Digital Design and Computer Architecture Solved Exercises

Chapter 1

Exercise 1.1

Explain in one paragraph at least three levels of abstractions that are used by

A) Biologists studying the operation of cells
B) Chemists studying the composition of matter

Solution

A) An example can be how the cells form tissue and the tissue forms organs.
B) The atoms are made by electrons, protons, and neutrons and the atoms react with other atoms to form molecules.

Exercise 1.2

Explain in one paragraph how the techniques of hierarchy, modularity and regularity may be used by

A) Automobile designers
B) Businesses to manage their operations

Solution

A) Automobile designers use hierarchy to define how a car must be assembled, use modularity to define their most basic components like the engine and the transmission, but use regularity when they define which range of operations and how must be supplied.
B) Businesses use hierarchy defining the structure of their employees, use modularity to split their workforce to many departments to focus on their workflow instead of their integration, but use regularity to assure the integration of all departments with protocols and supervisors.

Exercise 1.3

Ben Bitdiddle is building a house. Explain how he can use the principles of hierarchy, modularity, and regularity to save time and money during construction.

Solution

He needs to use bricks to get a wall, and walls to get a room. He also first laid the foundation before starting to build, and finally he needed to define the quality of the materials because using some damaged materials could be an important danger.

Exercise 1.4

An analog voltage is in the range of 0-5V, if it can be measured with an accuracy of ±50mV, at most, how many bits of information does it convey?

Solution

It can convey $(5V-0V)/0.05V=100V$ discrete values which can fit on 2^7=128, so it needs 7-bit information to represent that whole range of values.

Exercise 1.5

A classroom has an old clock on the wall whose minute hand broke off

A) If you can read the hour to the nearest 15 minutes, how many bits of information does the clock convey about time?
B) If you know whether it is before or after noon, how many additional bits of information do you know about the time?

Solution

A) If it can only measure every quarter of an hour then we get 12hrs/0.25hrs=48 discrete values, then it fits on 2^6=64, so it needs 6-bit information to represent those values.
B) You need an extra bit to represent the sign.

Exercise 1.6

The Babylonians developed the sexagesimal (base 60) number system about 4000 years ago. How many bits of information are conveyed with one sexagesimal digit? How do you write the number 4000_{10} in sexagesimal?

Solution

Considering that sexagesimal is base 60 it needs 6 bits to fit only one digit because it can store 64 values. To find the value of 4000 in sexagesimal it is necessary to be processed to find the integer part from the division of 60 and save the remains to form the digits, like its shows on the table.

Integer division	Remain
$4000/60=66$	40
$66/60=1$	6
$1/60=0$	1

The first remain is then the first digit form the result because is divided by 60 (regroup the number in quantity of sixty's) then the second digit is the number 6 cause is divided by 60^2 (group of sixty sixties) and the final digit is 1 cause is divided by 60^3.

Exercise 1.7

How many different numbers can be represented with 16 bits?

Solution

$2^{16}=65536$

Exercise 1.8

What is the largest unsigned 32 bit binary number?

Solution

$2^{32}-1=4294967295$.

Exercise 1.9

What is the largest 16 bit binary number that can be represented with

A) Unsigned numbers?
B) Two's complement numbers?
C) Sign/magnitude numbers?

Solution

A) $2^{16}-1=65535$
B) $2^{16-1}-1=32767$
C) $2^{16-1}-1=32767$

Exercise 1.10

What is the largest 32 bit binary number that can be represented with

A) Unsigned numbers?
B) Two's complement numbers?
C) Sign/magnitude numbers?

Solution

A) $2^{32}-1=4294967295$
B) $2^{32-1}-1=2147483647$
C) $2^{32-1}-1=2147483647$

Exercise 1.11

What is the smallest (most negative) 16 bit binary number that can be represented with

A. Unsigned numbers?
B. Two's complement numbers?
C. Sign/magnitude numbers?

Solution

A) 0
B) $-2^{16-1}=-32768$
C) $-2^{16-1}-1=-32767$

Exercise 1.12

What is the smallest (most negative) 32 bit binary number that can be represented with

A) Unsigned numbers?
B) Two's complement numbers?
C) Sign/magnitude numbers?

Solution

A) 0
B) $-2^{32-1}=-4294967296$
C) $-2^{32-1}-1=-4294967295$

Exercise 1.13

Convert the following unsigned binary numbers to decimal. Show your work

A) 1010_2

B) 110110_2
C) 11110000_2
D) 000100010100111_2

Solution

A) $1010_2 = 1\times2^3+0\times2^2+1\times2^1+0\times2^0 = 10_{10}$
B) $110110_2 = 1\times2^5+1\times2^4+0\times2^3+1\times2^2+1\times2^1+0\times2^0 = 54_{10}$
C) $11110000_2 = 1\times2^7+1\times2^6+1\times2^5+1\times2^4+0\times2^3+0\times2^2+0\times2^1+0\times2^0 = 240_{10}$
D) $000100010100111_2 = 1\times2^{11}+1\times2^7+1\times2^5+1\times2^2+1\times2^1+1\times2^0 = 2215_{10}$

Exercise 1.14

Convert the following unsigned binary numbers to decimal. Show your work

A) 1110_2
B) 100100_2
C) 11010111_2
D) 011101010100100_2

Solution

A) $1110_2 = 1\times2^3+1\times2^2+1\times2^1+0\times2^0 = 14_{10}$
B) $100100_2 = 1\times2^5+0\times2^4+0\times2^3+1\times2^2+0\times2^1+0\times2^0 = 36_{10}$
C) $11010111_2 = 1\times2^7+1\times2^6+0\times2^5+1\times2^4+0\times2^3+1\times2^2+1\times2^1+1\times2^0 = 215_{10}$
D) $011101010100100_2 = 1\times2^{13}+1\times2^{12}+1\times2^{11}+1\times2^9+1\times2^7+1\times2^5+1\times2^2 = 2215_{10}$

Exercise 1.15

Repeat Exercise 1.13, but convert to hexadecimal.

Solution

A) $1010_2 = A_{16}$
B) $110110_2 = 0011_2\times16^1+0110_2\times16^0 = 36_{16}$
C) $11110000_2 = 1111_2\times16^1+0000_2\times16^0 = F\,0_{16}$
D) $000100010100111_2 = 0000_2\times16^3+1000_2\times16^2+1010_2\times16^1+0111_2\times16^0 = 08\,A\,7_{16}$

Exercise 1.16

Repeat Exercise 1.14, but convert to hexadecimal.

Solution

A) $1110_2 = E_{16}$
B) $100100_2 = 0010_2\times16^1+0100_2\times16^0 = 24_{16}$
C) $11010111_2 = 1101_2\times16^1+0111_2\times16^0 = C\,7_{16}$
D) $011101010100100_2 = 0011_2\times16^3+1010_2\times16^2+1010_2\times16^1+0100_2\times16^0 = 3\,AA\,4_{16}$

Exercise 1.17

Convert the following hexadecimal numbers to decimal. Show your work.

A) $A\,5_{16}$
B) $3\,B_{16}$

C) $FFFF_{16}$
D) $D0000000_{16}$

Solution

A) $A5_{16} = 10 \times 16^1 + 5 \times 16^0 = 165_{10}$
B) $3B_{16} = 3 \times 16^1 + 11 \times 16^0 = 59_{10}$
C) $FFFF_{16} = 15 \times 16^3 + 15 \times 16^2 + 15 \times 16^1 + 15 \times 16^0 = 65535_{10}$
D) $D0000000_{16} = 13 \times 16^7 = 3489660928_{10}$

Exercise 1.18

Convert the following hexadecimal numbers to decimal. Show your work.

A) $4E_{16}$
B) $7C_{16}$
C) $ED3A_{16}$
D) $403FB001_{16}$

Solution

A) $4E_{16} = 4 \times 16^1 + 14 \times 16^0 = 78_{10}$
B) $7C_{16} = 7 \times 16^1 + 12 \times 16^0 = 124_{10}$
C) $ED3A_{16} = 14 \times 16^3 + 13 \times 16^2 + 3 \times 16^1 + 10 \times 16^0 = 60730_{10}$
D) $403FB001_{16} = 4 \times 16^7 + 0 \times 16^6 + 3 \times 16^5 + 15 \times 16^4 + 11 \times 16^3 + 1 \times 16^2 + 1 \times 16^1 + 1 \times 16^0 = 1077915649_{10}$

Exercise 1.19

Repeat Exercise 1.17, but convert to unsigned binary.

Solution

A) $A5_{16} = 1010_2 \times 16^1 + 0101_2 \times 16^0 = 10100101_2$
B) $3B_{16} = 0011_2 \times 16^1 + 1011_2 \times 16^0 = 00111011_2$
C) $FFFF_{16} = 1111_2 \times 16^3 + 1111_2 \times 16^2 + 1111_2 \times 16^1 + 1111_2 \times 16^0 = 1111111111111111_2$
D) $D0000000_{16} = 1101_2 \times 16^7 = 11010000000000000000000000000000_2$

Exercise 1.20

Repeat Exercise 1.18, but convert to unsigned binary.

Solution

A) $4E_{16} = 0100_2 \times 16^1 + 1110_2 \times 16^0 = 01001110_2$
B) $7C_{16} = 0111_2 \times 16^1 + 1100_2 \times 16^0 = 01111100_2$
C) $ED3A_{16} = 1110_2 \times 16^3 + 1101_2 \times 16^2 + 0011_2 \times 16^1 + 1010_2 \times 16^0 = 1110110100111010_2$
D) $403FB001_{16} = 0100_2 \times 16^7 + 0000_2 \times 16^6 + 0011_2 \times 16^5 + 1111_2 \times 16^4 + 1011_2 \times 16^3 + 0000_2 \times 16^2 + 0000_2 \times 16^1 + 0001_2 \times 16^0 = 01000000001111111011000000000001_2$

Exercise 1.21

Convert the following two’s complement binary numbers to decimal.

A) 1010_2
B) 110110_2
C) 01110000_2
D) 10011111_2

Solution

A) $1010_2 = -1\times2^3+0\times2^2+1\times2^1+0\times2^0 = -6_{10}$
B) $110110_2 = -1\times2^5+1\times2^4+0\times2^3+1\times2^2+1\times2^1+0\times2^0 = -10_{10}$
C) $01110000_2 = -0\times2^7+1\times2^6+1\times2^5+1\times2^4+0\times2^3+0\times2^2+0\times2^1+0\times2^0 = 112_{10}$
D) $10011111_2 = -1\times2^7+0\times2^6+0\times2^5+1\times2^4+1\times2^3+1\times2^2+1\times2^1+1\times2^0 = -97_{10}$

Exercise 1.22

Convert the following two's complement binary numbers to decimal.

A) 1110_2
B) 100011_2
C) 01001110_2
D) 10110101_2

Solution

A) $1110_2 = -1\times2^3+1\times2^2+1\times2^1+0\times2^0 = -2_{10}$
B) $100011_2 = -1\times2^5+0\times2^4+0\times2^3+0\times2^2+1\times2^1+1\times2^0 = -29_{10}$
C) $01001110_2 = -0\times2^7+1\times2^6+0\times2^5+0\times2^4+1\times2^3+1\times2^2+1\times2^1+0\times2^0 = 78_{10}$
D) $10110101_2 = -1\times2^7+0\times2^6+1\times2^5+1\times2^4+0\times2^3+1\times2^2+0\times2^1+1\times2^0 = -75_{10}$

Exercise 1.23

Repeat Exercise 1.21, assuming that the binary numbers are in sign/magnitude form rather than two's complement representation.

Solution

A) $1010_2 = -(0\times2^2+1\times2^1+0\times2^0) = -2_{10}$
B) $110110_2 = -(1\times2^4+0\times2^3+1\times2^2+1\times2^1+0\times2^0) = -22_{10}$
C) $01110000_2 = +(1\times2^6+1\times2^5+1\times2^4+0\times2^3+0\times2^2+0\times2^1+0\times2^0) = 112_{10}$
D) $10011111_2 = -(0\times2^6+0\times2^5+1\times2^4+1\times2^3+1\times2^2+1\times2^1+1\times2^0) = -31_{10}$

Exercise 1.24

Repeat Exercise 1.22, assuming that the binary numbers are in sign/magnitude form rather than two's complement representation.

Solution

A) $1110_2 = -(1\times2^2+1\times2^1+0\times2^0) = -6_{10}$
B) $100011_2 = -(0\times2^4+0\times2^3+0\times2^2+1\times2^1+1\times2^0) = -3_{10}$
C) $01001110_2 = +(1\times2^6+0\times2^5+0\times2^4+1\times2^3+1\times2^2+1\times2^1+0\times2^0) = 78_{10}$
D) $10110101_2 = -(0\times2^6+1\times2^5+1\times2^4+0\times2^3+1\times2^2+0\times2^1+1\times2^0) = -53_{10}$

Exercise 1.25

Convert the following decimal numbers to unsigned binary numbers

A) 42_{10}
B) 63_{10}
C) 229_{10}
D) 845_{10}

Solution

A) $42_{10} = 101010_2$

Integer division	Remainder
42/2=21	0
21/2=10	1
10/2=5	0
5/2=2	1
2/2=1	0
1/2=0	1

B) $63_{10} = 111111_2$

Integer division	Remainder
63/2=31	1
31/2=15	1
15/2=7	1
7/2=3	1
3/2=1	1
1/2=0	1

C) $229_{10} = 11100101_2$

Integer division	Remainder
229/2=114	1
114/2=57	0
57/2=28	1
28/2=14	0
14/2=7	0
7/2=3	1
3/2=1	1
1/2=0	1

D) $845_{10} = 1101001101_2$

Integer division	Remainder
845/2=422	1
422/2=211	0
211/2=105	1
105/2=52	1
52/2=26	0
26/2=13	0
13/2=6	1
6/2=3	0
3/2=1	1
1/2=0	1

Exercise 1.26

Convert the following decimal numbers to unsigned binary numbers

A) 14_{10}
B) 52_{10}
C) 339_{10}
D) 711_{10}

Solution

A) $14_{10} = 1110_2$

Integer division	Remainder
14/2=7	0
7/2=3	1
3/2=1	1
1/2=0	1

B) $52_{10} = 110100_2$

Integer division	Remainder
52/2=26	0
26/2=13	0
13/2=6	1
6/2=3	0
3/2=1	1
1/2=0	1

C) $339_{10} = 101010011_2$

Integer division	Remainder
339/2=169	1
169/2=84	1
84/2=42	0
42/2=21	0
21/2=10	1
10/2=5	0
5/2=2	1
2/2=1	0
1/2=0	1

D) $711_{10} = 1011000111_2$

Integer division	Remainder
711/2=355	1
355/2=177	1
177/2=88	1
88/2=44	0
44/2=22	0
22/2=11	0
11/2=5	1
5/2=2	1
2/2=1	0
1/2=0	1

Exercise 1.27

Repeat Exercise 1.25, but convert to hexadecimal.

Solution

A) $42_{10} = 101010_2 = 0010_2 \times 16^1 + 1010_2 \times 16^0 = 2A_{16}$

B) $63_{10} = 111111_2 = 0011_2 \times 16^1 + 1111_2 \times 16^0 = 3F_{16}$

C) $229_{10} = 11100101_2 = 1110_2 \times 16^1 + 0101_2 \times 16^0 = E5_{16}$

D) $845_{10} = 1101001101_2 = 0011_2 \times 16^2 + 0100_2 \times 16^1 + 1101_2 \times 16^0 = 34D_{16}$

Exercise 1.28

Repeat Exercise 1.26, but convert to hexadecimal.

Solution

A) $14_{10} = 1110_2 = E_{16}$
B) $52_{10} = 110100_2 = 0011_2 \times 16^1 + 0100_2 \times 16^0 = 34_{16}$
C) $339_{10} = 101010011_2 = 0001_2 \times 16^2 + 0101_2 \times 16^1 + 0011_2 \times 16^0 = 153_{16}$
D) $711_{10} = 1011000111_2 = 0010_2 \times 16^2 + 1100_2 \times 16^1 + 0111_2 \times 16^0 = 2C7_{16}$

Exercise 1.29

Convert the following decimal numbers to 8 bit two's complement numbers or indicate that the decimal number would overflow the range

A) 42_{10}
B) -63_{10}
C) 124_{10}
D) -128_{10}
E) 133_{10}

Solution

A) $42_{10} = 00101010_2$
B) $-63_{10} = 11000001_2$
C) $124_{10} = 01111100_2$
D) $-128_{10} = 10000000_2$
E) It overflows

Exercise 1.30

Convert the following decimal numbers to 8 bit two's complement numbers or indicate that the decimal number would overflow the range

A) 24_{10}
B) -59_{10}
C) 128_{10}
D) -150_{10}
E) 127_{10}

Solution

A) $24_{10} = 00011000_2$
B) $-59_{10} = 11000101_2$
C) It overflows
D) It overflows
E) $127_{10} = 01111111_2$

Exercise 1.31

Repeat Exercise 1.29, but convert to 8 bit sign/magnitude numbers.

Solution

A) $42_{10} = 00101010_2$
B) $-63_{10} = 11000001_2$
C) $124_{10} = 01111100_2$
D) It overflows

E) It overflows

Exercise 1.32

Repeat Exercise 1.30, but convert to 8 bit sign/magnitude numbers.

Solution

A) $24_{10} = 00011000_2$
B) $-59_{10} = 10111011_2$
C) It overflows
D) It overflows
E) $127_{10} = 01111111_2$

Exercise 1.33

Convert the following 4 bit two's complement numbers to 8 bit two's complement numbers.

A) 0101_2
B) 1010_2

Solution

For convert two's complement numbers to larger sized bits, is required to extend the sign bit

A) $0101_2 = 00000101_2$
B) $1010_2 = 11111010_2$

Exercise 1.34

Convert the following 4 bit two's complement numbers to 8 bit two's complement numbers.

A) 0111_2
B) 1001_2

Solution

A) $0111_2 = 00000111_2$
B) $1001_2 = 11111001_2$

Exercise 1.35

Repeat Exercise 1.33 if the numbers are unsigned rather than two's complement.

Solution

Only fill the new bits spaces with zeros

A) $0101_2 = 00000101_2$
B) $1010_2 = 00001010_2$

Exercise 1.36

Repeat Exercise 1.34 if the numbers are unsigned rather than two's complement.

Solution

A) $0111_2 = 00000111_2$
B) $1001_2 = 00001001_2$

Exercise 1.37

Base 8 is referred to as octal. Convert each of the numbers from Exercise 1.25 to octal.

Solution

A) $42_{10} = 52_8$

Integer division	Remainder
$42/8 = 5$	2
$5/8 = 0$	5

B) $63_{10} = 77_8$

Integer division	Remainder
$63/8 = 7$	7
$7/8 = 0$	7

C) $229_{10} = 345_8$

Integer division	Remainder
$229/8 = 28$	5
$28/8 = 3$	4
$3/8 = 0$	3

D) $845_{10} = 1515_8$

Integer division	Remainder
$845/8 = 105$	5
$105/8 = 13$	1
$13/8 = 1$	5
$1/8 = 0$	1

Exercise 1.38

Base 8 is referred to as octal. Convert each of the numbers from Exercise 1.26 to octal.

Solution

A) $14_{10} = 16_8$

Integer division	Remainder
$14/8 = 1$	6
$1/8 = 0$	1

B) $52_{10} = 46_8$

Integer division	Remainder

$52/8=6$	4
$6/8=0$	6

C) $339_{10}=523_8$

Integer division	Remainder
$339/8=42$	3
$42/8=5$	2
$5/8=0$	5

D) $711_{10}=1307_8$

Integer division	Remainder
$711/8=88$	7
$88/8=11$	0
$11/8=1$	3
$1/8=0$	1

Exercise 1.39

Convert each of the following octal numbers to binary, hexadecimal and decimal.

A) 42_8
B) 63_8
C) 255_8
D) 3047_8

Solution

$42_8=4\times8^1+2\times8^0=34_{10}$

A) $42_8=100_2\times8^1+010_2\times8^0=100010_2$
$42_8=100010_2=0010_2\times16^1+0010_2\times16^0=22_{16}$

$63_8=6\times8^1+3\times8^0=51_{10}$

B) $63_8=110_2\times8^1+011_2\times8^0=110011_2$
$63_8=110011_2=0011_2\times16^1+0011_2\times16^0=33_{16}$

$255_8=2\times8^2+5\times8^1+5\times8^0=173_{10}$

C) $255_8=010_2\times8^1+101_2\times8^1+101_2\times8^0=010101101_2$
$255_8=010101101_2=1010_2\times16^1+1101_2\times16^0=AD_{16}$

$3047_8=3\times8^3+0\times8^2+4\times8^1+7\times8^0=1575_{10}$

D) $3047_8=011_2\times8^2+000_2\times8^1+100_2\times8^1+111_2\times8^0=011000100111_2$
$3047_8=011000100111_2=0110_2\times16^2+0010_2\times16^1+0111_2\times16^0=627_{16}$

Exercise 1.40

Convert each of the following octal numbers to binary, hexadecimal and decimal.

A) 23_8
B) 45_8
C) 371_8
D) 2560_8

Solution

$23_8 = 2 \times 8^1 + 3 \times 8^0 = 19_{10}$

A) $23_8 = 010_2 \times 8^1 + 011_2 \times 8^0 = 010011_2$
$23_8 = 010011_2 = 0001_2 \times 16^1 + 0011_2 \times 16^0 = 13_{16}$

$45_8 = 4 \times 8^1 + 5 \times 8^0 = 37_{10}$

B) $45_8 = 100_2 \times 8^1 + 101_2 \times 8^0 = 100101_2$
$45_8 = 100101_2 = 0010_2 \times 16^1 + 0101_2 \times 16^0 = 25_{16}$

$371_8 = 3 \times 8^2 + 7 \times 8^1 + 1 \times 8^0 = 249_{10}$

C) $371_8 = 011_2 \times 8^1 + 111_2 \times 8^1 + 001_2 \times 8^0 = 011111001_2$
$371_8 = 011111001_2 = 1111_2 \times 16^1 + 1001_2 \times 16^0 = F\,9_{16}$

$2560_8 = 2 \times 8^3 + 5 \times 8^2 + 6 \times 8^1 + 0 \times 8^0 = 1392_{10}$

D) $2560_8 = 010_2 \times 8^2 + 101_2 \times 8^1 + 110_2 \times 8^1 + 000_2 \times 8^0 = 010101110000_2$
$2560_8 = 011000100111_2 = 0101_2 \times 16^2 + 0111_2 \times 16^1 + 0000_2 \times 16^0 = 570_{16}$

Exercise 1.41

How many 5-bit two's complement numbers are greater than 0? How many are less than 0? How would your answers differ for sign/magnitude numbers?

Solution

There are 15 numbers greater than 0 and 16 numbers lower than 0. It's different for sign/magnitude numbers because there are 2 different values to represent the zero, so there are just 15 numbers above and below from 0.

Exercise 1.42

How many 7-bit two's complement numbers are greater than 0? How many are less than 0? How would your answers differ for sign/magnitude numbers?

Solution

There are 63 numbers greater than 0 and 64 numbers lower than 0. It's different for sign/magnitude numbers because there are 2 different values to represent the zero, so there are just 63 numbers above and below from 0.

Exercise 1.43

How many bytes are in a 32-bit word? How many nibbles are in the 32-bit word?

Solution

There are 4 bytes in 32 bits and there are 8 nibbles as well.

Exercise 1.44

How many bytes are in a 64-bit word?

Solution

There are 8 bytes in 64 bits.

Exercise 1.45

A particular DSL modem operates at 768 kbits/sec. How many bytes can it receive in 1 minute?

Solution

$768\,Kbits/sec \times 60\,sec \times (1/8\,bytes/bits) = 5.76\,Megabytes$

Exercise 1.46

USB 3.0 can send data at 5 Gbits/sec. How many bytes can it send in 1 minute?

Solution

$5\,Gbits/sec \times 60\,sec \times (1/8\,bytes/bits) = 37.5\,Gigabytes$

Exercise 1.47

Hard drive manufactures use the term “megabyte” to mean 10^6 bytes and “gigabyte” to mean 10^9 bytes. How many real GBs (i.e., GiBs) of music can you store on a 50 GB hard drive?

Solution

The amount of GiB that you could store on that hard drive disk is $50 \times 10^9 / 2^{30} = 46.56\,GiB$.

Exercise 1.48

Estimate the value of 2^{31} without using a calculator. Million

Solution

$2^{31} = 2 \times 2^{30} \approx 2 \times 10^9 \approx 2\,Millions$

Exercise 1.49

A memory of the Pentium II microprocessor is organized as a rectangular array of bits with 2^8 rows and 2^9 columns. Estimate how many bits it has without using a calculator.

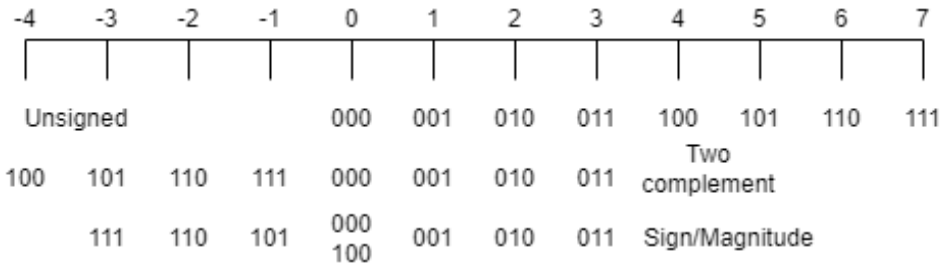
Solution

$2^8 \times 2^9 = 2^{17} = 2^{-3} \times 2^{20} \approx 1000000/8 = 125000\,bits$

Exercise 1.50

Draw a number line for 3-bit unsigned, two’s complement and sign/magnitude numbers.

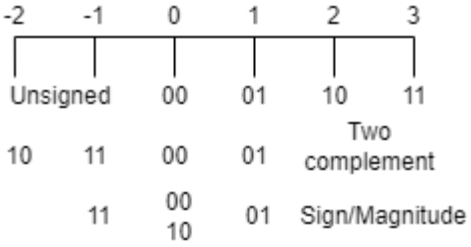
Solution



Exercise 1.51

Draw a number line for 2-bit unsigned, two's complement and sign/magnitude numbers.

Solution



Exercise 1.52

Perform the following additions of unsigned binary numbers. Indicate whether the sum overflows a 4-bit result.

A) $1001_2 + 0100_2$

B) $1101_2 + 1011_2$

Solution

A) $1001_2 + 0100_2 = 1101_2$

B) $1101_2 + 1011_2 = 1000_2$ Overflows

Exercise 1.53

Perform the following additions of unsigned binary numbers. Indicate whether the sum overflows a 8 bit result.

A) $10011001_2 + 01000100_2$

B) $11010010_2 + 10110110_2$

Solution

A) $10011001_2 + 01000100_2 = 11011101_2$

B) $11010010_2 + 10110110_2 = 10001000_2$ Overflows

Exercise 1.54

Repeat Exercise 1.52, assuming that the binary numbers are in two's complement form.

Solution

A) $1001_2 + 0100_2 = 1101_2$

B) $1101_2 + 1011_2 = 1000_2$

Exercise 1.55

Repeat Exercise 1.53, assuming that the binary numbers are in two's complement form.

Solution

A) $10011001_2 + 01000100_2 = 11011101_2$.
B) $11010010_2 + 10110110_2 = 10001000_2$

Exercise 1.56

Convert the following decimal numbers to 6 bit two's complement binary numbers and add them. Indicate whether the sum overflows a 6-bit result.

A) $16_{10} + 9_{10}$
B) $27_{10} + 31_{10}$
C) $-4_{10} + 19_{10}$
D) $3_{10} + -32_{10}$
E) $-16_{10} + -9_{10}$
F) $-27_{10} + -31_{10}$

Solution

A) $16_{10} + 9_{10} = 010000_2 + 001001_2 = 011001_2$
B) $27_{10} + 31_{10} = 011011_2 + 011111_2 = 111010_2$ Overflows
C) $-4_{10} + 19_{10} = -000100_2 + 010011_2 = 111100_2 + 010011_2 = 001111_2$
D) $3_{10} + -32_{10} = 000011_2 + -100000_2 = 000011_2 + 100000_2 = 100011_2$
E) $-16_{10} + -9_{10} = -010000_2 + -001001_2 = 110000_2 + 110111_2 = 100111_2$
F) $-27_{10} + -31_{10} = -011011_2 + -011111_2 = 100101_2 + 100001_2 = 000110_2$ Overflows

Exercise 1.57

Convert the following decimal numbers to 6 bit two's complement binary numbers and add them. Indicate whether the sum overflows a 6-bit result.

A) $7_{10} + 13_{10}$
B) $17_{10} + 25_{10}$
C) $-26_{10} + 8_{10}$
D) $31_{10} + -14_{10}$
E) $-19_{10} + -22_{10}$
F) $-2_{10} + -29_{10}$

Solution

A) $7_{10} + 13_{10} = 000111_2 + 001101_2 = 010100_2$
B) $17_{10} + 25_{10} = 010001_2 + 011001_2 = 101010_2$ Overflows
C) $-26_{10} + 8_{10} = -011010_2 + 001000_2 = 100110_2 + 001000_2 = 101110_2$
D) $31_{10} + -14_{10} = 011111_2 + -001110_2 = 011111_2 + 110010_2 = 010001_2$
E) $-19_{10} + -22_{10} = -010011_2 + -010110_2 = 101101_2 + 101010_2 = 010111_2$ Overflows
F) $-2_{10} + -29_{10} = -000010_2 + -011101_2 = 111110_2 + 100011_2 = 100001_2$

Exercise 1.58

Perform the following additions of unsigned hexadecimal numbers. Indicate whether the sum overflows an 8-bit (two hex digit) result.

A) $7_{16}+9_{16}$
B) $13_{16}+28_{16}$
C) $AB_{16}+3\,E_{16}$
D) $8\,F_{16}+A\,D_{16}$

Solution

A) $7_{16}+9_{16}=00000111_2+00001001_2=00010000_2=10_{16}$
B) $13_{16}+28_{16}=00010011_2+00101000_2=00111011_2=3\,B_{16}$
C) $AB_{16}+3\,E_{16}=10101011_2+00111110_2=11101001_2=E\,9_{16}$
D) $8\,F_{16}+AD_{16}=10001111_2+10101101_2=00111100_2=3\,C_{16}$ Overflows

Exercise 1.59

Perform the following additions of unsigned hexadecimal numbers. Indicate whether the sum overflows an 8-bit (two hex digit) result.

A) $22_{16}+8_{16}$
B) $73_{16}+2\,C_{16}$
C) $7\,F_{16}+7\,F_{16}$
D) $C\,2_{16}+A\,4_{16}$

Solution

A) $22_{16}+8_{16}=00100010_2+00001000_2=00101010_2=2\,A_{16}$
B) $73_{16}+2\,C_{16}=01110011_2+00101100_2=10011111_2=9\,F_{16}$
C) $7\,F_{16}+7\,F_{16}=01111111_2+01111111_2=11111110_2=FE_{16}$
D) $C\,2_{16}+A\,4_{16}=11000010_2+10100100_2=01100110_2=66_{16}$ Overflows

Exercise 1.60

Convert the following decimal numbers to 5 bit two's complement binary numbers and subtract them. Indicate whether the difference overflows a 5-bit result.

A) $9_{10}-7_{10}$
B) $12_{10}-15_{10}$
C) $-6_{10}-11_{10}$
D) $4_{10}--8_{10}$

Solution

A) $9_{10}-7_{10}=01001_2-00111_2=01001_2+11001_2=00010_2$
B) $12_{10}-15_{10}=01100_2-01111_2=01100_2+10001_2=11101_2$
C) $-6_{10}-11_{10}=-00110_2-01011_2=11010_2+10101_2=01111_2$ Overflows
D) $4_{10}--8_{10}=00100_2--01000_2=00100_2+01000_2=01100_2$

Exercise 1.61

Convert the following decimal numbers to 6 bit two's complement binary numbers and subtract them. Indicate whether the difference overflows a 6-bit result.

A) $18_{10} - 12_{10}$
B) $30_{10} - 9_{10}$
C) $-28_{10} - 3_{10}$
D) $-16_{10} - 21_{10}$

Solution

A) $18_{10} - 12_{10} = 010010_2 - 001100_2 = 010010_2 + 110100_2 = 000110_2$
B) $30_{10} - 9_{10} = 011110_2 - 001001_2 = 011110_2 + 110111_2 = 010101_2$
C) $-28_{10} - 3_{10} = -011100_2 - 000011_2 = 100100_2 + 111101_2 = 100001_2$
D) $-16_{10} - 21_{10} = -010000_2 - 010101_2 = 110000_2 + 101011_2 = 011011_2$ Overflows

Exercise 1.62

In a biassed N bit binary number system with bias B, positive and negative numbers are represented as their value plus the bias B. For example, for 5-bit numbers with a bias of 15, the number 0 is represented as 01111, 1 as 10000 and so forth. Biassed numbers systems are sometimes used in floating point mathematics, which will be discussed in chapter 5. Consider a biassed 8-bit binary numbers system with a bias of 127_{10}

A) What decimal value does the binary number 10000010_2 represent?
B) What binary number represents the value 0?
C) What is the representation and value of the most negative number?
D) What is the representation and value of the most positive number?

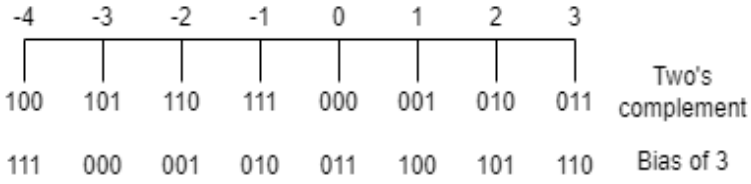
Solution

A) 3_{10}
B) 01111111_2
C) 01000000_2
D) 10111111_2

Exercise 1.63

Draw a number line for 3-bit biassed numbers with a bias of 3.

Solution



Exercise 1.64

In a binary coded decimal (BCD) system, 4 bits are used to represent a decimal digit from 0 to 9. For example, 37_{10} is written as 00110111_{BCD}.

A) Write 289_{10} in BCD
B) Convert 100101010001_{BCD} to decimal
C) Convert 01101001_{BCD} to binary
D) Explain why BCD might be a useful way to represent numbers

Solution

A) $289_{10} = 001010001001_{BCD}$
B) $100101010001_{BCD} = 951_{10}$
C) $01101001_{BCD} = 69_{10} = 01000101_{2}$
D) Because BCD use binary system but it maintain the binary lenght independent of the number so even if we discard any number greater than 9 we can write decimal numbers really easy from binary.

Exercise 1.65

Answer the following questions related to BCD systems.

A) Write 371_{10} in BCD
B) Convert 000110000111_{BCD} to decimal
C) Convert 10010101_{BCD} to binary
D) Explain the disadvantages of BCD when compared with binary representation of numbers

Solution

A) $371_{10} = 001101110001_{BCD}$
B) $000110000111_{BCD} = 187_{10}$
C) $10010101_{BCD} = 95_{10} = 01011111_{2}$
D) The disadvantage is the unused values of the binary system, so it needs more memory that a binary number could need to store the same information

Exercise 1.66

A flying saucer crashes in a Nebraska cornfield. The FBI investigates the wreckage and finds an engineering manual containing an equation in the Martian number system: 325+42=411. If this equation is correct, how many fingers would you expect Martians to have?

Solution

In order to know we need to check the sum of the last significant digit (5 from 425 and 2 from 42), if the last significant digit of the sum is less than these previous numbers (1 from 411) it means there is a carry, so that means that 7+2=11 on that number system, so that means that the numbers have a base system of 6.

Exercise 1.67

Ben Bitdiddle and Alyssa P. Hacker are having an argument. Ben says, “All integers greater than zero and exactly divisible by six have exactly two 1’s in their binary representation.” Alyssa disagrees. She says, “No, but all such numbers have an even number of 1’s in their representation.” Do you agree with Ben or Alyssa or both or neither? Explain.

Solution

The best way to solve this is by contradiction. $42_{10} = 101010_{2}$ so none of them are right.

Exercise 1.68

Ben Bitdiddle and Alyssa P. Hacker are having another argument. Ben says, “I can get the two’s complement of a number by subtracting 1, then inverting all the bits of the result.” Alyssa says, “No, I can do it by examining each bit of the number, starting with the least significant bit. When the first 1 is found, invert each subsequent bit.” Do you agree with Ben or Alyssa or both or neither? Explain.

Solution

Both of them are right. Ben do the inverse operations to do a two's complement so if he do a inverse function he gets the inverse by definition. Alyssa are right too because she uses a property of two's complement where from the less significant bit the two's complement left intact the rest of the string.

Exercise 1.69

Write a program in your favourite language to convert numbers from binary to decimal. The user should type in an unsigned binary number. The program should print the decimal equivalent.

Solution

```
def BinaryToDecimal(A):
    n=0
    for i in range (0,len(A)):
        n=n+int(A[i])*2**(len(A)-i-1)
    return n

A=str(input("Give a binary number: "))
n=BinaryToDecimal(A)
print(n)
```

Exercise 1.70

Repeat Exercise 1.69 but convert from an arbitrary base b_1 to another base b_2, as specified by the user. Support bases up to 16, using letters of the alphabet for digits greater than 9. The user should enter b_1, b_2 and then the number to convert in base b_1. The program should print the equivalent number in base b_2.

Solution

```
def ConvertBaseNumber(InitNumber, InitBase, FinalBase):
    NumSym = ('0', '1', '2', '3', '4', '5', '6', '7',
              '8', '9', 'A', 'B', 'C', 'D', 'E', 'F')
    Number=tuple(InitNumber)
    decimal=0
    for i in range (0,len(Number)):
        decimal+=NumSym.index(Number[i])*InitBase**(len(Number)-i-1)
    Number=""
    while (decimal>0):
        Number= NumSym[decimal%FinalBase] + Number
        decimal=int(decimal/FinalBase)
    return Number

InitBase=int(input("Give a base for a number system (2-16): "))
InitNumber=str(input("Give the number on that number system: "))
FinalBase=int(input("Give the base to convert (2-16): "))
FinalNumber=ConvertBaseNumber(InitNumber, InitBase, FinalBase)
print(FinalNumber)
```

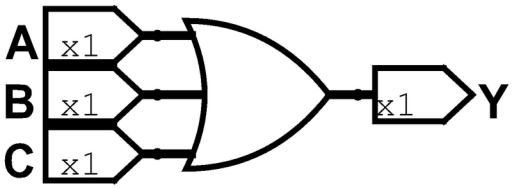
Exercise 1.71

Draw the symbol, Boolean equation and truth table for

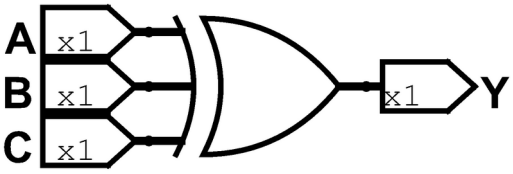
A) A three input OR gate
B) A three input exclusive OR (XOR) gate
C) A four input XNOR gate

Solution

A) A three input OR gate

Equation	Truth table			
$Y = A + B + C$	A	B	C	Y
Symbol	1	1	1	1
	1	1	0	1
	1	0	1	1
	1	0	0	1
	0	1	1	1
	0	1	0	1
	0	0	1	1
	0	0	0	0

B) A three input exclusive OR (XOR) gate

Equation	Truth table			
$Y = A \oplus B \oplus C$	A	B	C	Y
Symbol	1	1	1	1
	1	1	0	0
	1	0	1	0
	1	0	0	1
	0	1	1	0
	0	1	0	1
	0	0	1	1
	0	0	0	0

C) A four input XNOR gate

Equation	Truth table				
$Y = \overline{A \oplus B \oplus C \oplus D}$	A	B	C	D	Y
Symbol	1	1	1	1	1
	1	1	1	0	0
	1	1	0	1	0
	1	1	0	0	1
	1	0	1	1	0

	1	0	1	0	1
	1	0	0	1	1
	1	0	0	0	0
	0	1	1	1	0
	0	1	1	0	1
	0	1	0	1	1
	0	1	0	0	0
	0	0	1	1	1
	0	0	1	0	0
	0	0	0	1	0
	0	0	0	0	1

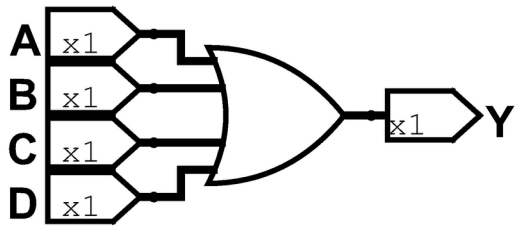
Exercise 1.72

Draw the symbol, Boolean equation and truth table for

A) A four input OR gate
B) A three input exclusive XNOR gate
C) A four input NAND gate

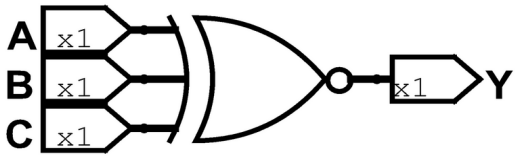
Solution

A) A four input OR gate

Equation	Truth table				
$Y = A + B + C + D$	A	B	C	D	Y
Symbol	1	1	1	1	1
	1	1	1	0	1
	1	1	0	1	1
	1	1	0	0	1
	…	…	…	…	1
	0	0	1	0	1
	0	0	0	1	1
	0	0	0	0	0

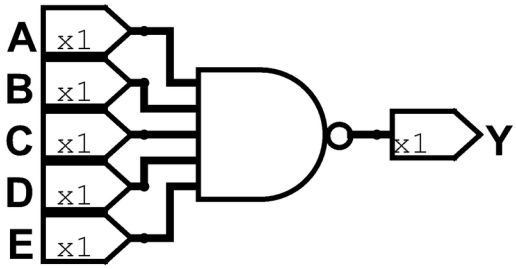
B) A three input exclusive XNOR gate

Equation	Truth table			
$Y = \overline{A \oplus B \oplus C}$	A	B	C	Y
Symbol	1	1	1	0



1	1	0	1
1	0	1	1
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	0
0	0	0	1

C) A four input NAND gate

Equation	Truth table					
$Y = \overline{A \cdot B \cdot C \cdot D \cdot E}$	A	B	C	D	E	Y
Symbol	1	1	1	1	1	1
	1	1	1	1	0	0
	1	1	1	0	1	0
	1	1	1	0	0	0
	...	...	...	...	...	0
	0	0	0	1	0	0
	0	0	0	0	1	0
	0	0	0	0	0	0

Exercise 1.73

A majority gate produces a TRUE output if and only if more than half of its input are TRUE. Complete a truth table for a three input majority gate shown in Figure 1.42

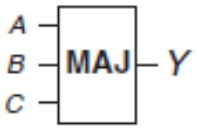


Figure 1.42 Three-input majority gate

Solution

A	B	C	Y
1	1	1	1
1	1	0	1
1	0	1	1
1	0	0	0

0	1	1	1
0	1	0	0
0	0	1	0
0	0	0	0

Exercise 1.74

A three input AND OR (AO) gate shown in Figure 1.43 produces a TRUE output if both A and B are TRUE or if C is TRUE. Complete a truth table for the gate.

Figure 1.43 Three-input AND-OR gate

Solution

A	B	C	Y
1	1	1	1
1	1	0	1
1	0	1	1
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	1
0	0	0	0

Exercise 1.75

A three input OR AND INVERT (OAI) gate shown in Figure 1.44 produces FALSE output if C is TRUE and A or B is TRUE. Otherwise, it produces a TRUE output. Complete a truth table for the gate.

Figure 1.44 Three-input OR-AND-INVERT gate

Solution

A	B	C	Y
1	1	1	0

1	1	0	1
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	1
0	0	1	1
0	0	0	1

Exercise 1.76

There are 16 different truth tables for Boolean function of two variables. List each truth table. Give each one a short descriptive name (such as OR, NAND and so on).

Solution

A	B	AND	OR	XOR	NAND	NOR	XNOR
1	1	1	1	0	0	0	1
1	0	0	1	1	1	0	0
0	1	0	1	1	1	0	0
0	0	0	0	0	1	1	1

Exercise 1.77

How many different truth tables exist for Boolean functions of N variables?

Solution

If all the boolean function is defined by the whole different output that outcome from N inputs variables then the number of combinations is 2^{2^N}.

Exercise 1.78

Is it possible to assign logic levels so that a device with the transfer characteristics shown in Figure 1.45 would serve as an inverter? If so, what are the input and output low and high levels (V_{IL}, V_{OL}, V_{IH} and V_{OH}) and noise margins (NM_L and NM_H)? If not, explain why not.

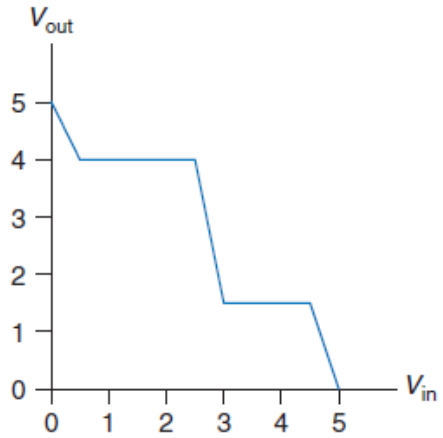


Figure 1.45 DC transfer characteristics

Solution

Here we had 3 point of interest were the slope of the current changes. The first transition happen when V_{IN} pass from 0V to 0.33V approx, V_{OUT} drops from V_{DD} to V_{OH} which is 4V. From here V_{OUT} voltage still remain while V_{IN} are between 0.33V and 2.5V. Here lies an gain point and V_{OUT} drops from 4V to 1.5V so we can assure that V_{IL} start on 2.5V. V_{OUT} voltage will still at 1.5V when V_{IN} are between 3V and 4.5V. Then other gain point appears so we know that V_{OL} is 1.5V and V_{IH} is 4.5V. With all set then NM_L is equal to 2.5V-1.5V=1V and NM_H is equal to 4V-4.5V=-0.5V. Now because the noise margin is violated then it isn't possible to assign logic levels to this device.

Exercise 1.79

Repeat Exercise 1.78 for the transfer characteristics shown in Figure 1.46.

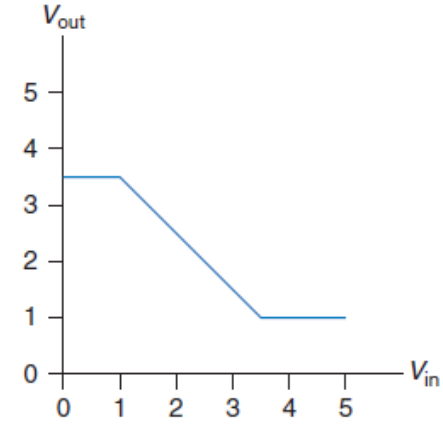


Figure 1.46 DC transfer characteristics

Solution

The first gain point happen when V_{IN} are between 0V to 1V. So apparently V_{DD} and V_{OH} are the same which is 3.5V and V_{IL} is equal to 1V. Then voltage still dropping until V_{IN} reaches 3.5V when it stabilize in 1V. So V_{OL} is equal to 1V and V_{IH} is equal to 3.5V. With all set then NM_L is equal to 1V-1V=0V and NM_H is equal to 3.5V-3.5V=0V. This devices work as a perfect inverter because the states transitions are seamless.

Exercise 1.80

Is it possible to assign logic levels so that a device with the transfer characteristics shown in Figure 1.47 would serve as a buffer? If so, what are the input and output low and high levels (V_{IL}, V_{OL}, V_{IH} and V_{OH}) and noise margins (NM_L and NM_H)? If not, explain why not.

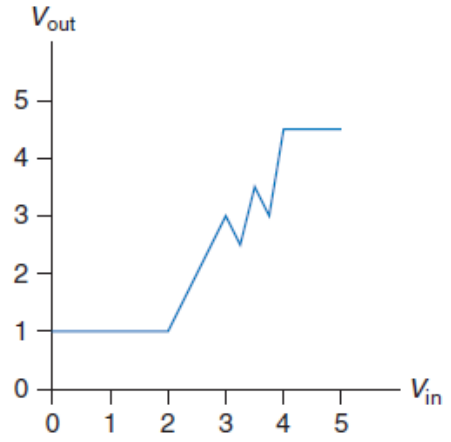


Figure 1.47 DC transfer characteristics

Solution

No. it isn't possible. Transitions happen inside the forbidden zone, any value inside the forbidden zone can not produce gain points.

Exercise 1.81

Ben Bitdiddle has invented a circuit with the transfer characteristics shown in Figure 1.48 that he would like to use as a buffer. Will it work? Why or why not? He would like to advertise that it is compatible with LVCMOS and LVTTL logic. Can its output properly drive those logic families? Explain.

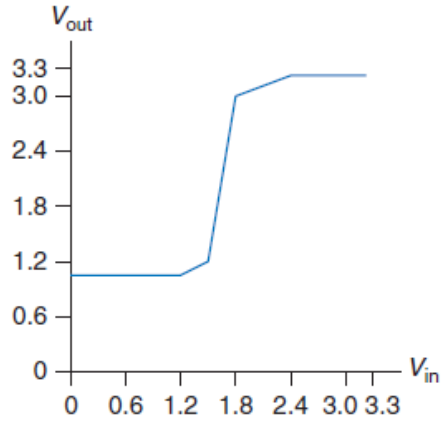


Figure 1.48 Ben's buffer DC transfer characteristics

Solution

Starting with the first gain point V_{OL} is 1.2V and V_{IL} is 1.5V approx. Then on the second gain point V_{OH} is 3V and V_{IH} is equal to 1.8V. With all set then NM_L is equal to 1.5V-1.2V=0.3V and NM_H is equal to

3V-1.8V=1.2V. At first this devices accomplish logical device behaviour. It can receive from LVCMOS and LVTTL however it cannot drive logical values to them.

Exercise 1.82

While walking down a dark alley, Ben Bitdiddle encounters a two input gate with the transfer function shown in Figure 1.49. The inputs are A and B and the output is Y.

A) What kind of logic gate did he find?
B) What are the approximate high and low logic levels?

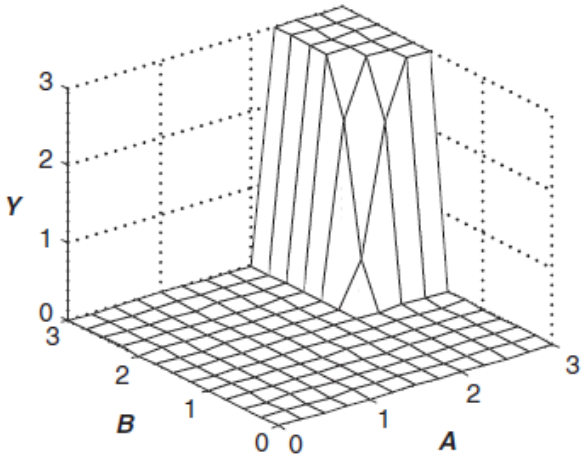


Figure 1.49 Two-input DC transfer characteristics

Solution

A) He found an AND gate because B and A need to be in high state to Y be as well
B) A LOW state is between 0 to 2 volts and a HIGH state is between 2..25 to 3 volts. For B LOW state is between 0 to 1.5 volts and a HIGH state is between 1.75 to 3 volts.

Exercise 1.83

Repeat Exercise 1.82 for Figure 1.50.

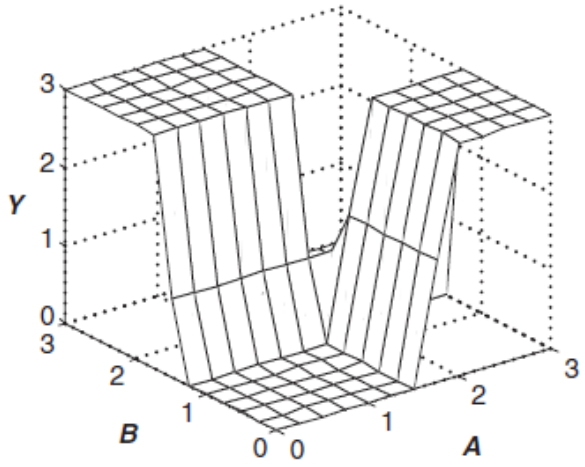


Figure 1.50 Two-input DC transfer characteristics

Solution

A) He found an XOR gate because the output only is TRUE, just one of the gates is TRUE.
B) A LOW state is between 0 to 1.5 volts and a HIGH state is between 2 to 3 volts. For B LOW state is between 0 to 1.25 volts and a HIGH state is between 1.75 to 3 volts.

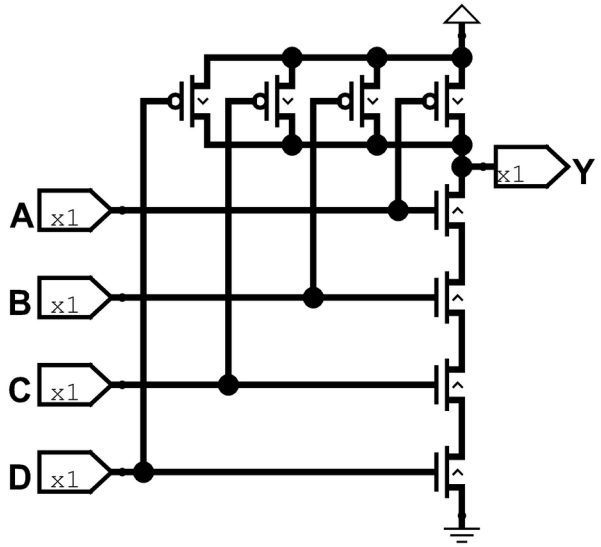
Exercise 1.84

Sketch a transistor level circuit for the following CMOS gates. Use a minimum number of transistors

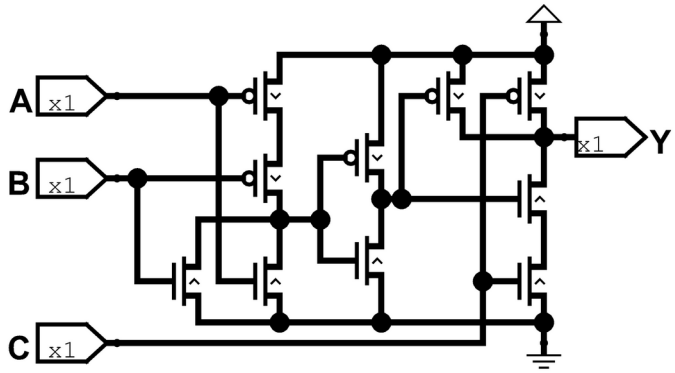
A) Four input NAND gate
B) Three input OR-AND-INVERT gate (see Exercise 1.75)
C) Three input AND-OR gate (see Exercise 1.74)

Solution

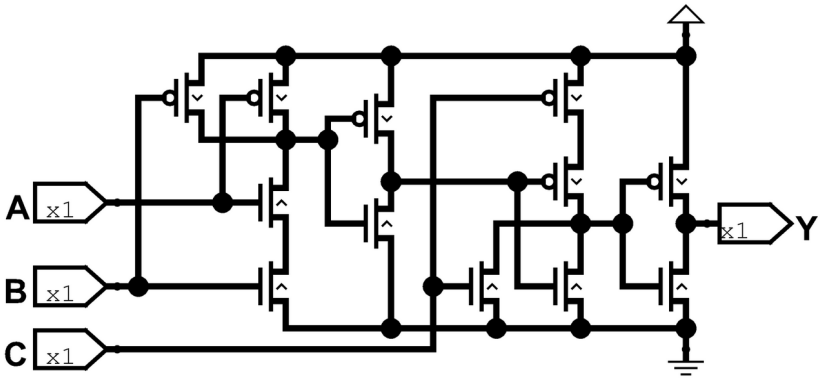
A) Four input NAND gate



B) Three input OR-AND-INVERT gate



C) Three input AND-OR gate



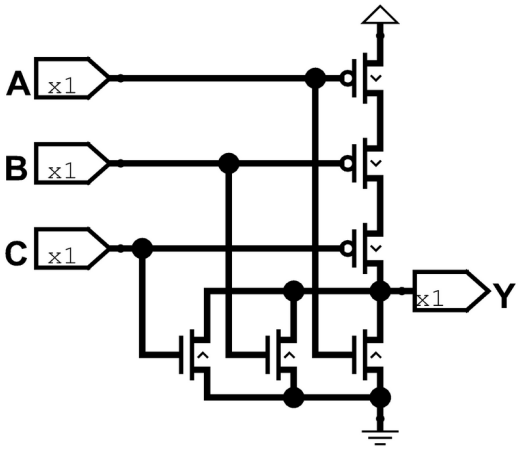
Exercise 1.85

Sketch a transistor level circuit for the following CMOS gates. Use a minimum number of transistors.

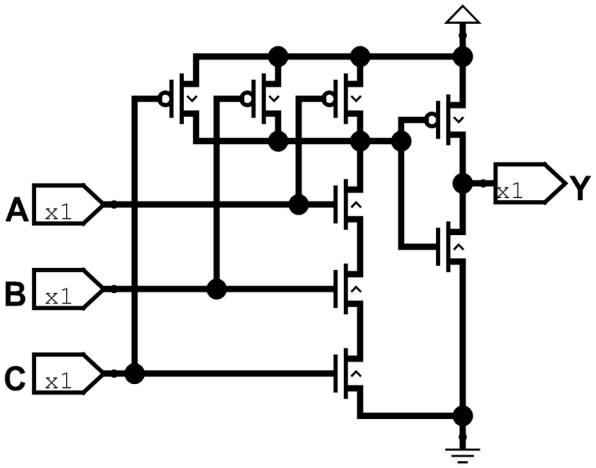
A) Three input NOR gate
B) Three input AND gate
C) Two input OR gate

Solution

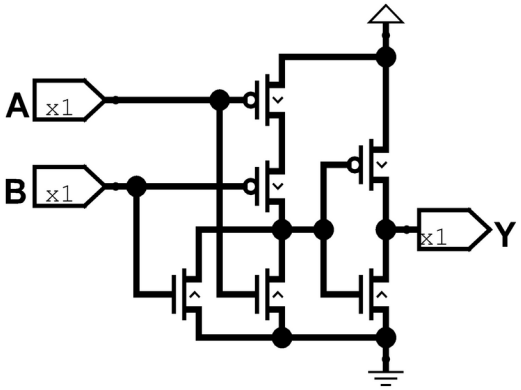
A) Three input NOR gate



B) Three input AND gate



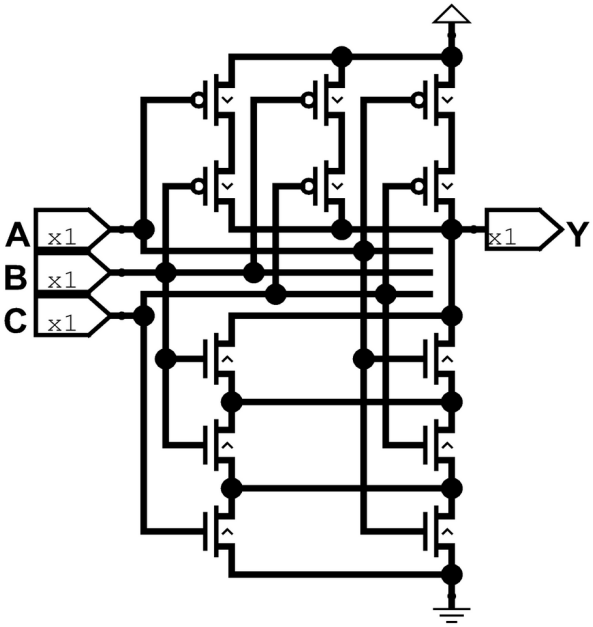
C) Two input OR gate



Exercise 1.86

A minority gate produces a TRUE output if and only if fewer than half of its inputs are TRUE. Otherwise, it produces a False output. Sketch a transistor level circuit for a three input CMOS minority gate. Use a minimum number of transistors.

Solution



Exercise 1.87

Write a truth table for the function performed by the gate in Figure 1.51. Truth table should have two inputs, A and B. What is the name of this function?

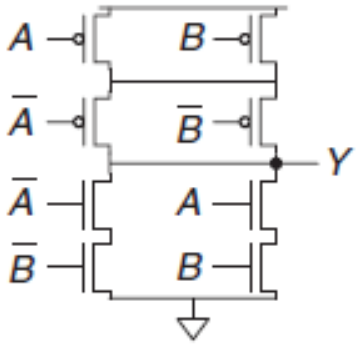


Figure 1.51 Mystery schematic

Solution

To know the values of this logical gate it is convenient to split the analysis in two, from the pull down and pull up perspectives for later merge together to get the whole picture. From pull down the gate throws LOW when neither A and B or inverted A and inverted B because it connects the ground. From the pull up the entries by default are inverted so we get HIGH when inverted A or Inverted B and double Inverted A which is A or double inverted B which is B. So if we merge the result of both tables we notice that is the behaviour of a XOR gate.

A	B	Y
1	1	0
1	0	1

0	1	1
0	0	0

Exercise 1.88

Write a truth table for the function performed by the gate in the figure. The truth table should have three inputs, A, B and C.

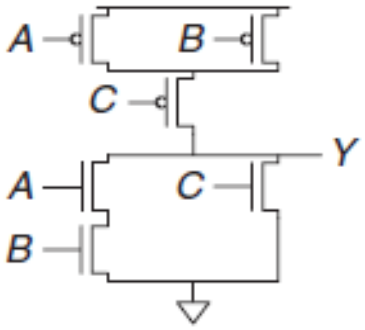


Figure 1.52 Mystery schematic

Solution

We do the same, split the process of pull up and pull down. From pull down we assure the LOW when A and B or C. We continue with a pull up perspective. As before I mentioned the entries for pull up are inverted so in order to get HIGH value we need to assure inverted A and inverted B or inverted C. If we merge the result we get

A	B	C	Y
1	1	1	0
1	1	0	0
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	1
0	0	1	0
0	0	0	1

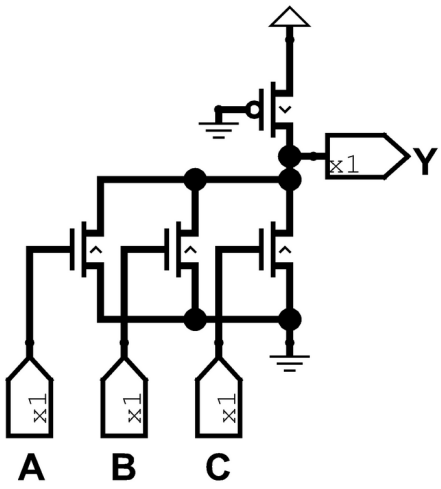
Exercise 1.89

Implement the following three-input gates using only pseudo nMOS logic gates. Your gates receive three inputs A, B and C. Use the minimum number of transistors.

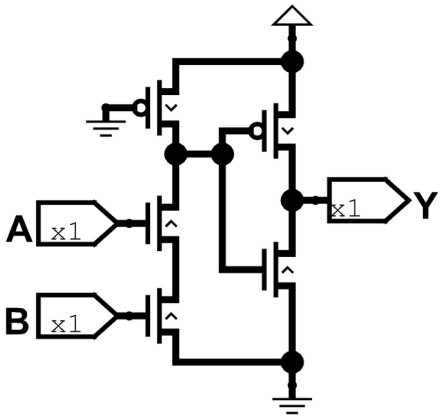
A) Three input NOR gate
B) Three input NAND gate
C) Three input AND gate

Solution

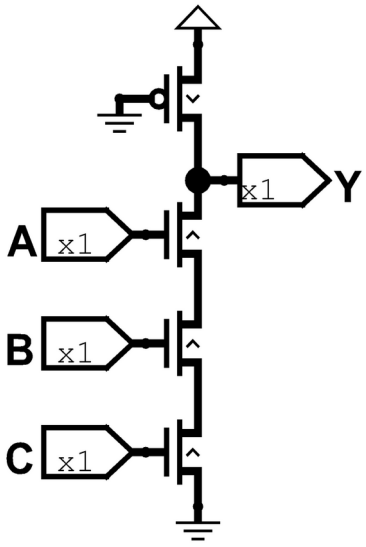
A) Three input NOR gate



B) Three input NAND gate



C) Three input AND gate



Exercise 1.90

Resistor Transistor Logic (RTL) uses nMOS transistors to pull the gate output LOW and weak a resistor to pull the output HIGH when none of the paths to ground are active. A NOT gate built using RTL is shown in Figure 1.53. Sketch a three input RTL NOR gate. Use a minimum number of transistors.

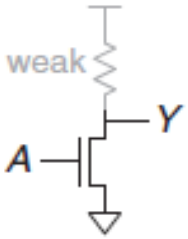
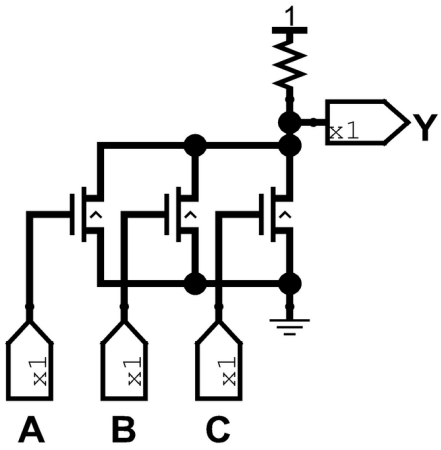


Figure 1.53 RTL NOT gate

Solution



Chapter 2

Exercise 2.1

Write a Boolean equation in sum of products canonical form for each of the truth table in Figure 2.80

(a)

A	B	Y
0	0	1
0	1	0
1	0	1
1	1	1

(b)

A	B	C	Y
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

(c)

A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

(d)

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	1
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	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

(e)

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Figure 2.80 Truth tables for Exercises 2.1, 2.3, and 2.41

Solution

A) $Y=(\overline{A}\cdot\overline{B})+(A\cdot\overline{B})+(A\cdot B)$

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

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

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

E) $Y=(\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D})+(\overline{A}\cdot\overline{B}\cdot C\cdot D)+(\overline{A}\cdot B\cdot\overline{C}\cdot D)+(\overline{A}\cdot B\cdot C\cdot\overline{D})+(A\cdot\overline{B}\cdot\overline{C}\cdot D)$
$+(A\cdot\overline{B}\cdot C\cdot\overline{D})+(A\cdot B\cdot\overline{C}\cdot\overline{D})+(A\cdot B\cdot C\cdot D)$

Exercise 2.2

Write a Boolean equation in sum of products canonical form for each of the truth tables in Figure 2.81

(a)

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

(b)

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

(c)

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

(d)

A	B	C	D	Y
0	0	0	0	1
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	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
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

(e)

A	B	C	D	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
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	0
1	1	1	0	0
1	1	1	1	0

Figure 2.81 Truth tables for Exercises 2.2 and 2.4

Solution

A) $Y=(\overline{A}\cdot B)+(A\cdot\overline{B})+(A\cdot B)$

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

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

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

E) $Y=(\overline{A}\cdot\overline{B}\cdot C\cdot D)+(\overline{A}\cdot B\cdot C\cdot\overline{D})+(\overline{A}\cdot B\cdot C\cdot D)+(A\cdot\overline{B}\cdot\overline{C}\cdot\overline{D})+(A\cdot\overline{B}\cdot\overline{C}\cdot D)$
$+(A\cdot\overline{B}\cdot C\cdot\overline{D})+(A\cdot\overline{B}\cdot C\cdot D)$

Exercise 2.3

Write a Boolean equation in products of sums canonical form for the truth table of the Exercise 2.1

Solution

A) $Y=(\overline{A}+\overline{B})\cdot(A+\overline{B})\cdot(A+B)$

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

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

D) $Y=(\overline{A}+\overline{B}+\overline{C}+\overline{D})\cdot(\overline{A}+\overline{B}+\overline{C}+D)\cdot(\overline{A}+\overline{B}+C+\overline{D})\cdot(\overline{A}+\overline{B}+C+D)\cdot(A+\overline{B}+\overline{C}+\overline{D})$
$\cdot(A+\overline{B}+C+\overline{D})\cdot(A+B+C+\underline{D})$

E) $Y=(\overline{A}+\overline{B}+\overline{C}+\overline{D})\cdot(\overline{A}+\overline{B}+C+D)\cdot(\overline{A}+B+\overline{C}+D)\cdot(\overline{A}+B+C+\overline{D})\cdot(A+\overline{B}+\overline{C}+D)$
$\cdot(A+\overline{B}+C+\overline{D})\cdot(A+B+\overline{C}+\overline{D})\cdot(A+\overline{B}+C+D)$

Exercise 2.4

Write a Boolean equation in products of sums canonical form for the truth table of the Exercise 2.2

Solution

A) $Y=(\overline{A}+B)\cdot(A+\overline{B})\cdot(A+B)$

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

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

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

E) $Y=(\overline{A}+\overline{B}+C+D)\cdot(\overline{A}+B+C+\overline{D})\cdot(\overline{A}+B+C+D)\cdot(A+\overline{B}+\overline{C}+\overline{D})\cdot(A+\overline{B}+\overline{C}+D)$
$\cdot(A+\overline{B}+C+\overline{D})\cdot(A+\overline{B}+C+D)$

Exercise 2.5

Minimise each of the Boolean equations form the Exercise 2.1

Solution

$Y=(\overline{A}\cdot\overline{B})+(A\cdot\underline{B})+(A\cdot B)$

A) $Y=(\overline{A}\cdot\overline{B})+((A\cdot\overline{B})+(A\cdot\overline{B}))+(A\cdot B)$
$Y=\overline{B}\cdot(\overline{A}+A)+A\cdot(B+\overline{B})$
$Y=\overline{B}+A$

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

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

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

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

E)
$$Y=(\overline{A}\cdot\overline{B})\cdot((\overline{C}\cdot\overline{D})+(C\cdot D))+(\overline{A}\cdot B)\cdot((\overline{C}\cdot D)+(C\cdot\overline{D}))+(A\cdot\overline{B})\cdot((\overline{C}\cdot D)+(C\cdot\overline{D}))$$
$$+(A\cdot B)\cdot((\overline{C}\cdot\overline{D})+(C\cdot D))$$
$$Y=(((\overline{A}\cdot\overline{B})+(A\cdot B))\cdot((\overline{C}\cdot\overline{D})+(C\cdot D)))+(((\overline{A}\cdot B)+(A\cdot\overline{B}))\cdot((\overline{C}\cdot D)+(C\cdot\overline{D})))$$
$$Y=\overline{(\overline{A\oplus B})\oplus(\overline{C\oplus D})}$$

Exercise 2.6

Minimise each of the Boolean equations form the Exercise 2.2

Solution

A.
$$Y=(\overline{A}\cdot B)+(A\cdot\overline{B})+(A\cdot B)$$
$$Y=(\overline{A}\cdot B)+(A\cdot\overline{B})+((A\cdot B)+(A\cdot B))$$
$$Y=B\cdot(A+\overline{A})+A\cdot(B+\overline{B})$$
$$Y=A+B$$

B.
$$Y=(\overline{A}\cdot\overline{B}\cdot C)+(\overline{A}\cdot B\cdot\overline{C})+(\overline{A}\cdot B\cdot C)+(A\cdot\overline{B}\cdot\overline{C})+(A\cdot B\cdot\overline{C})$$
$$Y=(\overline{A}\cdot\overline{B}\cdot C)+(\overline{A}\cdot B\cdot\overline{C})+((\overline{A}\cdot B\cdot C)+(\overline{A}\cdot B\cdot C))+(A\cdot\overline{B}\cdot\overline{C})+(A\cdot B\cdot\overline{C})$$
$$Y=(\overline{A}\cdot B)\cdot(C+\overline{C})+(\overline{A}\cdot C)\cdot(B+\overline{B})+(A\cdot\overline{C})\cdot(B+\overline{B})$$
$$Y=(\overline{A}\cdot B)+(\overline{A}\cdot C)+(A\cdot\overline{C})$$

C.
$$Y=(\overline{A}\cdot\overline{B}\cdot C)+(A\cdot B\cdot\overline{C})+(A\cdot B\cdot C)$$
$$Y=(\overline{A}\cdot\overline{B}\cdot C)+(A\cdot B)\cdot(C+\overline{C})$$
$$Y=(\overline{A}\cdot\overline{B}\cdot C)+(A\cdot B)$$

D.
$$Y=(\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D})+(\overline{A}\cdot\overline{B}\cdot C\cdot\overline{D})+(\overline{A}\cdot\overline{B}\cdot C\cdot D)+(\overline{A}\cdot B\cdot C\cdot\overline{D})+(\overline{A}\cdot B\cdot C\cdot D)$$
$$+(A\cdot\overline{B}\cdot\overline{C}\cdot\overline{D})+(A\cdot\overline{B}\cdot C\cdot\overline{D})$$
$$Y=(\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D})+((\overline{A}\cdot\overline{B}\cdot C\cdot\overline{D})+(\overline{A}\cdot\overline{B}\cdot C\cdot\overline{D}))+(\overline{A}\cdot\overline{B}\cdot C\cdot D)+(\overline{A}\cdot B\cdot C\cdot\overline{D})$$
$$+(\overline{A}\cdot B\cdot C\cdot D)+(A\cdot\overline{B}\cdot\overline{C}\cdot\overline{D})+(A\cdot\overline{B}\cdot C\cdot\overline{D})$$
$$Y=(\overline{A}\cdot C)\cdot((\overline{B}\cdot\overline{D})+(\overline{B}\cdot D)+(B\cdot\overline{D})+(B\cdot D))+(\overline{B}\cdot\overline{D})\cdot((\overline{A}\cdot\overline{C})+(\overline{A}\cdot C)+(A\cdot\overline{C})+(A\cdot C))$$
$$Y=(\overline{A}\cdot C)+(\overline{B}\cdot\overline{D})$$

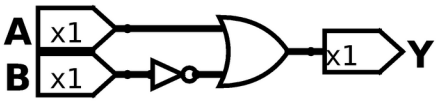
E.
$$Y=(\overline{A}\cdot\overline{B}\cdot C\cdot D)+(\overline{A}\cdot B\cdot C\cdot\overline{D})+(\overline{A}\cdot B\cdot C\cdot D)+(A\cdot\overline{B}\cdot\overline{C}\cdot\overline{D})+(A\cdot\overline{B}\cdot\overline{C}\cdot D)$$
$$+(A\cdot\overline{B}\cdot C\cdot\overline{D})+(A\cdot\overline{B}\cdot C\cdot D)$$
$$Y=(\overline{A}\cdot\overline{B}\cdot C\cdot D)+(\overline{A}\cdot B\cdot C\cdot\overline{D})+((\overline{A}\cdot B\cdot C\cdot D)+(\overline{A}\cdot B\cdot C\cdot D))+(A\cdot\overline{B}\cdot\overline{C}\cdot\overline{D})$$
$$+(A\cdot\overline{B}\cdot\overline{C}\cdot D)+(A\cdot\overline{B}\cdot C\cdot\overline{D})+(A\cdot\overline{B}\cdot C\cdot D)$$
$$Y=(A\cdot\overline{B})\cdot((C\cdot\overline{D})+(C\cdot D)+(\overline{C}\cdot\overline{D})+(\overline{C}\cdot D))+(\overline{A}\cdot C\cdot D)\cdot(B+\overline{B})+(\overline{A}\cdot B\cdot C)\cdot(\overline{D}+D)$$
$$Y=(A\cdot\overline{B})+(\overline{A}\cdot C\cdot D)+(\overline{A}\cdot B\cdot C)$$

Exercise 2.7

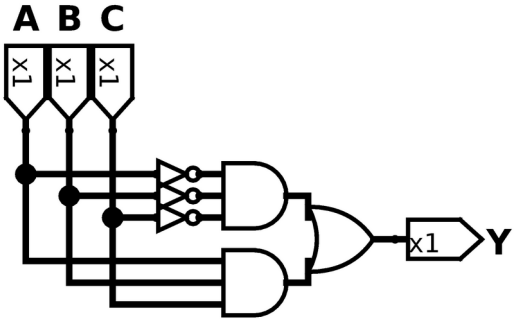
Sketch a reasonably simple combinational circuit implementing each of the functions from Exercise 2.5. Reasonably simple means that you are not wasteful of gates, but you don’t waste vast amounts of time checking every possible implementation of the circuit either.

Solution

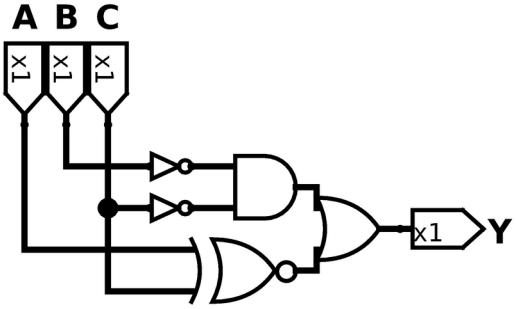
A) $Y=\overline{B}+A$



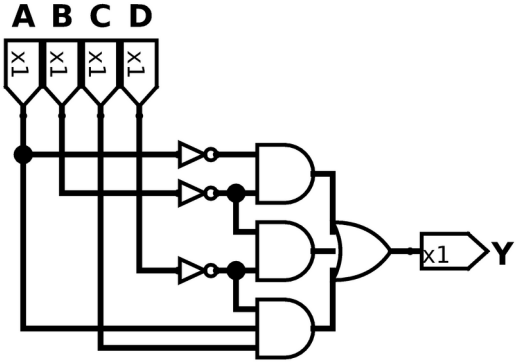
B) $Y=(\overline{A}\cdot\overline{B}\cdot\overline{C})+(A\cdot B\cdot C)$



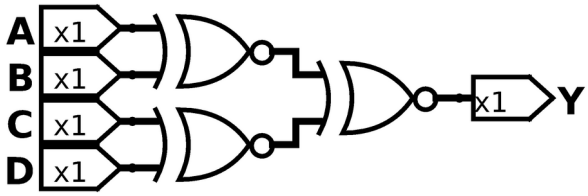
C) $Y=(\overline{B}\cdot\overline{C})+(\overline{A}\cdot\overline{C})+(A\cdot C)$



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



E) $Y=\overline{(\overline{A\oplus B})\oplus(\overline{C\oplus D})}$

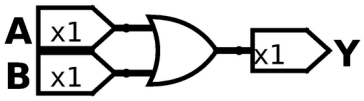


Exercise 2.8

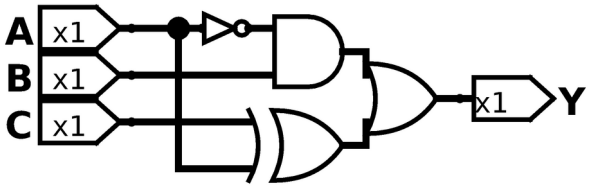
Sketch a reasonably simple combinational circuit implementing each of the functions from Exercise 2.6.

Solution

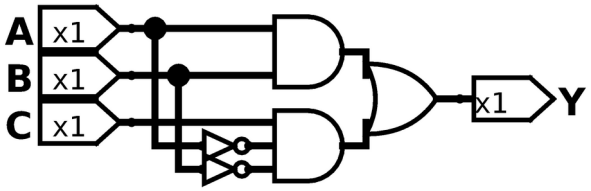
A) $Y = A + B$



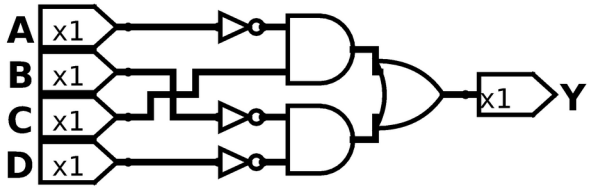
B) $Y = (\overline{A} \cdot B) + (\overline{A} \cdot C) + (A \cdot \overline{C})$



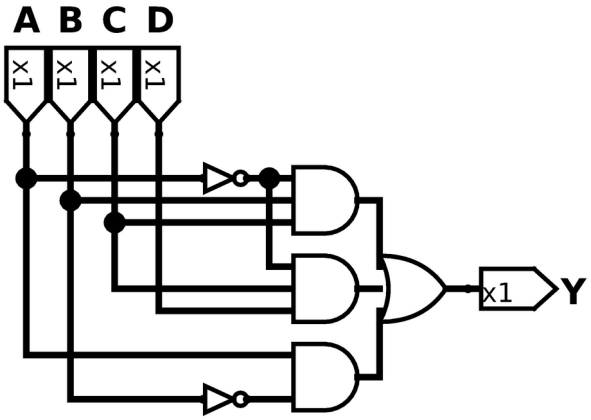
C) $Y = (\overline{A} \cdot \overline{B} \cdot C) + (A \cdot B)$



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



E) $Y = (A \cdot \overline{B}) + (\overline{A} \cdot C \cdot D) + (\overline{A} \cdot B \cdot C)$

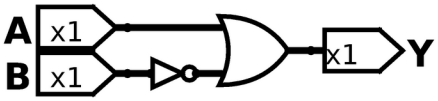


Exercise 2.9

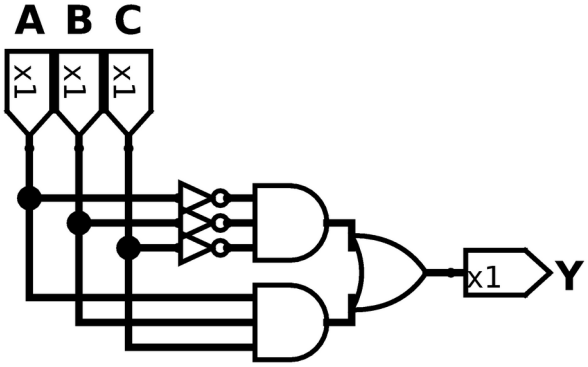
Repeat Exercise 2.7 using only NOT gates and AND gates and OR gates.

Solution

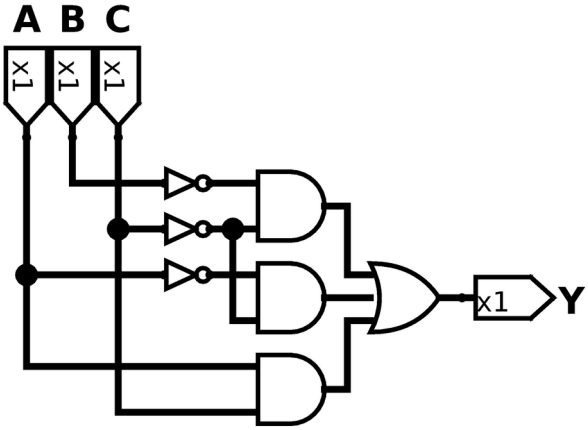
A) $Y = \overline{B} + A$



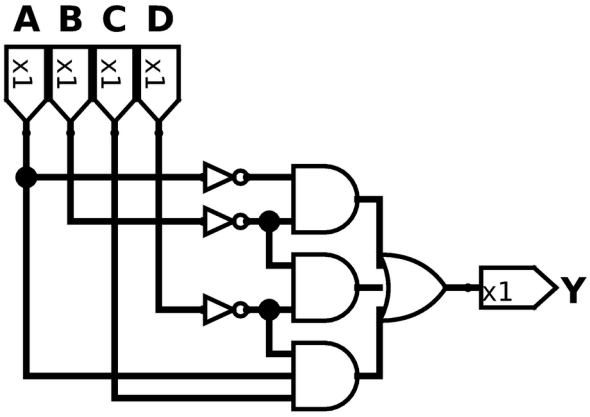
B) $Y = (\overline{A} \cdot \overline{B} \cdot \overline{C}) + (A \cdot B \cdot C)$



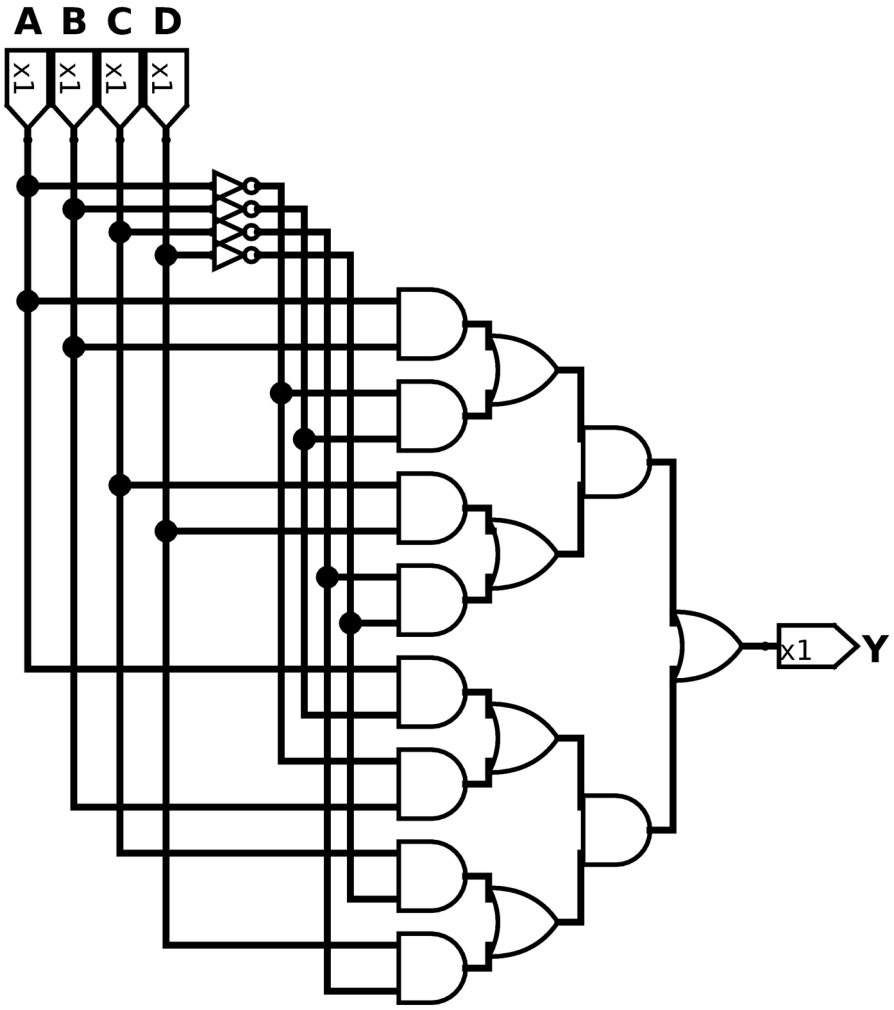
C) $Y = (\overline{B} \cdot \overline{C}) + (\overline{A} \cdot \overline{C}) + (A \cdot C)$



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



E) $Y = \overline{(\overline{A \oplus B}) \oplus (\overline{C \oplus D})}$

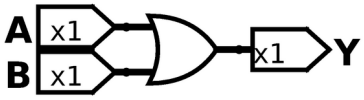


Exercise 2.10

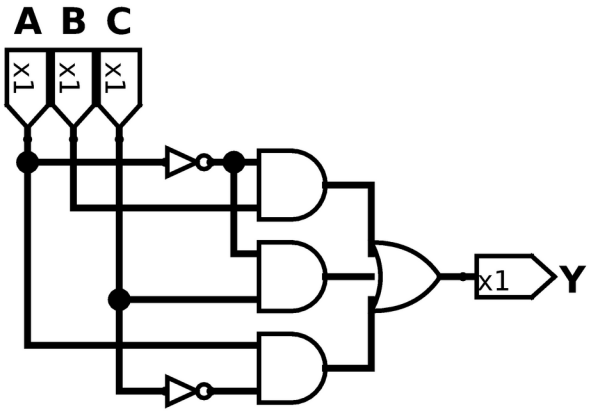
Repeat Exercise 2.8 using only NOT gates and AND gates and OR gates.

Solution

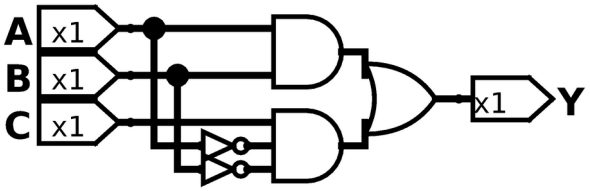
A) $Y = A + B$



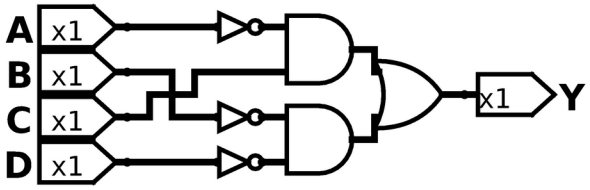
B) $Y = (\overline{A} \cdot B) + (\overline{A} \cdot C) + (A \cdot \overline{C})$



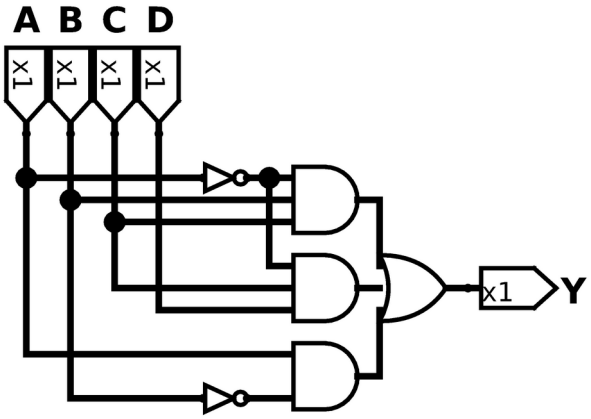
C) $Y=(\overline{A}\cdot\overline{B}\cdot C)+(A\cdot B)$



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



E) $Y=(A\cdot\overline{B})+(\overline{A}\cdot C\cdot D)+(\overline{A}\cdot B\cdot C)$

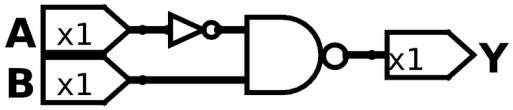


Exercise 2.11

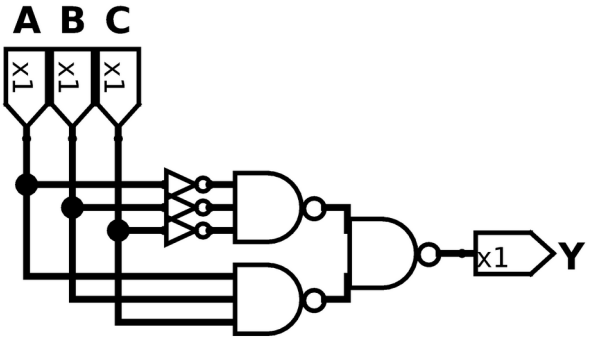
Repeat Exercise 2.7 using only NOT gates and NAND gates and NOR gates.

Solution

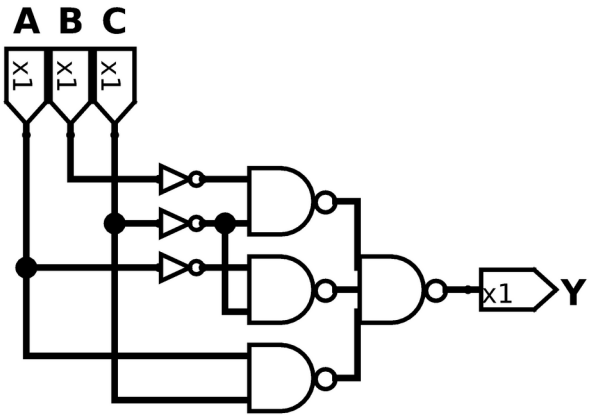
A) $Y=\overline{B}+A$



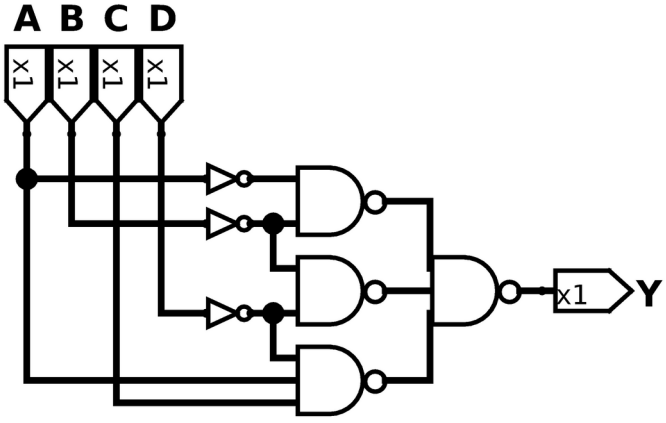
B) $Y=(\overline{A}\cdot\overline{B}\cdot\overline{C})+(A\cdot B\cdot C)$



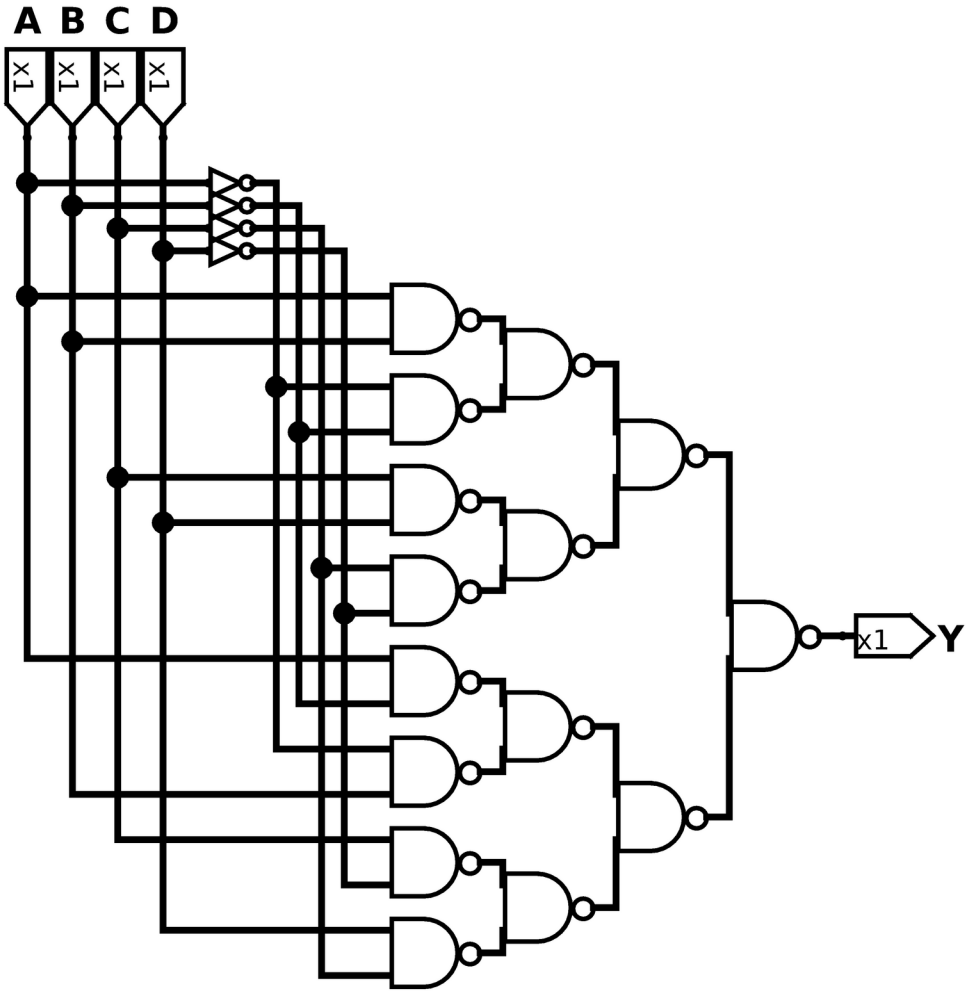
C) $Y=(\overline{B}\cdot\overline{C})+(\overline{A}\cdot\overline{C})+(A\cdot C)$



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



E) $Y=\overline{(\overline{A\oplus B})\oplus(\overline{C\oplus D})}$

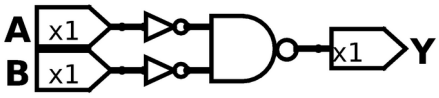


Exercise 2.12

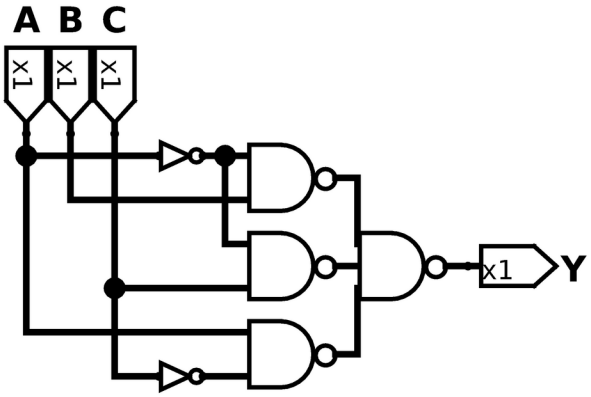
Repeat Exercise 2.8 using only NOT gates and NAND gates and NOR gates.

Solution

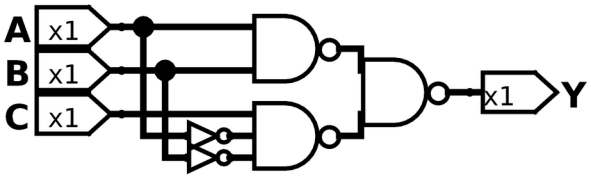
A) $Y = A + B$



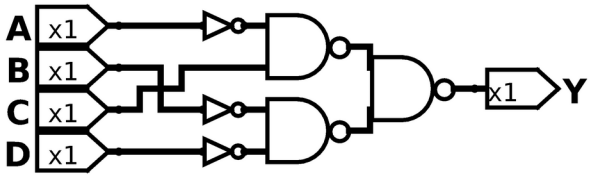
B) $Y = (\overline{A} \cdot B) + (\overline{A} \cdot C) + (A \cdot \overline{C})$



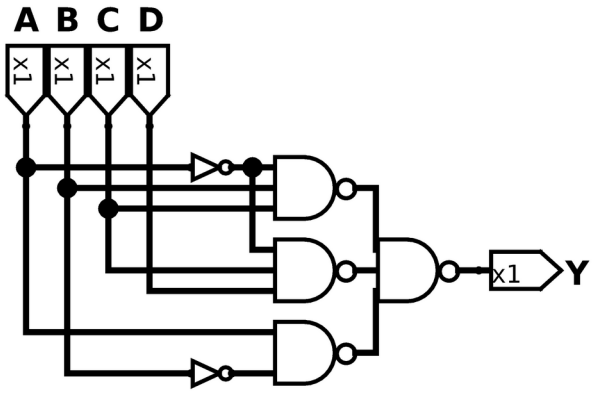
C) $Y=(\overline{A}\cdot\overline{B}\cdot C)+(A\cdot B)$



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



E) $Y=(A\cdot\overline{B})+(\overline{A}\cdot C\cdot D)+(\overline{A}\cdot B\cdot C)$



Exercise 2.13

Simplify the following Boolean equations using Boolean theorems. Check for correctness using a truth table or K-map.

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

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

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

Solution

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

A/BC	00	01	11	10
0	0	1	0	0
1	0	1	1	0

B) $Y = \overline{A}$

A/BC	00	01	11	10
0	1	1	1	1
1	0	0	0	0

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

AB/CD	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	0	1	1	0
10	1	1	0	1

Exercise 2.14

Simplify the following Boolean equations using Boolean theorems. Check for correctness using a truth table or K-map

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

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

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

Solution

A) $Y = \overline{A} \cdot B$

A/BC	00	01	11	10
0	0	0	1	1
1	0	0	0	0

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

A/BC	00	01	11	10
0	1	0	0	0
1	1	1	0	0

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

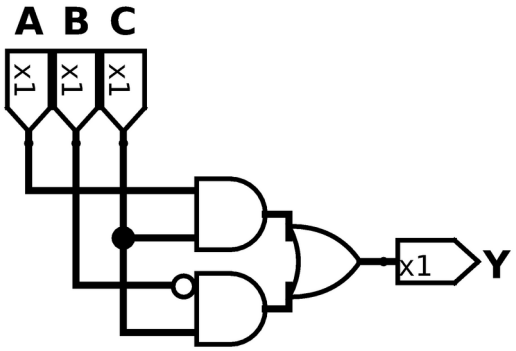
AB/CD	00	01	11	10
00	1	0	0	0
01	0	0	0	0
11	0	0	0	1
10	1	0	0	0

Exercise 2.15

Sketch a reasonably simple combinational circuit implementing each of the functions form the Exercise 2.13.

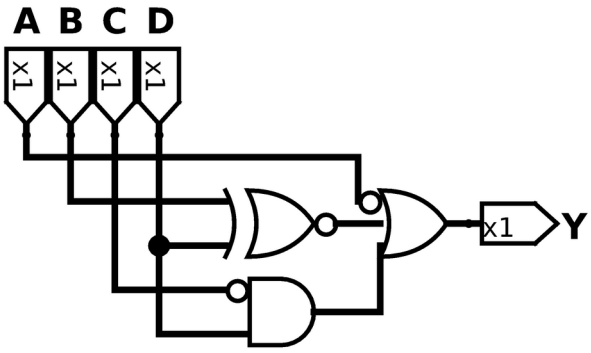
Solution

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



B) $Y=\overline{A}$

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

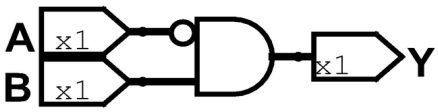


Exercise 2.16

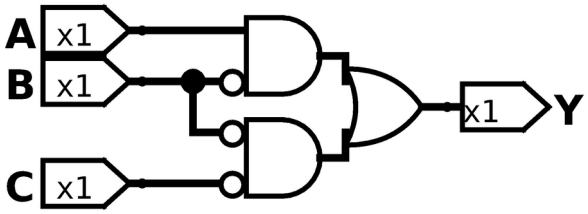
Sketch a reasonably simple combinational circuit implementing each of the functions from Exercise 2.14.

Solution

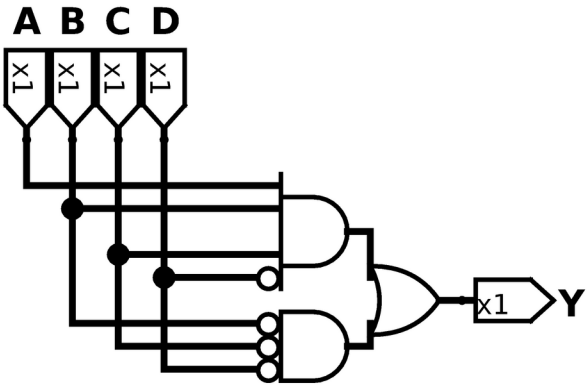
A) $Y = \overline{A} \cdot B$



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



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



Exercise 2.17

Simplify each of the following Boolean equations. Sketch a reasonably simple combinational circuit implementing the simplified equation.

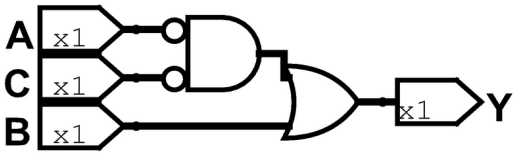
A) $Y = (B \cdot C) + (\overline{A} \cdot \overline{B} \cdot \overline{C}) + (B \cdot \overline{C})$

B) $Y = \overline{A + (\overline{A} \cdot B) + (\overline{A} \cdot \overline{B})} + (\overline{A + \overline{B}})$

C) $Y = (A \cdot B \cdot C) + (A \cdot B \cdot D) + (A \cdot B \cdot E) + (A \cdot C \cdot D) + (A \cdot C \cdot E) + (\overline{A + D + E}) + (\overline{B} \cdot \overline{C} \cdot D) + (\overline{B} \cdot \overline{C} \cdot E) + (\overline{B} \cdot \overline{D} \cdot \overline{E}) + (\overline{C} \cdot \overline{D} \cdot \overline{E})$

Solution

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



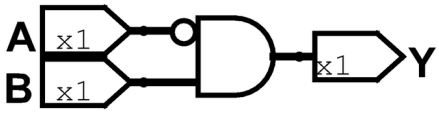
$$Y=\overline{A+(\overline{A}\cdot B)+(\overline{A}\cdot\overline{B})}+(\overline{A+\overline{B}})$$

$$Y=\overline{A+\overline{A}\cdot(B\cdot\overline{B})}+(\overline{A}\cdot B)$$

B) $$Y=\overline{A+\overline{A}}+(\overline{A}\cdot B)$$

$$Y=\overline{1}+(\overline{A}\cdot B)$$

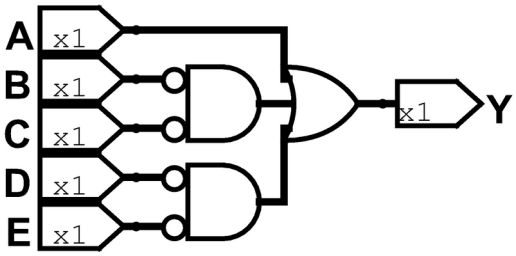
$$Y=\overline{A}\cdot B$$



$$Y=(A\cdot B\cdot C)+(A\cdot B\cdot D)+(A\cdot B\cdot E)+(A\cdot C\cdot D)+(A\cdot C\cdot E)+(\overline{A+D+E})+(\overline{B}\cdot\overline{C}\cdot D)$$

C) $$+(\overline{B}\cdot\overline{C}\cdot E)+(\overline{B}\cdot\overline{D}\cdot\overline{E})+(\overline{C}\cdot\overline{D}\cdot\overline{E})$$

$$Y=A+(\overline{B}\cdot\overline{C})+(\overline{D}\cdot\underline{E})$$



AB/CDE	000	001	011	010	110	111	101	100
00	1	1	1	1	0	0	0	1
01	1	0	0	0	0	0	0	1
11	1	1	1	1	1	1	1	1
10	1	1	1	1	1	1	1	1

Exercise 2.18

Simplify each of the following Boolean equations. Sketch a reasonably simple combinational circuit implementing the simplified equation.

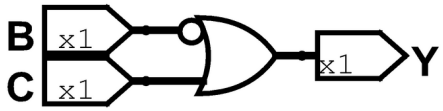
A) $Y=(\overline{A}\cdot B\cdot C)+(\overline{B\cdot\overline{C}})+(B\cdot C)$

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

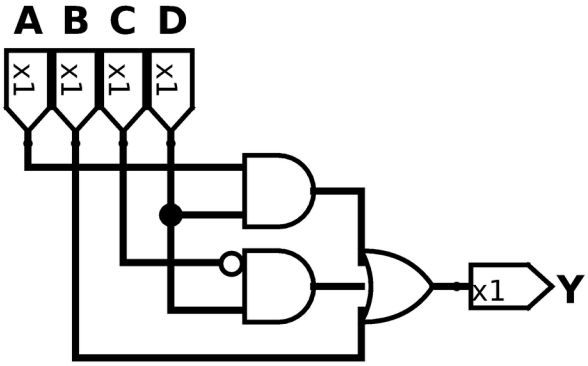
C) $Y=(A\cdot B\cdot C\cdot D)+(\overline{A}\cdot B\cdot\overline{C}\cdot D)+(\overline{\overline{B}+D})\cdot E$

Solution

A) $Y=(\overline{A}\cdot B\cdot C)+(\overline{B\cdot\overline{C}})+(B\cdot C)$
$Y=(\overline{A}\cdot B\cdot C)+(\overline{B}+C)+(B\cdot C)$
$Y=(\overline{B}+C)+((B\cdot C)\cdot(1+\overline{A}))$
$Y=\overline{B}+C+(B\cdot C)$
$Y=\overline{B}+C\cdot(B+1)$
$Y=\overline{B}+C$

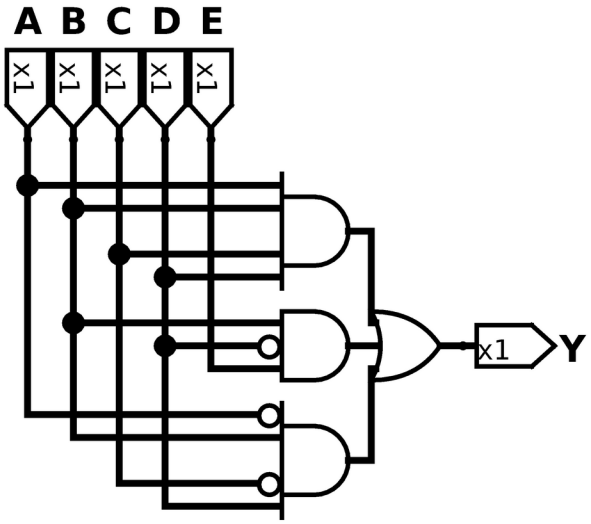


B) $Y=(\overline{A+B+C})\cdot D+(A\cdot D)+B$
$Y=B+(\overline{C}\cdot D)+(A\cdot D)$



AB/CD	00	01	11	10
00	0	1	0	0
01	1	1	1	1
11	1	1	1	1
10	0	1	1	0

C) $Y=(A\cdot B\cdot C\cdot D)+(\overline{A}\cdot B\cdot\overline{C}\cdot D)+(\overline{\overline{B}+D})\cdot E$
$Y=(A\cdot B\cdot C\cdot D)+(\overline{A}\cdot B\cdot\overline{C}\cdot D)+(B\cdot\overline{D}\cdot E)$



Exercise 2.19

Give an example of a truth table requiring between 3 billion and 5 billion rows can be constructed using fewer than 40 (but least 1) two-input gates.

Solution

If we build a tree of logic gates like XORs we get N inputs for every N-1 gates used and we get 2^N possible values (truth table rows). It easily reaches 4 billion when we use 31 gates.

Exercise 2.20

Give an example of a circuit with a cyclic path that is nevertheless combinational.

Solution

The most simple example is connecting an Inverter in cycle, the input and output node are the same but this contradict the gate value.

Exercise 2.21

Alyssa P. Hacker says that any Boolean function can be written in minimal sum of products form as the sum of the prime implicants of the function. Ben Bitdiddle says that there are some functions whose minimal equation does not involve all of the prime implicants. Explain why Alyssa is right or provide a counterexample demonstrating Ben's point.

Solution

Both are right. Any function can be described by a sum of product of every term from the truth table that gives us a TRUE output, however many of these terms share a logic operation that can be reduced. Then it could be simplified and it doesn't need all those prime implicants.

Exercise 2.22

Prove that the following theorems are true using perfect induction. You need not to prove their duals.

A) The idempotency theorem (T3)
B) The distributivity theorem (T8)
C) The combining theorem (T10)

Solution

A) The first theorem says $B \cdot B = B$. To prove it we could define the values of B (0 and 1) and see every scene. When B value is FALSE the operation result is also FALSE, it happens something similar when B is TRUE, the operation is TRUE, then if the result of the operation is the same as the value of the operating then $B \cdot B = B$.
B) The second theorem says $(B \cdot C) + (B \cdot D) = B \cdot (C + D)$. We need to imagine every situation in this equation. If B is FALSE then no matter what value is C or D because it is invalidated by the AND operation. Then if B is true then the operation reduces on only C or D need to be TRUE to the result be as well. Then we can conclude that $(B \cdot C) + (B \cdot D) = B \cdot (C + D)$.
C) The third theorem says $(B \cdot C) + (B \cdot \overline{C}) = B$. To analyse this is better to inspect C first, if C is TRUE then the first operating is TRUE and the result as well, but if FALSE then inverted C is TRUE and then the second operating is true, so the appearance of C or their inverted is meaningless, so the result of this operation depends only of B then $(B \cdot C) + (B \cdot \overline{C}) = B$.

Exercise 2.23

Prove De Morgan's Theorem (T12) for three variables A, B and C using perfect induction.

Solution

This problem, should say (For the specified amount of variables) that $\overline{A + B + C} = \overline{A} \cdot \overline{B} \cdot \overline{C}$. First we start to analyse the left side of the equation. Due to the way that NOR is defined, only need any of these to be TRUE to the result be FALSE. We move next to the right side of the equation. Due to the way that AND is defined, it only needs any of these to be FALSE to the result be FALSE, then if all of the variables are inverted then it only needs any of these to be TRUE to the result be FALSE. But as you can notice we had the same definition of both sides of the equation, then the theorem is true.

Exercise 2.24

Write Boolean equations for the circuit in Figure 2.82. You need not minimise the equations.

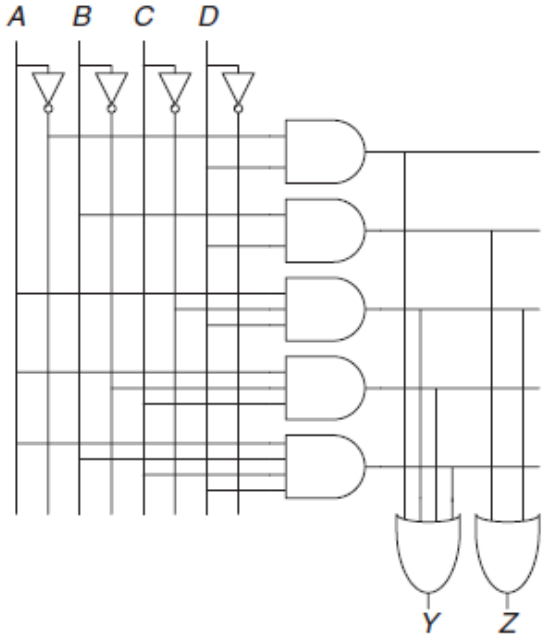


Figure 2.82 Circuit schematic for Exercise 2.24

Solution

$$Y=(\overline{A}\cdot D)+(B\cdot D)+(A\cdot\overline{C}\cdot D)+(A\cdot\overline{B}\cdot C)+(A\cdot B\cdot C\cdot D)$$
$$Z=(B\cdot D)+(A\cdot\overline{C}\cdot D)$$

Exercise 2.25

Minimise the Boolean equations from Exercise 2.24 and sketch an improved circuit with the same function.

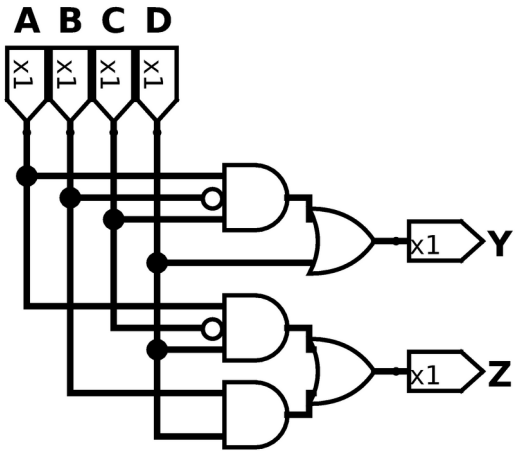
Solution

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

AB/CD	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	1	1	0
10	0	1	1	1

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

AB/CD	00	01	11	10
00	0	0	0	0
01	0	1	1	0
11	0	1	1	0
10	0	1	0	0



Exercise 2.26

Using De Morgan equivalent gates and bubble pushing methods, redraw the circuit in Figure 2.83 so that you can find the Boolean equation by inspection. Write the Boolean equation.

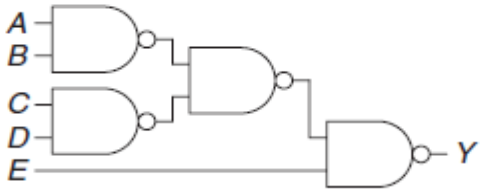
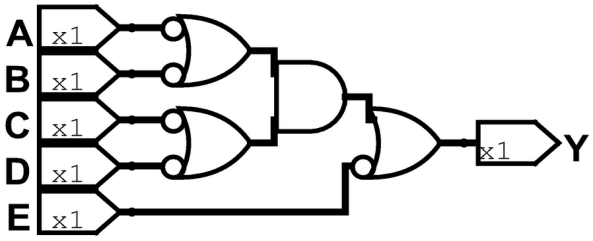


Figure 2.83 Circuit schematic for Exercises 2.26 and 2.43

Solution



Exercise 2.27

Repeat Exercise 2.26 for the circuit in the figure.

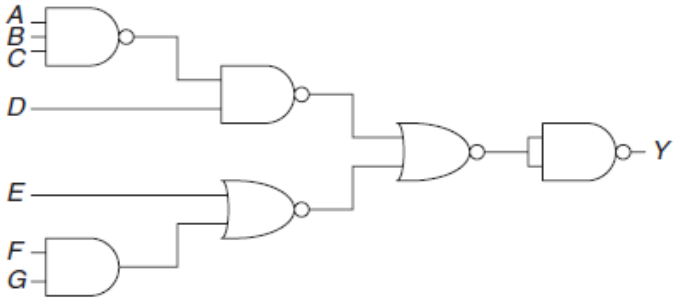
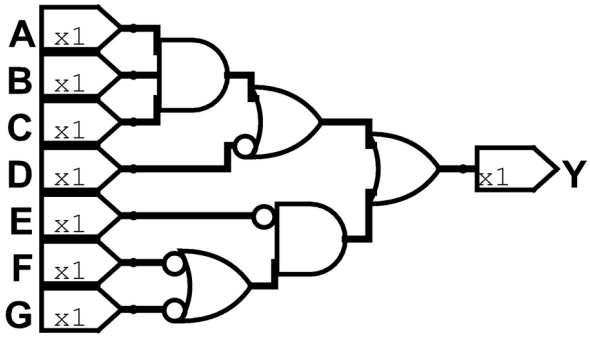


Figure 2.84 Circuit schematic for Exercises 2.27 and 2.44

Solution



Exercise 2.28

Find a minimal Boolean equation for the function in Figure 2.85. Remember to take advantage of don’t care entries.

A	B	C	D	Y
0	0	0	0	X
0	0	0	1	X
0	0	1	0	X
0	0	1	1	0
0	1	0	0	0
0	1	0	1	X
0	1	1	0	0
0	1	1	1	X
1	0	0	0	1
1	0	0	1	0
1	0	1	0	X
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	X
1	1	1	1	1

Figure 2.85 **Truth table for Exercise 2.28**

Solution

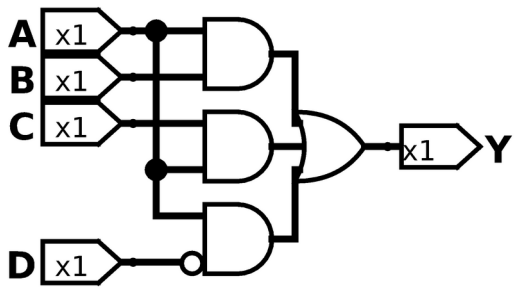
$Y = (A \cdot \overline{D}) + (A \cdot C) + (A \cdot B)$

AB/CD	00	01	11	10
00	X	X	0	X
01	0	X	X	0
11	1	1	1	X
10	1	0	1	X

Exercise 2.29

Sketch a circuit for the function from Exercise 2.28.

Solution



Exercise 2.30

Does your circuit from Exercise 2.29 have any potential glitches when one of the inputs changes? If not, explain why not. If so, show how to modify the circuit to eliminate the glitches.

Solution

There's no glitch and here is why. The expression that could we suspect is $(A \cdot \overline{D})$ because there is a delay because of the NOT gate. However if we take a look at the K-Map from above we can notice that 1100, 1110 and 1010 belongs to the other terms too, then only remain 1000, if we start to analyse we notice that the expression only will be true until both entries are TRUE (A TRUE and D FALSE). But no matter which of those entries start to modify it wont present a glitch.

Exercise 2.31

Find a minimal Boolean equation for the function in Figure 2.86. Remember to take advantage of the don't care entries.

A	B	C	D	Y
0	0	0	0	0
0	0	0	1	1
0	0	1	0	X
0	0	1	1	X
0	1	0	0	0
0	1	0	1	X
0	1	1	0	X
0	1	1	1	X
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	X
1	1	1	1	1

Figure 2.86 Truth table for Exercise 2.31

Solution

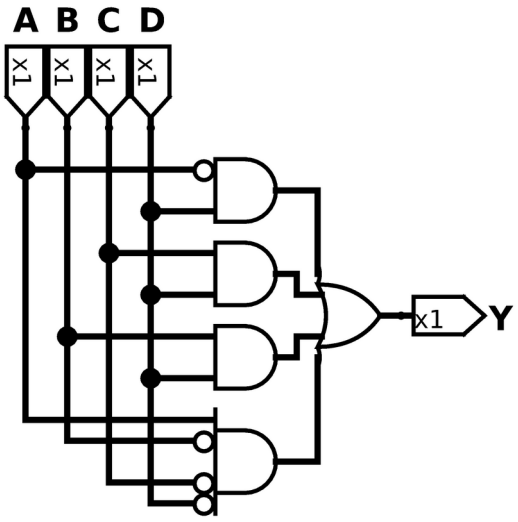
$Y = (\overline{A} \cdot D) + (C \cdot D) + (B \cdot D) + (A \cdot \overline{B} \cdot \overline{C} \cdot \overline{D})$

AB/CD	00	01	11	10
00	0	1	X	X
01	0	X	X	X
11	0	1	1	X
10	1	0	1	0

Exercise 2.32

Sketch a circuit for the function from Exercise 2.31.

Solution



Exercise 2.33

Ben Bitdiddle will enjoy his picnic on sunny days that have no ants. He will also enjoy his picnic any day he sees a hummingbird, as well as on days where there are ants and ladybugs. Write a Boolean equation for his enjoyment (E) in terms of sun (S), ants (A), hummingbirds (H) and ladybugs (L).

Solution

$E = (S \cdot \overline{A}) + H + (A \cdot L)$

Exercise 2.34

A seven segment display decoder takes a 4-bits data input $D_{3:0}$ and produces seven outputs to control light emitting diodes to display a digit from 0 to 9. The seven outputs are often called segment a trough g, as defined in Figure 2.47. Use K-maps to do the follow points, Assume that illegal input values (10-15) produce a blank readout:

A) Derive Boolean equations for the outputs S_A through S_G assuming that inputs greater than 9 must produce blank (0) outputs.
B) Derive Boolean equations for the outputs S_A through S_G assuming that inputs greater than 9 are don't cares.
C) Sketch a reasonably simple gate level implementation of part (B). Multiple outputs can share gates where appropriate.

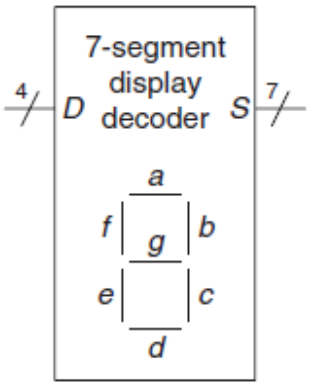


Figure 2.47 Seven-segment display decoder icon

Solution

A) Blank outputs

$$S_A=(\overline{D_3}\cdot D_1)+(D_3\cdot\overline{D_2}\cdot\overline{D_1})+(\overline{D_3}\cdot D_2\cdot D_0)+(\overline{D_3}\cdot\overline{D_2}\cdot\overline{D_0})$$

$D_{3:2}/D_{1:0}$	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	0	0	0	0
10	1	1	0	0

$$S_B=(\overline{D_3}\cdot\overline{D_2})+(\overline{D_2}\cdot\overline{D_1})+(\overline{D_3}\cdot\overline{D_1}\cdot\overline{D_0})+(\overline{D_3}\cdot D_1\cdot D_0)$$

$D_{3:2}/D_{1:0}$	00	01	11	10
00	1	1	1	1
01	1	0	1	0
11	0	0	0	0
10	1	1	0	0

$$S_C=(\overline{D_2}\cdot\overline{D_1})+(\overline{D_3}\cdot D_0)+(\overline{D_3}\cdot D_2)$$

$D_{3:2}/D_{1:0}$	00	01	11	10
00	1	1	1	0
01	1	1	1	1
11	0	0	0	0

10	1	1	0	0

$$S_D = (\overline{D_3} \cdot D_2 \cdot \overline{D_1} \cdot D_0) + (\overline{D_2} \cdot \overline{D_1} \cdot \overline{D_0}) + (\overline{D_3} \cdot \overline{D_2} \cdot D_1) + (\overline{D_3} \cdot D_1 \cdot \overline{D_0}) + (D_3 \cdot \overline{D_2} \cdot \overline{D_1})$$

$D_{3:2}/D_{1:0}$	00	01	11	10
00	1	0	1	1
01	0	1	0	1
11	0	0	0	0
10	1	1	0	0

$$S_E = (\overline{D_2} \cdot \overline{D_1} \cdot \overline{D_0}) + (\overline{D_3} \cdot D_1 \cdot \overline{D_0})$$

$D_{3:2}/D_{1:0}$	00	01	11	10
00	1	0	0	1
01	0	0	0	1
11	0	0	0	0
10	1	0	0	0

$$S_F = (\overline{D_2} \cdot \overline{D_1} \cdot \overline{D_0}) + (\overline{D_3} \cdot D_2 \cdot \overline{D_0}) + (\overline{D_3} \cdot D_2 \cdot \overline{D_1}) + (D_3 \cdot \overline{D_2} \cdot \overline{D_1})$$

$D_{3:2}/D_{1:0}$	00	01	11	10
00	1	0	0	0
01	1	1	0	1
11	0	0	0	0
10	1	1	0	0

$$S_G = (D_3 \cdot \overline{D_2} \cdot \overline{D_1}) + (\overline{D_3} \cdot D_2 \cdot \overline{D_1}) + (\overline{D_3} \cdot \overline{D_2} \cdot D_1) + (\overline{D_3} \cdot D_1 \cdot \overline{D_0})$$

$D_{3:2}/D_{1:0}$	00	01	11	10
00	0	0	1	1
01	1	1	0	1
11	0	0	0	0
10	1	1	0	0

B) Don’t cares

$S_A = D_3 + D_1 + (D_2 \cdot D_0) + (\overline{D_2} \cdot \overline{D_0})$

$D_{3:2}/D_{1:0}$	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	X	X	X	X
10	1	1	X	X

$S_B = \overline{D_2} + (\overline{D_1} \cdot \overline{D_0}) + (D_1 \cdot D_0)$

$D_{3:2}/D_{1:0}$	00	01	11	10
00	1	1	1	1
01	1	0	1	0
11	X	X	X	X
10	1	1	X	X

$S_C = D_2 + \overline{D_1} + D_0$

$D_{3:2}/D_{1:0}$	00	01	11	10
00	1	1	1	0
01	1	1	1	1
11	X	X	X	X
10	1	1	X	X

$S_D = D_3 + (\overline{D_2} \cdot \overline{D_0}) + (\overline{D_2} \cdot D_1) + (D_1 \cdot \overline{D_0}) + (D_2 \cdot \overline{D_1} \cdot D_0)$

$D_{3:2}/D_{1:0}$	00	01	11	10
00	1	0	1	1
01	0	1	0	1
11	X	X	X	X
10	1	1	X	X

$S_E = (\overline{D_2} \cdot \overline{D_0}) + (D_1 \cdot \overline{D_0})$

$D_{3:2}/D_{1:0}$	00	01	11	10

00	1	0	0	1
01	0	0	0	1
11	X	X	X	X
10	1	0	X	X

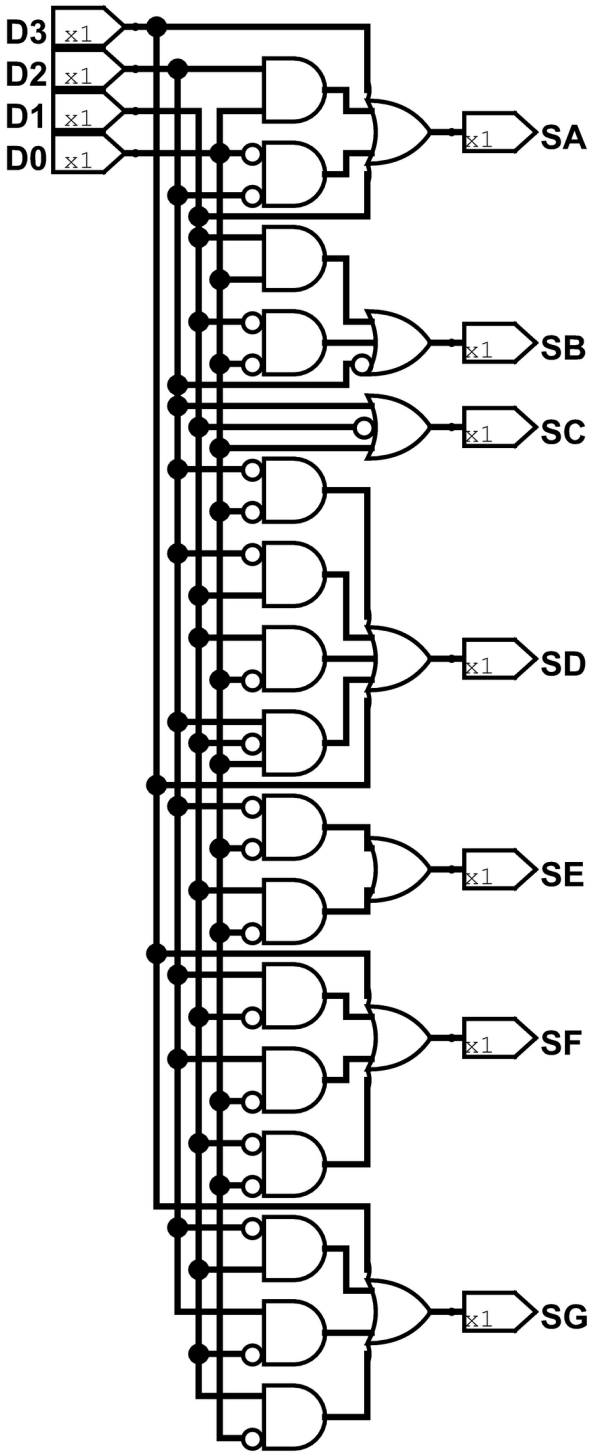
$$S_F = D_3 + (D_2 \cdot \overline{D_1}) + (D_2 \cdot \overline{D_0}) + (\overline{D_1} \cdot \overline{D_0})$$

$D_{3:2}/D_{1:0}$	00	01	11	10
00	1	0	0	0
01	1	1	0	1
11	X	X	X	X
10	1	1	X	X

$$S_G = D_3 + (\overline{D_2} \cdot D_1) + (D_2 \cdot \overline{D_1}) + (D_1 \cdot \overline{D_0})$$

$D_{3:2}/D_{1:0}$	00	01	11	10
00	0	0	1	1
01	1	1	0	1
11	X	X	X	X
10	1	1	X	X

C) Sketch



Exercise 2.35

A circuit has four inputs and two outputs. The $A_{3:0}$ represents a number from 0 to 15. Output P should be TRUE if the number is prime (0 and 1 are not prime, but 2, 3, 5 and so on are prime). Output D should be true if the number is divisible by 3. Give simplified Boolean equations for each output and sketch circuit.

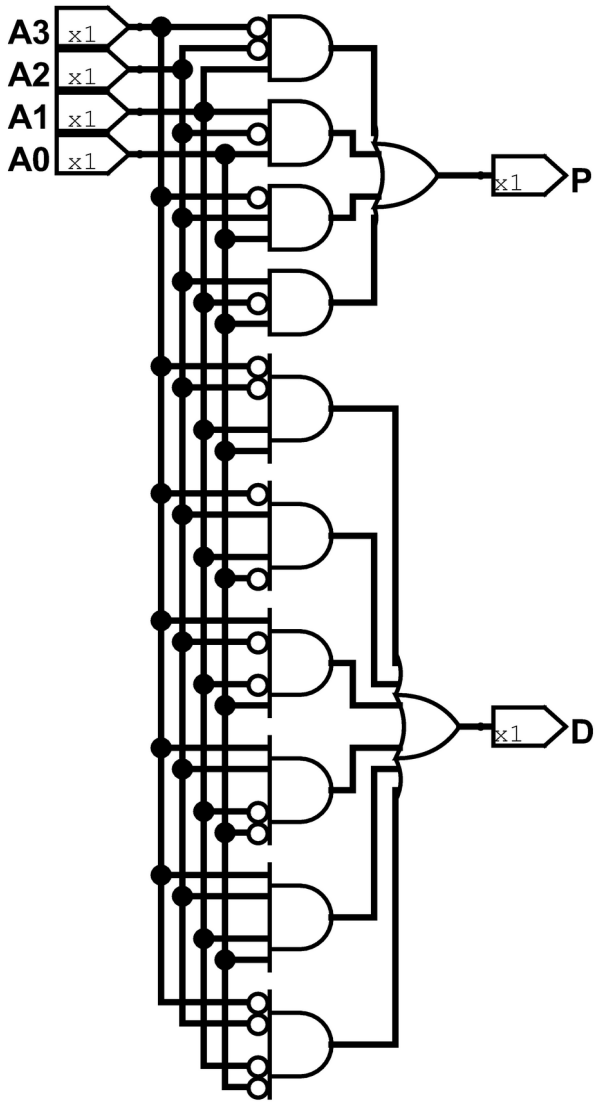
Solution

$$P=(A_2\cdot\overline{A_1}\cdot A_0)+(\overline{A_3}\cdot A_2\cdot A_0)+(\overline{A_2}\cdot A_1\cdot A_0)+(\overline{A_3}\cdot\overline{A_2}\cdot A_1)$$

$A_{3:2}/A_{1:0}$	00	01	11	10
00	0	0	1	1
01	0	1	1	0
11	0	1	0	0
10	0	0	1	0

$$D=(\overline{A_3}\cdot\overline{A_2}\cdot A_1\cdot A_0)+(\overline{A_3}\cdot A_2\cdot A_1\cdot\overline{A_0})+(A_3\cdot\overline{A_2}\cdot\overline{A_1}\cdot A_0)+(A_3\cdot A_2\cdot\overline{A_1}\cdot\overline{A_0})+(A_3\cdot A_2\cdot A_1\cdot A_0)+(\overline{A_3}\cdot\overline{A_2}\cdot\overline{A_1}\cdot\overline{A_0})$$

$A_{3:2}/A_{1:0}$	00	01	11	10
00	1	0	1	0
01	0	0	0	1
11	1	0	1	0
10	0	1	0	0



Exercise 2.36

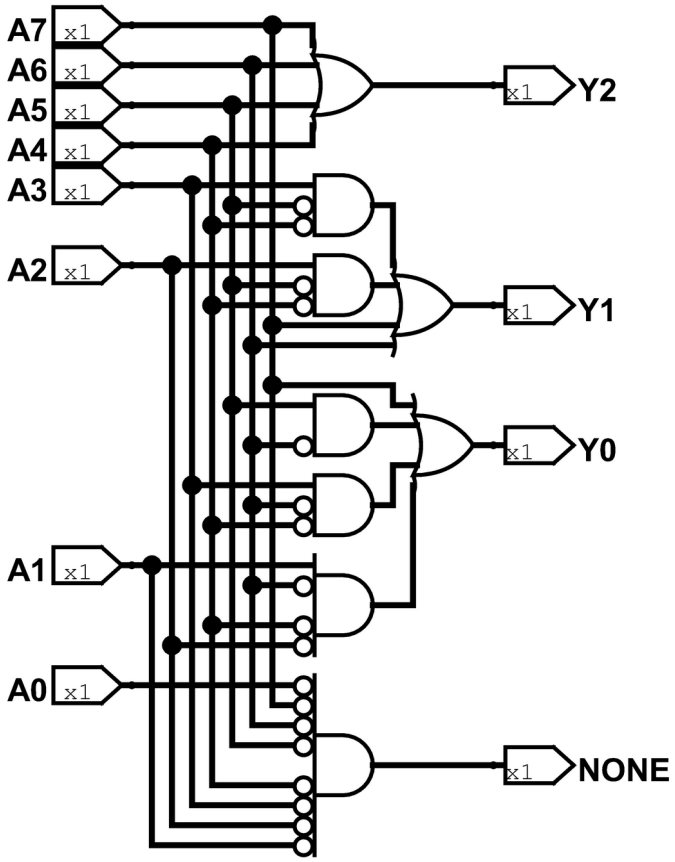
A priority encoder has 2^N inputs. It produces an N bit binary output indicating the most significant bit of the input that is TRUE or 0 if none of the inputs are TRUE. Design an eight input priority encoder with inputs $A_{7:0}$ and outputs $Y_{2:0}$ and NONE. For example, if the input is 00100000, the output Y should be 101 and NONE should be 0. Give a simplified Boolean equation for each output and sketch a schematic.

Solution

$$Y_2 = A_7 + (A_6 \cdot \overline{A_7}) + (A_5 \cdot \overline{A_6} \cdot \overline{A_7}) + (A_4 \cdot \overline{A_5} \cdot \overline{A_6} \cdot \overline{A_7})$$
$$Y_2 = A_7 + A_6 + A_5 + A_4$$
$$Y_1 = A_7 + (A_6 \cdot \overline{A_7}) + (A_3 \cdot \overline{A_4} \cdot \overline{A_5} \cdot \overline{A_6} \cdot \overline{A_7}) + (A_2 \cdot \overline{A_3} \cdot \overline{A_4} \cdot \overline{A_5} \cdot \overline{A_6} \cdot \overline{A_7})$$
$$Y_1 = A_7 + A_6 + (A_2 \cdot \overline{A_4} \cdot \overline{A_5}) + (A_3 \cdot \overline{A_4} \cdot \overline{A_5})$$
$$Y_0 = A_7 + (A_5 \cdot \overline{A_6} \cdot \overline{A_7}) + (A_3 \cdot \overline{A_4} \cdot \overline{A_5} \cdot \overline{A_6} \cdot \overline{A_7}) + (A_1 \cdot \overline{A_2} \cdot \overline{A_3} \cdot \overline{A_4} \cdot \overline{A_5} \cdot \overline{A_6} \cdot \overline{A_7})$$
$$Y_0 = A_7 + (A_5 \cdot \overline{A_6}) + (A_3 \cdot \overline{A_4} \cdot \overline{A_6}) + (A_1 \cdot \overline{A_2} \cdot \overline{A_4} \cdot \overline{A_6})$$

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Y_2	Y_1	Y_0	NONE

1	X	X	X	X	X	X	X	1	1	1	0
0	1	X	X	X	X	X	X	1	1	0	0
0	0	1	X	X	X	X	X	1	0	1	0
0	0	0	1	X	X	X	X	1	0	0	0
0	0	0	0	1	X	X	X	0	1	1	0
0	0	0	0	0	1	X	X	0	1	0	0
0	0	0	0	0	0	1	X	0	0	1	0
0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1



Exercise 2.37

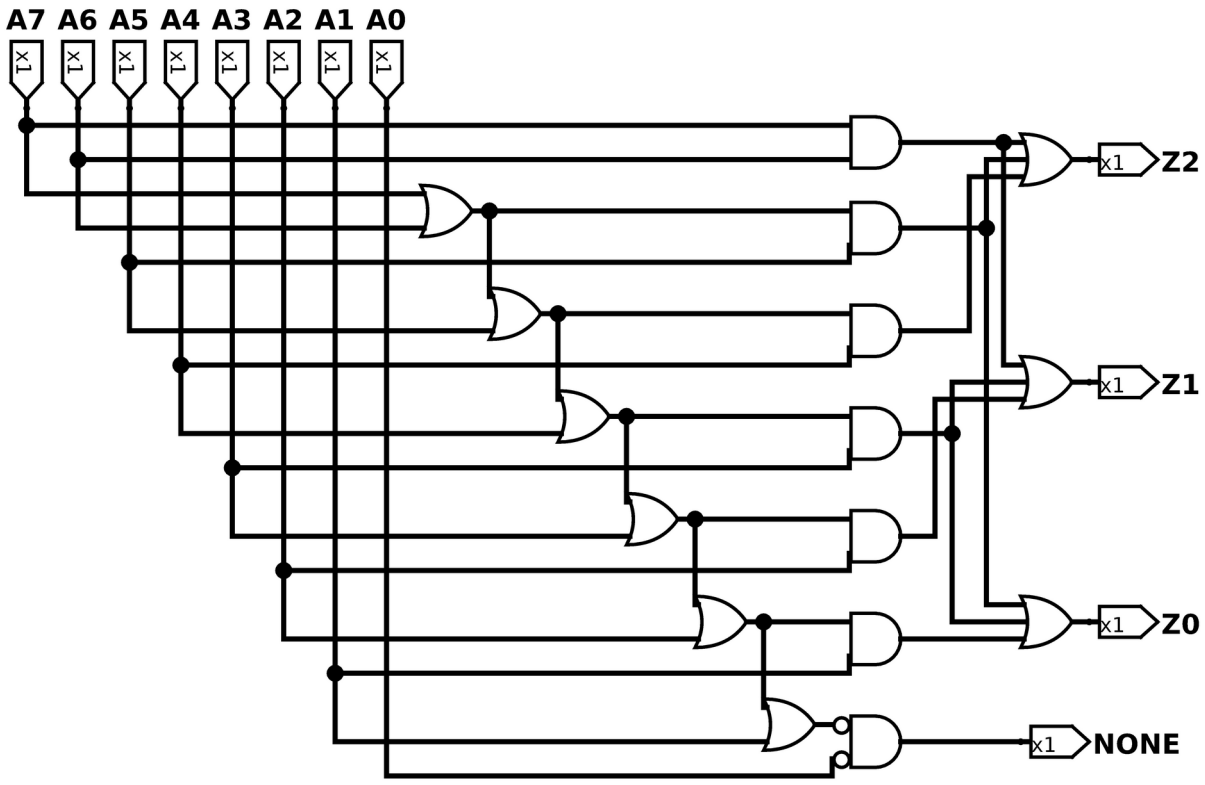
Design a modified priority encoder (see Exercise 2.36) that receives an 8 bit input, $A_{7:0}$ and produces two 3 bit outputs, $Y_{2:0}$ and $Z_{2:0}$. Y indicates the most significant bit of the input that is TRUE. Z indicates the second most significant bit of the input that is TRUE. Y should be 0 if none of the inputs are TRUE. Z should be 0 if no more than one of the inputs is TRUE. Give a simplified Boolean equation for each output and sketch a schematic.

Solution

$$Z_2 = A_7 \cdot A_6 + (A_7 + A_6) \cdot A_5 + (A_7 + A_6 + A_5) \cdot A_4$$

$$Z_1 = A_7 \cdot A_6 + (A_7 + A_6 + A_5 + A_4) \cdot A_3 + (A_7 + A_6 + A_5 + A_4 + A_3) \cdot A_2$$
$$Z_0 = (A_7 + A_6) \cdot A_5 + (A_7 + A_6 + A_5 + A_4) \cdot A_3 + (A_7 + A_6 + A_5 + A_4 + A_3 + A_2) \cdot A_1$$

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Z_2	Z_1	Z_0
1	1	X	X	X	X	X	X	1	1	0
1	0	1	X	X	X	X	X	1	0	1
0	1	1	X	X	X	X	X	1	0	1
1	0	0	1	X	X	X	X	1	0	0
0	1	0	1	X	X	X	X	1	0	0
0	0	1	1	X	X	X	X	1	0	0
1	0	0	0	1	X	X	X	0	1	1
0	1	0	0	1	X	X	X	0	1	1
0	0	1	0	1	X	X	X	0	1	1
0	0	0	1	1	X	X	X	0	1	1
1	0	0	0	0	1	X	X	0	1	0
0	1	0	0	0	1	X	X	0	1	0
0	0	1	0	0	1	X	X	0	1	0
0	0	0	1	0	1	X	X	0	1	0
0	0	0	0	1	1	X	X	0	1	0
1	0	0	0	0	0	1	X	0	0	1
0	1	0	0	0	0	1	X	0	0	1
0	0	1	0	0	0	1	X	0	0	1
0	0	0	1	0	0	1	X	0	0	1
0	0	0	0	1	0	1	X	0	0	1
0	0	0	0	0	1	1	X	0	0	1
1	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	1	0	0	0
0	0	0	0	1	0	0	1	0	0	0
0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	1	1	0	0	0



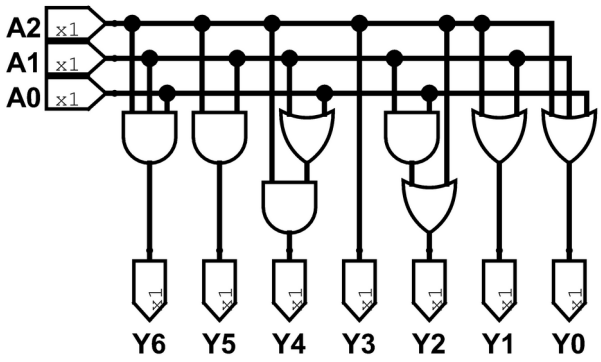
Exercise 2.38

An M bit thermometer code for the number k consist of k 1s in the least significant bit positions and 0s M-K in all the more significant bit positions. A binary to thermometer code converter has N inputs and 2^N-1 output. It produces a 2^N-1 bit thermometer code for the number specified by the input. For example, if the input is 110, the output should be 0111111. Design a 3:7 binary thermometer code converter. Give a simplified Boolean equation for each output and sketch a schematic.

Solution

$Y_6 = A_2 \cdot A_1 \cdot A_0$ $Y_5 = A_2 \cdot A_1$ $Y_4 = A_2 \cdot (A_1 + A_0)$ $Y_3 = A_2$
$Y_2 = A_2 + (A_1 \cdot A_0)$ $Y_1 = A_2 + A_1$ $Y_0 = A_2 + A_1 + A_0$

A_2	A_1	A_0	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
1	1	1	1	1	1	1	1	1	1
1	1	0	0	1	1	1	1	1	1
1	0	1	0	0	1	1	1	1	1
1	0	0	0	0	0	1	1	1	1
0	1	1	0	0	0	0	1	1	1
0	1	0	0	0	0	0	0	1	1
0	0	1	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0



Exercise 2.39

Write a minimised Boolean equation for the function performed by the circuit in Figure 2.87

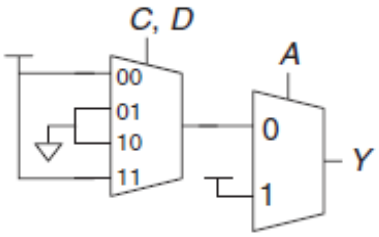


Figure 2.87 Multiplexer circuit for Exercise 2.39

Solution

$Y = (\overline{A} \cdot ((\overline{C} \cdot \overline{D}) + (C \cdot D))) + A$

Exercise 2.40

Write a minimised Boolean equation for the input performed by the circuit in Figure 2.88.

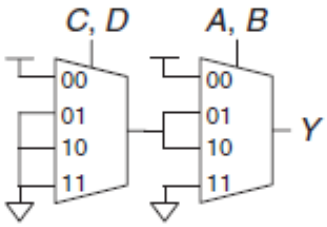


Figure 2.88 Multiplexer circuit for Exercise 2.40

Solution

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

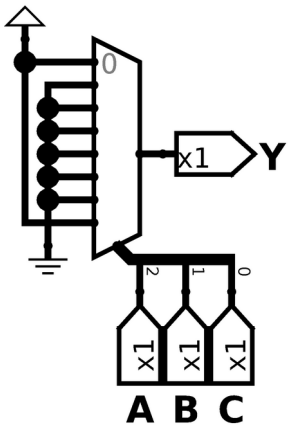
Exercise 2.41

Implement the function from Figure 2.80(B) using

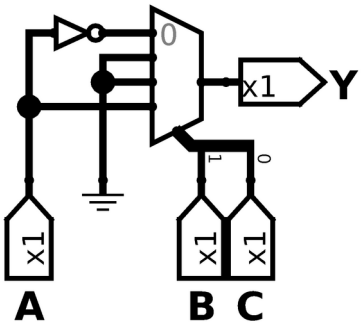
A) An 8:1 multiplexer
B) A 4:1 multiplexer and one inverter
C) A 2:1 multiplexer and two other logic gates

Solution

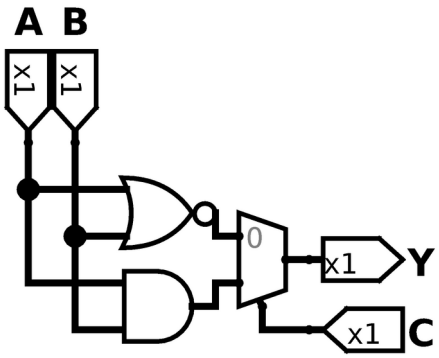
A) An 8:1 multiplexer



B) A 4:1 multiplexer and one inverter



C) A 2:1 multiplexer and two other logic gates



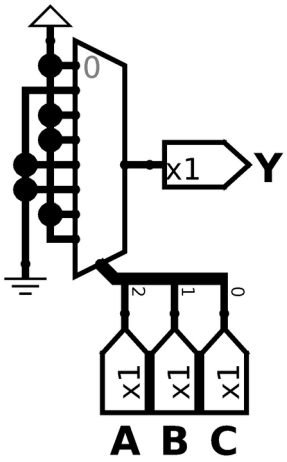
Exercise 2.42

Implement the function from Exercise 2.17(A) using

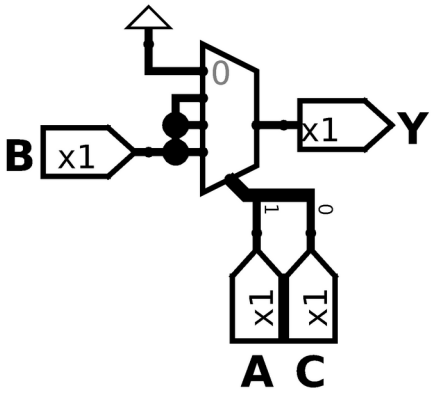
A) An 8:1 multiplexer
B) A 4:1 multiplexer and no other gates
C) A 2:1 multiplexer, one OR gate, and an inverter

Solution

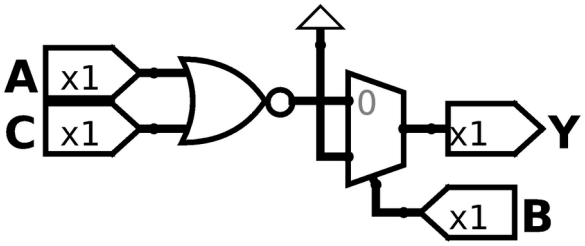
A) An 8:1 multiplexer



B) A 4:1 multiplexer and no other gates



C) A 2:1 multiplexer, one OR gate, and an inverter



Exercise 2.43

Determine the propagation delay and contamination delay of the circuit in Figure 2.83. Use the gate delays given in Table 2.8

Table 2.8 Gate delays for Exercises 2.43–2.45 and 2.47–2.48

Gate	t_{pd} (ps)	t_{cd} (ps)
NOT	15	10
2-input NAND	20	15
3-input NAND	30	25
2-input NOR	30	25
3-input NOR	45	35
2-input AND	30	25
3-input AND	40	30
2-input OR	40	30
3-input OR	55	45
2-input XOR	60	40

Solution

It's easy to notice that the worst path is whichever A, B, C or D and the best path is E.
$t_{pd} = 3 * 20\ ps = 60\ ps$
$t_{cd} = 1 * 15\ ps = 15\ ps$

Exercise 2.44

Determine the propagation delay and contamination delay of the circuit in Figure 2.84. Use the gate delays given in Table 2.8.

Solution

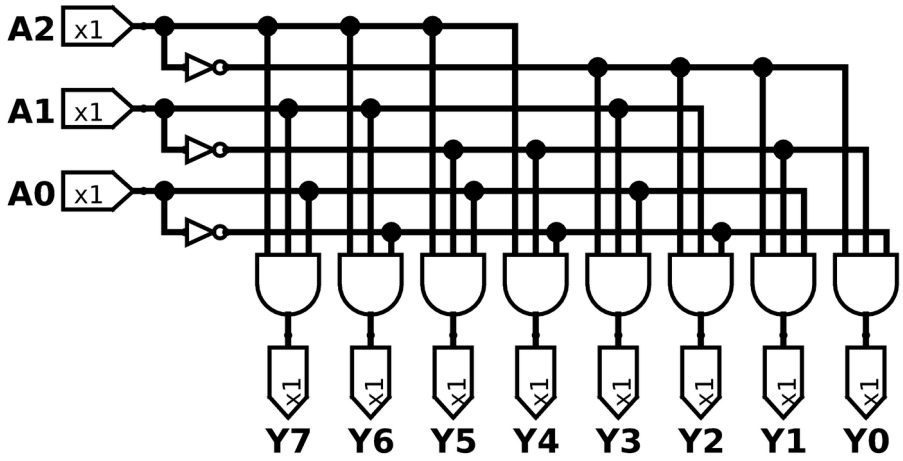
Here it isn't easy to know which is the best and worst path, then we need to evaluate every path, so we start from A, B and C.
$t_{pd} = 30\ ps + 20\ ps + 30\ ps + 20\ ps = 100\ ps$
$t_{cd} = 25\ ps + 15\ ps + 25\ ps + 15\ ps = 80\ ps$
From D
$t_{pd} = 20\ ps + 30\ ps + 20\ ps = 70\ ps$
$t_{cd} = 15\ ps + 25\ ps + 15\ ps = 55\ ps$
From E
$t_{pd} = 30\ ps + 30\ ps + 20\ ps = 80\ ps$
$t_{cd} = 25\ ps + 25\ ps + 15\ ps = 65\ ps$
From F and G
$t_{pd} = 20\ ps + 30\ ps + 30\ ps + 20\ ps = 100\ ps$
$t_{cd} = 15\ ps + 25\ ps + 25\ ps + 15\ ps = 80\ ps$
To get the total time of the circuit we need to select the worst time for propagation delay and the best time for contamination delay, so we get.
$t_{pd} = 100\ ps$
$t_{cd} = 55\ ps$

Exercise 2.45

Sketch a schematic for a fast 3:8 decoder. Suppose gate delays given in Table 2.8 (and only the gates in that table are available). Design your decoder to have the shortest possible critical path and indicate what that path is. What are its propagation delay and contamination delay?

Solution

$Y_7 = A_2 \cdot A_1 \cdot A_1$ $Y_6 = A_2 \cdot A_1 \cdot \overline{A_1}$ $Y_5 = A_2 \cdot \overline{A_1} \cdot A_0$ $Y_4 = A_2 \cdot \overline{A_1} \cdot \overline{A_1}$
$Y_3 = \overline{A_2} \cdot A_1 \cdot A_1$ $Y_2 = \overline{A_2} \cdot A_1 \cdot \overline{A_1}$ $Y_1 = \overline{A_2} \cdot \overline{A_1} \cdot A_0$ $Y_0 = \overline{A_2} \cdot \overline{A_1} \cdot \overline{A_0}$



Every path of this decoder uses 3 input AND gates and NOT gates, but some others like Y_7 output are made without NOT gates so the times we get are the following.

$t_{pd} = 15\,ps + 30\,ps = 45\,ps$

$t_{cd} = 25\,ps$

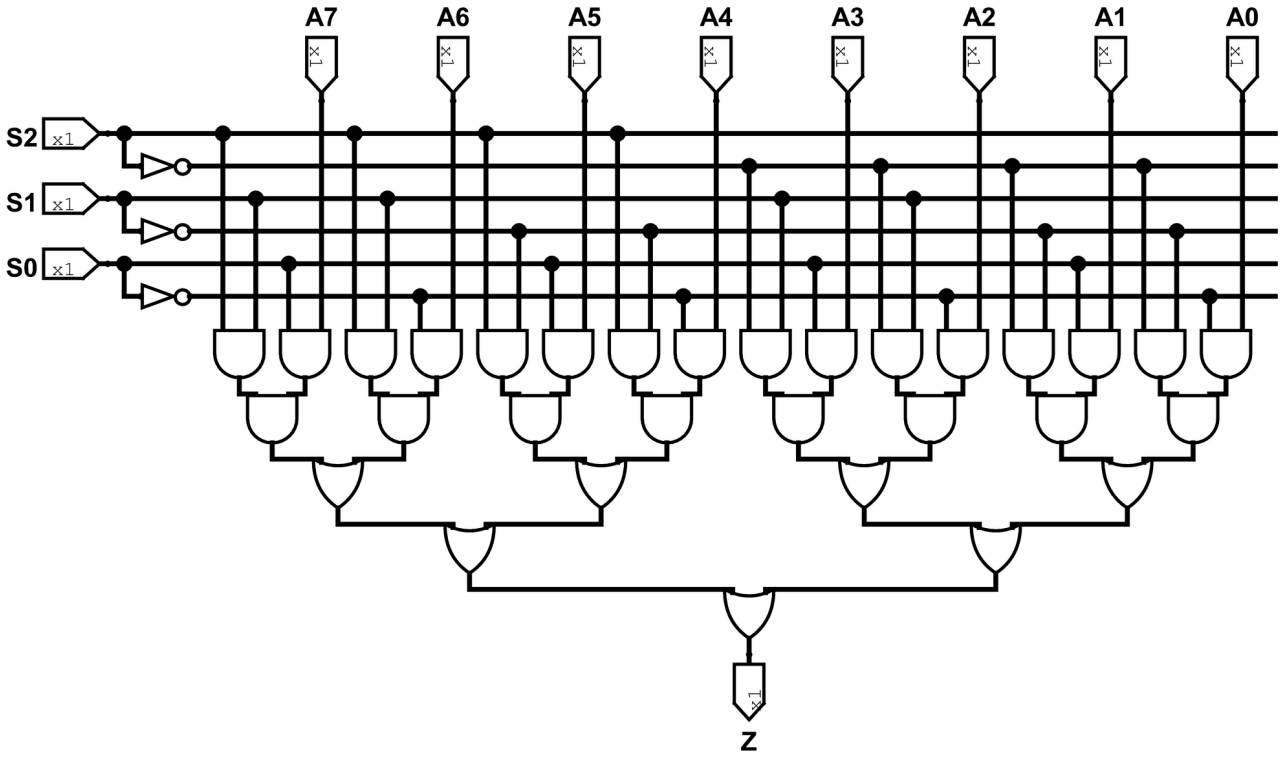
Exercise 2.46

Design an 8:1 multiplexer with the shortest possible delay from the data inputs to the output. You may use any of the gates from Table 2.7. Sketch a schematic. Using the gate delays from the table, determine this delay.

Solution

$t_{pd} = 15\,ps + 2*30\,ps + 3*40\,ps = 195\,ps$

$t_{cd} = 2*25\,ps + 3*30\,ps = 140\,ps$



Exercise 2.47

Redesign the circuit from Exercise 2.35 to be fast as possible. Use only gates from Table 2.8. Sketch the new circuit and indicate the critical path. What are its propagation delay and contamination delay?

Solution

For the redesign we just need to do some push bubble to convert OR gates in NAND and other tricks. Then we can start to analyse for every output, first we start with P. Every path passes through 3 input NAND, 2 input AND and 2 input NAND. Some of the entries are inverted so the times are the follow ones:

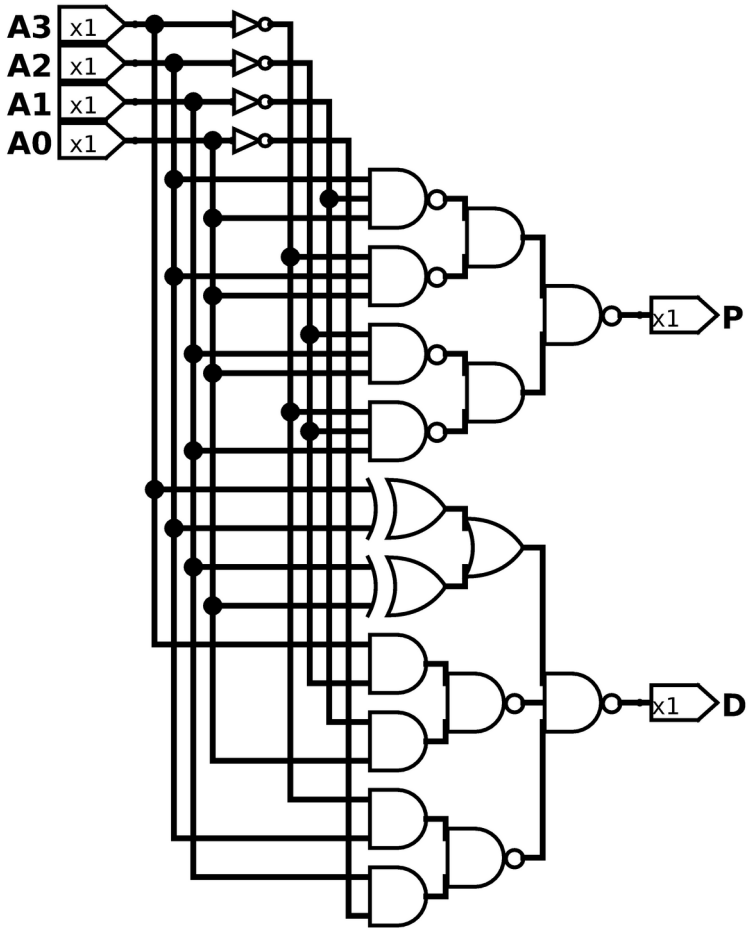
$t_{pd} = 30\,ps + 30\,ps + 20\,ps + 15\,ps = 95\,ps$

$t_{cd} = 25\,ps + 25\,ps + 15\,ps + 10\,ps = 75\,ps$

The next one is output D. It result obvious that the more faster way is from AND path so we get contamination delay from AND path and propagation delay from XOR path:

$t_{pd} = 60\,ps + 40\,ps + 30\,ps = 130\,ps$

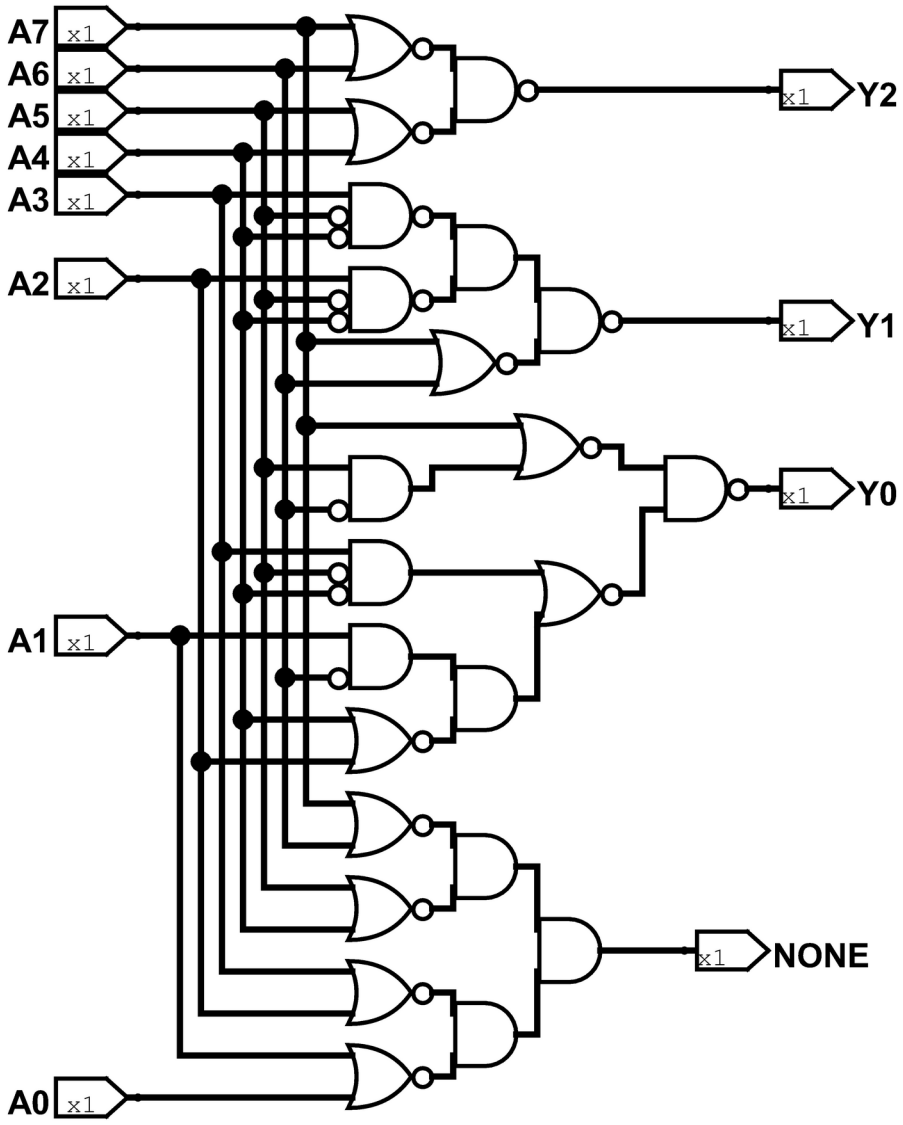
$t_{cd} = 25\,ps + 15\,ps + 25\,ps = 65\,ps$



Exercise 2.48

Redesign the priority encoder from Exercise 2.36 to be as fast as possible. You may use any of the gates from Table 2.8. Sketch the new circuit and indicate the critical path. What are its propagation delay and contamination delay?

Solution



$t_{pd} = 30\ ps + 30\ ps + 30\ ps + 20\ ps = 110\ ps$
$t_{cd} = 15\ ps + 25\ ps = 40\ ps$

Exercise 2.49

Another way to think about transistor level design is to use De Morgan's theorem to consider the pull-up and pull-down networks. Design the pull-down network of a transistor level gate directly from the equations below. Then, apply De Morgan's theorem to the equations and draw the pull-up network using that rewritten equation. Also, state the number of transistors used. Do not forget to draw (and count) the inverters needed to complement the inputs, if needed.

A) $W = \overline{A + (B \cdot C) + (\overline{C} \cdot D)}$
B) $X = \overline{\overline{A} \cdot (B + C + D) + (A \cdot \overline{D})}$
C) $Y = \overline{(\overline{A} \cdot ((B \cdot C) + (\overline{B} \cdot \overline{C}))) + (A \cdot \overline{B} \cdot C)}$

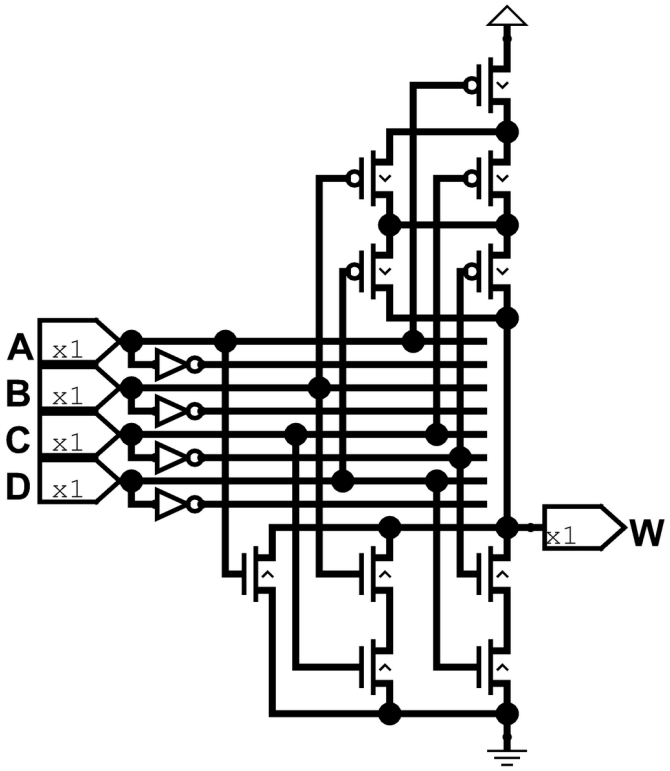
Solution

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

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

Pull Down: $W = \overline{A + (B \cdot C) + (\overline{C} \cdot D)}$

Pull Up: $W = \overline{A} \cdot (\overline{B} + \overline{C}) \cdot (C + \overline{D})$



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

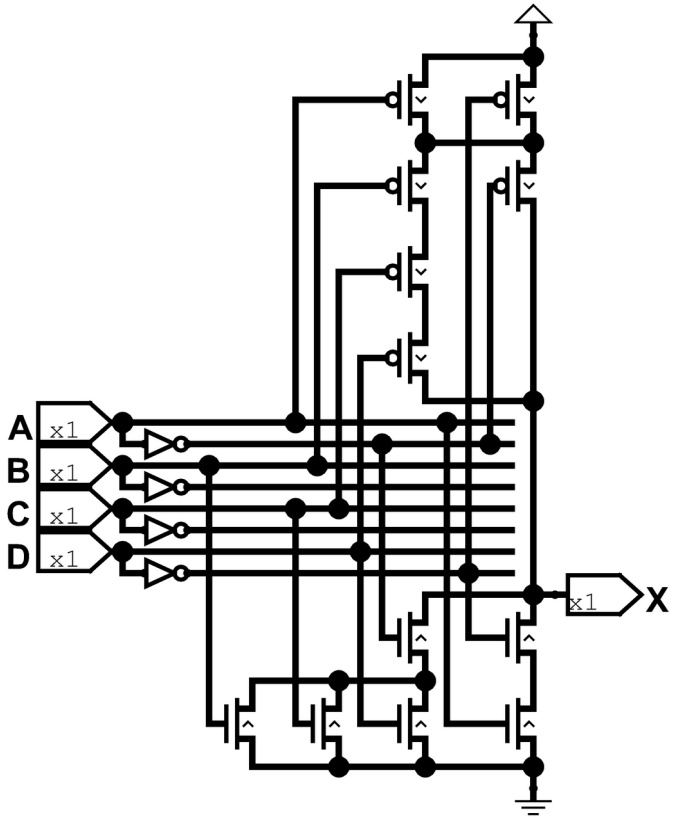
$X = (\overline{\overline{A} \cdot (B + C + D)}) \cdot (\overline{A \cdot \overline{D}})$

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

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

Pull Down: $X = \overline{\overline{A} \cdot (B + C + D) + (A \cdot \overline{D})}$

Pull Up: $X = (A + (\overline{B} \cdot \overline{C} \cdot \overline{D})) \cdot (\overline{A} + D)$



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

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

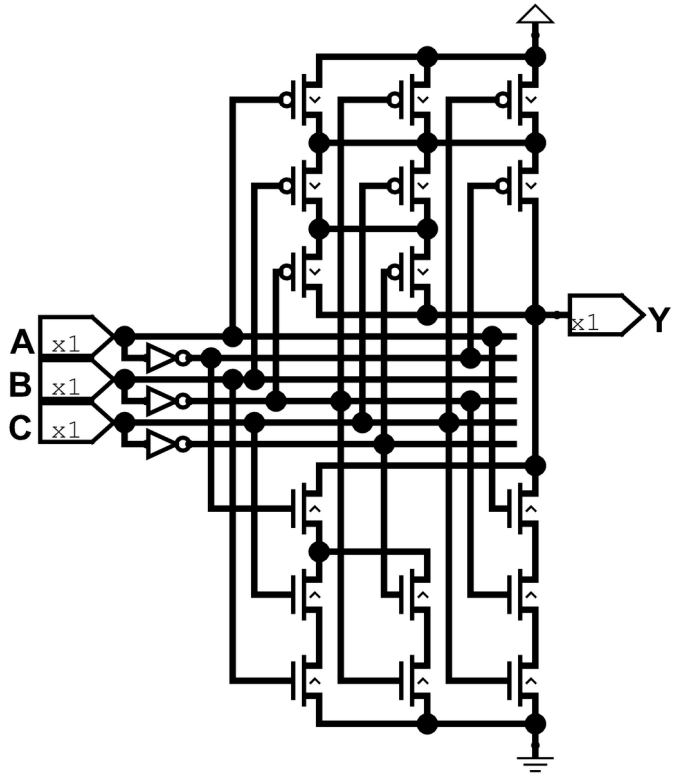
$Y=(A+\overline{((B\cdot C)+(\overline{B}\cdot\overline{C}))})\cdot(\overline{A}+B+\overline{C})$

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

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

Pull Down: $Y=\overline{(\overline{A}\cdot((B\cdot C)+(\overline{B}\cdot\overline{C})))+(A\cdot\overline{B}\cdot C)}$

Pull Up: $Y=(A+((\overline{B}+\overline{C})\cdot(B+C)))\cdot(\overline{A}+B+\overline{C})$



Exercise 2.50

Repeat Exercise 2.49 for the equations below.

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

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

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

Solution

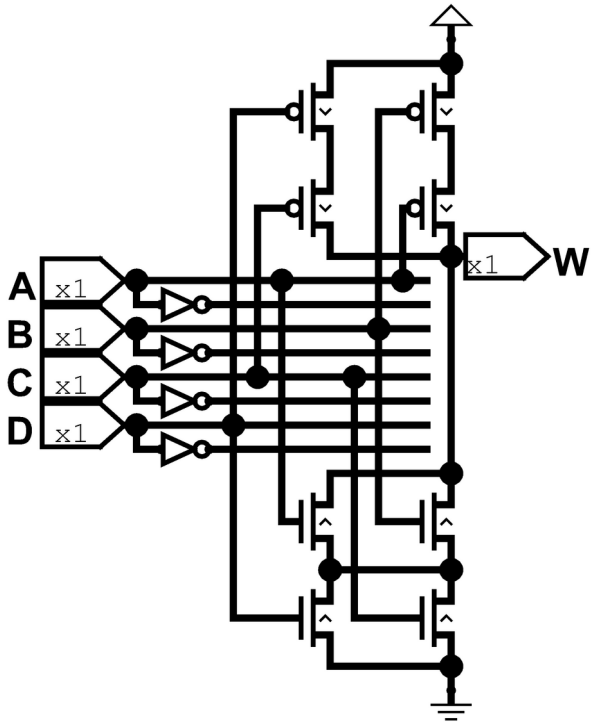
A) $W=\overline{(A+B)\cdot(C+D)}$

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

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

Pull Down: $W=\overline{(A+B)\cdot(C+D)}$

Pull Up: $W=(\overline{A}\cdot\overline{B})+(\overline{C}\cdot\overline{D})$



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

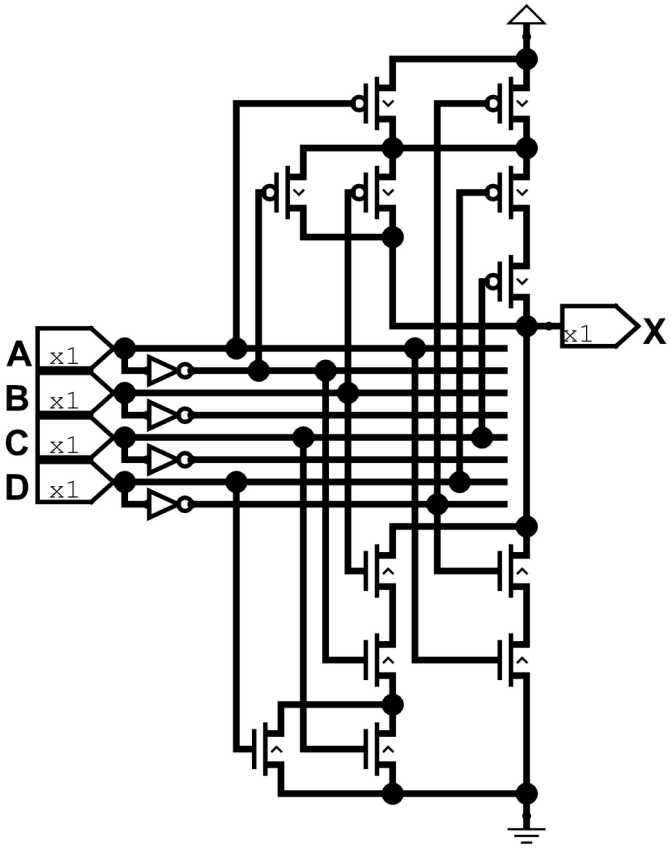
$X=((\overline{\overline{A}\cdot B})\cdot(C+D))\cdot(\overline{A\cdot\overline{D}})$

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

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

Pull Down: $X=\overline{((\overline{A}\cdot B)\cdot(C+D))+(A\cdot\overline{D})}$

Pull Up: $X=((A+\overline{B})+(\overline{C}\cdot\overline{D}))\cdot(\overline{\overline{A}}+D)$



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

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

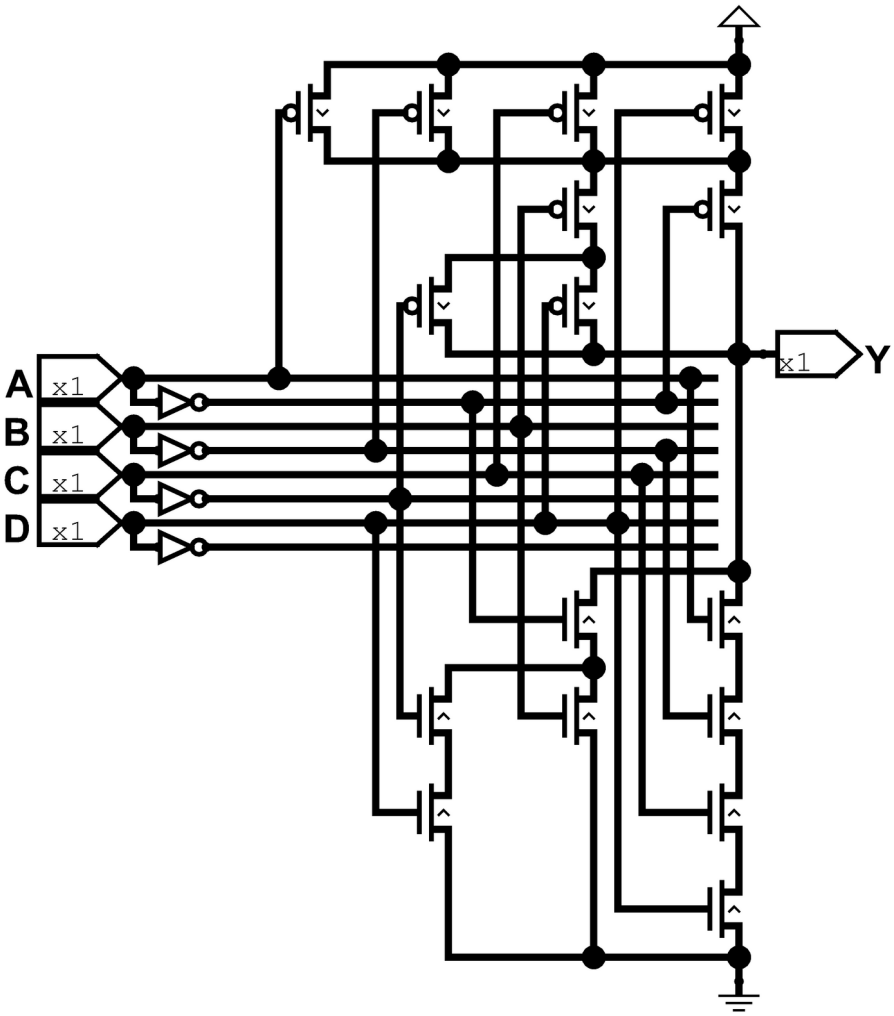
$Y=(A+(\overline{B+(\overline{C}\cdot D)}))\cdot(\overline{A}+B+\overline{C}+\overline{D})$

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

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

Pull Down: $Y=\overline{(\overline{A}\cdot(B+(\overline{C}\cdot D)))+(A\cdot\overline{B}\cdot C\cdot D)}$

Pull Up: $Y=(A+(\overline{B}\cdot(C+\overline{D})))\cdot(\overline{A}+B+\overline{C}+\overline{D})$


A
x1
B
x1
C
x1
D
x1
x1
Y

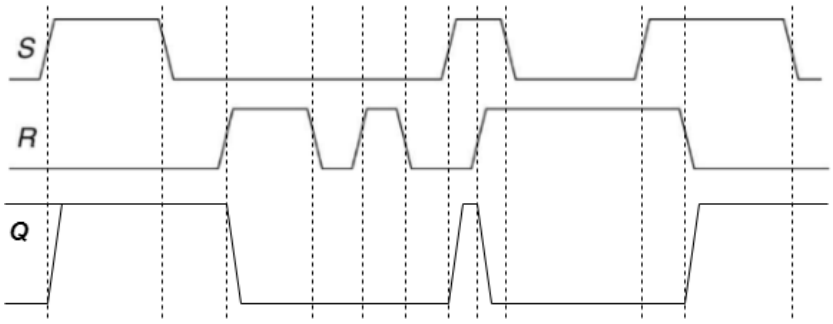
Chapter 3

Exercise 3.1

Given the input waveforms shown in Figure 3.61, sketch the output Q of an SR latch.

Figure 3.61 Input waveforms of SR latch for Exercise 3.1

Solution



Exercise 3.2

Given the input waveforms shown in Figure 3.62, sketch the output Q of an SR latch.

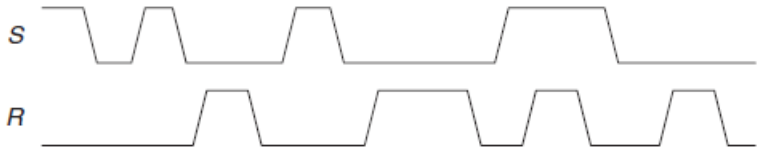
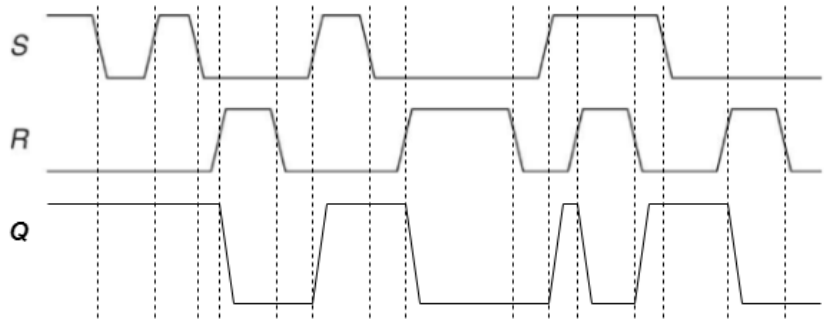


Figure 3.62 Input waveforms of SR latch for Exercise 3.2

Solution



Exercise 3.3

Given the input waveforms shown in Figure 3.63, sketch the output Q of a D latch.

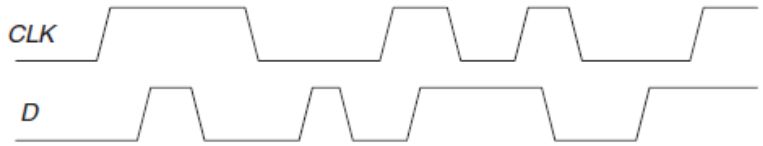
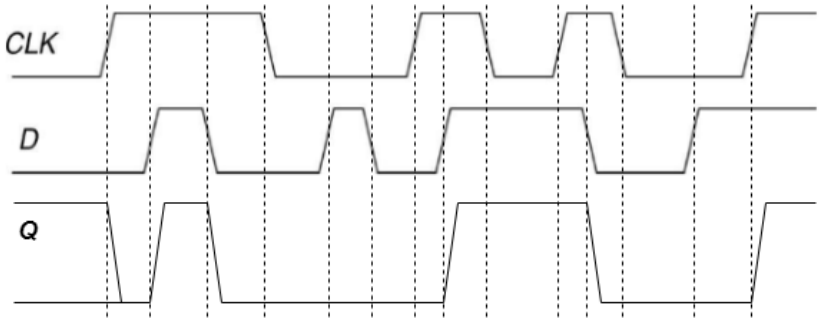


Figure 3.63 Input waveforms of D latch or flip-flop for Exercises 3.3 and 3.5

Solution



Exercise 3.4

Given the input waveforms shown in Figure 3.64, sketch the output Q of a D latch

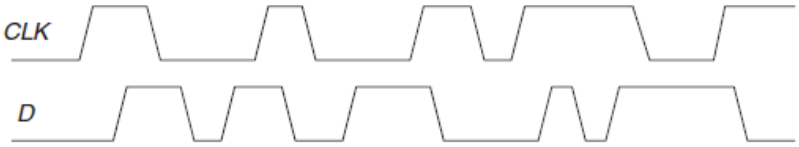
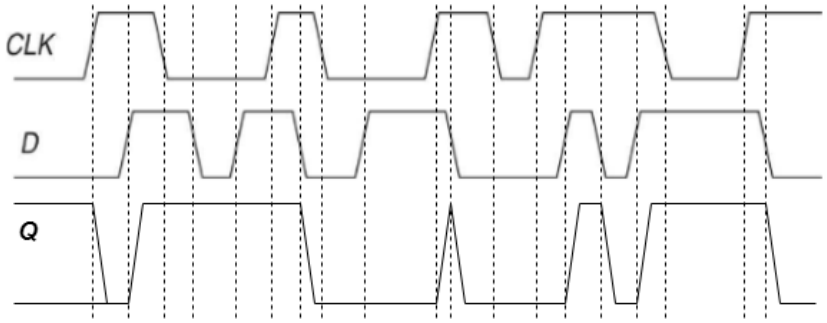


Figure 3.64 Input waveforms of D latch or flip-flop for Exercises 3.4 and 3.6

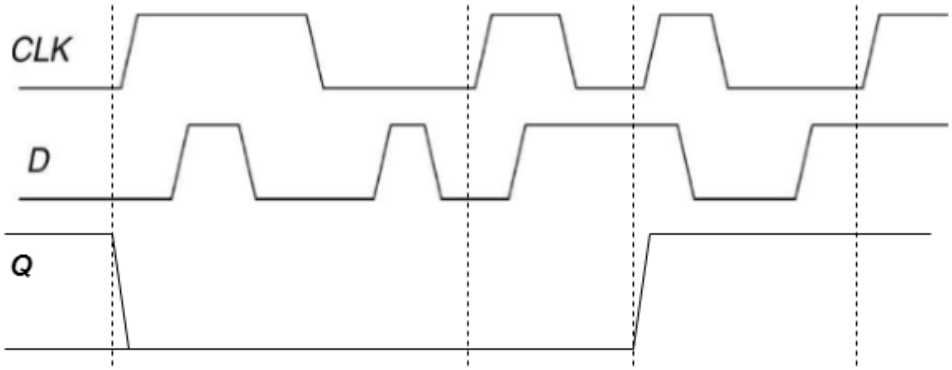
Solution



Exercise 3.5

Given the input waveforms shown in Figure 3.63, sketch the output Q of a D flip-flop.

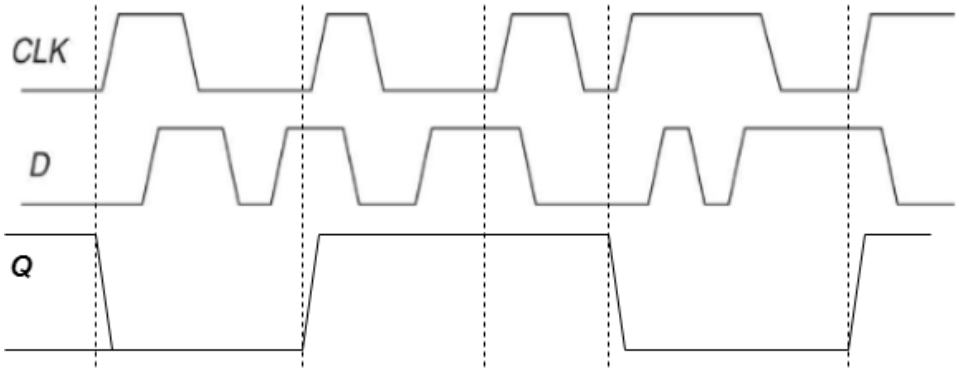
Solution



Exercise 3.6

Given the input waveforms shown in Figure 3.64, sketch the output Q of a D flip-flop.

Solution



Exercise 3.7

Is the circuit in Figure 3.65 combinational logic or sequential logic? Explain in a simple fashion what the relationship is between the inputs and outputs. What would you call this circuit?

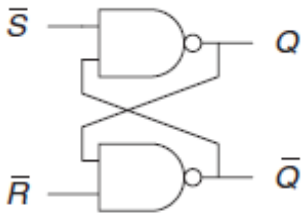


Figure 3.65 Mystery circuit

Solution

Is a sequential circuit because you need to know the previous state to know how it would behave. If we analyse the circuit we can notice that the NAND gates need all the inputs to be true to be false. Then we can imagine the follow situations:

- $\overline{S}=0$ and $\overline{R}=0$: In the first gate no matter the value of $\overline{Q}$ the output will be true then $Q=1$, then the second gate will also be true then $\underline{Q}=1$.
- $\overline{S}=0$ and $\overline{R}=1$: In the first gate no matter the value of $\overline{Q}$ the output will be true then $Q=1$, then the second gate will be false because Q and $\overline{R}$ are true then $\overline{Q}=0$.

- $\overline{S}=1$ and $\overline{R}=0$: In the first gate we can not know the output, then we move to the next gate, in the second gate no matter the value of Q the output will be true then $\overline{Q}=1$. So with this update we can know the value of the first gate which is false, $Q=0$.
- $\overline{S}=1$ and $\overline{R}=1$: In both cases we need to know the actual value of Q to know the process of this combinational circuit then we follow with:
    - $Q=0$: If we assume the value of Q we can know the value of the second gate which is true, $Q=1$. then the value of the first gate is false, $Q=0$.
    - $Q=1$: If we assume the value of Q we can know the value of the second gate which is false, $Q=0$. then the value of the first gate is true, $Q=1$.

This is the same behaviour of a SR latch then is the same circuit.

Exercise 3.8

Is the circuit in Figure 3.66 combinational logic or sequential logic? Explain in a simple fashion what the relationship is between the inputs and outputs. What would you call this circuit?

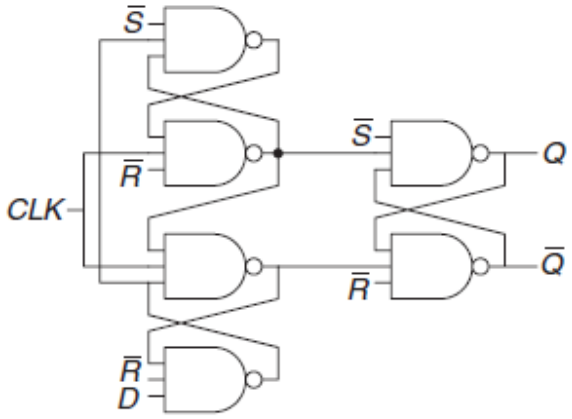


Figure 3.66 Mystery circuit

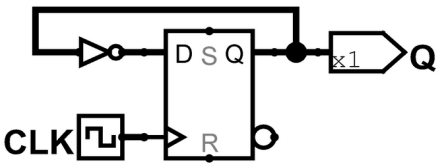
Solution

This is a sequential circuit which is made by 3 SR latches, the first two are the first part of the circuit which is bounded by a CLK signal before to stage to the final latch, as you can see if CLK is 0 no matter the other inputs signal the final SR will receive 11 which means that keep the previous state. Then we need CLK to be 1 to analyse the D input, D is bounded by S and R inputs, because it passes through the first and last gate. First the last gate R needs to be 0 to D propagates next to the first gate and third gate. But to pass the third gate some conditions need to be True because the third gate needs to be the second gate which in turn depends on the first gate, then we need to S be 0 and CLK be 1. However because when CLK needs to be 1 to pass the signal it results that only on the edge of the CLK signal D will pass because when it changes it blocks the first gate. Then we have a crazy loop that once D passes it blocks until the CLK Triggers again. Finally the S and R besides blocks D on the first phase, on the second it acts like a normal S and R inputs from a SR latch, so this is an asynchronous set, reset.

Exercise 3.9

The toggle (T) flip-flop has one input, CLK and one output Q. On each rising edge of CLK, Q toggles to the complement of its previous value. Draw a schematic for a T flip-flop using a D flip-flop and an inverter.

Solution



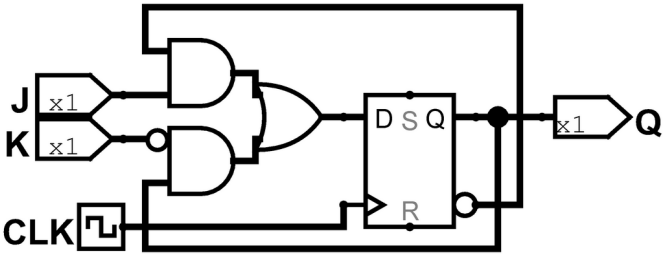
Exercise 3.10

A JK flip-flop receives a clock and two inputs, J and K. On the rising edge of the clock, it updates the output Q. If J and K are both 0, Q retains its old value, If only J is 1, Q becomes 1. If only K is 1, Q becomes 0. If both J and K are 1, Q becomes the opposite of its present state.

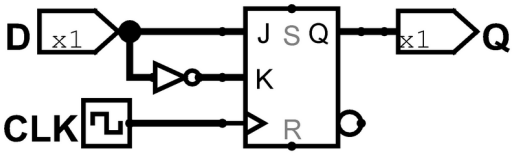
A) Construct a JK flip-flop, using a D flip-flop and some combinational logic.
B) Construct a D flip-flop, using a JK flip-flop and some combinational logic.
C) Construct a T flip-flop, using a JK flip-flop.

Solution

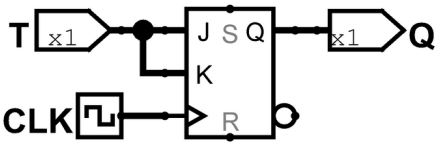
A) JK flip-flop



B) D flip-flop



C) T flip-flop



Exercise 3.11

The circuit in Figure 3.67 is called a Muller C-element. Explain in a simple fashion what the relationship is between the inputs and the output.

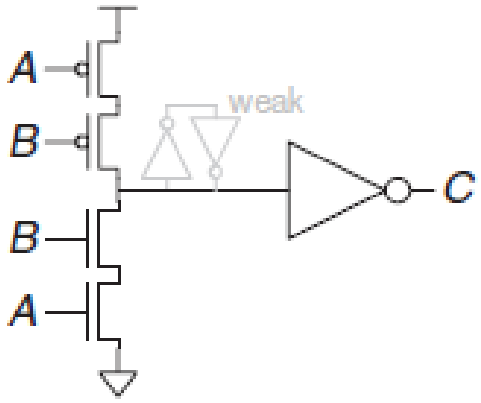


Figure 3.67 Muller C-element

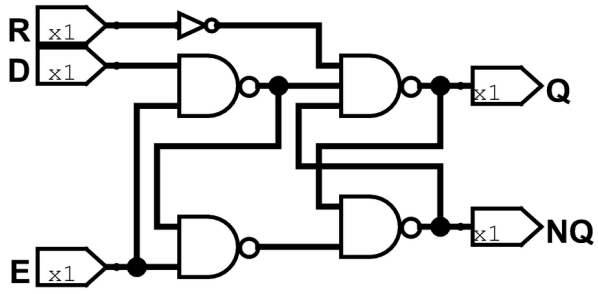
Solution

First is known how it will behave in the state 11 the pulldown network activates and passes through the inverter making the output 1, the same is true for 00 but it activates the pullup network and the output is 0. However there's an inverter loop connected which repeats the signal received. When the circuit gets a 01 or 10 none of the networks will activate and the value it's supposed to be indeterminate, but with this repeater it saves the previous value of the circuit.

Exercise 3.12

Design an asynchronously resettable D latch, using logic gates.

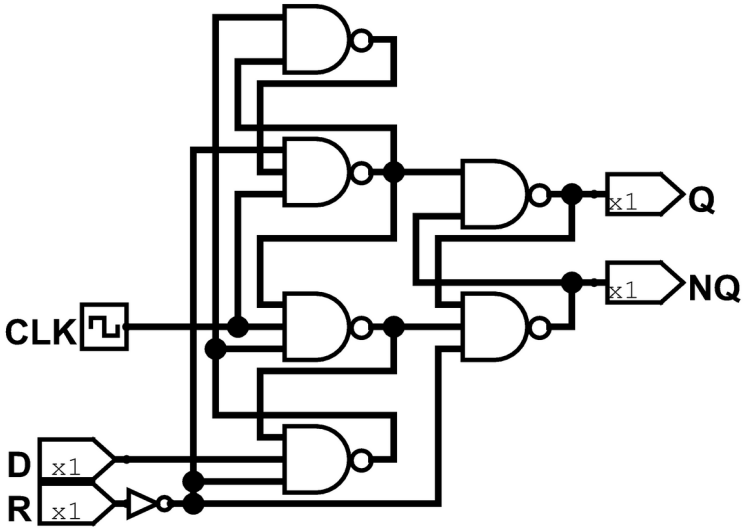
Solution



Exercise 3.13

Design an asynchronously resettable D flip-flop, using logic gates.

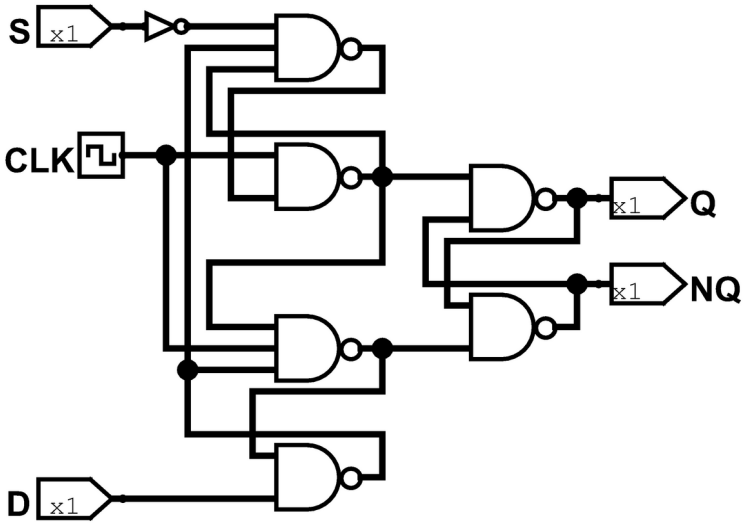
Solution



Exercise 3.14

Design a synchronously settable D flip-flop, using logic gates.

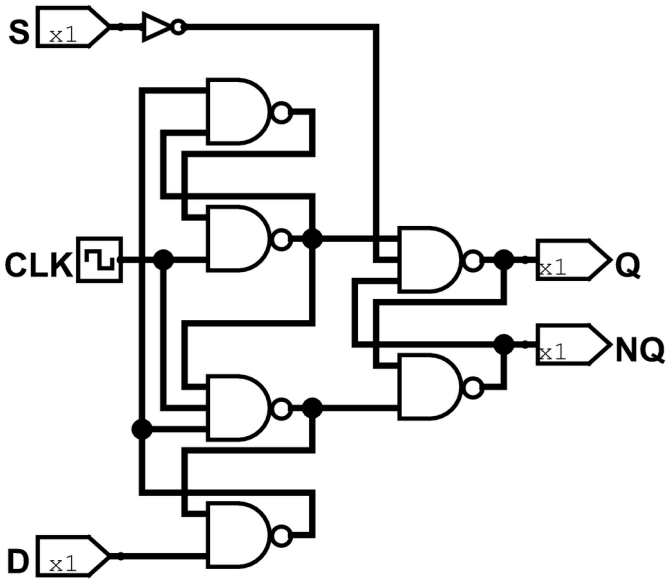
Solution



Exercise 3.15

Design an asynchronously settable D flip-flop, using logic gates.

Solution



Exercise 3.16

Suppose that a ring oscillator is built from N inverters connected in a loop. Each inverter has a minimum delay of t_{pd}. If N is odd, determine the range of frequencies at which the oscillator might operate.

Solution

The frequency of this ring would be a constant, no matter how much big is N, because every wire is alternating between turn on and off so, this time is $2*t_{pd}$, so the unique frequency that this oscillator can operate is $1/2*t_{pd}$.

Exercise 3.17

Why must N be odd in Exercise 3.16?

Solution

Because there will be a conflict with defining the value of one of its nodes. If N is odd then the values will oscillate but if N is even then because it can define the value eventually will broke the gates.

Exercise 3.18

Which of the circuits in Figure 3.68 are synchronous sequential circuits? Explain.

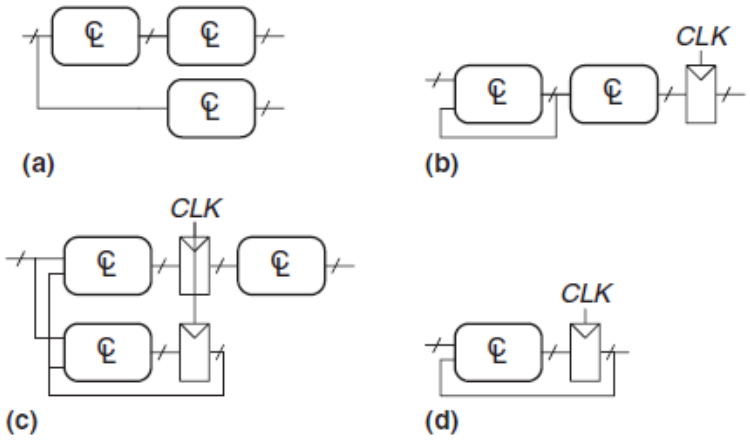


Figure 3.68 Circuits

Solution

A) This circuit is not synchronous because the behaviour it's not bounded by a clock signal. Neither is sequential because none of the CL networks supply any inputs to them.
B) This circuit is synchronous because all outputs are bounded by a clock signal, either is sequential because the output of one CL network supplies itself input.
C) This circuit is not synchronous because even if the output passes by the clock signal, it could cause some race conditions due to the propagation delay of the last CL. Finally it is sequential because some of the CL network provides input signal by itself and other networks.
D) This final circuit it's both because all outputs are treated by the clock signal and the CL provides its inputs.

Exercise 3.19

You are designing an elevator controller for a building with 25 floors. The controller has two inputs: UP and DOWN. It produces an output indicating the floor that the elevator is on. There is no floor 13. What is the minimum number of bits of state in the controller?

Solution

The number of states is the number of floors implied (24), so if you transform every state on a bit number you will need 5 bits of information.

Exercise 3.20

You are designing a FSM to keep track of the mood of four students working in the digital design lab. Each student's mood is either HAPPY (the circuit works), SAD (the circuit blew up), BUSY (working on the circuit), CLUELESS (confused about the circuit), or ASLEEP (face down on the circuit board). How many states does the FSM have? What is the minimum number of bits necessary to represent these states?

Solution

There's 4 students and if we try to divide every possible state then we could arrange the 4 students into a mix of 5 possible combinations. That's mean that we could get $5^4 = 625$ possible combinations, so it needs 10 bits.

Exercise 3.21

How would you factor the FSM from Exercise 3.20 into multiple simple machines? How many states does each simple machine have? What is the minimum total number of bits necessary in this factored design?

Solution

It can create 4 modules of individual mood trackers for every student so we could output every student, so if the mood states need 3 bits then we multiply for every module and we get 12 bits nedeed.

Exercise 3.22

Describe in words what the state machine in Figure 3.69 does. Using binary state encodings, complete a state transition table and output table for the FSM. Write Boolean equations for the next state and output and sketch a schematic of the FSM.

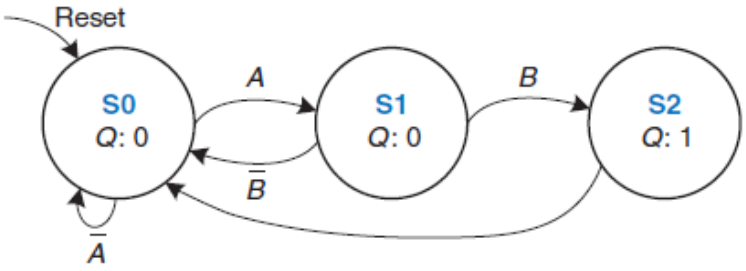


Figure 3.69 State transition diagram for Exercise 3.22

Solution

This FSM needs a sequence of A and B to Q be 1 else Q will be 0.

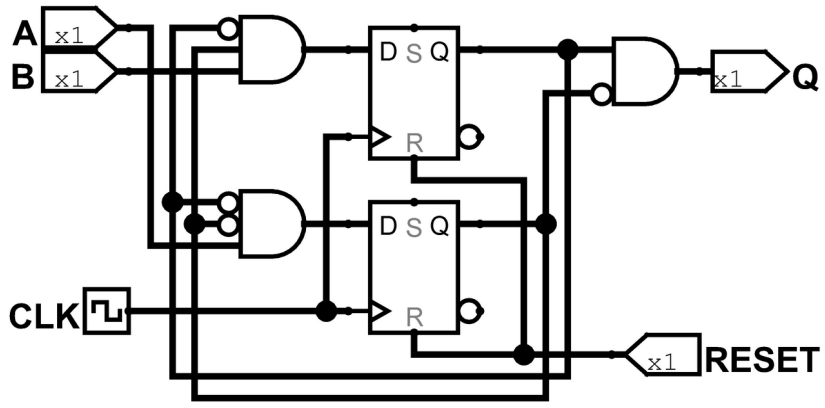
States	Encoding
S0	00
S1	01
S2	10

Actual State	Current state encoding		Inputs		Next State	Next state encoding	
	S_1	S_0	A	B		S'_1	S'_0
S0	0	0	0	X	S0	0	0
S0	0	0	1	X	S1	0	1
S1	0	1	X	0	S0	0	0
S1	0	1	X	1	S2	1	0
S2	1	0	X	X	S0	0	0

Current states		Output
S1	S0	Q

0	0	0
0	1	0
1	0	1

Next State Equation	Output equation
$S'_1 = \overline{S_1} \cdot S_0 \cdot B$	$Q = S_1 \cdot \underline{S_0}$
$S'_0 = \overline{S_1} \cdot \overline{S_0} \cdot A$	



Exercise 3.23

Describe in words what the state machine in Figure 3.70 does. Using binary state encodings, complete a state transition table and output table for the FSM. Write Boolean equations for the next state and output and sketch a schematic of the FSM.

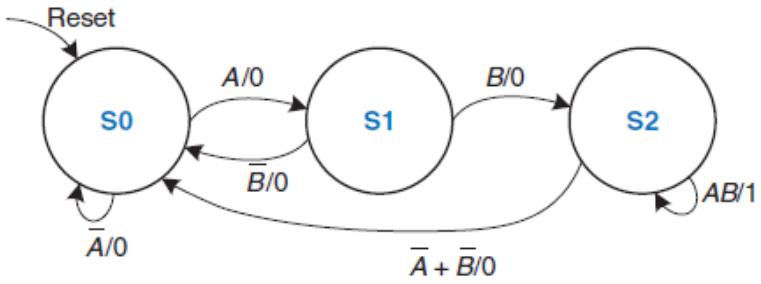


Figure 3.70 State transition diagram for Exercise 3.23

Solution

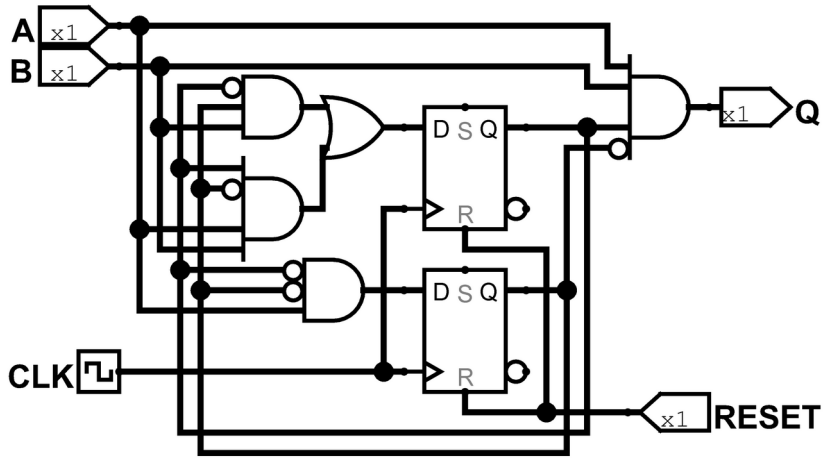
This FSM does exactly the same as the past exercise, but with the difference that if it holds A and B in the final state it will still be 1. From here we define the transition table and binary encoding.

States	Encoding
S0	00
S1	01

S2	10

Actual State	Current state encoding		Inputs		Next State	Next state encoding		Output
	S_1	S_0	A	B		S'_1	S'_0	Q
S0	0	0	0	X	S0	0	0	0
S0	0	0	1	X	S1	0	1	0
S1	0	1	X	0	S0	0	0	0
S1	0	1	X	1	S2	1	0	0
S2	1	0	1	1	S2	1	0	1
S2	1	0	0	X	S0	0	0	0
S2	1	0	X	0	S0	0	0	0

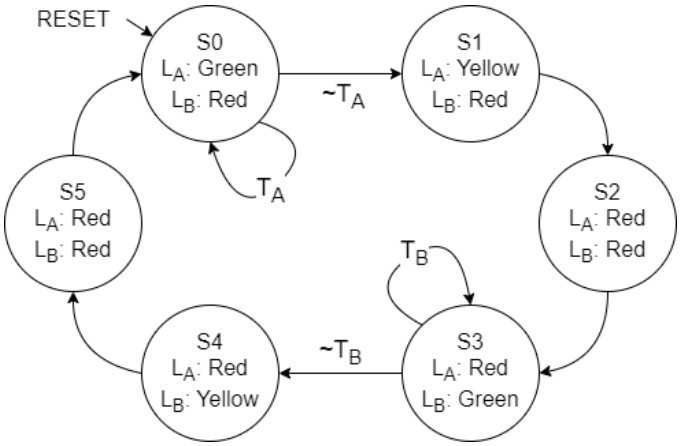
Next State Equation	Output equation
$S'_1 = \overline{S_1} \cdot S_0 \cdot B + S_1 \cdot \overline{S_0} \cdot A \cdot B$	$Q = S_1 \cdot \overline{S_0} \cdot A \cdot B$
$S'_0 = \overline{S_1} \cdot \overline{S_0} \cdot A$	



Exercise 3.24

Accidents are still occurring at the intersection of Academic Avenue and Bravado Boulevard. The football team is rushing into the intersection the moment light B turns green. They are colliding with sleep deprived CS majors who stagger into the intersection just before light A turns red. Extend the traffic light controller from Section 3.4.1 so that both lights are red for 5 seconds before either light turns green again. Sketch your improved Moore machine state transition diagram, state encodings, state transition table, output table, next state and output equations, and your FSM schematic.

Solution



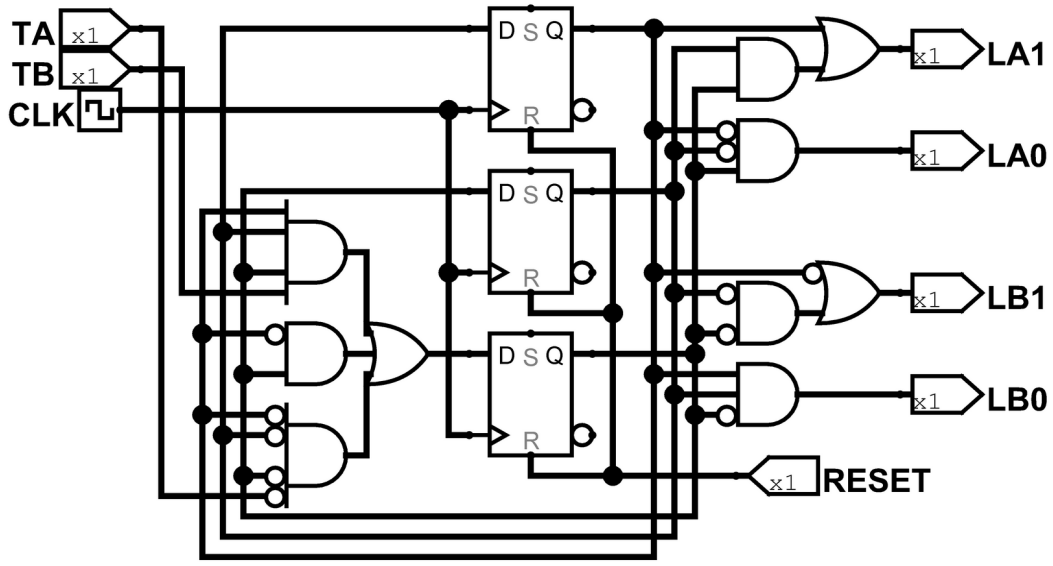
States	Encoding
S0	000
S1	001
S2	011
S3	111
S4	110
S5	100
Green	00
Yellow	01
Red	10

Actual State	Current state encoding			Inputs		Next State	Next state encoding		
	S_2	S_1	S_0	T_A	T_B		S'_2	S'_1	S'_0
S0	0	0	0	1	X	S0	0	0	0
S0	0	0	0	0	X	S1	0	0	1
S1	0	0	1	X	X	S2	0	1	1
S2	0	1	1	X	X	S3	1	1	1
S3	1	1	1	X	1	S3	1	1	1
S3	1	1	1	X	0	S4	1	1	0
S4	1	1	0	X	X	S5	1	0	0
S5	1	0	0	X	X	S0	0	0	0

Current state			Output			
S_2	S_1	S_0	L_{A1}	L_{A0}	L_{B1}	L_{B0}

0	0	0	0	0	1	0
0	0	1	0	1	1	0
0	1	1	1	0	1	0
1	1	1	1	0	0	0
1	1	0	1	0	0	1
1	0	0	1	0	1	0

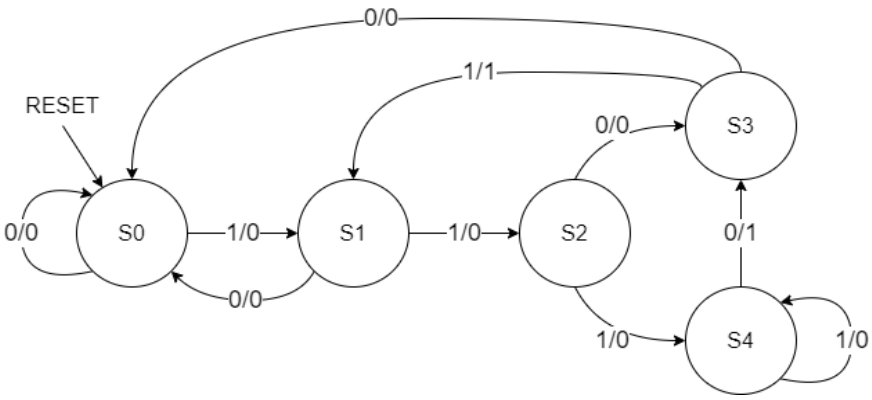
Next State Equation	Output equation	
$S'_2 = S_1$	$L_{A1} = S_2 + S_1 \cdot S_0$	$L_{B1} = \underline{S_2} + \overline{S_1} \cdot \overline{S_0}$
$S'_1 = S_0$	$L_{A0} = \overline{S_2} \cdot \overline{S_1} \cdot S_0$	$L_{B0} = S_2 \cdot S_1 \cdot \overline{S_0}$
$S'_0 = (\overline{S_2} \cdot \overline{S_1} \cdot \overline{S_0} \cdot \overline{T_A}) + (\overline{S_2} \cdot S_0) + (S_2 \cdot S_1 \cdot S_0 \cdot T_B)$		



Exercise 3.25

Alyssa P. Hacker's snail form section 3.4.3 has a daughter with a Mealy machine FSM brain. The daughter snail smiles whenever she slides over the pattern 1101 or the pattern 1110. Sketch the state transition diagram for this happy snail using as a few states as possible. Choose state encodings and write a combined state transition and output table, using your encodings. Write the next state and output equations and sketch your FSM schematic.

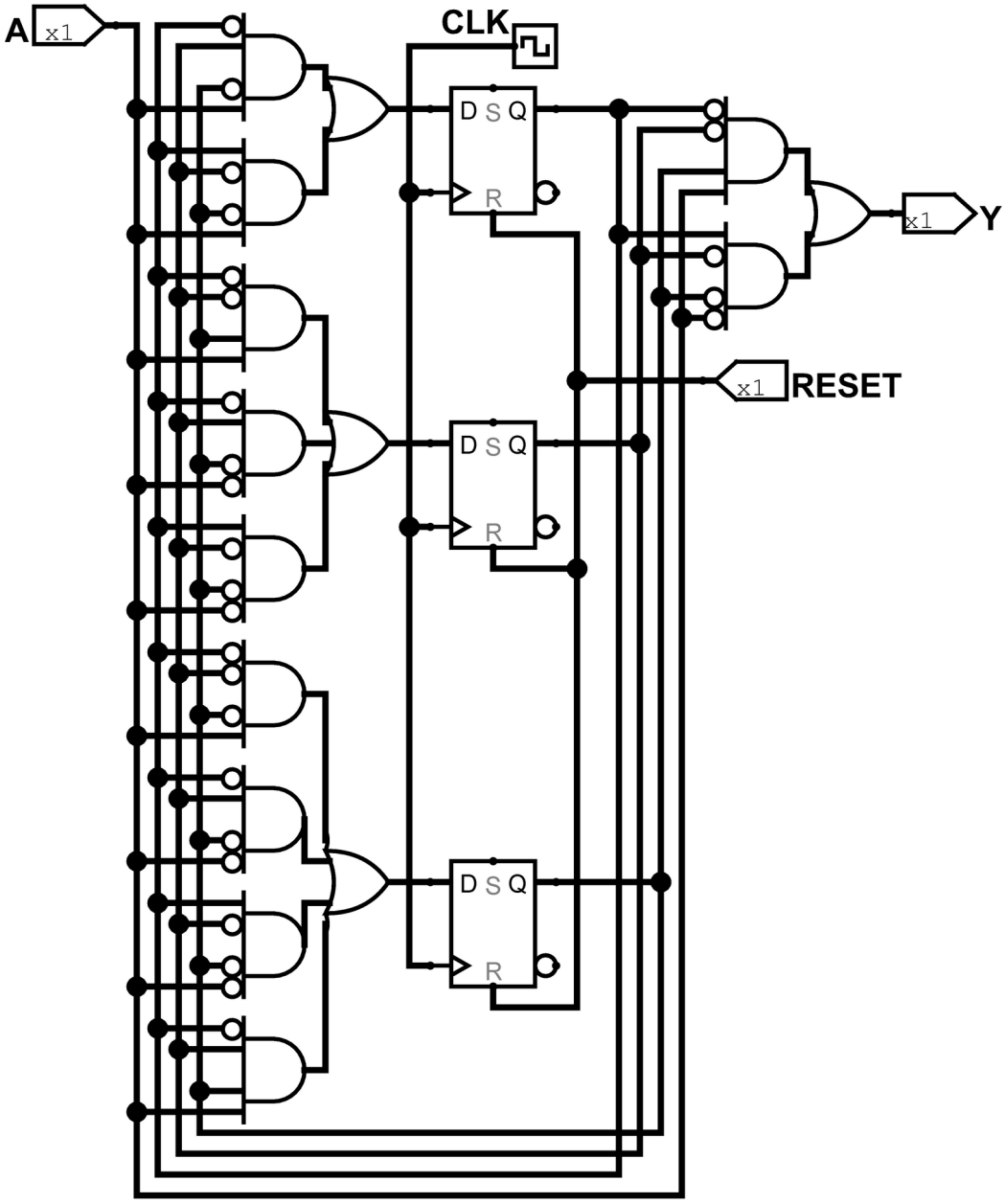
Solution



States	Encoding
S0	000
S1	001
S2	010
S3	011
S4	100

Actual State	Current state encoding			Inputs	Next State	Next state encoding			Output
	S_2	S_1	S_0	A		S'_2	S'_1	S'_0	Y
S0	0	0	0	0	S0	0	0	0	0
S0	0	0	0	1	S1	0	0	1	0
S1	0	0	1	0	S0	0	0	0	0
S1	0	0	1	1	S2	0	1	0	0
S2	0	1	0	0	S3	0	1	1	0
S2	0	1	0	1	S4	1	0	0	0
S3	0	1	1	0	S0	0	0	0	0
S3	0	1	1	1	S1	0	0	1	1
S4	1	0	0	0	S3	0	1	1	1
S4	1	0	0	1	S4	1	0	0	0

Next State Equation	Output equation
$S'_2=(\overline{S_2}\cdot S_1\cdot\overline{S_0}\cdot A)+(S_2\cdot\overline{S_1}\cdot\overline{S_0}\cdot A)$	$Y=(\overline{S_2}\cdot\overline{S_1}\cdot S_0\cdot A)+(\overline{S_2}\cdot\overline{S_1}\cdot\overline{S_0}\cdot A)$
$S'_1=(\overline{S_2}\cdot\overline{S_1}\cdot S_0\cdot A)+(\overline{S_2}\cdot S_1\cdot\overline{S_0}\cdot\overline{A})+(S_2\cdot\overline{S_1}\cdot\overline{S_0}\cdot\overline{A})$	
$S'_0=(\overline{S_2}\cdot\overline{S_1}\cdot\overline{S_0}\cdot A)+(\overline{S_2}\cdot S_1\cdot\overline{S_0}\cdot\overline{A})+(\overline{S_2}\cdot S_1\cdot S_0\cdot A)+(S_2\cdot\overline{S_1}\cdot\overline{S_0}\cdot\overline{A})$	

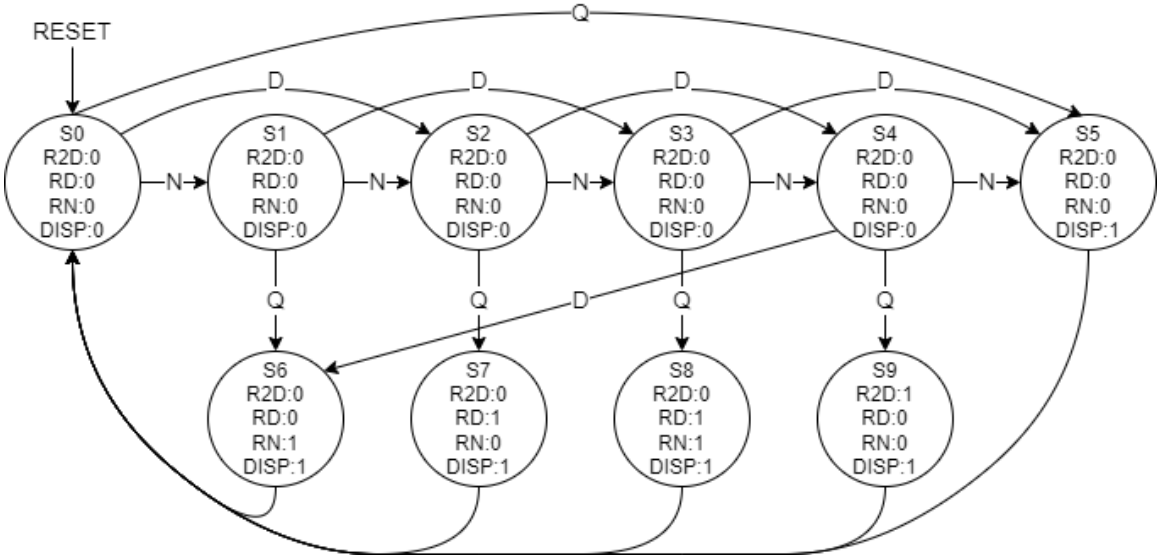


Exercise 3.26

You have been enlisted to design a soda machine dispenser for your department lounge. Sodas are partially subsidised by the students of the IEEE, so they cost only 25 cents. The machine accepts nickels, dimes and quarters. When enough coins have been inserted, it dispenses the soda and returns any necessary change. Design an FSM controller for the soda machine. The FSM inputs are Nickel, Dime and Quarter, indicating which coin was inserted. Assume that exactly one coin is inserted on each cycle. The outputs are Dispense, ReturnNickel, ReturnDime, and ReturnTwoDimes. When the FSM reaches 25 cents, it asserts Dispense and the necessary Return outputs required to deliver the appropriate change. Then, it should be ready to start accepting coins for another soda.

Solution

For this exercise we change the labels of the inputs and outputs to acronyms. So Nickel, Dime, Quarter, Dispense, ReturnNickel, ReturnDime, and ReturnTwoDimes are changed by N, D, Q, DISP, RN, RD, R2D respectively.



States	Encoding
S0	00_0000_0001
S1	00_0000_0010
S2	00_0000_0100
S3	00_0000_1000
S4	00_0001_0000
S5	00_0010_0000
S6	00_0100_0000
S7	00_1000_0000
S8	01_0000_0000
S9	10_0000_0000

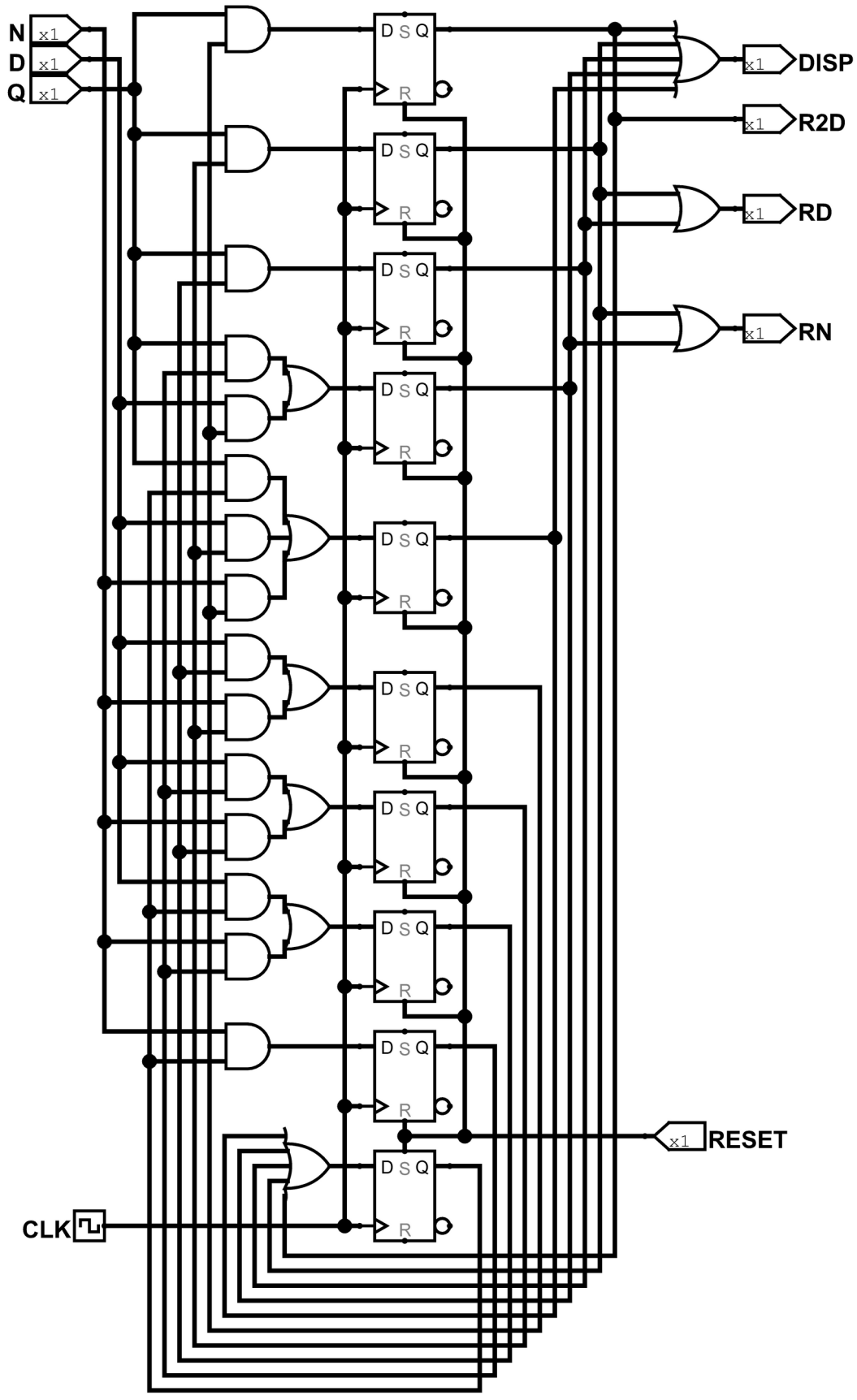
Actual State	Current state encoding	Inputs			Next State	Next state encoding
	$S_{9:0}$	N	D	Q		$S'_{9:0}$
S0	00_0000_0001	1	0	0	S1	00_0000_0010
S0	00_0000_0001	0	1	0	S2	00_0000_0100
S0	00_0000_0001	0	0	1	S5	00_0010_0000
S1	00_0000_0010	1	0	0	S2	00_0000_0100
S1	00_0000_0010	0	1	0	S3	00_0000_1000
S1	00_0000_0010	0	0	1	S6	00_0100_0000

S2	00_0000_0100	1	0	0	S3	00_0000_1000
S2	00_0000_0100	0	1	0	S4	00_0001_0000
S2	00_0000_0100	0	0	1	S7	00_1000_0000
S3	00_0000_1000	1	0	0	S4	00_0001_0000
S3	00_0000_1000	0	1	0	S5	00_0010_0000
S3	00_0000_1000	0	0	1	S8	01_0000_0000
S4	00_0001_0000	1	0	0	S5	00_0010_0000
S4	00_0001_0000	0	1	0	S6	00_0100_0000
S4	00_0001_0000	0	0	1	S9	10_0000_0000
S5	00_0010_0000	X	X	X	S0	00_0000_0001
S6	00_0100_0000	X	X	X	S0	00_0000_0001
S7	00_1000_0000	X	X	X	S0	00_0000_0001
S8	01_0000_0000	X	X	X	S0	00_0000_0001
S9	10_0000_0000	X	X	X	S0	00_0000_0001

Current state	Output			
$S_{9:0}$	RN	RD	R2D	DISP
00_0000_0001	0	0	0	0
00_0000_0010	0	0	0	0
00_0000_0100	0	0	0	0
00_0000_1000	0	0	0	0
00_0001_0000	0	0	0	0
00_0010_0000	0	0	0	1
00_0100_0000	1	0	0	1
00_1000_0000	0	1	0	1
01_0000_0000	1	1	0	1
10_0000_0000	0	0	1	1

Next State Equation	Output equation
$S'_9 = S_4 \cdot Q$	$RN = S_8 + S_6$
$S'_8 = S_3 \cdot Q$	$RD = S_8 + S_7$
$S'_7 = S_2 \cdot Q$	$R2D = S_9$
$S'_6 = S_1 \cdot Q + S_4 \cdot D$	$DISP = S_9 + S_8 + S_7 + S_6 + S_5$
$S'_5 = S_0 \cdot Q + S_3 \cdot D + S_4 \cdot N$	

$S'_4 = S_2 \cdot D + S_3 \cdot N$	
$S'_3 = S_1 \cdot D + S_2 \cdot N$	
$S'_2 = S_0 \cdot D + S_1 \cdot N$	
$S'_1 = S_0 \cdot N$	
$S'_0 = S_9 + S_8 + S_7 + S_6 + S_5$	


N
D
Q
x1
CLK
D S Q
R
DISP
R2D
RD
RN
RESET

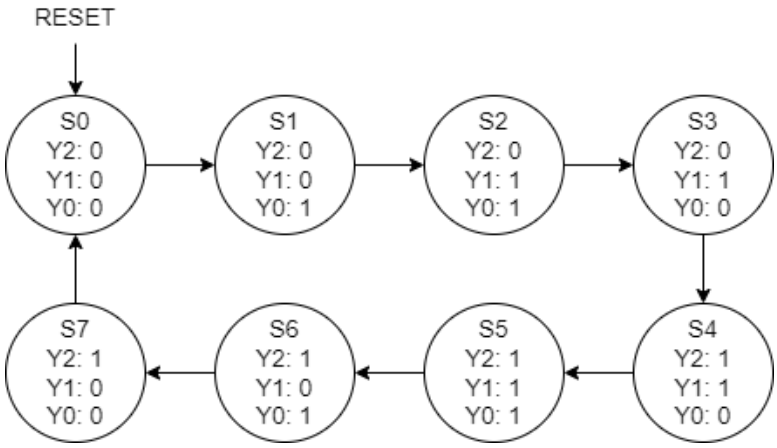
Exercise 3.27

Gray codes have a useful property in that consecutive numbers differ in only a single bit position. Table 3.23 lists a 3-bit Gray code representing the numbers 0 to 7. Design a 3-bit modulo 8 Gray code counter FSM with no inputs and three outputs. (A modulo N counter counts from 0 to N-1, then repeats. For example, a watch uses a modulo 60 counter for the minutes and seconds that counts from 0 to 59.) When reset, the output should be 000. On each clock edge, the output should advance to the next Gray code after reaching 100, it should repeat with 000.

Table 3.23 3-bit Gray code

Number	Gray code		
0	0	0	0
1	0	0	1
2	0	1	1
3	0	1	0
4	1	1	0
5	1	1	1
6	1	0	1
7	1	0	0

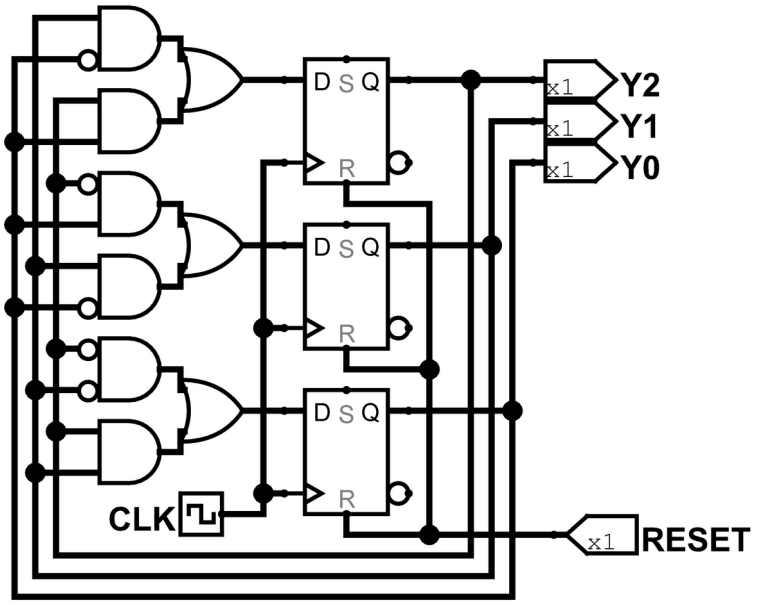
Solution



Actual State	Current state encoding			Next State	Next state encoding		
	S_2	S_1	S_0		S'_2	S'_1	S'_0
S0	0	0	0	S1	0	0	1
S1	0	0	1	S2	0	1	1
S2	0	1	1	S3	0	1	0
S3	0	1	0	S4	1	1	0

S4	1	1	0	S5	1	1	1
S5	1	1	1	S6	1	0	1
S6	1	0	1	S7	1	0	0
S7	1	0	0	S0	0	0	0

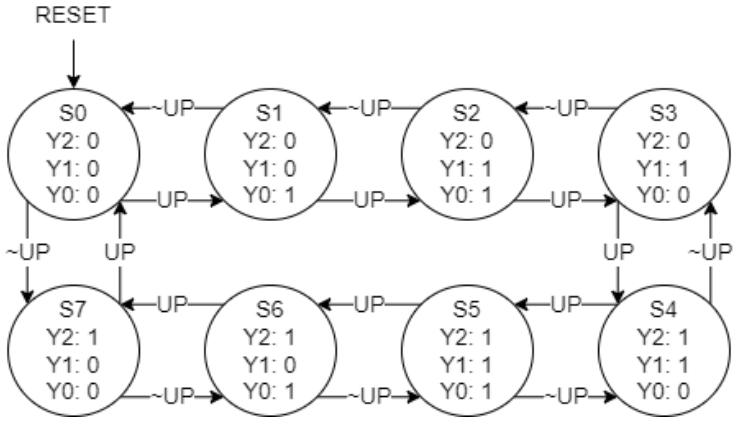
Next State Equation	Output equation
$S'_2 = S_1 \cdot \overline{S_0} + S_2 \cdot S_0$	$Y_{3:0} = S_{3:0}$
$S'_1 = \overline{S_2} \cdot S_0 + S_1 \cdot \overline{S_0}$	
$S'_0 = \overline{S_2} \cdot \overline{S_1} + S_2 \cdot S_1$	



Exercise 3.28

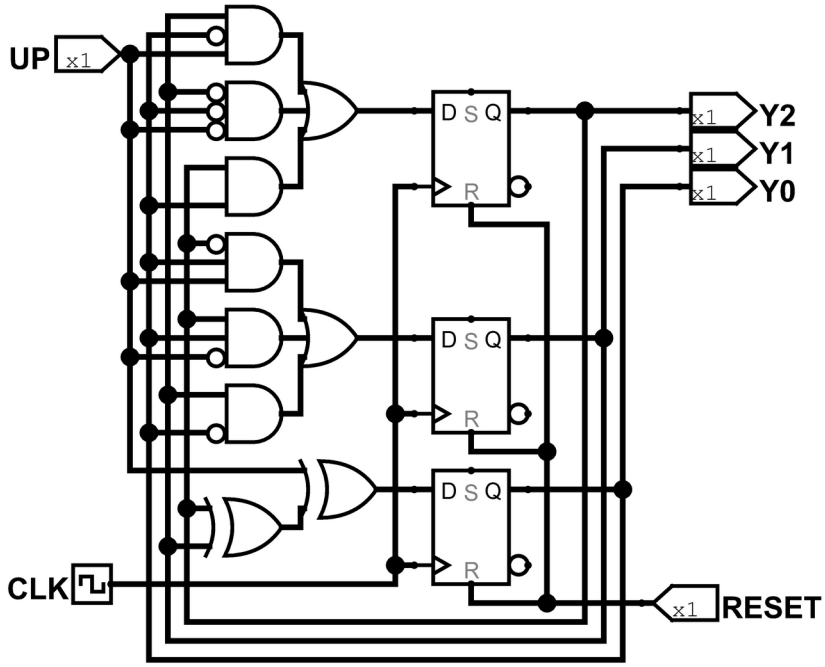
Extend your modulo 8 Gray code counter from Exercise 3.27 to be an UP/DOWN counter by adding an UP input. If UP=1, the counter advances to the next number. If UP=0, the counter retreats to the previous number.

Solution



Actual State	Current state encoding			Inputs	Next State	Next state encoding		
	S_2	S_1	S_0	UP		S'_2	S'_1	S'_0
S0	0	0	0	1	S1	0	0	1
S1	0	0	1	1	S2	0	1	1
S2	0	1	1	1	S3	0	1	0
S3	0	1	0	1	S4	1	1	0
S4	1	1	0	1	S5	1	1	1
S5	1	1	1	1	S6	1	0	1
S6	1	0	1	1	S7	1	0	0
S7	1	0	0	1	S0	0	0	0
S0	0	0	0	0	S7	1	0	0
S1	0	0	1	0	S0	0	0	0
S2	0	1	1	0	S1	0	0	1
S3	0	1	0	0	S2	0	1	1
S4	1	1	0	0	S3	0	1	0
S5	1	1	1	0	S4	1	1	0
S6	1	0	1	0	S5	1	1	1
S7	1	0	0	0	S6	1	0	1

Next State Equation	Output equation
$S'_2=(S_1\cdot\overline{S_0}\cdot UP)+(\overline{S_1}\cdot\overline{S_0}\cdot\overline{UP})+(S_2\cdot S_0)$	$Y_{3:0}=S_{3:0}$
$S'_1=(\overline{S_2}\cdot S_0\cdot UP)+(S_2\cdot S_0\cdot\overline{UP})+(S_1\cdot\overline{S_0})$	
$S'_0=(S_2\oplus S_1)\oplus UP$	



Exercise 3.29

Your company, Detect-o-rama, would like to design an FSM that takes two inputs, A and B, and generates one output, Z. The output in cycle n, Z_n, is either the Boolean AND or OR of the corresponding input A_n and the previous input A_{n-1}, depending on the other input, B_n:

$Z_n = A_n \cdot A_{n-1}$ if $B_n = 0$

$Z_n = A_n + A_{n-1}$ if $B_n = 1$

A) Sketch the waveform for Z, given the inputs shown in Figure 3.71

B) Is this FSM a Moore or a Mealy machine?

C) Design the FSM. Show your state transition diagram, encoded state transition table, next state and output equations, and schematic.

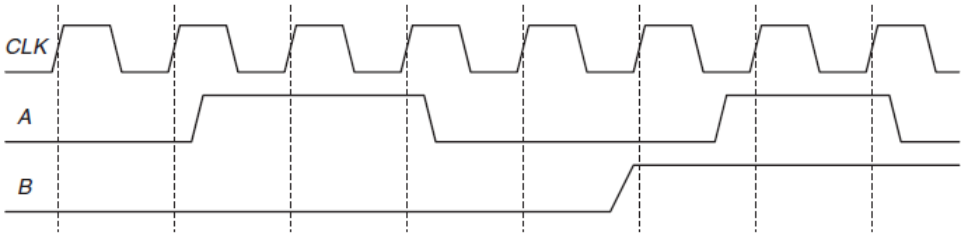
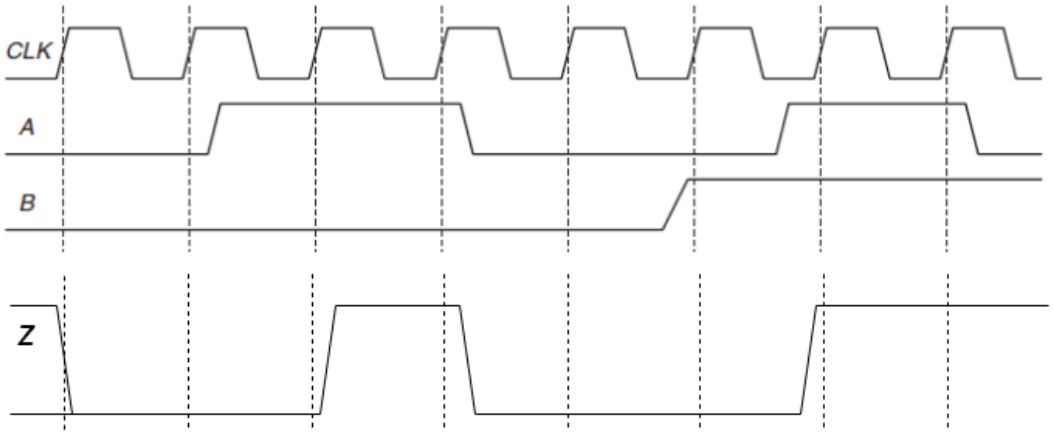


Figure 3.71 FSM input waveforms for Exercise 3.29

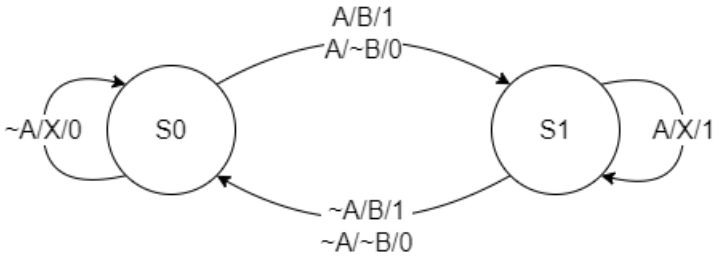
Solution

A) Waveform



B) Is a mealy machine because it depends of the actual input

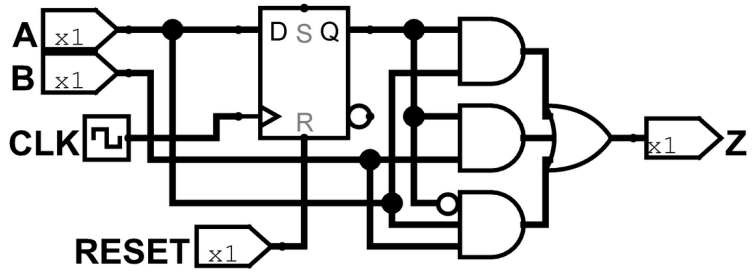
C) Design the FSM



States	Encoding
S0	0
S1	1

Actual State	Current state encoding	Inputs		Next State	Next state encoding	Output
	S_0	A	B		S'_0	Z
S0	0	0	X	S0	0	0
S0	0	1	1	S1	1	1
S0	0	1	0	S1	1	0
S1	1	1	X	S1	1	1
S1	1	0	1	S0	0	1
S1	1	0	0	S0	0	0

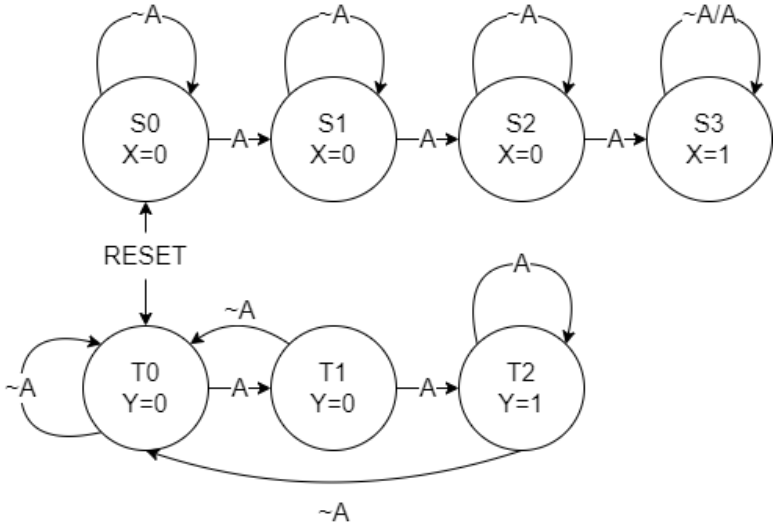
Next State Equation	Output equation
$S'_0 = A$	$Z = S_0 \cdot (A + B) + \overline{S_0} \cdot A \cdot B$



Exercise 3.30

Design an FSM with one input, A, and two outputs, X and Y. X should be 1 if A has been 1 for at least three cycles altogether (not necessarily consecutively). Y should be 1 if A has been 1 for at least two consecutive cycles. Show your state transition diagram, encoded state transition table, next state and output equations, and schematic.

Solution



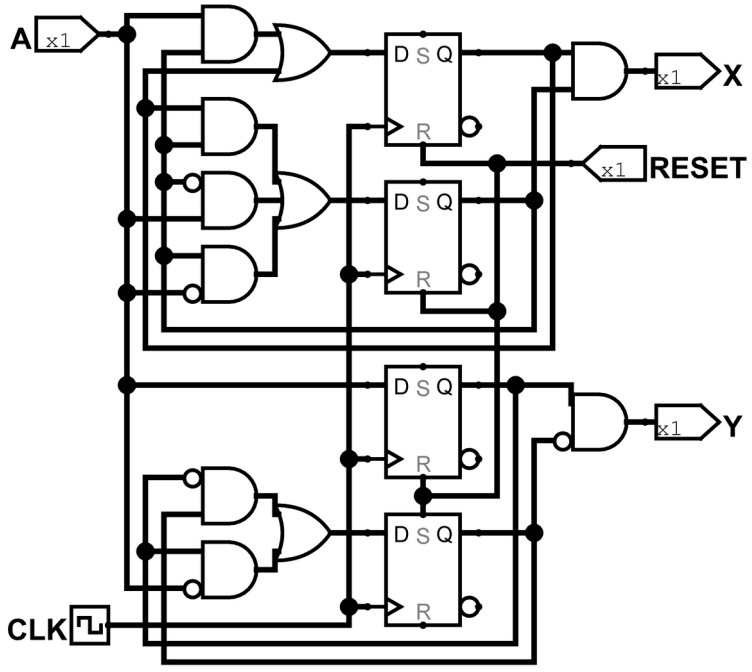
States	Encoding	States	Encoding
S0	00	T1	01
S1	01	T2	11
S2	10	T3	10
S3	11		

Actual State	Current state encoding		Inputs	Next State	Next state encoding		Output
	S_1	S_0	A		S'_1	S'_0	X
S0	0	0	1	S1	0	1	0
S0	0	0	0	S0	0	0	0
S1	0	1	1	S2	1	0	0

S1	0	1	0	S1	0	1	0
S2	1	0	1	S3	1	1	0
S2	1	0	0	S2	1	0	0
S3	1	1	X	S3	1	1	1

Actual State	Current state encoding		Inputs	Next State	Next state encoding		Output
	T_1	T_0	A		T'_1	T'_0	Y
T0	0	1	1	T1	1	1	0
T0	0	1	0	T0	0	1	0
T1	1	1	1	T2	1	0	0
T1	1	1	0	T0	0	1	0
T2	1	0	1	T2	1	0	1
T2	1	0	0	T0	0	1	1

Next State Equation	Output equation	Next State Equation	Output equation
$S'_1 = S_1 + S_0 \cdot A$	$X = S_1 \cdot S_0$	$T'_1 = A$	$Y = T_1 \cdot \overline{T_0}$
$S'_0 = S_1 \cdot S_0 + \overline{S_0} \cdot A + S_0 \cdot \overline{A}$		$T'_0 = \overline{T_1} \cdot T_0 + T_1 \cdot \overline{A}$	



Exercise 3.31

Analyse the FSM shown in Figure 3.72. Write the state transition and output tables and sketch the state transition diagram.

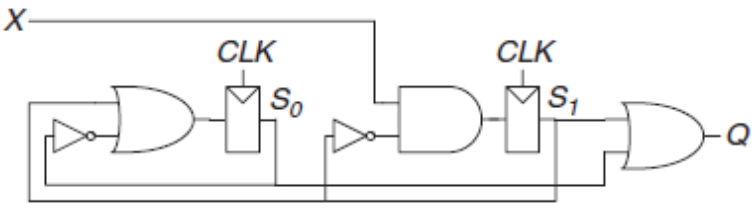
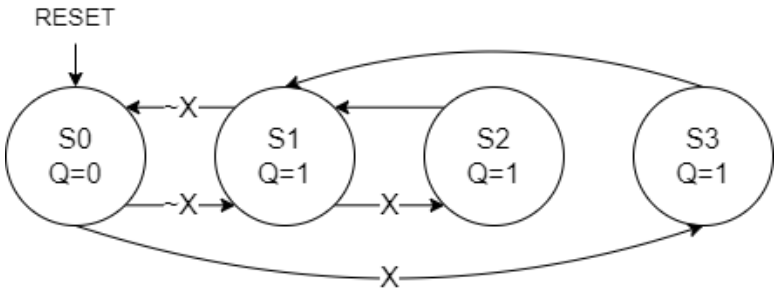


Figure 3.72 FSM schematic for Exercise 3.31

Solution

Next State Equation	Output equation
$S'_1 = \overline{S_1} \cdot X$	$Q = S_1 + S_0$
$S'_0 = S_1 + \overline{S_0}$	

Actual State	Current state encoding		Inputs	Next State	Next state encoding		Output
	S_1	S_0	X		S'_1	S'_0	Q
S0	0	0	1	S3	1	1	0
S0	0	0	0	S1	0	1	0
S1	0	1	1	S2	1	0	1
S1	0	1	0	S0	0	0	1
S2	1	0	1	S1	0	1	1
S2	1	0	0	S1	0	1	1
S3	1	1	1	S1	0	1	1
S3	1	1	0	S1	0	1	1



Exercise 3.32

Analyse the FSM shown in Figure 3.73. Recall that the s and r register inputs indicate set and reset, respectively.

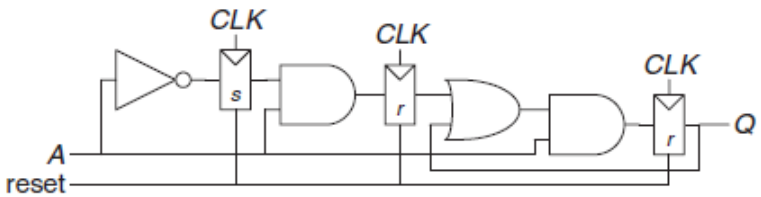


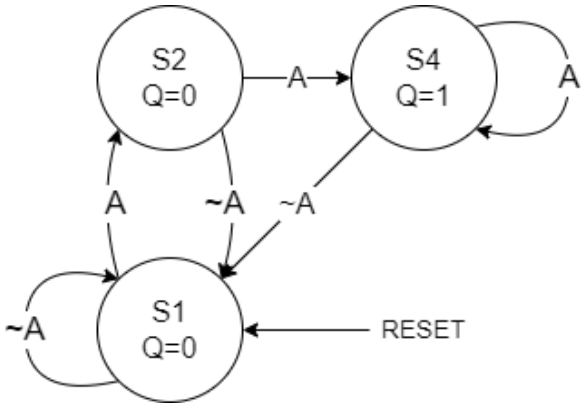
Figure 3.73 FSM schematic for Exercise 3.32

Solution

Next State Equation	Output equation
$S'_2=(S_2+S_1)\cdot A$	$Q=S_2$
$S'_1=S_0\cdot A$	
$S'_0=\overline{A}$	

Actual State	Current state encoding			Inputs	Next State	Next state encoding			Output
	S_2	S_1	S_0	A		S'_2	S'_1	S'_0	Q
S0	0	0	0	1	S0	0	0	0	0
S0	0	0	0	0	S1	0	0	1	0
S1	0	0	1	1	S2	0	1	0	0
S1	0	0	1	0	S1	0	0	1	0
S2	0	1	0	1	S4	1	0	0	0
S2	0	1	0	0	S1	0	0	1	0
S3	0	1	1	1	S6	1	1	0	0
S3	0	1	1	0	S1	0	0	1	0
S4	1	0	0	1	S4	1	0	0	1
S4	1	0	0	0	S1	0	0	1	1
S5	1	0	1	1	S6	1	1	0	1
S5	1	0	1	0	S1	0	0	1	1
S6	1	1	0	1	S4	1	0	0	1
S6	1	1	0	0	S1	0	0	1	1
S7	1	1	1	1	S6	1	1	0	1
S7	1	1	1	0	S1	0	0	1	1

This FSM has unused possible states because it starts at S1 there’s no way to reach S0, S3, S5, S6, S7 so it can be ignored. So there remain 4 states.



Exercise 3.33

Ben bitdiddle has designed the circuit in Figure 3.74 to compute a registered four-input XOR function. Each two-input XOR gate has a propagation delay of 100 ps and a contamination delay of 55 ps. Each flip flop has a setup time of 60 ps, a hold time of 20 ps, a clock to Q maximum delay of 70 ps, and a clock to minimum Q delay of 50 ps.

A) If there is no clock skew, what is the maximum operating frequency of the circuit?
B) How much clock skew can the circuit tolerate if it must operate at 2 GHz?
C) How much clock skew can the circuit tolerate before it might experience a hold time violation?
D) Alyssa P. Hacker points out that she can redesign the combinational logic between the registers to be faster and tolerate more clock skew. Her improved circuit also uses three two input XORs, but they are arranged differently. What is her circuit? What is its maximum frequency if there is no clock skew? How much clock skew can the circuit tolerate before it might experience a hold time violation?

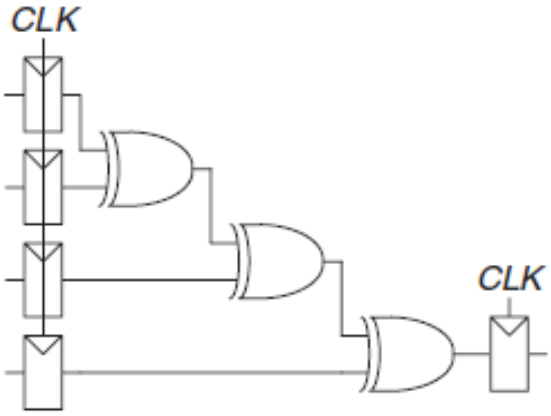


Figure 3.74 Registered four-input XOR circuit for Exercise 3.33

Solution

A) $T_c \geq t_{pcq} + t_{pd} + t_{setup} = 70\,ps + 3*100\,ps + 60\,ps = 430\,ps$
$f_c = 1/T_c = 1/430\,ps = 2.32\,GHz$
B) $T_c = 1/f_c = 1/2\,GHz = 500\,ps$
$t_{skew} \leq T_c - (t_{pcq} + t_{setup} + t_{pd}) = 500\,ps - (70\,ps + 60\,ps + 3*100\,ps) = 70\,ps$
C) $t_{skew} \leq t_{ccq} + t_{cd} - t_{hold} = 50\,ps + 55\,ps - 20\,ps = 85\,ps$

D) The maximum frequency without clock skew

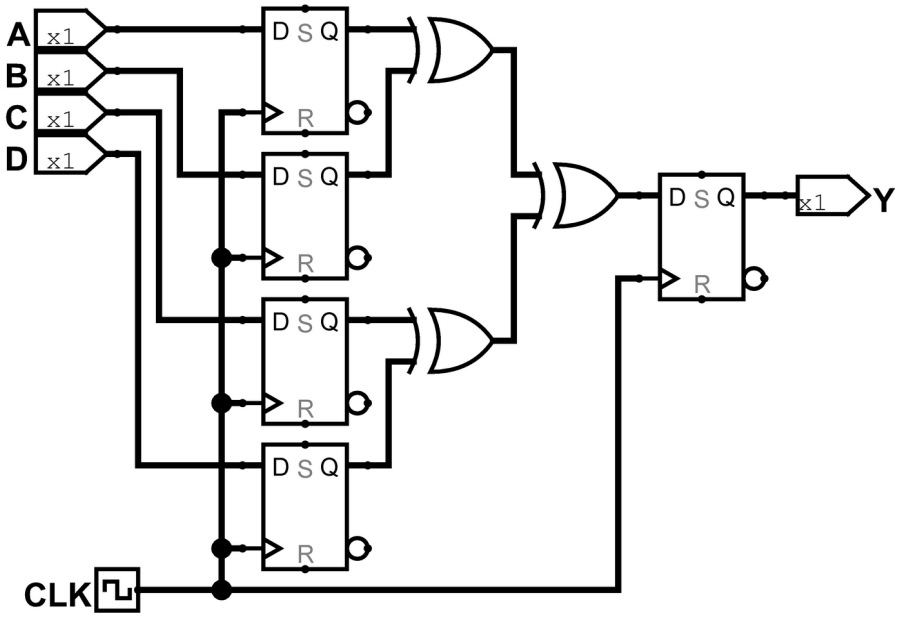
$T_c \geq t_{pcq} + t_{pd} + t_{setup} = 70\,ps + 2*100\,ps + 60\,ps = 330\,ps$

$f_c = 1/T_c = 1/330\,ps = 3.03\,GHz$

Clock stew tolerated

$t_{skew} \leq t_{ccq} + t_{cd} - t_{hold} = 50\,ps + 2*55\,ps - 20\,ps = 140\,ps$

The way to optimise is doing the follow arrangement



Exercise 3.34

You are designing an adder for the blindingly fast 2 bit RePentium Processor. The adder is built from two full adders, such that the carry out of the first adder is the carry in to the second adder, as shown in Figure 3.75. Your adder has input and output registers and must complete the addition in one clock cycle. Each full adder has the following propagation delays: 20 ps from C_{IN} to C_{OUT} or to Sum (S), 25 ps from A or B to C_{OUT}, and 30 ps from A or B to S. The adder has a contamination delay of 15 ps from C_{IN} to either output and 22 ps form A or B to either output. Each flip flop has a setup time of 30 ps, a hold time of 10 ps, a clock to Q propagation delay of 35 ps, and a clock to Q contamination delay of 21 ps.

A) If there is no clock skew, what is the maximum operating frequency of the circuit?
B) How much clock skew can the circuit tolerate if it must operate at 8 GHz?
C) How much clock skew can the circuit tolerate before it might experience a hold time violation?

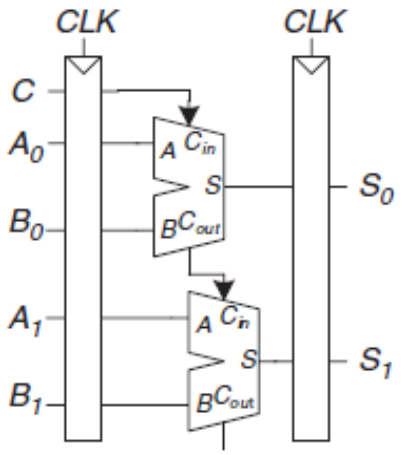


Figure 3.75 2-bit adder schematic for Exercise 3.34

Solution

A) $T_c \geq t_{pcq}+t_{pd}+t_{setup}=35\,ps+(25\,ps+30\,ps)+30\,ps=120\,ps$
$f_c=1/T_c=1/120\,ps=8.33\,GHz$

B) $T_c=1/f_c=1/8\,GHz=125\,ps$
$t_{skew}\leq T_c-(t_{pcq}+t_{setup}+t_{pd})=125\,ps-(35\,ps+(25\,ps+30\,ps)+30\,ps)=5\,ps$

C) $t_{skew}\leq t_{ccq}+t_{cd}-t_{hold}=21\,ps+(15\,ps+22\,ps)-10\,ps=48\,ps$

Exercise 3.35

A field programmable gate array (FPGA) uses configurable logic blocks (CLBs) rather than logic gates to implement combinational logic. The Xilinx Spartan 3 FPGA has propagation and contamination delays of 0.61 and 0.30 ns, respectively, for each CLB. It also contains flip flops with propagation and contamination delays of 0.72 and 0.50 ns, and setup and hold times of 0.53 and 0 ns, respectively.

A) If you are building a system that needs to run at 40 MHz, how many consecutive CLBs can you use between two flip flops? Assume that there is no clock skew and no delay through wires between CLBs.

B) Suppose that all paths between flip flops pass through at least one CLB. How much clock skew can the FPGA have without violating the hold time?

Solution

A) $T_c=1/f_c=1/40\,MHz=25\,ns$
$T_{pd}\leq T_c-(t_{pcq}+t_{setup})=25\,ns-(0.72\,ns+0.53\,ns)=23.75\,ns$
$N_{CLB}=T_{pd}/t_{pd}=23.75\,ns/0.61\,ns=38\,CLBs$

B) $t_{skew}\leq t_{ccq}+t_{cd}-t_{hold}=0.50\,ns+0.30\,ns-0\,ns=0.80\,ns$

Exercise 3.36

A synchronizer is built from a pair of flip flops with t_{setup}=50ps, T_0=20ps and τ=30ps. It samples an asynchronous input that changes 10^8 times per second. What is the minimum clock period of the synchronizer to achieve a mean time between failures (MTBF) of 100 years?

Solution

This equation is impossible to solve directly so it's necessary do some approximations

$$3.15*10^{9}s=\frac{T_C*e^{\frac{T_C-50*10^{-12}s}{30*10^{-12}s}}}{10^{8}t/s*20*10^{-12}s}$$
$$T_c\approx 1.138\,ns$$
$$f_c=1/T_c=0.88\,GHz$$

Exercise 3.37

You would like to build a synchronizer that can receive asynchronous inputs with an MTBF of 50 years. Your system is running at 1GHz, and you use a sampling flip flops with τ=100ps, T_0=110ps, t_{setup}=70ps,and . The synchronizer receives a new asynchronous input on average 0.5 times per second (i.e., once every 2 seconds). What is the required probability of failure to satisfy this MTBF? How many clock cycles would you have to wait before reading the sampled input signal to give that probability of error?

Solution

$$P(failure)/sec=1/MTBF=1/(1.58*10^{9}s)=6.34*10^{-10}$$
$$1.58*10^{9}s=\frac{T_C*e^{\frac{T_C-70*10^{-12}s}{100*10^{-12}s}}}{0.5t/s*110*10^{-12}s}$$
$$T_c\approx 1.837\,ns$$
$$f_c=1/T_c=0.54\,GHz$$

So because the system is running at 1GHz it needs to wait 2 clock cycles to assure this MTBF.

Exercise 3.38

You are walking down the hallway when you run into your lab partner walking in the other direction. The two of you first step one way and are still in each other's way. Then you both wait a bit, hoping the other person will step aside. You can model this situation as a metastable point and apply the same theory that has been applied to synchronizers and flip flops. Suppose you create a mathematical model for yourself and your lab partner. You start the unfortunate encounter in the metastable state. The probability that you remain in this state after t seconds is $e^{-t/\tau}$. τ indicates your response rate; today, your brian has been blurred by lack of sleep and has τ=20ps.

A) How long will it be until you have 99% of certainty that you will have resolved from metastability (i.e., figured out how to pass one another)?
B) You are not only sleepy, but also ravenously hungry. In fact, you will starve to death if you don't get to the cafeteria within 3 minutes. What is the probability that your lab partner will have to drag you to the morgue?

Solution

A) $P(failure)=e^{-t/\tau}$
$\ln(P(failure))=-t/\tau$
$t=-\tau*\ln(P(failure))=-20\,sec*\ln(0.01)=92.103\,sec$
B) $P(failure)=e^{-t/\tau}=e^{-(180\,sec/20\,sec)}=0.0001$

Exercise 3.39

You have built a synchronizer using flip-flops with T_0=20ps and τ=30ps. Your boss tells you that you need to increase the MTBF by a factor of 10. By how much do you need to increase the clock period?

Solution

$$\frac{MTBF_F}{MTBF_I}=10=\frac{\frac{T_{CF}*e^{\frac{T_{CF}-t_{setup}}{30\ ps}}}{N*20\ ps}}{\frac{T_{CI}*e^{\frac{T_{CI}-t_{setup}}{30\ ps}}}{N*20\ ps}}=\frac{T_{CF}*e^{\frac{T_{CF}}{30\ ps}}*e^{\frac{-t_{setup}}{30\ ps}}}{T_{CI}*e^{\frac{T_{CI}}{30\ ps}}*e^{\frac{-t_{setup}}{30\ ps}}}=\frac{T_{CF}}{T_{CI}}*e^{(T_{CF}-T_{CI})/30\ ps}$$

$$\ln(10)=\ln\left(\frac{T_{CF}}{T_{CI}}*e^{(T_{CF}-T_{CI})/30\ ps}\right)$$

$$\ln(10)=\ln\left(\frac{T_{CF}}{T_{CI}}\right)+(T_{CF}-T_{CI})/30\ ps$$

$$T_{CF}-T_{CI}=30\ ps*[\ln(10)-\ln\left(\frac{T_{CF}}{T_{CI}}\right)]$$

If we assume that the ratio between T_{CF} and T_{CI} is pretty close to 1 because it changes too little then we get that

$T_{CF}-T_{CI}\approx 69\ ps$

Exercise 3.40

Ben Bitdiddle invents a new and improved synchronizer in Figure 3.76 that he claims eliminates metastability in a single cycle. He explains that the circuit in box M is an analog "metastability detector" that produces a HIGH output if the voltage is in the forbidden zone between V_{IL} and V_{IH}. The metastability detector checks to determine whether the first flip flop has produced a metastable output on D2. If so, it asynchronously resets the flip flop to produce a good 0 at D2. The second flip flop then samples D2, always producing a valid logic level on Q. Alyssa P. Hacker tells Ben that there must be a bug in the circuit, because eliminating metastability is just as impossible as building a perpetual motion machine. Who is right? Explain, showing Ben's error or showing why Alyssa is wrong.

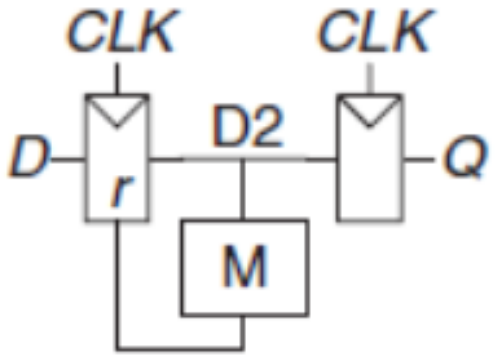


Figure 3.76 "New and improved" synchronizer for Exercise 3.40

Solution

It's totally impossible to avoid the metastability of an asynchronous input, because there is some way that the M box could be metastable if the first flip flop resolves the value of metastability when M activates then M becomes metastable because it will be between 2 ranges of values. If M is metastable the restet value it will aslo be metastable

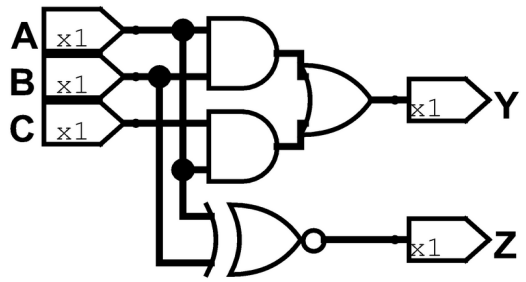
Chapter 4

Exercise 4.1

Sketch a schematic of the circuit described by the following HDL code. Simplify the schematic so that it shows a minimum number of gates.

```
 1 module exercise1
 2     (input logic a, b, c,
 3     output logic y, z);
 4         assign y = a & b & c | a & b & ~c | a & ~b & c;
 5         assign z = a & b | ~a & ~b;
 6 endmodule
```

Solution

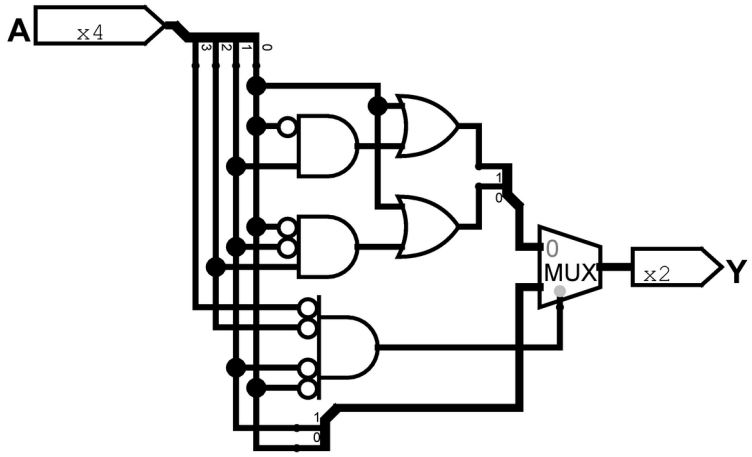


Exercise 4.2

Sketch a schematic of the circuit described by the following HDL code. Simplify the schematic so that it shows a minimum number of gates.

```
 1 module exercise2
 2     (input logic [3:0] a,
 3     output logic [1:0] y);
 4
 5     always_comb
 6         if (a[0]) y = 2'b11;
 7         else if (a[1]) y = 2'b10;
 8         else if (a[2]) y = 2'b01;
 9         else if (a[3]) y = 2'b00;
10         else y = a[1:0];
11 endmodule
```

Solution



Exercise 4.3

Write an HDL module that computes a four input XOR function. The input is $a_{3:0}$ and the output is y.

Solution

```
 1 module xor4
 2     (input logic[3:0] a,
 3     output logic y);
 4
 5     always_comb begin
 6         y = ^a;
 7     end
 8 endmodule
```

Exercise 4.4

Write a self checking testbench for Exercise 4.3. Create a test vector file containing all 16 test cases. Simulate the circuit and show that it works. Introduce an error in the test vector and show that the testbench reports a mismatch.

Solution

verilator --binary -j 0 --trace –assert <DESIGN_CODE> <TESTBENCH>

```
 1 module tb_xor4();
 2     localparam vindx = 16;
 3     logic[3:0] a;
 4     logic y, y_exp;
 5     logic[4:0] vector_in[0 : vindx - 1];
 6     xor4 dut(.a(a), .y(y));
 7     initial
 8     begin
 9         $readmemb("input_xor4.txt", vector_in);
10         $dumpfile("waveform_xor4.vcd");
11         $dumpvars(0, tb_xor4);
12         for (integer i=0; i < vindx; i++)
13         begin
14             {a, y_exp} = vector_in[i]; #10;
15             assert (y === y_exp) else $error("Failed at line %d: input %b
output %b", (i+1), a, y);
16         end
```

```
17     end
18 endmodule
```

Exercise 4.5

Write an HDL module called *minority*. It receives three inputs *a*, *b* and *c*. It produces one output y that is TRUE if at least two of the inputs are FALSE.

Solution

```
 1 module minority
 2     (input logic a, b, c,
 3     output logic y);
 4
 5     always_comb begin
 6         if ((~a & ~b) | (~b & ~c) | (~c & ~a))
 7             y = 1'b1;
 8         else
 9             y = 1'b0;
10     end
11 endmodule
```

Exercise 4.6

Write an HDL module for a hexadecimal seven-segment display decoder. The decoder should handle the digits A, B, C, D, E, and F, as well as 0–9.

Solution

```
 1 module seven_seg
 2     (input logic[3:0] data,
 3     output logic[6:0] seg);
 4
 5     always_comb begin
 6         case(data)
 7             'h0 : seg = 7'b1111110;
 8             'h1 : seg = 7'b0110000;
 9             'h2 : seg = 7'b1101101;
10             'h3 : seg = 7'b1111001;
11             'h4 : seg = 7'b0110011;
12             'h5 : seg = 7'b1011011;
13             'h6 : seg = 7'b1011111;
14             'h7 : seg = 7'b1110000;
15             'h8 : seg = 7'b1111111;
16             'h9 : seg = 7'b1111011;
17             'hA : seg = 7'b1110111;
18             'hB : seg = 7'b0011111;
19             'hC : seg = 7'b1001110;
20             'hD : seg = 7'b0111101;
21             'hE : seg = 7'b1001111;
22             'hF : seg = 7'b1000111;
23         endcase
24     end
25 endmodule
```

Exercise 4.7

Write a self-checking testbench for Exercise 4.6. Create a test vector file containing all 16 test cases. Simulate the circuit and show that it works. Introduce an error in the test vector file and show that the testbench reports a mismatch.

Solution

```
 1 module tb_seven_seg();
 2     localparam vindx = 16;
 3     logic[3:0] data;
 4     logic[6:0] seg, seg_exp;
 5     logic[10:0] vector_in[0 : vindx - 1];
 6     seven_seg dut(.data(data), .seg(seg));
 7     initial
 8     begin
 9         $readmemb("input_seven_seg.txt", vector_in);
10         $dumpfile("waveform_seven_seg.vcd");
11         $dumpvars(0, tb_seven_seg);
12         for (integer i=0; i < vindx; i++)
13         begin
14             {data, seg_exp} = vector_in[i]; #10;
15             assert (seg === seg_exp) else $error("Failed at line %d : input %b
output %b", (i+1), data, seg);
16         end
17     end
18 endmodule
```

Exercise 4.8

Write an 8:1 multiplexer module called *mux8* with inputs $s_{2:0}$, *d0*, *d1*, *d2*, *d3*, *d4*, *d5*, *d6*, *d7*, and output *y*.

Solution

```
 1 module mux8
 2     (input logic[2:0] sel,
 3     input logic d7, d6, d5, d4, d3, d2, d1, d0,
 4     output logic y);
 5
 6     always_comb begin
 7         case(sel)
 8             3'b111: y = d7;
 9             3'b110: y = d6;
10             3'b101: y = d5;
11             3'b100: y = d4;
12             3'b011: y = d3;
13             3'b010: y = d2;
14             3'b001: y = d1;
15             3'b000: y = d0;
16             default: y=1'bx;
17         endcase
18     end
19 endmodule
```

Exercise 4.9

Write a structural module to compute the logic function $Y = a \cdot \bar{b} + \bar{b} \cdot \bar{c} + \bar{a} \cdot b \cdot c$ using multiplexer logic. Use the 8:1 multiplexer from Exercise 4.8.

Solution

```
 1 module logic_mux8
 2     (input logic a, b, c,
 3     output logic y);
 4
 5     logic d7, d6, d5, d4, d3, d2, d1, d0;
 6     logic[2:0] sel;
 7
 8     mux8 mux(.sel(sel), .d7(d7), .d6(d6), .d5(d5), .d4(d4), .d3(d3), .d2(d2),
.d1(d1), .d0(d0), .y(y));
 9
10     always_comb begin
11         sel = {a ,b ,c};
12         {d7, d6, d5, d4, d3, d2, d1, d0} = 8'b00111001;
13     end
14 endmodule
```

Exercise 4.10

Repeat Exercise 4.9 using a 4:1 multiplexer and as many NOT gates as you need.

Solution

```
 1 module logic_mux4
 2     (input logic a, b, c,
 3     output logic y);
 4
 5     logic d3, d2, d1, d0;
 6     logic[1:0] sel;
 7
 8     mux4 mux(.sel(sel), .d3(d3), .d2(d2), .d1(d1), .d0(d0), .y(y));
 9
10     always_comb begin
11         sel = {a ,c};
12         d3 = ~b;
13         d2 = ~b;
14         d1 = b;
15         d0 = ~b;
16     end
17 endmodule
```

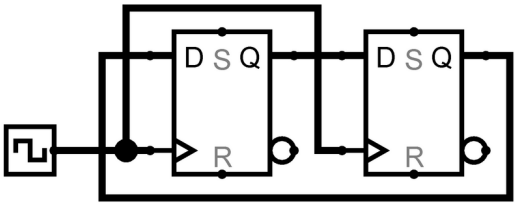
Exercise 4.11

Section 4.5.4 pointed out that a synchronizer could be correctly described with blocking assignments if the assignments were given in the proper order. Think of a simple sequential circuit that cannot be correctly described with blocking assignments, regardless of order.

Solution

One circuit that presents this problem is when you have 2 flip flops which share the same clock signal, it happens because both flip flops should pass the current value at the same time but because the

process is sequential it can’t execute both flip flop assignments at the same time so it may happen that the value losses.



Exercise 4.12

Write an HDL module for an eight-input priority circuit.

Solution

```
 1 module priority8
 2     (input logic[7:0] a,
 3     output logic[2:0] y);
 4
 5     always_comb begin
 6         casez (a)
 7             8'b1??????? : y = 3'b111;
 8             8'b01?????? : y = 3'b110;
 9             8'b001????? : y = 3'b101;
10             8'b0001???? : y = 3'b100;
11             8'b00001??? : y = 3'b011;
12             8'b000001?? : y = 3'b010;
13             8'b00000001? : y = 3'b001;
14             8'b00000001 : y = 3'b000;
15             default : y = 3'bxxx;
16         endcase
17     end
18 endmodule
```

Exercise 4.13

Write an HDL module for a 2:4 decoder.

Solution

```
 1 module decoder2_4
 2     (input logic[1:0] a,
 3     output logic[3:0] y);
 4
 5     always_comb begin
 6         case (a)
 7             2'b11 : y = 4'b1000;
 8             2'b10 : y = 4'b0100;
 9             2'b01 : y = 4'b0010;
10             2'b00 : y = 4'b0001;
11             default : y = 4'bx;
12         endcase
13     end
14 endmodule
```

Exercise 4.14

Write an HDL module for a 6:64 decoder using three instances of the 2:4 decoders from Exercise 4.13 and a bunch of three-input AND gates.

Solution

```
module decoder6_64
    (input logic[5:0] a,
    output logic[63:0] y);

    logic[11:0] layer;
    logic[63:0] gen_struct;
    genvar i, j, k;
    decoder2_4 dec2(.a(a[5:4]), .y(layer[11:8]));
    decoder2_4 dec1(.a(a[3:2]), .y(layer[7:4]));
    decoder2_4 dec0(.a(a[1:0]), .y(layer[3:0]));

    generate
        for(i=8; i<12; i++) begin
            for(j=4; j<8; j++) begin
                for(k=0; k<4; k++) begin
                    assign gen_struct[((i-8)*16+(j-4)*4+k)] = layer[i] &
layer[j] & layer[k];
                end
            end
        end
    endgenerate

    always_comb begin
        y = gen_struct;
    end
endmodule
```

Exercise 4.15

Write HDL modules that implement the Boolean equations from Exercise 2.13.

Solution

```
module bool_eq
    (input logic a, b, c, d,
    output logic y2, y1, y0);

    always_comb begin
        y2 = (a & c) | (~a & ~b & c);
        y1 = (~a & ~b) | (~a & b & ~c) | ~(a | ~c);
        y0 = (~a & ~b & ~c & ~d) | (a & ~b & ~c) | (a & ~b & c & ~d)
        | (a & b & d) |(~a & ~b & c & ~d) | (b & ~c & d) | ~a;
    end
endmodule
```

Exercise 4.16

Write an HDL module that implements the circuit from Exercise 2.26.

Solution

```
module bubble_push1
```

```
 2     (input logic a, b, c, d, e,
 3     output logic y);
 4
 5     always_comb begin
 6         y = ((a ~& b) ~& (c ~& d)) ~& e;
 7     end
 8 endmodule
```

Exercise 4.17

Write an HDL module that implements the circuit from Exercise 2.27.

Solution

```
 1 module bubble_push2
 2     (input logic a, b, c, d, e, f, g,
 3     output logic y);
 4
 5     always_comb begin
 6         y = ~& (~(~(~(a & b & c) & d) | ~(e | (f & g))));
 7     end
 8 endmodule
```

Exercise 4.18

Write an HDL module that implements the logic function from Exercise 2.28. Pay careful attention to how you handle don’t cares.

Solution

```
 1 module dont_care
 2     (input logic a, b, c, d,
 3     output logic y);
 4
 5     always_comb begin
 6         case ({a, b, c, d})
 7             4'b0000 : y = 1'bx;
 8             4'b0001 : y = 1'bx;
 9             4'b0010 : y = 1'bx;
10             4'b0011 : y = 1'b0;
11             4'b0100 : y = 1'b0;
12             4'b0101 : y = 1'bx;
13             4'b0110 : y = 1'b0;
14             4'b0111 : y = 1'bx;
15             4'b1000 : y = 1'b1;
16             4'b1001 : y = 1'b0;
17             4'b1010 : y = 1'bx;
18             4'b1011 : y = 1'b1;
19             4'b1100 : y = 1'b1;
20             4'b1101 : y = 1'b1;
21             4'b1110 : y = 1'bx;
22             4'b1111 : y = 1'b1;
23         endcase
24     end
25 endmodule
```

Exercise 4.19

Write an HDL module that implements the functions from Exercise 2.35.

Solution

```
module prime_division
    (input logic[3:0] a,
    output logic p, d);

    always_comb begin
        case (a)
            4'b0000 : {p, d} = 2'b01;
            4'b0001 : {p, d} = 2'b00;
            4'b0010 : {p, d} = 2'b10;
            4'b0011 : {p, d} = 2'b11;
            4'b0100 : {p, d} = 2'b00;
            4'b0101 : {p, d} = 2'b10;
            4'b0110 : {p, d} = 2'b01;
            4'b0111 : {p, d} = 2'b10;
            4'b1000 : {p, d} = 2'b00;
            4'b1001 : {p, d} = 2'b01;
            4'b1010 : {p, d} = 2'b00;
            4'b1011 : {p, d} = 2'b10;
            4'b1100 : {p, d} = 2'b01;
            4'b1101 : {p, d} = 2'b10;
            4'b1110 : {p, d} = 2'b00;
            4'b1111 : {p, d} = 2'b01;
            default : {p, d} = 2'bxx;
        endcase
    end
endmodule
```

Exercise 4.20

Write an HDL module that implements the priority encoder from Exercise 2.36.

Solution

```
module priority_enc
    (input logic[7:0] a,
    output logic[2:0] y,
    output logic none);

    always_comb begin
        casez (a)
            8'b00000000:
                y = 3'b000;
            8'b00000001:
                y = 3'b000;
            8'b00000001?:
                y = 3'b001;
            8'b000001??:
                y = 3'b010;
            8'b00001???:
                y = 3'b011;
            8'b0001????:
                y = 3'b100;
            8'b001?????:
                y = 3'b101;
```

```
22              8'b01??????:
23                  y = 3'b110;
24              8'b1???????:
25                  y = 3'b111;
26          endcase
27          none = ~|a;
28      end
29  endmodule
```

Exercise 4.21

Write an HDL module that implements the modified priority encoder from Exercise 2.37.

Solution

```
 1  module priority_mod
 2      (input logic[7:0] a,
 3      output logic[2:0] z,
 4      output logic none);
 5
 6      always_comb begin
 7          if((a[7] & a[6])==1'b1)
 8              z = 3'b110;
 9          else if(((a[7] | a[6]) & a[5])==1'b1)
10              z = 3'b101;
11          else if(((a[7] | a[6] | a[5]) & a[4])==1'b1)
12              z = 3'b100;
13          else if(((a[7] | a[6] | a[5] | a[4]) & a[3])==1'b1)
14              z = 3'b011;
15          else if(((a[7] | a[6] | a[5] | a[4] | a[3]) & a[2])==1'b1)
16              z = 3'b010;
17          else if(((a[7] | a[6] | a[5] | a[4] | a[3] | a[2]) & a[1])==1'b1)
18              z = 3'b001;
19          else
20              z = 3'b000;
21
22          none = ~|a;
23      end
24  endmodule
```

Exercise 4.22

Write an HDL module that implements the binary-to-thermometer code converter from Exercise 2.38.

Solution

```
 1  module thermometer
 2      #(parameter N = 3)
 3      (input logic[N-1:0] a,
 4      output logic[2**N-1:0] y);
 5
 6      always_comb begin
 7          y = 'b0;
 8          for(integer i = {{32-N{1'b0}}, a}; i >= 0; i--) begin
 9              y[i] = 1'b1;
10          end
11      end
12  endmodule
```

Exercise 4.23

Write an HDL module implementing the days-in-month function from Question 2.2.

Solution

```
module days_month
    (input logic[3:0] a,
    output logic y);

    always_comb begin
        case(a)
            4'b0001 : y = 1'b1;
            4'b0010 : y = 1'b0;
            4'b0011 : y = 1'b1;
            4'b0100 : y = 1'b0;
            4'b0101 : y = 1'b1;
            4'b0110 : y = 1'b0;
            4'b0111 : y = 1'b1;
            4'b1000 : y = 1'b1;
            4'b1001 : y = 1'b0;
            4'b1010 : y = 1'b1;
            4'b1011 : y = 1'b0;
            4'b1100 : y = 1'b1;
            default : y = 1'bx;
        endcase
    end
endmodule
```

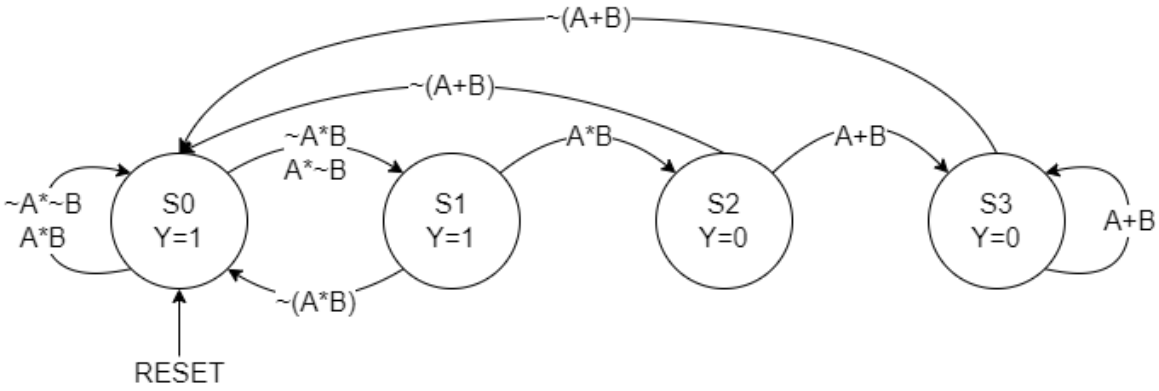
Exercise 4.24

Sketch the state transition diagram for the FSM described by the following HDL code.

```
module fsm2
    (input logic clk, reset,
    input logic a, b,
    output logic y);

    logic [1:0] state, nextstate;
    parameter S0 = 2'b00;
    parameter S1 = 2'b01;
    parameter S2 = 2'b10;
    parameter S3 = 2'b11;

    always_ff @(posedge clk, posedge reset) begin
        if (reset) state <= S0;
        else state <= nextstate;
    end

    always_comb begin
        case (state)
            S0: if (a ^ b) nextstate = S1;
            else nextstate = S0;
            S1: if (a & b) nextstate = S2;
            else nextstate = S0;
            S2: if (a | b) nextstate = S3;
            else nextstate = S0;
```

```
25             S3: if (a | b) nextstate = S3;
26             else nextstate = S0;
27         endcase
28         assign y = (state == S1) | (state == S2);
29     end
30 endmodule
```

Solution



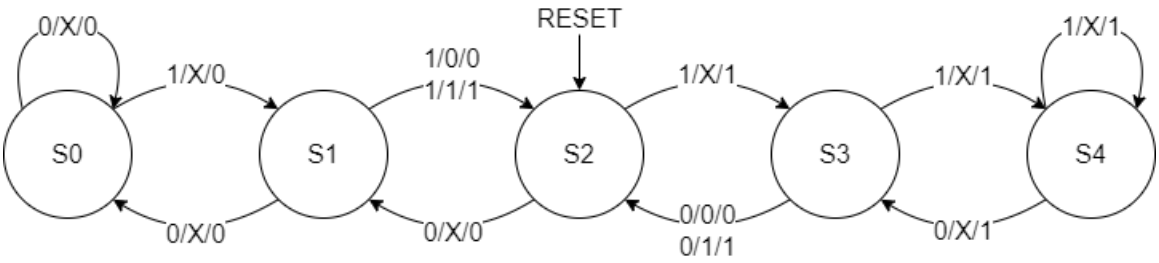
Exercise 4.25

Sketch the state transition diagram for the FSM described by the following HDL code. An FSM of this nature is used in a branch predictor on some microprocessors.

```
 1 module fsm1
 2     (input logic clk, reset,
 3     input logic taken, back,
 4     output logic predicttaken);
 5
 6     logic [4:0] state, nextstate;
 7     parameter S0 = 5'b00001;
 8     parameter S1 = 5'b00010;
 9     parameter S2 = 5'b00100;
10     parameter S3 = 5'b01000;
11     parameter S4 = 5'b10000;
12
13     always_ff @(posedge clk, posedge reset) begin
14         if (reset) state <= S2;
15         else state <= nextstate;
16     end
17
18     always_comb begin
19         case (state)
20             S0: if (taken) nextstate = S1;
21             else nextstate = S0;
22             S1: if (taken) nextstate = S2;
23             else nextstate = S0;
24             S2: if (taken) nextstate = S3;
25             else nextstate = S1;
26             S3: if (taken) nextstate = S4;
27             else nextstate = S2;
28             S4: if (taken) nextstate = S4;
29             else nextstate = S3;
30             default: nextstate = S2;
31         endcase
```

```
 32         assign predicttaken = (state == S4) | (state == S3) | (state == S2 &
back);
 33     end
 34 endmodule
```

Solution



Exercise 4.26

Write an HDL module for an SR latch.

Solution

```
 1 module sr_latch
 2     (input logic s, r,
 3     output logic q, nq);
 4
 5     assign nq = s ~| q;
 6     assign q = r ~| nq;
 7 endmodule
```

Exercise 4.27

Write an HDL module for a JK flip-flop. The flip-flop has inputs, *clk*, *J*, and *K*, and output *Q*. On the rising edge of the clock, *Q* keeps its old value if $J = K = 0$. It sets *Q* to 1 if $J = 1$, resets *Q* to 0 if $K = 1$, and inverts *Q* if $J = K = 1$.

Solution

```
 1 module jk_flip_flop
 2     (input logic j, k, clk,
 3     output logic q);
 4
 5     always_ff @(posedge clk) begin
 6         case ({j, k})
 7             2'b00 : q <= q;
 8             2'b01 : q <= 1'b0;
 9             2'b10 : q <= 1'b1;
10             2'b11 : q <= ~q;
11             default : q <= 1'b0;
12         endcase
13     end
14 endmodule
```

Exercise 4.28

Write an HDL module for the latch from Figure 3.18. Use one assignment statement for each gate. Specify delays of 1 unit or 1 ns to each gate. Simulate the latch and show that it operates correctly.

Then, increase the inverter delay. How long does the delay have to be before a race condition causes the latch to malfunction?

Solution

If we apply the same delays to all gates then it will malfunction. The delay of nclk cannot be more than n1 and n2 gates and q gate cannot be delay more than n1 and n2. And n2 delay should be grater than nclk and q together.

```
 1 module improved_latch
 2     (input logic d, clk,
 3     output logic q);
 4
 5     logic n1, n2, nclk;
 6
 7     assign  #5 n1 = clk & d;
 8     assign  #5 n2 = nclk & q;
 9     assign  #3 q = n1 | n2;
10     assign  #1 nclk = ~clk;
11 endmodule
```

Exercise 4.29

Write an HDL module for the traffic light controller from Section 3.4.1.

Solution

```
 1 module lights
 2     (input logic ta, tb,
 3     input logic clk, reset,
 4     output logic[1:0] la, lb);
 5
 6     typedef enum logic[1:0] {S0, S1, S2, S3} statetype;
 7     statetype state, nextstate;
 8
 9     always_ff @(posedge clk, posedge reset) begin
10         if (reset)
11             state <= S0;
12         else
13             state <= nextstate;
14     end
15
16     always_comb begin
17         case (state)
18             S0 : if (~ta) nextstate = S1;
19                  else nextstate = S0;
20             S1 : nextstate = S2;
21             S2 : if (~tb) nextstate = S3;
22                  else nextstate = S2;
23             S3 : nextstate = S0;
24         endcase
25     end
26
27     assign la = (state == S2 || state == S3) ? 2'b10 :
28                 (state == S1) ? 2'b01 : 2'b00;
29
30     assign lb = (state == S0 || state == S1) ? 2'b10 :
31                 (state == S3) ? 2'b01 : 2'b00;
32 endmodule
```

Exercise 4.30

Write three HDL modules for the factored parade mode traffic light controller from Example 3.8. The modules should be called controller, mode, and lights, and they should have the inputs and outputs shown in Figure 3.33(b).

Solution

```
module lights
    (input logic ta, tb, m,
    input logic clk, reset,
    output logic[1:0] la, lb);

    typedef enum logic[1:0] {S0, S1, S2, S3} statetype;
    statetype state, nextstate;

    always_ff @(posedge clk, posedge reset) begin
        if (reset)
            state <= S0;
        else
            state <= nextstate;
    end

    always_comb begin
        case (state)
            S0 : if (~ta) nextstate = S1;
                 else nextstate = S0;
            S1 : nextstate = S2;
            S2 : if (~tb && ~m) nextstate = S3;
                 else nextstate = S2;
            S3 : nextstate = S0;
        endcase
    end

    assign la = (state == S2 || state == S3) ? 2'b10 :
                (state == S1) ? 2'b01 : 2'b00;

    assign lb = (state == S0 || state == S1) ? 2'b10 :
                (state == S3) ? 2'b01 : 2'b00;
endmodule
```

Exercise 4.31

Write an HDL module describing the circuit in Figure 3.42.

Solution

```
module timming
    (input logic a, b, c, d,
    input logic clk,
    output logic x, y);

    logic a_reg, b_reg, c_reg, d_reg;
    logic n1, n2, n3;

    always_ff @(posedge(clk)) begin
        {a_reg, b_reg, c_reg, d_reg} <= {a, b, c, d};
        {x, y} <= {n2, n3};
    end
```

```
13
14      assign n1 = (a_reg & b_reg);
15      assign n2 = (n1 | c_reg);
16      assign n3 = ~(n2 | d_reg);
17  endmodule
```

Exercise 4.32

Write an HDL module for the FSM with the state transition diagram given in Figure 3.69 from Exercise 3.22.

Solution

```
 1  module moore_diag
 2      (input logic a, b,
 3      input logic clk, reset,
 4      output logic q);
 5
 6      typedef enum logic[1:0] {S0, S1, S2} statetype;
 7      statetype state, nextstate;
 8
 9      always_ff @(posedge clk, posedge reset) begin
10          if (reset)
11              state <= S0;
12          else
13              state <= nextstate;
14      end
15
16      always_comb begin
17          case (state)
18              S0 : if (a) nextstate = S1;
19                  else if (~a) nextstate = S0;
20              S1 : if (b) nextstate = S2;
21                  else if (~b) nextstate = S0;
22              S2 : nextstate = S0;
23              default: nextstate = S0;
24          endcase
25      end
26
27      assign q = (state == S2);
28  endmodule
```

Exercise 4.33

Write an HDL module for the FSM with the state transition diagram given in Figure 3.70 from Exercise 3.23.

Solution

```
 1  module mealy_diag
 2      (input logic a, b,
 3      input logic clk, reset,
 4      output logic q);
 5
 6      typedef enum logic[1:0] {S0, S1, S2} statetype;
 7      statetype state, nextstate;
 8
 9      always_ff @(posedge clk, posedge reset) begin
```

```
10         if (reset)
11             state <= S0;
12         else
13             state <= nextstate;
14     end
15
16     always_comb begin
17         case (state)
18             S0 : if (a) nextstate = S1;
19                 else if (~a) nextstate = S0;
20             S1 : if (b) nextstate = S2;
21                 else if (~b) nextstate = S0;
22             S2 : if (a & b) nextstate = S2;
23                 else nextstate = S0;
24             default: nextstate = S0;
25         endcase
26     end
27
28     assign q = (state == S2 & {a, b} == 2'b11);
29 endmodule
```

Exercise 4.34

Write an HDL module for the improved traffic light controller from Exercise 3.24.

Solution

```
 1 module improved_lights
 2     (input logic ta, tb,
 3     input logic clk, reset,
 4     output logic[1:0] la, lb);
 5
 6     typedef enum logic[2:0] {S0, S1, S2, S3, S4, S5} statetype;
 7     statetype state, nextstate;
 8
 9     always_ff @(posedge clk, posedge reset) begin
10         if (reset)
11             state <= S0;
12         else
13             state <= nextstate;
14     end
15
16     always_comb begin
17         case (state)
18             S0 : if (~ta) nextstate = S1;
19                 else nextstate = S0;
20             S1 : nextstate = S2;
21             S2 : nextstate = S3;
22             S3 : if (~tb) nextstate = S4;
23                 else nextstate = S3;
24             S4 : nextstate = S5;
25             S5 : nextstate = S0;
26             default: nextstate = S0;
27         endcase
28     end
29
30     assign la = (state == S2 || state == S3 || state == S4 || state == S5) ?
2'b10 :
31                 (state == S1) ? 2'b01 : 2'b00;
```

```
 32
 33     assign lb = (state == S0 || state == S1 || state == S2 || state == S5) ?
2'b10 :
 34                 (state == S4) ? 2'b01 : 2'b00;
 35 endmodule
```

Exercise 4.35

Write an HDL module for the daughter snail from Exercise 3.25.

Solution

```
  1 module improved_snail
  2     (input logic a,
  3     input logic clk, reset,
  4     output logic y);
  5
  6     typedef enum logic[2:0] {S0, S1, S2, S3, S4} statetype;
  7     statetype state, nextstate;
  8
  9     always_ff @(posedge clk, posedge reset) begin
 10         if (reset)
 11             state <= S0;
 12         else
 13             state <= nextstate;
 14     end
 15
 16     always_comb begin
 17         case (state)
 18             S0 : if (a) nextstate = S1;
 19                  else nextstate = S0;
 20             S1 : if (a) nextstate = S2;
 21                  else nextstate = S0;
 22             S2 : if (a) nextstate = S4;
 23                  else nextstate = S3;
 24             S3 : if (a) nextstate = S1;
 25                  else nextstate = S0;
 26             S4 : if (a) nextstate = S4;
 27                  else nextstate = S3;
 28             default: nextstate = S0;
 29         endcase
 30     end
 31
 32     assign y = ((state == S4 && a == 1'b0) || (state == S3 && a == 1'b1)) ?
1'b1 : 1'b0;
 33 endmodule
```

Exercise 4.36

Write an HDL module for the soda machine dispenser from Exercise 3.26.

Solution

```
  1 module dispenser
  2     (input logic N, D, Q,
  3     input logic clk, reset,
  4     output logic R2D, RD, RN, DISP);
  5
  6     typedef enum logic[3:0] {S0, S1, S2, S3, S4, S5, S6, S7, S8, S9} statetype;
```

```
 7     statetype state, nextstate;
 8
 9     always_ff @(posedge clk, posedge reset) begin
10         if (reset)
11             state <= S0;
12         else
13             state <= nextstate;
14     end
15
16     always_latch begin
17         case (state)
18             S0 : if (Q) nextstate = S5;
19                  else if (D) nextstate = S2;
20                  else if (N) nextstate = S1;
21                  else nextstate = nextstate;
22             S1 : if (Q) nextstate = S6;
23                  else if (D) nextstate = S3;
24                  else if (N) nextstate = S2;
25                  else nextstate = nextstate;
26             S2 : if (Q) nextstate = S7;
27                  else if (D) nextstate = S4;
28                  else if (N) nextstate = S3;
29                  else nextstate = nextstate;
30             S3 : if (Q) nextstate = S8;
31                  else if (D) nextstate = S5;
32                  else if (N) nextstate = S4;
33                  else nextstate = nextstate;
34             S4 : if (Q) nextstate = S9;
35                  else if (D) nextstate = S6;
36                  else if (N) nextstate = S5;
37                  else nextstate = nextstate;
38             S5 : nextstate = S0;
39             S6 : nextstate = S0;
40             S7 : nextstate = S0;
41             S8 : nextstate = S0;
42             S9 : nextstate = S0;
43             default: nextstate = nextstate;
44         endcase
45     end
46
47     assign DISP = (state == S5 || state == S6 || state == S7 || state == S8 ||
state == S9) ? 1'b1 : 1'b0;
48     assign R2D = (state == S9) ? 1'b1 : 1'b0;
49     assign RD = (state == S7 || state == S8) ? 1'b1 : 1'b0;
50     assign RN = (state == S6 || state == S8) ? 1'b1 : 1'b0;
51 endmodule
```

Exercise 4.37

Write an HDL module for the Gray code counter from Exercise 3.27.

Solution

```
 1 module gray_code
 2     (input logic clk, reset,
 3     output logic[2:0] y);
 4
 5     logic[2:0] state, nextstate;
 6
```

```
 7     parameter S0 = 3'b000;
 8     parameter S1 = 3'b001;
 9     parameter S2 = 3'b011;
10     parameter S3 = 3'b010;
11     parameter S4 = 3'b110;
12     parameter S5 = 3'b111;
13     parameter S6 = 3'b101;
14     parameter S7 = 3'b100;
15
16     always_ff @(posedge clk, posedge reset) begin
17         if (reset)
18             state <= S0;
19         else
20             state <= nextstate;
21     end
22
23     always_comb begin
24         case (state)
25             S0 : nextstate = S1;
26             S1 : nextstate = S2;
27             S2 : nextstate = S3;
28             S3 : nextstate = S4;
29             S4 : nextstate = S5;
30             S5 : nextstate = S6;
31             S6 : nextstate = S7;
32             S7 : nextstate = S0;
33         endcase
34     end
35
36     assign y = state;
37 endmodule
```

Exercise 4.38

Write an HDL module for the UP/DOWN Gray code counter from Exercise 3.28.

Solution

```
 1 module gray_code_up
 2     (input logic up,
 3     input logic clk, reset,
 4     output logic[2:0] y);
 5
 6     logic[2:0] state, nextstate;
 7
 8     parameter S0 = 3'b000;
 9     parameter S1 = 3'b001;
10     parameter S2 = 3'b011;
11     parameter S3 = 3'b010;
12     parameter S4 = 3'b110;
13     parameter S5 = 3'b111;
14     parameter S6 = 3'b101;
15     parameter S7 = 3'b100;
16
17     always_ff @(posedge clk, posedge reset) begin
18         if (reset)
19             state <= S0;
20         else
21             state <= nextstate;
```

```
22      end
23
24      always_comb begin
25          case (state)
26              S0 : if (up) nextstate = S1;
27                   else nextstate = S7;
28              S1 : if (up) nextstate = S2;
29                   else nextstate = S0;
30              S2 : if (up) nextstate = S3;
31                   else nextstate = S1;
32              S3 : if (up) nextstate = S4;
33                   else nextstate = S2;
34              S4 : if (up) nextstate = S5;
35                   else nextstate = S3;
36              S5 : if (up) nextstate = S6;
37                   else nextstate = S4;
38              S6 : if (up) nextstate = S7;
39                   else nextstate = S5;
40              S7 : if (up) nextstate = S0;
41                   else nextstate = S6;
42          endcase

43      end
44
45      assign y = state;
46  endmodule
```

Exercise 4.39

Write an HDL module for the FSM from Exercise 3.29.

Solution

```
 1  module ab_function
 2      (input a, b,
 3      input clk, reset,
 4      output z);
 5
 6      logic aprev;
 7
 8      always_ff @(posedge clk, posedge reset) begin
 9          aprev <= a;
10      end
11
12      assign z = b ? (aprev | a) : (aprev & a);
13  endmodule
```

Exercise 4.40

Write an HDL module for the FSM from Exercise 3.30.

Solution

```
 1  module fsmxy
 2      (input logic a,
 3      input logic clk, reset,
 4      output logic x, y);
 5
 6      typedef enum logic[1:0] {S0, S1, S2, S3} statetypeX;
```

```
 7     typedef enum logic[1:0] {T0, T1, T2} statetypeY;
 8
 9     statetypeX stateX, nextstateX;
10     statetypeY stateY, nextstateY;
11
12     always_ff @(posedge clk, posedge reset) begin
13         if (reset) begin
14             stateX <= S0;
15             stateY <= T0;
16         end
17         else begin
18             stateX <= nextstateX;
19             stateY <= nextstateY;
20         end
21     end
22
23     always_comb begin
24         case (stateX)
25             S0 : if (a) nextstateX = S1;
26                  else nextstateX = S0;
27             S1 : if (a) nextstateX = S2;
28                  else nextstateX = S1;
29             S2 : if (a) nextstateX = S3;
30                  else nextstateX = S2;
31             S3 : nextstateX = S3;
32         endcase
33         case (stateY)
34             T0 : if(a) nextstateY = T1;
35                  else nextstateY = T0;
36             T1 : if(a) nextstateY = T2;
37                  else nextstateY = T0;
38             T2 : if(a) nextstateY = T2;
39                  else nextstateY = T0;
40             default: nextstateY = T0;
41         endcase
42     end
43
44     assign x = (stateX == S3) ? 1'b1 : 1'b0;
45     assign y = (stateY == T2) ? 1'b1 : 1'b0;
46 endmodule
```

Exercise 4.41

Write an HDL module for the serial two's complementer from Question 3.2.

Solution

/*Pending*/

Exercise 4.42

Write an HDL module for the circuit in Exercise 3.31.

Solution

```
 1 module reverse_eng1
 2     (input logic x,
 3     input logic clk, reset,
 4     output logic q);
 5
```

```
 6     typedef enum logic[1:0] {S0, S1, S2, S3} statetype;
 7
 8     statetype state, nextstate;
 9
10     always_ff @(posedge clk, posedge reset) begin
11         if (reset) begin
12             state <= S0;
13         end
14         else begin
15             state <= nextstate;
16         end
17     end
18
19     always_comb begin
20         case (state)
21             S0 : if (x) nextstate = S3;
22                 else if (~x) nextstate = S1;
23                 else nextstate = S0;
24             S1 : if (x) nextstate = S2;
25                 else if (~x) nextstate = S0;
26                 else nextstate = S1;
27             S2 : nextstate = S1;
28             S3 : nextstate = S1;
29         endcase
30     end
31
32     assign q = (state == S1 || state == S2 || state == S3) ? 1'b1 : 1'b0;
33 endmodule
```

Exercise 4.43

Write an HDL module for the circuit in Exercise 3.32.

Solution

```
 1 module reverse_eng2
 2     (input logic a,
 3     input logic clk, reset,
 4     output logic q);
 5
 6     typedef enum logic[1:0] {S0, S1, S2} statetype;
 7
 8     statetype state, nextstate;
 9
10     always_ff @(posedge clk, posedge reset) begin
11         if (reset) begin
12             state <= S0;
13         end
14         else begin
15             state <= nextstate;
16         end
17     end
18
19     always_comb begin
20         case (state)
21             S0 : if (a) nextstate = S1;
22                 else if (~a) nextstate = S0;
23             S1 : if (a) nextstate = S2;
24                 else if (~a) nextstate = S0;
```

```
25              S2 : if (a) nextstate = S2;
26                  else if (~a) nextstate = S0;
27              default : nextstate = S0;
28          endcase
29      end
30
31      assign q = (state == S2) ? 1'b1 : 1'b0;
32  endmodule
```

Exercise 4.44

Write an HDL module for the circuit in Exercise 3.33.

Solution

```
 1  module four_xor_register
 2      (input logic a, b, c, d,
 3      input logic clk,
 4      output logic y);
 5
 6      logic reg_a, reg_b, reg_c, reg_d;
 7      logic n1, n2, n3;
 8
 9      always_ff @(posedge(clk)) begin
10          {reg_a, reg_b, reg_c, reg_d} <= {a, b, c, d};
11          y <= n3;
12      end
13
14      assign n1 = (reg_a ^ reg_b);
15      assign n2 = (n1 ^ reg_c);
16      assign n3 = (n2 ^ reg_d);
17  endmodule
```

Exercise 4.45

Write an HDL module for the circuit in Exercise 3.34. You may use the full adder from Section 4.2.5.

Solution

```
 1  module two_bit_adder
 2      (input logic[1:0] a, b,
 3      input logic clk, c,
 4      output logic[1:0] s);
 5
 6      logic cout1, cout0, reg_c;
 7      logic[1:0] reg_a, reg_b, s_bef;
 8
 9      fulladder bit_zero_adder(
10          .a(reg_a[0]),
11          .b(reg_b[0]),
12          .cin(reg_c),
13          .s(s_bef[0]),
14          .cout(cout0)
15      );
16
17      fulladder bit_one_adder(
18          .a(reg_a[1]),
19          .b(reg_b[1]),
20          .cin(cout0),
```

```
21          .s(s_bef[1]),
22          .cout(cout1)
23      );
24
25      always_ff @(posedge(clk)) begin
26          reg_a <= a;
27          reg_b <= b;
28          s <= s_bef;
29          reg_c <= c;
30      end
31  endmodule
```

Chapter 5

Exercise 5.1

What is the delay for the following types of 64-bit adders? Assume that each two-input gate delay is 150 ps and that a full adder delay is 450 ps.

A) a ripple-carry adder
B) a carry-lookahead adder with 4-bit blocks
C) a prefix adder

Solution

A) a ripple-carry adder
$t_d = 450\ ps * 64 = 28.8\ ns$
B) a carry-lookahead adder with 4-bit blocks
$t_d = 150\ ps + 6 * 150\ ps + (64/4 - 1) * 300\ ps + 4 * 450\ ps = 7350\ ps$
C) a prefix adder
$t_d = 150\ ps + \log_2(64) * 300\ ps + 150\ ps = 2100\ ps$

Exercise 5.2

Design two adders: a 64-bit ripple-carry adder and a 64-bit carrylookahead adder with 4-bit blocks. Use only two-input gates. Each two-input gate is 15 μm^2, has a 50 ps delay, and has 20 fF of total gate capacitance. You may assume that the static power is negligible.

A) Compare the area, delay, and power of the adders (operating at 100 MHz and 1.2 V).
B) Discuss the trade-offs between power, area, and delay.

Solution

A) Compare the area, delay, and power of the adders

Characteristics	64-bit ripple-carry	64-bit carrylookahead (4-bit blocks)
Number of Gates	Is the sum of the 64 bits by the number of gates needed to build a full adder multiplied by the number of gates needed to build a full adder. $N_G = 64 * 5 = 320\ gates$	Is the sum of 4 bits blocks that made 64 bits (Which each of them have 9 two input gates) and the number of full adders which are 4 (5 two input gates). $N_G = (64/4) * (9 + 4 * 5) = 464\ gates$
Area	$A_{total} = N_G * A_{gate}$ $A_{total} = 320\ gates * 15\ \mu m^2 = 4800\ \mu m^2$	$A_{total} = N_G * A_{gate}$ $A_{total} = 464\ gates * 15\ \mu m^2 = 6960\ \mu m^2$
Delay	$t_{delay} = N * t_{FA} = 64 * 50\ ps * 3 = 9600\ ps$	$t_{delay} = t_{pg} + t_{pgblock} + (N/k - 1) * t_{pgfinal} + k * t_{FA}$ $t_{delay} = 50\ ps + 6 * 50\ ps + (64/4 - 1) * 2 * 50\ ps + 4 * 3 * 50\ ps = 2450\ ps$
Power Consumption	We need to calculate the power consumption by every gate $P_{Dynamic} = \alpha * C * {V_{DD}}^2 * f$ $P_{Dynamic} = 1 * (20 * 10^{-15} F) * (1.2\ V)^2 * (100 * 10^6) = 2.88\ \mu W$	

	$P_{total} = N_G * P_{Dynamic} = 320 * 2.88\,\mu W$ $P_{total} = 921.6\,\mu W$	$P_{total} = N_G * P_{Dynamic} = 464 * 2.88\,\mu W$ $P_{total} = 1336.32\,\mu W$

B) Discuss the trade-offs between power, area, and delay.

I could say that even with the drawbacks is better the carry look ahead adder because the extra gates given to the adder is just 45% (Which means 45% more power consumption and area), but the performance is almost 4x speed performance.

Exercise 5.3

Explain why a designer might choose to use a ripple-carry adder instead of a carry-lookahead adder.

Solution

If the designer want to prioritize the size and consumption of energy of the chip instead the speed of the semiconductor.

Exercise 5.4

Design the 16-bit prefix adder of Figure 5.7 in an HDL. Simulate and test your module to prove that it functions correctly.

Solution

```
 1 module prefix_adder #(parameter N=4)
 2     (input logic [2**N-1:0] a,
 3     input logic [2**N-1:0] b,
 4     input logic cin,
 5     output logic [2**N-1:0] s,
 6     output logic cout);
 7
 8     genvar i, j;
 9
10     logic [2**N-1:0] p[N:0];
11     logic [2**N-1:0] g[N:0];
12
13     assign p[0][0] = 0;
14     assign g[0][0] = cin;
15
16     generate
17         for (j=0; j<(2**N-1); j++) begin
18             assign p[0][j+1] = a[j] | b[j];
19             assign g[0][j+1] = a[j] & b[j];
20         end
21
22         for(i=1; i<N+1; i++) begin
23             for (j=0; j<2**N; j++) begin
24                 if(j%2**i <= 2**(i+1) && j%2**i >= 2**(i-1)) begin
25                     assign p[i][j] = p[i-1][j] & p[i-1][j-1-(j%(2**i))+2**(i-
1)];
26                     assign g[i][j] = g[i-1][j] | p[i-1][j] & g[i-1][j-1-(j
%(2**i))+2**(i-1)];
27                 end
28                 else begin
29                     assign p[i][j] = p[i-1][j];
30                     assign g[i][j] = g[i-1][j];
31                 end
```

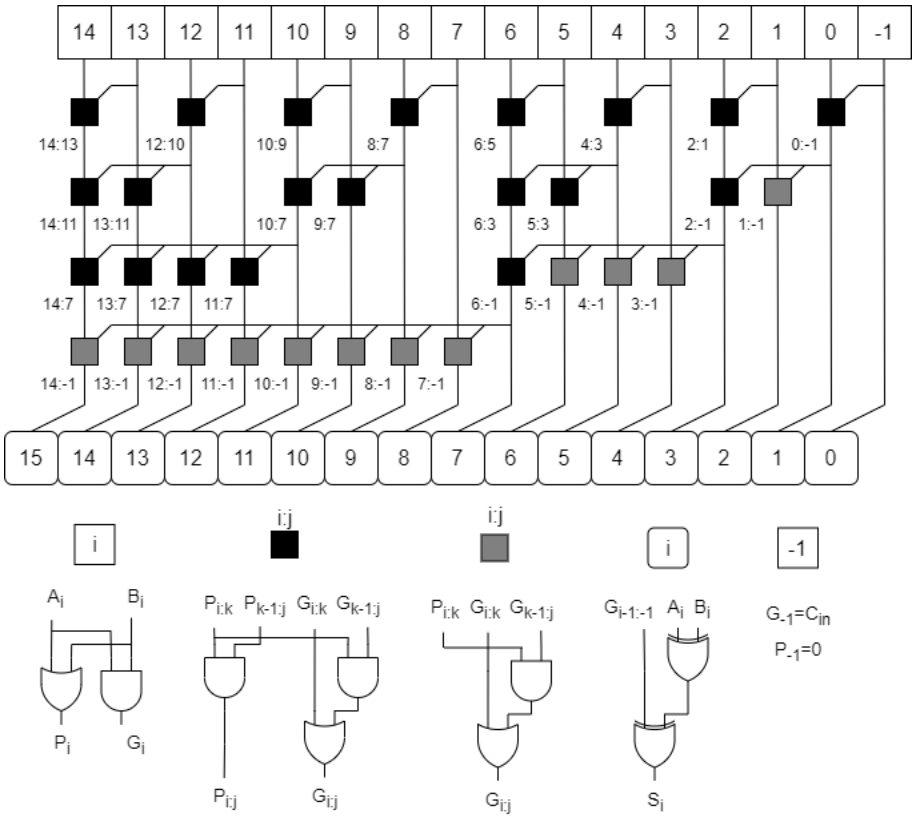
```
 32             end
 33         end
 34
 35         for (j=0; j<2**N; j++) begin
 36             assign s[j] = (a[j] ^ b[j]) ^ g[N][j];
 37         end
 38
 39         assign cout = ((a[2**N-1] ^ b[2**N-1]) & g[N][2**N-1]) | (a[2**N-1] &
b[2**N-1]);
 40     endgenerate
 41 endmodule
```

Exercise 5.5

The prefix network shown in Figure 5.7 uses black cells to compute all of the prefixes. Some of the block propagate signals are not actually necessary. Design a “gray cell” that receives G and P signals for bits i:k and k − 1:j but produces only $G_{i:j}$, not $P_{i:j}$. Redraw the prefix network, replacing black cells with gray cells wherever possible.

Solution

The gray cells must be the last cell before the sum because the propagate signal is needed to calculate the whole sum, that’s why cell 1:-1 is gray, is the last cell on the column i=1, but because cell 2:-1 is branched to calculate the next tier then it must be a black cell.

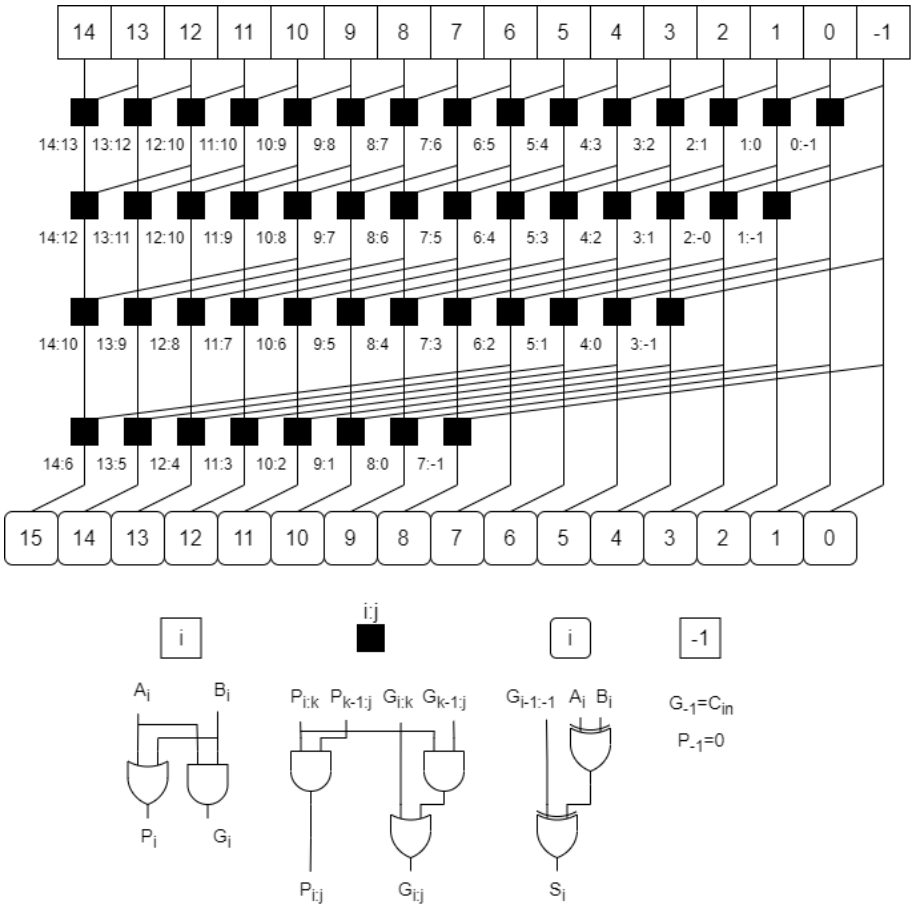


Exercise 5.6

The prefix network shown in Figure 5.7 is not the only way to calculate all of the prefixes in logarithmic time. The Kogge-Stone network is another common prefix network that performs the same

function using a different connection of black cells. Research Kogge-Stone adders and draw a schematic similar to Figure 5.7 showing the connection of black cells in a Kogge-Stone adder.

Solution



Exercise 5.7

Recall that an N-input priority encoder has $\log_2 N$ outputs that encode which of the N inputs gets priority (see Exercise 2.36).

A) Design an N-input priority encoder that has delay that increases logarithmically with N. Sketch your design and give the delay of the circuit in terms of the delay of its circuit elements.
B) Code your design in an HDL. Simulate and test your module to prove that it functions correctly.

Solution

/*Pending*/

Exercise 5.8

Design the following comparators for 32-bit unsigned numbers. Sketch the schematics.

A) Not equal
B) greater than or equal to
C) less than

Solution

/*Pending*/

Exercise 5.9

Consider the signed comparator of Figure 5.12.

A) Give an example of two 4-bit signed numbers A and B for which a 4-bit signed comparator correctly computes A < B.

B) Give an example of two 4-bit signed numbers A and B for which a 4-bit signed comparator incorrectly computes A < B.

C) In general, when does the N-bit signed comparator operate incorrectly?

Solution

A) $A<B=1110_2-0011_2=1011_2 \rightarrow 1$

B) $A<B \neq 1010_2-0101_2=0101_2 \rightarrow 0$

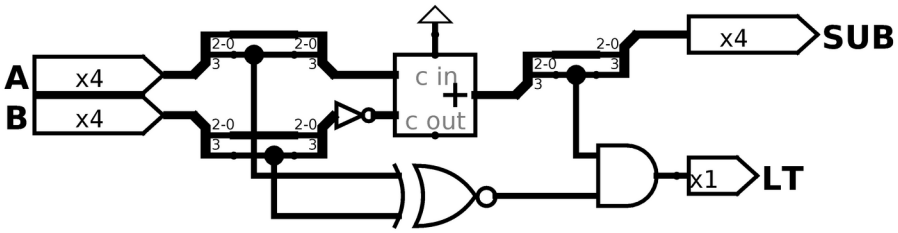
C) It occurs when happen a overflow, that mean when the result of subtraction is more than the bits that can hold the result.

Exercise 5.10

Modify the N-bit signed comparator of Figure 5.12 to correctly compute A < B for all N-bit signed inputs A and B.

Solution

To know if a signed number is less than other is necessary to know first if the operation overflows. To achieve that we need to compare the signs between the operands and the result. If this occurs we only need to see if the result is negative.



Exercise 5.11

Design the 32-bit ALU shown in Figure 5.15 using your favorite HDL. You can make the top-level module either behavioral or structural.

Solution

```
 1 module first_alu_32
 2     (input logic[31:0] a,
 3     input logic[31:0] b,
 4     input logic[1:0] control,
 5     output logic[31:0] result);
 6
 7     logic[31:0] sum;
 8
 9     always_comb begin
10         sum = a+((control[0]==1)?~b:b)+{31'b0,control[0]};
11         case (control)
12             2'b00 : result = sum;
13             2'b01 : result = sum;
14             2'b10 : result = a & b;
15             2'b11 : result = a | b;
16         endcase
```

```
    end
endmodule
```

Exercise 5.12

Design the 32-bit ALU shown in Figure 5.17 using your favorite HDL. You can make the top-level module either behavioral or structural.

Solution

```
module second_alu_32
    (input logic[31:0] a,
    input logic[31:0] b,
    input logic[1:0] control,
    output logic[31:0] result,
    output logic[3:0] flags);

    logic N,Z,C,V;
    logic[31:0] sum;
    logic[32:0] ext_a, ext_b, ext_c;
    logic cout;

    always_comb begin
        ext_a = {a[31],a};
        ext_b = {b[31],b};
        ext_c = {32'b0,control[0]};
        {cout, sum} = ext_a+((control[0]==1)?~ext_b:ext_b)+ext_c;
        case (control)
            2'b00 : result = sum;
            2'b01 : result = sum;
            2'b10 : result = a & b;
            2'b11 : result = a | b;
            default : result = 32'bx;
        endcase
        N = result[31];
        Z = &(~result);
        C = ~control[1] & cout;
        V = ~(a[31] ^ b[31] ^ control[0]) & (a[31] ^ sum[31]) & ~control[1];
        flags = {N,Z,C,V};
    end
endmodule
```

Exercise 5.13

Design the 32-bit ALU shown in Figure 5.18(a) using your favorite HDL. You can make the top-level module either behavioral or structural.

Solution

```
module third_alu_32
    (input logic[31:0] a,
    input logic[31:0] b,
    input logic[2:0] control,
    output logic[31:0] result);

    logic[31:0] sum;
    logic cout;

```

```
    always_comb begin
        sum = a+((control[0]==1)?~b:b)+{31'b0, control[0]};
        case (control)
            3'b000 : result = sum;
            3'b001 : result = sum;
            3'b010 : result = a & b;
            3'b011 : result = a | b;
            3'b101 : result = {31'b0, sum[31]};
            default : result = 32'bx;
        endcase
    end
endmodule
```

Exercise 5.14

Design the 32-bit ALU shown in Figure 5.18(b) using your favorite HDL. You can make the top-level module either behavioral or structural.

Solution

```
module fourth_alu_32
    (input logic[31:0] a,
    input logic[31:0] b,
    input logic[2:0] control,
    output logic[31:0] result);

    logic[31:0] sum;
    logic V, cout;

    always_comb begin
        sum = a+((control[0]==1)?~b:b)+{31'b0, control[0]};
        V = ~(a[31] ^ b[31] ^ control[0]) & (a[31] ^ sum[31]) & ~control[1];
        case (control)
            3'b000 : result = sum;
            3'b001 : result = sum;
            3'b010 : result = a & b;
            3'b011 : result = a | b;
            3'b101 : result = {31'b0, V ^ sum[31]};
            default : result = 32'bx;
        endcase
    end
endmodule
```

Exercise 5.15

Write a testbench to test the 32-bit ALU from Exercise 5.11. Then, use it to test the ALU. Include any test vector files necessary. Be sure to test enough corner cases to convince a reasonable skeptic that the ALU functions correctly.

Solution

```
module tb_first_alu_32();
    localparam vindx = 10;
    logic[31:0] a, b;
    logic[1:0] control;
    logic[31:0] result, result_exp;
    logic[97:0] vector_in[0 : vindx - 1];

```

```
 8      first_alu_32 dut(.a(a), .b(b), .control(control), .result(result));
 9
10      initial begin
11          $readmemh("input_first_alu_32.txt", vector_in);
12          $dumpfile("waveform_tb_first_alu_32.vcd");
13          $dumpvars(0, tb_first_alu_32);
14          for(int i=0; i<vindx; i++) begin
15              {control, a, b, result_exp} = vector_in[i]; #10;
16              assert
17                  (result === result_exp)
18              else
19                  $error("Failed at line %d : a=%h, b=%h, control=%b, result=%h",
(i+1), a, b, control, result);
20          end
21          $finish;
22      end
23  endmodule
```

Exercise 5.16

Repeat Exercise 5.15 for the ALU from Exercise 5.12.

Solution

```
 1  module tb_second_alu_32();
 2      localparam vindx = 10;
 3      logic[31:0] a, b;
 4      logic[1:0] control;
 5      logic[31:0] result, result_exp;
 6      logic[3:0] flags, flags_exp;
 7      logic[101:0] vector_in[0 : vindx - 1];
 8
 9      second_alu_32 dut(.a(a), .b(b), .control(control), .result(result),
.flags(flags));
10
11      initial begin
12          $readmemh("input_second_alu_32.txt", vector_in);
13          $dumpfile("waveform_tb_second_alu_32.vcd");
14          $dumpvars(0, tb_second_alu_32);
15          for(int i=0; i<vindx; i++) begin
16              {control, a, b, result_exp, flags_exp} = vector_in[i]; #10;
17              assert
18                  (result === result_exp && flags === flags_exp)
19              else
20                  $error("Failed at line %d : a=%h, b=%h, control=%b, result=%h,
flags=%b", (i+1), a, b, control, result, flags);
21          end
22          $finish;
23      end
24  endmodule
```

Exercise 5.17

Repeat Exercise 5.15 for the ALU from Exercise 5.13.

Solution

```
 1  module tb_third_alu_32();
 2      localparam vindx = 12;
```

```
 3     logic[31:0] a, b;
 4     logic[2:0] control;
 5     logic[31:0] result, result_exp;
 6     logic[98:0] vector_in[0 : vindx - 1];
 7
 8     third_alu_32 dut(.a(a), .b(b), .control(control), .result(result));
 9
10     initial begin
11         $readmemh("input_third_alu_32.txt", vector_in);
12         $dumpfile("waveform_tb_third_alu_32.vcd");
13         $dumpvars(0, tb_third_alu_32);
14         for(int i=0; i<vindx; i++) begin
15             {control, a, b, result_exp} = vector_in[i]; #10;
16             assert
17                 (result === result_exp)
18             else
19                 $error("Failed at line %d : a=%h, b=%h, control=%b, result=%h",
(i+1), a, b, control, result);
20         end
21         $finish;
22     end
23 endmodule
```

Exercise 5.18

Repeat Exercise 5.15 for the ALU from Exercise 5.14.

Solution

```
 1 module tb_fourth_alu_32();
 2     localparam vindx = 12;
 3     logic[31:0] a, b;
 4     logic[2:0] control;
 5     logic[31:0] result, result_exp;
 6     logic[98:0] vector_in[0 : vindx - 1];
 7
 8     fourth_alu_32 dut(.a(a), .b(b), .control(control), .result(result));
 9
10     initial begin
11         $readmemh("input_fourth_alu_32.txt", vector_in);
12         $dumpfile("waveform_tb_fourth_alu_32.vcd");
13         $dumpvars(0, tb_fourth_alu_32);
14         for(int i=0; i<vindx; i++) begin
15             {control, a, b, result_exp} = vector_in[i]; #10;
16             assert
17                 (result === result_exp)
18             else
19                 $error("Failed at line %d : a=%h, b=%h, control=%b, result=%h",
(i+1), a, b, control, result);

20         end
21         $finish;
22     end
23 endmodule
```

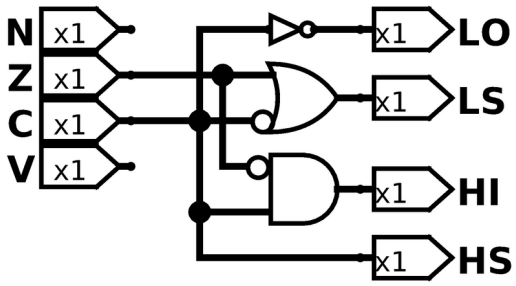
Exercise 5.19

Build an unsigned comparison unit that compares two unsigned numbers A and B. The unit's input is the Flags signal (N, Z, C, V) from the ALU of Figure 5.16, with the ALU performing subtraction: A − B. The unit's outputs are HS, LS, HI, and LO, which indicate that A is higher than or the same as (HS), lower than or the same as (LS), higher (HI), or lower (LO) than B.

A) Write minimal equations for HS, LS, HI, and LO in terms of N, Z, C, and V.
B) Sketch circuits for HS, LS, HI, and LO

Solution

A) Write minimal equations
$LO = \overline{C}$
$LS = Z + \overline{C}$
$HI = \overline{Z} \cdot C$
$HS = C$
B) Sketch circuits for HS, LS, HI, and LO



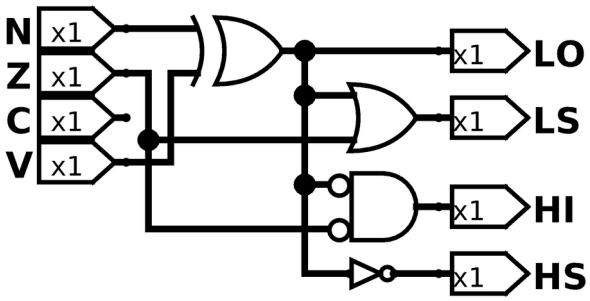
Exercise 5.20

Build a signed comparison unit that compares two signed numbers A and B. The unit's input is the Flags signal (N, Z, C, V) from the ALU of Figure 5.16, with the ALU performing subtraction: A − B. The unit's outputs are GE, LE, GT, and LT, which indicate that A is greater than or equal to (GE), less than or equal to (LE), greater than (GT), or less than (LT) B.

A) Write minimal equations for HS, LS, HI, and LO in terms of N, Z, C, and V.
B) Sketch circuits for HS, LS, HI, and LO

Solution

A) Write minimal equations
$LO = N \oplus V$
$LS = Z + \overline{(N \oplus V)}$
$HI = \overline{Z} \cdot \overline{(N \oplus V)}$
$HS = \overline{(N \oplus V)}$
B) Sketch circuits for HS, LS, HI, and LO

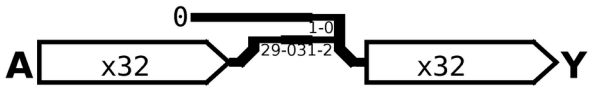


Exercise 5.21

Design a shifter that always shifts a 32-bit input left by 2 bits. The input and output are both 32 bits. Explain the design in words and sketch a schematic. Implement your design in your favorite HDL.

Solution

Is needed to wire the less significant 30 bits to the most significant 30 bits of the output and fill the blank bits with 0s.



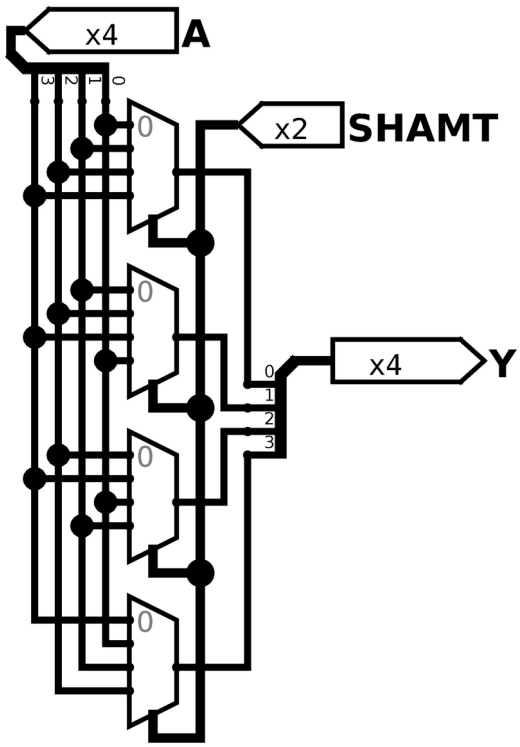

```
 1 module left_shift_2bit
 2     (input logic[31:0] a,
 3     output logic[31:0] y);
 4
 5     assign y = {a[29:0], 2'b00};
 6 endmodule
```

Exercise 5.22

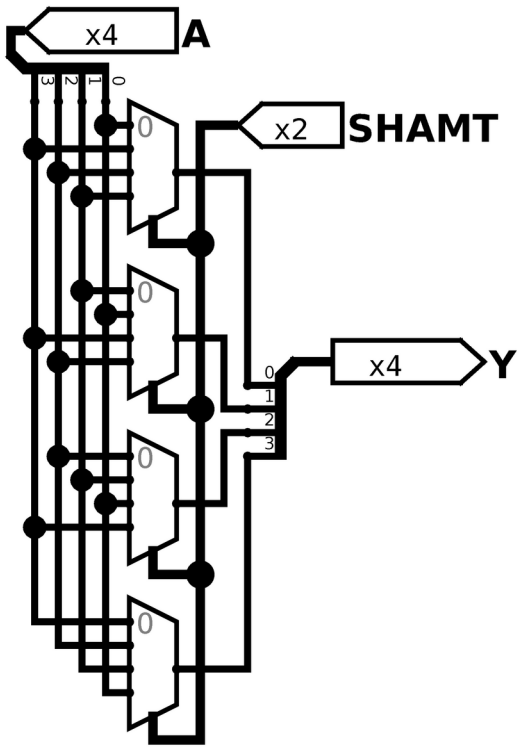
Design 4-bit left and right rotators. Sketch a schematic of each design. Implement your designs in your favorite HDL.

Solution

```
 1 module rr_shifter
 2     (input logic[3:0] a,
 3     input logic[1:0] shamt,
 4     output logic[3:0] y);
 5
 6     always_comb begin
 7         case (shamt)
 8             2'b00 : y = {a[3], a[2], a[1], a[0]};
 9             2'b01 : y = {a[0], a[3], a[2], a[1]};
10             2'b10 : y = {a[1], a[0], a[3], a[2]};
11             2'b11 : y = {a[2], a[1], a[0], a[3]};
12             default : y = 4'bx;
13         endcase
14     end
15 endmodule
```



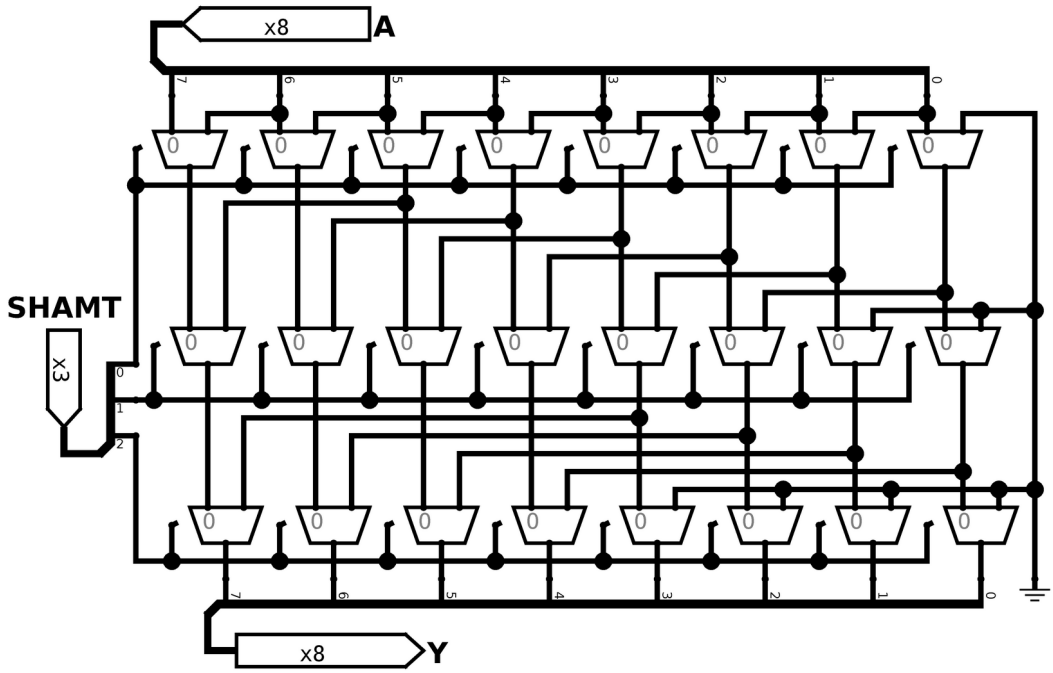

```
 1 module rl_shifter
 2     (input logic[3:0] a,
 3     input logic[1:0] shamt,
 4     output logic[3:0] y);
 5 
 6     always_comb begin
 7         case (shamt)
 8             2'b00 : y = {a[3], a[2], a[1], a[0]};
 9             2'b01 : y = {a[2], a[1], a[0], a[3]};
10             2'b10 : y = {a[1], a[0], a[3], a[2]};
11             2'b11 : y = {a[0], a[3], a[2], a[1]};
12             default : y = 4'bx;
13         endcase
14     end
15 endmodule
```



Exercise 5.23

Design an 8-bit left shifter using only 24 2:1 multiplexers. The shifter accepts an 8-bit input A and a 3-bit shift amount, $shamt_{2:0}$. It produces an 8-bit output Y. Sketch the schematic.

Solution



Exercise 5.24

Explain how to build any N-bit shifter or rotator using only $N\log_2 N$ 2:1 multiplexers.

Solution

You need to create a grid (N columns and $\log_2 N$ rows) of multiplexers. First we wire the bits of the input to the corresponding LSB input of every Multiplexer (for example we wire the bit 1 to the LSB input of the second multiplexer). Next we wire the MSB input of the multiplexer with a bit below (for our example the second multiplexer connect their MSB input to bit 0). As you can see the last multiplexer has not bit below so it wires to earth. We follow the same pattern but adding a distance of 2 powered to the number of the row you are actually working.

Exercise 5.25

The funnel shifter in Figure 5.66 can perform any N-bit shift or rotate operation. It shifts a 2N-bit input right by k bits. The output Y is the N least significant bits of the result. The most significant N bits of the input are called B and the least significant N bits are called C. By choosing appropriate values of B, C, and k, the funnel shifter can perform any type of shift or rotate. Explain what these values should be in terms of A, shamt, and N for

A) logical right shift of A by shamt
B) arithmetic right shift of A by shamt
C) left shift of A by shamt
D) right rotate of A by shamt
E) left rotate of A by shamt

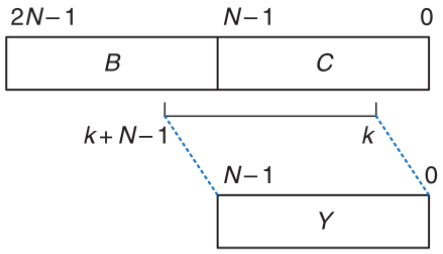


Figure 5.66 **Funnel shifter**

Solution

A) logical right shift of A by shamt
   B=0 C=A k=shamt
B) arithmetic right shift of A by shamt
   $B=A_{N-1}$ C=A k=shamt
C) left shift of A by shamt
   B=A C=0 k=-shamt
D) right rotate of A by shamt
   B=A C=A k=shamt
E) left rotate of A by shamt
   B=A C=A k=-shamt

Exercise 5.26

Find the critical path for the unsigned 4 × 4 multiplier from Figure 5.21 in terms of an AND gate delay (tAND) and a full adder delay (tFA). What is the delay of an N × N multiplier built in the same way?

Solution

$t_d = tAND + 8 * tFA$
$t_d = tAND + 2 * N * tFA$

Exercise 5.27

Find the critical path for the unsigned 4 × 4 divider from Figure 5.22 in terms of a 2:1 mux delay (tMUX), an adder delay (tFA), and an inverter delay (tINV). What is the delay of an N × N divider built in the same way?

Solution

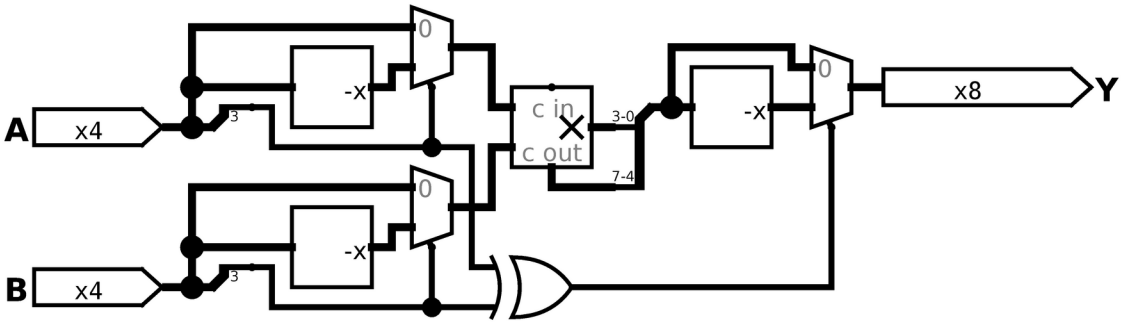
$t_d = 16 * tFA + 4 * tFA$
$t_d = N^2 * tFA + N * tFA$

Exercise 5.28

Design a multiplier that handles two's complement numbers.

Solution

The -X squares represent a two's complement block and the big X is a unsigned multiplier like the example before.

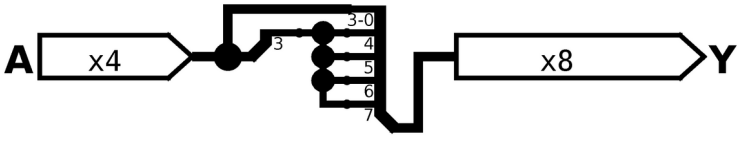


Exercise 5.29

A sign extension unit extends a two's complement number from M to N (N > M) bits by copying the most significant bit of the input into the upper bits of the output (see Section 1.4.6). It receives an M-bit input A and produces an N-bit output Y. Sketch a circuit for a sign extension unit with a 4-bit input and an 8-bit output. Write the HDL for your design.

Solution

```
1 module sign_extended #(parameter N=4, M=8)
2     (input logic[N-1:0] a,
3     output logic[M-1:0] y);
4
5     assign y = {{(M-N){a[N-1]}}, a[N-1:0]};
6 endmodule
```

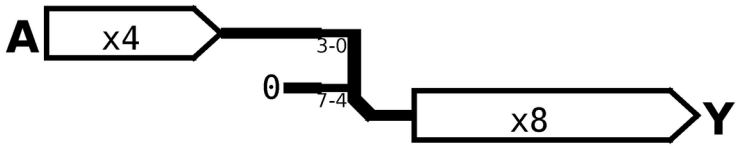


Exercise 5.30

A zero extension unit extends an unsigned number from M to N bits (N > M) by putting zeros in the upper bits of the output. Sketch a circuit for a zero extension unit with a 4-bit input and an 8-bit output. Write the HDL for your design.

Solution

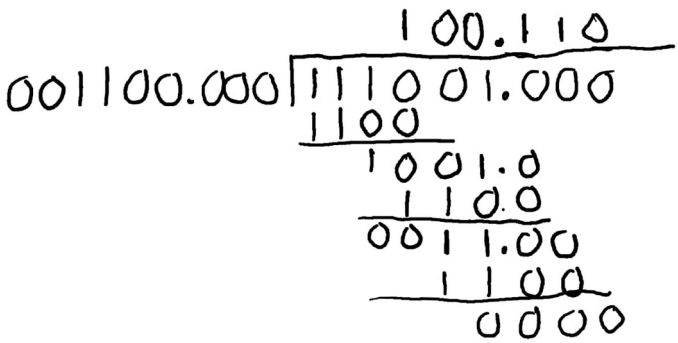
```
1 module zero_extended #(parameter N=4, M=8)
2     (input logic[N-1:0] a,
3     output logic[M-1:0] y);
4
5     assign y = {{(M-N){1'b0}}, a[N-1:0]};
6 endmodule
```



Exercise 5.31

Compute $111001.000_2/001100.000_2$ in binary using the standard division algorithm from elementary school. Show your work.

Solution



Exercise 5.32

What is the range of numbers that can be represented by the following number systems?

A) U12.12 format (24-bit unsigned fixed-point numbers with 12 integer bits and 12 fraction bits)
B) 24-bit sign/magnitude fixed-point numbers with 12 integer bits and 12 fraction bits
C) Q12.12 format (24-bit two's complement fixed-point numbers with 12 integer bits and 12 fraction bits)

Solution

A) You can represent 2^{24} distinct values from 0.0 to 4095.999755859375
B) You can represent 2^{24}-1 distinct values from -2047.999755859375 to 2047.999755859375
C) You can represent 2^{24}-1 distinct values from -2047.99951171875 to 2047.999755859375

Exercise 5.33

Express the following base 10 numbers in 16-bit fixed-point sign/magnitude format with eight integer bits and eight fraction bits. Express your answer in hexadecimal.

A) −13.5625
B) 42.3125
C) −17.15625

Solution

A) 0x8D90
$-13.5625 = -(2^3 + 2^2 + 2^0 + 2^{-1} + 2^{-4}) = 10001101.10010000_2$
B) 0x2A50
$42.3125 = 2^5 + 2^3 + 2^1 + 2^{-2} + 2^{-4} = 00101010.01010000_2$
C) 0x9128
$-17.15625 = -(2^4 + 2^0 + 2^{-3} + 2^{-5}) = 10010001.00101000_2$

Exercise 5.34

Express the following base 10 numbers in 12-bit fixed-point sign/magnitude format with six integer bits and six fraction bits. Express your answer in hexadecimal.

A) −30.5
B) 16.25
C) −8.078125

Solution

A) 0xFA0
$-30.5 = -(2^4 + 2^3 + 2^2 + 2^1 + 2^{-1}) = 111110.100000_2$
B) 0x410
$16.25 = 2^4 + 2^{-2} = 010000.010000_2$
C) 0xA05
$-8.078125 = -(2^3 + 2^{-4} + 2^{-6}) = 101000.000101_2$

Exercise 5.35

Express the base 10 numbers in Exercise 5.33 in Q8.8 format (16-bit fixed-point two's complement format with eight integer bits and eight fraction bits). Express your answer in hexadecimal.

Solution

A) 0xF270
$-13.5625 = -(2^3 + 2^2 + 2^0 + 2^{-1} + 2^{-4}) = -(00001101.1001000_2) = 11110010.01110000_2$
B) 0x2A50
$42.3125 = 2^5 + 2^3 + 2^1 + 2^{-2} + 2^{-4} = 00101010.01010000_2$
C) 0xEED8
$-17.15625 = -(2^4 + 2^0 + 2^{-3} + 2^{-5}) = -(00010001.00101000_2) = 11101110.11011000_2$

Exercise 5.36

Express the base 10 numbers in Exercise 5.34 in Q6.6 format (12-bit fixed-point two's complement format with six integer bits and six fraction bits). Express your answer in hexadecimal.

Solution

A) 0x860
$-30.5 = -(2^4 + 2^3 + 2^2 + 2^1 + 2^{-1}) = -(011110.100000_2) = 100001.100000$

B) 0x410
$16.25 = 2^4 + 2^{-2} = 010000.010000_2$

C) 0xDFB
$-8.078125 = -(2^3 + 2^{-4} + 2^{-6}) = -(001000.000101_2) = 110111.111011$

Exercise 5.37

Express the base 10 numbers in Exercise 5.33 in IEEE 754 single-precision floating-point format. Express your answer in hexadecimal.

Solution

A) 0xC1590000
$-13.5625 = -(2^3 + 2^2 + 2^0 + 2^{-1} + 2^{-4})$
$Sign = 1$
$Exponent = 127 + 3 = 130 = 10000010$
$Mantissa = 1.1011001$
$-13.5625 = 1 \;\; 10000010 \;\; 1011001 \ldots 0$

B) 0x42294000
$42.3125 = 2^5 + 2^3 + 2^1 + 2^{-2} + 2^{-4}$
$Sign = 0$
$Exponent = 127 + 5 = 132 = 10000100$
$Mantissa = 1.010100101$
$42.3125 = 0 \;\; 10000100 \;\; 010100101 \ldots 0$

C) 0xC1894000
$-17.15625 = -(2^4 + 2^0 + 2^{-3} + 2^{-5})$
$Sign = 1$
$Exponent = 127 + 4 = 131 = 10000011$
$Mantissa = 1.000100101$
$-17.15625 = 1 \;\; 10000011 \;\; 000100101 \ldots 0$

Exercise 5.38

Express the base 10 numbers in Exercise 5.34 in IEEE 754 single-precision floating-point format. Express your answer in hexadecimal.

Solution

A) 0xC1F40000
$-30.5 = -(2^4 + 2^3 + 2^2 + 2^1 + 2^{-1})$
$Sign = 1$
$Exponent = 127 + 4 = 131 = 10000011$
$Mantissa = 1.11101$
$-30.5 = 1 \;\; 10000011 \;\; 11101 \ldots 0$

B) 0x41808000

$16.25 = 2^4 + 2^{-2}$
$Sign = 0$
$Exponent = 127 + 4 = 131 = 10000011$
$Mantissa = 1.000001$
$16.25 = 0 \;\; 10000011 \;\; 000001\ldots0$

C) 0xC1014000
$-8.078125 = -\left(2^3 + 2^{-4} + 2^{-6}\right)$
$Sign = 1$
$Exponent = 127 + 3 = 130 = 10000010$
$Mantissa = 1.000000101$
$-8.078125 = 1 \;\; 10000010 \;\; 000000101\ldots0$

Exercise 5.39

Convert the following Q4.4 (two's complement binary fixed-point numbers) to base 10. The implied binary point is explicitly shown to aid in your interpretation.

A) 0101.1000
B) 1111.1111
C) 1000.0000

Solution

A) $0101.1000 = 2^2 + 2^0 + 2^{-1} = 5.5$
B) $1111.1111 = -2^3 + 2^2 + 2^1 + 2^0 + 2^{-1} + 2^{-2} + 2^{-3} + 2^{-4} = -0.0625$
C) $1000.0000 = -2^3 = -8$

Exercise 5.40

Repeat Exercise 5.39 for the following Q6.5 format (two's complement binary fixed-point) numbers.

A) 011101.10101
B) 100110.11010
C) 101000.00100

Solution

A) $011101.10101 = 2^4 + 2^3 + 2^2 + 2^0 + 2^{-1} + 2^{-3} + 2^{-5} = 29.65625$
B) $100110.11010 = -2^5 + 2^2 + 2^1 + 2^{-1} + 2^{-2} + 2^{-4} = -25.1875$
C) $101000.00100 = -2^5 + 2^3 + 2^{-3} = -23.875$

Exercise 5.41

When adding two floating-point numbers, the number with the smaller exponent is shifted. Why is this? Explain in words and give an example to justify your explanation.

Solution

This is because we want to preserve the most number of bits. So the number with the greater exponent get fixed and we shift the lower number by the difference of the exponent. The lower number just lose the number of bits shifted. So this is the best way to sum and preserve the most accurate value possible.

Exercise 5.42

Add the following IEEE 754 single-precision floating-point numbers.

A) C0123456 + 81C564B7
B) D0B10301 + D1B43203

C) 5EF10324 + 5E039020

Solution

A) 0xC0123456

$C0123456 = 1100\ \ 0000\ \ 0001\ \ 0010\ \ 0011\ \ 0100\ \ 0101\ \ 0110$
$Sign = 1$
$Exponent = 10000000 = 128 - 127 = 1$
$Mantissa = 1.00100100011010001010110 \gg 0$
$Mantissa = 1.00100100011010001010110$

$81C564B7 = 1000\ \ 0001\ \ 1100\ \ 0101\ \ 0110\ \ 0100\ \ 1011\ \ 0111$
$Sign = 1$
$Exponent = 00000011 = 3 - 127 = -124$
$Mantissa = 1.10001010110010010110111 \gg (1 - -124)$
$Mantissa = 0.00000000000000000000000$

$-1.00100100011010001010110$
$\underline{-0.00000000000000000000000}$
$-1.00100100011010001010110$

$Sign = 1$
$Exponent = 10000000 = 1$
$Mantissa = 1.00100100011010001010110$
$1100\ \ 0000\ \ 0001\ \ 0010\ \ 0011\ \ 0100\ \ 0101\ \ 0110 = C0123456$

B) 0xD1E072C3

$D0B10301 = 1101\ \ 0000\ \ 1011\ \ 0001\ \ 0000\ \ 0011\ \ 0000\ \ 0001$
$Sign = 1$
$Exponent = 10100001 = 161 - 127 = 34$
$Mantissa = 1.01100010000001100000001 \gg (36 - 34)$
$Mantissa = 0.01011000100000011000000$

$D1B43203 = 1101\ \ 0001\ \ 1011\ \ 0100\ \ 0011\ \ 0010\ \ 0000\ \ 0011$
$Sign = 1$
$Exponent = 10100011 = 163 - 127 = 36$
$Mantissa = 1.01101000011001000000011 \gg 0$
$Mantissa = 1.01101000011001000000011$

$-0.01011000100000011000000$
$\underline{-1.01101000011001000000011}$
$-1.11000000111001011000011$

$Sign = 1$
$Exponent = 10100011 = 36$
$Mantissa = 1.11000000111001011000011$
$1101\ \ 0001\ \ 1110\ \ 0000\ \ 0111\ \ 0010\ \ 1100\ \ 0011 = D1E072C3$

C) 0x5F19659A

$5EF10324 = 0101\ \ 1110\ \ 1111\ \ 0001\ \ 0000\ \ 0011\ \ 0010\ \ 0100$
$Sign = 0$
$Exponent = 10111101 = 189 - 127 = 62$
$Mantissa = 1.11100010000001100100100 \gg 0$
$Mantissa = 1.11100010000001100100100$

$5E039020 = 0101\ \ 1110\ \ 0000\ \ 0011\ \ 1001\ \ 0000\ \ 0010\ \ 0000$
$Sign = 0$
$Exponent = 10111100 = 188 - 127 = 61$
$Mantissa = 1.00000111001000000100000 \gg (62 - 61)$
$Mantissa = 0.10000011100100000010000$

$$\begin{array}{r} +1.11100010000001100100100 \\ \underline{+0.10000011100100000010000} \\ +10.01100101100101100110100 \end{array}$$

$Sign = 0$
$Exponent = 10111110 = 62 + 1$
$Mantissa = 1.00110010110010110011010$
$0101\ \ 1111\ \ 0001\ \ 1001\ \ 0110\ \ 0101\ \ 1001\ \ 1010 = 5F19659A$

Exercise 5.43

Add the following IEEE 754 single-precision floating-point numbers.

A) C0D20004 + 72407020
B) C0D20004 + 40DC0004
C) (5FBE4000 + 3FF80000) + DFDE4000 (Why is the result counterintuitive? Explain.)

Solution

A) 0x72407020

$C0D20004 = 1100\ \ 0000\ \ 1101\ \ 0010\ \ 0000\ \ 0000\ \ 0000\ \ 0100$
$Sign = 1$
$Exponent = 10000001 = 129 - 127 = 2$
$Mantissa = 1.10100100000000000000100 \gg (101 - 2)$
$Mantissa = 0.00000000000000000000000$

$72407020 = 0111\ \ 0010\ \ 0100\ \ 0000\ \ 0111\ \ 0000\ \ 0010\ \ 0000$
$Sign = 0$
$Exponent = 11100100 = 228 - 127 = 101$
$Mantissa = 1.10000000111000000100000 \gg 0$
$Mantissa = 1.10000000111000000100000$

$+\ 0.00000000000000000000000$
$+\ \underline{1.10000000111000000100000}$
$+\ 1.10000000111000000100000$

$Sign = 0$
$Exponent = 11100100 = 101$
$Mantissa = 1.10000000111000000100000$
$0111\ \ 0010\ \ 0100\ \ 0000\ \ 0111\ \ 0000\ \ 0010\ \ 0000 = 72407020$

B) 0x3EA00000

$C0D20004 = 1100\ \ 0000\ \ 1101\ \ 0010\ \ 0000\ \ 0000\ \ 0000\ \ 0100$
$Sign = 1$
$Exponent = 10000001 = 129 - 127 = 2$
$Mantissa = 1.10100100000000000000100 \gg (2 - 2)$
$Mantissa = 1.10100100000000000000100$

$40DC0004 = 0100\ \ 0000\ \ 1101\ \ 1100\ \ 0000\ \ 0000\ \ 0000\ \ 0100$
$Sign = 0$
$Exponent = 10000001 = 129 - 127 = 2$
$Mantissa = 1.10111000000000000000100 \gg (2 - 2)$
$Mantissa = 1.10111000000000000000100$

$-\ 1.10100100000000000000100$
$+\ \underline{1.10111000000000000000100}$
$+\ 0.00010100000000000000000$

$Sign = 0$
$Exponent = 01111101 = -2$
$Mantissa = 1.01000000000000000000000$
$0011\ \ 1110\ \ 1010\ \ 0000\ \ 0000\ \ 0000\ \ 0000\ \ 0000 = 3EA00000$

C) /*Pending*/

Exercise 5.44

Expand the steps in Section 5.3.2 for performing floating-point addition to work for negative as well as positive floating-point numbers.

Solution

1) Decode the sign, exponent and mantissa from the binary
2) Reform the mantissa adding the implicit 1
3) Compare exponents of both numbers and keep the larger exponent
4) Shift the smaller mantissa for the difference between exponents
5) Compare signs, if they have the same sign then 7) else 6)
6) Add both numbers
7) Subtract from the larger number the smaller one
8) Keep the sign of the number with larger mantissa
9) Adjust exponent by the number of exponent needed to normalize the mantissa
10) Encode the result with sign, exponent and mantissa

Exercise 5.45

Consider IEEE 754 single-precision floating-point numbers.

A) How many numbers can be represented by the IEEE 754 single-precision floating-point format? You need not count ±∞ or NaN.
B) How many additional numbers could be represented if ±∞ and NaN were not represented?
C) Explain why ±∞ and NaN are given special representations.

Solution

A) $2^{32}-2*2^{23}-2=4278190078$
B) $2*2^{23}+2=16777218$
C) Because there is operations with no real representation, NaN covers math operation with no solution on the real numbers and the infinite covers an overflow or underflow.

Exercise 5.46

Consider the following decimal numbers: 245 and 0.0625.

A) Write the two numbers using single-precision floating-point notation. Give your answers in hexadecimal.
B) Perform a magnitude comparison of the two 32-bit numbers from part (a). In other words, interpret the two 32-bit numbers as two's complement numbers and compare them. Does the integer comparison give the correct result?
C) You decide to come up with a new single-precision floating-point notation. Everything is the same as the IEEE 754 single-precision floating-point standard, except that you represent the exponent using two's complement instead of a bias. Write the two numbers using your new standard. Give your answers in hexadecimal.
D) Does integer comparison work with your new floating-point notation from part (c)?
E) Why is it convenient for integer comparison to work with floating-point numbers?

Solution

/*Pending*/

Exercise 5.47

Design a single-precision floating-point adder using your favorite HDL. Before coding the design in an HDL, sketch a schematic of your design. Simulate and test your adder to prove to a skeptic that it functions correctly. You may consider positive numbers only and use round toward zero (truncate). You may also ignore the special cases given in Table 5.4.

Solution

/*Pending*/

Exercise 5.48

In this problem, you will explore the design of a 32-bit floating- point multiplier. The multiplier has two 32-bit floating-point inputs and produces a 32-bit floating-point output. You may consider positive numbers only and use round toward zero (truncate). You may also ignore the special cases given in Table 5.4.

A) Write the steps necessary to perform 32-bit floating-point multiplication.
B) Sketch the schematic of a 32-bit floating-point multiplier.
C) Design a 32-bit floating-point multiplier in an HDL. Simulate and test your multiplier to prove to a skeptic that it functions correctly.

Solution

/*Pending*/

Exercise 5.49

In this problem, you will explore the design of a 32-bit prefix adder.

A) Sketch a schematic of your design.
B) Design the 32-bit prefix adder in an HDL. Simulate and test your adder to prove that it functions correctly.
C) What is the delay of your 32-bit prefix adder from part (a)? Assume that each two-input gate delay is 100 ps.
D) Design a pipelined version of the 32-bit prefix adder. Sketch the schematic of your design. How fast can your pipelined prefix adder run? You may assume a sequencing overhead (tpcq + tsetup) of 80 ps. Make the design run as fast as possible.
E) Design the pipelined 32-bit prefix adder in an HDL.

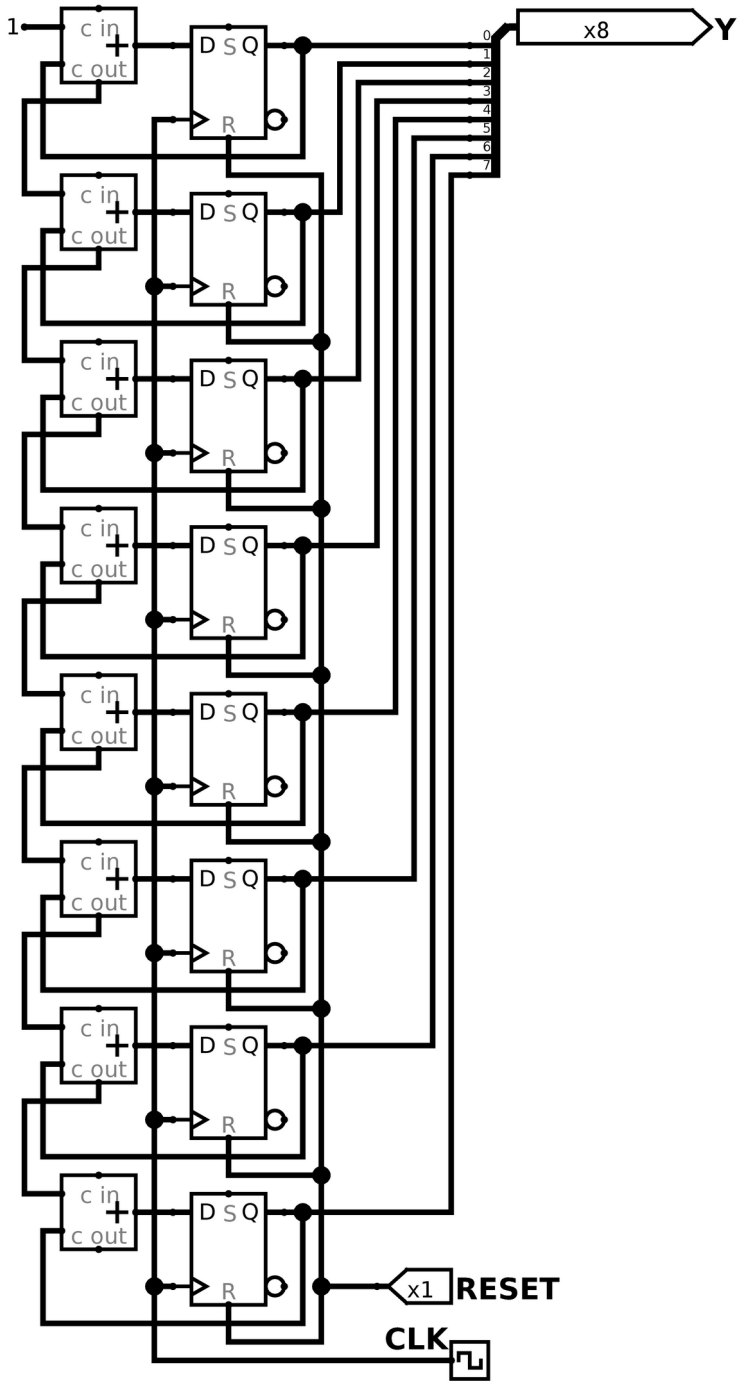
Solution

/*Pending*/

Exercise 5.50

An incrementer adds 1 to an N-bit number. Build an 8-bit incrementer using half adders.

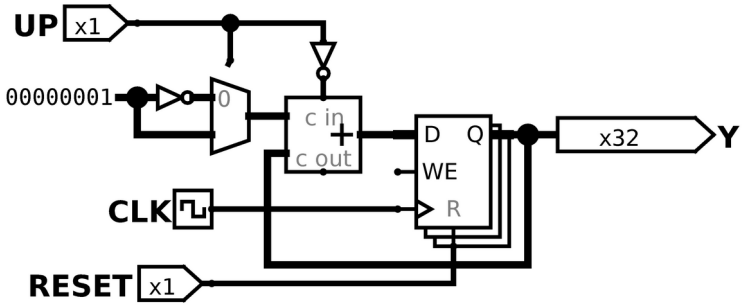
Solution



Exercise 5.51

Build a 32-bit synchronous Up/Down counter. The inputs are Reset and Up. When Reset is 1, the outputs are all 0. Otherwise, when Up = 1, the circuit counts up, and when Up = 0, the circuit counts down.

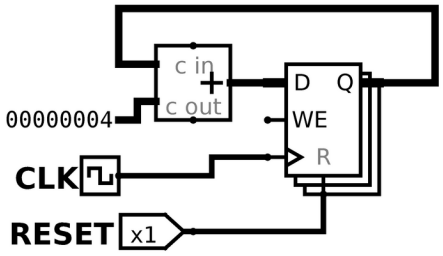
Solution



Exercise 5.52

Design a 32-bit counter that adds 4 at each clock edge. The counter has reset and clock inputs. Upon reset, the counter output is all 0.

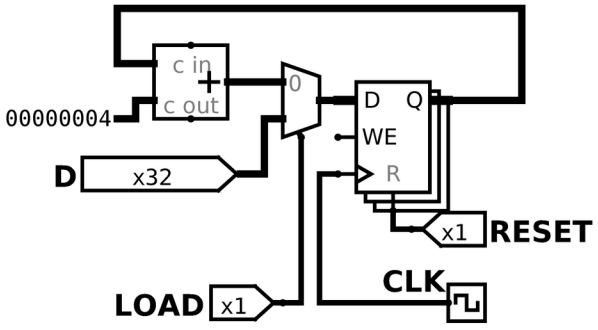
Solution



Exercise 5.53

Modify the counter from Exercise 5.52 such that the counter will either increment by 4 or load a new 32-bit value, D, on each clock edge, depending on a control signal Load. When Load = 1, the counter loads the new value D. Otherwise, it increments by 4.

Solution



Exercise 5.54

An N-bit Johnson counter consists of an N-bit shift register with a reset signal. The output of the shift register (S_{out}) is inverted and fed back to the input (S_{in}). When the counter is reset, all of the bits are cleared to 0.

A) Show the sequence of outputs, $Q_{3:0}$, produced by a 4-bit Johnson counter starting immediately after the counter is reset.
B) How many cycles elapse until an N-bit Johnson counter repeats its sequence? Explain.

C) Design a decimal counter using a 5-bit Johnson counter, ten AND gates, and inverters. The decimal counter has a clock, a reset, and ten one-hot outputs $Y_{9:0}$. When the counter is reset, Y_0 is asserted. On each subsequent cycle, the next output should be asserted. After ten cycles, the counter should repeat. Sketch a schematic of the decimal counter.

D) What advantages might a Johnson counter have over a conventional counter?

Solution

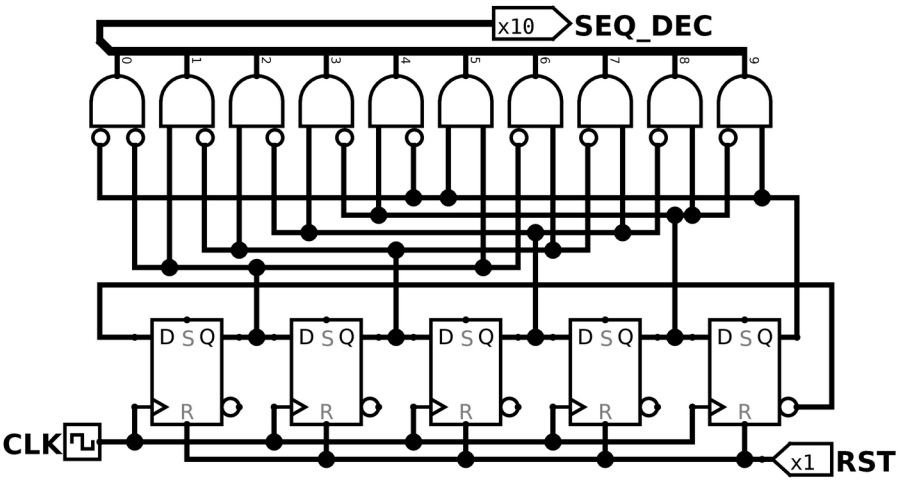
A) Sequence

RESET	Q3	Q2	Q1	Q0
1	0	0	0	0
0	0	0	0	1
0	0	0	1	1
0	0	1	1	1
0	1	1	1	1
0	1	1	1	0
0	1	1	0	0
0	1	0	0	0
0	0	0	0	0

B) It will repeat after 8 clock cycles

C) Decimal Johnson Counter

RESET	Q4	Q3	Q2	Q1	Q0	Y	Equation
1	0	0	0	0	0	0000000001	$Y_0=\overline{Q_4}\cdot\overline{Q_0}$
0	0	0	0	0	1	0000000010	$Y_1=\overline{Q_1}\cdot Q_0$
0	0	0	0	1	1	0000000100	$Y_2=\overline{Q_2}\cdot Q_1$
0	0	0	1	1	1	0000001000	$Y_3=\overline{Q_3}\cdot Q_2$
0	0	1	1	1	1	0000010000	$Y_4=\overline{Q_4}\cdot Q_3$
0	1	1	1	1	1	0000100000	$Y_5=Q_4\cdot Q_0$
0	1	1	1	1	0	0001000000	$Y_6=Q_1\cdot\overline{Q_0}$
0	1	1	1	0	0	0010000000	$Y_7=Q_2\cdot\overline{Q_1}$
0	1	1	0	0	0	0100000000	$Y_8=Q_3\cdot\overline{Q_2}$
0	1	0	0	0	0	1000000000	$Y_9=Q_4\cdot\overline{Q_3}$



D) The principal benefit are the minimal hardware needed to build it, also is easier to decode but the principal disadvantages are the delay of the signal to propagate

Exercise 5.55

Write the HDL for a 4-bit scannable flip-flop like the one shown in Figure 5.39. Simulate and test your HDL module to prove that it functions correctly.

Solution

/*Pending*/

Exercise 5.56

The English language has a good deal of redundancy that allows us to reconstruct garbled transmissions. Binary data can also be transmitted in redundant form to allow error correction. For example, the number 0 could be coded as 00000 and the number 1 could be coded as 11111. The value could then be sent over a noisy channel that might flip up to two of the bits. The receiver could reconstruct the original data because a 0 will have at least three of the five received bits as 0's; similarly, a 1 will have at least three 1's.

A) Propose an encoding to send 00, 01, 10, or 11 encoded using five bits of information such that all errors that corrupt one bit of the encoded data can be corrected. Hint: the encodings 00000 and 11111 for 00 and 11, respectively, will not work.
B) Design a circuit that receives your five-bit encoded data and decodes it to 00, 01, 10, or 11, even if one bit of the transmitted data has been changed.
C) Suppose you wanted to change to an alternative 5-bit encoding. How might you implement your design to make it easy to change the encoding without having to use different hardware?

Solution

/*Pending*/

Exercise 5.57

Flash EEPROM, simply called Flash memory, is a fairly recent invention that has revolutionized consumer electronics. Research and explain how Flash memory works. Use a diagram illustrating the floating gate. Describe how a bit in the memory is programmed. Properly cite your sources.

Solution

/*Pending*/

Exercise 5.58

The extraterrestrial life project team has just discovered aliens living on the bottom of Mono Lake. They need to construct a circuit to classify the aliens by potential planet of origin based on measured features available from the NASA probe: greenness, brownness, sliminess, and ugliness. Careful consultation with xenobiologists leads to the following conclusions:

- If the alien is green and slimy or ugly, brown, and slimy, it might be from Mars.
- If the critter is ugly, brown, and slimy, or green and neither ugly nor slimy, it might be from Venus.
- If the beastie is brown and neither ugly nor slimy or is green and slimy, it might be from Jupiter.

Note that this is an inexact science; for example, a life form which is mottled green and brown and is slimy but not ugly might be from either Mars or Jupiter.

A) Program a 4 × 4 × 3 PLA to identify the alien. You may use dot notation.
B) Program a 16 × 3 ROM to identify the alien. You may use dot notation.
C) Implement your design in an HDL.

Solution

/*Pending*/

Exercise 5.59

Implement the following functions using a single 16 × 3 ROM. Use dot notation to indicate the ROM contents.

A) $X = A \cdot B + B \cdot \overline{C} \cdot D + \overline{A} \cdot \overline{B}$
B) $Y = A \cdot B + B \cdot D$
C) $Z = A + B + C + D$

Solution

/*Pending*/

Exercise 5.60

Implement the functions from Exercise 5.59 using a 4 × 8 × 3 PLA. You may use dot notation.

Solution

/*Pending*/

Exercise 5.61

Specify the size of a ROM that you could use to program each of the following combinational circuits. Is using a ROM to implement these functions a good design choice? Explain why or why not.

A) a 16-bit adder/subtractor with C_{in} and C_{out}
B) an 8 × 8 multiplier
C) a 16-bit priority encoder (see Exercise 2.36)

Solution

/*Pending*/

Exercise 5.62

Consider the ROM circuits in Figure 5.67. For each row, can the circuit in column I be replaced by an equivalent circuit in column II by proper programming of the latter’s ROM?

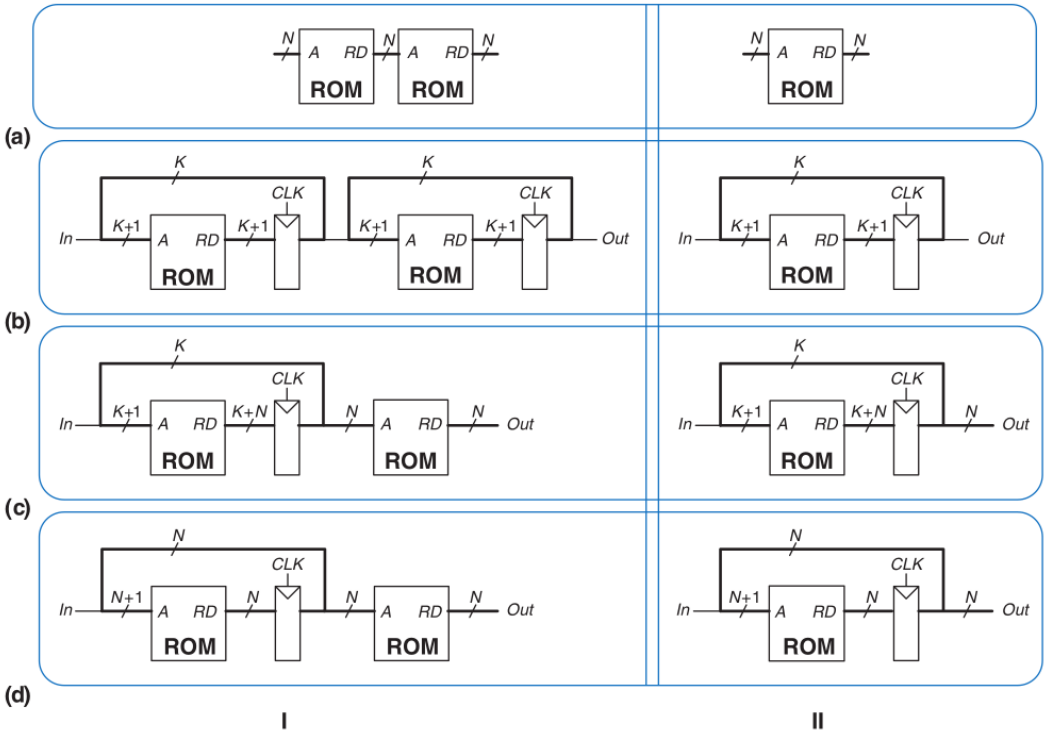


Figure 5.67 ROM circuits

Solution

/*Pending*/

Exercise 5.63

How many Cyclone IV FPGA LEs are required to perform each of the following functions? Show how to configure one or more LEs to perform the function. You should be able to do this by inspection, without performing logic synthesis.

A) the combinational function from Exercise 2.13(c)
B) the combinational function from Exercise 2.17(c)
C) the two-output function from Exercise 2.24
D) the function from Exercise 2.35
E) a four-input priority encoder (see Exercise 2.36)

Solution

/*Pending*/

Exercise 5.64

Repeat Exercise 5.63 for the following functions:

A) an eight-input priority encoder (see Exercise 2.36)
B) a 3:8 decoder
C) a 4-bit carry propagate adder (with no carry in or out)
D) the FSM from Exercise 3.22
E) the Gray code counter from Exercise 3.27

Solution

/*Pending*/

Exercise 5.65

Consider the Cyclone IV LE shown in Figure 5.60. According to the datasheet, it has the timing specifications given in Table 5.7.

A) What is the minimum number of Cyclone IV LEs required to implement the FSM of Figure 3.26?
B) Without clock skew, what is the fastest clock frequency at which this FSM will run reliably?
C) With 3 ns of clock skew, what is the fastest frequency at which the FSM will run reliably?

Table 5.7 Cyclone IV timing

Name	Value (ps)
t_{pcq}, t_{ccq}	199
t_{setup}	76
t_{hold}	0
t_{pd} (per LE)	381
t_{wire} (between LEs)	246
t_{skew}	0

Solution

/*Pending*/

Exercise 5.66

Repeat Exercise 5.65 for the FSM of Figure 3.31(b).

Solution

/*Pending*/

Exercise 5.67

You would like to use an FPGA to implement an M&M sorter with a color sensor and motors to put red candy in one jar and green candy in another. The design is to be implemented as an FSM using a Cyclone IV FPGA. According to the data sheet, the FPGA has timing characteristics shown in Table 5.7. You would like your FSM to run at 100 MHz. What is the maximum number of LEs on the critical path? What is the fastest speed at which the FSM will run?

Solution

/*Pending*/