9. If there is an open, find it and restore the continuity between these two points.

3) If after performing step two the technician finds that there is a good connection between Z2-11 and Z2-9, then one could conclude that either output Z2-11 or input Z2-9 is externally shorted to Vcc. Since the circuit still has the power turned off from the last check, the technician should make a continuity check to see if the trace between Z2-11 and Z2-9 is externally shorted to Vcc. If there is a short to Vcc, find it and eliminate it. If no external short to Vcc is found then either Z2-11 or Z2-9 or both must be internally short to Vcc or have an internal open. In any case the replacement of IC Z2 should be performed.

**4.46** This is a tough one. You have noticed that Z2-6 and Z2-11 will be at the same logic level except for the two cases that don't work. For those cases, Z2-6 and Z2-11 are supposed to be different. Since they measure indeterminate for those cases, it is likely that Z2-6 and Z2-11 are shorted together, probably by a solder bridge. The short will have no effect for all those cases where these two outputs are at the same level.

**4.47** (b) If Z1-2 was internally shorted to ground, whenever the passenger failed to fastened his/her seat-belt the circuit would've not detected this ALARM condition.
(c) Since this is a TTL logic circuit, if there was an open connection between Z2-6 and Z2-10, the circuit would've operated as if a logic HIGH was present at Z2-10. This would've caused the circuit to ALWAYS assume that a passenger was in the seat with the respective seat-belt fastened.

**4.48** Since the problem only manifests itself when an occupant is present in the car and the ignition is turned on, it can be deduced that IC Z2 is working properly. The problem must be with IC Z1. The following are the possible circuit failures:

(a) IC Z1 is not properly biased. } Most likely
(b) IC Z1 is plugged in backwards. } problems.

**Remote possibilities:**

(c) Z1-4 and Z1-2 are internally shorted to Vcc.
(d) Z1-4 and Z1-2 are internally open.
(e) An open connection from Z1-2 to Z2-5, and from Z1-4 to Z2-2.
(f) Connection from Z1-2 to Z2-5 is externally shorted to Vcc as well as the connection from Z1-4 to Z2-2.
(g) Z1-1 and Z1-3 are internally shorted to Ground.

**Procedure:**

1) Make the necessary voltage measurements to confirm proper IC Z1 bias. Check for proper IC Z1 orientation.

2) Check the logic levels at Z1-2 and Z1-4 with a logic probe. If IC Z1 is working properly then a TTL logic LOW should be present at these points.

3) If these logic levels are still HIGH, by using an ohmmeter check for any external shorts to Vcc or open PC traces.

4) Check the logic levels at Z1-1 and Z1-3 with a logic probe. If IC Z1 is to work properly then a TTL logic HIGH should be present at these points.

5) If these logic levels are LOW, use an ohmmeter to check for any external shorts to Ground.

6) If the above steps do not reveal a probable cause, Z1 must be internally damaged and it must be replaced.

**4.49** For some reason Z2-13 is always HIGH. The following are the possible circuit failures:

(a) Z2-13 is internally shorted to Vcc.

(b) Z2-8 is internally shorted to Vcc.

(c) Connection from Z2-8 to Z2-13 is open or externally shorted to Vcc.

(d) Z2-9 or Z2-10 are internally shorted to Ground.

(e) Z2-3 or Z2-6 are internally shorted to Ground.

(f) Connections from Z2-3 to Z2-9 or from Z2-6 to Z2-10 are externally shorted to Ground.

**Procedure:**

The first troubleshooting step is to make sure that all of the ICs are properly biased (Vcc and Ground) and oriented.

I) Isolate Z2-13 from Z2-8 by cutting the trace on the PC board or by clipping the proper pin on IC Z2 (either pin 8 or pin 13). Check the voltage level at Z2-13 with a VOM. It should be about 0v since it's floating at this point. If the voltage is ≈Vcc, Z2-13 is either internally or externally shorted to Vcc and it should be replaced.

II) If a fault is not found after performing step I, then check the logic level at Z2-8 with a logic probe. If it's HIGH, check the logic levels at Z2-9 and Z2-10. One of them or both should be LOW. If they are both HIGH, IC Z2-8 is internally or externally shorted to Vcc.

III) ***If Z2-9 is LOW***

Check the logic levels at Z2-1 and Z2-2. They should be both LOW. If they are LOW, isolate Z2-3 from Z2-9 by cutting the trace on the PC board or by clipping the appropriate pin (Z2-3 or Z2-9). Check the logic levels at Z2-3 and Z2-9 with a logic probe. If either input is LOW, one must conclude that IC Z2 pin 3 or pin 9 is externally or internally shorted to ground.

IV) If Z2-10 is LOW, the same test procedure should be used for the connection between Z2-10 and Z2-6.

**4.50** (a) True; (b) True; (c) False; (d) False; (e) True

**4.51** All text between the characters **%** **%** serves as comments.

**4.52** Comments in a VHDL design file are indicated by --.

**4.53** A special socket that allows you to drop the chip in and then clamp the contacts onto the pins.

**4.54** 1) Boolean equation; 2) Truth table; 3) Schematic diagram

**4.55** *JEDEC* - Joint Electronic Device Engineering Council; *HDL* - Hardware Description Language

**4.56**

```
(a) AHDL: gadgets[7..0]   :OUTPUT;
    VHDL  gadgets         :OUT BIT_VECTOR (7 DOWNTO 0);

(b) AHDL  buzzer          :OUTPUT;
    VHDL  buzzer          :OUT BIT;

(c) AHDL: altitude[15..0] :INPUT;
    VHDL  altitude        :IN INTEGER RANGE 0 TO 65535);

(d) AHDL  VARIABLE
          wire2           :NODE;
    VHDL  SIGNAL wire2    :BIT ;
```

**4.57**

```
(a) AHDL  H"98"  B"10011000"  152
    VHDL  X"98"  B"10011000"  152

(b) AHDL  H"254" B"1001010100"  596
    VHDL  X"254" B"1001010100"  596

(c) AHDL  H"3C4" B"1111000100"  964
    VHDL  X"3C4" B"1111000100"  964
```

**4.58**

```
SUBDESIGN hw
(
        inbits[3..0]        :INPUT;
        outbits[3..0]       :OUTPUT;
)
ENTITY hw IS
Port    (
        inbits              :IN BIT_VECTOR (3 downto 0);
        outbits             :OUT BIT_VECTOR (3 downto 0)
        );
END hw;

AHDL                    outbits[3]      =       inbits[1];
                        outbits[2]      =       inbits[3];
                        outbits[1]      =       inbits[0];
                        outbits[0]      =       inbits[2];

VHDL                    outbits(3)      <=      inbits(1);
                        outbits(2)      <=      inbits(3);
                        outbits(1)      <=      inbits(0);
                        outbits(0)      <=      inbits(2);
```

**4.59**

```
TABLE
                (a,b,c)         =>      y;
                (0,0,0)         =>      0;
                (0,0,1)         =>      0;
                (0,1,0)         =>      1;
                (0,1,1)         =>      1;
                (1,0,0)         =>      1;
                (1,0,1)         =>      0;
                (1,1,0)         =>      1;
                (1,1,1)         =>      1;
END TABLE;
```

**4.60**

```
BEGIN
        IF digital_value[] < 10 THEN
                z = VCC;                        --output a 1
        ELSE z = GND;                           --output a 0
        END IF;
END;
```

**4.61**

```
WITH in_bits SELECT
        y       <=      '0'  WHEN "000",
                        '0'  WHEN "001",
                        '1'  WHEN "010",
                        '1'  WHEN "011",
                        '1'  WHEN "100",
                        '0'  WHEN "101",
                        '1'  WHEN "110",
                        '1'  WHEN "111";
```

**4.62**

```
        PROCESS (digital_value)
        BEGIN
                IF (digital_value < 10) THEN z <= '1';
                ELSE z <= '0';
                END IF;
END PROCESS;
```

**4.63**

```
        %   Problem 4-63 in AHDL
        Digital Systems 10th ed
        Neal Widmer
                %
SUBDESIGN PROB4_63
(
        digital_value[3..0]     :INPUT;                 --define inputs to block
        y                       :OUTPUT;                --define block output
)

BEGIN
        IF digital_value[] > 5 & digital_value[] < 12 THEN
            y = vcc;                                    --output a 1
        ELSE y = gnd;                                   --output a 0
        END IF;
END;

-- NOTE: The digital_value[0] term drops out when this is simplified.
-- The compiler will issue a warning to this effect.
```

**4.63** (in VHDL)

```
-- USING PROCESS.
-- Digital Systems 10th ed
-- Tocci Widmer Moss

ENTITY prob4_63 IS
PORT(  digital_value        :IN INTEGER RANGE 0 TO 15;      --declare 4-bit input
            z               :OUT BIT);
END fig4_55;
```

```
ARCHITECTURE truth OF fig4_55 IS
BEGIN
        PROCESS (digital_value)
            BEGIN
                    IF (digital_value > 5) AND digit_value < 12) THEN
                            z <= '1';
                    ELSE
                            z <= '0';
                    END IF;
        END PROCESS ;
END truth;

-- NOTE: The digital_value[0] term drops out when this is simplified.
-- The compiler will issue a warning to this effect.
```

**4.64 (a)**

```
SUBDESIGN fig4_60
(
  a, b, c           :INPUT;                 --define inputs to block
  y                 :OUTPUT;                --define outputs
)
VARIABLE
status[2..0]        :NODE;                  --holds state of cold, moderate, hot
BEGIN
  status[]= (a, b, c);                      --link input bits in order
  CASE status[] IS
          WHEN b"010"           =>      y = VCC;
          WHEN b"011"           =>      y = VCC;
          WHEN b"111"           =>      y = VCC;
          WHEN OTHERS                   =>      y = GND;
  END CASE;
END;
```

**4.64 (b)**

```
ENTITY fig4_61 IS
port(
        a, b, c         :IN bit;        --declare 3 bits input
        y               :OUT BIT);
END fig4_61;

ARCHITECTURE copy OF fig4_61 IS
SIGNAL status           :BIT_VECTOR (2 downto 0);
BEGIN
        status <= a & b & c;            --link bits in order.
        PROCESS (status)
                BEGIN
                        CASE status IS
                                WHEN "010" => y <= '1';
                                WHEN "011" => y <= '1';
                                WHEN "111" => y <= '1';
                                WHEN OTHERS =>      y <= '0';
                        END CASE;
                END PROCESS ;
END copy;
```

**4.65** S=!P#(Q&R)

**4.66** $P = D_3\$D_2\$D_0\$D_1$

**4.67** (a) Two-dimensional form of a truth table used to simplify a sum-of-products expression.

(b) Logic expression consisting of two or more AND terms (products) that are ORed together.

(c) Logic circuit that produces an even or odd parity bit for a given set of input data bits.

(d) Group of eight 1s that are adjacent to each other within a Karnaugh map.

(e) Logic circuit that controls the passage of an input signal through to the output.

(f) Situation when a circuit's output level for a given set of input conditions can be assigned as either a 1 or 0.

(g) Input signal that is left disconnected in a logic circuit.

(h) Whenever a logic voltage level of a particular logic family falls out of the required range of voltages for either a logic 0 or logic 1.

(i) Signal contention is when two signals are "fighting" each other.

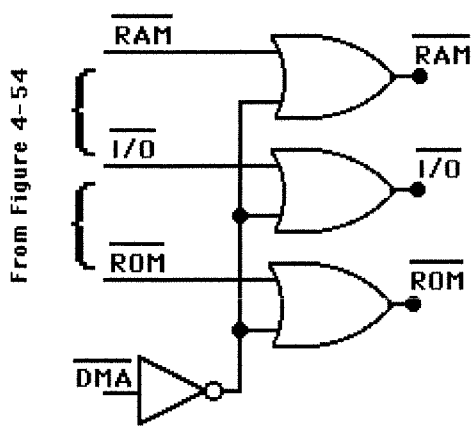
(j) Programmable Logic Device

(k) The TTL (Transistor-Transistor-Logic) family is the major family of bipolar digital ICs.

(l) The CMOS (Complementary Metal Oxide Semiconductor) family belongs to the class of unipolar digital ICs.

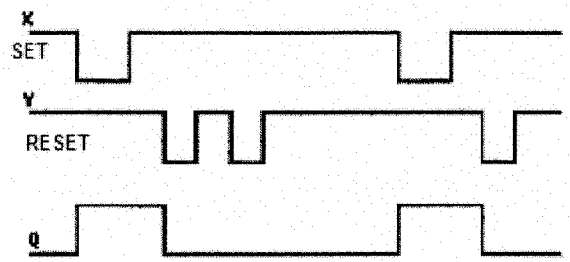
**4.68** RAM } $00000000_2$ - $11101111_2$ = $\mathbf{00_{16}}$ - $\mathbf{EF_{16}}$
I/O } $11110000_2$ = $\mathbf{F0_{16}}$
ROM } $11110001_2$ - $11111111_2$ = $\mathbf{F1_{16}}$ - $\mathbf{FF_{16}}$

**4.69**



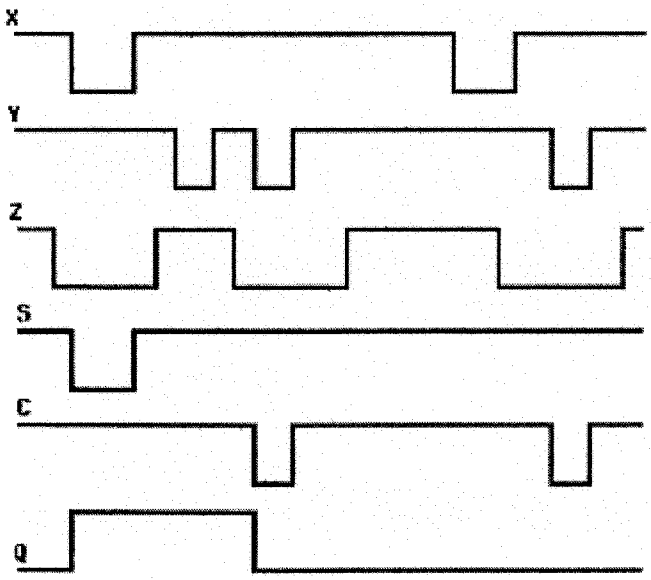
## CHAPTER FIVE - Flip-Flops and Related Devices

**5.1**

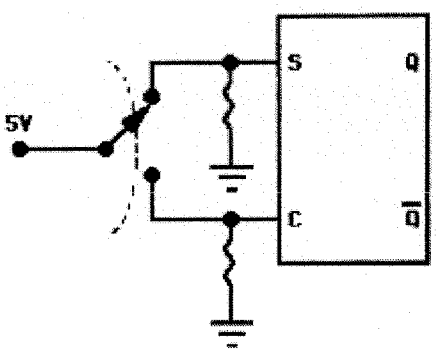



**5.2** Same Q output as 5.1.

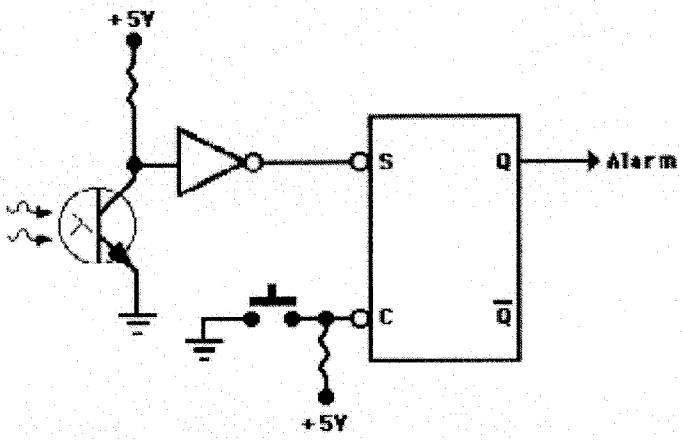
**5.3**




**5.4**



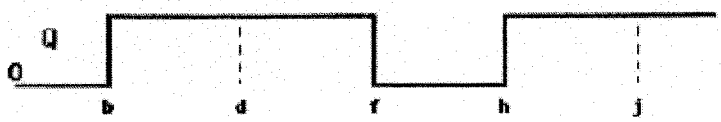
**5.5** One possibility:



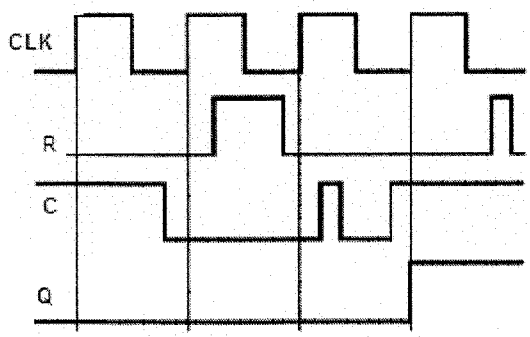

**5.6** The response shown would occur If the NAND latch is not working as a Flip-Flop. A permanent logic HIGH at IC Z1-4 will prevent the latch from working properly and therefore the switch bounce will appear at Z1-6. When the 1 KHz squarewave is high, the switch bounce will be present at Z2-6.

**5.7** Control inputs have to be stable for $t_S$=20ns prior to the clock transition.

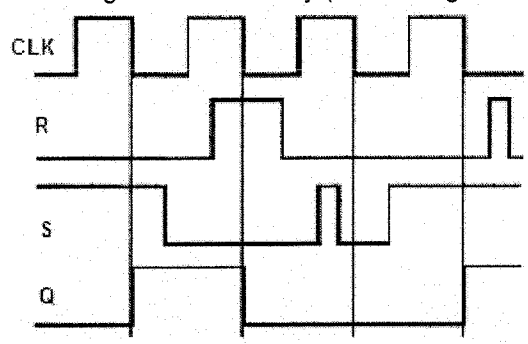
**5.8** The FF will respond at times b, d, f, h, j corresponding to negative-going CLK transitions.



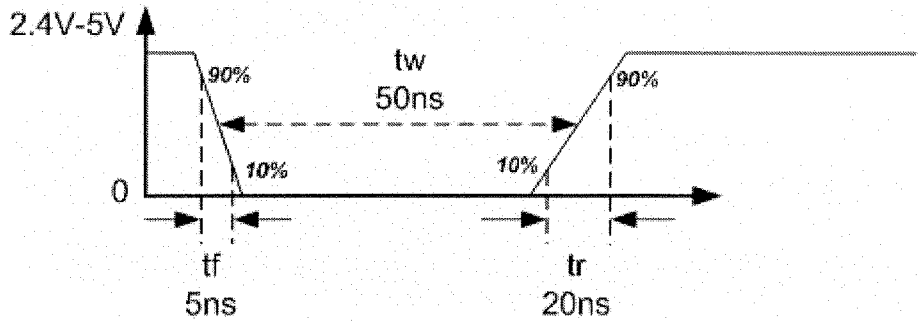

**5.9** Assuming that Q=0 initially (for the positive edge triggered S-C FF).



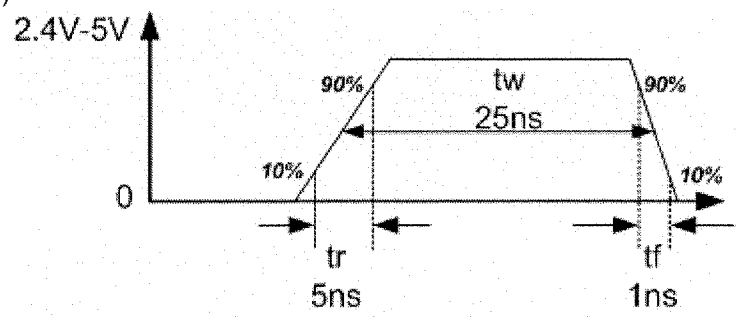

Assuming that Q=0 initially (for the negative edge triggered S-C FF).



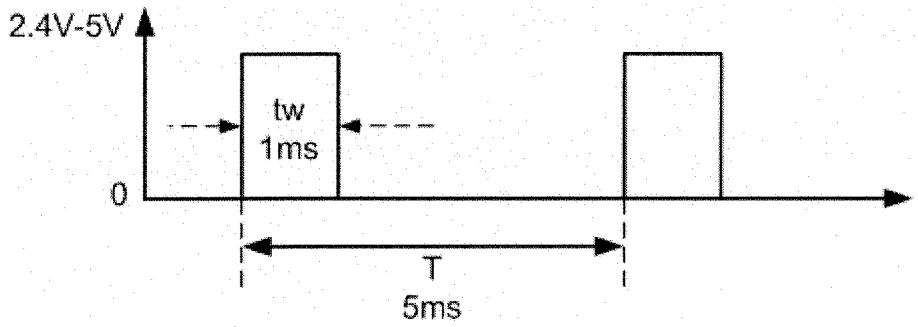
**5.10** (a)



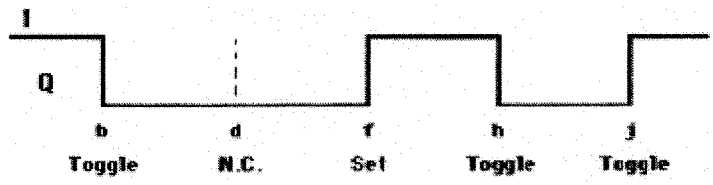

(b)




(c)



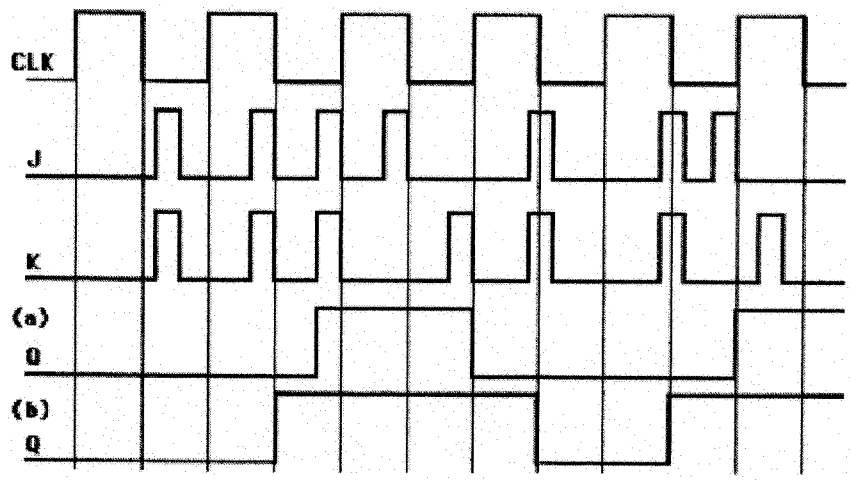

**5.11** FF can change state only at points b, d, f, h, j based on values of J and K inputs.



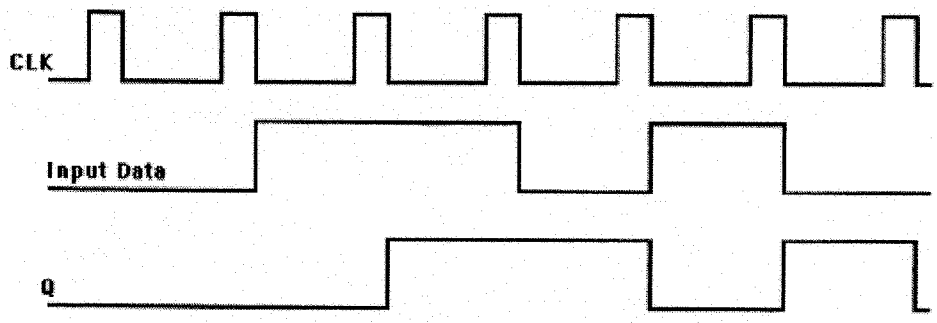

**5.12** (a) Connect the J and K inputs permanently HIGH. The Q output will be a squarewave with a frequency of 5 KHz.

(b) The Q output will be a squarewave with a frequency of 2.5 KHz.

**5.13**

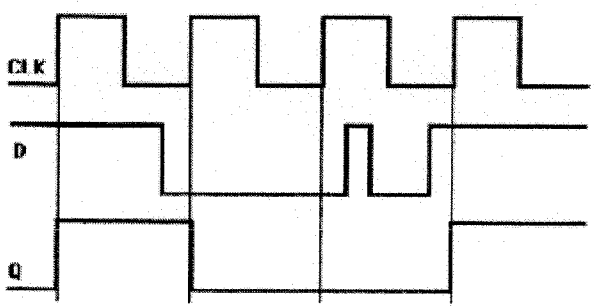



**5.14** (a) Since the FF has $t_H$=0, the FF will respond to the value present on the D input just prior to the NGT of the clock.

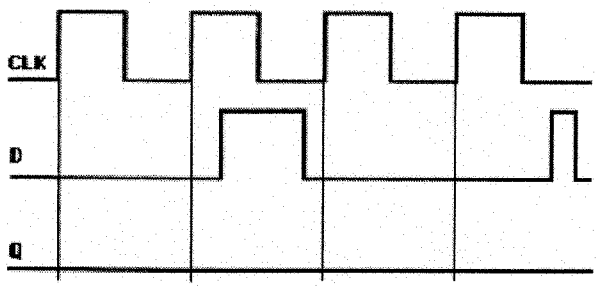



(b) Connect Q to the D input of a second FF, and connect the clock signal to the second FF. The output of the second FF will be delayed by 2 clock periods from the Input Data.

**5.15** (a)

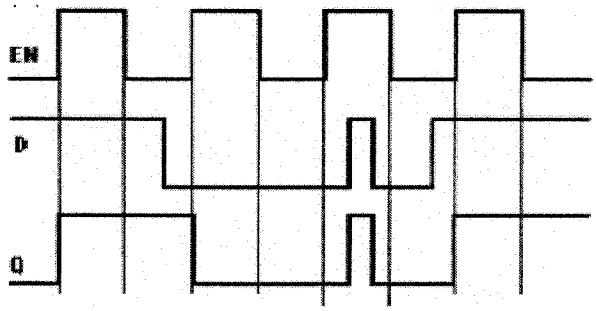


(b)

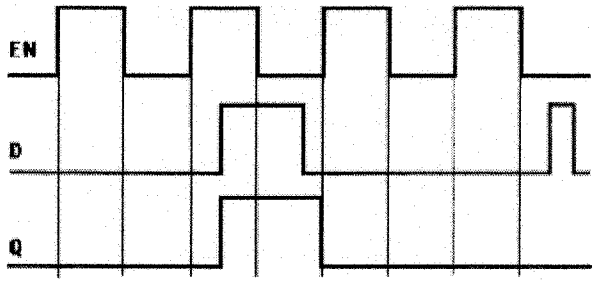



**5.16** Q is a 500 Hz square wave.

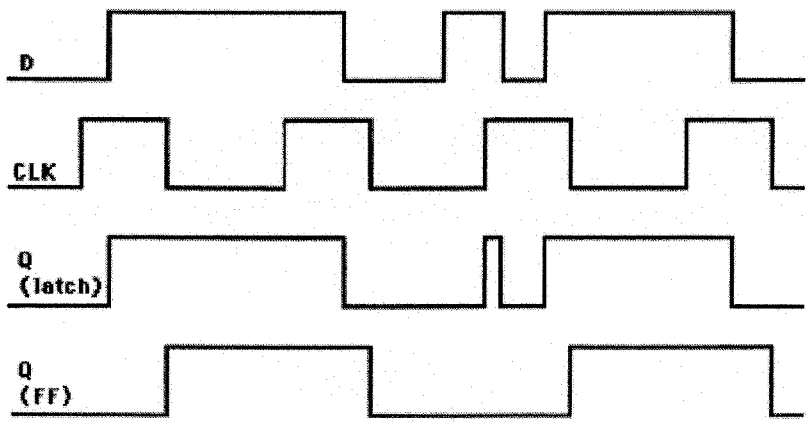
**5.17** (a)




(b)

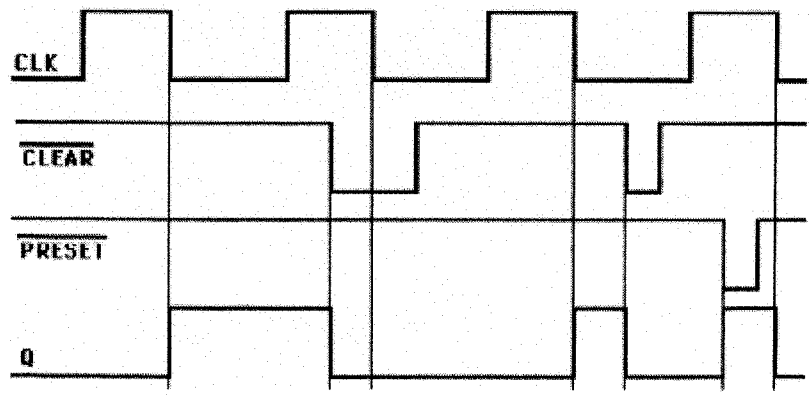



**5.18**

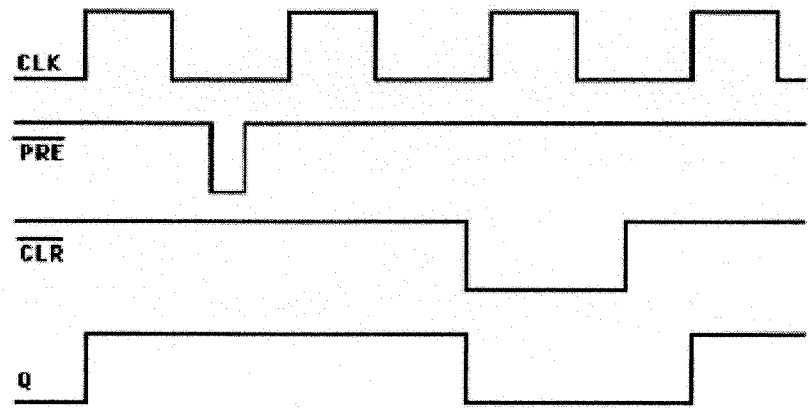


**5.19** If $\overline{Q}$ is connected back to D, the Q and $\overline{Q}$ outputs will oscillate while CLK is HIGH. This is because $\overline{Q}$ =1 will produce S=0, C=1 which will make $\overline{Q}$ =0. This $\overline{Q}$ =0 then will make S=1, C=0 which will make $\overline{Q}$ =1.

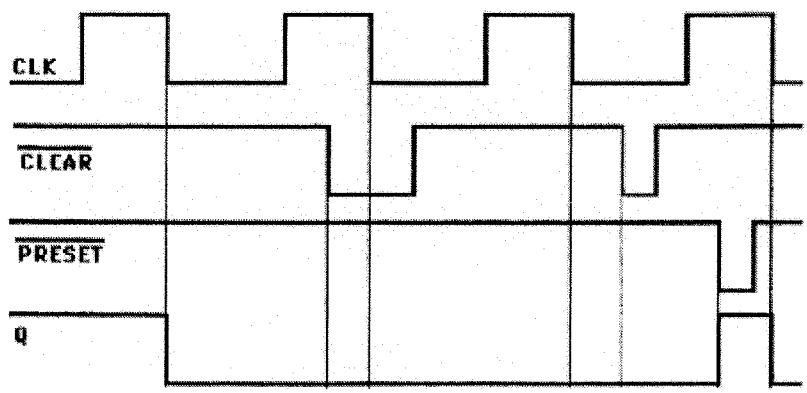
**5.20** J=K=1 so FF will toggle on each CLK negative-going edge, unless either $\overline{PRESET}$ or $\overline{CLEAR}$ inputs is LOW.




**5.21**



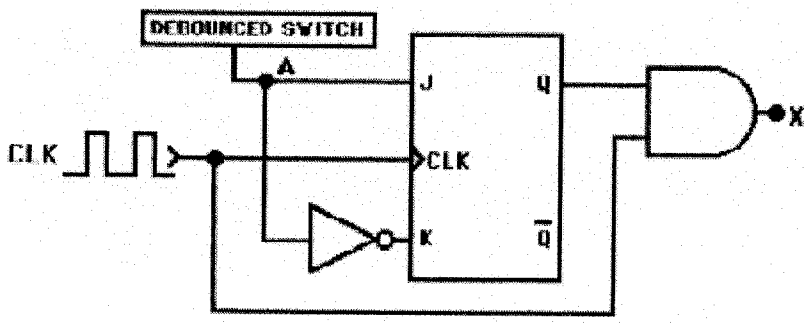

**5.22**



**5.23** (a) $t_{PLH}$ from CLK to Q is 200ns.
(b) With a $t_H$ = 5ns, the 7474 requires its control inputs to remain stable the longest time after the CLK transition.
With a $t_S$ = 60ns, the 74C74 requires its control inputs to remain stable the longest time before the CLK transition.
(c) $t_W(L)$ at $\overline{PRE}$ is 30ns.

**5.24** (a) $t_{PHL}$, CLR-Q = 24ns
(b) $t_{PLH}$, PRE-CLR-Q = 41ns
(c) $Tmin = \frac{1}{F\,max} = \frac{1}{15MHz} = 66.7ns$
(d) $tSU_{(min)}$ = 25ns. No. There is insufficient time.
(e) $t_{PLH}$ = 25ns CLR to Q

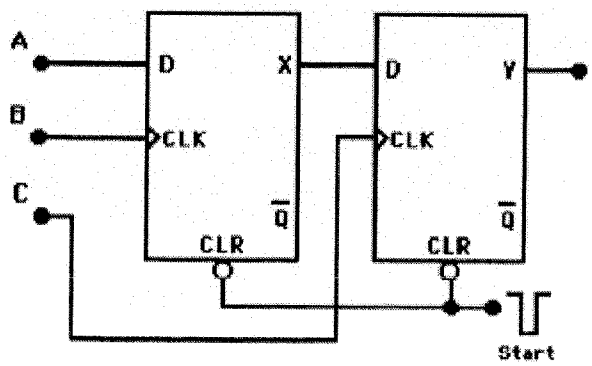
**5.25**



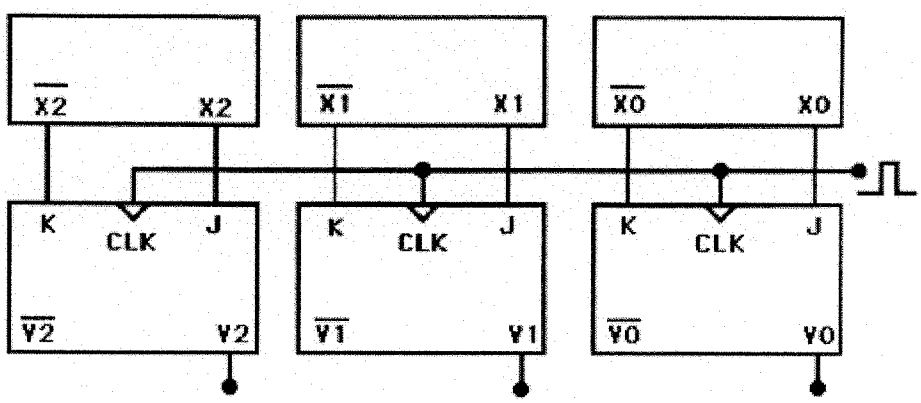

**5.26** (a) Y can go HIGH only when C goes HIGH while X is already HIGH. X can go HIGH only if B goes HIGH while A is HIGH. Thus, the correct sequence is A,B,C.

(b) The START pulse initially clears X and Y to 0 before applying the A,B,C signals.

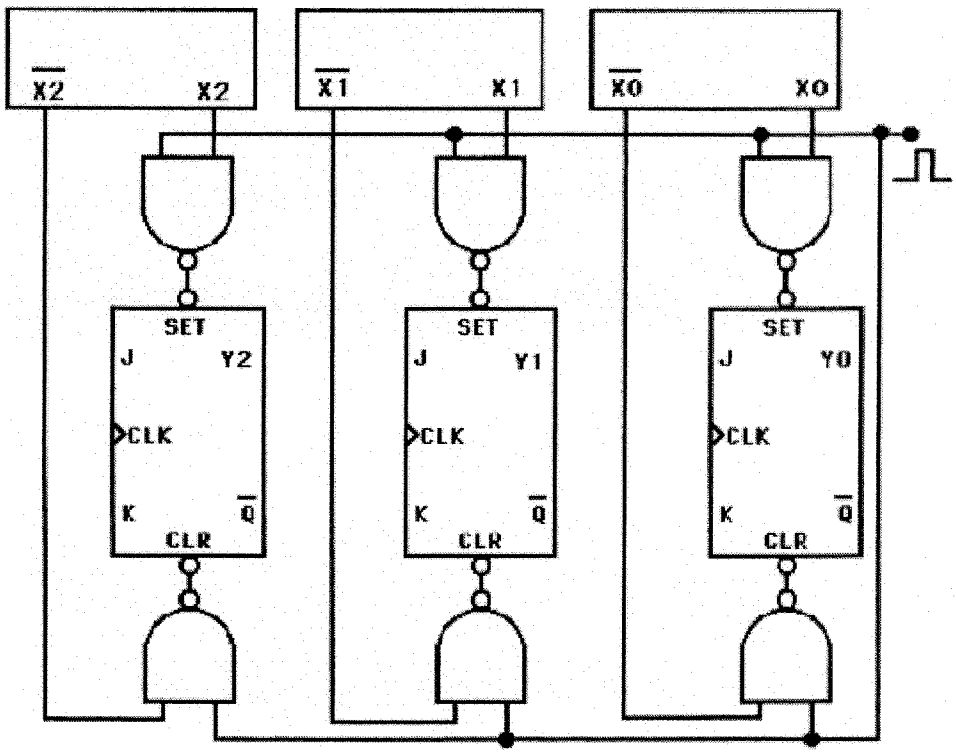
(c)



**5.27** (a)


X2
X2
X1
X1
X0
X0
K
CLK
J
K
CLK
J
K
CLK
J
Y2
Y2
Y1
Y1
Y0
Y0

(b)


X2
X2
X1
X1
X0
X0
SET
J
Y2
CLK
K
Q
CLR
SET
J
Y1
CLK
K
Q
CLR
SET
J
Y0
CLK
K
Q
CLR

**5.28** In this arrangement, the data shifts accordingly:


| X3 | X2 | X1 | X0 | |
|---|---|---|---|---|
| 1 | 0 | 1 | 1 | Initial State |
| 1 | 1 | 0 | 1 | Clock Pulse 1 |
| 1 | 1 | 1 | 0 | Clock Pulse 2 |
| 0 | 1 | 1 | 1 | Clock Pulse 3 |
| 1 | 0 | 1 | 1 | Clock Pulse 4 |
| 1 | 1 | 0 | 1 | Clock Pulse 5 |
| 1 | 1 | 1 | 0 | Clock Pulse 6 |
| 0 | 1 | 1 | 1 | Clock Pulse 7 |
| 1 | 0 | 1 | 1 | Clock Pulse 8 |

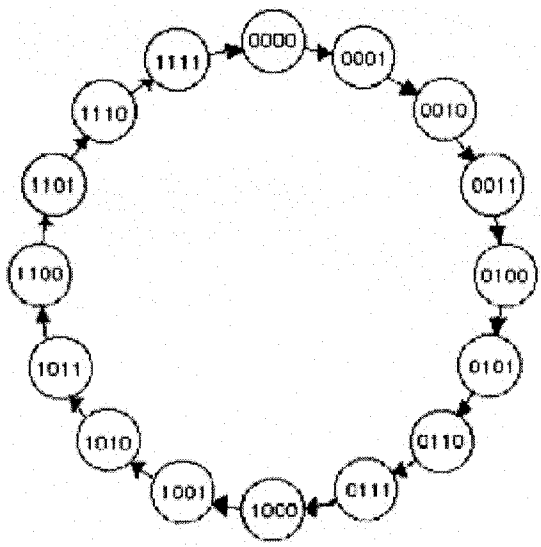
**5.29** Connect outputs X0 to D input of FF X2 so that the contents of the X register will be recirculated.

**5.30** This is a counter that will recycle every 8 pulses (MOD 8 counter).

(a) Count after 13 clock pulses is 5 (101); Count after 99 clock pulses is 3 (011); Count after 256 clock pulses is 0 (000).

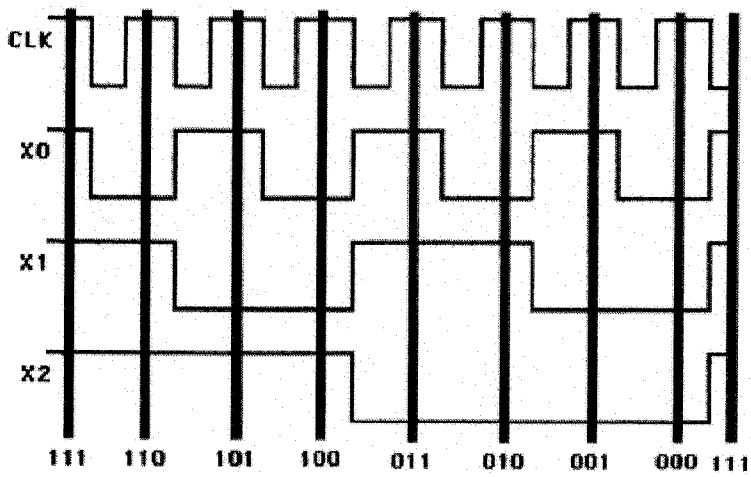
(b) Count after 13 clock pulses is 1 (001); Count after 99 clock pulses is 7 (111); Count after 256 clock pulses is 4 (100).

(c) State diagram for a MOD-16 counter

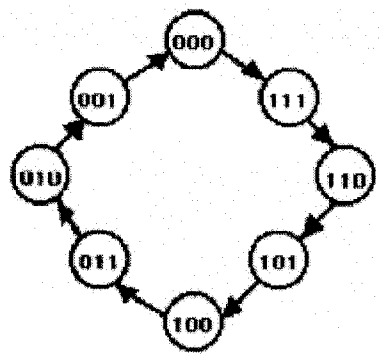



If the input frequency is 80 MHz the output waveform at X3 will be a squarewave with a frequency of 500 KHz (80 MHz/16).

**5.31**




**5.32**




**5.33** (a) $2^N-1=1023$, so that $2^N=1024$. Thus, N=10 flip-flops.

(b) With N FFs, the MOD-number is $2^N=1024$ so that the frequency division at the last FF will be 1/1024 relative to the input clock. Thus, output frequency = 2MHz/1024 = 1953 Hz.

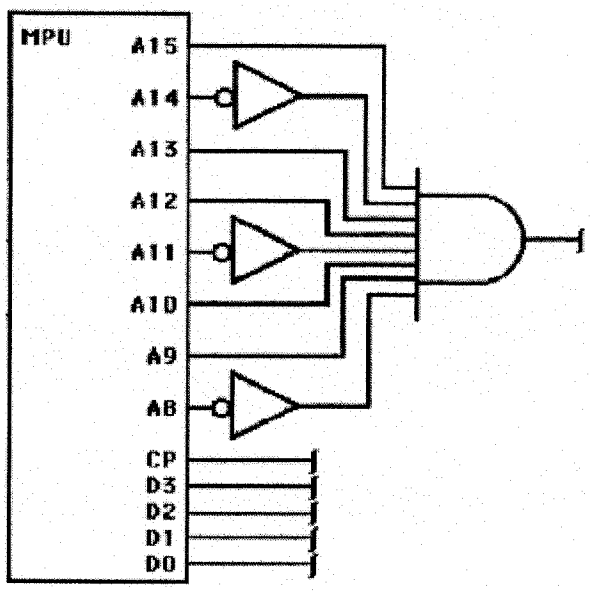
(c) MOD-number=$2^N=1024$.

(d) Every 1024 pulses the counter recycles through zero. Thus, after 2048 pulses the counter is back at count zero. Therefore, after 2060 pulses the counter will be at count 12 (i.e. 1024 + 1024 + 12 = 2060).

**5.34** (a) MOD-number = 256 KHz/2KHz = 128.
(b) $128=2^N$. The maximum count is $2^N-1=127$. Thus, the range is 0 to 127.

**5.35** The counter recycled back to 00000000 after $2^8=256$ customers.

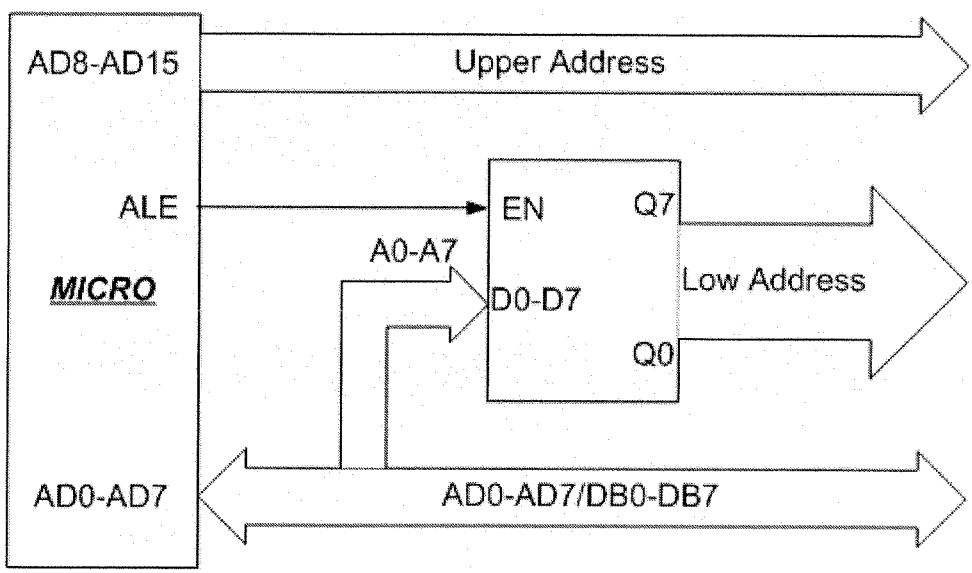
**5.36**



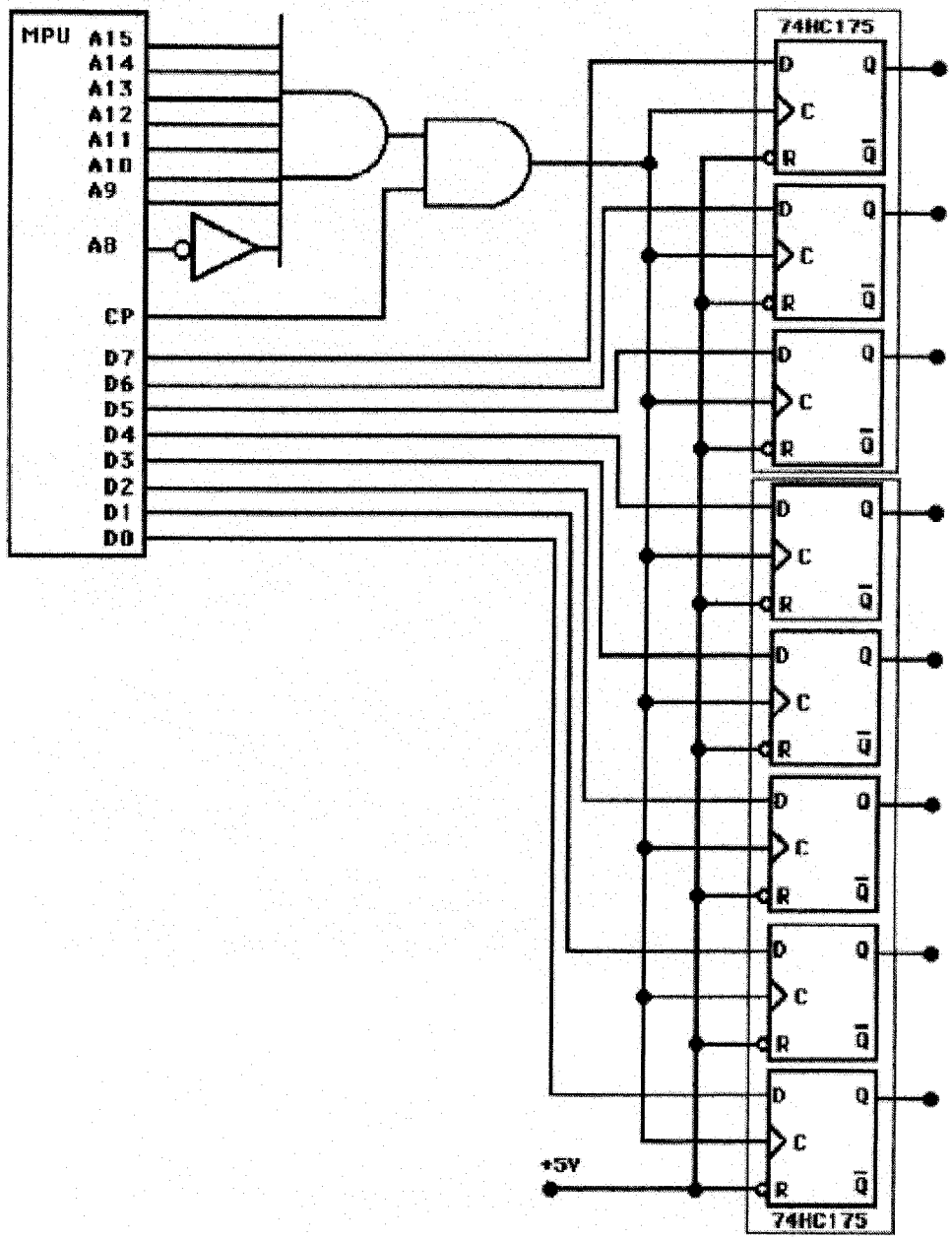

**5.37** Regardless of the logic state of the address line A8, data gets transferred from the MPU to the X register. Thus, the problem is in the connection between the address line A8 from the MPU and the 8-input AND gate. The following are some of the circuit faults that could cause this malfunction:

(a) External open on address line A8 between the MPU and the input of the Inverter.
(b) External short to Vcc on address line A8 between the MPU and the input of the Inverter.
(c) External open on the line connecting the output of the Inverter and the input of the AND gate.
(d) External short to Vcc on the line connecting the output of the Inverter and the input of the AND gate.
(e) Internal open or short to Vcc on the input of the Inverter.
(f) Internal open or short to Vcc on the output of the Inverter.
(g) Internal open or short to Vcc on the input of the AND gate.

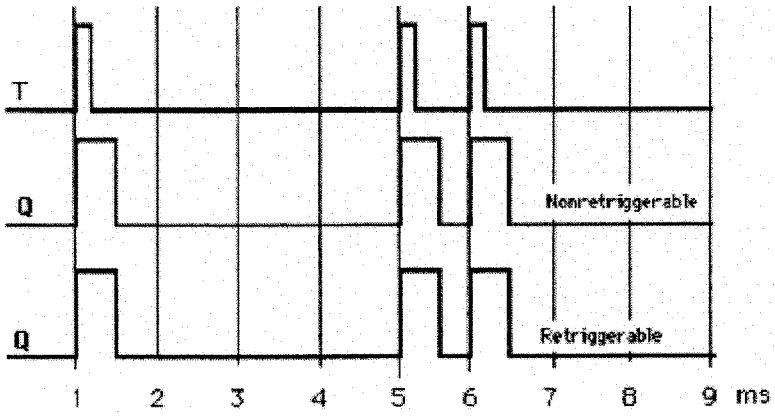
**5.38**



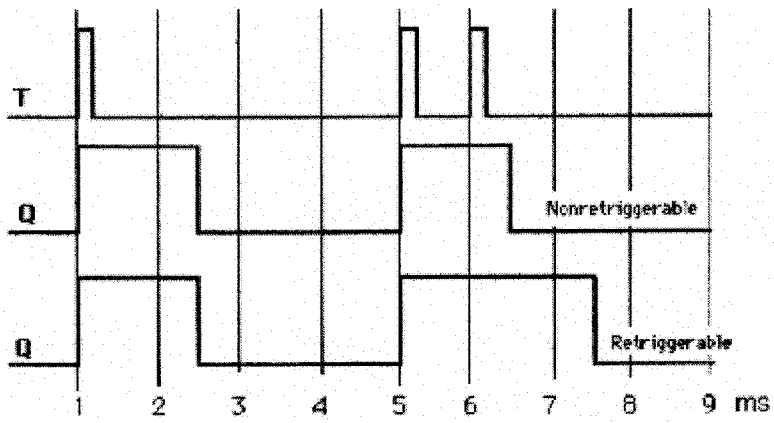
**5.39**


MPU
A15
A14
A13
A12
A11
A10
A9
A8
CP
D7
D6
D5
D4
D3
D2
D1
D0
74HC175
D
Q
C
R
Q
+5V
74HC175

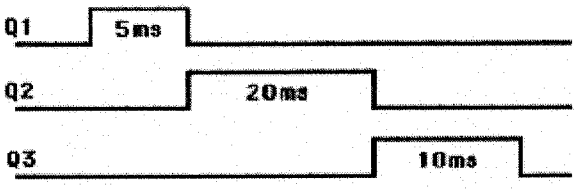
**5.40** With tp = 0.5ms




With tp = 1.5ms




**5.41**




**5.42** (a) Closing S1 clears X to 0. Since the OS has tp=1ms, the OS will be triggered before the end of the tp interval for frequencies greater than 1 KHz. Thus, $\overline{Q}$ will stay LOW.

(b) If the input frequency drops below 1 KHz, the $\overline{Q}$ will return HIGH before the OS is triggered again. This PGT at $\overline{Q}$ will clock X to the 1 state.

(c) Change tp to 1/50 KHz = 20μs.

**5.43** (a) A1 or A2 has to be LOW, and a PGT must occur at B.
(b) B and A2 have to be HIGH, and a NGT must occur at A1.

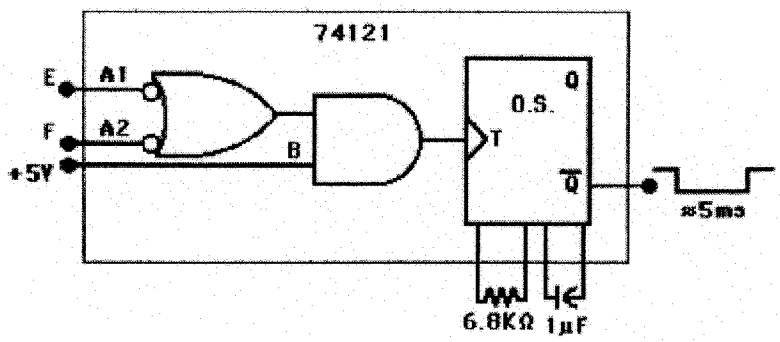
**5.44** (a) **<u>One possibility:</u>**

$0.7\ R_T C_T = 5ms$

Let $C_T = 1\mu F$; $0.7\ R_T = 5ms/1\mu F = 5000$

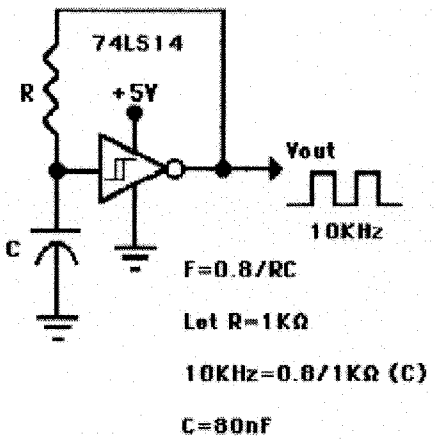
$R_T = 7143\Omega \approx 6.8K\Omega$ (std. value).

If an accurate 5ms is required, an adjustable $R_T$ should be used.




(b) Connect G to input B of 74121.

**5.45**




**5.46** **<u>One possibility:</u>**

F=40 KHz; T=25μs; $t_1 = t_2 = 12.5\mu s$

For a squarewave **RA<<RB**; Let RA=1KΩ and RB=10KΩ

$t_1 = 0.693(RB)(C)$: $12.5\mu s = 0.693(10K\Omega)(C)$: $C = 1800pF$

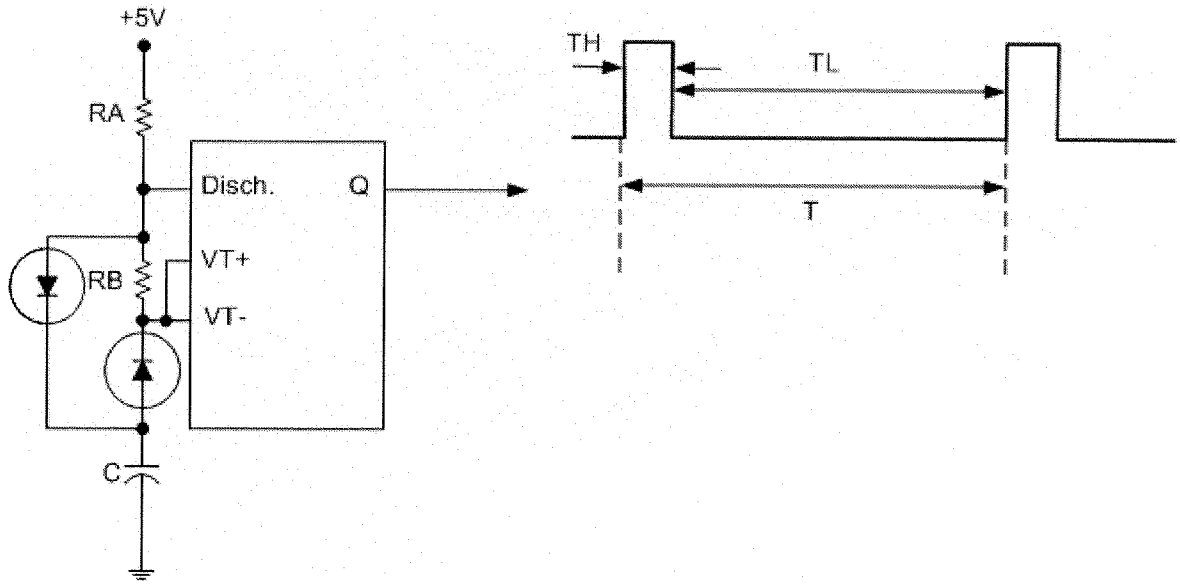
$T = 0.693(RA+2RB)C$: $T = 0.693(1K\Omega+20K\Omega)1800pF$

T=26.2μs; F=1/T; F=38 KHz (almost squarewave).

**5.47** **<u>One possibility:</u>**

Reduce by half the 1800pF. This will create a T=13.1μs or F=76.35 KHz (almost square wave). Now, take the output of the 555 Timer and connect it to the CLK input of a J-K FF wired in the toggle mode (J and K inputs connected to +5V). The result at the Q output of the J-K FF is a perfect 38.17 KHz square wave.

**5.48**




$$T = \frac{1}{F} = \frac{1}{5KHz} = 200\mu s$$

$$TH = (10\%)(200\mu s) = 20\mu s$$

$$TL = T - TH = 200\mu s - 20\mu s = 180\mu s$$

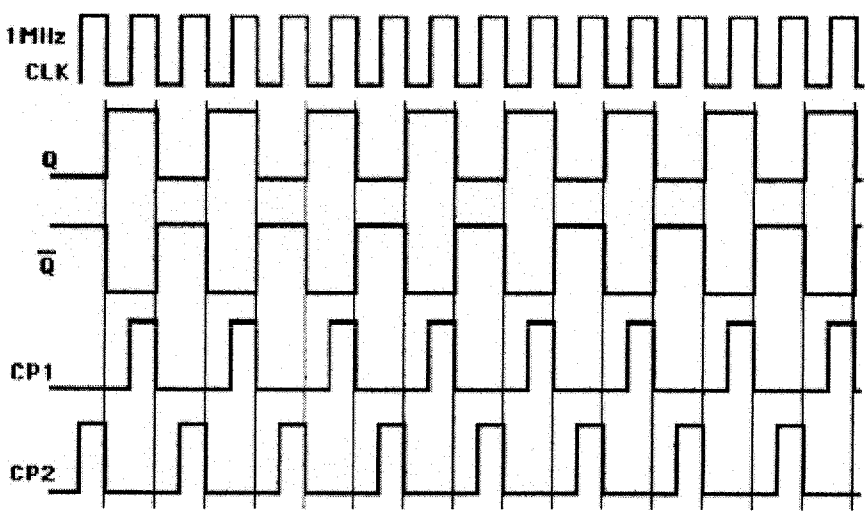
***Choose C = 0.01μF***

$$TL = 0.75(RB)(C)$$

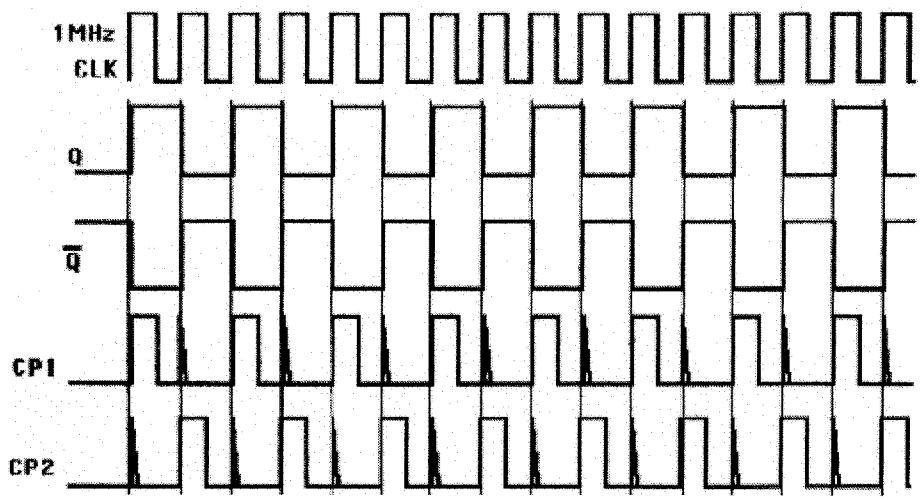
$$RB = \frac{TL}{0.75C} = \frac{180\mu s}{0.75(0.01\mu F)} = 13.5K\Omega$$

$$RA = \frac{TH}{0.75C} = \frac{20\mu s}{0.75(0.01\mu F)} = 1.5K\Omega$$

**5.49** (a)



(b)




**5.50** (a) No. An open on the CLR input would be the same as a TTL HIGH and would not cause FF X2 to clear on the fourth pulse.
(b) Yes. Since X1 provides the CLK input to FF X2, a slow transition on X1 could cause erratic clocking of X2.
(c) No. This would keep X2 at a permanent LOW.
(d) No. Since X2's J and K inputs are held HIGH.

**5.51** (a) Yes. Q2 will stay LOW because the set-up time for FF Q2 has to be equal to 5ns or longer and it was only 1ns (skew=13ns, $t_{pLH}$ for Q1=12ns)
(b) No. Q2 will go HIGH since the set-up time is 8ns which is greater than 5ns. Thus, when Q2 is clocked, Q1 has already been HIGH for 8ns and the level at Q1 will be transferred to Q2 (skew=18ns, $t_{PLH}$ for Q1=10ns).

**5.52** Two cascading Inverters between Q1 and D2. This would add 12ns or 14ns to the effective $t_{pLH}$ of Q1 (using propagation delays for the Inverters of problem 5.45 (a) and (b)). Now the skew time would be less than the effective propagation delay $t_{pLH}$ of Q1. Thus, by the time FF Q2 gets clocked, the signal at D2 hasn't yet changed.

**5.53** (a) No. If point X was always LOW inputs J and K would've been always HIGH and therefore FF Z2 would've toggled on each NGT of the clock.
(b) No. An internal short to Vcc at Z1-1 would make input K always LOW. Under these conditions FF Z2 would be cleared (J=0,K=1) or it wouldn't change states (J=0,K=0) on the NGT of the clock.
(c) Yes. This condition causes the J input to always be HIGH (floating TTL input). Any time a NGT on the clock occurs and B is LOW, FF Z2 will toggle. If the B input is HIGH FF Z2 will SET. This analysis agrees with the Q waveform.
(d) No. This would cause input K to always be LOW. Under this condition FF Z2 could either SET (J=1,K=0) or it wouldn't change states (J=0,K=0) on the NGT of the clock.

**5.54** SWA = 1 ; SWB = 0 ; SWC = 1 (First combination)
SWA = 0 ; SWB = 1 ; SWC = 0 (Second combination)

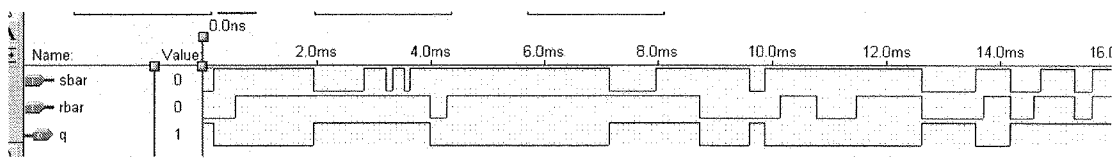
**5.55** (a) No. Switch bounce would have no effect since the D inputs of the FFs are not sensitive to transitions.
(b) No. An open on the CLR (HIGH for TTL) input of FF Q2 wouldn't cause Q2 to change during a PGT on the CLK.

(c) Yes. This fault would cause the switch bounce from the ENTER switch to be present at the CLK inputs of the D-type FFs. Since the input D of FF Q1 is at a logic LOW during the second combination, after the first bounce FF Q2 would get SET and after the second switch bounce it would get CLEAR.

**5.56** (a) NAND or NOR gate latch.
(b) Clocked J-K flip-flop.
(c) D Latch.
(d) Clocked D flip-flop or J-K flip-flop.
(e) Clocked D flip-flop.
(f) All types of flip-flops.
(g) Any edge-triggered flip-flop.
(h) J-K flip-flops.

**5.57** (a) ***Asynchronous Inputs*** - Flip-flop inputs that can affect the operation of the flip-flop independent of the synchronous and clock inputs.
(b) ***Edge-Triggered*** - Manner in which a flip-flop is activated by a signal transition. It may be either a positive or negative edge-triggered flip-flop.
(c) ***Shift Register*** - Digital circuit that accepts binary data from some input source and then shifts these data through a chain of flip-flops one bit at a time.
(d) ***Frequency division*** - Expression normally associated with counters. The frequency division ratio of a counter is equal to the total number of different states that counter can go through and is often referred to as the counter's MOD number.
(e) ***Asynchronous (Jam) Transfer*** - Data transfer performed without the aid of the clock.
(f) ***State transition diagram*** - Way to show pictorially the states of flip-flops change with each applied clock pulse.
(g) ***Parallel Data Transfer*** - Operation by which the entire contents of a register are transferred simultaneously to another register.
(h) ***Serial Data Transfer*** - When data are transferred from one place to another one bit at a time.
(i) ***Retriggerable One-Shot*** - Type of One-Shot that can be triggered while it is in the quasi-stable state, and it will begin a new $t_P$ interval.
(j) ***Schmitt-trigger inputs*** - Inputs on certain devices that accept slow-changing signals and produce oscillation-free transitions at the output.

**5.58**




This latch design always SETs when both inputs are active (LOW). It remains SET if the inputs change simultaneously to the no change mode.

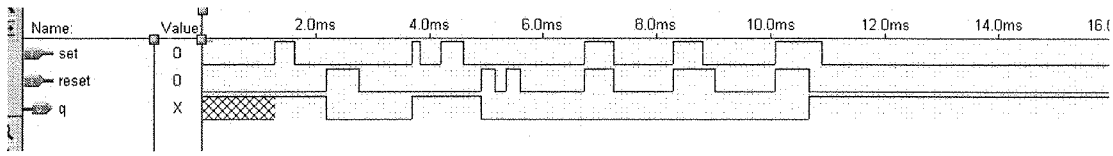
**5.59**

```
SUBDESIGN prob5_59
(
    set, reset              :INPUT;
    q                       :OUTPUT;
)
BEGIN
    IF      set == 1 THEN       q = VCC;    -- set or illegal command
    ELSIF   reset == 1  THEN    q = GND;    -- reset
    ELSE                        q = q;      -- hold
    END IF;
END;
```

```
ENTITY prob5_59 IS                    -- must compile with Multi-Level Synthesis for
                                      -- 7000 enabled
PORT (      set, reset          :IN BIT;
            q                   :BUFFER BIT);
END prob5_59;
ARCHITECTURE behavior OF prob5_59 IS
BEGIN
    PROCESS (set, reset)
            BEGIN
                IF     set = '1' THEN          q <= '1';       -- set or illegal command
                ELSIF  reset = '1' THEN        q <= '0';       -- reset
                ELSE                           q <= q;         -- hold
                END IF;
            END PROCESS;
END behavior;
```

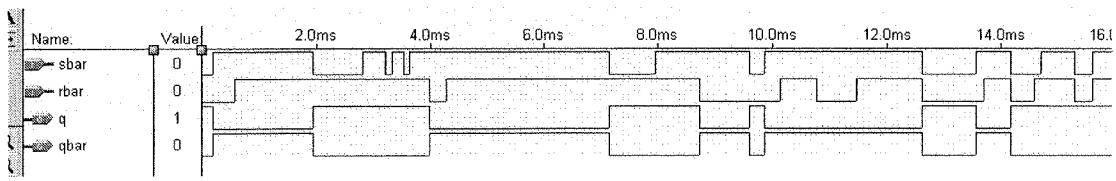
**5.60**

```
    SUBDESIGN prob5_60
(
    set, reset                        :INPUT;
    q                                 :OUTPUT;
)
BEGIN
    IF              reset == 1        THEN  q = GND;      -- reset or illegal command
    ELSIF   set == 1                  THEN  q = VCC;      -- set
    ELSE                                    q = q;        -- hold
    END IF;
END;

ENTITY prob5_60 IS                    -- must compile with Multi-Level Synthesis for
                                      -- 7000 enabled
PORT (      set, reset          :IN BIT;
            q                   :BUFFER BIT);
END prob5_60;
ARCHITECTURE behavior OF prob5_60 IS
BEGIN
    PROCESS (set, reset)
            BEGIN
                IF      reset = '1'       THEN  q <= '0';     -- reset or illegal command
                ELSIF   set = '1'         THEN  q <= '1';     -- set
                ELSE                            q <= q;       -- hold
                END IF;
            END PROCESS;
END behavior;
```

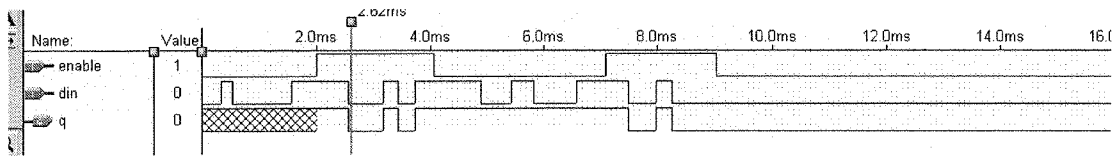


**5.61**

```
SUBDESIGN prob5_61
(
    sbar, rbar              :INPUT;
    q, qbar                 :OUTPUT;
)
BEGIN
    IF sbar == 0            THEN  q = VCC;    qbar = GND;    -- set or illegal command
    ELSIF  rbar == 0        THEN  q = GND;    qbar = VCC;    -- reset
    ELSE                          q = q;      qbar = qbar;   -- hold
    END IF;
END;

ENTITY prob5_61 IS                          -- must compile with Multi-Level Synthesis for
                                            -- 7000 enabled
PORT (      sbar, rbar      :IN BIT;
            q, qbar         :BUFFER BIT);
END prob5_61;
ARCHITECTURE behavior OF prob5_61 IS
BEGIN
    PROCESS (sbar, rbar)
            BEGIN
                IF sbar = '0'       THEN  q <= '1'; qbar <= '0';    -- set or illegal command
                ELSIF rbar = '0'    THEN  q <= '0'; qbar <= '1';    -- reset
                ELSE                      q <= q; qbar <= qbar;     -- hold
                END IF;
            END PROCESS;
END behavior;
```



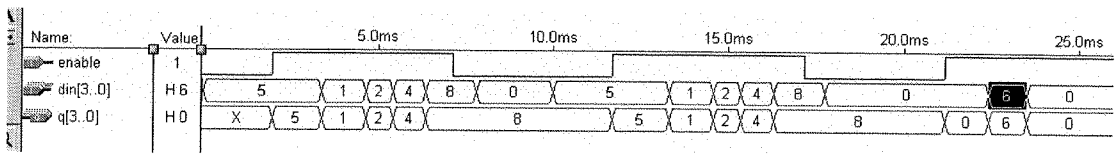

**5.62**




**5.63**

```
SUBDESIGN prob5_63
(enable, din[3..0]          :INPUT;
    q[3..0]                 :OUTPUT;)

VARIABLE
    q[3..0]                 :LATCH;
BEGIN
    q[].ena = enable;
    q[].d = din[];
END;
ENTITY  prob5_63 IS
PORT  (enable               :IN BIT;
```

```
                    din     :IN BIT_VECTOR (3 DOWNTO 0);
                    q       :OUT BIT_VECTOR (3 DOWNTO 0));
END prob5_63;

ARCHITECTURE v OF prob5_63 IS
BEGIN
    PROCESS (enable, din)
    BEGIN
            IF enable = '1' THEN
                    q <= din;
            END IF;
    END PROCESS;
END v;
```




**5.64** (a)

```
%T flip flop circuit %
SUBDESIGN prob5_64a
(
        clk, t          :INPUT;
        q, qbar         :OUTPUT;

)
VARIABLE ff1   :TFF;
BEGIN
        ff1.clk = clk;
        ff1.t = t;
        q = ff1.q;
        qbar = !q;
END;
```

**5.64** (b)

```
-- T flip flop circuit

ENTITY prob5_64b IS
port(   clk, t                  :IN BIT;
        q, qbar                 :OUT BIT);
END prob5_64b;

ARCHITECTURE a OF prob5_64b IS
signal qstate                   :BIT;
BEGIN
        PROCESS(clk)                            --respond to any of these signals
        BEGIN
                IF  clk = '1' AND clk'event THEN  -- on PGT clock edge
                IF t = '1' THEN
                IF qstate = '1' THEN qstate <= '0';
                ELSE qstate <= '1';
                END IF;
                END IF;
                END IF;
        END PROCESS;
```

```
        q <= qstate;                                --update output pin
        qbar <= NOT qstate;
END a;
```

**5.65** (a)

```
%Figure 5-45 Implemented to clearly show each connection %

SUBDESIGN prob5_65a
(
        clock, data_in      :INPUT;
        xff[3..0]           :OUTPUT;
)
VARIABLE
        xff[3..0] :JKFF;                    -- defines three JK FFs
        BEGIN
                xff[].clk = clock;          --synchronous (parallel) clocking
                xff[3].J = data_in;
                xff[3].K = !data_in;
                xff[2].J = xff[3].q;
                xff[2].K = !xff[3].q;
                xff[1].J = xff[2].q;
                xff[1].K = !xff[2].q;
                xff[0].J = xff[1].q;
                xff[0].K = !xff[1].q;
        END;
```

**5.65** (b)

```
--Figure 5-45 implemented to clearly show each connection
--"structural level of abstraction" using library primitive
-- for a JK flip flop.

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
LIBRARY altera;
USE altera.maxplus2. ALL;

ENTITY prob5_65b IS
PORT(
        clock, data_in          :IN std_logic ;
        xff                     :OUT std_logic_vector (3 DOWNTO 0));
END prob5_65b;

ARCHITECTURE a OF prob5_65b IS
SIGNAL  high, data_in_not       :std_logic;
SIGNAL          qnot            :std_logic_vector (3 DOWNTO 0);
SIGNAL          q               :std_logic_vector (3 DOWNTO 0);

BEGIN
        high <= '1';                                        --connection for Vcc
        qnot <= NOT q;
        data_in_not <= NOT data_in;
ff3: JKFF
        PORT MAP(       j => data_in,
                        k => data_in_not,
                        clk => clock,
                        clrn => high,
                        prn => high,
                        q => q(3));
```

```
ff2: JKFF
        PORT MAP(     j => q(3),                    -- toggle mode
                      k => qnot(3),
                      clk => clock,                 -- ripple clock connection
                      clrn => high,                 -- asynch inputs inactive
                      prn => high,
                      q => q(2));                   --connect to output signal
ff1: JKFF
        PORT MAP(     j => q(2),
                      k => qnot(2),
                      clk => clock,
                      clrn => high,
                      prn => high,
                      q => q(1));
ff0: JKFF
        PORT MAP(     j => q(1),
                      k => qnot(1),
                      clk => clock,
                      clrn => high,
                      prn => high,
                      q => q(0));
xff <= q;                                           -- connect ff out signals to output pins

END a;
```

**5.66** (a)

```
% Figure 5-46 implemented to clearly show each connection %

SUBDESIGN prob5_66a
(
        clock, data_in              :INPUT;
        xff[2..0], yff[2..0]        :OUTPUT;
)
VARIABLE
        xff[2..0], yff[2..0]:DFF;           -- defines two JK FFs

        BEGIN
                xff[].clk = clock;          --synchronous (parallel) clocking
                yff[].clk = clock;
                xff[2].D = data_in;
                xff[1].D = xff[2].q;
                xff[0].D = xff[1].q;
                yff[2].D = xff[0].q;
                yff[1].D = yff[2].q;
                yff[0].D = yff[1].q;
        END;
```

**5.66** (b)

```
--Figure 5-46 implemented to clearly show each connection
--"structural level of abstraction" using library primitive
-- for a D flip flop.

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
LIBRARY altera;
USE altera.maxplus2. ALL;
```

```
ENTITY prob5_66b IS
PORT(
        clock, data_in          :IN std_logic ;
        xff, yff                :OUT std_logic_vector (2 DOWNTO 0));
END prob5_66b;

ARCHITECTURE a OF prob5_66b IS
SIGNAL  high, data_in_not       :std_logic;
SIGNAL          x, y            :std_logic_vector (2 DOWNTO 0);

BEGIN
        high <= '1';                                    -- connection for Vcc
xff2: DFF
        PORT MAP(       d => data_in,
                        clk => clock,
                        clrn => high,
                        prn => high,
                        q => x(2));

xff1: DFF
        PORT MAP(       d => x(2),                      -- toggle mode
                        clk => clock,                   -- ripple clock connection
                        clrn => high,                   -- asynch inputs inactive
                        prn => high,
                        q => x(1));                     -- connect to output signal
xff0: DFF
        PORT MAP(       d => x(1),
                        clk => clock,
                        clrn => high,
                        prn => high,
                        q => x(0));
yff2: DFF
        PORT MAP(       d => x(0),
                        clk => clock,
                        clrn => high,
                        prn => high,
                        q => y(2));
yff1: DFF
        PORT MAP(       d => y(2),
                        clk => clock,
                        clrn => high,
                        prn => high,
                        q => y(1));
yff0: DFF
        PORT MAP(       d => y(1),
                        clk => clock,
                        clrn => high,
                        prn => high,
                        q => y(0));

        xff <= x;                                       -- connect ff out signals to output pins
        yff <= y;
END a;
```

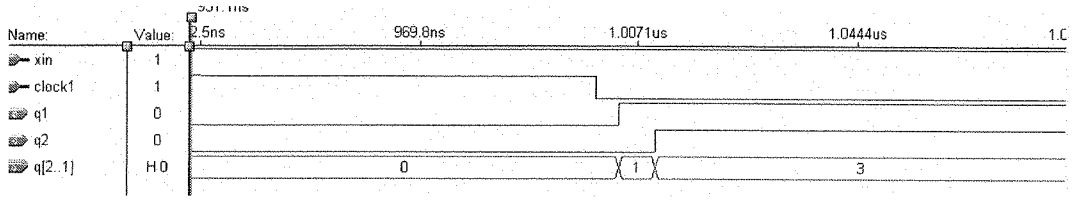
**5.67** (a)

```
% Figure 5-59 implemented to clearly show each connection %

SUBDESIGN prob5_67a
(
        clock1, xin             :INPUT;
        q1, q2                  :OUTPUT;
        )
VARIABLE
        q1, q2                  :DFF;           -- defines two D FFs
        clock2, nandout         :node;
        BEGIN
                q1 . clk = !clock1;
                q1 . d = VCC;
                q2 . d = q[1] . q;
                clock2 = !nandout;
                nandout = !(xin & clock1);
        END;
```

**5.67** (b)

```
--Answer to problem 5-67b

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
LIBRARY altera;
USE altera.maxplus2. ALL;

ENTITY prob5_67b IS
PORT(
        clock1, xin                                     :IN std_logic ;
        q1, q2                                          :OUT std_logic);
END prob5_67b;

ARCHITECTURE a OF prob5_67b IS
SIGNAL  high, clock2, nandout, clk1not, clk2not         :std_logic;
SIGNAL  qone, qtwo                                      :std_logic;

BEGIN
        high <= '1';                                    -- connection for Vcc
        nandout <= NOT (xin AND clock1);
        clock2 <= NOT nandout;
        clock2not <= NOT clock2;
        clocknot <= NOT clock1;
ff1: DFF
        PORT MAP (      d => high,
                        clk => clk1not,
                        clrn => high,
                        prn => high,
                        q => qone);
ff2: DFF
        PORT MAP (      d => qone,
                        clk => clk2not,
                        clrn => high,
                        prn => high,
                        q => qtwo);
q1 <= qone;                                             -- connect ff out signals to output pins
q2 <= qtwo;
END a;
```

**5.68**




**5.69** (a) % Figure 5-89 implemented to clearly show each connection %

```
SUBDESIGN prob5_69a
(
        swa, swb, swc, reset, enterNO, enterNC          :INPUT;
        lock                                            :OUTPUT;
)
VARIABLE
        q1, q2                  : DFF;                  -- defines two D FFs
        enter, enterbar         : node;

        BEGIN
                q1 . clrn = reset;
                q2 . clrn = reset;
                q1 . d = swa & !swb & swc;
                q2 . d = !swa & swb & !swc & q1 . q;
                enter = !enterNO # !enterbar;
                enterbar = !enterNC # !enter;
                q1 . clk = enter;
                q2 . clk = enter;
                lock = q2 . q;
        END;
```

**5.69** (b) --Answer to problem 5-69b

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
LIBRARY altera;
USE altera.maxplus2. ALL;

ENTITY prob5_69b IS
PORT(
        Swa, swb, swc, reset, enterNO, enterNC      :IN std_logic ;
        lock                                        :OUT std_logic);
END prob5_69b;

ARCHITECTURE a OF prob5_69b IS
SIGNAL  q1, q2, enter, enterbar,                    :std_logic;
SIGNAL  gate2, gate7, high                          :std_logic;
```

```
BEGIN
        high <= '1';                                        -- connection for Vcc
        gate2 <= swa AND (NOT swb) AND swc;
        gate7 <= (NOT swa) AND swb AND (NOT swc) AND q1;
        lock <= q2;
        enter <= NOT enterNO OR NOT enterbar;
        enterbar <= NOT enterNC OR NOT enter;
ff1: DFF
        PORT MAP (      d => gate2,
                        clk => enter,
                        clrn => reset,
                        prn => high,
                        q => q1);
ff2: DFF
        PORT MAP (      d => gate7,                         -- toggle mode
                        clk => enter,                       -- ripple clock connection
                        clrn => reset,                      -- asynch inputs inactive
                        prn => high,
                        q => q2);
END a;
```

## CHAPTER SIX - Digital Arithmetic: Operations and Circuits

**6.1**

| (a) | (b) | (c) | (d) |
|---|---|---|---|
| 1010 | 1111 | 1011.1101 | 0.1011 |
| + 1011 | + 0011 | + 0011.1000 | + 0.1111 |
| 10101 | 10010 | 1111.0101 | 1.1010 |

| (e) | (f) |
|---|---|
| 10011011 | 1010.01 |
| + 10011101 | + 0010.111 |
| 100111000 | 1101.001 |

**6.2**

(a) $+32_{10}=\underline{0}0100000_2$
(b) $-14_{10}=\underline{1}1110010_2$
(c) $+63_{10}=\underline{0}0111111_2$
(d) $-104_{10}=\underline{1}0011000_2$
(e) $+127_{10}=\underline{0}1111111_2$
(f) $-127_{10}=\underline{1}0000001_2$
(g) $+89_{10}=\underline{0}1011001_2$
(h) $-55_{10}=\underline{1}1001001_2$
(i) $-1_{10}=\underline{1}1111111_2$
(j) $-128_{10}=\underline{1}0000000_2$
(k) Can't be represented with eight bits.
(l) $0_{10}=\underline{0}0000000_2$
(m) $+84_{10}=\underline{0}1010100_2$
(n) $+3_{10}=\underline{0}0000011_2$
(o) $-3_{10}=\underline{1}1111101_2$
(p) Can't be represented with eight bits.

**6.3**

(a) $\underline{0}1101_2=13_{10}$
(b) $\underline{1}1101_2=-3_{10}$
(c) $\underline{0}1111011_2=+123_{10}$
(d) $\underline{1}0011001_2=-103_{10}$
(e) $\underline{0}1111111_2=+127_{10}$
(f) $\underline{1}0000000_2=-128_{10}$
(g) $\underline{1}1111111_2=-1_{10}$
(h) $\underline{1}0000001_2=-127_{10}$
(i) $\underline{0}1100011_2=99_{10}$
(j) $\underline{1}1011001_2=-39_{10}$

**6.4** (a) Eleven magnitude bits can represent decimal numbers from $-2^{11}$ to $+(2^{11}-1)$ or $-2048_{10}$ to $2047_{10}$.

(b) $-32{,}768=-2^N$---> N=15 (for magnitude). Thus, sixteen bits are required including sign bit.

**6.5** Four magnitude bits can represent numbers from $-16_{10}$ to $+15_{10}$.

**6.6**

| | Dec. number | 8-bit signed number | 2's-comp. (Negate) |
|---|---|---|---|
| (a) | $+73_{10}$ ----------> | 01001001-------> | $10110111_2 = (-73_{10})$ |
| (b) | $-12_{10}$ ----------> | 11110100-------> | $00001100_2 = (+12_{10})$ |
| (c) | $+15_{10}$ ----------> | 00001111-------> | $11110001_2 = (-15_{10})$ |
| (d) | $-1_{10}$ ----------> | 11111111-------> | $00000001_2 = (+1_{10})$ |

(e) $-128_{10}$ ----------> $10000000_2$ -----> It requires nine binary bits to represent $+128_{10}= 010000000_2$

(f) $+127_{10}$ ----------> $01111111_2$ -----> $10000001_2 = (-127_{10})$

**6.7** (a) With 10 bits, we can represent any unsigned number from 0 to $1023_{10}$. With 10 bits, we can represent signed numbers from $-2^9$ to $+(2^9-1)$, or $-512_{10}$ to $+511_{10}$.

(b) With 8 bits, we can represent any unsigned number from 0 to $255_{10}$. With 8 bits, we can represent signed numbers from $-2^7$ to $+(2^7-1)$, or $-128_{10}$ to $+127_{10}$.Using 8-bits

**6.8** (a) $+12_{10}$= **0**0001100 (b) $-12_{10}$= **1**0001100
(c) (**0**0001100 + **1**0001100) = **10011000 ≠ 0 ??**

**6.9** (a)
```
+9 =   00001001
+6 =   00000110
     +________
   = 00001111 = +15
```

(b)
```
+14 =  00001110
-17 =  11101111
     +________
   = 11111101 = -3
```

(c)
```
+19 =  00010011
-24 =  11101000
     +________
   = 11111011 = -5
```

(d)
```
-48 =  11010000
-80 =  10110000
     +________
   = 10000000 = -128
```

(e)
```
+17 =  00010001
-16 =  11110000
     +________
   = 00000001 = +1
```

(f)
```
-13 =  11110011
-21 =  11101011
     +________
   = 11011110 = -34
```

(g)
```
+47 =  00101111
-47 =  11010001
     +________
   = 00000000 = 0
```

(h)
```
-15 =  11110001
+36 =  00100100
     +________
   = 00010101 = +21
```

(i)
```
+17 =  00010001
-17 =  11101111
     +________
   = 00000000 = 0
```

(j) Same as (i).

**6.10** (a)
```
+37 =  00100101
+95 =  01011111
     +________
   = 10000100 (Sign bit=1 indicates overflow.)
```

(b)
```
-95      = 10100001
+(-37)   = 11011011
         +________
         = 01111100 (Sign bit=0 indicates overflow.)
```

(c)
```
-37      = 11011011
+(-95)   = 10100001
         +________
         = 01111100 (Sign bit=0 indicates overflow.)
```

(d)
```
 95      = 01011111
-(-37)   = 00100101
         +________
         = 10000100 (Sign bit=1 indicates overflow.)
```

**6.11**

(a) 7x5=35

```
   111
 x 101
   111
  000
 111
100011 =35
```

(b) 11x11=121

```
  1011
 x1011
  1011
 1011
0000
1011
1111001 =121
```

(c) 5.625x6.25=35.15625

```
   101.101
 x 110.010
   000000
  101101
 000000
000000
101101
101101
100011.00101=35.15625
```

(d) 0.8125x0.6875=0.55859375

```
   .1101
 x .1011
   1101
  1101
 0000
1101
.10001111=.55859375
```

**6.12** *12/4=3* *63/9=7* *23/4=5.75*

(a)
```
      11
100 1100
   100
   100
   100
     0
```

(b)
```
     111
1001 111111
   1001
   1101
   1001
   1001
      0
```

(c)
```
     101.11
100 10111.0
   100
    111
    100
     110
     100
     100
       0
```

*22.8125/1.5=15.1875*

(d)
```
          1111.0011
1.1. 10110.1.1010
     11
      101
      11
     100
      11
     011
     011
      0101
       11
      100
       11
       01
```

**6.13**

a)
```
0111  0100
0010  0011
+
1001  0111(BCD)
```

b)
```
 0101  1000
 0011  0111
+
 1000  1111
 +     0110
 1001  0101(BCD)
```

c)
```
 0001  0100  0111
 0011  1000  0000
+
 0100  1100  0111
 +           0110
 0101  0010  0111(BCD)
```

d)
```
  0011  1000  0101
  0001  0001  1000
+-------------------
  0100  1001  1101
            + 0110
-------------------
  0100  1010  0011
      + 0110
-------------------
  0101  0000  0011(BCD)
```

e)
```
  1001  1001  1000
  0000  0000  0011
+-------------------
  1001  1001  1011
            + 0110
-------------------
  1001  1010  0001
      + 0110
-------------------
  1010  0000  0001
+ 0110
-------------------
0001  0000  0000  0001(BCD)
```

f)
```
  0110  0010  0011
  0101  1001  1001
+-------------------
  1011  1011  1100
  0110  0110  0110
-------------------
0001  0010  0010  0010(BCD)
```

**6.14** (a) 3E91 + 2F93 = 6E24 (b) 91B + 6F2 = 100D (c) ABC + DEF = 18AB

(d) 2FFE + 0002 = 3000 (e) FFF + 0FF = 10FE (f) D191 + AAAB = 17C3C

**6.15** (a)
```
3E91 ---------------------------- 3E91
2F93 ---------------------------- D06D (2's complement)
 -                                +
(Ignore carry) ->                 10EFE
```

(b)
```
91B ----------91B
6F2----------90E
 -          +
(Ignore carry) -> 1229
```

(c)
```
0300 ----------0300
005A ----------FFA6
 -             +
(Ignore carry) ---> 102A6
```

(d)
```
0200----------0200
0003----------FFFD
 -             +
(Ignore carry) ---> 101FD
```

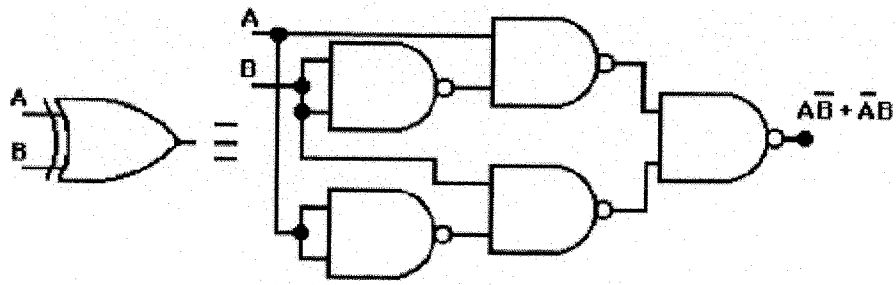
(e)
```
F000 ----------F000
EFFF ----------1001
 -             +
(Ignore carry) ---> 10001
```

(f)
```
2F00 ----------2F00
4000----------C000
 -             +
               EF00
```

**6.16**
```
  03FF          7FD0          } 0200
  0200          4000          } 3FD1
-------       -------         } +----
  01FF          3FD0          } 41D1(16) locations=1684910
    +1            +1          }
-------       -------         }
  0200          3FD1          }
```

**6.17** (a) $77_{16} = 119_{10}$ (b) $77_{16} = +119_{10}$ (c) $E5_{16} = 229_{10}$ ; $E5_{16} = -27_{10}$

**6.18** One possibility is to convert each EX-OR to its NAND equivalents shown below:




Then, convert ANDs and OR gate for $C_{OUT}$ to their equivalent NAND representation.

**6.19**

| A | B | SUM | CARRY |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

$SUM = \bar{A}B + A\bar{B} = A \oplus B$

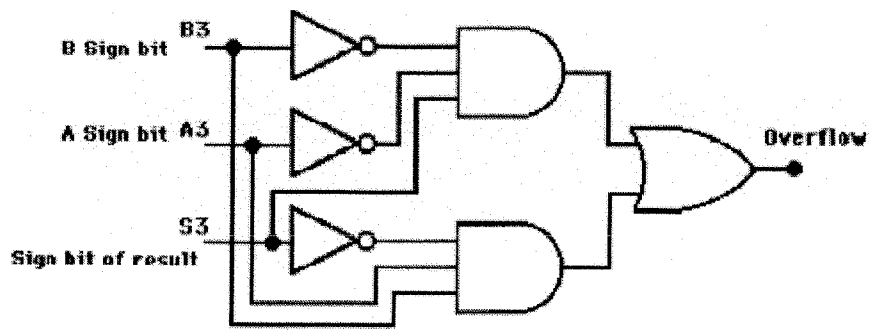
$CARRY = AB$

**6.21**

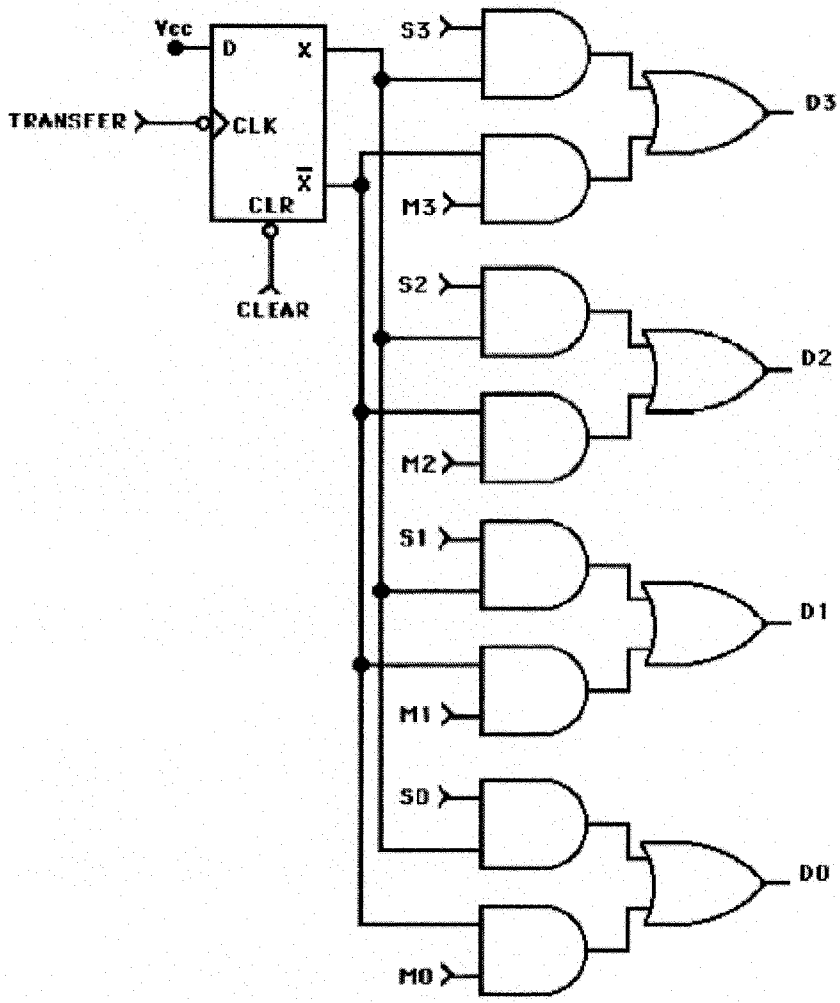
```
 0000 [A]          0101 [A]
+0101 [B]         +1011 [B]
 0101 [S]  ---->  10000 [S] ----> [A] = 0000
```

**6.22** After the PGT of the LOAD pulse, the FFs in the B register require 30ns to produce proper levels to the FAs. The FAs produce stable outputs in 4x40ns or 160ns (allowing for carry propagation). These outputs have to be present at the inputs of the A register for 10ns (set-up time) before the PGT of the TRANSFER pulse. The total time, then, is 200ns.

**6.23** **Overflow Circuit**



**6.24** One possibility:




**6.25**

$$C1 = [A0\,B0 + A0\,C0 + B0\,C0]$$

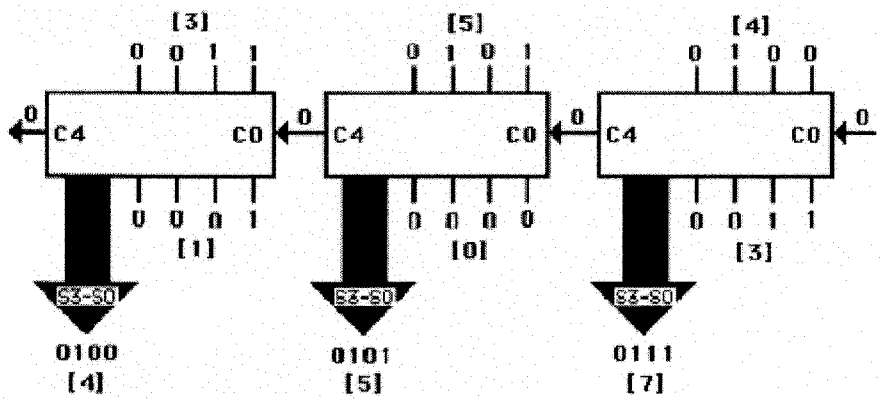
$$C2 = (A1\,B1 + A1\,C1 + B1\,C1) = A1\,B1 + (A1 + B1)[C1]$$

$$C3 = A2\,B2 + A2\,C2 + B2\,C2$$

$$C3 = A2\,B2 + (A2+B2)\{A1B1 + (A1+B1)[A0B0 + B0C0 + A0C0]\}$$

Final expression for C3 can be put into S-of-P form by multiplying all terms out. This results in a circuit with TWO levels of gating. The arrangement of Figure 6.9 requires that A0, B0, and C0 propagate through as many as 6 levels of gates before producing C3.

**6.26**




**6.27** (a) SUM = 0111 (b) SUM = 1010 (-6) (c) SUM = 1100 (-4)

**6.28** (a)

| C4 | | | C0 | |
|---|---|---|---|---|
| *1* | *1* | *1* | *1* | |
| **1** | **1** | **0** | **1** | A |
| **1** | **1** | **0** | **0** | $\overline{B}$ |
| **1** | **0** | **1** | **0** | **Σ (-6)** |

(b)

| C4 | | | C0 | |
|---|---|---|---|---|
| *0* | | | *0* | |
| **1** | **1** | **0** | **0** | A |
| **0** | **0** | **1** | **0** | B |
| **1** | **1** | **1** | **0** | **Σ (-2)** |

(c)

| C4 | | | C0 | |
|---|---|---|---|---|
| *1* | | *1* | *1* | *1* |
| **1** | **0** | **1** | **1** | A |
| **1** | **0** | **1** | **1** | $\overline{B}$ |
| **0** | **1** | **1** | **1** | **Σ (+7)** |

**6.29** (a)

| | | *1* | *1* | |
|---|---|---|---|---|
| **0** | **1** | **0** | **1** | A |
| **0** | **0** | **0** | **1** | $\overline{B}$ |
| **0** | **1** | **1** | **1** | **Σ (+7)** |

*No Overflow*

(b)

| *1* | | | | |
|---|---|---|---|---|
| **1** | **1** | **0** | **0** | A |
| **1** | **1** | **1** | **0** | B |
| **1** | **0** | **1** | **0** | **Σ (-6)** |

*No Overflow*

(c)

| | | | | |
|---|---|---|---|---|
| **1** | **1** | **0** | **0** | A |
| **0** | **0** | **0** | **0** | B |
| **1** | **1** | **0** | **0** | **Σ (-4)** |

*No Overflow*

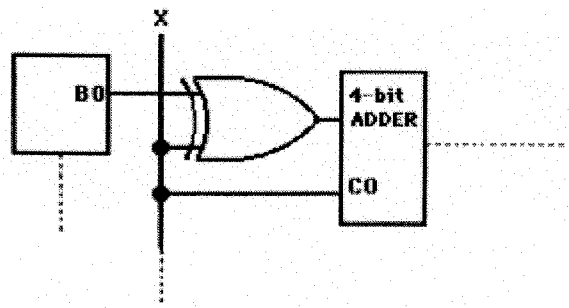
**6.30** (a) No Overflow (b) No Overflow (c) Overflow

**6.31** Three 74HC00 chips will have a total of twelve 2-input NAND gates.

I. Replace all of the 2-input AND gates (gates 1, 2, 3, 4, 5, 6, 7, and 8) with 2-input NAND gates.

II. Replace all of the 2-input OR gates (gates 9, 10, 11, and 12) with 2-input NAND gates.

**6.32** An EX-OR used as a controlled inverter with X as the control input can be connected as shown below for each B FF.




**6.33** (a) [S]=011 will select the A plus B operation: [A]=0110; [B]=0011; therefore, F=1001, $C_{N+4}$=0, OVR=1

(b) [S]=001 will select the B minus A operation: [A]=0110; [B]=0011; therefore, F=1101, $C_{N+4}$=0, OVR=0

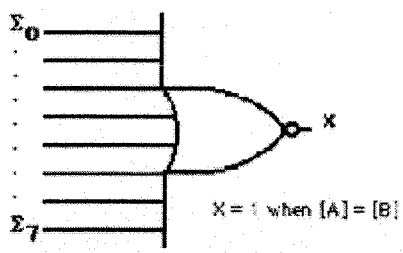
(c) [S]=010 will select the A minus B operation: [A]=0110; [B]=0011; therefore, F=0011, $C_{N+4}$=1, OVR=0

**6.34** [S] = 100 will select the Exclusive-OR operation: [A] = XXXX; [B] = 1111.
Therefore, F = $\overline{[A]}$.

**6.35** (a) [S] = 110 will select the AND operation: [A] = 10101100; [B] = 00001111; therefore, $\Sigma$=00001100.

(b) [S] = 100 will select the Exclusive-OR operation:[A] = 11101110; [B] = 00110010; therefore, $\Sigma$=11011100.

**6.36** [S] = 100 to select the Exclusive-OR operation. Thus, the $\Sigma$ outputs will be zero when [A] = [B]. The NORed result of the output sums $\Sigma_0$-$\Sigma_7$ will indicate whether or not the binary numbers are equal (X=1). The output X of the NOR gate is HIGH when [A]=[B].



**6.37** (a) After 0010 is transferred into the A register, [A] = 0010. After [A] is added to 0011 the result should be 0101. However, because bit A2 is stuck LOW the final results in [A] = 0001.

(b) [A] = 1010

(c) After 0111 is transferred into the A register, [A] = 0011. After [A] is added to 0011 the result should be 0110. However, because bit A2 is stuck LOW the final results in [A] = 0010.

(d) [A] = 1011

(e) After 1001 is transferred into the A register, [A] = 1001. After [A] is added to 0011 the result should be 1100. However, because bit A2 is stuck LOW the final results in [A] = 1000.

**6.38** The technician most likely connected input $C_0$ of the 4-bit parallel adder to the ADD signal instead of the SUB signal.

**6.39** (a) B[3..0] would all be HIGH.

(b) $C_0$ would be HIGH.

(c) Σ [3..0] would always contain the value from the accumulator, which would never change because it is adding to the accumulator $1111_2$ plus a carry in of 1 resulting in the same value that was in the accumulator.

(d) $C_4$ will always be HIGH.

**6.40** **AHDL**

```
                a [7..0], b[7..0]      :INPUT;
                z[7..0]                :OUTPUT;
VHDL
PORT(           a, b                   :IN    BIT_VECTOR (7 DOWNTO 0;
                z                      :OUT   BIT_VECTOR (7 DOWNTO 0);
```

**6.41** (a) 000100 (b)10111111 (c) 1000100 (d) 1000000 (e) 0101110

**6.42** (a) a[3..0] = 0111 (b) b[0] = 0 (c) a[7] = 1

**6.43** (a) 0 (b) 1 (c) 0010110

**6.44** **AHDL**

```
z[6..0] = a[7..1]
z[7] = a[0];
```

**VHDL**

```
z(6..0) <= a(7..1);
z(7) <= a(0);
```

**6.45** **AHDL**

```
Z= (B "0000" , b[7..4])
```

**VHDL**

```
Z= B "0000" & b(7 DOWNTO 4);
```

**6.46**

```
ovr = a[3] & b[3] & !z[3]  #  !a[3] & !b[3] & z[3];

ovr = ((a(3) AND b(3)) AND NOTz(3))  OR  (NOTa(3) AND NOTb(3)) AND z(3));

%  ADDER in AHDL  %

SUBDESIGN prob6_46
(
    cin                 :INPUT;                 --Carry in
    a[3..0]             :INPUT;                 --augend
    b[3..0]             :INPUT;                 --addend
    s[3..0]             :OUTPUT;                --sum
    cout                :OUTPUT;                --carry OUT
    ovr                 :OUTPUT;                --overflow OUT

)
VARIABLE
    c[4..0]   :node;                            -- carry array is 5 bits long!

BEGIN
    c[0] = cin;
    s[] = a[] $ b[] $ c[3..0];                  --generate sum
    c[4..1] = (a[] & b[]) # (a[] & c[3..0]) # (b[] & c[3..0]);
    cout = c[4];                                --carry out
    ovr = a[3] & b[3] & !s[3]  #  !a[3] & !b[3] & s[3];

END;
```

```
-- ADDER IN VHDL
-- USING GROUPS of bits (arrays).
--constant N :integer:=3;
ENTITY prob6_46 IS
PORT(
        cin             :IN bit;
        a               :IN BIT_VECTOR(3 DOWNTO 0);
        b               :IN BIT_VECTOR(3 DOWNTO 0);
        s               :BUFFER BIT_VECTOR(3 DOWNTO 0);
        cout, ovr       :OUT bit);
END prob6_46;

ARCHITECTURE a OF prob6_46 IS
SIGNAL c :bit_vector (4 DOWNTO 0);                  --carries require 5 bit array

BEGIN
        c(0) <= cin;                                --Read the carry in to bit array
        s <= a XOR b  XOR c(3 DOWNTO 0);            -- Generate the sum bits
        c(4 DOWNTO 1) <= (a AND b) OR (a AND c(3 DOWNTO 0)) OR (b AND c(3 DOWNTO
0));
        cout <= c(4);                               -- output the carry of the MSB.
        ovr <= (a(3) AND b(3) AND (NOT s(3)))  OR  ((NOT a(3)) AND (NOT b(3)) AND s(3));
END a;
```

**6.47**

```
ovr <= c[4] $ c[3];

ovr <= c(4) XOR c(3);

-- ADDER IN VHDL
-- USING GROUPS of bits (arrays).
--constant N :integer:=3;
ENTITY prob6_47 IS
PORT(
    cin       :IN bit;
    a         :IN BIT_VECTOR(3 DOWNTO 0);
    b         :IN BIT_VECTOR(3 DOWNTO 0);
    s         :BUFFER BIT_VECTOR(3 DOWNTO 0);
    cout, ovr         :OUT bit);
END prob6_47;

ARCHITECTURE a OF prob6_47 IS
SIGNAL c :bit_vector (4 DOWNTO 0);                    --carries require 5 bit array

BEGIN
    c(0) <= cin;                                      --Read the carry in to bit array
    s <= a XOR b  XOR c(3 DOWNTO 0);                  -- Generate the sum bits
    c(4 DOWNTO 1) <= (a AND b) OR (a AND c(3 DOWNTO 0)) OR (b AND c(3 DOWNTO
0));
    cout <= c(4);                                     -- output the carry of the MSB.
    ovr <= c(4) XOR c(3);
END a;


%  ADDER in AHDL  %

SUBDESIGN prob6_47
(
    cin                   :INPUT;                     --Carry in
    a[3..0]               :INPUT;                     --augend
    b[3..0]               :INPUT;                     --addend
    s[3..0]               :OUTPUT;                    --sum
    cout                  :OUTPUT;                    --carry OUT
    ovr                   :OUTPUT;                    --overflow OUT

)
VARIABLE
    c[4..0]               :node;                      -- carry array is 5 bits long!

BEGIN
    c[0] = cin;
    s[] = a[] $ b[] $ c[3..0];                        --generate sum
    c[4..1] = (a[] & b[]) # (a[] & c[3..0]) # (b[] & c[3..0]);
    cout = c[4];                                      --carry out
    ovr =  c[3] $ c[4];

END;
```

**6.48**

```
-- ADDER IN VHDL
-- USING GROUPS of bits (arrays).
--constant N :integer:=3;
-- Answer to problem 6-48 This is a 4-bit accumulator like Figure 6-10
ENTITY prob6_48 IS
PORT(
    cin                     :IN bit;
    bin                     :IN BIT_VECTOR(3 DOWNTO 0);
    load, transfer, clear_not :INPUT BIT;
    a                       :OUT BIT_VECTOR(3 DOWNTO 0);
    cout                    :OUT bit
END prob6_48 ;

ARCHITECTURE a OF prob6_48  IS
SIGNAL c                    : BIT_VECTOR (4 DOWNTO 0); -- carries require 5 bit array
SIGNAL s                    : BIT_VECTOR(3 DOWNTO 0);
SIGNAL s, b                 : BIT_VECTOR(3 DOWNTO 0);
PROCESS (load, transfer, clear_not)

BEGIN
        IF (clear_not) THEN a <= "0000";
        ELSIF (transfer = 1 AND transfer'EVENT)THEN a = s;
        END IF;
        IF (load = '1' AND load'EVENT) THEN b <= bin;  -- load into B register
        END IF;
END PROCESS;
BEGIN
    c(0) <= cin;                                          -- Read the carry in to bit array
    s <= a XOR b  XOR c(3 DOWNTO 0);                      -- Generate the sum bits
    c(4 DOWNTO 1) <= (a AND b) OR (a AND c(3 DOWNTO 0)) OR (b AND c(3 DOWNTO
0));
    cout <= c(4);                                         -- output the carry of the MSB.
END a;


%  ADDER in AHDL
    Answer to problem 6-48 This is a 4-bit accumulator like Figure 6-10 %

SUBDESIGN prob6_48
(
    cin                     :INPUT;                       -- Carry in
    bin[3..0]               :INPUT;                       -- data from memory
    load                    :INPUT;                       -- load control
    transfer                :INPUT;                       -- transfer control
    clear_not               :INPUT;                       -- clear accumulator control
    a[3..0]                 :OUTPUT;                      -- Accumulator output
    cout                    :OUTPUT;                      -- Carry OUT
)
VARIABLE
    c[4..0]                 :node;                        -- carry array is 5 bits long!
    s[3..0]                 :node;
    a[3..0]                 :DFF;                         -- A register
    b[3..0]                 :DFF;                         -- B register

BEGIN
    b[].clk = load;
```

```
    b[].d = bin[];
    c[0] = cin;
    s[] = a[] $ b[] $ c[3..0];                          -- generate sum
    c[4..1] = (a[] & b[]) # (a[] & c[3..0]) # (b[] & c[3..0]);
    cout = c[4];                                        -- carry out
    a[].d = s[];                                        -- connect adder outputs to acc inputs
    a[].clk = transfer;
    a[].clrn = clear_not;
END;
```

**6.49**

```
-- ADDER IN VHDL
-- USING GROUPS of bits (arrays).

ENTITY  prob6_49.vhd IS
PORT(       cin                 :IN bit;
            a                   :IN BIT_VECTOR(11 DOWNTO 0);
            b                   :IN BIT_VECTOR(11 DOWNTO 0);
            s                   :OUT BIT_VECTOR(11 DOWNTO 0);
            cout                :OUT bit);
END prob6_49.vhd;

ARCHITECTURE a OF prob6_49.vhd IS
SIGNAL c :bit_vector (12 DOWNTO 0);                     -- carries require 5 bit array

BEGIN
    c(0) <= cin;                                        -- Read the carry in to bit array
    s <= a XOR b  XOR c(11 DOWNTO 0);                   -- Generate the sum bits
    c(12 DOWNTO 1) <= (a AND b) OR (a AND c(11 DOWNTO 0)) OR (b AND c(11 DOWNTO
0));
    cout <= c(12);                                      -- output the carry of the MSB.
END a;


%   -- ADDER IN AHDL
Problem 6_49 12-bit adder  %

SUBDESIGN prob6_49
(
    cin                         :INPUT;                 -- Carry in
    a[11..0]                    :INPUT;                 -- augend
    b[11..0]                    :INPUT;                 -- addend
    s[11..0]                    :OUTPUT;                -- sum
    cout                        :OUTPUT;                -- Carry OUT
)
VARIABLE
    c[12..0]                    :node;                  -- carry array is 5 bits long!

BEGIN
    c[0] = cin;
    s[] = a[] $ b[] $ c[11..0];                         -- generate sum
    c[12..1] = (a[] & b[]) # (a[] & c[11..0]) # (b[] & c[11..0]);
    cout = c[12];                                       -- carry out
END;
```

**6.50**

```
-- ADDER IN VHDL with variable number of bits
-- USING GROUPS of bits (arrays).
-- constant N :integer:=11;

PACKAGE const IS
    CONSTANT number_of_bits    :INTEGER:=12;                    -- set total number of bits
    CONSTANT n                 :INTEGER:=number_of_bits - 1; -- MSB index number
END const;

USE work.const.all;

ENTITY prob6_50 IS
PORT(
    cin                        :IN bit;
    a                          :IN BIT_VECTOR(n DOWNTO 0);
    b                          :IN BIT_VECTOR(n DOWNTO 0);
    s                          :OUT BIT_VECTOR(n DOWNTO 0);
    cout                       :OUT bit);
END prob6_50;

ARCHITECTURE a OF prob6_50 IS
SIGNAL c                       :bit_vector (n+1 DOWNTO 0);-- carries require 5 bit array

BEGIN
    c(0) <= cin;                                                -- Read the carry in to bit array
    s <= a XOR b  XOR c(n DOWNTO 0);                            -- Generate the sum bits
    c(n+1 DOWNTO 1) <= (a AND b) OR (a AND c(n DOWNTO 0)) OR (b AND c(n DOWNTO
0));
    cout <= c(n+1);                                             -- output the carry of the MSB.
END a;


%  ADDER in AHDL using a variable number of bits set for 12 bits [11..0]  %

CONSTANT number_of_bits = 12;                                   -- set total number of bits
CONSTANT n = number_of_bits - 1;                                -- n is highest bit index

SUBDESIGN prob6_50
(
    cin                        :INPUT;                          -- Carry in
    a[n..0]                    :INPUT;                          -- augend
    b[n..0]                    :INPUT;                          -- addend
    s[n..0]                    :OUTPUT;                         -- sum
    cout                       :OUTPUT;                         -- Carry OUT
)
    VARIABLE
    c[n+1..0]                  :node;                           -- carry array is 5 bits long!

BEGIN
    c[0] = cin;
    s[] = a[] $ b[] $ c[n..0];                                  -- generate sum
    c[n+1..1] = (a[] & b[]) # (a[] & c[n..0]) # (b[] & c[n..0]);
    cout = c[n+1];                                              -- carry out
END;
```

**6.51**

```
-- 74382 ALU IN VHDL
-- USING GROUPS of bits (arrays).

ENTITY prob6_51 IS
PORT(       cin             :IN bit;
            s               :IN BIT_VECTOR(2 DOWNTO 0);
            a               :IN BIT_VECTOR(3 DOWNTO 0);
            b               :IN BIT_VECTOR(3 DOWNTO 0);
            f               :OUT BIT_VECTOR(3 DOWNTO 0);
            cout            :OUT bit;
            ovr             :OUT BIT);
END prob6_51 ;

ARCHITECTURE a OF prob6_51  IS
SIGNAL c                    :bit_vector (4 DOWNTO 0);       -- carries require 5 bit array

BEGIN
PROCESS (a,b,s)
BEGIN
CASE s IS
    WHEN "000" =>
            f <= "0000";
            cout <= '0';
            ovr <= '0';
    WHEN B"001" =>
            c(0) <= cin;
            f <= (NOT a) XOR b XOR c(3 DOWNTO 0);       -- generate sum
            c(4 DOWNTO 1) <= ((NOT a) AND b) OR ((NOT A) AND c(3 DOWNTO 0)) OR (b
            AND c(3 DOWNTO 0));
            cout <= c(4);                               -- carry out
            ovr <= c(4) XOR c(3);                       -- generate signed overflow indicator
    WHEN B"010"  =>
            c(0) <= cin;
            f <= a XOR NOT b XOR c(3 DOWNTO 0);         -- generate sum
            c(4 DOWNTO 1) <= (a AND NOT b) OR (a AND c(3 DOWNTO 0)) OR (NOT b
            AND c(3 DOWNTO 0));
            cout <= c(4);                               -- carry out
            ovr <= c(4) XOR c(3);                       -- generate signed overflow indicator
    WHEN B"011"  =>
            c(0) <= cin;
            f <= a XOR b XOR c(3 DOWNTO 0);     -- generate sum
            c(4 DOWNTO 1) <= (a AND b) OR (a AND c(3 DOWNTO 0)) OR (b AND c(3
            DOWNTO 0));
            cout <= c(4);                               -- carry out
            ovr <= c(4) XOR c(3);                       -- generate signed overflow indicator
    WHEN B"100"  =>
            f <= a XOR b;
            cout <= '0';
            ovr <= '0';
    WHEN B"101"  =>
            f <= a OR b;
            cout <= '0';
            ovr <= '0';
    WHEN B"110"  =>
            f <= a AND b;
            cout <= '0';
```

```
            ovr <= '0';
        WHEN B"111" =>
            f <= "1111";
            cout <= '0';
            ovr <= '0';
END CASE;
END PROCESS;
END a;

%   74382 ALU in AHDL  %

SUBDESIGN prob6_51
(
    cin                 :INPUT;             -- Carry in
    s[2..0]             :INPUT;             -- select operation
    a[3..0]             :INPUT;             -- operand A
    b[3..0]             :INPUT;             -- operand B
    f[3..0]             :OUTPUT;            -- result
    cout                :OUTPUT;            -- Carry OUT
    ovr                 :OUTPUT;            -- overflow
)
VARIABLE
    c[4..0]  :node;                         -- carry array is 5 bits long!
BEGIN
CASE s[] IS
    WHEN B"000"    =>
            f[] = 0;

    WHEN B"001"    =>
            c[0] = cin;
            f[] = !a[] $ b[] $ c[3..0];             -- generate sum
            c[4..1] = (!a[] & b[]) # (!a[] & c[3..0]) # (b[] & c[3..0]);
            cout = c[4];                            -- carry out
            ovr = c[4] $ c[3];                      -- generate signed overflow indicator
    WHEN B"010"   =>
            c[0] = cin;
            f[] = a[] $ !b[] $ c[3..0];             -- generate sum
            c[4..1] = (a[] & !b[]) # (a[] & c[3..0]) # (!b[] & c[3..0]);
            cout = c[4];                            -- carry out
            ovr = c[4] $ c[3];                      -- generate signed overflow indicator
    WHEN B"011"   =>
            c[0] = cin;
            f[] = a[] $ b[] $ c[3..0];              -- generate sum
            c[4..1] = (a[] & b[]) # (a[] & c[3..0]) # (b[] & c[3..0]);
            cout = c[4];                            -- carry out
            ovr = c[4] $ c[3];                      -- generate signed overflow indicator
    WHEN B"100"   =>
            f[] = a[] $ b[];
            cout = GND;
            ovr = GND;
    WHEN B"101"   =>
            f[] = a[] # b[];
            cout = GND;
            ovr = GND;
    WHEN B"110"   =>
            f[] = a[] & b[];
```

```
            cout = GND;
            ovr = GND;
    WHEN B"111" =>
            f[] = B"1111";
            cout = GND;
            ovr = GND;
END CASE;
END;
```

**6.52**

(a) ***Full Adder*** - Logic circuit with three inputs and two outputs. The inputs are a carry bit ($C_{IN}$) from a previous stage, a bit from the Augend, and a bit from the addend, respectively. The outputs are the sum bit produced by the addition of the bit from the addend with the bit from the Augend and the resulted carry ($C_{OUT}$) bit which will be added to the next stage.

(b) ***2's-Complement Form*** - Result obtained when a 1 is added to the least significant bit position of a binary number in the 1's-complement form.

(c) ***Arithmetic/Logic Unit*** - Digital circuit used in computers to perform various arithmetic and logic operations.

(d) ***Sign Bit*** - Binary bit that is added to the leftmost position of a binary number to indicate whether that number represents a positive or a negative quantity.

(e) ***Overflow*** - When in the process of adding signed binary numbers a 1 is generated from the MSB position of the number into the sign bit position.

(f) ***Accumulator*** - Principal register of an Arithmetic Logic Unit (ALU).

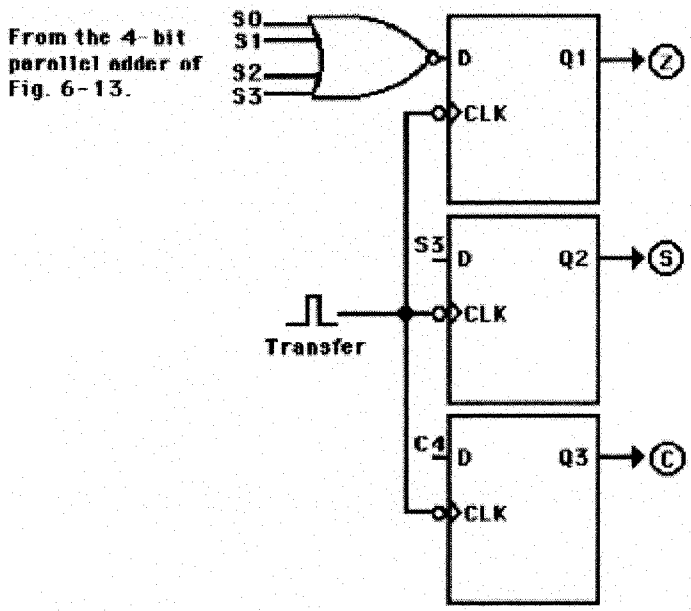
(g) ***Parallel Adder*** - Digital circuit made from full adders and used to add all the bits from the addend and the Augend together and simultaneously.

(h) ***Look-Ahead Carry*** - Ability of some parallel adders to predict, without having to wait for the carry to propagate through the full adders, whether or not a carry bit ($C_{OUT}$) will be generated as a result of the addition, thus reducing the overall propagation delays.

(i) ***Negation*** (2's complementing)- It's the operation of converting a positive binary number to its negative equivalent or a negative binary number to its positive equivalent.

(j) ***B-Register*** - One of two flip-flop registers used by the ALU (Arithmetic-Logic Unit).

**6.53**




**6.54** $01001001_2 = 00000000\underline{0}1001001_2 = +73_{10}$
$10101110_2 = 11111111\underline{1}0101110_2 = -82_{10}$

**6.55** The general rule used to convert 8-bit to 16-bit signed binary numbers is as follows:

1. If the signed bit of the 8-bit signed number is positive (0), then 8 more 0s are added in front of the 8-bit number thereby, making it a 16-bit number with the same sign as the original 8-bit number.

2. If the signed bit of the 8-bit signed number is negative (1), then 8 more 1s are added in front of the 8-bit number thereby, making it a 16-bit number with the same sign as the original 8-bit number.

## CHAPTER SEVEN - Counters and Registers

**7.1** (a) 250 kHz; 50% (b) same as (a) (c) 1 MHz (d) 32

**7.2** Need to divide by 64; use MOD-64, 6-bit counter

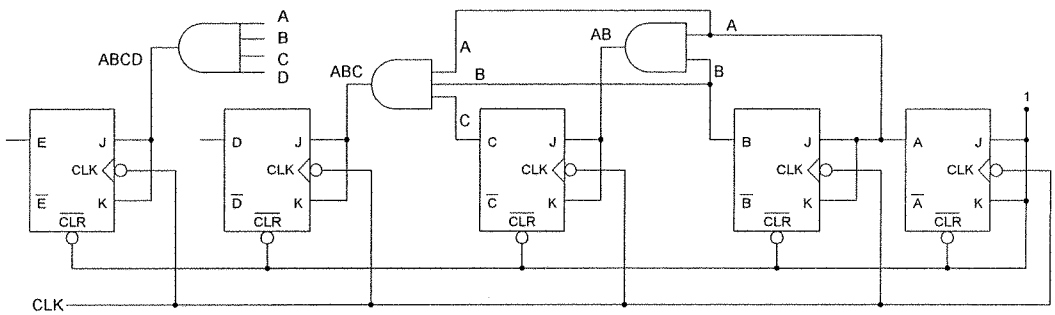
**7.3** $10000_2$

**7.4** (a) 1024 (b) 250 Hz (c) 50% (d) 3E8

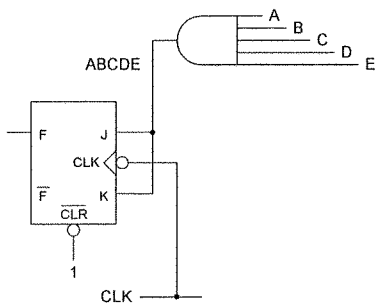
**7.5** 1000 and 0000 states never occur

**7.6** (a) 12.5 MHz (b) 8.33 MHz

**7.7** (a) See schematic (b) 33 MHz

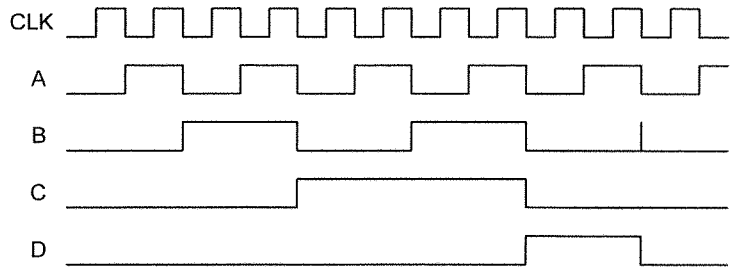



**7.8** (a) Add one more FF & gate to Problem 7-7(a) schematic (b) 33 MHz



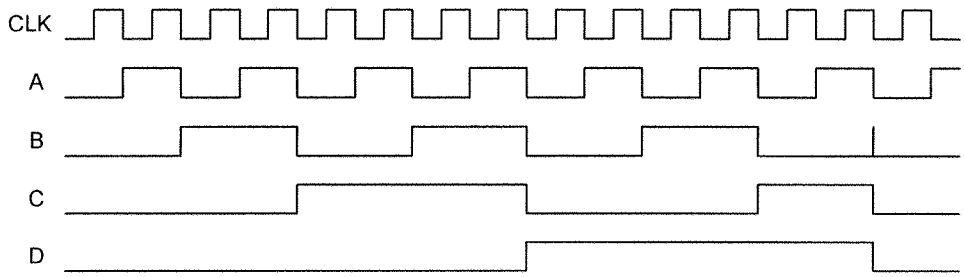
**7.9**

Frequency at D = 100 Hz



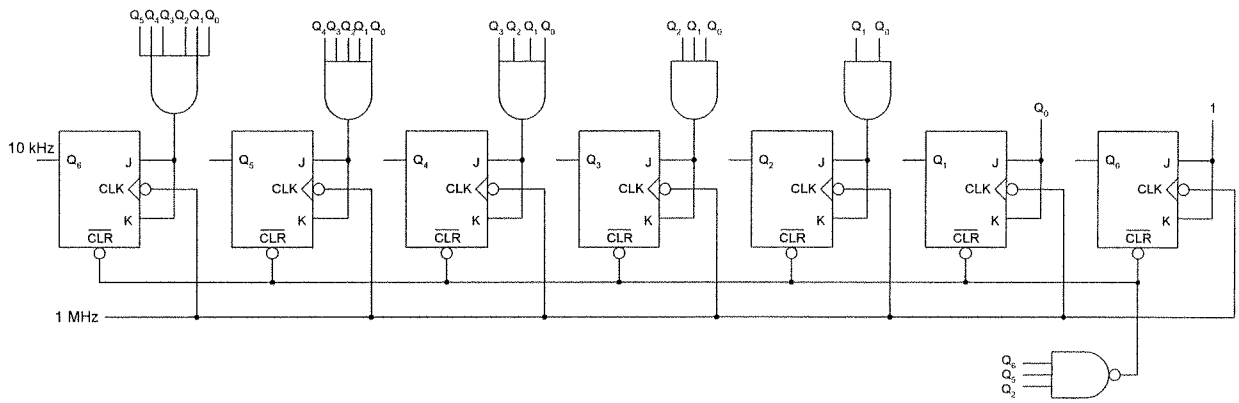

**7.10**

Frequency at D = 71.4 Hz

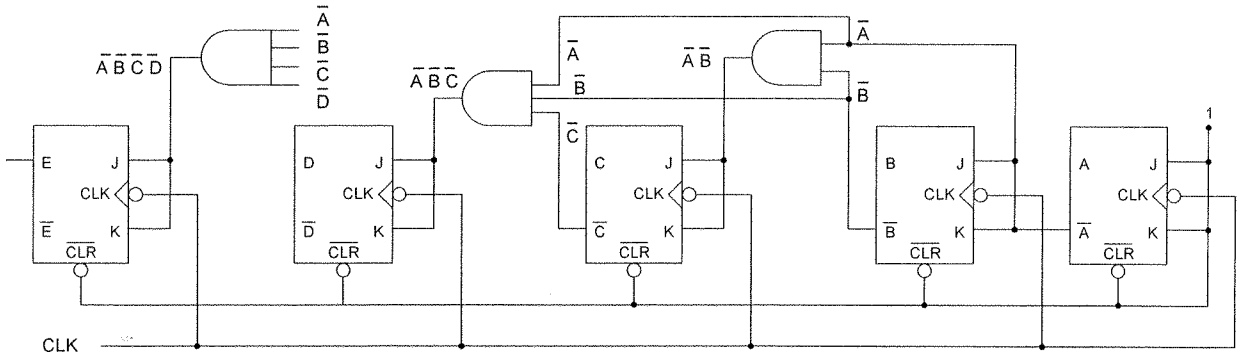



**7.11** Replace 4-input NAND with a 3-input NAND driving all FF CLRs and whose inputs are Q5, Q4, and Q1.

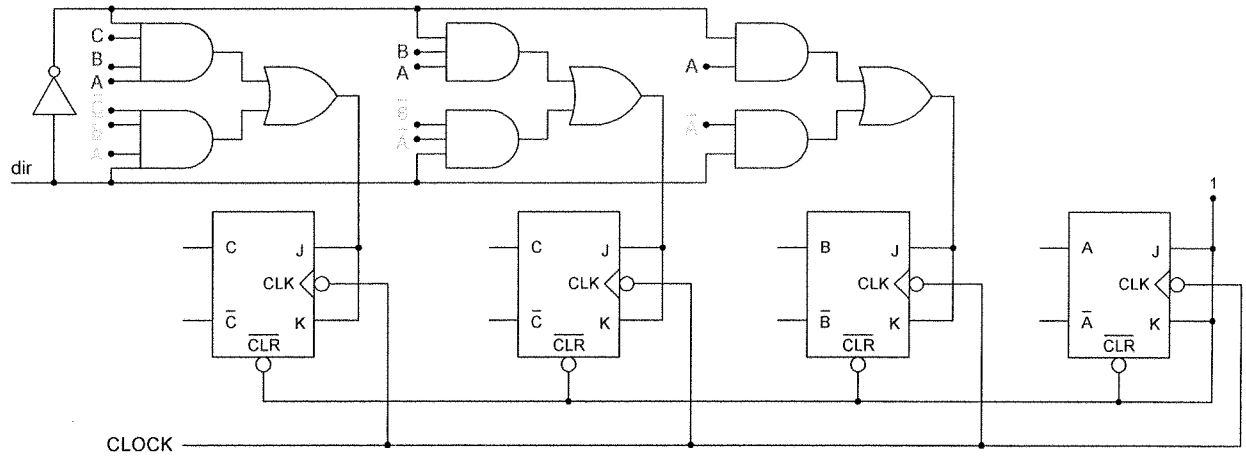
**7.12**



**7.13**

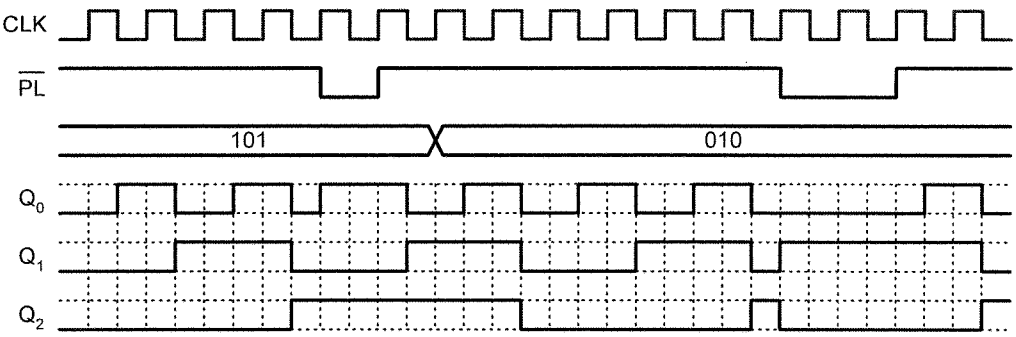



**7.14**

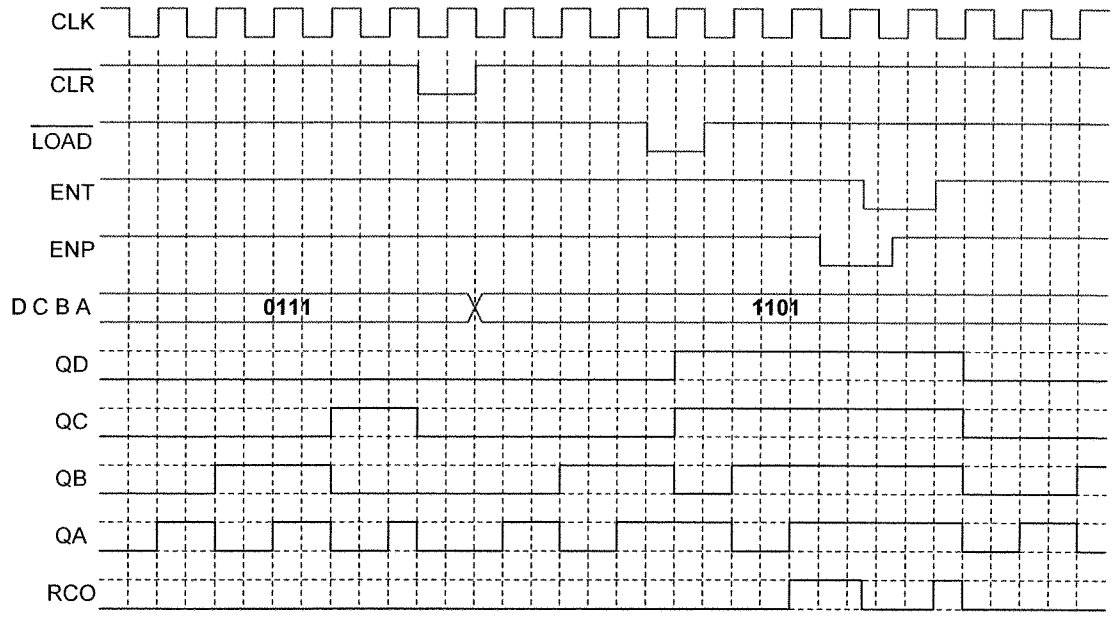



**7.15** Counter switches states between 000 and 111 on each clock pulse.

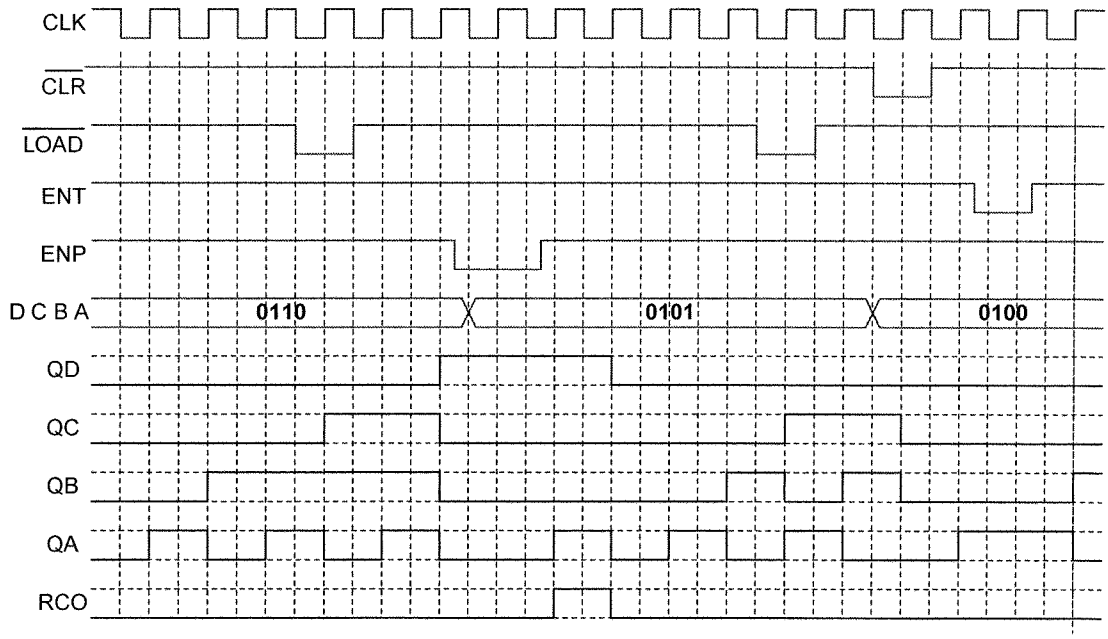
**7-16**



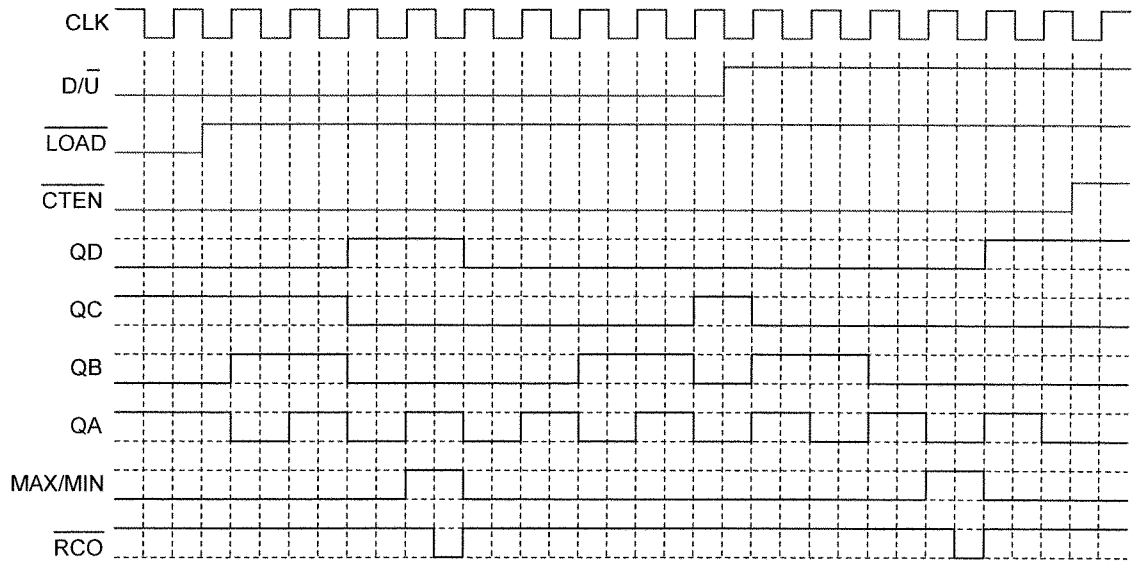
**7.17**


CLK
CLR
LOAD
ENT
ENP
D C B A
0111
1101
QD
QC
QB
QA
RCO

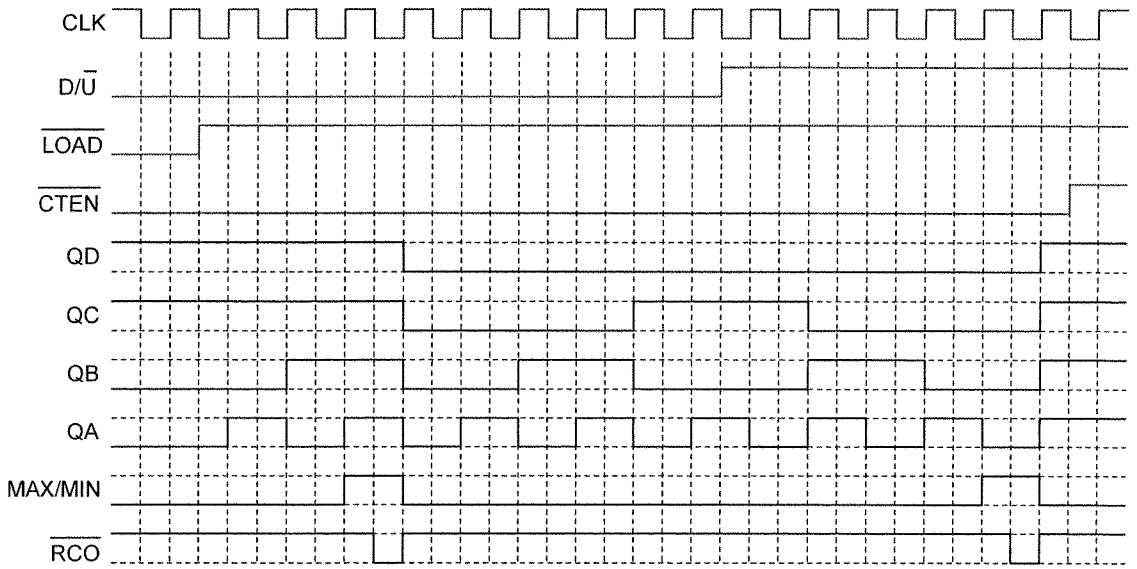
**7.18**


CLK
CLR
LOAD
ENT
ENP
D C B A
0110
0101
0100
QD
QC
QB
QA
RCO

**7.19**




**7.20**



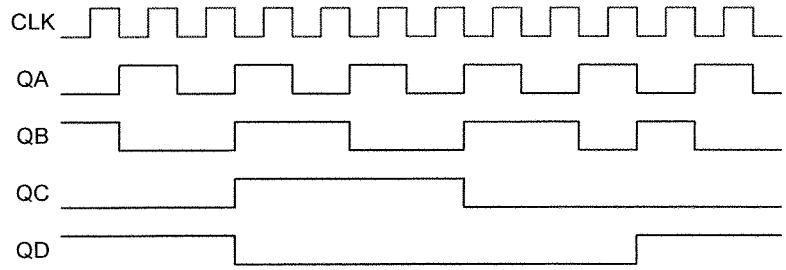

**7.21**

(a) 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, & repeat
(b) MOD-12
(c) frequency at QD (MSB) is 1/12 of CLK frequency
(d) 33.3%

**7.22**

(a) 0000, 0001, 0010, 0011, 0100, 0101, 0110, 1001, 1010, 1011, 1100, 1101, 1110, 0001, & repeat
(b) MOD-12
(c) frequency at QD (MSB) is 1/12 of CLK frequency
(d) 50%

**7.23** (a)

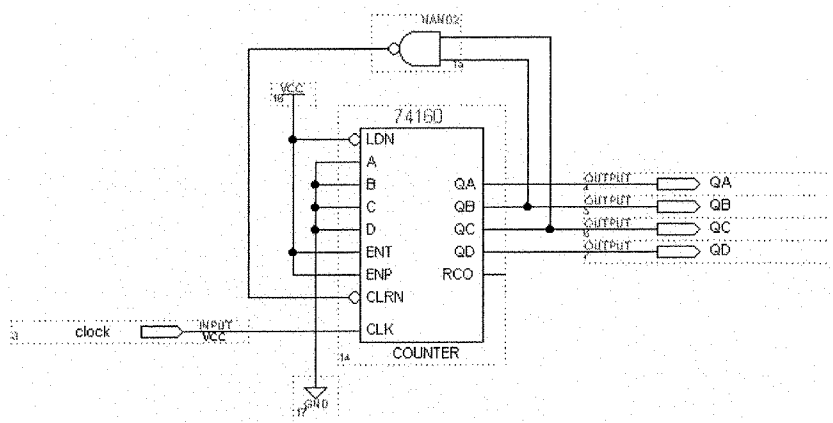



(b) MOD-10
(c) 10 down to 1
(d) Can produce MOD-10, but not same sequence.

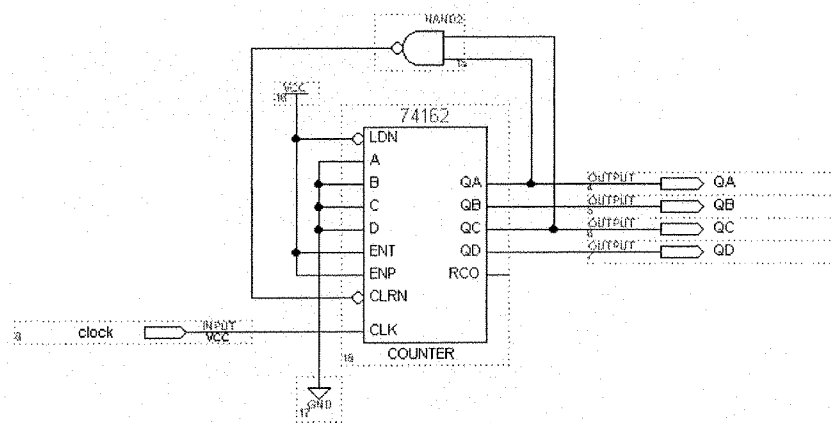
**7.24** (a) Output will be 0000 as long as $\overline{\text{START}}$ is LOW.
(b) Counter will count from 0000 up to 1001 on each CLK pulse and stop at 1001.
(c) MOD-10; it is a self-stopping counter not a recycling counter.

**7.25**

(a)

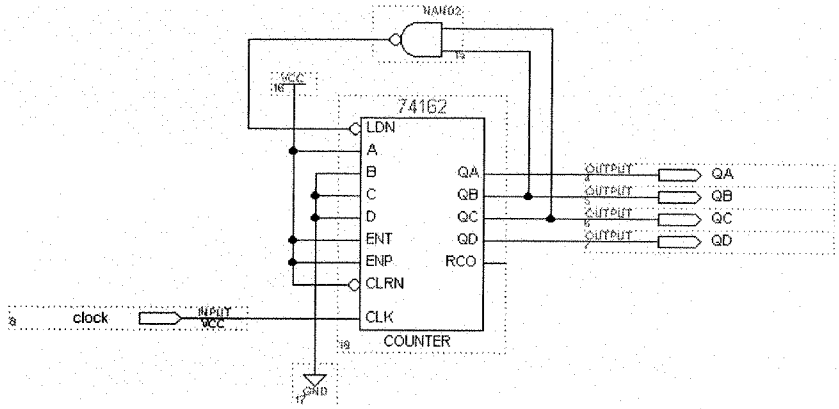



(b)

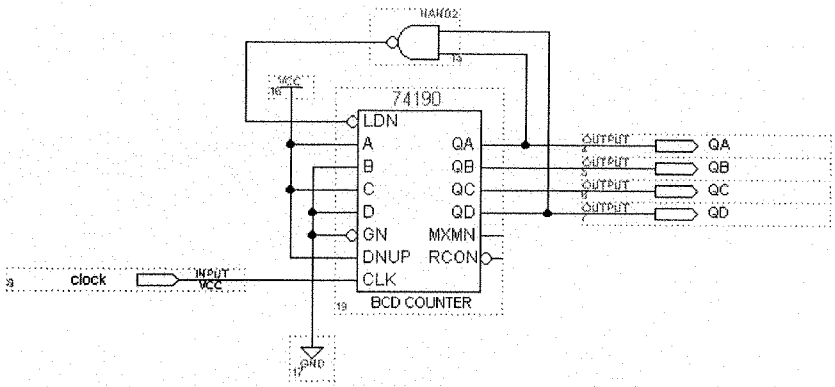


**7.26**

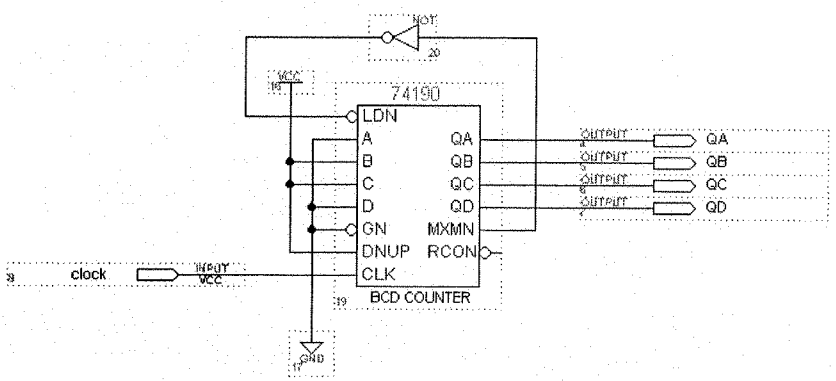
(a)


NAND2
74162
LDN
A
B
C
D
ENT
ENP
CLRN
CLK
QA
QB
QC
QD
RCO
clock
INPUT
VCC
COUNTER
GND
OUTPUT

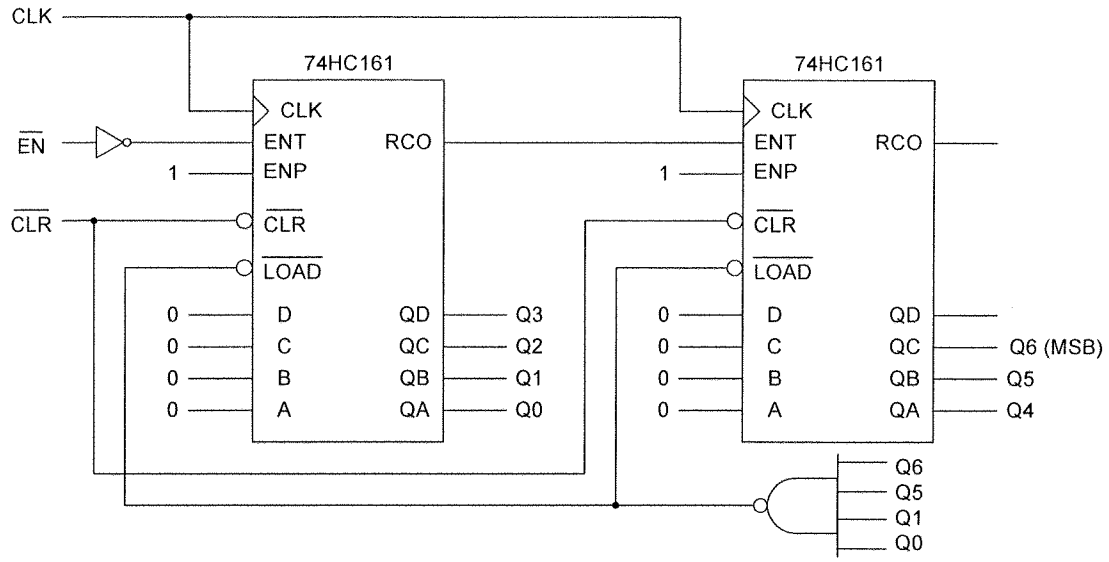
(b)


NAND2
74190
LDN
A
B
C
D
GN
DNUP
CLK
QA
QB
QC
QD
MXMN
RCON
clock
INPUT
VCC
BCD COUNTER
GND
OUTPUT

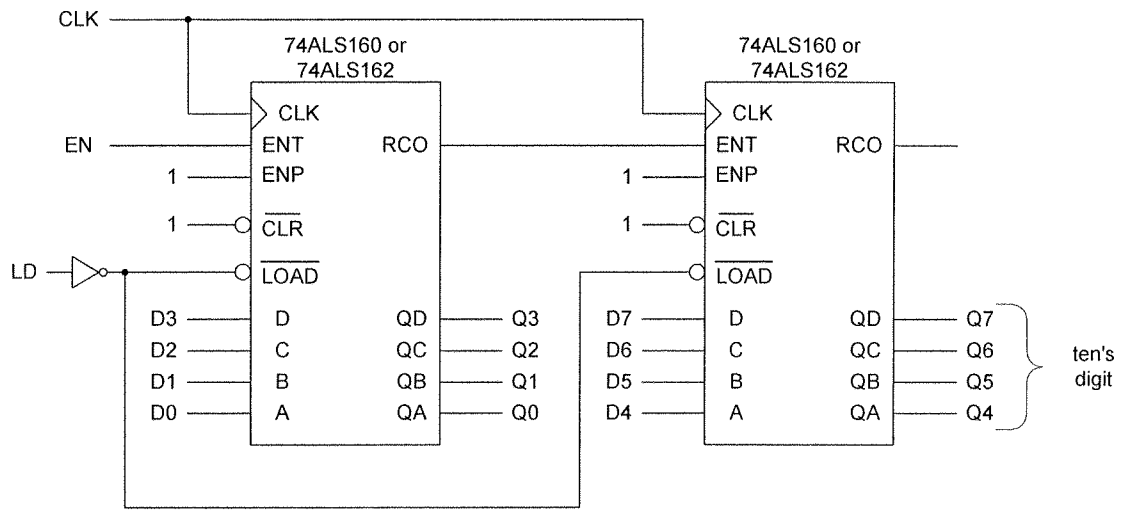
(c)


NOT
74190
LDN
A
B
C
D
GN
DNUP
CLK
QA
QB
QC
QD
MXMN
RCON
clock
INPUT
VCC
BCD COUNTER
GND
OUTPUT

**7.27**




**7.28**




**7.29**

| Output: | QA | QB | QC | QD | RCO |
|---|---|---|---|---|---|
| Frequency: | 3 MHz | 1.5 MHz | 750 kHz | 375 kHz | 375 kHz |
| Duty Cycle: | 50% | 50% | 50% | 50% | 6.25% |

**7.30**

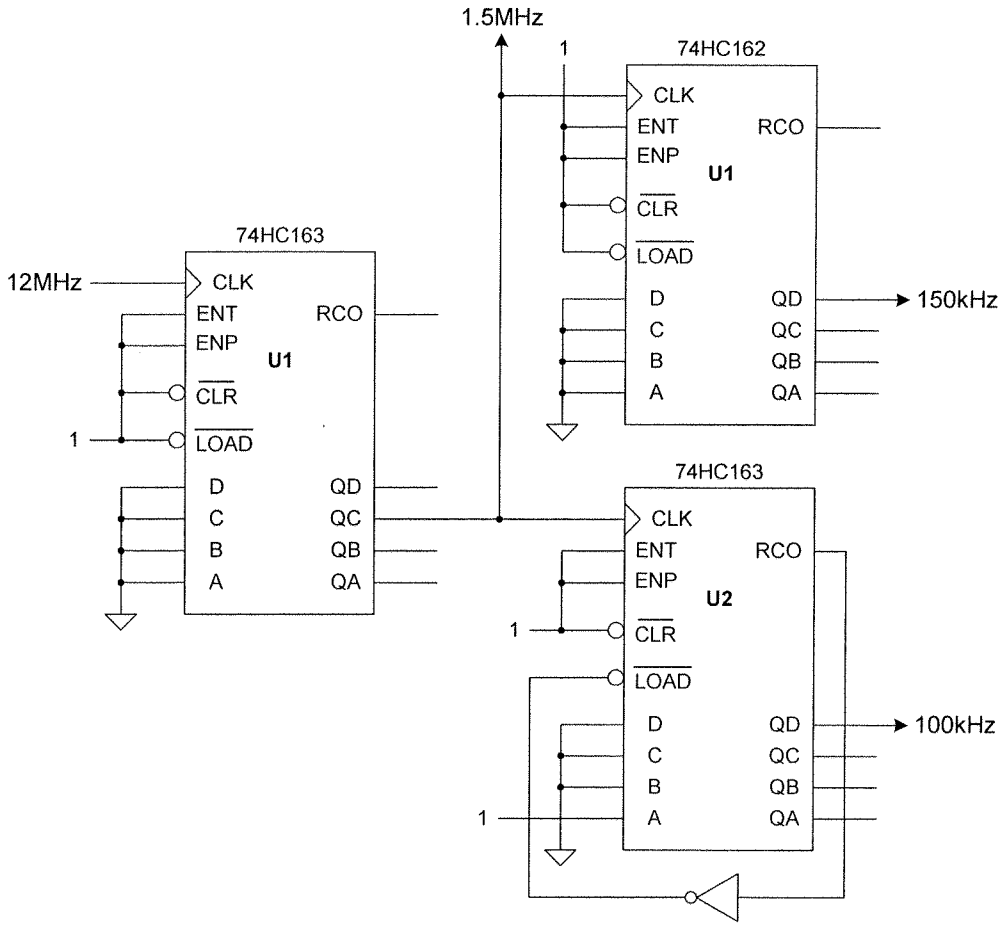
| Output: | QA | QC | QD | RCO |
|---|---|---|---|---|
| Frequency: | 3 MHz | 600 kHz | 600 kHz | 600 kHz |
| Duty Cycle: | 50% | 40% | 20% | 10% |

Output QB has an irregular pattern.

**7.31** Frequency at $f_{out1}$ = 500 kHz, at $f_{out2}$ = 100 kHz

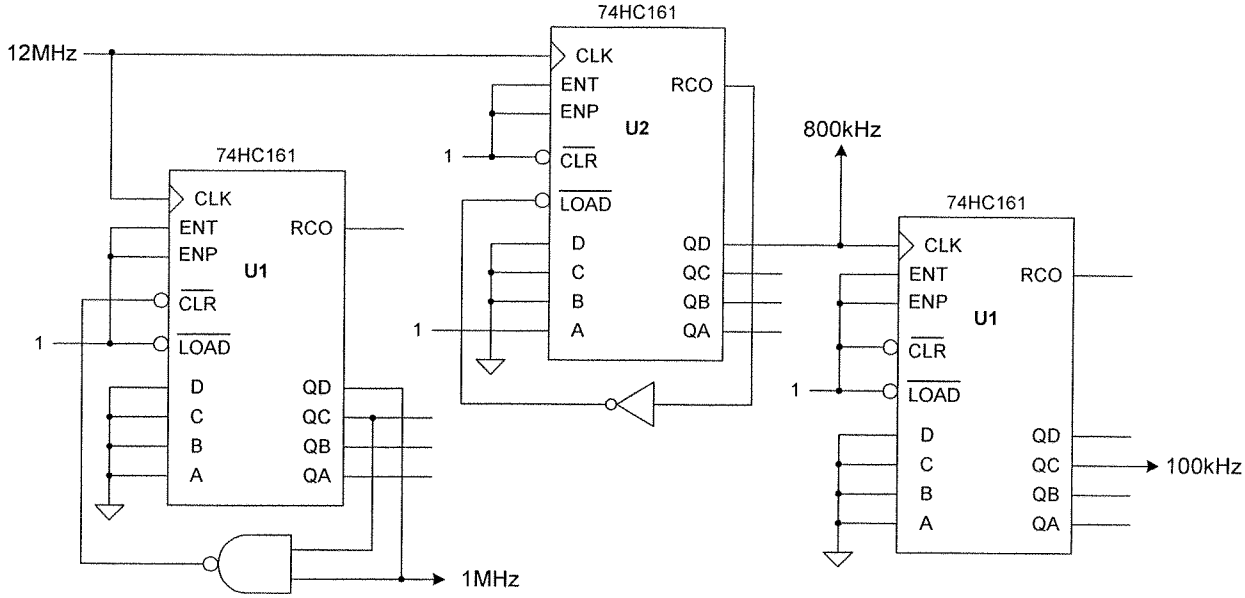
**7.32** Frequency at $f_{out1}$ = 100 kHz, at $f_{out2}$ = 10 kHz

**7.33** 12M/8 = 1.5M  1.5M/10 = 150k  1.5M/15 = 100k

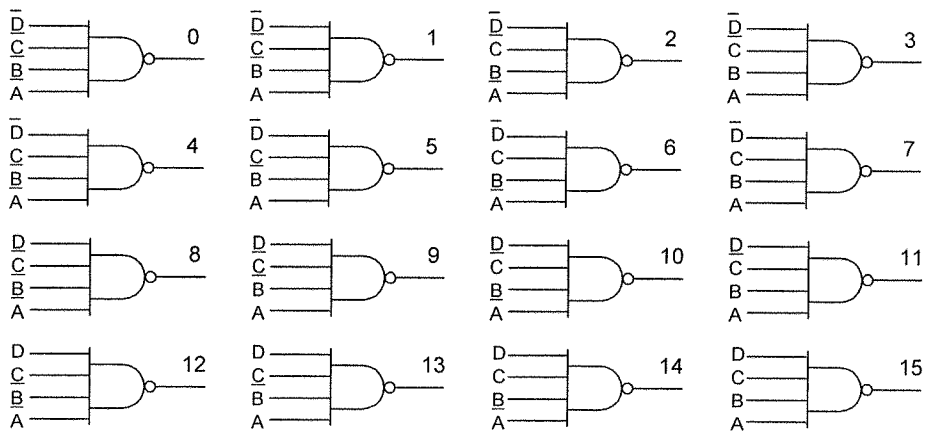


**7.34**

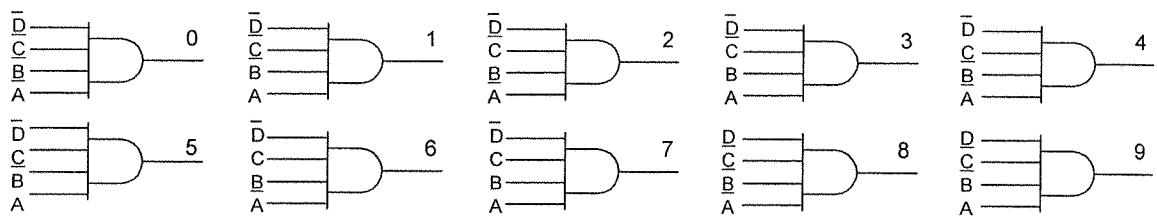
12M/12 = 1M  12M/15 = 800k  800k/8 = 100k



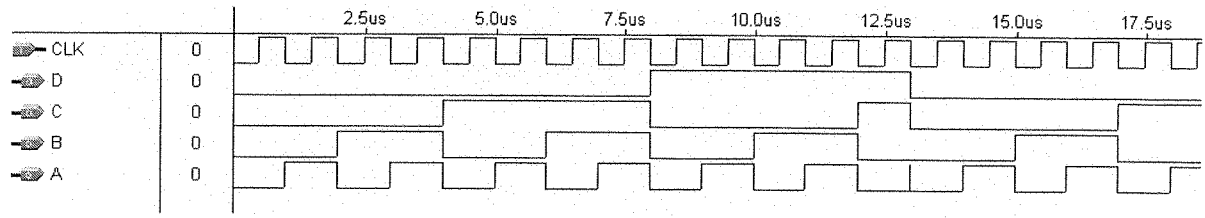

**7.35**



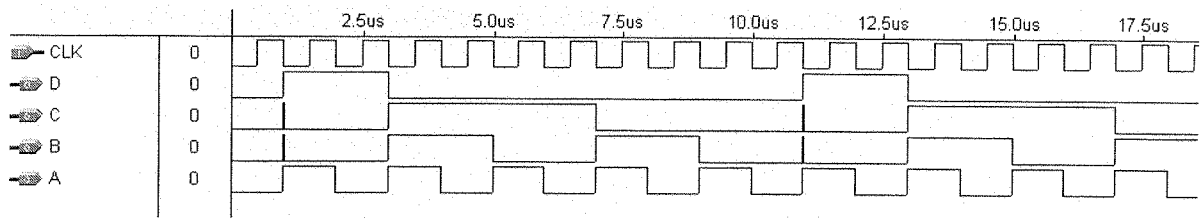

**7.36**



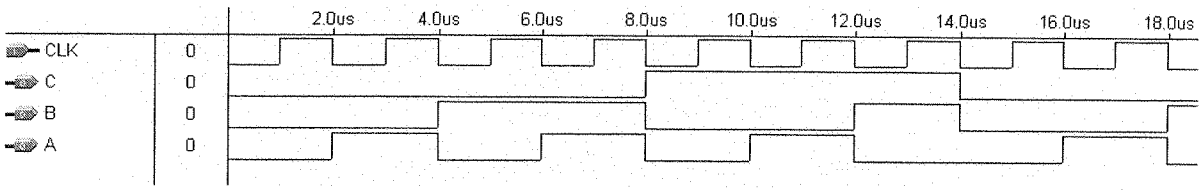
**7.37** MOD-13 counter.



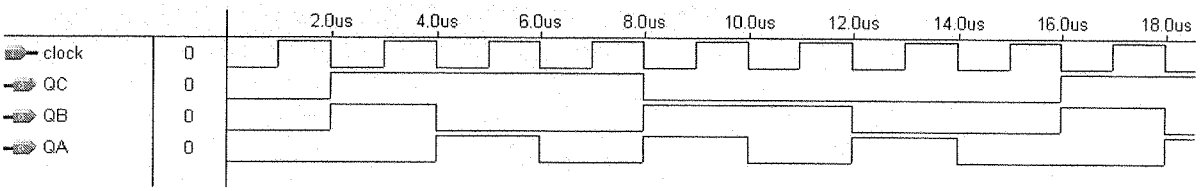

**7.38** MOD-10 down counter.



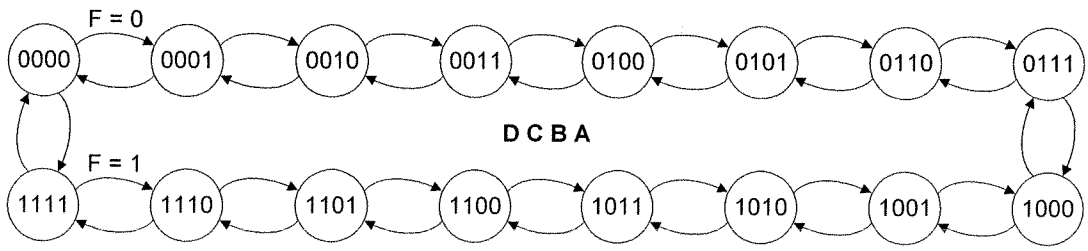

**7.39** MOD-7 counter.



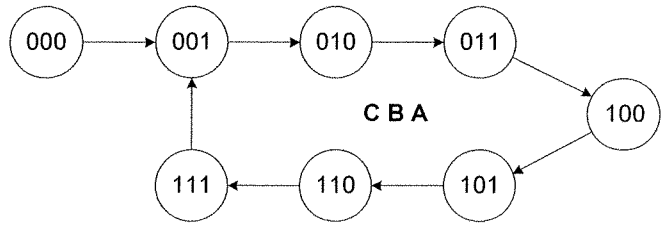

**7.40** MOD-7 down counter.




**7.41** MOD-16 up/down counter.



**7.42** MOD-7, self-correcting counter




**7.43**

(a) $JA = B\overline{C},\ KA = 1,\ JB = CA + \overline{C}\,\overline{A},\ KB = 1,\ JC = B\overline{A},\ KC = B + \overline{A}$

(b) $JA = B\overline{C},\ KA = 1,\ JB = KB = 1,\ JC = KC = B$

**7.44**

(a) $JA = \overline{C}B + C\overline{B}, KA = 1,\ JB = C\overline{A},\ KB = C + \overline{A},\ JC = \overline{B}\,\overline{A},\ KC = 1$

(b) $JA = B + C\ , KA = 1,\ JB = C,\ KB = \overline{A},\ JC = \overline{B}\,\overline{A},\ KC = 1$ (self-correcting)

**7.45**

$JA = KA = 1,\ JB = C\overline{A} + D\overline{A},\ KB = \overline{A},\ JC = D\overline{A},\ KC = \overline{A}\,\overline{B},\ JD = \overline{C}\,\overline{B}\,\overline{A},\ KD = \overline{A}$

**7.46**

$JA = \overline{C}B + DC + \overline{D}\,\overline{B},\ KA = 1,\ JB = D\overline{A} + \overline{D}A,\ KB = C + \overline{D}A + D\overline{A},$
$JC = \overline{A}\,\overline{B}D + \overline{D}BA,\ KC = AB + B\overline{D} + D\overline{B}\,\overline{A}$

**7.47**

$DA = \overline{A},\ DB = BA + \overline{B}\,\overline{A},\ DC = CA + CB + \overline{C}\,\overline{B}\,\overline{A}$

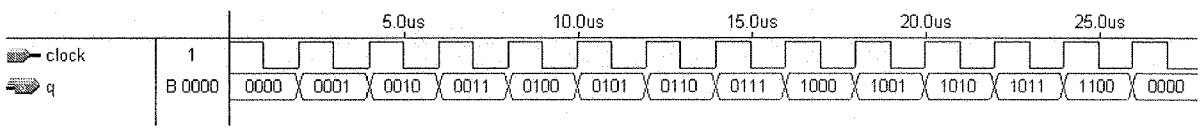
**7.48**

$DA = \overline{A},\ DB = \overline{B}A + B\overline{A},\ DC = C\overline{A} + C\overline{B} + \overline{D}BA,\ DD = D\overline{B} + D\overline{A} + CBA$

**7.49**

```
SUBDESIGN  mod13_ahdl
(     clock             :INPUT;
      q[3..0]           :OUTPUT;    )
VARIABLE
      q[3..0]           :DFF;
BEGIN
      q[].clk  = clock;
      IF  q[].q == 12  THEN         -- check for terminal state
            q[].d = B"0000";        -- recycle
      ELSE
            q[].d = q[].q + 1;      -- increment
      END IF;
END;
```

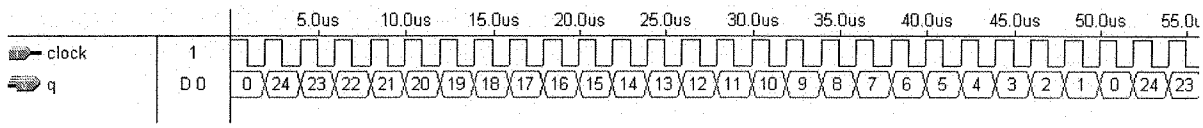
```
ENTITY mod13_vhdl IS
PORT (      clock       :IN BIT;
            q           :OUT INTEGER RANGE 0 TO 15    );
END mod13_vhdl;
ARCHITECTURE vhdl OF mod13_vhdl IS
BEGIN
      PROCESS (clock)
      VARIABLE  counter :INTEGER RANGE 0 TO 15;
      BEGIN
            IF (clock'EVENT AND clock = '1') THEN
                  IF  (counter = 12)  THEN      -- terminal state?
                        counter := 0;           -- recycle
                  ELSE  counter := counter + 1; -- increment
                  END IF;
            END IF;
            q <= counter;
      END PROCESS;
END vhdl;
```



**7.50**

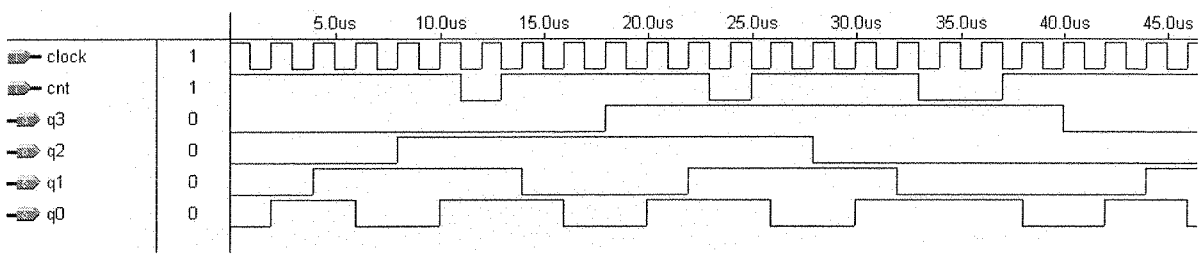
```
SUBDESIGN  mod25_ahdl
(     clock              :INPUT;
      q[4..0]            :OUTPUT;    )
VARIABLE
      q[4..0]            :DFF;
BEGIN
      q[].clk  = clock;
      IF  q[].q == 0  THEN          -- terminal state?
            q[].d = B"11000";       -- recycle
      ELSE
            q[].d = q[].q - 1;      -- decrement
      END IF;
END;
```

```
ENTITY mod25_vhdl IS
PORT (      clock       :IN BIT;
            q           :OUT INTEGER RANGE 31 DOWNTO 0      );
END mod25_vhdl;
ARCHITECTURE vhdl OF mod25_vhdl IS
BEGIN
      PROCESS (clock)
      VARIABLE  counter :INTEGER RANGE 31 DOWNTO 0;
      BEGIN
            IF (clock'EVENT AND clock = '1') THEN
                  IF  (counter = 0)  THEN       -- terminal state?
                        counter := 24;          -- recycle
                  ELSE  counter := counter - 1; -- decrement
                  END IF;
            END IF;
            q <= counter;
      END PROCESS;
END vhdl;
```



**7.51**

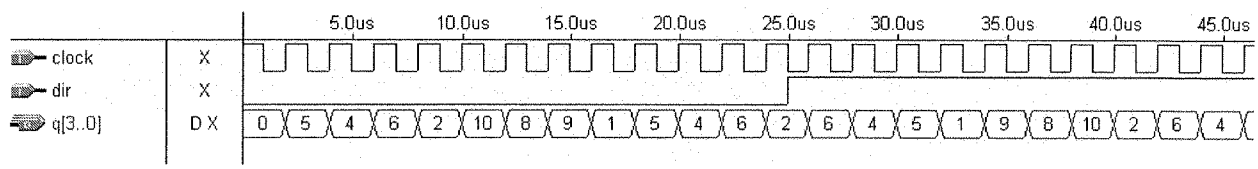
```
SUBDESIGN  gray_ahdl
(   clock, cnt :INPUT;
    q[3..0]    :OUTPUT;       )
VARIABLE
gray:       MACHINE OF BITS (q[3..0])
    WITH STATES (s0 = B"0000", s1 = B"0001", s2 = B"0011", s3 = B"0010",
           s4 = B"0110", s5 = B"0111", s6 = B"0101", s7 = B"0100",
           s8 = B"1100", s9 = B"1101", s10 = B"1111", s11 = B"1110",
           s12 = B"1010", s13 = B"1011", s14 = B"1001", s15 = B"1000");
BEGIN
    gray.clk = clock;
    IF cnt  THEN
        CASE  gray  IS
            WHEN  s0    =>      gray = s1;
            WHEN  s1    =>      gray = s2;
            WHEN  s2    =>      gray = s3;
            WHEN  s3    =>      gray = s4;
            WHEN  s4    =>      gray = s5;
            WHEN  s5    =>      gray = s6;
            WHEN  s6    =>      gray = s7;
            WHEN  s7    =>      gray = s8;
            WHEN  s8    =>      gray = s9;
            WHEN  s9    =>      gray = s10;
            WHEN  s10   =>      gray = s11;
            WHEN  s11   =>      gray = s12;
            WHEN  s12   =>      gray = s13;
            WHEN  s13   =>      gray = s14;
            WHEN  s14   =>      gray = s15;
            WHEN  s15   =>      gray = s0;
        END CASE;
    ELSE
        CASE  gray  IS
            WHEN  s0    =>      gray = s0;
            WHEN  s1    =>      gray = s1;
            WHEN  s2    =>      gray = s2;
            WHEN  s3    =>      gray = s3;
            WHEN  s4    =>      gray = s4;
            WHEN  s5    =>      gray = s5;
            WHEN  s6    =>      gray = s6;
            WHEN  s7    =>      gray = s7;
            WHEN  s8    =>      gray = s8;
            WHEN  s9    =>      gray = s9;
            WHEN  s10   =>      gray = s10;
            WHEN  s11   =>      gray = s11;
            WHEN  s12   =>      gray = s12;
            WHEN  s13   =>      gray = s13;
            WHEN  s14   =>      gray = s14;
            WHEN  s15   =>      gray = s15;
        END CASE;
    END IF;
END;
```



```
ENTITY gray_vhdl IS
PORT ( clock, cnt       :IN BIT;
       q                :OUT BIT_VECTOR (3 DOWNTO 0)  );
END gray_vhdl;
ARCHITECTURE vhdl OF gray_vhdl IS
BEGIN
PROCESS (clock)
VARIABLE seq            :BIT_VECTOR (3 DOWNTO 0);
BEGIN
    IF (clock'EVENT AND clock = '1')     THEN
       IF (cnt = '1')   THEN
          CASE seq IS
             WHEN "0000" =>             seq := "0001";
             WHEN "0001" =>             seq := "0011";
             WHEN "0011" =>             seq := "0010";
             WHEN "0010" =>             seq := "0110";
             WHEN "0110" =>             seq := "0111";
             WHEN "0111" =>             seq := "0101";
             WHEN "0101" =>             seq := "0100";
             WHEN "0100" =>             seq := "1100";
             WHEN "1100" =>             seq := "1101";
             WHEN "1101" =>             seq := "1111";
             WHEN "1111" =>             seq := "1110";
             WHEN "1110" =>             seq := "1010";
             WHEN "1010" =>             seq := "1011";
             WHEN "1011" =>             seq := "1001";
             WHEN "1001" =>             seq := "1000";
             WHEN "1000" =>             seq := "0000";
          END CASE;
       ELSE
          CASE seq IS
             WHEN "0000" =>             seq := "0000";
             WHEN "0001" =>             seq := "0001";
             WHEN "0011" =>             seq := "0011";
             WHEN "0010" =>             seq := "0010";
             WHEN "0110" =>             seq := "0110";
             WHEN "0111" =>             seq := "0111";
             WHEN "0101" =>             seq := "0101";
             WHEN "0100" =>             seq := "0100";
             WHEN "1100" =>             seq := "1100";
             WHEN "1101" =>             seq := "1101";
             WHEN "1111" =>             seq := "1111";
             WHEN "1110" =>             seq := "1110";
             WHEN "1010" =>             seq := "1010";
             WHEN "1011" =>             seq := "1011";
             WHEN "1001" =>             seq := "1001";
             WHEN "1000" =>             seq := "1000";
          END CASE;
       END IF;
    END IF;
    q <= seq;
END PROCESS;
END vhdl;
```

**7.52**

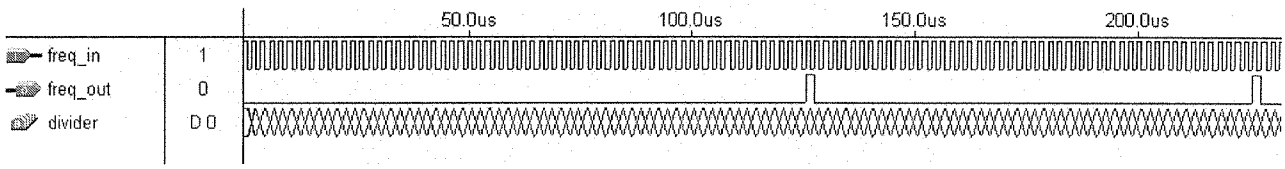
```
SUBDESIGN  stepper_ahdl
(
      clock, dir          :INPUT;
      q[3..0]             :OUTPUT;
)
VARIABLE
stepper                   :MACHINE OF BITS (q[3..0])
      WITH STATES (initial = B"0000", s1 = B"0101", s2 = B"0001",
            s3 = B"1001", s4 = B"1000", s5 = B"1010", s6 = B"0010",
            s7 = B"0110", s8 = B"0100");
BEGIN
stepper.clk = clock;
IF  dir == VCC    THEN
      CASE  stepper  IS
            WHEN  initial      =>    stepper = s1;
            WHEN  s1           =>    stepper = s2;
            WHEN  s2           =>    stepper = s3;
            WHEN  s3           =>    stepper = s4;
            WHEN  s4           =>    stepper = s5;
            WHEN  s5           =>    stepper = s6;
            WHEN  s6           =>    stepper = s7;
            WHEN  s7           =>    stepper = s8;
            WHEN  s8           =>    stepper = s1;
      END CASE;
ELSE
      CASE  stepper  IS
            WHEN  initial      =>    stepper = s1;
            WHEN  s1           =>    stepper = s8;
            WHEN  s2           =>    stepper = s1;
            WHEN  s3           =>    stepper = s2;
            WHEN  s4           =>    stepper = s3;
            WHEN  s5           =>    stepper = s4;
            WHEN  s6           =>    stepper = s5;
            WHEN  s7           =>    stepper = s6;
            WHEN  s8           =>    stepper = s7;
      END CASE;
END IF;
END;
```



```
ENTITY stepper_vhdl IS
PORT (
      clock, dir         :IN BIT;
      q                  :OUT BIT_VECTOR (3 DOWNTO 0));
END stepper_vhdl;
ARCHITECTURE vhdl OF stepper_vhdl IS
BEGIN
   PROCESS (clock)
   VARIABLE stepper  :BIT_VECTOR (3 DOWNTO 0);
   BEGIN
      IF (clock'EVENT AND clock= '1')     THEN
         IF dir = '0'   THEN
            CASE stepper IS
               WHEN "0101"    =>    stepper := "0100";
               WHEN "0100"    =>    stepper := "0110";
               WHEN "0110"    =>    stepper := "0010";
               WHEN "0010"    =>    stepper := "1010";
               WHEN "1010"    =>    stepper := "1000";
               WHEN "1000"    =>    stepper := "1001";
               WHEN "1001"    =>    stepper := "0001";
               WHEN "0001"    =>    stepper := "0101";
               WHEN OTHERS => stepper := "0101";
            END CASE;
         ELSIF dir = '1'      THEN
            CASE stepper IS
               WHEN "0101"    =>    stepper := "0001";
               WHEN "0001"    =>    stepper := "1001";
               WHEN "1001"    =>    stepper := "1000";
               WHEN "1000"    =>    stepper := "1010";
               WHEN "1010"    =>    stepper := "0010";
               WHEN "0010"    =>    stepper := "0110";
               WHEN "0110"    =>    stepper := "0100";
               WHEN "0100"    =>    stepper := "0101";
               WHEN OTHERS => stepper := "0101";
            END CASE;
         END IF;
      END IF;
      q <= stepper;
   END PROCESS;
END vhdl;
```

**7.53**

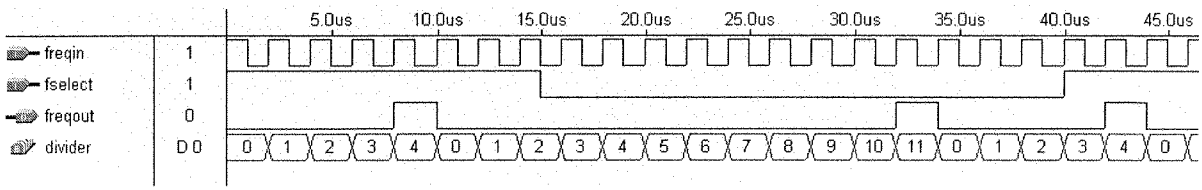
```
SUBDESIGN  divide_by50_ahdl
(
      freq_in           :INPUT;
      freq_out          :OUTPUT;
)
VARIABLE
      divide_by[5..0]   :DFF;
BEGIN
      divide_by[].clk  = freq_in;
      IF  divide_by[] == 1  THEN
            divide_by[].d = 50;
            freq_out = VCC;
      ELSE  divide_by[].d = divide_by[].q - 1;
      END IF;
END;
```




```
ENTITY  divide_by50_vhdl  IS
PORT (
      freq_in           :IN BIT;
      freq_out          :OUT BIT
);
END divide_by50_vhdl;
ARCHITECTURE vhdl OF divide_by50_vhdl IS
BEGIN
      PROCESS (freq_in)
      VARIABLE  divider :INTEGER RANGE 0 TO 50;
      BEGIN
            IF (freq_in'EVENT AND freq_in='1')  THEN
                  IF  divider = 1  THEN
                        divider := 50;
                  ELSE
                        divider := divider - 1;
                  END IF;
            END IF;
            IF divider = 1  THEN
                        freq_out <= '1';
            ELSE
                        freq_out <= '0';
            END IF;
      END PROCESS;
END vhdl;
```

**7.54**

```
SUBDESIGN variable_div_ahdl
(     freqin, fselect                :INPUT;
      freqout                        :OUTPUT;     )
VARIABLE
      divider[3..0]                  :DFF;
BEGIN
DEFAULTS
      freqout = GND;
END DEFAULTS;
divider[].clk = freqin;
IF  fselect == GND      THEN
      IF  divider[].q == 11    THEN  divider[].d = 0;  freqout = VCC;
      ELSE         divider[].d = divider[].q + 1;
      END IF;
ELSE
      IF  divider[].q == 4     THEN  divider[].d = 0;  freqout = VCC;
      ELSE         divider[].d = divider[].q + 1;
      END IF;
END IF;
END;
```




```
ENTITY  variable_div_vhdl  IS
PORT (      freqin, fselect          :IN BIT;
            freqout                  :OUT BIT);
END variable_div_vhdl;
ARCHITECTURE vhdl OF variable_div_vhdl IS
BEGIN
PROCESS (freqin)
VARIABLE
      divider                        :INTEGER RANGE 0 TO 12;
BEGIN
IF (freqin'EVENT AND freqin = '1')  THEN
      IF  fselect = '0' THEN
            IF  divider = 11  THEN        divider := 0;
            ELSE        divider := divider + 1;
            END IF;
      ELSE
            IF  divider = 4   THEN        divider := 0;
            ELSE        divider := divider + 1;
            END IF;
      END IF;
END IF;
IF (divider = 11 AND fselect = '0')       THEN  freqout <= '1';
ELSIF (divider = 4 AND fselect = '1')     THEN  freqout <= '1';
ELSE              freqout <= '0';
END IF;
END PROCESS;
END;
```

**7.55**

```
SUBDESIGN  mod256_ahdl
(
      clock, clear, load, cntenabl, down, din[7..0]    :INPUT;
      q[7..0], term_ct                                 :OUTPUT;
)
VARIABLE
      count[7..0]                    :DFF;
BEGIN
count[].clk = clock;
count[].clrn = !clear;
IF  load  THEN  count[].d = din[];
ELSIF  !cntenabl  THEN  count[].d = count[].q;
ELSIF  !down  THEN
      count[].d = count[].q + 1;
ELSE
      count[].d = count[].q - 1;
END IF;
IF ((count[].q == 0) & down # (count[].q == 255) & !down) & cntenabl
      THEN  term_ct = VCC;
ELSE  term_ct = GND;
END IF;
q[] = count[].q;
END;
```

```
ENTITY  mod256_vhdl  IS
PORT (clock, clear, load, cntenabl, down  :IN BIT;
      din                                 :IN INTEGER RANGE 0 TO 255;
      q                                   :OUT INTEGER RANGE 0 TO 255;
      term_ct                             :OUT BIT);
END mod256_vhdl;
ARCHITECTURE  vhdl OF  mod256_vhdl  IS
BEGIN
PROCESS (clock, clear, down)
      VARIABLE  count                     :INTEGER RANGE 0 TO 255;
      BEGIN
      IF  clear = '1'   THEN  count := 0;
      ELSIF  (clock = '1' AND clock'EVENT)      THEN
            IF  load = '1'    THEN  count := din;
            ELSIF  cntenabl = '1'   THEN
                  IF  down = '0'    THEN  count := count + 1;
                  ELSE                    count := count - 1;
                  END IF;
            END IF;
      END IF;
      IF (((count = 0) AND (down = '1')) OR
         ((count = 255) AND (down = '0'))) AND cntenabl = '1'
                  THEN        term_ct <= '1';
      ELSE                    term_ct <= '0';
      END IF;
      q <= count;
END PROCESS;
END vhdl;
```

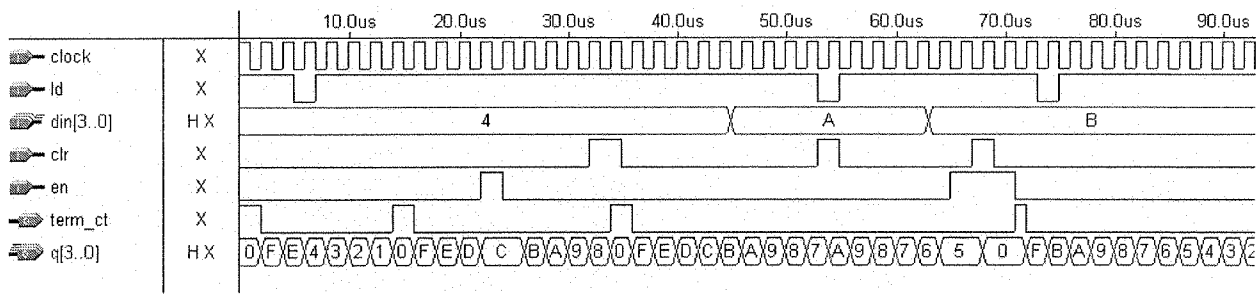
**7.56**

```
SUBDESIGN  mod1024_ahdl
(
      clock, clear, load, cntenabl, down, din[9..0]          :INPUT;
      q[9..0], term_ct                                       :OUTPUT;
)
VARIABLE
      count[9..0]                    :DFF;
BEGIN
count[].clk = clock;
count[].clrn = !clear;
IF  load  THEN  count[].d = din[];
ELSIF  !cntenabl  THEN  count[].d = count[].q;
ELSIF  !down  THEN
      count[].d = count[].q + 1;
ELSE
      count[].d = count[].q - 1;
END IF;
IF ((count[].q == 0) & down # (count[].q == 1023) & !down) & cntenabl
      THEN  term_ct = VCC;
ELSE  term_ct = GND;
END IF;
q[] = count[].q;
END;
```

```
ENTITY  mod1024_vhdl  IS
PORT (clock, clear, load, cntenabl, down  :IN BIT;
      din                                 :IN INTEGER RANGE 0 TO 1023;
      q                                   :OUT INTEGER RANGE 0 TO 1023;
      term_ct                             :OUT BIT);
END mod1024_vhdl;
ARCHITECTURE  vhdl OF  mod1024_vhdl  IS
BEGIN
PROCESS (clock, clear, down)
      VARIABLE  count                     :INTEGER RANGE 0 TO 1023;
      BEGIN
      IF  clear = '1'   THEN  count := 0;
      ELSIF  (clock = '1' AND clock'EVENT)      THEN
            IF  load = '1'    THEN  count := din;
            ELSIF  cntenabl = '1'   THEN
                  IF  down = '0'    THEN  count := count + 1;
                  ELSE                    count := count - 1;
                  END IF;
            END IF;
      END IF;
      IF (((count = 0) AND (down = '1')) OR
         ((count = 1023) AND (down = '0'))) AND cntenabl = '1'
                  THEN        term_ct <= '1';
      ELSE                    term_ct <= '0';
      END IF;
      q <= count;
END PROCESS;
END vhdl;
```

**7.57**

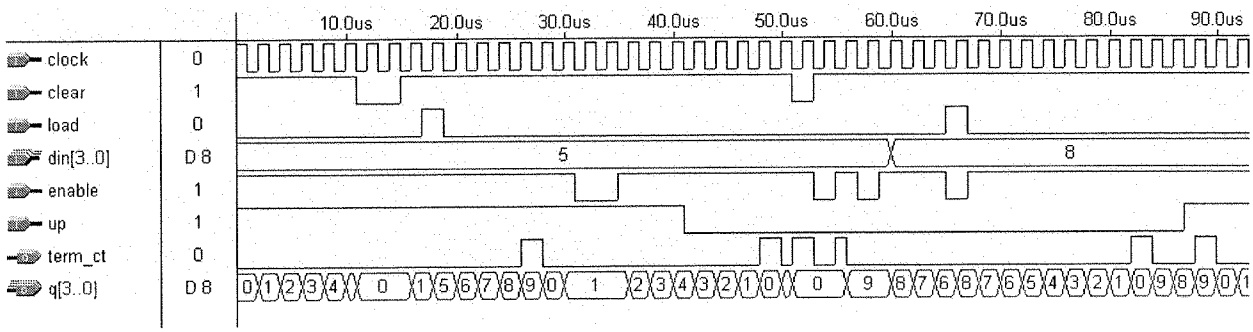
```
SUBDESIGN  mod16_ahdl
(      clock, clr, ld, en, din[3..0]        :INPUT;
       q[3..0], term_ct                     :OUTPUT;     )
VARIABLE
       count[3..0]              :DFF;
BEGIN
       count[].clk = clock;
       IF      !ld        THEN  count[].d = din[];
       ELSIF   clr        THEN  count[].d = 0;
       ELSIF   !en        THEN  count[].d = count[].q - 1;
       ELSE         count[].d = count[].q ;
       END IF;
       IF (count[].q == 0 & en == GND)      THEN  term_ct = VCC;
       ELSE         term_ct = GND;
       END IF;
       q[] = count[];
END;
```




```
ENTITY  mod16_vhdl  IS
PORT( clock, clr, ld, en            :IN BIT;
           din                      :IN INTEGER RANGE 15 DOWNTO 0;
           q                        :OUT INTEGER RANGE 15 DOWNTO 0;
           term_ct                  :OUT BIT);
END mod16_vhdl;
ARCHITECTURE  vhdl  OF  mod16_vhdl  IS
BEGIN
PROCESS (clock, en)
       VARIABLE  count              :INTEGER RANGE 15 DOWNTO 0;
       BEGIN
       IF    (clock'EVENT AND clock = '1') THEN
             IF          ld = '0'     THEN  count := din;
             ELSIF clr   = '1' THEN   count := 0;
             ELSIF en = '0'    THEN   count := count - 1;
             END IF;
       END IF;
       IF (count = 0 AND en = '0')    THEN  term_ct <= '1';
       ELSE         term_ct <= '0';
       END IF;
       q <= count;
END PROCESS;
END vhdl;
```

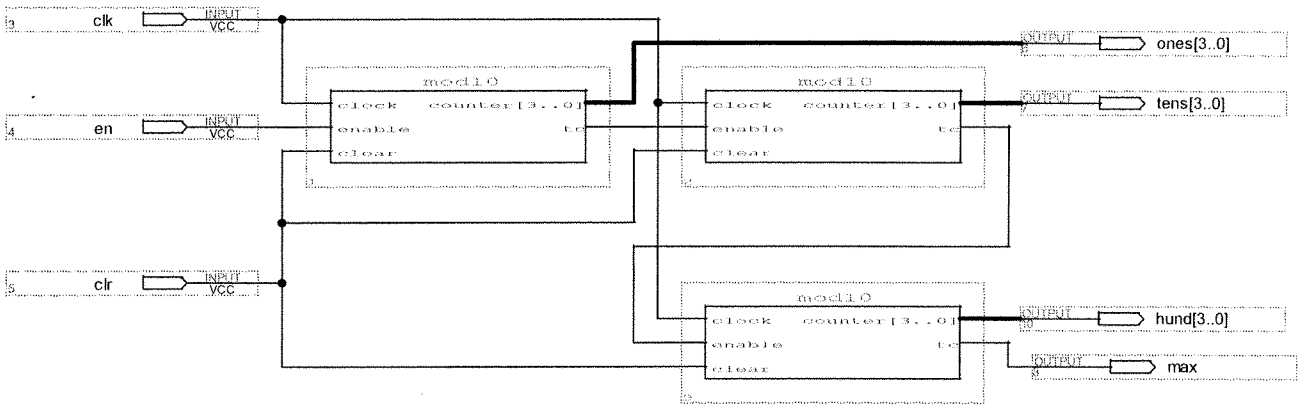
**7.58**

```
SUBDESIGN  mod10_ahdl
(
      clock, clear, load, enable, up, din[3..0]        :INPUT;
      q[3..0], term_ct                                 :OUTPUT;
)
VARIABLE
      count[3..0]                    :DFF;
BEGIN
      count[].clk = clock;
      count[].clrn = clear;
      IF    load  THEN  count[].d = din[];
      ELSIF  enable     THEN
            IF    up    THEN
                  IF  count[].q == 9      THEN
                              count[].d = 0;
                  ELSE        count[].d = count[].q + 1;
                  END IF;
            ELSE
                  IF  count[].q == 0  THEN
                              count[].d = 9;
                  ELSE        count[].d = count[].q - 1;
                  END IF;
            END IF;
      ELSE                    count[].d = count[].q;
      END IF;
      IF ((count[].q == 9 & up) # (count[].q == 0 & !up)) & enable
            THEN  term_ct = VCC;
      ELSE        term_ct = GND;
      END IF;
      q[] = count[];
END;
```

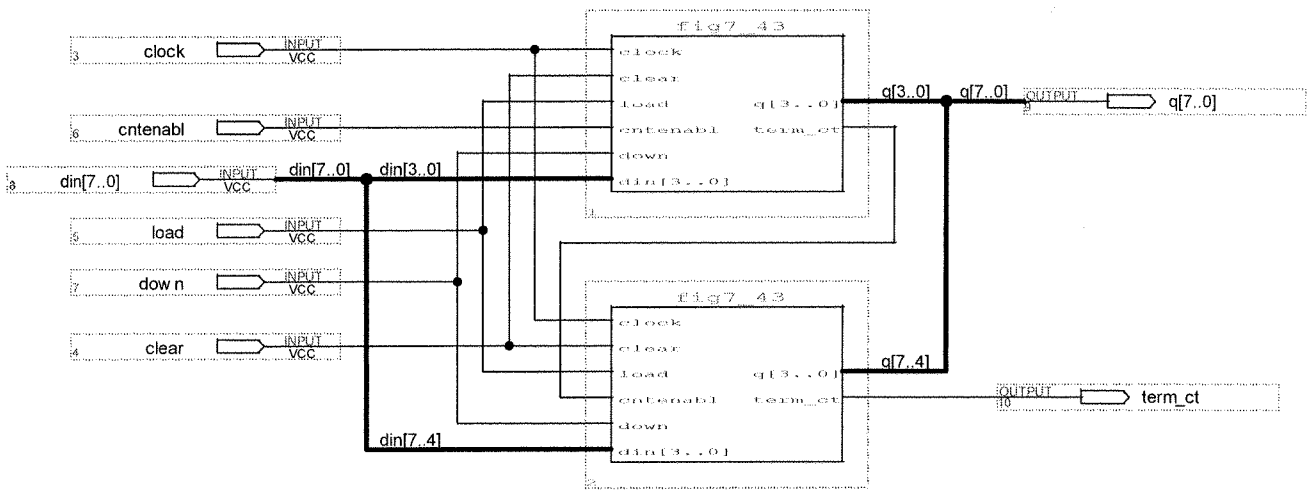


```
ENTITY  mod10_vhdl  IS
PORT(clock, clear, load, enable, up       :IN BIT;
     din                                  :IN INTEGER RANGE 0 TO 9;
     q                                    :OUT INTEGER RANGE 0 TO 9;
     term_ct                              :OUT BIT);
END mod10_vhdl;
ARCHITECTURE vhdl OF mod10_vhdl IS
BEGIN
PROCESS (clock, clear, enable, up)
VARIABLE  count                           :INTEGER RANGE 0 TO 9;
BEGIN
     IF    clear = '0' THEN  count := 0;
     ELSIF (clock'EVENT AND clock = '1') THEN
           IF    load = '1'  THEN  count := din;
           ELSIF  enable = '1'     THEN
                 IF    up = '1'    THEN
                       IF  count = 9     THEN  count := 0;
                       ELSE                    count := count + 1;
                       END IF;
                 ELSE
                       IF  count = 0     THEN  count := 9;
                       ELSE                    count := count - 1;
                       END IF;
                 END IF;
           END IF;
     END IF;
     IF ((count = 9 AND up = '1') OR
        (count = 0 AND up = '0')) AND enable = '1'
           THEN  term_ct <= '1';
     ELSE        term_ct <= '0';
     END IF;
     q <= count;
END PROCESS;
END vhdl;
```

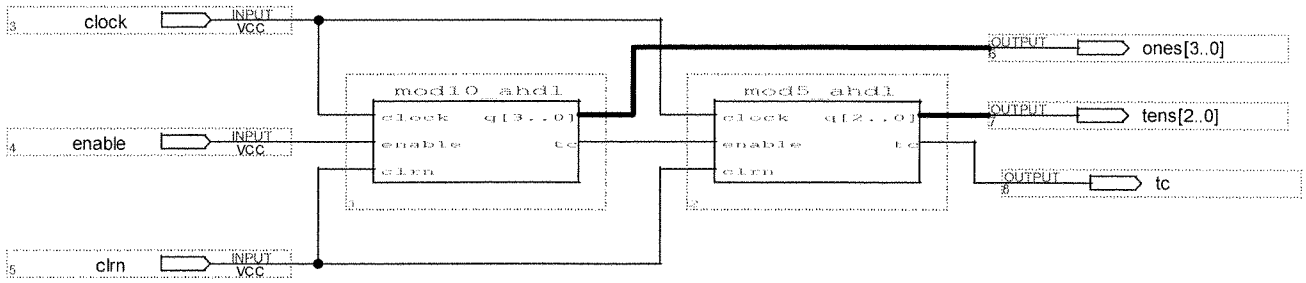
## 7.59


clk
INPUT
VCC
en
INPUT
VCC
clr
INPUT
VCC
mod10
clock
enable
clear
counter[3..0]
tc
OUTPUT
ones[3..0]
tens[3..0]
hund[3..0]
max

## 7.60


clock
cntenabl
din[7..0]
load
down
clear
INPUT
VCC
fig7_43
clock
clear
load
cntenabl
down
din[3..0]
q[3..0]
term_ct
din[7..0]
din[3..0]
din[7..4]
q[7..0]
q[7..4]
OUTPUT
q[7..0]
term_ct

**7.61**

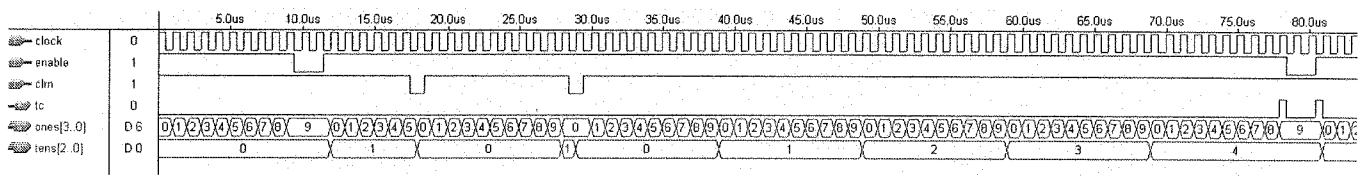



```
SUBDESIGN  mod10_ahdl
(
      clock, enable, clrn           :INPUT;
      q[3..0], tc                   :OUTPUT;
)
VARIABLE
      q[3..0]                       :DFF;
BEGIN
      q[].clk  = clock;
      IF q[].q == 9 & enable  THEN  tc = VCC;
      ELSE                          tc = GND;
      END IF;
      IF  !clrn  THEN               q[].d = 0;
      ELSIF  enable     THEN
            IF  q[].q == 9  THEN    q[].d = 0;
            ELSE                    q[].d = q[].q + 1;
            END IF;
      ELSE                          q[].d = q[].q;
      END IF;
END;
```

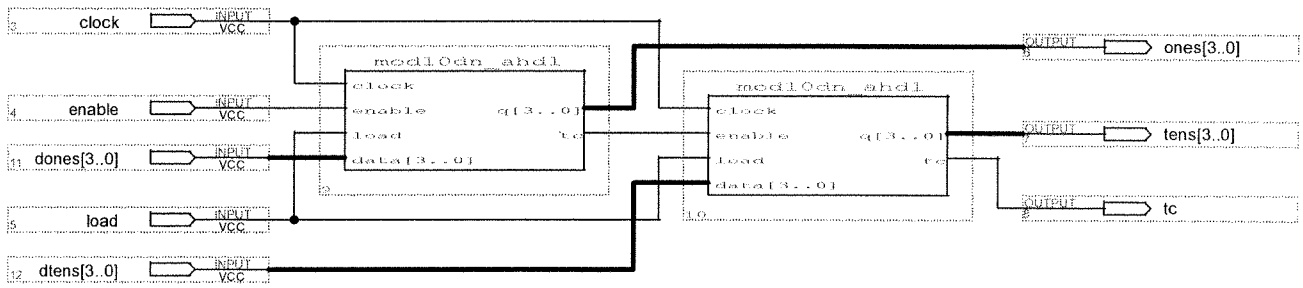
```
SUBDESIGN  mod5_ahdl
(
      clock, enable, clrn           :INPUT;
      q[2..0], tc                   :OUTPUT;
)
VARIABLE
      q[2..0]                       :DFF;
BEGIN
      q[].clk  = clock;
      IF q[].q == 4 & enable  THEN  tc = VCC;
      ELSE                          tc = GND;
      END IF;
      IF  !clrn  THEN               q[].d = 0;
      ELSIF  enable     THEN
            IF  q[].q == 4  THEN    q[].d = 0;
            ELSE                    q[].d = q[].q + 1;
            END IF;
      ELSE                          q[].d = q[].q;
      END IF;
END;
```

```
ENTITY  mod50_vhdl  IS
PORT(clock, enable, clrn            :IN BIT;
      tc                            :OUT BIT;
      ones                          :OUT INTEGER RANGE 0 TO 9;
      tens                          :OUT INTEGER RANGE 0 TO 4);
END mod50_vhdl;
ARCHITECTURE  vhdl  OF  mod50_vhdl  IS
SIGNAL  cascade_node                :BIT;
COMPONENT  mod10_vhdl
      PORT(clock, enable, clrn      :IN BIT;
      tc                            :OUT BIT;
      q                             :OUT INTEGER RANGE 0 TO 9);
END COMPONENT;
COMPONENT  mod5_vhdl
      PORT(clock, enable, clrn      :IN BIT;
      tc                            :OUT BIT;
      q                             :OUT INTEGER RANGE 0 TO 4);
END COMPONENT;
BEGIN
digit1:      mod10_vhdl PORT MAP (clock => clock, enable => enable,
      clrn => clrn, tc => cascade_node, q => ones);
digit2:  mod5_vhdl  PORT MAP (clock => clock, enable => cascade_node,
      clrn => clrn, tc => tc, q => tens);
END vhdl;
-------------------------------------------------------
ENTITY  mod10_vhdl  IS
PORT(clock, enable, clrn            :IN BIT;
      tc                            :OUT BIT;
      q                             :OUT INTEGER RANGE 0 TO 9);
END mod10_vhdl;
ARCHITECTURE lsd  OF  mod10_vhdl  IS
BEGIN
PROCESS (clock, enable)
VARIABLE count                      :INTEGER RANGE 0 TO 9;
BEGIN
      IF    (clock'EVENT AND clock = '1') THEN
            IF  clrn = '0'  THEN    count := 0;
            ELSIF  enable = '1'     THEN
                  IF  count = 9  THEN     count := 0;
                  ELSE                    count := count + 1;
                  END IF;
            END IF;
      END IF;
      IF (count = 9 AND enable = '1')     THEN  tc <= '1';
      ELSE                                              tc <= '0';
      END IF;
      q <= count;
END PROCESS;
END lsd;
-------------------------------------------------------
```

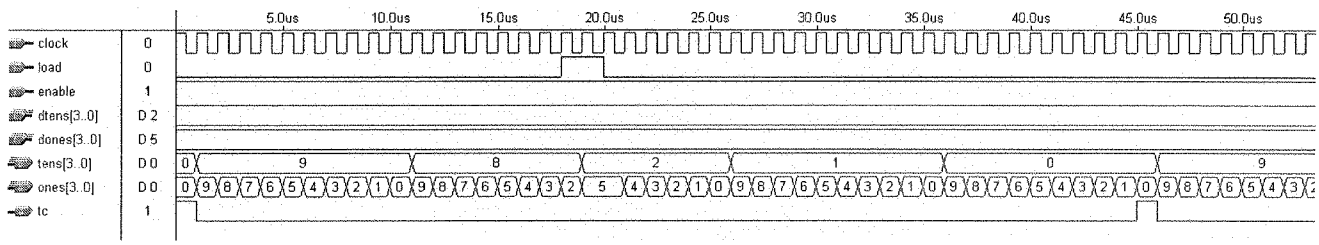
```
ENTITY  mod5_vhdl  IS
PORT(clock, enable, clrn            :IN BIT;
     tc                             :OUT BIT;
     q                              :OUT INTEGER RANGE 0 TO 4);
END mod5_vhdl;
ARCHITECTURE msd  OF  mod5_vhdl  IS
BEGIN
PROCESS (clock, enable)
VARIABLE count                      :INTEGER RANGE 0 TO 4;
BEGIN
     IF    (clock'EVENT AND clock = '1') THEN
           IF  clrn = '0'  THEN    count := 0;
           ELSIF  enable = '1'     THEN
                 IF  count = 4  THEN     count := 0;
                 ELSE                    count := count + 1;
                 END IF;
           END IF;
     END IF;
     IF (count = 4 AND enable = '1')     THEN  tc <= '1';
     ELSE                                      tc <= '0';
     END IF;
     q <= count;
END PROCESS;
END msd;
```



**7.62**




```
SUBDESIGN  mod10dn_ahdl
(
      clock, enable, load           :INPUT;
      data[3..0]                    :INPUT;
      q[3..0], tc                   :OUTPUT;
)
VARIABLE
      q[3..0]                       :DFF;
BEGIN
      q[].clk  = clock;
      IF q[].q == 0 & enable  THEN  tc = VCC;
      ELSE                          tc = GND;
      END IF;
      IF  load  THEN                q[].d = data[];
      ELSIF  enable     THEN
            IF  q[].q == 0  THEN    q[].d = 9;
            ELSE                    q[].d = q[].q - 1;
            END IF;
      ELSE                          q[].d = q[].q;
      END IF;
END;
```



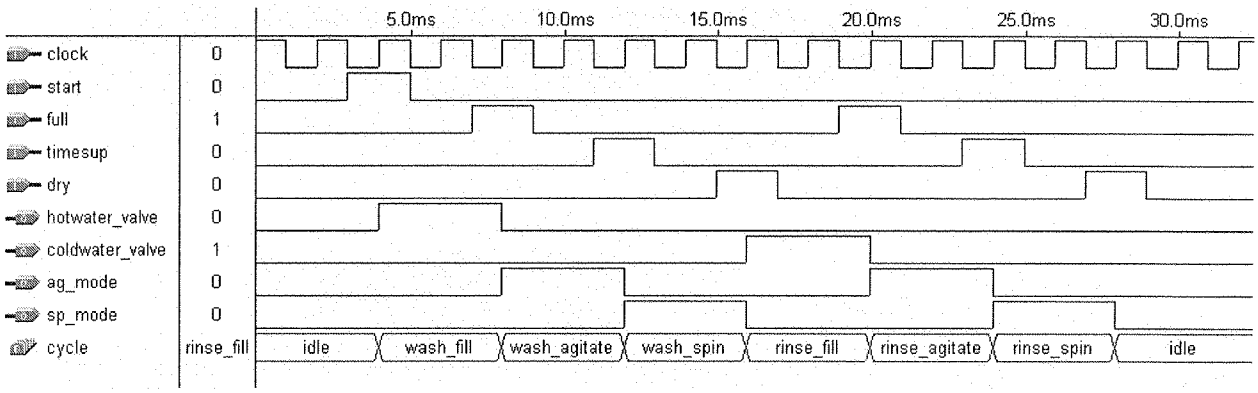
```
ENTITY  mod100dn_vhdl  IS
PORT(clock, enable, load            :IN BIT;
      dtens, dones                  :IN INTEGER RANGE 0 TO 9;
      tc                            :OUT BIT;
      ones                          :OUT INTEGER RANGE 0 TO 9;
      tens                          :OUT INTEGER RANGE 0 TO 9);
END mod100dn_vhdl;
ARCHITECTURE  vhdl  OF  mod100dn_vhdl  IS
SIGNAL  cascade_node                :BIT;
COMPONENT  mod10dn_vhdl
      PORT(clock, enable, load      :IN BIT;
      data                          :IN INTEGER RANGE 0 TO 9;
      tc                            :OUT BIT;
      q                             :OUT INTEGER RANGE 0 TO 9);
END COMPONENT;
BEGIN
digit1:     mod10dn_vhdl      PORT MAP (clock => clock,
            enable => enable, load => load, data => dones,
            tc => cascade_node, q => ones);
digit2:     mod10dn_vhdl      PORT MAP (clock => clock,
            enable => cascade_node, load=> load, data => dtens,
            tc => tc, q => tens);
END vhdl;
--------------------------------------------------------------------------
ENTITY  mod10dn_vhdl  IS
PORT(clock, enable, load            :IN BIT;
      data                          :IN INTEGER RANGE 0 TO 9;
      tc                            :OUT BIT;
      q                             :OUT INTEGER RANGE 0 TO 9);
END mod10dn_vhdl;
ARCHITECTURE  bcd  OF  mod10dn_vhdl  IS
BEGIN
PROCESS (clock, enable)
VARIABLE count                      :INTEGER RANGE 0 TO 9;
BEGIN
      IF    (clock'EVENT AND clock = '1') THEN
            IF  load = '1'  THEN          count := data;
            ELSIF  enable = '1'     THEN
                  IF  count = 0  THEN     count := 9;
                  ELSE                    count := count - 1;
                  END IF;
            END IF;
      END IF;
      IF (count = 0 AND enable = '1')     THEN  tc <= '1';
      ELSE                                      tc <= '0';
      END IF;
      q <= count;
END PROCESS;
END bcd;
```

**7.63**

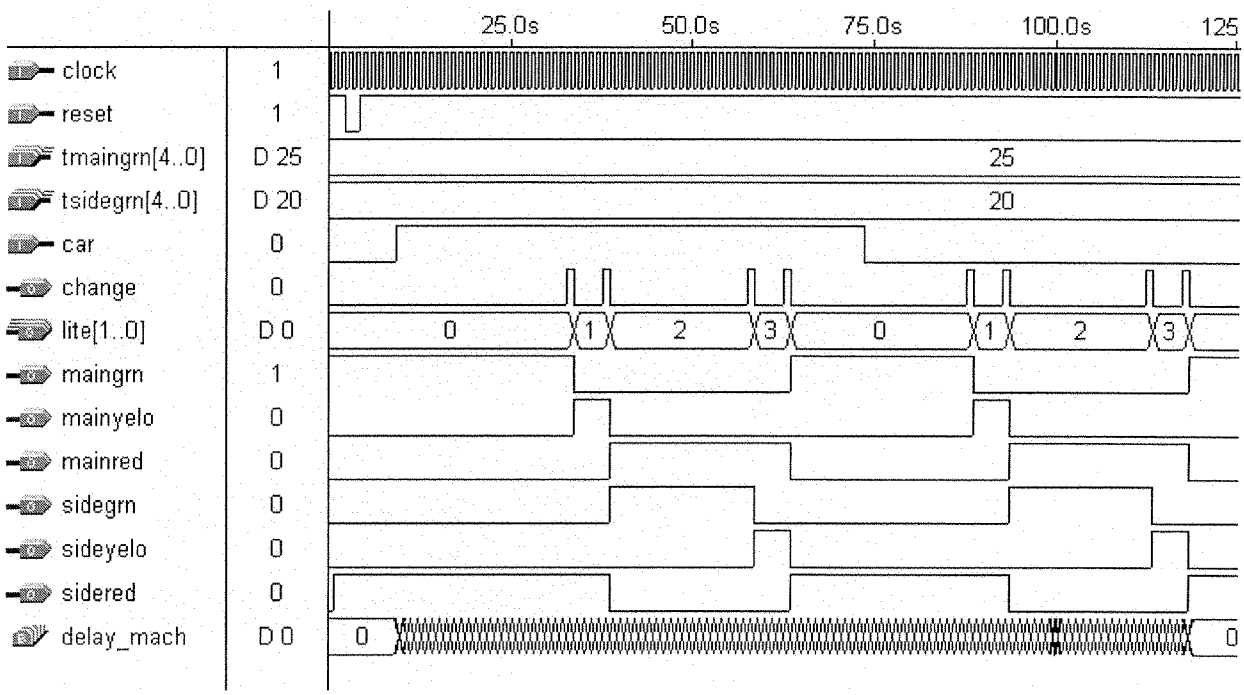
```
SUBDESIGN  wash_mach_delux
(          clock, start, full, timesup, dry                          : INPUT;
           hotwater_valve, coldwater_valve, ag_mode, sp_mode         : OUTPUT;     )
VARIABLE
           cycle:    MACHINE              -- "buried" machine
                 WITH STATES (idle, wash_fill, wash_agitate, wash_spin,
                              rinse_fill, rinse_agitate, rinse_spin);
BEGIN
           cycle.clk = clock;

           CASE  cycle  IS
              WHEN idle =>
                 IF start THEN    cycle = wash_fill;
                 ELSE             cycle = idle;
                 END IF;
              WHEN wash_fill =>
                 IF full THEN     cycle = wash_agitate;
                 ELSE             cycle = wash_fill;
                 END IF;
              WHEN wash_agitate =>
                 IF timesup THEN cycle = wash_spin;
                 ELSE             cycle = wash_agitate;
                 END IF;
              WHEN wash_spin =>
                 IF dry THEN      cycle = rinse_fill;
                 ELSE             cycle = wash_spin;
                 END IF;
              WHEN rinse_fill =>
                 IF full THEN     cycle = rinse_agitate;
                 ELSE             cycle = rinse_fill;
                 END IF;
              WHEN rinse_agitate =>
                 IF timesup THEN cycle = rinse_spin;
                 ELSE             cycle = rinse_agitate;
                 END IF;
              WHEN rinse_spin =>
                 IF dry THEN      cycle = idle;
                 ELSE             cycle = rinse_spin;
                 END IF;
              WHEN OTHERS =>      -- all other states to idle
                 cycle = idle;
           END CASE;
           TABLE
              cycle          => hotwater_valve, coldwater_valve, ag_mode, sp_mode;
              idle           => GND,           GND,              GND,     GND;
              wash_fill      => VCC,           GND,              GND,     GND;
              wash_agitate   => GND,           GND,              VCC,     GND;
              wash_spin      => GND,           GND,              GND,     VCC;
              rinse_fill     => GND,           VCC,              GND,     GND;
              rinse_agitate  => GND,           GND,              VCC,     GND;
              rinse_spin     => GND,           GND,              GND,     VCC;
           END TABLE;
END;
```

```
ENTITY  wash_mach_delux  IS
PORT (clock, start, full, timesup, dry                          :IN BIT;
      hotwater_valve, coldwater_valve, ag_mode, sp_mode         :OUT BIT);
END wash_mach_delux;
ARCHITECTURE  vhdl  OF  wash_mach_delux  IS
TYPE  state_machine  IS  (idle, wash_fill, wash_agitate, wash_spin,
                          rinse_fill, rinse_agitate, rinse_spin);
BEGIN
      PROCESS (clock)
      VARIABLE  cycle                       :state_machine;
      BEGIN
      IF (clock'EVENT  AND  clock = '1')  THEN
            CASE  cycle  IS
                  WHEN idle =>
                        IF start = '1' THEN       cycle := wash_fill;
                        ELSE                      cycle := idle;
                        END IF;
                  WHEN wash_fill =>
                        IF full = '1' THEN        cycle := wash_agitate;
                        ELSE                      cycle := wash_fill;
                        END IF;
                  WHEN wash_agitate =>
                        IF timesup = '1' THEN     cycle := wash_spin;
                        ELSE                      cycle := wash_agitate;
                        END IF;
                  WHEN wash_spin =>
                        IF dry = '1' THEN         cycle := rinse_fill;
                        ELSE                      cycle := wash_spin;
                        END IF;
                  WHEN rinse_fill =>
                        IF full = '1' THEN        cycle := rinse_agitate;
                        ELSE                      cycle := rinse_fill;
                        END IF;
                  WHEN rinse_agitate =>
                        IF timesup = '1' THEN     cycle := rinse_spin;
                        ELSE                      cycle := rinse_agitate;
                        END IF;
                  WHEN rinse_spin =>
                        IF dry = '1' THEN         cycle := idle;
                        ELSE                      cycle := rinse_spin;
                        END IF;
            END CASE;
      END IF;
```

```
      CASE  cycle  IS
               WHEN idle            =>    hotwater_valve <= '0';  coldwater_valve
                                          <= '0';
                                          ag_mode <= '0';  sp_mode <= '0';
               WHEN wash_fill       =>    hotwater_valve <= '1';  coldwater_valve
                                          <= '0';
                                          ag_mode <= '0';  sp_mode <= '0';
               WHEN wash_agitate    =>    hotwater_valve <= '0';  coldwater_valve
                                          <= '0';
                                          ag_mode <= '1';  sp_mode <= '0';
               WHEN wash_spin       =>    hotwater_valve <= '0';  coldwater_valve
                                          <= '0';
                                          ag_mode <= '0';  sp_mode <= '1';
               WHEN rinse_fill      =>    hotwater_valve <= '0';  coldwater_valve
                                          <= '1';
                                          ag_mode <= '0';  sp_mode <= '0';
               WHEN rinse_agitate   =>    hotwater_valve <= '0';  coldwater_valve
                                          <= '0';
                                          ag_mode <= '1';  sp_mode <= '0';
               WHEN rinse_spin      =>    hotwater_valve <= '0';  coldwater_valve
                                          <= '0';
                                          ag_mode <= '0';  sp_mode <= '1';
      END CASE;
      END PROCESS;
END vhdl;
```



**7.64**

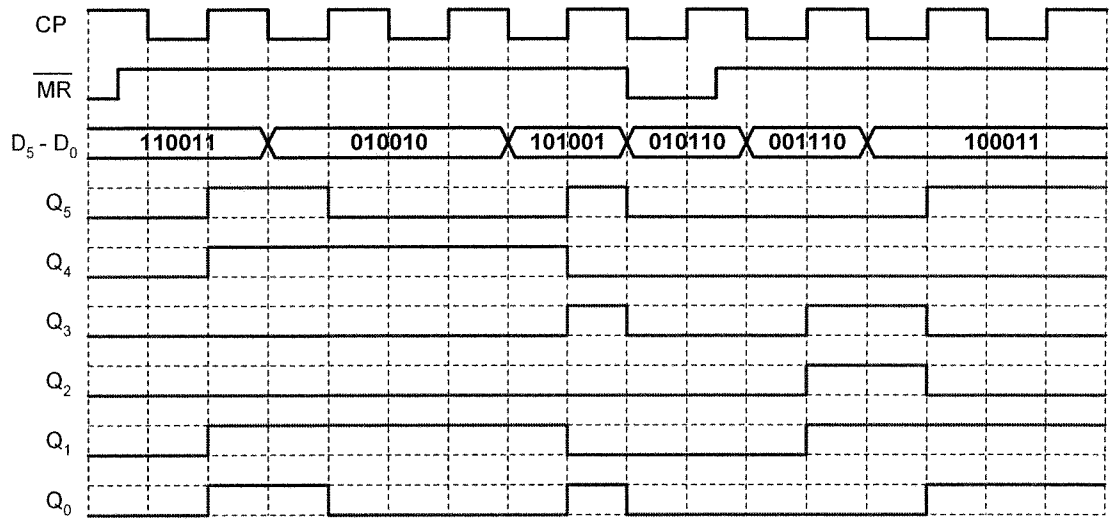



**7.65**

Parallel data transfer; 4 clock pulses are needed to move data on I to Z

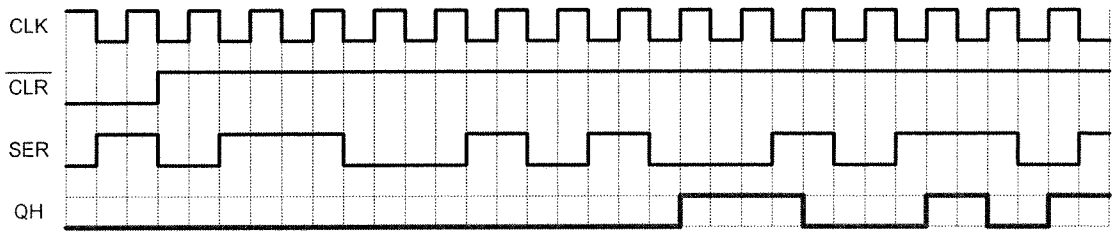
| ↑ CLK | $\overline{MR}$ | I5 – I0 | W5 – W0 | X5 – X0 | Y5 – Y0 | Z5 – Z0 |
|---|---|---|---|---|---|---|
| X | 0 | 101010 | 000000 | 000000 | 000000 | 000000 |
| CP1 | 1 | 101010 | 101010 | 000000 | 000000 | 000000 |
| CP2 | 1 | 010101 | 010101 | 101010 | 000000 | 000000 |
| CP3 | 1 | 000111 | 000111 | 010101 | 101010 | 000000 |
| CP4 | 1 | 111000 | 111000 | 000111 | 010101 | 101010 |
| CP5 | 1 | 011011 | 011011 | 111000 | 000111 | 010101 |
| CP6 | 1 | 001101 | 001101 | 011011 | 111000 | 000111 |
| CP7 | 1 | 000000 | 000000 | 001101 | 011011 | 111000 |
| CP8 | 1 | 000000 | 000000 | 000000 | 001101 | 011011 |

**7.66** The register is cleared immediately when Master Reset is active.

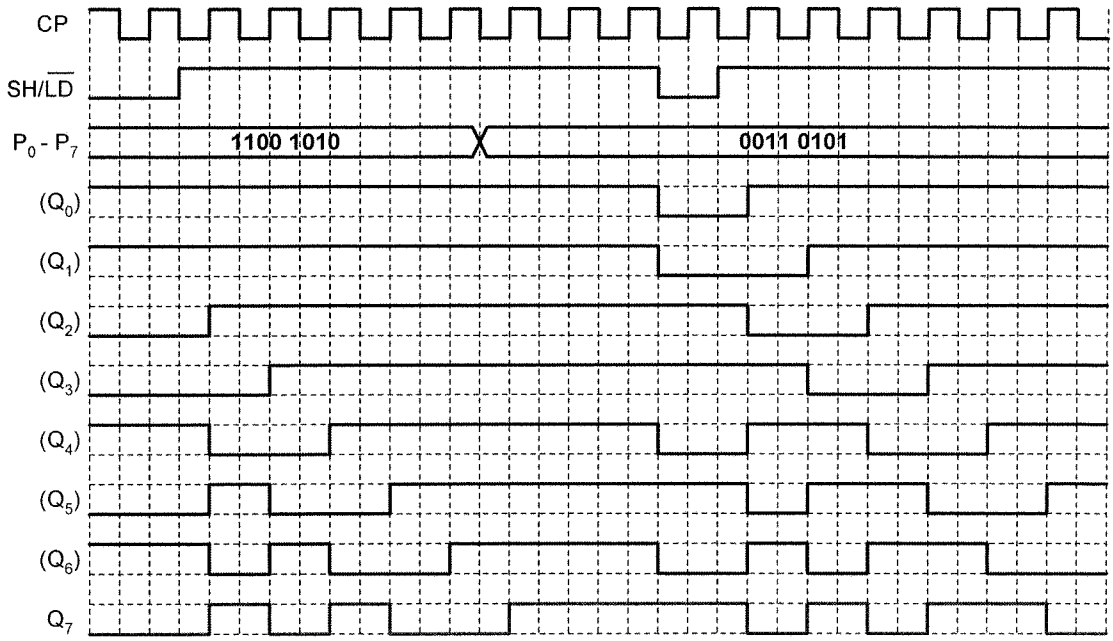



**7.67** 8 clock pulses are needed to serially load a 74166 since there are 8 FFs in the chip.

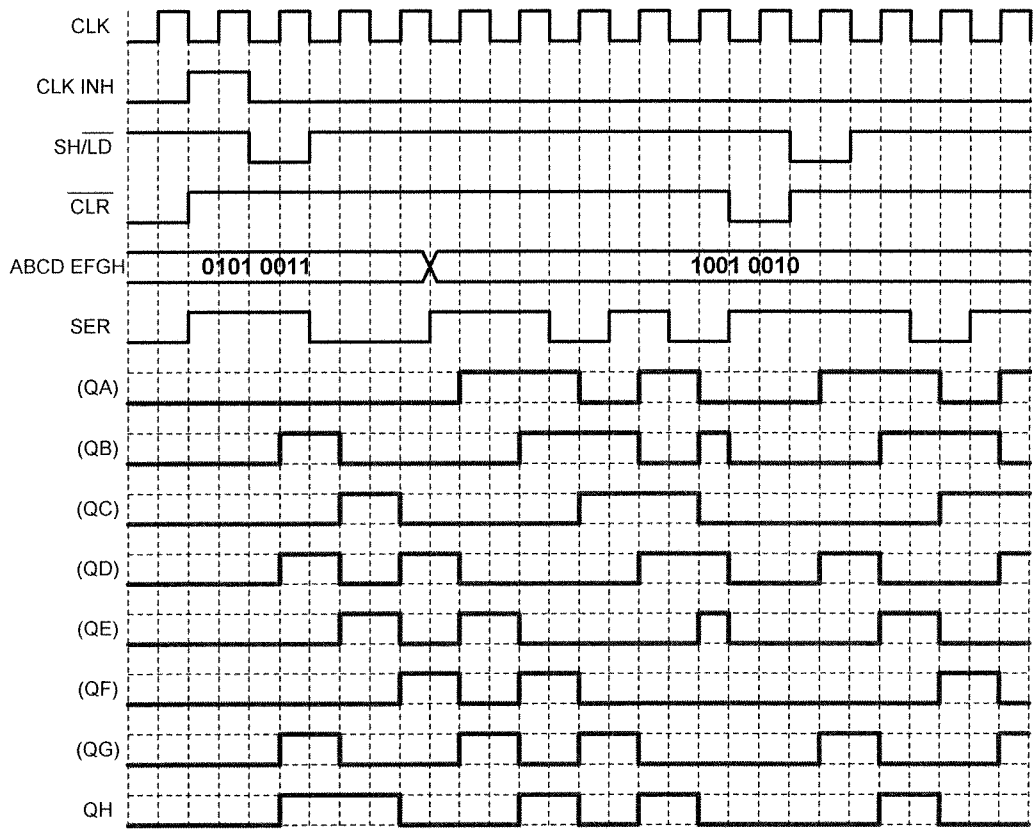
**7.68**



**7.69**


CP
SH/LD
P0 - P7
1100 1010
0011 0101
(Q0)
(Q1)
(Q2)
(Q3)
(Q4)
(Q5)
(Q6)
Q7

**7.70**


CLK
CLK INH
SH/LD
CLR
ABCD EFGH
0101 0011
1001 0010
SER
(QA)
(QB)
(QC)
(QD)
(QE)
(QF)
(QG)
QH

**7.71**

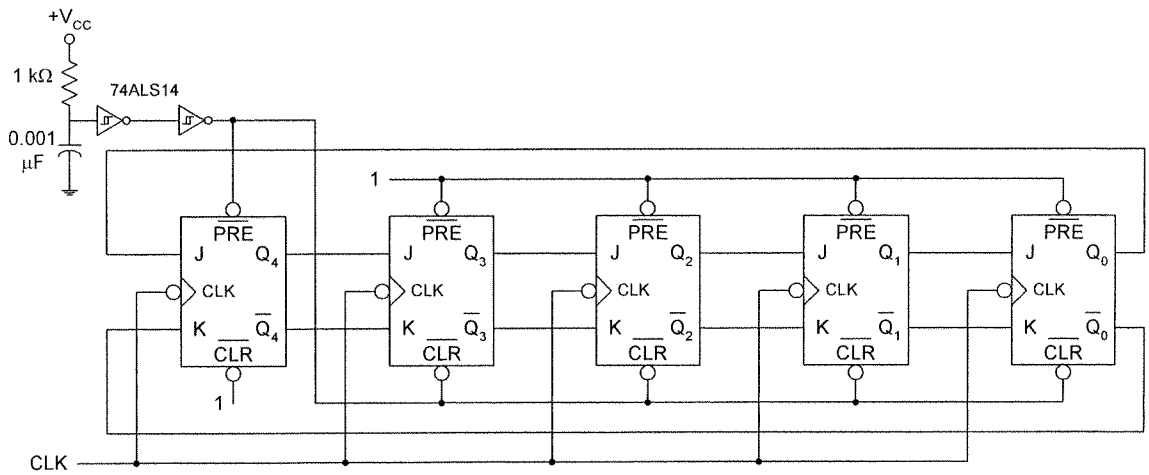
(a) asynch.
(b) True

| part | Starting at: | After 1 CLK: | After 2 CLKs: | After 3 CLKs: | After 4 CLKs: |
|---|---|---|---|---|---|
| (c) | 1011 | 0111 | 1111 | 1111 | 1111 |
| (d) | 1011 | 0101 | 0010 | 0001 | 0000 |
| (e) | 1011 | 0110 | 0110 | 0110 | 0110 |
| (f) | 1011 | 1011 | 1011 | 1011 | 1011 |
| (g) | 1011 | 0111 | 1110 | 1101 | 1011 |

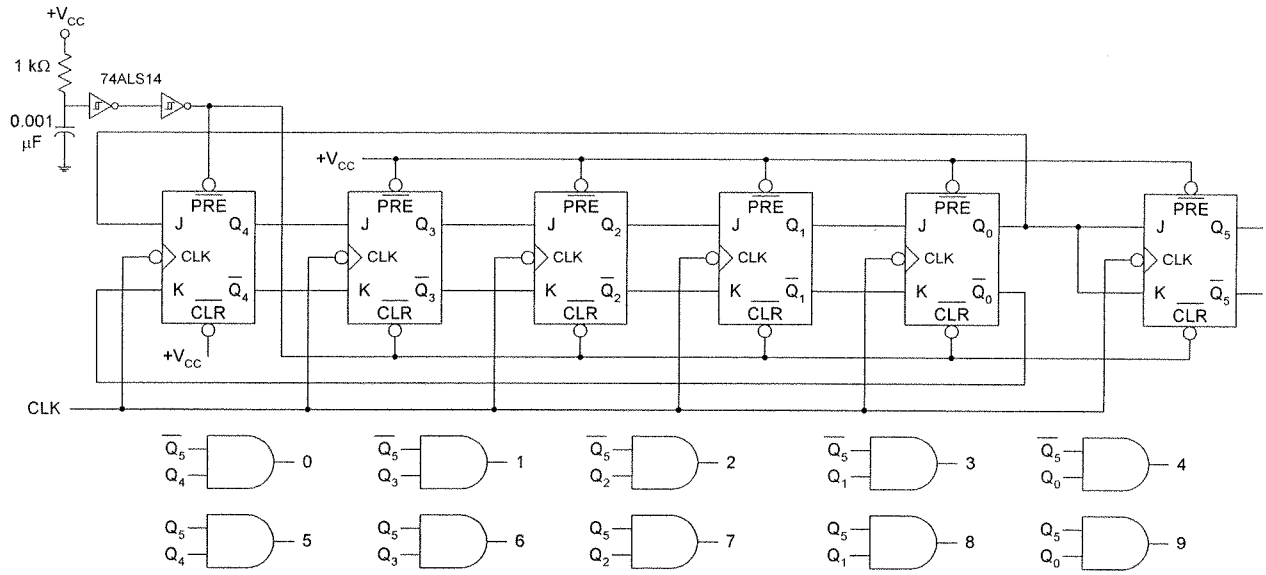
**7.72**

(a) Loads 00000000.
(b) Loads 11111111.
(c) Shifts in a 1.
(d) Shifts in a 0.
(e) Output will change states if input is switched to the same logic level (in = out).
(f) Input logic level must be maintained for at least 8 clock pulses.
(g) The output will not switch states.
(h) Output will not switch states until input signal is stable; pulsing input condition will not be recognized.

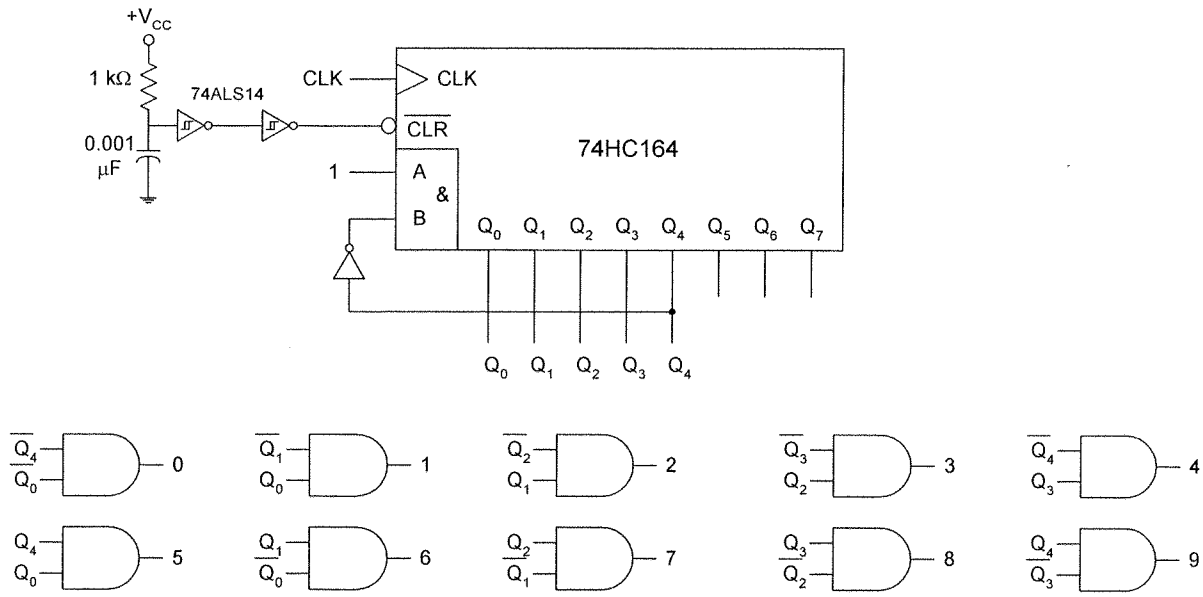
**7.73**



**7.74**




**7.75**




**7.76** 1 Hz, 50% duty cycle.

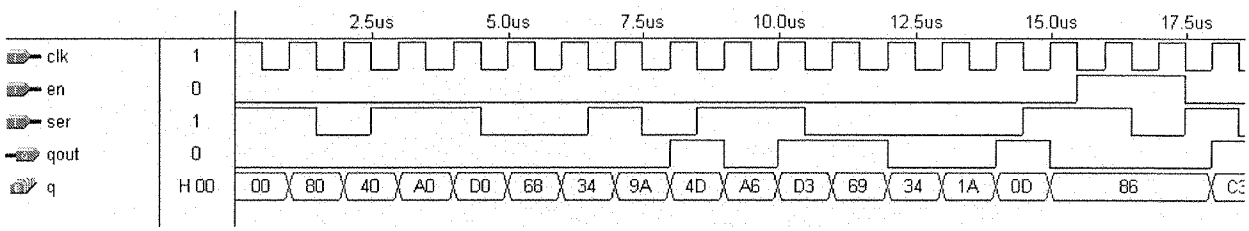
**7.77** output of 3-in AND or J, K inputs to FF D shorted to ground, FF D output shorted to ground, CLK input on FF D open, B input to NAND is open.

**7.78** Output of 2-input AND or J, K inputs shorted to VCC, output from 2-input AND is open.

**7.79**

```
SUBDESIGN  siso8_ahdl
(     clk, en, ser       :INPUT;
      qout               :OUTPUT;     )
VARIABLE
      q[7..0]            :DFF;
BEGIN
      q[].clk = clk;
      qout = q0.q;
      IF (en == GND)     THEN
            q[7..0].d = (ser, q[7..1].q);
      ELSE
            q[7..0].d = (q[7..0].q);
      END IF;
END;
```

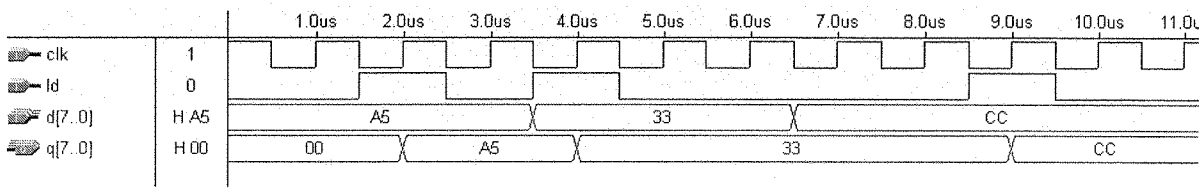
```
ENTITY  siso8_vhdl  IS
PORT (      clk, en, ser       :IN BIT;
            qout               :OUT BIT     );
END siso8_vhdl;
ARCHITECTURE  vhdl  OF  siso8_vhdl  IS
BEGIN
PROCESS (clk)
      VARIABLE  q              :BIT_VECTOR (7 DOWNTO 0);
      BEGIN
      qout <= q(0);
      IF (clk'EVENT AND clk = '1')  THEN
            IF (en = '0')  THEN  q := (ser & q(7 DOWNTO 1));
            END IF;
      END IF;
END PROCESS;
END vhdl;
```



**7.80**

```
SUBDESIGN  pipo8_ahdl
(     clk, ld, d[7..0]          :INPUT;
      q[7..0]                   :OUTPUT;    )
VARIABLE
      q[7..0]                   :DFF;
BEGIN
      q[].clk = clk;
      IF (ld == VCC)     THEN
            q[7..0].d = d[7..0];
      ELSE
            q[7..0].d = (q[7..0].q);
      END IF;
END;
```

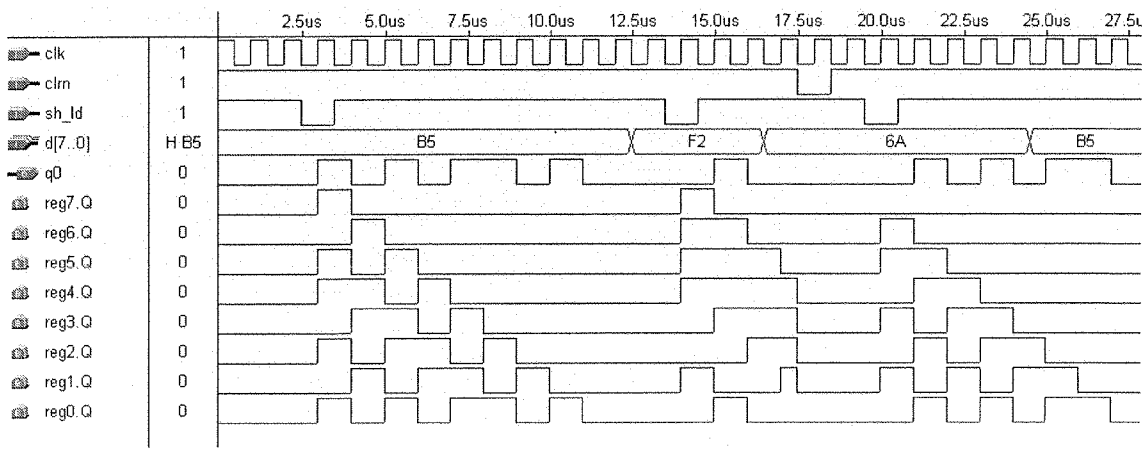
```
ENTITY  pipo8_vhdl  IS
PORT (      clk, ld           :IN BIT;
            d                 :IN BIT_VECTOR (7 DOWNTO 0);
            q                 :OUT BIT_VECTOR (7 DOWNTO 0));
END pipo8_vhdl;
ARCHITECTURE  vhdl  OF  pipo8_vhdl  IS
BEGIN
PROCESS (clk)
      VARIABLE  reg           :BIT_VECTOR (7 DOWNTO 0);
      BEGIN
      IF (clk'EVENT AND clk = '1')  THEN
            IF (ld = '1')  THEN  reg := d;
            END IF;
      END IF;
      q <= reg;
END PROCESS;
END vhdl;
```



**7.81**

```
SUBDESIGN  piso8_ahdl
(      clk, sh_ld, clrn, d[7..0]      :INPUT;
       q0                             :OUTPUT;     )
VARIABLE
       q[7..0]                        :DFF;
       ser                            :NODE;
BEGIN
       q[].clk = clk;
       q[].clrn = clrn;
       ser = GND;
       IF (sh_ld == GND) THEN
             q[7..0].d = d[7..0];
       ELSE
             q[7..0].d = (ser, q[7..1].q);
       END IF;
END;
```

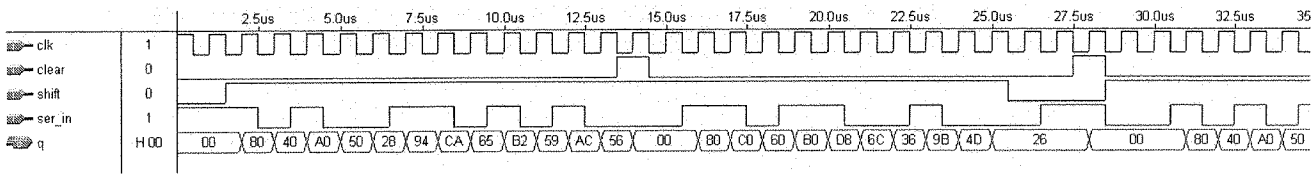
```
ENTITY  piso8_vhdl  IS
PORT (       clk, sh_ld, clrn          :IN BIT;
             d                         :IN BIT_VECTOR (7 DOWNTO 0);
             q0                        :OUT BIT);
END piso8_vhdl;
ARCHITECTURE  vhdl  OF  piso8_vhdl  IS
SIGNAL  ser                            :BIT;
BEGIN
ser <= '0';
PROCESS (clk, clrn)
       VARIABLE  reg                   :BIT_VECTOR (7 DOWNTO 0);
       BEGIN
       IF clrn = '0'      THEN  reg := "00000000";
       ELSIF (clk'EVENT AND clk = '1')      THEN
             IF (sh_ld = '0')  THEN  reg := d;
             ELSE reg := (ser & reg(7 DOWNTO 1));
             END IF;
       END IF;
       q0 <= reg(0);
END PROCESS;
END vhdl;
```



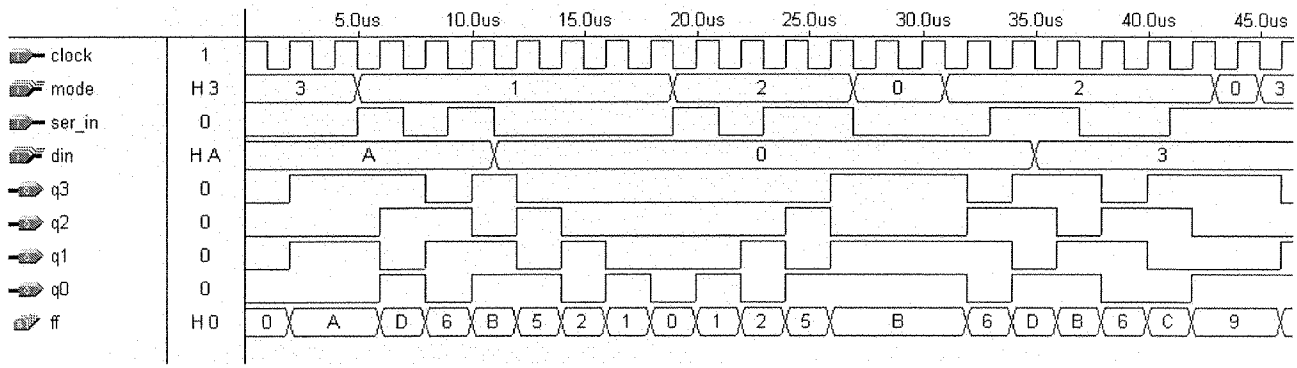
**7.82**

```
SUBDESIGN  sipo8_ahdl
(     clk, shift, clear, ser_in          :INPUT;
      q[7..0]                            :OUTPUT;     )
VARIABLE
      q[7..0]                            :DFF;
BEGIN
      q[].clk = clk;
      IF (clear == VCC) THEN  q[] = 0;
      ELSIF (shift == VCC)    THEN
            q[7..0].d = (ser_in, q[7..1]);
      ELSE
            q[7..0].d = q[7..0].q;
      END IF;
END;
```

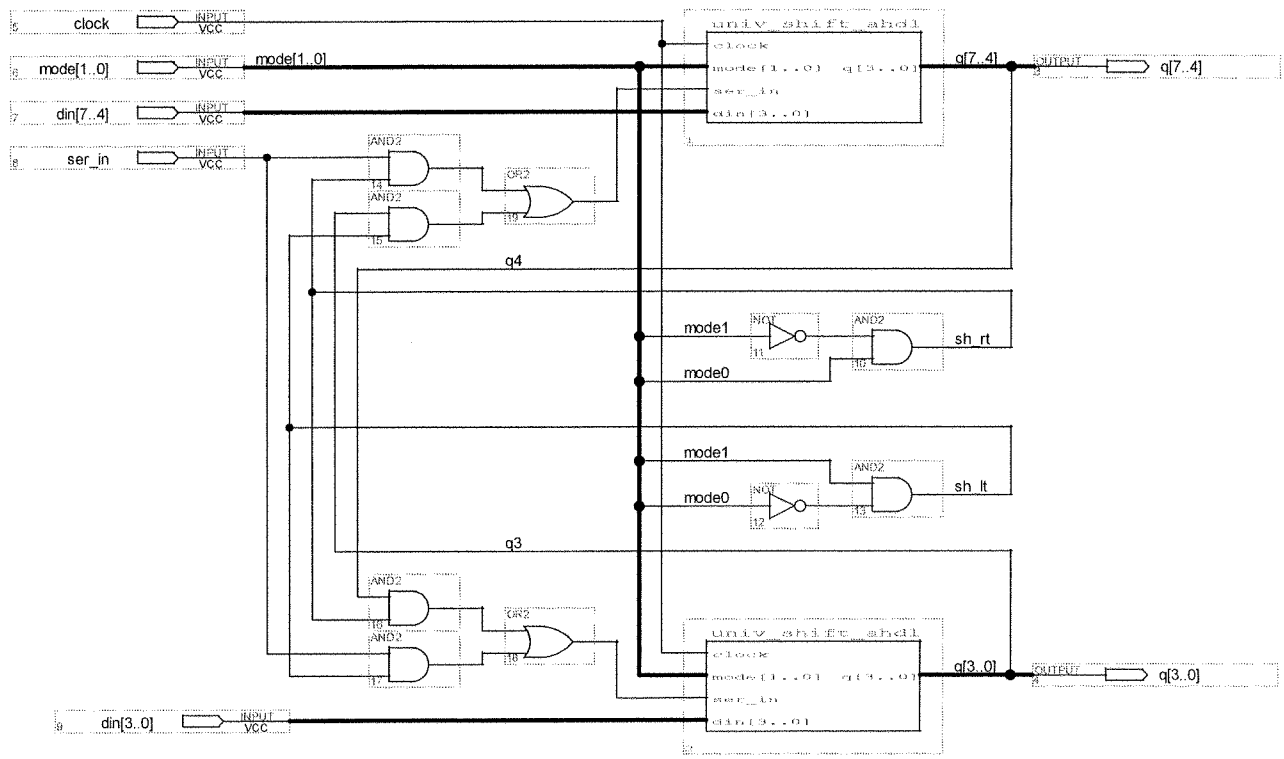
```
ENTITY  sipo8_vhdl  IS
PORT (clk, shift, clear, ser_in     :IN BIT;
      q                             :OUT BIT_VECTOR (7 DOWNTO 0));
END sipo8_vhdl;
ARCHITECTURE  vhdl  OF  sipo8_vhdl  IS
BEGIN
PROCESS (clk)
      VARIABLE  reg                 :BIT_VECTOR (7 DOWNTO 0);
      BEGIN
      IF (clk'EVENT AND clk = '1')  THEN
            IF (clear = '1')        THEN  reg := "00000000";
            ELSIF (shift = '1')     THEN
                              reg := (ser_in & reg(7 DOWNTO 1));
            END IF;
      END IF;
      q <= reg;
END PROCESS;
END vhdl;
```

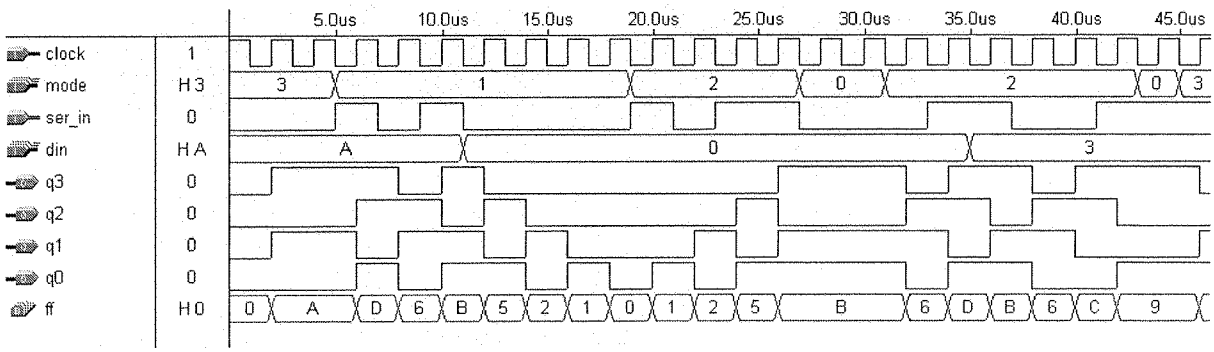


**7.83**


5.0us
10.0us
15.0us
20.0us
25.0us
30.0us
35.0us
40.0us
45.0us
clock
mode
ser_in
din
q3
q2
q1
q0
ff

**7.84**


clock
mode[1..0]
din[7..4]
ser_in
din[3..0]
univ_shift_abdl
q[7..4]
q[3..0]
q4
q3
mode1
mode0
sh_rt
sh_lt
AND2
OR2
NOT

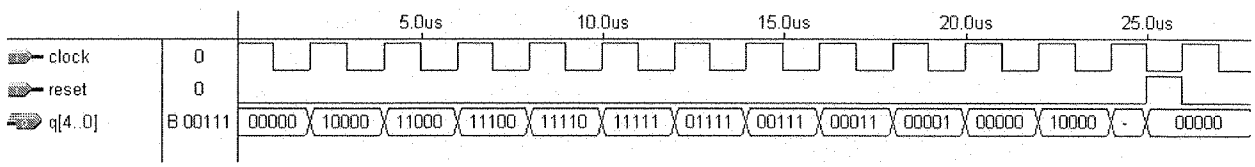

5.0us
10.0us
15.0us
20.0us
25.0us
30.0us
35.0us
40.0us
45.0us
clock
mode
ser_in
din
q3
q2
q1
q0
ff

```
ENTITY  univ_8_vhdl  IS
PORT (
      clock       :IN BIT;
      din         :IN BIT_VECTOR (7 DOWNTO 0);  -- parallel data in
      ser_in      :IN BIT;                      -- serial data in L or R
      mode        :IN INTEGER RANGE 0 TO 3;     -- 0=hold 1=rt 2=lt 3=load
      q           :BUFFER BIT_VECTOR (7 DOWNTO 0));
END univ_8_vhdl;
ARCHITECTURE  byte  OF univ_8_vhdl  IS
COMPONENT  univ_shift_vhdl
PORT (
      clock       :IN BIT;
      din         :IN BIT_VECTOR (3 DOWNTO 0);
      ser_in      :IN BIT;
      mode        :IN INTEGER RANGE 0 TO 3;
      q           :OUT BIT_VECTOR (3 DOWNTO 0));
END COMPONENT;
SIGNAL  ser_msn, ser_lsn            :BIT;
BEGIN
ser_msn <= ser_in  WHEN (mode = 1)  ELSE q(3)   WHEN (mode = 2)   ELSE '0';
ser_lsn <= ser_in  WHEN (mode = 2)  ELSE q(4)   WHEN (mode = 1)   ELSE '0';
msn:  univ_shift_vhdl   PORT MAP (clock => clock, din => din(7 DOWNTO 4),
                        ser_in => ser_msn, mode => mode, q => q(7 DOWNTO 4));
lsn:  univ_shift_vhdl   PORT MAP (clock => clock, din => din(3 DOWNTO 0),
                        ser_in => ser_lsn, mode => mode, q => q(3 DOWNTO 0));
END byte;
--------------------------------------------------------------------------------
ENTITY  univ_shift_vhdl  IS
PORT (
      clock             :IN BIT;
      din               :IN BIT_VECTOR (3 DOWNTO 0);
      ser_in            :IN BIT;
      mode              :IN INTEGER RANGE 0 TO 3;
      q                 :OUT BIT_VECTOR (3 DOWNTO 0));
END univ_shift_vhdl;
ARCHITECTURE vhdl  OF univ_shift_vhdl  IS
BEGIN
      PROCESS (clock)
      VARIABLE  ff      :BIT_VECTOR (3 DOWNTO 0);
      BEGIN
      IF (clock'EVENT AND clock = '1')  THEN
         CASE  mode  IS
            WHEN  0  =>    ff := ff;                          -- hold data
            WHEN  1  =>    ff(2 DOWNTO 0) := ff(3 DOWNTO 1);  -- shift right
                           ff(3) := ser_in;
            WHEN  2  =>    ff(3 DOWNTO 1) := ff(2 DOWNTO 0);  -- shift left
                           ff(0) := ser_in;
            WHEN  3  =>    ff := din;                         -- parallel load
         END CASE;
      END IF;
      q <= ff;                                                -- update outputs
      END PROCESS;
END vhdl;
```

**7.85**

```
SUBDESIGN  johnson_ahdl
(
      clock, reset              :INPUT;
      q[4..0]                   :OUTPUT;
)
VARIABLE
      q[4..0]                   :DFF;
      ser                       :NODE;
BEGIN
      q[].clk = clock;
      q[].clrn = !reset;
-- due to power-on reset, automatically self-starting
-- design is not self-correcting
      ser = !q0;
      q[4..0].d = (ser, q[4..1].q);
END;
```

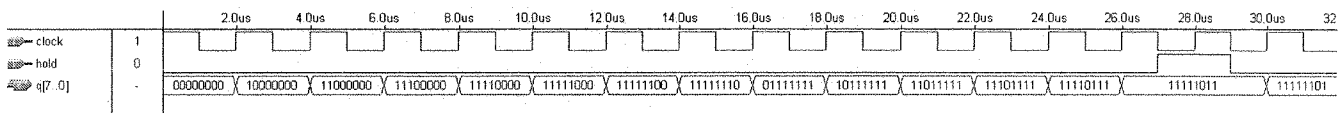
```
ENTITY  johnson_vhdl  IS
PORT (
      clock, reset              :IN BIT;
      q                         :OUT BIT_VECTOR (4 DOWNTO 0)
);
END johnson_vhdl;

ARCHITECTURE vhdl OF johnson_vhdl  IS
SIGNAL  ser                     :BIT;
BEGIN
PROCESS (clock)
      VARIABLE  reg             :BIT_VECTOR (4 DOWNTO 0);
      BEGIN
            ser <= NOT reg(0);
            IF (reset = '1') THEN  reg := "00000";
            ELSIF (clock'EVENT AND clock = '1') THEN
                  reg := (ser & reg(4 DOWNTO 1));
            END IF;
            q <= reg;
END PROCESS;
END vhdl;
```



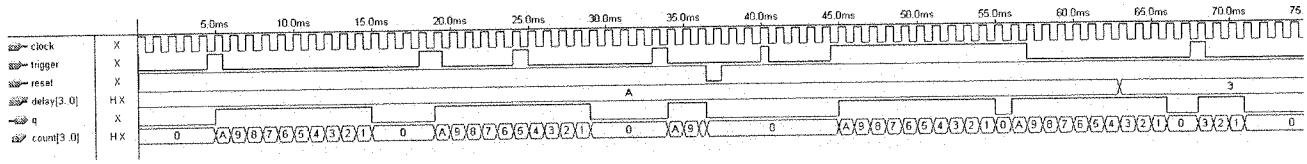
**7.86**

```
SUBDESIGN  ring_ahdl
(
      clock, hold       :INPUT;
      q[7..0]           :OUTPUT;
)
VARIABLE
      q[7..0]           :DFF;
      ser               :NODE;
BEGIN
      q[].clk = clock;
-- self-starting by filling with ones
      IF q[7..1].q == B"1111111"  THEN  ser = GND;
      ELSE  ser = VCC;
      END IF;
      IF  hold  THEN    q[].d = q[].q;
      ELSE  q[7..0].d = (ser, q[7..1].q);
      END IF;
END;
```

```
ENTITY  ring_vhdl  IS
PORT (
      clock, hold             :IN BIT;
      q                       :OUT BIT_VECTOR (7 DOWNTO 0)
);
END ring_vhdl;

ARCHITECTURE vhdl OF ring_vhdl IS
SIGNAL      ser               :BIT;
BEGIN
PROCESS (clock)
      VARIABLE  reg           :BIT_VECTOR (7 DOWNTO 0);
      BEGIN
            -- self-starting by filling with all ones
            IF (reg(7 DOWNTO 1) = "1111111")    THEN  ser <= '0';
            ELSE  ser <= '1';
            END IF;

            IF (clock'EVENT AND clock = '1')    THEN
                  IF  hold = '0'    THEN
                        reg(7 DOWNTO 0) := (ser & reg(7 DOWNTO 1));
                  END IF;
            END IF;
      q <= reg;
END PROCESS;
END vhdl;
```



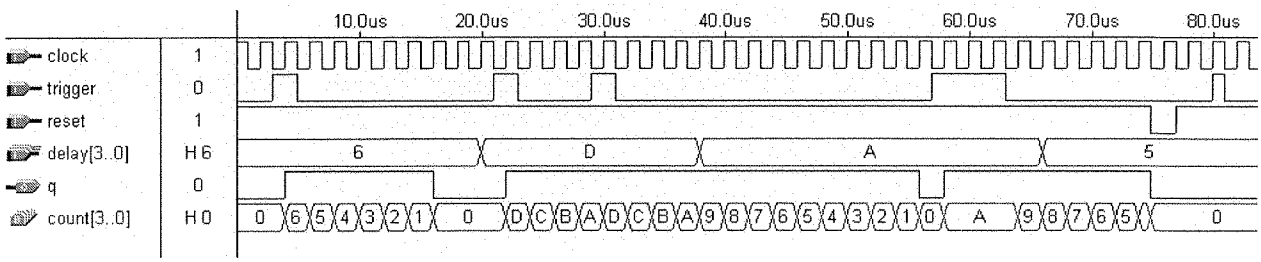
**7.87**




**7.88**

```
% retriggerable, level-sensitive digital one-shot %
SUBDESIGN  one_shot_a
(
      clock, trigger, reset          :INPUT;
      delay[3..0]                    :INPUT;
      q                              :OUTPUT;
)
VARIABLE count[3..0]                 :DFF;
BEGIN
      count[].clk = clock;
      count[].clrn = reset;

      IF trigger THEN   count[].d = delay[];
      ELSIF count[].q == B"0000" THEN count[].d = B"0000";
      ELSE count[].d = count[].q - 1;
      END IF;

      q = count[].q != B"0000";
END;
```

```
-- retriggerable, level-sensitive digital one-shot
ENTITY one_shot_v IS
PORT (
      clock, trigger, reset          :IN BIT;
      delay                          :IN INTEGER RANGE 0 TO 15;
      q                              :OUT BIT
);
END one_shot_v;

ARCHITECTURE vhdl OF one_shot_v IS
BEGIN
      PROCESS (clock, reset)
      VARIABLE count                 : INTEGER RANGE 0 TO 15;
      BEGIN
            IF reset = '0' THEN count := 0;
            ELSIF (clock'EVENT AND clock = '1') THEN
                  IF trigger = '1' THEN   count := delay;
                  ELSIF count = 0 THEN count := 0;
                  ELSE count := count - 1;
                  END IF;
            END IF;
            IF count /= 0     THEN q <= '1';
            ELSE q <= '0';
            END IF;
      END PROCESS;
END vhdl;
```




**7.89**

(a) Parallel
(b) Binary
(c) MOD-8 down
(d) MOD-10, BCD, decade
(e) Asynchronous, ripple
(f) Ring
(g) Johnson
(h) All
(i) Presettable
(j) Up/down
(k) Asynchronous, ripple
(l) MOD-10, BCD, decade
(m) Synchronous, parallel

## CHAPTER EIGHT - Integrated-Circuit Logic Families

*(Data values used to answer the questions in this chapter were obtained from one of the following sources: Data tables found throughout chapter 8; www.TI.com)*

**8.1** (a) Circuit A has $V_{NL}$ = 0.5V and $V_{NH}$ = 0.6V.
Circuit B has $V_{NL}$ = 0.4V and $V_{NH}$ = 0.7V.
(b) Circuit A because it has lower tp values.
(c) $I_{(Supply)} = (P_D)/(V_{(Supply)})$
Circuit A has $I_{(Supply)}$ = 2.67mA, and circuit B has $I_{(Supply)}$ = 2mA.

**8.2** Sample calculations (using max. values) for the 7432 IC:
$Icc_{(avg)}$ = (22mA+38mA)/2 = 30mA
$PD_{(avg)}$ for the IC = $Icc_{(avg)}$xVcc = 30mAx5.25 = 157.5mW
$PD_{(avg)}$ for one gate = 157.5mW/4 = 39.37mW
$tpd_{(avg)} = (t_{PLH}+t_{PHL})/2$ = (15ns+22ns)/2 =18.5ns

| | IC | $PD_{(avg.)}$ | $tpd_{(avg.)}$ |
|---|---|---|---|
| (a) | 7432 | 39.37 mW | 18.5 ns |
| (b) | 74S32 | 65.62 mW | 7.0 ns |
| (c) | 74LS20 | 3.93 mW | 15.0 ns |
| (d) | 74ALS20 | 2.61 mW | 10.5 ns |
| (e) | 74AS20 | 14.16 mW | 4.75 ns |

**8.3** $V_{OH(min)}$ = 4.9V ; $V_{IH(min)}$ = 3.5V
$V_{OL(max)}$ = 0.1V ; $V_{IL(max)}$ = 1.0V

(a) A positive noise spike can drive the voltage above 1.0 V level if the amplitude is greater than:

$$V_{NL} = V_{IL(max)} - V_{OL(max)}$$
$$V_{NL} = 1.0V - 0.1V = 0.9V$$

(b) A negative noise spike can drive the voltage below 3.5V level if the amplitude is greater than:

$$V_{NH} = V_{OH(min)} - V_{IH(min)}$$
$$V_{NH} = 4.9V - 3.5V = 1.4V$$

**8.4** (a) $I_{IH}$
(b) $I_{CCH}$
(c) $t_{pHL}$
(d) $V_{NH}$ (High-state noise margin)
(e) Surface mount
(f) Current-Sinking Action
(g) Fan-out
(h) Totem-pole output circuit.
(i) Current-sinking transistor
(j) 4.75V to 5.25V
(k) $V_{OH(min)}$ = 2.5V; $V_{IH(min)}$ = 2.0V
(l) $V_{IL(max)}$= 0.8V; $V_{OL(max)}$= 0.5V
(m) Current-sourcing action.

**8.5** $V_{NH} = V_{OH(min)} - V_{IH(min)}$; $V_{NL} = V_{IL(max)} - V_{OL(max)}$

(a) $V_{NH}$ = 2.7V (for LS) - 2.0V (for ALS) = 0.7V
$V_{NH}$ = 0.8V (for ALS) - 0.5V (for LS) = 0.3V
(b) $V_{NH}$ = 2.5V (for ALS) - 2.0V (for LS) = 0.5V
$V_{NL}$ = 0.8V (for LS) - 0.4V (for ALS) = 0.4V
(c) $V_{NH}$ = 0.5V; $V_{NL}$ = 0.3V
(d) 74 and 74ALS (from table 8-6 in the textbook)

**8.6** (a) Maximum number of standard logic inputs that the output of a digital circuit can drive reliably.
(b) NANDs and ANDs.
(c) Any input to a TTL circuit that is left disconnected (open) is said to be floating.
(d) Whenever a totem-pole TTL output goes from a LOW to HIGH, a high-amplitude current spike is drawn from the Vcc supply. This is because for a short period of time (about 2ns) both Q3 and Q4 are conducting. It can cause serious malfunctions during switching transitions unless some type of filtering is used. The most common technique uses small radio frequency capacitors connected from Vcc to Ground to essentially short out these high-frequency spikes.
(e) $I_{OL}$ comes from the TTL input that is being driven.
$I_{OH}$ goes into the TTL input that is being driven.

**8.7** (a) ***74AS to 74AS***

Fanout in the HIGH state (2mA/20μA) = 100
Fanout in the LOW state (20mA/.5mA) = 40
Therefore, the overall fanout is 40.

(b) ***74F to 74F***

Fanout in the HIGH state (1mA/20μA) = 50
Fanout in the LOW state (20mA/.6mA) = 33.3
Therefore, the overall fanout is 33.

(c) ***74AHC to 74AS***

Fanout in the HIGH state (8mA/20μA) = 400
Fanout in the LOW state (8mA/.5mA) = 16
Therefore, the overall fanout is 16.

(d) ***74HC to 74ALS***

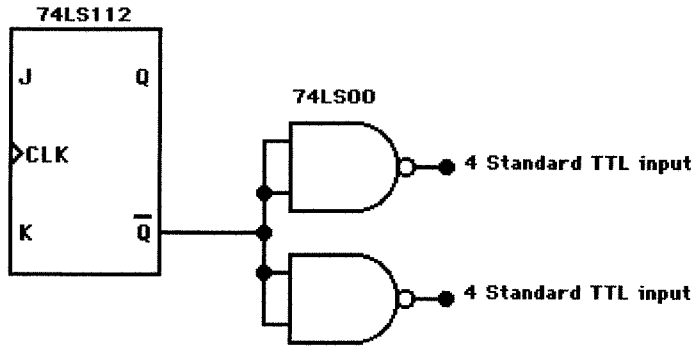
Fanout in the HIGH state (4mA/20μA) = 200
Fanout in the LOW state (4mA/.1mA) = 40
Therefore, the overall fanout is 40.

**8.8** (a) J and K inputs: 20μA in the HIGH state and 0.4mA in the LOW state.
(b) Clock inputs: 80μA in the HIGH state and 0.8mA in the LOW state.
Clear inputs: 60μA in the HIGH state and 0.8mA in the LOW state.
(c) Fan-Out: 400μA in the HIGH state and 8mA in the LOW state.
In the HIGH state: 400μA/80μA = Five 74LS112s
In the LOW state: 8mA/0.8mA = Ten 74LS112s

HIGH state is more restrictive. Thus, the answer is ***FIVE***.

**8.9** (a) Fan-Out (74LS37) = 30 standard TTL inputs in the HIGH state and 15 standard TTL inputs in the LOW state.
(b) $I_{OL}$=15x1.6mA=24mA

**8.10** **One possibility:**




**8.11** Tied together 74LS20 inputs act like 1LS input load in the LOW state and as separate LS input loads in the HIGH state. Thus, the 74LS86 output drives only 5LS input loads in the LOW state and 12LS input loads in the HIGH state. This is okay since the 74LS86 fan-out is 20LS input loads both in the HIGH and in the LOW state.

The 74AS86 output can sink in the LOW state 20mA and in the HIGH state it can supply 2mA. The 74AS20 has an input requirement of 0.5mA in the LOW state and 20μA in the HIGH state. Thus, the 74AS86 can drive 100 'AS20 inputs in the HIGH state and 40 'AS20 inputs in the LOW state.

**8.12** If a positive-going transition is applied to the input of a 74LS04, then the output will change in 10ns ($t_{PHL}$=10ns).

**8.13** **Case I:** 74LS86 output going from LOW to HIGH
$t_{AW}=t_{PLH}(max\ 74LS86)+t_{PHL}(max\ 74LS20)+t_{PLH}(max\ 74LS20)$
$t_{AW}$= 30ns+15ns+15ns = 60ns

**Case II:** 74LS86 output going from HIGH to LOW.
Same as Case I except use $t_{PHL(max\ 74LS86)}$ = 22ns.
This gives $t_{PAW}$ = 52ns.
Case I is longer. Thus, answer is 60ns.

**Case I:** 74ALS86 output going from LOW to HIGH
$t_{AW}=t_{PLH(max\ 74ALS86)}+t_{PHL(max\ 74ALS20)}+t_{PLH(max\ 74ALS20)}$
$t_{AW}$= 17ns+10ns+11ns = 38ns

**Case II:** 74ALS86 output going from HIGH to LOW.
Same as Case I except use $t_{PHL\ (max\ 74ALS86)}$ = 12ns.
This gives $t_{PAW}$ = 33ns.
Case I is longer. Thus, answer is 38ns.

**8.14** (a) MR input has an $I_{IL}$=0.4mA and a $V_{IL(max)}$=0.8V.
Thus, $R_{max}= V_{IL}(max)/I_{IL}$.
$R_{max}$= (0.8V/0.4mA) = 2KΩ
(b) MR has an $I_{IL}$=0.1mA and $V_{IL(max)}$=0.8V.
Thus, $R_{max}= V_{IL(max)}/I_{IL}$.
$R_{max}$= 0.8V/0.1mA = 8KΩ.

**8.15** (a) The circuit is used to convert a 60Hz sinewave to a 60 pps signal. The diode and voltage divider produces a positive half-cycle with reduced amplitude to drive the TTL inverter. The 74LS14 is a Schmitt Trigger which converts the slow changing input to fast-changing pulses.

(b) The $V_X$ waveform rides on a 1V baseline produced by $I_{IL}$ of the 74LS14 flowing through the bottom 4.7KΩ resistor to ground. $I_{IL\ (max)}$= 0.4mA which could produce a maximum voltage of1.88V. In practice, however, $I_{IL}$ will be about half that value. $V_X$ apparently is not dropping below $V_{T-}$ (0.6V-1.1V) needed to produce a HIGH at the 74LS14 output.

**8.16** (a) Amplitude is too low.
(b) $T_P$ of 10ns is less than $t_{W(H)}$=20ns min. value stated on the Texas Instruments data sheet.
(c) Clock LOW time is not given on the data sheet. However, $f_{max}$ is given as 30MHz. Hence, the minimum period for the clock is T=33.3ns. Consequently, $t_W(L) = T-t_W(H)$ or 33.3ns-20ns = 13.3ns. Therefore, 10ns is less than $t_W(L)$=13.3ns minimum.

**8.17** Noise is probably caused by totem-pole outputs switching from LOW to HIGH and producing ICC spikes. The technician probably forgot to connect de-coupling capacitors from Vcc to ground.

**8.18** (a) N-channel MOSFET; (b) P-channel MOSFET

**8.19** (a), (c), (e), (f), (g), (h).

**8.20** Since power drain increases with both an increase in frequency and $V_{DD}$, the best choice is (b).

**8.21** The total power dissipation for the LS04 chip is approximately equal to $I_{CCH} \times V_{CC(max)}$ or 2.4mA x 5.25V = 12.6mW. (This approximation neglects 12μA ($I_{IH}$ x 2 inputs x 6 AND gates) supplied by the 74LS04 package.)

**8.22** $V_{NH} = V_{OH(min)} - V_{IH(min)}$; $V_{NH}$ = 4.9V - 2.0V = 2.9V

**8.23** Latch-up can be triggered by high-voltage spikes or ringing at the device inputs and outputs.

When latch-up occurs, a large current may flow and destroy the IC.

In order to prevent latch-up, clamping diodes can be connected externally. A well-regulated power supply will minimize spikes on the VDD line. A current-limiting feature will limit current should latch-up occur.

**8.24** Calculations (using max. values) for the 74HC20 IC:

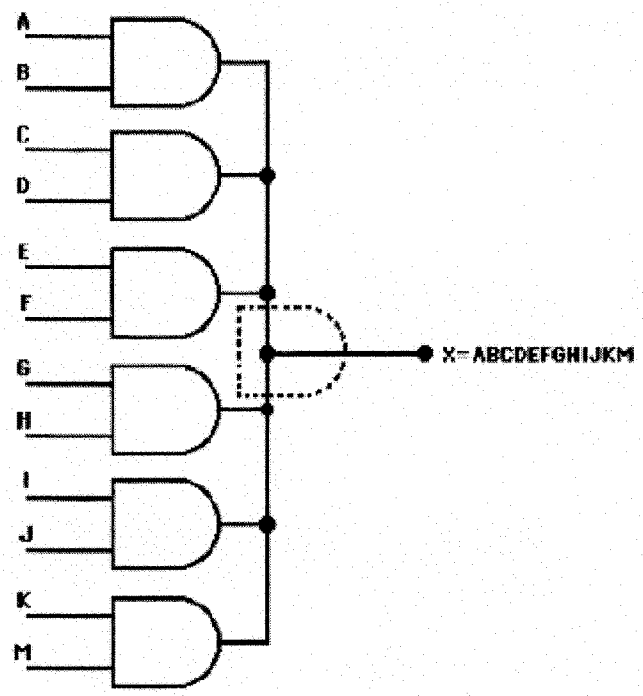
$ICC_{(avg)}$ = 20μA
$PD_{(avg)}$ for the IC = $ICC_{(avg)}$ x Vcc = 20μAx6V = 120μW
$PD_{(avg)}$ for one gate = 120μW/2 = 60μW
$tpd_{(avg@6V)}$ =24ns

Therefore, when compared with the values calculated in problem 8.2 for TTL, the 74HC20 draws less power and it is slower.

**8.25** (a) Term used to describe the logic function created when TTL open-collector outputs are tied together.
(b) It is a resistor that is used to keep a certain node in a circuit at a specific logic level. It is used to prevent that particular node from floating to an undetermined logic level as well as picking up voltage noises.

(c) Open Collector and Tristate outputs.
(d) Bus contention is that situation in which the outputs of two or more active devices are placed on the same bus line at the same time.

**8.26**




**8.27** Output of wired-AND is $\overline{AB} \cdot \overline{CD} \cdot \overline{FG}$. Thus, $X = \overline{\overline{AB} \cdot \overline{CD} \cdot \overline{FG}}$, or $X = AB + CD + FG$

**8.28** (a) Tying the output to +5V: As the output changes from HIGH to LOW the sinking-transistor (Q4) will be turned ON while the sourcing-transistor (Q3) turns OFF. If the output is tied to +5V, transistor Q4 will probably be sinking more current than it can handle and therefore be destroyed.
(b) Tying the output to ground: This would not cause any damage since the output is switching to a ground potential.
(c) Applying an input of 7V: This would cause the PN junction of one of the emitters in the multi-emitter-input transistor to be reversed bias. This reverse biasing is the normal situation when +5V is applied to the input of any TTL gate. Most likely, a slight increase of the reverse leakage current ($I_{IH}$) would occur.
(d) Tying the output to another TTL totem-pole output: If both outputs are ALWAYS at the same level no damage is likely to occur. However, if one output is trying to go to a logic LOW while the other is trying to go to a logic HIGH, then damage is likely to occur to both totem-pole outputs. This situation can cause a relatively high current to flow (55 mA) from Vcc to ground through Q3 of one gate and Q4 of the other.

**8.29** (a) 5V since the 7406 output is open-collector.
(b) Design for nominal LED current = 20 mA. $V_{RS}$ = 5V-2.4V-0.4V=2.2V. $R_S$ = 2.2V/20mA = 110Ω

**8.30** (a) ≈ +12V. (b) 40mA. ($I_{OL(7406)}$)
(c) The input to the 7407 non-inverting buffer will have to be changed from Q to $\overline{Q}$.

**8.31** With **DIRECTION = 0**, bottom buffer is disabled and upper buffer is enabled, so that signal applied to A will be transmitted to B. With **DIRECTION = 1**, B is transmitted to A.

**8.32** (a)

| *X* | *Y* | *EA* | *EB* | *EC* | *Data on Bus* |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | C |
| 0 | 1 | 0 | 1 | 0 | B |
| 1 | 0 | 0 | 1 | 0 | B |
| 1 | 1 | 1 | 0 | 0 | A |

(b) Both EA and EC would be activated (HIGH) for X=Y=1.

**8.33** A 3-bit ring counter

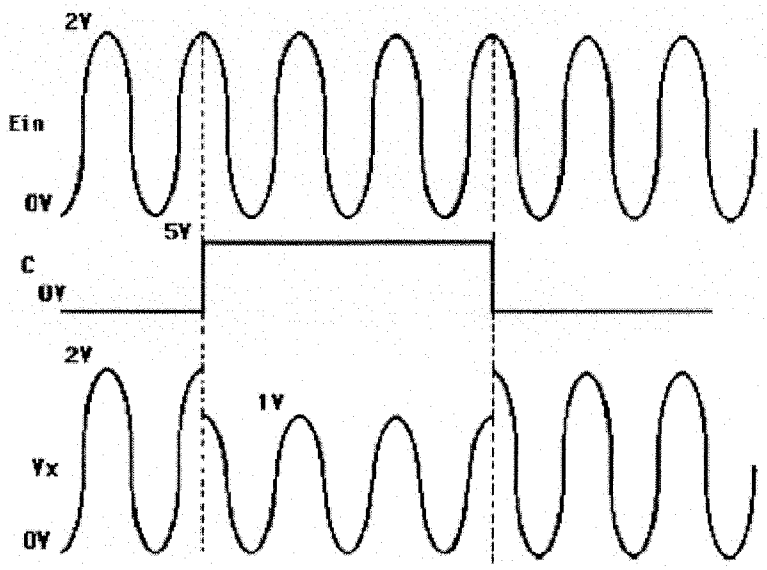
**8.34** (a) ECL (1ns); (b) ECL (25mW/gate); (c) 74AS (1.7ns); (d) 74AHC (3.7ns); (e) 74AHC (0.02pJ)

**8.35** 1. ECL outputs are nominally -0.8V and -1.7V for the logic 1 and 0.
2. An ECL logic block usually produces an output and its complement.

**8.36** Transmission gate's $R_{on}$ = 200Ω; Transmission gate's $R_{off} = 10^{12}\Omega$

**When CONTROL = 1:** $V_{OUT}$ = 5V (22KΩ)/(22KΩ + 68KΩ + 200Ω) = 1.22V
**When CONTROL = 0:** $V_{OUT}$ = 5V (22KΩ)/(22KΩ + 68KΩ + $10^{12}\Omega$) =$11\times10^{-9}$V ≈ 0 V

**8.37** When C=0, the upper switch is closed so that we have: $V_x$=(10KΩ/(10KΩ + 200Ω)) x $E_{in}$. $V_x \approx e_{IN}$
When C=1, the lower switch is closed so that: $V_x$= (10KΩ/(10KΩ+10KΩ+200Ω)) x $e_{IN}$ = 0.5 $e_{IN}$




**8.38** With **GAIN SELECT=0**, the switch is open so the op-amp gain is -(100KΩ/100KΩ) = -1
With **GAIN SELECT=1**, the switch is closed so that the op-amp gain is -(100KΩ/50KΩ) = -2

**8.39** (a) 74HCT
(b) Circuit that is designed to take a certain voltage input and translates it to a different voltage output. It is often used to interface circuits of different logic families.
(c) Because some CMOS series (i.e. 4000B) have an $I_{OL}$ capability that is not sufficient to drive even one input of the 74 or 74AS series.
(d) False.

**8.40** (c)

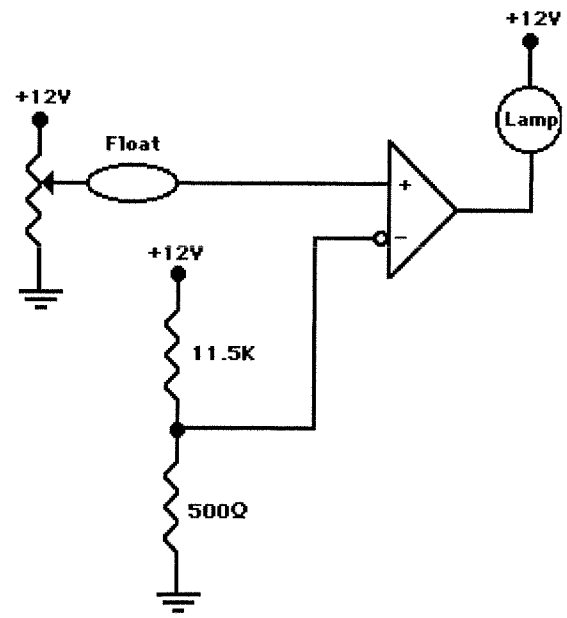
**8.41** (a) $I_{OL(4000B)}$ = 0.4mA ; $I_{IL(74AS)}$ = 2mA. Thus, a 4000B output cannot drive 74AS input directly.
(b) $I_{OL(74HC)}$ = 4mA ; $I_{IL(74AS)}$ = 2mA. Thus, a 74HC output can drive 2 (4mA/2mA) 74AS inputs.

**8.42**
1. B input of 74121 is always in logic 1 state.
2. Two unused gates on 4001B chip should have their inputs connected to ground or +5V.
3. Fan-out of the top 7400 NAND gate is exceeded.
4. Cannot wire-AND the 7400 NAND gates since they have totem-pole outputs.
5. Total current drain of all chips exceeds the capability of the power supply.
6. 74S112 (TTL) outputs are driving CMOS gates without pull-up resistors.
7. CMOS outputs are driving TTL inputs directly with no buffering.
8. Unused JK inputs of the 74S112 flip-flops should be tied to +5V through pull-up resistors.

**8.43**
1. B input of 74121 is always in logic 1 state (There is NO 74LS121).
2. Cannot wire-AND the 74LS00 NAND gates since they have totem-pole outputs.
3. Unused JK inputs of the 74LS112 flip-flops should be tied to +5V through pull-up resistors.
4. Unused inputs of 74HCT02 cannot be floating.

**8.44** The 74HC00 NAND gate is connected to 3 TTL input loads. When the output of the high-speed CMOS gate (74HC00) is LOW, it must be capable of sinking 4.8mA (3x1.6mA). However, according to table 8-12, the 74HC00 has an $I_{OL(max)}$ = 4mA. Eliminating one of the 3 TTL input loads could solve the problem. Simply disconnecting pin 2 from pin 3 of the 7402 and tying it permanently to ground can do this. Thus, the 74HC00 will be sinking 3.2mA (2x1.6mA), which is well within its $I_{OL(max)}$ of 4mA. Note that the 7402 gate is still being used as an inverter.

**8.45**




**8.46** LM35 voltage out at 38°C is 0.38V. Therefore, $R_2$=1.5KΩ and $R_1$=18KΩ.

**8.47** Use the logic probe to determine if point X can go HIGH when inputs A-F are all LOW. If X can go HIGH, then place the probe on the output of the NAND gate and inject a pulse on H. If the output still stays HIGH, probe input H while injecting the pulse at H. If a pulse is detected at H, the NAND gate is bad. No pulse indication on the probe at H means a hard short (probably on the circuit board).

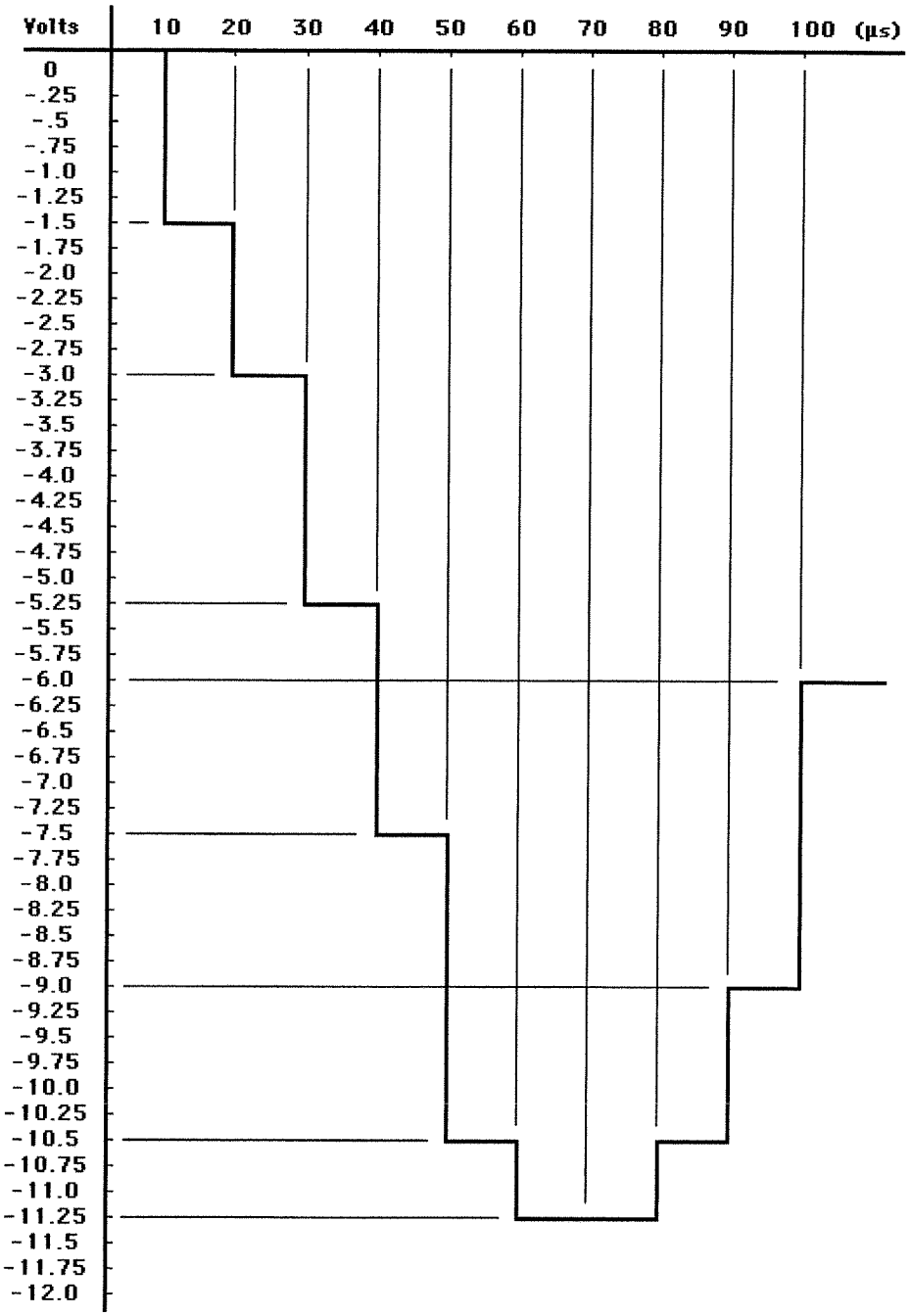
If point X will not go HIGH when A-F are LOW, inject a pulse at any output of the 74HC05 IC while probing X. If X pulses HIGH, then replace the 74HC05 IC. If the problem persists, there must be an internal short to ground on the NAND gate input. If the probe at X cannot detect a pulse injected anywhere on that node, look for a hard short to ground on the circuit board.

**8.48** Inject a Pulse anywhere along the trace between the NAND and the flip flop while monitoring the same node with a logic probe. The fact that the probe indicates a constant LOW and does not detect a pulse indicates a hard wire short to ground.

**8.49** The choice in (b) is a possible fault.

**8.50** Output A will be attenuated by 10KΩ/20KΩ = 0.5
Output B will be attenuated by 10KΩ/40KΩ = 0.25
Output C will be attenuated by 10KΩ/80KΩ = 0.125
Output D will be attenuated by 10KΩ/160KΩ = 0.0625

| μs | $x_3$ | $x_2$ | $x_1$ | $x_0$ | Vout |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0V |
| 10 | 0 | 0 | 1 | 0 | -12V x 0.125V=-1.5V |
| 20 | 0 | 1 | 0 | 0 | -12V x 0.25V=-3.0V |
| 30 | 0 | 1 | 1 | 1 | -12V x (0.25+0.125+0.0625)=-5.25V |
| 40 | 1 | 0 | 1 | 0 | -12V x (0.5+0.125)=-7.50V |
| 50 | 1 | 1 | 1 | 0 | -12V x (0.5+0.25+0.125)=-10.5V |
| 60 | 1 | 1 | 1 | 1 | -12V x (0.5+0.25+0.125+0.0625)=-11.25V |
| 70 | 1 | 1 | 1 | 1 | -12V x (0.5+0.25+0.125+0.0625)=-11.25V |
| 80 | 1 | 1 | 1 | 0 | -12V x (0.5+0.25+0.125)=-10.50V |
| 90 | 1 | 1 | 0 | 0 | -12V x (0.5+0.25)=-9.0V |
| 100 | 1 | 0 | 0 | 0 | -12V x (0.5)=-6V |


Volts
10
20
30
40
50
60
70
80
90
100 (µs)
0
-.25
-.5
-.75
-1.0
-1.25
-1.5
-1.75
-2.0
-2.25
-2.5
-2.75
-3.0
-3.25
-3.5
-3.75
-4.0
-4.25
-4.5
-4.75
-5.0
-5.25
-5.5
-5.75
-6.0
-6.25
-6.5
-6.75
-7.0
-7.25
-7.5
-7.75
-8.0
-8.25
-8.5
-8.75
-9.0
-9.25
-9.5
-9.75
-10.0
-10.25
-10.5
-10.75
-11.0
-11.25
-11.5
-11.75
-12.0

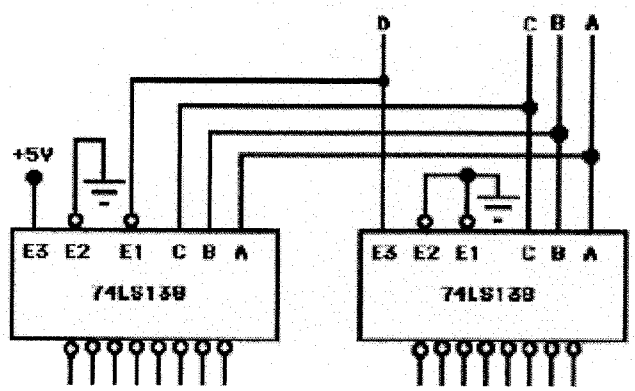
## CHAPTER NINE - MSI Logic Circuits

**9.1** (a) All of the outputs are HIGH. (b) $\overline{O}0 = 0$, $\overline{O}1 - \overline{O}7 = 1$ (c) $\overline{O}0 - \overline{O}6 = 1$, $\overline{O}7 = 0$. (d) Same as (a).

**9.2** Inputs = 6: Outputs = 64

**9.3** (a) [$\overline{O}6$] -> A2=1, A1=1, A0=0, E3=1, $\overline{E2} = 0$, $\overline{E1} = 0$
(b) [$\overline{O}3$] -> A2=0, A1=1, A0=1, E3=1, $\overline{E2} = 0$, $\overline{E1} = 0$
(c) [$\overline{O}5$] -> A2=1, A1=0, A0=1, E3=1, $\overline{E2} = 0$, $\overline{E1} = 0$
(d) [$\overline{O}0$ and $\overline{O}7$] -> Only ONE output can be active at a time.

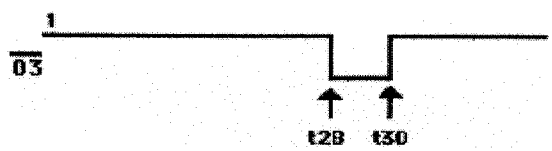
**9.4**



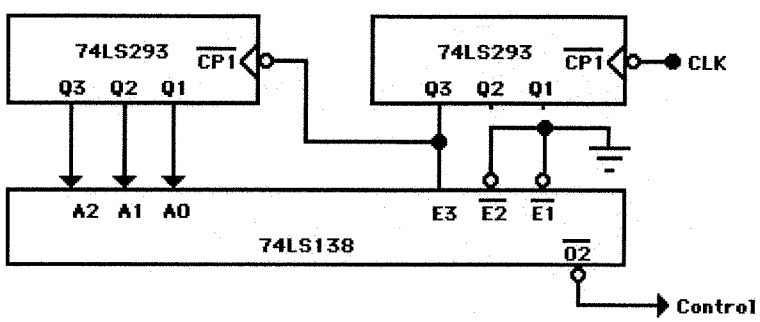

**9.5** Each 74LS293 is connected as a MOD-8 (Q0 not used). The output $\overline{O3}$ will be LOW only when A2 A1 A0 = 011, E3=1, $\overline{E2}$ =0.
This condition is present after the 28th and 29th NGTs of the clock. That is:

$28_{10} = 011100_2$ $29_{10} = 011101_2$

Thus, $\overline{O3}$ will appear as shown below:

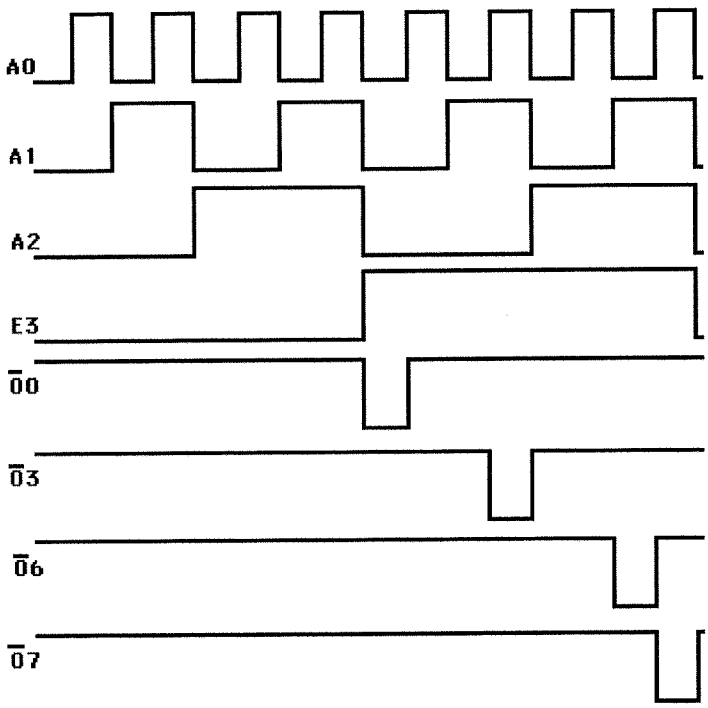



**9.6** There are several possible solutions. One is given below.



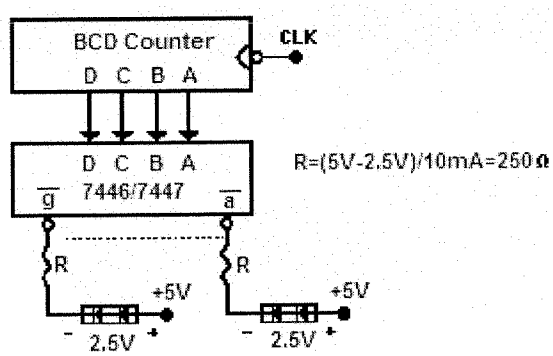
**9.7** When D=0, the C,B and A inputs determine which of the outputs $\overline{O7}$ - $\overline{O0}$ will be activated. With D held HIGH, all of these outputs will be inactive (HIGH) because the input code will be greater than $0111_2$ ($7_{10}$).

**9.8**

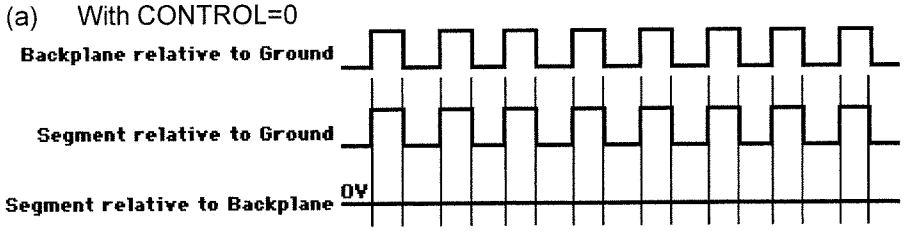



**9.9** For Relay K1 to be energized from $t_3$ to $t_5$, tie outputs $\overline{O3}$ and $\overline{O4}$ of 7445 together. For Relay K2 to be energized from $t_6$ to $t_9$ tie outputs $\overline{O6}$, $\overline{O7}$ and $\overline{O8}$ of 7445 together. This can be done because the 7445 has open-collector outputs and only one of its outputs will be active (LOW) at any one time.

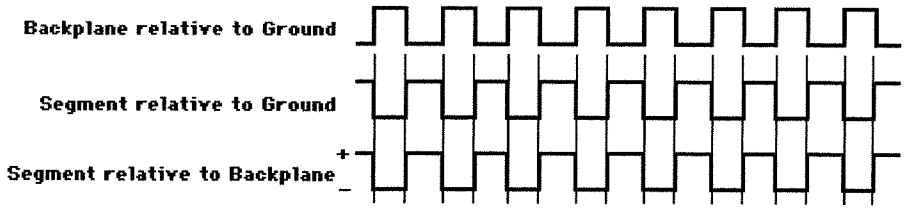
**9.10**



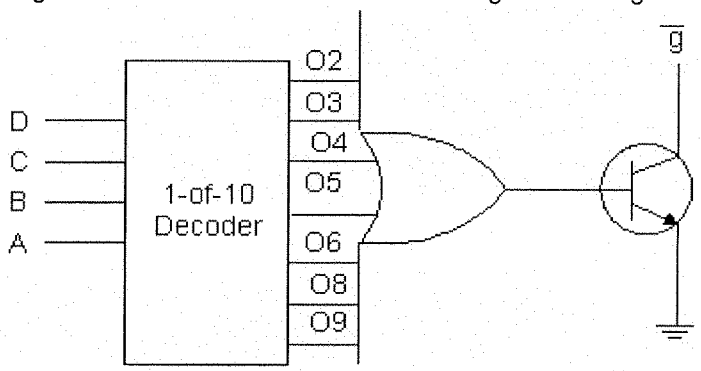

**9.11** (a) With CONTROL=0



(b) With CONTROL=1



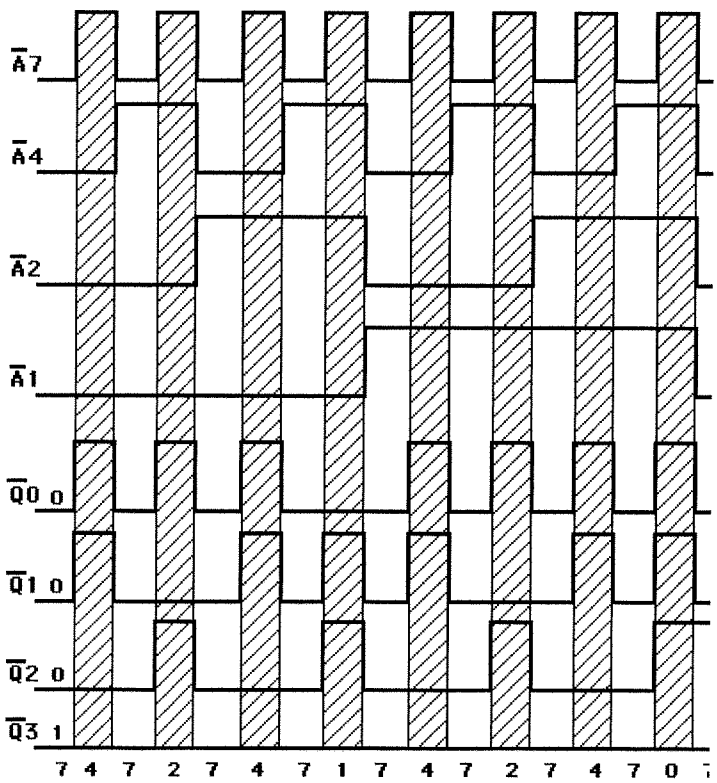

**9.12** The 'g' segment should be active for the following decimal digits: 2,3,4,5,6,8,9.



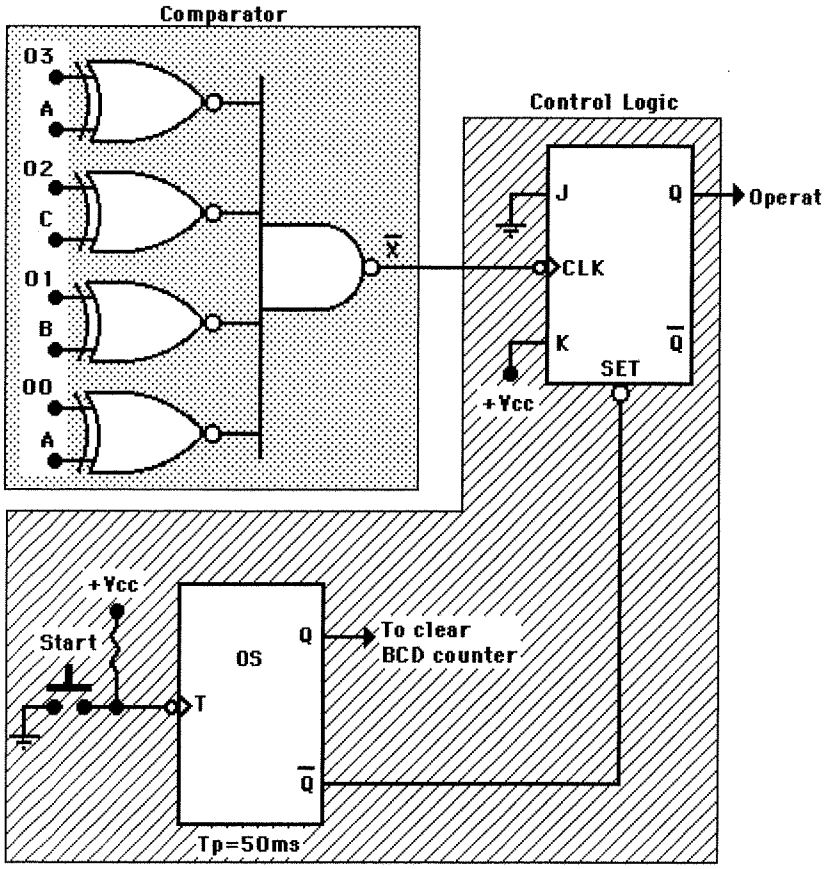

**9.13** (a) Encoder (b) Encoder (c) Decoder (d) BCD-to-7-segment decoder (e) Decoder/driver

**9.14** The 74147 responds to higher-order $\overline{A}8$. Thus, the encoder output will be the complement of the code for 8. That is $\overline{1}\,\overline{0}\,\overline{0}\,\overline{0} = 0111$.

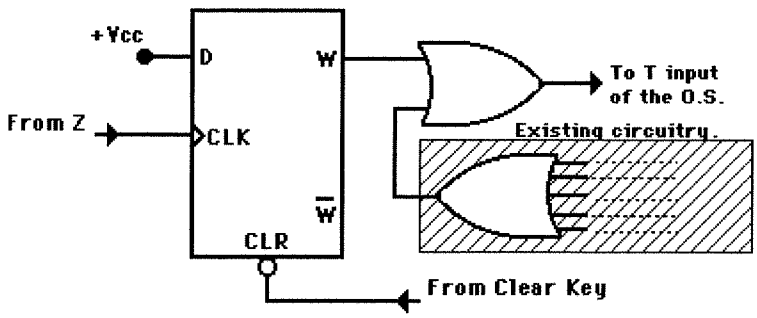
**9.15**



**9.16** One possible design is shown below:




**9.17** If the fourth key were actuated, it would be entered into the MSD. For example, if you entered 3095, the 5 would replace the 3 in the MSD so you would see 509 on the displays. The following circuit will prevent the fourth key actuation from having any effect (until CLEAR is again actuated). The PGT at Z will set W=1. This will hold gate output HIGH and prevent further key actuation from clocking the ring counter.




**9.18** Choice (b) is the only one, which would give the symptoms described. If Q is triggering the X,Y,Z register, a negative-going transition will occur immediately upon the actuation of a digit key. This would cause the shift register to shift and produce a positive-transition at one of the X,Y or Z outputs before the outputs of the encoder have stabilized due to switch bounce.

Choice (c) could not be the cause of the symptoms described, since it would trigger the OS more than once for a given key actuation, thereby affecting more than one register.

**9.19** Choice (a) could cause this symptom because the open input to the OR gate would produce a constant HIGH at its output (for TTL), thereby preventing any clocking of the ring counter. Choice (d) could also cause this symptom. The LOW at Y could pull down the HIGH at Z so that the ring counter stays at $000_2$.

**9.20** (a) Yes. An open would result in a constant HIGH at the D inputs of the FFs. Since this is the LSB, the result would always be an odd-numbered entry.
(b) No. OR-gate output would be stuck HIGH, freezing all operation
(c) No. Same as (b).

**9.21** $A_2$ Bus line is open between IC-Z2 and IC-Z3.

**9.22** Either the output of the INVERTER or input (2) on the NAND gate of Z4 is internally, or externally shorted to Ground.

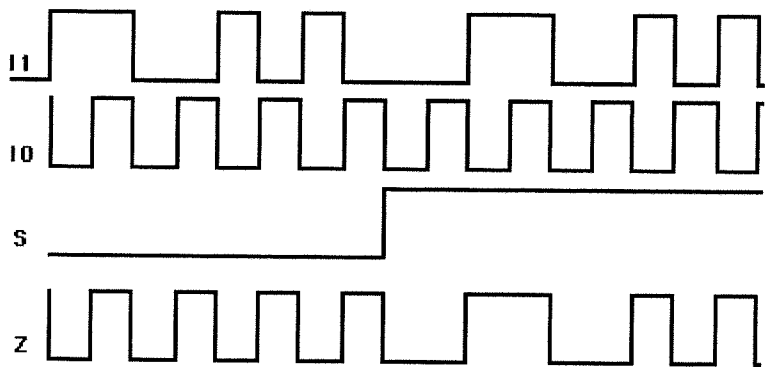
**9.23** (a) Segment **'g'** would be much brighter than the other six segments.
(b) Segment **'g'** of display and/or output transistor of 7446/7447 could burn out.

**9.24** Inputs D and C to the BCD-to-7-segment decoder/driver have been wired in reversed order.

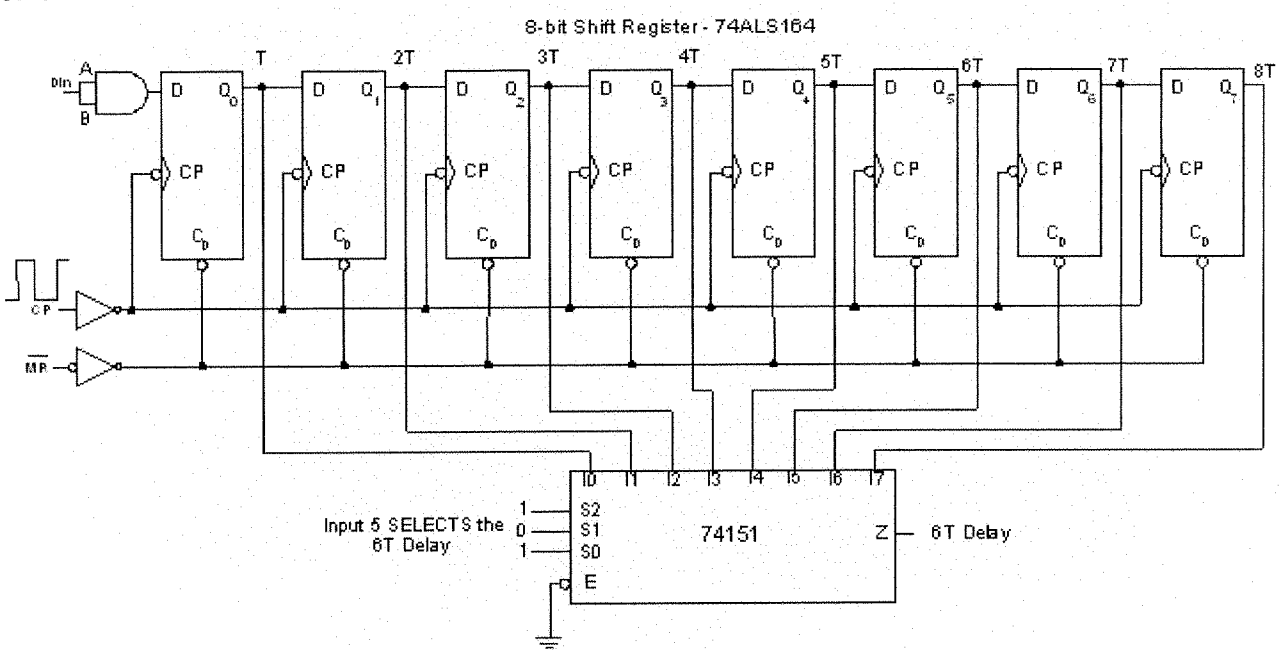
**9.25** Segments **'a'** and **'b'** of the display are always ON. A short between the cathodes of segments 'a' and 'b' must exist.

**9.26** One possibility: The unused inputs of the remaining XOR gate were left floating (to get 7 XOR gates you need 2 quad 2-input XOR ICs). Most probable cause: Connection **'f'** from the BCD-to-7 segment dec./driver to the XOR gate is open.

**9.27**



**9.28**

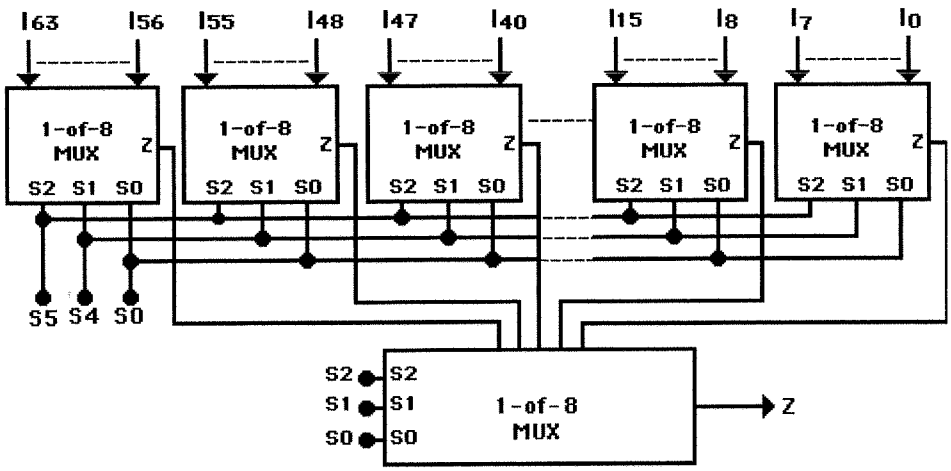



**9.29**

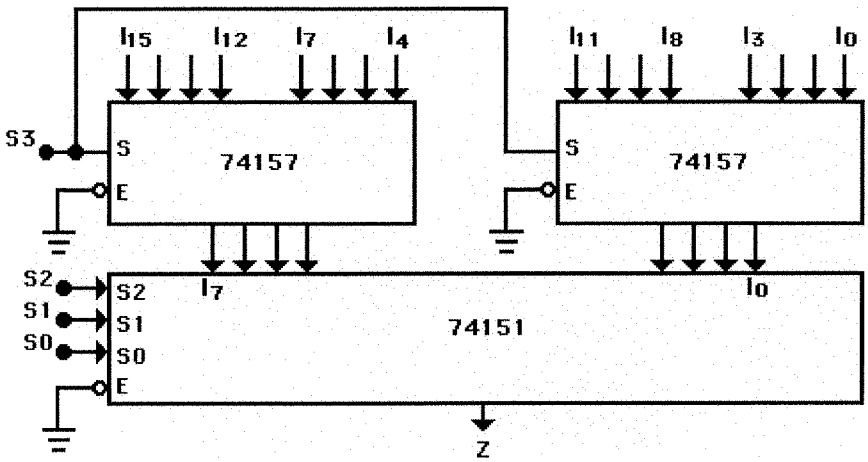
| S1 | S0 | Output |
|---|---|---|
| 0 | 0 | $I_0$ |
| 0 | 1 | $I_2$ |
| 1 | 0 | $I_1$ |
| 1 | 1 | $I_3$ |

The circuit of figure 9.61 functions as a 4-input Multiplexer.

**9.30**

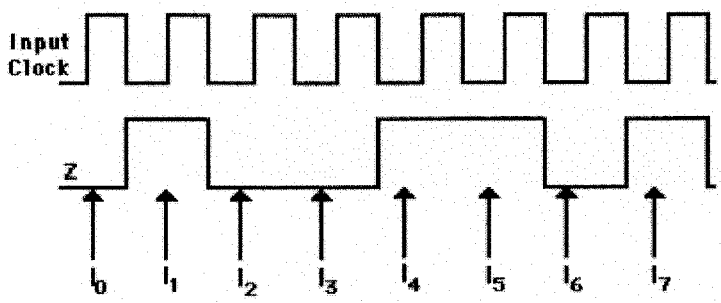


**9.31**




**9.32** (b) The total number of connections in the circuit using the Multiplexers is 63, not including Vcc and GND. The total number required for the circuit using separate decoder/drivers and displays for each BCD counter is 66.

**9.33** Assume counter starts at $000_2$.




**9.34** When MSB of counter goes HIGH, it disables the MUX output so that Z=0.


**9.35**

| C | B | A | Z | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | |
| 0 | 0 | 1 | 0 | |
| 0 | 1 | 0 | 0 | |
| 0 | 1 | 1 | 1 | $\rightarrow I_3$ |
| 1 | 0 | 0 | 0 | |
| 1 | 0 | 1 | 1 | $\rightarrow I_5$ |
| 1 | 1 | 0 | 1 | $\rightarrow I_6$ |
| 1 | 1 | 1 | 1 | $\rightarrow I_7$ |

Therefore, connect $I_3$, $I_5$, $I_6$ and $I_7$ to Vcc. Connect all other MUX inputs to ground.

**9.36**

| D | C | B | A | Z | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | |
| 0 | 0 | 0 | 1 | 1 | → $I_1$ |
| 0 | 0 | 1 | 0 | 0 | |
| 0 | 0 | 1 | 1 | 0 | |
| 0 | 1 | 0 | 0 | 0 | |
| 0 | 1 | 0 | 1 | 1 | → $I_5$ |
| 0 | 1 | 1 | 0 | 0 | |
| 0 | 1 | 1 | 1 | 0 | |
| 1 | 0 | 0 | 0 | 1 | → $I_8$ |
| 1 | 0 | 0 | 1 | 0 | |
| 1 | 0 | 1 | 0 | 0 | |
| 1 | 0 | 1 | 1 | 1 | → $I_{11}$ |
| 1 | 1 | 0 | 0 | 0 | |
| 1 | 1 | 0 | 1 | 0 | |
| 1 | 1 | 1 | 0 | 1 | → $I_{14}$ |
| 1 | 1 | 1 | 1 | 1 | → $I_{15}$ |

Connect inputs $I_1$, $I_5$, $I_8$, $I_{11}$, $I_{14}$ and $I_{15}$ to Vcc. Connect all other MUX inputs to ground.

**9.37**

| D | C | B | A | Z | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | |
| 0 | 0 | 0 | 1 | 0 | |
| 0 | 0 | 1 | 0 | 1 | → $\overline{A}\,B\,\overline{C}\,\overline{D}$ |
| 0 | 0 | 1 | 1 | 0 | |
| 0 | 1 | 0 | 0 | 1 | → $\overline{A}\,\overline{B}\,C\,\overline{D}$ |
| 0 | 1 | 0 | 1 | 0 | |
| 0 | 1 | 1 | 0 | 0 | |
| 0 | 1 | 1 | 1 | 0 | |
| 1 | 0 | 0 | 0 | 0 | |
| 1 | 0 | 0 | 1 | 1 | → $A\,\overline{B}\,\overline{C}\,D$ |
| 1 | 0 | 1 | 0 | 1 | → $\overline{A}\,B\,\overline{C}\,D$ |
| 1 | 0 | 1 | 1 | 0 | |
| 1 | 1 | 0 | 0 | 0 | |
| 1 | 1 | 0 | 1 | 0 | |
| 1 | 1 | 1 | 0 | 0 | |
| 1 | 1 | 1 | 1 | 0 | |

$$Z = \overline{A}\,B\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,C\,\overline{D} + A\,\overline{B}\,\overline{C}\,D + \overline{A}\,B\,\overline{C}\,D$$

$$Z = \overline{A}\,B\,\overline{C} + \overline{A}\,\overline{B}\,C\,\overline{D} + A\,\overline{B}\,\overline{C}\,D$$

**9.38** (a) $Z = \overline{C}\,B\,\overline{A} + D\,\overline{C}\,\overline{B}\,A + \overline{D}\,C\,\overline{B}\,\overline{A}$

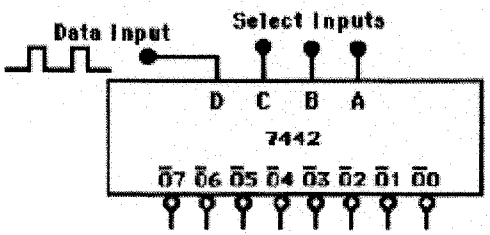
| | *D=0* | | *D=1* | |
|---|---|---|---|---|
| *DCBA* | *Z* | *DCBA* | *Z* | *$I_{CBA}$* |
| 0 0 0 0 | 0 | 1 0 0 0 | 0 | $I_0$=0 |
| 0 0 0 1 | 0 | 1 0 0 1 | 1 | $I_1$=D |
| 0 0 1 0 | 1 | 1 0 1 0 | 1 | $I_2$=1 |
| 0 0 1 1 | 0 | 1 0 1 1 | 0 | $I_3$=0 |
| 0 1 0 0 | 1 | 1 1 0 0 | 0 | $I_4=\overline{D}$ |
| 0 1 0 1 | 0 | 1 1 0 1 | 0 | $I_5$=0 |
| 0 1 1 0 | 0 | 1 1 1 0 | 0 | $I_6$=0 |
| 0 1 1 1 | 0 | 1 1 1 1 | 0 | $I_7$=0 |

(b)

| | D=0 | | D=1 | |
|---|---|---|---|---|
| DCBA | Z | DCBA | Z | $I_{CBA}$ |
| 0 0 0 0 | 1 | 1 0 0 0 | 0 | $I_0=\overline{D}$ |
| 0 0 0 1 | 0 | 1 0 0 1 | 0 | $I_1=0$ |
| 0 0 1 0 | 1 | 1 0 1 0 | 1 | $I_2=1$ |
| 0 0 1 1 | 1 | 1 0 1 1 | 1 | $I_3=1$ |
| 0 1 0 0 | 1 | 1 1 0 0 | 1 | $I_4=1$ |
| 0 1 0 1 | 1 | 1 1 0 1 | 1 | $I_5=1$ |
| 0 1 1 0 | 0 | 1 1 1 0 | 0 | $I_6=0$ |
| 0 1 1 1 | 0 | 1 1 1 1 | 1 | $I_7=D$ |

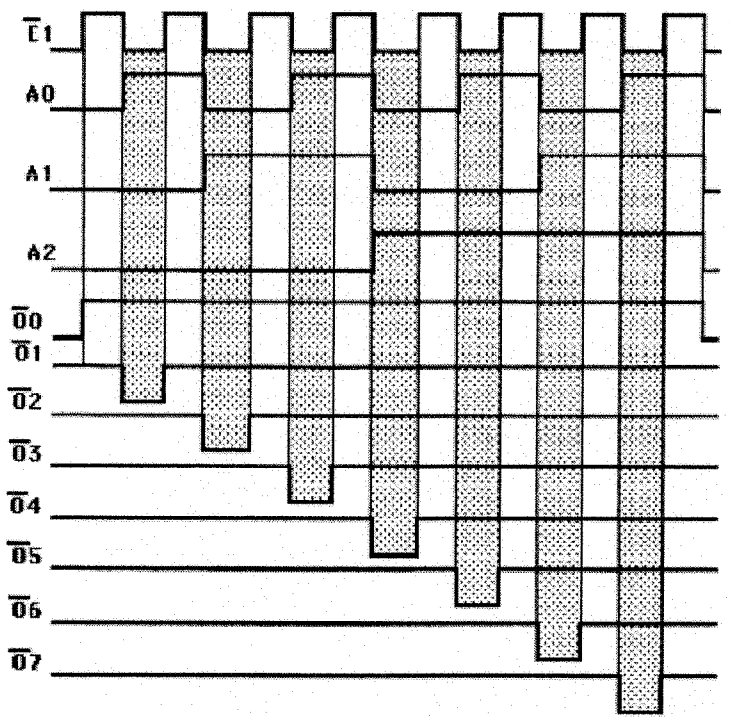
**9.39** (a) Encoder, Multiplexer (b) Multiplexer, Demultiplexer (c) Multiplexer (d) Encoder (e) Decoder, Demultiplexer (f) Demultiplexer (g) Multiplexer

**9.40**




When A=B=C=0, output $\overline{Q}0$ will follow the DATA INPUT; all other outputs stay HIGH.

**9.41**

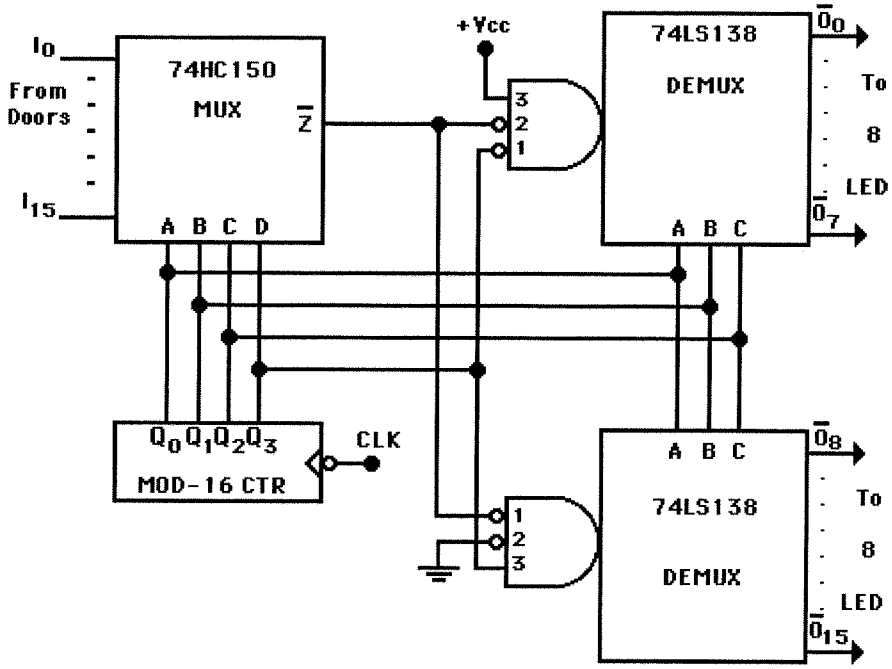


**9.42** (a) All of the LEDs will be off.

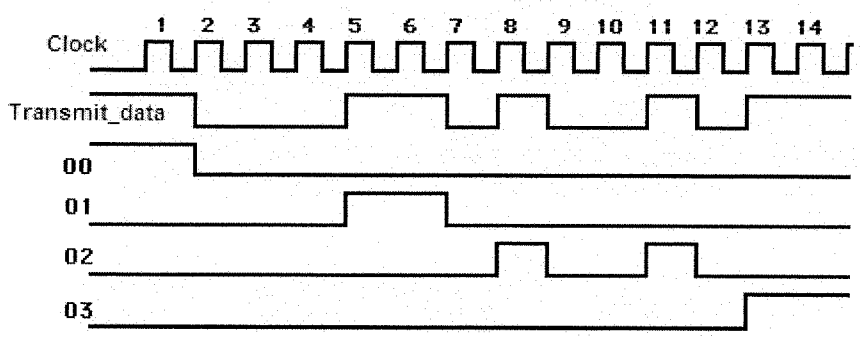
(b) Each LED will flash in sequence (0 through 7) for 0.1s.

(c) When the MOD-8 CTR reaches the count of 2 (010), the output of the MUX will be the complement of I2 (LOW). At this time the DEMUX will be active and O2 will be LOW allowing LED #2 to be lit for 0.1s. When the counter reaches the count of $6_{10}$ ($110_2$), which will be 0.4s later, LED #6 will lit for 0.1s. This will continue as the counter sequences through its 8 states.

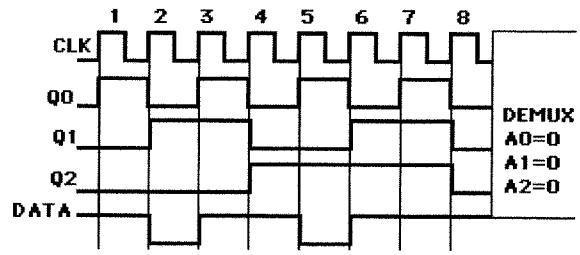
**9.43** As the circuit below shows, five lines will go to the remote monitoring panel.

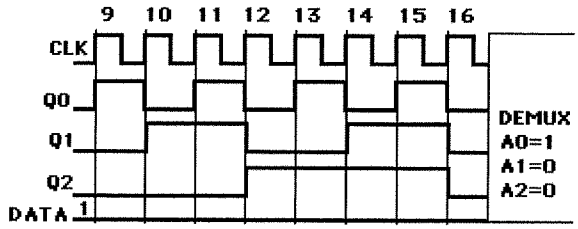


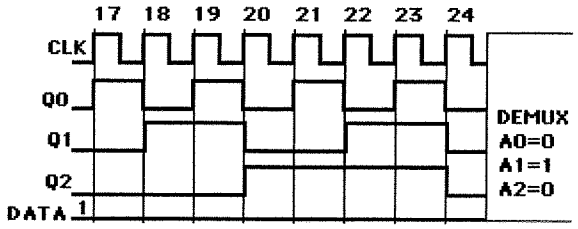

**9.44**

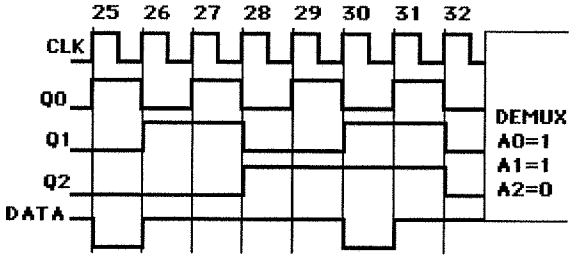


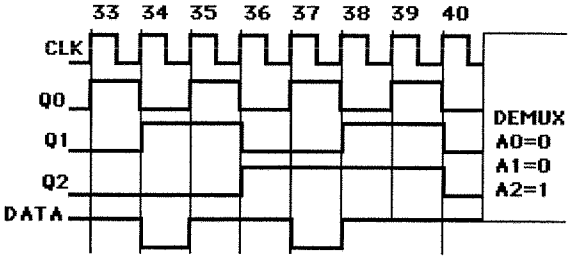
**9.45**

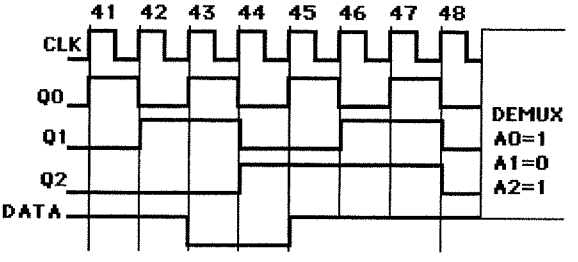

1 2 3 4 5 6 7 8
CLK
Q0
Q1
Q2
DATA
DEMUX
A0=0
A1=0
A2=0


9 10 11 12 13 14 15 16
CLK
Q0
Q1
Q2
DATA
1
DEMUX
A0=1
A1=0
A2=0


17 18 19 20 21 22 23 24
CLK
Q0
Q1
Q2
DATA
1
DEMUX
A0=0
A1=1
A2=0


25 26 27 28 29 30 31 32
CLK
Q0
Q1
Q2
DATA
DEMUX
A0=1
A1=1
A2=0


33 34 35 36 37 38 39 40
CLK
Q0
Q1
Q2
DATA
DEMUX
A0=0
A1=0
A2=1


41 42 43 44 45 46 47 48
CLK
Q0
Q1
Q2
DATA
DEMUX
A0=1
A1=0
A2=1

**9.46** (a) Sensor output 3 will not be allowed to go HIGH when Actuator #3 is activated at the count of 3. This will not allow the counter to be incremented. Thus, the process sequence will be terminated at this time.

(b) When Actuator #3 is activated sensor #3 and $I_4$ of MUX will be HIGH. Since at this time the select inputs of the MUX are at the count of $3_{10}$ ($011_2$) its output will reflect the status of $I_3$, which is LOW. Thus, the counter won't get incremented and the process sequence will halt.

**9.47** Clearly, the MSB (Za) of the 74157 MUX (tens) never goes HIGH. A possible cause could be that the connection from MUX (tens) to the BCD-to-7 segment decoder/driver is shorted to ground.

**9.48** Connections $Q_0$ and $Q_1$ of the MOD-8 CTR are reversibly connected to the select inputs of the MUX and DEMUX. This will cause the select inputs to change in the sequence: 0, 2, 1, 3, 4, 6, 5, 7.

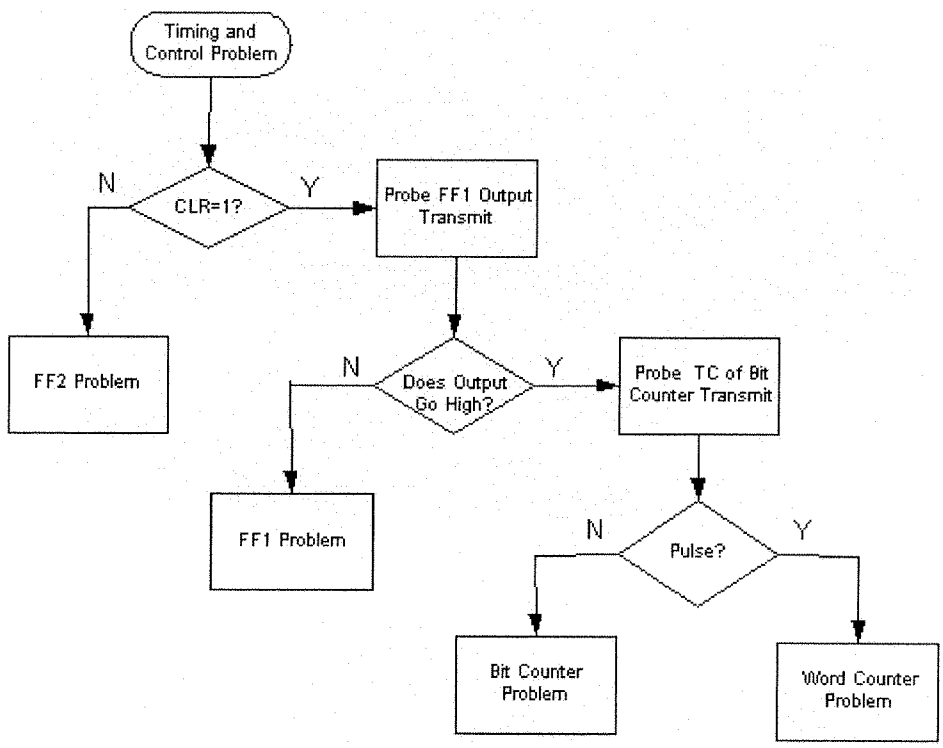
**9.49** There is a short between inputs $I_6$ and $I_7$ of the 74HC151 MUX.

**9.50** By examining the waveforms the following is observed:

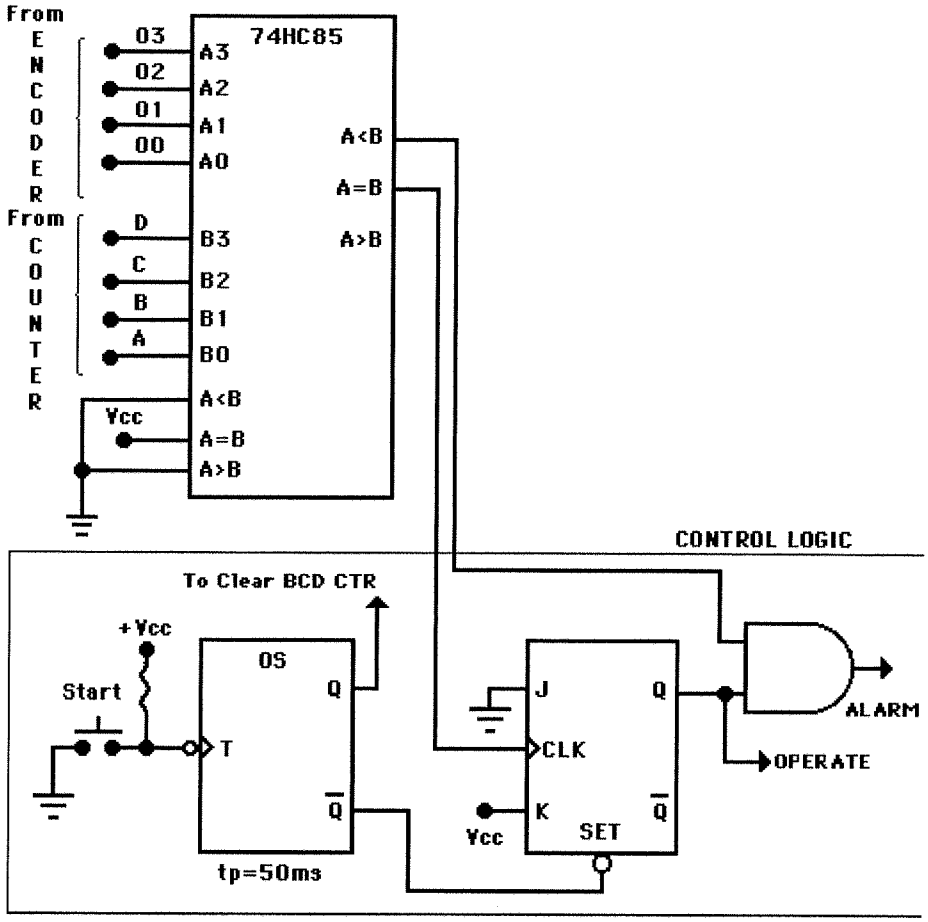
I. All of the signals appear to be correct between $t_0$ and $t_9$, with the $O_0$ signal containing the serial data from register A, and $O_1$ containing the serial data from register B.
II. The $O_2$ and $O_3$ outputs are never activated.
III. Between $t_{10}$ and $t_{14}$, the $O_0$ output, rather than $O_2$, contains the serial data from register C.
IV. Between $t_{14}$ and $t_{18}$, the $O_1$ output, rather than $O_3$, contains the serial data from register D.

It appears that the select inputs of the receiver's DEMUX are selecting only $O_0$ and $O_1$. This would happen if $S_1$ were stuck in the LOW state. This stuck node could be caused by an internal short to ground at $S_1$, $Q_1$, or an external short to ground.

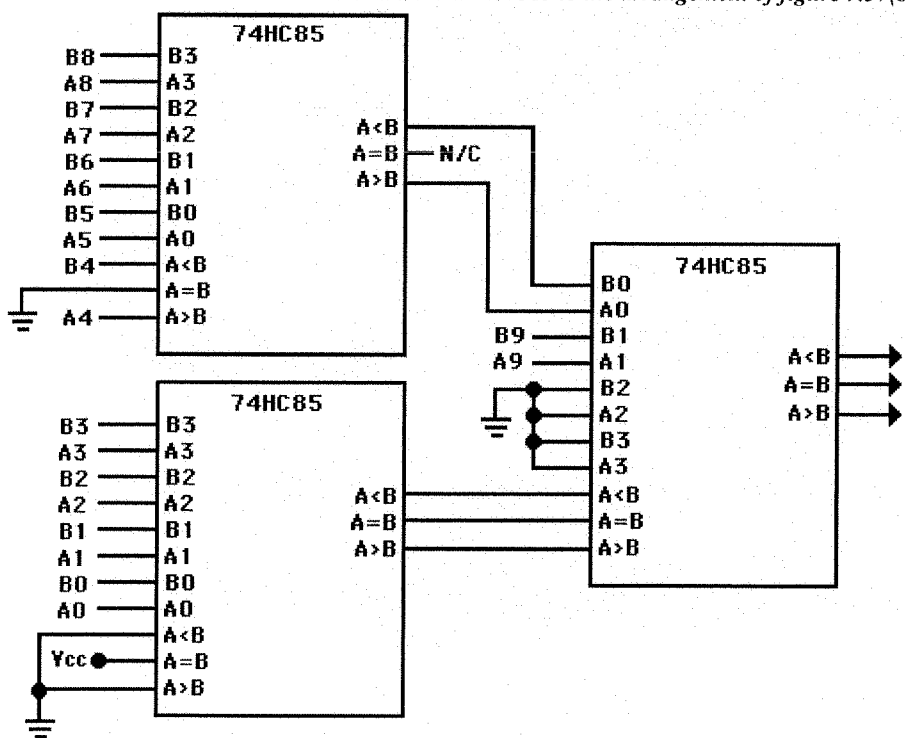
**9.51**



**9.52**




**9.53** * *another solution would be to add a third 74HC85 to the arrangement of figure 9.37(b).*



**9.54**

| | $D_1$ | $C_1$ | $B_1$ | $A_1$ | $D_0$ | $C_0$ | $B_0$ | $A_0$ |
|---|---|---|---|---|---|---|---|---|
| BCD input = | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |

| | c | $S_3$ | $S_2$ | $S_1$ | $S_0$ |
|---|---|---|---|---|---|
| Sum at the output of the first 75LS83 = | 0 | 1 | 1 | 1 | 0 |

| | $S_3$ | $S_2$ | $S_1$ | $S_0$ |
|---|---|---|---|---|
| Sum at the output of the second 75LS83 = | 1 | 0 | 0 | 0 |

| | $b_6$ | $b_5$ | $b_4$ | $b_3$ | $b_2$ | $b_1$ | $b_0$ |
|---|---|---|---|---|---|---|---|
| Binary output = | 1 | 0 | 0 | 0 | 1 | 0 | 1 |

**9.55** By looking at the results at the "Binary output" it can be concluded that a problem exists for the first two BCD conversions (52 and 95), and that the last conversion for the BCD number 27 is without fault. Further investigation can bring to conclusion that for the conversions that exhibit a fault the actual condition of $b_0$ is always the opposite of what it should be with the exception of the last conversion for the BCD number 27. Now, what than is the major difference between the two first BCD numbers (52, 95) and the last one (27)?
The answer is that the last BCD number (27) is the only one with $A_0$=$b_0$. Thus, the most probable cause for the fault is $b_0$ being connected to $B_0$ instead of $A_0$.

**9.56** True: (a), (c), (e) False: (b), (d)

**9.57** *Register C:* $\overline{OE_B} = 0; \overline{IE_C} = 1$

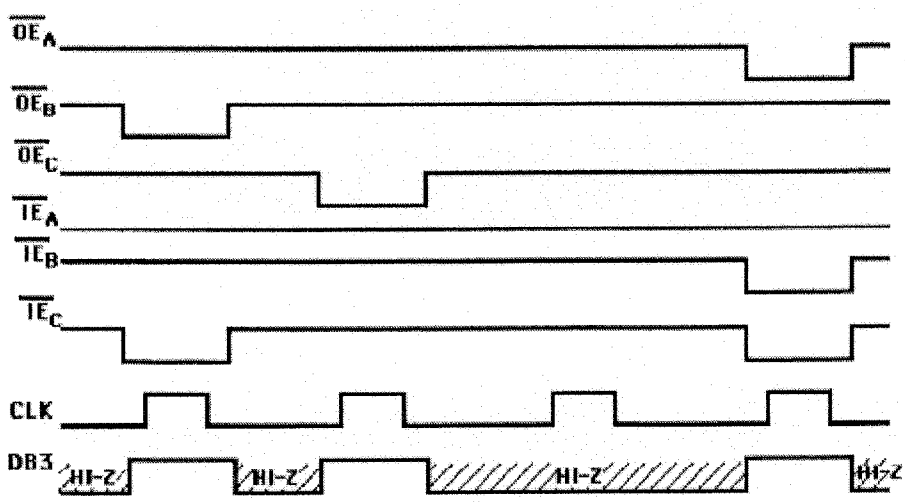
*Registers A and B:* $\overline{OE_B} = \overline{OE_A} = 1; \overline{IE_B} = \overline{IE_A} = 0$

The contents of register [C] will be transferred to registers [A] and [B] when a CLOCK pulse is applied.

**9.58** (a) @ t1 [A] = 1011; [B] = 1000; [C] = 1000{ [B]→[C] }
@ t2 [A] = 1011; [B] = 1000; [C] = 1000
@ t3 [A] = 1011; [B] = 1000; [C] = 1000
@ t4 [A] = 1011; [B] = 1011; [C] = 1011{ [A]→[B]→[C] }

(b) Since the registers' outputs are all in their HI-Z mode, register A would be loaded with unpredictable data (noise) that would be floating on the Bus.

**9.59**



**9.60** (a) Set switches at SW3=1, SW2=0, SW1=SW0=1. Make $\overline{E_{SW}} = 0$ and $\overline{IE_A} = 0$; all other input and output enables are kept HIGH. Apply CLOCK pulse. Set switches at 0001. Make $\overline{E_{SW}} = 0$ and $\overline{IE_B} = 0$. Apply CLOCK pulse. Set switches at 1110. Make $\overline{E_{SW}} = 0$ and $\overline{IE_B} = 0$. Apply CLOCK pulse.

(b) The 74HC174 register is clocked by the same CLOCK signal as the A, B, C registers. Thus, each time data is transferred to one of these registers, the 74HC174 will latch the same data. Since the last operation transferred 1110 to register C, the 74HC174 will also hold $1110_2$.

**9.61** (a) At $t_1$, $\overline{E_{SW}} = 0$ and $\overline{IE_A} = 0$ because of the levels at $OS_1$, $OS_0$, $IS_1$ and $IS_0$. This will transfer levels from the switches to register A. Thus, [A] = 1001. The 74HC174 register will also hold $1001_2$. Other registers have $0000_2$. At $t_2$, $\overline{OE_A} = 0$ and $\overline{IE_C} = 0$. Thus, [A]→[C] so that [C]=$1001_2$, as does the 74HC174 register. At $t_3$, $\overline{OE_C} = 0$ and $\overline{IE_B} = 0$. Thus, [C]→[B] so that [B]=1001. All registers are now holding $1001_2$.

(b) In theory, the answer is "no" because only one decoder output can go LOW at one time, and so only one device's outputs can be enabled. However, in practice there will be a very short overlap interval where two devices' outputs are enabled as the output select code changes from one code to another.

**9.62** 1. Connect $\overline{E_{SW}}$ from figure 9-67 to the $\overline{ENABLE}$ input of the 74HC541.
2. Connect $S_{W0}$-$S_{W3}$ to the inputs of the first four-tristate buffers of the 74HC541 IC. Connect the other four unused inputs of the 74HC541 IC to ground.

**9.63** (a) $57FA_{16}$

To activate the memory module $\overline{O2}$ of the 74LS138 must be LOW. Therefore, A0=0, A1=1, A2=0, E3=1, $\overline{E2} = 0$, $\overline{E1} = 0$. Thus, CP must go LOW at the same time the address bus has the following:

| A15 | A14 | A13 | A12 | A11 | A10 | A9 | A8 | A7 | A6 | A5 | A4 | A3 | A2 | A1 | A0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | x | x | x | x | x | x | x | x | x | x | x |

(b) Hence, any address from $5000_{16}$ to $57FF_{16}$ will be acceptable.

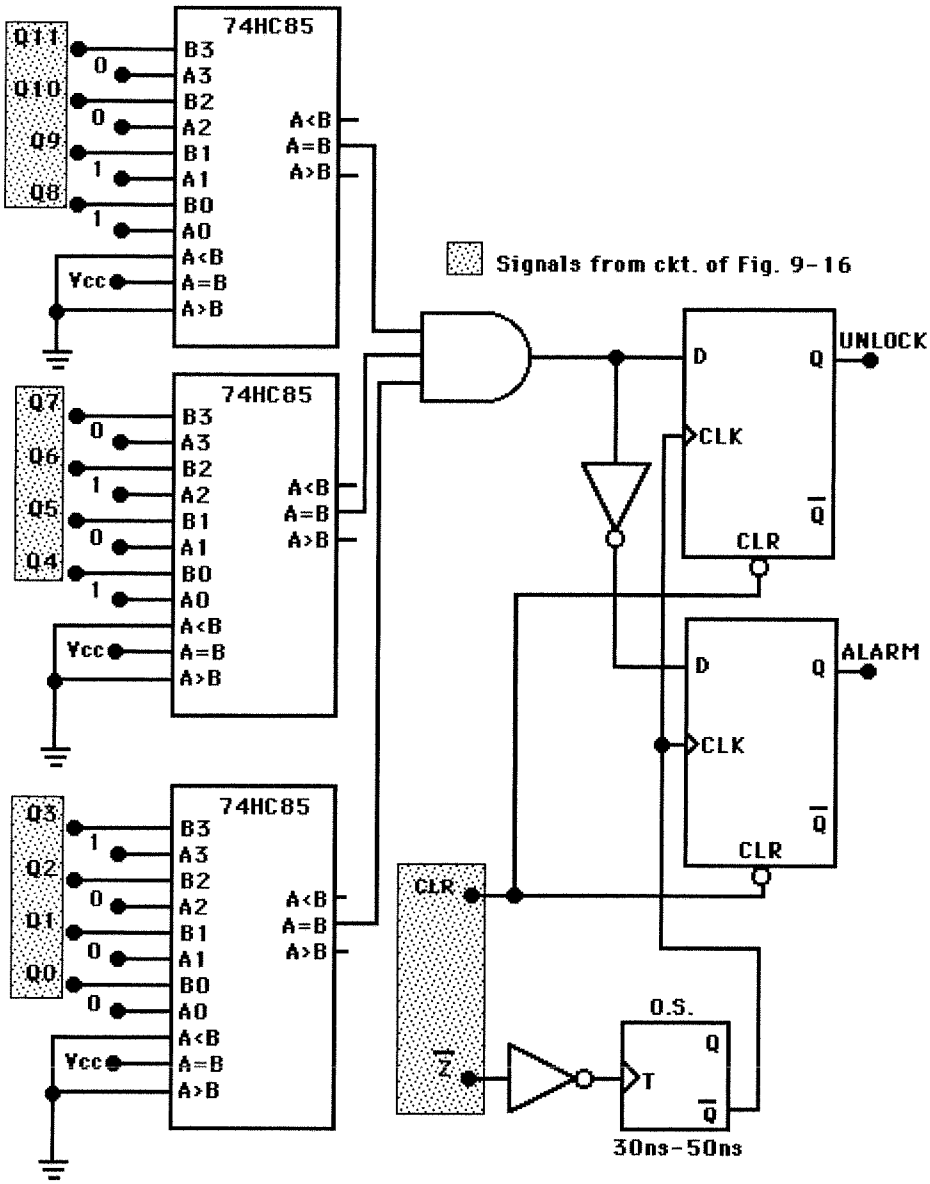
(c) To activate the memory module $\overline{O4}$ of the 74LS138 must be LOW. Therefore, A0=0, A1=0, A2=1, E3=1, $\overline{E2} = 0$, $\overline{E1} = 0$. Thus, CP must go LOW at the same time the address bus has the following:

| A15 | A14 | A13 | A12 | A11 | A10 | A9 | A8 | A7 | A6 | A5 | A4 | A3 | A2 | A1 | A0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | x | x | x | x | x | x | x | x | x | x | x |

Thus, any address from $9000_{16}$ to $97FF_{16}$ will activate the second module.

(d) No. In order for the MPU to READ from or WRITE to both modules at the same time both memory-modules would have to be active at the same time. This is impossible because only one output from the decoder can be active (LOW) at any one time.
Therefore, both $\overline{O2}$ and $\overline{O4}$ of the decoder 74LS138 cannot be LOW at the same time.

**9.64** **One possible design is shown below:**




**9.65**

```
% BCD TO DECIMAL DECODER SIMILAR TO A 7442 %

SUBDESIGN  prob9_65
(
        a[3..0]                                         :INPUT;         --binary inputs

        O9,O8,O7,O6,O5,O4,O3,O2,O1,O0                   :OUTPUT;        --decoded outputs
)
BEGIN
        DEFAULTS
                O8=VCC;O9=VCC;
                O7=VCC;O6=VCC;O5=VCC;O4=VCC;
                O3=VCC;O2=VCC;O1=VCC;O0=VCC;                            --defaults all HIGH out
```

```
        END DEFAULTS;
        CASE a[] IS
                WHEN 0      =>      O0 = GND;
                WHEN 1      =>      O1 = GND;
                WHEN 2      =>      O2 = GND;
                WHEN 3      =>      O3 = GND;
                WHEN 4      =>      O4 = GND;
                WHEN 5      =>      O5 = GND;
                WHEN 6      =>      O6 = GND;
                WHEN 7      =>      O7 = GND;
                WHEN 8      =>      O8 = GND;
                WHEN 9      =>      O9 = GND;
        END CASE;
END;


-- BCD to decimal decoder similar to a 7447

ENTITY prob9_65 IS
PORT    (
                a                               :IN BIT_VECTOR (3 DOWNTO 0);
                O                               :OUT BIT_VECTOR (9 DOWNTO 0)
        );
END prob9_65 ;

ARCHITECTURE truth OF prob9_65  IS
        BEGIN
                WITH a SELECT
                        O       <=      "1111111110"  WHEN "0000",    --O0 active
                                        "1111111101"  WHEN "0001",    --O1 active
                                        "1111111011"  WHEN "0010",    --O2 active
                                        "1111110111"  WHEN "0011",    --O3 active
                                        "1111101111"  WHEN "0100",    --O4 active
                                        "1111011111"  WHEN "0101",    --O5 active
                                        "1110111111"  WHEN "0110",    --O6 active
                                        "1101111111"  WHEN "0111",    --O7 active
                                        "1011111111"  WHEN "1000",    --O8 active
                                        "0111111111"  WHEN "1001",    --O9 active
                                        "1111111111"  WHEN OTHERS;  --disabled
        END truth;
```

**9.66**

```
-- HEX decoder driver for a 7-seg display

SUBDESIGN  prob9_66
(
        hex[3..0]                       :INPUT;         --4-bit number
        lt, bi, rbi                     :INPUT;         --3 independent controls
        a,b,c,d,e,f,g,rbo               :OUTPUT;        --individual outputs
)
BEGIN
        IF !bi THEN
                (a,b,c,d,e,f,g,rbo) = (1,1,1,1,1,1,1,0);        % blank all %
        ELSIF   !lt THEN
                (a,b,c,d,e,f,g,rbo) = (0,0,0,0,0,0,0,1);        % test segments %
        ELSIF !rbi & hex[] == 0   THEN
```

```
        (a,b,c,d,e,f,g,rbo) = (1,1,1,1,1,1,1,0);        % blank leading 0's %
    ELSE
        TABLE                                           % display 7 segment Common Anode pattern %
        hex[]   =>      a,b,c,d,e,f,g,rbo;
        0               =>      0,0,0,0,0,0,1,1;
        1               =>      1,0,0,1,1,1,1,1;
        2               =>      0,0,1,0,0,1,0,1;
        3               =>      0,0,0,0,1,1,0,1;
        4               =>      1,0,0,1,1,0,0,1;
        5               =>      0,1,0,0,1,0,0,1;
        6               =>      1,1,0,0,0,0,0,1;
        7               =>      0,0,0,1,1,1,1,1;
        8               =>      0,0,0,0,0,0,0,1;
        9               =>      0,0,0,1,1,0,0,1;
        10              =>      0,0,0,1,0,0,0,1;
        11              =>      1,1,0,0,0,0,0,1;
        12              =>      1,1,1,0,0,1,0,1;
        13              =>      1,0,0,0,0,1,0,1;
        14              =>      0,1,1,0,0,0,0,1;
        15              =>      0,1,1,1,0,0,0,1;
        END TABLE;
    END IF;
END;


-- HEX to 7-segment decoder

ENTITY prob9_66 IS
PORT (
        hex                             :IN INTEGER RANGE 0 TO 15;
        lt, bi, rbi                     :IN BIT;
        a,b,c,d,e,f,g,rbo               :OUT BIT;
    );
END prob9_66 ;

ARCHITECTURE vhdl OF prob9_66  IS
BEGIN
PROCESS (hex, lt, bi, rbi)
VARIABLE segments                       :BIT_VECTOR (0 TO 6);
    BEGIN
        IF  bi = '0' THEN
            segments := "1111111";  rbo <= '0';     -- blank all
        ELSIF lt = '0' THEN
            segments := "0000000";  rbo <= '1';     -- test segments
        ELSIF (rbi = '0' AND hex = 0)   THEN
            segments := "1111111";  rbo <= '0';     -- blank leading 0's
        ELSE
            rbo <= '1';
            CASE hex IS                             -- display 7 segment Common Anode pattern
                WHEN 0      =>      segments := "0000001";
                WHEN 1      =>      segments := "1001111";
                WHEN 2      =>      segments := "0010010";
                WHEN 3      =>      segments := "0000110";
                WHEN 4      =>      segments := "1001100";
                WHEN 5      =>      segments := "0100100";
                WHEN 6      =>      segments := "1100000";
```

```
                        WHEN 7      =>      segments := "0001111";
                        WHEN 8      =>      segments := "0000000";
                        WHEN 9      =>      segments := "0001100";
                        WHEN 10     =>      segments := "0001000";
                        WHEN 11     =>      segments := "1100000";
                        WHEN 12     =>      segments := "1110010";
                        WHEN 13     =>      segments := "1000010";
                        WHEN 14     =>      segments := "0110000";
                        WHEN 15     =>      segments := "0111000";
                END CASE;
        END IF;
    a <= segments(0);                                   --assign bits of array to output pins
    b <= segments(1);
    c <= segments(2);
    d <= segments(3);
    e <= segments(4);
    f <= segments(5);
    g <= segments(6);
        END PROCESS;
END vhdl;
```

**9.67**

```
% LOW priority encoder. Encodes lowest order input that is activated. %

SUBDESIGN prob9_67
(
    sw[9..0], oe                    :INPUT;
    d[3..0]                         :OUTPUT;
)
VARIABLE
    buffers[3..0]                   :TRI;
BEGIN
    IF      !sw[0]  THEN  buffers[].in = 0;
    ELSIF   !sw[1]  THEN  buffers[].in = 1;
    ELSIF   !sw[2]  THEN  buffers[].in = 2;
    ELSIF   !sw[3]  THEN  buffers[].in = 3;
    ELSIF   !sw[4]  THEN  buffers[].in = 4;
    ELSIF   !sw[5]  THEN  buffers[].in = 5;
    ELSIF   !sw[6]  THEN  buffers[].in = 6;
    ELSIF   !sw[7]  THEN  buffers[].in = 7;
    ELSIF   !sw[8]  THEN  buffers[].in = 8;
    ELSIF   !sw[9]  THEN  buffers[].in = 9;
    ELSE                  buffers[].in = 0;

    END IF;
    buffers[].oe = oe & sw[]!=b"1111111111";        -- enable when OE AND key pressed
    d[] = buffers[].out;                            -- connect to outputs
END;
```

```
-- LOW Priority encoder

LIBRARY        ieee;
USE ieee.std_logic_1164.ALL;

ENTITY prob9_67 IS
PORT(
                sw      :IN BIT_VECTOR (9 DOWNTO 0);          --standard logic not needed
                oe      :IN BIT;                              --standard logic not needed
                d       :OUT STD_LOGIC_VECTOR (3 DOWNTO 0)    --must use std logic for hi-Z
        );
END prob9_67;

ARCHITECTURE a OF prob9_67 IS
        BEGIN
                d       <=      "ZZZZ"  WHEN ((oe = '0') OR (sw = "1111111111")) ELSE
                                "0000"  WHEN  sw(0) = '0'       ELSE
                                "0001"  WHEN  sw(1) = '0'       ELSE
                                "0010"  WHEN  sw(2) = '0'       ELSE
                                "0011"  WHEN  sw(3) = '0'       ELSE
                                "0100"  WHEN  sw(4) = '0'       ELSE
                                "0101"  WHEN  sw(5) = '0'       ELSE
                                "0110"  WHEN  sw(6) = '0'       ELSE
                                "0111"  WHEN  sw(7) = '0'       ELSE
                                "1000"  WHEN  sw(8) = '0'       ELSE
                                "1001"  WHEN  sw(9) = '0';
END a;
```

**9.68**

```
-- modified Fig9-66 to make an 8-bit comparator

SUBDESIGN prob9_68
(
        a[7..0], b[7..0]                :INPUT;
        gtin, ltin, eqin                :INPUT;         % cascade inputs %
        agtb, altb, aeqb                :OUTPUT;
)
        % standard cascade inputs:      gtin = ltin = GNDeqin = VCC %

BEGIN
        IF    a[] > b[] THEN
                        agtb = VCC;             altb = GND;             aeqb = GND;
        ELSIF  a[] < b[] THEN
                        agtb = GND;             altb = VCC;             aeqb = GND;
        ELSE   agtb = gtin;       altb = ltin;      aeqb = eqin;
        END IF;
END;

-- modified Fig9-66 to make an 8-bit comparator

ENTITY prob9_68 IS
PORT ( a, b                             : IN INTEGER RANGE 0 TO 255;
                gtin, ltin, eqin        : IN BIT;               -- cascade inputs
                agtb, altb, aeqb        : OUT BIT);
END prob9_68;
                                                        -- standard cascade inputs: gtin = ltin = '0' eqin = '1'
```

```
ARCHITECTURE vhdl OF prob9_68 IS
BEGIN
        PROCESS (a, b, gtin, ltin, eqin)
        BEGIN
                IF      a < b  THEN     altb <= '1';    agtb <= '0';    aeqb <= '0';
                ELSIF   a > b  THEN     altb <= '0';    agtb <= '1';    aeqb <= '0';
                ELSE                    altb <= ltin;   agtb <= gtin;   aeqb <= eqin;
                END IF;
        END PROCESS;
END vhdl;
```

**9.69**

```
-- A 4-bit binary to 2-digit BCD converter

SUBDESIGN fig9_69
(               binary[3..0]                    :INPUT;
                ones[3..0], tens[3..0]          :OUTPUT;
)

BEGIN
        IF binary > 9 THEN
                tens[] = B"0001" ;
                ones[] = binary[] - 10;
        ELSE tens[] = B"0000";
                ones = binary[];
END;

--A 4-bit binary to 2-digit BCD converter

ENTITY prob9_69 IS
PORT ( binary                           :IN INTEGER RANGE 0 TO 15;
                ones, tens              :buffer INTEGER RANGE 0 TO 9);
END prob9_69;

ARCHITECTURE vhdl OF prob9_69 IS

BEGIN
        PROCESS (binary)
        BEGIN
                IF binary > 9 THEN
                        tens <= 1;
                        ones <= binary - 10;
                ELSE
                        tens <= 0;
                        ones <= binary  ;
                END IF;
        END PROCESS;
END vhdl;
```

**9.70**

```
-- 3-digit BCD to 8-bit binary code converter handles input values 0 - 255

SUBDESIGN prob9_70
(       hundreds[1..0], tens[3..0], ones[3..0]          :INPUT;
        binary[7..0]                                    :OUTPUT;
)

VARIABLE timesten[7..0]                                 :NODE; % variable for tens digit times 10 %
                timeshund[7..0]                         :NODE; % variable for hundreds digit times 100 %
BEGIN
        timeshund[] = (hundreds[],B"000000") + (B"0",hundreds[],B"00000") + (B"0000",hundreds[],B"00");
                        % shift left 6X (times 64) + shift left 5 X (time 32) + shift left 2X (times4) %
        timesten[] = (B"0",tens[],B"000") + (B"000",tens[],B"0");
                        % shift left 3X (times 8) + shift left 1X (times 2) %
        binary[] = timeshund[] + timesten[] + (B"0000",ones[]);
                        % tens digit times 10 + ones digit %
END;

-- 3-digit BCD to 8-bit binary code converter handles input values 0 - 255

ENTITY prob9_70 IS
PORT ( ones, tens                                       :IN INTEGER RANGE 0 TO 9;
                hundreds                                :IN INTEGER RANGE 0 TO 2;
                binary                                  :OUT INTEGER RANGE 0 TO 255);
END prob9_70 ;

ARCHITECTURE vhdl OF prob9_70  IS
SIGNAL timesten                                         :INTEGER RANGE 0 TO 90;
SIGNAL timeshund                                        :INTEGER RANGE 0 TO 200;
BEGIN
        timeshund       <= hundreds * 100;
        timesten        <= tens * 10;
        binary          <= timeshund + timesten + ones;
END vhdl;
```

## CHAPTER TEN - Digital System Projects Using HDL

**10.1** (a) This project is a security system that monitors the open/closed status of a number of doors in the building. The status of each door must be monitored in a remote security shack. When any door is securely closed, the corresponding LED in the guard's shack should be off. When the door is open, the corresponding LED should blink. Specifications for this system:

- Number of doors: 8
- Number of LED indicators: 8
- Blink rate: 2.5 Hz
- Door sensors: Door open/contacts open, door closed/contacts closed.

(b) Three major blocks:
- MUX
- Timing and control (counter)
- DEMUX

(c) Block interconnections:

| | | |
|---|---|---|
| • MUX | INPUTS: | 8 door sensors (HIGH = OPEN, LOW = CLOSED)<br>3 binary select lines |
| | OUTPUT: | 1 serial data line |
| • Timing | INPUTS: | Clock |
| | OUTPUTS: | 3 binary select lines counting 0-7 |
| • DEMUX | INPUTS: | 3 binary select lines<br>1 serial data line |
| | OUTPUTS: | 8 LED drivers (active LOW) |

(d) 20 Hz
(e) Only one LED will ever be lit at any time.

**10.2** 24 steps

**10.3** 4 states = 4 steps x 15°/step = 60° of rotation.

**10.4** 8 states = 8 steps x 7.5°/step = 60° of rotation.

**10.5** 3 state transitions x 15°/step = 45° of rotation.

**10.6** Connect a de-bounced push button switch to the *step* input, a toggle switch to the *dir* input, two toggle switches to the mode (*m1, m0*) inputs, and four LEDs to the outputs *cout.*

*a. set m1, m0 to [0,0] and dir to 0. Apply pulses to step and compare the LED states to Table 10-1 Full step mode. Repeat with dir=1.*
*b. set m1, m0 to [0,1] and dir to 0. Apply pulses to step and compare the LED states to Table 10-1 Wave drive mode.*
*c. set m1, m0 to [1,01] and dir to 0. Apply pulses to step and compare the LED states to Table 10-1 Half step mode.*

Set m1, m0 to [1,1]. Connect toggle switches to each of the inputs *Cin*[3..0]. Using a logic probe, monitor each *cout* line while toggling the input switches on *Cin*. Each *Cout* line should follow the corresponding *Cin* line. The step and direction lines should have no effect.

**10.7**

```
% Complete stepper motor driver
MODES: 00 - Full step;   01 - Wave drive;  02 - Half step;  03 - direct drive %
SUBDESIGN  prob10_7
(
        step, dir                       :INPUT;
        m[1..0], cin[3..0]              :INPUT;
        cout[3..0], q[2..0]             :OUTPUT;
)
VARIABLE
        count[2..0]                     : DFF;
BEGIN
        count[].clk = step;
        IF dir THEN     count[].d = count[].q + 1;
        ELSE            count[].d = count[].q - 1;
        END IF;
        q[] = count[].q;
        IF m[] == 0 THEN
                TABLE                                           -- FULL STEP
                        count[] =>      cout[];
                        B"000"  =>      B"1010";
                        B"001"  =>      B"1001";
                        B"010"  =>      B"0101";
                        B"011"  =>      B"0110";
                        B"100"  =>      B"1010";
                        B"101"  =>      B"1001";
                        B"110"  =>      B"0101";
                        B"111"  =>      B"0110";
                END TABLE;
        ELSIF m[] == 1 THEN
                TABLE                                           -- WAVE DRIVE
                        count[] =>      cout[];
                        B"000"  =>      B"1000";
                        B"001"  =>      B"0001";
                        B"010"  =>      B"0100";
                        B"011"  =>      B"0010";
                        B"100"  =>      B"1000";
                        B"101"  =>      B"0001";
                        B"110"  =>      B"0100";
                        B"111"  =>      B"0010";
                END TABLE;
        ELSIF m[] == 2 THEN
                TABLE
                        count[] =>      cout[];                 -- HALF STEP
                        B"000"  =>      B"1010";
                        B"001"  =>      B"1000";
                        B"010"  =>      B"1001";
                        B"011"  =>      B"0001";
                        B"100"  =>      B"0101";
                        B"101"  =>      B"0100";
                        B"110"  =>      B"0110";
                        B"111"  =>      B"0010";
                END TABLE;
        ELSE    cout[] = cin[];                                 --DIRECT DRIVE
        END IF;
END;
```

```
-- Universal stepper motor driver
-- MODES: 00 - Full step;  01 - Wave drive;  02 - Half step;  03 -  direct drive
--        Digital Systems 10th ed
--        Tocci  Widmer  Moss

ENTITY prob10_7 IS
PORT ( step, dir                                    :IN BIT;
             m                                      :IN BIT_VECTOR (1 DOWNTO 0);
             cin                                    :IN BIT_VECTOR (3 DOWNTO 0);
             q                                      :OUT INTEGER RANGE 0 TO 7;
             cout                                   :OUT BIT_VECTOR (3 DOWNTO 0) );
END prob10_7;

ARCHITECTURE vhdl OF prob10_7 IS
BEGIN
        PROCESS (step)
        VARIABLE count                              :INTEGER RANGE 0 TO 7;
        BEGIN
                IF (step'EVENT AND step = '1' THEN
                        IF dir = '1' THEN           count  := count + 1;
                        ELSE                        count  := count – 1;
                        END IF;
                END IF;
                q <= count;
                IF m = "00"     THEN                                -- FULL STEP
                        IF      count = 0 THEN cout <= "1010";
                        ELSIF   count = 1 THEN cout <= "1001";
                        ELSIF   count = 2 THEN cout <= "0101";
                        ELSIF   count = 3 THEN cout <= "0110";
                        ELSIF   count = 4 THEN cout <= "1010";
                        ELSIF   count = 5 THEN cout <= "1001";
                        ELSIF   count = 6 THEN cout <= "0101";
                        ELSIF   count = 7 THEN cout <= "0110";
                        END IF;

                ELSIF m = "01"  THEN                                -- WAVE DRIVE
                        IF      count = 0 THEN cout <= "1000";
                        ELSIF   count = 1 THEN cout <= "0001";
                        ELSIF   count = 2 THEN cout <= "0100";
                        ELSIF   count = 3 THEN cout <= "0010";
                        ELSIF   count = 4 THEN cout <= "1000";
                        ELSIF   count = 5 THEN cout <= "0001";
                        ELSIF   count = 6 THEN cout <= "0100";
                        ELSIF   count = 7 THEN cout <= "0010";
                        END IF;

                ELSIF m = "10"  THEN                                -- HALF STEP
                        IF      count = 0 THEN cout <= "1010";
                        ELSIF   count = 1 THEN cout <= "1000";
                        ELSIF   count = 2 THEN cout <= "1001";
                        ELSIF   count = 3 THEN cout <= "0001";
                        ELSIF   count = 4 THEN cout <= "0101";
                        ELSIF   count = 5 THEN cout <= "0100";
                        ELSIF   count = 6 THEN cout <= "0110";
                        ELSIF   count = 7 THEN cout <= "0010";
                        END IF;
```

```
                    ELSE    cout <= cin                         -- DIRECT DRIVE
                    END IF;
            END PROCESS
END vhdl;
-- original design by TW Schultz
-- modified by NS Widmer
```

**10.8**

```
% Complete stepper motor driver
MODES: 00 - Full step;   01 - Wave drive;  02 - Half step;  03 - direct drive %
SUBDESIGN  prob10_8
(
        step, dir, oe                       :INPUT;
        m[1..0], cin[3..0]                  :INPUT;
        cout[3..0], q[2..0]                 :OUTPUT;
)
VARIABLE
        buffers[3..0]                       : TRI;
        count[2..0]                         : DFF;
BEGIN
        count[].clk = step;
        IF dir THEN     count[].d = count[].q + 1;
        ELSE            count[].d = count[].q - 1;
        END IF;
        q[] = count[].q;
        CASE m[] IS
        WHEN 0 =>
                CASE count[] IS                                 -- FULL STEP
                WHEN  B"000"  =>    buffers[].in = B"1010";
                WHEN  B"001"  =>    buffers[].in = B"1001";
                WHEN  B"010"  =>    buffers[].in = B"0101";
                WHEN  B"011"  =>    buffers[].in = B"0110";
                WHEN  B"100"  =>    buffers[].in = B"1010";
                WHEN  B"101"  =>    buffers[].in = B"1001";
                WHEN  B"110"  =>    buffers[].in = B"0101";
                WHEN  B"111"  =>    buffers[].in = B"0110";
                END CASE;
        WHEN 1 =>
                CASE count[] IS                                 -- WAVE DRIVE
                WHEN  B"000"  =>    buffers[].in = B"1000";
                WHEN  B"001"  =>    buffers[].in = B"0001";
                WHEN  B"010"  =>    buffers[].in = B"0100";
                WHEN  B"011"  =>    buffers[].in = B"0010";
                WHEN  B"100"  =>    buffers[].in = B"1000";
                WHEN  B"101"  =>    buffers[].in = B"0001";
                WHEN  B"110"  =>    buffers[].in = B"0100";
                WHEN  B"111"  =>    buffers[].in = B"0010";
                END CASE;
        WHEN 2 =>
                CASE count[] IS                                 -- HALF STEP
                WHEN  B"000"  =>    buffers[].in = B"1010";
                WHEN  B"001"  =>    buffers[].in = B"1000";
                WHEN  B"010"  =>    buffers[].in = B"1001";
                WHEN  B"011"  =>    buffers[].in = B"0001";
                WHEN  B"100"  =>    buffers[].in = B"0101";
                WHEN  B"101"  =>    buffers[].in = B"0100";
                WHEN  B"110"  =>    buffers[].in = B"0110";
```

```
                    WHEN  B"111"  =>      buffers[].in = B"0010";
                    END CASE;
        WHEN 3 =>   buffers[].in = cin[];                          --DIRECT DRIVE
        END CASE;
        cout[]= buffers[].out ;
        buffers[].oe = oe;
END;

-- Universal stepper motor driver
-- MODES: 00 - Full step;  01 - Wave drive;  02 - Half step;  03 -  direct drive
--       Digital Systems 10th ed
--       Tocci  Widmer  Moss
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY prob10_8 IS
PORT ( step, dir,  oe                  :IN BIT;
            m                          :IN BIT_VECTOR (1 DOWNTO 0);
            cin                        :IN STD_LOGIC_VECTOR (3 DOWNTO 0);
            q                          :OUT INTEGER RANGE 0 TO 7;
            cout                       :OUT STD_LOGIC_VECTOR (3 DOWNTO 0) );
END prob10_8 ;

ARCHITECTURE vhdl OF prob10_8 IS
BEGIN
        PROCESS (step)
        VARIABLE count                 :INTEGER RANGE 0 TO 7;
        BEGIN
            IF (step'EVENT AND step = '1' THEN
                IF dir = '1' THEN      count  := count + 1;
                ELSE                   count  := count – 1;
                END IF;
            END IF;
            q <= count;
            IF oe = '1'       THEN
               CASE m IS
                  WHEN "00" =>                          -- FULL STEP
                        CASE count IS
                              WHEN 0 =>    cout <= "1010";
                              WHEN 1 =>    cout <= "1001";
                              WHEN 2 =>    cout <= "0101";
                              WHEN 3 =>    cout <= "0110";
                              WHEN 4 =>    cout <= "1010";
                              WHEN 5 =>    cout <= "1001";
                              WHEN 6 =>    cout <= "0101";
                              WHEN 7 =>    cout <= "0110";
                        END CASE;

                  WHEN "01" =>                          -- WAVE DRIVE
                        CASE count IS
                              WHEN 0 =>    cout <= "1000";
                              WHEN 1 =>    cout <= "0001";
                              WHEN 2 =>    cout <= "0100";
                              WHEN 3 =>    cout <= "0010";
                              WHEN 4 =>    cout <= "1000";
                              WHEN 5 =>    cout <= "0001";
```

```
                                    WHEN 6 =>       cout <= "0100";
                                    WHEN 7 =>       cout <= "0010";
                                END CASE;

                            WHEN "10" =>                            -- HALF STEP
                                CASE count IS
                                    WHEN 0 =>       cout <= "1010";
                                    WHEN 1 =>       cout <= "1000";
                                    WHEN 2 =>       cout <= "1001";
                                    WHEN 3 =>       cout <= "0001";
                                    WHEN 4 =>       cout <= "0101";
                                    WHEN 5 =>       cout <= "0100";
                                    WHEN 6 =>       cout <= "0110";
                                    WHEN 7 =>       cout <= "0010";
                                END CASE;

                            WHEN "11" =>    cout <= cin             -- DIRECT DRIVE
                        END CASE;
                    ELSE cout <= "ZZZZ";
                    END IF;
            END PROCESS;
    END vhdl;
    -- original design by TW Schultz
    -- modified by NS Widmer
```

**10.9**

| R3 | R2 | R1 | R0 |
|---|---|---|---|
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

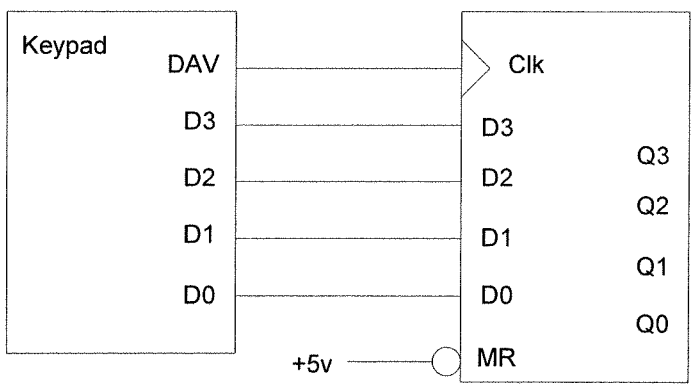
**10.10** 1111

**10.11** Yes

**10.12** (a) 1011 (b) $10_2$ (row 2) (c) $01_2$ (row 1) (d) 1001

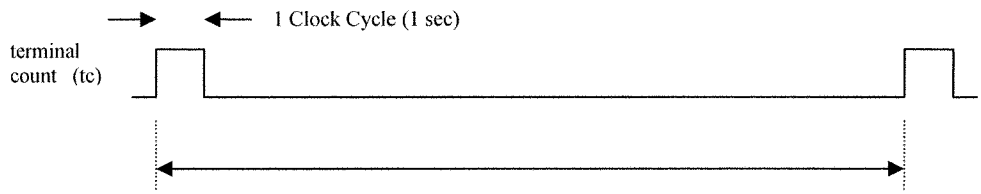
**10.13** No

**10.14** DAV



**10.15** The data goes away (high-Z) before the DAV goes LOW. The high-Z is latched.

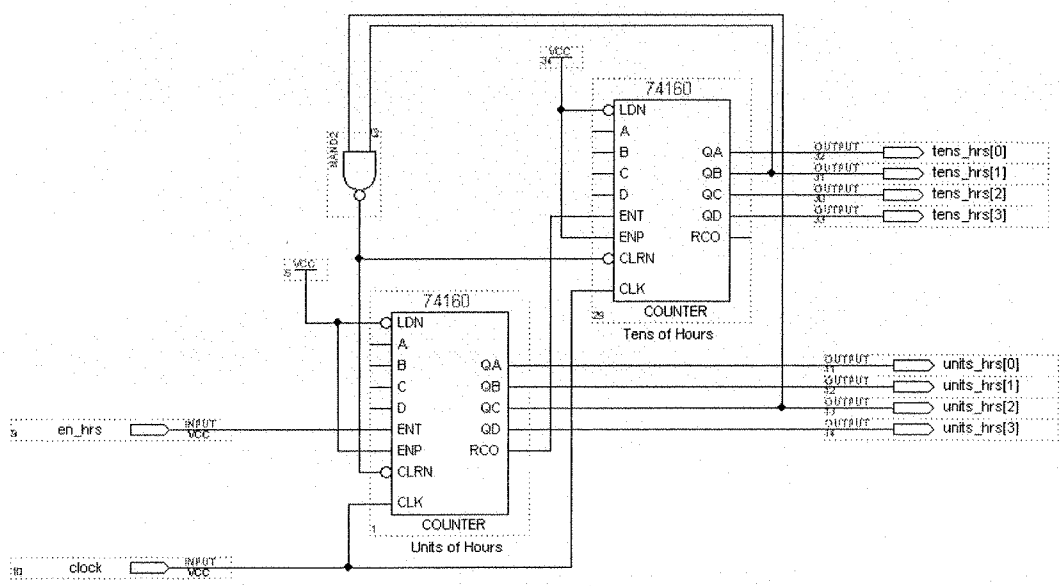
**10.16** (a) 60 clock cycles (b) 600 clock cycles (c) 3600 clock cycles



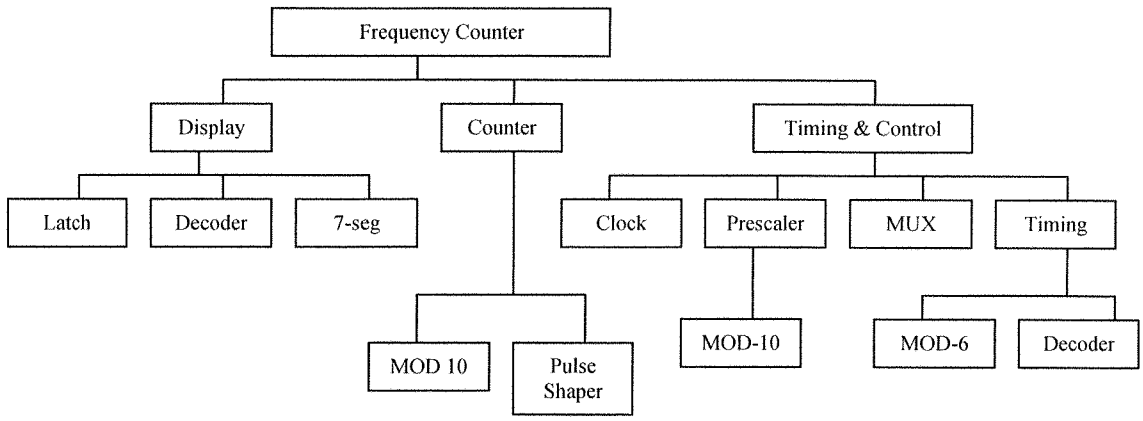

**10.17** 60 cycles/sec x 60 sec/min x 60 min/hr x 24 hr/day = 5,184,000 cycles/day. This will take a long time to generate a simulation file.

**10.18** When the set input is active, bypass the prescaler and feed the 60 Hz clock directly into the units of seconds counter.

**10.19**



**10.20**




**10.21**

```
% MOD 6 used in FREQ COUNTER project
counts 0-5 decodes three states: clear = 0, enable = 2, store = 4 %

SUBDESIGN PROB10_21
(
        clock                           :INPUT;             -- synch clock
        q[2..0]                         :OUTPUT;            -- 3-bit counter
        clear, enable, store:OUTPUT;                        -- timing signals
)
VARIABLE
        count[2..0]     :DFF;                               -- declare a register of D flip flops.

BEGIN
        count[].clk = clock;                                -- connect all clocks to synchronous source
        IF count[].q < 5 THEN
                count[].d = count[].q + 1;                  -- increment current value by one
        ELSE count[].d = 0;                                 -- force unused states to 0
        END IF;
        q[] = count[].q;                                    -- connect register to outputs

        CASE count[] IS
                WHEN 0          =>      clear = VCC; enable = GND; store = GND;
                WHEN 2          =>      clear = GND; enable = VCC; store = GND;
                WHEN 4          =>      clear = GND; enable = GND; store = VCC;
                WHEN OTHERS     =>      clear = GND; enable = GND; store = GND;
        END CASE;
END;


--      MOD 6 for FREQ COUNTER PROBLEM
--      counts 0-5 decodes three states: clear = 0, enable = 2, store = 4


ENTITY prob10_21 IS
PORT(  clock                            :IN BIT ;
                q                       :OUT INTEGER RANGE 0 TO 5;
                clear, enable, store    :OUT BIT        );
END prob10_21;
```

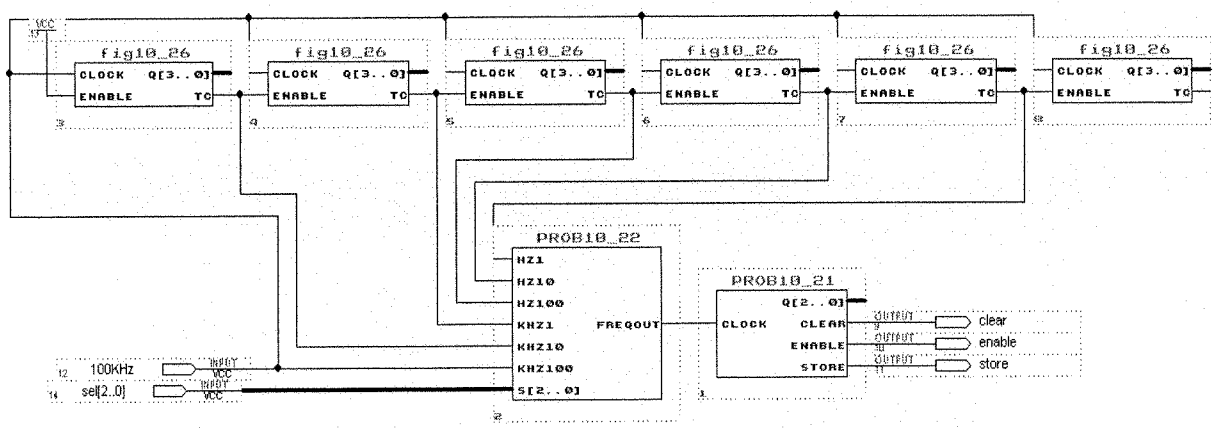
```
ARCHITECTURE a OF prob10_21 IS
BEGIN
        PROCESS ( clock)                                        -- respond to clock
        VARIABLE count              :INTEGER RANGE 0 TO 5;

        BEGIN
                IF (clock = '1' AND clock'event) THEN
                        IF count < 5 THEN                       -- maximum (terminal) count
                                count := count + 1;
                        ELSE
                                count := 0;
                        END IF;
                END IF;

                q <= count;                                     -- update outputs
                CASE count IS
                        WHEN 0                  =>      clear <= '1'; enable <= '0'; store <= '0';
                        WHEN 2                  =>      clear <= '0'; enable <= '1'; store <= '0';
                        WHEN 4                  =>      clear <= '0'; enable <= '0'; store <= '1';
                        WHEN OTHERS             =>      clear <= '0'; enable <= '0'; store <= '0';
                END CASE;
        END PROCESS;
END a;
```

**10.22**

```
% MUX for Freq Counter %

SUBDESIGN prob10_22
(
        1Hz, 10Hz, 100Hz, 1KHz, 10KHz, 100KHz           :INPUT;
        s[2..0]                                         :INPUT;-- select inputs
        freqout                                         :OUTPUT;
)
BEGIN
        CASE s[] IS
                        WHEN 0 =>       freqout = 1Hz;
                        WHEN 1 =>       freqout = 10Hz;
                        WHEN 2 =>       freqout = 100Hz;
                        WHEN 3 =>       freqout = 1KHz;
                        WHEN 4 =>       freqout = 10KHz;
                        WHEN 5 =>       freqout = 100KHz;
        END CASE;
END;


--      MUX for Freq Counter

ENTITY prob10_22 IS
PORT(
                Hz1, Hz10, Hz100, KHz1, KHz10, KHz100   :IN BIT;
                s                                       :IN INTEGER RANGE 0 TO 5;
                freqout                                 :OUT BIT        );
END prob10_22;
```

```
ARCHITECTURE truth OF prob10_22 IS
        BEGIN
                WITH s SELECT
                        freqout  <=     Hz1     WHEN 0,         --switch 1 to output y
                                        Hz10    WHEN 1,         --switch 10 to output y
                                        Hz100   WHEN 2,         --switch 100 to output y
                                        KHz1    WHEN 3,         --switch 1k to output y
                                        KHz10   WHEN 4,         --switch 10k to output y
                                        KHz100  WHEN 5;         --switch 100k to output y
        END truth;
```

**10.23**




**10.24**

```
--      prescaler made from 5 MOD10  modules
--      freq cntr project in Chapter 10
--      used in problem 10-24

include "fig10_26.inc";                                 -- mod-10 counter module

SUBDESIGN prescaler     (
                        clk                     : INPUT;
                        freqs[5..0]             : OUTPUT;
                                )
VARIABLE
        KHZ10                   : fig10_26;             -- mod-10
        KHZ1                    : fig10_26;
        HZ100                   : fig10_26;
        HZ10                    : fig10_26;
        HZ1                     : fig10_26;

BEGIN
        KHZ10.clock = clk;                              -- synchronous clocking
        KHZ10.enable = VCC;
        KHZ1.clock = clk;
        KHZ1.enable = KHZ10.tc;
        Hz100.clock = clk;
        HZ100.enable = KHZ1.tc;
```

```
        Hz10.clock = clk;
        HZ10.enable = HZ100.tc;
        Hz1.clock = clk;
        HZ1.enable = HZ10.tc;
        freqs[] = (clk, KHZ10.tc, KHZ1.tc, HZ100.tc, HZ10.tc, HZ1.tc);

END;

--      Timing and Control section of freq cntr project in Chapter 10

include "prescaler.inc";
include "prob10_22.inc";
include "prob10_21.inc";
SUBDESIGN T_and_C    (
                                clk, freq_range[2..0]       : INPUT;
                                clear, enable, store        : OUTPUT;
                                        )
VARIABLE
        presc                           : prescaler;            -- frequency prescaler
        fmux                            : prob10_22;            -- multiplexer selects freq
        control                         : prob10_21;            -- control signal generator
BEGIN
        presc.clk = clk;
        (fmux.KHZ100, fmux.KHZ10, fmux.KHZ1, fmux.HZ100, fmux.HZ10, fmux.HZ1) = presc.freqs[];
        control.clock = fmux.freqout;
        clear = control.clear;
        enable = control.enable;
        store = control.store;
        fmux.s[] = freq_range[];
END;
```

## CHAPTER ELEVEN - Interfacing With the Analog World

**11.1** (a) Analog output = (K) x (digital input)
(b) Smallest change that can occur in the analog output as a result of a change in the digital input.
(c) Same as (b).
(d) Maximum possible output value of a DAC.
(e) Ratio of the step size to the full-scale value of a DAC. Percentage resolution can also be defined as the reciprocal of the maximum number of steps of a DAC.
(f) False.
(g) False (It is the same).

**11.2** $01100100_2 = 100_{10}$
$10110011_2 = 179_{10}$
(179/100) = (X/2V)
X = 3.58V

**11.3** LSB = 2V/100 = 20mV
Other bits: 40mV, 80mV, 160mV, 320mV, 640mV, 1280mV, and 2560mV.

**11.4** Resolution = Weight of LSB = 20mV; % Resolution = $[1/(2^8-1)] \times 100\% \approx 0.4\%$

**11.5** 10 bits---> $2^{10}-1 = 1023$ steps; Resolution = 5V/1023 ≈5mV

**11.6** Assume resolution = 40μA. The number of steps required to produce 10mA F.S. = 10mA/40μA = 250. Therefore, it requires 8 bits.

**11.7** Number of steps = 7; % Resolution = 1/7 = 14.3%; Step-size = 2V/7 = 0.286V

**11.8** The glitches are caused by the temporary states of the counter as FFs change in response to clock.

**11.9** 12-bit DAC gives us $2^{12}-1$ steps = 4095. Step-Size = F.S/# of steps = 2mA/4095 = 488.4nA
To have exactly 250 RPM the output of the DAC must be 500μA. ((250 x 2mA)/1000RPM)

In order to have 500μA at the output of the DAC, the computer must increment the input of the DAC to the count of 1023.75. (500μA/488.4nA)

Thus, the motor will rotate at 250.061 RPM when the computer's output has incremented 1024 steps.

**11.10** Step Size (resolution) = $V_{FS} / (2^{12} - 1) = 3.66$ mV
% Resolution = step size / full scale x 100% = 3.66 mV / 15.0 V x 100% = 0.024%
$011010010101_2 = 1685_{10}$
Vout = 1685 x 15 / 4095 = 6.17 V

**11.11** The most significant 8 bits: DAC[9..2] => PORT[7..0].
Full scale is still 10 volts and step size is 39 mV.

**11.12** Number of steps = 12 V / 20mV = 600
$2^n - 1 > 600$, Thus, n = 10 bits.

**11.13** (a) Step-Size = $R_F$ x (5V/8KΩ) = 0.5V. Therefore, $R_F$ = 800Ω
(b) No. Percentage resolution is independent of $R_F$.

**11.14** (a) $I_O = V_{REF}/R$ = 250μA
LSB = $I_O/8$ = 31.25μA
$V_{OUT(LSB)}$ = -31.25μA x 10KΩ = -0.3125V
$V_{OUT(Full\ Scale)}$ = -10KΩ(31.25+62.5+125+250)μA = -4.6875V

(b) (-2V/-4.6875V) = $R_F$/10KΩ
$R_F$ = 4.27KΩ

(c) $V_{OUT} = K(V_{REF} \times B)$
-2V = K(5V x 15)
K = -0.0267

**11.15** With the current IC fabrication technology, it is very difficult to produce resistance values over a wide resistance range. Thus, this would be the disadvantage of the circuit of figure 11.8, especially if it was to have a large number of inputs.

**11.16** (a) Absolute error = 0.2% x 10mA = 20μA
(b) Step-Size = (F.S./# of steps) = 10mA/255 = 39.2μA. Ideal output for $00000001_2$ is 39.2μA. The possible range is 39.2μA ± 20μA = 19.2μA-59.2μA. Thus, 50μA is within this range.

**11.17** (a) 0.1 inches out of a total of 10 inches is a percentage resolution of 1%.
Thus, $(1/2^n-1) \times 100\% < 1\%$. The smallest integer value of n which satisfies this criteria is n=7.
(b) The potentiometer will not give a smoothly changing value of $V_P$ but will change in small jumps due to the granularity of the material used as the resistance.

**11.18** (a) Resistor network used in simple DAC using a an op-amp summing amplifier. Starting with the MSB resistor, the resistor values increase by a factor of 2.
(b) Type of DAC where its internal resistance values only span a range of 2 to 1.
(c) Amount of time that it takes the output of a DAC to go from zero to within 1/2 step size of its full-scale value as the input is changed from all 0s to all 1s.
(d) Term used by some DAC manufacturers to specify the accuracy of a DAC. It's defined as the maximum deviation of a DAC's output from its expected ideal value.
(e) Under ideal conditions the output of a DAC should be zero volts when the input is all 0s. In reality, there is a very small output voltage for this situation. This deviation from the ideal zero volts is called the offset error.

**11.19** Step-Size = 1.26V/63 = 20mV; ±0.1% F.S. = ±1.26mV = ±1mV
Thus, maximum error will be ±2.26 mV.

$000010_2$ → 2 x 20mV = 40mV [41.5mV is within specs.]
$000111_2$ → 7 x 20mV = 140mV [140.2mV is within specs.]
$001100_2$ → 12 x 20mV = 240mV [242.5mV isn't within specs.]
$111111_2$ → 63 x 20mV = 1.26V [1.258 V is within specs.]

**11.20** The actual offset voltage is greater than 2mV. In fact, it appears to be around 8mV.

**11.21** The DAC's binary input next to the LSB (00000000) is always HIGH. It is probably open.

**11.22** The graph of Figure 11.32 would've resulted, if the two least significant inputs of the DAC were reversed ($0000000\underline{0}_2$). Thus, the staircase would've incremented in the following sequence: 0,2,1,3,4,6,5,7,8,10,9,11,12,14,13,15.

**11.23** A START pulse is applied to *reset* the counter and to keep *pulses* from passing through the AND gate into the *counter.* At this point, the DAC output, $V_{AX}$, is *zero* and $\overline{EOC}$ is *high.* When START returns *low*, the AND gate is *enabled*, and the counter is allowed to *count.* The $V_{AX}$ signal is increased one *step* at a time until it *exceeds* $V_A$. At that point, $\overline{EOC}$ goes LOW to *prevent* further pulses from *being counted.* This signals the end of conversion, and the digital equivalent of $V_A$ is present at the *counter output.*

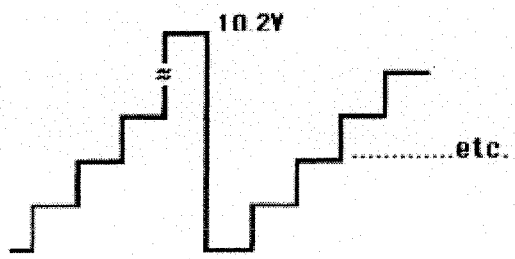
**11.24** (a) (Digital value) x (resolution) $\geq V_A+V_T$; (Digital value) x (40mV) $\geq$ 6.001V = 6001mV. Therefore, Digital value $\geq$ 150.025. This indicates a digital value of 151 or written in binary $10010111_2$.

(b) Using same method as in (a) the digital value is again $10010111_2$.

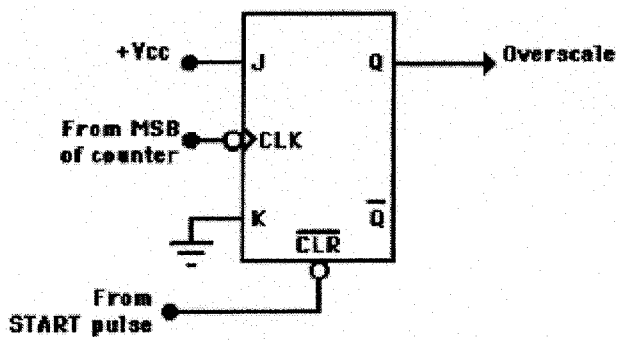
(c) Maximum conversion time =(max. # of steps)x($T_{CLOCK}$); $T_{CLOCK} = (2^8-1) \times (0.4\mu s) = 102\mu s$. Average conversion time = 102μs/2 = 51μs.

**11.25** Because the difference in the two values of $V_A$ was smaller than the resolution of the converter.

**11.26** The A/D converter has a full-scale value of $(2^8-1) \times 40mV=10.2V$. Thus, a $V_A$ of 10.853V would mean that the comparator output would never switch LOW. The counter would keep counting indefinitely producing the waveform below at the D/A output.




The circuit below can be used to indicate an over-scale condition.

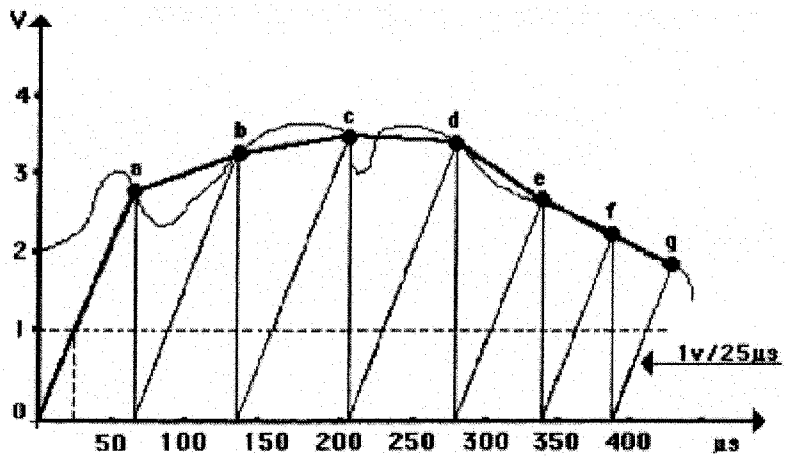


**11.27** (a) With 12 bits, percentage resolution is $(1/(2^{12}-1)) \times 100\% = 0.024\%$.
Thus, quantization error = 0.024% x 5V = 1.2mV.
(b) Error due to .03% inaccuracy = .03% x 5V = 1.5mV.
Total Error = 1.2mV + 1.5mV = 2.7mV.

**11.28** (a) With $V_A$ = 5.022V, the value of $V_{AY}$ must equal or exceed 5.023V to switch COMP. Thus, $V_{AX}$ must equal or exceed 5.018V. This requires 5.018V/10mV = 501.8 = 502 steps. This gives $V_{AX}$ = 5.02V and digital value $0111110110_2$.
(b) $V_{AY} \geq 5.029V$, $V_{AX} \geq 5.024V$; # of steps = 5.024V/10mV = 502.4 = 503 steps ($V_{AX}$ = 5.03V). This gives digital value $0111110111_2$.
(c) In (a) quantization error is $V_{AX} - V_A$ = 5.02V - 5.022V = -2mV. In (b) $V_{AX} - V_A$ = 5.03V - 5.028V = +2mV

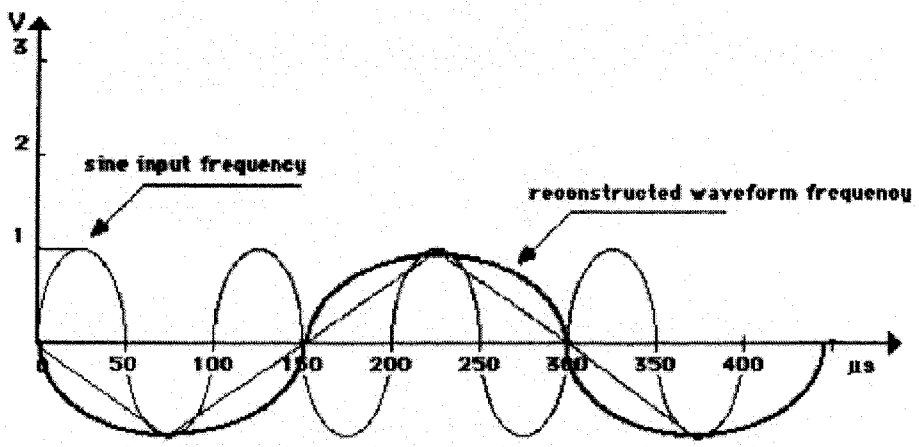
**11.29** $0100011100_2 = 284_{10}$; At count of $284_{10}$, $V_{AY}$ = 2.84V + 5mV = 2.845V; At count of $283_{10}$, $V_{AY}$ = 2.83V + 5mV = 2.835V. Thus, the range of $V_A$ = 2.8341V ---> 2.844V

**11.30**




For a more accurate reproduction of the signal, we must have an A/D converter with much shorter conversion times. An increase in the number of bits of the converter will also help, especially during those times when the original waveform changes rapidly.

**11.31**

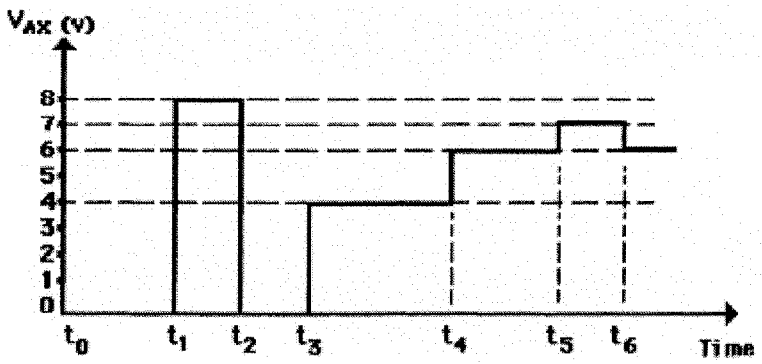


(a) Since the Flash ADC samples at intervals of 75μs, the sample frequency is 1/75μs =13.33 kHz.
(b) The sine wave has a period of 100 μs or a F=10 kHz. Therefore, the difference between the sample frequency and the input sine wave frequency is 3.3 kHz.
(c) The frequency of the reconstructed waveform is approximately 1/300 μs or 3.33 kHz.

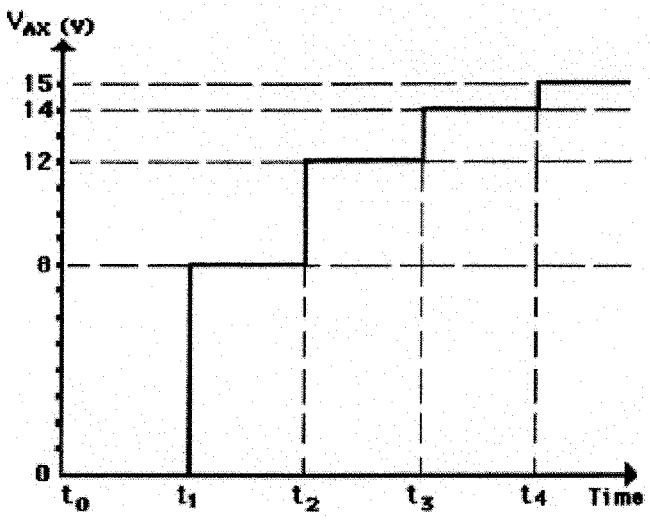
**11.32** (a) Input signal = 5 kHz; (b) Input signal = 9.9 kHz; (c) Input signal = 9.8 kHz
(d) Input signal = 5 kHz; (e) Input signal = 900 Hz; (f) Input signal = 800 Hz

**11.33** (a) digital-ramp ADC; (b) successive approximation ADC; (c) successive approximation ADC
(d) both; (e) both; (f) digital-ramp ADC; (g) successive approximation ADC; (h) both

**11.34**




**11.35**




**11.36** 80μs: Conversion time is independent of $V_A$.

**11.37** $t_0$: Set MSB (bit 5); $t_1$: Set bit 4; clear bit 4; $t_2$: Set bit 3; clear bit 3; $t_3$: Set bit 2
$t_4$: Set bit 1; clear bit 1; $t_5$: Set LSB; Digital result = $100101_2$

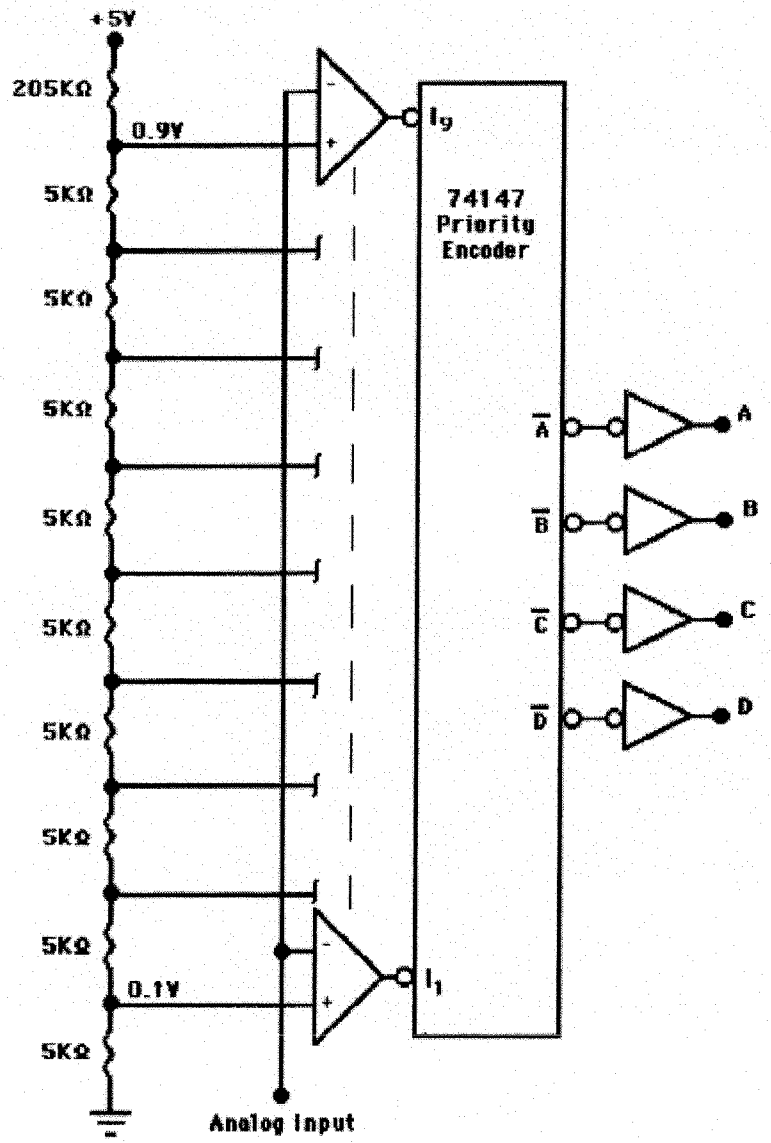
**11.38** The range is 3.0V ; The offset is 0.5V.; The Resolution = 3V/255 = 11.76mV : $10010111_2=151_{10}$ Thus, the value of the analog input is approximately ($151_{10}$ x 11.76mV) + 0.5V = **2.276V**

**11.39** With $V_{REF}/2$ = 2.0V, the range is = 4V ; The offset is 0.5V.
The Resolution = 4V/255 = 15.69mV : $10010111_2=151_{10}$. Thus, the value of the analog input is approximately ($151_{10}$ x 15.69mV) + 0.5V = **2.869V**

**11.40** (a) Since we must measure accurately from 50°F to 101°F, the digital value for 50°F for the best resolution should be $00000000_2$.
(b) The voltage applied to the input $V_{in}(-)$ should be 500mV. With $V_{in}(-)$ = 500mV, when the temperature is 50°F the ADC output will be $00000000_2$.
(c) The full range of voltage that will come in is: (101°F x 0.01V) - (50°F x 0.01V) = 510mV.
(d) A voltage of **255mV** (full range/2) should be applied to $V_{ref}/2$ input.
(e) An input temperature of 72°F causes the LM34 sensor to output a voltage of (72°F x 0.01V) = 720mV. However, since there is an offset voltage of 500mV, the ADC will convert (720mV-500mV) = 220mV. Since one step of the ADC corresponds to an input change of ($2^8$-1)/510mV = 2mV, the ADC output binary value will be $01101110_2 = 110_{10}$.
(f) The sensor will change by 10mV for every input change of 1°F. Therefore, a change of one step of the ADC (2mV) corresponds to an input temperature change of 0.2°F. Thus, the resolution is 0.2°F/step. From step (e), the resolution in volts is 2mV/step.

**11.41** Since a conversion would take place every 1µs rather than the 1V/25µs rate of conversion, the result would've been a much closer reproduction of the analog signal.

**11.42**

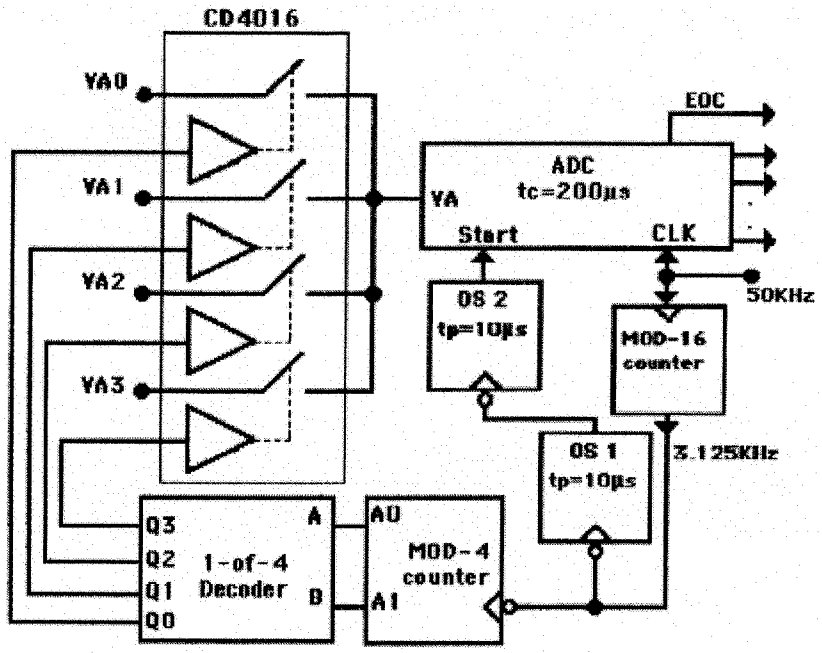



**11.43** (a) flash. (b) digital-ramp and SAC; (c) flash. (d) flash; (e) digital-ramp. (f) digital-ramp, SAC, and flash; (g) SAC and flash.

**11.44** (a) up/down digital-ramp ADC (tracking ADC); (b) flash ADC; (c) voltage-to-frequency ADC (d) voltage-to-frequency ADC; (e) dual-slope ADC. (f) dual-slope ADC.

**11.45** If the switch is stuck closed, the output will follow $V_A$. If the switch is stuck open, or if $C_h$ is shorted, the output will be 0V.

**11.46**




A MOD-16 counter is used between the 50KHz clock and the clock input of the MOD-4 counter because a 320µs time delay is needed for the proper operation of the circuit. The 320µs was determined according to the following requirements:

(a) 200µs for the time conversion (10-bits x clock period).
(b) The outputs must remain stable for 100µs after the conversion is complete.
(c) A 10µs delay (OS1) is needed in order to allow the analog signal VA to stabilize before the ADC is given a Start pulse
(d) Finally, a 10µs-duration Start pulse is required (OS2).

**11.47** (a) The $\overline{CS}$ signal is LOW only when ALE=0 and the following address is on the address bus:

A15 A14 A13 A12 A11 A10 A9 A8 A7-->A0

1 1 1 0 1 0 1 0 x--->x = $EAXX_{16}$

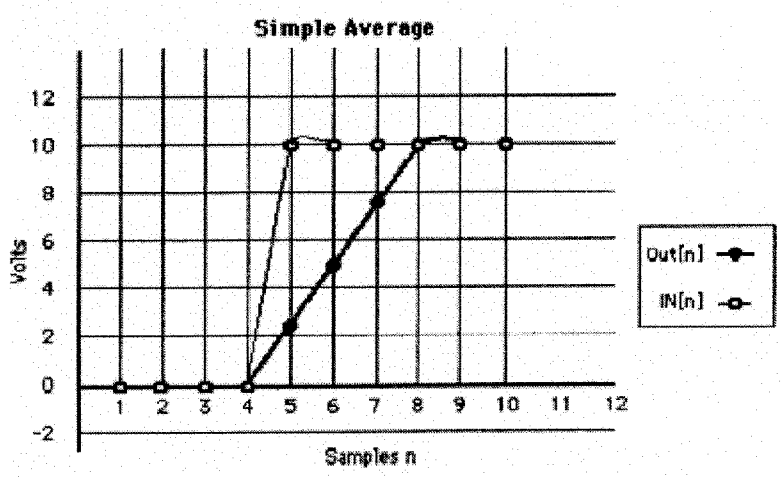
(b) Add an inverter between address line A9 and input A1 of the 74LS138.

(c) 1. Remove the inverter between address line A12 and the NAND gate.
2. Change $\overline{CS}$ from output 2 of the 74LS138 to output 7.

**11.48** Yes. Connect the two least significant bits (b0 and b1) to ground. Attach b2 through b9 from the ADC to the port.

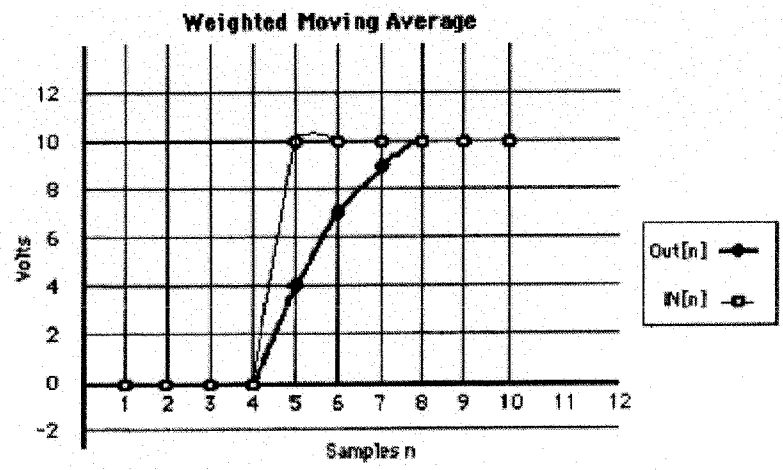
**11.49**

| *Sample* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* |
|---|---|---|---|---|---|---|---|---|---|---|
| IN[n] (v) | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 |
| OUT[n] (v) | 0 | 0 | 0 | 0 | 2.5 | 5 | 7.5 | 10 | 10 | 10 |




**11.50**

| *Sample* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* |
|---|---|---|---|---|---|---|---|---|---|---|
| IN[n] (v) | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 |
| OUT[n] (v) | 0 | 0 | 0 | 0 | 4 | 7 | 9 | 10 | 10 | 10 |




**11.51** Multiply Accumulate

**11.52** (a) F (b) T (c) T (d) T (e) F (f) T (g) F (h) T

## CHAPTER TWELVE - Memory Devices

**12.1** 6x1,024 = 16,384 words; 32 bits/word; 16,384x32 = 524,288 cells

**12.2** 16,384 addresses; one per word.

**12.3** $2^{16}$= 65,536 words = 64K. Thus, memory capacity is 64Kx4.

**12.4** Data input lines = 16; Data output lines = 16; Address lines = 13 ($2^N$ = 8192)
Capacity in bytes = 16,384 ((8192x16)/8)

**12.5** (a) Random Access Memory (RAM) - Memory in which the access time is the same for any location.
(b) Read/Write Memory (RWM) - Any memory that can be read from and written into with equal ease.
(c) Read-Only Memory (ROM) - Memory devices that are designed for applications where the ratio of read operations to write operations is very high.
(d) Internal Memory - This is also referred to as the computer's main memory. It stores the instructions and data that the CPU is currently working on.
(e) Auxiliary Memory - This type of memory is also referred to as mass storage. It stores large amounts of data without the need for electrical power.
(f) Capacity - A way of specifying how many bits can be stored in a particular memory device or complete memory system.
(g) Volatile - Any type of memory that requires the application of electrical power in order to store information.
(h) Density - Another term for Capacity.
(i) Read - The operation whereby the binary word stored in a specific memory location is sensed and then transferred to another device.
(j) Write - The operation whereby a new word is placed into a particular memory location.

**12.6** (a) Address bus, Data bus, and Control bus; (b) Address bus; (c) Data bus. (d) CPU.

**12.7** (a) CS=1 produces Hi-Z state outputs; (b) Data out = 11101101

**12.8** (a) Only register 11 will have both enable inputs activated; (b) Input address code 0100 will activate both enable inputs of register 4.

**12.9** (a) 16K = 16,384. (b) There are 4 bits per register; (c) 16,384 = $2^{14}$ = $2^7$x$2^7$ = 128x128. Thus, two 1-of-128 decoders are required.

**12.10** (a) True; (b)Process of entering data into the ROM; (c) The delay between the application of a ROM's inputs and the appearance of the data outputs during a READ operation.
(d) Data Inputs = 4; Data Outputs = 4; Address Inputs = 10; (e) Its function is to activate one row-select line and one column-select line.

**12.11** Since the address inputs to the ROM are stable 500ns prior to the TRANSFER pulse, then our only concern is to accommodate the $t_{OE}$ delay of 120ns. Thus, the PGT of TRANSFER should not occur for at least 120ns after its NGT. This neglects the set-up time requirement of the 74ALS273.

**12.12** Since the address inputs will have changed only 70ns prior to the NGT of the TRANSFER pulse, we have to accommodate the access time requirement of 250ns. Thus, the PGT of the TRANSFER PULSE should occur for at least 180ns after the NGT.

**12.13** (a) PROM; (b) MROM; (c) All of these memories are nonvolatile; (d) EPROM, EEPROM, FLASH. (e) EEPROM; (f) EPROM; (g) EEPROM, FLASH; (h) PROM; (i) FLASH; (j) EEPROM, FLASH. (k) PROM, EPROM, EEPROM, FLASH. (l) EPROM

**12.14** Row 3 will be active (HIGH). Thus, transistors Q13, Q14 and Q15 will be conducting.

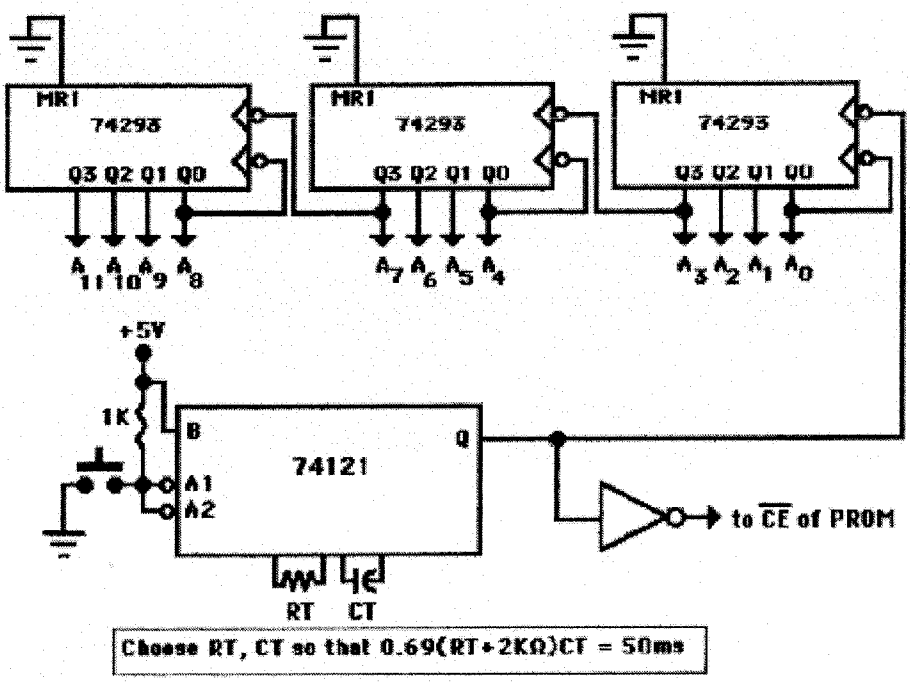
**12.15**

| (X) | | (Y = 3x + 5) | | | |
|---|---|---|---|---|---|
| A1 | A0 | D3 | D2 | D1 | D0 |
| 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 |

Row 0: Connections to the bases of transistors Q3 and Q1 will be made. Connections to the bases of transistors Q0 and Q2 will be unconnected.
Row 1: Connection to the base of transistor Q4 will be made. Connections to the bases of transistors Q5, Q6 and Q7 will be unconnected.
Row 2: Connections to the bases of transistors Q8, Q10 and Q11 will be made. Connection to the base of transistor Q9 will be unconnected.
Row 3: Connections to the bases of transistors Q12, Q13 and Q14 will be made. Connection to the base of transistor Q15 will be unconnected.

**12.16** (a) Counter is initially RESET so that address inputs to EPROM are 000000000000. Switches are set for desired data. The PROGRAM push-button is depressed and released. This triggers OS to apply an inverted 50ms PROGRAM PULSE to the EPROM. The NGT of the PROGRAM PULSE increments the counter to the next address. Likewise, the inverted PROGRAM PULSE programs the 2732 EPROM with data from the switches. This process is repeated until each EPROM address has been programmed with desired data.

(b)



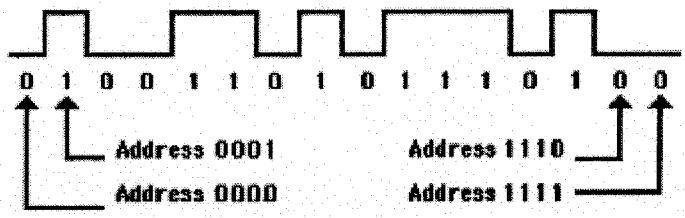
(c) Switch bounce should have no effect because the bounce duration will not exceed 50ms.

**12.17** (a) Writing code $20_{16}$ to the command register of the 28F256A flash memory sets up the chip for the erase operation. Again, writing the code $20_{16}$ to the command register commences the erase operation. This 2-step process will take about 20ms. After the memory is erased, all locations will be loaded with data $FF_{16}$. Thus, this sequence of operations is called the "Set-up Erase/Erase Command"

(b) Writing code $40_{16}$ to the command register of the 28F256A flash memory sets up the chip for the byte programming (Set-up Program/Program Command). After, writing data $3C_{16}$ at the time when address $2300_{16}$ is on the address bus, will cause data $3C_{16}$ to be loaded into memory location $2300_{16}$.

**12.18** Each data output waveform will change according to the truth table as the counter sequences through the various addresses. The D0 waveform is shown below.

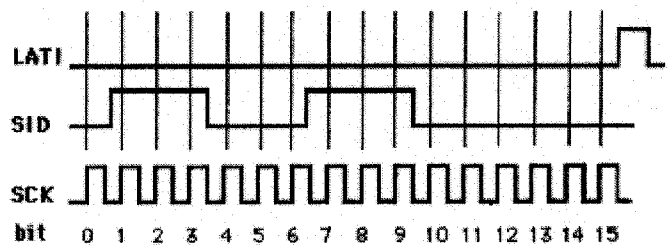

**12.19**



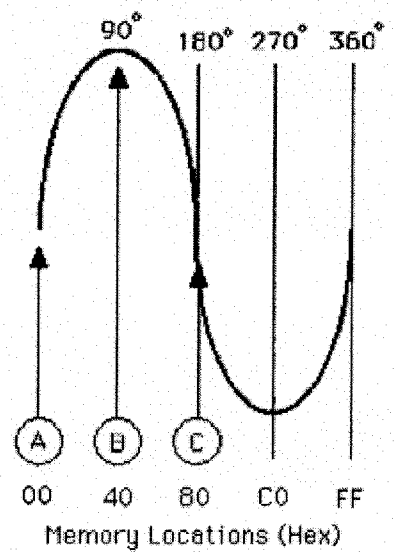

Change D7 in truth-table of figure 12.6 (b) so that it has the levels shown above at the various addresses. Therefore, the hex data will be 5E, BA, 05, 2F, 99, FB, 00, ED, 3C, FF, B8, C7, 27, EA, 52, 5B.

**12.20** (a) 100Hz x 256 = 25.6KHz; (b) Adjust $V_{ref}$.

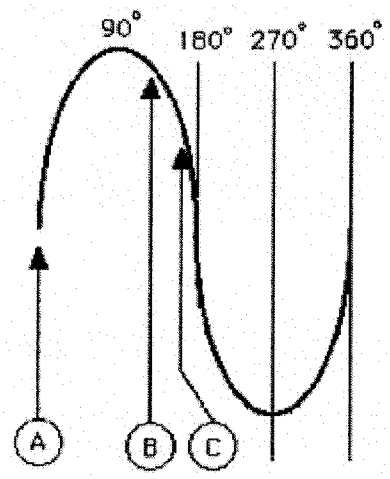
**12.21**



**12.22** (a)




(b)




$\frac{B}{FFH} = \frac{120}{360}$. Thus, B = $85_{10}$ = 55H

$\frac{C}{FFH} = \frac{240}{360}$. Thus, C = $170_{10}$ = AAH

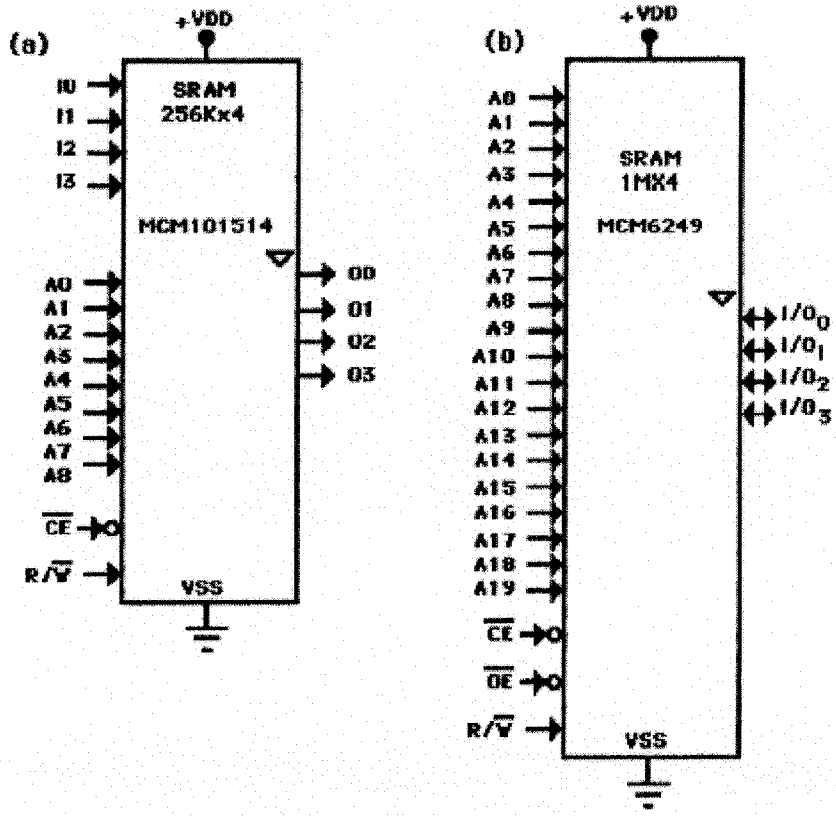
(c) 1-Cycle = 256 points 60 cycles/sec = 60 x 256 points/sec = 15,360 Hz

(d) Sequencer outputs A, B, and C must remain active long enough to allow for tpd of the counter, tacc of the ROM, and tpd of the octal latch.
$T_{CK(min)}$ = 10+20+5 = 35ns
$F_{CK(max)} = 1/T_{CK(min)}$ = 1/35ns = 28.6 MHz

(e) It takes 4 cycles of CK for each DAC-OUT pulse and 256 DAC-OUT pulses per cycle.
$T_{SINE} = T_{CK}$ x 4 x 256 = 35.84 µs
$F_{SINE} = 1/T_{SINE}$ = 1/35.84 µs = 27.9 kHz

(f) It supplies the Most-significant address bits to ROM to select the type of waveform.

**12.23**

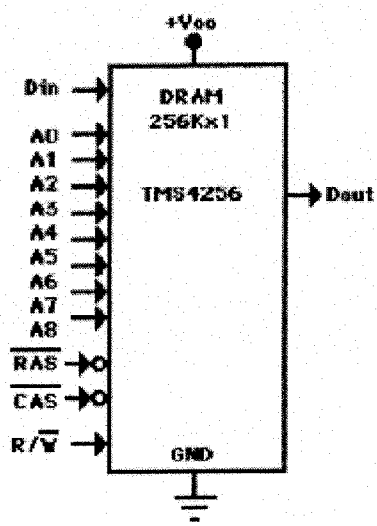



**12.24** Refer to figure 12.22:

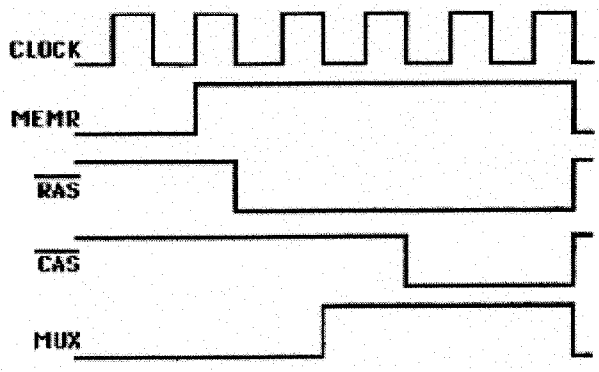
(a) $t_{ACC}$=100ns: (b) $t_{OD}$=30ns: (c) $t_{RC}$=100ns; 1/100ns=10 million: (d) $t_{AS}$=20ns:
(b) (e) $t_{DS}+t_{DH}$=30ns: (f) $t_{AH}=t_{WC}-(t_{AS}+t_W)$=40ns: (g) $t_{WC}$=100ns; 1/100ns=10 million

**12.25**

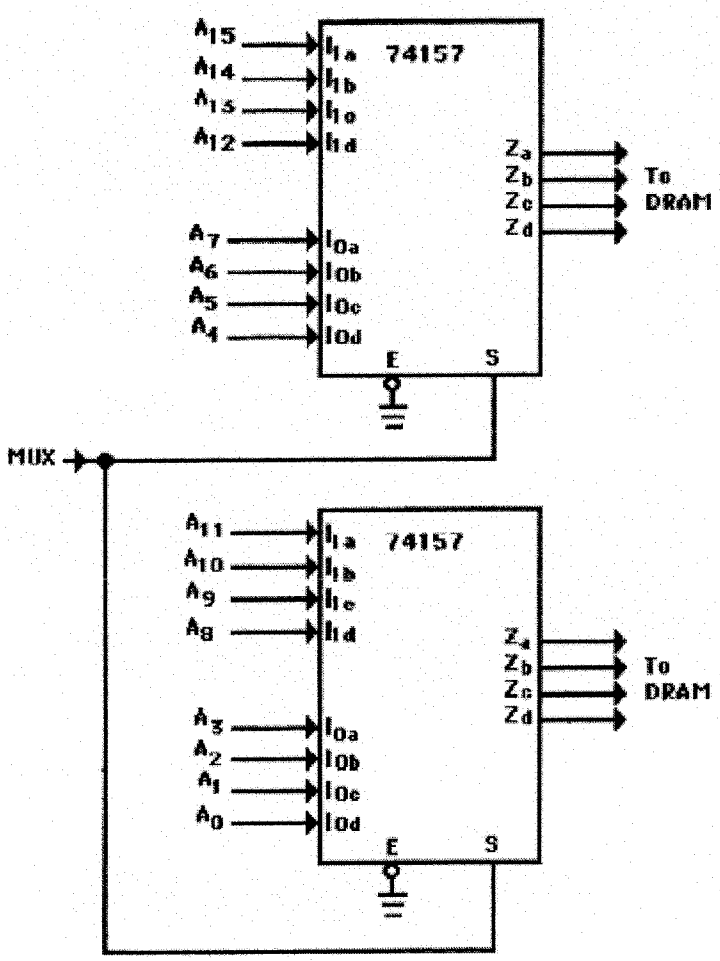
We save 9 address pins by using address multiplexing, but we need $\overline{RAS}$ and $\overline{CAS}$ signals instead of a single CS. Thus, we save a Net Total of 8 pins.



**12.26**




**12.27**

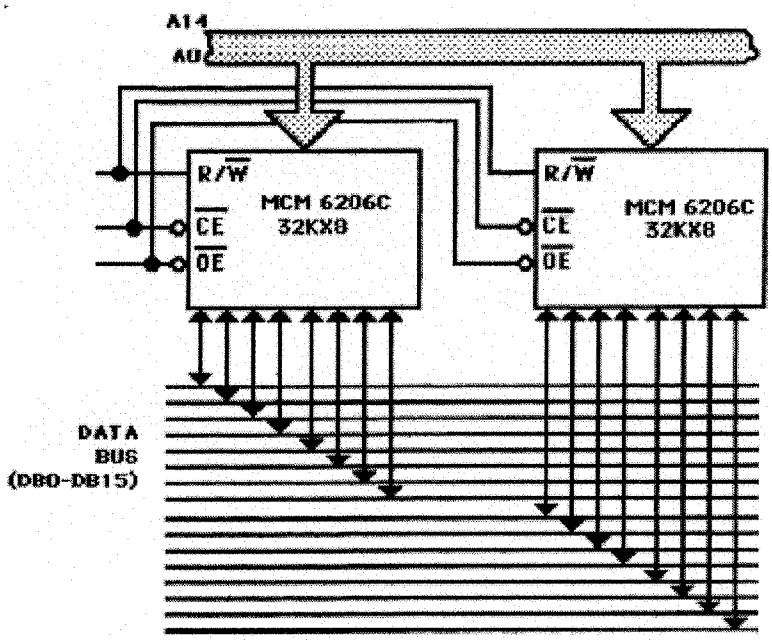



**12.28/29** See section 12.15 of text book.

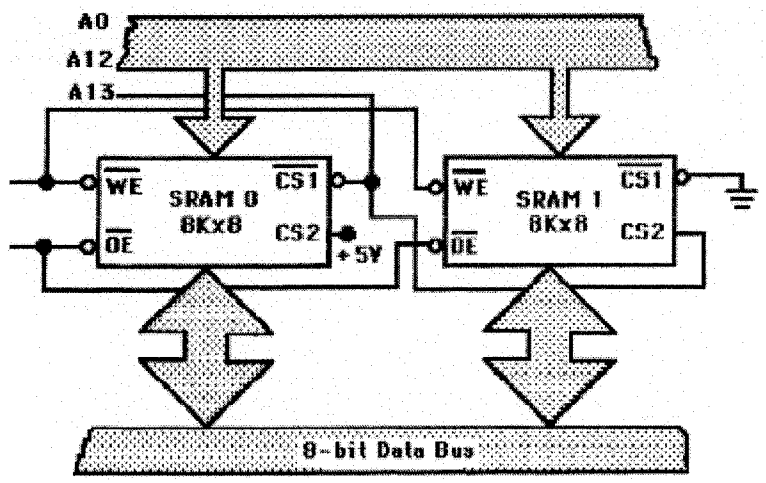
**12.30** The cycles of $\overline{CAS}$-before-$\overline{CAS}$ must be applied at least every 7.8μs (4ms/512) in order for the data to be retained.

**12.31** (a) $2^8$ groups of 16 MUXed columns = 256x16 = 4096 columns. $2^{10}$ rows = 1024 rows.
(b) 2048 rows (c) The total refresh time would double to 226μs.

**12.32**




**12.33**



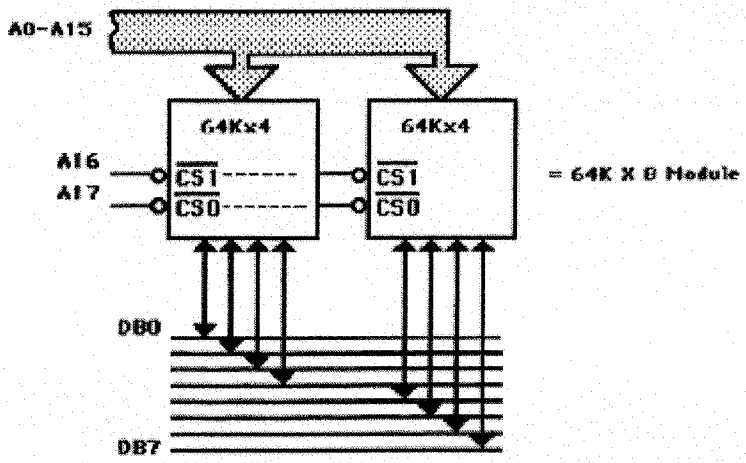

SRAM 0 has an address range from $0000_{16}$-$1FFF_{16}$
SRAM 1 has an address range from $2000_{16}$-$3FFF_{16}$

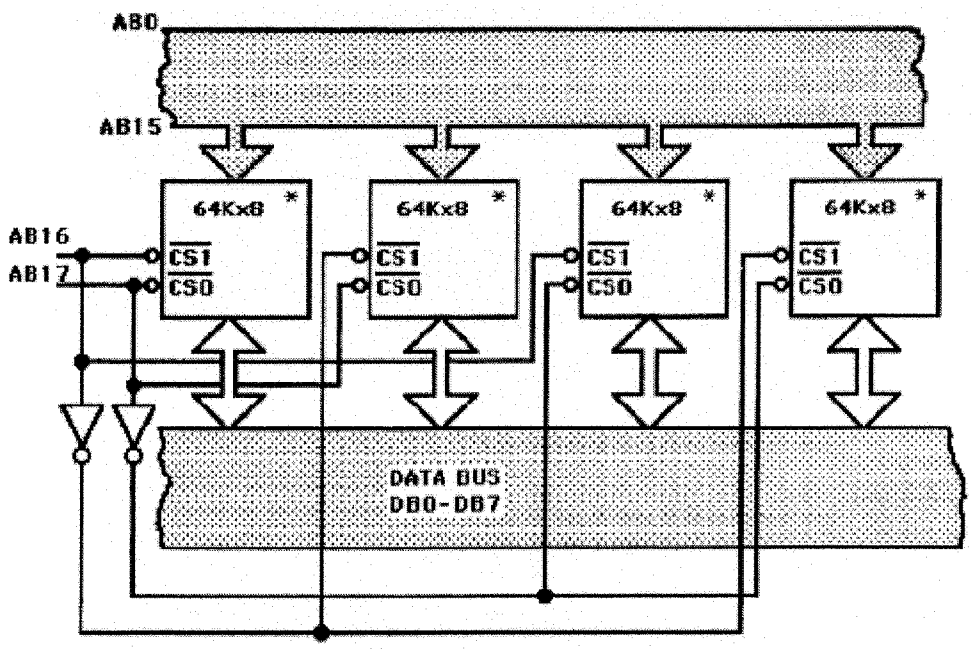
**12.34** 1)Add four more PROMs (PROM-4 through PROM-7) to the circuit of fig.12.37.
2) Connect $AB_{13}$ to C input of the 3-line-to-8-line decoder.
3) Connect outputs 4 through 7 of the decoder to the CS inputs of PROMS 4 through 7 respectively.

**12.35** (1) Connect $AB_{13}$, $AB_{14}$, and $AB_{15}$ to the inputs of a 3-input OR gate.
(2) Replace the existing LOW at input C of the decoder with the output of the OR gate.

**12.36** (a) The RAM
(b) No. The chip is completely decoded, only one address accesses this memory location.
(c) 6007 stores in the EEPROM. Since A11, A12 are not involved in decoding (don't cares) there are other addresses in the system that will access the same memory location in the EEPROM as address 6007. They are 6807, 7007, and 7807.
(d) Storing a byte at 6800 will actually write over (and destroy) the byte that was previously stored at address 6000. A careful programmer will avoid this situation.

**12.37**



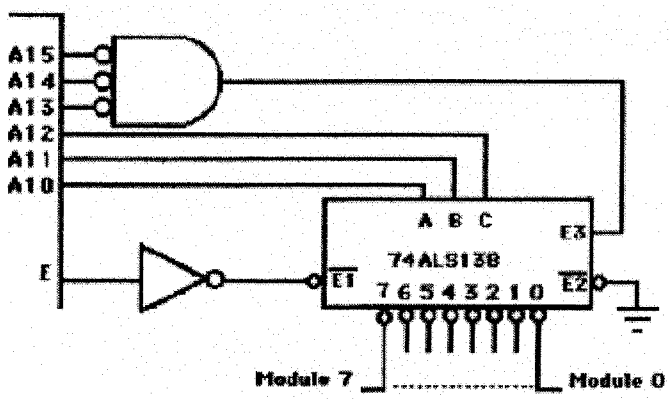

** Each 64Kx8 module consists of two 64Kx4 RAM chips with their address inputs, chip select inputs and R/W inputs tied in parallel.*

**12.38** The 74ALS138 is enabled only when E is active and $AB_{12}$-$AB_{15}$=HIGH.

| ADDRESS BUS LINES | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 1 | 1 | 1 | 1 | X | X | X | X | X | X | X | X | X | X | X | X | F000-FFFF |
| 1 | 1 | 1 | 1 | 0 | 0 | X | X | X | X | X | X | X | X | X | X | |
| 1 | 1 | 1 | 1 | 0 | 1 | X | X | X | X | X | X | X | X | X | X | |
| 1 | 1 | 1 | 1 | 1 | 0 | X | X | X | X | X | X | X | X | X | X | |
| 1 | 1 | 1 | 1 | 1 | 1 | X | X | X | X | X | X | X | X | X | X | |

F000-F3FF Module 0
F400-F7FF Module 1
F800-FBFF Module 2
FC00-FFFF Module 3

**12.39** Four more 1Kx8 modules ( 4 through 7) will be added to the circuit of figure 12.42. Their inputs and outputs will be connected the same way modules 0 through 3 are already connected. Their CS inputs will be connected to outputs 4 through 7 of the 74ALS138. The new decoding logic is shown below:




| ADDRESS BUS LINES | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 0 | 0 | 0 | X | X | X | X | X | X | X | X | X | X | X | X | X | 0000-1FFFF |
| 0 | 0 | 0 | 0 | 0 | 0 | X | X | X | X | X | X | X | X | X | X | |
| 0 | 0 | 0 | 0 | 0 | 1 | X | X | X | X | X | X | X | X | X | X | |
| 0 | 0 | 0 | 0 | 1 | 0 | X | X | X | X | X | X | X | X | X | X | |
| 0 | 0 | 0 | 0 | 1 | 1 | X | X | X | X | X | X | X | X | X | X | |
| 0 | 0 | 0 | 1 | 0 | 0 | X | X | X | X | X | X | X | X | X | X | |
| 0 | 0 | 0 | 1 | 0 | 1 | X | X | X | X | X | X | X | X | X | X | |
| 0 | 0 | 0 | 1 | 1 | 0 | X | X | X | X | X | X | X | X | X | X | |
| 0 | 0 | 0 | 1 | 1 | 1 | X | X | X | X | X | X | X | X | X | X | |

0000-03FF Module 0
0400-07FF Module 1
0800-0BFF Module 2
0C00-0FFF Module 3
1000-13FF Module 4
1400-17FF Module 5
1800-1BFF Module 6
1C00-1FFF Module 7

**12.40** The problem exists because of a permanent logic HIGH at the B input of the 74ALS138. The B input of the 74ALS138 could:

(a) be internally or externally shorted to Vcc; (b) be floating (open).

**12.41** The problem exists because of a permanent logic LOW at the C input of the 74ALS138.

(a) The C input of the 74ALS138 could be internally or externally shorted to Ground.

(b) The OR gate could have its output internally or externally grounded.

**12.42** (a) If a break exists in the connection to the A input of the 74ALS138, this input will act as if it was always HIGH (A floating input in TTL assumes a logic HIGH). Thus, the binary sequence at the decoder's input would be as follows: $001_2$,$001_2$,$011_2$,$011_2$,$101_2$,$101_2$,$111_2$,$111_2$. This would cause the SELF-TEST program to perform a checkerboard test sequence for RAM module-1 twice in a row, as well as for RAM module-3. Thus, RAM modules 0 and 2 never get tested. However, the SELF-TEST program has no way of detecting this fault, and therefore assumes that all four modules test OK.

(b) The following software implementation can be done in order to detect the preceding fault:

(1) Clear or Set all memory.
(2) Write a checkerboard pattern to an address (ie.$0000_{16}$).
(3) Read all other memory locations to see if only one memory location contains the checkerboard pattern.
(4) Clear or Set all memory.
(5) Write a checkerboard pattern to the next address (ie.$0001_{16}$).
(6) Go to step (3) and continue until all memory has been checked.

**12.43** The 1Kx4 RAM chip with data outputs 4 through 7 which is one half of the RAM module 2, shows a fault in every location.

Some possible causes:

(a) An unconnected Vcc or Ground on the 1Kx4 RAM chip.
(b) The 1Kx4 RAM chip does not respond to the CS signal.
(c) The CS node connection to the 1Kx4 RAM chip is open.
(d) The R/W node connection to the 1Kx4 RAM chip is open.

**12.44** (a) Connection from output 7 of RAM module-3 to data bus line D7 is externally open.
(b) Output 7 of RAM module-3 is internally open or in permanent HI-Z state.

**12.45** Since $\overline{K3}$ and $\overline{K2}$ are shorted together, they will always be at the same voltage level. When either one is forced to go LOW by the application of its input code, there will be two possibilities: both $\overline{K3}$ and $\overline{K2}$ will go to a valid LOW level; or both $\overline{K3}$ and $\overline{K2}$ will be at some indeterminate level. In the first case, both RAM modules 2 and 3 will be activated at the same time whenever the CPU is writing to or reading from either one. This will probably not be detected by the checkerboard self-test since both RAM modules will respond in exactly the same way to each write and read operation.

In the second case, the indeterminate levels at $\overline{K2}$ and $\overline{K3}$ will probably result in erratic write and read operations at all locations in both modules. The self-test messages will probably look like this:

module-0 test OK
module-1 test OK
address 0800 faulty at bits 0-7
address 0801 faulty at bits 0-7
" " " " " "
address 0BFE faulty at bits 0-7
address 0BFF faulty at bits 0-7
address 0C00 faulty at bits 0-7
address 0C01 faulty at bits 0-7
" " " " " "
address 0FFE faulty at bits 0-7
address 0FFF faulty at bits 0-7

**12.46** Checksum=DE+3A+85+AF+19+7B+00+ED+3C+FF+B8+C7+27+6A+D2=$EA_{16}$
Therefore, checksum at address $1111_2$=$11101010_2$=$EA_{16}$

## CHAPTER THIRTEEN - Programmable Logic Device Architectures

**13.1** (a) Standard logic refers to SSI and MSI chips that provide basic digital functions.
(b) ASICs are ICs that are designed to implement a specific application.
(c) Microprocessor/DSP devices control components in a system and manipulate data by executing a program of instructions,

**13.2** The necessary speed of operation for the circuit, cost of manufacturing, system power consumption, system size, amount of time available to design the product, etc.

**13.3** Because its functionality is determined by the program of instructions, the "software."

**13.4** Speed of operation.

**13.5** (a) PLDs use programmable electronic switches to create the desired functionality using the logic hardware available on the IC.
(b) Gate arrays use customized interconnections, created during IC fabrication, between the prefabricated gates on a silicon wafer to create the desired functionality.
(b) Standard cells use predefined logic function building blocks to create the desired functionality in an IC.
(c) Full custom employs layout of components and interconnections to design an IC for the desired application.

**13.6** Advantages: highest speed and smallest die area.
Disadvantages: design/development time and expense.

**13.7** Fuse, EPROM, EEPROM, Flash, SRAM, Antifuse;
OTP: Fuse and Antifuse;
Volatile: SRAM

**13-8** SRAM-based PLDs must be configured (programmed) upon power-up

**13-9** (a) LAB (Logic Array Block) is a set of 16 macrocells.
(b) PIA (Programmable Interconnect Array) is a bus that connects signal sources and destinations within the CPLD.
(c) Macrocell is the programmable logic block containing an AND/OR circuit & a flip-flop to create desired logic functions.

**13-10** In a PLD programmer or in-system (via JTAG interface)

**13-11** Joint Test Action Group (JTAG) interface

**13-12** pin 1 – GCLRn (Global Clear)
pin 2 – OE2/GCLK2 (Output Enable 2/Global Clock 2)
pin 83 – GCLK1 (Global Clock 1)
pin 84 – OE1 (Output Enable 1)

**13-13** Provides high-speed clocking

**13-14** Logic cell in MAX7000S is AND/OR circuit vs. lookup table in FLEX10K;
EEPROM (MAX7000S) & SRAM (FLEX10K);
MAX7000S is non-volatile;
FLEX10K has greater logic resources