Step 1 of 6

1.03DP

The following is the expansion of the acronyms:

ASIC – Application Specific Integrated Circuit

ASIC is an integrated circuit designed for specific application, and not for general purpose.

CAD – Computer Aided Design

CAD is used to create a technical drawing with the use of computer software. It is used to assist in creation, modification, analysis or optimization of a design.

CD – Compact Disc

CD is an optical disc, used to store digital data. It is flat, circular disc which encodes the binary data.

CO – Central Office (or) Company

CO is the building, and it has all the equipment used for telephone system to work.

Example: Making connections and relaying the speech information.

CPLD – Complex Programmable Logic Device

CPLDs are integrated circuits or chips that application designers configure to implement digital hardware. It contains macrocells with a sum-of-product combinatorial logic function, and an optional flip-flop.

Step 2 of 6

DIP – Dual Inline Package

DIP is a chip encased in a hard plastic casing with pins along each side of the plastic casing.

Example: DIP found on a computer motherboard that has been soldered into place.

DVD – Digital Versatile Disc

DVD a small plastic disc used for the storage of digital data. A DVD can have more than 100 times the storage capacity of a CD. When compared to CD technology, DVD allows for better graphics and greater resolution.

Step 3 of 6

FPGA – Field Programmable Gate Array

FPGA is a type of logic chip that can be programmed. An FPGA is similar to a PLD, but PLDs are generally limited to hundreds of gates, FPGAs support thousands of gates. They are especially popular for prototyping integrated circuit designs. Once the design is set, hardwired chips are produced for faster performance.

HDL – Hardware Description Language

HDL is a formal description and design of electronic circuits, and digital logic. It can describe the circuit's operation, its design and organization, and tests the operation by means of simulation.

IC – Integrated Circuit

IC is also known as chip, it contains an electronic circuit fabricated in a semiconductor material normally silicon. It is very compact, since it normally has billions of transistor and other electronic components.

Step 4 of 6

IP – Internet Protocol

IP is a numerical label assigned to each device (e.g., computer, printer) participating in a computer network that uses the IP for communication. An IP address serves two principal functions that are host or network interface identification and location addressing.

LSI – Large Scale Integration

LSI refers to the placement of thousands of electronic components on a single integrated circuit, approximately from 3,000 to 10,000 electronic components per chip.

MCM – Multi-Chip Module

MCM is a specialized electronic package where multiple integrated circuits (ICs), semiconductor dies or other discrete components are packaged and used as a single component (as a larger IC).

MSI – Medium Scale Integration

MSI refers to the placement of hundreds of electronic components on a single integrated circuit, approximately from 100 to 3,000 electronic components per chip.

NRE – Non-Recurring Engineering

Non-recurring engineering (NRE) refers to the one-time cost to research, develop, design and test a new product.

Step 5 of 6

OK – All Correct

OK is also spelled as okay. It is a colloquial word denoting approval, acceptance, agreement, assent, or acknowledgment.

PBX – Private Branch Exchange

A PBX is a telephone system within an enterprise that switches calls between enterprise users on local lines, while allowing all users to share certain number of external phone lines. The main purpose of PBX is to save the cost of requiring a line for each user to the telephone company's central office.

PCB – Printed Circuit Board

A PCB is also called printed wiring board or etching wiring board. It electrically connects electronic components using conductive pathways from copper sheets laminated on to a non-conductive substrate.

PLD – Programmable Logic Device

PLD is an integrated circuit that you program using a hardware description language such as VHDL or Verilog. Other languages that you may have heard of are CUPL or ADA.

PWB – Printed Wiring Board

A PWB is also called printed circuit board or etching wiring board. It electrically connects electronic components using conductive pathways from copper sheets laminated on to a non-conductive substrate.

SMT – Surface Mount Technology

SMT is a design standard for constructing electronic circuits where the components are mounted directly onto the surface of the printed circuit board. The components have small metal tabs that are soldered directly to the surface of the printed circuit board on tin-lead, silver, or gold plated copper pads, called solder pads.

Step 6 of 6

SSI – Small Scale Integration

SSI refers to the placement of tens of electronic components on a single integrated circuit, approximately from 1 to 100 electronic components per chip.

VHDL – VHSIC Hardware Description Language (or) Very High Speed Integrated Circuit Hardware Description Language.

VHDL is one of the Hardware Description Languages (HDLs) and it is used to design digital & mixed signal systems such as field programmable gate arrays & integrated circuit. In simple terms VHDL gives the text models that describe logic circuit.

VLSI – Very Large Scale Integration

VLSI refers to the placement of more than thousands of electronic components on a single integrated circuit, approximately from 100,000 to 1,000,000 electronic components per chip.

Step 1 of 1 **1.05DP**

The following is the expansion and definition of the acronyms:

ABEL – Advanced Boolean Expression Language

ABEL allows logic designs to be implemented in programmable logic devices. ABEL can be used to program any type of SPLD and is therefore a device- independent language. ABEL provides three different text-based formats for describing and entering a logic design from the computer keyboard: equations, truth tables, and state diagrams.

CMOS – Complementary Metal Oxide Semiconductor

CMOS is a low-power, low-heat semiconductor technology used in contemporary microchips, especially useful for battery-powered devices.

JPEG – Joint Photographic Experts Group

JPEG is a standardized image compression mechanism. JPEG compresses either full-color or grayscale images, or works best with photographs and artwork. Making image files smaller is important for storing and transmitting files.

MPEG – Moving Picture Experts Group

A set of standards established for the compression of digital video and audio data.

OK – All Correct

OK is also spelled as okay. It is a colloquial word denoting approval, acceptance, agreement, assent, or acknowledgment.

VHDL – VHSIC Hardware Description Language (or) Very High Speed Integrated Circuit Hardware Description Language.

VHDL is one of the Hardware Description Language (HDL) and it is used to design digital & mixed signal systems such as field programmable gate arrays & integrated circuit. In simple terms VHDL gives the text models that describe logic circuit.

Step 1 of 2

1.06DP

A digital circuit of two inputs AND gate and three inverters whereas an inverter is connected to each inputs of AND gate and to its output is shown in Figure 1.

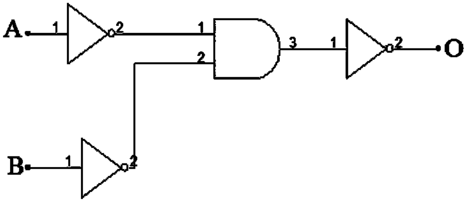


Figure 1

Step 2 of 2

The following table gives the four possible combinations of primary inputs, and the value of their corresponding primary output:

Inputs	Output	
A	B	O
0	0	0
0	1	1
1	0	1
1	1	1

According to the input and output behavior of the circuit, the circuit can be modified as a simple two input OR gate.

Step 1 of 1

1.07DP

Pin diagram shows the assignment of device signals to package pins, whereas, schematic diagram is the simplified, conventional, graphical representation of a digital circuit.

Pin diagram should be used for mechanical references, whenever the designer wants to know the pin number of particular IC.

Step 1 of 1

The following is the relationship between die and dice:

Die is nothing but an IC (Integrated Circuit) or chip, whereas, dice is the plural form of die (more than one IC).

Step 1 of 9

2.01DP

(a) The given binary number, 1101011_2

To convert the given binary number to its hexadecimal equivalent, separate the bits in the given binary number into groups of four bits and replace each group with corresponding hexadecimal digit. We can freely add zeroes on the left to make the total number of bits a multiple of 4 as required.

$$\begin{aligned}1101011_2 &= 0110\ \ 1011_2\\ &= 6B_{16}\end{aligned}$$

Thus, the required hexadecimal equivalent of the given binary number is $\boxed{6B_{16}}$.

Step 2 of 9

(b) The given octal number, 174003_8

To convert the given octal number to its binary equivalent, simply replace each octal digit with the corresponding 3 bit binary value string.

$$\begin{aligned}174003_8 &= 001\ 111\ 100\ 000\ 000\ 011_2\\ &= 1111100000000011_2\end{aligned}$$

Thus, the required binary equivalent of the given octal number is

$\boxed{1111100000000011_2}$.

Step 3 of 9

(c) The given binary number, 10110111_2

To convert the given binary number to its hexadecimal equivalent, separate the bits in the given binary number into groups of four bits and replace each group with corresponding hexadecimal digit.

$$\begin{aligned}10110111_2 &= 1011\ \ 0111_2\\ &= B7_{16}\end{aligned}$$

Thus, the required hexadecimal equivalent of the given binary number is $\boxed{B7_{16}}$.

Step 4 of 9

(d) The given octal number, 67.24_8

To convert the given octal number to its binary equivalent, simply replace each octal digit with the corresponding 3 bit binary value string.

$$\begin{aligned}67.24_8 &= 110\ 111.010\ 100_2\\ &= 110111.0101_2\end{aligned}$$

Thus, the required binary equivalent of the given octal number is $\boxed{110111.0101_2}$.

Step 5 of 9

(e) The given binary number, 10100.1101_2

To convert the given binary number to its hexadecimal equivalent, separate the bits in the given binary number into groups of four bits and replace each group with corresponding hexadecimal digit

$$\begin{aligned}10100.1101_2 &= 0001\ 0100.1101_2\\ &= 14.D_{16}\end{aligned}$$

Thus, the required hexadecimal equivalent of the given binary number is $\boxed{14.D_{16}}$.

Step 6 of 9

(f) The given hexadecimal number, $F3A5_{16}$

To convert the given octal number to its binary equivalent, simply replace each octal digit with the corresponding 4 bit binary value string.

$$\begin{aligned}F3A5_{16} &= 1111\ 0011\ 1010\ 0101_2\\ &= 1111001110100101_2\end{aligned}$$

Thus, the required binary equivalent of the given hexadecimal number is

$\boxed{1111001110100101_2}$.

Step 7 of 9

(g) The given binary number, 11011001_2

To convert the given binary number to its octal equivalent, separate the bits in the given binary number into groups of three bits and replace each group with corresponding octal digit. We can freely add zeroes on the left to make the total number of bits a multiple of three as required.

$$\begin{aligned}11011001_2 &= 011\ 011\ 001_2\\ &= 331_8\end{aligned}$$

Thus, the required octal equivalent of the given binary number is $\boxed{331_8}$.

(h) The given hexadecimal number, $AB3D_{16}$

To convert the given octal number to its binary equivalent, simply replace each octal digit with the corresponding 4 bit binary value string.

$$\begin{aligned}AB3D_{16} &= 1010\ 1011\ 0011\ 1101_2\\ &= 1010101100111101_2\end{aligned}$$

Thus, the required binary equivalent of the given hexadecimal number is

$\boxed{1010101100111101_2}$.

Step 8 of 9

(i) The given binary number, 101111.0111_2

To convert the given binary number to its octal equivalent, separate the bits in the given binary number into groups of three bits and replace each group with corresponding octal digit.

$$\begin{aligned}101111.0111_2 &= 101\ 111.011\ 100_2\\ &= 57.34_8\end{aligned}$$

Thus, the required octal equivalent of the given binary number is $\boxed{57.34_8}$.

Step 9 of 9

(j) The given hexadecimal number, $15C.38_{16}$

To convert the given octal number to its binary equivalent, simply replace each octal digit with the corresponding 4 bit binary value string.

$$\begin{aligned}15C.38_{16} &= 0001\ 0101\ 1100.0011\ 1000_2\\ &= 101011100.00111_2\end{aligned}$$

Thus, the required binary equivalent of the given hexadecimal number is

$\boxed{101011100.00111_2}$.

Step 1 of 11

(a) The given octal number, 1234_8

To determine binary equivalent of the given octal number replace each octal digit with the corresponding 3 bit binary value string.

$$\begin{aligned}1234_8 &= 001\ 010\ 011\ 100 \\ &= 1010011100_2\end{aligned}$$

Thus, the required binary equivalent of the given octal number is $\boxed{1010011100_2}$.

Step 2 of 11

Now, separate the bits in the binary equivalent of the given number into groups of four bits and replace each group with corresponding hexadecimal digit and freely add zeroes on the left to make the total of bits a multiple of four.

$$\begin{aligned}1010011100_2 &= 0010\ 1001\ 1100_2 \\ &= 29C_{16}\end{aligned}$$

Thus, the required hexadecimal equivalent of the given octal number is $\boxed{29C_{16}}$.

Step 3 of 11

(b) The given octal number, 174637_8

To determine binary equivalent of the given octal number replace each octal digit with the corresponding 3 bit binary value string.

$$\begin{aligned}174637_8 &= 001\ 111\ 100\ 110\ 011\ 111_2 \\ &= 1111100110011111_2\end{aligned}$$

Thus, the required binary equivalent of the given octal number is

$\boxed{1111100110011111_2}$.

Step 4 of 11

Now, separate the bits in the binary equivalent of the given number into groups of four bits and replace each group with corresponding hexadecimal digit.

$$\begin{aligned}1111100110011111_2 &= 1111\ 1001\ 1001\ 1111_2 \\ &= F99F_{16}\end{aligned}$$

Thus, the required hexadecimal equivalent of the given octal number is $\boxed{F99F_{16}}$.

Step 5 of 11

(c) The given octal number, 365517_8

To determine binary equivalent of the given octal number replace each octal digit with the corresponding 3 bit binary value string.

$$\begin{aligned}365517_8 &= 011\ 110\ 101\ 101\ 001\ 111_2 \\ &= 11110101101001111_2\end{aligned}$$

Thus, the required binary equivalent of the given octal number is

$\boxed{11110101101001111_2}$.

Step 6 of 11

Now, separate the bits in the binary equivalent of the given number into groups of four bits and replace each group with corresponding hexadecimal digit and freely add zeroes on the left to make the total of bits a multiple of four.

$$\begin{aligned}11110101101001111_2 &= 0001\ 1110\ 1011\ 0100\ 1111_2 \\ &= 1EB4F_{16}\end{aligned}$$

Thus, the required hexadecimal equivalent of the given octal number is $\boxed{1EB4F_{16}}$.

Step 7 of 11

(d) The given octal number, 2535321_8

To determine binary equivalent of the given octal number replace each octal digit with the corresponding 3 bit binary value string.

$$\begin{aligned}2535321_8 &= 010\ 101\ 011\ 101\ 011\ 010\ 001_2 \\ &= 10101011101011010001_2\end{aligned}$$

Thus, the required binary equivalent of the given octal number is

$\boxed{10101011101011010001_2}$.

Step 8 of 11

Now, separate the bits in the binary equivalent of the given number into groups of four bits and replace each group with corresponding hexadecimal digit.

$$\begin{aligned}10101011101011010001_2 &= 1010\ 1011\ 1010\ 1101\ 0001_2 \\ &= ABAD1_{16}\end{aligned}$$

Thus, the required hexadecimal equivalent of the given octal number is

$\boxed{ABAD1_{16}}$.

(e) The given octal number, 7436.11_8

To determine binary equivalent of the given octal number replace each octal digit with the corresponding 3 bit binary value string.

$$\begin{aligned}7436.11_8 &= 111\ 100\ 011\ 110.001\ 001_2 \\ &= 111100011110.001001_2\end{aligned}$$

Thus, the required binary equivalent of the given octal number is

$\boxed{111100011110.001001_2}$.

Step 9 of 11

Now, separate the bits in the binary equivalent of the given number into groups of four bits and replace each group with corresponding hexadecimal digit.

$$\begin{aligned}111100011110.001001_2 &= 1111\ 0001\ 1110.0010\ 0100_2 \\ &= F1E.24_{16}\end{aligned}$$

Thus, the required hexadecimal equivalent of the given octal number is

$\boxed{F1E.24_{16}}$.

Step 10 of 11

(f) The given octal number, 45316.7414_8

To determine binary equivalent of the given octal number replace each octal digit with the corresponding 3 bit binary value string.

$$\begin{aligned}45316.7414_8 &= 100\ 101\ 011\ 001\ 110.111\ 100\ 001\ 100_2 \\ &= 100101011001110.1111000011_2\end{aligned}$$

Thus, the required binary equivalent of the given octal number is

$\boxed{100101011001110.1111000011_2}$.

Step 11 of 11

Now, separate the bits in the binary equivalent of the given number into groups of four bits and replace each group with corresponding hexadecimal digit.

$$\begin{aligned}100101011001110.1111000011_2 &= 0100\ 1010\ 1100\ 1110.1111\ 0000\ 1100_2 \\ &= 4ACE.F0E_{16}\end{aligned}$$

Thus, the required hexadecimal equivalent of the given octal number is

$\boxed{4ACE.F0E_{16}}$.

Step 1 of 11

2.03DP

(a) The given hexadecimal number, $\mathbf{1023_{16}}$

To convert the given hexadecimal number to its binary equivalent, simply replace each hexadecimal digit with the corresponding 4 bit binary value string.

$$\begin{aligned}\mathbf{1023_{16}} &= \mathbf{0001\ 0000\ 0010\ 0011_2}\\ &= \mathbf{1000000100011_2}\end{aligned}$$

Thus, the required binary equivalent of the given hexadecimal number is

$\boxed{\mathbf{1000000100011_2}}$.

Step 2 of 11

Now, separate the bits in the binary equivalent of the given number into groups of three bits and replace each group with corresponding octal digit and freely add zeroes on the left to make the total of bits a multiple of three.

$$\begin{aligned}\mathbf{1000000100011_2} &= \mathbf{001\ 000\ 000\ 100\ 011_2}\\ &= \mathbf{10043_8}\end{aligned}$$

Thus, the required octal equivalent of the given hexadecimal number is $\boxed{\mathbf{10043_8}}$.

Step 3 of 11

(b) The given hexadecimal number, $\mathbf{7E6A_{16}}$

To convert the given hexadecimal number to its binary equivalent, simply replace each hexadecimal digit with the corresponding 4 bit binary value string.

$$\begin{aligned}\mathbf{7E6A_{16}} &= \mathbf{0111\ 1110\ 0110\ 1010_2}\\ &= \mathbf{111111001101010_2}\end{aligned}$$

Thus, the required binary equivalent of the given hexadecimal number is

$\boxed{\mathbf{111111001101010_2}}$.

Step 4 of 11

Now, separate the bits in the binary equivalent of the given number into groups of three bits and replace each group with corresponding octal digit and freely add zeroes on the left to make the total of bits a multiple of three.

$$\begin{aligned}\mathbf{111111001101010_2} &= \mathbf{111\ 111\ 001\ 101\ 010_2}\\ &= \mathbf{77152_8}\end{aligned}$$

Thus, the required octal equivalent of the given hexadecimal number is $\boxed{\mathbf{77152_8}}$.

Step 5 of 11

(c) The given hexadecimal number, $\mathbf{ABCD_{16}}$

To convert the given hexadecimal number to its binary equivalent, simply replace each hexadecimal digit with the corresponding 4 bit binary value string.

$$\begin{aligned}\mathbf{ABCD_{16}} &= \mathbf{1010\ 1011\ 1100\ 1101_2}\\ &= \mathbf{1010101111001101_2}\end{aligned}$$

Thus, the required binary equivalent of the given hexadecimal number is

$\boxed{\mathbf{1010101111001101_2}}$.

Step 6 of 11

Now, separate the bits in the binary equivalent of the given number into groups of three bits and replace each group with corresponding octal digit and add zeroes on the left to make the total of bits a multiple of three.

$$\begin{aligned}\mathbf{1010101111001101_2} &= \mathbf{001\ 010\ 101\ 111\ 001\ 101_2}\\ &= \mathbf{125715_8}\end{aligned}$$

Thus, the required octal equivalent of the given hexadecimal number is $\boxed{\mathbf{125715_8}}$.

Step 7 of 11

(d) The given hexadecimal number, $\mathbf{C350_{16}}$

To convert the given hexadecimal number to its binary equivalent, simply replace each hexadecimal digit with the corresponding 4 bit binary value string.

$$\begin{aligned}\mathbf{C350_{16}} &= \mathbf{1100\ 0011\ 0101\ 0000_2}\\ &= \mathbf{1100001101010000_2}\end{aligned}$$

Thus, the required binary equivalent of the given hexadecimal number is

$\boxed{\mathbf{1100001101010000_2}}$.

Step 8 of 11

Now, separate the bits in the binary equivalent of the given number into groups of three bits and replace each group with corresponding octal digit and add zeroes on the left to make the total of bits a multiple of three.

$$\begin{aligned}\mathbf{1100001101010000_2} &= \mathbf{001\ 100\ 001\ 101\ 010\ 000_2}\\ &= \mathbf{141520_8}\end{aligned}$$

Thus, the required octal equivalent of the given hexadecimal number is $\boxed{\mathbf{141520_8}}$.

(e) The given hexadecimal number, $\mathbf{9E36.7A_{16}}$

To convert the given hexadecimal number to its binary equivalent, simply replace each hexadecimal digit with the corresponding 4 bit binary value string.

$$\begin{aligned}\mathbf{9E36.7A_{16}} &= \mathbf{1001\ 1110\ 0011\ 0110.0111\ 1010_2}\\ &= \mathbf{1001111000110110.01111010_2}\end{aligned}$$

Thus, the required binary equivalent of the given hexadecimal number is

$\boxed{\mathbf{1001111000110110.01111010_2}}$.

Step 9 of 11

Now, separate the bits in the binary equivalent of the given number into groups of three bits and replace each group with corresponding octal digit and add zeroes on the left to make the total of bits a multiple of three.

$$\begin{aligned}\mathbf{1001111000110110.01111010_2} &= \mathbf{001\ 001\ 111\ 000\ 110\ 110.011\ 110\ 100_2}\\ &= \mathbf{117066.364_8}\end{aligned}$$

Thus, the required octal equivalent of the given hexadecimal number is

$\boxed{\mathbf{117066.364_8}}$.

Step 10 of 11

(f) The given hexadecimal number, $\mathbf{DEAD.BEEF_{16}}$

To convert the given hexadecimal number to its binary equivalent, simply replace each hexadecimal digit with the corresponding 4 bit binary value string.

$$\begin{aligned}\mathbf{DEAD.BEEF_{16}} &= \mathbf{1101\ 1110\ 1010\ 1101.1011\ 1110\ 1110\ 1111_2}\\ &= \mathbf{1101111010101101.1011111011101111_2}\end{aligned}$$

Thus, the required binary equivalent of the given hexadecimal number is

$\boxed{\mathbf{1101111010101101.1011111011101111_2}}$.

Step 11 of 11

Separate the bits in the binary equivalent of the given number into groups of three bits and replace each group with corresponding octal digit and add zeroes on the left to make the total of bits a multiple of three.

$$\begin{aligned}&\mathbf{1101111010101101.1011111011101111_2}\\ &= \mathbf{(001\ 101\ 111\ 010\ 101\ 101.101\ 111\ 101\ 110\ 111\ 100_2)}\\ &= \mathbf{157255.575654_8}\end{aligned}$$

Thus, the required octal equivalent of the given hexadecimal number is

$\boxed{\mathbf{157255.575654_8}}$.

Step 1 of 2

The given 32 bit number with octal representation, 32107654321_8 .

To obtain binary equivalent of the given octal number, simply replace each octal digit with the corresponding 3 bit binary value string.

$$32107654321_8 = 011\ 010\ 001\ 000\ 111\ 110\ 101\ 100\ 011\ 010\ 001_2$$

Now, the four 8-bit bytes of the of the 32 bit number is determined as follows,

$$32107654321_8 = 11010001\ \ 00011111\ \ 01011000\ \ 11010001_2$$

Step 2 of 2

Now, convert each 8 bit bytes into its octal equivalent. Binary to octal form is done by separating the binary bits into groups of three bits and replace each group with corresponding octal digit.

$$11010001_2 = 011\ 010\ 001_2 = 321_8$$
$$00011111_2 = 000\ 011\ 111_2 = 037_8$$
$$01011000_2 = 001\ 011\ 000_2 = 130_8$$
$$11010001_2 = 011\ 010\ 001_2 = 321_8$$

Thus, the required octal values of the four 8-bit bytes in the 32-bit number are

$\boxed{321_8, 037_8, 130_8, \text{ and } 321_8}$.

Step 1 of 10

2.05DP

The number conversion formula from any radix to decimal (radix 10) is

$$D=\sum_{i=-n}^{p-1} d_i r^i$$

Where,

$r=$ Radix of the number

$p=$ Digits to the left of the radix point and *n* to the right

Step 2 of 10

(a) The given binary number, 1101011_2

The conversion from the binary to decimal number is as follows,

$$\begin{aligned}1101011_2&=1\times2^6+1\times2^5+0\times2^4+1\times2^3+0\times2^2+1\times2^1+1\times2^0\\&=64+32+8+2+1\\&=107\end{aligned}$$

Thus, the required decimal number of the given binary number is $\boxed{107_{10}}$.

Step 3 of 10

(b) The given octal number, 174003_8

The conversion from octal number to decimal number is as follows,

$$\begin{aligned}174003_8&=1\times8^5+7\times8^4+4\times8^3+0\times8^2+0\times8^1+3\times8^0\\&=32,768+28,672+2,048+3\\&=63,491\end{aligned}$$

Thus, the required decimal equivalent of the given octal number is $\boxed{63,491_{10}}$.

Step 4 of 10

(c) The given binary number, 10110111_2

The conversion from the binary to decimal number is as follows,

$$\begin{aligned}10110111_2&=1\times2^7+0\times2^6+1\times2^5+1\times2^4+0\times2^3+1\times2^2+1\times2^1+1\times2^0\\&=128+0+32+16+0+4+2+1\\&=183\end{aligned}$$

Thus, the required decimal equivalent of the given binary number is $\boxed{183_{10}}$.

Step 5 of 10

(d) The given octal number, 67.24_8

The conversion from octal number to decimal number is as follows,

$$\begin{aligned}67.24_8&=6\times8^1+7\times8^0+2\times8^{-1}+4\times8^{-2}\\&=48+7+0.25+0.0625\\&=55.3125\end{aligned}$$

Thus, the required decimal equivalent of the given octal number is $\boxed{55.3125_{10}}$.

Step 6 of 10

(e) The given binary number, 10100.1101_2

The conversion from the binary to decimal number is as follows,

$$\begin{aligned}10100.1101_2&=1\times2^4+0\times2^3+1\times2^2+0\times2^1+0\times2^0+1\times2^{-1}+1\times2^{-2}+0\times2^{-3}+1\times2^{-4}\\&=16+0+4+0+0+0.5+0.25+0+0.0625\\&=20.8125\end{aligned}$$

Thus, the required decimal equivalent of the given octal number is $\boxed{20.8125_{10}}$.

Step 7 of 10

(f) The given hexadecimal number, $F3A5_{16}$

The conversion from the hexadecimal to decimal number is as follows,

$$\begin{aligned}F3A5_{16}&=15\times16^3+3\times16^2+10\times16^1+5\times16^0\\&=61,440+768+160+5\\&=62,373\end{aligned}$$

Thus, the required decimal equivalent of the given hexadecimal number is

$\boxed{62,373_{10}}$.

Step 8 of 10

(g) The given number, 12010_3

The decimal equivalent of the given number is determined as follows,

$$\begin{aligned}12010_3&=1\times3^4+2\times3^3+0\times3^2+1\times3^1+0\times3^0\\&=81+54+0+3+0\\&=138\end{aligned}$$

Thus, the required decimal equivalent of the given number is $\boxed{138_{10}}$.

(h) The given hexadecimal number, $AB3D_{16}$

The decimal equivalent of the given hexadecimal number is determined as follows,

$$\begin{aligned}AB3D_{16}&=10\times16^3+11\times16^2+3\times16^1+13\times16^0\\&=40,960+2,816+48+13\\&=43,837\end{aligned}$$

Thus, the required decimal equivalent of the given hexadecimal number is

$\boxed{43,837_{10}}$.

Step 9 of 10

(i) The given octal number, 7156_8

The decimal equivalent of the given octal number is determined as follows,

$$\begin{aligned}7156_8&=7\times8^3+1\times8^2+5\times8^1+6\times8^0\\&=3584+64+40+6\\&=3,694\end{aligned}$$

Thus, the required decimal equivalent of the given octal number is $\boxed{3,694_{10}}$.

Step 10 of 10

(j) The given hexadecimal number, $15C.38_{16}$

The decimal equivalent of the given hexadecimal number is determined as follows,

$$\begin{aligned}15C.38_{16}&=1\times16^2+5\times16^1+12\times16^0+3\times16^{-1}+8\times16^{-2}\\&=256+80+12+0.1875+0.03125\\&=348.21875\end{aligned}$$

Thus, the required decimal equivalent of the given hexadecimal number is

$\boxed{348.21875_{10}}$.

Step 1 of 10

The number conversion from decimal to other radix form is done by successive division of radix and the remainder will yield successive digits of corresponding radix form from left to right.

Step 2 of 10

(a) The given decimal number, 125_{10}

Decimal to binary conversion is done by successive division of radix 2 and the remainder will yield successive digits from left to right.

$125 \div 2 = 62$ remainder 1 (Least significant bit)
$62 \div 2 = 31$ remainder 0
$31 \div 2 = 15$ remainder 1
$15 \div 2 = 7$ remainder 1
$7 \div 2 = 3$ remainder 1
$3 \div 2 = 1$ remainder 1
$1 \div 2 = 0$ remainder 1 (Most significant bit)

Thus, the required binary equivalent of the given decimal number is $\boxed{1111101_2}$.

Step 3 of 10

(b) The given decimal number, $3{,}489_{10}$

Decimal to octal conversion is done by successive division of radix 8 and the remainder will yield successive digits from left to right.

$3{,}489 \div 8 = 436$ remainder 1 (Least significant bit)
$436 \div 8 = 54$ remainder 4
$54 \div 8 = 6$ remainder 6
$6 \div 8 = 0$ remainder 6 (Most significant bit)

Thus, the required octal equivalent of the given decimal number is $\boxed{6641_8}$.

Step 4 of 10

(c) The given decimal number, 209_{10}

Decimal to binary conversion is done by successive division of radix 2 and the remainder will yield successive digits from left to right.

$209 \div 2 = 104$ remainder 1 (Least significant bit)
$104 \div 2 = 52$ remainder 0
$52 \div 2 = 26$ remainder 0
$26 \div 2 = 13$ remainder 0
$13 \div 2 = 6$ remainder 1
$6 \div 2 = 3$ remainder 0
$3 \div 2 = 1$ remainder 1
$1 \div 2 = 0$ remainder 1 (Most significant bit)

Thus, the required binary equivalent of the given decimal number is $\boxed{11010001_2}$.

Step 5 of 10

(d) The given decimal number, $9{,}714_{10}$

Decimal to octal conversion is done by successive division of radix 8 and the remainder will yield successive digits from left to right.

$9{,}714 \div 8 = 1214$ remainder 2 (Least significant bit)
$1{,}214 \div 8 = 151$ remainder 6
$151 \div 8 = 18$ remainder 7
$18 \div 8 = 2$ remainder 2
$2 \div 8 = 0$ remainder 2 (Most significant bit)

Thus, the required octal equivalent of the given decimal number is $\boxed{22762_8}$.

Step 6 of 10

(e) The given decimal number, 132_{10}

Decimal to binary conversion is done by successive division of radix 2 and the remainder will yield successive digits from left to right.

$132 \div 2 = 66$ remainder 0 (Least significant bit)
$66 \div 2 = 33$ remainder 0
$33 \div 2 = 16$ remainder 1
$16 \div 2 = 8$ remainder 0
$8 \div 2 = 4$ remainder 0
$4 \div 2 = 2$ remainder 0
$2 \div 2 = 1$ remainder 0
$1 \div 2 = 0$ remainder 1 (Most significant bit)

Thus, the required binary equivalent of the given decimal number is $\boxed{10000100_2}$.

(f) The given decimal number, $23{,}851_{10}$

Decimal to hexadecimal conversion is done by successive division of radix 16 and the remainder will yield successive digits from left to right.

$23{,}851 \div 16 = 1{,}490$ remainder B (Least significant bit)
$14{,}90 \div 16 = 93$ remainder 2
$93 \div 16 = 5$ remainder D
$5 \div 16 = 0$ remainder 5 (Most significant bit)

Thus, the required hexadecimal equivalent of the given decimal number is

$\boxed{5D2B_{16}}$.

Step 7 of 10

(g) The given decimal number, 727_{10}

Decimal to radix 5 conversion is done by successive division of radix 5 and the remainder will yield successive digits from left to right.

$727 \div 5 = 145$ remainder 2 (Least significant bit)
$145 \div 5 = 29$ remainder 0
$29 \div 5 = 5$ remainder 4
$5 \div 5 = 1$ remainder 0
$1 \div 5 = 0$ remainder 1 (Most significant bit)

Thus, the required radix 5 equivalent of the given decimal number is $\boxed{10402_5}$.

Step 8 of 10

(h) The given decimal number, $57{,}190_{10}$

Decimal to hexadecimal conversion is done by successive division of radix 16 and the remainder will yield successive digits from left to right.

$57{,}190 \div 16 = 35{,}74$ remainder 6 (Least significant bit)
$35{,}74 \div 16 = 223$ remainder 6
$223 \div 16 = 13$ remainder F
$13 \div 16 = 0$ remainder D (Most significant bit)

Thus, the required hexadecimal equivalent of the given decimal number is

$\boxed{DF66_{16}}$.

Step 9 of 10

(i) The given decimal number, $1{,}435_{10}$

Decimal to octal conversion is done by successive division of radix 8 and the remainder will yield successive digits from left to right.

$1{,}435 \div 8 = 179$ remainder 3 (Least significant bit)
$179 \div 8 = 22$ remainder 3
$22 \div 8 = 2$ remainder 6
$2 \div 8 = 0$ remainder 2 (Most significant bit)

Thus, the required octal equivalent of the given decimal number is $\boxed{2633_8}$.

Step 10 of 10

(j) The given decimal number, 65113_{10}

Decimal to hexadecimal conversion is done by successive division of radix 16 and the remainder will yield successive digits from left to right.

$65{,}113 \div 16 = 4{,}069$ remainder 9 (Least significant bit)
$4{,}069 \div 16 = 254$ remainder 5
$254 \div 16 = 15$ remainder E
$15 \div 16 = 0$ remainder F (Most significant bit)

Thus, the required hexadecimal equivalent of the given decimal number is

$\boxed{FE59_{16}}$.

Step 1 of 6

The binary addition of two numbers X and Y is shown in Table (1).

Table 1:

C_{in}	X	Y	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Step 2 of 6

Where X and Y are two binary numbers, C_{in} is Carry input to the addition of binary numbers, C_{out} is Carry output from the addition of the binary numbers and Sum is the addition of two binary numbers.

Addition of two binary numbers are done by adding two binary number X and Y, together with least significant bits with an initial carry (C_{in}) of 0, producing carry (C_{out}) and sum (S) according to the Table (1).

Step 3 of 6

(a) The addition of given two binary numbers is determined as follows,

$$\begin{array}{lr} C & 1100100 \\ X & 110011 \\ Y & +11010 \\ \hline \text{Sum} & 1001101 \end{array}$$

Thus, the addition of given two binary numbers results in the sum, $\boxed{1001101}$ and a carry $\boxed{1100100}$.

Step 4 of 6

(b) The addition of given two binary numbers is determined as follows,

$$\begin{array}{lr} C & 1011100 \\ X & 100111 \\ Y & +101010 \\ \hline \text{Sum} & 1010001 \end{array}$$

Thus, the addition of given two binary numbers results in the sum, $\boxed{1010001}$ and a carry $\boxed{1011100}$.

Step 5 of 6

(c) The addition of given two binary numbers is determined as follows,

$$\begin{array}{lr} C & 111111110 \\ X & 11100011 \\ Y & +1011101 \\ \hline \text{Sum} & 101000000 \end{array}$$

Thus, the addition of given two binary numbers results in the sum, $\boxed{101000000}$ and a carry $\boxed{111111110}$.

Step 6 of 6

(d) The addition of given two binary numbers is determined as follows,

$$\begin{array}{lr} C & 11000000 \\ X & 1100110 \\ Y & +1111001 \\ \hline \text{Sum} & 11011111 \end{array}$$

Thus, the addition of given two binary numbers results in the sum, $\boxed{11011111}$ and a carry $\boxed{11000000}$.

Step 1 of 6

The binary subtraction of two numbers X and Y is shown in Table (1).

Table 1:

B_{in}	X	Y	B_{out}	D
0	0	0	0	0
0	0	1	1	1
0	1	0	0	1
0	1	1	0	0
1	0	0	1	1
1	0	1	1	0
1	1	0	0	0
1	1	1	1	1

Step 2 of 6

Where X and Y are two binary numbers, B_{in} is borrow input to the subtraction of binary numbers, B_{out} is borrow output from the subtraction of the binary numbers and *D* is the subtraction of two binary numbers.

Subtraction of two binary numbers is done by difference between two binary numbers X and Y. If $X-Y$ produces a binary out of the most significant bit position then X is less than Y otherwise X is greater than or equal to Y.

Step 3 of 6

(a) The subtraction of the given two binary numbers is determined as follows,

$$\begin{array}{lr} B & 0110000 \\ X & 110011 \\ Y & -\ 11010 \\ \hline D & 011001 \end{array}$$

Thus, the subtraction of given two binary numbers results in the difference, $\boxed{0011001}$ and a borrow, $\boxed{0110000}$.

Step 4 of 6

(b) The subtraction of the given two binary numbers is determined as follows,

$$\begin{array}{lr} B & 1110000 \\ X & 100111 \\ Y & -101010 \\ \hline D & 111101 \end{array}$$

Thus, the subtraction of given two binary numbers results in the difference, $\boxed{111101}$ and a borrow, $\boxed{1110000}$.

Step 5 of 6

(c) The subtraction of the given two binary numbers is determined as follows,

$$\begin{array}{lr} B & 00111000 \\ X & 11100011 \\ Y & -1011101 \\ \hline \mathrm{D} & 10000110 \end{array}$$

Thus, the subtraction of given two binary numbers results in the difference, $\boxed{10000110}$ and a borrow, $\boxed{00111000}$.

Step 6 of 6

(d) The subtraction of the given two binary numbers is determined as follows,

$$\begin{array}{lr} B & 11110010 \\ X & 1100110 \\ Y & -1111001 \\ \hline D & 1101101 \end{array}$$

Thus, the subtraction of given two binary numbers results in the difference, $\boxed{1101101}$ and a borrow, $\boxed{11110010}$.

Step 1 of 4

2.09DP

(a) The addition of the given two octal numbers is determined as follows,

C	**1**	**1**	**1**	**0**
X	**1**	**7**	**7**	**6**
Y	**1**	**4**	**3**	**2**
	3	**12**	**11**	**8**
	3	**8+4**	**8+3**	**8+0**
sum	**3**	**4**	**3**	**0**

Thus, the addition of given two octal numbers results in the sum, $\boxed{3430}$ and a carry, $\boxed{1110}$.

Step 2 of 4

(b) The addition of the given two octal numbers is determined as follows,

C	**1**	**1**	**1**	**1**	**0**
X	**5**	**7**	**7**	**3**	**4**
Y	**+**	**1**	**0**	**6**	**6**
	6	**9**	**8**	**10**	**10**
	6	**8+1**	**8+0**	**8+2**	**8+2**
Sum	**6**	**1**	**0**	**2**	**2**

Thus, the addition of given two octal numbers results in the sum, $\boxed{61022}$ and a carry, $\boxed{11110}$.

Step 3 of 4

(c) The addition of the given two octal numbers is determined as follows,

C	**1**	**1**	**1**	**1**	**1**	**0**
X	**2**	**5**	**2**	**7**	**5**	**7**
Y	**4**	**6**	**5**	**5**	**2**	**1**
	7	**12**	**8**	**13**	**8**	**8**
	7	**8+4**	**8+0**	**8+5**	**8+0**	**8+0**
Sum	**7**	**4**	**0**	**5**	**0**	**0**

Thus, the addition of given two octal numbers results in the sum, $\boxed{740500}$ and a carry, $\boxed{111110}$.

Step 4 of 4

(d) The addition of the given two octal numbers is determined as follows,

C	**0**	**1**	**0**	**1**	**1**	**0**
X	**5**	**1**	**1**	**0**	**4**	**2**
Y	**+**	**5**	**7**	**6**	**4**	**7**
	5	**7**	**8**	**7**	**9**	**9**
	5	**7**	**8+0**	**7**	**8+1**	**8+1**
Sum	**5**	**7**	**0**	**7**	**1**	**1**

Thus, the addition of given two octal numbers results in the sum, $\boxed{570711}$ and a carry, $\boxed{010110}$.

2.10DP

Step 1 of 4

(a) The addition of the given two hexadecimal numbers is determined as follows,

C	0	0	0	0
X	1	7	7	6
Y	1	4	3	2
	2	11	10	8
	2	B	A	8
sum	2	B	A	8

Thus, the addition of the given two hexadecimal numbers results in the sum, $\boxed{2BA8}$ and a carry, $\boxed{0000}$.

Step 2 of 4

(b) The addition of the given two hexadecimal numbers is determined as follows,

C	1	0	1	0	0
X	4	F	1	A	5
Y	+	B	8	D	5
	5	26	10	23	10
	5	16+10	A	16+7	A
Sum	5	A	A	7	A

Thus, the addition of the given two hexadecimal numbers results in the sum, $\boxed{5AA7A}$ and a carry, $\boxed{10100}$.

Step 3 of 4

(c) The addition of the given two hexadecimal numbers is determined as follows,

C	1	0	1	1	0
X		F	3	5	B
Y	+	2	7	E	6
	1	17	11	20	17
	1	16+1	B	16+4	16+1
Sum	1	1	B	4	1

Thus, the addition of the given two hexadecimal numbers results in the sum, $\boxed{11B41}$ and a carry, $\boxed{10110}$.

Step 4 of 4

(d) The addition of the given two hexadecimal numbers is determined as follows,

C	1	0	0	1	0
X	1	B	9	0	F
Y	+	C	4	4	E
	2	23	13	5	29
	2	16+7	D	5	16+13
Sum	2	7	D	5	D

Thus, the addition of the given two hexadecimal numbers results in the sum, $\boxed{27D5D}$ and a carry, $\boxed{10010}$.

2.11DP

Step 1 of 23

(a) Determine the binary representation of +25

$$\begin{array}{l} 25 \div 2 = 12 \text{ remainder } 1 \text{ (Least significant bit)} \\ 12 \div 2 = 6 \text{ remainder } 0 \\ 6 \div 2 = 3 \text{ remainder } 0 \\ 3 \div 2 = 1 \text{ remainder } 1 \\ 1 \div 2 = 0 \text{ remainder } 1 \text{ (Most significant bit)} \end{array}$$

Thus, the binary representation of decimal number +25 is 11001_2.

Step 2 of 23

The following is the representation of 8-bit signed magnitude:

$$+25_{10} = 00011001_2$$

The most significant bit represents the sign, if it is '0' then the number is positive, else the number is negative.

Thus, the 8-bit signed magnitude of decimal number +25 is $\boxed{00011001_2}$.

Step 3 of 23

Find the two's complement of +25.

$$\begin{array}{r} +25_{10} = 00011001_2 \\ \Downarrow \\ 11100110_2 \quad \text{Complement bits} \\ 1\,+ \\ \hline 11100111_2 \end{array}$$

Thus, the two's complement of decimal number +25 is $\boxed{11100111_2}$.

Step 4 of 23

Find the one's complement of +25

$$\begin{array}{r} +25_{10} = 00011001_2 \\ \Downarrow \quad \text{Complement bits} \\ 11100110_2 \end{array}$$

Thus, the one's complement of decimal number +25 is $\boxed{11100110_2}$

Step 5 of 23

(b) Determine the binary representation of +120

$$\begin{array}{l} 120 \div 2 = 60 \text{ remainder } 0 \text{ (Least significant bit)} \\ 60 \div 2 = 30 \text{ remainder } 0 \\ 30 \div 2 = 15 \text{ remainder } 0 \\ 15 \div 2 = 7 \text{ remainder } 1 \\ 7 \div 2 = 3 \text{ remainder } 1 \\ 3 \div 2 = 1 \text{ remainder } 1 \\ 1 \div 2 = 0 \text{ remainder } 1 \text{ (Most significant bit)} \end{array}$$

Thus, the binary representation of decimal number +120 is given by 1111000_2.

Step 6 of 23

The following is the representation of 8-bit signed magnitude:

$$+120_{10} = 01111000_2$$

The most significant bit represents the sign, if it is '0' then the number is positive else the number is negative.

Thus, the 8-bit signed magnitude of decimal number +120 is $\boxed{01111000_2}$.

Step 7 of 23

Find the two's complement of +120

$$\begin{array}{r} +120_{10} = 01111000_2 \\ \Downarrow \quad \text{Complement bits} \\ 10000111_2 \\ 1\,+ \\ \hline 10001000_2 = -120_{10} \end{array}$$

Thus, the two's complement of decimal number +120 is $\boxed{10001000_2}$.

Step 8 of 23

Find the one's complement of +120

$$\begin{array}{r} +120_{10} = 01111000_2 \\ \Downarrow \quad \text{Complement bits} \\ 10000111_2 \end{array}$$

Thus, the one's complement of decimal number +120 is $\boxed{10000111_2}$.

Step 8 of 23

(c) Determine the binary representation of +82

$$\begin{array}{l} 82 \div 2 = 41 \text{ remainder } 0 \text{ (Least significant bit)} \\ 41 \div 2 = 20 \text{ remainder } 1 \\ 20 \div 2 = 10 \text{ remainder } 0 \\ 10 \div 2 = 5 \text{ remainder } 0 \\ 5 \div 2 = 2 \text{ remainder } 1 \\ 2 \div 2 = 1 \text{ remainder } 0 \\ 1 \div 2 = 0 \text{ remainder } 1 \text{ (Most significant bit)} \end{array}$$

Thus the binary representation of decimal number +82 is 1010010_2.

The following is the representation of 8-bit signed magnitude:

$$+82_{10} = 01010010_2$$

The most significant bit represents the sign, if it is '0' then the number is positive else the number is negative.

Thus, the 8-bit signed magnitude of decimal number +82 is $\boxed{01010010_2}$.

Step 10 of 23

Find the two's complements of +82

$$\begin{array}{r} +82_{10} = 01010010_2 \\ \Downarrow \\ 10101101_2 \quad \text{Complement bits} \\ 1\,+ \\ \hline 10101110_2 \end{array}$$

Thus, the two's complement of decimal number +82 is $\boxed{10101110_2}$.

Step 11 of 23

Find the one's complement of +82

$$\begin{array}{r} +82_{10} = 01010010_2 \\ \Downarrow \quad \text{Complement bits} \\ 10101101_2 \end{array}$$

Thus, the one's complement of decimal number +82 is $\boxed{10101101_2}$.

Step 12 of 23

(d) Determine the binary representation of 42.

$$\begin{array}{l} 42 \div 2 = 21 \text{ remainder } 0 \text{ (Least significant bit)} \\ 21 \div 2 = 10 \text{ remainder } 1 \\ 10 \div 2 = 5 \text{ remainder } 0 \\ 5 \div 2 = 2 \text{ remainder } 1 \\ 2 \div 2 = 1 \text{ remainder } 0 \\ 1 \div 2 = 0 \text{ remainder } 1 \text{ (Most significant bit)} \end{array}$$

Thus the binary representation of decimal number 42 is 101010_2.

Step 13 of 23

The following is the representation of 8-bit signed magnitude:

$$-42_{10} = 10101010_2$$

The most significant bit represents the sign, if it is '0' then the number is positive else the number is negative.

Thus, the 8-bit signed magnitude of decimal number -42 is given by $\boxed{10101010_2}$.

Step 14 of 23

Find the binary value of -42_{10}

$$\begin{array}{r} 42_{10} = 00101010_2 \\ \Downarrow \\ 11010101_2 \\ 1\,+ \\ \hline 11010110_2 \end{array}$$

Thus, the binary value of -42_{10} is 11010110_2

Find the two's complements of -42_{10}

$$\begin{array}{r} -42_{10} = 11010110_2 \\ \Downarrow \\ 00101001_2 \\ 1\,+ \\ \hline 00101010_2 \end{array}$$

Thus, the two's complement of decimal number -42_{10} is $\boxed{00101010_2}$.

Step 15 of 23

Find the one's complement.

$$\begin{array}{r} -42_{10} = 11010110_2 \\ \Downarrow \quad \text{Complement bits} \\ 00101001_2 \end{array}$$

Thus, the one's complement of decimal number -42_{10} is $\boxed{00101001_2}$.

Step 16 of 23

(e) Determine the binary representation of 6.

$$\begin{array}{l} 6 \div 2 = 3 \text{ remainder } 0 \text{ (Least significant bit)} \\ 3 \div 2 = 1 \text{ remainder } 1 \\ 1 \div 2 = 0 \text{ remainder } 1 \text{ (Most significant bit)} \end{array}$$

Thus the binary representation of decimal number 6 is 110_2.

Step 17 of 23

The following is the representation of 8-bit signed magnitude:

$$-6_{10} = 10000110_2$$

The most significant bit represents the sign, if it is '0' then the number is positive else the number is negative.

Thus, the 8-bit signed magnitude of decimal number -6 is given by $\boxed{10000110_2}$.

Find the binary value of -6_{10}.

$$\begin{array}{r} 6_{10} = 00000110_2 \\ \Downarrow \\ 11111001_2 \quad \text{Complement bits} \\ 1\,+ \\ \hline 11111010 \end{array}$$

Thus, the binary value of decimal number -6_{10} is $\boxed{11111010_2}$.

Find the two's complement of decimal number -6_{10}

$$\begin{array}{r} -6_{10} = 11111010_2 \\ \Downarrow \\ 00000101_2 \\ 1\,+ \\ \hline 00000110 \end{array}$$

Thus the two's complement of -6_{10} is $\boxed{00000110_2}$.

Step 18 of 23

Find the one's complement.

$$\begin{array}{r} -6_{10} = 11111010_2 \\ \Downarrow \quad \text{Complement bits} \\ 00000101_2 \end{array}$$

Thus, the one's complement of decimal number -6 is $\boxed{00000101_2}$.

Step 19 of 23

(f) Determine the binary representation of 111

$$\begin{array}{l} 111 \div 2 = 55 \text{ remainder } 1 \text{ (Least significant bit)} \\ 55 \div 2 = 27 \text{ remainder } 1 \\ 27 \div 2 = 13 \text{ remainder } 1 \\ 13 \div 2 = 6 \text{ remainder } 1 \\ 6 \div 2 = 3 \text{ remainder } 0 \\ 3 \div 2 = 1 \text{ remainder } 1 \\ 1 \div 2 = 0 \text{ remainder } 1 \text{ (Most significant bit)} \end{array}$$

Thus the binary representation of decimal number 111 is 1101111_2.

Step 20 of 23

The following is the representation of 8-bit signed magnitude.

$$-111_{10} = 11101111_2$$

The most significant bit represents the sign, if it is '0' then the number is positive else the number is negative.

Thus, the 8-bit signed magnitude of decimal number -111 is $\boxed{11101111_2}$.

Step 21 of 23

Find the binary value of -111_{10}

$$\begin{array}{r} 111_{10} = 01101111_2 \\ \Downarrow \\ 10010000_2 \\ 1\,+ \\ \hline 10010001_2 \end{array}$$

Thus, the binary value for -111_{10} is $\boxed{10010001_2}$.

Step 22 of 23

Find the two's complement.

$$\begin{array}{r} -111_{10} = 10010001_2 \\ \Downarrow \\ 01101110_2 \quad \text{Complement bits} \\ 1\,+ \\ \hline 01101111_2 \end{array}$$

Thus, the two's complement of decimal number -111 is $\boxed{01101111_2}$.

Step 23 of 23

Find the one's complement.

$$\begin{array}{r} -111_{10} = 10010001_2 \\ \Downarrow \quad \text{Complement bits} \\ 01101110_2 \end{array}$$

Thus, the one's complement of decimal number -111 is $\boxed{01101110_2}$.

2.12DP

Step 1 of 4

Overflow occurs only when carry in and carry out are different logic.

(a)

The following is the addition of two 8-bit numbers:

$$\begin{array}{rr} C= & 1000000 \\ x= & 11010100 \\ y= & 11101011 \\ \hline s= & 110111111 \end{array}$$

Here *x* & *y* are the two 8-bit two's complement numbers, *s* is the result of adding two 8-bit number, *C* is the carry carried over during addition.

Auxiliary and final carry are same logic (indicated in red color). So there is **no overflow occurs.**

Step 2 of 4

(b)

The following is the addition of two 8-bit numbers:

$$\begin{array}{rr} C= & 1111111 \\ x= & 10111111 \\ y= & 11011111 \\ \hline s= & 110011110 \end{array}$$

Here *x* & *y* are the two 8-bit two's complement numbers, *s* is the result of adding two 8-bit number, *C* is the carry carried over during addition.

Auxiliary and final carry are same logic (indicated in red color).. So there is **no overflow occurs.**

Step 3 of 4

(c)

The following is the addition of two 8-bit numbers:

$$\begin{array}{rr} C= & 11100010 \\ x= & 01011101 \\ y= & 00110001 \\ \hline s= & 010001110 \end{array}$$

Here *x* & *y* are the two 8-bit two's complement numbers, *s* is the result of adding two 8-bit number, *C* is the carry carried over during addition.

Auxiliary and final carry are differ logic (indicated in red color). So there is **overflow occurs.**

Step 4 of 4

(d)

The following is the addition of two 8-bit numbers:

$$\begin{array}{rr} C= & 11111110 \\ x= & 01100001 \\ y= & 00011111 \\ \hline s= & 010000000 \end{array}$$

Here *x* & *y* are the two 8-bit two's complement numbers, *s* is the result of adding two 8-bit number, *C* is the carry carried over during addition.

Auxiliary and final carry are differ logic(indicated in red color). So there is **overflow occurs.**

Step 1 of 1 **2.13DP**

If the minimum distance of a code is $2c+d+1$, then the code can be used to correct errors up to *c* bits and to detect errors in up to *d* additional bits.

Thus, the number errors that can be detected by a code with minimum distance $d+1$ is

$\boxed{d}$.

Step 1 of 2

2.14DP

The following table gives the word sizes of distance-4 extended Hamming code:

Table 1

Minimum – Distance – 4 codes

ParityBits	***Total Bits***
3	4
4	≤ 8
5	≤ 16
6	≤ 32
7	≤ 64
8	≤ 128

Step 2 of 2

From Table 1, we can arrive at the following condition to be satisfied, between parity bit *p* and the number of information bits, *n* in a two dimensional code with minimum distance-4:

$$n \leq 2^{(p-1)}$$

Derive the number of parity bits required by taking natural logarithm on both sides.

$$\ln n \leq \ln 2^{(p-1)}$$
$$\ln n \leq (p-1)\ln 2$$
$$p-1 \geq \frac{\ln n}{\ln 2}$$
$$p \geq \frac{\ln n}{\ln 2}+1$$

$$p \geq (1.443\ln n+1)$$

Thus, the minimum number of parity bits required to obtain a distance-4, two dimensional code with *n* information bits, *p* is $\boxed{1.443\ln n+1}$.

Step 1 of 1

2.15DP

Sometimes the terms Dec, Oct, Hex are placed in front of numbers to indicate their base. However some digital engineers get confusion in holiday time because octal number 31 is equal to Decimal number 25.

So the programmer used to write as Oct 31 = Dec 25, which also seems like October 31 that is, Halloween day is equal to December 25 that is Christmas day.

Step 1 of 1

Steps for finding the hexadecimal equivalent of 12648430_{10} as follows,

$$\begin{aligned}
12648430 \div 16 &= 790526 \text{ remainder E (Least significant bit)}\\
790526 \div 16 &= 49407 \text{ remainder E}\\
49407 \div 16 &= 3087 \text{ remainder F}\\
3087 \div 16 &= 192 \text{ remainder F}\\
192 \div 16 &= 12 \text{ remainder 0}\\
12 \div 16 &= 0 \text{ remainder C (Most significant bit)}
\end{aligned}$$

Thus the hexadecimal equivalent of 12648430_{10} is $\boxed{C0FFEE_{16}}$.

Step 1 of 2

Consider a 8 bit binary number 00011001_2 (In decimal value is -25) and its negative value is 11100111_2.

Steps for finding out two's complements are as follows,

$$\begin{array}{rl} +25_{10} = 00011001_2 & \\ \Downarrow & \\ 11100110_2 & \text{Complement bits} \\ 1\ + & \\ \hline 11100111_2 & \end{array}$$

Thus the two's complement of 8 bit binary number 00011001_2 is given by $\boxed{11100111_2}$ and in decimal -25. Hence an 8-bit binary number is having same negative value when interpret either a decimal or two's complement number.

Step 2 of 2

Similarly consider a 8 bit binary number 01111000_2 (In decimal value is +120) and its negative value is 10001000_2.

Steps for finding out two's complements are as follows,

$$\begin{array}{rl} +120_{10} = 01111000_2 & \\ \Downarrow & \\ 10000111_2 & \text{Complement bits} \\ 1\ + & \\ \hline 10001000_2 = -120_{10} & \end{array}$$

Thus the two's complement of 8 bit binary number 01111000_2 is given by $\boxed{10001000_2}$ and in decimal -120. Hence an 8-bit binary number is having same negative value when interpret either a decimal or two's complement number.

Step 1 of 6

The possible radices for below arithmetic operations are as follows,

(a) Arithmetic operation, $1234+5432=6666$

The greatest number used in the above equation is 6. Therefore the above arithmetic operation is possible in **radices more than 6**, example radix 7 (terms are between 0 and 6) & above.

Step 2 of 6

(b) Arithmetic operation, $\frac{41}{3}=13$

Let us take radix as *x*. In radices, a string of digits such as $d_1d_2d_3$ are denotes the decimal as $d_1\cdot x^2+d_2\cdot x+d_3$, whereas *x* is the radix of the system. Therefore the above arithmetic operation can be written as follows.

$$\frac{4x+1}{3}=x+3$$

Solve the equation as follows.

$$4x+1=3(x+3)$$
$$4x+1=3x+9$$
$$4x+1-3x-9=0$$
$$x-8=0$$
$$x=8$$

Thus the given arithmetic operation is possible in **radix 8 or octal** representation.

Step 3 of 6

(c) Arithmetic operation, $\frac{33}{3}=11$

The greatest number used in the above equation is 3. Therefore the above arithmetic operation is possible in radices more than 3, example radix 4(terms are between 0 and 3) & above.

Step 4 of 6

(d) Arithmetic operation, $23+44+14+32=223$

Let us take radix as x. In radices, a string of digits such as $d_1d_2d_3$ are denotes the decimal as $d_1\cdot x^2+d_2\cdot x+d_3$, whereas *x* is the radix of the system. Therefore the above arithmetic operation can be written as follows,

$$(2x+3)+(4x+4)+(x+4)+(3x+2)=(2x^2+2x+3)$$

Solve the equation as follows.

$$(2x+3)+(4x+4)+(x+4)+(3x+2)=(2x^2+2x+3)$$
$$10x+13=2x^2+2x+3$$
$$2x^2+2x+3-10x-13=0$$
$$2x^2-8x-10=0$$
$$x^2-4x-5=0$$
$$(x+1)(x-5)=0$$
$$x=-1\text{ or }5$$

Radix must be positive. Therefore $x=5$.

Thus the given arithmetic operation is possible in **radix 5**.

Step 5 of 6

(e) Arithmetic operation, $\frac{302}{20}=12.1$

Let us take radix as x. In radices, a string of digits such as $d_1d_2d_3$ are denotes the decimal as $d_1\cdot x^2+d_2\cdot x+d_3$, whereas *x* is the radix of the system. Therefore the given arithmetic operation can be written as follows,

$$\frac{(3x^2+2)}{2x}=x+2+\frac{1}{x}$$

Solve the equation as follows.

$$\frac{(3x^2+2)}{2x}=\frac{x^2+2x+1}{x}$$

Cancelling x in denominator on both sides,

$$(3x^2+2)=2(x^2+2x+1)$$
$$3x^2+2=2x^2+4x+2$$
$$3x^2+2-2x^2-4x-2=0$$
$$x^2-4x=0$$
$$x(x-4)=0$$
$$x=0\text{ or }4$$

Radix must be greater than 0. Therefore $x=4$.

Thus the given arithmetic operation is possible in **radix 4**.

Step 6 of 6

(f) Arithmetic operation, $\sqrt{41}=5$

Let us take radix as x. In radices, a string of digits such as $d_1d_2d_3$ are denotes the decimal as $d_1\cdot x^2+d_2\cdot x+d_3$, whereas *x* is the radix of the system. Therefore the given arithmetic operation can be written as follows,

$$\sqrt{(4x+1)}=5$$

Solving the equation as follows,

Squaring the equation on both sides,

$$(4x+1)=25$$
$$4x+1-25=0$$
$$4x-24=0$$
$$x=\frac{24}{4}$$
$$x=6$$

Thus the given arithmetic operation is possible in **radix 6**.

Step 1 of 2 **2.020E**

Most of the people believe that we use base 10 since we have 10 fingers. Assuming this let us say that Martians have *n* fingers and thus use a base *n* number system.

Consider the following equation found from the Martian mathematics:

$$5x^2-50x+125=0$$

Consider the following solutions for the equation:

$$x=5 \text{ and } x=8$$

Convert this equation into base *b* which yields the following equation:

$$5x^2-(5n)x+\left(n^2+2n+5\right)=0$$

Substitute 5 for *x* in the equation.

$$5\left(5^2\right)-(5n)5+\left(n^2+2n+5\right)=0$$
$$n^2-23n+130=0$$
$$(n-13)(n-10)=0$$
$$n=13 \text{ or } 10$$

Note that it is not possible to have $n=10$, since in base 10, the other solution $x=8$ should not be a solution. Hence $n=13$.

Step 2 of 2

Martians must have 13 fingers. Indeed this makes sense because if 50 and 125 are in base 13.

Convert them to base 10.

$$5(13)=65_{10}$$
$$13^2+2(13)+5=200_{10}$$

So, the following is the resulting equation:

$$5x^2-65x+200=0$$
$$x^2-13x+40=0$$
$$(x-5)(x-8)=0$$
$$x=5 \text{ or } 8$$

Here $x=8$ is a justified solution; therefore, the Martians had **13** fingers.

Step 1 of 2

In a radix *r* complement system, the complement of *n*-digit number *D* is obtained by subtracting it from r^n as follows:

$$-D = r^n - \sum_{i=0}^{n-1} D_i \cdot r^i \quad \text{......} \ (1)$$

According to context, in order to prove a 4*n* bit binary number *B* is represented by an *n*-digit hexadecimal number *H* and then the one's complement of *B* is represented by the 15's complement of *H*. It can be proved as shown below.

Given that 4*n* bit binary number *B* is represented by an *n*-digit hexadecimal number *H*, which can be represented as follows.

$$\begin{aligned} B &= \sum_{i=0}^{4n-1} b_i \cdot 2^i \\ &= \sum_{i=0}^{n-1} h_i \cdot 16^i \\ &= H \end{aligned}$$

The one's complement of *B* is represented by using equation (1) as follows:

$$-B = 1^{4n} - \sum_{i=0}^{4n-1} b_i \cdot 2^i$$

Step 2 of 2

Solve the above equation.

$$-B = (2-1)^{4n} - \sum_{i=0}^{4n-1} b_i \cdot 2^i$$

$$-B = 2^{4n} - 1^{4n} - \sum_{i=0}^{4n-1} b_i \cdot 2^i$$

$$-B = 16^n - 1^n - \sum_{i=0}^{4n-1} b_i \cdot 2^i$$

$$-B = (16-1)^n - \sum_{i=0}^{4n-1} b_i \cdot 2^i$$

$$-B = 15^n - \sum_{i=0}^{4n-1} b_i \cdot 2^i$$

$$-B = 15^n - \sum_{i=0}^{n-1} h_i \cdot 16^i \qquad \left(\text{Since } \sum_{i=0}^{4n-1} b_i \cdot 2^i = \sum_{i=0}^{n-1} h_i \cdot 16^i\right)$$

The above equation also gives the value of 15's complement of *H*. Thus a 4*n* bit binary number *B* is represented by an *n*-digit hexadecimal number, then one's complement of *B* is represented by the 15's complement of *H* is proved.

$$\boxed{\text{1's complement of } B = \text{15's complement of } H}$$

Step 1 of 5

2.022E

The range an integer *x* is in the range of $-2^{n-1} \le x \le \left(2^{n-1}-1\right)$.

Consider the definition of $[x]$ which is two's complement representation of an integer *x*:

$$[x]=\begin{cases} x, & \text{if } x \ge 0 \\ 2^n-|x| & \text{if } x<0 \end{cases} \quad \text{...... (1)}$$

Here, $|x|$ is the absolute value of *x*.

Similarly, definition of $[y]$ which is two's complement representation of an integer *y*:

$$[y]=\begin{cases} y, & \text{if } y \ge 0 \\ 2^n-|y| & \text{if } y<0 \end{cases} \quad \text{...... (2)}$$

Here, $|y|$ is the absolute value of *y*.

Step 2 of 5

Case I:

Consider both *x* and *y* are negative integers and their maximum values are -2^{n-1}.

For the two negative integer *x* and *y*, its two`s complement representation from the definition of $[x]$ in the equation (1) and $[y]$ in the equation (2),

$$[x]=2^n-|x| \quad \text{...... (3)}$$

$$[y]=2^n-|y| \quad \text{...... (4)}$$

Summation of the equations (3) and (4)

$$\begin{aligned}[x]+[y]&=\left(2^n-|x|\right)+\left(2^n-|y|\right)\\ &=2^n+2^n-\left(|x|+|y|\right)\end{aligned}$$

Take modulus operation to the base 2^n on both sides of the equation

$$[x]+[y]\text{mod } 2^n=\left(2^n+2^n-\left(|x|+|y|\right)\right)\text{mod } 2^n \quad \text{...... (5)}$$

Modulus operation returns the remainder of the division operation.

$$\begin{aligned}[x]+[y]\text{modulo } 2^n&=\left(2^n\right)\text{modulo } 2^n+\left(2^n-\left(|x|+|y|\right)\right)\text{modulo } 2^n\\ &=0+0-\left(|x|+|y|\right)\text{modulo } 2^n\end{aligned}$$

$$[x]+[y]\text{modulo } 2^n=2^n-\left(|x|+|y|\right) \quad \text{...... (6)}$$

By means of the equation (3) expressing the negative number in two`s complement form write the equation (6) as follows,

$$[x]+[y]\text{modulo } 2^n=[x+y] \quad \text{...... (7)}$$

Thus, the two complement's additional rule when adding two negative integers is proved.

Step 3 of 5

Case II:

Consider *x* as negative integer with maximum range value -2^{n-1} and y as positive integer with the maximum range value $2^{n-1}-1$.

Write the two`s complement representation for the negative integer *x* and positive integer *y*, from the definition of $[x]$ in the equation (1) and $[y]$ in the equation (2),

$$[x]=2^n-|x| \quad \text{...... (8)}$$

$$[y]=y=|y| \quad \text{...... (9)}$$

Summation of the equation (8) and (9):

$$[x]+[y]=2^n-|x|+|y| \quad \text{...... (10)}$$

Take modulus operation to the base 2^n on both sides of the equation

$$[x]+[y]\text{modulo } 2^n=\left(2^n-|x|+|y|\right)\text{modulo } 2^n$$

Modulus operation returns the remainder of the division operation.

Modulus operation is distributive over addition as same as division operation. As well as consider the absolute value of x and y otherwise $|x|>|y|$.

$$\begin{aligned}[x]+[y]\text{modulo } 2^n&=\left(2^n-\left(|x|-|y|\right)\right)\text{modulo } 2^n\\ &=2^n\text{modulo } 2^n-\left(|x|-|y|\right)\text{modulo } 2^n\\ &=0+2^n-\left(|x|-|y|\right)\end{aligned}$$

$$[x]+[y]\text{modulo } 2^n=2^n-\left(|x|-|y|\right) \quad \text{...... (11)}$$

Rewrite the equation (11) using the equation (9), express the negative number in two`s complement form as follows,

$$[x]+[y]\text{modulo}2^n=[x+y]$$

Thus, the two complement's additional rule when adding negative integer and positive integer is proved.

Step 4 of 5

Case III:

Consider positive integer *x* and negative integer *y*, its two`s complement representation from equation (1) and (2).

$$[x]=x=|x| \quad \text{..... (12)}$$

$$[y]=2^n-|y| \quad \text{..... (13)}$$

Summation of the equation (2) and (3):

$$[x]+[y]=2^n-|y|+|x|$$

Take modulus operation to the base 2^n on both sides of the equation

$$[x]+[y]\text{modulo } 2^n=\left(2^n-|y|+|x|\right)\text{modulo } 2^n$$

$$[x]+[y]\text{modulo } 2^n=\left(2^n+|x|-|y|\right)\text{modulo } 2^n \quad \text{..... (14)}$$

Modulus operation returns the remainder of the division operation.

$|x|>|y|$ and hence the two`s complement of $\left(|x|-|y|\right)$ is greater than 0. Hence $2^n+\left(|x|-|y|\right)$ is greater than 2^n. Since the modulus operation is distributive over addition, rewrite the equation (14) as follows

$$\begin{aligned}[x]+[y]\text{modulo } 2^n&=\left(2^n+\left(|x|-|y|\right)\right)\text{modulo } 2^n\\ &=0+\left(|x|-|y|\right)\end{aligned}$$

$$[x]+[y]\text{modulo } 2^n=\left(|x|-|y|\right) \quad \text{..... (15)}$$

In equation (15), consider the absolute value considering the variables along with the sign we can write the equation as follows,

$$[x+y]=[x]+[y]\text{modulo } 2^n$$

Thus, the two complement's additional rule when adding positive integer and negative integer is proved.

Step 5 of 5

Case IV:

Let us consider both *x* and *y* are positive integers and their maximum values are $2^{n-1}-1$.

From the definition of $[x]$ in the equation (1) and $[y]$ in the equation (2),

$$[x]=x \quad \text{...... (16)}$$

$$[y]=y \quad \text{...... (17)}$$

Summation of the equations (16) and (17):

$$[x]+[y]=x+y$$

Take modulus operation to the base 2^n on both sides.

$$[x]+[y]\text{modulo } 2^n=x+y\text{ modulo } 2^n \quad \text{...... (18)}$$

Modulus operation returns the remainder of the division operation.

The Right hand side of equation (18), $x+y$ is always less than 2^n because x and y only take the values less than 2^{n-1}. In two`s complement addition carry is discarded. When the number of bits of the addition result is $n+1$ then leftmost first bit is discarded from the result.

$$[x]+[y]\text{modulo } 2^n=[x+y]$$

$$[x+y]=[x]+[y]\text{modulo } 2^n$$

Example for this proof:

Let us consider 4-bit positive numbers. Their maximum values of positive decimal numbers are 7. So the resultant of these positive integers is 14; which is also a 4-bit.

Hence, the statement is true for the four possible cases of addition operation over two`s complement of signed numbers.

Step 1 of 5

The range of integer *x* is in the range of $-2^{n-1} \le x \le \left(2^{n-1}-1\right)$.

It is assumed that *x* is greater than *y*.

In case of negative integer *x*, [x] is 2^n-1-|x|. One's complement addition is illustrated as follows:

$$[x+y]\text{mod}\,2^n = \begin{cases} [x]+[y] & [x]+[y]<2^n \\ [x]+[y]\text{mod}\,2^n+1 & [x]+[y]>2^n \end{cases} \quad \text{...... (1)}$$

Case I:

Consider both *x* and *y* as negative integers.

The one's complement representation of the negative integers *x* and *y* is as follows:

$$[x]=2^n-1-|x| \quad \text{...... (2)}$$

$$[y]=2^n-1-|y| \quad \text{...... (3)}$$

Summate equation (2) and (3).

$$\begin{aligned}[x]+[y]&=2^n-1-|x|+2^n-1-|y| \\ [x]+[y]&=2^n+2^n-2-(|x|+|y|)\end{aligned} \quad \text{...... (4)}$$

Since one's complement addition is to be obtained, add 1 to equation (4) on both sides.

$$\begin{aligned}[x]+[y]\text{mod}\,2^n &= 2^n+2^n-2-(|x|+|y|)\text{mod}\,2^n \\ [x]+[y]\text{mod}\,2^n &= 2^n-2-(|x|+|y|) \\ [x]+[y]\text{mod}\,2^n+1 &= 2^n-2-(|x|+|y|)+1 \\ [x]+[y]\text{mod}\,2^n+1 &= 2^n-1-(|x|+|y|)\end{aligned}$$

Thus, one's complement representation of $|x+y|$ is $2^n-1-(|x|+|y|)$.

Step 2 of 5

Case II:

Consider both *x* and *y* as positive integers.

For the positive integers, its absolute value is same as positive value.

Then,

$$[x]=x \quad \text{...... (5)}$$

$$[y]=y \quad \text{...... (6)}$$

Summate equations (5) and (6).

$$[x]+[y]=x+y \quad \text{...... (7)}$$

Modulus operation returns the remainder of the division operation.

Take modulus operation to the base 2^n on both sides of equation (7).

$$[x]+[y]\text{mod}\,2^n = x+y \;\;\text{mod}\,2^n \quad \text{...... (8)}$$

Right hand side of the above equation is less than 2^n, hence,

$$[x]+[y]\text{mod}\,2^n = x+y$$

Right hand side of the above equation is the one`s complement representation of $x+y$. Hence,

$$[x]+[y]\text{mod}\,2^n = [x+y]$$

Step 3 of 5

Case III:

Consider *x* as negative integer and *y* as positive integer.

One's complement representation of *x* and *y* is as follows:

$$[x]=2^n-1-|x| \quad \text{...... (9)}$$

$$[y]=y \quad \text{...... (10)}$$

Summate equation (9) and (10).

$$[x]+[y]=2^n-1-|x|+|y| \quad \text{...... (11)}$$

Since |x| is greater than |y|, -|x|+|y| become negative, hence right hand side of the equation (11) is less than 2^{n-1}.

Modulus operation returns the remainder of the division operation. Take modulus operation to the base 2^n on both sides of equation (11).

$$[x]+[y]\text{mod}\;2^n = \left(2^n-1-|x|+|y|\right)\text{mod}\;2^n \quad \text{...... (12)}$$

Modulus operation is distributive over addition and same as division operation, considering the absolute values of *x* and *y* equation (12) is modified as follows:

$$[x]+[y]\text{mod}\;2^n = 2^n-1-(|x|-|y|) \quad \text{...... (13)}$$

The one's complement form representation of the equation (13) is,

$$[x]+[y]\text{mod}\;2^n = [x+y]$$

Step 4 of 5

Case IV:

Consider the final case, positive integer x and negative integer y, its one`s complement representation is as follows:

$$[x]=x \quad \text{...... (14)}$$

$$[y]=2^n-1-|y| \quad \text{...... (15)}$$

Step 5 of 5

Summate equation (14) and (15).

$$[x]+[y]=2^n-1-|y|+|x| \quad \text{...... (16)}$$

Since |x| is greater than |y| and hence |x|-|y| is positive and the right hand side of the equation (16) is greater than 2^n.

Hence take modulus operation and add 1 to the remainder.

Modulus operation returns the remainder of the division operation. Take modulus operation to the base 2^n on both sides of equation (16) and add 1 on both sides.

$$[x]+[y]\text{mod}\;2^n = \left(2^n-1-|y|+|x|\right)\text{mod}\;2^n$$

$$\begin{aligned}[x]+[y]\text{mod}\;2^n &= -1-(|y|-|x|) \\ &= -1-(|y|-|x|)+1 \\ &= -(|y|-|x|) \\ &= (|x|-|y|)\end{aligned}$$

$$[x]+[y]\text{mod}\;2^n = (|x|-|y|) \quad \text{...... (17)}$$

By means of one`s Complement representation, equation (17) is modified as follows:

$$[x]+[y]\text{mod}\;2^n = [x]+[y]$$

Hence proved that in all the cases of *x* and *y*, the equation, $[x]+[y]\text{mod}\;2^n = [x]+[y]$ is always true.

Step 1 of 2

Below diagram shows circular modular representation. Addition & subtraction of numbers using circular modular representation is very easy. For example when the arrow is pointing to +2 and addition of +4 to that number can be done by moving the arrow up to 4 positions clockwise. Similarly for subtraction from any number can be done by moving the arrow in anticlockwise according to the number. The circular modular representation gives the correct result only if the numbers are small.

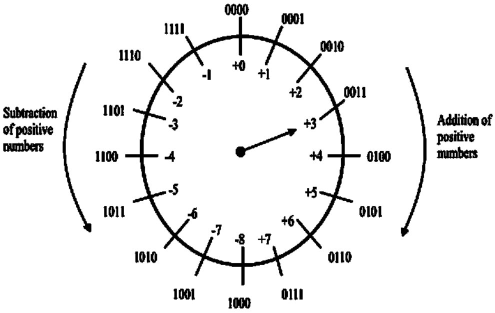


Figure 1: Circular modular representation diagram

Step 2 of 2

An overflow rule for addition of two's complement numbers in terms of circular modular operations are as follows,

(i) If the arrow is pointing in +6 and need to add +2 causes overflow, since arrow cannot advance from +8, if so it will go to negative numbers gives the wrong result or we can say overflow occurs. Thus if the arrow is pointing to any number and adding a positive number causes overflow if the arrow is advanced through the +7 to -8 transition.

(ii) If the arrow is in -6 and need to add -3(that is subtracting 3 from -6) with that causes overflow. Thus if the arrow is pointing to any number and adding a negative number (subtraction of positive number) causes overflow if the arrow is advance through the -8 to +7 transition.

Step 1 of 4

In two`s complement number representation Most Significant Bit (MSB) is sign bit. Hence the MSB represent the sign of the bit and its positional value is $-\left(2^{(n-1)}\right)$

Consider an *n*-bit two`s complement number *X* by standard formula,

$$D_{10n} = -2^{n-1} \cdot X_{n-1} + \sum_{i=0}^{N-2} 2^i X_i$$

$$D_{10n} = -2^{n-1} \cdot X_{n-1} + 2^{n-2} \cdot X_{n-2} + \dots\dots + 2^1 \cdot X_1 + 2^0 \cdot X_0 \text{ ...... (1)}$$

Here, *D* is an *n*-bit integer in the form of $X_{n-1}X_{n-2}\dots\dots X_0$.

Extend the *n*-bit representation to *m*-bit representation where $m > n$ by appending $m-n$ leftmost bits which are same as sign bits.

$$D_{10m} = \begin{pmatrix} -2^{m-1} \cdot X_{m-1} + 2^{m-2} \cdot X_{m-2} + \dots + 2^n \cdot X_n + 2^{n-1} \cdot X_{n-1} + \dots + \\ 2 \cdot X_1 + 2^0 \cdot X_0 \end{pmatrix} \text{ ...... (2)}$$

Step 2 of 4

Here, $X_{m-1}, X_{m-2}, X_{m-3} \dots X_{n-2}, X_{n-1}$ bits values are same. So, $X_{m-1} = X_{m-2} = \dots = X_{n-2} = X_{n-1} = X_n$ ...... (3)

Step 3 of 4

Step 1:

Subtract the equation (2) from the equation (1).

$$\begin{aligned} D_{10m} - D_{10n} &= \begin{bmatrix} \begin{pmatrix} -2^{m-1} \cdot X_{m-1} + 2^{m-2} \cdot X_{m-2} + \dots + 2^n \cdot X_n + \\ 2^{n-1} \cdot X_{n-1} + 2^{n-2} \cdot X_{n-2} + \dots + 2 \cdot X_1 + 2^0 \cdot X_0 \end{pmatrix} - \\ \left(-2^{n-1} \cdot X_{n-1} + 2^{n-2} \cdot X_{n-2} + \dots\dots + 2^1 \cdot X_1 + 2^0 \cdot X_0\right) \end{bmatrix} \\ &= \begin{pmatrix} -2^{m-1} \cdot X_{m-1} + 2^{m-2} \cdot X_{m-2} + \dots + 2^{n+1} \cdot X_{n+1} + 2^n \cdot X_n + \\ 2^{n-1} \cdot X_{n-1} + 2^{n-1} \cdot X_{n-1} \end{pmatrix} \\ &= \begin{pmatrix} -2^{m-1} \cdot X_{m-1} + 2^{m-2} \cdot X_{m-2} + \dots + 2^{n+1} \cdot X_{n+1} + \\ 2^n \cdot X_n + 2.2^{n-1} \cdot X_{n-1} \end{pmatrix} \\ &= \begin{pmatrix} -2^{m-1} \cdot X_{m-1} + 2^{m-2} \cdot X_{m-2} + \dots + 2^{n+1} \cdot X_{n+1} + \\ 2^n \cdot X_n + 2^n \cdot X_{n-1} \end{pmatrix} (\because X_n = X_{n-1}) \\ &= -2^{m-1} \cdot X_{m-1} + 2^{m-2} \cdot X_{m-2} + \dots + 2^{n+1} \cdot X_{n+1} + 2 \cdot 2^n \cdot X_n \end{aligned}$$

Step 4 of 4

Continue simplifying the last expression in step1 using the relationship from equation (3), by adding right most two adjacent terms up to $m-n$ terms to get the following expression:

$$\begin{aligned} D_{10m} - D_{10n} &= -2^{m-1} \cdot X_{m-1} + 2^{m-2} \cdot 2 \cdot X_{m-2} \\ &= -2^{m-1} \cdot X_{m-1} + 2^{m-1} \cdot X_{m-2} \\ &= 2^{m-1}\left(-X_{m-1} + X_{m-1}\right) \\ &= 2^{m-1}(0) \end{aligned}$$

$$= 0$$

Therefore, the two different representations of a number using m-bits and n-bits are equal where $m > n$, m bits are obtained by appending $m-n$ sign bits at leftmost end.

Step 1 of 3

In two`s complement number representation Most Significant Bit (MSB) is sign bit. Hence the MSB represent the sign of the bit and its positional value is $-\left(2^{(n-1)}\right)$

Consider an *n*-bit two`s complement number *X* by standard formula,

$$X=-2^{n-1}\cdot X_{n-1}+\sum_{i=0}^{N-2}2^i.X_i$$

$$X=\begin{pmatrix}-2^{n-1}\cdot X_{n-1}+2^{n-2}\cdot X_{n-2}+......+2^m\cdot X_m+2^{m-1}\cdot X_{m-1}+......+\\ 2^1\cdot X_1+2^0\cdot X_0\end{pmatrix} \text{ ...... (1)}$$

Remove the leftmost consecutive *d* bits, which are same as sign bits of *Y*, from the *n*-bit representation then the representation has *m* bits.

$$Y=-2^{m-1}\cdot X_{m-1}+2^{m-2}\cdot X_{m-2}+....+2\cdot X_1+2^0\cdot X_0 \text{ ...... (2)}$$

Here, $m=n-d$ and $X_{n-1},X_{n-2},X_{n-3}......X_m,X_{m-1}$ bits have same value and equal to signed bit value of *Y*. So,

$$X_{n-1}=X_{n-2}=.....=X_{m-1}=X_m \text{ ...... (3)}$$

Step 2 of 3

Step 1:

Subtract the equation (2) from the equation (1).

$$\begin{aligned}X-Y&=\begin{pmatrix}-2^{n-1}\cdot X_{n-1}+2^{n-2}\cdot X_{n-2}+.....+2^m\cdot X_m+\\ 2^{m-1}\cdot X_{m-1}+2^{m-1}\cdot X_{m-1}\end{pmatrix}\\ &=\begin{pmatrix}-2^{n-1}\cdot X_{n-1}+2^{n-2}\cdot X_{n-2}+.....+2^{m+1}\cdot X_{m+1}+\\ 2^m\cdot X_m+2.2^{m-1}\cdot X_{m-1}\end{pmatrix}\\ &=\begin{pmatrix}-2^{n-1}\cdot X_{n-1}+2^{n-2}\cdot X_{n-2}+.....+2^{m+1}\cdot X_{m+1}+\\ 2^m\cdot X_m+2^m\cdot X_{m-1}\end{pmatrix} \quad (\because X_n=X_{n-1})\\ &=-2^{n-1}\cdot X_{n-1}+2^{n-2}\cdot X_{n-2}+.....+2^{m+1}\cdot X_{m+1}+2\cdot 2^m\cdot X_m\end{aligned}$$

Step 3 of 3

Continue simplifying the last expression in step1 using the relationship from equation (3), by adding right most two adjacent terms up to $n-m$ terms to get the following expression:

$$\begin{aligned}X-Y&=-2^{n-1}\cdot X_{n-1}+2^{n-2}\cdot X_{n-2}+.....+2^{m+1}\cdot X_{m+1}+2\cdot 2^m\cdot X_m\\ &=-2^{n-1}\cdot X_{n-1}+2^{n-2}\cdot 2\cdot X_{n-2}\\ &=-2^{n-1}\cdot X_{n-1}+2^{n-1}\cdot X_{n-2}\\ &=2^{n-1}\left(-X_{n-1}+X_{n-2}\right)\end{aligned}$$

Simplify the equation further by substituting $X_{n-1}=X_{n-2}$ from equation (3).

$$X-Y=2^{n-1}\left(-X_{n-1}+X_{n-1}\right)$$
$$X-Y=2^{n-1}(0)$$
$$X-Y=0$$

$$X=Y \text{ ...... (4)}$$

Hence, from the equation (3) and (4), *showed that the m-bit two complement number Y obtained by discarding the d leftmost bits of X represents the same number as X if and only if the discarded bits all equal to the sign bit of Y.*

Hence the representations using *m*-bits and *n*-bits are equal, where $n>m$ if and only if the discarded leftmost *d* bits are equal to sign bits. Otherwise equation (3) is not satisfied. Therefore it cannot possible to simplify the equation $X-Y$. Hence it cannot say value of *X* is equal to *Y*.

Hence, two's complement number converted to a representation with fewer bits by removing high-order bits if and only if the discarded bits all equal to the sign bit.

Step 1 of 1

The inconsistency of the punctuation of “two’s complement” and “ones’ complement” is due to their significance in digital operations.

(a) Both of these complements are used to perform digital subtraction.

(b) One’s complement has a special operational significance and usefulness as well. It can detect unidirectional error which is useful in communication channel like internet. Unidirectional error means all the erroneous bits change in the same direction from low to high or vice versa. Ones’ – complement checksum codes are used in internet to check for the errors that occur during the transmission of the information.

The advantages and usage of Ones’ complement is different from the two’s complement.

The procedure to perform digital subtraction using ones’ complement is different from the two’s complement.

Hence, to signify the difference in its operation, there exists an inconsistency in the punctuation of two’s complement from ones’ complement.

Step 1 of 2 **2.028E**

The *n*-bit binary adder can be used to perform *n*-bit unsigned subtraction operation $X - Y$ by performing the operation $X - \overline{Y} + 1$.

Here, $\overline{Y}$ is the bit by bit complement of *Y*.

Binary Subtraction operation performed by adding the two`s complement of subtrahend to the Minuend. Two`s complement of the subtrahend is found as follows $\overline{Y} + 1$

Here $\overline{Y}$ is the bit by bit complement of *Y*.

In two` s complement representation we are discarding the carry 1 if it occurs. We can say that we are discarding the bit in the $(n+1)^{th}$ position. Hence,

$$X - Y = \left(X + \overline{Y} + 1\right) \textbf{modulo } 2^n$$

Its positional value is 2^n since we are starting the first bit from the 0^{th} Position. Hence instead of discarding the carry 1 in the $(n+1)^{th}$ position we can subtract 2^n from the equation. Otherwise multiplicative inverse of subtraction is division as,

$$X - Y = \left(X + \overline{Y} + 1\right) - \left(1 \cdot 2^n + 0 \cdot 2^{n-1} + \ldots\ldots + 0 \cdot 2 + 0\right) \text{ ...... (1)}$$

$$X - Y = \left(X + \overline{Y} + 1\right) - 2^n \text{ ...... (2)}$$

Hence, the expression for unsigned subtraction operation is proved,

$$\boxed{X - Y = \left(X + \overline{Y} + 1\right) - 2^n}.$$

Step 2 of 2

Carry out at the end of the operation $X + \overline{Y} + 1$ is called carry. Carry out at the end of the operation $\left(X + \overline{Y} + 1\right) - 2^n$ is called borrow.

In the equation (2), if $X + \overline{Y} + 1$ produces carry 1 in the $(n+1)^{th}$ position, after completing the operation in equation (2) borrow as zero.

If $X + \overline{Y} + 1$ produces carry 0 in the $(n+1)^{th}$ position, after completing the operation in equation (2) get borrow as one.

Subtraction in the right hand side of the equation (1) replaced by two`s complement addition.

$$\begin{aligned} X - Y &= \left(X + \overline{Y} + 1\right) + \overline{\left(1 \cdot 2^n + 0 \cdot 2^{n-1} + \ldots\ldots + 0 \cdot 2 + 0\right)} + 1 \\ &= \left(X + \overline{Y} + 1\right) + \left(0 \cdot 2^n + 1 \cdot 2^{n-1} + \ldots\ldots + 1 \cdot 2 + 1\right) + 1 \\ &= \left(X + \overline{Y} + 1\right) + \left(1 \cdot 2^n + 0 \cdot 2^{n-1} + \ldots\ldots + 0 \cdot 2 + 0\right) \end{aligned}$$

In equation (3), if $X + \overline{Y} + 1$ produces carry 1 in the position in the $(n+1)^{th}$ position, borrow will be 0. Else it will be 1. Hence the carry and borrow of the operation is complementary.

Step 1 of 5

In a *n*-bit two`s complement representation, the range of integer values is,

$$-2^{n-1} \le X \le 2^{n-1}-1$$

Two`s complement representation of integer *X*,

$$X = -2^{n-1}\cdot X_{n-1} + 2^{n-2}\cdot X_{n-2} + \ldots + 2\cdot X_1 + 2^0\cdot X_0 \text{ ...... (1)}$$

Two`s complement representation of integer *Y*,

$$Y = -2^{n-1}\cdot Y_{n-1} + 2^{n-2}\cdot Y_{n-2} + \ldots + 2\cdot Y_1 + 2^0\cdot Y_0 \text{ ...... (2)}$$

Step 2 of 5

Multiply two`s complement number X with Y.

$$X\cdot Y = -2^{n-1}\cdot Y_{n-1}\cdot X + 2^{n-2}\cdot Y_{n-2}\cdot X + \ldots + 2\cdot Y_1\cdot X + 2^0\cdot Y_0\cdot X \text{ ...... (3)}$$

Substitute $-2^{n-1}\cdot X_{n-1} + 2^{n-2}\cdot X_{n-2} + \ldots + 2\cdot X_1 + 2^0\cdot X_0$ for *X*.

$$XY = \begin{bmatrix} -2^{n-1}\cdot\left(-2^{n-1}\cdot X_{n-1} + 2^{n-2}\cdot X_{n-2} + \ldots + 2\cdot X_1 + 2^0\cdot X_0\right)\cdot Y_{n-1} + \\ 2^{n-2}\cdot\left(-2^{n-1}\cdot X_{n-1} + 2^{n-2}\cdot X_{n-2} + \ldots + 2\cdot X_1 + 2^0\cdot X_0\right)\cdot Y_{n-2} + \\ \ldots + 2\cdot Y_1\cdot\left(-2^{n-1}\cdot X_{n-1} + 2^{n-2}\cdot X_{n-2} + \ldots + 2\cdot X_1 + 2^0\cdot X_0\right) + \\ 2^0\cdot Y_0\cdot\left(-2^{n-1}\cdot X_{n-1} + 2^{n-2}\cdot X_{n-2} + \ldots + 2\cdot X_1 + 2^0\cdot X_0\right) \end{bmatrix}$$

Simplify the equation further.

$$XY = \begin{bmatrix} 2^{2n-2}X_{n-1}Y_{n-1} - 2^{2n-3}\left(X_{n-2}Y_{n-1} + X_{n-1}Y_{n-2}\right) + \\ 2^{2n-4}\left(-X_{n-3}Y_{n-1} + X_{n-1}Y_{n-3} + X_{n-2}Y_{n-2}\right) + \\ \ldots + 2^1\left(X_1Y_0 + X_0Y_1\right) + 2^0 X_0Y_0 \end{bmatrix}$$

Step 3 of 5

Case I:

Consider both *X* and *Y* are positive and their maximum values are $2^{n-1}-1$,

$$X = Y = 2^{n-1}-1.$$

$$\begin{aligned} X\cdot Y &= \left(2^{n-1}-1\right)\cdot\left(2^{n-1}-1\right) \\ &= \left(2^{n-1}-1\right)^2 \\ &= 2^{2n-2} - 2\cdot 2^{n-1} + 1 \\ &= 2^{2n-2} - 2^n + 1 \end{aligned}$$

$$X\cdot Y = 2^{2n-2} - \left(2^n - 1\right) \text{ ...... (4)}$$

From equation (4), it is evident that product $X\cdot Y$ can be represented with less than $2n$ number of bits. Since the positive result is less than $2^{2n-2}-1$.

Step 4 of 5

Case II:

Consider *X* as positive integer with maximum value $2^{n-1}-1$ and *Y* as negative integer with the maximum value -2^{n-1} .

$$\begin{aligned} X\cdot Y &= \left(2^{n-1}-1\right)\cdot\left(-2^{n-1}\right) \\ &= \left(-2^{2n-2}+2^{n-1}\right) \\ &= -\left(2^{2n-2}-2^{n-1}\right) \end{aligned}$$

$$X\cdot Y = -\left(2^{2n-2}-2^{n-1}\right) \text{ ...... (5)}$$

Case III:

Consider *X* as negative integer with maximum value -2^{n-1} and *Y* as positive integer with the maximum value $2^{n-1}-1$.

$$\begin{aligned} X\cdot Y &= \left(2^{n-1}-1\right)\cdot\left(-2^{n-1}\right) \\ &= \left(-2^{2n-2}+2^{n-1}\right) \\ &= -\left(2^{2n-2}-2^{n-1}\right) \end{aligned}$$

$$X\cdot Y = -\left(2^{2n-2}-2^{n-1}\right) \text{ ...... (6)}$$

From equation (5) and (6), the product $X\cdot Y$ can be represented with less than $2n$ number of bits. Since the negative value $\left|-\left(2^{2n-2}-2^{n-1}\right)\right|$ is less than $\left|-2^{2n-2}\right|$.

Step 5 of 5

Case IV:

Consider both *X* and *Y* are negative integers and their maximum values are -2^{n-1} .

$$\begin{aligned} X\cdot Y &= \left(-2^{n-1}\right)\cdot\left(-2^{n-1}\right) \\ &= 2^{2n-2} \end{aligned}$$

The product $X\cdot Y$ cannot be represented with less than $2n$ number of bits. Since the value $\left|\left(2^{2n-2}\right)\right|$ is greater than $\left|2^{2n-2}-1\right|$. But it can be represented with $2n$ number of bits in two`s complement representation.

*Hence, in case IV that is **both X and Y are negative integers** and their maximum value is equal to* -2^{n-1} *needs 2n bits for its representation.*

Step 1 of 2

Statement: A one's complement number can be multiplied by 2 is same as the left shifting the one`s complement representation and transferring the Previous Most significant bit to current least significant bit.

Proof for statement:

Write the general representation of *n*-bit one`s complement number.

$$X = -\left(2^{n-1} - 1\right) X_{n-1} + 2^{n-2} X_{n-2} + \ldots + 2^1 X_1 + 2^0 X_0 \quad \ldots\ldots (1)$$

Multiply one`s complement number by 2.

$$X \cdot 2 = \left(-\left(2^{n-1} - 1\right) X_{n-1} + 2^{n-2} X_{n-2} + \ldots + 2^1 X_1 + 2^0 X_0 \right) \cdot 2 \quad \ldots\ldots (2)$$

Multiplicative operation is distributive over addition. So,

$$X \cdot 2 = \left(-\left(2^{n-1} - 1\right) \cdot 2X_{n-1} + 2^{n-2} \cdot 2X_{n-2} + \ldots + 2^1 \cdot 2X_1 + 2^0 \cdot 2X_0 \right)$$

$$X \cdot 2 = \left(-\left(2^{n} - 2\right) X_{n-1} + 2^{n-1} \cdot X_{n-2} + \ldots + 2^2 \cdot X_1 + 2^1 \cdot X_0 \right) \quad \ldots\ldots (3)$$

From the equation (3), multiplication by 2 alone doubles the positional value of each bit. It indicates the left shifting of bits by one position and also Least Significant bit position is vacant.

Step 2 of 2

Assume that there is no overflow then for a *n*-bit one`s complement number, bit present in the $(n-1)^{th}$ position is most significant bit and it has the positional value -2^{n-1}.

Hence rewrite the equation (3) by relocating the bit in the $(n+1)^{th}$ position to 0th position as Least Significant bit.

$$X \cdot 2 = \left(-2^{n-1} \cdot X_{n-2} + \ldots + 2^2 \cdot X_1 + 2^1 \cdot X_0 + 2^0 \cdot X_{n-1} \right) \quad \ldots\ldots (4)$$

In One`s Complement Numbers, multiplication by 2 shifts the bits by one position left side and append previous MSB (currently in the $(n+1)^{th}$ position) to the least significant bit position. In vice versa, the left shifting the one`s complement representation and transferring the previous Most significant bit to current least significant bit is equivalent to multiplying the one`s complement number by 2.

Therefore, the statement is proved.

Maximum value represented by n bit one`s complement number is $\pm\left(2^{(n-1)} - 1\right)$. When it is multiplied by 2, maximum value obtain is $\pm 2^n - 2$, it is represented by *n*-bit. Hence the overflow does not occur.

Step 1 of 3 **2.032E**

The following is the procedure that must be followed while doing subtract Binary Coded Decimal (BCD) numbers:

1. Determine the ten`s complement of the subtrahend and then add it to the corresponding minuend.

2. If the resultant is less than 9, take ten`s complement of the result and append negative sign to the final result.

3. If the carry occurs then just discard it.

4. Else check whether the result is greater than 9, if it is so add binary equivalent of +6 to the result, discard the carry if it occurs in this addition.

Now apply these rules to the following problems.

Determination of subtraction $8-3$ **:**

Binary Coded Decimal equivalent of 8 is 1000

Ten`s complement of 3 is $10-3=7$ and Binary Coded Decimal equivalent of 7 is 0111

Add the Binary coded decimal of 8 and ten`s complement of 3.

$$\begin{array}{r} 1000 \\ 0111 \\ \hline 1111 \end{array}$$

Result has no carry but it exceeds the value 9.

So, add binary coded decimal equivalent of 6.

$$\begin{array}{r} 1111 \\ 0110 \\ \hline \underline{1}|0101 \\ 0101 \rightarrow 5 \end{array}$$

Carry occurs in the addition and hence discard it. Then the result is 0101 , which is

Binary coded decimal equivalent of the decimal number 5 .

Therefore, the result for the arithmetic operation $8-3$ is $\boxed{5}$.

Step 2 of 3

Determination of subtraction, $4-8$ **:**

Binary Coded Decimal equivalent of 4 is 0100

Ten`s complement of 8 is $10-8=2$ and Binary Coded Decimal equivalent of 2 is 0010

Add the Binary coded decimal of 4 and ten`s complement of 8.

$$\begin{array}{r} 0100 \\ 0010 \\ \hline 0110 \end{array}$$

Result has no carry but it is less than 9, so find the ten`s complement of 0110 by subtracting the binary number 1010 from 0110 .

$$\begin{array}{r} 1010 \\ 0110 \\ \hline 0100 \\ 0100 \rightarrow 4 \end{array}$$

Therefore, the result for the arithmetic operation $4-8$ is $\boxed{-4}$.

Step 3 of 3

Determination of subtraction, $5-9$ **:**

Binary Coded Decimal equivalent of 5 is 0101

Ten`s complement of 9 is $10-9=1$ and Binary Coded Decimal equivalent of 1 is 0001 .

Add the binary numbers, the Binary coded decimal of 5 and ten`s complement of 9.

$$\begin{array}{r} 0101 \\ 0001 \\ \hline 0110 \end{array}$$

Result has no carry but it is less than 9, so find the ten`s complement of 0110 .

$$\begin{array}{r} 1010 \\ 0110 \\ \hline 0100 \\ 0100 \rightarrow 4 \end{array}$$

Taking ten`s complement denotes the negative sign of the result just like one`s complement and two`s complement representation of binary numbers. According to the fourth statement append negative sign.

Therefore, the result for the arithmetic operation $5-9$ is $\boxed{-4}$.

Determination of subtraction, $2-7$ **:**

Binary Coded Decimal equivalent of 2 is 0010

Ten`s complement of 7 is $10-7=3$ and Binary Coded Decimal equivalent of 3 is 0011

Add the binary numbers, the Binary coded decimal of 2 and ten`s complement of 7.

$$\begin{array}{r} 0010 \\ 0011 \\ \hline 0101 \end{array}$$

Result has no carry but it is less than 9 and so by second rule find the ten`s complement of 0101 by subtracting 1010 from 0101 .

$$\begin{array}{r} 1010 \\ 0101 \\ \hline 0101 \\ 0101 \rightarrow 5 \end{array}$$

According to the second statement append negative sign to the result, -5

Therefore, the result for the arithmetic operation $2-7$ is $\boxed{-5}$.

Step 1 of 4 **2.033E**

Usually for a system with *n* bits encode 2^n states for it. Hence by means of three bits the total number states to be encode is $2^3(8)$.

If the system needs only five states it may be done by more than one possible way of state encoding. It can be found using Combinational formula,

$${}^{n}C_{r}=\frac{n!}{r!(n-r)!} \quad \text{......} (1)$$

Here,

n is the total number of states that can be encoded with three bits

r is the required number of states for the system

For three bit, the number of possible encode states are eight. So, $n=8$. Out of eight we need only five states and hence $r=5$.

Substitute the 5 for *r* and 8 for *n* in equation (1).

$$\begin{aligned}{}^{8}C_{5}&=\frac{8!}{5!(8-5)!}\\&=\frac{8!}{5!(3)!}\\&=\frac{40320}{120(6)}\\&=56\end{aligned}$$

Hence, $\boxed{56}$ different ways of assigning 3-bit state encodings are possible in a controller with 5 states.

Step 2 of 4

If the system needs only seven states it can be done by more than one possible way of state encoding.

For three bit, the number of possible encode states are eight. So, $n=8$. Out of eight we need only seven states and hence $r=7$.

Substitute the 7 for *r* and 8 for *n* in equation (1).

$$\begin{aligned}{}^{8}C_{7}&=\frac{8!}{7!(8-7)!}\\&=\frac{8(7!)}{7!}\\&=8\end{aligned}$$

Step 3 of 4

Therefore, $\boxed{8}$ different ways of encoding 7 states out of available 8 states are possible.

Step 4 of 4

If the system needs all the eight states only one possible way of state encoding.

Substitute the 8 for *r* and 8 for *n* in equation (1).

$$\begin{aligned}{}^{8}C_{8}&=\frac{8!}{8!}\\&=1\end{aligned}$$

Hence, only one possible way of encoding 8 states with three bits is possible.

Step 1 of 1 **2.034E**

The traffic-light controller of Table 2-12 in the textbook contains 6 code words to represent different states in the traffic-light controller. Usually for a system with *n* bits it is possible to encode 2^n states. Hence by means of 3 bits there are $2^3\left(\text{that is } 8\right)$ encode states possible.

Consider that every state must be encoded with at least one zero in the code word to save power. So out of the eight states use only seven states leaving the 111 state. As our traffic system needs only 6 states, we can have more than one possible way of state encoding.

Use the following combinational formula to find the combinations.

$$^{n}C_{r}=\frac{n!}{r!(n-r)!} \quad \text{...... (1)}$$

Here,

The following is the total number of states that have at least one zero in its code word.

$$\begin{aligned} n &= 2^3-1 \qquad \left(\text{except 111 state}\right)\\ &= 8-1\\ &= 7 \end{aligned}$$

The required number of states for the traffic system, $r=6$.

Substitute 7 for n and 6 for r in equation (1).

$$\begin{aligned} ^{7}C_{6} &= \frac{7!}{(6!)(7-6)!}\\ &= \frac{(7)(6)(5)(4)(3)(1)(2)}{\left[(6)(5)(4)(3)(1)(2)(1)\right](1)}\\ &= \frac{5040}{720}\\ &= 7 \end{aligned}$$

Hence, 7 different ways of assigning 6 three-bit binary state encodings are possible.

Step 1 of 3

Consider the following diagram which shows a mechanical encoding disk using a 3-bit binary code.

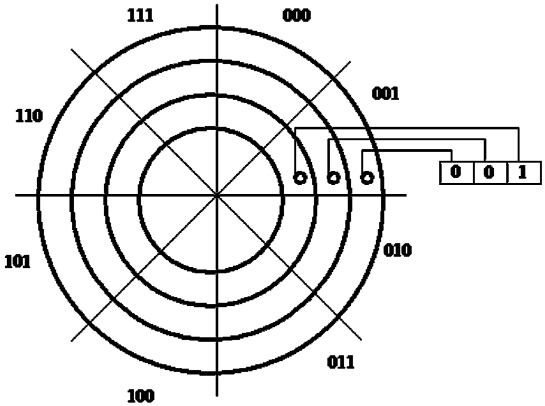


Figure 1: Mechanical encoding disk with 3-bit binary code

Step 2 of 3

When the contacts made on the disc while rotating, the values in the disc are read according to the connection of the signal source in the disc. The dark areas of the disk gives the value of 1 or logic 1 which is connected to the signal source and light areas in the disc gives the value of 0 or logic 0 which is not connected to the signal source.

When the disc is reading at boundaries for example between 101 and 110, two bits are changed between two numbers. When contact is made at boundaries, it reads 1 on the both sides and gives an incorrect reading. These types of boundaries are said to be bad boundaries. This problem can be solved by using gray code. Because only one bit, changes at time. For example when contacts made between the boundaries of 000 and 001, disc will read as 001.

Step 3 of 3

When we look in to the diagram, the below list are said to be bad boundaries since two or more of the encoded bits changes.

Boundaries	Number of encoded bits changed	
From	To	
001	010	2
011	100	3
101	110	2
111	000	3

Table 1: List of bad boundaries in mechanical encoded disk with 3 bit.

Thus, from the Table 1 the number of bad boundaries are said to be $\boxed{4}$.

Step 1 of 4

2.037E

Consider the following diagram which shows a mechanical encoding disk using a 3-bit binary code:

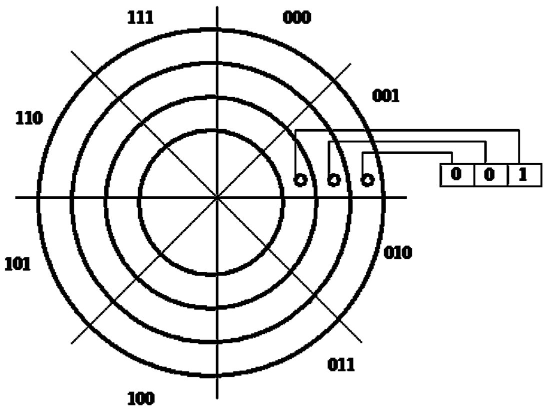


Figure 1: Mechanical encoding disk with 3 bit binary code

Step 2 of 4

When the contacts made on the disc while rotating the values in the disc are read according to the connection of the signal source in the disc. The dark areas of the disk gives the value of 1 or logic 1 which is connected to the signal source and light areas in the disc gives the value of 0 or logic 0 which is not connected to the signal source.

When the disc is reading at boundaries for example between 101 and 110, two bits are changed between two numbers. When contact is made at boundaries, it reads 1 on the both sides and gives an incorrect reading. These types of boundaries are said to be bad boundaries. This problem can be solved by using gray code. Because only one bit, changes at time. For example when contacts made between the boundaries of 000 and 001, disc will read as 001.

Step 3 of 4

When we look in to the diagram in Figure 1, the list in Table 1 are said to be bad boundaries since two or more of the encoded bits changes.

Boundaries	Number of encoded bits changed	
From	To	
001	010	2
011	100	3
101	110	2
111	000	3

Table 1: List of bad boundaries in mechanical encoding disk with 3 bit

Step 4 of 4

So in 3-bit binary code, the numbers of bad boundaries are 4. Similarly for four bit binary code, the numbers of bad boundaries are 8 as shown in Table 2.

Boundaries	Number of encoded bits changes	
From	To	
0001	0010	2
0011	0100	3
0101	0110	2
0111	1000	3
1001	1010	2
1011	1100	3
1101	1110	2
1111	0000	4

Table 2: List of bad boundaries in mechanical encoding disk with 4 bit

So in general we can say the number of bad boundaries in a mechanical encoding disk that uses an *n*-bit binary code is,

$$\text{Number bad boundaries} = \frac{2^n}{2}$$
$$= 2^{n-1}$$

Thus, the number of bad boundaries in a mechanical encoding disk that uses an *n*-bit binary code are $\boxed{2^{n-1}}$.

Step 1 of 1 2.038E

A transponder in aircraft can be used to provide altitude information in a gray code format either in a serial or a parallel format (or we can say that, in an advanced gray code format called as Gilham code). This will help the radar to identify the aircraft and to avoid collision. The reason for using gray code format is because of only one bit differs from previous transmission. Whenever the altitude needs to change, transmission is differing by only one bit. Hence it is easy to read the information. If altitude information transmission is not in gray code format, then there are more possibilities for error in reading each and every bit in information. Suppose logic 1 may consider as 0 or logic 0 may consider as 1. Thus there are more possibilities of reading the information in complemented form. Hence a gray code format is an efficient way for a transponder or altitude information transmission in aircraft.

Step 1 of 2 **2.040E**

Follow the binary codes in the traditional while assigning codes to the three way bulbs. Each bulb requires two states and hence three bulbs require six states.

The number of bits required to represent N states by the binary convention is,

$n = \log_2 N$ ...... (1)

Substitute 6 for N in equation (1).

$$\begin{aligned} n &= \log_2 6 \\ &= 3 \end{aligned}$$

Using $n = 3$ assigning states by means of gray codes that change bit by bit for corresponding states we can reduce the number of on/off cycles. **Assigning one output signal for one way of the bulb, performing Ex-OR operation will complement the previous output if the state signal from any one of the three way changes.** Since in gray code each consecutive code will differ by a single bit we have only six consecutive on/off cycles for one full rotation. Two codes are assigned to the states 4, 5. It says that the possibility of changing input signals. Those input combination also provides the desired result.

Gray code assignment for the three way bulb is shown in Table 1.

Table 1

States	Gray Code
0	000
1	001
2	011
3	010
4	110/100
5	111/101

In case of binary code to represent three ways we need two Most Significant Bits to denote the way. Least significant bit is meant to denote the state of the way. Most Significant Bit assignment for the three ways is shown in Table 2.

Table 2

First two MSB	Way assigned
00	First way
01	Second way
10	Third way

State assignment for the three ways is shown in Table 3.

Table 3

States	Binary code	
First way	OFF	000
	ON	001
Second way	OFF	010
	ON	011
Third way	OFF	100
	ON	101

Step 2 of 2

Number of transitions required for the three ways is shown in Table 4.

Table 4

Transition between states	Number of On/Off transitions required		
ON- way	OFF-way	Binary code	Gray Code
Third	First	2	1
Third	second	3	1
Second	First	2	1
Second	Third	3	1
First	Second	2	1
First	third	2	1

In binary code to control the bulb from the way that is different from the one we have done just previously, we need two-to-three changes in the signal. Hence to step over six states it requires minimum of 14 on/off cycles. While using the gray code representation, numbers of transitions get reduced by half and hence, the life time of the bulb will get double.

Step 1 of 2

Below diagram shows 3-cube (3- bit strings) and none of the lines cross each other.

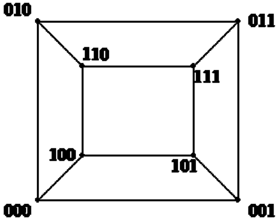


Figure 1: 3-cube diagram

Step 2 of 2

The 3-cube means possibility of 3 bit strings forming a cube. It is having 2^3 (8) vertices and each vertex that is connected is differing by only one bit. The above diagram shows that vertices connected lines do not cross each other.

Step 1 of 2

The parity bits for hamming code are obtained by Ex-OR operation of the bits whose binary equivalent of positional value has 1 in the same position. The parity bits are inserted from the right hand side of data bits in the position of 2^m where $m = 0,\ 1,\ 2,\ ...,\ n$ and $n = \lfloor \log_2 N \rfloor$.

The following is the general format for the distance-3 hamming code with 11 information bits:

$$D_{15}D_{14}D_{13}D_{12}D_{11}D_{10}D_9P_8D_7D_6D_5P_4D_3P_2P_1$$

Where

The information bits are $D_{15},\ D_{14},\ D_{13},\ D_{12},\ D_{11},\ D_{10},\ D_9,\ D_7,\ D_6,\ D_5,\ D_3$ and

The parity bits are $P_8,\ P_4,\ P_2,\ P_1$.

Step 2 of 2

Obtain the parity bit P_1 by performing Ex-OR operation with the digits whose binary equivalent of the positions have 1 in its least significant bit (LSB). Similarly, obtain the parity bits $P_8,\ P_4,$ **and** P_2 by performing Ex-OR operation with the digits whose binary equivalent for the positions have 1 in the second bit, third bit and fourth bit from right hand side of the information bits respectively.

Write the parity bits as follows:

$$P_1 = D_3 \oplus D_5 \oplus D_7 \oplus D_9 \oplus D_{11} \oplus D_{13} \oplus D_{15}$$
$$P_2 = D_3 \oplus D_6 \oplus D_7 \oplus D_{10} \oplus D_{11} \oplus D_{14} \oplus D_{15}$$
$$P_4 = D_5 \oplus D_6 \oplus D_7 \oplus D_{12} \oplus D_{13} \oplus D_{14} \oplus D_{15}$$
$$P_4 = D_9 \oplus D_{10} \oplus D_{11} \oplus D_{12} \oplus D_{13} \oplus D_{14} \oplus D_{15}$$

Step 1 of 3

In hamming code, Parity bit is obtained by the Ex-OR operation of the bits whose binary equivalent of positional value has 1 in the same position. General format for the hamming code word for one information bit (3, 1) is

$D_3\ P_2\ P_1$

Where,

D_3 is information bits

P_2, P_1 are parity bits

Step 2 of 3

Parity bits are obtained using the following expressions.

$$P_1 = D_3 \oplus D_5$$
$$P_2 = D_3 \oplus D_6$$

For the single bit higher order bits,

$$D_5 = D_6$$
$$= 0$$

Therefore,

$P_1 = D_3 \oplus 0$ ...... (1)

$P_2 = D_3 \oplus 0$ ...... (2)

The truth table of Ex-OR gate is shown in Table 1.

Table 1

X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

When one of the inputs is zero, the output of the operation is equal to the input of other gate. Rewrite the equation (1) and (2).

$P_1 = D_3$ ...... (3)

$P_2 = D_3$ ...... (4)

Step 3 of 3

By means of equation (3) and (4) we can say that the parity bits are same as data bits. And hence for the information bit 0, Parity bits are

$$P_1 = 0$$
$$P_2 = 0$$

Hamming codeword for the information bit 0 is,

$$\begin{matrix} D_3 & P_2 & P_1 \\ 0 & 0 & 0 \end{matrix}$$

Write the Parity bits for the information bit 0 from equation (3) and (4).

$$P_1 = 1$$
$$P_2 = 1$$

Write the Hamming codeword for the information bit 0.

$$\begin{matrix} D_3 & P_2 & P_1 \\ 1 & 1 & 1 \end{matrix}$$

Therefore, Hamming Code word for the information bit 0 and 1 is $\boxed{000}$ and $\boxed{111}$ respectively.

In two dimensional codes parity bits are appended at the end of each row and column. At the lower right corner we have check bits for the row and column parity bits. When one of the data bits in the two dimensional matrix is corrupted its corresponding row and column parity bits are also get corrupted. Then the horizontal row checking and vertical column checking corresponding to the data bits will not detect the error. But the **horizontal parity row and vertical parity column with corner parity bits at its end will detect the error by producing 1for even parity checker and 0 for odd parity checker.**

The Three bit error pattern that can be detected with corner parity bit is shown in Figure 1.

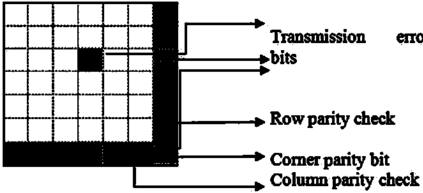


Figure 1

The two dimensional code received at the receiver, while performing the parity checking on rows, third row cannot be detected because in that row data bit and the row parity gets corrupted during the transmission. Similarly, the column parity is also corrupted and hence the parity checking on columns cannot detect the error.

Finally checking on row parity column and column parity row with corner bits will detect the error by producing high or low output corresponds to the parity.

Step 1 of 3

Distance 2 parity codes will detect a single bit error and it is not able to correct it. One parity bit is appended for each byte (8 bits). Hence, number of parity bits to be appended for 100 information bits.

Parity bits for distance-2 code is,

$$n_p = \frac{\textbf{total number of data bits}}{8}$$
$$= \frac{100}{8}$$
$$= 12.5$$
$$= 13$$

The rate code for Distance-2 codes is shown in Table 1.

Table 1

Number Of Data Bits (d)	Number Of Parity Bits (p)	Total Number Of bits (N)	Code rate $r = \frac{d}{N}$
=8	1	=9	0.89
=16	2	=18	0.89
=24	3	=27	0.89
=32	4	=36	0.89
=40	5	=45	0.89
=48	6	=54	0.89
=56	7	=63	0.89
=64	8	=72	0.89
=72	9	=81	0.89
=80	10	=90	0.89
=88	11	=99	0.89
=96	12	=108	0.89
=104	13	=117	0.89

From the Table 1, for 100 information bits we have to add 13 parity bits to 100 information bits and hence finally we get 113 bits.

Step 2 of 3

In the distance-3 (Hamming) code using the check bit 'i' we will get $2^i - 1$ bit code with $(2^i - 1 - i)$ information bits. The rate code for Distance-3 codes is shown in Table 2.

Table 2

Number Of Parity Bits (p)	Number Of data Bits $d = (2^i - 1 - i)$	Total Number Of bits $N = (2^i - 1)$	Code rate $r = \frac{d}{N}$
2	1	3	0.33
3	2	5	0.4
	3	6	0.5
	4	7	0.571
4	5 to 11	9 to 15	0.555 to 0.733
5	12 to 26	17 to 31	0.706 to 0.8387
6	27 to 57	33 to 63	0.8181 to 0.9048
7	58 to 120	65 to 127	0.8923 to 0.944

Step 3 of 3

From the Table 2 for 100 information bits we have to add 7 parity bits to 100 information bits and hence finally we have to send 107 bits. Distance 4 Hamming code appends one more extra parity bit to the Distance 3 hamming code. The extra parity bit is generated for the already existing parity bits such that it satisfies even parity code.

The rate code for Distance-4 codes is shown in Table 3.

Table 3

Number Of Parity Bits (p)	Number Of data Bits $d = (2^i - 1 - i)$	Total Number Of bits $N = (2^i - 1)$	Code rate $r = \frac{d}{N}$
3	1	4	0.25
4	2	6	0.33
	3	7	0.429
	4	8	0.5
5	5 to 11	10 to 16	0.5 to 0.6875
6	12 to 26	18 to 32	0.67 to 0.8125
7	27 to 57	34 to 64	0.794 to 0.891
8	58 to 120	66 to 128	0.878 to 0.9375

From the Table 3 for 100 information bits we have to add 8 parity bits to 100 information bits and hence finally we have to transmit 108 bits. To construct the graph for code rate versus the number of information bits D we can create the following table 4 from the data in the table 3.

Table 4

Number Of Data Bits (d)	Code rate		
	Distance 2 code	Distance 3 code	Distance 4 code
1	0.5	0.33	0.25
2	0.67	0.4	0.33
3	0.75	0.5	0.429
4	0.8	0.571	0.5
5	0.833	0.555	0.5
11	0.89	0.733	0.6875
12	0.89	0.706	0.67
26	0.89	0.8387	0.8125
27	0.89	0.8181	0.794
57	0.89	0.9048	0.891
58	0.89	0.8923	0.878
120	0.89	0.944	0.9375

Plot the graph for the data in the table 4.

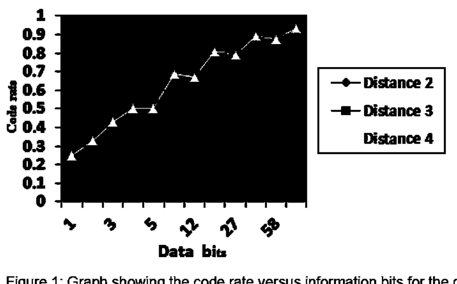


Figure 1: Graph showing the code rate versus information bits for the data in table 4

Step 1 of 4

The rate of a code is the ratio of the number of information bits to the total number of bits in a code word.

High rate means number of information bits is higher compared to the total number of parity bits. It means number of information bits should approach nearly equal to total number of bits.

A two dimensional code has higher rate compared to Hamming code because in two dimensional code check bits (row and column) can be varied, so the rate of a code is high whereas in Hamming code they are related by some formula. It means number of check bits are fixed, so the rate of a code is lower.

Step 2 of 4

Write the formula for the rate of the code as shown in equation (1).

$$\text{Rate} = \frac{\text{Information bits}}{\text{Information bits} + \text{Parity bits}}$$

Hamming code formula for the information bits, check bits, total bits are as follows.

Information bits: $2^i - 1 - i$

Check bits (Parity bits): $i + 1$

Total bits: $2^i - 1$

Step 3 of 4

Use the formulae of the rate of code for the hamming code and two dimension code and draw the table that shows the rate, information bits, and parity bits for two types of codes for the minimum distance as 4.

Hamming code	Two dimensional code				
Information bits	Parity bits	Rate of code	Information bits(original)	Parity bits(row and column)	Rate of code
1	3	0.25	1	1x2=2	0.33
<=4	4	<=0.57	<=4	1x2=2	<=0.67
<=11	5	<=0.69	<=11	1x2=2	<=0.85
<=26	6	<=0.81	<=26	1x2=2	<=0.92
<=57	7	<=0.89	<=57	1x2=2	<=0.96
<=100	8	<=0.92	<=100	1x2=2	<=0.98

Table 1

Step 4 of 4

It is observed that the number of parity bits in the hamming code is more compared to two dimensional codes.

It is clear from Table 1 that the rate of code is higher for the two dimensional code than the hamming code for the corresponding number of information bits and the variation is obtained because of the variation in the number of parity bits from hamming code to two dimensional code.

Thus, the two dimension code of distance 4 code has the higher rate compared to hamming code.

Step 1 of 6

Two dimensional codes are used to construct a distance 6 code.

It means distance-3 and distance-2 code has to be formed either in C_{row} or C_{column} codes.

The two dimensional code is constructed as shown in Figure 1.

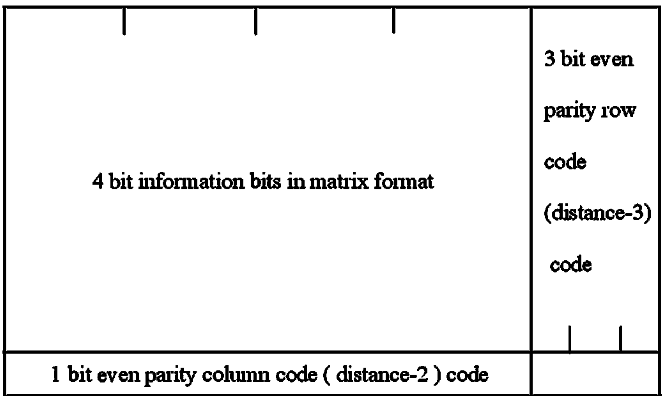

4 bit information bits in matrix format
3 bit even parity row code (distance-3) code
1 bit even parity column code (distance-2) code

Figure 1

Step 2 of 6

Generate distance- 3 code using Hamming code as follows.

Information bits: $2^i - 1 - i$

Check bits (parity bits): *i*

Total bits: $2^i - 1$

It is clear that to generate a distance-3 code, the value of *i* must be equal to the value 3.

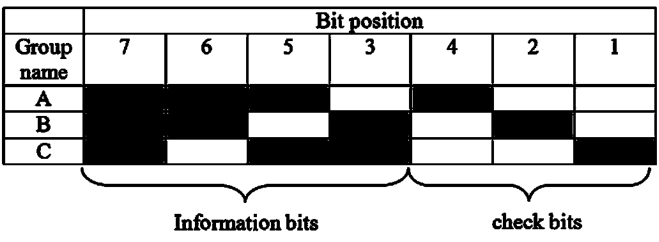
Write some distance 3 code words as shown in Table 1.

Information bits	Parity(check) bits
0000	000
0001	011
0010	101
0011	110
0100	110
0101	101
0110	011

Table 1

Step 3 of 6

Follow the parity check matrix and draw the table as follows.



	Bit position						
Group name	7	6	5	3	4	2	1
A	■	■	■		■		
B	■	■		■		■	
C	■		■	■			■

Information bits
check bits

Step 4 of 6

Figure 2

Step 5 of 6

Write some distance -2 code words using even parity as shown in Table 2.

Information bits	Parity(even) bits
000	0
001	1
010	1
011	0
100	1
101	0
110	0

Table 2

Step 6 of 6

Thus, the distance-6 code can be constructed using the distance 3 code and distance 2 code and the code words are listed in Table 1 and Table 2.

Step 1 of 2

In RAID system *'n'* numbers of disks are stacked one over the other and finally $(n+1)^{th}$ disk is placed for error detection and correction. Disk consists of blocks of data which has own CRC code. Hence to write the data in to information block *b* in drive *d*, we need the cyclic redundancy check code for the above data. It can be obtained by the following operation. *'n'* bit Data to be written in *'b'* block of the disk *'d'* can be generated by the following equation.

$C = DG$ ...... (1)

Where,

D is the row data matrix $1 \times n$

G is the Generator matrix $n \times k$, value of *k* indicates the sum of data bits and parity Bits

Write the generator matrix.

$G = [I_n \mid p_{k-n}]$ ...... (2)

Where,

I_n is the Identity matrix

Substitute equation (2) in (1).

$C = [I_n \mid p_{k-n}]$ ...... (3)

The operation results in data bits appended by the *n-k* parity bits. Each block is having 512 data bytes. Hence

Total number of data bits in a block is,

$$\begin{aligned} n &= (512)(8) \\ &= 4096 \text{ bits} \end{aligned}$$

4096 bits are appended by *k-n* parity bits and it consists of *k* bits can be directly return to the block.

Step 2 of 2

Hence in total *k* bits are available in a block *'b'* in disk*'d'*. In disk $d+1$ *b*th block will have parity codes for bits in block*'b'* in all disks. Obtaining the even parity of *b*th block in disk $d+1$ with the data obtained in equation (3) we will get the even parity of all *b*th block in disk 1 to *d*.

$D_{d+1} = C \oplus D'_{d+1}$

Where,

D'_{d+1} is the old parity data available in *b*th block in disk $d+1$

D_{d+1} is the new parity data including the data given by equation (3) available in

b th block in disk $d+1$

Step 1 of 2

The waveform bit pattern for 10101110, when sent serially using the NRZ, NRZI, RZ, BPRZ and Manchester codes are shown in Figure 1.

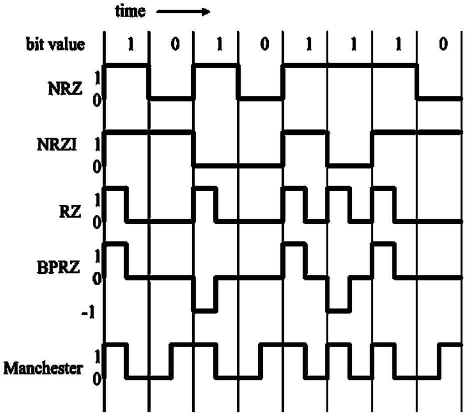


Figure 1: Waveform Pattern for 10101110

Step 2 of 2

NRZ (Non Return to Zero) code: When 1 is transmitted, then waveform at level 1 and 0 is transmitted, then waveform at level 0.

NRZI (Non Return to Zero Invert on 1's) code: When 1 is transmitted, then waveform is opposite of the level that was sent in previous and for 0 as the same level.

RZ (Return to Zero) code: When 1 is transmitted then the waveform at level 1 for the half (or we can say fraction) of the bit time and for 0, no transition.

BPRZ (Bipolar Return-to-Zero) code: This code transmits three signal levels +1, 0 and -1. When 1 is transmitted then the waveform at level 1 for the half (or we can say fraction) of the bit time but 1's are alternatively transmit as +1 & -1 and for 0, no transition.

Manchester Code: When 1 is transmitted, the waveform moves from 1 to 0 per bit cell and when 0 is transmitted, 0 to 1 transition occurs per bit cell.

Step 1 of 8 3.02DP

In a SDRAM module, as per the context the LOW signal is said to be in the range of 0.0-0.7V and HIGH signal to be in the range of 1.7 – 2.5V. In a negative logic LOW is consider as a 1 and HIGH is consider as 0. Using negative-logic convention, the below signal levels are indicated as 1 (LOW) or 0 (HIGH).

(a) Given signal level, **0.0 V**

It is in the range of $\mathbf{0.0-0.7\ V}$, which is defined as a **LOW** signal.

Therefore, from the definition of negative logic convention the logic value of $\mathbf{0.0\ V}$ is $\boxed{1}$.

Step 2 of 8

(b) Given signal level, **0.7 V**

It is in the range of $\mathbf{0.0-0.7\ V}$, which is defined as a **LOW** signal.

Therefore, from the definition of negative logic convention the logic value of $\mathbf{0.7\ V}$ is $\boxed{1}$.

Step 3 of 8

(c) Given signal level, **1.7 V**

It is in the range of $\mathbf{1.7-2.5\ V}$, which is defined as a **HIGH** signal.

Therefore, from the definition of negative logic convention the logic value of $\mathbf{1.7\ V}$ is $\boxed{0}$.

Step 4 of 8

(d) Given signal level, $\mathbf{-0.6\,V}$.

It is undefined signal level. That is a circuit may interpret them as either 0 or 1.

Therefore, the logic value of $\mathbf{-0.6\ V}$ is $\boxed{\textbf{either 0 or 1}}$.

Step 5 of 8

(e) Given signal level, $\mathbf{1.6\,V}$.

It is in the intermediate range $\mathbf{0.7-1.7\ V}$ are not expected to occur except during signal transitions, and yield undefined logic values.

Therefore, the circuit may interpret them as either 0 or 1.

Step 6 of 8

(f) Given signal level, $\mathbf{-2.0\,V}$.

It is undefined signal level. That is a circuit may interpret them as either 0 or 1.

Therefore, the logic value of $\mathbf{-2.0\ V}$ is $\boxed{\textbf{either 0 or 1}}$.

Step 7 of 8

(g) Given signal level, **2.5 V**

It is in the range of $\mathbf{1.7-2.5\ V}$, which is defined as a **HIGH** signal.

Therefore, from the definition of negative logic convention the logic value of $\mathbf{2.5\ V}$ is $\boxed{0}$.

Step 8 of 8

(h) Given signal level, **3.3 V**

It is undefined signal level. That is a circuit may interpret them as either 0 or 1.

Therefore, the logic value of $\mathbf{3.3\ V}$ is $\boxed{\textbf{either 0 or 1}}$.

Step 1 of 1 **3.03DP**

The logic buffer amplifier is the one which works as a non-linear amplifier that maps the entire set of possible analog input voltages into just two output voltages, HIGH and LOW. An audio amplifier has a linear response over its specified operating range, mapping each input voltage into an output voltage that is directly proportional to the input voltage.

Step 1 of 2 3.04DP

Yes, buffer amplifier equivalent to a 1-input AND gate or a 1-input OR gate.

Showing buffer amplifier equivalent to 1-input AND gate:

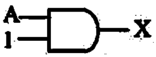


Figure 1

In the Figure 1 the second terminal always constant input which is equal to logic HIGH.

Table 1:

A	2nd terminal	X
1	1	1
0	1	0

From the table 1, it is observed that, the output X is follows the input A. Hence, buffer amplifier equivalent to a 1-input AND gate.

Step 2 of 2

Showing buffer amplifier equivalent to 1-input OR gate:

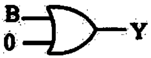


Figure 2

In the Figure 2 the second terminal always constant input which is equal to logic LOW.

Table 2:

B	2nd terminal	X
0	0	0
1	0	1

From the table 2, it is observed that, the output X is follows the input B. Hence, buffer amplifier equivalent to a 1-input OR gate.

Step 1 of 1

A NAND gate produces the same output as an OR gate with inverted inputs.

For example, consider the two inputs of NAND gate as A &B.

The output equation of NAND gate is

$$\mathbf{F=(A\cdot B)'}$$

According to DeMorgan's theorem, the above equation can be written as,

$$\mathbf{F=(A\cdot B)'=A'+B'} \quad \text{......}\ (1)$$

From the equation, it is clear that a NAND gate produces the same output as an OR gate with inverted inputs.

Therefore, the statement "For a given set of input values, a NAND gate produces the opposite output as an OR gate with inverted inputs." is $\boxed{\textbf{False}}$.

Step 1 of 1

3.07DP

The two completely different definitions of “gate” used in this context are

(1) A circuit with multiple inputs and one output that is energized only when a designated set of input pulses is received.

(2) A terminal used to control output current (that is flow of carriers in the channel) in the field effect transistor is called **gate**.

Step 1 of 2

3.08DP

The two input CMOS NAND gate is drawn as follows:

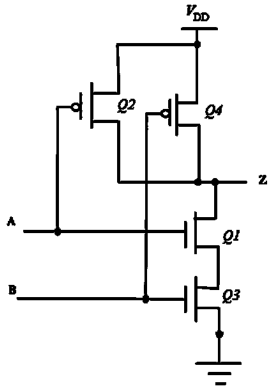


Figure 1: 2-input CMOS NAND gate

In above circuit, the numbers of transistors used are four. 2 PMOS and 2 NMOS transistors are used.

Step 2 of 2

The equivalent circuit for CMOS NAND gate using Single pole double throw relays is as follows:

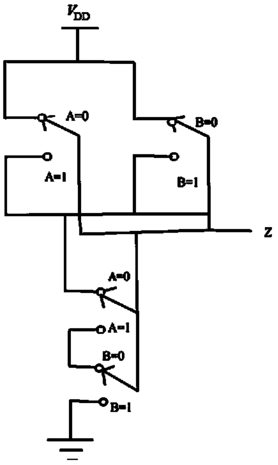


Figure 2: 2-input CMOS NAND gate using single pole double throw relays.

Step 1 of 1

3.09DP

CMOS NAND and NOR gates do not have identical electrical performance. For a given silicon area, an *n*-channel transistor has lower "on" resistance than a *p*-channel transistor. Therefore, when transistors are put in series, *k* *n*-channel transistors have lower "on" resistance than do *k* *p*-channel ones. As a result, a *k*-input NAND gate is generally faster than and preferred over a k-input NOR gate.

Step 1 of 1

Fan-in: The number of inputs that a gate can have in a particular logic family is called the logic family's fan-in.

Fan-out: It is defined as the number of logic gate inputs that can be driven from a single gate output of the same type.

Mostly for a circuit, fan-out is likely to be calculated. Fan-in is rarely used.

Step 1 of 4

3.11DP

Below diagram shows the CMOS AND-OR-INVERT gate.

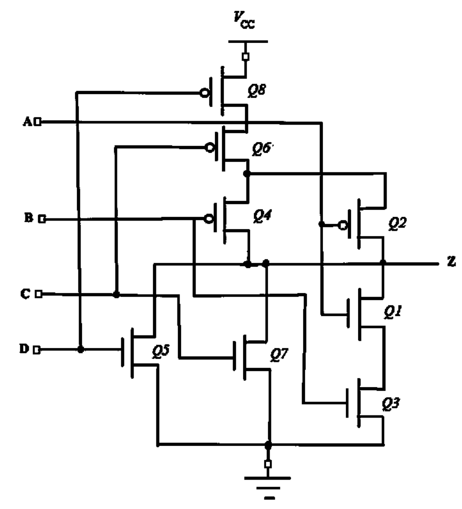


Figure 1: CMOS AND-OR-INVERT gate.

Step 2 of 4

The function table for the above circuit is given by as follows,

A	B	C	D	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Z
L	L	L	L	off	on	off	on	off	on	off	on	H
L	L	L	H	off	on	off	on	on	on	off	off	L
L	L	H	L	off	on	off	on	off	off	on	on	L
L	L	H	H	off	on	off	on	on	off	on	off	L
L	H	L	L	off	on	on	off	off	on	off	on	H
L	H	L	H	off	on	on	off	on	on	off	off	L
L	H	H	L	off	on	on	off	off	off	on	on	L
L	H	H	H	off	on	on	off	on	off	on	off	L
H	L	L	L	on	off	off	on	off	on	off	on	H
H	L	L	H	on	off	off	on	on	on	off	off	L
H	L	H	L	on	off	off	on	off	off	on	on	L
H	L	H	H	on	off	off	on	on	off	on	off	L
H	H	L	L	on	off	on	off	off	on	off	on	L
H	H	L	H	on	off	on	off	on	on	off	off	L
H	H	H	L	on	off	on	off	off	off	on	on	L
H	H	H	H	on	off	on	off	on	off	on	off	L

Step 3 of 4

The logic function for the above circuit can be written as follows:

$Z = ((A \cdot B) + C + D)'$

The logic diagram for the above circuit using AND, OR and NOT gates is as follows:

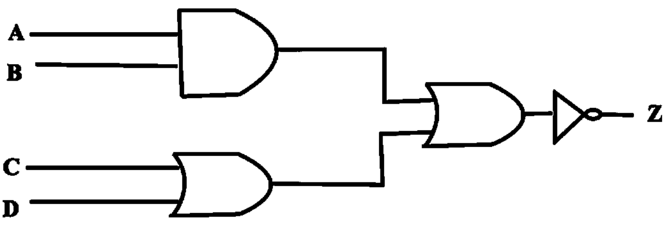


Figure 2: AND-OR-NOT Circuit for the function $Z = ((A \cdot B) + C + D)'$.

Step 4 of 4

Figure 2 can also be redrawn as follows:

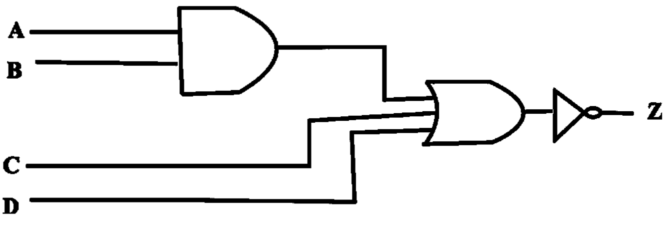


Figure 3: AND-OR-NOT Circuit for the function $Z = ((A \cdot B) + C + D)'$.

Step 1 of 4

Below diagram shows the CMOS OR-AND-INVERT gate.

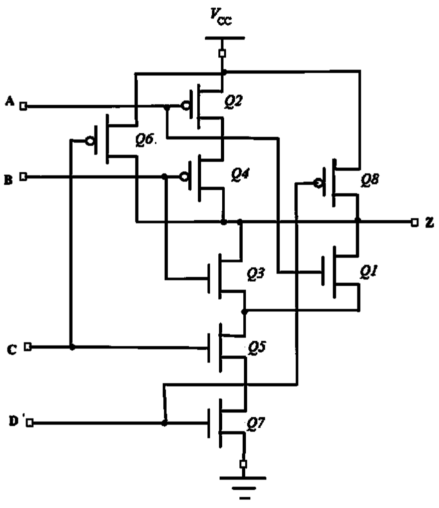


Figure 1: CMOS AND-OR-INVERT gate.

Step 2 of 4

The function table for the above circuit is given by as follows,

A	B	C	D	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Z
L	L	L	L	off	on	off	on	off	on	off	on	H
L	L	L	H	off	on	off	on	off	on	on	off	H
L	L	H	L	off	on	off	on	on	off	off	on	H
L	L	H	H	off	on	off	on	on	off	on	off	H
L	H	L	L	off	on	on	off	off	on	off	on	H
L	H	L	H	off	on	on	off	off	on	on	off	H
L	H	H	L	off	on	on	off	on	off	off	on	H
L	H	H	H	off	on	on	off	on	off	on	off	L
H	L	L	L	on	off	off	on	off	on	off	on	H
H	L	L	H	on	off	off	on	off	on	on	off	H
H	L	H	L	on	off	off	on	on	off	off	on	H
H	L	H	H	on	off	off	on	on	off	on	off	L
H	H	L	L	on	off	on	off	off	on	off	on	H
H	H	L	H	on	off	on	off	off	on	on	off	H
H	H	H	L	on	off	on	off	on	off	off	on	H
H	H	H	H	on	off	on	off	on	off	on	off	L

Step 3 of 4

The logic function for the above circuit is written below,

$Z = ((A+B) \cdot C \cdot D)'$

The logic diagram for the above circuit using OR, AND and NOT gates are as follows:

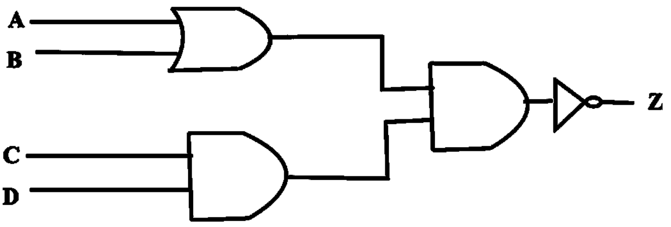


Figure 2: The OR-AND-NOT circuit for the function $Z = ((A+B) \cdot C \cdot D)'$

Step 4 of 4

Figure 2 can also be redrawn as follows:

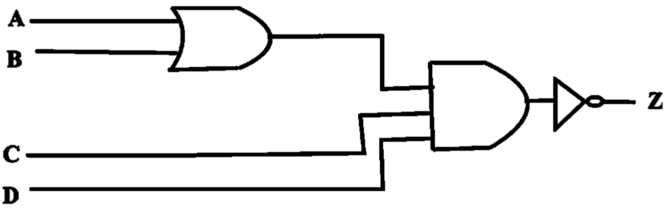


Figure 2: The OR-AND-NOT circuit for the function $Z = ((A+B) \cdot C \cdot D)'$

Step 1 of 4

3.13DP

Below diagram shows the 3-input CMOS NOR gate.

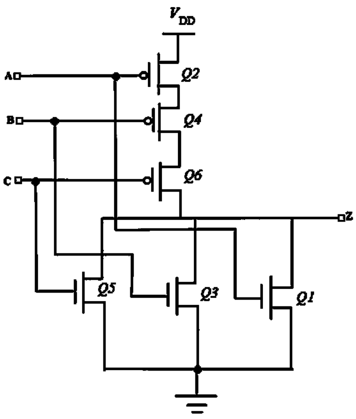


Figure 1: 3-input CMOS NOR gate.

Step 2 of 4

The function table for the above circuit is given by as follows:

A	B	C	Q1	Q2	Q3	Q4	Q5	Q6	Z
L	L	L	off	on	off	on	off	on	H
L	L	H	off	on	off	on	on	off	L
L	H	L	off	on	on	off	off	on	L
L	H	H	off	on	on	off	on	off	L
H	L	L	on	off	off	on	off	on	L
H	L	H	on	off	off	on	on	off	L
H	H	L	on	off	on	off	off	on	L
H	H	H	on	off	on	off	on	off	L

Step 3 of 4

Table 1: The 3-input NOR gate function table

Step 4 of 4

The logic symbol for NOR gate is as follows:

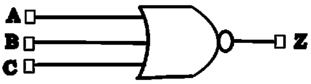


Figure 2: The logic symbol for NOR gate.

Step 1 of 2 **3.15DP**

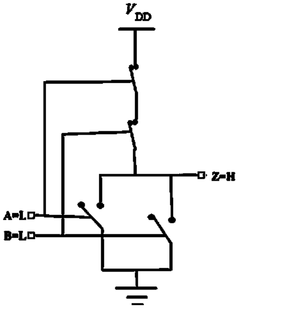
The four input combinations are shown in the following table.

A	B
L	L
L	H
H	L
H	H

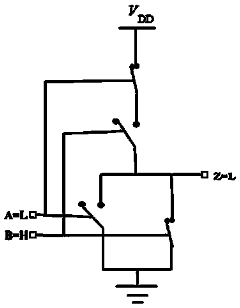
Table 1

Step 2 of 2

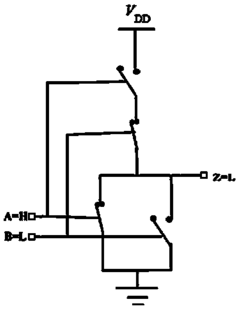
Switch models of 2-input CMOS NOR gate for all four input combination is as follows,



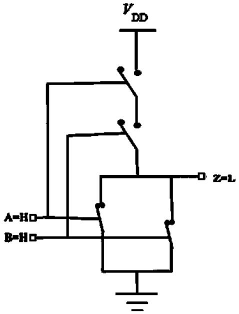
CMOS 2-input NOR GATE with both the inputs low



CMOS 2-input NOR GATE with one input low



CMOS 2-input NOR GATE with one input low



CMOS 2-input NOR GATE with both the inputs high

Figure 1: Switch models of 2-input CMOS OR gate.

Step 1 of 1

3.16DP

In a CMOS circuit, the outputs of the all logic gates are in inversion form. In order to get the expected result, need to put inverter in the output which require further 2 more transistors. Thus CMOS inverting gate is having fewer transistor than non-inverting gate.

Step 1 of 4

3.17DP

Four different 3-inputs CMOS logic symbols that each use six transistors are

(i) 3-inputs NAND logic,

(ii) 3-inputs NOR logic,

(iii) 3-inputs AND-OR-INVERT logic,

(iv) 3-inputs OR-AND-INVERT logic.

Step 2 of 4

The 3-inputs CMOS NAND logic symbols are given by as follows,

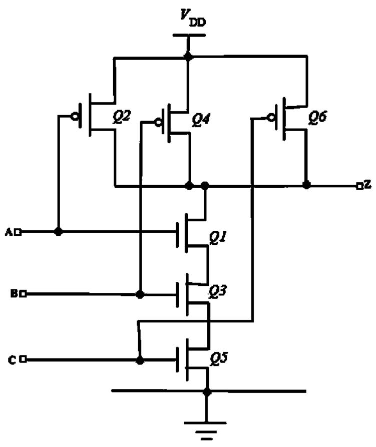


Figure 1: 3-inputs CMOS NAND Logic symbols

Step 3 of 4

The 3-inputs CMOS NOR logic symbols are given by as follows,

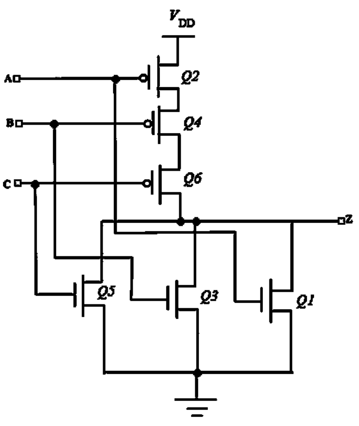


Figure 2: The 3-inputs CMOS NOR logic symbols.

Step 4 of 4

The 3-inputs CMOS AND-OR-INVERT logic is given by as follows,

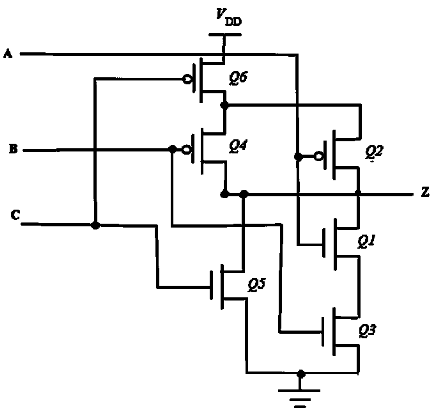


Figure 3: The 3-inputs CMOS AND-OR-INVERT logic.

The 3-inputs CMOS OR-AND-INVERT logic is given by as follows,

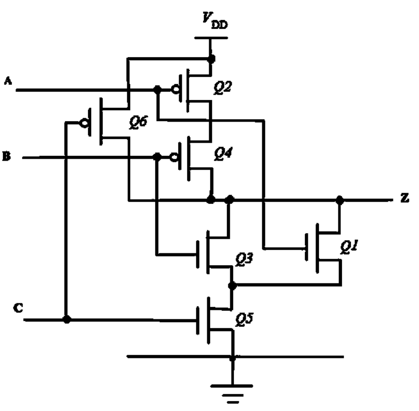


Figure 4: The 3-inputs CMOS OR-AND-INVERT logic.

Step 1 of 1

3.19DP

In general in a CMOS gate, the output is in inverted form. In order to, get the expected result we need to put inverter at the output, which requires further two transistors.

Hence, when compared to 8-input CMOS NAND gate and AND gate, AND gate requires extra transistors to get the expected result. This situation will leads to extra propagation delay.

Thus, the 8-input $\boxed{\text{CMOS NAND}}$ gate is faster than AND gate.

3.21DP

Step 1 of 3

The worst-case LOW state and HIGH state DC noise margin is calculated by the formula as below,

The HIGH-state DC noise margin is given by, $\mathbf{V_H = V_{OH} - V_{IH}}$.

Whereas V_{OH} = Minimum output Voltage and V_{IH} = Minimum Input Voltage.

The LOW-state DC noise margin is given by, $\mathbf{V_L = V_{IL} - V_{OL}}$.

Whereas V_{IL} = Maximum Input Voltage and V_{OL}= Maximum output Voltage.

Step 2 of 3

For 74HC00 below are the noise margins:

For CMOS Load:

(i) HIGH-state DC noise margin:

$$\begin{aligned}\mathbf{V_H} &= \mathbf{V_{OH} - V_{IH}}\\ &= \mathbf{4.499 - 3.15}\\ &= \mathbf{1.349\ V}\end{aligned}$$

(ii) LOW-state DC noise margin:

$$\begin{aligned}\mathbf{V_L} &= \mathbf{V_{IL} - V_{OL}}\\ &= \mathbf{1.35 - 0.1}\\ &= \mathbf{1.25\ V}\end{aligned}$$

Thus for CMOS Load, the HIGH-state DC noise margin is $\boxed{\mathbf{1.349\ V}}$ and the LOW-state DC noise margin is $\boxed{\mathbf{1.25\ V}}$.

Step 3 of 3

For TTL Load:

(i) HIGH-state DC noise margin:

$$\begin{aligned}\mathbf{V_H} &= \mathbf{V_{OH} - V_{IH}}\\ &= \mathbf{3.84 - 3.15}\\ &= \mathbf{0.69\ V}\end{aligned}$$

(ii) LOW-state DC noise margin:

$$\begin{aligned}\mathbf{V_L} &= \mathbf{V_{IL} - V_{OL}}\\ &= \mathbf{1.35 - 0.33}\\ &= \mathbf{1.02\ V}\end{aligned}$$

Thus for TTL Load, the HIGH-state DC noise margin is $\boxed{\mathbf{0.69\ V}}$ and the LOW-state DC noise margin is $\boxed{\mathbf{1.02\ V}}$.

Step 1 of 1 **3.22DP**

The worst case value for the electrical parameters for CMOS circuits 74HC00 is given by as follows:

Electrical Parameters	TTL load	CMOS Load
V_{OH}	3.84V	4.4V
V_{IH}	3.15V	3.15V
V_{OL}	0.33V	0.1V
V_{IL}	1.35V	1.35V
I_{IH}	1μA	1μA
I_{IL}	-1μA	-1μA
I_{OL}	4mA	20μA
I_{OH}	-4mA	-20μA

Step 1 of 1 **3.23DP**

The Sink current is the current which flows from power supply to ground through the load, and through the device output, whereas the Source current which flows from power supply to ground out of the device output and through the load.

It is known that, the output current in HIGH state is represented with a negative number, by convention the current flow measured at a device terminal is positive if positive current flows into the device and in HIGH state, current flows output of the output terminal.

Thus, if the current at a device output is specified as a negative number, then the output is considered to be a $\boxed{\textbf{sourcing current}}$.

Step 1 of 2

3.24DP

It is known that, the maximum output Voltage or maximum power consumption is happen when the input voltage is same as the output voltage. That is,

$$V_{IN} = V_{OUT}$$

Where,

V_{IN} is the input voltage of the CMOS circuit

V_{OUT} is the output voltage of the CMOS circuit

It is also known that, the maximum output voltage is equal to the half of the supply voltage.

$$\begin{aligned} V_{IN} &= V_{OUT} \\ &= \frac{V_{DD}}{2} \\ &= \frac{V_{OH}}{2} \end{aligned}$$

Where,

V_{OH} is the maximum output Voltage

Step 2 of 2

From the datasheet of CMOS device 74HC00, the value of the HIGH voltage is $\mathbf{4.4\ V}$.

Now, calculate the input level of the 74HC00 to consume the most power.

$$\begin{aligned} V_{IN} &= \frac{V_{OH}}{2} \\ &= \frac{4.4}{2} \\ &= 2.2\text{ V} \end{aligned}$$

Thus, the input level V_{IN} of the 74HC00 to consume the most power is $\boxed{2.2\text{ V}}$.

Step 1 of 5

The LOW state DC fan out of a CMOS gate is defined as the ratio of the minimum output current (I_{OL}) of the CMOS gate which is used to drive the another gate and the minimum input current (I_{IL}) of the gate, which is required for working or operate.

That is,

$$\text{LOW state DC fan-out} = \frac{I_{OL}}{I_{IL}}$$

Step 2 of 5

The current I_{OL} of 74HC00 is given by $4\ \text{mA}$ which is used to drive the 74LS00 and I_{IL} of 74LS00 is given by $0.2\ \text{mA}$ which is required to operate.

Now, calculate the LOW state DC fan-out when 74HC00 is driving 74LS00 (TTL).

$$\begin{aligned}\text{LOW state DC fan-out} &= \frac{I_{OL}}{I_{IL}} \\ &= \frac{4\times10^{-3}}{0.2\times10^{-3}} \\ &= 20\end{aligned}$$

Thus, the LOW sate DC fan-out when 74HC00 is driving 74LS00 is $\boxed{20}$.

Step 3 of 5

The HIGH state DC fan-out of a CMOS gate is defined as the ratio of the maximum output current (I_{OH}) of the CMOS gate which is used to drive the another gate and the maximum input current (I_{IH}) of the gate, which is required for working or operate.

That is,

$$\text{HIGH state DC fan-out} = \frac{I_{OL}}{I_{IL}}$$

Step 4 of 5

The current I_{OH} of 74HC00 is given by $4\ \text{mA}$ which is used to drive the 74LS00 and I_{IH} of 74LS00 is given by $20\ \mu\text{A}$ which is required to operate.

Now, calculate the HIGH state DC fan-out when 74HC00 is driving 74LS00 (TTL).

$$\begin{aligned}\text{HIGH state DC fan-out} &= \frac{I_{OL}}{I_{IL}} \\ &= \frac{4\times10^{-3}}{20\times10^{-6}} \\ &= 200\end{aligned}$$

Thus, the HIGH sate DC fan-out when 74HC00 is driving 74LS00 is $\boxed{200}$.

Step 5 of 5

The Overall DC fan out is minimum of the LOW-state and HIGH-state DC fan-out.

In this case, the minimum of 20, 200 is 20. So, the overall DC fan-out is 20.

Thus, the DC fan-out when 74HC00 is driving 74LS00 is $\boxed{20}$.

Step 1 of 2 **3.26DP**

Consider the following formula to obtain the "on" resistance of the p-channel output transistor of the 74HC00.

$$R_{p(on)} = \frac{V_{DD} - V_{OHminT}}{|I_{OHmaxT}|} \text{ ...... (1)}$$

From Table 3-3 in the text book, we have the following values.

$$V_{DD} = 5 \text{ V}$$
$$V_{OHminT} = 3.84 \text{ V}$$
$$I_{OHmaxT} = -4 \text{ mA}$$

Substitute 5 V for V_{DD}, 3.84 V for V_{OHminT}, and -4 mA for I_{OHmaxT} in equation (1).

$$\begin{aligned} R_{p(on)} &= \frac{5 - 3.84}{|-4 \times 10^{-3}|} \\ &= \frac{1.16}{4 \times 10^{-3}} \\ &= 290\ \Omega \end{aligned}$$

Thus, the "on" resistance $R_{p(on)}$ of the p-channel output transistor of the 74HC00 is $\boxed{290\ \Omega}$.

Step 2 of 2

Consider the following formula to obtain the "on" resistance of the n-channel output transistor of the 74HC00.

$$R_{n(on)} = \frac{V_{OLmaxT}}{I_{OLmaxT}} \text{ ...... (2)}$$

From Table 3-3 in the text book, we have the following values.

$$V_{OLmaxT} = 0.33 \text{ V}$$
$$I_{OLmaxT} = 4 \text{ mA}$$

Substitute 0.33 V for V_{OLmaxT} and 4 mA for I_{OLmaxT} in equation (2).

$$\begin{aligned} R_{n(on)} &= \frac{0.33 \text{ V}}{4 \times 10^{-3}} \\ &= 0.0825 \times 10^{3} \\ &= 82.5\ \Omega \end{aligned}$$

Thus, the "on" resistance $R_{n(on)}$ of the n-channel output transistor of the 74HC00 is $\boxed{82.5\ \Omega}$.

Step 1 of 34

Refer to Figure 3.27 (a) and Table 3.3 in the textbook.

(a)

The resistive load of 120 Ω is connected to V_{CC}.

A 74x-220 is the commercial part, so the the source voltage is $V_{CC} = 5 \pm 5\%$, but for the output high voltage test conditions are $V_{CC} = \text{MIN}$

Where, Min is the minimum value of V_{CC}

Calculate the minimum value of V_{CC}

$$\begin{aligned} V_{CC} &= 5 - 5\% \\ &= 5 - 5\left(\frac{5}{100}\right) \\ &= 5 - 0.25 \\ &= 4.75\ \text{V} \end{aligned}$$

Step 2 of 34

Draw the circuit with the resistive load of 120 Ω to V_{CC}

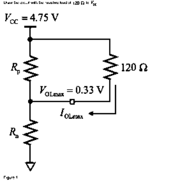


Figure 1

Step 3 of 34

Calculate the sinking current, I_{OLmax}

$$\begin{aligned} I_{OLmax} &= \frac{V_{CC} - V_{OLmax}}{R_1} \\ &= \frac{4.75 - 0.33}{120} \\ &= 0.03683 \\ &= 36.83\ \text{mA} \end{aligned}$$

Therefore, the value of I_{OLmax} is very far beyond the commercial range. Hence, the output drive specifications exceed over the commercial range.

Step 4 of 34

(b)

The resistive load of 270 Ω is connected to V_{CC} and 330 Ω is connected to GND

Draw the circuit with the resistive loads.

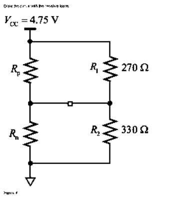


Figure 2

Step 5 of 34

Convert the voltage divider circuit in Figure 2 to Thevenin's equivalent.

Calculate the Thevenin's voltage V_{Thev}

$$\begin{aligned} V_{Thev} &= \left(\frac{R_2}{R_1 + R_2}\right) V_{CC} \\ &= \left(\frac{330}{270 + 330}\right)(4.75) \\ &= \frac{1567.5}{600} \\ &= 2.6125\ \text{V} \end{aligned}$$

Step 6 of 34

Calculate the Thevenin's resistance R_{Thev}

$$\begin{aligned} R_{Thev} &= \frac{R_1 R_2}{R_1 + R_2} \\ &= \frac{(270)(330)}{270 + 330} \\ &= \frac{89100}{600} \\ &= 148.5\ \Omega \end{aligned}$$

Step 7 of 34

Draw the modified circuit with Thevenin equivalent to calculate the sinking current.

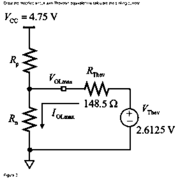


Figure 3

Step 8 of 34

Calculate the sinking current I_{OLmax}

$$\begin{aligned} I_{OLmax} &= \frac{V_{Thev} - V_{OLmax}}{R_{Thev}} \\ &= \frac{2.6125 - 0.33}{148.5} \\ &= 0.01537 \\ &= 15.37\ \text{mA} \end{aligned}$$

Therefore, the value of I_{OLmax} is very far beyond the commercial range.

Step 9 of 34

Draw the modified circuit with Thevenin equivalent to calculate the sourcing current.

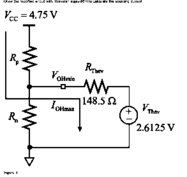


Figure 4

Step 10 of 34

Calculate the Sourcing current I_{OHmax}

$$\begin{aligned} I_{OHmax} &= \frac{V_{OHmin} - V_{Thev}}{R_{Thev}} \\ &= \frac{3.84 - 2.6125}{148.5} \\ &= 0.008266 \\ &= 8.266\ \text{mA} \end{aligned}$$

Therefore, the value of I_{OHmax} is very far beyond the commercial range. Hence, the output drive specifications exceed over the commercial range.

Step 11 of 34

(c)

The resistive load of 820 Ω is connected to GND

Draw the modified circuit with 820 Ω to GND to calculate the Sourcing current.

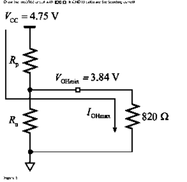


Figure 5

Step 12 of 34

Calculate the Sourcing current I_{OHmax}

$$\begin{aligned} I_{OHmax} &= \frac{V_{CC} - V_{OHmin}}{R_1} \\ &= \frac{4.75 - 3.84}{820} \\ &= 0.00111 \\ &= 1.11\ \text{mA} \end{aligned}$$

Therefore, the value of I_{OHmax} is beyond the commercial range. Hence, the output drive specifications are in the commercial range.

Step 13 of 34

(d)

The resistive load of 470 Ω is connected to V_{CC} and 470 Ω is connected to GND

Draw the circuit with the resistive loads.

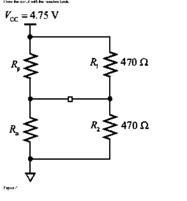


Figure 7

Step 14 of 34

Convert the voltage divider circuit in Figure 6 to Thevenin's equivalent.

Calculate the Thevenin's voltage V_{Thev}

$$\begin{aligned} V_{Thev} &= \left(\frac{R_2}{R_2 + R_1}\right) V_{CC} \\ &= \left(\frac{470}{470 + 470}\right)(4.75) \\ &= \frac{2232.5}{940} \\ &= 2.375\ \text{V} \end{aligned}$$

Calculate the Thevenin's resistance R_{Thev}

$$\begin{aligned} R_{Thev} &= \frac{R_1 R_2}{R_1 + R_2} \\ &= \frac{(470)(470)}{470 + 470} \\ &= 235\ \Omega \end{aligned}$$

Step 15 of 34

Draw the modified circuit with Thevenin equivalent to calculate the sinking current.

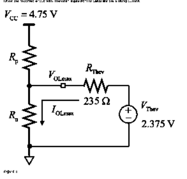


Figure 8

Step 16 of 34

Calculate the sinking current, I_{OLmax}

$$\begin{aligned} I_{OLmax} &= \frac{V_{Thev} - V_{OLmax}}{R_{Thev}} \\ &= \frac{2.375 - 0.33}{235} \\ &= 0.0087 \\ &= 8.7\ \text{mA} \end{aligned}$$

Therefore, the value of I_{OLmax} is very far beyond the commercial range.

Step 17 of 34

Draw the modified circuit with Thevenin equivalent to calculate the sourcing current.

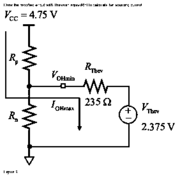


Figure 9

Step 18 of 34

Calculate the Sourcing current I_{OHmax}

$$\begin{aligned} I_{OHmax} &= \frac{V_{OHmin} - V_{Thev}}{R_{Thev}} \\ &= \frac{3.84 - 2.375}{235} \\ &= 0.006234 \\ &= 6.234\ \text{mA} \end{aligned}$$

Therefore, the value of I_{OHmax} is very far beyond the commercial range. Hence, the output drive specifications exceed over the commercial range.

Step 19 of 34

(e)

The resistive load of 1 kΩ is connected to V_{CC}

Draw the circuit with the resistive load of 1 kΩ to V_{CC}

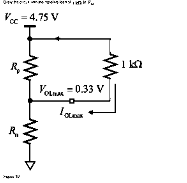


Figure 10

Step 20 of 34

Calculate the sinking current I_{OLmax}

$$\begin{aligned} I_{OLmax} &= \frac{V_{CC} - V_{OLmax}}{R_1} \\ &= \frac{4.75 - 0.33}{1.0 \times 10^3} \\ &= 4.42 \times 10^{-3} \\ &= 4.42\ \text{mA} \end{aligned}$$

Therefore, the value of I_{OLmax} is nearer to the commercial range. Hence the output drive specifications not exceeded the commercial range.

Step 21 of 34

(f)

The resistive load of 1.2 kΩ is connected to V_{CC} and 820 Ω is connected to GND

Draw the circuit with the resistive loads.

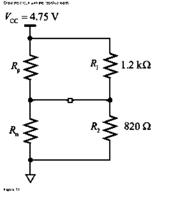


Figure 11

Step 22 of 34

Convert the voltage divider circuit in Figure 11 to Thevenin's equivalent.

Calculate the Thevenin's voltage V_{Thev}

$$\begin{aligned} V_{Thev} &= \left(\frac{R_2}{R_1 + R_2}\right) V_{CC} \\ &= \left(\frac{820}{1200 + 820}\right)(4.75) \\ &= \frac{3895}{2020} \\ &= 1.928\ \text{V} \end{aligned}$$

Calculate the Thevenin's resistance R_{Thev}

$$\begin{aligned} R_{Thev} &= \frac{R_1 R_2}{R_1 + R_2} \\ &= \frac{(820)(1200)}{1200 + 820} \\ &= \frac{984000}{2020} \\ &= 487.13\ \Omega \end{aligned}$$

Step 23 of 34

Draw the modified circuit with Thevenin equivalent to calculate the sinking current.

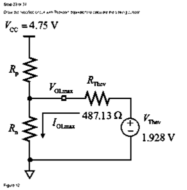


Figure 12

Step 24 of 34

Calculate the sinking current, I_{OLmax}

$$\begin{aligned} I_{OLmax} &= \frac{V_{Thev} - V_{OLmax}}{R_{Thev}} \\ &= \frac{1.928 - 0.33}{487.13} \\ &= 0.00328 \\ &= 3.28\ \text{mA} \end{aligned}$$

Therefore, the value of I_{OLmax} is in the commercial range.

Step 25 of 34

Draw the modified circuit with Thevenin equivalent to calculate the sourcing current.

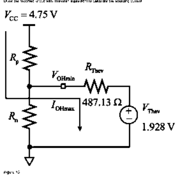


Figure 13

Step 26 of 34

Calculate the Sourcing current I_{OHmax}

$$\begin{aligned} I_{OHmax} &= \frac{V_{OHmin} - V_{Thev}}{R_{Thev}} \\ &= \frac{3.84 - 1.928}{487.13} \\ &= 0.00392 \\ &= 3.92\ \text{mA} \end{aligned}$$

Therefore, the value of I_{OHmax} is very far beyond the commercial range. Hence, the output drive specifications exceed over the commercial range.

Step 27 of 34

(g)

The resistive load of 4.7 kΩ is connected to V_{CC}

Draw the circuit with the resistive load of 4.7 kΩ to V_{CC}

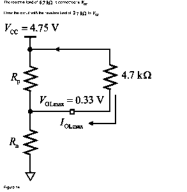


Figure 14

Step 28 of 34

Calculate the sinking current, I_{OLmax}

$$\begin{aligned} I_{OLmax} &= \frac{V_{CC} - V_{OLmax}}{R_1} \\ &= \frac{4.75 - 0.33}{4.7 \times 10^3} \\ &= 0.94 \times 10^{-3} \\ &= 0.94\ \text{mA} \end{aligned}$$

Therefore, the value of I_{OLmax} is in the commercial range. Hence the output drive specifications are in the commercial range.

Step 29 of 34

(h)

The resistive load of 1.2 kΩ is connected to V_{CC} and 1 kΩ is connected to GND

Draw the circuit with the resistive loads.

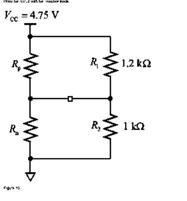


Figure 15

Step 30 of 34

Convert the voltage divider circuit in Figure 15 to Thevenin's equivalent.

Calculate the Thevenin's voltage V_{Thev}

$$\begin{aligned} V_{Thev} &= \left(\frac{R_2}{R_1 + R_2}\right) V_{CC} \\ &= \left(\frac{1000}{1200 + 1000}\right)(4.75) \\ &= \frac{4750}{2200} \\ &= 2.16\ \text{V} \end{aligned}$$

Calculate the Thevenin's resistance R_{Thev}

$$\begin{aligned} R_{Thev} &= \frac{R_1 R_2}{R_1 + R_2} \\ &= \frac{(1000)(1200)}{1200 + 1000} \\ &= \frac{1200000}{2200} \\ &= 545.45\ \Omega \end{aligned}$$

Step 31 of 34

Draw the modified circuit with Thevenin equivalent to calculate the sinking current.

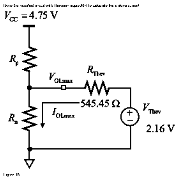


Figure 16

Step 32 of 34

Calculate the sinking current I_{OLmax}

$$\begin{aligned} I_{OLmax} &= \frac{V_{Thev} - V_{OLmax}}{R_{Thev}} \\ &= \frac{2.16 - 0.33}{545.45} \\ &= 0.00335 \\ &= 3.35\ \text{mA} \end{aligned}$$

Therefore, the value of I_{OLmax} is in the commercial range.

Step 33 of 34

Draw the modified circuit with Thevenin equivalent to calculate the sourcing current.

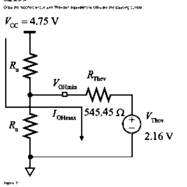


Figure 17

Step 1 of 1

Under any circumstances it is not safe to allow an unused CMOS input to float because the floating input may appear as LOW signal applied to it when it is probed with an oscilloscope or voltmeter. Therefore, we might think that an unused OR or NOR input can be left floating. But, the CMOS inputs have very high impedance, it takes only a small amount of circuit noise to temporarily make a floating input look HIGH.

Therefore, we can conclude that, under any circumstances it is not safe to allow an unused CMOS input to float.

Step 1 of 1

3.29DP

Latch-up problem is nothing but formation of virtual short circuit between power supply & ground.

The following are the circumstances to the occurrence of latch-up problem:

1. When the input voltage is changing from 0V to 5V or either way, there is a possibility of both PMOS and NMOS conducting at the same time, which can be act as “Silicon Controlled Rectifier (SCR)”. This parasitic SCR acts as short-circuit when the input voltage is less than the ground or more than the V_{CC} . Thus, it results creating low resistance path between power supply & ground or we can say that short circuit occurring between power supply & ground.

2. The latch-up problem can also occur when CMOS inputs are driven by the outputs of another system with a separate power supply.

3. The latch-up problem is still possible with the large amount of sourcing current by the driving output.

Step 1 of 1 3.30DP

The decoupling capacitors are added between power supply & ground, in order to avoid the noise on the power supply and ground connections in a CMOS circuit caused by current variations during the CMOS output transitions between LOW and HIGH states. The small size distributed capacitors are faster than large filtering capacitors. Because, the stray wiring inductance prevents the larger capacitors from supplying currents fast enough.

Thus, the larger filter capacitances are replaced with a physically distributed system of decoupling capacitances.

Step 1 of 1

3.31DP

It is important to hold hands with a friend, when a problem cannot be solved individually. Similarly, the decoupling capacitors in a circuit holds together to form a distributed system of large capacitance. Because, the large filtering capacitances are prevented from supplying the currents fast enough due to stray wiring inductance.

Step 1 of 1

3.32DP

The following are the two components that affect the CMOS logic gates delay:

(i) The internal resistance of the transistor

(ii) The capacitance of the transistor

The internal resistance of PMOS and NMOS transistor is given by R_n and R_p. According to the resistance of the circuit, the rise time and fall time varies and creates the delay.

The capacitance in the transistor is considered as AC load. It determines how much time it takes from one logic level to another. Again this cause delay in the circuit.

Step 1 of 4

3.33DP

(a)

Consider the following resistor and capacitor values,

$$R = 100\ \Omega$$
$$C = 50\ \text{pF}$$

Now, calculate the RC time constant of this resistor-capacitor combination.

$$\begin{aligned} RC \text{ time constant} &= RC \\ &= (100\ \Omega)(50\ \text{pF}) \\ &= 5{,}000 \times 10^{-12}\ \text{s} \\ &= 5 \times 10^{-9}\ \text{s} \end{aligned}$$

$$= 5\ \text{ns}$$

Thus, the required RC time constant is $\boxed{5\ \text{ns}}$.

Step 2 of 4

(b)

Consider the following resistor and capacitor values,

$$R = 4.7\ \text{k}\Omega$$
$$C = 150\ \text{pF}$$

Now, calculate the RC time constant of this resistor-capacitor combination.

$$\begin{aligned} RC \text{ time constant} &= RC \\ &= (4.7\ \text{k}\Omega)(150\ \text{pF}) \\ &= \left(4.7 \times 10^{3}\ \Omega\right)\left(150 \times 10^{-12}\ \text{F}\right) \\ &= 705 \times 10^{-9}\ \text{s} \end{aligned}$$

$$= 705\ \text{ns}$$

Thus, the required RC time constant is $\boxed{705\ \text{ns}}$.

Step 3 of 4

(c)

Consider the following resistor and capacitor values,

$$R = 47\ \Omega$$
$$C = 47\ \text{pF}$$

Now, calculate the RC time constant of this resistor-capacitor combination.

$$\begin{aligned} RC \text{ time constant} &= RC \\ &= (47\ \Omega)(47\ \text{pF}) \\ &= 2{,}209 \times 10^{-12}\ \text{s} \\ &= 2.209 \times 10^{-9}\ \text{s} \end{aligned}$$

$$= 2.209\ \text{ns}$$

Thus, the required RC time constant is $\boxed{2.209\ \text{ns}}$.

Step 4 of 4

(d)

Consider the following resistor and capacitor values,

$$R = 1\ \text{k}\Omega$$
$$C = 100\ \text{pF}$$

Now, calculate the RC time constant of this resistor-capacitor combination.

$$\begin{aligned} RC \text{ time constant} &= RC \\ &= (1\ \text{k}\Omega)(100\ \text{pF}) \\ &= \left(1 \times 10^{3}\ \Omega\right)\left(100 \times 10^{-12}\ \text{F}\right) \\ &= 100 \times 10^{-9}\ \text{s} \end{aligned}$$

$$= 100\ \text{ns}$$

Thus, the required RC time constant is $\boxed{100\ \text{ns}}$.

Step 1 of 1

3.34DP

The power consumption of the CMOS circuit is given by,

$$P_T = C_{PD} \cdot V_{CC}^{\,2} \cdot f$$

Where C_{PD} is the capacitance in the circuit, V_{CC} is the power-supply voltage and f is the switching frequency between the logic levels.

From the formula, since $P_T \;\alpha\; C_{PD}$ and $P_T \;\alpha\; V_{CC}^2$, 5% increase in power supply voltage give bigger effect on power consumption that 5% increase in internal and load capacitance .

Step 1 of 1

3.35DP

Fan-out: It is defined as the number of inputs it can able to drive or we can say that number of other gate inputs you can connect to the gate output.

In general CMOS devices are for less power consumption. So when we calculate the DC fan-out for CMOS device which is driving other CMOS, gives more number of devices or we can say that virtually unlimited, because the power consumption for transitions (from HIGH to LOW and from LOW to HIGH) is very less.

Step 1 of 1

74AHC is also called as 74VHC where V stands for 'very'. It can operate with the supply voltage in the range $2-5.5\text{ V}$. So, it is possible to operate 74VHC CMOS devices with a 2.5-volt power supply.

The power consumption of CMOS device can be of two types, static power consumption and dynamic power consumption. Static power consumption occurs during transition. Because at some logic levels, both the transistors gets valid same logic and starts conduct at the same time.

Dynamic power consumption occurs when transitions occur at high frequency since charging and discharging of capacitance takes place. In general the power consumption of CMOS device is given by the formula,

$$P=C\cdot V^2\cdot f$$

Where, *C* is the capacitance that occurs during the transition and capacitive load,

V is the supply voltage and

f is the number of transitions (from low to high & high to low) occur per second.

Thus, if 74VHC CMOS devices are reducing the supply voltage from 5 V to 2.5 V, then power consumption is reduced from $P=25\cdot C\cdot f$ to $P=6.25\cdot C\cdot f$. Since the power consumption is directly proportional to the square of the supply voltage.

The power consumed is $\boxed{\text{4 times}}$ less than actual power consumption.

Step 1 of 1

Consider the following data:

The Low-level input voltage V_{ILmax} is $\mathbf{0.8\ V}$

The High-level input voltage V_{IHmin} is $\mathbf{2.0\ V}$

The switching threshold for negative-going changes V_{T-} is $\mathbf{1.2\ V}$

The switching threshold for positive-going changes V_{T+} is $\mathbf{1.7\ V}$

Calculate the hysteresis.

The difference between two thresholds is called hysteresis.

$$\textbf{hysteresis} = (V_{T+}) - (V_{T-}) \text{ ...... (1)}$$

Substitute $\mathbf{1.7\ V}$ for V_{T+} and $\mathbf{1.2\ V}$ for V_{T-} in equation (1)

$$\begin{aligned}\textbf{hysteresis} &= (1.7) - (1.2)\\ &= 0.5\ \text{V}\end{aligned}$$

Therefore, the Schmitt trigger has $\boxed{0.5\ \text{V}}$ hysteresis.

Step 1 of 1 **3.38DP**

The three state outputs are having three logic or three state, they are LOW (logic 0), HIGH (logic 1) and High Impedance state (Hi-Z) or floating state. The three state outputs can be obtained by extra input and the output is predictable, since third input is controlling the output.

If three state outputs is turned ON faster than they turned OFF, then non-logic voltage is produced on the bus. Because one device's output is enable before the second device's output is disable. Thus two devices are enabling at same time, which in tum creates a leakage current of $\mathbf{10\ \mu A}$ for a moment of time, till it gets normal.

Step 1 of 1

3.39DP

The difference between smaller and larger pull-up resistors for open-drain CMOS outputs are as follows.

Larger pull-up resistors	**Smaller pull-up resistors**
When resistance is more value, then RC value also increases which in turn makes longer rise-time or we can say that slower transition from LOW to HIGH.	The RC value is less when compared with larger one, thus shorter rise-time or we can say that faster transition from LOW to HIGH.
The power consumption is low in LOW state when compared with open state.	The power consumption is high in LOW state when compare with open state.
More space is required, more heat and more price.	Less space is required, less heat and less price when compared with larger pull-up resistor.

Step 1 of 1 3.40DP

Consider the following data:

The value of LED voltage drop, V_{LED} is $\mathbf{2.0\ V}$

The value of LED current, I_{LED} is $\mathbf{5\ mA}$

For 74AC and 74ACT CMOS families, V_{OL} is $\mathbf{0.37\ V}$

Assume the value of the supply voltage, V_{CC} is $\mathbf{5\ V}$

Calculate the pull-up resistor value for LED connected to a 74AC00 NAND gate.

$$R = \frac{V_{\mathrm{CC}} - V_{\mathrm{OL}} - V_{\mathrm{LED}}}{I_{\mathrm{LED}}}$$

Substitute $5\ \mathrm{V}$ for V_{CC}, $0.37\ \mathrm{V}$ for V_{OL}, $5\ \mathrm{mA}$ for I_{LED} and $2.0\ \mathrm{V}$ for V_{LED} in R.

$$\begin{aligned} R &= \frac{5-0.37-2}{5\times10^{-3}} \\ &= \frac{2630}{5} \\ &= 526 \end{aligned}$$

Therefore, the value of the pull-up resistor for LED is $\boxed{526\ \Omega}$.

Step 1 of 1 **3.41DP**

Consider the following data:

The value of LED voltage drop, V_{LED} is 2.0 V

The value of LED current, I_{LED} is 2 mA

For 74HC and 74HCT CMOS families, V_{OL} is 0.33 V

Assume the value of the supply voltage, V_{CC} is 5 V

Calculate the pull-up resistor value for LED connected to a 74HC00 NAND gate.

$$R=\frac{V_{\text{CC}}-V_{\text{OL}}-V_{\text{LED}}}{I_{\text{LED}}}$$

Substitute 5 V for V_{CC}, 0.33 V for V_{OL}, 2 mA for I_{LED} and 2.0 V for V_{LED} in R.

$$\begin{aligned}R&=\frac{5-0.33-2}{2\times10^{-3}}\\&=\frac{2670}{2}\\&=1335\ \Omega\\&=1.335\text{ k}\Omega\end{aligned}$$

Therefore, the value of the pull-up resistor for LED is $\boxed{1.335\text{ k}\Omega}$.

Step 1 of 1

3.42DP

A wired-AND function is obtained simply by tying two or more open-drain or open-collector output together, without going through another level of transistor circuitry but we need additional pull-up resistor to drive the load.

A wired-AND logic is nothing but the outputs of many open drain are connected together such that to form a AND logic with single pull-up resistor. Thus larger pull-up resistance is required to drive the load which in tum increases the RC time constant and longer rise time.

So, as a result the wired-AND logic is slower than a discrete AND logic.

Step 1 of 1

3.43DP

As per the context, FCT and FCT-T family has the strongest output driving capability. It can sink and sourcing current a lot up to 64mA and -15mA. It drives heavy DC load and also TTL devices. It is having very fast rise and fall time and it is controlled. The FCT-T family is reduced output High level voltage, thereby reducing power consumption since power consumption is directly proportional to the square supply voltage.

Therefore, TTL logic family has the strongest output driving capability.

Step 1 of 1 **3.44DP**

The difference between ACT and HC logic families are given as follows.

ACT logic families	**HC logic families**
More sink and source current of 24mA when compared with HC logic families.	Sink and Source current is less when compared with ACT logic families and it is 4mA.
Ability to drive heavy DC loads, including TTL loads.	Ability to drive loads but not as heavy as ACT logic families.
TTL compatibility.	Not TTL compatibility.
Very fast rise and fall time.	Slower rise and fall time compared with ACT.
Price is high; since it is fast propagation delay is less.	Price is less; propagation delay is more compared with ACT.

Step 1 of 1 3.45DP

The main advantage of FCT to meet the speed and output drive capability of TTL families. FCT family has the drawback of high power dissipation while producing high 5V CMOS, V_{OH} and circuit noise as the output swings from 0 V to 5 V. Later FCT-T was introduced with reduced level of V_{OH} and it is compatible with TTL loads.

Hence, the FCT devices do not include some output parameters with CMOS load.

Step 1 of 1

3.46DP

The High-level output voltage is $5\ \mathbf{V}$ for FCT. Thus power consumption is more since power consumption is directly proportional to square of the voltage. Whereas the FCT-T has reduced its High-level output voltage to $3.3\ \mathbf{V}$, thus the power consumption is less and noise also reduced since the FCT-T is having faster transition time.

Therefore, by reducing the High-level output voltage, FCT-T had reduced the power consumption compared to FCT devices.

Step 1 of 2

Draw the following n-input diode AND gate.

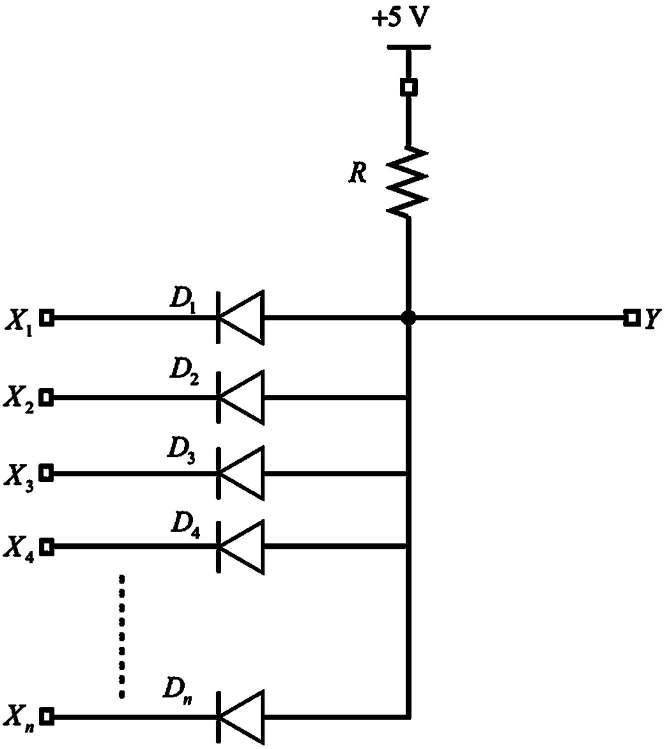


Figure 1

Step 2 of 2

From Figure 1, $X_1, X_2, X_3, X_4 \cdots X_n$ are *n*- number of inputs and Y is the output of *n*-input diode AND gate.

For *n*-input diode AND gate *n*- number of diodes are required.

Therefore, the number of diodes required for an *n*-input diode AND gate are $\boxed{n}$.

Step 1 of 1 **3.48DP**

The case where the current flows into a TTL output in the LOW state is called sinking current.

The case where the current flows into a TTL output in the HIGH state is called sinking current.

TTL outputs are more capable of both sourcing and sinking current. But when we compare, TTL output is having more sinking current than sourcing current.

3.49DP

Step 1 of 15

The LOW-state and High-state fan out can be calculated by using the following formulae:

(i) The LOW-state $\textbf{fanout} = \frac{I_{OLmax}}{I_{ILmax}}$, where I_{OLmax} is the the maximum output current of the TTL which is used to drive another gate or we can say that given as input to another gate and I_{ILmax} is equal to the maximum input current required to pull it LOW.

(ii) The HIGH-state $\textbf{fanout} = \frac{I_{OHmax}}{I_{IHmax}}$, where I_{OHmax} is the maximum output current of the TTL which is used to drive another gate or we can say that given as input to another gate and I_{IHmax} is the maximum input current required to pull it HIGH.

(iii) The Overall fan out is minimum of the LOW-state and HIGH-state fanouts.

Step 2 of 15

(a)

Determine the LOW-state fanout for 74LS driving 74AS.

The LOW-state $\textbf{fanout} = \frac{I_{OLmax}}{I_{ILmax}}$

The I_{OLmax} of 74LS is given by 8 mA and I_{ILmax} of 74AS is given by 0.5 mA which are required to operate it.

Substitute 8 for I_{OLmax} and 0.5 for I_{ILmax} .

$$\begin{aligned}\textbf{The LOW-state fanout} &= \frac{I_{OLmax}}{I_{ILmax}} \\ &= \frac{8}{0.5} \\ &= 16\end{aligned}$$

Thus, the LOW-state fanout is 16.

Step 3 of 15

Determine the HIGH-state fanout for 74LS driving 74AS.

$$\textbf{HIGH-state fanout} = \frac{I_{OHmax}}{I_{IHmax}}$$

The I_{OHmax} of 74LS is given by 400 μA and I_{IHmax} of 74AS is given by 20 μA which are required to operate it.

Substitute 400 for I_{OHmax} and 20 for I_{IHmax} .

$$\begin{aligned}\textbf{HIGH-state fanout} &= \frac{I_{OHmax}}{I_{IHmax}} \\ &= \frac{400}{20} \\ &= 20\end{aligned}$$

Thus, the HIGH-state fanout is 20.

The maximum fanout is the minimum of LOW state fan-out and HIGH state fan-outs, that is, 16.

Thus, the maximum fanout of 74LS driving 74F is $\boxed{16}$.

The excess current is in HIGH state and value is $\boxed{80\ \mu\text{A}}$.

Step 4 of 15

(b)

Determine the LOW-state fanout for 74LS driving 74F.

The LOW-state $\textbf{fanout} = \frac{I_{OLmax}}{I_{ILmax}}$

The I_{OLmax} of 74LS is given by 8 mA and I_{ILmax} of 74F is given by 0.6 mA which are required to operate it.

Substitute 8 for I_{OLmax} and 0.5 for I_{ILmax} .

$$\begin{aligned}\textbf{The LOW-state fan out} &= \frac{I_{OLmax}}{I_{ILmax}} \\ &= \frac{8}{0.6} \\ &= 13\end{aligned}$$

Thus, the LOW-state fanout is 13.

Step 5 of 15

Determine the HIGH-state fanout for 74LS driving 74F

$$\textbf{HIGH-state fanout} = \frac{I_{OHmax}}{I_{IHmax}}$$

The I_{OHmax} of 74LS is given by 400 μA and I_{IHmax} of 74AS is given by 20 μA which are required to operate it.

Substitute 400 for I_{OHmax} and 20 for I_{IHmax} .

$$\begin{aligned}\textbf{HIGH-state fanout} &= \frac{I_{OHmax}}{I_{IHmax}} \\ &= \frac{400}{20} \\ &= 20\end{aligned}$$

Thus, the HIGH-state fanout is 20.

The maximum fanout is the minimum of LOW state fan-out and HIGH state fan-outs, that is, 13.

Thus, the maximum fanout of 74F driving 74LS is $\boxed{13}$.

The excess current is in HIGH state and value is $\boxed{140\ \mu\text{A}}$

Step 6 of 15

(c)

Determine the LOW-state fanout for 74F driving 74LS.

The LOW-state $\textbf{fanout} = \frac{I_{OLmax}}{I_{ILmax}}$

The I_{OLmax} of 74F is given by 20 mA and I_{ILmax} of 74LS is given by 0.4 mA, which are required to operate it.

Substitute 20 for I_{OLmax} and 0.4 for I_{ILmax} .

$$\begin{aligned}\textbf{The LOW-state fan out} &= \frac{I_{OLmax}}{I_{ILmax}} \\ &= \frac{20}{0.4} \\ &= 50\end{aligned}$$

Thus, the LOW-state fanout is 50.

Step 7 of 15

Determine the HIGH-state fanout for 74F driving 74LS.

$$\textbf{HIGH-state fanout} = \frac{I_{OHmax}}{I_{IHmax}}$$

The I_{OHmax} of 74F is given by 1000 μA and I_{IHmax} of 74LS is given by 20 μA which are required to operate it.

Substitute 1000 for I_{OHmax} and 20 for I_{IHmax} .

$$\begin{aligned}\textbf{HIGH-state fanout} &= \frac{I_{OHmax}}{I_{IHmax}} \\ &= \frac{1000}{20} \\ &= 50\end{aligned}$$

Thus, the HIGH-state fanout is 50.

The maximum fanout is the minimum of LOW state fan-out and HIGH state fan-outs, that is, 50.

Thus, the maximum fanout of 74F driving 74LSis $\boxed{50}$.

The excess current is $\boxed{0\ \mu\text{A}}$.

Step 8 of 15

(d)

Determine the LOW-state fanout for 74F driving 74AS.

The LOW-state $\textbf{fanout} = \frac{I_{OLmax}}{I_{ILmax}}$

The I_{OLmax} of 74F is given by 20 mA and I_{ILmax} of 74AS is given by 0.5 mA, which are required to operate it.

Substitute 20 for I_{OLmax} and 0.5 for I_{ILmax} .

$$\begin{aligned}\textbf{The LOW-state fan out} &= \frac{I_{OLmax}}{I_{ILmax}} \\ &= \frac{20}{0.5} \\ &= 40\end{aligned}$$

Thus, the LOW-state fanout is 40.

Step 9 of 15

Determine the HIGH-state fanout for 74F driving 74AS.

$$\textbf{HIGH-state fanout} = \frac{I_{OHmax}}{I_{IHmax}}$$

The I_{OHmax} of 74F is given by 1000 μA and I_{IHmax} of 74AS is given by 20 μA which are required to operate it.

Substitute 1000 for I_{OHmax} and 20 for I_{IHmax} .

$$\begin{aligned}\textbf{HIGH-state fanout} &= \frac{I_{OHmax}}{I_{IHmax}} \\ &= \frac{1000}{20} \\ &= 50\end{aligned}$$

Thus, the HIGH-state fanout is 50.

The maximum fanout is the minimum of LOW state fan-out and HIGH state fan-outs, that is, 40.

Thus, the maximum fanout of 74F driving 74AS is $\boxed{40}$.

The excess current is in HIGH state and value is $\boxed{200\ \mu\text{A}}$.

(e)

Determine the LOW-state fanout for 74AS driving 74S.

The LDW-state $\textbf{fanout} = \frac{I_{OLmax}}{I_{ILmax}}$

The I_{OLmax} of 74AS is given by 20 mA and I_{ILmax} of 74S is given by 2 mA, which are required to operate it.

Substitute 20 for I_{OLmax} and 2 for I_{ILmax} .

$$\begin{aligned}\textbf{The LOW-state fan out} &= \frac{I_{OLmax}}{I_{ILmax}} \\ &= \frac{20}{2} \\ &= 10\end{aligned}$$

Thus, the LOW-state fanout is 10.

Step 10 of 15

Determine the HIGH-state fanout for 74AS driving 74S.

$$\textbf{HIGH-state fanout} = \frac{I_{OHmax}}{I_{IHmax}}$$

The I_{OHmax} of 74AS is given by 2000 μA and I_{IHmax} of 74S is given by 50 μA which are required to operate it.

Substitute 2000 for I_{OHmax} and 50 for I_{IHmax} .

$$\begin{aligned}\textbf{HIGH-state fanout} &= \frac{I_{OHmax}}{I_{IHmax}} \\ &= \frac{2000}{50} \\ &= 40\end{aligned}$$

Thus, the HIGH-state fanout is 40.

The maximum fanout is the minimum of LOW state fan-out and HIGH state fan-outs, that is, 10.

Thus, the maximum fanout of 74F driving 74AS is $\boxed{10}$.

The excess current is in HIGH state and value is $\boxed{1500\ \mu\text{A}}$.

Step 11 of 15

(f)

Determine the LOW-state fanout for 74S driving 74ALS.

The LOW-state $\textbf{fanout} = \frac{I_{OLmax}}{I_{ILmax}}$

The I_{OLmax} of 74S is given by 20 mA and I_{ILmax} of 74ALS is given by 0.2 mA, which are required to operate it.

Substitute 20 for I_{OLmax} and 0.2 for I_{ILmax} .

$$\begin{aligned}\textbf{The LOW-state fan out} &= \frac{I_{OLmax}}{I_{ILmax}} \\ &= \frac{20}{0.2} \\ &= 100\end{aligned}$$

Thus, the LOW-state fanout is 100.

Step 12 of 15

Determine the HIGH-state fanout for 74AS driving 74S.

$$\textbf{HIGH-state fanout} = \frac{I_{OHmax}}{I_{IHmax}}$$

The I_{OHmax} of 74S is given by 1000 μA and I_{IHmax} of 74ALS is given by 20 μA which are required to operate it.

Substitute 1000 for I_{OHmax} and 20 for I_{IHmax} .

$$\begin{aligned}\textbf{HIGH-state fanout} &= \frac{I_{OHmax}}{I_{IHmax}} \\ &= \frac{1000}{20} \\ &= 50\end{aligned}$$

Thus, the HIGH-state fanout is 50.

The maximum fanout is the minimum of LOW state fan-out and HIGH state fan-outs, that is, 50.

Thus, the maximum fanout of 74F driving 74AS is $\boxed{50}$.

The excess current is in LOW state and value is $\boxed{10\ \text{mA}}$.

Step 13 of 15

(g)

Determine the LOW-state fanout for 74ALS driving 74S.

The LOW-state $\textbf{fanout} = \frac{I_{OLmax}}{I_{ILmax}}$

The I_{OLmax} of 74ALS is given by 8 mA and I_{ILmax} of 74S is given by 2 mA, which are required to operate it.

Substitute 8 for I_{OLmax} and 2 for I_{ILmax} .

$$\begin{aligned}\textbf{The LOW-state fan out} &= \frac{I_{OLmax}}{I_{ILmax}} \\ &= \frac{8}{2} \\ &= 4\end{aligned}$$

Thus, the LOW-state fanout is 4.

Determine the HIGH-state fanout for 74ALS driving 74S.

$$\textbf{HIGH-state fanout} = \frac{I_{OHmax}}{I_{IHmax}}$$

The I_{OHmax} of 74ALS is given by 400 μA and I_{IHmax} of 74S is given by 50 μA which are required to operate it.

Substitute 400 for I_{OHmax} and 50 for I_{IHmax} .

$$\begin{aligned}\textbf{HIGH-state fanout} &= \frac{I_{OHmax}}{I_{IHmax}} \\ &= \frac{400}{50} \\ &= 8\end{aligned}$$

Thus, the HIGH-state fanout is 8.

The maximum fanout is the minimum of LOW state fan-out and HIGH state fan-outs, that is, 4.

Thus, the maximum fanout of 74F driving 74AS is $\boxed{4}$.

The excess current is in HIGH state and value is $\boxed{200\ \mu\text{A}}$.

Step 14 of 15

(h)

Determine the LOW-state fanout for 74F driving 74F.

The LDW-state $\textbf{fanoul} = \frac{I_{OLmax}}{I_{ILmax}}$

The I_{OLmax} of 74F is 20 mA and I_{ILmax} of 74F is given by 0.6 mA, which are required to operate it.

Substitute 20 for I_{OLmax} and 0.6 for I_{ILmax} .

$$\begin{aligned}\textbf{The LOW-state fan out} &= \frac{I_{OLmax}}{I_{ILmax}} \\ &= \frac{20}{0.6} \\ &= 33\end{aligned}$$

Thus, the LOW-state fanout is 33.

The excess driving capable is 0.2mA.

Step 15 of 15

Determine the HIGH-state fanout for 74F driving 74F.

$$\textbf{HIGH-state fanout} = \frac{I_{OHmax}}{I_{IHmax}}$$

The I_{0Hmax} of 74F is given by 1000 μA and I_{IHmax} of 74F is given by 20 μA which are required to operate it.

Substitute 1000 for I_{OHmax} and 20 for I_{IHmax} .

$$\begin{aligned}\textbf{HIGH-state fanout} &= \frac{I_{OHmax}}{I_{IHmax}} \\ &= \frac{1000}{20} \\ &= 50\end{aligned}$$

Thus, the HIGH-state fanout is 50.

The maximum fanout is the minimum of LOW state fan-out and HIGH state fan-outs, that is, 33.

Thus, the maximum fanout of 74F driving 74AS is $\boxed{33}$.

The excess current is in HIGH state and value is $\boxed{340\ \mu\text{A}}$.

Step 1 of 2 **3.50DP**

The maximum allowable value of the pull up resistor for an unused LS-TTL NAND gate input is calculated in the HIGH state and it is the ratio of the minimum input voltage to the maximum input current in the High state.

Determine maximum value of the pull up resistor for an unused LS-TTL NAND gate.

$$\begin{aligned}R_P &= \frac{V_{\text{IH min}}}{I_{\text{IH max}}}\\ &= \frac{2\text{ V}}{20\ \mu\text{A}} \quad \left(LS-\text{series family values from reference of table 3-10}\right)\\ &= \frac{2.0}{20\times10^{-6}}\\ &= 100\text{ k}\Omega\end{aligned}$$

Calculate the power dissipated in pull up resistor using the following formula.

$$\begin{aligned}P &= \frac{V^2_{\text{IH min}}}{R_P}\\ &= \frac{2^2}{100\times10^3}\\ &= 0.04\text{ mW}\end{aligned}$$

Step 2 of 2

The maximum allowable value of the pull down resistor for an unused LS-TTL NOR gate input is calculated in the LOW state and it is the ratio of the maximum input voltage to the maximum input current in the LOW state.

Determine maximum value of the pull down resistor for an unused LS-TTL NOR gate.

$$\begin{aligned}R_N &= \frac{V_{IL\max}}{I_{IL\max}} \quad \left(LS-\text{series family values from reference of table 3-10}\right)\\ &= \frac{0.8\text{ V}}{0.4\text{ mA}}\\ &= \frac{0.8}{0.4\times10^{-3}}\\ &= 2\text{ k}\Omega\end{aligned}$$

Calculate the power dissipated in pull down resistor using the following formula.

$$\begin{aligned}P &= \frac{V^2_{\text{IL max}}}{R_N}\\ &= \frac{(0.8)^2}{2\times10^3}\\ &= 0.32\text{ mW}\end{aligned}$$

Thus, when comparing the values of power dissipated by both resistors, pull down resistor for an unused **LS-TTL NOR** dissipates more power when compared with the pull up resistor unused LS-TTL NAND gate.

Step 1 of 1 **3.51DP**

When we compare CMOS AND gate and CMOS AND-OR-INVERT gate, the second one is expected to be faster because we require two more transistors for AND gate to get the inverted output.

For example, 3 input AND-OR-INVERT needs 6 transistors for performing the logic whereas 3-input CMOS AND gate needs 6 transistors for NAND logic and 2 more for inverting that, so totally 8 transistors are required for a 3-input AND gate.

Thus, depending on the number of transistors required, **AND-OR-INVERT** is faster than **AND**.

Step 1 of 1

3.52DP

Schottky transistor is a combination of transistor and schottky diode. The following are the key benefit of schottky transistor:

• Prevents the transistor from saturating; it reducing excess amount of current (sink and source current) or power consumption.

• It is very fast when compared with other device producing more switching speed; further propagation delay time is decreasing.

• The place occupied for the fabrication or size of schottky transistor is small.

Step 1 of 4

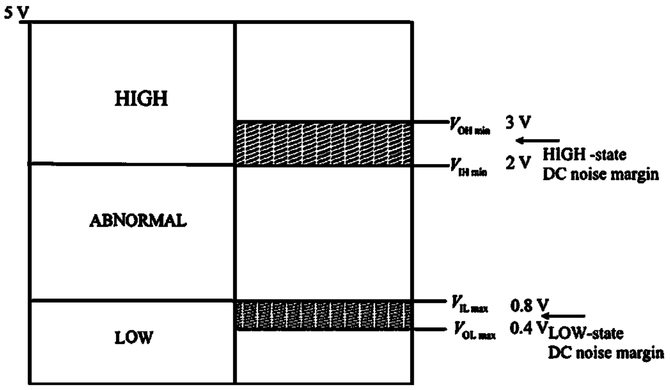
According to www.ti.com the electrical parameter values for 74ALS00-Quadruple 2-input positive NAND gate are:

Electrical Parameters	**values**
HIGH-level output voltage, V_{OHmin}	3 V
HIGH-level input voltage, V_{IHmin}	2 V
LOW-level output voltage, V_{OLmax}	0.4 V
LOW-level input voltage, V_{ILmax}	0.8 V
HIGH-level input current, I_{IHmax}	20 μA
LOW-level input current, I_{ILmax}	−100 μA
LOW-level output current, I_{OLmax}	8 mA
HIGH-level output current, I_{OHmax}	−0.4 mA

Table 1: The Electrical parameters of 74ALS00

Step 2 of 4

DC Noise margin of 74ALS00:



5 V
HIGH
ABNORMAL
LOW
$V_{\mathrm{OH\,min}}$ 3 V
$V_{\mathrm{IH\,min}}$ 2 V
HIGH -state DC noise margin
$V_{\mathrm{IL\,max}}$ 0.8 V
$V_{\mathrm{OL\,max}}$ 0.4 V
LOW-state DC noise margin

Figure 1

Step 3 of 4

Calculate the High state DC noise margin.

$$\begin{aligned}\text{High state DC Noise margin} &= V_{\mathrm{OHmin}} - V_{\mathrm{IHmin}} \\ &= 3-2 \\ &= 1\,\mathrm{V}\end{aligned}$$

Thus, the High state DC noise margin of 74ALS00 is $\boxed{1\ \mathrm{V}}$.

Step 4 of 4

Calculate the Low state DC noise margin.

$$\begin{aligned}\text{Low state DC Noise margin} &= V_{\mathrm{ILmax}} - V_{\mathrm{OLmax}} \\ &= 0.8-0.4 \\ &= 0.4\,\mathrm{V}\end{aligned}$$

Thus, the LOW state DC noise margin of 74ALS00 is $\boxed{0.4\ \mathrm{V}}$.

Step 1 of 8 **3.54DP**

The following are the eight electrical parameters for Transistor-Transistor Logic (TTL):

$V_{\mathrm{OH\,min}}$: The minimum output voltage that is guaranteed to be recognized as a HIGH, and its value **2.7 V** for most of the TTL circuits.

Step 2 of 8

V_{IHmin} : The minimum input voltage that is guaranteed to be recognized as a HIGH, and its value **2.0 V** for most of the TTL circuits.

Step 3 of 8

V_{ILmax} : The maximum input voltage that is guaranteed to be recognized as a LOW, and its value **0.8 V** for most of the TTL circuits.

Step 4 of 8

V_{OLmax} : The maximum input voltage in the LOW state, and its value **0.5 V** for most of the TTL circuits.

Step 5 of 8

I_{ILmax} : The maximum current that an input to the TTL requires to pull it LOW state and it has negative value of $\mathbf{0.4\,mA}$. I_{IHmax} : The maximum input current required that is given to the TTL, hence to pull it to the HIGH state, its value is $\mathbf{20\,\mu A}$.

Step 6 of 8

I_{OLmax} : The maximum current an output can sink in the LOW state while maintaining an output voltage no more than V_{OLmax} . For TTL circuits its value, $\mathbf{8\,mA}$.

Step 7 of 8

I_{OHmax} : The maximum current an output can source in the HIGH state while maintaining an output voltage no more than V_{OHmin} . For TTL circuits its value, $\mathbf{-400\,\mu A}$.

Step 8 of 8

According to www.ti.com the electrical parameter values for 74ALS00-Quadruple 2-input positive NAND gate are:

Electrical Parameters	**values**
HIGH-level output voltage, V_{OHmin}	$\mathbf{3\,V}$
HIGH-level input voltage, V_{IHmin}	$\mathbf{2\,V}$
LOW-level output voltage, V_{OLmax}	$\mathbf{0.4\,V}$
LOW-level input voltage, V_{ILmax}	$\mathbf{0.8\,V}$
HIGH-level input current, I_{IHmax}	$\mathbf{20\,\mu A}$
LOW-level input current, I_{ILmax}	$\mathbf{-100\,\mu A}$
LOW-level output current, I_{OLmax}	$\mathbf{8\,mA}$
HIGH-level output current, I_{OHmax}	$\mathbf{-0.4\,mA}$

Table 1: The Electrical parameters of 74ALS00

3.55DP

Step 1 of 18

As there are two resistances added in the current flow path, the current and voltage at the output side affected. **If the percentage change in the resistance is small while calculated on the added series resistance to the total resistances before adding any resistances then neglect the affected voltage and current on the output side.**

Refer Table 3-11 in text book, for the operating conditions of the 74LS00.

Of the all output parameters, most affected output parameters are I_{OL} and I_{OH} .

Step 2 of 18

Refer Figure 3-73 in text book, one TTL gate is driving a low input of other gate. Hence, the total resistance of I_{OL} is calculated from this circuit.

From Figure 3-73, calculate the total resistance in the current flow path tiefore adding any resistance.

$$(20K\|8K)+(1.5K\|3K\|0)$$
$$\Rightarrow \frac{(20K)(8K)}{28K}+0$$
$$\Rightarrow \frac{160K}{28}$$
$$\Rightarrow 5.71\ K\Omega$$

Step 3 of 18

Refer Figure 3-74 in text book, one TTL gate is driving a high input of other gate. Hence, the total resistance of I_{OH} is calculated from this circuit.

From Figure 3-74, calculate the total resistance in the current flow path tiefore adding any resistance.

$$(8K\|120\|20K\|8K)+(12K\|0)+1.5K$$
$$\Rightarrow (0.118K\|5.71K)+0+1.5K$$
$$\Rightarrow 0.1156K+1.5K$$
$$\Rightarrow 1.6156\ K\Omega$$
$$\Rightarrow 1.62\ K\Omega$$

If the percentage is appreciatile (above 10%) then it is known that affect the parameters otherwise not.

Step 4 of 18

(a)

Calculate the percentage change in the resistance with respect to I_{OL} .

$$\left(\frac{470}{5.71K}\right)\times 100\% = 8.2\%$$

The percentage change in the resistance is less than 10%. Hence, the resistive load does not affect I_{OL} .

Calculate the percentage change in the resistance with respect to I_{OH} .

$$\left(\frac{470}{1.62K}\right)\times 100\% = 29\%$$

The percentage change in the resistance is greater than 10%. Hence, the resistive load affects I_{OH} value.

Calculate the new value of I_{OH} , which is 71% of I_{OH} .

$$\begin{aligned} I_{OH} &= 0.71(-0.4\ \text{mA}) \\ &= -0.284\ \text{mA} \end{aligned}$$

Thus, the output drive parameter I_{OH} is changed and its new value is $-0.284\ \text{mA}$.

Step 5 of 18

(b)

Calculate the percentage change in the resistance with respect to I_{OL} .

$$\left(\frac{470+330}{5.71K}\right)\times 100\% = 14\%$$

The percentage change in the resistance is greater than 10%. Hence, the resistive load affect I_{OL} .

Calculate the new value of I_{OL} , which is 86% of I_{OL} .

$$\begin{aligned} I_{OL} &= 0.86(8\ \text{mA}) \\ &= 6.88\text{mA} \end{aligned}$$

Step 6 of 18

Calculate the percentage change in the resistance with respect to I_{OH} .

$$\left(\frac{470+330}{1.62K}\right)\times 100\% = 49.39\%$$

The percentage change in the resistance is greater than 10%. Hence, the resistive load affects.

Calculate the new value of I_{OH} , which is 51% of I_{OH} .

$$\begin{aligned} I_{OH} &= 0.51(-0.4\ \text{mA}) \\ &= -0.204\ \text{mA} \end{aligned}$$

Thus, the output drive parameter I_{OH} is changed and its new value is $-0.204\ \text{mA}$.

Step 7 of 18

(c)

Calculate the percentage change in the resistance with respect to I_{OL} .

$$\left(\frac{6.8K}{5.71K}\right)\times 100\% = 119\%$$

The percentage change in the resistance is greater than 10%. Hence, the resistive load affect I_{OL} .

The new value of I_{OL} , which is -19% of I_{OL} , is practically zero.

Step 8 of 18

Calculate the percentage change in the resistance with respect to I_{OH} .

$$\left(\frac{6.8K}{1.62K}\right)\times 100\% = 419.75\%$$

Thus I_{OH} would tie practically equal to zero.

Thus, the output drive parameters I_{OH} and I_{OL} are changed.

Step 9 of 18

(d)

Calculate the percentage change in the resistance with respect to I_{OL} .

$$\left(\frac{910+1200}{5.71K}\right)\times 100\% = 36.95\%$$

The percentage change in the resistance is greater than 10%. Hence, the resistive load affect I_{OL} .

The new value of I_{OL} , which is -19% of I_{OL} , is practically zero.

Step 10 of 18

Calculate the percentage change in the resistance with respect to I_{OH} .

$$\left(\frac{910+1200}{1.62K}\right)\times 100\% = 130.02\%$$

The percentage change in the resistance is greater than 10%. Hence, the resistive load affect I_{OH} . Thus I_{OH} would be practically equal to zero.

Thus, the output drive parameters I_{OH} and I_{OL} are changed.

Step 11 of 18

(e)

Calculate the percentage change in the resistance with respect to I_{OL} .

$$\left(\frac{620}{5.71K}\right)\times 100\% = 10.86\%$$

The percentage change in the resistance is around 10%. Hence, the resistive load not affect I_{OL} .

Step 12 of 18

Calculate the percentage change in the resistance with respect to I_{OH} .

$$\left(\frac{620}{1.62K}\right)\times 100\% = 38.27\%$$

The percentage change in the resistance is greater than 10%. Hence, the resistive load affects.

Calculate the new value of I_{OH} , which is 62% of I_{OH} .

$$\begin{aligned} I_{OH} &= 0.62(-0.4\ \text{mA}) \\ &= -0.248\ \text{mA} \end{aligned}$$

Thus, the output drive parameter I_{OH} is changed and its new value is $-0.248\ \text{mA}$.

Step 13 of 18

(f)

Calculate the percentage change in the resistance with respect to I_{OL} .

$$\left(\frac{470+510}{5.71K}\right)\times 100\% = 17.16\%$$

The percentage change in the resistance is greater than 10%. Hence, the resistive load affect I_{OL} .

Calculate the new value of I_{OL} , which is 82.84% of I_{OL} .

$$\begin{aligned} I_{OL} &= 0.83(8\ \text{mA}) \\ &= 6.64\ \text{mA} \end{aligned}$$

Step 14 of 18

Calculate the percentage change in the resistance with respect to I_{OH} .

$$\left(\frac{510+470}{1.62K}\right)\times 100\% = 60.49\%$$

The percentage change in the resistance is greater than 10%. Hence, the resistive load affects.

Calculate the new value of I_{OH} , which is 39.51% of I_{OH} .

$$\begin{aligned} I_{OH} &= 0.4(-0.4\ \text{mA}) \\ &= -0.16\ \text{mA} \end{aligned}$$

Thus, the output drive parameters I_{OH} and I_{OL} are exceeded their operating range.

Step 15 of 18

(g)

Calculate the percentage change in the resistance with respect to I_{OL} .

$$\left(\frac{5.1K}{5.71K}\right)\times 100\% = 89.31\%$$

The percentage change in the resistance is greater than 10%. Hence, the resistive load affect I_{OL} .

Calculate the new value of I_{OL} , which is 10.69% of I_{OL} .

$$\begin{aligned} I_{OL} &= 0.106(8\ \text{mA})I_{OL} \\ &= 0.848\ \text{mA} \end{aligned}$$

Step 16 of 18

Calculate the percentage change in the resistance with respect to I_{OH} .

$$\left(\frac{5.1K}{1.62K}\right)\times 100\% = 314.81\%$$

The percentage change in the resistance is greater than 10%. Hence, the resistive load affects.

Thus I_{OH} would be practically equal to zero.

Thus, the output drive parameters I_{OH} and I_{OL} are exceeded their operating range.

Step 17 of 18

(h)

Calculate the percentage change in the resistance with respect to I_{OL} .

$$\left(\frac{464+510}{5.71K}\right)\times 100\% = 17.05\%$$

The percentage change in the resistance is greater than 10%. Hence, the resistive load affect I_{OL} .

Calculate the new value of I_{OL} , which is 82.95% of I_{OL} .

$$\begin{aligned} I_{OL} &= 0.83(8\ \text{mA})I_{OL} \\ &= 6.64\ \text{mA} \end{aligned}$$

Step 18 of 18

Calculate the percentage change in the resistance with respect to I_{OH} .

$$\left(\frac{510+464}{1.62K}\right)\times 100\% = 60.12\%$$

The percentage change in the resistance is greater than 10%. Hence, the resistive load affects.

Calculate the new value of I_{OH} , which is 39.88% of I_{OH} .

$$\begin{aligned} I_{OH} &= 0.4(-0.4\ \text{mA}) \\ &= -0.16\ \text{mA} \end{aligned}$$

Thus, the output drive parameters I_{OH} and I_{OL} are exceeded their operating range.

Step 1 of 5

3.56DP

The DC Noise margin is given by the following formula:

$$\textbf{High state DC Noise margin} = V_{OHmin} - V_{IHmin}$$

Here,

V_{OHmin} is the driving output minimum high voltage and

V_{IHmin} is the driven input minimum high voltage

$$\textbf{Low state DC Noise margin} = V_{ILmax} - V_{OLmax}$$

Here,

V_{OLmax} is the driving output maximum low voltage and

V_{ILmax} is the driven input maximum low voltage

Step 2 of 5

(a)

The voltage, V_{ILmax} of 74LS is 0.8 V

The voltage, V_{IHmin} of 74LS is 2.0 V

The voltage, V_{OLmax} of 74HCT is 0.33 V

The voltage, V_{OHmin} of 74HCT is 84 V

Determine the noise margins of 74HCT driving 74LS

$$\begin{aligned}\textbf{Low state DC Noise margin} &= V_{ILmax} - V_{OLmax}\\ &= 0.8 - 0.33\\ &= 0.47\text{ V}\end{aligned}$$

$$\begin{aligned}\textbf{High state DC Noise margin} &= V_{OHmin} - V_{IHmin}\\ &= 3.84 - 2.0\\ &= 1.84\text{ V}\end{aligned}$$

Thus, the LOW-state noise margin is $\boxed{0.47\text{ V}}$ and the HIGH-state noise margin is $\boxed{1.84\text{ V}}$.

Step 3 of 5

(b)

The voltage, V_{ILmax} of 74HCT is 0.8 V

The voltage, V_{IHmin} of 74HCT is 2.0 V

The voltage, V_{OLmax} of 74 ALS is 0.5 V

The voltage, V_{OHmin} of 74 ALS is 2.7 V

$$\begin{aligned}\textbf{Low state DC Noise margin} &= V_{ILmax} - V_{OLmax}\\ &= 0.8 - 0.5\\ &= 0.3\text{ V}\end{aligned}$$

$$\begin{aligned}\textbf{High state DC Noise margin} &= V_{OHmin} - V_{IHmin}\\ &= 2.7 - 2.0\\ &= 0.7\text{ V}\end{aligned}$$

Thus, the LOW-state noise margin is $\boxed{0.3\text{ V}}$ and the HIGH-state noise margin is $\boxed{0.7\text{ V}}$.

Step 4 of 5

(c)

The voltage, V_{ILmax} of 74VHCT is 0.8 V

The voltage, V_{IHmin} of 74VHCT is 2.0 V

The voltage, V_{OLmax} of 74AS is 0.5 V

The voltage, V_{OHmin} of 74AS is 2.7 V

$$\begin{aligned}\textbf{Low state DC Noise margin} &= V_{ILmax} - V_{OLmax}\\ &= 0.8 - 0.5\\ &= 0.3\text{V}\end{aligned}$$

$$\begin{aligned}\textbf{High state DC Noise margin} &= V_{OHmin} - V_{IHmin}\\ &= 2.7 - 2.0\\ &= 0.7\text{V}\end{aligned}$$

Thus, the LOW-state noise margin is $\boxed{0.3\text{ V}}$ and the HIGH-state noise margin is $\boxed{0.7\text{ V}}$.

Step 5 of 5

(d)

The voltage, V_{ILmax} of 74F is 0.8 V

The voltage, V_{IHmin} of 74F is 2.0 V

The voltage, V_{OLmax} of 74VHCT is 0.44 V

The voltage, V_{OHmin} of 74VHCT is 3.80 V

$$\begin{aligned}\textbf{Low state DC Noise margin} &= V_{ILmax} - V_{OLmax}\\ &= 0.8 - 0.44\\ &= 0.36\text{V}\end{aligned}$$

$$\begin{aligned}\textbf{High state DC Noise margin} &= V_{OHmin} - V_{IHmin}\\ &= 3.8 - 2.0\\ &= 1.8\text{V}\end{aligned}$$

Thus, the LOW-state noise margin is $\boxed{0.36\text{ V}}$ and the HIGH-state noise margin is $\boxed{1.8\text{ V}}$.

Step 1 of 9 **3.57DP**

The LOW-state and High-state fan out can be calculated by using the following formulae:

(i) The LOW-state $\textbf{fanout} = \frac{I_{OLmax}}{I_{ILmax}}$, where I_{OLmax} is the the maximum output current of the TTL which is used to drive another gate or we can say that given as input to another gate and I_{ILmax} is equal to the maximum input current required to pull it LOW.

(ii) The HIGH-state $\textbf{fanout} = \frac{I_{OHmax}}{I_{IHmax}}$, where I_{OHmax} is the maximum output current of the TTL which is used to drive another gate or we can say that given as input to another gate and I_{IHmax} is the maximum input current required to pull it HIGH.

(iii) The Overall fan out is minimum of the LOW-state and HIGH-state fanouts.

Step 2 of 9

(a)

Determine the LOW-state fanout for 74HCT driving 74LS.

The LOW-state $\textbf{fanout} = \frac{I_{OLmax}}{I_{ILmax}}$

The I_{OLmax} of 74HCT is given by 4 mA and I_{ILmax} of 74LS is given by 0.4 mA which are required to operate it.

Substitute 4 for I_{OLmax} and 0.4 for I_{ILmax} .

$$\begin{aligned}\textbf{The LOW-state fanout} &= \frac{I_{OLmax}}{I_{ILmax}}\\ &= \frac{4}{0.4}\\ &= 10\end{aligned}$$

Thus, the LOW-state fanout is 10.

Step 3 of 9

Determine the HIGH-state fanout for 74HCT driving 74LS.

$$\textbf{HIGH-state fanout} = \frac{I_{OHmax}}{I_{IHmax}}$$

The I_{OHmax} of 74HCT is given by 4000 μA and I_{IHmax} of 74LS is given by 20 μA which are required to operate it.

Substitute 400 for I_{OHmax} and 20 for I_{IHmax} .

$$\begin{aligned}\textbf{HIGH-state fanout} &= \frac{I_{OHmax}}{I_{IHmax}}\\ &= \frac{4000}{20}\\ &= 200\end{aligned}$$

Thus, the HIGH-state fanout is 200.

The maximum fanout is the minimum of LOW state fan-out and HIGH state fan-outs, that is, 10.

Thus, the maximum fanout of 74HCT driving 74LS is $\boxed{10}$.

The excess current is in HIGH state and value is $\boxed{3800\ \mu A}$.

Step 4 of 9

(b)

Determine the LOW-state fanout for 74VHCT driving 74S.

The LOW-state $\textbf{fanout} = \frac{I_{OLmax}}{I_{ILmax}}$

The I_{OLmax} of 74VHCT is given by 8 mA and I_{ILmax} of 74S is given by 2 mA which are required to operate it.

Substitute 8 for I_{OLmax} and 2 for I_{ILmax} .

$$\begin{aligned}\textbf{The LOW-state fanout} &= \frac{I_{OLmax}}{I_{ILmax}}\\ &= \frac{8}{2}\\ &= 4\end{aligned}$$

Thus, the LOW-state fanout is 4.

Step 5 of 9

Determine the HIGH-state fanout for 74VHCT driving 74S.

$$\textbf{HIGH-state fanout} = \frac{I_{OHmax}}{I_{IHmax}}$$

The I_{OHmax} of 74VHCT is given by 8000 μA and I_{IHmax} of 74S is given by 50 μA which are required to operate it.

Substitute 8000 for I_{OHmax} and 50 for I_{IHmax} .

$$\begin{aligned}\textbf{HIGH-state fanout} &= \frac{I_{OHmax}}{I_{IHmax}}\\ &= \frac{8000}{50}\\ &= 160\end{aligned}$$

Thus, the HIGH-state fanout is 160.

The maximum fanout is the minimum of LOW state fan-out and HIGH state fan-outs, that is, 4.

Thus, the maximum fanout of 74VHCT driving 74S is $\boxed{4}$.

The excess current is in HIGH state and value is $\boxed{7800\ \mu A}$.

Step 6 of 9

(c)

Determine the LOW-state fanout for 74VHCT driving 74ALS.

The LOW-state $\textbf{fanout} = \frac{I_{OLmax}}{I_{ILmax}}$

The I_{OLmax} of 74VHCT is given by 8 mA and I_{ILmax} of 74ALS is given by 0.2 mA which are required to operate it.

Substitute 8 for I_{OLmax} and 0.2 for I_{ILmax} .

$$\begin{aligned}\textbf{The LOW-state fanout} &= \frac{I_{OLmax}}{I_{ILmax}}\\ &= \frac{8}{0.2}\\ &= 40\end{aligned}$$

Thus, the LOW-state fanout is 40.

Step 7 of 9

Determine the HIGH-state fanout 74VHCT driving 74ALS.

$$\textbf{HIGH-state fanout} = \frac{I_{OHmax}}{I_{IHmax}}$$

The I_{OHmax} of 74VHCT is given by 8000 μA and I_{IHmax} of 74ALS is given by 20 μA which are required to operate it.

Substitute 8000 for I_{OHmax} and 20 for I_{IHmax} .

$$\begin{aligned}\textbf{HIGH-state fanout} &= \frac{I_{OHmax}}{I_{IHmax}}\\ &= \frac{8000}{20}\\ &= 400\end{aligned}$$

Thus, the HIGH-state fanout is 400.

The maximum fanout is the minimum of LOW state fan-out and HIGH state fan-outs, that is, 40.

Thus, the maximum fanout of 74VHCT driving 74ALS is $\boxed{40}$.

The excess current is in HIGH state and value is $\boxed{3840\ \mu A}$.

Step 8 of 9

(d)

Determine the LOW-state fanout for 74HCT driving 74AS.

The LOW-state $\textbf{fanout} = \frac{I_{OLmax}}{I_{ILmax}}$

The I_{OLmax} of 74HCT is given by 4 mA and I_{ILmax} of 74AS is given by 0.5 mA which are required to operate it.

Substitute 4 for I_{OLmax} and 0.5 for I_{ILmax} .

$$\begin{aligned}\textbf{The LOW-state fanout} &= \frac{I_{OLmax}}{I_{ILmax}}\\ &= \frac{4}{0.5}\\ &= 8\end{aligned}$$

Thus, the LOW-state fanout is 8.

Step 9 of 9

Determine the HIGH-state fanout 74HCT driving 74LS.

$$\textbf{HIGH-state fanout} = \frac{I_{OHmax}}{I_{IHmax}}$$

The I_{OHmax} of 74VHCT is given by 4000 μA and I_{IHmax} of 74AS is given by 20 μA which are required to operate it.

Substitute 4000 for I_{OHmax} and 20 for I_{IHmax} .

$$\begin{aligned}\textbf{HIGH-state fanout} &= \frac{I_{OHmax}}{I_{IHmax}}\\ &= \frac{4000}{20}\\ &= 200\end{aligned}$$

Thus, the HIGH-state fanout is 200.

The maximum fanout is the minimum of LOW state fan-out and HIGH state fan-outs, that is, 8.

Thus, the maximum fanout of 74HCT driving 74AS is $\boxed{8}$.

The excess current is in HIGH state and value is $\boxed{3840\ \mu A}$.

Step 1 of 1

The dynamic power dissipation occurs during transition. The formula for finding out dynamic power dissipation is given by,

$$P_D = C \cdot V^2 \cdot f$$

Here,

C is the charging and discharging of capacitance during transition,

V is the supply voltage, and

f is the transition rate per second.

From the formula, we observe that the device with more supply voltage and higher transition has more dynamic power dissipation. Thus, FCT device has more dynamic power dissipation according to the context, because the output high voltage is 5 V for FCT and transition rate is very high. Thus dynamic power dissipation is more.

Compared to low dynamic power dissipated devices, highest power dissipated device has its power reduced, heat is reduced and efficiency is increased.

Step 1 of 2

The functional behavior of a logic circuit with 3-inputs is shown in Figure 1.

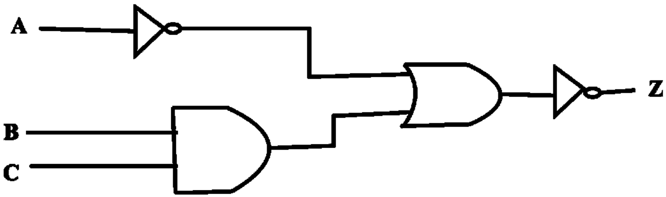


Figure 1: The 3-input logic circuit.

The logic function for the above circuit can be written as follows:

$$Z = \left[A' + (B \cdot C)\right]'$$

Step 2 of 2

The CMOS circuit that has the functional behavior shown in Figure 1 is shown in

Figure 2.

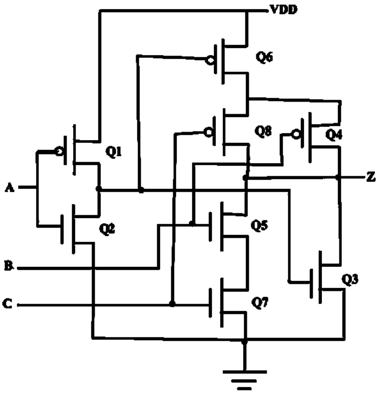


Figure 2: CMOS circuit for the function $Z = \left[A' + (B \cdot C)\right]'$

Thus, the CMOS circuit is designed.

Step 1 of 2

The functional behavior of a logic circuit with 3-inputs is shown in Figure 1.

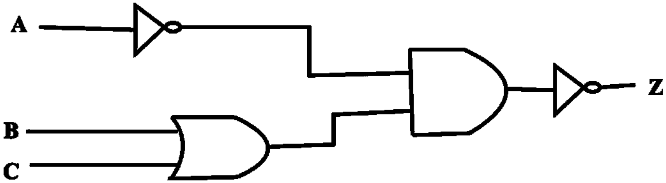


Figure 1: The 3-input logic circuit.

The logic function for the above circuit can be written as follows,

$$Z=\left[A'\cdot(B+C)\right]'$$

Step 2 of 2

The CMOS circuit that has the functional behavior shown in Figure 1 is shown in

Figure 2.

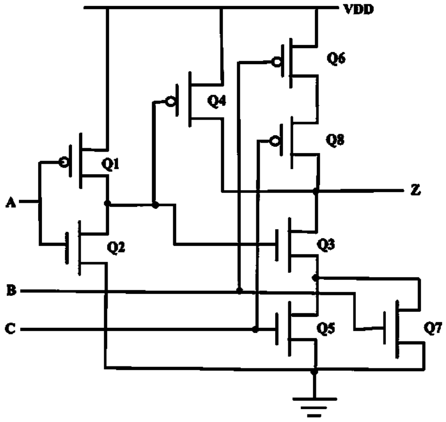


Figure 2: CMOS circuit for the function $Z=\left[A'\cdot(B+C)\right]'$

Thus, the CMOS circuit is designed.

Step 1 of 4

The functionality of a logic circuit for two inputs A & B and an output is that the output,

$\mathbf{Z=1}$ when $\mathbf{A=1 \text{ and } B=1}$ and $\mathbf{Z=0}$ otherwise.

Write the logic equation representing the functionality.

$$\begin{aligned} Z &= ((A'\cdot B)')' \\ &= (A''+B')' \\ &= (A+B')' \end{aligned}$$

Thus, the logic function is written as $\boxed{Z=(A+B')'}$.

Step 2 of 4

The CMOS circuit that has the functional behavior $Z=(A+B')'$ is shown in Figure 1.

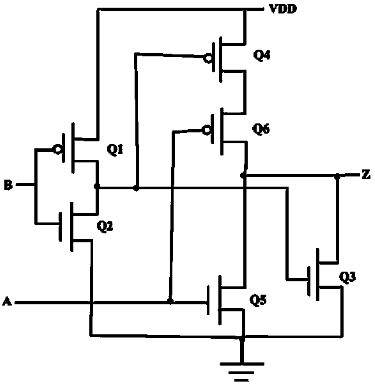


Figure 1: CMOS logic circuit for the function $Z=(A+B')'$

Step 3 of 4

The function table for the circuit in Figure 1 is shown in Table 1.

Table 1: The function table for $Z=(A+B')'$

A	B	Q1	Q2	Q3	Q4	Q5	Q6	Z
L	L	on	off	on	off	off	on	L
L	H	off	on	off	on	off	on	H
H	L	on	off	on	off	on	off	L
H	H	off	on	off	on	on	off	L

Step 4 of 4

The logic symbol or diagram for the function $Z=(A+B')'$ is shown in Figure 2.

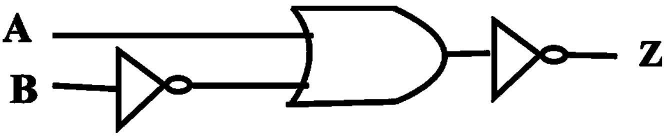


Figure 2: The logic symbol or diagram for function $Z=(A+B')'$

Thus, the circuit diagram, function table and logic symbol of the functionality are determined.

Step 1 of 4

The functionality of a logic circuit for two inputs A & B and an output is that the output,

$\mathbf{Z=0}$ when $\mathbf{A=1\text{ and }B=0}$ and $\mathbf{Z=1}$ otherwise.

Write the logic equation representing the functionality.

$$\begin{aligned}\mathbf{Z} &= \mathbf{A'+B}\\ &= \mathbf{(AB')'}\end{aligned}$$

Thus, the logic function is written as $\boxed{\mathbf{Z=(AB')'}}$.

Step 2 of 4

The CMOS circuit that has the functional behavior $\mathbf{Z=(AB')'}$ is shown in Figure 1.

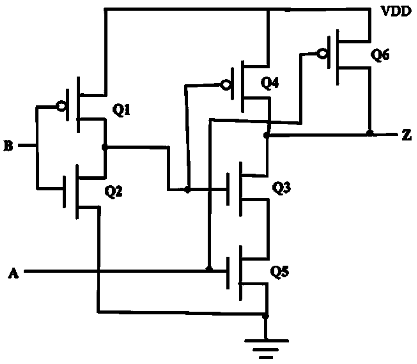


Figure 1: CMOS logic circuit for the function $\mathbf{Z=(AB')'}$

Step 3 of 4

The function table for the circuit in Figure 1 is shown in Table 1.

Table 1: The function table for $\mathbf{Z=(AB')'}$

A	B	Q1	Q2	Q3	Q4	Q5	Q6	Z
L	L	on	off	on	off	off	on	H
L	H	off	on	off	on	off	on	H
H	L	on	off	on	off	on	off	L
H	H	off	on	off	on	on	off	H

Step 4 of 4

The logic symbol or diagram for the function $\mathbf{Z=(AB')'}$ is shown in Figure 2.

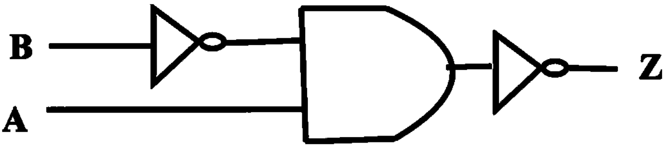


Figure 2: The logic symbol or diagram for function $\mathbf{Z=(AB')'}$

Thus, the circuit diagram, function table and logic symbol of the functionality are determined.

Step 1 of 3

3.063E

Logic structure of an 8-input $(A_1, B_1, C_1, D_1, A_2, B_2, C_2, D_2)$ CMOS NAND gate has to be implemented by means of 4-input CMOS NAND gate and 2-input NOR gate. 8-input CMOS NAND gate to be implemented as follows:

$$F = \overline{A_1 \cdot B_1 \cdot C_1 \cdot D_1 \cdot A_2 \cdot B_2 \cdot C_2 \cdot D_2}$$

$$= \overline{(A_1 \cdot B_1 \cdot C_1 \cdot D_1) \cdot (A_2 \cdot B_2 \cdot C_2 \cdot D_2)} \qquad \begin{pmatrix} \textbf{DeMorgan's Law} \\ \overline{a \cdot b} = \overline{a} + \overline{b} \end{pmatrix}$$

$$= \overline{(A_1 \cdot B_1 \cdot C_1 \cdot D_1)} + \overline{(A_2 \cdot B_2 \cdot C_2 \cdot D_2)}$$

To implement 8-input NAND gate, two 4-input NAND gates and two 2-input OR gate are required. But need to design with 2-input NOR gate.

Implement OR operation with NOR gate.

$$A + B = \overline{\overline{A + B}} \qquad \begin{pmatrix} \textbf{Complement Law} \\ \overline{\overline{a}} = a \end{pmatrix}$$

Step 2 of 3

Hence inverting the NOR gate output will be equivalent to OR operation. Inverting operation can be done by two input NOR gate by connecting both the inputs to the same value. NOR gate can also do the same NOT operation for single input or two same inputs.

Consider the following truth table for NOR and OR gates:

Table 1

X	Y	NOR Gate output	OR Gate output
0	0	1	0
0	1	0	1
1	0	0	1
1	1	0	1

Step 3 of 3

Draw the designed 8-input CMOS NAND gate by two 4-input NAND gates and two 2-input NOR gates.

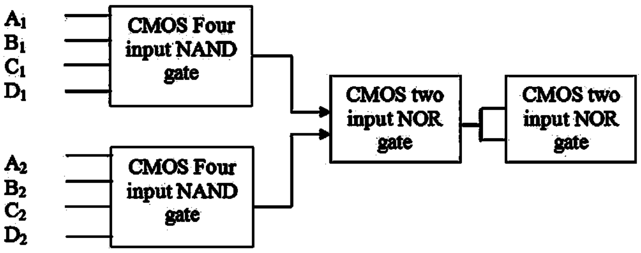


Figure 1

Therefore, 8-input CMOS NAND gate by two 4-input NAND gates and two 2-input NOR gates is designed.

Step 1 of 1

3.064E

Here CMOS is driving a TTL gate. At each time its need to take care of the input voltage level of TTL by keeping it within specified limit.

If keep the voltage level of "on" transistor of CMOS gate at HIGH state as same as in LOW state, it means the resistance of p-channel transistor becomes smaller and which enhance the input current to some large extent which affect the TTL device.

In any device always try to operate the device at much below value than the specified maximum limit. Here if we make the voltage level of p-channel transistor at HIGH state as same as of voltage level of "on" transistor of LOW state, we will make the TTL input voltage level to reach its maximum value.

Input voltage range of TTL gate is 2 to 5 V. When the voltages at HIGH and LOW states are equal, the CMOS output voltage at HIGH state will be $\mathbf{5-0.4=4.6V}$. This voltage is almost equal to the maximum voltage 5 V.

Step 1 of 2

Refer to Figure 3-32 (b) in the text book.

Calculate the wasted current in the circuit, I_{wasted} .

$$\begin{aligned} I_{\text{wasted}} &= \frac{V_{CC}}{4000+200} \\ &= \frac{5}{4200} \\ &= 0.00119 \\ &= 1.19 \text{ mA} \end{aligned}$$

Therefore, wasted current in the circuit, I_{wasted} is $\boxed{1.19 \text{ mA}}$.

Step 2 of 2

Calculate the wasted power in the circuit, P_{wasted} .

$$P_{\text{wasted}} = V_{CC} I_{\text{wasted}}$$

Substitute 5 V for V_{CC} and 1.19 mA for I_{wasted} .

$$\begin{aligned} P_{\text{wasted}} &= (5)\left(1.19 \times 10^{-3}\right) \\ &= 5.95 \times 10^{-3} \\ &= 5.95 \text{ mW} \end{aligned}$$

Therefore, wasted current in the circuit, P_{wasted} is $\boxed{5.95 \text{ mW}}$.

Step 1 of 15

3.066E

Refer to Figure 3-33 in the textbook.

As the circuit contains two dc sources, V_{CC} and V_{Thev} . So, need to apply superposition theorem to calculate the output voltage, V_{OUT} .

First, short the Thevenin's voltage and calculate the output voltage V_{OUT1} .

Draw the modified circuit with shorted Thevenin's voltage.

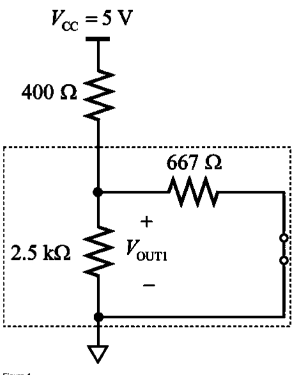


Figure 1

Step 2 of 15

From Figure 1, the resistors $667\ \Omega$ and $2.5\ \mathrm{k\Omega}$ are connected in parallel

Calculate the resultant resistance, R_1 .

$$\begin{aligned}R_1 &= (667\ \Omega)\parallel(2.5\ \mathrm{k\Omega})\\ &= \frac{(667)(2500)}{667+2500}\\ &= \frac{1667500}{3167}\\ &= 526.52\ \Omega\end{aligned}$$

Step 3 of 15

Draw the modified circuit with the resultant resistance.

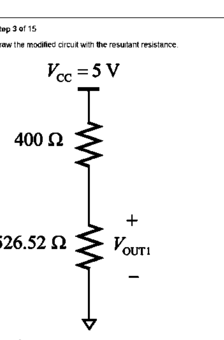


Figure 2

Step 4 of 15

Apply voltage division rule to calculate the output voltage, V_{OUT1} .

$$\begin{aligned}V_{OUT1} &= \left(\frac{526.52}{400+526.52}\right)V_{CC}\\ &= \left(\frac{526.52}{926.52}\right)(5)\\ &= 2.84\ \mathrm{V}\end{aligned}$$

Therefore, the output voltage due to the supply voltage, V_{CC} is 2.84 V .

Step 5 of 15

Now, short the supply voltage and calculate the output voltage, V_{OUT2} .

Draw the modified circuit with shorted supply voltage.

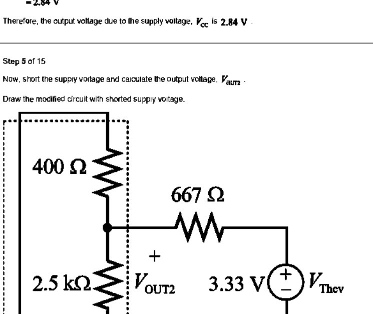


Figure 3

Step 6 of 15

From Figure 3, the resistors $400\ \Omega$ and $2.5\ \mathrm{k\Omega}$ are connected in parallel.

Calculate the resultant resistance, R_2 .

$$\begin{aligned}R_2 &= (400\ \Omega)\parallel(2.5\ \mathrm{k\Omega})\\ &= \frac{(400)(2500)}{400+2500}\\ &= \frac{1000000}{2900}\\ &= 344.83\ \Omega\end{aligned}$$

Step 7 of 15

Draw the modified circuit with the resultant resistance.

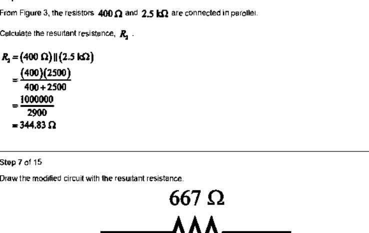


Figure 4

Step 8 of 15

Apply voltage division rule to calculate the output voltage, V_{OUT2} .

$$\begin{aligned}V_{OUT2} &= \left(\frac{344.83}{667+344.83}\right)V_{Thev}\\ &= \left(\frac{344.83}{1011.83}\right)(3.33)\\ &= \frac{1148.284}{1011.83}\\ &= 1.135\ \mathrm{V}\end{aligned}$$

Therefore, the output voltage due to the Thevenin's voltage, V_{Thev} is 1.135 V.

Calculate the output voltage by superposition theorem, V_{OUT} .

$$V_{OUT} = V_{OUT1} + V_{OUT2}$$

Substitute 2.84 V for V_{OUT1} and 1.135 V for V_{OUT2} .

$$\begin{aligned}V_{OUT} &= 2.84 + 1.135\\ &= 3.975\ \mathrm{V}\end{aligned}$$

Therefore, the output voltage of the CMOS inverter is $\boxed{3.975\ \mathrm{V}}$.

Step 9 of 15

Refer to Figure 3-34 in the textbook.

As the circuit contains two dc sources, V_{CC} and V_{Thev} . So, need to apply superposition theorem to calculate the output voltage, V_{OUT} .

First, short the Thevenin's voltage and calculate the output voltage V_{OUT1} .

Draw the modified circuit with shorted Thevenin's voltage.

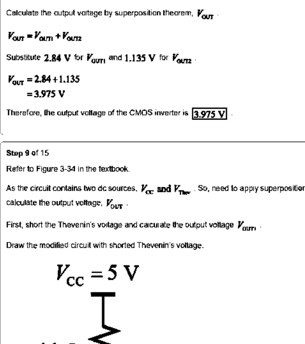


Figure 5

Step 10 of 15

From Figure 1, the resistors $667\ \Omega$ and $200\ \Omega$ are connected in parallel.

Calculate the resultant resistance, R_3 .

$$\begin{aligned}R_3 &= (667\ \Omega)\parallel(200\ \Omega)\\ &= \frac{(667)(200)}{667+200}\\ &= \frac{133400}{867}\\ &= 153.86\ \Omega\end{aligned}$$

Step 11 of 15

Draw the modified circuit with the resultant resistance.

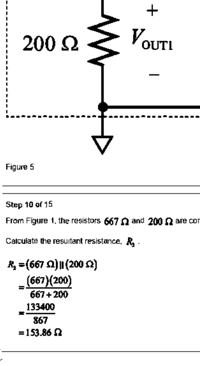


Figure 6

Step 12 of 15

Apply voltage division rule to calculate the output voltage, V_{OUT1} .

$$\begin{aligned}V_{OUT1} &= \left(\frac{153.86}{4000+153.86}\right)V_{CC}\\ &= \left(\frac{153.86}{4153.86}\right)(5)\\ &= 0.185\ \mathrm{V}\end{aligned}$$

Therefore, the output voltage due to the supply voltage, V_{CC} is 0.185 V .

Step 13 of 15

Now, short the supply voltage and calculate the output voltage, V_{OUT2} .

Draw the modified circuit with shorted supply voltage.

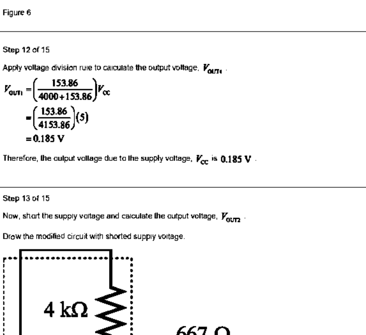


Figure 7

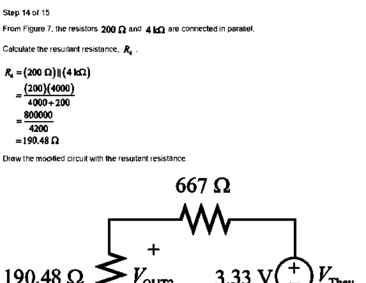
Step 14 of 15

From Figure 7, the resistors $200\ \Omega$ and $4\ \mathrm{k\Omega}$ are connected in parallel.

Calculate the resultant resistance, R_4 .

$$\begin{aligned}R_4 &= (200\ \Omega)\parallel(4\ \mathrm{k\Omega})\\ &= \frac{(200)(4000)}{4000+200}\\ &= \frac{800000}{4200}\\ &= 190.48\ \Omega\end{aligned}$$

Draw the modified circuit with the resultant resistance.



Figvre 6

Step 15 of 15

Apply voltage division rule to calculate the output voltage, V_{OUT2} .

$$\begin{aligned}V_{OUT2} &= \left(\frac{190.48}{667+190.48}\right)V_{Thev}\\ &= \left(\frac{190.48}{857.48}\right)(3.33)\\ &= \frac{634.3}{857.48}\\ &= 0.74\ \mathrm{V}\end{aligned}$$

Therefore, the output vallage due to the Thevenin's voltage, V_{Thev} is 0.74 V .

Calculate the ovtput voltage by svperposition theorem, V_{OUT} .

$$V_{OUT} = V_{OUT1} + V_{OUT2}$$

Substitute 0.185 V for V_{OUT1} and 0.74 V for V_{OUT2} .

$$\begin{aligned}V_{OUT} &= 0.185 + 0.74\\ &= 0.925\ \mathrm{V}\end{aligned}$$

Therefore, the output voltage of the CMOS inverter is $\boxed{0.925\ \mathrm{V}}$.

Step 1 of 1 **3.067E**

Consider the expression for rise and fall times.

$$t = -RC\ln\left(\frac{V_{DD} - V_{OUT}}{V_{DD}}\right)$$

The rise time or fall time is directly proportional to the resistance.

$$t \propto R$$

The resistance of the charging path, R_r is double the resistance of the discharging path, R_f .

$$R_r = 2R_f$$

Calculate the relation between the rise time and fall time.

$$t \propto R$$
$$\frac{t_r}{t_f} = \frac{R_r}{R_f}$$

Substitute $2R_f$ for R_r .

$$\frac{t_r}{t_f} = \frac{2R_f}{R_f}$$
$$\frac{t_r}{t_f} = 2$$
$$t_r = 2t_f$$

Therefore, if the resistance of the charging path is double the resistance of the discharging path the rise time also exactly twice the fall time.

Step 1 of 6

4.01DP

Refer to Figure 3-37 in the textbook.

Draw the circuit in s-domain.

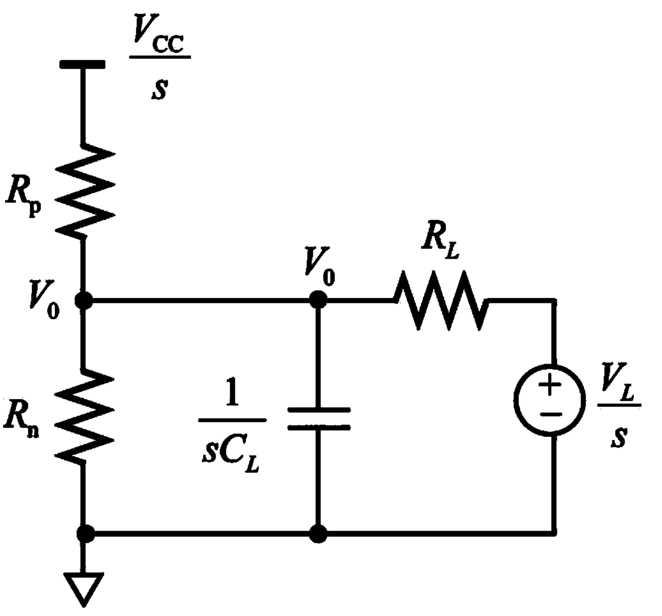


Figure 2

Step 2 of 6

Apply Kirchhoff's current Law at the node V_0 .

$$\frac{V_0-\frac{V_{CC}}{s}}{R_p}+\frac{V_0}{R_n}+\frac{V_0}{\frac{1}{sC_L}}+\frac{V_0-\frac{V_L}{s}}{R_L}=0$$

$$V_0\left[\frac{1}{R_p}+\frac{1}{R_n}+sC_L+\frac{1}{R_L}\right]=\frac{V_{CC}}{R_p s}+\frac{V_L}{R_L s}$$

$$V_0\left[\frac{1}{R_p}+\frac{1}{R_n}+sC_L+\frac{1}{R_L}\right]=\left[V_{CC}\left(\frac{1}{R_p}\right)+V_L\left(\frac{1}{R_L}\right)\right]\frac{1}{s}$$

Here,

The Load voltage, V_L is $2.0\ \text{V}$

The supply voltage, V_{CC} is $5.0\ \text{V}$

The load resistance, R_L is $900\ \Omega$

The load capacitance, C_L is $100\ \text{pF}$

The value of the resistor, R_n is $100\ \Omega$

The value of the resistor, R_p is $>1\ \text{M}\Omega$

As the value of the resistor R_p is $>1\ \text{M}\Omega$ means very large, so the value of $\frac{1}{R_p}$ is almost zero.

Step 3 of 6

Substitute $900\ \Omega$ for R_L , $100\ \Omega$ for R_n , $2.0\ \text{V}$ for V_L , $100\ \text{pF}$ for C_L , $5.0\ \text{V}$ for V_{CC} , and 0 for $\frac{1}{R_p}$.

$$V_0\left[0+\frac{1}{100}+s\left(100\times10^{-12}\right)+\frac{1}{900}\right]=\left[5(0)+2\left(\frac{1}{900}\right)\right]\frac{1}{s}$$

$$V_0\left[\frac{900s\left(100\times10^{-12}\right)+10}{900}\right]=\frac{2}{900s}$$

$$V_0\left[s\left(9\times10^{-8}\right)+10\right]=\frac{2}{s}$$

$$V_0=\frac{2}{s\left[s\left(9\times10^{-8}\right)+10\right]}$$

$$V_0=\frac{22.2\times10^6}{s\left[s+\left(111.1\times10^6\right)\right]}$$

$$V_0=\left(\frac{22.2\times10^6}{111.1\times10^6}\right)\left[\frac{1}{s}-\frac{1}{\left[s+\left(111.1\times10^6\right)\right]}\right]$$

$$V_0=(0.2)\left[\frac{1}{s}-\frac{1}{\left[s+\left(111.1\times10^6\right)\right]}\right]$$

Apply inverse Laplace transforms.

$$V_{OUT}=0.2\left[1-e^{-\left(111.1\times10^6\right)t}\right]$$

Step 4 of 6

To obtain the fall time, solve the preceding equation for $V_{OUT}=3.5\ \text{V}$ and $V_{OUT}=1.5\ \text{V}$.

Calculate the fall time for $V_{OUT}=3.5\ \text{V}$.

$$3.5=0.2\left[1-e^{-\left(111.1\times10^6\right)t_{3.5}}\right]$$

$$1-e^{-\left(111.1\times10^6\right)t_{3.5}}=17.5$$

$$e^{-\left(111.1\times10^6\right)t_{3.5}}=-16.5$$

$$-\left(111.1\times10^6\right)t_{3.5}=-\ln(16.5)$$

$$-\left(111.1\times10^6\right)t_{3.5}=-2.8$$

$$\begin{aligned}t_{3.5}&=\frac{2.8}{111.1\times10^6}\\&=0.0252\times10^{-6}\\&=25.2\ \text{ns}\end{aligned}$$

Step 5 of 6

Calculate the fall time for $V_{OUT}=1.5\ \text{V}$.

$$1.5=0.2\left[1-e^{-\left(111.1\times10^6\right)t_{1.5}}\right]$$

$$1-e^{-\left(111.1\times10^6\right)t_{1.5}}=3$$

$$e^{-\left(111.1\times10^6\right)t_{1.5}}=-2$$

$$-\left(111.1\times10^6\right)t_{1.5}=-\ln(2)$$

$$-\left(111.1\times10^6\right)t_{1.5}=-0.693$$

$$\begin{aligned}t_{1.5}&=\frac{0.693}{111.1\times10^6}\\&=0.00623\times10^{-6}\\&=6.23\times10^{-9}\\&=6.23\ \text{ns}\end{aligned}$$

Step 6 of 6

The fall time, t_f is the difference between time at $V_{OUT}=3.5\ \text{V}$ and time at $V_{OUT}=1.5\ \text{V}$.

Calculate the fall time, t_f .

$$t_f=t_{3.5}-t_{1.5}$$

Substitute $25.2\ \text{ns}$ for $t_{3.5}$ and $6.23\ \text{ns}$ for $t_{1.5}$.

$$\begin{aligned}t_f&=25.2\ \text{ns}-6.23\ \text{ns}\\&=18.97\ \text{ns}\end{aligned}$$

Therefore, the fall time of the CMOS inverter output is $\boxed{18.97\ \text{ns}}$.

Step 1 of 5 4.02DP

Consider the following axioms that are used to prove theorems T2-T5 using perfect induction:

$A1 \Rightarrow X = 0 \text{ if } X = 1$	$A1' \Rightarrow X = 1 \text{ if } X = 0$
$A2 \Rightarrow X = 0, \text{ then } X' = 1$	$A2' \Rightarrow X = 1, \text{then } X' = 0$
$A3 \Rightarrow 0 \cdot 0 = 0$	$A3' \Rightarrow 1 + 1 = 1$
$A4 \Rightarrow 1 \cdot 1 = 1$	$A4' \Rightarrow 0 + 0 = 0$
$A5 \Rightarrow 0 \cdot 1 = 1 \cdot 0 = 0$	$A5' \Rightarrow 1 + 0 = 0 + 1 = 1$

Step 2 of 5

Proof for the theorem T2 using perfect induction method $(T2 \Rightarrow X + 1 = 1)$:

To prove the theorem 2 $(T2 \Rightarrow X + 1 = 1)$ involving a single variable X by proving that it is true for both $X = 0$ and $X = 1$ as follows,

X	X+1=1
0	0+1=1
1	1+1=1

(According to axioms $A3'$ and $A5'$)

Thus, the theorem $\boxed{X+1=1}$ is proved.

Step 3 of 5

Proof for the theorem T3 using perfect induction method $(T3 \Rightarrow X + X = X)$:

To prove the theorem 3 $(T3 \Rightarrow X + X = X)$ involving a single variable X by proving that it is true for both $X = 0$ and $X = 1$ as follows,

X	X+X=X
0	0+0=0
1	1+1=1

(According to axioms $A3'$ and $A4'$)

Thus, the theorem T3, $\boxed{X+X=X}$ is proved.

Step 4 of 5

Proof for the theorem T4 using perfect induction method $(T4 \Rightarrow (X')' = X)$:

To prove the theorem 4 $(T4 \Rightarrow (X')' = X)$ involving a single variable X by proving it is true for both $X = 0$ and $X = 1$ as follows,

X	X'	(X')'
0	1	0=0
1	0	1=1

(According to axioms $A2$ and $A2'$)

Thus, the theorem T4, $\boxed{(X')'=X}$ is proved.

Step 5 of 5

Proof for the theorem T5 using perfect induction method $(T5 \Rightarrow X + X' = 1)$:

To prove the theorem $(T5 \Rightarrow X + X' = 1)$ involving a single variable X by proving it is true for both $X = 0$ and $X = 1$ as follows,

X	X'	X+X'=1
0	1	1+0=1
1	0	0+1=1

(According to axioms $A2$, $A2'$, and $A5'$)

Thus, the theorem T5 $\boxed{X+X'=1}$ is proved.

Step 1 of 5

Consider the following axioms that are used to prove theorems $T1'-T3'$ and $T5'$ using perfect induction:

$A1 \Rightarrow X=0 \text{ if } X=1$	$A1' \Rightarrow X=1 \text{ if } X=0$
$A2 \Rightarrow X=0, \text{ then } X'=1$	$A2' \Rightarrow X=1, \text{then } X'=0$
$A3 \Rightarrow 0\cdot 0=0$	$A3' \Rightarrow 1+1=1$
$A4 \Rightarrow 1\cdot 1=1$	$A4' \Rightarrow 0+0=0$
$A5 \Rightarrow 0\cdot 1=1\cdot 0=0$	$A5' \Rightarrow 1+0=0+1=1$

Step 2 of 5

Proof for the theorem $T1'$ using perfect induction method $(T1' \Rightarrow X\cdot 1=X)$:

To prove the theorem $(T1' \Rightarrow X\cdot 1=X)$ involving a single variable X by proving it is true for both X=0 and X=1 as follows,

X	$X\cdot 1=X$
0	$0\cdot 1=0$
1	$1\cdot 1=1$

(According to axioms $A5$ and $A4$)

Thus, the theorem $T1'$ $\boxed{X\cdot 1=X}$ is proved.

Step 3 of 5

Proof for the theorem $T2'$ using perfect induction method $(T2' \Rightarrow X\cdot 0=0)$:

To prove the theorem $(T2' \Rightarrow X\cdot 0=0)$ involving a single variable X by proving it is true for both X=0 and X=1 as follows,

X	$X\cdot 0=0$
0	$0\cdot 0=0$
1	$1\cdot 0=0$

(According to axioms $A5$ and $A3$)

Thus, the theorem $T2'$ $\boxed{X\cdot 0=0}$ is proved.

Step 4 of 5

Proof for the theorem $T3'$ using perfect induction method $(T3' \Rightarrow X\cdot X=X)$:

To prove the theorem $(T3' \Rightarrow X\cdot X=X)$ involving a single variable X by proving it is true for both X=0 and X=1 as follows,

X	$X\cdot X=X$
0	$0\cdot 0=0$
1	$1\cdot 1=1$

(According to axioms $A3$ and $A4$)

Thus, the theorem $T3'$ $\boxed{X\cdot X=X}$ is proved.

Step 5 of 5

Proof for the theorem $T5'$ using perfect induction method $(T5' \Rightarrow X\cdot X'=0)$:

To prove the theorem $(T5' \Rightarrow X\cdot X'=0)$ involving a single variable X by proving it is true for both X=0 and X=1 as follows,

X	X'	$X\cdot X'=0$
0	1	$0\cdot 1=0$
1	0	$1\cdot 0=0$

(According to axiom $A5$)

Thus, the theorem $T5'$ $\boxed{X\cdot X'=0}$ is proved.

Step 1 of 1

The DeMorgan's theorem for two variables A and B states that,

$$(A \cdot B)' = A' + B'$$
$$(A + B)' = A' \cdot B'$$

According to DeMorgan's theorem, the complement of $W \cdot X + Y \cdot Z$ is $W' + X' \cdot Y' + Z'$. Both functions are 1 for $W = 1,\ X = 1,\ Y = 1,$ and $Z = 0$.

$$\begin{aligned} W \cdot X + Y \cdot Z &= 1 \cdot 1 + 1 \cdot 0 \qquad (\text{Since } 1 \cdot 1 = 1, 1 \cdot 0 = 0 \text{ and } 1 + 0 = 1) \\ &= 1 + 0 \\ &= 1 \end{aligned}$$

Calculate the complement, $W' + X' \cdot Y' + Z'$

$$\begin{aligned} W' + X' \cdot Y' + Z' &= 1' + 1' \cdot 1' + 0' \qquad \left(\text{Since } 1' = 0, 0' = 1, 0 \cdot 0 = 0 \text{ and } 0 + 1 = 1\right) \\ &= 0 + 0 \cdot 0 + 1 \\ &= 0 + 0 + 1 \\ &= 1 \end{aligned}$$

The function and its complement are 1 for the same input combination because the precedence of operator '$\cdot$' is more than operator '$+$'. The wrong here is parenthesization is missing here. The correct result will get only parenthesization is used in proper place as, $\boxed{(W \cdot X + Y \cdot Z)' = \left(W' + X'\right) \cdot \left(Y' + Z'\right)}$.

Step 1 of 5

To simplify the Boolean function or logic function the following switching algebra theorems are used:

$$\begin{array}{ll}
T1 \Rightarrow X+0=X & (T1')\Rightarrow X\cdot 1=X \\
T2 \Rightarrow X+1=1 & T2' \Rightarrow X\cdot 0=0 \\
T3 \Rightarrow X+X=X & T3' \Rightarrow X\cdot X=X \\
T4 \Rightarrow (X')'=X & \\
T5 \Rightarrow X+X'=1 & T5' \Rightarrow X\cdot X'=0 \\
T6 \Rightarrow X+Y=Y+X & T6' \Rightarrow X\cdot Y=Y\cdot X \\
T7 \Rightarrow (X+Y)+Z=X+(Y+Z) & T7' \Rightarrow (X\cdot Y)\cdot Z=X\cdot(Y\cdot Z) \\
T8 \Rightarrow X\cdot Y+X\cdot Z=X\cdot(Y+Z) & T8' \Rightarrow (X+Y)\cdot(X+Z)=X(Y+Z) \\
T9 \Rightarrow X+X\cdot Y=X & T9' \Rightarrow X\cdot(X+Y)=X
\end{array}$$

Step 2 of 5

(a) Steps for simplifying the logic function, using switching algebra theorems as follows,

$$F=W\cdot X\cdot Y\cdot Z\cdot(W\cdot X\cdot Y\cdot Z'+W\cdot X'\cdot Y\cdot Z+W'\cdot X\cdot Y\cdot Z+W\cdot X\cdot Y'\cdot Z)$$

To take the parenthesize of the above function, multiply $W\cdot X\cdot Y\cdot Z$ with function inside parenthesize as shown,

$$\begin{aligned}
F&=\begin{Bmatrix} W\cdot X\cdot Y\cdot Z\cdot W\cdot X\cdot Y\cdot Z'+W\cdot X\cdot Y\cdot Z\cdot W\cdot X'\cdot Y\cdot Z+ \\ W\cdot X\cdot Y\cdot Z\cdot W'\cdot X\cdot Y\cdot Z+W\cdot X\cdot Y\cdot Z\cdot W\cdot X\cdot Y'\cdot Z \end{Bmatrix} \\
&=\begin{Bmatrix} W\cdot X\cdot Y\cdot Z\cdot Z'+W\cdot X\cdot X'\cdot Y\cdot Z+ \\ W\cdot W'\cdot X\cdot Y\cdot Z+W\cdot X\cdot Y\cdot Y'\cdot Z \end{Bmatrix} \text{ (According to T5')} \\
&=W\cdot X\cdot Y\cdot 0+W\cdot 0\cdot Y\cdot Z+0\cdot X\cdot Y\cdot Z+W\cdot X\cdot 0\cdot Z \text{ (According to T2')} \\
&=0+0+0+0 \\
&=0
\end{aligned}$$

Thus, the simplified value of the logic function F is $\boxed{0}$.

Step 3 of 5

(b) Steps for simplifying the logic function, using switching algebra theorems as follows,

$$\begin{aligned}
F&=A\cdot B+A\cdot B\cdot C'\cdot D+A\cdot B\cdot D\cdot E'+A'\cdot B\cdot C'\cdot E+A'\cdot B'\cdot C'\cdot E \\
&=A\cdot B(1+C'\cdot D)+A\cdot B\cdot D\cdot E'+A'\cdot C'\cdot E(B+B') \\
&=AB+A\cdot B\cdot D\cdot E'+A'\cdot C'\cdot E \quad \text{(According toT2 and T5)} \\
&=AB(1+D\cdot E')+A'\cdot C'\cdot E \quad \text{(According toT2)} \\
&=AB+A'\cdot C'\cdot E
\end{aligned}$$

Thus, the simplified logic function F is $\boxed{AB+A'\cdot C'\cdot E}$.

Step 4 of 5

(c) Steps for simplifying the logic function, using switching algebra theorems as follows,

$$\begin{aligned}
F&=M\cdot R\cdot P+Q\cdot O'\cdot R'+M\cdot N+O\cdot N\cdot M+Q\cdot P\cdot M\cdot O' \\
&=M\cdot R\cdot P+Q\cdot O'\cdot R'+M\cdot N(1+O)+Q\cdot P\cdot M\cdot O' \\
&=M\cdot R\cdot P+Q\cdot O'\cdot R'+M\cdot N+Q\cdot P\cdot M\cdot O' \quad \text{(According to T2)} \\
&=M\cdot R\cdot P+Q\cdot O'\cdot R'+M\cdot N+Q\cdot P\cdot M\cdot O'(R+R') \quad \text{(Since } R+R'=1)
\end{aligned}$$

Further simplification yields,

$$\begin{aligned}
F&=M\cdot R\cdot P+Q\cdot O'\cdot R'+M\cdot N+Q\cdot P\cdot M\cdot O'\cdot R+Q\cdot P\cdot M\cdot O'\cdot R' \\
&=M\cdot R\cdot P+Q\cdot O'\cdot R'(1+P\cdot M)+M\cdot N+Q\cdot P\cdot M\cdot O'\cdot R \\
&=M\cdot R\cdot P+Q\cdot O'\cdot R'+M\cdot N+Q\cdot P\cdot M\cdot O'\cdot R \quad \text{(According to T2)} \\
&=M\cdot R\cdot P(1+Q\cdot O')+Q\cdot O'\cdot R'+M\cdot N \\
&=M\cdot R\cdot P+Q\cdot O'\cdot R'+M\cdot N
\end{aligned}$$

Thus, the simplified function F is $\boxed{M\cdot R\cdot P+Q\cdot O'\cdot R'+M\cdot N}$.

Step 5 of 5

Step 1 of 15 **4.08DP**

Consider the following axioms that are used to find the truth table for the given logic functions:

$A1 \Rightarrow X = 0 \text{ if } X = 1$	$A1' \Rightarrow X = 1 \text{ if } X = 0$
$A2 \Rightarrow X = 0, \text{ then } X' = 1$	$A2' \Rightarrow X = 1, \text{then } X' = 0$
$A3 \Rightarrow 0 \cdot 0 = 0$	$A3' \Rightarrow 1 + 1 = 1$
$A4 \Rightarrow 1 \cdot 1 = 1$	$A4' \Rightarrow 0 + 0 = 0$
$A5 \Rightarrow 0 \cdot 1 = 1 \cdot 0 = 0$	$A5' \Rightarrow 1 + 0 = 0 + 1 = 1$

Step 2 of 15

(a) Consider the following function $F = X' \cdot Y' \cdot Z' + X \cdot Y \cdot Z + X \cdot Y' \cdot Z$

The truth table for the function F can be found by finding the value for $X' \cdot Y' \cdot Z'$, $X \cdot Y \cdot Z$ & $X \cdot Y' \cdot Z$ and then adding them as follows,

X	Y	Z	$X' \cdot Y' \cdot Z'$	$X \cdot Y \cdot Z$	$X \cdot Y' \cdot Z$	$F = X' \cdot Y' \cdot Z' + X \cdot Y \cdot Z + X \cdot Y' \cdot Z$
0	0	0	1	&	0	1
0	0	1	&	0	0	0
0	1	0	&	0	0	0
0	1	1	0	0	0	0
1	0	0	0	0	0	0
1	0	1	0	0	1	1
1	1	0	0	0	0	0
1	1	1	&	1	0	1

Table 1

Step 3 of 15

Thus, the truth table for a particular logic function $F(X, Y, Z)$ is:

X	Y	Z	$F = X' \cdot Y' \cdot Z' + X \cdot Y \cdot Z + X \cdot Y' \cdot Z$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	&
1	0	1	1
1	1	0	0
1	1	1	1

Table 2

Step 4 of 15

(b) Consider the following funciwn $F = M' \cdot N' + M \cdot P + N' \cdot P$

The truth table for the function F can be found by finding the value for $M' \cdot N'$, $M \cdot P$ & $N' \cdot P$ and then ad4ing them as follows,

M	N	P	$M' \cdot N'$	$M \cdot P$	$N' \cdot P$	$F = M' \cdot N' + M \cdot P + N' \cdot P$
0	&	0	1	0	&	1
0	&	1	1	0	1	1
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	&	0	0	0	0	0
1	0	1	0	1	1	1
1	1	0	0	0	0	0
1	1	1	0	1	0	1

Table 3

Step 5 of 15

Thus, the truth table is written as follows.

M	N	P	$F = M' \cdot N' + M \cdot P + N' \cdot P$
0	0	0	1
0	&	1	1
0	1	0	0
0	1	1	0
1	&	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Step 6 of 15

Table 4

Step 7 of 15

(c) Consider the following function $F = A \cdot B + A \cdot B' \cdot C' + A' \cdot B \cdot C$

The truth table for the function F is fin4ing the value for $A \cdot B$, $A \cdot B' \cdot C'$, and $A' \cdot B \cdot C$ then adding them as follows,

A	B	C	$A \cdot B$	$A \cdot B' \cdot C'$	$A' \cdot B \cdot C$	$F = A \cdot B + A \cdot B' \cdot C' + A' \cdot B \cdot C$
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	1	0	1
1	0	1	0	0	0	0
1	1	0	1	0	0	1
1	1	1	1	0	0	1

Table 5

Thus, the truth table is written as follows,

A	B	C	$F = A \cdot B + A \cdot B' \cdot C' + A' \cdot B \cdot C$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Table 6

Step 8 of 15

(d) Consider the following function $F = A' \cdot B \cdot (C \cdot B \cdot A' + B \cdot C')$

The truth table for the function F is finding the values of $A' \cdot B$, $C \cdot B \cdot A'$ and $B \cdot C'$ then calculate them according to the function as follows.

A	B	C	$A' \cdot B$	$A' \cdot B \cdot C$	$B \cdot C'$	$F = A' \cdot B \cdot (C \cdot B \cdot A' + B \cdot C')$
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	1	0	1	1
0	1	1	1	1	0	1
1	0	0	0	8	0	0
1	0	1	0	8	0	0
1	1	0	0	8	1	0
1	1	1	0	0	0	0

Table 7

Step 9 of 15

Thus the truth table is written as follows,

A	0	C	$F = A' \cdot B \cdot (C \cdot B \cdot A' + B \cdot C')$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Table 8

Step 10 of 15

(e) Consider the following function,

$$F = X \cdot Y \cdot (X' \cdot Y \cdot Z + X \cdot Y' \cdot Z + X \cdot Y \cdot Z' + X' \cdot Y' \cdot Z)$$

The truth table for the function F is finding the value $X' \cdot Y \cdot Z, X \cdot Y' \cdot Z$

$X \cdot Y \cdot Z', X' \cdot Y' \cdot Z$ and $X \cdot Y$ then calculate them according to the function F:

X	Y	Z	$X' \cdot Y \cdot Z$	$X \cdot Y' \cdot Z$	$X \cdot Y \cdot Z'$	$X' \cdot Y' \cdot Z$	$X \cdot Y$	F
0	0	0	&	0	0	0	0	0
0	0	1	&	0	0	1	0	0
0	1	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0
1	0	0	0	0	0	0	0	0
1	0	1	0	1	0	0	0	0
1	1	0	0	8	1	0	1	1
1	1	1	0	8	0	0	1	0

Table 9

Thus the truth table is written as follows,

X	Y	Z	F
0	0	0	&
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	&

Table 10

Step 11 of 15

(f) Consider the following function $F = M \cdot N + M' \cdot N' \cdot P$

The truth table for the function F is finding the value for $M \cdot N$ and $M' \cdot N' \cdot P$ then adding them as follows,

M	N	P	$M \cdot N$	$M' \cdot N' \cdot P$	F
0	&	0	0	0	0
0	&	1	0	1	1
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	0	0
1	1	0	1	0	1
1	1	1	1	0	1

Table 11

Thus the truth table is writen as follows,

M	N	P	F
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

Table 12

Step 12 of 15

(g) Consider the following funciwn $F = (A + A') \cdot B + B \cdot A \cdot C' + C \cdot (A + B)' \cdot (A + B)$

The truth table for function F is finding the value $(A + A') \cdot B$, $B \cdot A \cdot C'$, and $C \cdot (A + B)' \cdot (A' + B)$ then adding them as follows,

A	B	C	$(A+B)'$	$(A'+B)$	$C \cdot (A+B)' \cdot (A'+B)$	$(A+A') \cdot B$	$B \cdot A \cdot C'$	F
0	0	0	1	1	&	0	0	0
0	0	1	1	1	1	&	0	1
0	1	0	0	1	0	1	0	1
0	1	1	0	1	0	1	0	1
1	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0
1	1	0	0	1	0	1	1	1
1	1	1	0	1	0	1	0	1

Table 13

Step 13 of 15

Thus the truth table is written as follows,

A	B	C	$F = (A + A') \cdot B + B \cdot A \cdot C' + C \cdot (A + B)' \cdot (A + B)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Table 14

Step 14 of 15

(h) Consider the following function $F = X \cdot Y' + Y \cdot Z + Z' \cdot X$

Thus truth table can be found by finding the value for $X \cdot Y'$, $Y \cdot Z$, and $Z' \cdot X$ then adding them as follows,

X	Y	Z	$X \cdot Y'$	$Y \cdot Z$	$Z' \cdot X$	F
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	1	0	1
1	0	0	1	0	1	1
1	0	1	1	0	&	1
1	1	0	0	0	1	1
1	1	1	0	1	&	1

Table 15

Step 15 of 15

Thus the truth table is written as follows,

X	Y	Z	$F = X \cdot Y' + Y \cdot Z + Z' \cdot X$
0	0	0	0
0	0	1	&
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Table 16

Step 1 of 9 4.09DP

(a) Consider the following function, $\mathbf{F}=\sum_{X,Y}(1,2)$

The canonical sum for the function F is,

$$\begin{aligned}F&=\sum_{X,Y}(1,2)\\&=X\cdot Y'+Y'\cdot X\end{aligned}$$

Thus, the canonical sum for the function F is $\boxed{X\cdot Y'+Y'\cdot X}$.

Write the function F in terms of maxterms as,

$$F=\sum_{X,Y}(1,2)=\prod_{X,Y}(0,3)$$

The canonical product for the function F is,

$$\begin{aligned}F&=\prod_{X,Y}(0,3)\\&=(X+Y)\cdot(X'+Y')\end{aligned}$$

Thus, the canonical product of the function F is $\boxed{(X+Y)\cdot(X'+Y')}$.

Step 2 of 9

(b) Consider the following function, $\mathbf{F}=\prod_{A,B}(0,1,2)$

The canonical product for the function F is,

$$\begin{aligned}F&=\prod_{A,B}(0,1,2)\\&=(A+B)\cdot(A+B')\cdot(A'+B)\end{aligned}$$

Thus, the canonical product for the function F is $\boxed{(A+B)\cdot(A+B')\cdot(A'+B)}$.

Write the function F in terms of minterms as,

$$F=\prod_{A,B}(0,1,2)=\sum_{A,B}(3)$$

The canonical sum for the function F is,

$$\begin{aligned}F&=\sum_{A,B}(3)\\&=AB\end{aligned}$$

Thus, the canonical sum of the function F is $\boxed{AB}$.

Step 3 of 9

(c) Consider the following function, $\mathbf{F}=\sum_{A,B,C}(1,2,4,6)$

The canonical sum for the function F is,

$$\begin{aligned}F&=\sum_{A,B,C}(1,2,4,6)\\&=A'\cdot B'\cdot C+A'\cdot B\cdot C'+A\cdot B'\cdot C'+A\cdot B\cdot C'\end{aligned}$$

Thus, the canonical sum for the function F is $\boxed{A'\cdot B'\cdot C+A'\cdot B\cdot C'+A\cdot B'\cdot C'+A\cdot B\cdot C'}$.

Write the function F in terms of maxterms as

$$\begin{aligned}F&=\sum_{A,B,C}(1,2,4,6)\\&=\prod_{A,B}(0,3,5)\end{aligned}$$

The canonical product for the function F is,

$$\begin{aligned}F&=\prod_{A,B}(0,3,5)\\&=(A+B+C)\cdot(A+B'+C')\cdot(A'+B+C')\end{aligned}$$

Thus, the canonical product of the function F is

$\boxed{(A+B+C)\cdot(A+B'+C')\cdot(A'+B+C')}$.

Step 4 of 9

(d) Consider the following function, $\mathbf{F}=\prod_{W,X,Y}(0,2,3,6,7)$

The canonical product for the function F is,

$$\begin{aligned}F&=\prod_{W,X,Y}(0,2,3,6,7)\\&=(W+X+Y)\cdot(W+X'+Y)\cdot(W+X'+Y')\cdot(W'+X'+Y)\cdot(W'+X'+Y')\end{aligned}$$

Thus, the canonical sum for the function F is

$\boxed{(W+X+Y)\cdot(W+X'+Y)\cdot(W+X'+Y')\cdot(W'+X'+Y)\cdot(W'+X'+Y')}$.

Write the function F in terms of maxterms as

$$\begin{aligned}F&=\prod_{W,X,Y}(0,2,3,6,7)\\&=\sum_{W,X,Y}(1,4,5)\end{aligned}$$

The canonical product for the function F is,

$$\begin{aligned}F&=\sum_{W,X,Y}(1,4,5)\\&=W'\cdot X'\cdot Y+W\cdot X'\cdot Y'+W\cdot X'\cdot Y\end{aligned}$$

Thus, the canonical product of the function F is $\boxed{W'\cdot X'\cdot Y+W\cdot X'\cdot Y'+W\cdot X'\cdot Y}$.

Step 5 of 9

(e) Consider the following function, $\mathbf{F}=X'+Y\cdot Z$

The steps for finding truth table for function F is,

X	Y	Z	X'	$Y\cdot Z$	$F=X'+Y\cdot Z$
0	0	0	1	0	1
0	0	1	1	0	1
0	1	0	1	0	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	1	1	0	1	1

Table 1

Thus, the truth table is written as,

X	Y	Z	$F=X'+Y\cdot Z$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Table 2

Step 6 of 9

Thus from the truth table 2, the minterms are written as,

$$F=\sum_{X,Y,Z}(0,1,2,3,7)$$

The canonical sum for the function F is,

$$\begin{aligned}F&=\sum_{X,Y,Z}(0,1,2,3,7)\\&=X'\cdot Y'\cdot Z'+X'\cdot Y'\cdot Z+X'\cdot Y\cdot Z'+X'\cdot Y\cdot Z+X\cdot Y\cdot Z\end{aligned}$$

Thus, the canonical sum for the function F is

$\boxed{X'\cdot Y'\cdot Z'+X'\cdot Y'\cdot Z+X'\cdot Y\cdot Z'+X'\cdot Y\cdot Z+X\cdot Y\cdot Z}$

Write the function F in terms of maxterm is

$$\begin{aligned}F&=\sum_{X,Y,Z}(0,1,2,3,7)\\&=\prod_{X,Y,Z}(4,5,6)\end{aligned}$$

The canonical product for the function F is,

$$\begin{aligned}F&=\prod_{X,Y,Z}(4,5,6)\\&=(X'+Y+Z)\cdot(X'+Y+Z')\cdot(X'+Y'+Z)\end{aligned}$$

Thus, the canonical product of the function F is

$\boxed{(X'+Y+Z)\cdot(X'+Y+Z')\cdot(X'+Y'+Z)}$.

Step 7 of 9

(f) Consider the following function, $\mathbf{F}=V+(W.X')'$

The steps for finding truth table for the function F,

V	W	X	$W\cdot X'$	$(W\cdot X')'$	$F=V+(W.X')'$
0	0	0	0	1	1
0	0	1	0	1	1
0	1	0	1	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	0	1	1

Table 3

Step 8 of 9

Thus the truth table is written as,

V	W	X	$F=V+(W.X')'$
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Table 4

Step 9 of 9

Thus from the truth table 4, the minterms is written as,

$$F=\sum_{V,W,X}(0,1,3,4,5,6,7)$$

The canonical sum for the function F is,

$$\begin{aligned}F&=\sum_{V,W,X}(0,1,3,4,5,6,7)\\&=\left\{\begin{matrix}V'\cdot W'\cdot X'+V'\cdot W'\cdot X+V'\cdot W\cdot X+\\V\cdot W'\cdot X'+V\cdot W'\cdot X+V\cdot W\cdot X'+V\cdot W\cdot X\end{matrix}\right\}\end{aligned}$$

Thus, the canonical sum for the function F is

$\boxed{V'\cdot W'\cdot X'+V'\cdot W'\cdot X+V'\cdot W\cdot X+V\cdot W'\cdot X'+V\cdot W'\cdot X+V\cdot W\cdot X'+V\cdot W\cdot X}$.

Write the function F in terms of maxterms as,

$$\begin{aligned}F&=\sum_{V,W,X}(0,1,3,4,5,6,7)\\&=\prod_{V,W,X}(2)\end{aligned}$$

The canonical product for the function F is,

$$\begin{aligned}F&=\prod_{V,W,X}(2)\\&=V+W'+X\end{aligned}$$

Thus, the canonical product of the function F is $\boxed{V+W'+X}$.

Step 1 of 9

4.10DP

(a) Consider the following function, $F=\sum_{X,Y,Z}(0,3)$

The canonical sum for the function F is,

$$F=\sum_{X,Y,Z}(0,3)$$
$$=X'\cdot Y'\cdot Z'+X'\cdot Y\cdot Z$$

Thus, the canonical sum for the function F is $\boxed{X'\cdot Y'\cdot Z'+X'\cdot Y\cdot Z}$

Write the function F in terms of maxterms as,

$$F=\sum_{X,Y,Z}(0,3)$$
$$=\prod_{X,Y,Z}(1,2,4,5,6,7)$$

The canonical product for the function F is,

$$F=\prod_{X,Y,Z}(1,2,4,5,6,7)$$
$$=\begin{Bmatrix}(X+Y+Z')\cdot(X+Y'+Z)\cdot(X'+Y+Z)\cdot\\(X'+Y+Z')\cdot(X'+Y'+Z)\cdot(X'+Y'+Z')\end{Bmatrix}$$

Thus, the canonical product of the function F is

$$\boxed{\begin{Bmatrix}(X+Y+Z')\cdot(X+Y'+Z)\cdot(X'+Y+Z)\cdot\\(X'+Y+Z')\cdot(X'+Y'+Z)\cdot(X'+Y'+Z')\end{Bmatrix}}.$$

Step 2 of 9

(b) Consider the following function, $F=\prod_{A,B,C}(1,2,4)$

The canonical product for the function F is,

$$F=\prod_{A,B,C}(1,2,4)$$
$$=(A+B+C')\cdot(A+B'+C)\cdot(A'+B+C)$$

Thus, the canonical product of the function F is

$$\boxed{(A+B+C')\cdot(A+B'+C)\cdot(A'+B+C)}$$

Write the function F in terms of minterms as,

$$F=\prod_{A,B,C}(1,2,4)$$
$$=\sum_{A,B,C}(0,3,5,6,7)$$

The canonical sum for the function F is,

$$F=\sum_{A,B,C}(0,3,5,6,7)$$
$$=A'\cdot B'\cdot C'+A'\cdot B\cdot C+A\cdot B'\cdot C+A\cdot B\cdot C'+A\cdot B\cdot C$$

Thus, the canonical sum of the function F is

$$\boxed{A'\cdot B'\cdot C'+A'\cdot B\cdot C+A\cdot B'\cdot C+A\cdot B\cdot C'+A\cdot B\cdot C}$$

Step 3 of 9

(c) Consider the following function, $F=\sum_{A,B,C,D}(1,2,5,6)$

The canonical sum for the function F is ,

$$F=\sum_{A,B,C,D}(1,2,5,6)$$
$$=A'\cdot B'\cdot C'\cdot D+A'\cdot B'\cdot C\cdot D'+A'\cdot B\cdot C'\cdot D+A'\cdot B\cdot C\cdot D'$$

Thus, the canonical sum of the function F is

$$\boxed{A'\cdot B'\cdot C'\cdot D+A'\cdot B'\cdot C\cdot D'+A'\cdot B\cdot C'\cdot D+A'\cdot B\cdot C\cdot D'}$$

Write the function F in terms of maxterms as,

$$F=\sum_{A,B,C,D}(1,2,5,6)$$
$$=\prod_{A,B,C,D}(0,3,4,7,8,9,10,11,12,13,14,15)$$

The canonical product for the function F is,

$$F=\prod_{A,B,C,D}(0,3,4,7,8,9,10,11,12,13,14,15)$$
$$=\begin{Bmatrix}(A+B+C+D)\cdot(A+B+C'+D')\cdot(A+B'+C+D)\cdot(A+B'+C'+D')\cdot\\(A'+B+C+D)\cdot(A'+B+C+D')\cdot(A'+B+C'+D)\cdot(A'+B+C'+D')\cdot\\(A'+B'+C+D)\cdot(A'+B'+C+D')\cdot(A'+B'+C'+D)\cdot(A'+B'+C'+D')\end{Bmatrix}$$

Thus, the canonical product of the function F is

$$\boxed{\begin{Bmatrix}(A+B+C+D)\cdot(A+B+C'+D')\cdot(A+B'+C+D)\cdot(A+B'+C'+D')\cdot\\(A'+B+C+D)\cdot(A'+B+C+D')\cdot(A'+B+C'+D)\cdot(A'+B+C'+D')\cdot\\(A'+B'+C+D)\cdot(A'+B'+C+D')\cdot(A'+B'+C'+D)\cdot(A'+B'+C'+D')\end{Bmatrix}}.$$

Step 4 of 9

(d) Consider the following function, $F=\prod_{M,N,P}(0,1,3,6,7)$

The canonical product of function F is,

$$F=\prod_{M,N,P}(0,1,3,6,7)$$
$$=(M+N+P)\cdot(M+N+P')\cdot(M+N'+P')\cdot(M'+N'+P)\cdot(M'+N'+P')$$

Thus, the canonical product of the function F is

$$\boxed{(M+N+P)\cdot(M+N+P')\cdot(M+N'+P')\cdot(M'+N'+P)\cdot(M'+N'+P')}$$

Write the function F in terms of minterms as,

$$F=\prod_{M,N,P}(0,1,3,6,7)$$
$$=\sum_{M,N,P}(2,4,5)$$

The canonical sum for the function F is, $F=\sum_{M,N,P}(2,4,5)$
$$=M'\cdot N\cdot P'+M\cdot N'\cdot P'+M\cdot N'\cdot P$$

Thus, the canonical sum of the function F is

$$\boxed{M'\cdot N\cdot P'+M\cdot N'\cdot P'+M\cdot N'\cdot P}$$

(e) Consider the following function, $F=X'+Y\cdot Z'+Y.Z'$

The steps for finding truth table for the function F is,

X	Y	Z	X'	$Y\cdot Z'$	$Y\cdot Z'$	$F=X'+Y\cdot Z'+Y.Z'$
0	0	0	1	0	0	1
0	0	1	1	0	0	1
0	1	0	1	1	1	1
0	1	1	1	0	0	1
1	0	0	0	0	0	0
1	0	1	0	0	0	0
1	1	0	0	1	1	1
1	1	1	0	0	0	0

Table 1

Step 5 of 9

Thus the truth table is written as,

X	Y	Z	$F=X'+Y\cdot Z'+Y.Z'$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Table 2

Step 6 of 9

Thus from the truth table 2, the minterms are written as,

$$F=\sum_{X,Y,Z}(0,1,2,3,6)$$

The canonical sum for the function F is,

$$F=\sum_{X,Y,Z}(0,1,2,3,6)$$
$$=X'\cdot Y'\cdot Z'+X'\cdot Y'\cdot Z+X'\cdot Y\cdot Z'+X'\cdot Y\cdot Z+X\cdot Y\cdot Z'$$

Thus, the canonical sum for the function F is

$$\boxed{X'\cdot Y'\cdot Z'+X'\cdot Y'\cdot Z+X'\cdot Y\cdot Z'+X'\cdot Y\cdot Z+X\cdot Y\cdot Z'}.$$

Write the function F in terms of maxterms as,

$$F=\sum_{X,Y,Z}(0,1,2,3,6)$$
$$=\prod_{X,Y,Z}(4,5,7)$$

The canonical product for the function F is,

$$F=\prod_{X,Y,Z}(4,5,7)$$
$$=(X'+Y+Z)\cdot(X'+Y+Z')\cdot(X'+Y'+Z')$$

Thus, the canonical product of the function F is,

$$\boxed{(X'+Y+Z)\cdot(X'+Y+Z')\cdot(X'+Y'+Z')}.$$

Step 7 of 9

(f) Consider the following function, $F=A'\cdot B+B'\cdot C+A$

The step to find the truth table for above function F is,

A	B	C	$A'\cdot B$	$B'\cdot C$	$F=A'\cdot B+B'\cdot C+A$
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	1	0	1
0	1	1	1	0	1
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	0	0	0
1	1	1	0	0	0

Table 3

Step 8 of 9

Thus the truth table written as,

A	B	C	$F=A'\cdot B+B'\cdot C+A$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Table 4

Step 9 of 9

Thus from the truth table 4, the minterms are written as,

$$F=\sum_{A,B,C}(1,2,3,5)$$

The canonical sum for the function F is,

$$F=\sum_{A,B,C}(1,2,3,5)$$
$$=A'\cdot B'\cdot C+A'\cdot B\cdot C'+A'\cdot B\cdot C+A\cdot B'\cdot C$$

Thus, the canonical sum for the function F is

$$\boxed{A'\cdot B'\cdot C+A'\cdot B\cdot C'+A'\cdot B\cdot C+A\cdot B'\cdot C}$$

Write the function F in terms of maxterms as,

$$F=\sum_{A,B,C}(1,2,3,5)$$
$$=\prod_{A,B,C}(0,4,6,7)$$

The canonical product for the function F is,

$$F=\prod_{A,B,C}(0,4,6,7)$$
$$=(A+B+C)\cdot(A'+B+C)\cdot(A'+B'+C)\cdot(A'+B'+C')$$

Thus, the canonical product of the function F is,

$$\boxed{(A+B+C)\cdot(A'+B+C)\cdot(A'+B'+C)\cdot(A'+B'+C')}.$$

4.11DP

Step 1 of 2

In general a 4-bit prime number detector, the input combinations of $N = ABCD$, produces a 1 output for $N = 1,2,3,5,7,11,13$ and 0 for other values.

Thus the minterm list is given by,

$$F = \sum_{A,B,C,D}(1,2,3,5,7,11,13)$$

But according to the context, mathematicians will tell that 1 is not really a prime number. Therefore, the minterm list is given by,

$$F=\sum_{A,B,C,D}(2,3,5,7,11,13)$$

Thus the canonical sum of the function F is given by,

$$F = \sum_{A,B,C,D}(2,3,5,7,11,13)$$
$$= \begin{bmatrix} (A'\cdot B'\cdot C\cdot D')+(A'\cdot B'\cdot C\cdot D)+(A'\cdot B\cdot C'\cdot D)+ \\ (A'\cdot B\cdot C\cdot D)+(A\cdot B'\cdot C\cdot D)+(A\cdot B\cdot C'\cdot D) \end{bmatrix}$$

Step 2 of 2

The following is the logic diagram of above canonical sum:

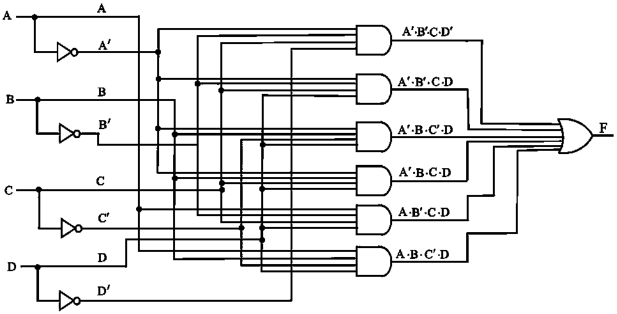


Figure 1

Step 1 of 1

4.12DP

Canonical sum is the sum of minterms corresponding to truth table for which the function produces a high (digit '1') output. Minimal sum is the sum of product terms (min terms) expression for the function. The logic function has fewer product terms. Do not have the other sum of product expression for the same function. Minterm list is also known as on-set of the logic function.

The minterms will also have all the *n*-input literals when the Canonical sum for an *n*-input logic function is same as minimal sum. In other words, the Minterms in the function does not differ by a single literal with other minterms in the function. Any sum of the expressions with same number of product terms has at least as many literals. Hence the function cannot be simplified using switching algebra axioms and theorems and therefore, the minimal sum is same as canonical sum.

Step 1 of 2 4.13DP

The NOT gate is also known as inverter; because it changes the input from a logic to its opposite logic (inverts it). In a digital logic, inverter or NOT gate is a logical gate which implements logical negation.

The truth table for NOT gate is as follows:

Input	Output
0	1
1	0

Step 2 of 2

The reasons for the cost of inverters are not included in the definition of 'minimal' for logic minimization is as follows:

(1) In LSI circuit and more than that, when compared with the logic gates the inverter cost is much less. So the cost of inverter is not included in the definition.

(2) Including the cost of the inverter in the each & every definition of minimal logic need more explanation and it makes the minimal logic more complex.

Step 1 of 11 **4.14DP**

(a) Consider the following data:

The logic function is,

$$F = å_{X,Y,Z}(1,3,5,6,7)$$

The *K-map* for function F is shown in Figure 1.

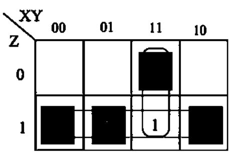


Figure 1

Step 2 of 11

From the *K-map* in Figure 1, the minimum sum of products expression is,

$$F = Z + X{\times}Y$$

Thus, the minimum sum of products expression is $\boxed{F = Z + X{\times}Y}$.

A distinguished 1-cell of a logic function is an input combination that is covered by only one prime implicant.

The distinguished 1- cells are highlighted with blue color in Figure 1.

Step 3 of 11

(b) Consider the following data:

The logic function is,

$$F = \sum\nolimits_{W,X,Y,Z}(1,4,5,6,7,9,14,15)$$

The *K-map* for function F is shown in Figure 2.

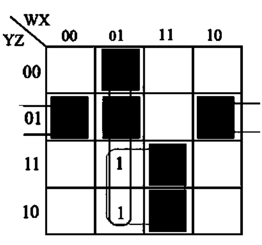


Figure 2

Step 4 of 11

From the *K-map* in Figure 2, the minimum sum of products expression is,

$$F = W' \cdot X + X \cdot Y + X' \cdot Y' \cdot Z$$

Thus, the minimum sum of products expression is

$\boxed{F = W¢×X + X×Y + X¢×Y¢×Z}$.

A distinguished 1-cell of a logic function is an input combination that is covered by only one prime implicant.

The distinguished 1- cells of function $F = \sum_{W,X,Y,Z}(1,4,5,6,7,9,14,15)$ are highlighted with blue color in Figure 2.

Step 5 of 11

(c) Consider the following data:

The logic function is,

$$F = \prod\nolimits_{W,X,Y}(1,4,5,6,7)$$

The function F can also written as,

$$F = \prod\nolimits_{W,X,Y}(1,4,5,6,7) = \sum\nolimits_{W,X,Y}(0,2,3)$$

The *K-map* for function F is shown in Figure 3.

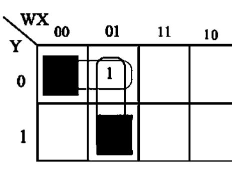


Figure 3

From the *K-map* in Figure 3, the minimum sum of products expression is,

$$F = W' \cdot Y' + W' \cdot X$$

Thus, the minimum sum of products expression is $\boxed{F = W¢×Y¢+ W¢×X}$.

A distinguished 1-cell of a logic function is an input combination that is covered by only one prime implicant.

The distinguished 1- cells of function $F = \prod_{W,X,Y}(1,4,5,6,7)$ are highlighted with blue color in Figure 3.

Step 6 of 11

(d) Consider the following data:

The logic function is,

$$F = \sum\nolimits_{W,X,Y,Z}(0,1,6,7,8,9,14,15)$$

The *K-map* for function F is shown in Figure 4.

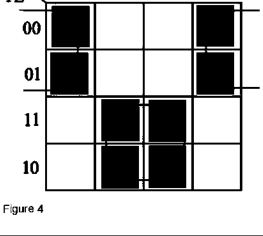


Figure 4

Step 7 of 11

From the *K-map* in Figure 4, the minimum sum of products expression is,

$$F = X \cdot Y + X' \cdot Y'$$

Thus, the minimum sum of products expression is $\boxed{F = X×Y + X¢×Y¢}$.

A distinguished 1-cell of a logic function is an input combination that is covered by only one prime implicant.

The distinguished 1- cells of function $F = \sum_{W,X,Y,Z}(0,1,6,7,8,9,14,15)$ are highlighted with blue color in Figure 4.

Step 8 of 11

(e) Consider the following data:

The logic function is,

$$F = \prod\nolimits_{A,B,C,D}(4,5,6,13,15)$$

The function F can also written as,

$$F = \prod\nolimits_{A,B,C,D}(4,5,6,13,15) = \sum\nolimits_{A,B,C,D}(0,1,2,3,7,8,9,10,11,12,14)$$

The *K-map* for function F is shown in Figure 5.

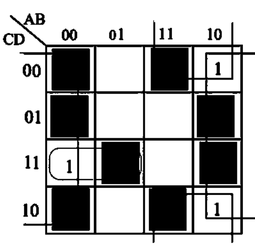


Figure 5

Step 9 of 11

From the *K-map* in Figure 5, the minimum sum of products expression is,

$$F = B' + A' \cdot D' + A' \cdot C \cdot D$$

Thus, the minimum sum of products expression is $\boxed{F = B¢+ A¢×D¢+ A¢×C×D}$.

A distinguished 1-cell of a logic function is an input combination that is covered by only one prime implicant.

The distinguished 1- cells of function $F = \prod_{A,B,C,D}(4,5,6,13,15)$ are highlighted with blue color in Figure 5.

Step 10 of 11

(f) Consider the following data:

The logic function is,

$$F = \sum\nolimits_{A,B,C,D}(4,5,6,11,13,14,15)$$

The *K-map* for function F is shown in Figure 6.

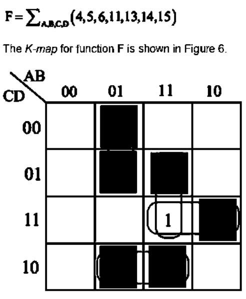


Figure 6

Step 11 of 11

From the *K-map* in Figure 6, the minimum sum of products expression is,

$$F = A' \cdot B \cdot C' + B \cdot C \cdot D' + A \cdot B \cdot D + A \cdot C \cdot D$$

Thus, the minimum sum of products expression is $\boxed{F = A¢×B×C¢+ B×C×D¢+ A×B×D + A×C×D}$.

A distinguished 1-cell of a logic function is an input combination that is covered by only one prime implicant.

The distinguished 1- cells of function $F = \sum_{A,B,C,D}(4,5,6,11,13,14,15)$ are highlighted with blue color in Figure 6.

Step 1 of 10

(a) Consider the following data:

The logic function is,

$F=\sum_{A,B,C}(0,1,2,4)$

The *K-map* for function F is shown in Figure 1.

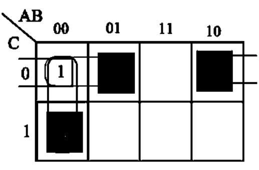


Figure 1

Step 2 of 10

From the *K-map* in Figure 1, the minimum sum of products expression is,

F= A×B+ AϕCϕ+ BϕC¢

Thus, the minimum sum of products expression is $\boxed{\text{F= A×B+ AϕCϕ+ BϕC¢}}$.

A distinguished 1-cell of a logic function is an input combination that is covered by only one prime implicant.

The distinguished 1- cells of function $F=\sum_{A,B,C}(0,1,2,4)$ are highlighted with blue color in Figure 1.

Step 3 of 10

(b) Consider the following data:

The logic function is,

$F=\sum_{W,X,Y,Z}(1,4,5,6,11,12,13,14)$

The *K-map* for function F is shown in Figure 2.

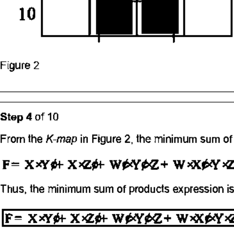


Figure 2

Step 4 of 10

From the *K-map* in Figure 2, the minimum sum of products expression is,

F= X×Yϕ+ X×Zϕ+ WϕXϕYϕZ+ W×XϕY×Z

Thus, the minimum sum of products expression is

$\boxed{\text{F= X×Yϕ+ X×Zϕ+ WϕXϕYϕZ+ W×XϕY×Z}}$.

A distinguished 1-cell of a logic function is an input combination that is covered by only one prime implicant.

The distinguished 1- cells of function $F=\sum_{W,X,Y,Z}(1,4,5,6,11,12,13,14)$ are highlighted with blue color in Figure 2.

Step 5 of 10

(c) Consider the following data:

The logic function is,

$F=\prod_{A,B,C}(1,2,6,7)$

The function F can also written as,

$F=\prod_{A,B,C}(1,2,6,7)=\sum_{A,B,C}(0,3,4,5)$

The *K-map* for function F is shown in Figure 3.

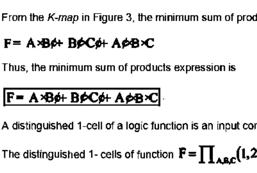


Figure 3

From the *K-map* in Figure 3, the minimum sum of products expression is,

F= A×Bϕ+ BϕCϕ+ AϕB×C

Thus, the minimum sum of products expression is

$\boxed{\text{F= A×Bϕ+ BϕCϕ+ AϕB×C}}$.

A distinguished 1-cell of a logic function is an input combination that is covered by only one prime implicant.

The distinguished 1- cells of function $F=\prod_{A,B,C}(1,2,6,7)$ are highlighted with blue color in Figure 3.

Step 6 of 10

(d) Consider the following data:

The logic function is,

$F=\sum_{W,X,Y,Z}(0,1,2,3,7,8,10,11,15)$

The *K-map* for function F is shown in Figure **4**.

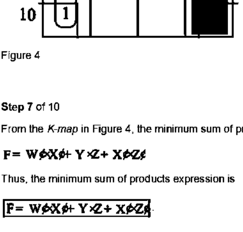


Figure 4

Step 7 of 10

From the *K-map* in Figure 4, the minimum sum of products expression is,

F= WϕXϕ+ Y×Z+ XϕZ¢

Thus, the minimum sum of products expression is

$\boxed{\text{F= WϕXϕ+ Y×Z+ XϕZ¢}}$.

A distinguished 1-cell of a logic function is an input combination that is covered by only one prime implicant.

The distinguished 1- cells of function $F=\sum_{W,X,Y,Z}(0,1,2,3,7,8,10,11,15)$ are highlighted with blue color in Figure 4.

Step 8 of 10

(e) Consider the following data:

The logic function is,

$F=\sum_{W,X,Y,Z}(1,2,4,7,8,11,13,14)$

The *K-map* for function F is shown in Figure 5.

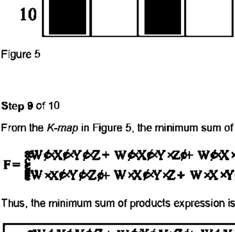


Figure 5

Step 9 of 10

From the *K-map* in Figure 5, the minimum sum of products expression is,

F= WϕXϕYϕZ+ WϕXϕY×Zϕ+ WϕX×YϕZϕ+ WϕX×Y×Z+
W×XϕYϕZϕ+ W×XϕY×Z+ W×X×YϕZ+ W×X×Y×Z¢

Thus, the minimum sum of products expression is

$\boxed{\begin{array}{l}\text{F= WϕXϕYϕZ+ WϕXϕY×Zϕ+ WϕX×YϕZϕ+ WϕX×Y×Z+}\\ \text{W×XϕYϕZϕ+ W×XϕY×Z+ W×X×YϕZ+ W×X×Y×Z¢}\end{array}}$.

A distinguished 1-cell of a logic function is an input combination that is covered by only one prime implicant.

The distinguished 1- cells of function $F=\sum_{W,X,Y,Z}(1,2,4,7,8,11,13,14)$ are highlighted with blue color in Figure 5.

(f) Consider the following data:

The logic function is,

$F=\prod_{A,B,C,D}(1,3,4,5,6,7,9,12,13,14)$

The function F can also written as,

$F=\prod_{A,B,C,D}(1,3,4,5,6,7,9,12,13,14)=\sum_{A,B,C,D}(0,2,8,10,11,15)$

The *K-map* for function F is shown in Figure 6.

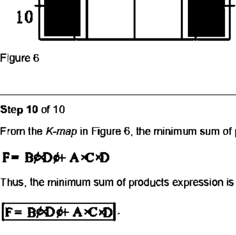


Figure 6

Step 10 of 10

From the *K-map* in Figure 6, the minimum sum of products expression is,

F= BϕDϕ+ A×C×D

Thus, the minimum sum of products expression is

$\boxed{\text{F= BϕDϕ+ A×C×D}}$.

A distinguished 1-cell of a logic function is an input combination that is covered by only one prime implicant.

The distinguished 1- cells of function $F=\prod_{A,B,C,D}(1,3,4,5,6,7,9,12,13,14)$ are highlighted with blue color in Figure 6.

Step 1 of 3 **4.16DP**

In a 4-bit prime number detector, the input combinations of $N = ABCD$, produces a 1 output for $N = 1,2,3,5,7,11,13$ and 0 for other values.

Thus, the minterm list is,

$$F=\sum_{A,B,C,D}(1,2,3,5,7,11,13)$$

But according to the context, mathematicians will tell that 1 is not really a prime number.

Therefore, the minterm list is,

$$F=\sum_{A,B,C,D}(2,3,5,7,11,13)$$

The canonical sum of a logic function is a sum of the minterms corresponding to truth table rows (input combinations) for which the function produces a 1 output.

Thus, the canonical sum of the function F is,

$$F=\sum_{A,B,C,D}(2,3,5,7,11,13)$$
$$=\begin{pmatrix}A'\cdot B'\cdot C\cdot D'+A'\cdot B'\cdot C\cdot D+A'\cdot B\cdot C'\cdot D+A'\cdot B\cdot C\cdot D+\\ A\cdot B'\cdot C\cdot D+A\cdot B\cdot C'\cdot D\end{pmatrix}$$

Step 2 of 3

The *K-map* for function $F=\sum_{A,B,C,D}(2,3,5,7,11,13)$ is shown in Figure 1.

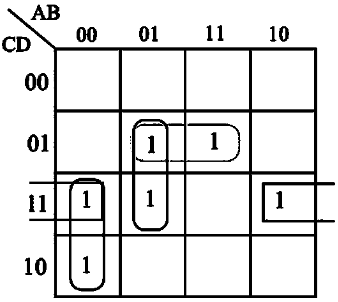


Figure 1

Step 3 of 3

From the *K-map* in Figure 1, the minimum sum of products expression is,

$$F=A'\cdot B'\cdot C+A'\cdot B\cdot D+B'\cdot C\cdot D+B\cdot C'\cdot D$$

The following is the design for minimal sum for 4 bit prime detector:

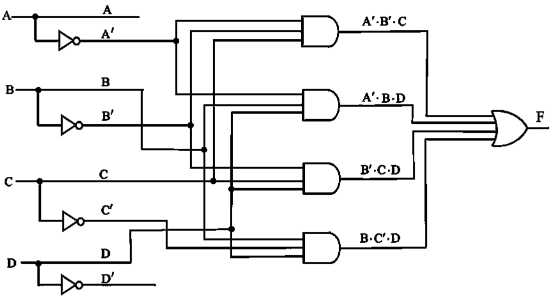


Figure 2

4.17DP

Step 1 of 4

(a) Consider the following data:

The logical function is,

$$F=\sum_{W,X,Y,Z}(0,1,2,3,7,8,10,11,15)$$

The sum of all the prime implicant of a logic function is called the complete sum.

The *K-map* for function F is shown in Figure 1.

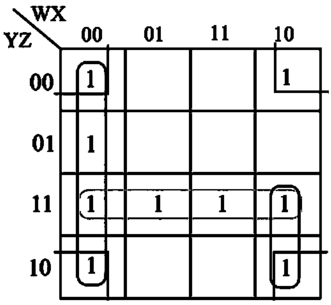


Figure 1

Step 2 of 4

From the *K-map* in Figure 1, the complete sum for the function is,

$$F=W'\cdot X'+Y\cdot Z+X'\cdot Z'+W\cdot X'\cdot Y$$

Thus, the complete sum for the function $F=\sum_{W,X,Y,Z}(0,1,2,3,7,8,10,11,15)$ is

$\boxed{F=W'\cdot X'+Y\cdot Z+X'\cdot Z'+W\cdot X'\cdot Y}$.

Step 3 of 4

(b) Consider the following data:

The logical function is,

$$F=\sum_{W,X,Y,Z}(1,2,4,7,8,11,13,14)$$

The sum of all the prime implicant of a logic function is called the complete sum.

The *K-map* for function F is shown in Figure 2.

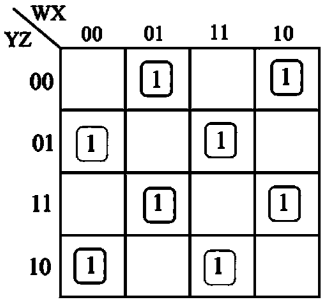


Figure 2

Step 4 of 4

From the *K-map* in Figure 2, the complete sum for the function is,

$$F=\begin{pmatrix}W'\cdot X'\cdot Y'\cdot Z+W'\cdot X'\cdot Y\cdot Z'+W'\cdot X\cdot Y'\cdot Z'+W'\cdot X\cdot Y\cdot Z+\\ W\cdot X'\cdot Y'\cdot Z'+W\cdot X'\cdot Y\cdot Z+W\cdot X\cdot Y'\cdot Z+W\cdot X\cdot Y\cdot Z'\end{pmatrix}$$

Thus, the complete sum for the function $F=\sum_{W,X,Y,Z}(1,2,4,7,8,11,13,14)$ is

$$\boxed{F=\begin{pmatrix}W'\cdot X'\cdot Y'\cdot Z+W'\cdot X'\cdot Y\cdot Z'+W'\cdot X\cdot Y'\cdot Z'+W'\cdot X\cdot Y\cdot Z+\\ W\cdot X'\cdot Y'\cdot Z'+W\cdot X'\cdot Y\cdot Z+W\cdot X\cdot Y'\cdot Z+W\cdot X\cdot Y\cdot Z'\end{pmatrix}}.$$

From the Karnaugh map it is observed that complete sum of the function is same as minimal function, because 1's are not able to combine in to a group.

Step 1 of 9

4.18DP

(a) Consider the following data:

The logic function is,

$$F=\sum\nolimits_{W,X,Y,Z}(0,1,3,5,14)+d(8,15)$$

The *K-map* for function F is shown in Figure 1.

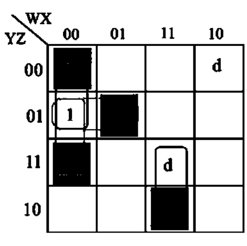


Figure 1

Step 2 of 9

From the *K-map* in Figure 1, the minimum sum of products expression is,

F= W¢×X¢×Y¢+ W¢×X¢×Z+ W¢×Y¢×Z+ W×X×Y

Thus, the minimum sum of products expression is

$\boxed{\text{F= W¢×X¢×Y¢+ W¢×X¢×Z+ W¢×Y¢×Z+ W×X×Y}}$.

A distinguished 1-cell of a logic function is an input combination that is covered by only one prime implicant.

The distinguished 1- cells of function $F=\sum\nolimits_{W,X,Y,Z}(0,1,3,5,14)+d(8,15)$ are highlighted with blue color in Figure 1.

Step 3 of 9

(b) Consider the following data:

The logic function is,

$$F=\sum\nolimits_{W,X,Y,Z}(0,1,2,8,11)+d(3,9,15)$$

The *K-map* for function F is shown in Figure 2.

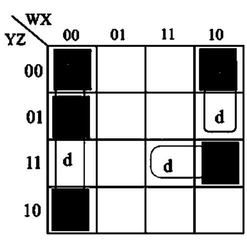


Figure 2

Step 4 of 9

From the *K-map* in Figure 2, the minimum sum of products expression is,

F= W¢×X¢+ W×X¢×Y¢+ W×Y×Z

Thus, the minimum sum of products expression is

$\boxed{\text{F= W¢×X¢+ W×X¢×Y¢+ W×Y×Z}}$.

A distinguished 1-cell of a logic function is an input combination that is covered by only one prime implicant.

The distinguished 1- cells of function $F=\sum\nolimits_{W,X,Y,Z}(0,1,2,8,11)+d(3,9,15)$ are highlighted with blue color in Figure 2.

Step 5 of 9

(c) Consider the following data:

The logic function is,

$$F=\sum\nolimits_{A,B,C,D}(4,6,7,9,13)+d(12)$$

The *K-map* for function F is shown in Figure 3.

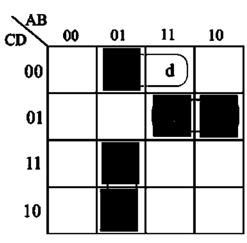


Figure 3

From the *K-map* in Figure 3, the minimum sum of products expression is,

$$F=A\cdot C'\cdot D+A'\cdot B\cdot C+B\cdot C'\cdot D'$$

Thus, the minimum sum of products expression is

$\boxed{\text{F= A×C¢×D + A¢×B×C + B×C¢×D¢}}$.

A distinguished 1-cell of a logic function is an input combination that is covered by only one prime implicant.

The distinguished 1- cells of function $F=\sum\nolimits_{A,B,C,D}(4,6,7,9,13)+d(12)$ are highlighted with blue color in Figure 3.

Step 6 of 9

(d) Consider the following data:

The logic function is,

$$F=\sum\nolimits_{A,B,C,D}(1,5,12,13,14,15)+d(7,9)$$

The *K-map* for function F is shown in Figure 4.

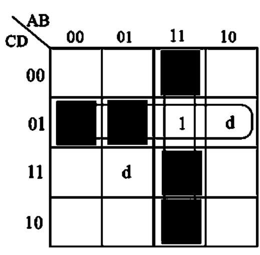


Figure 4

Step 7 of 9

From the *K-map* in Figure 4, the minimum sum of products expression is,

$$F=A\cdot B+C'\cdot D$$

Thus, the minimum sum of products expression is

$\boxed{\text{F= A×B + C¢×D}}$.

A distinguished 1-cell of a logic function is an input combination that is covered by only one prime implicant.

The distinguished 1- cells of function $F=\sum\nolimits_{A,B,C,D}(1,5,12,13,14,15)+d(7,9)$ are highlighted with blue color in Figure 4.

Step 8 of 9

(e) Consider the following data:

The logic function is,

$$F=\sum\nolimits_{W,X,Y,Z}(4,5,9,13,15)+d(0,1,7,11,12)$$

The *K-map* for function F is shown in Figure 5.

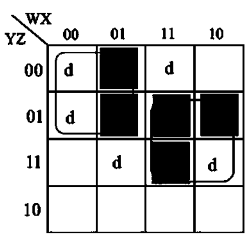


Figure 5

Step 9 of 9

From the *K-map* in Figure 5, the minimum sum of products expression is,

$$F=W'\cdot Y'+W\cdot Z$$

Thus, the minimum sum of products expression is

$\boxed{\text{F= W¢×Y¢+ W×Z}}$.

A distinguished 1-cell of a logic function is an input combination that is covered by only one prime implicant.

The distinguished 1- cells of function $F=\sum\nolimits_{W,X,Y,Z}(4,5,9,13,15)+d(0,1,7,11,12)$ are highlighted with blue color in Figure 5.

Step 2 of 10

(b) $F = W \cdot X' \cdot Y' + X \cdot Y' \cdot Z + X \cdot Y$

In order to find static hazards in a two level circuit, Karnaugh map is used.
The Karnaugh map for the above function can be drawn as follows,

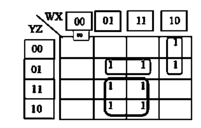


Figure 3: Karnaugh map for the function $F = W \cdot X' \cdot Y' + X \cdot Y' \cdot Z + X \cdot Y$

From the Karnaugh map, it is observed that there is possibility of glitches or static 1 hazard presence for the input combinations W,X,Y,Z= 0101 & 0111, W,X,Y,Z= 1101 & 1111 and W,X,Y,Z=1001 &1101. In order to eliminate the hazards from the function, an extra AND gate should be included. So that it covers all the input combinations that are creating hazards.
An extra AND gate where to be added or an extra product term where to be included in the function are as shown below in the Karnaugh map (highlighted with red color).

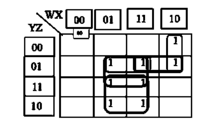


Figure 4: Karnaugh map for hazard free function $F = W \cdot X' \cdot Y' + X \cdot Y' \cdot Z + X \cdot Y$

Thus the hazard free function is designed and from the Karnaugh map the function can be written as follows,

$\boxed{F = W \cdot X' \cdot Y' + X \cdot Y' \cdot Z + X \cdot Y + X \cdot Z + W \cdot Y' \cdot Z}$.

Step 3 of 10

(c) $F = W \cdot Y + W' \cdot Z' + X \cdot Y' \cdot Z$

In order to find static hazards in a two level circuit, Karnaugh map is used.
The Karnaugh map for the above function can be drawn as follows,

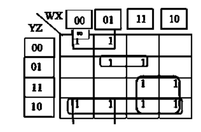


Figure 5: Karnaugh map for the function $F = W \cdot Y + W' \cdot Z' + X \cdot Y' \cdot Z$

Step 4 of 10

From the Karnaugh map, it is observed that there is possibility of glitches or static 1 hazard presence for the input combinations W,X,Y,Z= 0100 & 0101 and W,X,Y,Z=1101 &1111. In order to eliminate the hazards from the function, an extra AND gate should be included. So that it covers all the input combinations that are creating hazards.
An extra AND gate where to be added or an extra product term where to be included in the function are as shown below in the Karnaugh map (highlighted with red color).

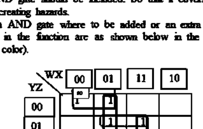


Figure 6: Karnaugh map for hazard free function $F = W \cdot Y + W' \cdot Z' + X \cdot Y' \cdot Z$

Thus the hazard free function is designed and from the Karnaugh map the function can be written as follows,

$\boxed{F = W \cdot Y + W' \cdot Z' + X \cdot Y' \cdot Z + W' \cdot X \cdot Y' + W \cdot X \cdot Z}$.

Step 5 of 10

(d) $F = W' \cdot X' + Y' \cdot Z + W' \cdot X \cdot Y \cdot Z + W \cdot X \cdot Y \cdot Z'$

In order to find static hazards in a two level circuit, Karnaugh map is used.
The Karnaugh map for the above function can be drawn as follows,

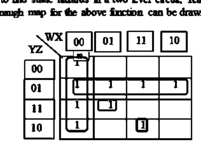


Figure 7: Karnaugh map for the function $F = W' \cdot X' + Y' \cdot Z + W' \cdot X \cdot Y \cdot Z + W \cdot X \cdot Y \cdot Z'$

From the Karnaugh map, it is observed that there is possibility of glitches or static 1 hazard presence for the input combinations W,X,Y,Z= 0101 & 0111 and W,X,Y,Z= 0111 & 0011. In order to eliminate the hazards from the function, an extra AND gate should be included. So that it covers all the input combinations that are creating hazards.
An extra AND gate where to be added or an extra product term where to be included in the function are as shown below in the Karnaugh map (highlighted with red color).

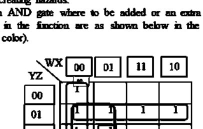


Figure 8: K- map for hazard free function $F = W' \cdot X' + Y' \cdot Z + W' \cdot X \cdot Y \cdot Z + W \cdot X \cdot Y \cdot Z'$

Thus the hazard free function is designed and from the Karnaugh map the function can be written as follows,

$\boxed{F = W' \cdot X' + Y' \cdot Z + W' \cdot X \cdot Y \cdot Z + W \cdot X \cdot Y \cdot Z' + W' \cdot Z}$.

Step 6 of 10

(e) $F = (W' + X + Y') \cdot (X' + Z')$

In order to find static hazards in a two level circuit, Karnaugh map is used.
The Karnaugh map for the above function can be drawn as follows,

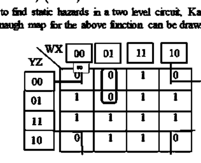


Figure 9: Karnaugh map for the function $F = (W' + X + Y') \cdot (X' + Z')$

From the Karnaugh map, it is observed that there is possibility of glitches or static 0 hazard presence for the input combinations W,X,Y,Z= 0000 & 0100. In order to eliminate the hazards from the function, an extra OR gate should be included. So that it covers all the input combinations that are creating hazards.
An extra OR gate where to be added or an extra sum term where to be included in the function are as shown below in the Karnaugh map (highlighted with red color).

Step 7 of 10

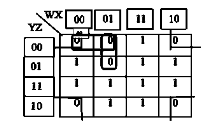


Figure 10: K- map for hazard free function $F = (W' + X + Y') \cdot (X' + Z')$

Thus the hazard free function is designed and from the Karnaugh map the function can be written as follows, $\boxed{F = (W' + X + Y') \cdot (X' + Z') \cdot (W' + Y' + Z')}$

Step 8 of 10

(f) $F = (W + Y' + Z') \cdot (W' + X' + Z') \cdot (X' + Y + Z)$

In order to find static hazards in a two level circuit, Karnaugh map is used.
The Karnaugh map for the above function can be drawn as follows,

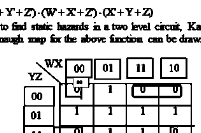


Figure 11: Karnaugh map for the function $F = (W + Y' + Z') \cdot (W' + X' + Z') \cdot (X' + Y + Z)$

From the Karnaugh map, it is observed that there is possibility of glitches or static 0 hazard presence for the input combinations W,X,Y,Z= 0000 & 1000 and W,X,Y,Z= 0010 & 0011. In order to eliminate the hazards from the function, an extra OR gate should be included. So that it covers all the input combinations that are creating hazards.
An extra OR gate where to be added or an extra sum term where to be included in the function are as shown below in the Karnaugh map (highlighted with red color).

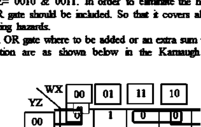


Figure 12: K- map for hazard free function $F = (W + Y' + Z') \cdot (W' + X' + Z') \cdot (X' + Y + Z)$

Thus the hazard free function is designed and from the Karnaugh map the function can be written as follows,

$\boxed{F = (W + Y' + Z') \cdot (W' + X' + Z') \cdot (X' + Y + Z) \cdot (X' + Y' + Z') \cdot (W' + X' + Y)}$.

Step 9 of 10

(g) $F = (W + Y + Z') \cdot (W' + X' + Y + Z) \cdot (X' + Y') \cdot (X + Z)$

In order to find static hazards in a two level circuit, Karnaugh map is used.
The Karnaugh map for the above function can be drawn as follows,

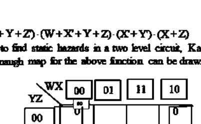


Figure 13: K- map for the function $F = (W + Y + Z') \cdot (W' + X' + Y + Z) \cdot (X' + Y') \cdot (X + Z)$

From the Karnaugh map, it is observed that there is possibility of glitches or static 0 hazard presence for the input combinations W,X,Y,Z= 0001 & 0101, W,X,Y,Z= 1101 & 1001, W,X,Y,Z= 1111 & 1011, W,X,Y,Z= 1111 & 1110 and W,X,Y,Z= 1011 & 1010. In order to eliminate the hazards from the function, an extra OR gate should be included. So that it covers all the input combinations that are creating hazards.
An extra OR gate where to be added or an extra sum term where to be included in the function are as shown below in the Karnaugh map (highlighted with red color).

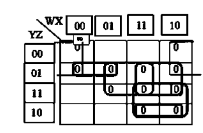


Figure 14: K- map for hazard free function $F = (W + Y + Z') \cdot (W' + X' + Y + Z) \cdot (X' + Y') \cdot (X + Z)$

Step 10 of 10

Thus the hazard free function is designed and from the Karnaugh map the function can be written as follows,

$\boxed{F = (W + Y + Z') \cdot (W' + X' + Y + Z) \cdot (X' + Y') \cdot (X + Z) \cdot (W' + Y' + Z) \cdot (W + Z) \cdot (W + Y)}$.

Step 1 of 2

Below diagram shows a non-trivial looking logic circuit that contains a feedback loop.

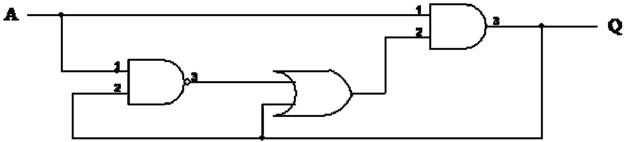


Figure1: Non-trivial logic circuit.

Step 2 of 2

The above circuit looks like non-trivial circuit also called as cycle, it means that the output is depends up on the previous input combination. But actually when we look in to the logic, the output depends only on its current input.

$$\begin{aligned} Q^{*} &= A\cdot((A\cdot Q)'+Q) \\ &= A\cdot(A'+Q'+Q) \\ &= A\cdot(A'+1) \\ &= A \end{aligned}$$

(Since $X+X'=1$ and $1+X=1$)

Thus from the above logic equation, the output is depend only on present input.

Step 1 of 1 **4.022E**

Below theorem is a combining theorem (According to context, its number is T10).

$\boxed{X \cdot Y + X \cdot Y' = X}$

In order to prove combining theorem, assume the below theorems are true.

$T1 \Rightarrow X + 0 = X \qquad T1' \Rightarrow X \cdot 1 = X$

$T2 \Rightarrow X + 1 = 1 \qquad T2' \Rightarrow X \cdot 0 = 0$

$T3 \Rightarrow X + X = X \qquad T3' \Rightarrow X \cdot X = X$

$T4 \Rightarrow (X')' = X$

$T5 \Rightarrow X + X' = 1 \qquad T5' \Rightarrow X \cdot X' = 0$

$T6 \Rightarrow X + Y = Y + X \qquad T6' \Rightarrow X \cdot Y = Y \cdot X$

$T7 \Rightarrow (X + Y) + Z = X + (Y + Z) \qquad T7' \Rightarrow (X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$

$T8 \Rightarrow X \cdot Y + X \cdot Z = X \cdot (Y + Z) \qquad T8' \Rightarrow (X + Y) \cdot (X + Z) = X(Y+Z)$

$T9 \Rightarrow X + X \cdot Y = X \qquad T9' \Rightarrow X \cdot (X + Y) = X$

Proof for combining theorem:

$$X \cdot Y + X \cdot Y' = X \cdot (Y + Y') \quad \text{(Taking variable X as common)}$$
$$= X \cdot 1 \quad \text{(According to T5)}$$
$$= X \quad \text{(According to T1')}$$

Thus the combining theorem is proved.

Step 1 of 1 **4.024E**

In order to prove the below equation without using perfect induction, the theorems T1-T11 and T1'-T11' are considered to be true.

$$(X+Y')\cdot Y = X\cdot Y$$

The theorems are shown below:

$$T1 \Rightarrow X+0=X \qquad T1' \Rightarrow X\cdot 1 = X$$
$$T2 \Rightarrow X+1=1 \qquad T2' \Rightarrow X\cdot 0 = 0$$
$$T3 \Rightarrow X+X=X \qquad T3' \Rightarrow X\cdot X = X$$
$$T4 \Rightarrow (X')'=X$$
$$T5 \Rightarrow X+X'=1 \qquad T5' \Rightarrow X\cdot X' = 0$$
$$T6 \Rightarrow X+Y=Y+X \qquad T6' \Rightarrow X\cdot Y = Y\cdot X$$
$$T7 \Rightarrow (X+Y)+Z=X+(Y+Z) \qquad T7' \Rightarrow (X\cdot Y)\cdot Z = X\cdot(Y\cdot Z)$$
$$T8 \Rightarrow X\cdot Y+X\cdot Z=X\cdot(Y+Z) \qquad T8' \Rightarrow (X+Y)\cdot(X+Z) = X(Y+Z)$$
$$T9 \Rightarrow X+X\cdot Y=X \qquad T9' \Rightarrow X\cdot(X+Y) = X$$
$$T10 \Rightarrow X\cdot Y+X\cdot Y'=X \qquad T10' \Rightarrow (X+Y)\cdot(X+Y') = X$$
$$T11 \Rightarrow X\cdot Y+X'\cdot Z+Y\cdot Z=X\cdot Y+X'\cdot Z$$
$$T11' \Rightarrow (X+Y)\cdot(X'+Z)\cdot(Y+Z)=(X+Y)\cdot(X'+Z)$$

Proof for the equation is given below.

$$\begin{aligned}(X+Y')\cdot Y &= (X\cdot Y)+(Y'\cdot Y)\\ &= (X\cdot Y)+0 \qquad \text{(Since } X\cdot X'=1\text{)}\\ &= X\cdot Y \qquad \text{(Since } X+0=X\text{)}\end{aligned}$$

Hence,

$$\boxed{(X+Y')\cdot Y = X\cdot Y}.$$

Thus the equation is proved.

Step 1 of 2 **4.025E**

The function of *n*-input OR gate with variables $x_1, x_2, x_3, \ldots, x_n$ can be written as follows,

$$\mathbf{F} = x_1 + x_2 + x_3 + \ldots + x_n$$

Consider when *n*=5, then the function can be written as follows,

$$\mathbf{F} = x_1 + x_2 + x_3 + x_4 + x_5 \quad \ldots\ldots (1)$$

The function of 2-input OR gate with variables $x_1, x_2, x_3 \ldots x_n$ can be written as follows,

$$\mathbf{F} = ((\ldots((x_1 + x_2) + x_3) + \ldots + x_n)$$

The enclosed number of each pair of bracket shown in above function gives the number of OR gate used in the function.

Consider when *n*=5, then the function can be written as follows,

$$\begin{aligned}\mathbf{F} &= ((((x_1 + x_2) + x_3) + x_4) + x_5) \\ &= (((x_1 + x_2 + x_3) + x_4) + x_5) \\ &= ((x_1 + x_2 + x_3 + x_4) + x_5)\end{aligned}$$

Hence,

$$\mathbf{F} = \left(x_1 + x_2 + x_3 + x_4 + x_5\right) \quad \ldots\ldots (2)$$

Compare equations (1) and (2). Both the functions yield the same value. It is observed that for *n*=5, we need four 2-input OR gate. Similarly for *n*=6, we need five 2-input OR gate. Thus in general we can say that, *n*-input OR gate can be replaced by (*n*-1) 2- input OR gate.

Step 2 of 2

The function of *n*-input NOR gate with variables $x_1, x_2, x_3, \ldots, x_n$ can be written as follows,

$$\mathbf{F} = \left(x_1 + x_2 + x_3 + \ldots + x_n\right)'$$

Consider when *n*=5, then the function can be written as follows,

$$\mathbf{F} = \left(x_1 + x_2 + x_3 + x_4 + x_5\right)' \quad \ldots\ldots (3)$$

The function of 2-input NOR gate with variables $x_1, x_2, x_3 \ldots x_n$ can be written as follows,

$$\mathbf{F} = ((\ldots.((x_1 + x_2)' + x_3)' + \ldots.. + x_n)'$$

The enclosed number of each pair of bracket shown in above function gives the number of NOR gates used in the function.

Consider when *n*=5, then the function can be written as follows,

$$\begin{aligned}\mathbf{F} &= ((((x_1 + x_2)' + x_3)' + x_4)' + x_5)' \\ &= (((x_1' \cdot x_2' + x_3)' + x_4)' + x_5)' \\ &= (((x_1' \cdot x_2')' \cdot x_3' + x_4)' + x_5)' \\ &= ((((x_1 + x_2) \cdot x_3')' \cdot x_4') + x_5)'\end{aligned}$$

Simplify further.

$$\mathbf{F} = ((((x_1 + x_2)' + x_3) \cdot x_4') + x_5)'$$

Hence,

$$\mathbf{F} = ((x_1' \cdot x_2' + x_3) \cdot x_4' + x_5)' \quad \ldots\ldots (4)$$

Compare equations (3) and (4), the function $\mathbf{F}$ yields different values.

Thus, an *n*-input NOR gate cannot be replaced by (*n*-1) 2- input NOR gate.

Step 1 of 3 **4.026E**

A function with similar operation can be implemented by more than one way using similar gates. There are three physically different ways to realize the function.

$$F = V \cdot W \cdot X \cdot Y \cdot Z$$

Between the variables similar AND operation has to done. By means of associative law,

$$A \cdot (B \cdot C) = (A \cdot B) \cdot C$$

Since we have a single output series connection is needed at the end. Another one is the given gates is similar maximum of two input gates. Therefore, maximum of two gates can be parallel. Hence we can implement the circuit such that

• Input and output Series implementation.

• Input Parallel and output series arrangement

• Few Series Input and some other parallel inputs, output series implementation

Step 2 of 3

Implementation of the function F using serial connection of AND gates is shown in Figure 1

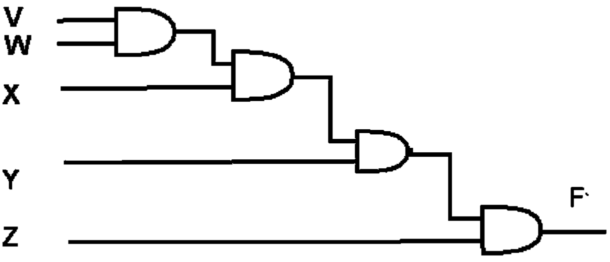


Figure 1: Serial Implementation of Function F

Implementation of the function F in parallel input and serial output connection of AND gates is shown in Figure 2.

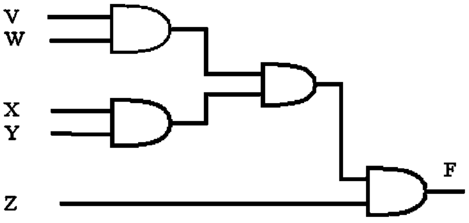


Figure 2: Parallel input and serial output Implementation of Function F

Step 3 of 3

Implementation of the function F in Few Series Input and some other parallel inputs, series output is shown in Figure 3.

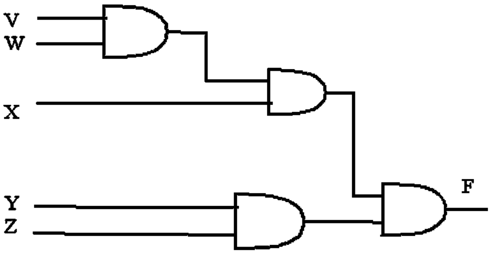


Figure 2: Combined parallel and serial input and serial output Implementation of Function F

Hence, $\boxed{\text{three physically different ways}}$ of implementation are possible for function F using 4 two input AND gates.

Step 1 of 3

In order to prove using switching algebra, switching theorems are used. Switching theorems are given as follows:

$\mathrm{T1} \Rightarrow X+0=X$	$\mathrm{T1'} \Rightarrow X\cdot 1=X$
$\mathrm{T2} \Rightarrow X+1=1$	$\mathrm{T2'} \Rightarrow X\cdot 0=0$
$\mathrm{T3} \Rightarrow X+X=X$	$\mathrm{T3'} \Rightarrow X\cdot X=X$
$\mathrm{T4} \Rightarrow (X')'=X$	
$\mathrm{T5} \Rightarrow X+X'=1$	$\mathrm{T5'} \Rightarrow X\cdot X'=0$
$\mathrm{T6} \Rightarrow X+Y=Y+X$	$\mathrm{T6'} \Rightarrow X\cdot Y=Y\cdot X$
$\mathrm{T7} \Rightarrow (X+Y)+Z=X+(Y+Z)$	$\mathrm{T7'} \Rightarrow (X\cdot Y)\cdot Z=X\cdot(Y\cdot Z)$
$\mathrm{T8} \Rightarrow X\cdot Y+X\cdot Z=X\cdot(Y+Z)$	$\mathrm{T8'} \Rightarrow (X+Y)\cdot(X+Z)=X(Y+Z)$
$\mathrm{T9} \Rightarrow X+X\cdot Y=X$	$\mathrm{T9'} \Rightarrow X\cdot(X+Y)=X$
$\mathrm{T10} \Rightarrow X\cdot Y+X\cdot Y'=X$	$\mathrm{T10'} \Rightarrow (X+Y)\cdot(X+Y')=X$

$$\mathrm{T11} \Rightarrow X\cdot Y+X'\cdot Z+Y\cdot Z=X\cdot Y+X'\cdot Z$$

$$\mathrm{T11'} \Rightarrow (X+Y)\cdot(X'+Z)\cdot(Y+Z)=(X+Y)\cdot(X'+Z)$$

Step 2 of 3

The function of *n*-input AND gate with variables $x_1, x_2, x_3, \ldots, x_n$ can be written as follows,

$$\mathrm{F}=x_1\cdot x_2\cdot x_3 \ldots x_{n-1}\cdot x_n \text{ ...... (1)}$$

The function of (*n*+1)-input AND gate with variables $x_1, x_2, x_3 \ldots x_n$ can be written as follows,

$$\mathrm{F}=x_1\cdot x_2\cdot x_3 \ldots x_{n-1}\cdot x_n\cdot x_{n+1}$$

Consider the two inputs of an (*n*+1)-input AND gate are tied together, let us take *n* and *n*+1 input. Then the function can be written as follows,

$$\mathrm{F}=x_1\cdot x_2\cdot x_3 \ldots x_{n-1}\cdot x_n\cdot x_n$$

Solving the above function using switching theorems as follows,

$$\begin{aligned}\mathrm{F}&=x_1\cdot x_2\cdot x_3 \ldots x_{n-1}\cdot(x_n\cdot x_n) && \textbf{(According to T7')}\\ &=x_1\cdot x_2\cdot x_3 \ldots x_{n-1}\cdot x_n && \textbf{(According to T3')}\end{aligned}$$

Thus from equation (1), we can say that *n*-input AND gate and (*n*+1)-input AND gate are equal if the two inputs of an (*n*+1)-input AND gate are tied together.

Step 3 of 3

The function of *n*-input OR gate with variables $x_1, x_2, x_3 \ldots x_n$ can be written as follows,

$$\mathrm{F}=x_1+x_2+x_3+\ldots+x_{n-1}+x_n \text{ ...... (2)}$$

The function of (*n*+1)-input OR gate with variables $x_1, x_2, x_3 \ldots x_n$ can be written as follows,

$$\mathrm{F}=x_1+x_2+x_3 \ldots x_{n-1}+x_n+x_{n+1}$$

Consider the two inputs of an (*n*+1)-input OR gate are tied together, let us take *n* and *n*+1 input. Then the function can be written as follows,

$$\mathrm{F}=x_1+x_2+x_3 \ldots x_{n-1}+x_n+x_n$$

Solving the above function using switching theorems as follows,

$$\begin{aligned}\mathrm{F}&=x_1+x_3+x_3 \ldots x_{n-1}+(x_n+x_n) && \textbf{(According to T7)}\\ &=x_1+x_2+x_3 \ldots x_{n-1}+x_n && \textbf{(According to T3)}\end{aligned}$$

Thus from equation (2), we can say that *n*-input OR gate and (*n*+1)-input OR gate are equal if the two inputs of an (*n*+1)-input AND gate are tied together.

Step 1 of 3 **4.028E**

To prove the DeMorgan's theorems using finite induction, we need to solve by two steps. First step is basis step, considering n=2 and proving the theorem using axioms. The axioms are given by as follows:

$A1 \Rightarrow X=0$ if $X=1$	$A1' \Rightarrow X=1$ if $X=0$
$A2 \Rightarrow X=0$, then $X'=1$	$A2' \Rightarrow X=1$, then $X'=0$
$A3 \Rightarrow 0 \cdot 0=0$	$A3' \Rightarrow 1+1=1$
$A4 \Rightarrow 1 \cdot 1=1$	$A4' \Rightarrow 0+0=0$
$A5 \Rightarrow 0 \cdot 1=1 \cdot 0=0$	$A5' \Rightarrow 1+0=0+1=1$

Second step is induction step, assuming the theorem is true for n variable and proving the same for n+1 variable. For this step switching theorems are used. Switching theorems are given as follows:

$T1 \Rightarrow X+0=X$	$T1' \Rightarrow X \cdot 1=X$
$T2 \Rightarrow X+1=1$	$T2' \Rightarrow X \cdot 0=0$
$T3 \Rightarrow X+X=X$	$T3' \Rightarrow X \cdot X=X$
$T4 \Rightarrow (X')'=X$	
$T5 \Rightarrow X+X'=1$	$T5' \Rightarrow X \cdot X'=0$
$T6 \Rightarrow X+Y=Y+X$	$T6' \Rightarrow X \cdot Y=Y \cdot X$
$T7 \Rightarrow (X+Y)+Z=X+(Y+Z)$	$T7' \Rightarrow (X \cdot Y) \cdot Z=X \cdot (Y \cdot Z)$
$T8 \Rightarrow X \cdot Y+X \cdot Z=X \cdot (Y+Z)$	$T8' \Rightarrow (X+Y) \cdot (X+Z)=X(Y+Z)$
$T9 \Rightarrow X+X \cdot Y=X$	$T9' \Rightarrow X \cdot (X+Y)=X$
$T10 \Rightarrow X \cdot Y+X \cdot Y'=X$	$T10' \Rightarrow (X+Y) \cdot (X+Y')=X$

$T11 \Rightarrow X \cdot Y+X' \cdot Z+Y \cdot Z=X \cdot Y+X' \cdot Z$

$T11' \Rightarrow (X+Y) \cdot (X'+Z) \cdot (Y+Z)=(X+Y) \cdot (X'+Z)$

Step 2 of 3

The DeMorgan's theorem is as follows:

$$(X_1 \cdot X_2 \cdot ... \cdot X_n)' = X_1' + X_2' + ... + X_n'$$
$$(X_1 + X_2 + ... + X_n)' = X_1' \cdot X_2' \cdot ... \cdot X_n'$$

Basis Step:

Considering n=2 then DeMorgan's theorem is written as follows:

$$(X_1 \cdot X_2)' = X_1' + X_2'$$
$$(X_1 + X_2)' = X_1' \cdot X_2'$$

Proof for the above equations using perfect induction method:

X_1	X_2	X_1'	X_2'	$(X_1 \cdot X_2)'$	$X_1'+X_2'$
0	0	1	1	1	1
0	1	1	0	1	1
1	0	0	1	1	1
1	1	0	0	0	0

Table 1: Perfect induction method for theorem $(X_1 \cdot X_2)' = X_1' + X_2'$

Thus the theorem $(X_1 \cdot X_2)' = X_1' + X_2'$ is proved using perfect induction method.

Then for second one is given by as follows:

X_1	X_2	X_1'	X_2'	$(X_1 + X_2)'$	$X_1' \cdot X_2'$
0	0	1	1	1	1
0	1	1	0	0	0
1	0	0	1	0	0
1	1	0	0	0	0

Table 2: Perfect induction method for theorem $(X_1 + X_2)' = X_1' \cdot X_2'$

Thus the theorem $(X_1 + X_2)' = X_1' \cdot X_2'$ is proved using perfect induction method.

Step 3 of 3

The second step is induction step, assuming the theorem is true for n variable and proving the same for n+1 variable.

Thus for n+1 variable, the DeMorgan's theorem is written by

$$(X_1 \cdot X_2 \cdot ... \cdot X_n \cdot X_{n+1})' = X_1' + X_2' + ... + X_n' + X_{n+1}'$$
$$(X_1 + X_2 + ... + X_n + X_{n+1})' = X_1' \cdot X_2' \cdot ... \cdot X_n' \cdot X_{n+1}'$$

Proof for n+1 variable is given by,

$$\begin{aligned}(X_1 \cdot X_2 \cdot ... \cdot X_n \cdot X_{n+1})' &= ((X_1 \cdot X_2 \cdot ... \cdot X_n) \cdot X_{n+1})' \\ &= (X_1 \cdot X_2 \cdot ... \cdot X_n)' + X_{n+1}' \quad \text{(Since theorem is proved for two variable)} \\ &= X_1' + X_2' + ... + X_n' + X_{n+1}' \text{ (Since assumed that theorem is true for n variable)}\end{aligned}$$

For second one,

$$\begin{aligned}(X_1 + X_2 + ... + X_n + X_{n+1})' &= ((X_1 + X_2 + ... + X_n) + X_{n+1})' \\ &= (X_1 + X_2 + ... \cdot X_n)' \cdot X_{n+1}' \quad \text{(Since theorem is proved for two variable)} \\ &= X_1' \cdot X_2' \cdot ... \cdot X_n' \cdot X_{n+1}' \text{ (Since assumed that theorem is true for n variable)}\end{aligned}$$

Thus the DeMorgan's theorem is proved by finite induction method.

Step 1 of 2

4.029E

The internal realization of a TTL NOR gate is shown in Figure 1.

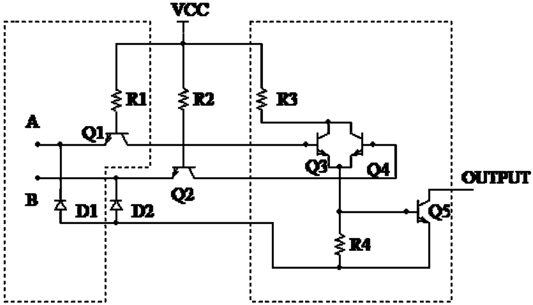


Figure1: Internal realization of a TTL NOR gate

Step 2 of 2

The logic symbol more closely to the internal realization of a TTL NOR gate is AND gate with inverted inputs. As shown in figure, the two inputs are given to invertor and inverted inputs are connected to AND gate. The inverter of a TTL realization is shown in dotted line (green color).

Step 1 of 2 **4.030E**

In order to prove the function using switching algebra, switching theorems are used. The switching algebra theorems are shown in Figure 1:

$T1 \Rightarrow X+0=X$ $\quad T1' \Rightarrow X\cdot 1=X$

$T2 \Rightarrow X+1=1$ $\quad T2' \Rightarrow X\cdot 0=0$

$T3 \Rightarrow X+X=X$ $\quad T3' \Rightarrow X\cdot X=X$

$T4 \Rightarrow (X')'=X$

$T5 \Rightarrow X+X'=1$ $\quad T5' \Rightarrow X\cdot X'=0$

$T6 \Rightarrow X+Y=Y+X$ $\quad T6' \Rightarrow X\cdot Y=Y\cdot X$

$T7 \Rightarrow (X+Y)+Z=X+(Y+Z)$ $\quad T7' \Rightarrow (X\cdot Y)\cdot Z=X\cdot(Y\cdot Z)$

$T8 \Rightarrow X\cdot Y+X\cdot Z=X\cdot(Y+Z)$ $\quad T8' \Rightarrow (X+Y)\cdot(X+Z)=X(Y+Z)$

$T9 \Rightarrow X+X\cdot Y=X$ $\quad T9' \Rightarrow X\cdot(X+Y)=X$

$T10 \Rightarrow X\cdot Y+X\cdot Y'=X$ $\quad T10' \Rightarrow (X+Y)\cdot(X+Y')=X$

$T11 \Rightarrow X\cdot Y+X'\cdot Z+Y\cdot Z=X\cdot Y+X'\cdot Z$

$T11' \Rightarrow (X+Y)\cdot(X'+Z)\cdot(Y+Z)=(X+Y)\cdot(X'+Z)$

Figure 1

Step 2 of 2

The function to solve by switching algebra is given by,

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

Solving the function using switching theorems contains following steps,

$$\begin{aligned}F&=B'\cdot C+A\cdot C\cdot D'+A'\cdot C+E\cdot B'+(A\cdot E+C\cdot E)\cdot(A'+D') \quad \textbf{(According to T8)}\\ &=B'\cdot C+A\cdot C\cdot D'+A'\cdot C+E\cdot B'+(A'\cdot A\cdot E+A'\cdot C\cdot E+A\cdot D'\cdot E+D'\cdot C\cdot E)\\ &=B'\cdot C+A\cdot C\cdot D'+A'\cdot C+E\cdot B'+A'\cdot C\cdot E+A\cdot D'\cdot E+D'\cdot C\cdot E \quad \textbf{(According to T5)}\end{aligned}$$

Arrange the terms according to the common terms. Then,

$$\begin{aligned}F&=A\cdot C\cdot D'+A\cdot D'\cdot E+A'\cdot C+A'\cdot C\cdot E+B'\cdot C+E\cdot B'+D'\cdot C\cdot E\\ &=A\cdot D'\cdot(C+E)+A'\cdot C\cdot(1+E)+B'\cdot(C+E)+D'\cdot C\cdot E\\ &=A\cdot D'\cdot(C+E)+A'\cdot C+B'\cdot(C+E)+D'\cdot C\cdot E \quad \textbf{(According to T2)}\\ &=(C+E)\cdot(A\cdot D'+B')+A'\cdot C+D'\cdot C\cdot E\end{aligned}$$

Thus, using the switching theorems the function is reduced as follows,

$$\boxed{F=(C+E)\cdot(A\cdot D'+B')+A'\cdot C+D'\cdot C\cdot E}.$$

Step 1 of 2

Shannon's expansion theorem is given by,

$$F(X_1, X_2, \ldots X_{n-1}, X_n) = X_1 \cdot F(1, X_2 \ldots X_n) + X_1' \cdot F(0, X_2 \ldots X_n)$$

A Variable X_1 can hold two values 1 & 0 or we can say true or complementary form and the function can appear any of one form.

Shannon's expansion theorem can be proved by perfect induction method as follows,

Case 1: When X_1 =0 value is substituted in Shannon's expansion theorem, both left hand side(LHS) of the equation and right hand side (RHS) of the equation should be equal.

$$\begin{aligned} F(0, X_2, \ldots X_n) &= 0 \cdot F(1, X_2 \ldots X_n) + 1 \cdot F(0, X_2 \ldots X_n) \\ &= 0 + F(0, X_2 \ldots X_n) \quad (\textit{Since} \quad 0 \cdot X = 0 \quad \textit{and} \quad 1 \cdot X = X) \\ &= F(0, X_2 \ldots X_n) \quad (\textit{Since} \quad 0 + X = X) \end{aligned}$$

Thus the LHS and RHS are equal, and Shannon's expansion theorem is proved.

Step 2 of 2

Case2: When X_1 =1 value is substituted in Shannon's expansion theorem, both left hand side(LHS) of the equation and right hand side (RHS) of the equation should be equal.

$$\begin{aligned} F(1, X_2, \ldots X_n) &= 1 \cdot F(1, X_2 \ldots X_n) + 0 \cdot F(0, X_2 \ldots X_n) \\ &= F(1, X_2 \ldots X_n) + 0 \quad (\textit{Since} \quad 0 \cdot X = 0 \quad \textit{and} \quad 1 \cdot X = X) \\ &= F(1, X_2 \ldots X_n) \quad (\textit{Since} \quad 0 + X = X) \end{aligned}$$

Thus, the LHS and RHS are equal, and Shannon's expansion theorem is proved.

Step 1 of 2

4.032E

The general form of Shannon's expansion theorem is,

$$F(X_1,X_2,...X_{n-1},X_n) = X_1 \cdot F(1,X_2...X)_n + X_1' \cdot F(0,X_2...X_n) \quad ...... (1)$$

The other form or duality of Shannon's expansion theorem is,

$$F(X_1,X_2,...X_{n-1},X_n) = \begin{Bmatrix} [X_1 + F(0,X_2,...X_{n-1},X_n)] \cdot \\ [X_1' + F(1,X_2,...X_{n-1},X_n)] \end{Bmatrix} \quad ...... (2)$$

Step 2 of 2

Shannon's expansion theorem can be done by more than one variable. For example from equation (1) expanding the function with respect to X_1 and X_2 as follows,

$$F(X_1,X_2,...X_{n-1},X_n) = \begin{Bmatrix} X_1 \cdot X_2 \cdot F(1,1,...X_n) + X_1' X_2 \cdot F(0,1,...X_n) + \\ X_1 \cdot X_2' \cdot F(1,0,...X_n) + X_1' \cdot X_2' \cdot F(0,0...X_n) \end{Bmatrix}$$

Similarly from the equation (2),

$$F(X_1,X_2,...X_{n-1},X_n) = \begin{Bmatrix} [X_1 + X_2 + F(0,0,...X_n)] \cdot [X_1' + X_2 + F(1,0,...X_n)] \cdot \\ [X_1 + X_2' \cdot F(0,1,...X_n)] \cdot [X_1' + X_2' + F(1,1...X_n)] \end{Bmatrix}$$

Similarly when expand the equation (1) or equation (2) with respect to three terms X_1 , X_2 , and X_3 then we get sum or product of eight terms.

By observing the results of the Shannon's expansion theorem when we expanding the function with respect to 1,2, and 3 variables or pull out 1,2, and 3 variables then the logic function can be expressed as a sum or product of 2, 2^2 , and 2^3 terms.

Thus, in general when expanding the function with respect to *i* variable or pull out *i* variable then the logic function can be expressed as a sum or product of 2^i terms.

Step 1 of 2

The general form of Shannon's expansion theorem is,

$$F(X_1, X_2, \dots X_{n-1}, X_n) = X_1 \cdot F(1, X_2 \dots X_n) + X_1' \cdot F(0, X_2 \dots X_n) \quad \dots\dots (1)$$

The other form or duality of Shannon's expansion theorem is,

$$F(X_1, X_2, \dots X_{n-1}, X_n) = \begin{Bmatrix} [X_1 + F(0, X_2, \dots X_{n-1}, X_n)] \cdot \\ [X_1' + F(1, X_2, \dots X_{n-1}, X_n)] \end{Bmatrix} \quad \dots\dots (2)$$

Step 2 of 2

The Shannon's expansion itself has in a form of canonical sum or product representation of logic function. Write the general Shannon's expansion for the three variable functions.

$$F(X_1, X_2, X_3) = X_1 \cdot F(1, X_2, X_3) + X_1' \cdot F(0, X_2, X_3)$$

When Shannon's expansion is done in terms of all n variables then the function gives the canonical sum or product representation of logic function. For example, take one function.

$$F(X_1, X_2, X_3) = X_1X_2 + X_1X_3' + X_1'X_3$$

Expand the function using the Shannon's expansion theorem with respect to the variable X_1.

$$\begin{aligned} F(X_1, X_2, X_3) &= X_1 \cdot F(1, X_2, X_3) + X_1' \cdot F(0, X_2, X_3) \\ &= X_1(1 \cdot X_2 + 1 \cdot X_3' + 0 \cdot X_3) + X_1' \cdot (0 \cdot X_2 + 0 \cdot X_3' + 1 \cdot X_3) \\ &= X_1(X_2 + X_3') + X_1' \cdot X_3 \end{aligned}$$

Thus, the Shannon's expansion leads to the canonical sum and canonical product representations of logic functions.

Step 1 of 2

A 2-input XOR gate gives the output 1 if one of its input is 1. Thus the truth table for the XOR gate is given by as follows. Let X &Y are the inputs and Z is the output of the XOR gate.

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

Table 1: Truth table for XOR gate

Step 2 of 2

From the truth table, the sum of product expression is given by,

$$Z = X' \cdot Y + X \cdot Y'.$$

The AND-OR circuit of XOR gate is shown in Figure 1:

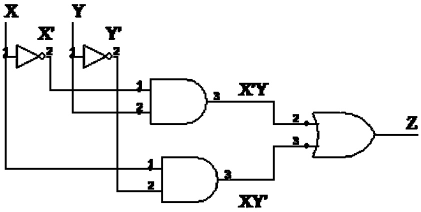


Figure 1: AND-OR circuit for XOR gate

Step 1 of 2

A 2-input XNOR gate gives the output 1 if both the inputs are equal. Thus the truth table for the XNOR gate is given by as follows. Let X &Y are the inputs and Z is the output of the XOR gate. The truth table for XNOR gate is shown in Table 1:

X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	1

Table 1

Step 2 of 2

From the truth table, the sum of product expression is given by,

$$Z = X' \cdot Y' + X \cdot Y.$$

The AND-OR circuit of XNOR gate is shown in Figure 1:

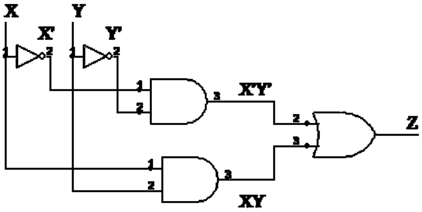


Figure 1: AND-OR circuit for XNOR gate

Step 1 of 2 4.038E

When one of the input in 2-input XNOR gate is grounded, then it will act as an inverter. The truth table for the XOR inverter is shown in Table 1:

X(input)	Y(grounded)	Z(output)
0	0	1
1	0	0

Table 1

Step 2 of 2

One of the input of XNOR gate, let's take Y is grounded.

When $\mathbf{X=0}$, both the inputs are equal thus the output of XNOR gate is 1. When $\mathbf{X=1}$, both the inputs are not equal thus the output of XNOR gate is 0.

From the Table 1, we can analyze that, the XNOR gate is perfectly acts as an inverter.

Thus, the designer realize the inverter function using the available $\boxed{\text{XNOR}}$ gate.

Step 1 of 4

The 2-input NAND gate and 2-input NOR gate are complete set of logic gates or we can say that they are universal gates. That means any function can be realized from these gates. For example 2-input NAND gate can be realized in to AND, OR, NOT and XOR.

The diagram for AND gate using NAND gate is shown in Figure 1:

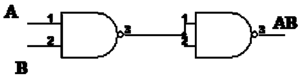


Figure 1

Step 2 of 4

The OR gate using NAND gate is shown in Figure 2:

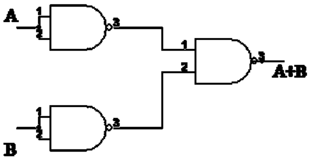


Figure 2: OR gate using NAND gate

Step 3 of 4

The NOT gate realization using NAND gate is shown in Figure 3:

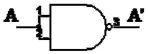


Figure 3: NOT gate using NAND gate

Step 4 of 4

The XOR gate using NAND gate is shown in Figure 4:

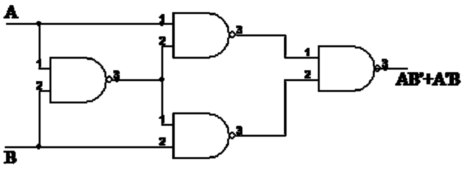


Figure 4: XOR gate using NAND gate

Thus, it has been proved that a 2-input NAND gate is forms a complete set of logic gates.

Step 1 of 7

Consider the following 2-input AND gate which have one input in inverted form:

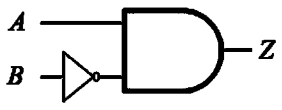


Figure 1

Step 2 of 7

In Figure 1, *A* and *B* are inputs to the gate whereas *Z* is the output. The gate in Figure 1 can also represented as shown in Figure 2.

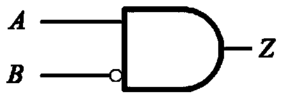


Figure 2

Step 3 of 7

This type of gate represented in Figure 2 is called inhibit gate or anti-coincidence circuit or enable/disable circuit. Because, one input acts as a controller for the circuit that is, according to the input *B* the gate enables or disables.

If $B=0$, the gate conducts and the output varies according to the input *A* else if $B=1$, then the gate does not conduct and it gives the result 0, whatever will be the *A* value. Thus this is called as inhibitor circuit or enable/disable circuit. Multiple input inhibitor circuit also possible and in a single gate it is possible to have more than one inhibited terminal. The standard AND gate does not be an inhibitor circuit and it is called as uninhibited.

The 2-input AND gate with one input inverted or inhibited does not form a complete set because using this inhibited gate it is not possible to implement AND, OR and NOT gates. Because if the inhibit gate is connected in any order, the result will be same as shown in the Figure 1 or value 0. The output of 2-input AND inhibit gate gives the result as $\mathbf{Z=A\cdot B'}$.

Step 4 of 7

Try to connect two inhibited gates as shown in Figure 3.

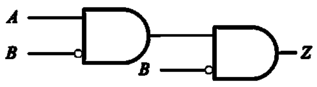


Figure 3

Step 5 of 7

Find the output of circuit in Figure 3.

$$\begin{aligned}\mathbf{Z_1} &= \mathbf{(A\cdot B')\cdot B'}\\ &= \mathbf{A\cdot(B'\cdot B')} \quad \text{since } (x\cdot y)\cdot z = x\cdot(y\cdot z)\\ &= \mathbf{A\cdot B'} \quad \text{since } x\cdot x = x\end{aligned}$$

Thus, the output of the gate in Figure 3 is same as the output of the gate in Figure 1.

Step 6 of 7

Try to connect three inhibited gates as shown in Figure 4.

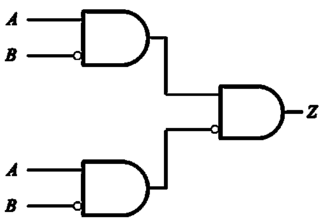


Figure 4

Step 7 of 7

Find the output of circuit in Figure 4.

$$\begin{aligned}\mathbf{Z} &= \mathbf{(A\cdot B')\cdot(A\cdot B')'}\\ &= \mathbf{(A\cdot B')\cdot(A'+B)} \quad \text{since } (x\cdot y)' = x'+y'\\ &= \mathbf{A\cdot B'\cdot A' + A\cdot B'\cdot B}\\ &= \mathbf{0} \quad \text{since } x\cdot x' = 0\end{aligned}$$

Thus, observe that the result of the inhibit gate connected in any manner is same as the result of gate in Figure 1 or value 0. Hence, the 2-input AND gate with one input inverted or inhibited gate does not form a complete set.

Step 1 of 1

4.042E

The 2-input XNOR gate alone does not form a complete set of logic gates. The complete set of logic gates means, XNOR gate have to design AND, OR and NOT gate but it is not possible to possible to realize all the gates using XNOR gate alone. XNOR gate along with OR gate form a complete set of logic gates.

For example:

• Realization of NOT gate:

$$\mathbf{NOT(X)=(0\ XNOR\ X)}$$

• Realization of AND gate:

$$\begin{aligned}\mathbf{X\ AND\ Y} &= \mathbf{NOT(NOT(X)\ OR\ NOT(Y))}\\ &= \mathbf{\{0\ XNOR[(0\ XNOR\ x)\ OR\ (0\ XNOR\ y)]\}}\end{aligned}$$

Step 1 of 4

Consider the BUT gate contains 4 inputs and 2 outputs:

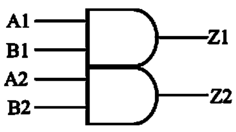


Figure 1: BUT gate logic schematic diagram

Step 2 of 4

The BUT gate function is symmetric with respect to the A and B inputs. BUT gate function logic is

• The output Z1 is high, when inputs A1, B1 are high and inputs A2, B2 are not equal to 1.

• The output Z2 is high, when inputs A2, B2 are high and inputs A1, B1 are not equal to 1.

The truth table for the BUT gate function:

A1	B1	A2	B2	Z1	Z2
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	0	1
0	1	0	0	0	0
0	1	0	1	0	0
0	1	1	0	0	0
0	1	1	1	0	1
1	0	0	0	0	0
1	0	0	1	0	0
1	0	1	0	0	0
1	0	1	1	0	1
1	1	0	0	1	0
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	0	0

Table 1: Truth table of BUT gate

Step 3 of 4

The simplified logic expression for the Z1 after applying the Karnaugh's map:

$$\begin{aligned} Z1 &= A1 \cdot B1 \cdot A2' + A1 \cdot B1 \cdot B2' \\ &= A1 \cdot B1 \cdot (A2' + B2') \end{aligned}$$

The simplified logic expression for the Z2 after applying the Karnaugh's map:

$$\begin{aligned} Z2 &= A2 \cdot B2 \cdot B1' + A2 \cdot B2 \cdot A1' \\ &= A2 \cdot B2 \cdot (B1' + A1') \end{aligned}$$

Step 4 of 4

The following is the logic diagram BUT gate drawing using the AND-OR gates and inverters:

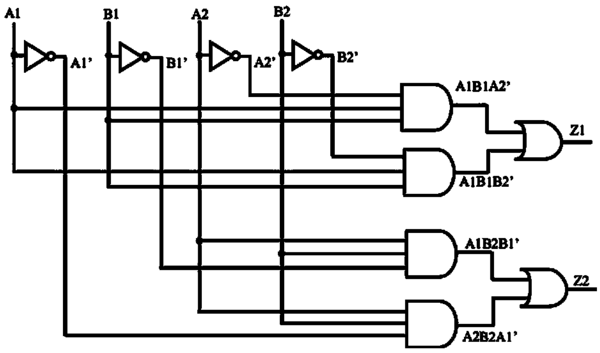


Figure 2: Equivalent AND-OR-INVERTER circuit for BUT gate

4.046E

Step 1 of 14

A digital system frequently its input combination will have two or more high inputs out of its three inputs.

^nC_r

Where

n is the total number of input signals

r is the required conditional number of inputs.

Therefore,

$$^nC_r = \frac{n!}{r!(n-r)!}$$

$$^3C_2 + {}^3C_3 = \frac{3\times2\times1}{2!(3-2)!} + \frac{3\times2\times1}{3\times2\times1}$$
$$= 3+1$$
$$= 4$$

Since the system needs two and more input signals we are summing the possible combinations of two and more inputs. Hence, out of eight only four combinations are most useful.

Step 2 of 14

Logical functions are accepted as different only if their outputs must differ at least by one within four frequently occurring input combination. So different logical function with the four frequently occurring input combinations are $16(2^4)$. Hence out of the $256(2^8)$ logical function with three input variables, only $\boxed{16}$ different logic functions can be obtained because most of the times, two or more of its inputs are always high. In order to, obtain the logical expression for the different functions, assuming don't care conditions for the rarely occurring input combinations

input			Possible output functions															
X	Y	Z	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}
0	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0	0	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0	1	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
1	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1	0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Table 1: Truth table of possible output functions with frequently occurring input combination

Step 3 of 14

To obtain the simplified logic function for the possible output functions we are using the karnaugh map. For the function F_1 only the don't-cares and 0 is present

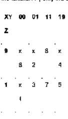


Figure 1: K-map for the function F_1

Hence, the simplified expression is $\boxed{F_1 = 0}$.

Step 4 of 14

Similarly for the function F_2 we can obtain the k map as follows

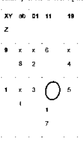


Figure 2: K map for the function F_2

Hence, the simplified expression is $\boxed{F_2 = XYZ}$.

Step 5 of 14

For the function F_3, we have to plot 1 in the cell 6 and hence we can group it as follows

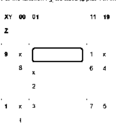


Figure 3: K map for the function F_3

Hence, the simplified expression is $\boxed{F_3 = YZ'}$.

Step 6 of 14

For the function F_4, we have to plot 1 in the cells 6,7 and hence we can group it as follows

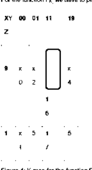


Figure 4: K-map for the function F_4

Hence, the simplified expression is $\boxed{F_4 = XY}$.

Step 7 of 14

For the function F_5, we have to plot 1 in the cells corresponding to the minterm 5 and hence we can group it as follows.

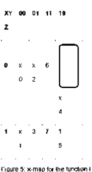


Figure 5: K-map for the function F_5

Hence the simplified expression is $\boxed{F_5 = XY'}$.

Step 8 of 14

For the function F_6, we have to plot 1 in the cells corresponding to the minterm 5, 7

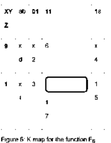


Figure 6: K map for the function F_6

Hence the simplified expression is $\boxed{F_6 = XZ}$.

Step 9 of 14

Same as before, we are going to plot 1 in the cells corresponding to the minterm 5,6 in the function F_7.

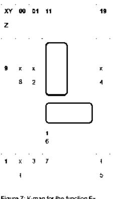


Figure 7: K-map for the function F_7

Hence, the simplified expression is $\boxed{F_7 = XZ' + XY'}$

Step 10 of 14

For the function F_8, we have to plot 1 in the cells corresponding to the minterm 5,6,7 we will obtain the k-map as shown just below

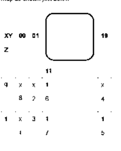


Figure 8: K-map for the function F_8

Hence, the simplified expression is $\boxed{F_8 = X}$.

For the function F_9, we have to plot 1 in the cell corresponding to the minterm 3.

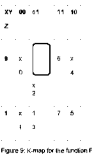


Figure 9: K-map for the function F_9

Hence, the simplified expression is $\boxed{F_9 = X'Y}$

Step 11 of 14

For the function F_{10}, we have to plot 1 in the cells corresponding to the minterms 3,7 and hence

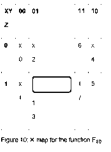


Figure 10: K map for the function F_{10}

Hence, the simplified expression is $\boxed{F_{10} = ZY}$.

Step 12 of 14

For the function F_{11}, we have to plot 1 in the cells corresponding to the minterms 3,6.

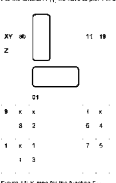


Figure 11: K-map for the function F_{11}

Hence, the simplified expression is $\boxed{F_{11} = X'Z + YZ'}$

Step 13 of 14

For the function F_{12}, we have to plot 1 in the cells corresponding to the minterms 3, 6, 7

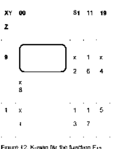


Figure 12. K-map for the function F_{12}

Hence, the simplified expression is $\boxed{F_{12} = Y}$.

Step 14 of 14

For the function F_{13}, we have to plot 1 in the cells corresponding to the minterms 3,5

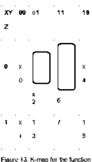


Figure 13. K-map for the function F_{13}

Hence, the simplified expression is $\boxed{F_{13} = XY' + YX'}$.

For the function F_{14}, we have to plot 1 in the cells corresponding to the minterms 3,5,7

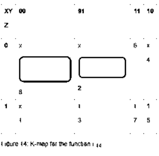


Figure 14: K-map for the function F_{14}

Hence, the simplified expression is $\boxed{F_{14} = YZ + XZ}$

For the function F_{15}, we have to plot 1 in the cells corresponding to the minterms 3,5,6

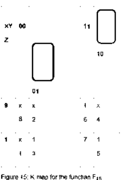


Figure 15: K map for the function F_{15}

Hence, the simplified expression is $\boxed{F_{15} = X'Y + XZ' + XY'}$.

Finally for function F_{16}, we have to plot 1 in the cells corresponding to the minterms 3,5,6,7

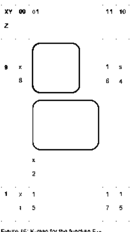


Figure 16: K-map for the function F_{16}

Hence, the simplified expression is $\boxed{F_{16} = Y + X}$.

Step 1 of 7 **4.047E**

(a)

Write the function.

$\mathbf{F=X}$ ...... (1)

Dual for the given function F is obtained by replacing $\mathbf{+ \text{ with } \bullet, \bullet \text{ with } +}$ in the function F

$\mathbf{F^D=X}$ ...... (2)

Compare equation (1) and (2).

$\mathbf{F=F^D}$

Hence, the function F is **self-dual logic function.**

Step 2 of 7

(b)

Write the second function.

$\mathbf{F=\sum_{X,Y,Z}(1,2,5,7)}$ ...... (3)

$\mathbf{F=X'Y'Z+X'YZ'+XY'Z+XYZ}$

The function (3) denotes that the function F produces 1 output for the minterms 1,2,5,7. Hence we can say that it produces 0 outputs for the minterms 0,3,4,6.

Dual for the given function F is obtained by replacing $\mathbf{+ \text{ with } \bullet, \bullet \text{ with } +}$ in the function F

$\mathbf{F^D=(X'+Y'+Z)\bullet(X'+Y+Z')\bullet(X+Y'+Z)\bullet(X+Y+Z)}$

$\mathbf{F^D=\prod_{X,Y,Z}(0,2,5,6)}$ ...... (4)

Function in equation (4) produces 0 outputs for the minterms 0,2,5,6 in the truth table. Compare equations (3) and (4) we can say that $\mathbf{F\neq F^D}$. Hence, the given function F is not a **self-dual logic function**.

Step 3 of 7

(c)

Write the function.

$\mathbf{F=X'\cdot Y'\cdot Z'+X\cdot Y'\cdot Z'+X\cdot Y}$ ...... (5)

Express the equation (5) in standard form.

$$\begin{aligned}\mathbf{F}&\mathbf{=X'\cdot Y'\cdot Z'+X\cdot Y'\cdot Z'+X\cdot Y\cdot(Z+Z')}\\ &\mathbf{=X'\cdot Y'\cdot Z'+X\cdot Y'\cdot Z'+X\cdot Y\cdot Z+X\cdot Y\cdot Z`}\end{aligned}$$

Therefore,

$\mathbf{F=\sum_{X,Y,Z}(0,4,6,7)}$ ...... (6)

Equation (6) denotes that the function F produces 1 output for the minterms 0,4,6,7. Hence we can say that it produces 0 outputs for the minterms **1**,2,3,5.

Dual for the given function F is obtained by replacing $\mathbf{+ \text{ with } \bullet, \bullet \text{ with } +}$ in the function F given by equation (6).

$\mathbf{F^D=(X'+Y'+Z')\bullet(X+Y'+Z')\bullet(X+Y+Z)\bullet(X+Y+Z')}$

$\mathbf{F^D=\prod_{X,Y,Z}(0,1,3,7)}$ ...... (7)

Function in equation (7) produces 0 outputs for the minterms 0,1,3,7 in the truth table. We can say that $\mathbf{F\neq F^D}$. Function F does not satisfy the condition $\mathbf{F=F^D}$.

Hence, the function F is not a **self-dual logic function.**

Step 4 of 7

(d)

Write the function.

$\mathbf{F=(W\cdot(X\oplus Y\oplus Z\oplus W'))+(X\oplus Y\oplus Z)}$

The function will produce high output in positive logic whenever strictly one of the inputs X, Y, Z must be 1 or at the instant W is 1and strictly one of the inputs X, Y, Z is 1. Hence we can complete the truth table by the following logic. **First condition says that strictly one of the inputs X, Y, Z needs to be high which produces output 1 for the minterms 1, 2, 4, 9, 10, 12. Second condition insists the output to be high whenever W is 1 and strictly one of the inputs X, Y, Z needs to be high. Hence it will produce 1 output for the minterms 9, 10, 12. First condition overlaps the second one.** Since we are having OR operation between the conditions, if any one of the conditions gets satisfied by the inputs, high output will be produced.

$\mathbf{F=\sum_{W,X,Y,Z}(1,2,4,9,10,12)}$

$\mathbf{F=W'X'Y'Z+W'X'YZ'+W'XY'Z'+WX'Y'Z+WX'YZ'+WXY'Z'}$ ...... (8)

From equation (8) function F produces **1** output for the minterms 1,2,4,9,10,12 and 0 for the minterms 0,3,5,6,7,8,11,13,14,15.

Step 5 of 7

Dual for the given function F is obtained by replacing $\mathbf{+ \text{ with } \bullet, \bullet \text{ with } +}$ in the function F given by equation (8).

$$\mathbf{F=\left\{\begin{matrix}(W'+X'+Y'+Z)\bullet(W'+X'+Y+Z')\bullet(W'+X+Y'+Z')\bullet\\ (W+X'+Y'+Z)\bullet(W+X'+Y+Z')\bullet(W+X+Y'+Z')\end{matrix}\right\}}$$

$\mathbf{F=\prod_{W,X,Y,Z}(3,5,6,11,13,14)}$ ...... (9)

Function in equation (9) produces 0 outputs for the minterms 3, 5, 6,11,13,14 in the truth table. Equation (9) gives 1 output for the minterms 0,1,2,4,7,8,9,10,12,15. With respect to F, F^D gives 1 output 0, 7, 8, 15 instead of 0. We can say that $\mathbf{F\neq F^D}$. Hence, the function F is not a **self-dual logic function**.

(e)

We have been given a function F such that it will produce high output only if three or more inputs are high in positive logic. Hence for five input literals, possible number of input combinations having 3 or more ones is given by,

$\mathbf{nC_r}$

Where,

n is the number of input literals

r is the required number of positive logic high

Since we need more than three inputs to be high in the total six input literals, we can write

$\mathbf{6C_3+6C_4+6C_5+6C_6}$ ...... (10)

By means of the formula,

$$\begin{aligned}&\mathbf{nC_r=nC_{n-r}}\\ &\mathbf{nC_1=n}\\ &\mathbf{nC_n=1}\end{aligned}$$

Rewrite the equation (10).

$$\begin{aligned}\mathbf{6C_3+6C_4+6C_5+6C_6}&\mathbf{=6C_3+6C_2+6C_1+6C_6}\\ &\mathbf{=\frac{6\times5\times4}{3\times2}+\frac{6\times5}{2}+6+1}\\ &\mathbf{=20+15+6+1}\\ &\mathbf{=42}\end{aligned}$$

Out of 2^6 input combinations, function F produces high 1 output for the 42 combinations where number of ones in the input is 3 or more. For the Remaining 22 combinations output is 0 where the number of Positive logic high input is less than 3 otherwise we can say that number of zeroes is greater than $\mathbf{6-3=3}$.

When we take the dual of the function F, 42 input combinations where the number of negative logic high (0) is 3 or more, will produce 0 output. Remaining 22 input combinations where the number of negative logic Low (1) is greater than 3 will produce **1** output. Hence, in the input combinations where number of 1 is equal to number of zeros function F will produce 1 output, whereas function F^D will produce 0 output. Thus we can say that $\mathbf{F\neq F^D}$.

Hence, the function F is not a **self-dual logic function**

Step 6 of 7

(f)

The function F with nine input variables will produce high output only if five or more of its inputs are high in positive logic. Hence for nine input literals, possible number of input combinations having 5 or more ones is given by,

$\mathbf{nC_r}$

Where,

n is the number of input literals

r is the required number of positive logic high

Since we need more than five inputs to be high in the available nine input literals, we can write

$\mathbf{9C_5+9C_6+9C_7+9C_8+9C_9}$ ...... (11)

By means of the formula

$$\begin{aligned}&\mathbf{nC_r=nC_{n-r}}\\ &\mathbf{nC_1=n}\\ &\mathbf{nC_n=1}\end{aligned}$$

Step 7 of 7

Rewrite the equation (11).

$$\begin{aligned}\mathbf{9C_5+9C_6+9C_7+9C_8+9C_9}&\mathbf{=9C_4+9C_3+9C_2+9C_1+9C_0}\\ &\mathbf{=\frac{9\times8\times7\times6}{4\times3\times2}+\frac{9\times8\times7}{3\times2}+\frac{9\times8}{2}+9+1}\\ &\mathbf{=126+84+36+9+1}\\ &\mathbf{=256}\end{aligned}$$

Out of 2^9 input combinations, function F produces positive logic high (1) output for the 256 combinations where number of ones in the input is 5 or more. For the remaining 256 combinations output is positive logic Low (0) where the number of Positive logic high input (1) is less than 5 otherwise we can say that number of Positive logic Low input (0) is greater than $\mathbf{9-5=4}$.

When we take the dual for the function F, 256 input combinations where the number of negative logic high (0) is 5 or more, which means that greater than 4, will produce negative logic high (0) output. Remaining 256 input combinations where the number of negative logic Low (1) is greater than 4 which mean that 5 or more will produce negative logic Low (1) output. Hence we can say that output remains same for all the possible input combinations. Thus we can say that $\mathbf{F=F^D}$.

Hence, the function **F is a self-dual logic function.**

Step 1 of 1

4.048E

Self-dual functions have to satisfy the following condition.

$$F = F^D$$

In order to satisfy the condition the function F strictly gives same output value for half of the possible input combination.

Otherwise we can say that if and only if the numbers of high and low outputs are equal it can be a self – dual function. Hence, the function F while representing in standard form will have $\frac{2^n}{2}$ minterms and hence, it must have 2^{n-1} minterms. Since in dual functions, positive high logic become negative low logic and vice versa.

Complement of the output of 2^{n-1} minterms will occur for the next half. Output of the 2^{n-1} minterms can take $2^{2^{n-1}}$ combination. Each combination will produce a unique output function F.

Hence, the possible self –dual functions are $\boxed{2^{2^{n-1}}}$.

Step 1 of 1

Write the function.

$$F = X_1 \cdot G(X_2, X_3 \dots X_n) + X_1' \cdot G^D(X_2, X_3 \dots X_n)$$ ...... (1)

Write the Dual of the function F.

$$F^D = \left(X_1 + G^D(X_2, X_3 \dots X_n)\right) \bullet \left(X_1' + G^{D^D}(X_2, X_3 \dots X_n)\right)$$ ...... (2)

Simplify further.

$$F^D = \left(X_1 + G^D(X_2, X_3 \dots X_n)\right) \bullet \left(X_1' + G(X_2, X_3 \dots X_n)\right)$$
$$= \begin{Bmatrix} X_1 \bullet X_1' + X_1 \bullet G(X_2, X_3 \dots X_n) + X_1' \bullet G^D(X_2, X_3 \dots X_n) + \\ G(X_2, X_3 \dots X_n) \bullet G^D(X_2, X_3 \dots X_n) \end{Bmatrix} \quad (\text{since } A \cdot A' = 0)$$

Therefore,

$$F^D = \begin{Bmatrix} X_1 \bullet G(X_2, X_3 \dots X_n) + X_1' \bullet G^D(X_2, X_3 \dots X_n) + \\ G(X_2, X_3 \dots X_n) \bullet G^D(X_2, X_3 \dots X_n) \end{Bmatrix}$$ ...... (3)

Consider the cases for equation (3).

$$G(X_2, X_3 \dots X_n) = G^D(X_2, X_3 \dots X_n)$$
$$= 0$$

Therefore,

$$F^D = X_1 \bullet G(X_2, X_3 \dots X_n) + X_1' \bullet G^D(X_2, X_3 \dots X_n)$$

Hence,

$$F = F^D$$

Consider the following case when

$$G(X_2, X_3 \dots X_n) = G^D(X_2, X_3 \dots X_n)$$

$$G(X_2, X_3 \dots X_n) = 1$$ ...... (4)

Therefore,

$$F^D = X_1 \bullet 1 + X_1' \bullet 1 + 1$$
$$= 1$$

Substitute the equation (4) in (1).

$$F = X_1 \cdot 1 + X_1' \cdot 1$$
$$= 1$$

Therefore,

$$F = F^D$$

Consider the following case when

$$G(X_2, X_3 \dots X_n) = 0$$

$$G^D(X_2, X_3 \dots X_n) = 1$$ ...... (5)

Therefore,

$$F^D = X_1 \bullet 0 + X_1' \bullet 1 + 0 \bullet 1$$
$$= X_1'$$

Substitute the equation (5) in (1).

$$F = X_1 \cdot 0 + X_1' \cdot 1$$
$$= X_1'$$

Hence,

$$F = F^D$$

Similarly, consider the following function.

$$G(X_2, X_3 \dots X_n) = 1$$

$$G^D(X_2, X_3 \dots X_n) = 0$$ ...... (6)

Therefore,

$$F^D = X_1 \bullet 1 + X_1' \bullet 0 + 1 \bullet 0$$
$$= X_1$$

Substitute the equation (5) in (1).

$$F = X_1 \cdot 1 + X_1' \cdot 0$$
$$= X_1$$

Therefore,

$$F = F^D$$

Hence, the function F that can be represented in the form

$F = X_1 \cdot G(X_2, X_3 \dots X_n) + X_1' \cdot G^D(X_2, X_3 \dots X_n)$ is self-dual.

Step 1 of 3

4.050E

The inverting gate has the propagation delay of 5 ns and non-inverting gate has the propagation delay of 8 ns. First way to physically realize the logic function is shown in the Figure 4.24 (a) in the text book. Referring the circuit, first two and gates are parallel and they are processed simultaneously and they have the single propagation delay of 8 ns and the second set of OR and AND gates process the output of the first set gates and hence it has to wait for 8 ns (when the first set gates process the inputs) and then it takes another 8 ns to produce the final output.

The total Propagation delay is sum of the propagation of first and second set gates.

$$\begin{aligned}\textbf{Total Propagation Delay } T_D &= 8\text{ ns} + 8\text{ ns}\\ &= 16\text{ns}\end{aligned}$$

Therefore, the net propagation delay is $\boxed{16\text{ns}}$.

Step 2 of 3

Another way to realize the above circuit is given by Figure 4-24(c) in the textbook. Two NAND gates and an inverter is processing the inputs parallel and hence the propagation delay is 5 ns another set of inverters are also processing the previous gates output simultaneously with the delay of 5 ns .Finally the non-inverting OR and AND gate will take 8 ns to produce the final output.

$$\begin{aligned}\textbf{Total Propagation Delay}(T_D) &= 5\text{ ns} + 5\text{ ns} + 8\text{ ns}\\ &= 18\text{ ns}\end{aligned}$$

Therefore, the net propagation delay is $\boxed{18\text{ ns}}$.

Step 3 of 3

Third way to realize the above circuit is shown in Figure 4-24(d) in the text book. Referring it we can say that Two NAND gates and inverters are processing the inputs parallel and hence the propagation delay is 5 ns another set of five inverters are also processing the previous gates output simultaneously with the delay of 5 ns .Finally the non-inverting OR and AND gate will take 8 ns to produce the final output.

$$\begin{aligned}\textbf{Total Propagation Delay}(T_D) &= 5\text{ ns} + 5\text{ ns} + 8\text{ ns}\\ &= 18\text{ns}\end{aligned}$$

Therefore, the net propagation delay is $\boxed{18\text{ns}}$.

Step 1 of 4

Write the general Shannon's expansion theorem for a function Z with the input variables $u_1, u_2, \dots u_n$.

$$Z(u_1, u_2, \dots u_{n-1}, u_n) = u_1 Z(1, u_2 \dots u_n) + u_1' Z(0, u_2, \dots u_{n-1}, u_n)$$

Write the other form or duality of Shannon's expansion theorem.

$$Z(u_1, u_2, \dots u_{n-1}, u_n) = [u_1 + Z(1, u_2, \dots u_{n-1}, u_n)][u_1' + Z(0, u_2, \dots u_{n-1}, u_n)]$$

Simplify the above two equations further to *n*-1 variable only if

$$Z(1, u_2 \dots u_n) = Z(0, u_2 \dots u_n)$$

Use the switching axioms.

$$(x+y)\cdot(x+y) = x$$
$$xy + xy' = x$$

In order to reduce the function by one literal, two cells in a group are necessary. It can be obtained only by two adjacent cells that differ by a single literal. It means that 2^1 cells are necessary.

Step 2 of 4

Shannon's expansion theorem can be done by more than one variable. For example from the first equation expanding the function with respect to u_1 and u_2 gives as follows.

$$\begin{aligned} Z(u_1, u_2, \dots u_{n-1}, u_n) = {} & u_1 u_2 Z(1, 1, \dots u_n) + u_1' u_2' Z(0, 1, \dots u_n) \\ & + u_1 u_2' Z(1, 0, \dots u_n) + u_1' u_2' Z(0, 0 \dots u_n) \end{aligned}$$

For second equation expanding the function with respect to u_1 and u_2

$$\begin{aligned} Z(u_1, u_2, \dots u_{n-1}, u_n) = {} & [u_1 + u_2 + Z(0, 0, \dots u_n)][u_1' + u_2 + Z(1, 0, \dots u_n)] \\ & [u_1 + u_2' \cdot Z(0, 1, \dots u_n)][u_1' + u_2' + Z(1, 1 \dots u_n)] \end{aligned}$$

Step 3 of 4

Simplify the above two equations further to n-2 variable only if

$$\begin{aligned} Z(0, 0 \dots, u_{n-1}, u_n) &= Z(0, 1, u_2 \dots, u_{n-1}, u_n) \\ &= Z(1, 0, u_2 \dots, u_{n-1}, u_n) \\ &= Z(1, 1, u_2 \dots, u_{n-1}, u_n) \end{aligned}$$

Use the switching axioms.

$$(x+y)\cdot(x+y') = x$$
$$xy + x'y = y$$

In order to reduce the function by two literals, four cells in a group are necessary. It can be obtained only by four adjacent cells that differ by a two literal. It means that 2^2 cells are necessary.

Step 4 of 4

Similarly, when we expand the first and second equation with respect to three terms X_1, X_2 and X_3,

$$\begin{aligned} Z(u_1, u_2, \dots u_{n-1}, u_n) = {} & u_1' \cdot u_2' \cdot u_3' . Z(0, 0, 0, \dots u_n) + u_1' \cdot u_2' \cdot u_3 . Z(0, 0, 1 \dots u_n) \\ & + u_1' \cdot u_2 \cdot u_3' \cdot Z(0, 1, 0 \dots u_n) + u_1' \cdot u_2 \cdot u_3 \cdot Z(0, 1, 1, \dots u_n) \\ & + u_1 \cdot u_2' \cdot u_3' \cdot Z(1, 0, 0, \dots u_n) + u_1 \cdot u_2' \cdot u_3 \cdot Z(1, 0, 1, \dots u_n) \\ & + u_1 \cdot u_2 \cdot u_3' \cdot Z(1, 1, 0, \dots u_n) + u_1 \cdot u_2 \cdot u_3 \cdot Z(1, 1, 1, \dots u_n) \end{aligned}$$

$$\begin{aligned} Z(u_1, u_2, \dots u_{n-1}, u_n) = {} & [u_1 + u_2 + u_3 + Z(0, 0, 0 \dots u_n)]\left[u_1 + u_2 + u_3' + Z(0, 0, 1 \dots u_n)\right] \cdot \\ & \left[u_1 + u_2' + u_3 + Z(0, 1, 0 \dots u_n)\right] \cdot \left[u_1 + u_2' + u_3' + Z(0, 1, 1 \dots u_n)\right] \cdot \\ & [u_1' + u_2 + u_3 + Z(1, 0, 0 \dots u_n)] \cdot \left[u_1' + u_2 + u_3' + Z(1, 0, 1 \dots u_n)\right] \cdot \\ & \left[u_1' + u_2' + u_3 + Z(1, 1, 0 \dots u_n)\right] \cdot \left[u_1' + u_2' + u_3' + Z(1, 1, 1 \dots u_n)\right] \end{aligned}$$

And to reduce the product or sum term by three literals we need eight (2^3) adjacent cells. Hence by means of switching algebra and Shannon`s theorem we can conclude that to reduce `*i*' number of literals in the product term or sum term of the logical functions 2^i adjacent cells in K-map needs to be grouped.

Step 1 of 1

4.052E

The Karnaugh-map for four variables is shown in Figure 1:

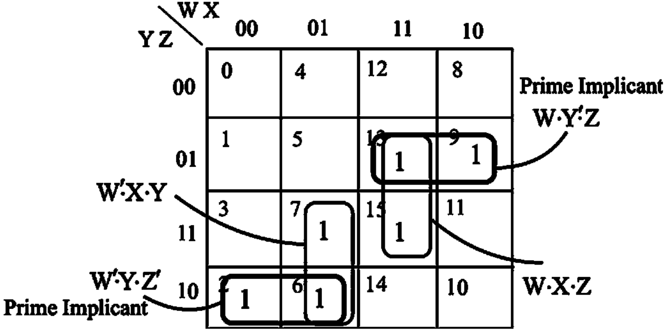


Figure 1: Karnaugh map

Irredundant sum is the one in which removal of any one the essential prime implicants cannot implies the same function F. Therefore, from Figure 1, the irredundant sum is,

$$F = W' \cdot Y \cdot Z' + W' \cdot X \cdot Y + W \cdot X \cdot Z + W \cdot Y' \cdot Z$$

Therefore, the irredundant sum is $\boxed{F = W' \cdot Y \cdot Z' + W' \cdot X \cdot Y + W \cdot X \cdot Z + W \cdot Y' \cdot Z}$.

Step 1 of 4 **4.053E**

The Karnaugh-map for four variables is shown in Figure 1:

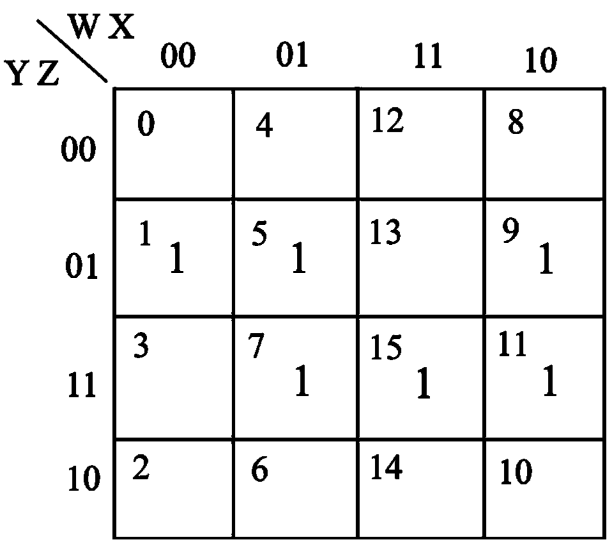


Figure 1: Karnaugh map

Irredundant sum is the one in which removal of any one the essential prime implicants cannot implies the same function F. It will occur in the K-map which has 1-cells that can be grouped together in minimal sum and it can be separated grouped with other cells and forms new groups.

One possibility of irredundant sum is shown in Figure 2.

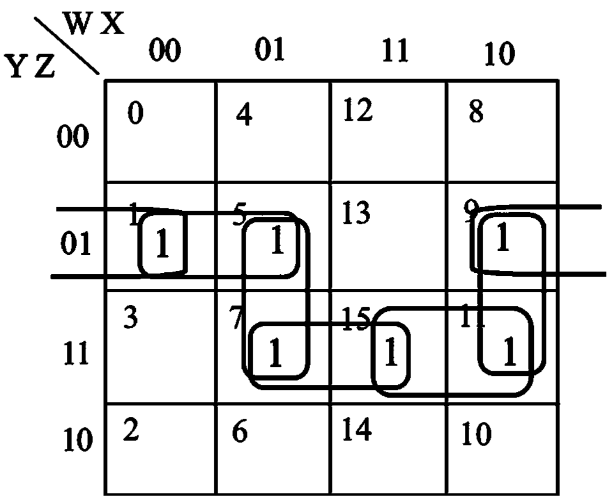


Figure 2: First possibility of Irredundant Sum Karnaugh map

Therefore, the logic function is $\boxed{F = W'\cdot X\cdot Z + X\cdot Y\cdot Z + W\cdot X'\cdot Z + X'\cdot Y'\cdot Z}$.

Step 2 of 4

Another irredundant sum for the K-map is shown in Figure 3.

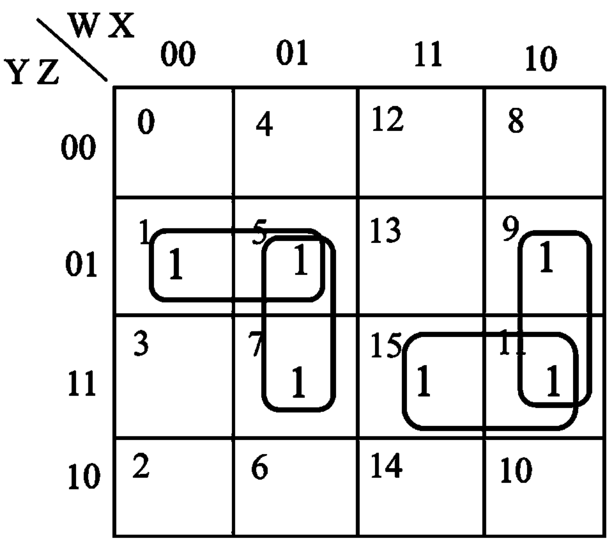


Figure 3: Second possible Irredundant Sum for the Karnaugh map

Step 3 of 4

Therefore, the logic function is $\boxed{F = W'\cdot Y'\cdot Z + W'\cdot X\cdot Z + W\cdot Y\cdot Z + W\cdot X'\cdot Z}$.

Step 4 of 4

Another one possibility of irredundant sum for the K-map is shown in Figure 4.

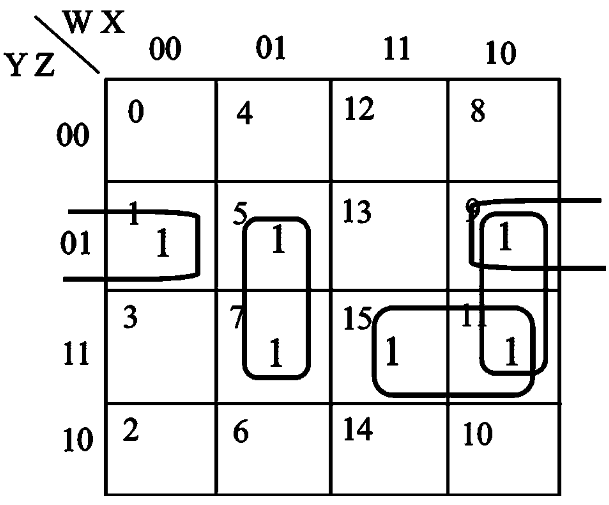


Figure 4: Third way of grouping to form Irredundant Sum for the Karnaugh map

Therefore, the logic function is $\boxed{F = W'\cdot X\cdot Z + W\cdot Y\cdot Z + W\cdot X'\cdot Z + X'\cdot Y'\cdot Z}$.

Step 1 of 3

Draw the OR-XOR logic circuit:

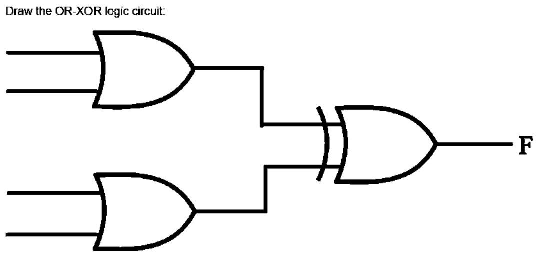


Figure 1: OR-XOR circuit

Step 2 of 3

The output is,

$$F = \sum_{W,X,Y,Z}(2,3,8,9)$$

The K-map for the output is shown in Figure 2.

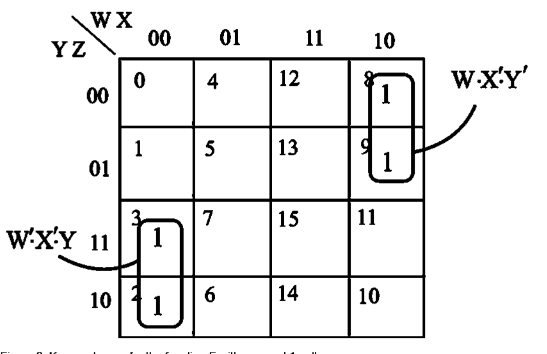


Figure 2: Karnaugh map for the function F with grouped 1-cells

Step 3 of 3

Therefore, the logic function F is,

$$\begin{aligned} F &= W\cdot X'\cdot Y' + W'\cdot X'\cdot Y \\ &= X'\cdot(W\cdot Y' + W'\cdot Y) \\ &= X'\cdot(W\oplus Y) \end{aligned}$$

Assign suitable inputs to the OR gate in accordance with the function and the schematic. Minimal sum obtained from the K-map is having AND, as well as, OR function otherwise AND and Ex-OR function. There is no such combination in the schematic. According to the schematic, two level OR-Ex-OR gates are available. It is not possible to implement simplified function directly in schematic. According to the Truth table of Ex-OR gate when one of its inputs is set to zero it will act as non-inverting transmission gate. One of the inputs is one it will act as inverter.

X	Y	XT Y
0	0	0
0	**1**	1
1	0	1
1	**1**	0

Table 1: Truth table of Ex-OR gate

When set the output of one OR gate as 1, then implement the function $\mathbf{F'}$ in the next OR gate. Hence, the Ex-OR gate will implement the function F. Now the function $\mathbf{F'}$ is obtained by replacing the dot with plus sign and inverting the literals.

$$F' = X + \overline{(W\oplus Y)}$$

Draw the OR-XOR circuit with labeled inputs.

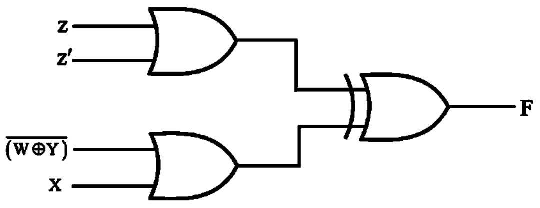
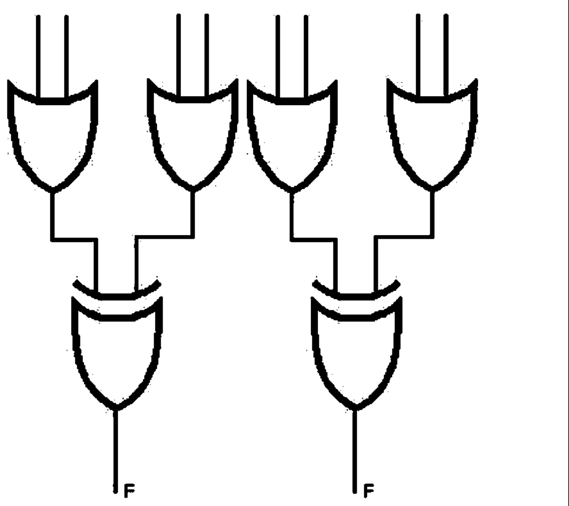
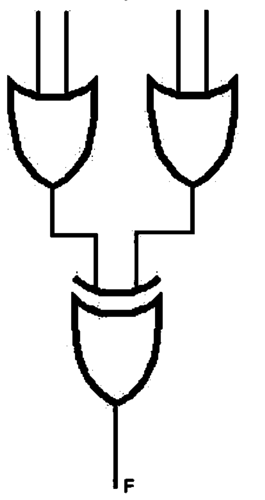


Figure 3: OR-XOR circuit with labeled inputs

The first OR gate performs the operation between $\mathbf{z}$ **and** $\mathbf{z'}$. Hence by the Boolean algebra $z + z' = 1$ output is always 1.





Implement the same function using two level NOR-Ex-OR schematic. Similarly when the output of one NOR gate is 0, Then implement the function F in the next NOR gate. The Ex-OR gate will implement the function F by NOR gate is shown in Figure 4.

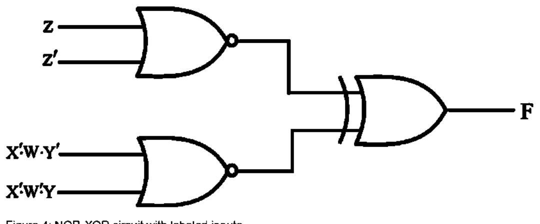


Figure 4: NOR-XOR circuit with labeled inputs

Step 1 of 3

Equality of two bits say, **P and Q** is denoted by $X = PQ + P'Q'$.

If $X = 1$, the bits are equal else unequal.

The truth table for 3-bit comparator is shown in Table 1:

P_2Q_2	P_1Q_1	P_0Q_0	$P<Q$	$P=Q$	$P>Q$
$P_2 < Q_2$	×	×	0	0	1
$P_2 = Q_2$	$P_1 > Q_1$	×	0	0	1
$P_2 = Q_2$	$P_1 = Q_1$	$P_0 > Q_0$	0	0	1
$P_2 = Q_2$	$P_1 = Q_1$	$P_0 = Q_0$	0	1	0
$P_2 < Q_2$	×	×	1	0	0
$P_2 = Q_2$	$P_1 < Q_1$	×	1	0	0
$P_2 = Q_2$	$P_1 = Q_1$	$P_0 < Q_0$	1	0	0

Table 1

Step 2 of 3

One binary number says $P(P2P1P0)$ is less than $Q(Q2Q1Q0)$ if either $(P2 < Q2)$ else if $(P2 = Q2)$ then $(P1 < Q1)$ else if $(P1 = Q1)$ then $(P0 < Q0)$ is denoted by the following Boolean expression.

$$(P < Q) = P2'Q2 + X2P1'Q1 + X2X1P0'Q0$$

Step 3 of 3

The implementation of this Boolean expression is as follows,

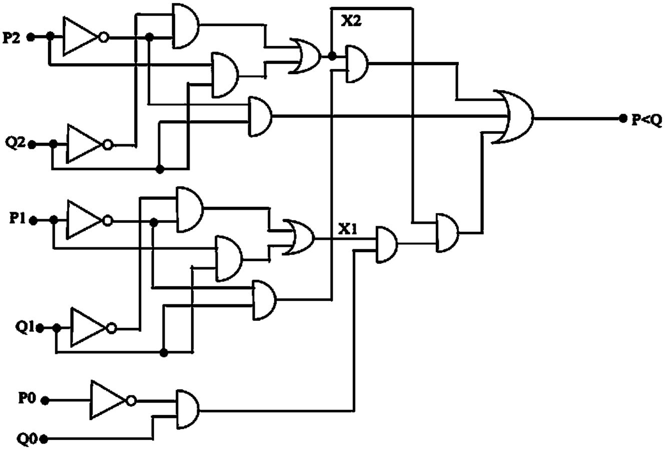


Figure 1

4.056E

Step 1 of 10

Consider the following function.

$F = \sum_{X,Y,Z}(0,1,2)$

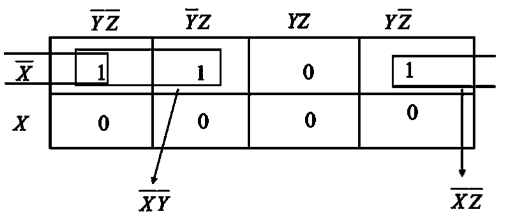
Write the truth table as shown in Table 1:

Table 1

X	Y	Z	F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Step 2 of 10

Draw the Karnaugh Map as shown in Figure 1:



$\overline{Y}\overline{Z}$
$\overline{Y}Z$
YZ
$Y\overline{Z}$
$\overline{X}$
X
1
1
0
1
0
0
0
0
$\overline{X}\overline{Y}$
$\overline{X}\overline{Z}$

Figure 1

Step 3 of 10

From Figure 1,

F= group1 (pink color line) + group2 (blue color line)

$$F = \overline{X}\overline{Y} + \overline{X}\overline{Z}$$
$$= \overline{X}(\overline{Y} + \overline{Z})$$

Therefore, the final Boolean expression is $\boxed{F = \overline{X}(\overline{Y} + \overline{Z})}$.

Step 4 of 10

Consider the following function.

$G = \sum_{X,Y,Z}(0,4,6)$

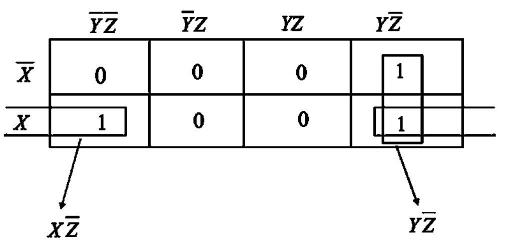
Write the truth table as shown in Table 2:

Table 2

X	Y	Z	G
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Step 5 of 10

Draw the Karnaugh Map as shown in Figure 2:



Step 6 of 10

Figure 2

Step 7 of 10

From Figure 2,

G= group1 (pink color line) + group2 (blue color line)

$$G = X\overline{Z} + Y\overline{Z}$$
$$= \overline{Z}(X + Y)$$

Therefore, the final Boolean expression is $\boxed{G = \overline{Z}(X + Y)}$.

Step 8 of 10

Consider the following function.

$H = \sum_{X,Y,Z}(0,1,2,4,6)$

Write the truth table as shown in Table 3:

Table 3

X	Y	Z	H
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Step 9 of 10

Draw the Karnaugh Map as shown in Figure 3:

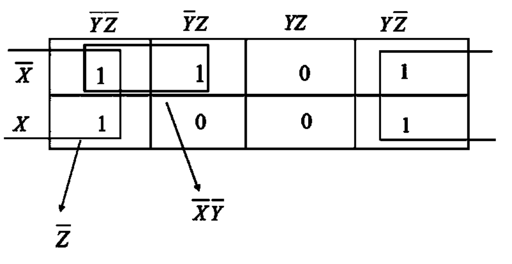


Figure 3

Step 10 of 10

From Figure 3,

Write the Boolean expression as follows:

H= group1 (blue color line) + group2 (pink color line)

$$H = \overline{X}\overline{Y} + \overline{Z}$$

Therefore, the final Boolean expression is $\boxed{H = \overline{X}\overline{Y} + \overline{Z}}$.

Step 1 of 1

4.057E

Write the following expression.

$$F = S'\cdot T\cdot U\cdot V\cdot W + S'\cdot T\cdot U'\cdot W\cdot Y + S'\cdot T\cdot V\cdot W\cdot X'\cdot Y$$

Simplify the expression algebraically.

$$\begin{aligned}F &= S'\cdot T\cdot U\cdot V\cdot W + S'\cdot T\cdot U'\cdot W\cdot Y + S'\cdot T\cdot V\cdot W\cdot X'\cdot Y\\ &= S'\cdot T\cdot\left(U\cdot V\cdot W + U'\cdot W\cdot Y + V\cdot W\cdot X'\cdot Y\right) \qquad (\text{since } X + X' = 1)\\ &= S'\cdot T\cdot\left(U\cdot V\cdot W + U'\cdot W\cdot Y + (U+U')V\cdot W\cdot X'\cdot Y\right)\\ &= S'\cdot T\cdot\left(U\cdot V\cdot W + U'\cdot W\cdot Y + U\cdot V\cdot W\cdot X'\cdot Y + U'\cdot V\cdot W\cdot X'\cdot Y\right)\end{aligned}$$

Simplify further.

$$\begin{aligned}F &= S'\cdot T\cdot\left(U\cdot V\cdot W(1+X'\cdot Y) + U'\cdot W\cdot Y(1+V\cdot X')\right) \qquad (\text{since } 1 + A = 1)\\ &= S'\cdot T\cdot\left(U\cdot V\cdot W + U'\cdot W\cdot Y\right)\\ &= S'\cdot T\cdot U\cdot V\cdot W + S'\cdot T\cdot U'\cdot W\cdot Y\end{aligned}$$

Therefore, the minimal sum is $\boxed{F = S'\cdot T\cdot U\cdot V\cdot W + S'\cdot T\cdot U'\cdot W\cdot Y}$.

Hence, proved

Step 1 of 1

(a)

Consider the following five variable expression.

$$F=\sum\nolimits_{V,W,X,Y,Z}(5,7,13,15,16,20,25,27,29,31)$$

Karnaugh map for 5-variable function is shown in Figure 1.

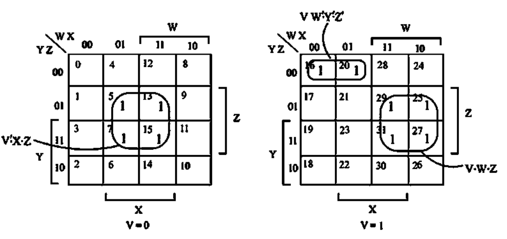


Figure 1

From K-map at V is equal to 0.

$$F_0 = V'\cdot X\cdot Z$$

From K-map at V is equal to 1.

$$F_1 = V\cdot W\cdot Z + V\cdot W'\cdot Y'\cdot Z'$$

Therefore,

$$\begin{aligned}F &= F_0 + F_1\\ &= V'\cdot X\cdot Z + V\cdot W\cdot Z + V\cdot W'\cdot Y'\cdot Z'\end{aligned}$$

Therefore, the minimized final expression is $\boxed{V'\cdot X\cdot Z + V\cdot W\cdot Z + V\cdot W'\cdot Y'\cdot Z'}$.

(b)

Consider the following five variable expression.

$$F=\sum\nolimits_{V,W,X,Y,Z}(0,7,8,9,12,13,15,16,22,23,30,31)$$

Karnaugh map for 5-variable function is shown in Figure 2.

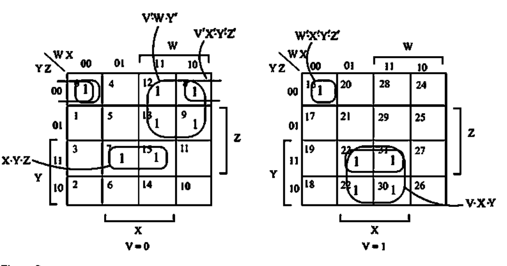


Figure 2

1-cells encircled with same color indicate the members of the same group.

Therefore, the final simplified expression is

$$\boxed{X\cdot Y\cdot Z + V'\cdot W\cdot Y' + V\cdot X\cdot Y + W'\cdot X'\cdot Y'\cdot Z'}.$$

(c)

Consider the following five variable expression.

$$F=\sum\nolimits_{V,W,X,Y,Z}(0,1,2,3,4,5,10,11,14,20,21,24,25,26,27,28,29,30)$$

Karnaugh map for 5-variable function is shown in Figure 3.

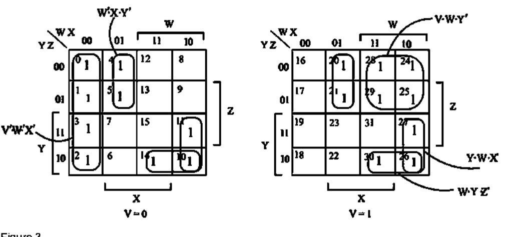


Figure 3

1-cells encircled with same color indicate the members of the same group.

Therefore, the simplified expression is

$$\boxed{W'\cdot X\cdot Y' + V'\cdot W'\cdot X' + V\cdot W\cdot Y' + Y\cdot W\cdot X' + Y\cdot W\cdot Z'}.$$

(d)

Consider the following five variable expression.

$$F=\sum\nolimits_{V,W,X,Y,Z}(0,2,4,6,7,8,10,11,12,13,14,16,18,19,29,30)$$

Karnaugh map for 5-variable function is shown in Figure 4.

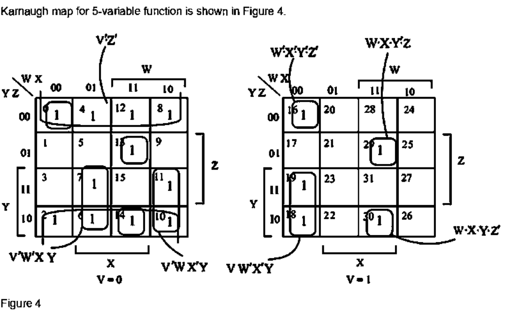


Figure 4

1-cells encircled with same color indicate the members of the same group.

Therefore, the simplified expression is

$$\boxed{\begin{array}{l}V'\cdot Z' + V'\cdot W'\cdot X\cdot Y + V'\cdot W\cdot X'\cdot Y + W'\cdot X'\cdot Y'\cdot Z' +\\ W\cdot X\cdot Y'\cdot Z + V\cdot W'\cdot X'\cdot Y + W\cdot X\cdot Y\cdot Z'\end{array}}.$$

(e)

Consider the following five variable expression.

$$F=\prod\nolimits_{V,W,X,Y,Z}(4,5,10,12,13,16,17,21,25,26,27,29)$$

Karnaugh map for 5-variable function is shown in Figure 5.

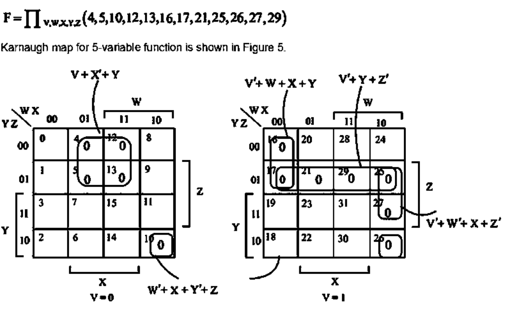


Figure 5

Since it is product of sum terms 0-cells encircled with same color indicate the members of the same group.

Therefore, the simplified expression is

$$\boxed{(V+X'+Y)\cdot(V'+W+X+Y)\cdot(V'+Y+Z')\cdot(V'+W'+X+Z')\cdot(W'+X+Y'+Z)}.$$

(f)

Consider the following five variable expression.

$$F=\sum\nolimits_{V,W,X,Y,Z}(4,6,7,9,11,12,13,14,15,20,22,25,27,28,30)+d(1,5,29,31)$$

Karnaugh map for 5-variable function is shown in Figure 6.

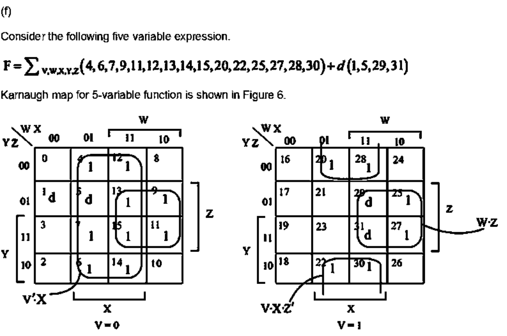


Figure 6

1-cells encircled with same color indicate the members of the same group.

Therefore, the final expression is $\boxed{V'X + WZ + VXZ'}$.

Step 1 of 2

(a)

Write the given function.

$$F=\sum_{U,V,W,X,Y,Z}(1,5,9,13,21,23,29,31,37,45,53,61)$$

Karnaugh map for 6-variable function is shown in Figure 1.

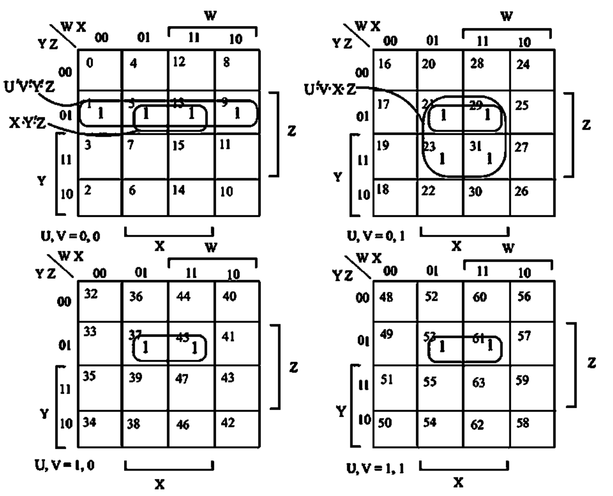


Figure 1

In the logical expression involving six variables, K map will get three dimensional picture with the overlapping of two dimensional planes and hence the cells that are in the same relative position of two dimensional (four variable) K-map will become adjacent one that can be grouped together. Hence the squares that are adjacent vertically as shown above can be grouped together.

Therefore, from Figure 1, the final expression *F* is

$$\boxed{U'\cdot V'\cdot Y'\cdot Z+U'\cdot V\cdot X\cdot Z+X\cdot Y'\cdot Z}$$

Step 2 of 2

(b)

Write the given function.

$$F=\sum_{U,V,W,X,Y,Z}(0,4,8,16,24,32,34,36,37,39,40,48,50,56)$$

Karnaugh map for 6-variable function is shown in Figure 2.

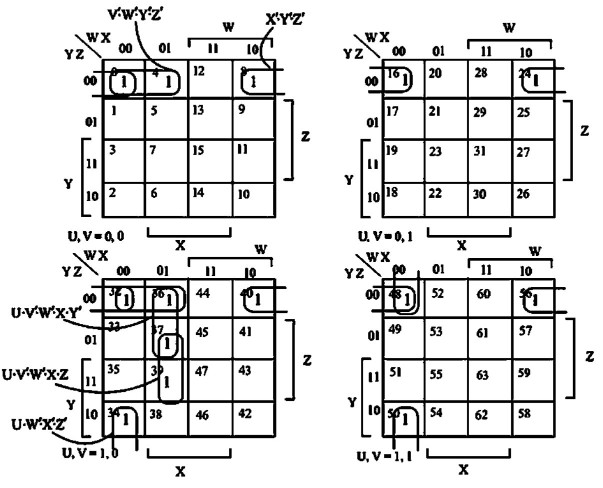


Figure 2

In the logical expression involving six variables, K map will get three dimensional picture with the overlapping of two dimensional planes and hence the cells that are in the same relative position of two dimensional (four variable) K-map will become adjacent one that can be grouped together. Hence the squares that are adjacent vertically as shown above can be grouped together.

Therefore, the final expression *F* is

$$\boxed{X'\cdot Y'\cdot Z'+V'\cdot W'\cdot Y'\cdot Z'+U\cdot V'\cdot W'\cdot X\cdot Z+U\cdot W'\cdot X'\cdot Z'}$$

(c)

Write the given function.

$$F=\sum_{U,V,W,X,Y,Z}(2,4,5,6,12-21,28-31,34,38,50,51,60-63)$$

Karnaugh map for 6-variable function is shown in Figure 3.

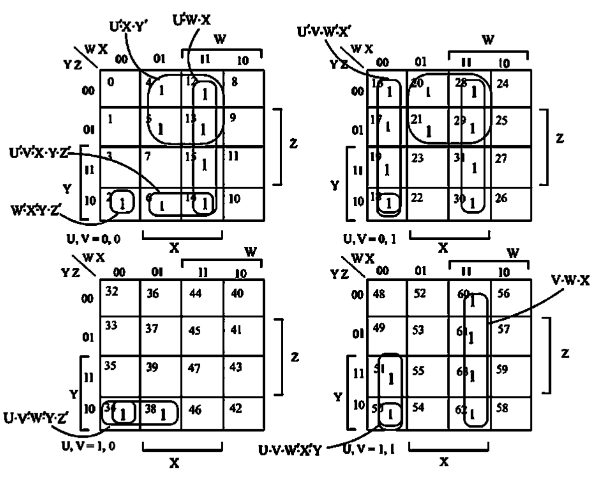


Figure 3

In the logical expression involving six variables, K map will get three dimensional picture with the overlapping of two dimensional planes and hence the cells that are in the same relative position of two dimensional (four variable) K-map will become adjacent one that can be grouped together. Hence the squares that are adjacent vertically as shown above can be grouped together.

Therefore, the final expression *F* is

$$\boxed{\begin{array}{l}U'\cdot X\cdot Y'+U'\cdot W\cdot X+U'\cdot V'\cdot X\cdot Y\cdot Z'+W'\cdot X'\cdot Y\cdot Z'+\\ U'\cdot V\cdot W'\cdot X'+U\cdot V'\cdot W'\cdot Y\cdot Z'+U\cdot V\cdot W'\cdot X'\cdot Y\end{array}}.$$

Step 1 of 5

Consider the following Hamlet circuit:

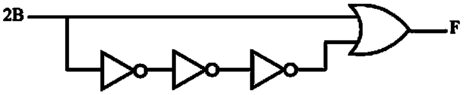


Figure 1: Hamlet Circuit

Step 2 of 5

Redraw the Figure **1** by labeling the NOT gates as Z1, Z2 and Z3 to consider the output each NOT gate:

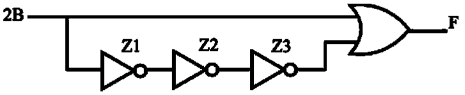


Figure 2: Hamlet circuit with labeled NOT gates

Step 3 of 5

The following is the timing diagram of the Hamlet circuit in sequence of steps of each gate from input 2B to the F in the Figure 2:

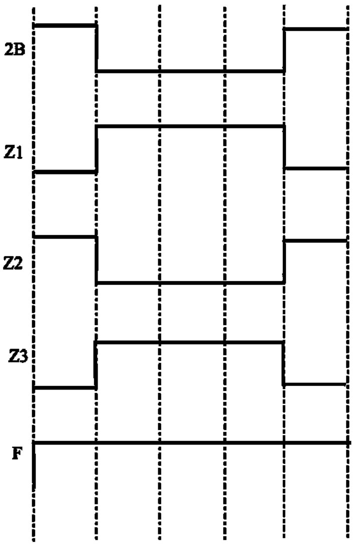


Figure 3: The timing diagram of ideal Hamlet circuit

Step 4 of 5

The signal 2B is given as input to the hamlet circuit, when 2B is high at this time interval Z1 signal goes low, Z2 goes high, and Z3 goes low. This Z3 signal (inverse of the input signal 2B) and 2B signal are given as input to the OR gate. As we know that the functioning of the OR gate if any one of the input is at logic High then the output is High.

Similarly when the 2B signal is at Low then the Output F is high because the inputs of the OR gate are the logic Low and its inverse Logic High.

Thus, the output F is always Logic is High even the input signal 2B is at logic High or Low.

Step 5 of 5

The reason for keeping the name Hamlet for this circuit is because hamlet means small village in Britain. Thus, the groups of NOT gate looks like small village and thus the name suits it.

Step 1 of 3

4.063E

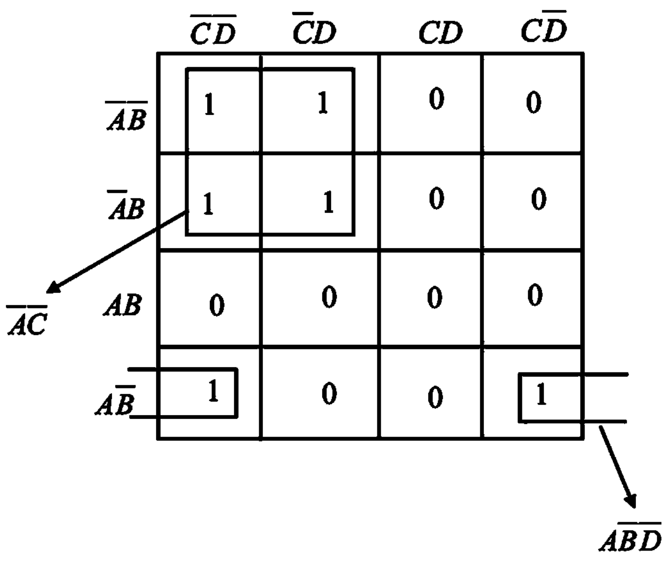
Consider the truth table is shown in Table 1:

Table 1

A	B	C	D	F
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	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

The Karnaugh Map is shown in Table 2:

Table 2



$\overline{C}\,\overline{D}$
$\overline{C}D$
CD
$C\overline{D}$
$\overline{A}\,\overline{B}$
$\overline{A}B$
AB
$A\overline{B}$
$\overline{A}\,\overline{C}$
$A\overline{B}\,\overline{D}$

Step 2 of 3

From Table 1, write the simplified expression as follows:

$$F = \text{group1 (pink color line)} + \text{group2 (red color line)}$$
$$= \overline{A}\,\overline{C} + A\overline{B}\,\overline{D}$$

The final minimal SOP expression is $\overline{A}\,\overline{C} + A\overline{B}\,\overline{D}$.

Step 3 of 3

The hazard free circuit means inclusion of those product terms of K-map in the final Boolean expression which were not included in the final Boolean expression. So no way the number of product terms can be less than the original SOP expression because the original SOP expression is itself a minimal Boolean expression.

In this example shows a four-variable logic function which is **not hazard free** (simplified expression).

Note: In case of hazard free circuits all prime implicates should be included.

Step 1 of 6

Consider the following Karnaugh map:

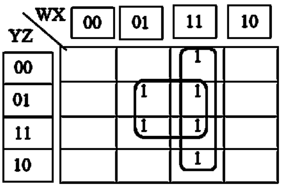


Figure 1: Karnaugh map for function $\sum_{W,X,Y,Z}(5,7,12,13,14,15)$

Step 2 of 6

The function of the Karnaugh map in Figure 1 is

$$\begin{aligned}F &= \sum\nolimits_{W,X,Y,Z}(5,7,12,13,14,15)\\ &= XZ + WX\end{aligned}$$

Step 3 of 6

Consider all the 1's shown in Figure 1 of Karnaugh map raised up one row. That is the 1's location 5, 13, and 12 go to 4, 12, and 14 respectively.

Redraw the Figure 1 after imagines 1's are moved one row up:

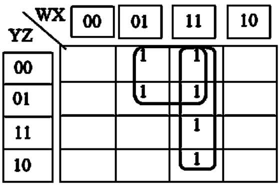


Figure 2: Karnaugh map for function $\sum_{W,X,Y,Z}(4,5,12,13,14,15)$

Step 4 of 6

When comparing Figure 1 and Figure 2, the 1's in place of 5 is moved to place of 4, 7 is moved to 5, 12 is moved to 14, 13 is moved to 12, 15 is moved to 13 and 14 is moved to 15.

The function of the Karnaugh map in Figure 2 is

$$\begin{aligned}F &= \sum\nolimits_{W,X,Y,Z}(4,5,12,13,14,15)\\ &= XY' + WX\end{aligned}$$

Step 5 of 6

Redraw the Figure 1 after shifting of contents from the 1's are moved to left one column:

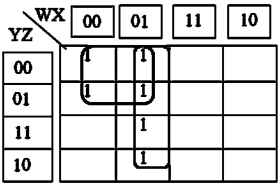


Figure 3: Karnaugh map for function $\sum_{W,X,Y,Z}(0,1,4,5,6,7)$

Step 6 of 6

The function of the Karnaugh map in Figure 3 is

$$\begin{aligned}F &= \sum\nolimits_{W,X,Y,Z}(0,1,4,5,6,7)\\ &= Y'W' + WX\end{aligned}$$

Step 1 of 2

Consider the following 2-to-4 decoder logic diagram:

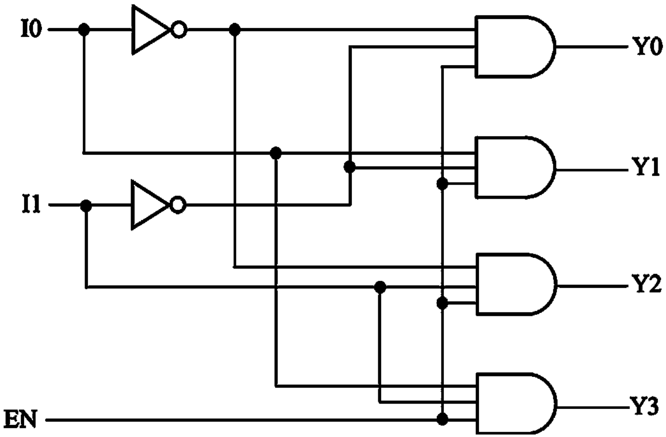


Figure 1: The 2-to-4 decoder circuit

Step 2 of 2

Write the ABEL program to the 2-to-4 decoder shown in Figure 1.

```
module decoder2_4

title ‘2-to-4 decoder’

“input and ouput pins

I0, I1, EN pin;

Y0, Y1, Y2, Y3 pin is type ‘com’;

equations

Y0 = !I0 & !I1 & EN;

Y1 = I0 & !I1 & EN;

Y2 = !I0 & I1 & EN;

Y3 = I0 & I1 & EN;

end decoder2_4
```

Step 1 of 2

Below ABEL program gives the prime number detector.

```
module PrimeDet

title '4-Bit Prime Number Detector'

"input and ouput pins

N0,N1,N2,N3 pin;

F pin istype 'com';

"Definiton

NUM= [N3, N2, N1, N0];

truth_table (NUM -> F)

1-> 1;

2-> 1;

3-> 1;

5-> 1;

7-> 1;

11-> 1;

13-> 1;

end PrimeDet;
```

Step 2 of 2

The test vectors associate various input combinations to the expected output values.

The test vectors for Prime number Detector is as shown:

```
test_vectors

( [N3, N2, N1, N0] -> [F] )

[ 0, 0, 0, 0] -> [ 0 ] "1

[ 0, 0, 0, 1] -> [ 1 ] "2

[ 0, 0, 1, 0] -> [ 1 ] "3

[ 0, 0, 1, 1] -> [1 ] "4

[ 0, 1, 0, 0] -> [ 0 ] "5

[ 0, 1, 0, 1] -> [1 ] "6

[ 0, 1, 1, 0] -> [ 0 ] "7

[ 0, 1, 1, 1] -> [ 1 ] "8

[ 1, 0, 0, 0] -> [ 0 ] "9

[ 1, 0, 0, 1] -> [ 0 ] "10

[ 1, 0, 1, 0] -> [ 0 ] "11

[ 1, 0, 1, 1] -> [ 1 ] "12

[ 1, 1, 0, 0] -> [ 0 ] "13

[ 1, 1, 0, 1] -> [ 1 ] "14

[ 1, 1, 1, 0] -> [ 0 ] "15

[ 1, 1, 1, 1] -> [ 0 ] "16
```

Step 1 of 5

Draw the logic circuit for the Alarm:

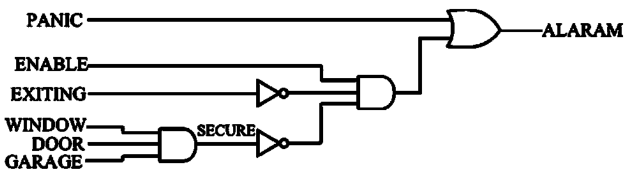


Figure 1

Step 2 of 5

The following is the structural VHDL program for Alarm circuit:

```
library IEEE;
use IEEE.std_logic_1164.all;
entity alarm_circuit is
port ( PANIC, ENABLE, EXITING, WINDOW, DOOR, GARAGE: in STD_LOGIC;
ALARM: out STD_LOGIC);
end alarm_circuit;
architecture alarm_circuit_arch of alarm_circuit is
signal SECURE, n1, n2, n3: STD_LOGIC;
component INV port( I: in STD_LOGIC; O: out STD_LOGIC);
end component;
component AND3 port( inp1, inp2, inp3 : in STD_LOGIC;
out1: out STD_LOGIC);
end component;
component OR2 port(inp1, inp2: in STD_LOGIC; out1: out STD_LOGIC);
end component;
begin
U1: AND3 port map ( WINDOW, DOOR, GARAGE, SECURE);
U2: INV port map( SECURE, n1);
U3: INV port map( EXITING, n2);
U4: AND3 port map(ENABLE, n1, n2,n3);
U5: OR2 port map( PANIC, n3, ALARM);
end alarm_circuit_arch;
```

Step 3 of 5

```
– component description
– AND3
entity AND3 is
    port  ( inp1, inp2, inp3: in STD_LOGIC;
          out1: out STD_LOGIC);
    end AND3;

architecture AND3_arch of AND3 is
begin
out1 <= inp1 and (inp2 and inp3);
end AND3_arch;
```

Step 4 of 5

```
– component description
--- INV
entity INV is
    port  ( I: in STD_LOGIC;
          O: out STD_LOGIC);
 end INV;

architecture INV_arch of INV is
begin
O <= not ( I );
end INV_arch;
```

Step 5 of 5

```
– component description
--- OR2
entity OR2 is
    port  ( inp1, inp2: in STD_LOGIC;
          out1: out STD_LOGIC);
    end OR2;

architecture OR2_arch of OR2 is
begin
out1 <= inp1 or inp2;
end OR2_arch;
```

Step 1 of 2 **5.05DP**

In dataflow design or description, continuous-assignment statements are used to evaluate the value on the right hand side of the statement to the left hand side statement, continuously. This is very important in simulation as it occurs in zero simulation time.

Consider the following circuit diagram:

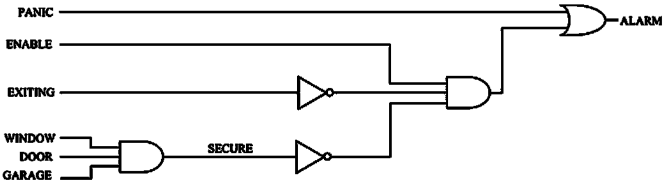


Figure 1

Step 2 of 2

The dataflow style of VHDL program for Alarm circuit is as follows:

```
library IEEE;

use IEEE.std_logic_1164.all;

entity alarm_circuit is

port ( panic, enable1, exiting, window, door, garage: in STD_LOGIC;

alarm: out STD_LOGIC);

end alarm_circuit;

architecture alarm_circuit_arch of alarm_circuit is

signal secure, n1, n2, n3: STD_LOGIC;

secure <= window and ( door and garage);

n1 <= not(secure);

n2 <= not(exiting);

n3 <= enable1 and (n1 and n2);

alarm <= panic or n3;

end alarm_circuit_arch;
```

5.06DP

Step 1 of 5

Draw the following logic circuit.

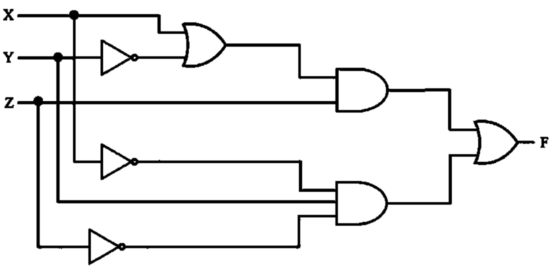


Figure 1: A 3-Input, 1-Output logic circuit

Step 2 of 5

The structural style of VHDL program for 3-Input, 1-Output logic circuit is as follows:

```
library IEEE;
use IEEE.std_logic_1164.all;
entity Inp3_out1 is
port ( inp1,inp2,inp3: in STD_LOGIC;
out1: out STD_LOGIC);
end Inp3_out1;
architecture Inp3_out1_arch of Inp3_out1 is
signal n1, n2, n3,n4,n5,n6: STD_LOGIC;
component INV port( I: in STD_LOGIC; O: out STD_LOGIC);
end component;
component AND3 port( inp1, inp2, inp3 : in STD_LOGIC;
out1: out STD_LOGIC);
end component;
component AND2 port( inp1, inp2: in STD_LOGIC;
out1: out STD_LOGIC);
end component;
component OR2 port( inp1, inp2: in STD_LOGIC;
out1: out STD_LOGIC);
end component;
begin
U1: INV port map ( inp1, n1);
U2: INV port map ( inp2, n2);
U3: INV port map ( inp3, n3);
U4: OR2 port map ( inp1, n2, n4);
U5:AND2 port map ( n4, inp3, n5);
U6: AND3 port map(n1, inp2, n3, n6);
U7: OR2 port map( n5, n6, out1);
end Inp3_out1_arch;
```

Step 3 of 5

```
-- component description
-- AND3
entity AND3 is
   port  ( inp1, inp2, inp3: in STD_LOGIC;
        out1: out STD_LOGIC);
   end AND3;

architecture AND3_arch of AND3 is
begin
out1 <= inp1 and (inp2 and inp3);
end AND3_arch;
```

Step 4 of 5

```
-- component description
-- AND2
entity AND2 is
   port  ( inp1, inp2, inp3: in STD_LOGIC;
        out1: out STD_LOGIC);
   end AND2;

architecture AND2_arch of AND3 is
begin
out1 <= inp1 and inp2 ;
end AND2_arch;
```

Step 5 of 5

```
-- component description
--- INV
entity INV is
   port  ( I: in STD_LOGIC;
        O: out STD_LOGIC);
 end INV;

architecture INV_arch of INV is
begin
O <= not ( I );
end INV_arch;
-- component description
--- OR2
entity OR2 is
   port  ( inp1, inp2: in STD_LOGIC;
        out1: out STD_LOGIC);
   end OR2;

architecture OR2_arch of OR2 is
begin
out1 <= inp1 or inp2;
end OR2_arch;
```

Step 1 of 2

Draw the following logic circuit.

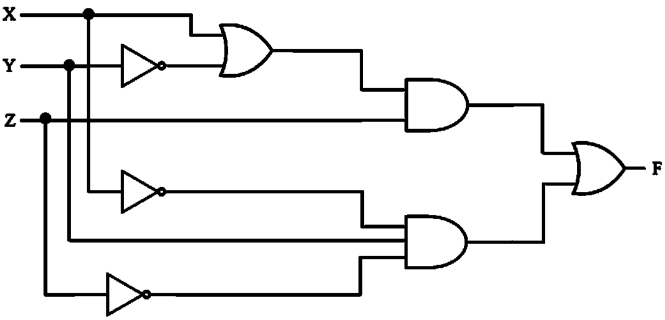


Figure 1: A 3-Input, 1-Output logic circuit

Step 2 of 2

The dataflow style of VHDL program for 3-Input, 1-Output logic circuit is as follows:

```
library IEEE;

use IEEE.std_logic_1164.all;

entity Inp3_out1 is

port ( inp1,inp2,inp3: in STD_LOGIC;

out1: out STD_LOGIC);

end Inp3_out1;

architecture Inp3_out1_arch of Inp3_out1 is

signal n1, n2, n3,n4,n5,n6: STD_LOGIC;

begin

n1 <= not (inp1);

n2<= not (inp2);

n3<= not (inp3);

n4<= inp1 or n2;

n5<= n4 and inp3;

n6<= n1 and (inp2 and n3);

out1<= n5 or n6;

end Inp3_out1_arch;
```

Step 1 of 2

5.08DP

Draw the following logic circuit.

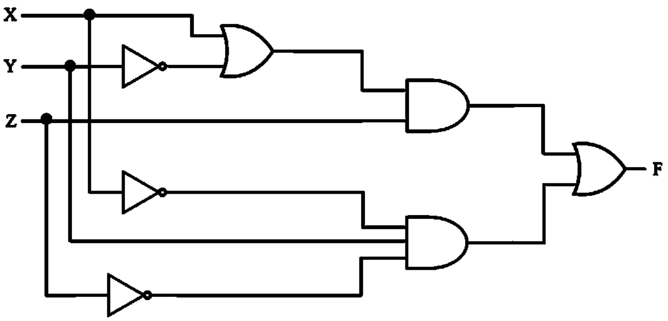


Figure 1: A 3-Input, 1-Output logic circuit

Step 2 of 2

The dataflow style of VHDL program for 3-Input, 1-Output logic circuit is as follows:

```
library IEEE;

use IEEE.std_logic_1164.all;

entity Inp3_out1 is

port ( inp1, inp2, inp3: in STD_LOGIC;

out1: out STD_LOGIC);

end Inp3_out1;

architecture Inp3_out1_arch of Inp3_out1 is

begin

process (inp1, inp2, inp3)

variable n1, n2, n3, n4, n5, n6: bit : = '0';

begin

n1 <= not (inp1);

n2<= not (inp2);

n3<= not (inp3);

n4<= inp1 or n2;

n5<= n4 and inp3;

n6<= n1 and (inp2 and n3);

out1<= n5 or n6;

end process;

end Inp3_out1_arch;
```

Step 1 of 3

Draw the following logic circuit.

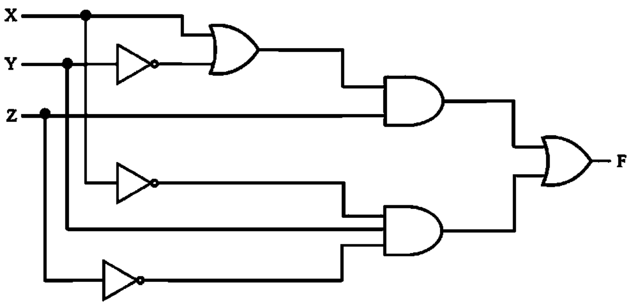


Figure 1: A 3-Input, 1-Output logic circuit

Step 2 of 3

The dataflow style of VHDL program for 3-Input, 1-Output logic circuit is as follows:

```
library IEEE;
use IEEE.std_logic_1164.all;
entity Inp3_out1 is
port ( inp1,inp2,inp3: in STD_LOGIC;
out1: out STD_LOGIC);
end Inp3_out1;
architecture Inp3_out1_arch of Inp3_out1 is
signal n1, n2, n3,n4,n5,n6: STD_LOGIC;
begin
n1 <= not (inp1);
n2<= not (inp2);
n3<= not (inp3);
n4<= inp1 or n2;
n5<= n4 and inp3;
n6<= n1 and (inp2 and n3);
out1<= n5 or n6;
end Inp3_out1_arch;
```

Step 3 of 3

The test bench for the program is as follows:

```
library IEEE;
use IEEE.standard_logice_1164.all;
library unisim;
use unisim.vcomponents.all;
entity Inp3_out1_tb is
end Inp3_out1_tb;
architecture Inp3_tb_arch of Inp3_out1_tb is
component Inp3_out1 port ( inp1,inp2,inp3: in STD_LOGIC;
out1: out STD_LOGIC);
end component;
signal xt, yt, zt, f: BIT;
begin
U1: Inp3_out1 port map( xt, yt, zt, f );
process
begin
xt<='0'; yt<='0'; zt<='0';
wait for 10 ns;
xt<='0'; yt<='0'; zt<='1';
wait for 10 ns;
xt<='0'; yt<='1'; zt<='0';
wait for 10 ns;
xt<='0'; yt<='1'; zt<='1';
wait for 10 ns;
xt<='1'; yt<='0'; zt<='0';
wait for 10 ns;
xt<='1'; yt<='0'; zt<='1';
wait for 10 ns;
xt<='1'; yt<='1'; zt<='0';
wait for 10 ns;
xt<='1'; yt<='1'; zt<='1';
wait;
end process;
end Inp3_tb_arch;
```

Step 1 of 5

5.10DP

Consider the NAND gate based logic circuit,

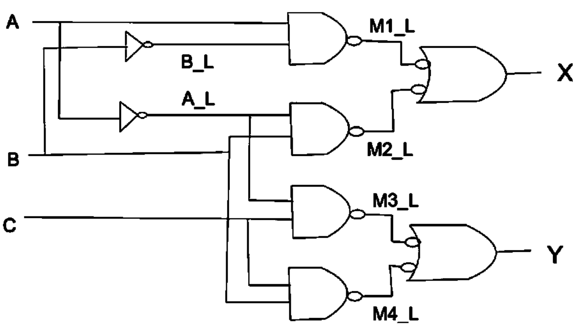


Figure 1: NAND gate based logic circuit

Step 2 of 5

The structural VHDL program for the above NAND gate based logic circuit is as shown below,

```
library IEEE;
use IEEE.standard_logic_1164.all;
entity nand_circuit is
port ( A, B, C : in STD_LOGIC;
X,Y : out STD_LOGIC);
end nand_circuit;
architecture nand_circuit_arch of nand_circuit is
signal A_L, B_L, M1_L,M2_L,M3_L,M4_L : STD_LOGIC;
component INV port( I: in STD_LOGIC;
O: out STD_LOGIC);
end component;
component NAND2 port( inp1, inp2: in STD_LOGIC;
out1: out STD_LOGIC);
end component;
component INV_OR2 port( inp1, inp2: in STD_LOGIC;
out1: out STD_LOGIC);
end component;
begin
U1: INV port map (A, A_L);
U2: INV port map (B, B_L);
U3: NAND2 port map (A, B_L, M1_L);
U4: NAND2 port map (A_L, B, M2_L);
U5: NAND2 port map (A_L, C, M3_L);
U6: NAND2 port map (B, C, M4_L);
U7: INV_OR2 port map (M1_L, M2_L, X);
U8: INV_OR2 port map (M3_L, M4_L, Y);
end nand_circuit_arch;
```

Step 3 of 5

```
-- component description
-- INV
entity INV is
    port  ( I: in STD_LOGIC;
          O: out STD_LOGIC);
 end INV;

architecture INV_arch of INV is
begin
O <= not ( I );
end INV_arch;
```

Step 4 of 5

```
-- component description
--- NAND2
entity NAND2 is
    port  ( inp1, inp2: in STD_LOGIC;
          out1: out STD_LOGIC);
    end NAND2;

architecture NAND2_arch of NAND2 is
begin
out1 <= not ( inp1 and inp2);
end NAND2_arch;
```

Step 5 of 5

```
-- component description
--INV_OR2
entity INV_OR2 is
    port  ( inp1, inp2: in STD_LOGIC;
          out1: out STD_LOGIC);
    end INV_OR2;

architecture INV_OR2_arch of INV_OR2 is
begin
out1 <= (not (inp1)) or (not (inp2));
end INV_OR2_arch;
```

Step 1 of 1 **5.11DP**

In a Verilog program, always block is having the assignment operator of '=' symbol means, it is a blocking assignments.

Blocking assignment means that the code should be executed sequentially. So the execution of the next step will proceed only if the current step is executed. The non-blocking assignment '<=, makes the statement can be executed in concurrent or parallel. Or we can say that it allow scheduling assignments without blocking the procedural ◊ow.

In a combinational logic, blocking assignment (=) should be used when you want to change an output that changes its value as soon as one or more of its inputs change. Whereas in Non-blocking assignment '<=', it takes the previous value.

Thus, the Verilog always block is intended to synthesize combinational logic is $\boxed{=}$.

Step 1 of 1

5.12DP

In a Verilog program, if multiple values are assigned to the same signal in a Verilog combinational always block. When it completes the execution, the signal value is the last value assigned. Because in the combinational logic always block, blocking assignment operator is used '=' and it makes the steps to be executed sequentially. So the last value assigned will be the signal's value.

Thus, the correct option is $\boxed{\textbf{(c) the last value assigned}}$.

Step 1 of 2

5.14DP

Draw the following logic circuit.

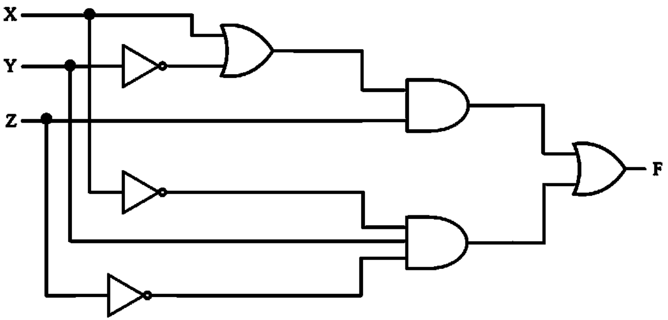


Figure 1: A 3-Input, 1-Output logic circuit

Step 2 of 2

From Figure 1, the structural Verilog module for 3-Input, 1-Output logic circuit is as follows:

```
module Inp3_out1 ( inp1, inp2, inp3, out1);

input inp1,inp2,inp3;

output out1;

wire n1, n2, n3,n4,n5,n6;

not U1 (n1, inp1);

not U2( n2, inp2);

not U3( n3, inp3);

or U4( n4, inp1, n2);

and U5( n5, n4, inp3);

and U6( n6, n1, inp2, n3);

or U7( out1, n5,n6);

endmodule
```

Step 1 of 2

5.16DP

Draw the following logic circuit:

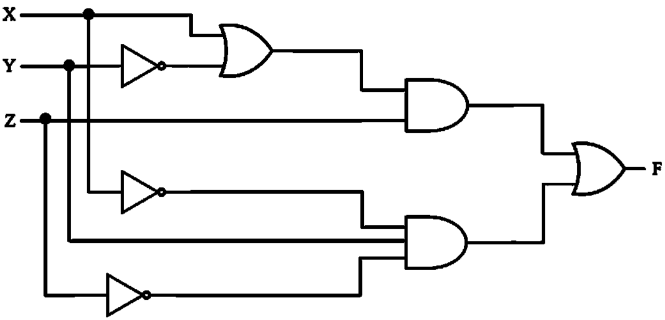


Figure 1: A 3-Input, 1-Output logic circuit

Step 2 of 2

From Figure 1, the dataflow Verilog module for 3-Input, 1-Output logic circuit using always block is as follows:

```
module Inp3_out1 ( inp1, inp2, inp3, out1);

input inp1,inp2,inp3;

output out1;

reg out1;

always @(inp1,inp2,inp3)

begin

reg n1, n2, n3;

n1= inp1 | ~(inp2);

n2= n1 & inp3;

n3= ~(inp1) & inp2 & ~(inp3);

out1= n2 | n3;

end

endmodule
```

Step 1 of 3

5.17DP

Below diagram gives the NAND gate based logic circuit,

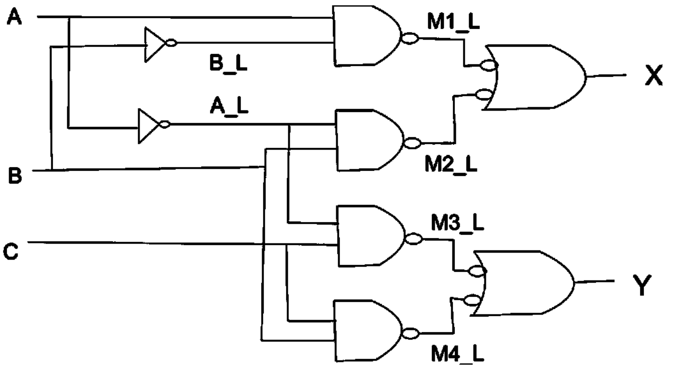


Figure 1: NAND gate based logic circuit

Step 2 of 3

The structural Verilog program for the above NAND gate based logic circuit is as shown below,

```
module nand_circuit (A, B, C, X, Y);

input A, B, C;

output X,Y;

wire A_L, B_L, M1_L, M2_L, M3_L, M4_L;

not U1(A_L, A);

not U2( B_L, B);

nand U3( M1_L, A, B_L);

nand U4( M2_L, A_L, B);

nand U5( M3_L, A_L, C);

nand U6( M4_L, C, B);

inv_or U7(M1_L, M2_L, X);

inv_or U8(M3_L, M4_L, Y);

endmodule
```

Step 3 of 3

The component inv_or's Verilog program as below,

```
module inv_or ( inp1, inp2, out1);

input inp1, inp2;

output out1;

wire n1, n2;

not U1(n1, inp1);

not U2(n2, inp2);

or U3(out1, n1, n2);

endmodule
```

Step 1 of 3

5.18DP

The verilog program for XOR gate given as per context is as follows,

```
module vxor(inpl,inp2,out);

input inpl,inp2;

output out;

wire Il,I2,notI2,notout;

vinh UI .out(Il),.inp(inpl),.inpv(inp2));

vinh U2 .out(I2),.inp(inp2),.inpv(inpl));

not U3 (notI2,I2);

vinh U4 (.out(notout),.in(notI2),.inpv(Il));

not U5 (out, notout);

endmodule
```

Step 2 of 3

The verilog program for component vrinh circuit is given by,

```
module Vinh(inp,inpv,out);

input inp,inpv;

output out;

wire notinpv;

not U1(notinpv, inpv);

and U2(out, in,notinpv);

endmodule
```

Step 3 of 3

When we synthesize the above circuit, synthesizer realizes the XOR function but no single XOR gate. It is realize the XOR function using 2 AND gate and 4 NOT gate.

Step 1 of 1

5.019E

A combinational logic function which has six input bits, consider as N5-N0 representing an integer between 0 and 63. There are two outputs for the function consider as M3 and M5, which indicate whether the number is a multiple of 3 or 5 respectively.

The following is the ABEL program for this logic function:

```
module mul3and5

title 'Numbers multiple of 3 and 5'

" Input and Output pins

N0, N1, N2, N3, N4, N5 pin;

M3, M5 pin is type 'com';

"Definition

NUM = [N5,N4,N3,N2,N1,N0];

equations

when (( NUM % 3) == 0) then M3=1;

else M3=0;

when (( NUM % 5) == 0) then M5=1;

else M5=0;

end mul3and5
```

Step 1 of 2

A combinational logic function which has six input bits, consider as N5-N0 representing an integer between 0 and 63. There are two outputs for the function consider as M3 and M5, which indicate whether the number is a multiple of 3 or 5 respectively.

ABEL program for this logic function is given below:

```
module mul3and5

title 'Numbers multiple of 3 and 5'

" Input and Output pins

N0, N1, N2, N3, N4, N5 pin;

M3, M5 pin is type 'com';

"Definition

NUM = [N5,N4,N3,N2,N1,N0];

equations

when (( NUM % 3) == 0) then M3=1;

else M3=0;

when (( NUM % 5) == 0) then M5=1;

else M5=0;

end mul3and5
```

Step 2 of 2

The test vector for the above ABEL program that checks the output of circuit for all multiples of 7 is given by as follows:

```
test_vectors

([N5,N4,N3,N2,N1,N0] -> [M3,M5])

[ 0, 0, 0, 1, 1, 1] -> [ 0, 0];

[ 0, 0, 1, 1, 1, 0] -> [ 0, 0];

[ 0, 1, 0, 1, 0, 1] -> [ 1, 0];

[ 0, 1, 1, 1, 0, 0] -> [ 0, 0];

[ 1, 0, 0, 0, 1, 1] -> [ 0, 1];

[ 1, 0, 1, 0, 1, 0] -> [ 1, 0];

[ 1, 1, 0, 0, 0, 1] -> [ 0, 0];

[ 1, 1, 1, 0, 0, 0] -> [ 0, 0];

[ 1, 1, 1, 1, 1, 1] -> [ 1, 0];
```

Step 1 of 4

Below ABEL program gives the WHEN statement examples as per context,

```
module Whentry
title 'Example of WHEN statement'
"Input pins
P, Q, R, S, T, U pin;
"output pins
X1, X1A, X2, X2A, X3, X3A, X4 pin istype 'com';
X5, X6, X7, X8, X9, X10 pin istype 'com';
equations
when (!P # Q) then X1 = R & !S;
X1A = (!P # Q) & (R & !S);
when (P & Q) then X2 = R # S;
else X2 = T # U;
X2A = (P & Q) & ( R # S)
# !(P & Q) & (T # U);
```

Step 2 of 4

```
when ( P ) then X3= S;
else when ( Q ) then X3=T;
else when ( R ) then X3=U;
X3A = ( P ) & ( S )
# !( P ) & ( Q ) & ( T )
# !( P ) & !( Q ) & ( R ) & ( U );
when ( P ) then
( when ( Q ) then X4 = S;}
else X4 = T;
when ( P & Q ) then X5 = S;
else when ( P # !R ) then X6 = T;
else when ( Q # R ) then X7 = U;
when ( P ) then {
X8 = S & T & U;
when ( Q ) then X8 = 1; else { X9= S; X10 = T;}
} else {
X8 = !S # !T;
When ( S ) then X9 = 1;
{X10 = R & S;}
}
end Whentry
```

Step 3 of 4

The above program can be rewritten by removing WHEN statement for the variable X4 through X10 as follows,

```
module Whentry
title 'Example of WHEN statement'
"Input pins
P, Q, R, S, T, U pin;
"output pins
X1, X1A, X2, X2A, X3, X3A, X4 pin istype 'com';
X5, X6, X7, X8, X9, X10 pin istype 'com';
equations
when (!P # Q) then X1 = R & !S;
X1A = (!P # Q) & (R & !S);
when (P & Q) then X2 = R # S;
else X2 = T # U;
X2A = (P & Q) & ( R # S)
# !(P & Q) & (T # U);
```

Step 4 of 4

```
when ( P } then X3= S;
else when ( Q ) then X3=T;
else when ( R ) then X3=U;
X3A = ( P ) & ( S )
# !( P ) & ( Q ) & ( T )
# !( P ) & !( Q ) & ( R ) & ( U );
X4= ( P ) & ( Q ) & ( S )
# !( P ) & ( T );
X5 = (P & Q) & ( S );
X6= !(P & Q) & ( P # !R ) & ( T );
X7= !(P & Q) & !( P # !R ) & ( Q # R ) & ( U );
X8 = ( P ) & ( S & T & U )
# (P) & (Q)
# ! (P) & (!S # !T);
X9 = ( P ) & ! ( Q ) & S
# ! ( P ) & ( S ) ;
X10 = ( P ) & !( Q ) & ( T )
# ! ( P ) & ( R & S );
end Whentry
```

Step 1 of 5 5.022E

The equation for alarm circuit according to the context is given by,

$$\text{Alarm} = \text{Panic} + (\text{Enable}\cdot\text{Exiting}'\cdot\text{Secure}') \;\ldots\ldots\; (1)$$

Secure can also be written according to the context as,

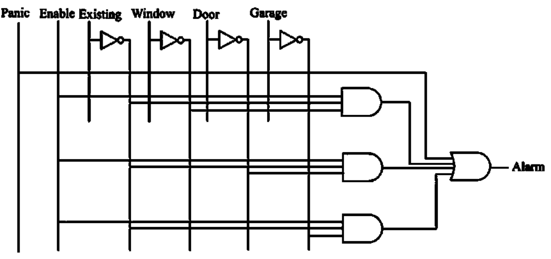
$$\text{Secure}=\text{Window}\cdot\text{Door}\cdot\text{Garrage} \;\ldots\ldots\; (2)$$

Substitute (2) in (1).

$$\text{Alarm} = \text{Panic} + (\text{Enable}\cdot\text{Exiting}'\cdot(\text{Window}\cdot\text{Door}\cdot\text{Garrage})')$$

$$\begin{aligned}\text{Alarm} &= \text{Panic} + (\text{Enable}\cdot\text{Exiting}'\cdot(\text{Window}\cdot\text{Door}\cdot\text{Garrage})') \\ &= \text{Panic} + (\text{Enable}\cdot\text{Exiting}'\cdot(\text{Window}' + \text{Door}' + \text{Garrage}')) \\ &= \begin{bmatrix}\text{Panic} + \text{Enable}\cdot\text{Exiting}'\cdot\text{Window}' + \text{Enable}\cdot\text{Exiting}'\cdot\text{Door}' \\ +\text{Enable}\cdot\text{Exiting}'\cdot\text{Garrage}'\end{bmatrix}\end{aligned}$$

Thus, two level AND-OR circuit for the above logic expression can be drawn as follows:



Step 2 of 5

Figure 1: Two level AND-OR circuit for Alarm circuit.

Step 3 of 5

The pair of numbers is written in below circuits for input and output of the gate. A pair of numbers consider as (t0, t1), where t0 is the test number that detect stuck-at-0 fault on that line and t1 is the test number that detect stuck-at-1 fault.

Thus, the Figure 2 gives the stuck at 0 and 1 fault in Panic line:

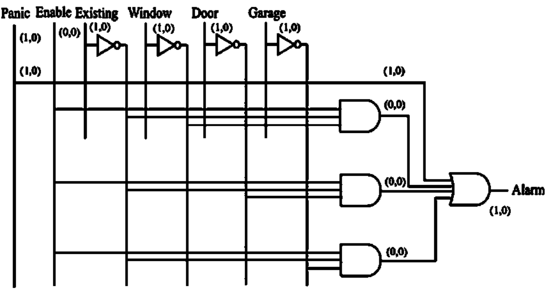


Figure 2: Test vector for panic line of the Alarm circuit

Step 4 of 5

Thus, the Figure 3 gives the Stuck at 0 and 1 fault in Enable, Exiting and Window line:

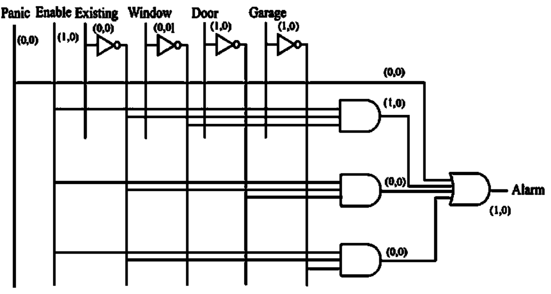


Figure 3: Test vector for Enable, Exiting and Window line of the Alarm circuit

Step 5 of 5

Thus, the Figure 4 gives the Stuck at 0 and 1 fault in Enable, Exiting and Door line.

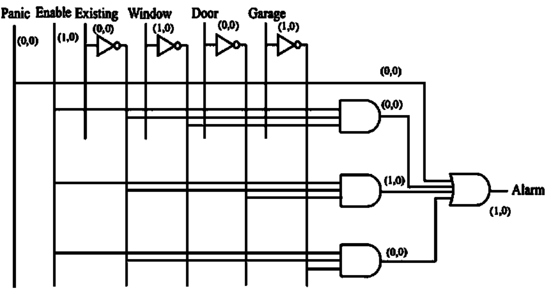


Figure 4: Test vector for Enable, Exiting and Door line of the Alarm circuit

Thus, the Figure 5 gives the Stuck at 0 and 1 fault in Enable, Exiting and Garage line.

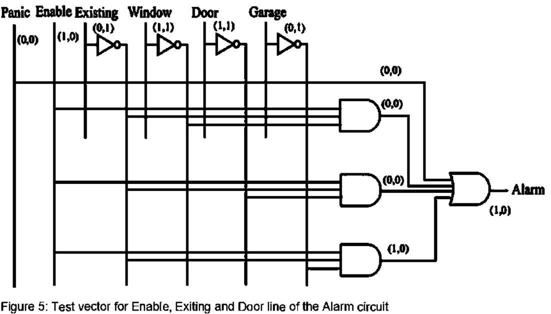


Figure 5: Test vector for Enable, Exiting and Door line of the Alarm circuit

Consider the VHDL program for the prime number detector circuit using a while statement:

```
library IEEE;

use IEEE.std_logic_1164.all;

entity prime9 is

Port ( N: in STD_LOGIC_VECTOR (15 downto 0);

F: out STD_LOGIC);

end prime9;

architecture prime9_arch of prime9 is

begin

process(N)

variable NI, i: INTEGER;

variable prime: Boolean;

begin

NI: CONV_INTEGER(N);

prime:= true;

if NI=1 or NI=2 then null;

else while i<=253 loop

if( NI mod i = 0) and (NI /= i) then

prime := false; exit;

end if;

i:=i+1;

end loop;

end if;

if prime then F<= ‘1’; else F<=’0’; end if;

end process;

end prime9_arch;
```

Step 1 of 2

A combinational logic function which has six input bits, consider as N5-N0 representing an integer between 0 and 63. There are two outputs for the function consider as M3 and M5, which indicate whether the number is a multiple of 3 or 5 respectively. VHDL program for this logic function is given below,

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mul3and5 is

port( N: in STD_LOGIC_VECTOR(5 downto 0);

M3,M5:out STD_LOGIC);

end mul3and5;
```

Step 2 of 2

```
architecture of mul3and5_arch of mul3and5 is

signal int_N: integer;

begin

process (N)

begin

int_N<= CONV_INTEGER (N);

if( int_N mod 3 = 0) then

M3<= 1;

else M3<=0;

end if;

if( int_N mod 5 = 0) then

M5<=1;

else M5<=0;

end if;

end process;

end mul3and5_arch;
```

Step 1 of 3

A combinational logic function which has six input bits, consider as N5-N0 representing an integer between 0 and 63. There are two outputs for the function consider as M3 and M5, which indicate whether the number is a multiple of 3 or 5 respectively. VHDL program for this logic function is given below,

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL
entity mul3and5 is
port( N: in STD_LOGIC_VECTOR(5 downto 0);
M3,M5:out STD_LOGIC);
end mul3and5;
```

Step 2 of 3

```
architecture of mul3and5_arch of mul3and5 is
signal int_N: integer;
begin
process (N)
begin
int_N<= CONV_INTEGER (N);
if( int_N mod 3 = 0) then
M3<= 1;
else M3<=0;
end if;
if( int_N mod 5 = 0) then
M5<=1;
else M5<=0;
end if;
end process;
end mul3and5_arch;
```

Step 3 of 3

The test bench for the above circuit can be written as follows,

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL
entity mul3and5_tb is
end mul3and5_tb;
architecture mul3and5_tb_arch of mul3and5_tb is
component mul3and5 port( N:in STD_LOGIC_VECTOR(5 downto 0);
M3,M5: out STD_LOGIC);
end component;
signal NT: STD_LOGIC_VECTOR(5 downto 0);
signal M3T, M5T: STD_LOGIC;
begin
U1: mul3and5 port map(NT,M3T,M5T);
Process
Begin
report "Beginning of test bench" severity NOTE;
NT<="000000"; wait for 10ns;
for i in 1 to 63 loop
NT<= NT+1; wait for 10ns;
assert(CONV_INTEGER (NT)> 0) report "error
NT< 0"& NT severity ERROR;
assert(CONV_INTEGER (NT)< 63) report "error
NT> 63"& NT severity ERROR;
end loop;
Report " Ending of testbench" severity NOTE;
wait;
end process;
end mul3and5_tb_arch;
```

Step 1 of 2

5.026E

Below program gives the prime number detector in structural modeling as per context

```
library IEEE;
use IEEE.std_logic_1164.all;
–The entity declaration for the input and output
entity prime_find is
port( Inp: in std_logic_vector ( 3 downto 0);
Out: out std_logic);
end prime_find;
– The architecture description for the prime number detector
architecture prime_arch of prime_find is
signal I1,I2,I3: STD_LOGIC;
signal A1, A2, A3, A4: STD_LOGIC;
-- Component declaration.
component INV port ( I: in STD_LOGIC; O: out STD_LOGIC);
end component;
component AND2 port (I0,I1: in STD_LOGIC;
O: out STD_LOGIC); end component;
component AND3 port (I0,I1,I2: in STD_LOGIC;
O: out STD_LOGIC); end component;
component OR4 port (I0,I1,I2,I3: in STD_LOGIC;
O: out STD_LOGIC); end component;
begin
-- port map for inverters.
U1: INV port map (Inp(3), I3);
U2: INV port map (Inp(2), I2);
U3: INV port map (Inp(1), I1);
-- port map for AND gate.
U4: AND2 port map (I3, Inp(0), A1);
U5: AND3 port map (I3, I2, Inp(1), A2);
U6: AND3 port map (I2, Inp(1), Inp(0), A3);
U7: AND3 port map (Inp(2), I1, Inp(0), A4);
-- port map for OR gate.
U8: OR4 port map (A1, A2, A3, A4, Out);
end prime_arch;
```

Step 2 of 2

The test bench for the above program is given by as follows,

```
library IEEE;
use IEEE.std_logic_1164.all;
--The entity declaration
entity prime_tb is
end prime_tb;
– The architecture description for the prime number detector test bench
architecture prime_tb_arch of prime_tb is
component prime_find
port( Inp: in std_logic_vector ( 3 downto 0);
Out: out std_logic);
end component;
signal NT: std_logic_vector ( 3 downto 0);
signal FT: std_logic;
begin
U1: prime port map (NT, FT);
process
begin
-- The input value for the prime number detector.
NT<="0000"; wait for 10ns;
for i in 1 to 15 loop
NT<= NT+1; wait for 10ns;
end loop;
-- Ending of test bench.
Report " Ending of testbench" severity NOTE;
wait;
end process;
end prime_tb_arch;
```

Step 1 of 4

The test bench VHDL program for prime number detector circuit is written below,

```
library IEEE;
use IEEE.std_logic_1164.all;
entity prime_tb is
end prime_tb;
architecture prime_tb_arch of tb is
component prime port (N: in STD_LOGIC_VECTOR ( 3 downto 0);
F: out STD_LOGIC);
end component;
signal Num: STD_LOGIC_VECTOR ( 3 downto 0), prime_out: bit;
//
begin
U1: prime port map ( Num, prime_out);
process
begin
report “Beginning of test bench” severity NOTE;
```

Step 2 of 4

```
//Continue the above program
Num<=”0000” ; wait for 10 ns;
Assert ( prime_out = ‘0’) report “ Failed ---- 0000” severity ERROR;
Num<=”0001” ; wait for 10 ns;
Assert ( prime_out = ‘1’) report “ Failed ---- 0001” severity ERROR;
Num<=”0010” ; wait for 10 ns;
Assert ( prime_out = ‘1’) report “ Failed —— 0010” severity ERROR;
Num<=”0011“ ; wait for 10 ns;
Assert ( prime_out = ‘1’) report “ Failed ---- 0011” severity ERROR;
Num<=”0100” ; wait for 10 ns;
Assert ( prime_out = ‘0’) report “ Failed ---- 0100” severity ERROR;
Num<=”0101“ ; wait for 10 ns;
Assert ( prime_out = ‘1’) report “ Failed ---- 0101” severity ERROR;
Num<=”0110” ; wait for 10 ns;
Assert ( prime_out = ‘0’) report “ Failed ---- 0110” severity ERROR;
```

Step 3 of 4

```
//Continue the above program
Num<=”0111” ; wait for 10 ns;
Assert ( prime_out = ‘1’) report “ Failed —— 0111” severity ERROR;
Num<=”1000” ; wait for 10 ns;
Assert ( prime_out = ‘0’) report ” Failed ---- 1000” severity ERROR;
Num<=”1001” ; wait for 10 ns;
Assert ( prime_out = ‘0’) report “ Failed ---- 1001” severity ERROR;
Num<=”1010“ ; wait for 10 ns;
Assert ( prime_out = ‘0’) report “ Failed ---- 1010” severity ERROR;
Num<=”1011” ; wait for 10 ns;
Assert ( prime_out = ’1’) report “ Failed ---- 1011” severity ERROR;
```

Step 4 of 4

```
//Continue the above program
Num<=”1100” ; wait for 10 ns;
Assert ( prime_out = ‘0’) report “ Failed —— 1100” severity ERROR;
Num<=”1101“ ; wait for 10 ns;
Assert ( prime_out = ‘1’) report “ Failed ---- 1101” severity ERROR;
Num<=”1110” ; wait for 10 ns;
Assert ( prime_out = ‘0’) report “ Failed ---- 1110” severity ERROR;
Num<=”1111“ ; wait for 10 ns;
Assert ( prime_out = ‘0’) report “ Failed ---- 1111” severity ERROR;
Report “ Ending of testbench” severity NOTE;
Wait;
End process;
End prime_tb_arch ;
```

Step 1 of 5

Draw the following gate level circuit diagram for full adder:

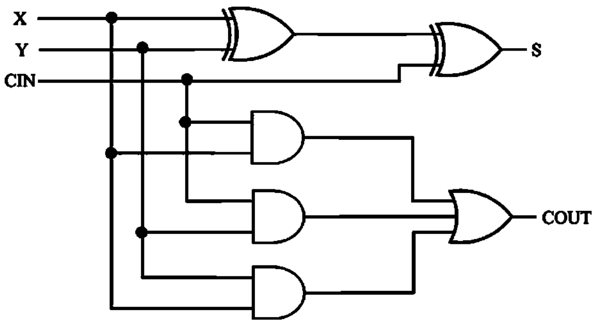


Figure 1 Gate level circuit diagram

Step 2 of 5

Draw the following logic diagram for full adder:

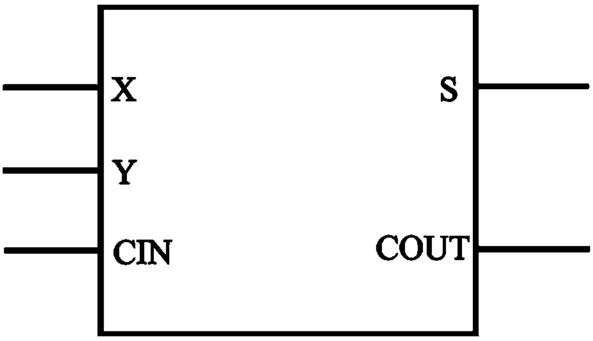
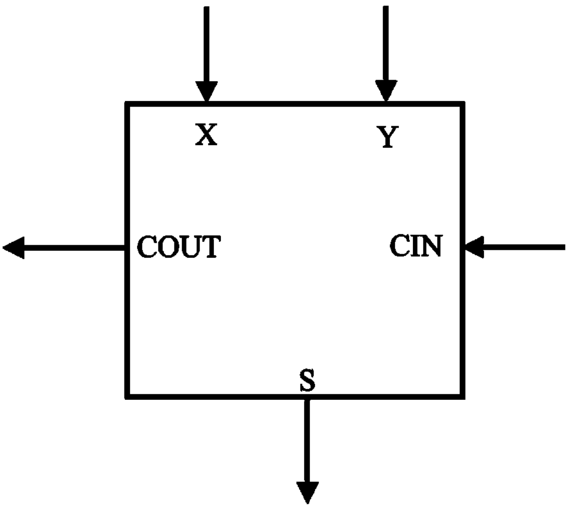


Figure 2 Logic diagram

Step 3 of 5

Draw the following alternate logic symbol for full adder:



Step 4 of 5

Figure 3 Alternate logic symbol suitable for cascading

Step 5 of 5

From Figure 1, the dataflow-style of VHDL program (entity and architecture) for Full adder circuit is as follows:

```
library IEEE;
use IEEE.std_logic_1164.all;
entity full_adder is
port (X, Y, CIN : in STD_LOGIC;
S, COUT: out STD_LOGIC);
end full_adder;
architecture full_adder_arch of full_adder is
begin
S<= CIN XOR (X XOR Y);
COUT<=(X AND Y) OR (CIN AND X) OR (CIN AND Y);
end full_adder_arch;
```

Step 1 of 2

5.029E

The dataflow style of VHDL program for Full adder circuit is written below,

```
library IEEE;

use IEEE.std_logic_1164.all;

– entity declaration of full adder circuit.

entity full_adder is

port ( xin, yin, cin : in STD_LOGIC;

sum,cout: out STD_LOGIC);

end full_adder;

– architecture declaration of full adder circuit.

architecture full_adder_arch of full_adder is

begin

sum<= cin XOR ( xin XOR yin);

cout<=(xin AND yin) OR (cin AND xin) OR (cin AND yin);

end full_adder_arch;
```

Step 2 of 2

The structural VHDL program for a 4-bit ripple carry adder using the above adder circuit is given below,

```
library IEEE;

use IEEE.std_logic_1164.all;

– The entity declaration of ripple carry adder circuit.

entity adder_rc_4bit is

port( x, y: in STD_LOGIC_VECTOR ( 3 downto 0);

s : out STD_LOGIC_VECTOR ( 3 downto 0);

cout: out STD_LOGIC);

end adder_rc_4bit;

– The architecture declaration of ripple carry adder.

architecture adder_rc_arch of adder_rc_4bit is

signal c: STD_LOGIC_VECTOR ( 4 downto 0);

– Component declaration of full adder.

component full_adder port ( xin, yin, cin : in STD_LOGIC; sum,cout: out STD_LOGIC);

end component;

begin

c(0)<='0';

– port mapping the full adder circuit.

U1: full_adder port map(x(0),y(0),c(0),s(0),c(1));

U2: full_adder port map(x(1),y(1),c(1),s(1),c(2));

U3: full_adder port map(x(2),y(2),c(2),s(2),c(3));

U4: full_adder port map(x(3),y(3),c(3),s(3),c(4));

cout<=c(4);

end adder_rc_arch;
```

Step 1 of 3 5.030E

The dataflow style of VHDL program for Full adder circuit is written below,

```
library IEEE;
use IEEE.std_logic_1164.all;
– entity declaration of full adder circuit.
entity full_adder is
port ( xin, yin, cin : in STD_LOGIC;
sum,cout: out STD_LOGIC);
end full_adder;
-- architecture declaration of full adder circuit.
architecture full_adder_arch of full_adder is
begin
sum<= cin XOR ( xin XOR yin);
cout<=(xin AND yin) OR (cin AND xin) OR (cin AND yin);
end full_adder_arch;
```

Step 2 of 3

The structural VHDL program for a 4-bit ripple carry adder using the above adder circuit is given below,

```
library IEEE;
use IEEE.std_logic_1164.all;
– The entity declaration of ripple carry adder circuit.
entity adder_rc_4bit is
port( x, y: in STD_LOGIC_VECTOR ( 3 downto 0);
s : out STD_LOGIC_VECTOR ( 3 downto 0);
cout: out STD_LOGIC);
end adder_rc_4bit;
-- The architecture declaration of ripple carry adder.
architecture adder_rc_arch of adder_rc_4bit is
signal c: STD_LOGIC_VECTOR ( 4 downto 0);
– Component declaration of full adder.
component full_adder port ( xin, yin, cin : in STD_LOGIC; sum,cout: out STD_LOGIC);
end component;
begin
c(0)<='0';
– port mapping the full adder circuit.
U1: full_adder port map(x(0),y(0),c(0),s(0),c(1));
U2: full_adder port map(x(1),y(1),c(1),s(1),c(2));
U3: full_adder port map(x(2),y(2),c(2),s(2),c(3));
U4: full_adder port map(x(3),y(3),c(3),s(3),c(4));
cout<=c(4);
end adder_rc_arch;
```

Step 3 of 3

The test bench for the above circuit or program is written as follows,

```
library IEEE;
use IEEE.std_logic_1164.all;
– The entity declaration of the test bench for the 4bit ripple carry adder.
entity adder_rc_tb is
end adder_rc_tb;
-- The architecture declaration of the test bench.
architecture adder_tb_arch of adder_rc_tb is
component adder_rc_4bit port
( x, y: in STD_LOGIC_VECTOR ( 3 downto 0);
s : out STD_LOGIC_VECTOR ( 3 downto 0);
cout: out STD_LOGIC);
end component;
signal xt, yt, st: STD_LOGIC_VECTOR ( 3 downto 0),
signal ct: STD_LOGIC;
begin
U1: adder_rc_4bit port map ( xt, yt, st, ct);
process
variable i, j : integer;
begin
report "Beginning of test bench" severity NOTE;
– The input value for the adder circuit.
for i in 0 to 3 loop
xt<=i;
for j in 0 to 3 loop
yt<=j; wait for 10ns;
assert(st=xt+yt)report"Failed --%b,%b",xt,yt,
severity ERROR;
end loop;
end loop;
report " Ending of testbench" severity NOTE;
Wait;
end process;
end adder_tb_arch;
```

Step 1 of 2

The dataflow style of VHDL program for Full adder circuit is written below,

```
library IEEE;

use IEEE.std_logic_1164.all;

entity full_adder is

port ( xin, yin, cin : in STD_LOGIC;

sum,cout: out STD_LOGIC);

end full_adder;

architecture full_adder_arch of full_adder is

begin

sum<= cin XOR ( xin XOR yin);

cout<=(xin AND yin) OR (cin AND xin) OR (cin AND yin);

end full_adder_arch;
```

Step 2 of 2

The structural VHDL program for a 16-bit ripple carry adder using the above adder circuit is given below,

```
library IEEE;

use IEEE.std_logic_1164.all;

entity ripple_16 is

port (x, y: in STD_LOGIC_VECTOR ( 15 downto 0);

s: out STD_LOGIC_VECTOR ( 15 downto 0);

cout: out STD_LOGIC);

end ripple_16;

architecture ripple_16_arch of ripple_16 is

component full_adder

port ( xin, yin, cin : in STD_LOGIC;

sum,cout: out STD_LOGIC);

end component;

signal c: STD_LOGIC_VECTOR ( 16 downto 0);

begin

c(0) <= ‘0’;

g1: for i in 0 to 15 generate

U1:full_adder port map( x(i),y(i),c(i),s(i),c(i+1));

end generate;

cout<= c(16);

end ripple_16_arch;
```

Step 1 of 7

Draw the following gate level circuit diagram for full adder:

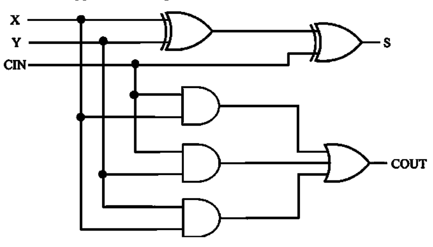


Figure 1 Gate level circuit diagram

Step 2 of 7

Draw the following logic diagram for full adder:

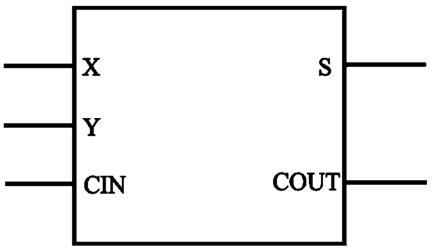
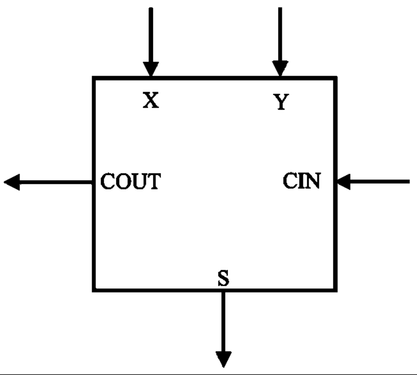


Figure 2 Logic diagram

Step 3 of 7

Draw the following alternate logic symbol for full adder:



Step 4 of 7

Figure 3 Alternate logic symbol suitable for cascading

Step 5 of 7

The dataflow style of VHDL program for Full adder circuit is as follows:

```
library IEEE;
use IEEE.std_logic_1164.all;
– The entity declaration of full adder circuit.
entity full_adder is
port ( xin, yin, cin : in STD_LDGIC;
sum,cout: out STD_LOGIC);
end full_adder;
– The architecture declaration of full adder circuit.
architecture full_adder_arch of full_adder is
begin
sum<= cin XOR ( xin XDR yin);
cout<=(xin AND yin) OR (cin AND xin) OR (cin AND yin);
end full_adder_arch;
```

Step 6 of 7

The structural VHDL program for a 16-bit ripple carry adder using the above adder circuit is as follows:

```
library IEEE;
use IEEE.std_logic_1164.all;
– The entity declaration of ripple carry adder circuit.
entity ripple_16 is
port (x, y: in STD_LOGIC_VECTOR ( 15 downto 0);
s: out STD_LOGIC_VECTOR ( 15 downto 0);
cout: out STD_LOGIC);
end ripple_16;
–The architecture declaration of ripple adder circuit.
architecture ripple_16_arch of ripple_16 is
–The component declaration.
component full_adder
port ( xin, yin, cin : in STD_LDGIC;
sum,cout: out STD_LOGIC);
end component;
signal c: STD_LOGIC_VECTDR ( 16 downto 0);
begin
c(0) <= '0';
–The component is called using generate statement.
g1: for i in 0 to 15 generate
U1:full_adder port map( x(i),y(i),c(i),s(i),c(i+1));
end generate;
cout<= c(16);
end ripple_16_arch;
```

Step 7 of 7

The test vector for a 16-bit ripple carry adder is given as follows:

```
library IEEE;
use IEEE.std_logic_1164.all;
–The entity declaration of test bench.
entity ripple_16_tb is
end ripple_16_tb;
architecture ripple_tb_arch of ripple_16_tb is
–The component declaration for a test bench.
component ripple_16
port (x, y: in STD_LOGIC_VECTOR ( 15 downto 0);
s: out STD_LOGIC_VECTOR ( 15 downto 0);
cout: out STD_LOGIC);
end component;
signal xt, yt, st: STD_LOGIC_VECTOR ( 15 downto 0),
signal ct: STD_LOGIC;
begin
U1: ripple_16 port map ( xt, yt, st, ct);
Process
Variable i, j : Integer;
begin
report "Beginning of test bench" severity NDTE;
–Giving all possible inputs to the ripple adder module.
for i in 0 to 15 loop
xt<=i;
for j in 0 to 15 loop
yt<=j; wait for 10ns;
assert(st=xt+yt)report"Failed --%b,%b",xt,yt,
severity ERROR;
end loop;
end loop;
report " Ending of testbench" severity NOTE;
Wait;
end process;
end ripple_tb_arch;
```

Step 1 of 2

Program for four way, 2-bit transceiver as per context is as follows:

```
module vbus(w,x,y,z,s,at,bt,ct,dt,et);

input [2:0] s;

input at,bt,ct,dt,et;

inout [1:8] w,x,y,z;

reg [1:8] ibus;

always@(w or x or y or z or s) begin

if (s[2] == 0) ibus = {4{s[1:0]}};

else case ( s[1:0] )

0: ibus = w;

1: ibus = x;

2: ibus = y;

3: ibus = z;

endcase

end

assign w=((~at & ~et) && (s[2:0] !=4)) ? ibus :8'bz;

assign x=((~bt & ~et) && (s[2:0] !=5)) ? ibus :8'bz;

assign y=((~ct & ~et) && (s[2:0] !=6)) ? ibus :8'bz;

assign z=((~dt & ~et) && (s[2:0] !=7)) ? ibus :8'bz;

endmodule
```

Step 2 of 2

If the experienced digital design engineer reviewed the above code, will give comment on logical condition if (s[2] == 0). Because s[2] == 0 can also written as !s[2] which reduce the number of character. Thus the above program can be modified as below,

```
module vbus(w,x,y,z,s,at,bt,ct,dt,et);

input [2:0] s;

input at,bt,ct,dt,et;

inout [1:8] w,x,y,z;

reg [1:8] ibus;

always@(w or x or y or z or s) begin

if ( !s[2] ) ibus = {4{s[1:0]}};

else case ( S[1:0] )

0: ibus = w;

1: ibus = x;

2: ibus = y;

3: ibus = z;

endcase

end

assign w=((~at & ~et) && (s[2:0] !=4)) ? ibus :8'bz;

assign x=((~bt & ~et) && (s[2:0] !=5)) ? ibus :8'bz;

assign y=((~ct & ~et) && (s[2:0] !=6)) ? ibus :8'bz;

assign z=((~dt & ~et) && (s[2:0] !=7)) ? ibus :8'bz;

endmodule
```

Step 1 of 1 **5.034E**

A combinational logic function which has six input bits, consider as N5-N0 representing an integer between 0 and 63. There are two outputs for the function consider as M3 and M5, which indicate whether the number is a multiple of 3 or 5 respectively. VHDL program for this logic function is given below,

```
module mul3and5 (N,M3,M5);

input [5:0] N;

output M3,M5;

reg M3,M5;

integer I;

always @( N)

begin

if( (N % 3) = = 0)

M3= 1;

else M3=0;

if( (N % 5) = = 0)

M5=1;

else M5=0;

end

endmodule
```

Step 1 of 2 5.035E

A combinational logic function which has six input bits, consider as N5-N0 representing an integer between 0 and 63. There are two outputs for the function consider as M3 and M5, which indicate whether the number is a multiple of 3 or 5 respectively. Verilog program for this logic function is given below,

```
module mul3and5 (N,M3,M5);

input [5:0] N;

output M3,M5;

reg M3,M5;

integer I;

always @( N)

begin

if( (N % 3) = = 0)

M3= 1;

else M3=0;

if( (N % 5) = = 0)

M5=1;

else M5=0;

end

endmodule
```

Step 2 of 2

The test bench for the above program is given by,

```
~ timescale 1ns/100ps

module mul3and5_tb( );

reg [5:0] NT;

wire M3T,M5T;

tast check;

input expect;

If ((M3T!= expect) and (M5T != expect))

$ display (" Error: NT= %b, expected= %b" NT,expect);

endtask

mul3and5 UUT( .N(NT), .M3(M3T), .M5(M5T));

initial begin : TB

integer i;

for ( i=0; i<=63; i=i+1) begin

NT= i;

#10 // wait for 10ns per iteration

Check (NT);

end

end

endmodule
```

Step 1 of 2

5.036E

The program for to find out 8-bit prime numbers as per the textbook is as follows:

```
module prime_8( inp, out);

input [7:0] inp;

output out;

reg out,z;

integer i;

always@ (inp) begin

z=1;

if((inp ==1) || (inp ==2))

z=1;

else if((inp%2) ==0)

z=0;

else for ( i=3; i<=15; i=i+2)

if ((inp%i) ==0) && inp !=i)

z=0;

if (z ==1) out =1; else out = 0;

end

endmodule
```

Step 2 of 2

The corresponding test bench for the above program is as follows:

```
timescale 1 ns/ 100 ps

module prime_tb( );

reg [7:0] inpt;

wire zt;

prime_8 UUT ( .inp(inpt), .out(zt));

initial begin: TB

integer i;

for ( i=0; i<=255; i=i+1) begin

inpt=i;

#10

If(zt) $display ("The number %d is prime", inpt);

end

end

endmodule
```

Step 1 of 5

Draw the following gate level circuit diagram for full adder:

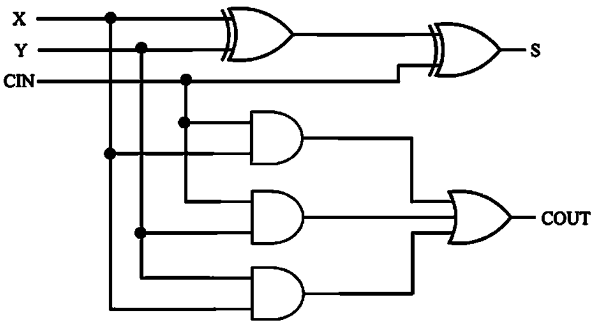


Figure 1 Gate level circuit diagram

Step 2 of 5

Draw the following logic diagram for full adder:

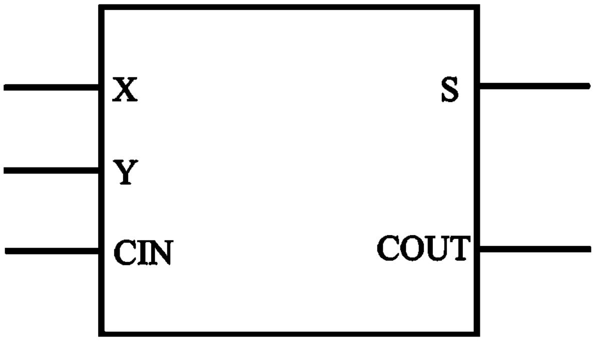
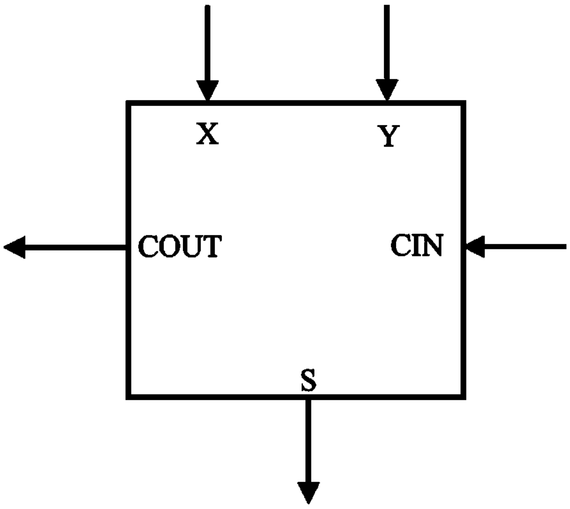


Figure 2 Logic diagram

Step 3 of 5

Draw the following alternate logic symbol for full adder:



Step 4 of 5

Figure 3 Alternate logic symbol suitable for cascading

Step 5 of 5

From Figure 1, the dataflow-style of verilog program corresponding to Full adder circuit is as follows:

```
module full_adder ( xin, yin, cin, sum, cout);

input xin, yin, cin;

output sum, cout;

assign sum = cin ^ ( xin ^ yin);

assign cout=(xin & yin) | (yin & cin) | (cin & xin);

endmodule;
```

Step 1 of 2

5.038E

The dataflow style of verilog program for Full adder circuit is written below,

```
module full_adder ( xin, yin, cin, sum, cout);

input xin, yin, cin;

output sum, cout;

assign sum = cin ^ ( xin ^ yin);

assign cout=( xin & yin ) | ( yin & cin ) | ( cin & xin );

endmodule;
```

Step 2 of 2

The Verilog program for a 4-bit ripple adder using the above full adder circuit is given by as follows,

```
module ripple_4(x,y,sum,cout);

input [3:0] x;

input [3:0] y;

output [3:0] sum;

output cout;

wire [4:0] c;

c[0]=0;

full_adder U1 (x[0],y[0],c[0],s[0],c[1]);

full_adder U2 (x[1],y[1],c[1],s[1],c[2]);

full_adder U1 (x[2],y[2],c[2],s[2],c[3]);

full_adder U1 (x[3],y[3],c[3],s[3],c[4]);

cout = c[4];

endmodule
```

Step 1 of 3

5.040E

The dataflow style of verilog program for Full adder circuit is written below,

```
module full_adder ( xin, yin, cin, sum, cout);

input xin, yin, cin;

output sum, cout;

assign sum = cin ^ ( xin ^ yin);

assign cout=( xin & yin ) | ( yin & cin ) | ( cin & xin );

endmodule;
```

Step 2 of 3

The Verilog program for a 4-bit ripple adder using the above full adder circuit is given by as follows,

```
module ripple_4(x,y,sum,cout);

input [3:0] x;

input [3:0] y;

output [3:0] sum;

output cout;

wire [4:0] c;

c[0]=0;

full_adder U1 (x[0],y[0],c[0],s[0],c[1]);

full_adder U2 (x[1],y[1],c[1],s[1],c[2]);

full_adder U1 (x[2],y[2],c[2],s[2],c[3]);

full_adder U1 (x[3],y[3],c[3],s[3],c[4]);

cout = c[4];

endmodule
```

Step 3 of 3

The test bench for the above circuit is given by as follows,

```
~ timescale 1ns/100ps

module ripple_4_tb( );

reg [3:0] xt, [3:0] yt;

wire [3:0] st,ct;

tast check;

input expect;

If ((st!= expect) and (ct != expect))

$ display (" Error: xt=%b,yt=%b, expected= %b" xt,yt,expect);

endtask

ripple_4 UUT( .x(xt), .y(yt),.sum(st),.cout(ct));

initial begin : TB

integer i;

for ( i=0; i<=15; i=i+1) begin

xt= i;

for ( j=0; j<=15; j=j+1) begin

yt=j;

#10 // wait for 10ns per iteration

check (xt,yt);

end

end

end

endmodule
```

5.041E

Step 1 of 7

Structural Verilog program is to be written for the 16 bit ripple adder, first declare one module of full adder and then use generate statement to design the 16 bit ripple adder.

The Verilog module for full_adder module in Xilinx is as follows:

```
module full_adder ( xin, yin, cin, sum, cout);

input xin, yin, cin;

output sum, cout;

assign sum = cin ^ ( xin ^ yin);

assign cout=( xin & yin ) | ( yin & cin ) | ( cin & xin );

endmodule
```

Step 2 of 7

The Verilog program for a 16-bit ripple adder using the full adder module is as follows:

```
module ripple_16(x,y,c,sum,cout);

input [15:0] x;

input [15:0] y;

output [15:0] sum;

output cout;

inout wire [16:0] c;

genvar i;

assign c[0]=0;

generate

for(i=0; i<=15; i=i+1) begin : U

full_adder U1 (x[i],y[i],c[i],sum[i],c[i+1]);

end

endgenerate

assign cout = c[16];

endmodule
```

Step 3 of 7

write the following test bench program to test the adder for 2^{32} possible pairs of 16 bit addends.

```
module test;

// Inputs

reg [15:0] x;

reg [15:0] y;

// Outputs

wire [15:0] sum;

wire cout;

// Bidirs

wire [16:0] c;

// Instantiate the Unit Under Test (UUT)

ripple_16 uut (

.x(x),

.y(y),

.c(c),

.sum(sum),

.cout(cout)

);

integer i,j;

initial begin: m

x=0;

y=0;

#10 //delay of 10 ns

for (i=1; i<65536; i=i+1) begin

#10 x= i;

#10 //delay of 10 ns

for (j=1; j<65536; j=j+1) begin

#10 y=j; //delay of 10 ns

if(sum>65535)

disable m;

$display ("sum exceeds the limit");

end

end

end

endmodule
```

Step 4 of 7

Thus, the required output is that the test bench should stop and display the actual and expected outputs if there is any mismatch.

Since, in test bench the input combinations are given in such a way that every time both the inputs x and y are less than 16 bit, so an error or mismatch can't come in the input.

But after certain combination of inputs, the sum value exceeds the total declared range of 16 bit that is declared as the range for the output.

Thus, it is necessary to obtain the display whenever sum exceeds the limit.

The delay between every input is given in the range of nanoseconds.

So, simulate the behavioral model of the test bench and observe the output, run the output for almost 12 milliseconds so that within this time, there occurs a condition that sum exceeds the limit.

Step 5 of 7

Figure 1 represents the simulated output waveform executed for 12 milliseconds.

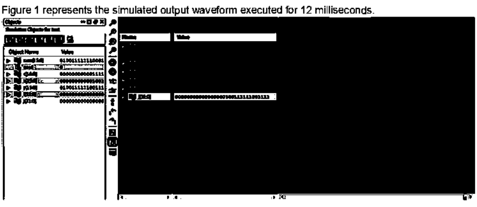
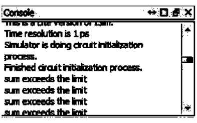


Figure 1

Step 6 of 7

Since during execution, sum value exceeds the limit of 16 bit.

Thus, Figure 2 represents the dialog box which is displayed when the sum is exceeding the limit range of 16 bit.



```
This is a Lite version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization
process.
Finished circuit initialization process.
sum exceeds the limit
sum exceeds the limit
sum exceeds the limit
sum exceeds the limit
```

Figure 2

Step 7 of 7

Thus, the test bench waveform is written such that the test bench stops and displays that there is a mismatch in the actual and expected outputs.

Step 1 of 2

In Verilog HDL, the logic operators results to a 1 bit. It may be 0 (false) or 1(true) or x(ambiguous) value. The ambiguous values means the value which is not equal to 0 and not equal to one. If the operand in any of the operation is having the value of x, then the resultant will be with the value x. The following Verilog program shows how Verilog handles the value x:

```
module veri_x( inp, out);

input [4:0] inp;

output out;

reg out,z;

always@ (inp) begin

z=4'b1xx0;

if((inp ==z))

$display("Condition is satisfied, input is %b",inp);

else

$display("Condition is not satisfied, input is %b",inp);

end

endmodule
```

Step 2 of 2

The test bench for the above program is as follows:

```
timescale 1 ns/ 100 ps

module verix_tb( );

reg [4:0] inpt;

wire zt;

veri_x UUT ( .inp(inpt), .out(zt));

initial begin: TB

integer i;

for ( i=0; i<=15; i=i+1) begin

inpt=i;

#10

end

end

endmodule
```

Step 1 of 2

The following is the program to find all the 8-bit prime numbers:

```
module prime_8( inp, out);

input [7:0] inp;

output out;

reg out,z;

integer i;

always@ (inp) begin

z=1;

if((inp ==1) || (inp ==2))

z=1;

else if((inp%2) ==0)

z=0;

else for ( i=3; i<=15; i=i+2)

if ((inp%i) ==0) && inp !=i)

z=0;

if (z ==1) out =1; else out = 0;

end

endmodule
```

Step 2 of 2

The test bench for the above program using Verilog I/O file is as follows:

```
timescale 1 ns/ 100 ps

module prime_tb( );

reg [7:0] inpt;

wire zt;

reg expected_out;

integer m,r;

prime_8 UUT ( .inp(inpt), .out(zt));

initial begin: text_file

// To read text file into memory

m=$fopenr("mem_in.txt");

end

initial begin: TB

// Till the end of file, need to read the text.

while(!$feof(m))

begin

r=$fscanf(m, "%b,%b
", inpt, expected_out;

If(zt==expected_out)

$display ("correct output %b", inpt);

else

$display ("error output %b", inpt);

end

end

endmodule
```

Billions of billons of rows in a truth table means at least 4 billion or approximately 2^{32} of rows are needed; it is possible only when the input bits are at least 32.

Following are the three examples of combinational logic circuits that require "billions and billions" of rows to describe in a truth table.

Combinational Logic Inputs Outputs Rows (Truth Table)

Circuits

1. 128 to 1 Multiplexer 128 1 2^{128}

2. 32:5 Encoder 32 5 2^{32}

3. 64 bit Binary Adder 64 32 2^{128}

The truth table of 64 bit Binary Adder has 2^{128} rows for the input combination of addend 64 bits and augend 64 bits in total of 128 input combinations.

Step 1 of 3

6.03DP

The Function Table for a **74x27** 3-input NOR gate is given in Table 1.

INPUTS	OUTPUT		
A	B	C	Z
1	x	x	0
x	1	x	0
x	x	1	0
0	0	0	1

Table 1

Step 2 of 3

In the Function Table, representation of symbols is given as follows:

A-C: inputs to the gate

Z: output of the gate

0: Low Voltage level

1: High Voltage level

x: Don’t care condition

The output for the above NOR gate can be represented as:

$$\begin{aligned} Z &= \overline{A+B+C} \\ &= \bar{A}\bar{B}\bar{C} \end{aligned}$$

Thus, for 3-input NOR gate, the De-Morgan’s equivalent equation is $Z = \bar{A}\bar{B}\bar{C}$

Step 3 of 3

The functional diagram and the DeMorgan equivalent symbol for a **74x27** 3-input NOR gate is shown in Figure 1.

Figure1

Thus, the DeMorgan equivalent symbol for a **74x27** 3-input NOR gate is drawn.

Step 1 of 1

6.04DP

In real system circuits signal names are well chosen so as to convey the information to someone reading the logic circuit. A signal's name indicates an action that is controlled (GO, PAUSE), a condition that it detects (READY, ERROR), or data that it carries (INBUS[31:0]). The difference between a signal and an expression is that a signal is just a name, an alphanumeric label, whereas a logic expression combines signal names using the operators of switching algebra namely, AND, OR and NOT.

The problem with **READY′** is that it is an expression, not a signal since ′ is an unary operator.

Step 1 of 1

6.05DP

It is easier to think of using active high names for all the logic signals in a logic circuit but while realizing the circuit it may be necessary to deal with active-low signals due to the requirement of the environment.

While designing a logic circuit on board, speed is a key factor. Inverting gates are faster than the non-inverting gates. So, if only non-inverting gates are used to keep all the signals active-high, for convenience purpose, the speed of the circuit reduces. Hence, there is a significant performance payoff in carrying some signals in active-low form while designing real time circuits.

Step 1 of 2

6.06DP

When the discrete gates are designed at the board or ASIC level the speed of the gates is the main constraint. Inverting gates at the output are typically faster than the non-inverting, so there is often significance payoff in carrying some signals in active low form. Hence to increase the speed and the performance of the gates, gates are designed with a bubble output a non-bubble input.

Draw the logic gate in which x and y are non-bubble inputs and z is bubble output.

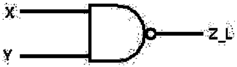


Figure 1

Step 2 of 2

Therefore, inverting gates like NAND and NOR are generally preferred over AND and OR gates respectively.

Step 1 of 2

Consider X and Y as inputs and Z as output:

Consider the following combination of bubble and non-bubble inputs and outputs.

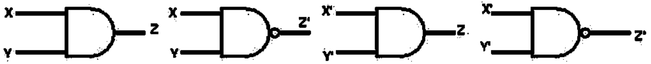


Figure 1

Step 2 of 2

The first logic gate has active high inputs and outputs.

The second logic gate has high inputs and active low output.

The third logic gate has active low inputs and active high output.

The fourth logic gate has active low inputs and active low output.

From all these combination of logic gates, it is clear that ***the all the inputs to a logic gate must have either bubble or non-bubble is TRUE.***

Step 1 of 1 **6.08DP**

Hierarchical schematic structure is preferred, because it is used when the network ports are identical. In Hierarchical schematic structure, top level schematic (in which only blocks) connected to the low level pages (in which gate level descriptions). In this particular low level hierarchy is used more than once that is one network port in twelve identical ports is designed using single gate level description and reused twelve times by the higher level hierarchy.

Step 1 of 2

6.09DP

Consider the circuit given below:

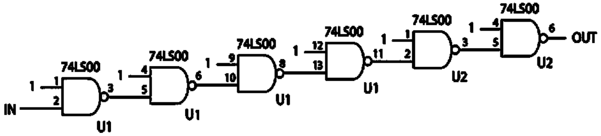


Figure1

Step 2 of 2

For the circuit given in Figure1 both LOW-to-HIGH and HIGH-to-LOW transitions cause positive transitions on the output of three gates (alternate gates) and negative transitions on the output of other three gates.]

Hence, the total delay is the same in both the cases and is given by,

$t_p = 3t_{\text{pLH(LS00)}} + 3t_{\text{pHL(LS00)}}$ ...... (1)

For 74LS00, maximum values for $t_{\text{pHL}} = 15\text{ ns}$ and $t_{\text{pLH}} = 15\text{ ns}$

Substitute 15 for t_{pHL} and 15 for t_{pLH} .

$$\begin{aligned} t_p &= (3)(15) + (3)(15) \\ &= 45 + 45 \\ &= 90\text{ ns} \end{aligned}$$

Thus, the exact propagation delay from IN to OUT of the circuit is $\boxed{90\text{ ns}}$.

Since, t_{pLH} and t_{pHL} for 74LS00 are equal, the same result is obtained using a single worst case delay of 15 ns

Step 1 of 2

Consider the circuit given below:

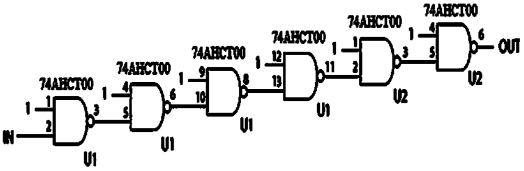


Figure1

Step 2 of 2

For the circuit given in Figure1 both LOW-to-HIGH and HIGH-to-LOW transitions cause positive transitions on the output of three gates (alternate gates) and negative transitions on the output of other three gates.

Hence, the total delay will be same in both the cases and is given by,

$t_p = 3t_{\text{pLH(LS00)}} + 3t_{\text{pHL(LS00)}}$ ...... (1)

For 74AHCT00, $t_{\text{pHL}} = 9\text{ ns}$ and $t_{\text{pLH}} = 9\text{ ns}$

Substitute 9 for t_{pHL} and 9 for t_{pLH} .

$$\begin{aligned} t_p &= 3\times 9 + 3\times 9 \\ &= 27 + 27 \\ &= 54\text{ ns} \end{aligned}$$

Thus, the minimum propagation delay from IN to OUT for the circuit is $\boxed{54\text{ ns}}$.

Since, t_{pLH} and t_{pHL} for a 74AHCT00 are equal, the same result is obtained using a single worst case delay of 9 ns.

Step 1 of 4

6.11DP

Refer Figure X6.9 and Table 6.2 in text book.

Design:

Replace 74LS00 gates in the Figure X6.9 with 74LS21, draw the modified circuit of Figure X6.9:

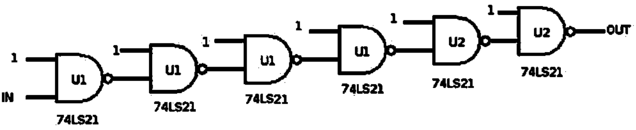


Figure 1

Step 2 of 4

Using the information in Table 6.2 and Figure X6.9, the values of t_{pLH} and t_{pHL} for 74LS21 are as follows,

$t_{\text{pLH}} = 15 \text{ ns}$ and $t_{\text{pHL}} = 20 \text{ ns}$

For the circuit Figure X6.9, both LOW-to-HIGH and HIGH-to-LOW transitions cause positive transitions on the output of three gates (alternate gates) and negative transitions on the output of other three gates.

Hence, the total delay is the same in both the cases and is,

$$\begin{aligned} t_p &= 3t_{\text{pLH(74LS21)}} + 3t_{\text{pHL(74LS21)}} \\ &= 3(15) + 3(20) \\ &= 45 + 60 \\ &= 105 \text{ ns} \end{aligned}$$

Thus, the exact propagation delay from IN to OUT of the circuit is $\boxed{105 \text{ ns}}$.

Step 3 of 4

Single worst case delay specification is the maximum of t_{pLH} and t_{pHL} specifications. So, single worst case delay for each gate is **20 ns.**

"The worst case delay through a circuit is computed as the sum of the worst case delays through the individual components, independent of the transition direction and other circuit conditions."

There are six components in the Figure X6.9, so the worst case delay through the circuit is,

$$\begin{aligned} t_p &= 20(6) \\ &= 120 \text{ ns} \end{aligned}$$

Thus, the maximum propagation delay from IN to OUT of the circuit using single worst case delay is $\boxed{120 \text{ ns}}$.

Step 4 of 4

On comparison of the results of the calculation of maximum propagation delays using the timing information (105 ns) and using the worst case delay of the circuit (120 ns), calculation of propagation delay using the worst case delay is high. Hence the designer keeps the worst case delay at the time of designing.

Step 1 of 1 **6.12DP**

Refer Figure X6.9 and Table 6.2 in text book.

Propagation delay is the summation of time a gate takes to reflect the change of voltage level (logic level) at the output side for a corresponding change at the input side and the time delay caused by the signal path.

Here 74HCT02's (U1) output will remain at the same constant level (1's) irrespective of the value of IN bit as one of the input is at logic level 0's. The constant output (1's) of U1 gate is feeding to the input of gates U2 which produces the constant output '1' according to the input, as one of the input of the U2 gates are 0's.

Thus, we have seen there is a constant output irrespective of the value of IN bit. It means there is no transition propagation delay.

Thus, the propagation delay of this circuit is **zero**.

Step 1 of 2

6.13DP

Consider the circuit given below:

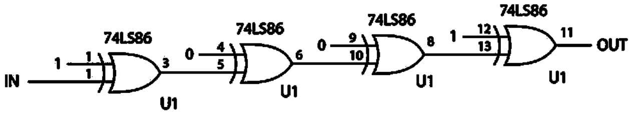


Figure1

Step 2 of 2

The smallest typical delay through one 74LS86 is 10 ns (for HIGH to LOW transition). Use the rule of thumb which states that "minimum delay is equal to 1/4th to 1/3rd of typical delay. For a typical delay of 10 ns estimate the minimum delay to be 3 ns through one gate. Hence, the minimum delay at the output because of all the four gates is 12ns.

The estimation is limited in the fact that it does not take into account the actual transitions of the circuit shown.

For a LOW-to-HIGH input transition, the four gates have typical delays of 13, 10, 10, and 20 ns in their sequence from input to output; hence the total delay comes out to be

$$13+10+10+20=52\text{ ns}$$

And the minimum delay is estimated at one-fourth of this, 13 ns.

For a HIGH-to-LOW input transition, the four gates have typical delays of 20, 12, 12, and 13 ns in their sequence from input to output; hence the total delay comes out to be $20+12+12+13=57\text{ ns}$

And the minimum delay is estimated at approximately one-fourth of this, 14 ns.

Thus, the minimum propagation delay from IN to OUT for the circuit is $\boxed{13\text{ ns}}$.

Step 1 of 3 **6.15DP**

Refer Figure X6.13 and Table 6.2 in text book.

Design:

Using the information in Table 6.2 and Figure X6.13, the values of and for 74LS86 are as follows,

and

Both Low-to-High and High-to-Low transition causes positive transitions and negative transitions on the outputs of each gate. Thus the exact propagation delay from IN to OUT of the circuit is sum of the propagation delays of the first and third gates low to high transition delays and second and fourth gates high to low transition delays or the vice versa summation.

Hence the propagation delay from IN to OUT of the circuit in either case is,

Thus, the propagation delay from IN to OUT of the circuit is .

Step 2 of 3

Single worst case delay specification is the maximum of and specifications. So, single worst case delay for each gate is **23 ns**.

"The worst case delay through a circuit is computed as the sum of the worst case delays through the individual components, independent of the transition direction and other circuit conditions." Since there are four components in Figure X6.13, the worst case delay through the circuit is,

Thus, the maximum propagation delay from IN to OUT of the circuit using single worst case delay is .

Step 3 of 3

On comparison of the results of calculation of maximum propagation delays using the timing information (80 ns) and using the worst case delay of the circuit (92 ns), calculation of propagation delay using the worst case delay is high. Hence the designer keeps the worst case delay at the time of designing.

Step 1 of 3

6.16DP

Propagation delay is the summation of time a gate takes to reflect the change of voltage level (logic level) at the output side for a corresponding change at the input side and the time delay caused by the signal path.

Refer Figure X6.9 and Table 6.2 in text book.

The propagation delay is counted as follows, using table 6-2 for 74HCT86

Table 1

74HCT	Typical	Maximum	Worst			
Part no	t_{pLH} (ns)	t_{pHL} (ns)	t_{pLH} (ns)	t_{pHL} (ns)	t_{pLH} (ns)	t_{pHL} (ns)
'86(2 level)	13	13	40	40	40	40
'86(3 level)	13	13	40	40	40	40

Step 2 of 3

The smallest typical delay through one 74HCT86 is 13 ns (for HIGH to LOW transition). Use the rule of thumb which states that "minimum delay is equal to 1/4th to 1/3rd of typical delay. For a typical delay of 13 ns estimate the minimum delay to be 3.5 ns through one gate.

Hence, the minimum delay at the output because of all the four gates is **14 ns**.

Total propagation delay based on our assumption at the input side is calculated by the propagation delay caused by the each gate in the circuit.

Calculate the minimum propagation delay when the transition is from low to high.

$$\begin{aligned} T_{typical\ \text{LOW to HIGH}} &= U1_{3pHL} + U1_{6pHL} + U1_{8pHL} + U1_{11pLH} \\ &= (13+13+13+13)\ \text{ns} \\ &= 52\ \text{ns} \end{aligned}$$

Thus, the minimum delay estimated is at one-fourth of this, **13 ns**.

Step 3 of 3

Calculate the minimum propagation delay when the transition is from high to low.

$$\begin{aligned} T_{typical\ \text{HIGH to LOW}} &= U1_{3pLH} + U1_{6pLH} + U1_{8pLH} + U1_{11pHL} \\ &= (13+13+13+13)\ \text{ns} \\ &= 52\ \text{ns} \end{aligned}$$

The minimum delay estimated is at one-fourth of this, **13 ns**.

Hence, the estimated minimum propagation delay is 13 ns to 14 ns, which is approximately same as the typical propagation delay.

Step 1 of 5

6.17DP

Refer Figure 6-37 for 5-to-32 decoder circuit and Table 6-3 for propagation delays information.

Propagation delay is the summation of time a gate takes to reflect the change of voltage level (logic level) at the output side for a corresponding change at the input side and the time delay caused by the signal path.

To calculate the delay of 5-to-32 decoder, calculate the delay of 74LS138 from Table 6-3.

Step 2 of 5

The propagation delay of 74LS138 is counted as follows, using table 6-3.

Table 1

74LS	Typical	Maximum	Worst			
Part no	From	To	t_{pLH} (ns)	t_{pHL} (ns)	t_{pLH} (ns)	t_{pHL} (ns)
'138	Any select	Output(2)	20	41	41	41
	Any select	Output(3)	27	39	39	39
	$\overline{\mathbf{G2A}},\overline{\mathbf{G2B}}$	Output	18	32	32	32
	G1	Output	26	38	38	38

Step 3 of 5

The propagation delay of 5-to-32 decoder of Figure 6-37 is twice the propagation delay of 74LS138 as in 5-to-32 decoder the input signal has to pass through two 74LS138 to reach to output side.

Refer to table 6-6,

For any change of level at the input *A,B* or *C* , the corresponding changes are reflected at the two output bits because other 6 bits remain at same level. Out of the eight combinations only one output signal passed through 2 gates otherwise 3 gates.

For any changes at $\overline{\mathbf{G2A}},\overline{\mathbf{G2B}}$ and G1 there are no changes at the output side. We will ignore the presence of $\overline{\mathbf{G2A}},\overline{\mathbf{G2B}}$ and G1

Now from Figure 6-35, it is clear that except one output bit (Y0_L) all others require.

Table 2

Output bit	Number of 2 level gates	Number of 3 level gates
Y0_L	3	0
Y1_L	2	1
Y2_L	2	1
Y3_L	1	2
Y4_L	1	2
Y5_L	1	2
Y6_L	1	2
Y7_L	0	3

Step 4 of 5

Calculate the propagation delays for one of the inputs A (LSB), B and C(MSB) changing while others are at constant value.

Consider that the change at input side is from 100 to 101. It means the change of Y4_L and Y5_L bit values.

Calculate the propagation delay.

$$\begin{aligned} T_{\text{maximumLOWtoHIGH}} &= \mathrm{Y4\_L}_{pLH} + \mathrm{Y5\_L}_{pHL} \\ &= \{(1\times 20)+(2\times 27)+(1\times 41)+(2\times 39)\}\text{ns} \\ &= 193\text{ns} \end{aligned}$$

This delay is for one 74 LS138.

Step 5 of 5

Consider the change at input side of 5-to-32 decoder is from 00100(N4N3N2N1N0) to 00101(N4N3N2N1N0). It means the change of Y4_L and Y5_L bit values in U2,U3,U4 and U5.

There is no change at the output bits of U1.

Thus for this transition change at the input side, calculate the propagation delay of one 74LS138 and then multiplying it by 4 produces the total propagation delay of 5-to-32 decoder.

Calculate the total propagation delay from any input to any output.

$$(193\times 4)\text{ns} = 772\text{ns}$$

By using the above procedure calculation of propagation delays for different set of input combinations is done.

Step 1 of 6 6.18DP

Refer Figure 6-37 for 5-to-32 decoder circuit and Table 6-3 for propagation delays information.

Propagation delay is the summation of time a gate takes to reflect the change of voltage level (logic level) at the output side for a corresponding change at the input side and the time delay caused by the signal path.

To calculate the delay of 5-to-32 decoder, calculate the delay of 74AHCT 138 from table 6-3.

Step 2 of 6

The propagation delay of 74LS138 is counted as follows, using table 6-3.

Table **1**

74AHCT	Typical	Maximum	Worst			
Part no	From	To	t_{pLH} (ns)	t_{pHL} (ns)	t_{pLH} (ns)	t_{pHL} (ns)
'138	Any select	Output(2)	13	13	13	13
	Any select	Output(3)	13	13	13	13
	$\overline{\text{G2A}}, \overline{\text{G2B}}$	Output	12	12	12	12
	G1	Output	11.5	11.5	11.5	11.5

Step 3 of 6

The propagation delay of 5-to-32 decoder of Figure 6-37 is twice the propagation delay of 74AHCT138 as in 5-to-32 decoder the input signal has to pass through two 74AHCT138 to reach to output side.

Refer to table 6-6,

For any change of level at the input A,B or C , the corresponding changes are reflected at the two output bits because other 6 bits remain at same level. Out of the eight combinations only one output signal passed through 2 gates otherwise 3 gates.

For any changes at $\overline{\text{G2A}}, \overline{\text{G2B}}$ and G1 there are no changes at the output side. We will ignore the presence of $\overline{\text{G2A}}, \overline{\text{G2B}}$ and G1.

Now from Figure 6-35, it is clear that except one output bit (Y0_L) all others require.

Table 2

Output bit	Number of 2 level gates	Number of 3 level gates
Y0_L	3	0
Y1_L	2	1
Y2_L	2	1
Y3_L	1	2
Y4_L	1	2
Y5_L	1	2
Y6_L	1	2
Y7_L	0	3

Step 4 of 6

We have to calculate the propagation delays for one of the inputs A(LSB), B and C(MSB) changing while others are at constant value.

Consider that the change at input side is from 100 to 101. It means the change of Y4_L and Y5_L bit values.

Calculate the propagation delay.

$$\begin{aligned} T_{\text{maximumLOWtoHIGH}} &= \text{Y4\_L}_{\text{pLH}} + \text{Y5\_L}_{\text{pHL}} \\ &= \{(1\times13)+(2\times13)+(1\times13)+(2\times13)\}\text{ns} \\ &= 78\text{ns} \end{aligned}$$

This delay is for one 74AHCT138.

Step 5 of 6

Consider the change at input side of 5-to-32 decoder is from 00100(N4N3N2N1N0) to 00101(N4N3N2N1N0). It means the change of Y4_L and Y5_L bit values in U2,U3,U4 and U5.

There is no change at the output bits of U1.

Thus for this transition change at the input side, calculate the propagation delay of one 74LS138 and then multiplying it by 4 will produce the total propagation delay of 5-to-32 decoder.

Calculate the total propagation delay from any input to any output.

$$(78\times4)\text{ns} = 312\text{ns}$$

Step 6 of 6

By using 74AHCT device instead of 74LS, we are getting time improvement of more than half proportion.

In 74AHCT devices time for transition from LOW to HIGH and HIGH to LOW are same or small enough to be ignored.

Step 1 of 2

6.19DP

From the Figure 6-35 in the textbook, write the truth table for 74 x 138 inside logic function is shown in table 1.

Table 1

Control pins			Input pins			Output pins							
6	4	5	3	2	1	7	9	10	11	12	13	14	15
1	X	X	X	X	X	0	0	0	0	0	0	0	0
X	0	X	X	X	X	0	0	0	0	0	0	0	0
X	X	0	X	X	X	0	0	0	0	0	0	0	0
0	1	1	1	1	1	0	0	0	0	0	0	0	1
0	1	1	1	1	0	0	0	0	0	0	0	1	0
0	1	1	1	0	1	0	0	0	0	0	1	0	0
0	1	1	1	0	0	0	0	0	0	1	0	0	0
0	1	1	0	1	1	0	0	0	1	0	0	0	0
0	1	1	0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	1	0	1	0	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0	0	0	0

Step 2 of 2

Here the pins 1, 2, 3 are the inputs A, B, C respectively, pins 6, 4, 5 are the selection inputsG1, G2A_L and G2B_L respectively and pins 7, 9, 10, 11, 12, 13, 14, 15 are the outputs Y7_L, Y6_L, Y5_L, Y4_L, Y3_L, Y2_L, Y1_L, Y0_L respectively.

Step 1 of 12 6.21DP

The functions can be built using one or more 74x138 binary decoders and NAND gates as shown in the figures. The binary decoder generates all minterms of n-variable logic function. The canonical form (sum of minterms) of a logic function is obtained by adding all minterms of that function by matching the order of input bits.

(a)

Consider the following logic function:

$$F=\sum_{X,Y,Z}(2,4,7)$$
$$=m_2+m_4+m_7$$
$$=\overline{X}Y\overline{Z}+X\overline{Y}\overline{Z}+XYZ$$

Step 2 of 12

The expression can be realized as shown in Figure 1.

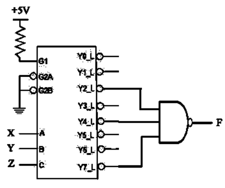


Figure 1

The function F is asserted when the function corresponds to the input combinations 010 or 100 or 111.

Step 3 of 12

(b)

Consider the following logic function:

$$F=\prod_{A,B,C}(3,4,5,6,7)$$
$$=\sum_{A,B,C}(0,1,2)$$
$$=m_0+m_1+m_2$$
$$=\overline{A}\overline{B}\overline{C}+\overline{A}\overline{B}C+\overline{A}B\overline{C}$$

Step 4 of 12

The expression can be realized as shown in Figure 2.

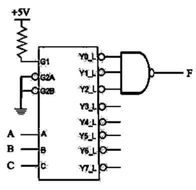


Figure 2

The function F is asserted when the function corresponds to the input combinations 000 or 001 or 010.

Step 5 of 12

(c)

Consider the following logic function:

$$F=\sum_{A,B,C,D}(0,2,10,12)$$
$$=m_0+m_1+m_{10}+m_{12}$$
$$=\overline{A}\overline{B}\overline{C}\overline{D}+\overline{A}\overline{B}C\overline{D}+A\overline{B}C\overline{D}+AB\overline{C}\overline{D}$$

Step 6 of 12

The expression can be realized as shown in Figure 3.

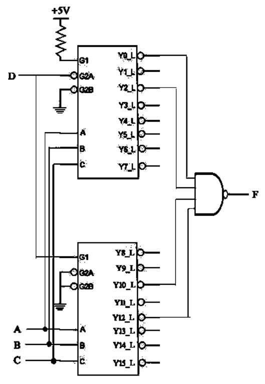


Figure 3

The function F is asserted when the function corresponds to the input combinations 0000 or 0010 or 1010 or 1100. If D is 0 it enables the upper decoder else it enables the lower decoder.

Step 7 of 12

(d)

Consider the following logic function:

$$F=\sum_{W,X,Y,Z}(2,3,4,5,8,10,12,14)$$
$$=m_2+m_3+m_4+m_5+m_8+m_{10}+m_{12}+m_{14}$$
$$=\overline{W}\overline{X}Y\overline{Z}+\overline{W}\overline{X}YZ+\overline{W}X\overline{Y}\overline{Z}+\overline{W}X\overline{Y}Z+W\overline{X}\overline{Y}\overline{Z}+W\overline{X}Y\overline{Z}+WX\overline{Y}\overline{Z}+WXY\overline{Z}$$

The expression can be realized as shown in Figure 4.

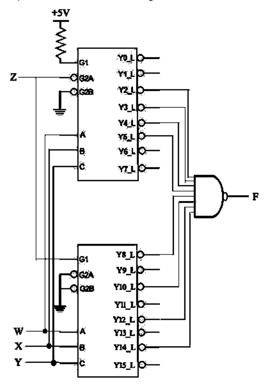


Figure 4

Step 8 of 12

The function F is asserted, when the function corresponds to the input combinations 0010 or 0011 or 0100 or 0101 or 1000 or 1010 or 1100 or 1110. If Z is 0 it enables the upper decoder else it enables the lower decoder.

Step 9 of 12

(e)

Consider the following logic function:

$$F=\sum_{W,X,Y}(0,2,4,5)$$
$$=m_0+m_2+m_4+m_5$$
$$=\overline{W}\overline{X}\overline{Y}+\overline{W}X\overline{Y}+W\overline{X}\overline{Y}+W\overline{X}Y$$
$$G=\sum_{W,X,Y}(1,2,3,6)$$
$$=m_1+m_2+m_3+m_6$$
$$=\overline{W}\overline{X}Y+\overline{W}X\overline{Y}+\overline{W}XY+WX\overline{Y}$$

Step 10 of 12

The expressions can be realized in a single decoder as shown in Figure 5.

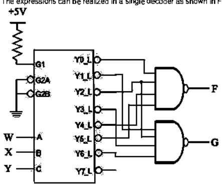


Figure 5

The function F is asserted when the function corresponds to the input combinations 000 or 010 or 100 or 101 and the function G is asserted when the function corresponds to the input combinations 001 or 010 or 011 or 110.

Step 11 of 12

(f)

Consider the following logic function:

$$F=\sum_{A,B,C}(2,6)$$
$$=m_2+m_6$$
$$=\overline{A}B\overline{C}+AB\overline{C}$$
$$G=\sum_{A,B,C}(0,2,3)$$
$$=m_0+m_2+m_3$$
$$=\overline{A}\overline{B}\overline{C}+\overline{A}B\overline{C}+\overline{A}BC$$

Step 12 of 12

The expressions can be realized in a single decoder as shown in Figure 6.

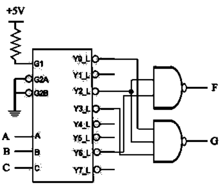


Figure 6

The function F is asserted when the function corresponds to the input combinations 010 or 110 and G is asserted when the function corresponds to the input combinations 000 or 010 or 011

Step 1 of 4

6.22DP

Refer 32-to-1 multiplexer as shown in figure 6-62 in the text book consists three stages. The first stage contains 74 x 138(3-to-8 decoder), the second stages are made of four 74 x 151((8-to-1 multiplexer), and last stage consists of one 4 input NAND gate, 74 x 20.

Consider the worst time delay nothing but selecting the highest time delay among low - to-high and high - to-low for the 74LS components from the Table 6-2 and Table 6-3 in the textbook. Write the scrutinized propagation delay list is as shown in table 1.

Table 1

Part number	From	To	t_{pLH} (ns)	t_{pHL} (ns)
'138	Any select	Dutput(2)	41	41
	Any select	Output(3)	39	39
	$\overline{G2A}, \overline{G2B}$	Output	32	32
	G1	Output	38	38
'151	Any select	$\overline{Y}$	32	32
	Any data	$\overline{Y}$	21	21
	Enable	$\overline{Y}$	30	30
'20			15	15

Step 2 of 4

Now calculate the delay in transmission signal from any input to output by considering each device separately. Assume each of this signals are transmitted one after another not simultaneously as in parallel process because we are trying to find out the worst case delays.

Find the propagation delay for the stage 74 x 138.

Enable delays of the device we will calculate as $(32+32+38)\text{ns} = 102\text{ns}$

From the figure 6-35 in the text book, it is clear that the data transmission delay of 74×138 is by considering the signal transmitted through 3 NAND gates internally as we are trying to find out worst case scenario.

As there are three input lines, the total delays will be $(39\times3)\text{ns} = 87\text{ns}$

Thus total delay of 74 x 138 is $(102+87)\text{ns} = 189\text{ns}$

Step 3 of 4

In second stages the total delay of four 74 x 151 is the summation of enable delay, select line delay and data delay.

This total delay we will calculate using the above table as follows,

$$4\times\{30+(32\times3)+21\}\text{ ns} = 588\text{ns}$$

As only one data line D6 is selected here in each 74 x 151.

Step 4 of 4

In last stage we will calculate the delay of one 4 input NAND gate 74 x 20 as follows,

$$(4\times15)\text{ns} = 60\text{ns}$$

Thus the total delay of 32-to1 MUX is $(189+588+60)\text{ns} = 837\text{ns}$

The worst case delay of 32-to-1 MUX of figure 6-62 is $\boxed{837\text{ ns}}$.

Step 1 of 5

6.23DP

Select the main part of the device as 74HCT which will sort out those problems of non-availability of part '20 and single package device parts. From table 6-2 and 6-3 the concerned information for worst case is shown in table 1.

Table 1

Part number	From	To	t_{pLH} (ns)	t_{pHL} (ns)
'138	Any select	Output(2)	45	45
	Any select	Output(3)	45	45
	$\overline{\mathrm{G2A}}, \overline{\mathrm{G2B}}$	Output	42	42
	G1	Output	42	42
'151	Any select	$\overline{\mathrm{Y}}$	54	54
	Any data	$\overline{\mathrm{Y}}$	45	45
	Enable	$\overline{\mathrm{Y}}$	45	45
'20			35	35

Step 2 of 5

The Figure 6-62 in the textbook contains three stages. The first stage contains 74 x 138(3-to-8 decoder), second stages are made of four 74 x 151(8-to-1 multiplexer), and last stage consists of one 74 x 20(4 input NAND gate).calculate the delay in transmission signal from any input to output by considering each device separately. Assume each of this signals are transmitted one after another not simultaneously as in parallel process because we are trying to find out the worst case delays.

Step 3 of 5

Consider first 74 x 138 device first.

Calculate the Enable delays of the device of the 74 x 138.

$$(42+42+42)\text{ns} = 126\text{ns}$$

The data transmission delay of 74 x 138 we will calculate by considering the signal transmitted through 3 NAND (refer figure 6-35) gates internally as we are trying to find out worst case scenario.

As there are three select lines, the total delays will be $(45\times3)\text{ns} = 135\text{ns}$

Thus total delay of 74 x 138 is $(126+135)\text{ns} = 261\text{ns}$

Step 4 of 5

In second stages the total delay of four 74 x 151 is the summation of enable delay, select line delay and data delay.

This total delay we will calculate using the above table as follows,

$$4\times\{45+(54\times3)+45\}\text{ns} = 252\text{ns}$$

As only one data line D6 is selected here in each 74 x 151.

Step 5 of 5

In last stage we will calculate the delay of one 4 input NAND gate 74 x 20 as follows,

$$(4\times35)\text{ns} = 140\text{ns}$$

Thus the total delay of 32-to1 MUX is $(261+252+140)\text{ns} = 653\text{ns}$

The worst case delay of 32-to-1 MUX of figure 6-62 is $\boxed{653\text{ ns}}$.

Step 1 of 4

6.24DP

Instead of n XNOR gates, use three XNOR gates to construct the 4 input and a single output logic circuit as shown in Figure 1.

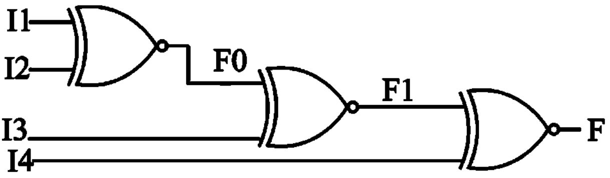


Figure 1

Step 2 of 4

Draw the truth table for the figure 1 by applying all possible combinations (levels) as shown in table 1.

Table 1

I1	I2	F0	I3	F1	I4	F
0	0	1	0	0	0	1
0	0	1	0	0	1	0
0	0	1	1	1	0	0
0	0	1	1	1	1	1
0	1	0	0	1	0	0
0	1	0	0	1	1	1
0	1	0	1	0	0	1
0	1	0	1	0	1	0
1	0	0	0	1	0	0
1	0	0	0	1	1	1
1	0	0	1	0	0	1
1	0	0	1	0	1	0
1	1	1	0	0	0	1
1	1	1	0	0	1	0
1	1	1	1	1	0	0
1	1	1	1	1	1	1

Step 3 of 4

The inputs are indicated with blue color and the output indicated as red color.

Step 4 of 4

From the Table 1, input and output relation it is clear that if the XOR gates shown in Figure 6-70 in the text book replaced by the XNOR produces the $\boxed{\textbf{even parity}}$.

For the $n+1$ inputs to construct the even parity requires n XNOR gates connected as daisy chain connection.

Step 1 of 4

6.25DP

Refer error-correcting circuit for a 7-bit hamming code is as shown in figure 6-73 in the text book consists three stages. The first stage contains of three 74 x 280(9 bit odd/even parity generator), the second stages are made of 74 x 138(3-to-8 decoder) and last stage consists of seven 74 x 86(2 input EX-OR gate).

Select the main part of the device as 74LS from the table 6-2 and 6-3 in the textbook. The concerned information for worst case delay is shown in table 1.

Table 1

Part number	From	To	t_{pLH} (ns)	t_{pHL} (ns)
'138	Any select	Output(2)	41	41
	Any select	Output(3)	39	39
	$\overline{G2A}, \overline{G2B}$	Output	32	32
	G1	Output	38	38
'86(3 levels)			30	30
'280	Any input	ODD	50	50

Step 2 of 4

Now calculate the delay in transmission signal from any input to output by considering each device separately. Assume each of this signals are transmitted one after another not simultaneously as in parallel process because we are trying to find out the worst case delays.

In first stage is summation of delay of three 74 x 280. The delay of one 74 x 280 is summation of four data bits delay and one common data bit delay as other five data bits are shorted to ground.

Find the propagation delay for the stage 74 x 280.

This total delay we will calculate using the above table as follows:

$$\{(4\times 50)+50\}\text{ns} = 250\text{ns}$$

Have three parity generators (74 x 280) leads to:

$$(3\times 250)\text{ns} = 750\text{ns}$$

Step 3 of 4

In second stages the total delay of 74 x 138 is the summation of enable delay, select line delay and data delay.

Enable delays of the device we will calculate as $(32+32+38)\text{ns} = 102\text{ns}$

From the figure 6-35 in the text book, it is clear that the data transmission delay of 74×138 is by considering the signal transmitted through 3 NAND gates internally as we are trying to find out worst case scenario.

As there are three input lines, the total delays will be $(39\times 3)\text{ns} = 87\text{ns}$

Thus total delay of 74 x 138 is $(102+87)\text{ns} = 189\text{ns}$ As here used eight output bits of

74 x 138,

The total delay incurred by this device is for worst case scenario, as follows:

$$102+(87\times 8)\text{ns} = 798\text{ ns}$$

Here enable delays are common to all the bits and once the device is enabled, it is applicable to all the bits. Thus enable delay is counted as one time.

Step 4 of 4

In last stage we will calculate the delay of seven 74 x 86, considering internal three NAND gates operation. In each 74 x 86 device there are two delays caused by DU and enable bits.

We calculate the delay of one 74 x 86 is as follows,

$$(2\times 30)\text{ns} = 60\text{ns}$$

Thus the delay of last stage is as follows,

$$(7\times 60)\text{ns} = 420\text{ns}$$

Thus the total delay of is $(798+750+420)\text{ns} = 1968\text{ ns}$

The total worst case delay of circuit of figure 6-73 is $\boxed{1968\text{ ns}}$.

Step 1 of 1 **6.26DP**

Refer to the Table 6-2 and 6-3 for propagation delay of 74AHCT components. Select the main part of the device as 74AHCT. The information for worst case delay for 74AHCT is shown in Table 1.

Table 1

Part number	From	To	t_{pLH} (ns)	t_{pHL} (ns)
'138	Any select	Output(2)	13	13
	Any select	Output(3)	13	13
	$\overline{G2A}, \overline{G2B}$	Output	12	12
	G1	Output	11.5	11.5
'86(3 levels)			10	10
'280(HCT)	Any input	ODD	56	56

Refer to the Figure 6-73 for error correcting circuit. The Figure 6-73 contains three stages. The first stage contains three 74x280. The second stages are made of one 74x138 and last stage consists of seven 74x86.

Calculate the delay in transmission signal from any input to output by considering each device separately.

We assume each of this signals are transmitted one after another not simultaneously as in parallel process because we are trying to find out the worst case delays.

Consider 74x138 device first that is existed in the second stage. Calculate the Enable delays of the 74x138.

$$(12+12+11.5)\text{ ns} = 35.5\text{ ns}$$

Find out the data transmission delay of 74x138 by considering the signal transmitted through 3 NAND gates internally as we are trying to find out worst case scenario. As there are three select lines $\mathbf{A,B,C}$, the total delays will be $(13\times3)\text{ ns} = 39\text{ ns}$

Thus total delay of one74 x 138 is,

$$(35.5+39)\text{ ns} = 74.5\text{ ns}$$

Here eight output bits of 74 x 138, the total delay incurred by this device is for worst case scenario as,

$$35.5\text{ ns} + (39\times8)\text{ ns} = 347.5\text{ ns}$$

Here enable delays are common to all the bits and once the device is enabled, it is applicable to all the bits. Thus enable delay is counted as one time.

In first stage is summation of delay of three 74x280. The delay of one 74x280 is summation of four data bits delay and one common data bit delay as other five data bits are shorted to ground.

Here HCT device is considered as part number '280 is unavailable in AHCT device.

Calculate the total delay using the Table 1.

$$\{(4\times56)+56\}\text{ ns} = 280\text{ ns}$$

Thus, the total delay of first stage is,

$$(3\times280)\text{ ns} = 840\text{ ns}$$

In last stage, calculate the delay of seven 74x86, considering internal three NAND gates operation. In each 74x86 device there are two delays caused by DU and enable bits.

Calculate the delay of one 74x86.

$$(2\times10)\text{ ns} = 20\text{ ns}$$

Thus, the delay of last stage is,

$$(7\times20)\text{ ns} = 140\text{ ns}$$

Thus, the total delay of is,

$$(347.5+840+140)\text{ ns} = 1327.5\text{ ns}$$

Therefore, the total worst case delay of circuit of Figure 6-73 is $\boxed{1327.5\text{ ns}}$.

Step 1 of 1 **6.27DP**

The logic function of EX-NOR gate is such that when two inputs bits are equal, output bit will be HIGH otherwise LOW.

The truth table of EX-NOR gate is shown in Table 1.

Table 1

X	Y	F
0	0	1
0	1	0
1	0	0
1	1	1

The output expression of EX-NOR gate is,

$$\mathbf{F = X \cdot Y + \overline{X} \cdot \overline{Y}}$$

Refer to the 6-78 for 4-bit74x85 comparator. It has three outputs like the less than output $\mathbf{ALTBOUT}$, greater than output $\mathbf{AGTBOUT}$, and an equal output $\mathbf{AEQBOUT}$. Inputs are $\mathbf{ALTBIN}$, $\mathbf{AGTBIN}$, and $\mathbf{AEQBIN}$.

The numbers $\mathbf{A(A3A2A1A0)\ and\ B(B3B2B1B0)}$ will be treated as equal if each of the corresponding pair of bits is equal separately.

The equality expression of A and B is evaluated as,

$$\begin{aligned}
&\mathbf{x3 = A3 \cdot B3 + \overline{A3} \cdot \overline{B3}} && \mathbf{here\ A3 = B3} \\
&\mathbf{x2 = A2 \cdot B2 + \overline{A2} \cdot \overline{B2}} && \mathbf{here\ A2 = B2} \\
&\mathbf{x1 = A1 \cdot B1 + \overline{A1} \cdot \overline{B1}} && \mathbf{here\ A1 = B1} \\
&\mathbf{x0 = A0 \cdot B0 + \overline{A0} \cdot \overline{B0}} && \mathbf{here\ A0 = B0}
\end{aligned}$$

$$\mathbf{(A = B) = x3 \cdot x2 \cdot x1 \cdot x0}$$

But, the expression for $\mathbf{AEQBOUT}$ is,

$$\begin{aligned}
\mathbf{AEQBOUT} &= \mathbf{(A = B) \cdot AEQBIN} \\
&= \mathbf{x3 \cdot x2 \cdot x1 \cdot x0 \cdot AEQBIN}
\end{aligned}$$

Therefore, the logic expression of AEQBOUT is $\boxed{\mathbf{AEQBOUT} = x3 \cdot x2 \cdot x1 \cdot x0 \cdot \mathbf{AEQBIN}}$.

Step 1 of 2

Refer to the Figure 6-80 in the textbook for logic symbol of 74x682.

The logic function of EX-NOR gate is such that when two inputs bits are equal, output bit will be HIGH otherwise LOW.

Truth table of EX-NOR gate is shown in Table 1.

Table 1

X	Y	F
0	0	1
0	1	0
1	0	0
1	1	1

Step 2 of 2

The output expression of EX-NOR gate is,

$$F = X \cdot Y + \overline{X} \cdot \overline{Y}$$

The numbers $P(P7 \dots P0)$ and $Q(Q7 \dots Q0)$ will be treated as equal if each of the corresponding pair of bits is equal separately.

Thus the equality expression of A and B is evaluated as follows.

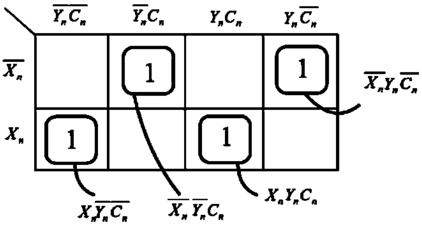
$$x7 = P7 \cdot Q7 + \overline{P7} \cdot \overline{Q7}$$
$$x6 = P6 \cdot Q6 + \overline{P6} \cdot \overline{Q6}$$
$$x5 = P5 \cdot Q5 + \overline{P5} \cdot \overline{Q5}$$
$$x4 = P4 \cdot Q4 + \overline{P4} \cdot \overline{Q4}$$

$$x3 = P3 \cdot Q3 + \overline{P3} \cdot \overline{Q3}$$
$$x2 = P2 \cdot Q2 + \overline{P2} \cdot \overline{Q2}$$
$$x1 = P1 \cdot Q1 + \overline{P1} \cdot \overline{Q1}$$
$$x0 = A0 \cdot B0 + \overline{A0} \cdot \overline{B0}$$

Therefore, the logic expression of PEQQ_L is $\boxed{\overline{x_7 \cdot x_6 \cdot x_5 \cdot x_4 \cdot x_3 \cdot x_2 \cdot x_1 \cdot x_0}}$.

Step 1 of 3

Evaluate the expression of carry output and sum bits as C_{n+1} and S_n respectively. The Karnaugh map for sum is shown in Figure 1.



$\overline{Y_n}\,\overline{C_n}$
$\overline{Y_n}C_n$
Y_nC_n
$Y_n\overline{C_n}$
$\overline{X_n}$
X_n
1
1
1
1
$\overline{X_n}Y_n\overline{C_n}$
$X_n\overline{Y_n}\,\overline{C_n}$
$\overline{X_n}\,\overline{Y_n}C_n$
$X_nY_nC_n$

Figure 1

Step 2 of 3

From Figure 1, the Boolean expression for sum S_n is,

$$S_n = X_n\overline{Y_n}\,\overline{C_n} + \overline{X_n}\,\overline{Y_n}C_n + X_nY_nC_n + \overline{X_n}Y_n\overline{C_n}$$

Step 3 of 3

Karnaugh map for carry is shown in Figure 2.

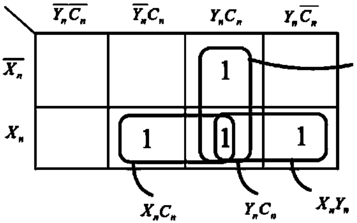


Figure 2

From Figure 2, the Boolean expression for carry C_{n+1} is,

$$C_{n+1} = X_nC_n + Y_nC_n + X_nY_n$$

The above expression is for n^{th} bit. The full addition operation will be started between X_0 and Y_0 and the carry output will be treated as the carry input for X_1 and Y_1. This operation will be carried over to the next subsequent bits as well.

Therefore, the final expression for S_3 is $\boxed{X_3\overline{Y_3}\,\overline{C_3} + \overline{X_3}\,\overline{Y_3}C_3 + X_3Y_3C_3 + \overline{X_3}Y_3\overline{C_3}}$.

Step 1 of 8

In a SDRAM module, as per the context the LOW signal is said to be in the range of 0.0-0.7V and HIGH signal to be in the range of 1.7 – 2.5V. In a positive logic LOW is consider as a 0 and HIGH is consider as 1.

Using positive-logic convention, the below signal levels are indicated as 0 (low) or 1(high).

(a) Given signal level, **0.0 V**

It is in the range of $\mathbf{0.0-0.7\ V}$, which is defined as a **LOW** signal.

Therefore, from the definition of positive logic convention the logic value of $\mathbf{0.0\ V}$ is $\boxed{0}$.

Step 2 of 8

(b) Given signal level, **0.7 V**

It is in the range of $\mathbf{0.0-0.7\ V}$, which is defined as a **LOW** signal.

Therefore, from the definition of positive logic convention the logic value of $\mathbf{0.7\ V}$ is $\boxed{0}$.

Step 3 of 8

(c) Given signal level, **1.7 V**

It is in the range of $\mathbf{1.7-2.5\ V}$, which is defined as a **HIGH** signal.

Therefore, from the definition of positive logic convention the logic value of $\mathbf{1.7\ V}$ is $\boxed{1}$.

Step 4 of 8

(d) Given signal level, $\mathbf{-0.6\ V}$.

It is undefined signal level. That is a circuit may interpret them as either 0 or 1.

Therefore, the logic value of $\mathbf{-0.6\ V}$ is $\boxed{\text{either 0 or 1}}$.

Step 5 of 8

(e) Given signal level, $\mathbf{1.6\ V}$.

It is in the intermediate range $\mathbf{0.7-1.7\ V}$ are not expected to occur except during signal transitions, and yield undefined logic values.

Therefore, the circuit may interpret them as either 0 or 1.

Step 6 of 8

(f) Given signal level, $\mathbf{-2.0\ V}$.

It is undefined signal level. That is a circuit may interpret them as either 0 or 1.

Therefore, the logic value of $\mathbf{-2.0\ V}$ is $\boxed{\text{either 0 or 1}}$.

Step 7 of 8

(g) Given signal level, **2.5 V**

It is in the range of $\mathbf{1.7-2.5\ V}$, which is defined as a **HIGH** signal.

Therefore, from the definition of positive logic convention the logic value of $\mathbf{2.5\ V}$ is $\boxed{1}$.

Step 8 of 8

(h) Given signal level, **3.3 V**

It is undefined signal level. That is a circuit may interpret them as either 0 or 1.

Therefore, the logic value of $\mathbf{3.3\ V}$ is $\boxed{\text{either 0 or 1}}$.

Step 1 of 2

Refer to the Table 6-3 for propagation delay of 74LS components. Select the main part of the device as 74LS. The information for worst case delay for 74LS is shown in Table 1.

Table 1

Part number	From	To	t_{pLH} (ns)	t_{pHL} (ns)
'283	C0	Any Si	24	24
	Any Ai , Bi	Any Si	24	24
	C0	C4	17	22
	Any Ai , Bi	C4	17	17

Refer to the Figure 6-89 for 16-bit group ripple adder.

The 16-bit group-ripple adder contains four 74x283 ICs as U1, U2, and U3 and U4. The C4 bit signal is serially passing from U1 to U2, from U2 to U3 and finally from U3 to U4. It means the delay of U1 affects U2, delay of U2 affects U3 and delay of U3 affects U4.

Calculate the total delay of one 74x283. Here, it is observed that $16 = 2^4$ hence $n = 4$. To evaluate the total delay of 16-bit group-ripple adder, first multiply propagation delay with four. The delay of one 74x283 is calculated in Table 2.

Step 2 of 2

The propagation delay from C0 to S0, S1, S2, and S3 is calculated as,

$$t_{pLH}(\text{ns}) = (24)(4) \\ = 96$$

$$t_{pHL}(\text{ns}) = (24)(4) \\ = 96$$

Calculate the propagation delay from four pair of Ai, Bi to four Si.

$$t_{pLH}(\text{ns}) = (24)(4) \\ = 96$$

$$t_{pHL}(\text{ns}) = (24)(4) \\ = 96$$

Calculate the propagation delay from four pair of Ai, Bi to C4.

$$t_{pLH}(\text{ns}) = (17)(4) \\ = 68$$

$$t_{pHL}(\text{ns}) = (17)(4) \\ = 68$$

Therefore, the total propagation delay for one 74x283 is,

$$t_{pd} = 96 + 96 + 22 + 68 \\ = 282 \text{ ns}$$

Here, in the 16-bit group ripple adder, there are four 74x283 ICs are used. Therefore, the total propagation delay is,

$$t_{pd} = (282 \text{ ns})(4) \\ = 1128 \text{ ns}$$

Therefore, the maximum propagation delay is $\boxed{1128 \text{ ns}}$.

Step 1 of 8 6.032E

A possible definition of a BUT gate is **"Y1 is 1 if A1 and B1 are 1 but either A2 or B2 is 0; Y2 is defined symmetrically."**

Find a gate-level design for the BUT gate defined, that uses a minimum number of transistors when realized in CMOS. You may use inverting gates with up to 4 inputs, AOI or OAI gates, transmission gates, or other transistor level tricks. Write the output expressions (which need not be two-level sums of products), and draw the logic diagram.

Step 2 of 8

BUT gate definition: "Y1 is 1 if A1 and B1 are 1 but either A2 or B2 is 0; Y2 is defined symmetrically."

Draw the following logical diagram of BUT gate.

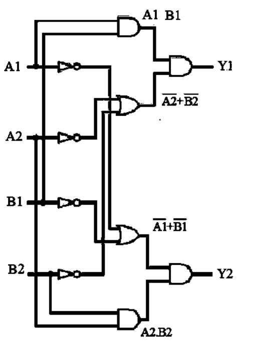


Figure 1: BUT gate logical diagram

Step 3 of 8

Write the expressions for the outputs of BUT gate.

$$Y1=(A1.B1)(\overline{A2}+\overline{B2})$$
$$Y2=(A2.B2)(\overline{A1}+\overline{B1})$$

Write the expression $Y1$ in suitable form to implement in CMOS structure.

$$\begin{aligned}Y1&=(A1.B1)(\overline{A2}+\overline{B2})\\&=\overline{\overline{(A1.B1)(\overline{A2}+\overline{B2})}}\\&=\overline{\overline{(A1.B1)}+\overline{(\overline{A2}+\overline{B2})}}\\&=\overline{\overline{(A1.B1)}+(\overline{\overline{A2}}.\overline{\overline{B2}})}\end{aligned}$$

Step 4 of 8

Simplify $Y1$ further.

$$\begin{aligned}Y1&=\overline{\overline{(A1.B1)}+(\overline{\overline{A2}}.\overline{\overline{B2}})}\\&=\overline{\overline{(A1.B1)}+(A2.B2)}\end{aligned}$$

Draw the following CMOS circuit to implement Y1.

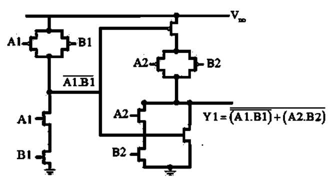


Figure 2: CMOS gate-level design circuit for Y1.

Step 5 of 8

Draw the following **Logic Diagram** for $Y1=A1.B1(\overline{A2}+\overline{B2})$.

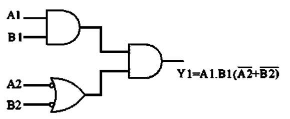


Figure 3: **Logic Diagram** for $Y1=A1.B1(\overline{A2}+\overline{B2})$.

Write the expression $Y2$ in suitable form to implement in CMOS structure.

$$\begin{aligned}Y2&=(A2.B2)(\overline{A1}+\overline{B1})\\&=\overline{\overline{(A2.B2)(\overline{A1}+\overline{B1})}}\\&=\overline{\overline{(A2.B2)}+\overline{(\overline{A1}+\overline{B1})}}\\&=\overline{\overline{(A2.B2)}+(\overline{\overline{A1}}.\overline{\overline{B1}})}\end{aligned}$$

Step 6 of 8

Simplify $Y2$ further.

$$\begin{aligned}Y2&=\overline{\overline{(A2.B2)}+(\overline{\overline{A1}}.\overline{\overline{B1}})}\\&=\overline{\overline{(A2.B2)}+(A1.B1)}\end{aligned}$$

Draw the following CMOS circuit to implement Y2.

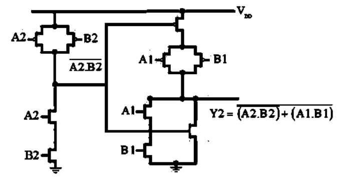


Figure 4: CMOS gate-level design for Y2.

Step 7 of 8

Draw the following **Logic Diagram** for $Y2=A2.B2(\overline{A1}+\overline{B1})$.

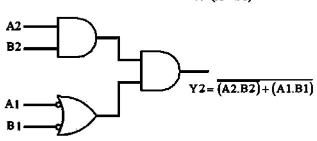


Figure 5: **Logic diagram** for Y2. **Logic Diagram** for $Y2=A2.B2(\overline{A1}+\overline{B1})$.

Step 8 of 8

Thus, the gate-level design for the BUT gate using CMOS are shown in $\boxed{\text{Figure 2, Figure 4}}$.

The output expressions are $\boxed{\begin{aligned}Y1&=(A1.B1)(\overline{A2}+\overline{B2})\\Y2&=(A2.B2)(\overline{A1}+\overline{B1})\end{aligned}}$

The logic diagrams are shown in $\boxed{\text{Figure 3, Figure 5}}$.

Step 1 of 8

A possible definition of a BUT gate is "Y1 is 1 if A1 and B1 are 1 but either A2 or B2 is 0; Y2 is defined symmetrically."

Find a gate-level design for the BUT gate defined, that uses a minimum number of transistors when realized in CMOS. You may use inverting gates with up to 4 inputs, AOI or OAI gates, transmission gates, or other transistor level tricks. Write the output expressions (which need not be two-level sums of products), and draw the logic diagram.

Step 2 of 8

BUT gate definition: "Y1 is 1 if A1 and B1 are 1 but either A2 or B2 is 0; Y2 is defined symmetrically."

Draw the following logical diagram of BUT gate.

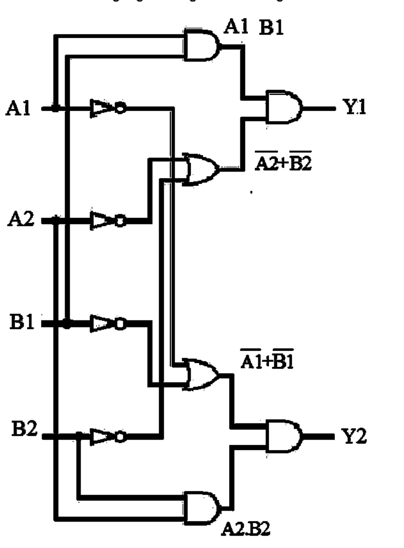


Figure 1: BUT gate logical diagram

Step 3 of 8

Write the expressions for the outputs of BUT gate.

$$Y1=(A1.B1)(\overline{A2}+\overline{B2})$$
$$Y2=(A2.B2)(\overline{A1}+\overline{B1})$$

Write the expression $Y1$ in suitable form to implement in CMOS structure.

$$\begin{aligned}Y1&=(A1.B1)(\overline{A2}+\overline{B2})\\&=\overline{\overline{(A1.B1)(\overline{A2}+\overline{B2})}}\\&=\overline{\overline{(A1.B1)}+\overline{(\overline{A2}+\overline{B2})}}\\&=\overline{\overline{(A1.B1)}+(\overline{\overline{A2}.\overline{B2}})}\end{aligned}$$

Step 4 of 8

Simplify $Y1$ further.

$$\begin{aligned}Y1&=\overline{\overline{(A1.B1)}+(\overline{\overline{A2}.\overline{B2}})}\\&=\overline{\overline{(A1.B1)}+(A2.B2)}\end{aligned}$$

Draw the following CMOS circuit to implement Y1.

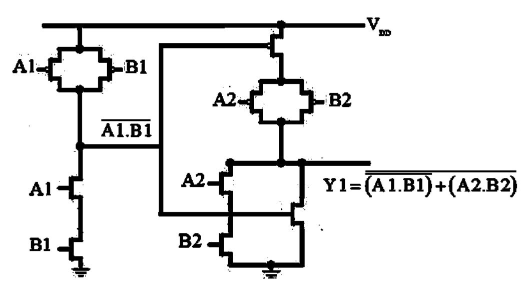


Figure 2: CMOS gate-level design circuit for Y1.

Step 5 of 8

Draw the following **Logic Diagram** for $Y1=A1.B1(\overline{A2}+\overline{B2})$.

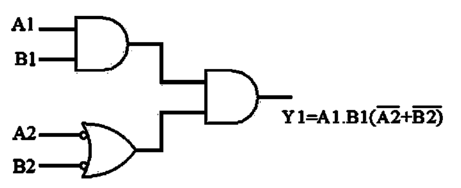


Figure 3: **Logic Diagram** for $Y1=A1.B1(\overline{A2}+\overline{B2})$.

Write the expression $Y2$ in suitable form to implement in CMOS structure.

$$\begin{aligned}Y2&=(A2.B2)(\overline{A1}+\overline{B1})\\&=\overline{\overline{(A2.B2)(\overline{A1}+\overline{B1})}}\\&=\overline{\overline{(A2.B2)}+\overline{(\overline{A1}+\overline{B1})}}\\&=\overline{\overline{(A2.B2)}+(\overline{\overline{A1}.\overline{B1}})}\end{aligned}$$

Step 6 of 8

Simplify $Y2$ further.

$$\begin{aligned}Y2&=\overline{\overline{(A2.B2)}+(\overline{\overline{A1}.\overline{B1}})}\\&=\overline{\overline{(A2.B2)}+(A1.B1)}\end{aligned}$$

Draw the following CMOS circuit to implement Y2.

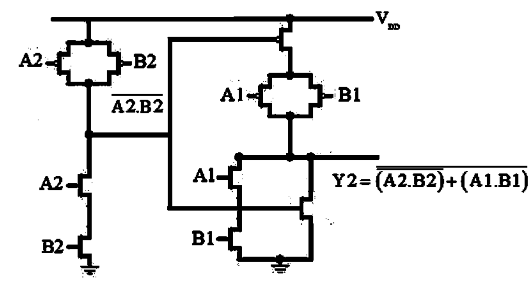


Figure 4: CMOS gate-level design for Y2.

Step 7 of 8

Draw the following **Logic Diagram** for $Y2=A2.B2(\overline{A1}+\overline{B1})$.

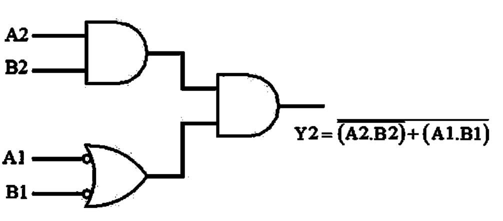


Figure 5: **Logic diagram** for Y2. **Logic Diagram** for $Y2=A2.B2(\overline{A1}+\overline{B1})$.

Step 8 of 8

Thus, the gate-level design for the BUT gate using CMOS are shown in $\boxed{\text{Figure 2, Figure 4}}$.

The output expressions are $\boxed{\begin{aligned}Y1=(A1.B1)(\overline{A2}+\overline{B2})\\Y2=(A2.B2)(\overline{A1}+\overline{B1})\end{aligned}}$.

The logic diagrams are shown in $\boxed{\text{Figure 3, Figure 5}}$.

Step 1 of 3

Write a behavioural-style VHDL or Verilog program for the BUT gate defined in Exercise 6.31

Step 2 of 3

BUT gate definition as in Figure 3.1 from Exercise 6.31

"Y1 is 1 if A1 and B1 are 1 but either A2 or B2 is 0; Y2 is defined symmetrically."

BUT gate

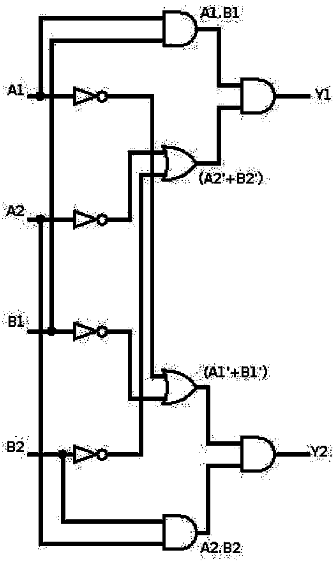


Figure 3.4

Step 3 of 3

Behavioural-style Verilog Program for the above Figure 3.4

```
module gate ( A1, B1, A2, B2, Y1, Y2 );

input A1,B1,A2,B2;

output Y1,Y2;

reg Y1,Y2;

always @( A1,B1,A2,B2)

begin

#10 Y1=A1 & B1 ( ~A2 ^ ~B2 );

#10 Y2=A2 & B2 ( ~A1 ^ ~B1 );

end

end module
```

Step 1 of 3

6.035E

Write a structural VHDL or Verilog program that instantiates a single 2-input OR gate and the BUT gate component of Exercise 6.34 to realize the logic function in Exercise 6.33. Write a test bench that checks your circuit's output for all 16 possible input combinations and displays a message if there's an error.

Step 2 of 3

A single 2-input OR gate and the BUT gate component of Exercise 6.34 to realize the logic function in Exercise 6.33. (Refer to Figure 3.31 in the Exercise 6.33)

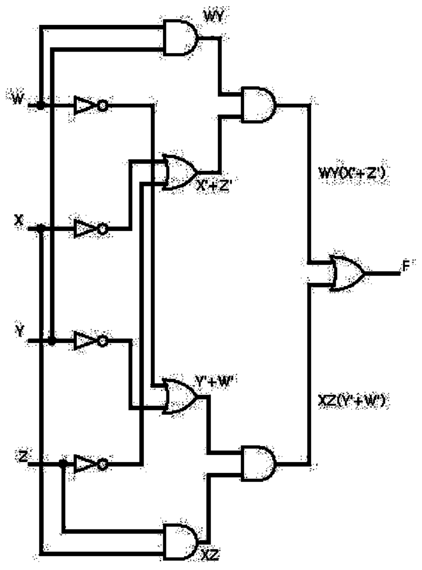


Figure 3.4

Step 3 of 3

Note: In the above circuit t1= WY, t2=X'+Z', t3=WY(X'+Z')

t4=XZ, t5=Y'+W',t6=XZ(Y'+W')

Verilog Program (Structural) of the for the above Figure 3.4,

```
module component (W,X,Y,Z,F);

input W,X,Y,Z

output F;

wire t1,t2,t3,t4,t5,t6,F;

and #10 (t3,t1,t2);

and #10 (t6,t4,t5);

or #10 (F,t3,t6);

end module
```

Verilog test bench for the above Figure 3.4,

```
module main;

reg W,X,Y,Z;

Wire F;

block gate(W,X,Y,Z,F);

begin

$(display="Output is Incorrect")

end

initial

begin

#5 W=0; X=0; Y=0; Z=0; // for all 16 possible combinations

#5 W=0; X=0; Y=0; Z=1;

#5 W=0; X=0; Y=1; Z=0;

#5 W=0; X=0; Y=1; Z=1;

#5 W=0; X=1; Y=0; Z=0;

#5 W=0; X=1; Y=0; Z=1;

#5 W=0; X=1; Y=1; Z=0;

#5 W=0; X=1; Y=1; Z=1;

#5 W=1; X=0; Y=0; Z=0;

#5 W=1; X=0; Y=0; Z=1;

#5 W=1; X=0; Y=1; Z=0;

#5 W=1; X=0; Y=1; Z=1;

#5 W=1; X=1; Y=0; Z=0;

#5 W=1; X=1; Y=0; Z=1;

#5 W=1; X=1; Y=1; Z=0;

#5 W=1; X=1; Y=1; Z=1;

end

end module
```

Step 1 of 4 **6.036E**

Write a VHDL program for a generic 3-to-8 binary decoder with active-high inputs and outputs, based on Table 6-17 but with only a single enable input. Then write a second VHDL program, based on Table 6-16, that instantiates the first module to emulate a74x138, including multiple enable inputs. Draw a block diagram similar to Figure 6-42 that shows the relationship between the modules. Synthesize both Table 6-16 and the second module for a CPLD of your choice, and compare the synthesized results. Explain any differences.

Step 2 of 4

VHDL program for a generic 3-to-8 binary decoder, active-high inputs and outputs, based on Table 6-17 (Refer chapter 6 in the textbook) but with only a single enable input.

VHDL program;

```
library IEEE;
use IEEE.std_logic_1164.all;
entity decoder is
port(G1.G2,G3: in STD_LOGIC;
A : in std_logic_vector(2 downto 0);
Y : out std_logic_vector(0 to 7));
end decoder;
architecture behavioral of decoder is
signal Y_S : std_logic_vector(0 to 7);
with A select Y_s <=
"10000000" when "000",
"01000000" when "001",
"00100000" when "010",
"00010000" when "011",
"00001000" when "100",
"00000100" when "101",
"00000010" when "110",
"00000001" when "111",
"00000000" when others;
--with only a single enable input.
Y <= Y_s when ((G1='1') & (G2 and G3='0') )else "00000000";
end behavioral;
```

Step 3 of 4

VHDL program for a generic 3-to-8 binary decoder, based on Table 6-16 (Refer chapter 6 in the textbook), that instantiates the first module to emulate a 74x138, including multiple enable inputs.

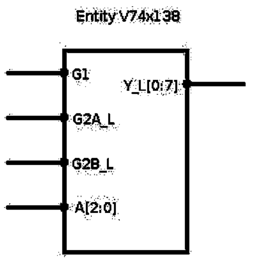
VHDL program;

```
library IEEE;
use IEEE.std_logic_1164.all;
entity decoder is
port(G1,G2A_L,G2B_L: in STD_LOGIC;
A : in std_logic_vector(2 downto 0);
Y : out std_logic_vector(0 to 7));
end decoder;
architecture behavioral of decoder is
signal G2A,G2B: std_logic;
signal Y: std_logic_vector(0 to 7);
signal Y_S : std_logic_vector(0 to 7);
begin
G2A < = not G2A_L;
G2B <= not G2B_L;
Y_L <= not Y;
with A select Y_s <= "10000000" when "000",
"01000000" when "001",
"00100000" when "010",
"00010000" when "011",
"00001000" when "100",
"00000100" when "101",
"00000010" when "110",
"00000001" when "111",
"00000000" when others;
--with multiple enable inputs
Y <= Y_s when (G1 and G2A and G2B) = '1' else "00000000";
end behavioral;
```

Step 4 of 4

Block Diagram

The below block diagram shows the relationship between the modules.



Comparison of the synthesized results is as follows;

With the first module the cascading and the parallel expansion will be less because of only one enable input

With the second module the cascading and the parallel expansion will be more because of multiple enable input

Step 1 of 2

6.037E

Repeat Exercise 6.36 using Verilog and Tables 6-24 and 6-23 and Figure 6-43.

Step 2 of 2

For a generic 3-to-8 binary decoder, active-high inputs and outputs, based on Table 6.24 Refer chapter 6 in the textbook) but with only a single enable input.

Verilog Program;

```
module decoder(G1,G2A_L,G2B_L,A,Y_L);
input G1,G2A_L,G2B_L;
input [2:0] A;
output [0:7]Y;
reg G2A,G2B;
reg [0:7] Y;
assign G2A= ~G2A_L;
assign G2B=~G2B_L;
assign Y_L= ~ Y
// with only a single enable input.
always @(G1 or G2A_L or G2B_L or A or Y) begin
if (G1 & G2A & G2B)
case(A)
0:Y=8'b10000000;
1:Y=8'b01000000;
2:Y=8'b00100000;
3:Y=8'b00010000;
4:Y=8'b00001000;
5:Y=8'b00000100;
6:Y=8'b00000010;
7:Y=8'b00000001;
default :Y=8'b00000000;
endcase
else Y=8'b00000000;
end
endmodule
```

For a generic 3-to-8 binary decoder, based on Table 6-23, that instantiates the first module to emulate a74x138, including multiple enable inputs.

Verilog program;

```
module dec74138 (G1,G2A_L,G2B_L,A,Y_L);
input G1,G2A_L,G2B_L;
input [2:0] A;
output [0:7]Y_L;
wire G2A,G2B;
wire [0:7] Y;
assign G2A= G2A_L;
assign G2B=G2B_L;
assign Y_L= Y
decoder U1(G1,G2A,G2B,A,Y);
endmodule
module decoder(G1,G2,G3,A,Y);
input G1,G2,G3;
input [2:0] A;
output [0:7]Y;
reg [0:7] Y;
// with only a multiple enable input.
always @(G1 or G2 or G3 or A) begin
if (G1 & G2 & G3)
case(A)
0:Y=8'b10000000;
1:Y=8'b01000000;
2:Y=8'b00100000;
3:Y=8'b00010000;
4:Y=8'b00001000;
5:Y=8'b00000100;
6:Y=8'b00000010;
7:Y=8'b00000001;
default :Y=8'b00000000;
endcase
else Y=8'b00000000;
end
endmodule
```

Block Diagram

The below Figure 3.7 shows the relationship between two modules.

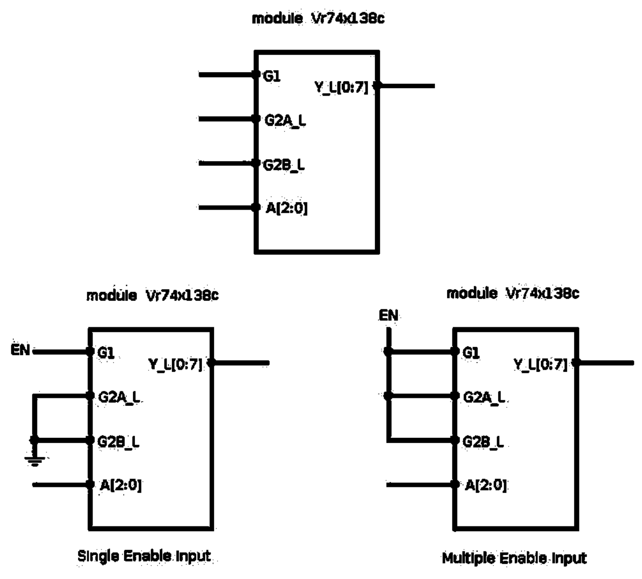


Figure 3.7

Step 1 of 2

4-to-16 decoder with six outputs removed is a 4-to-10 decoder without enable input. It is designed by the following logic equations.

$Y0=\overline{ABCD}$ $Y4=A\overline{B}C\overline{D}$

$Y1=A\overline{BCD}$ $Y5=\overline{A}BC\overline{D}$

$Y2=\overline{A}B\overline{CD}$ $Y6=ABC\overline{D}$

$Y3=AB\overline{CD}$ $Y7=\overline{ABC}D$

$Y8=A\overline{BC}D$

Step 2 of 2

Here Y0-8 are the outputs.
The gate level diagram with A, B, C and D as inputs and Y-0 to 9 as outputs is shown in Figure 1.

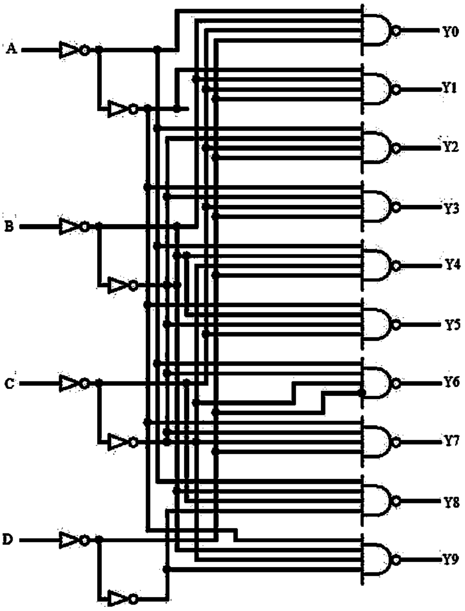


Figure 1
The cost of such a decoder be minimized compared to 4-to-16 decoder because this is simply a 4-to-16 decoder with six outputs removed. As the number of pins is reduced, the circuit design is reduced, and thus the cost of the circuit is reduced.

Step 1 of 2

6.039E

The gate level diagram with A, B, C and D as inputs and Y-0 to 9 as outputs is shown in Figure 1.

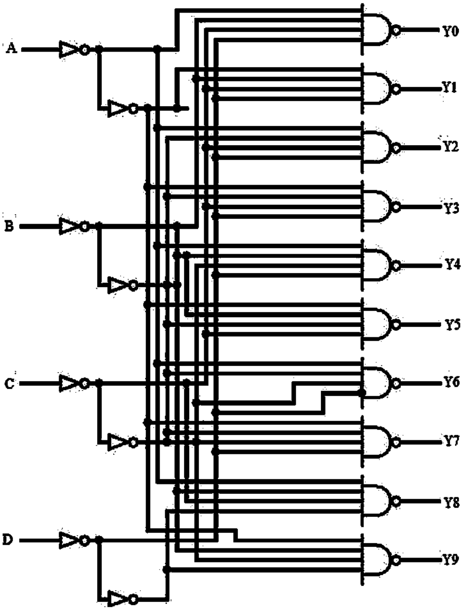


Figure 1

Step 2 of 2

From Figure 1, we observe that **three** four-variable Kamaugh maps are required to complete the multiple-output minimization procedure.

Step 1 of 1

6.040E

A system requires a 5-to-32 binary decoder with a single enable active-low input. When the EN1 input pulled HIGH, either the EN2-L or the EN3-L inputs could be used as enable, with the other input grounded.

The following are the pros and cons of using EN2-L versus EN3_L:

• It simplifies cascading and/or data reception.

• This multiple enable function allows easy parallel expansion of the device

• It is possible to enable one of the decoder out of many decoders.

• In some cases, to combine decoders to make large one, without any additional components, especially if the decoders have the large enable inputs.

Step 1 of 3

The function of the decoder is shown in Table 1.

CS_L	A2	A1	A0	*Output to Assert*
1	x	x	x	none
0	0	0	x	BILL_L
0	0	x	0	MART_L
0	0	1	x	JOAN_L
0	0	x	1	PAUL_L
0	1	0	x	ANNA_L
0	1	x	0	FRED_L
0	1	1	x	DAVE_L
0	1	x	1	KATE_L

Table 1

Step 2 of 3

ABEL Equations:

Decoders can be customized as in Table x6.41 using ABEL, a single GAL16V8 and single 74x138 MSI decoder

module decoder

“Input pins

!CS,!RD,A0,A1,A2

“Output pins

!BILL,!MARRY,!JOAN,!PAUL

!ANNA,!FRED,!DAVE,!KATE

Equations

BILL =CS & RD & (!A2 & !A1 & !A0);

BILL =CS & RD & (!A2 & !A1 & A0);

MARRY=CS & RD & (!A2 & !A1 & !A0);

MARRY=CS & RD & (!A2 & A1 & !A0);

JOAN=CS & RD & (!A2 & A1 & !A0);

JOAN=CS & RD & (!A2 & A1 & A0);

PAUL=CS & RD & (!A2 & !A1 & A0);

PAUL=CS & RD & (!A2 & A1 & A0);

ANNA=CS & RD & (A2 & !A1 & !A0);

ANNA=CS & RD & (A2 & !A1 & A0);

FRED=CS & RD & (A2 & !A1 & !A0);

FRED=CS & RD & (A2 & A1 & !A0);

DAVE=CS & RD & (A2 & A1 & !A0);

DAVE=CS & RD & (A2 & A1 & A0);

KATE =CS & RD & (A2 & !A1 & A0);

KATE =CS & RD & (A2 & A1 & A0);

end decoder

Step 3 of 3

Customised Decoder Circuit:

Depending upon the pins CS_L, RD_L and A0, A1, A2 the outputs are enabled according to Table X6.41 (or by above ABEL equations) and this inputs can be applied to Figure 1.

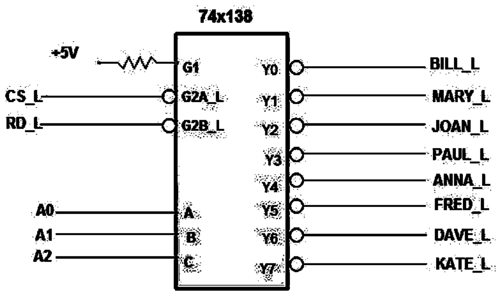


Figure 1

Step 1 of 4

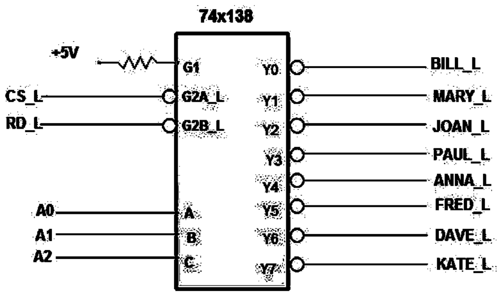
The function of the decoder is shown in Table 1.

CS_L	A2	A1	A0	*Output to Assert*
1	x	x	x	none
0	0	0	x	BILL_L
0	0	x	0	MART_L
0	0	1	x	JOAN_L
0	0	x	1	PAUL_L
0	1	0	x	ANNA_L
0	1	x	0	FRED_L
0	1	1	x	DAVE_L
0	1	x	1	KATE_L

Table 1

Step 2 of 4

The customized decoder circuit is shown in Figure 2.



Step 3 of 4

Figure 2

Step 4 of 4

Decoders can be customized as in Table x6.41 using Verilog and a single GAL16V8 and single 74x138 MSI decoder as shown in Figure 4.2

The following is the verilog program:

```
module design(G1,G2A_L,G2B_L,A,Y_L);

input G1,G2A_L,G2B_L;

input [2:0] A;

output [0:7]Y_L;

reg G2A,G2B;

reg [0:7] Y_L,Y;

always @(G1 or G2A_L or G2B_L or A or Y) begin

G2A=~G2A_L;

G2B=~G2B_L;

Y_L=~Y;

if (G1 & G2A & G2B)

case(A)

BILL_L:Y=3'b00X;

MARRY_L:Y=3'b0X0;

JOAN_L:Y=3'b01X;

PAUL_L:Y=3'b0X1;

ANNA_L:Y=3'b10X;

FRED_L:Y=3'b1X0;

DAVE_L:Y=3'b11X;

KATE_L:Y=3'b1X1;

default :Y=3'bXXX;

endcase

else Y=3'bXXX;

end

endmodule
```

Step 1 of 2

6.043E

Show how to build all four of the following functions using one SSI package (four 2-input gates) and one 74x138.

$$F1 = \overline{X.Y.Z} + X.Y.Z \qquad F2 = \overline{X.Y}Z + X.Y.\overline{Z}$$

$$F3 = \overline{X}.Y.\overline{Z} + X.\overline{Y}Z \qquad F4 = X.\overline{Y.Z} + \overline{X}.Y.Z$$

Step 2 of 2

The following is the design of the circuit:

The below circuit can be designed by applying the following functions that is equations (4) using one SSI package (four 2-input gates) and one 74x138.

Here in the following design single 7400 IC package and single 74x138 is used in which A,B and C are active high inputs and F1 , F2 , F3 and F4 are active low outputs.

$$\begin{aligned} F1 &= \overline{X.Y.Z} + X.Y.Z \qquad F2 = \overline{X.Y}Z + X.Y.\overline{Z} \\ F3 &= \overline{X}.Y.\overline{Z} + X.\overline{Y}Z \qquad F4 = X.\overline{Y.Z} + \overline{X}.Y.Z \end{aligned} \quad \text{...... (4)}$$

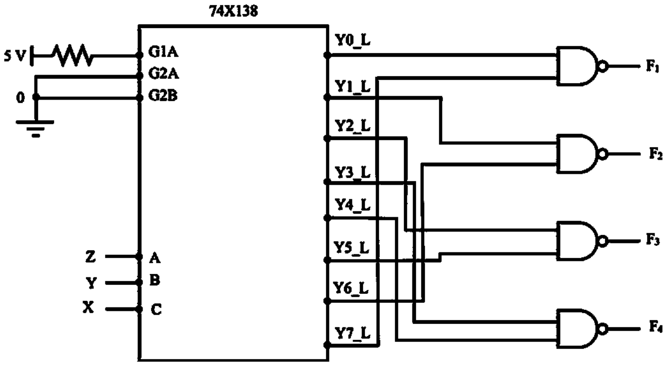


Figure 1

Step 1 of 5

6.044E

(a)

The inputs and outputs of the circuit are shown in Table 1:

Table 1

Input (from top)	Output (from top)
Z	$\overline{XYZEN4}$
Y	$\overline{\overline{X}YZEN4}$
X	$\overline{X\overline{Y}ZEN4}$
EN1	$\overline{\overline{X}\overline{Y}ZEN4}$
EN2	$\overline{XY\overline{Z}EN4}$
EN3	$\overline{\overline{X}Y\overline{Z}EN4}$
EN4	$\overline{X\overline{Y}\overline{Z}EN4}$
	$\overline{\overline{X}\overline{Y}\overline{Z}EN4}$
	$\overline{EN1EN2EN3}$

Step 2 of 5

(b)

The circuit is a 3 to 8 decoder. In Table 1, the top eight min-terms are decoder's output terms. The last one is enable signal term. If this enable term is HIGH, the decoder circuit will be in disabled state.

Step 3 of 5

(c)

The logic symbol is shown in Figure 1:

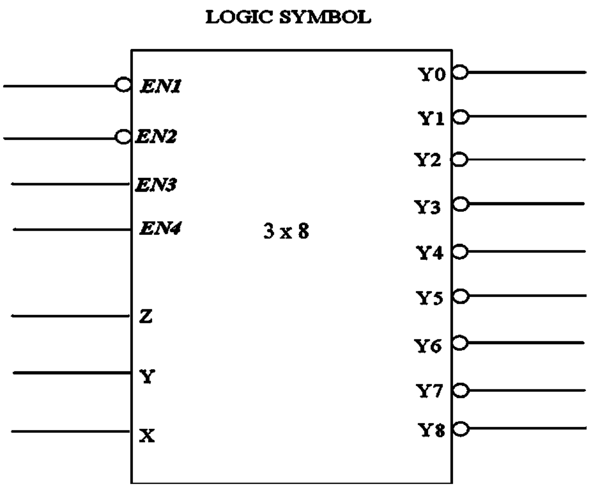


Figure 1

Step 4 of 5

(d)

Write the VHDL program as follows:

```
Library IEEE;
use IEEE.std_logic_1164.all ;
entity 3x8 is
port (EN1,EN2,EN3,EN4: in STD_LOGIC; -- enable inputs
X,Y,Z : in STD_LOGIC ; – select inputs
Y : out STD_LOGIC_VECTOR (0 to 7) ) ; – decoded outputs
end 3x8 ;
architecture 3x8_a of 3x8 is
signal Y_i : STD_LOGIC_VECTOR( 0 to 7 ) ;
begin
with XYZEN4 select Y_i <=
“01111111” when “1111” ,
“01111111” when “1111” ,
“01111111” when “1111” ,
“01111111” when “1111” ,
“01111111” when “1111” ,
“01111111” when “1111” ,
“01111111” when “1111” ,
“01111111” when “1111” ,
“11111111” when others ;
Y <= Y_i when ( not( EN1 and EN2 and EN3 and EN4)) =’0’ else ”11111111” ;
end 3x8_a ;
```

Step 5 of 5

(e)

The **IC74×138** is completely represented as the circuit shown in Figure X6.44 in the text book.

Step 1 of 1

Consider the following Verilog program of a lucky/prime encoder:

```
module primenum3(o,i);

output o; input [10:0]i; integer k; reg o;

always @(i)

begin

k=i;

if(i[0]==1'b0)

begin o=1'b0; $display("not prime"); end

else

begin

if(k==3 | k==5 | k==7 | k==11 | k==13 | k==17 | k==19)

begin o=1'b1; $display("prime"); end

else if(k%3==0 | k%5==0 | k%7==0 | k%11==0 | k%13==0 | k%17==0 | k%19==0)

begin o=1'b0; $display("not prime"); end

else

begin o=1'b1; $display("prime"); end

end

if(i==10'b00 | i==10'b010)

begin o=1'b1; $display("prime"); end

end

endmodule
```

Step 1 of 1

The following is the VHDL program for the seven-segment decoder:

```
LIBRARY IEEE;USE IEEE.STD_LOGIC_1164.ALL;

entity bcd2led is port (a,b,c,d,en:in std_logic; segs:out std_logic_vector (6 downto 0));end bcd2led;

architecture rtl of bcd2led isbegin process (a or b or c or d or en) begincase (d,c,b,a) is when "0000" =>
segs <="1111110"; when "0001"=> segs <="0110000"; when "0010"=> segs <="1101101"; when "0011"=>
segs <="1111001"; when "0100"=> segs <="0110011"; when "0101"=> segs <="1011011"; when "0110"=>
segs <="x011111"; when "0111"=> segs <="1110000"; when "1000"=> segs <="1111111"; when "1001"=>
segs <="111x011"; when others=> segs <="xxxxxxx";end case;end process;end rtl;
```

Step 1 of 3

Consider the following digital number:

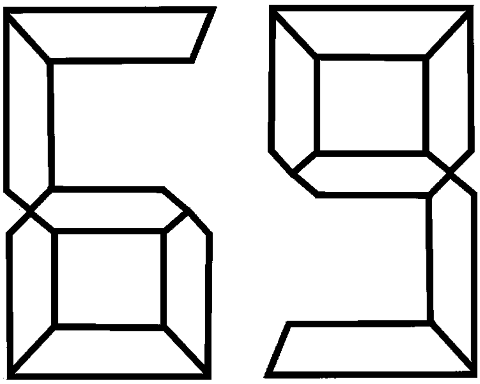


Figure 1

Step 2 of 3

Redesign the Verilog seven-segment decoder so that the digits 6 and 9 have tails as shown in Figure 1. In addition display the character "E" for non-decimal inputs 1010 through 1111.

Step 3 of 3

The following is the Verilog code for the same:

```
module segment(A,B,C,D,EN,sega,segb,segc,segd,sege,sege,segf,segg);

input A,B,C,D,EN;

output sega,segb,segc,segd,sege,segf,segg;

reg sega,segb,segc,segd,sege,segf,segg;

reg [1:7] segs;

always @(A or B or C or D or EN)

begin

if (EN)

case ([D,C,B,A})
```

// segment patterns abcdefg

```
0:segs=7'b1111110; //0

1:segs=7'b0110000; //1

2:segs=7'b1101101; //2

3:segs=7'b1111001; //3

4:segs=7'b0110011; //4

5:segs=7'b1011011; //5

6:segs=7'b1011111; //6 (with 'tail')

7:segs=7'b1110000; //7

8:segs=7'b1111111; //8

9:segs=7'b1111011; //9 (with 'tail')
```

// Character display for non-decimal inputs 1010 through 1111

```
4'b1010:segs = 7'b0001000; //A

4'b1011:segs = 7'b1000010; //B

4'b1100:segs = 7'b0000111; //C

4'b1101:segs = 7'b0000001; //D

4'b1110:segs = 7'b0110000; //E

4'b1111:segs = 7'b0000110; //F

default segs=7'bx;

endcase

else segs=7'b0;

[ sega,segb,segc,segd,sege,sege,segf,segg }=segs;

end

endmodule
```

Step 1 of 2

6.048E

The letters of A to F is shown in Figure 1:

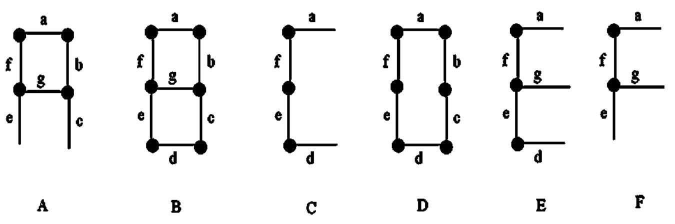


Figure 1

Step 2 of 2

The VHDL program for seven-segment decoder with the enhancements as follows:

```
module Vr7seg ( A,B,C,D,ENHEX,ERRDET,SEGA,SEGB,SEGC,SEGD,
SEGE,SEGF,SEGG) ;
input A,B,C,D,ENHEX,ERRDT ;
output SEGA,SEGB,SEGC,SEGD, SEGE,SEGF,SEGG ;
reg SEGA,SEGB,SEGC,SEGD, SEGE,SEGF,SEGG ;
reg [ 1:7 ] SEGS, SEGS_L ;
always @ (A or B or C or D or ENHEX or ERRDT ) begin
SEGS = ~ SEGS_L
if ( not ENHEX)
case ([D,C,B,A})
// Segment patterns abcdefg
0 : SEGS_L = 7'b0000001 ;
1 : SEGS_L = 7'b1001111 ;
2 : SEGS_L = 7'b0010010 ;
3 : SEGS_L = 7'b0000110 ;
4 : SEGS_L = 7'b1001100 ;
5 : SEGS_L = 7'b0100100 ;
6 : SEGS_L = 7'b1100000 ; // no tail
7 : SEGS_L = 7'b0001111 ;
8 : SEGS_L = 7'b0000000 ;
9 : SEGS_L = 7'b0001100 ; // no tail
default SEGS_L = 7 'b1111111 ;
end case ;
else if ( not (ENHEX & ERRDT) )
case ([D,C,B,A})
0 : SEGS_L = 7'b0000001 ;
1 : SEGS_L = 7'b1001111 ;
2 : SEGS_L = 7'b0010010 ;
3 : SEGS_L = 7'b0000110 ;
4 : SEGS_L = 7'b1001100 ;
5 : SEGS_L = 7'b0100100 ;
6 : SEGS_L = 7'b0100000 ; // tail
7 : SEGS_L = 7'b0001111 ;
8 : SEGS_L = 7'b0000000 ;
9 : SEGS_L = 7'b0000100 ; // tail
10 : SEGS_L = 7'b0001001 ; // A
11 : SEGS_L = 7'b0000000 ; // B
12 : SEGS_L = 7'b0110001 ; // C
13 : SEGS_L = 7'b0000001 ; // D
14 : SEGS_L = 7'b0110000 ; // E
15 : SEGS_L = 7'b0100100 ; // F
default SEGS_L = 7 'b1111111 ;
end case ;
else
case ([D,C,B,A})
0 : SEGS_L = 7'b0000001 ;
1 : SEGS_L = 7'b1001111 ;
2 : SEGS_L = 7'b0010010 ;
3 : SEGS_L = 7'b0000110 ;
4 : SEGS_L = 7'b1001100 ;
5 : SEGS_L = 7'b0100100 ;
6 : SEGS_L = 7'b0100000 ; // tail
7 : SEGS_L = 7'b0001111 ;
8 : SEGS_L = 7'b0000000 ;
9 : SEGS_L = 7'b0000100 ; // tail
10 : SEGS_L = 7'b0100100 ; // A
11 : SEGS_L = 7'b0100100 ; // B
12 : SEGS_L = 7'b0100100 ; // C
13 : SEGS_L = 7'b0100100 ; // D
14 : SEGS_L = 7'b0100100 ; // E
15 : SEGS_L = 7'b0100100 ; // F
default SEGS_L = 7 'b1111111 ;
end case ;
[ SEGA,SEGB,SEGC,SEGD, SEGE,SEGF,SEGG } = SEGS ;
end ;
endmodule
```

Step 1 of 1

6.049E

The verilog module for the seven segment decoder can be written starting with the specifications given in Exercise 6.48 (Refer to chapter 6) for all the conditions mentioned in the problem as follows by keeping in mind that all the outputs are active low.

Verilog Program:

Write the **VHDL** program as follows:

```
module segment(A,B,C,D,EN,sega,segb,segc,segd,sege,sege,segf,segg);

input A,B,C,D,EN,ENHEX,ERRDET;

output sega,segb,segc,segd,sege,segf,segg;

reg sega,segb,segc,segd,sege,segf,segg;

reg [1:7] segs;

always @(A or B or C or D or EN or ENHEX or ERRDET)

begin

if (EN==’1’ & ENHEX == ‘0’)

begin

case ([D,C,B,A})

// segment patterns abcdefg

0:segs=7’b1111110; //0

1:segs=7’b0110000; //1

2:segs=7’b1101101; //2

3:segs=7’b1111001; //3

4:segs=7’b0110011; //4

5:segs=7’b1011011; //5

6:segs=7’b1011111; //6 (with ‘tail’)

7:segs=7’b1110000; //7

8:segs=7’b1111111; //8

9:segs=7’b1111011; //9 (with ‘tail’)

4'b1010:segs = 7'b0001000; //A

4'b1011:segs = 7'b1000010; //B

4'b1100:segs = 7'b0000111; //C

4'b1101:segs = 7'b0000001; //D

4'b1110:segs = 7'b0110000; //E

4'b1111:segs = 7'b0000110; //F

Default segs=7’bx;

endcase

else segs=7’b0;

[ sega,segb,segc,segd,sege,sege,segf,segg }=segs;

end

// when ENHEX = 1 and ERRDET = 0, then the outputs for digits A-F look like the letters A-F as in the original program.

else If( ENHEX == ‘1’ & ERRDET ==’ 0’)

begin

4'b1010:segs = 7'b0001000; //A

4'b1011:segs = 7'b1000010; //B

4'b1100:segs = 7'b0000111; //C

4'b1101:segs = 7'b0000001; //D

4'b1110:segs = 7'b0110000; //E

4'b1111:segs = 7'b0000110; //F

Default segs=7’bx;

endcase

else segs=7’b0;

[ sega,segb,segc,segd,sege,sege,segf,segg }=segs;

End

// when ENHEX = 1 and ERRDET = 1, then the digits A-F look like the letter S.

else If (ENHEX == ‘1’ and ERRDET == ‘1’)

begin

case ([D,C,B,A})

// segment patterns abcdefg

6:segs=7’b1011111; //6 (with ‘tail’)

9:segs=7’b1111011; //9 (with ‘tail’)

4'b1010:segs = 7'b0001000; //A

4'b1011:segs = 7'b1000010; //B

4'b1100:segs = 7'b0000111; //C

4'b1101:segs = 7'b0000001; //D

4'b1110:segs = 7'b0110000; //E

4'b1111:segs = 7'b0000110; //F

Default segs=7’bx;

endcase

else segs=7’b0;

[ sega,segb,segc,segd,sege,sege,segf,segg }=segs;

end

endmodule
```

Step 1 of 4

Design:

The below Figure shows 10 to 4 line decoder in which the inputs I0 to I7 are active high inputs and Y0 to Y3 are active high outputs and the last two input lines 8 and 9 encoded into ‘E’ and ‘F’ respectively.

Step 2 of 4

Truth Table:

The truth table is shown in Table 1:

	INPUTS										OUTPUTS			
Decimal	I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	Y3	Y2	Y1	Y0
0	1	1	1	1	1	1	1	1	1	0	0	0	0	0
1	1	1	1	1	1	1	1	1	0	1	0	0	0	1
2	1	1	1	1	1	1	1	0	1	1	0	0	1	0
3	1	1	1	1	1	1	0	1	1	1	0	0	1	1
4	1	1	1	1	1	0	1	1	1	1	0	1	0	0
5	1	1	1	1	0	1	1	1	1	1	0	1	0	1
6	1	1	1	0	1	1	1	1	1	1	0	1	1	0
7	1	1	0	1	1	1	1	1	1	1	0	0	1	1
8	1	0	1	1	1	1	1	1	1	1	1	1	1	0
9	0	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 1

Step 3 of 4

Logic Diagram:

In the 10 to 4 line decoder the inputs I0 to I7 are active high inputs and Y0 to Y3 are active high outputs. The last two inputs I8 and I9 are encoded into E and F respectively that is $Y3Y2Y1Y0 = 1110$ and $Y3Y2Y1Y0 = 1111$ respectively.

Step 4 of 4

Write the Boolean expression for the outputs.

$$Y0 = \overline{I8} + \overline{I6} + \overline{I4} + \overline{I2} + \overline{I0}$$
$$Y1 = \overline{I7} + \overline{I6} + \overline{I3} + \overline{I2} + \overline{I1} + \overline{I0}$$
$$Y2 = \overline{I5} + \overline{I4} + \overline{I3} + \overline{I1} + \overline{I0}$$
$$Y3 = \overline{I0} + \overline{I1}$$

The design of 10 to 4 line decoder for required specifications is shown in Figure 1:

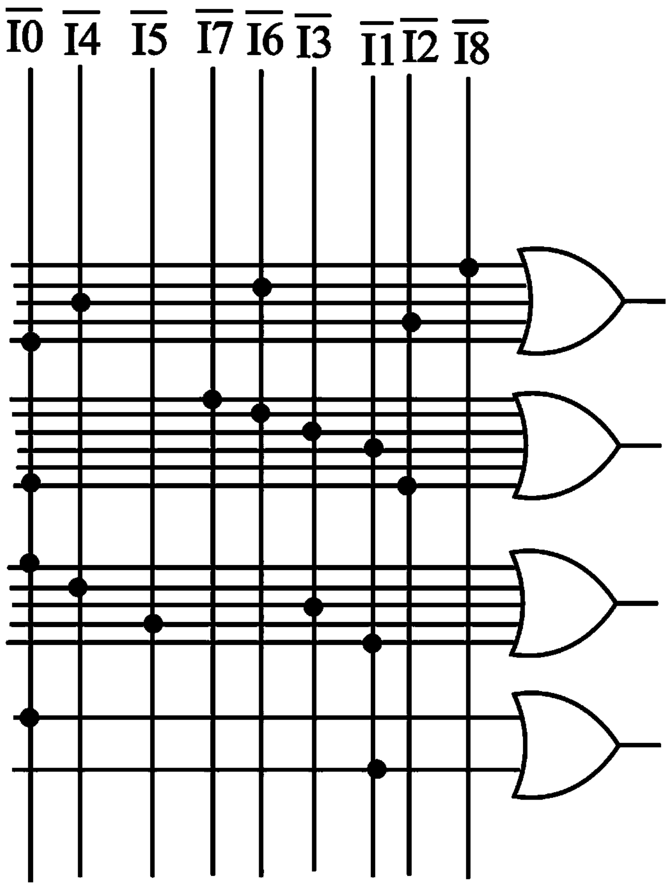


Figure 1

Thus, desired decoder is implemented.

Step 1 of 3

Design:

The equations for an 16-to-4 encoder using just four 8-input NAND gates are,

$$Y0\_L = \overline{I15}+\overline{I13}+\overline{I11}+\overline{I9}+\overline{I7}+\overline{I5}+\overline{I3}+\overline{I1}$$

$$Y1\_L = \overline{I15}+\overline{I14}+\overline{I11}+\overline{I10}+\overline{I7}+\overline{I6}+\overline{I3}+\overline{I2}$$

$$Y2\_L = \overline{I15}+\overline{I14}+\overline{I13}+\overline{I12}+\overline{I7}+\overline{I6}+\overline{I5}+\overline{I4}$$

$$Y3\_L = \overline{I15}+\overline{I14}+\overline{I13}+\overline{I12}+\overline{I11}+\overline{I10}+\overline{I9}+\overline{I8}$$

Step 2 of 3

16-to-4 encoder using just four 8-input NAND gates is shown in Figure 1.

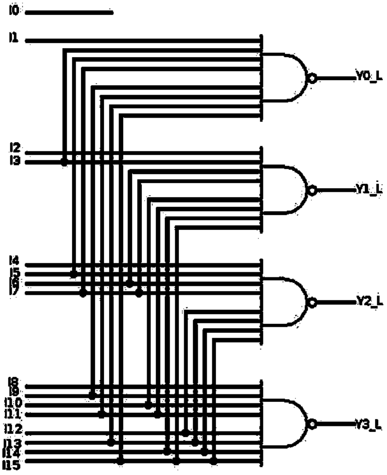


Figure 1

Step 3 of 3

The active levels of the inputs and outputs in the design are as follows.

Inputs: Active High

Outputs: Active Low

Step 1 of 3

The 74x148 uses eight active low inputs I7 to I0 where I7 is holding the highest priority, active low address outputs A2 to A0, two low active enables EI for input and the EO for output and a low active group select GS.

The low active inputs can be made active if we make them pass through the NOT gates before it reaches 74x148 and the low active address outputs must be inverted so that they can indicate the number of highest priority asserted input. The GS acts as the IDLE in 74x148. When no input is asserted A2 – A0, then it will be 111 and IDLE will be asserted as shown in Figure 1.

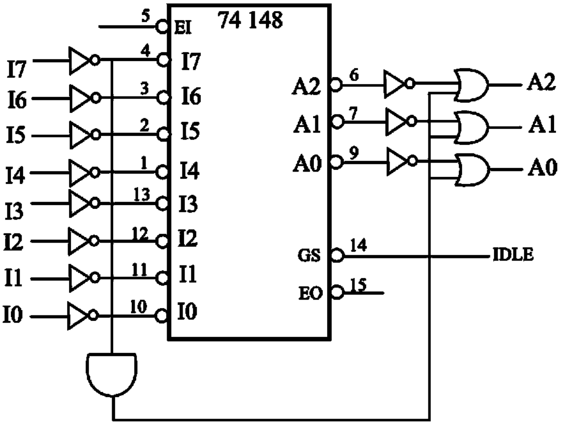


Figure 1

Step 2 of 3

The design works as shown in Table 1.

Table 1

Inputs	Outputs													
EI	I0	I1	I2	I3	I4	I5	I6	I7		A2	A1	A0	IDLE	EO
1	x	x	x	x	x	x	x	x		0	0	0	1	1
0	x	x	x	x	x	x	x	1		1	1	1	0	1
0	x	x	x	x	x	x	1	0		1	1	0	0	1
0	x	x	x	x	x	1	0	0		1	0	1	0	1
0	x	x	x	x	1	0	0	0		1	0	0	0	1
0	x	x	x	1	0	0	0	0		0	1	1	0	1
0	x	x	1	0	0	0	0	0		0	1	0	0	1
0	x	1	0	0	0	0	0	0		0	0	1	0	1
0	1	0	0	0	0	0	0	0		0	0	0	0	1
0	0	0	0	0	0	0	0	0		1	1	1	1	0

Step 3 of 3

When no input is asserted A2–A0, then it will be 111 and IDLE will be asserted. The high active address outputs indicates the number of the highest priority asserted input as shown in Table 1.

Step 1 of 3

The given Figure 5.3 below, which uses only 74x148 IC and no other gates to resolve priority among eight active-high inputs, I0 to I7, where I0 has the highest priority.

Naming of input and output signals:

• A0-A2 are active high outputs.

• I0-I7 are active high inputs.

• If at least one input is asserted, then an AVALID output can be asserted.

• EO_L is used for cascading with the other Priority Encoder.

• EN is enable input to select among the many encodes.

• EO_L is asserted if EN is asserted but no request input is asserted.

Step 2 of 3

Logic Diagram:

In the below Figure 5.3 the circuit is a priority encoder where I0_L is having the highest priority that uses GS output which is a group select output where one or more request inputs are asserted, and has a Enable input that must be asserted for any of its outputs to be asserted. In this example IDLE output is asserted if no inputs are asserted. If at least one input is asserted, then an AVALID output can be asserted as shown in the Figure and EO_L is used for cascading with the other Priority Encoder.

Step 3 of 3

The circuit diagram of encoder is shown in Figure 1:

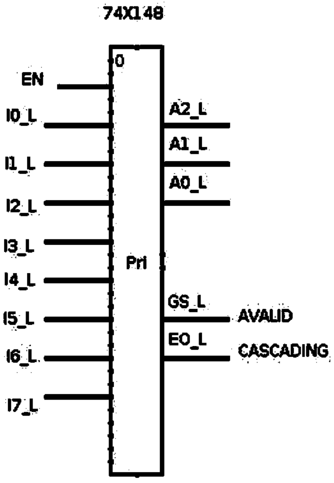


Figure 1

Thus desired circuit is implemented.

Step 1 of 2

Yes, according to Exercise 6.53 it is not always possible to maintain consistency in active-level notation unless you are willing to define alternate logic symbols for MSI parts that can be used in different ways.

So encoders can be designed using equations for intermediate variables to detect highest priority asserted input, for is purpose a single GAL20V8 can be used as follows as shown in Figure 1 , which gives alternate to 74X148 symbol.

Step 2 of 2

The pin diagram of GAL20V8 is shown in Figure 1:

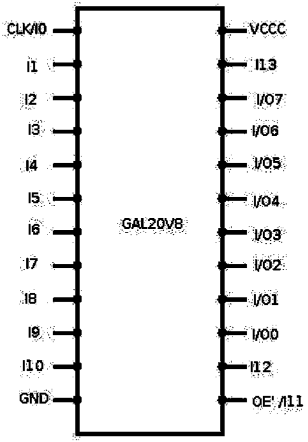


Figure 1

Step 1 of 2 **6.055E**

Consider the following conditions to design a combinational circuit with eight active-low request inputs as follows:

• R0_L-R7_L, A2-A0, AVALID defines the first highest priority as defined in Exercise 6.53 (Refer to chapter 6 in the text book).

• R8_L-R15_L, B2-B0, and BVALID defines the second highest priority.

• GS_L (Group Select-AVALID) output is asserted when then the device is enabled and one or more of the request inputs are asserted.

• EN (Enable Input) is asserted for any of it outputs to be asserted.

• EO_L (Enable output) used for cascading as it is connected to the previous enable input.

Step 2 of 2

Use two MSI 74x148 components (as mentioned in the problem not to use more than six).

The design of combinational circuit with eight active low request inputs is shown in Figure 1:

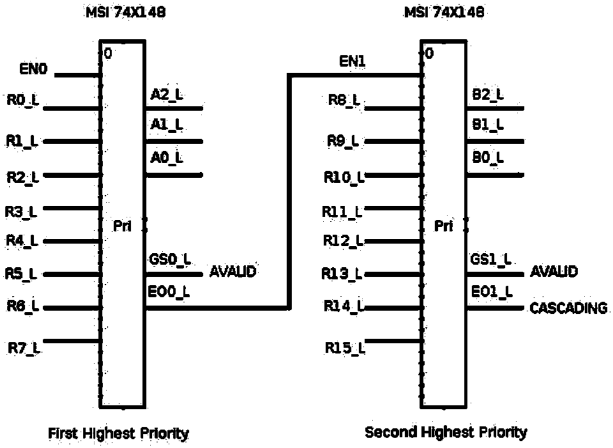


Figure 1

Step 1 of 3

ABEL Program:

It can be written by keeping Figure 1 in mind. The following figure is designed as mentioned in the Exercise 6.55 (Refer to chapter 6 in the textbook.)

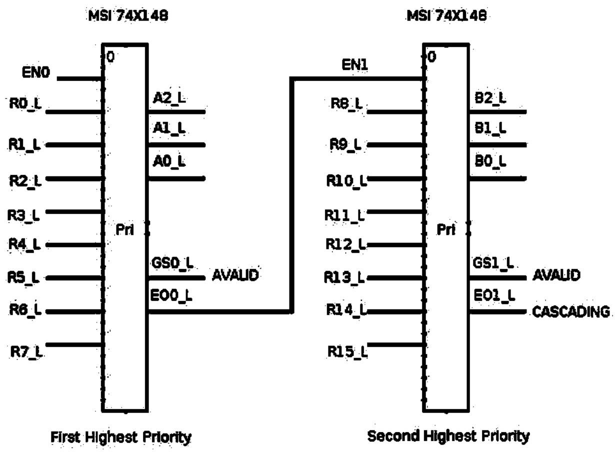


Figure 1

Step 2 of 3

Write the ABEL program as follows:

```
MODULE ENCODER
TITLE '15-INPUT PRIORITY ENCODER'
"INPUT AND OUTPUT PINS
R0....R14,EN0,EN1
A0....A2,B0....B2
GS0_L,GS1_L
"SET DEFINITION
GS0_L=AVALID
GS1_L=BVALID
WHEN !EN THEN AVALID = 0
ELSE WHEN R15 THEN BVALID=15;
ELSE WHEN R14 THEN BVALID=14;
ELSE WHEN R13 THEN BVALID=13;
ELSE WHEN R12 THEN BVALID=12;
ELSE WHEN R11 THEN BVALID=11;
ELSE WHEN R10 THEN BVALID=10;
ELSE WHEN R9 THEN BVALID=9;
ELSE WHEN R8 THEN BVALID=8;
ELSE WHEN R7 THEN AVALID=7;
ELSE WHEN R6 THEN AVALID=6;
ELSE WHEN R5 THEN AVALID=5;
ELSE WHEN R4 THEN AVALID=4;
ELSE WHEN R3 THEN AVALID=3;
ELSE WHEN R2 THEN AVALID=2;
ELSE WHEN R1 THEN AVALID=1;
ELSE WHEN R0 THEN AVALID=0;
ELSE {AVALID = 0; BVALID = 0;};
GS=EN&(R14#R13#R12#R11#R10#R9#R8#R7#R6#R5#R4#R3#R2#R1);
END ENCODER
```

Step 3 of 3

Therefore, the design fits into two GAL20V8 because it having many request inputs.

Step 1 of 3 **6.057E**

Consider the following conditions to design a combinational circuit with eight active-low request inputs as follows:

- R0_L-R7_L, A2-A0, AVALID defines the first highest priority as defined in Exercise 6.53 in the text book.
- R8_L-R15_L, B2-B0, and BVALID define the second highest priority.
- GS_L (Group Select-AVALID) output is asserted when then the device is enabled and one or more of the request inputs are asserted.
- EN (Enable Input) is asserted for any of it outputs to be asserted.
- EO_L (Enable output) used for cascading as it is connected to the previous enable input.

Step 2 of 3

Write the VHDL code for designing combinational circuit:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use ieee.numeric_std.all;
entity priority_encoder is
port(R : in STD_LOGIC_VECTOR(7 downto 0);
A : out STD_LOGIC_VECTOR(2 downto 0);
B : out STD_LOGIC_VECTOR(2 downto 0);
AVALID,BVALID : out STD_LOGIC);
end priority_encoder;
architecture priority_enc_arc of priority_encoder is
begin
pri_enc : process (R) is
begin
if (R(7)='0') then
A <= "000";
AVALID <= '1';
if (R(6)='0') then
B <= "001";
BVALID <= '1';
elsif (R(5)='0') then
B <= "010";
BVALID <= '1';
elsif (R(4)='0') then
B <= "011";
BVALID <= '1';
elsif (R(3)='0') then
B <= "100";
BVALID <= '1';
elsif (R(2)='0') then
B <= "101";
BVALID <= '1';
elsif (R(1)='0') then
B <= "110";
BVALID <= '1';
elsif (R(0)='0') then
B <= "111";
BVALID <= '1';
end if;
end if;
if (R(6)='0') then
A <= "001";
AVALID <= '1';
if (R(5)='0') then
B <= "010";
BVALID <= '1';
elsif (R(4)='0') then
B <= "011";
BVALID <= '1';
elsif (R(3)='0') then
B <= "100";
BVALID <= '1';
elsif (R(2)='0') then
B <= "101";
BVALID <= '1';elsif (R(1)='0') then
B <= "110";
BVALID <= '1';
elsif (R(0)='0') then
B <= "111";
BVALID <= '1';
end if;
end if;
if (R(5)='0') then
A <= "010";
AVALID <= '1';
if (R(4)='0') then
B <= "011";
BVALID <= '1';
elsif (R(3)='0') then
B <= "100";
BVALID <= '1';
elsif (R(2)='0') then
B <= "101";
BVALID <= '1';
elsif (R(1)='0') then
B <= "110";
BVALID <= '1';
elsif (R(0)='0') then
B <= "111";
BVALID <= '1';
```

Step 3 of 3

```
end if;
end if;
if (R(4)='0') then
A <= "011";
AVALID <= '1';
if (R(3)='0') then
B <= "100";
BVALID <= '1';
elsif (R(2)='0') then
B <= "101";
BVALID <= '1';
elsif (R(1)='0') then
B <= "110";
BVALID <= '1';
elsif (R(0)='0') then
B <= "111";
BVALID <= '1';
end if;
end if;
if (R(3)='0') then
A <= "100";
AVALID <= '1';
if (R(2)='0') then
B <= "101";
BVALID <= '1';
elsif (R(1)='0') then
B <= "110";
BVALID <= '1';
elsif (R(0)='0') then
B <= "111";
BVALID <= '1';
end if;
end if;
if (R(2)='0') then
A <= "101";
AVALID <= '1';
if (R(1)='0') then
B <= "110";
BVALID <= '1';
elsif (R(0)='0') then
B <= "111";
BVALID <= '1';
end if;
end if;
if (R(1)='0') then
A <= "110";
AVALID <= '1';
if (R(0)='0') then
B <= "111";
BVALID <= '1';
end if;
end if;
if (R(0)='0') then
A <= "111";
AVALID <= '1';
end if;
end process pri_enc;
end priority_enc_arc;
```

Hence, the VHDL code is written for desinging the combinational circuit.

Step 1 of 2 6.058E

Refer to Table 6-30 from the textbook for the VHDL program of priority encoder. In this table, for loop is used for higher priority order.

Write the VHDL program in which for loop starts from lowest-priority input.

```
library IEEE;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_1164.all;
entity V74x148 is
port (
I_L : in std_logic_vector(7 downto 0);
EI_L: in std_logic;
A_L: out std_logic_vector (2 downto 0);
EO_I, GS_L : out STD_LOGIC
);
end V74x148;
architecture V74x148_p of V74x148 is
signal EI, EO,GS: STD_LOGIC;
signal I: STD_LOGIC_VECTOR(7 downto 0);
signal A: STD_LOGIC_VECTOR(2 downto 0);
begin
process (EI_L, I_L, EI, EO, GS, I, A)
variable j: INTEGER range 0 downto 7;
begin
EI <= not EI_L;
I <= not I_L;
A <= "000";
GS <= '0';
EO <= '0';
if (EI = '0') then
else for j in 0 downto 7 loop
if I(j) = '1' then
GS <= '1';
EO <= '0';
A <= CONV_STD_LOGIC_VECTOR(j,3);
exit;
end if;
end loop;
end if;
EO_L <= not EO;
GS_L <= not GS;
A_L <= not A;
end process;
end V74x148_p;
```

Step 2 of 2

Therefore, the VHDL program as shown in Table 6-30, searches for the highest priority whereas if start the for loop with the lowest priority input, then it checks for the lowest priority input.

Step 1 of 2

6.059E

Refer to Table 6-31 from the textbook for the Verilog code of priority encoder. In this, the for loop starts with highest priority input.

Write the Verilog code for priority encoder.

```
module Vr74x148(EI_L, I_L, A_L, EO_L, GS_L);
input EI_L;
input [7:0] I_L;
ouput [2:0] A_L;
output EO_L, GS_L;
reg [7:0] I;
reg [2:0] A, A_L;
reg EI, EO_L, EO, GS_L, GS;
integer j;
always @ (EI_L or EI or I_L or I or A or EO or GS) begin
EI = ~EI_L;
I = ~I_L;
EO_L = ~EO;
GS_L = ~GS;
A_L = ~A;
EO = 1;
GS = 0;
A = 0;
begin
if (EI == 0) EO = 0;
else for (j=7; j>=0; j=j-1)
if (I[j] == 1)
begin
GS = 1;
EO = 0;
A = j;
end
end
end
end module
```

Step 2 of 2

The disable statement is used to exit the for loop. The disable statement was not supported in the Xilinx XST software. But disable statement works in Verilog.

Therefore, the Verilog program as shown in Table 6-31, searches for the lowest priority whereas if start the for loop with the highest priority input then it checks for the highest priority input.

Step 1 of 2 **6.060E**

Select device as 74FCT.

Table 1

Part number	From	To	t_{pLH} (ns)	t_{pHL} (ns)
'541(octal three- state buffer)	G1	Any Yi	8	8
	Any Ai	Any Yi	8	8
	G2	Any Yi	8	8

As the FCT device (three state buffer) is driving ten FCT device inputs, two 74 x 541 FCTS are required. First FCT has eight outputs and the second has two outputs. In the second 74 x 541 only two inputs and corresponding two outputs are used. Other inputs or outputs remain disconnected.

The outputs of 74 x 541 are connected to the inputs of FCT devices. If the output of 74 x 541 changes from LOW to Hi-Z, it means that the particular output bit is not selected. As a pull up resistor is used here, it provides stable HIGH logic state for floating or Hi-Z state.

Thus the delay in changing from Hi-Z to HIGH is zero or very much negligible because electricity flows through a resistor with the velocity of light.

Step 2 of 2

Calculate the total delay of two 74 x 541 from input to output as follows:

Table 2

Part number	From	To	t_{pLH} (ns)	t_{pHL} (ns)
'541(octal three- state buffer)1st device	G1	8 Yi	8x8=64	8x8=64
	8 Ai	8 Yi	8x8=64	8x8=64
	G2	8 Yi	8x8=64	8x8=64
'541(octal three- state buffer)2nd device	G1	2 Yi	8x2=16	8x2=16
	2 Ai	2 Yi	8x2=16	8x2=16
	G2	2 Yi	8x2=16	8x2=16
Total delay			{(64+16)x3}=240	

Thus, the FCT takes $\boxed{240\text{ ns}}$ to see HIGH output.

A three state bus is driving ten FCT inputs. When the status of active bus is Hi-Z, it means no buffers connected to this bus are enabled.

In case of TTL gates Hi-Z state is treated as acceptable or clear - '1'. But in CMOS floating state CMOS input is undecided. It means that it can be treated as high voltage ('1') or low voltage ('0') of electrostatic type.

The solution of this problem is to use a pull up or pull down resistor to make this Hi-Z state as stable HIGH or stable LOW states respectively.

Thus, the delay in changing from Hi-Z to stable HIGH is zero or negligible because electricity flows through the resistor with the velocity of light.

Therefore, the bus signal remains at a stable HIGH state **as long as the bus signal status is not changed by the inputs of three-state bus.**

6.062E

Step 1 of 5

The following is the truth table.

Step 2 of 5

Inputs		Outputs				
S1	S0	1Y	2Y	3Y	4Y	5Y
0	0	1D0	2D0	3D0	4D0	5D0
0	1	1D1	2D1	3D1	4D1	5D1
1	0	1D2	2D2	3D2	4D2	5D2
1	1	0	0	0	0	0

Step 3 of 5

Table 1

Step 4 of 5

Logic Diagram:

The following is the logic diagram.

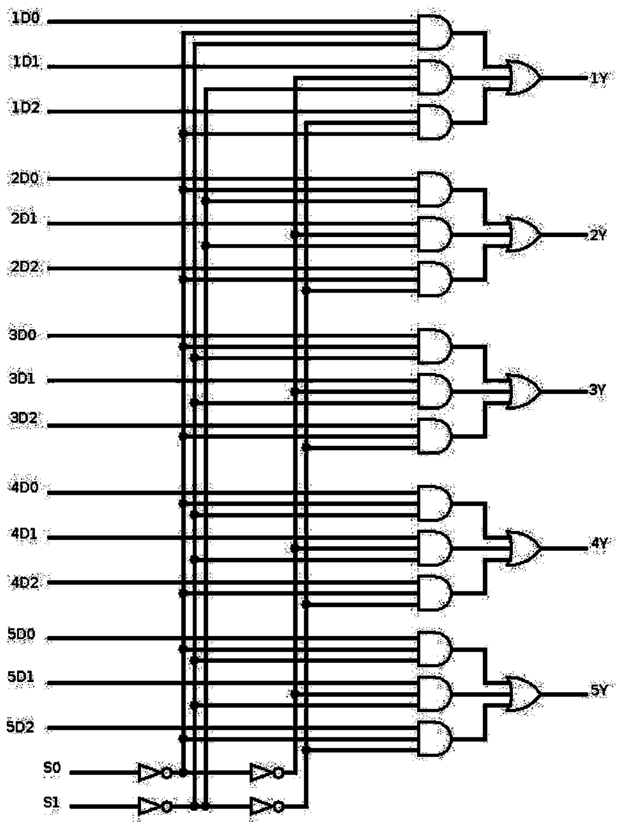


Figure 1

Step 5 of 5

Logic Symbol:

The following is the logic symbol.

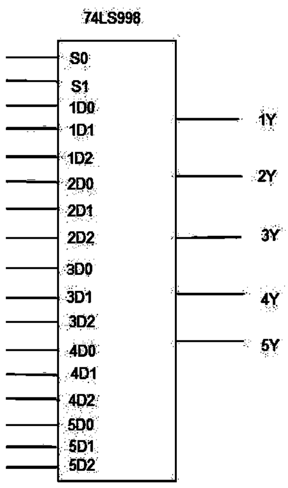


Figure 2

Step 1 of 2

Refer to Table 6-43 from the text book for the functional table of 74×157 multiplexer.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity V74x157 is
port ( EN_L : in STD_LOGIC;
S : in STD_LOGIC;
D1 : in STD_LOGIC_Vector(1 downto 0);
D2 : in STD_LOGIC_Vector(1 downto 0);
D3 : in STD_LOGIC_Vector(1 downto 0);
D4 : in STD_LOGIC_Vector(1 downto 0);
Y1 : out STD_LOGIC;
Y2 : out STD_LOGIC;
Y3 : out STD_LOGIC;
Y4 : out STD_LOGIC);
end V74x157;
architecture arc_V74x157 of V74x157 is
begin
process (EN_L, S)
begin
if (EN_L = '1') then
Y1 <= '0';
Y2 <= '0';
Y3 <= '0';
Y4 <= '0';
else
if (S = '0') then
Y1 <= D1(0);
Y2 <= D2(0);
Y3 <= D3(0);
Y4 <= D4(0);
end if;
if (S = '1') then
Y1 <= D1(1);
Y2 <= D2(1);
Y3 <= D3(1);
Y4 <= D4(1);
end if;
end if;
end process;
end arc_V74x157;
```

Step 2 of 2

Hence, the VHDL program for 74×157 multiplexer is written and tested.

Step 1 of 2

6.065E

Refer Table 6-46 from the textbook.

The 4-input, 18-bit multiplexer with the functionality described in Table 6-46 using 18 74x151s is as shown in Figure 1.

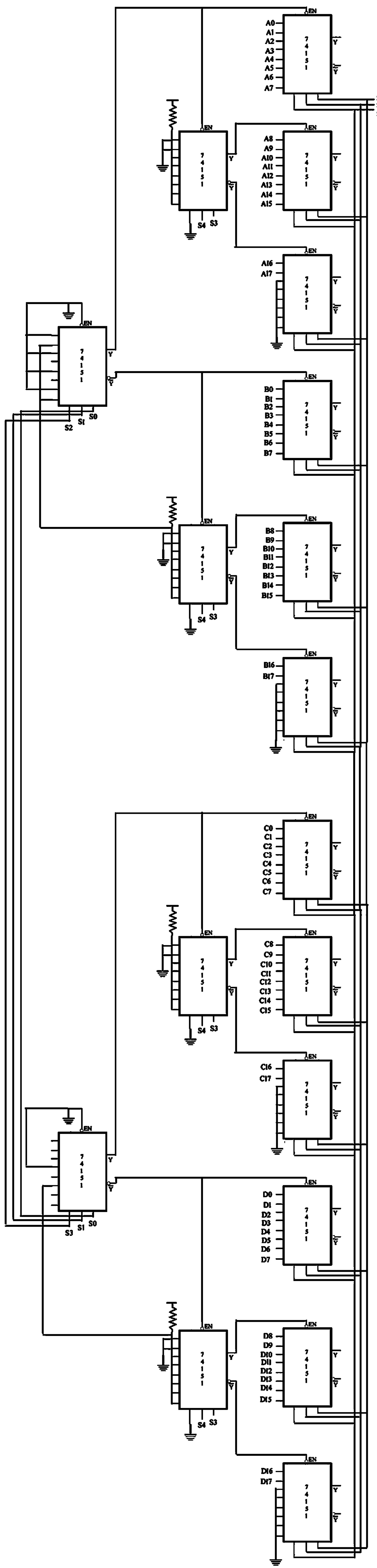


Figure 1

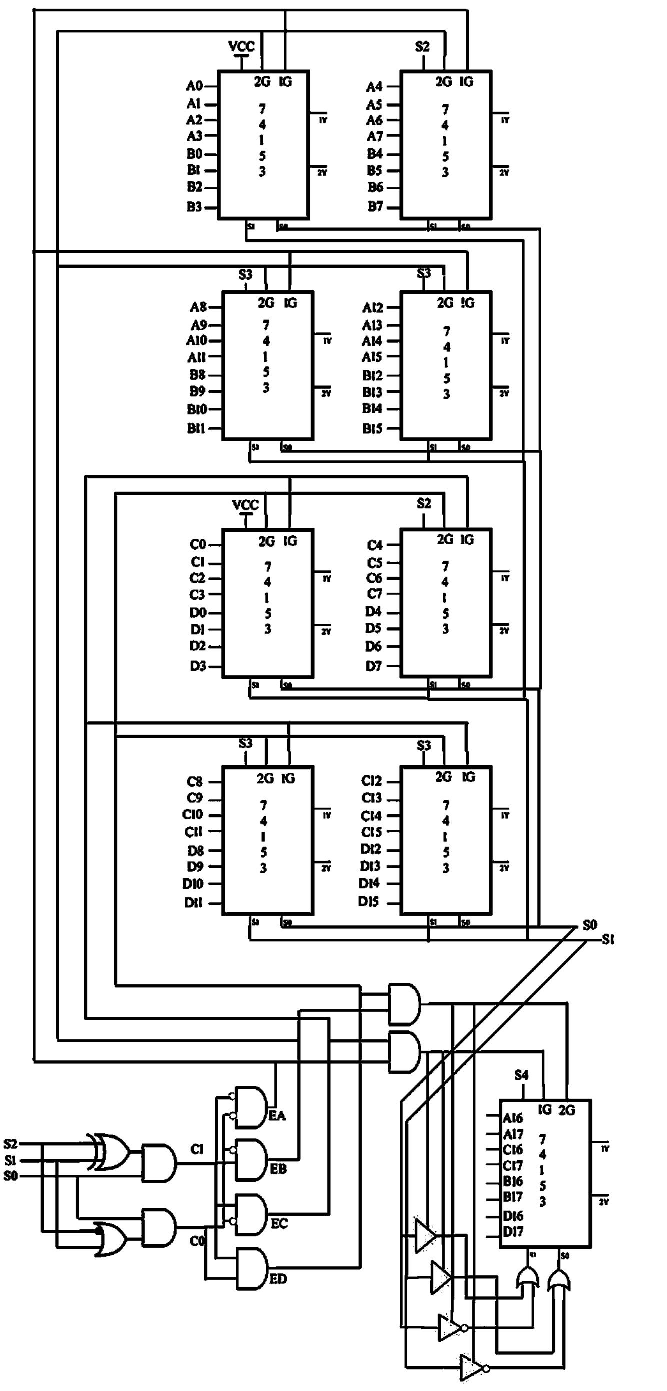
Step 2 of 2

The 4-input MUX will have three selection lines which are given as the input to the 2 74x151s which are located to the extreme left of the Figure 1. The input is of 18-bit which are selected by the five selection lines. The output is taken from the Ys of all the 74x151s which are located to the extreme right of the Figure 1.

Step 1 of 2

Refer Table 6-46 from the textbook.

The 4-input, 18-bit multiplexer with the functionality of Table 6-46 using nine 74x153s and a code convertor is as shown in Figure 1.



VCC
S2
S3
S4
2G 1G
1G 2G
74153
1Y
2Y
S1 S0
A0 A1 A2 A3 B0 B1 B2 B3
A4 A5 A6 A7 B4 B5 B6 B7
A8 A9 A10 A11 B8 B9 B10 B11
A12 A13 A14 A15 B12 B13 B14 B15
C0 C1 C2 C3 D0 D1 D2 D3
C4 C5 C6 C7 D4 D5 D6 D7
C8 C9 C10 C11 D8 D9 D10 D11
C12 C13 C14 C15 D12 D13 D14 D15
A16 A17 C16 C17 B16 B17 D16 D17
S0
S1
S2
C1
C0
EA
EB
EC
ED

Figure 1

Step 2 of 2

The code convertor does the selection part of the 4 inputs. The code convertor selects the 4 inputs as shown in Table 1.

Table 1

Input	Intermediate signals	Output			
S2	*S1*	*S0*	*C1*	*C0*	
0	0	0	0	0	A
0	0	1	0	1	B
0	1	0	0	0	A
0	1	1	1	1	C
1	0	0	0	0	A
1	0	1	1	0	D
1	1	0	0	0	A
1	1	1	0	1	B

Each input is of 18-bit are selected by the five selection lines. The output is taken from all the Ys of all the 74x153s in Figure 1.

Step 1 of 4

6.067E

Refer to Table 6-46 in the text book.

The following is function table for a specialized 3-input, 2-output combinational circuit.

Table 1

Inputs	Outputs			
S_2	S_1	S_0	C_1	C_0
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	1	0
1	1	0	0	0
1	1	1	0	1

Step 2 of 4

Draw the k-map to find the expression for output C_1.

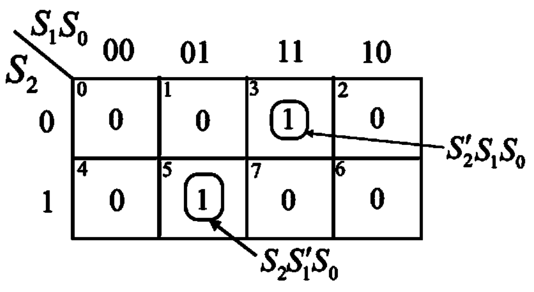


Figure 1

From Figure 1, the expression for output C_1 is,

$$C_1 = S_2'S_1S_0 + S_2S_1'S_0$$

Step 3 of 4

Draw the k-map to find the expression for output C_0.

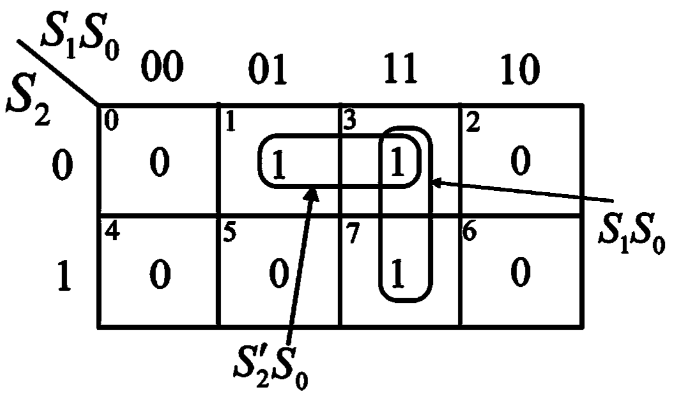


Figure 2

From Figure 2, the expression for output C_0 is,

$$C_0 = S_1S_0 + S_2'S_0$$

Step 4 of 4

The design of the code convertor with the equations C_1 and C_0 is shown in Figure 3.

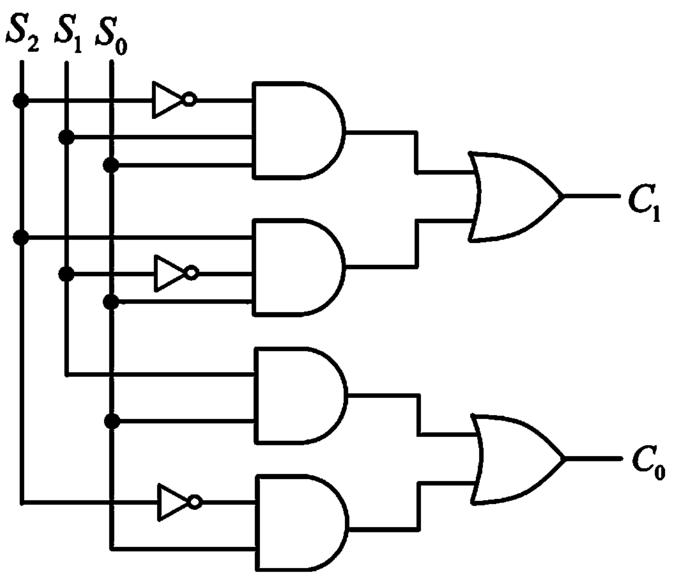


Figure 3

Thus, the 3-input, 2-output combiantiona; circuit that performs the code converter is designed.

Step 1 of 3 **6.068E**

The logic function performed by the CMOS circuit shown in Figure 1 can be found as follows:

Truth Table:

In the truth table S is select input and Z is the output.

S	Z
0	B
1	A

Table 1.1

Step 2 of 3

Logic Equations:

From Table 1.1, the Boolean equations for the 2 to 1 Mux is given as follows:

$$Z = AS + B\overline{S}$$

Step 3 of 3

Logic Diagram:

The logic diagram for the given CMOS circuit is shown in Figure 2:

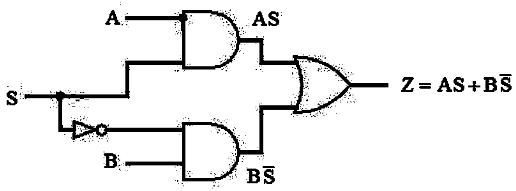


Figure 2

Step 1 of 4

Consider the following circuit diagram:

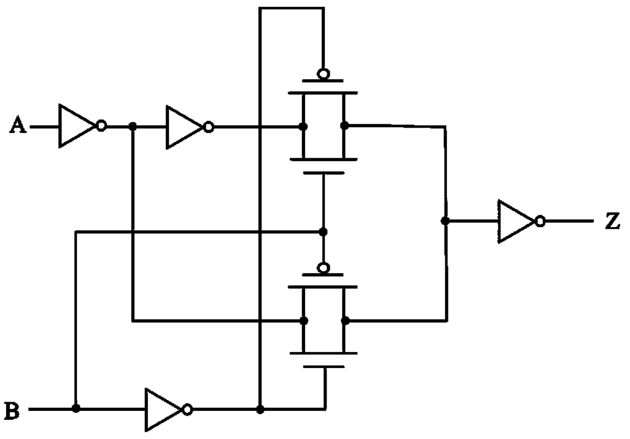


Figure 1

Step 2 of 4

The logic function is performed by the CMOS circuit shown in Figure 1, where A and B are inputs, and Z is the output which performs the XOR operation. The

Truth Table:

Inputs Output

A	B	Z
0	0	0
0	1	1
1	0	1
1	1	0

Step 3 of 4

Table 1.2

Step 4 of 4

Logic Equation:

From table 1.2, this performs the XOR operation

$$\mathbf{Z = \overline{A}B + A\overline{B}}$$

Logic Diagram:

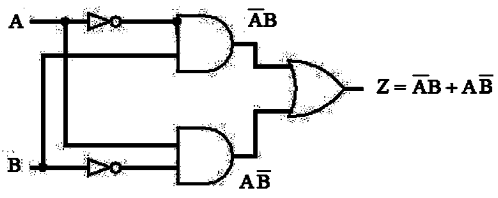


Figure 2

Step 1 of 5 **6.070E**

Refer Figure X6.70 from the textbook.

(a)

The following are the inputs and outputs of the circuit with appropriate signal names, including active-level indications:

Inputs : $\overline{1Ea}$, $\overline{2Ea}$, A0, A1, $\overline{1Eb}$, $\overline{2Eb}$

Outputs : $\overline{a0}$, $\overline{a1}$, $\overline{a2}$, $\overline{a3}$, b0, b1, b2, b3 respectively.

Step 2 of 5

(b)

The circuit acts as a dual 1 of 4 demultiplexer.

The truth table of the circuit is as shown in Table 1.

Address $A1$ $A0$	Enable "a" $\overline{1Ea}$ $\overline{2Ea}$	Output of "a" $\overline{a0}$ $\overline{a1}$ $\overline{a2}$ $\overline{a3}$	Enable "b" $\overline{1Eb}$ $\overline{2Eb}$	Output of "b" b0, b1, b2. b3
X X	H X	H H H H	H H	L L L L
X X	X H	H H H H	- -	- - - -
L L	L L	L H H H	L X	H L L L
L H	L L	H L H H	L X	L H L L
H L	L LL L	H H L H	L X	L L H L
H H		H H H L	L X	L L L H

Table 1

The truth table describes the operation of the circuit.

Step 3 of 5

(c)

The logic symbol of the circuit is as shown in Figure 1.

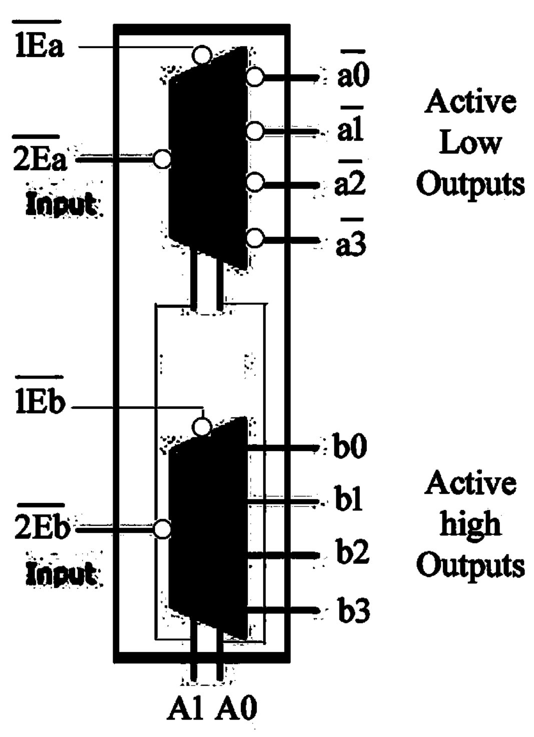


Figure 1

Step 4 of 5

(d)

The following is the VHDL program for the circuit:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity demultiplexer_case is
port(
ea : in STD_LOGIC_vector(1 downto 0);
eb : in STD_LOGIC_vector(1 downto 0);
sel : in STD_LOGIC_VECTOR(1 downto 0);
a : out STD_LOGIC_VECTOR(3 downto 0);
b : out STD_LOGIC_VECTOR(3 downto 0)
);
end demultiplexer_case;
architecture demultiplexer_case_arc of demultiplexer_case is
begin
demux : process (sel, ea, eb) is
begin
case sel is
when "00" => if(ea="00") then a <= "0111"; elsif(eb="0-") then b <= "1000"; end if;
when "01" => if(ea="00") then a <= "1011"; elsif(eb="0-") then b <= "0100"; end if;
when "10" => if(ea="00") then a <= "1101"; elsif(eb="0-") then b <= "0010"; end if;
when others => if(ea="00") then a <= "1110"; elsif(eb="0-") then b <= "0001"; end if;
end case;
end process demux;
end demultiplexer_case_arc;
```

Step 5 of 5

(e)

Yes, it would be successful as an MSI part and it competes with 74LS139, 74LS156, 74LS155 etc.

Step 1 of 1

The first set of 74157 ICs is controlled by S0 to shift the input word by 1 or 0 bits. The data outputs of this set are connected to the inputs of a second set. The second set of 74157 ICs is controlled by S1, which shifts its input word left by 0 or 2 bits. Continuing the cascade, a third and fourth set are controlled by S2 and S3 respectively to shift 4 and 8 bits as shown in Figure 1.

The 74157-based approach requires only half as many MSI packages and has far less data inputs and loading on the control inputs. It has the longest data-path delay, since each data bit must pass through four 74x157 ICs. Halfway between the two approaches, we can use eight 74x153 4-input, 2-bit multiplexers two build a 4-input, 16 -bit multiplexer. Cascading two sets of these, we can use S[3:2] to shift selectively by 0, 4, 8, or 12 bits, and S[1:0] to shift by 0–3 bits.

Observe that the required total number of **74157 ICs (type)** is **16**.

Step 1 of 1 6.072E

Write the VHDL program for the barrel shifter.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity bs16 is
port (
DIN: in STD_LOGIC_VECTOR(15 downto 0); – Data inputs
S: in STD_LOGIC_VECTOR (3 downto 0); – Shift amount, 0-15
DOUT: out STD_LOGIC_VECTOR(15 downto 0) – Data bus output);
end bs16;
architecture rol16_arch of bs16 is
begin
process(DIN, S)
variable X: STD_LOGIC_VECTOR(15 downto 0);
begin
if S = "0000" then
DOUT <= DIN;
end if;
if S = "0001" then
X := DIN(14 downto 0) & DIN(15);
end if;
if S = "0010" then
X := DIN(13 downto 0) & DIN(15 downto 14);
end if;
if S = "0011" then
X := DIN(12 downto 0) & DIN(15 downto 13);
end if;
if S = "0100" then
X := DIN(11 downto 0) & DIN(15 downto 12);
end if;
if S = "0101" then
X := DIN(10 downto 0) & DIN(15 downto 11);
end if;
if S = "0110" then
X := DIN(9 downto 0) & DIN(15 downto 10);
end if;
if S = "0111" then
X := DIN(8 downto 0) & DIN(15 downto 9);
end if;
if S = "1000" then
X := DIN(7 downto 0) & DIN(15 downto 8);
end if;
if S = "1001" then
X := DIN(6 downto 0) & DIN(15 downto 7);
end if;
if S = "1010" then
X := DIN(5 downto 0) & DIN(15 downto 6);
end if;
if S = "1011" then
X := DIN(4 downto 0) & DIN(15 downto 5);
end if;
if S = "1100" then
X := DIN(3 downto 0) & DIN(15 downto 4);
end if;
if S = "1101" then
X := DIN(2 downto 0) & DIN(15 downto 3);
end if;
if S = "1110" then
X := DIN(1 downto 0) & DIN(15 downto 2);
end if;
if S = "1111" then
X := DIN(0) & DIN(15 downto 1);
end if;
DOUT <= X;
end process;
end rol16_arch;
```

Hence, the VHDL code is written for the berrel shifter.

Step 1 of 1 **6.073E**

327-6-73E AID: 4132 | 11/05/2014

RID: 1988 | 22/05/2014

Refer to Table 6-49 from the textbook for the VHDL multiplexer program.

Rewrite the VHDL program for 4-input, 8-bit multiplexer including with three-state output control input OE.

```
library IEEE;

use IEEE.std_logic_1164.all;

entity mux4in8bit is

port (

OE : in STD_LOGIC;

S: in STD_LOGIC_VECTOR (1 downto 0); –Select inputs

A, B, C, D: in STD_LOGIC_VECTOR (7 downto 0);

– Data bus inputs

Y: out STD_LOGIC_VECTOR (7 downto 0)– Data bus output

);

end mux4in8bit;

architecture mux4in8p of mux4in8bit is

begin

process(OE,S, A, B, C, D)

begin

if(OE='1') then

case S is

when "00" => Y <= A;

when "01" => Y <= B;

when "10" => Y <= C;

when "11" => Y <= D;

when others => Y <= (others => 'U');-- 8-bit vector of 'U'

end case;

else

Y<=(others =>'U');

end if;

end process;

end mux4in8p;
```

Hence, the VHDL program for 4-input, 8-bit multiplexer including with three-state output control input OE has been written.

Step 1 of 1

Refer to Table X6.74 from the textbook.9014356785

Write the ABEL program for designing of customised multiplexer.

```
module chap6mux

title 'Eight-input multiplexer'

CHAP6MUX device 'P16V8';

"Input and Output pins

P1, P2, P3 pin istype 'reg';

Q1, Q2, Q3 pin istype 'reg';

R1, R2, R3 pin istype 'reg';

T1, T2, T3 pin istype 'reg';

S1, S2, S3 pin istype 'reg';

Y1, Y2, Y3 pin istype 'reg';

"Definitions

P = [P1, P2, P3];

Q = [Q1, Q2, Q3];

R = [R1, R2, R3];

T = [T1, T2, T3];

S = [S1, S2, S3];

Y = [Y1, Y2, Y3];

"Equations

WHEN S == [0, 0, 0] THEN Y = P;

ELSE WHEN S == [0, 0, 1] THEN Y = P;

ELSE WHEN S == [0, 1, 0] THEN Y = P;

ELSE WHEN S == [0, 1, 1] THEN Y = Q;

ELSE WHEN S == [1, 0, 0] THEN Y = P;

ELSE WHEN S == [1, 0, 1] THEN Y = P;

ELSE WHEN S == [1, 1, 0] THEN Y = R;

ELSE Y=T;

end chap6mux
```

Therefore, ABEL program is written for designing of multiplexer.

Step 1 of 1 **6.075E**

Refer to Table X6.74 from the textbook.

Write the VHDL code for multiplexer:

```
library ieee;

use ieee.std_logic_1164.all;

entity X674 is

port(

P,Q,R,T: in STD_LOGIC_VECTOR(7 downto 0);

S:in STD_LOGIC_VECTOR(3 downto 0);

Y: out STD_LOGIC_VECTOR(7 downto 0)

);

end X674;

architecture beh of X674 is

begin

process(S,P,Q,R,T)

begin

if(S="000" or S="001" or S="010" or S="100" or S="101") then Y<=P;

elsif(S="011") then Y<=Q;

elsif(S="110") then Y<=R;

elsif(S="111") then Y<=T;

end if;

end process;

end beh;
```

Step 1 of 1 **6.076E**

Write the following VHDL program for a customized multiplexer with four 8-bit input buses(P,Q,R,T), three select inputs(S2-S0) and two control inputs(C1-C0) and an output bus Y.

```
library ieee;

use ieee.std_logic_1164.all;

entity X674 is

port(

p,q,r,t: in STD_LOGIC_VECTOR(7 downto 0);

s:in STD_LOGIC_VECTOR(3 downto 0);

c:in STD_LOGIC_VECTOR(1 downto 0);

y: out STD_LOGIC_VECTOR(7 downto 0)

);

end X674;

architecture beh of X674 is

signal y1: STD_LOGIC_VECTOR(7 downto 0);

begin

process(s,p,q,r,t,c)

begin

if(s="000" or s="001" or s="010" or s="100" or s="101") then y1<=p;

elsif(s="011") then y1<=q;

elsif(s="110") then y1<=r;

elsif(s="111") then y1<=t;

end if;

if(c="00") then y<=(others=>'0');

elsif(c="01") then y<=(others=>'1');

elsif(c="10") then y<=y1;

else y<=y1 xor "11111111";

end if;

end process;

end beh;
```

Step 1 of 1

Refer to the table X6.77 from the text book.

A five 4-bit input buses A, B, C, D and E selecting one of the buses to drive a 4 bit output bus according to Table X6.77 is designed using two 74X153 and one 74X157 as shown in Figure 1.

Consider that $S0, S1, S2$ represents the selection lines, A, B, C, D and E are the input buses and T represents the output bus.

The customized multiplexer with designed specifications is as shown in Figure 1.

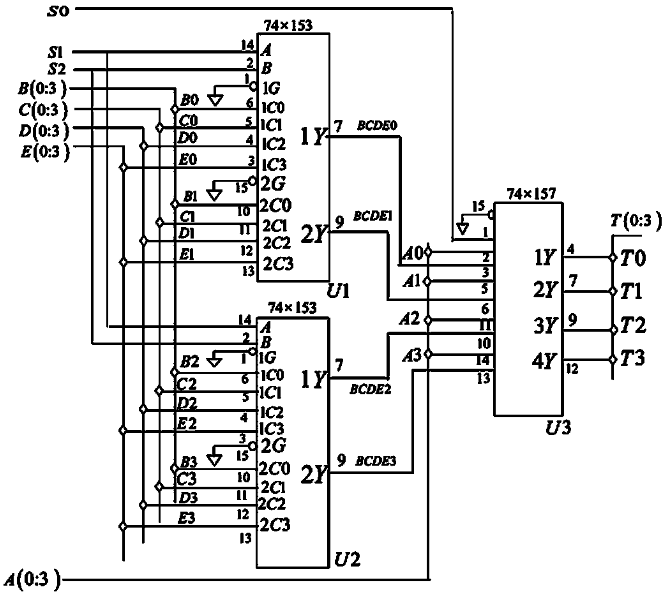


Figure 1

Step 1 of 1 **6.078E**

Refer to Table X6.77 from the textbook.

Write the ABEL code for the customized multiplexer with 4-bit input buses $A, B, C, D,$ **and** E and one output bus T .

```
module mux3in4b

title 'Specialized 4-bit, 3-input Multiplexer'

" Input pins

S2, S1, S0, A0, A1, A2, A3, B0, B1, B2, B3, C0, C1, C2, C3, D0, D1, D2, D3, E0, E1, E2, E3 pin;

" Output pins

F0, F1, F2, F3 pin;

" State Definitions

SEL = [S2, S1, S0];

A = [A3, A2, A1, A0];

B = [B3, B2, B1, B0];

C = [C3, C2, C1, C0];

D = [D3, D2, D1, D0];

E = [E3, E2, E1, E0];

F = [F3, F2, F1, F0];

equations

when (SEL==0) # (SEL==2) # (SEL==4) # (SEL==6) then F := A;

else when (SEL==1) then F := B;

else when (SEL==3) then F := C;

else when (SEL==5) then F := D;

else when (SEL==7) then F := E;

end mux3in4b
```

Thus, the ABEL code for the customized multiplexer is provided.

Step 1 of 1

6.079E

Refer to Table X6.77 from the textbook.

Write the ABEL code for the customized multiplexer with 4-bit input buses $A, B, C, D,$ **and** E and one output bus T .

```
library ieee;

use ieee.std_logic_1164.all;

entity X677 is

port(

a,b,c,d, e: in STD_LOGIC_VECTOR(7 downto 0);

s:in STD_LOGIC_VECTOR(3 downto 0);

y: out STD_LOGIC_VECTOR(7 downto 0)

);

end X677;

architecture beh of X677 is

begin

process(s,a,b,c,d,e)

begin

if(s="000" or s="010" or s="100" or s="110") then y<=a;

elsif(s="001") then y<=b;

elsif(s="011") then y<=c;

elsif(s="101") then y<=d;

elsif(s="111") then y<=e;

end if;

end process;

end beh;
```

Thus, the VHDL code for the customized multiplexer is provided.

Step 1 of 1

Refer to the Figure 6-72 from the text book.

The 74x00 has the same pin-out as the 74x08, but the outputs have the opposite polarity. The change in level at pin 3 of U1 is equivalent to a change at pin 4 of U2 (the input of an XOR tree), which is equivalent to a change at pin 6 of U2 (the parity-generator output).

Thus, the circuit simply generates and checks odd parity instead of even.

The change in level at pin 6 of U1 changed the active level of the ERROR signal.

Thus, due to the mentioned reasons, though the designer used 74x00 instead of 74xx08, except for the change in active level of ERROR SIGNAL, the circuit still worked.

Step 1 of 3

Refer to the Figure X6.81 from the textbook for a CMOS circuit.

Implement the truth table for the circuit.

Table 1

Input	**Output**		
A	**B**	**C**	**Z**
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Step 2 of 3

Use Karnaugh map on the output Z in the Table 1 to obtain the logic diagram.

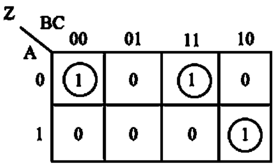


Figure 1

Step 3 of 3

Find the logic expression for the circuit.

$$Z = \bar{A}\bar{B}\bar{C} + \bar{A}BC + AB\bar{C}$$

Draw the logic diagram for the circuit:

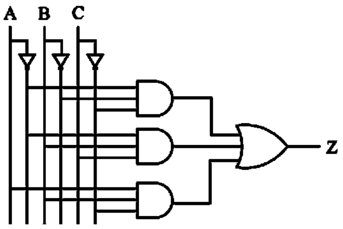


Figure 2

Thus, the logic diagram for the function of the circuit X6.81 is shown in Figure 2.

Step 1 of 8 **6.082E**

The number of 1's at input are odd number than it is called odd parity. The odd parity is identified using the XOR gate.

The truth table of odd parity is shown in Table 1:

Table 1

I_0	I_1	$I_0 \oplus I_1$	Odd parity
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

Step 2 of 8

From table 1, it is clear that **one XOR gate** is required for **two bit input**. In generally an odd parity circuit with 2^n inputs can be built with $2^n - 1$ XOR gates. The structure of odd parity can be implemented in two methods.

First method:

The logical expression of odd parity is,

$$\text{Odd parity} = I_0 \oplus I_1 \oplus I_2 \oplus I_3 \oplus \cdots \oplus I_n$$

In this method the odd parity circuit can be directly implemented that is 2^n inputs can be built with $2^n - 1$ XOR gates.

Step 3 of 8

The odd-parity circuit diagram using first method is shown in Figure 1:

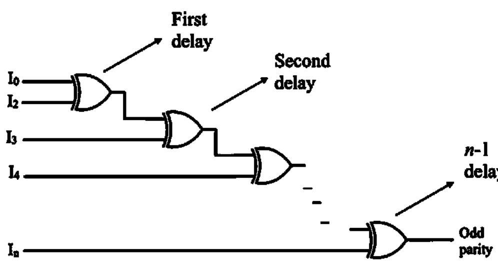


Figure 1

Step 4 of 8

From Figure 1, it is clear that one input takes **"n – 1" number of XOR gate delays**.

Step 5 of 8

Second method:

The logical expression of odd parity is,

$$\begin{aligned}\text{Odd parity} &= I_0 \oplus I_1 \oplus I_2 \oplus I_3 \oplus \cdots \oplus I_n \\ &= \left[\left(\left(I_0 \oplus I_1\right) \oplus \left(I_2 \oplus I_3\right)\right) \oplus \cdots\right] \oplus \left[I_n\right]\end{aligned}$$

In this method, the circuit can be built like a tree as shown in Figure 2:

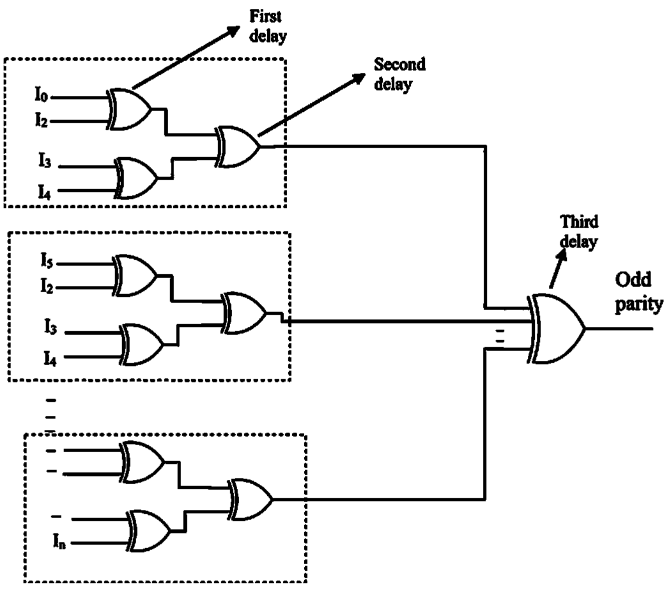


Figure 2

Step 6 of 8

From Figure 2, it is clear that one input takes **"3" number of XOR gate delays**.

Step 7 of 8

The delay numbers of the circuits in Method 1 and Method 2 are shown in Table 2:

Table 2

Circuit	**Number of XOR gate delays with "n" inputs**
Figure 1	n-1
Figure 2	3

Step 8 of 8

From Table 2, the input to output propagation delay is **minimum in second method** and **maximum in first method**.
Hence, the odd parity circuit diagram in **second method is preferred** than the circuit in first method.

Step 1 of 1

Refer to Figure 9-45 from the textbook for the XC4000 configurable logic block.

The main logic block (macrocell) in a Xilinx XC4000 field programmable gate array (FPGA) is the configurable logic block (CLB). The configurable logic blocks contain the programmable elements called the logic-function generators F, G and H. the blocks F and G perform any combinational logic function of their 4 inputs and H performs any combinational logic function of its 3 inputs. The outputs of F and G are directed to the inputs of H by multiplexers M1-M3.

Find the largest number of inputs that can be accommodated or realized by F, G and H in a single configurable logic block. Consider that the “steered” product terms from other macrocells are not used.

Some functions of up to 9 variables for two 4-bit inputs can be realized by a single configurable logic block in the XC4000 FPGA.

Thus, the largest number of inputs that can accommodate in a single macrocell is **2** four-bit inputs.

The following are the two situations in which there are up to 9 variables, including parity checking and a cascadable equality checker for two 4-bit inputs:

Realization of an equality checker for two 4-bit operands and

Realization of a 9-bit even parity function generator.

The macrocell resources that are used to achieve this result are counters, clock, reset, D or SR flip-flops and Enable output.

Step 1 of 1 6.088E

Refer to Table 6-58 from the textbook for the behavioral VHDL program for Hamming error correction.

Use the program to write the VHDL program for a Hamming encoder entity with 4-bit data inputs and 7-bit data outputs.

```
library IEEE;

use IEEE.STD_LOGIC_1164.all;

entity Hamming is

port

(

D: in STD_LOGIC_vector(3 downto 0);

O: out STD_LOGIC_VECTOR(6 downto 0)

);

end Hamming;

architecture Hamming_arc of Hamming is

signal P: STD_LOGIC_VECTOR(2 downto 0);

begin

process(D)

begin

P(2) <= D(2) XOR D(3);

P(1) <= D(1) XOR D(2) XOR P(2);

P(0) <= D(2) XOR D(2) XOR P(1);

O <= D(0) & P(0) & P(1) & D(1) & P(2) & D(2) & D(3);

end process;

end Hamming_arc;
```

Thus, the VHDL code for the Hamming encoder entity is provided.

Step 1 of 1

6.090E

Refer to the Figure 6-77 from the text book.

The four step iterative algorithm corresponding to the iterative comparator circuit of Figure 6-77 is,

1. Set EQ_0 to 1 and set *i* to 0.

2. If EQ_i is 1 and X_i and Y_i are equal then set EQ_{i+1} to 1,Else set EQ_{i+1} to 0.

3. Increment *i*.

4. If $i < n$, go to step 2.

Step 1 of 4

The logic symbol of 74×682 is shown in Figure 1:

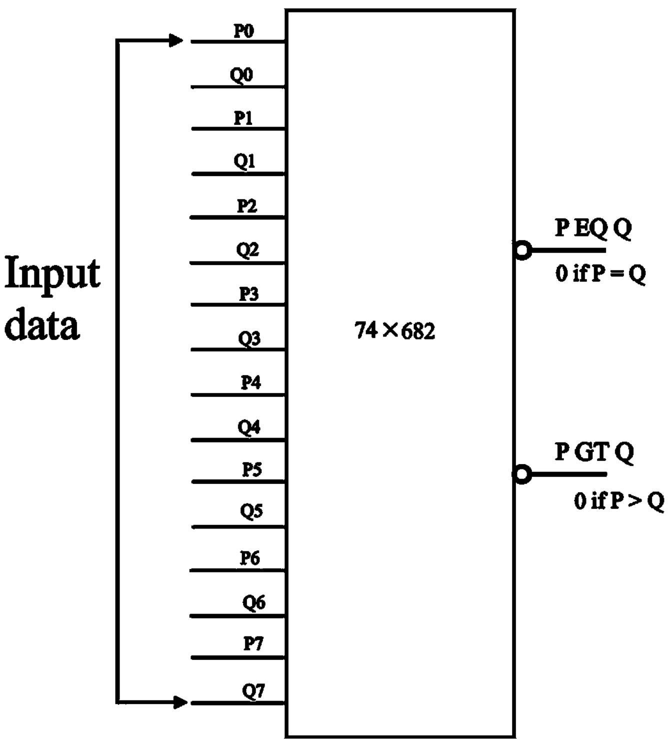


Figure 1

Step 2 of 4

From Figure 1, the inputs of 74×682 compartator are $[P0,P1\cdots P7]$ and $[Q0,Q1\cdots Q7]$.

The output provides $PEQQ=0$ if P is equal to Q and $PGTQ=0$ if P is greater than Q.

In the 64-bit comparator, total 64 input bits are divides in to eight 8-bit segments and eqach segment is connected to each 74×682 compartator. MSB bit is connected to first compartor and LSB bit is connected to last compartaor.

The first input is, $P_{64}P_{63}P_{64}P_{62}P_{61}P_{60}P_{59}P_{58}\ \cdots\ P_7P_6P_5P_4P_3P_2P_1P_0$

The second input is, $Q_{64}Q_{63}Q_{64}Q_{62}Q_{61}Q_{60}Q_{59}Q_{58}\ \cdots\ Q_7Q_6Q_5Q_4Q_3Q_2Q_1Q_0$

Initially compare first 8-bits, if $(P_{64}P_{63}P_{64}P_{62}P_{61}P_{60}P_{59}P_{58})>(Q_{64}Q_{63}Q_{64}Q_{62}Q_{61}Q_{60}Q_{59}Q_{58})$ than $PGTQ7=0$. If $(P_{64}P_{63}P_{64}P_{62}P_{61}P_{60}P_{59}P_{58})=(Q_{64}Q_{63}Q_{64}Q_{62}Q_{61}Q_{60}Q_{59}Q_{58})$ than $PGTQ7=0$.

Same procedure is followed for remaining seven 74×682 compartaors.

Step 3 of 4

The outputs of seven comratos are taken as inputs for 9th comparator $[P0,P1\cdots P7]$ and $[Q0,Q1\cdots Q7]$. Refer to Figure 6-81 in the text book. In this circuit diagram the airthemetic conditions are derived from 74×682 outputs.

For example, If first 64-bits of P are $(P_{64}P_{63}P_{64}P_{62}P_{61}P_{60}P_{59}P_{58}\ \cdots\ P_7P_6P_5P_4P_3P_2P_1P_0)$ greater than Q $(Q_{64}Q_{63}Q_{64}Q_{62}Q_{61}Q_{60}Q_{59}Q_{58}\ \cdots\ Q_7Q_6Q_5Q_4Q_3Q_2Q_1Q_0)$ than $[P0,P1\cdots P7]$ of 9th comparator is high and $[Q0,Q1\cdots Q7]$ of 9th comparator is low.

According to input data of 9th comparator the folloeing conditions are occurred.

• If both inputs $[P0,P1\cdots P7]$ and $[Q0,Q1\cdots Q7]$ of 9th compartaor are equal than the output is $PEQQ=1$.

• If both inputs $[P0,P1\cdots P7]$ and $[Q0,Q1\cdots Q7]$ of 9th compartaor are not equal than the output is $PNEQ=1$.

• If both inputs $[P0,P1\cdots P7]>[Q0,Q1\cdots Q7]$ of 9th compartaor than the output is $PGTQ=1$.

• If both inputs $[P0,P1\cdots P7]<[Q0,Q1\cdots Q7]$ of 9th compartaor than the output is $PLTQ=1$.

• If both inputs $[P0,P1\cdots P7]\geq[Q0,Q1\cdots Q7]$ of 9th compartaor than the output is $PGEQ=1$.

• If both inputs $[P0,P1\cdots P7]\leq[Q0,Q1\cdots Q7]$ of 9th compartaor than the output is $PLEQ=1$.

Step 4 of 4

A 64-bit comparator using nine 74x682s and an additional combinational logic is shown in Figure 2:

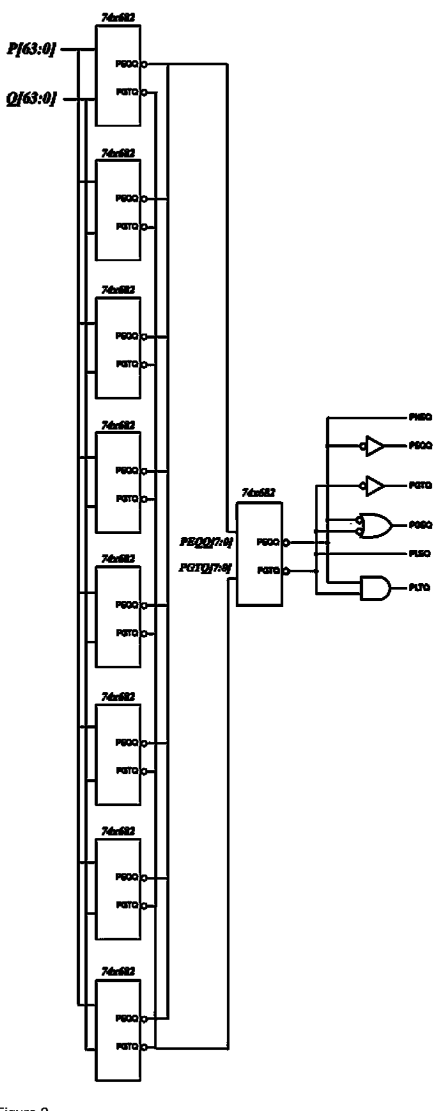


Figure 2

Step 1 of 2

Write the following VHDL program for 8-bit equality and inequaltity checkers.

```
library IEEE;

use IEEE.std_logic_1164.all;

use IEEE.std_logic_unsigned.all;

entity check is

port

(

A, B: in STD_LOGIC_VECTOR (7 downto 0);

EQ, GT: out STD_LOGIC

);

end check;

architecture check _arch of check is

begin

EQ <= '1' when A = B else '0';

GT <= '1' when A > B else '0';

end check _arch;
```

Step 2 of 2

If EQ is 1 then the A and B are equal else they are not equal. Hence, both checkers need the same number of product terms. And even if the device does not have any output polarity control the output is same as the magnitude of the inputs are compared.

Step 1 of 2

6.094E

Write the following VHDL program for the device with functionality of 74x85.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity comparator is
port
(
altbin : in STD_LOGIC;
aeqbin : in STD_LOGIC;
agtbin : in STD_LOGIC;
A : in STD_LOGIC_VECTOR (3 downto 0);
B : in STD_LOGIC_VECTOR (3 downto 0);
agtbout : out STD_LOGIC;
aeqbout : out STD_LOGIC;
altbout : out STD_LOGIC
);
end comparator;
architecture behavioral of comparator is
begin
process(A,B,agtbin,aeqbin,altbin)
begin
if(A=B and agtbin='0' and aeqbin='0' and altbin='0') then
agtbout<='0';
aeqbout<='0';
altbout<='0';
end if;
if(A
agtbout<='0';
aeqbout<='0';
altbout<='1';
end if;
if(A>B and agtbin='0' and aeqbin='0' and altbin='0') then
agtbout<='1';
aeqbout<='0';
altbout<='0';
end if;
if(A=B and agtbin='0' and aeqbin='0' and altbin='0') then
agtbout<='0';
aeqbout<='0';
altbout<='0';
end if;
if(A=B and agtbin='0' and aeqbin='0' and altbin='1') then
agtbout<='0';
aeqbout<='0';
altbout<='1';
end if;
if(A
agtbout<='0';
aeqbout<='0';
altbout<='1';
end if;
if(A>B and agtbin='0' and aeqbin='0' and altbin='1') then
agtbout<='1';
aeqbout<='0';
altbout<='0';
end if;
if(A=B and agtbin='0' and aeqbin='0' and altbin='1') then
agtbout<='0';
aeqbout<='0';
altbout<='1';
end if;
if(A=B and agtbin='0' and aeqbin='1' and altbin='0') then
agtbout<='0';
aeqbout<='1';
altbout<='0';
end if;
if(A
agtbout<='0';
aeqbout<='0';
altbout<='1';
end if;
if(A>B and agtbin='0' and aeqbin='1' and altbin='0') then
agtbout<='1';
aeqbout<='0';
altbout<='0';
end if;
if(A=B and agtbin='0' and aeqbin='1' and altbin='0') then
agtbout<='0';
aeqbout<='1';
altbout<='0';
end if;
if(A=B and agtbin='1' and aeqbin='0' and altbin='0') then
agtbout<='1';
aeqbout<='0';
altbout<='0';
end if;
if(A
agtbout<='0';
aeqbout<='0';
altbout<='1';
end if;
if(A>B and agtbin='1' and aeqbin='0' and altbin='0') then
agtbout<='1';
aeqbout<='0';
```

Step 2 of 2

```
altbout<='0';
end if;
if(A=B and agtbin='1' and aeqbin='0' and altbin='0') then
agtbout<='1';
aeqbout<='0';
altbout<='0';
end if;
end process;
end behavioral;
```

Step 1 of 2

6.095E

The truth table of 74×85 is shown in Table 1:

Cascading inputs	Input conditions			AGTBOUT	AEQBOUT	ALTBOUT
	A>B	A=B	A<B			
AGTBIN=1	X	X	X	1	0	0
AEQBIN=1	1	0	0	1	0	0
	0	1	0	0	1	0
	0	0	1	0	0	1
ALTBIN=1	X	X	X	0	0	1

Table 1

Step 2 of 2

Refer to Figure 6-79 in the text book.

The following are outexpressions of 12-bit comparator.

$$\text{AGTBOUT} = (A > B) + (A = B) \cdot \text{AGTBIN}$$
$$\text{AEQBOUT} = (A = B) \cdot \text{AEQBIN}$$
$$\text{ALTBOUT} = (A < B) + (A = B) \cdot \text{ALTBIN}$$

When the input $\text{AGTBIN} = 1$ than the output AGTBOUT is equal to one. When the input $\text{ALTBIN} = 1$ than the output ALTBOUT is equal to one. When the input $\text{AEQBIN} = 1$ than it checks three conditions as shown in Table 1.

In the design the cascading outputs of the higher order comparator would drive the cascading inputs of the mid-order comparator and the mid-order outputs would drive the low-order inputs. The logical combinational circuit helps to generate ALTBOUT, AEQBOUT and AGTBOUT.

Step 1 of 2

A 24-bit comparator is designed using three 74X682 one 74X27 and one 74X02 as shown in Figure 1.

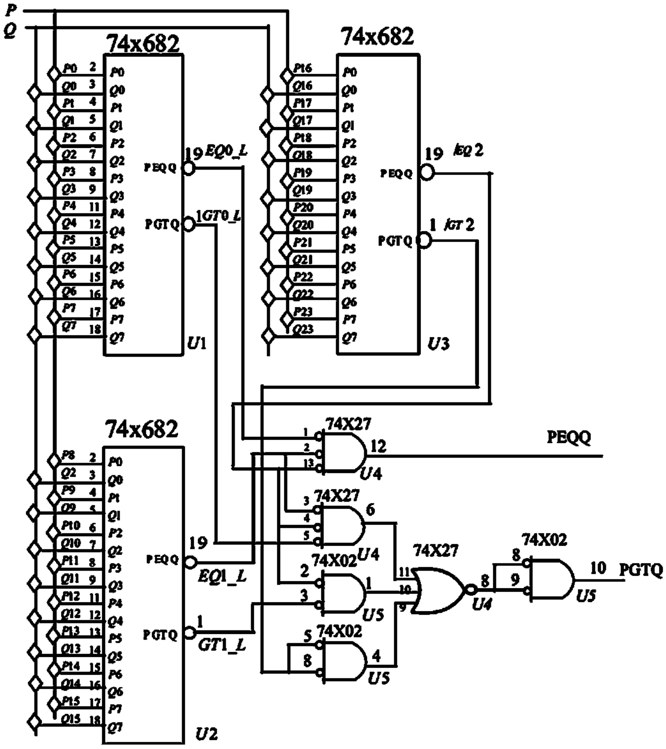


Figure 1

Step 2 of 2

Figure 1 shows that the output is asserted when P = Q or P > Q.

PEQQ is asserted when P = Q,

and PGTQ is asserted when P > Q.

6.097E

Step 1 of 3

Refer to Table 6-3 in the text book.

The obvious solution is to use a 74FCT682, which has a maximum delay of $\mathbf{11\ ns}$ to its $\overline{\mathbf{PEQQ}}$ output. In particular, the 74FCT151 has a delay of only 9 ns from any select input to Y or $\overline{\mathbf{Y}}$. The $\mathbf{74\times138}$ decoder is used to decode the SLOT inputs statically and apply the resulting eight signals to the data inputs of the 74FCT151. Apply GRANT [2–0] to the select inputs of the 74FCT151 and obtain the MATCH_L output (as well as an active-high MATCH) in only $\mathbf{9\ ns}$.

Step 2 of 3

The design of 3-bit equality checker is shown in Figure 1:

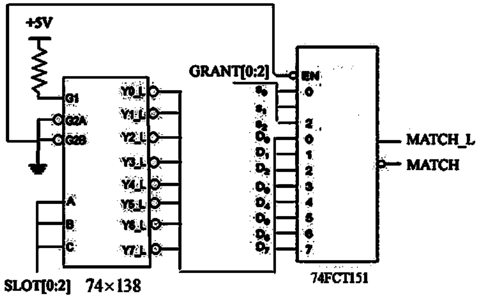


Figure 1

Step 3 of 3

The design of the 3-bit equality checker which saves $\mathbf{2\,ns\quad (11\,ns - 2\,ns = 9\,ns)}$ off the critical path delay is designed.

Step 1 of 2

According to the "worst-case" delay analysis method, the worst case delay through a circuit is computed as the sum of the worst case delays through the individual components. The worst case delay is independent of the transition direction and other circuit conditions.

Refer to Table 6-3 in the text book.

The maximum propagation delay from any A or B bus input to any F bus output of the 16-bit carry look ahead adder of can be calculated using the following values.

$$t_{pHL} = 23\,\text{ns}$$
$$t_{pLH} = 30\,\text{ns}$$

Refer to Figure 6-93 in the text book. The circuit diagram consists of **four** 74x381 components.

The maximum propagation delay is,

$$t_{p(74x381)} = 4t_{pLH(74LS381)} + 4t_{pHL(74LS381)}$$

Substitute $30\,\text{ns}$ for t_{pLH} and $23\,\text{ns}$ for t_{pHL}.

$$\begin{aligned} t_{p(74x381)} &= 4(30\,\text{ns}) + 4(23\,\text{ns}) \\ &= 120\,\text{ns} + 92\,\text{ns} \\ &= 212\,\text{ns} \end{aligned}$$

Step 2 of 2

Refer to Figure 6-93 in the text book. The output C8 of 74x182 is connected to CIN of 74x381.

The propagation delays of 74x182 are,

$$t_{pHL} = 10.5\,\text{ns}$$
$$t_{pLH} = 10\,\text{ns}$$

The IC 74x182 shares output with all the inputs of the 74x381.

The propagation delay is,

$$t_{p(74x182)} = 4t_{pLH(74x182)} + 4t_{pHL(74x182)}$$

Substitute $10.5\,\text{ns}$ for t_{pHL} and $10\,\text{ns}$ for t_{pLH}.

$$\begin{aligned} t_{p(74x182)} &= 4(10\,\text{ns}) + 4(10.5\,\text{ns}) \\ &= 40\,\text{ns} + 42\,\text{ns} \\ &= 82\,\text{ns} \end{aligned}$$

The worst case delay of A, B buses to F bus of the circuit in the Figure 6-93 is sum of delays of propagation delays of 74×182 and 74×381.

$$\begin{aligned} t_{(A,B\to F)} &= t_{p(74x381)} + t_{p(74x182)} \\ &= 212\,\text{ns} + 82\,\text{ns} \\ &= 294\,\text{ns} \end{aligned}$$

Therefore, the maximum worst case delay from A, B buses to F bus is $\boxed{294\,\text{ns}}$.

Step 1 of 1 **6.100E**

Refer to Figure 6-87 in the text book.

Starting with the logic diagram for the 74x283 in, the logic expression for S3 output in terms of the inputs can be calculated as follows:

The sum expression is,

$S_i = x_i \oplus y_i \oplus c_i$ ...... (1)

Consider $hs_i = x_i \oplus y_i$

The modified expression is,

$S_i = hs_i \oplus c_i$ ...... (2)

Substitute 3 for *i* in equation (1).

$$\begin{aligned} S_3 &= x_3 \oplus y_3 \oplus c_3 \\ &= hs_3 \oplus c_3 \\ &= hs_3 \oplus \left[p_3 \cdot (g_2 + p_1) \cdot (g_2 + g_1 + p_0) \cdot (g_0 + g_1 + g_2 + c_0) \right] \\ &= hs_3 \oplus \left[p_3 \cdot (g_2 + p_1) \cdot (g_2 + g_1 + p_0) \cdot (g_0 + g_1 + g_2 + 0) \right] \qquad (\text{since } c_0 = 0) \\ &= (x_3 \oplus y_3) \oplus \left[\begin{array}{l} (x_3 + y_3) \cdot (x_2 y_2 + (x_1 + y_1)) \cdot (x_2 y_2 + x_1 y_1 + (x_0 + y_0)) \cdot \\ (x_0 y_0 + x_1 y_1 + x_2 y_2) \end{array} \right] \end{aligned}$$

Therefore, the expression of S3 output in terms of the inputs is

$$\boxed{(x_3 \oplus y_3) \oplus \left[\begin{array}{l} (x_3 + y_3) \cdot (x_2 y_2 + (x_1 + y_1)) \cdot (x_2 y_2 + x_1 y_1 + (x_0 + y_0)) \cdot \\ (x_0 y_0 + x_1 y_1 + x_2 y_2) \end{array} \right]}.$$

Step 1 of 1 **6.102E**

Refer to Figure 6-90 from the textbook for the logic symbol of **74×181** 4-bit ALU.

The following is the VHDl program for the ALU:

```
library ieee;
use ieee.std_logic_1164.all;
entity ALU74181 is
port(
a,b,op: in STD_LOGIC_VECTOR(3 downto 0);
cin, m:in STD_LOGIC;
dout: out STD_LOGIC_VECTOR(3 downto 0);
cout, eq:out STD_LOGIC
);
end ALU74181;
architecture alubeh of ALU74181 is
signal ta, tb: STD_LOGIC_VECTOR(3 downto 0);
signal aux: STD_LOGIC;
begin
process(m,cin,op,a,b)
begin
if(m='1') then
case op is
when "0000" => dout <= not(a);
when "0001" => dout <= not(a and b);
when "0010" => dout <= not(a) or b;
when "0011" => dout <= "0001";
when "0100" => dout <= not(a or b);
when "0101" => dout <= not(b);
when "0110" => dout <= not(a xor b);
when "0111" => dout <= a or not(b);
when "1000" => dout <= not(a) and b;
when "1001" => dout <= a xor b;
when "1010" => dout <= b;
when "1011" => dout <= a or b;
when "1100" => dout <= "0000";
when "1101" => dout <= a and not(b);
when "1110" => dout <= a and b;
when "1111" => dout <= a;
when others => null;
end case;
else
case op is
when "0000" => ta<=a; tb<="1111";
when "0001" => ta<=a and b; tb<="1111";
when "0010" => ta<=a and not(b); tb<="1111";
when "0011" => ta<="1111"; tb<="0000";
when "0100" => ta<=a; tb<=a or not(b);
when "0101" => ta<=a and b; tb<=a or not(b);
when "0110" => ta<=a;tb<=not(b);
when "0111" => ta <= a or not(b);tb<="0000";
when "1000" => ta<=a;tb<=a or b;
when "1001" => ta<=a;tb<=b;
when "1010" => ta<=a and not(b);tb<=a or b;
when "1011" => ta<=a or b;tb<="0000";
when "1100" => ta<=a;aux<='0';
when "1101" => ta<=a and b;tb<=a;
when "1110" => ta<=a and not(b);tb<=a;
when "1111" => ta<=a;tb<="0000";
when others => null;
end case;
end if;
if(a=b) then eq<='0';
else eq<='1';
end if;
end process;
end alubeh;
```

Hence, the VHDL program for the ALU is provided.

6.104E

Step 1 of 3

Write the Verilog program for the sum and differences.

```
module alu74x138(s,a,b,cin,f,g_l,p_l);
input [2:0] s;
input [7:0] a,b;
input cin;
output [7:0] f;
reg [7:0] c; /* variable c */
output g_l,p_l;
reg [7:0] f;
reg g_l,p_l,gen,pro;
reg [7:0] g,p;
integer i;
always @ (s or a or b or c or g_l or p_l or gen or pro)
begin
c[0]=1;
for (i=0; i<= 7; i=i+1) begin
g[i] = a[i] & b[i];
p[i]= a[i] | b[i];
end
gen = g[0]; pro=p[0];
for (i=1;i<= 7;i=i+1) begin
gen = g[i] ^ (gen & p[i]);
pro=p[i] & pro;
if (gen | (pro & cin))
c[i]= 1;
else
c[i]=0;
end
g_l = ~ gen ;
p_l= ~pro;
case (s)
3'd0 : f = 8'b0;
3'd1: f=b-a-1+cin;
3'd2: f=a-b-1+cin;
3'd3: f=a^b^c; /* addition */
3'd4: f= ~a^b^c; /* subtraction */
3'd4: f= a|b;
3'd4: f= a&b;
3'd7:f=8'b11111111;
default : f = 8'b0;
endcase
end
endmodule
```

Step 2 of 3

Write the test Bench for the Verilog code.

```
module sgewrqg;
// Inputs
reg [2:0] s;
reg [7:0] a;
reg [7:0] b;
reg cin;
// Outputs
wire [7:0] f;
wire g_l;
wire p_l;
// Instantiate the Unit Under Test (UUT)
alu74x138 uut (
.s(s),
.a(a),
.b(b),
.cin(cin),
.f(f),
.g_l(g_l),
.p_l(p_l)
);
initial begin
$monitor ($time, "a=%b, b=%b, s=%b, cin=%b, f=%b, g_l=%b, p_l=%b ", a, b, s, cin, f, g_l, p_l);
#100 a=3;b=6;s=1;cin=1;
# 200 $display($time);
#100 a=5;b=9;s=3;cin=0;
# 200 $display($time);
#100 a=7;b=8;s=4;cin=1;
# 200 $display($time);
#100 a=4;b=11;s=2;cin=1;
# 200 $display($time);
end
endmodule
```

Step 3 of 3

Draw the simulated output.

Figure 1

Step 1 of 2

6.105E

Refer to Figure 6-96 in the textbook.

Refer to Table-6.3 in the textbook.

From Table-6.3 in the textbook and the worst-case delay analysis method, the worst case delay through a circuit is computed as the sum of the worst case delays through the individual components, independent of the transition direction and other circuit conditions.

The maximum propagation delay from any adder input to its sum output is twice as long as delay to the carry output.

The reverse relation can be calculated as follows:

$t_{pd(COUT)} = 2t_{pd(SOUT)}$ ...... (1)

Consider the following value from the Table 6.3 of part 283 in the textbook.

$$t_{pLH} = 24 \text{ ns}$$
$$t_{pHL} = 24 \text{ ns}$$

Step 2 of 2

Calculate the total delay of S_{out} .

$$t_{pd(SOUT)} = t_{pLH} + t_{pHL}$$

Substitute 24 ns for t_{pLH} and 24 ns for t_{pHL} .

$$\begin{aligned} t_{pd(SOUT)} &= 24 \text{ ns} + 24 \text{ ns} \\ &= 48 \text{ ns} \end{aligned}$$

The propagation delay of the adder is the propagation delay of C_{OUT} .

Calculate the propagation delay of C_{OUT} .

$$\begin{aligned} t_{pd} &= t_{pd(COUT)} \\ &= 2t_{pd(SOUT)} \end{aligned}$$

Substitute 48 ns for $t_{pd(SOUT)}$.

$$\begin{aligned} t_{pd} &= 2(48 \text{ ns}) \\ &= 96 \text{ ns} \end{aligned}$$

From Figure 6-96, the worst case delay path goes through 20 adders.

Calculate the worst case delay.

$$\begin{aligned} t_d &= 20t_{pd} \\ &= 20(96 \text{ ns}) \\ &= 1920 \text{ ns} \end{aligned}$$

Therefore, the worst case delay of the circuit in the Figure 6-96 is $\boxed{1920 \text{ ns}}$.

Step 1 of 1 **7.01DP**

Metastability means a stage or a state that will be stable only for few minutes or finite time before it reached a most stable state.

When we think about sports like golf and football, ball can reach metastable state before it reaches the most stable state of goal point.

Step 1 of 1

From the title song in Devo's freedom of choice album, the lines that refers to metastability is as follows,

"Then if you got it you don't want it

Seems to be the rule of thumb

Don't be tricked by what you see

You got two ways to go"

The above lines refers the metastable state and it is also refers that to obtain stable state two ways are there.

Step 1 of 1

In a Lovin' spoonful music, metastability is stated in the album "Do you believe in magic in a young girl's heart".

The lines are as follows,

"If you believe in magic don't bother to choose

If it's jug band music or rhythm and blues"

The above line discussed about the metastability state of mind to choose a stable state of either a jug band music or rhythm and blues.

Step 1 of 3

An SR latch of the type shown in Figure 7-5 in textbook has the following assumptions.

(i) Propagation delay of NOR gate is $t_{N_1} = t_{N_2} = 10\text{ ns}$.

(ii) Input & output of SR latch rise and fall time is consider to be zero.

(iii) Time division is 10 ns

The input waveforms are shown in Figure X7.5 in textbook.

In general the rise time and fall time for SR latch with two NOR gate can be calculated using propagation delay of two NOR gates N_1 and N_2 .

Rise time is given as follows,

$t_Q = t_{N_1} + t_{N_2}$ and $t_{\bar{Q}} = t_{N_1} + t_{N_2}$

Fall time is given as follows,

$t_Q = t_{N_2}$ and $t_{\bar{Q}} = t_{N_1}$

It is also noted that when both inputs (S and R) are changed simultaneously, then the output is ambiguous. The output will oscillate from 1 to 0 and 0 to 1.

Step 2 of 3

The truth table for SR latch is,

S	*R*	*Q*	*QN*
0	0	last Q	last QN
0	1	0	1
1	0	1	0
1	1	0	0

Table 1

Step 3 of 3

Considering the NOR gate delay and truth table, the output waveform for SR latch is given as follows,

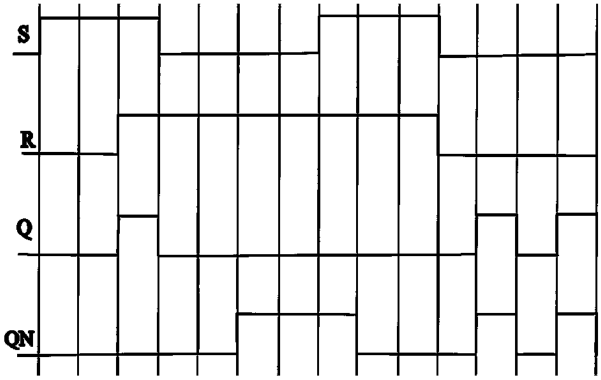


Figure 1: Output waveform for SR latch.

Step 1 of 4

To convert T flip flop to JK flip flop, find the input for T flip flop from the behavior of JK flip flop. This can be done by present state and next state output of JK flips flop as follows.

J	K	Qp	Qp+1	T
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	1

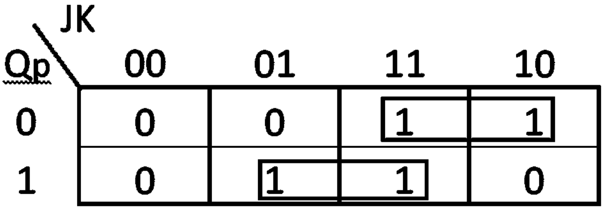
Table 1: Conversion table for D to JK flip flop.

Step 2 of 4

From Table 1, the next state (Qp+1) is calculated from JK flip flop input and present state (Qp). Then input to T flip flop is calculated from present state (Qp) and next state (Qp+1). The characteristic of T flip flop is such that when the input is 1, the output will toggle. Thus if the present state and next state is having the same value then $\mathbf{T=0}$ and if it differs then $\mathbf{T=1}$.

Step 3 of 4

From the above table, the equation for T flip flop can be calculated using Karnaugh map as follows,



Qp \ JK	00	01	11	10
0	0	0	1	1
1	0	1	1	0

Figure 1: Kamaugh map entry for conversion of D to JK flip flop

From Karnaugh map, the equation for T can be written as follows,

$$\mathbf{T = K \cdot Q_p + J \cdot Q_p{}'}$$

Step 4 of 4

Thus the diagram can be drawn as shown in Figure 2.

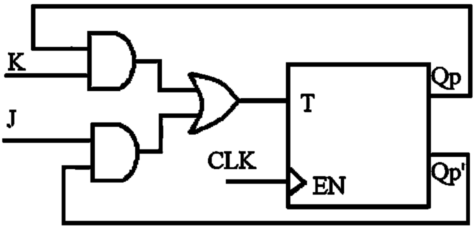


Figure 2: Conversion of T flip flop into JK flip flop

Step 1 of 1 7.08DP

The operation of flip-flop and latch is same and the circuits are also almost similar but the only differnce is clock signal. Flipflops need clock signal but latch doesn't need it. Flip flop can be designed by using latch is giving a clock signal to the latch but latch cannot be created using flip-flop because a flip-flop need an external clock to operate where as latch require an enable which must be always at level '1'.

Hence, one cannot drive a 74x74 dual edge triggered D flip-flop with the inputs of a latch. So designing a S-R latch using a single 74x74 positive edge triggered flip-flop is not possible.

Step 1 of 4 7.09DP

Yes, it is possible to build a master slave SR flip-flop using a single 74X74 positive edge triggred flip-flop and a combinational circuit.

The truth table is same as the SR flip-flop except the way it is clocked is quite different. S, R applied directly to the Master flip-flop, the outputs of master flip-flop acts as the control inputs to the slave flipflop. The inverted clock pulse of the slave makes it responsible to respond to the control signals produced by the master flip-flop.

Step 2 of 4

The truth table of the SR flip-flop is as follows:

Table 1

External Inputs	Present State	Next State	Flip-flop Input	
S	R	Q_n	Q_{n+1}	D
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	X	X
1	1	1	X	X

Step 3 of 4

Convert D flip flop into SR flip-flop and draw the Karnaugh map for the input, D

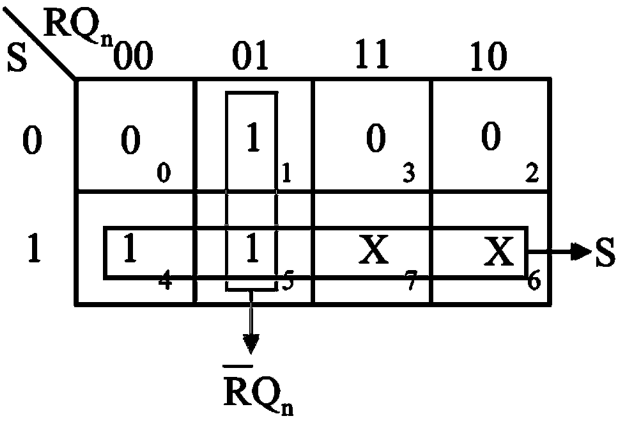


Figure 1

The relation between between the D flip flop and SR flip-flop is,

$$D = S + \overline{R}Q_n$$

Step 4 of 4

Draw the following designed circuit:

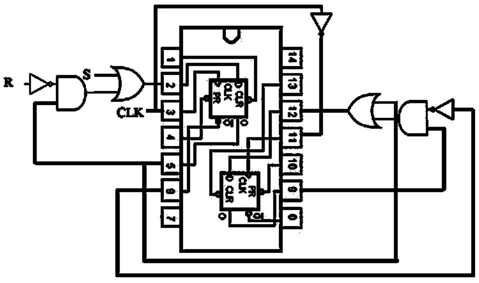


Figure 2

Therefore, designed the Master Slave SR flip-flop using single 74X74 and a combinational logic.

Step 1 of 5 **7.11DP**

To Convert JK flip flop to D flip flop, find the input for JK flip flop from the behavior of D flip flop. This can be done by present state and next state output of D flips flop as follows.

D	Qp	Qp+1	J	K
0	0	0	0	X
0	1	0	X	1
1	0	1	1	X
1	1	1	X	0

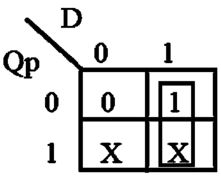
Table 1: Conversion table for JK to D flip flop.

Step 2 of 5

From Table 1, the next state (Qp+1) is calculated from D flip flop input and present state (Qp). Then input to JK flip flop is calculated from present state (Qp) and next state (Qp+1). The characteristic of JK flip flop is such that when the input of J = 0 or K = 1, then output will be 0. According to the values of the present state and next state, the values of J and K applied, where ‘X’ is don’t care. When the input of J = 1 or K = 0, then output will be 1. According to the values of the present state and next state, the values of J and K applied, where ‘X’ is don’t care.

Step 3 of 5

From Table 1, the equation for JK flip flop can be calculated using Karnaugh map as follows.



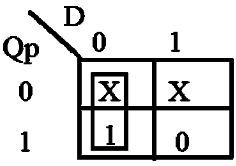


Figure 1: Kamaugh map entries for conversion of JK to D flip flop

Step 4 of 5

From Karnaugh map, the equation for J and K can be written as follows.

$$J = D$$
$$K = D'$$

Step 5 of 5

Thus the diagram can be drawn as shown in Figure 2.

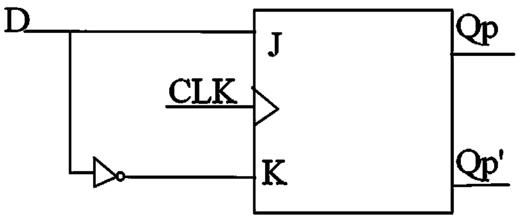


Figure 2: Conversion of JK flip flop into D flip flop

Step 1 of 7

The clocked synchronous state machine obtained by changing AND gates to NAND gates and OR gates to NOR gates in Figure X7.12 in text book is shown in Figure 1.

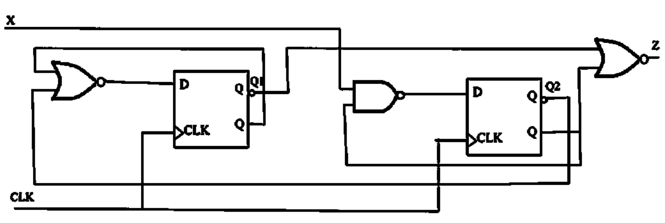


Figure 1: Clocked synchronous state machine.

Step 2 of 7

The excitation equation for the state machine can be given as follows,

$$D_1 = (Q_1 + Q_2')'$$
$$= Q_1' \cdot Q_2$$

$$D_2 = (Q_2 \cdot X)'$$
$$= Q_2' + X'$$

$$Z = (Q_1' + Q_2)'$$
$$= Q_1 \cdot Q_2'$$

Step 3 of 7

The next state for D flip flop is same as input, thus the equation for the next state (Q*) is given as follows,

$$Q_1^{*} = D_1$$
$$= Q_1' \cdot Q_2$$

$$Q_2^{*} = D_2$$
$$= Q_2' + X'$$

Step 4 of 7

The excitation / transition table is given by as follows,

Q1	Q2	X	Z	
		0	1	
0	0	01	01	0
0	1	11	10	0
1	0	01	01	1
1	1	01	00	0
		Q1* Q2*		

Table 1: Excitation / transitions table for Figure 1

Step 5 of 7

The state table name is taken as A-D according to the values of Q1 and Q2 from 00-11.

The state table for above transition table is,

S	X	Z	
	0	1	
A	B	B	0
B	D	C	0
C	B	B	1
D	B	B	0

Table 2: State/Output table for Figure 1

Step 6 of 7

The original clocked synchronous state machine is shown in Figure X7.12 in text book. The excitation equation for this state machine can be given as follows,

$$D_1 = Q_1' + Q_2$$
$$D_2 = Q_2' \cdot X$$
$$Z = Q_1 + Q_2'$$

The next state for D flip flop is same as input, thus the equation for the next state (Q*) is given as follows,

$$Q_1^{*} = D_1$$
$$= Q_1' + Q_2$$

And

$$Q_2^{*} = D_2$$
$$= Q_2' \cdot X$$

The excitation / transition table is as follows.

Q1	Q2	X	Z	
		0	1	
0	0	10	11	1
0	1	10	10	0
1	0	00	01	1
1	1	10	10	1
		Q1* Q2*		

Table 3: Excitation / transitions table

Step 7 of 7

Consider the state table names as A-D according to the values of Q1 and Q2 from 00-11. The state/output table for the transition table is as follows.

S	X	Z	
	0	1	
A	C	D	1
B	C	C	0
C	A	B	1
D	C	C	1

Table 4: State/Output table

When we compare tables 2 and 4, everything is complemented due to complemented form of gates and the outputs of flip flop.

Step 1 of 2

7.15DP

Consider the state table / output table shown in Table 7-9 of text book.

XY					
S	**00**	**01**	**11**	**10**	**Z**
S0	S0	S1	S2	S1	1
S1	S1	S2	S3	S2	0
S2	S2	S3	S0	S3	0
S3	S3	S0	S1	S0	0
	S*				

Table 1: State table for 1's counting system.

Step 2 of 2

The state diagram which indicates the states, the transitions between states and the outputs of the state table is shown in Figure 1.

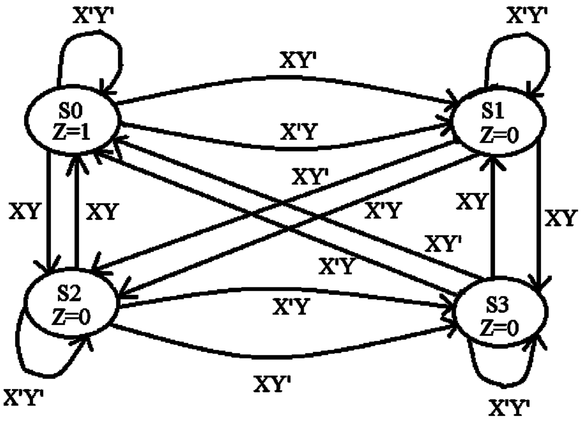


Figure 1: State diagram for the table 1

Step 1 of 2

7.17DP

Consider the state diagram in Figure X7.17 in textbook.

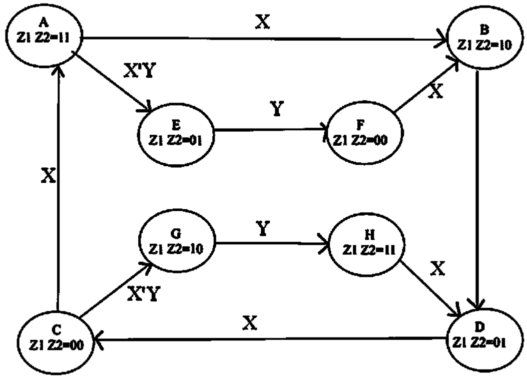


Figure 1: State diagram

Step 2 of 2

The state diagram in Figure 1 is drawn with the convection that the state does not change except for input conditions that are explicitly shown. Depending on the states, the transitions between states and the outputs indicated in the state diagram, the state table is drawn as shown in Table 1.

S	XY				
	00	01	11	10	Z1Z2
A	A	E	B	B	11
B	B	B	D	D	10
C	C	G	A	A	00
D	D	D	C	C	01
E	E	F	F	E	01
F	F	F	B	B	00
G	G	H	H	G	10
H	H	H	D	D	11

Table 1: State table for figure 1

Step 1 of 4

Refer to the Figure X7.19 from the text book.

It is clear that Figure X7.19 represents the clocked synchronous state machine.

The excitation equation for the state machine shown in Figure X7.19 is as follows:

$$D_1 = X$$
$$D_2 = (Q_1 + Y)\cdot Q_3'$$
$$D_3 = (Q_2'\cdot Y) + Q_1'$$

Step 2 of 4

The next state for D flip flop is same as input.

Consider that the next state is represented by Q*.

Thus the equation for the next state (Q*) is as follows.

$$\begin{aligned} Q_1^* &= D_1 \\ &= X \\ Q_2^* &= D_2 \\ &= (Q_1 + Y)\cdot Q_3' \\ Q_3^* &= D_3 \\ &= (Q_2'\cdot Y) + Q_1' \end{aligned}$$

Step 3 of 4

Write the excitation or the transition table as shown in Table 1.

Present state	Next state			
	XY			
Q1Q2Q3	00	01	11	10
000	001	011	111	101
001	001	001	101	101
010	001	011	111	101
011	001	001	101	101
100	010	011	111	110
101	000	001	101	100
110	010	010	110	110
111	000	000	100	100
	Q1*Q2*Q3*			

Table 1: Excitation / transitions table for figure 1.

Step 4 of 4

Use the state table name is as A-D according to the values of Q1 and Q2 from 00-11.

The state table for transition table (Table 1) is as shown in Table 2.

Present state	Next state			
	XY			
Q1Q2Q3	00	01	11	10
A	B	D	H	F
B	B	B	F	F
C	B	D	H	F
D	B	B	F	F
E	C	D	H	G
F	A	B	F	E
G	C	C	G	G
H	A	A	E	E
	Q1*Q2*Q3*			

Table 2: State table for Figure 1

Thus, the excitations equations, excitation table and the state table are obtained for the Figure X7.19.

Step 1 of 4

If all of the input combinations are covered, the logical sum of the expressions on all the transitions leaving a state must be 1. The sum is 0 for the combinations that are uncovered. For double-covered input combinations, we look at all possible pairs of transitions leaving a state. The product of a pair of transition equations is 1 for any double-covered input combinations.

(a)

Refer Figure X7.21 (a) from the textbook.

At state D, $Y = 0$ is uncovered.

Step 2 of 4

(b)

Refer Figure X7.21 (b) from the textbook.

At state A, $(X + Z') = 0$ is uncovered. At state B, $W = 1$ is double-covered; $(W + X) = 0$ is uncovered. At state C, $(W + X + Y + Z) = 0$ is uncovered; $(W \cdot X + W \cdot Y + Z \cdot X + Z \cdot Y) = 1$ is double covered. State D, $(X \cdot Y + W \cdot Z + W \cdot X' \cdot Z) = 0$ is uncovered.

Step 3 of 4

(c)

Refer Figure X7.21 (c) from the textbook.

At state D, $(XY + XY') = 0$ is uncovered.

Step 4 of 4

(d)

Refer Figure X7.21 (d) from the textbook.

At state C, $(Z + W \cdot Z' + W' \cdot X' \cdot Y') = 0$ is uncovered.

Step 1 of 4 **7.22DP**

Refer Figure 7-58 from the textbook.

Using the figure, the transition list is obtained as shown in table 1.

S	$LALBLC$	$RARB$ RC	Tranisition equation	S^*	LA^*LB^* LC^*	RA^*RB^* RC^*
IDLE	000	000	$(LEFT + RIGHT + HAZ)'$	IDLE	000	000
IDLE	000	000	$LEFT \cdot HAZ' \cdot RIGHT'$ $HAZ + LEFT \cdot RIGHT$	L1	001	000
IDLE	000	000	$RIGHT \cdot HAZ' \cdot LEFT'$ HAZ'	LR3	111	111
IDLE	000	000	HAZ	R1	000	100
L1	001	000	HAZ'	L2	011	000
L1	001	000	HAZ	LR3	111	111
L2	011	000	1	L3	111	000
L2	011	000	HAZ'	LR3	111	111
L3	111	000	HAZ	IDLE	000	000
R1	000	100	HAZ'	R2	000	110
R1	000	100	HAZ	LR3	111	111
R2	000	110	1	R3	000	111
R2	000	110	1	LR3	111	111
R3	000	111		IDLE	000	000
LR3	111	111		IDLE	000	000

Table 1

Step 2 of 4

Using the $V^* = \sum p-terms$ where $V^* = 1$ and the transition table in Table 1, the transition equations are as follows:

$$LA^* = \begin{pmatrix} LA' \cdot LB' \cdot LC' \cdot (HAZ + LEFT \cdot RIGHT) \\ +LA' \cdot LB' \cdot HAZ \\ +LA' \cdot LB \cdot LC \end{pmatrix}$$

$$LB^* = \begin{pmatrix} LA' \cdot LB' \cdot LC' \cdot (HAZ + LEFT \cdot RIGHT) \\ +LA' \cdot LB' \cdot LC \cdot HAZ' \\ +LA' \cdot LB' \cdot HAZ \\ +LA' \cdot LB \cdot LC \end{pmatrix}$$

$$LC^* = \begin{pmatrix} LA' \cdot LB' \cdot LC' \cdot (HAZ' \cdot LEFT \cdot RIGHT') \\ +LA' \cdot LB' \cdot LC' \cdot (HAZ + LEFT \cdot RIGHT) \\ +LA' \cdot LB' \cdot LC \cdot HAZ' \\ +LA' \cdot LB' \cdot HAZ \\ +LA' \cdot LB \cdot LC \end{pmatrix}$$

$$RA^* = \begin{pmatrix} RA' \cdot RB' \cdot RC' \cdot (RIGHT \cdot HAZ' \cdot LEFT') \\ +RA' \cdot RB' \cdot RC' \cdot (HAZ + LEFT \cdot RIGHT) \\ +RA \cdot RB' \cdot RC' \\ +RA \cdot RB \cdot RC' \end{pmatrix}$$

$$RB^* = \begin{pmatrix} RA' \cdot RB' \cdot RC' \cdot (HAZ + LEFT \cdot RIGHT) \\ +RA \cdot RB' \cdot RC' \\ +RA \cdot RB \cdot RC' \end{pmatrix}$$

$$RC^* = \begin{pmatrix} RA' \cdot RB' \cdot RC' \cdot (HAZ + LEFT \cdot RIGHT) \\ +RA \cdot RB' \cdot RC' \cdot HAZ \\ +RA \cdot RB \cdot RC' \end{pmatrix}$$

Step 3 of 4

The design can be done using two 74x138s and a combinational logic as shown in Figure 1.

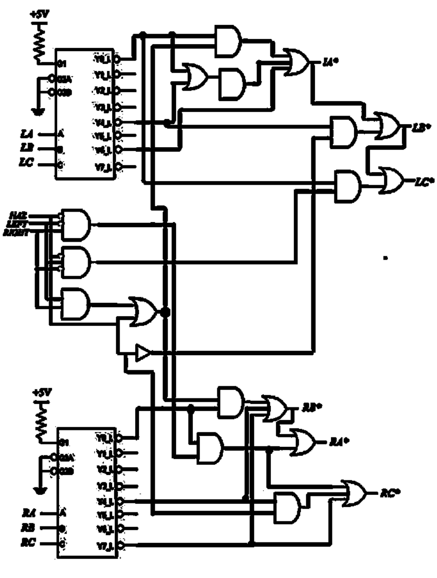


Figure 1

Step 4 of 4

In the Figure 1 he Top 74x138 is used decode the states of *LA*, *LB* and *LC*. The Bottom 74x138 is used decode the states of *RA*, *RB* and *RC*. The combinational logic is used to develop the functional requirements and are buit based on the derived transition equtions.

7.23DP

Step 1 of 2

Refer Table 7-15 in the textbook.

If the other two unused states were considered as don't-cares then the three variable map is completely filled resulting in

$$Q2^{*} = 1$$

It makes the guessing game machine run with a combinational logic if we consider the unused states as don't-cares, as the value of $Q2^{*}$ is always 1 irrespective of the inputs and the transition expressions as shown in the Table 7-15.

Step 2 of 2

If always $Q2^{*} = 1$ in the guessing game machine then the machine will never go back to game it oscillates in the states SOK and SERR, which makes the machine useless.

7.24DP

Step 1 of 2

If the LEFT and RIGHT are asserted simultaneously during a turn then the state will immediately go to IDLE. The modified state diagram is as shown in Figure 1.

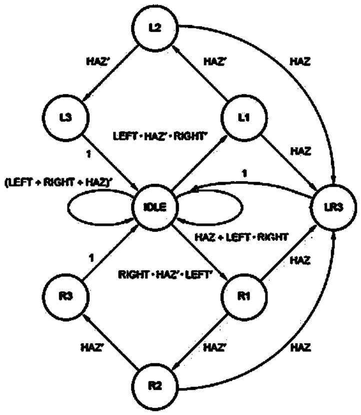


Figure 1

Step 2 of 2

The transition state for the above state diagram is as shown in Table 1.

S	$Q2Q1Q0$	Transition Expression	S^*	$Q2^*Q1^*Q0^*$
IDLE	000	$(\mathrm{LEFT}+\mathrm{RIGHT}+\mathrm{HAZ})'$	IDLE	000
IDLE	000	$\mathrm{LEFT}\cdot\mathrm{RIGHT}'\cdot\mathrm{HAZ}'$	L1	001
IDLE	000	$\mathrm{LEFT}\cdot\mathrm{RIGHT}+\mathrm{HAZ}$	IDLE	000
IDLE	000	$\mathrm{LEFT}'\cdot\mathrm{RIGHT}\cdot\mathrm{HAZ}'$	R1	101
L1	001	HAZ'	L2	011
L1	001	HAZ	LR3	100
L2	011	HAZ'	L3	010
L2	011	HAZ	LR3	100
L3	010	1	IDLE	000
R1	101	HAZ'	R2	111
R1	101	HAZ	LR3	100
R2	111	HAZ'	R3	110
R2	111	HAZ	LR3	100
R3	110	1	IDLE	000
LR3	100	1	IDLE	000

Table 1

From the transition list, we can say that if the LEFT and RIGHT are asserted simultaneously during a turn then the state will immediately go to IDLE state as in 1st and 3rd rows.

Step 1 of 8

7.25DP

The state table or the output table for ones-counting machine is shown in Table 1.

S	XY				
	00	01	11	10	Z
A	A	B	C	B	1
B	B	C	D	C	0
C	C	D	A	D	0
D	D	A	B	A	0
	S*				

Table 1: State and output table for ones- counting machine.

Step 2 of 8

The ABEL program for ones- counting machine state table is as follows.

```
module CM1
title 'one's counting machine'
"Input and Output pins
Clk,X,Y pin;
Q1,Q2 pin istype 'reg';
2 pin istype 'com';
"Definitions
S=[Q1,Q2]; "state variables
A=[0,0]; "state encoding
B=[0,1];
C=[1,0];
D=[1,1];
State_diagram S "state defining
State A: 2=1;
if (X==0) &(Y==0) then A
else if (X==0)&(Y==1) then B
else if (X==1)&(Y==1) then C
else if (X==1)&(Y==0) then B;
State B: Z=0;
if (X==0) &(Y==0) then B
else if (X==0)&(Y==1) then C
else if (X==1)&(Y==1) then D
else if (X==1)&(Y==0) then C;
State C: Z=0;
if (X==0) &(Y==0) then C
else if (X==0)&(Y==1) then D
else if (X==1)&(Y==1) then A
else if (X==1)&(Y==0) then D;
State D: Z=0;
if (X==0) &(Y==0) then D
else if (X==0)&(Y==1) then A
else if (X==1)&(Y==1) then B
else if (X==1)&(Y==0) then A;
equations
S.Clk = Clk; S.OE=1;
end CM1
```

Step 3 of 8

The VHDL program for one's Counting machine state table is as follows.

```
library IEEE;
use IEEE.std_logic_1164.all;
–ENTITY DECLARATION
entity CM1 is
port(CLK,X,Y : in STD_LOGIC;
Z: out STD_LOGIC);
end;
--ARCHITECTURE DECLARATION
architecture CM1_arch of CM1 is
type state_type is (A,B,C,D); -- state declaration
signal sreg, snext: state_type;
-- current state and next state
begin
```

Step 4 of 8

```
process(CLK) – state memory
begin
if CLK'event and CLK='1' then
sreg <= snext;
end if;
end process;
process (X,Y,sreg) – next state logic
begin
case sreg is
when A => if X='0' and Y='0' then
```

Step 5 of 8

```
snext<= A;
elsif X='0' and Y='1' then
snext<= B;
elsif X='1' and Y='1' then
snext<= C;
elsif X='1' and Y='0' then
snext<= B;
end if;
when B => if X='0' and Y='0' then
snext<= B;
elsif X='0' and Y='1' then
snext<= C;
elsif X='1' and Y='1' then
snext<= D;
elsif X='1' and Y='0' then
snext<= C;
end if;
```

Step 6 of 8

```
when C => if X='0' and Y='0' then
snext<= C;
elsif X='0' and Y='1' then
snext<= D;
elsif X='1' and Y='1' then
snext<= A;
elsif X='1' and Y='0' then
snext<= D;
end if;
when D => if X='0' and Y='0' then
snext<= D;
elsif X='0' and Y='1' then
snext<= A;
elsif X='1' and Y='1' then
snext<= B;
elsif X='1' and Y='0' then
snext<= A;
end if;
when others=> snext<= A;
end case;
end process;
with sreg select
Z<='1' when A,
'0' when others;
end CM1_arch;
```

Step 7 of 8

The Verilog program for ones-Counting machine state table is as follows.

```
module CM1(CLK,X,Y,Z) // module declaration
input CLK,X,Y;
output Z;
reg Z;
reg[1:0] sreg,snext;
parameter[1:0] A=2'b00, // state declaration
B=2'b01,
C=2'b10,
D=2'b11;
always@( posedge CLK)
sreg<= snext;
always@(X,Y,sreg) begin //state definition
case (sreg)
A: if((X==0)&(Y==0)) snext=A;
else if((X==0)&(Y==1)) snext=B;
else if((X==1)&(Y==1)) snext=C;
else if((X==1)&(Y==0)) snext=D;
B: if((X==0)&(Y==0)) snext=B;
else if((X==0)&(Y==1)) snext=C;
else if((X==1)&(Y==1)) snext=D;
else if((X==1)&(Y==0)) snext=C;
```

Step 8 of 8

```
C: if((X==0)&(Y==0)) snext=C;
else if((X==0)&(Y==1)) snext=D;
else if((X==1)&(Y==1)) snext=A;
else if((X==1)&(Y==0)) snext=D;
D: if((X==0)&(Y==0)) snext=D;
else if((X==0)&(Y==1)) snext=A;
else if((X==1)&(Y==1)) snext=B;
else if((X==1)&(Y==0)) snext=A;
default snext =A;
endcase
end
always@(sreg) begin //output logic
case(sreg)
A: Z=1;
Default Z=0;
endcase
end
endmodule
```

Thus, the ABEL, VHDL and Verilog programs are written for ones-counting state machine.

7.26DP

Step 1 of 11

Refer to the Table 7-11 from the text book.

Table 7-11 represents the state table for combination- lock machine.

The ABEL program for combination-lock state machine is as follows.

```
module CLM
title 'Combination-lock machine'
"Input and Output pins
Clk,X pin;
Q1,Q2,Q3 pin istype 'reg';
Unik,Hint pin istype 'com';
"Definitions
S=[Q1,Q2,Q3]; "state variables
A=[0,0,0]; "state encoding
B=[0,0,1];
C=[0,1,0];
D=[0,1,1];
E=[1,0,0];
F=[1,0,1];
G=[1,1,0];
H=[1,1,1];
```

Step 2 of 11

```
State_diagram S "state defining
State A: if X==0 then B with {Unlk=0; Hint=1}
else A with {Unlk=0; Hint=0}
State B: if X==0 then B with {Unlk=0; Hint=0}
else C with {Unlk=0; Hint=1}
State C: if X==0 then B with [Unlk=0; Hint=0}
else D with {Unlk=0; Hint=1]
State D: if X==0 then E with [Unlk=0; Hint=1}
else A with [Unlk=0; Hint=0]
State E: if X==0 then B with {Unlk=0; Hint=0}
else F with [Unik=0; Hint=1}
State F: if X==0 then B with {Unlk=0; Hint=0}
else G with {Unlk=0; Hint=1}
State G: if X==0 then E with [Unlk=0; Hint=0}
else H with {Unlk=0; Hint=1}
State H: if X==0 then B with [Unlk=1; Hint=1]
else A with [Unlk=0; Hint=0]
equations
S.Clk = Clk;
end CLM
```

Step 3 of 11

The VHDL program for combination lock machine state table is as follows.

```
library IEEE;
use IEEE.std_logic_1164.all;
-ENTITY DECLARATION
entity CLM is
port(CLK,X : in STD_LOGIC;
Unlk,Hint : out STD_LOGIC);
end;
-ARCHITECTURE DECLARATION
architecture CLM_arch of CLM is
type state_type is (A,B,C,D,E,F,G,H);
-- state declaration
signal sreg, snext: state_type;
--current state and next state
begin
process(CLK) -- state memory
begin
if CLK'event and CLK='1' then
sreg <= snext;
end if;
end process;
```

Step 4 of 11

```
process (X,sreg) -- next state logic
begin
case sreg is
when A => if X='0'then
snext<= B; Unlk=0; Hint=1;
else
snext<= A; Unlk=0; Hint=0;
end if;
when B => if X='0'then
```

Step 5 of 11

```
snext<= B; Unlk=0; Hint=0;
else
snext<= C; Unlk=0; Hint=1;
end if;
when C => if X='0'then
snext<= B; Unlk=0; Hint=0;
else
snext<= D; Unlk=0; Hint=1;
end if;
when D => if X='0'then
snext<= E; Unlk=0; Hint=1;
else
snext<= A; Unlk=0; Hint=0;
end if;
```

Step 6 of 11

```
when E => if X='0'then
snext<= B; Unlk=0; Hint=0;
else
snext<= F; Unlk=0; Hint=1;
end if;
when F => if X='0'then
snext<= B; Unlk=0; Hint=0;
else
snext<= G; Unik=0; Hint=1;
end if;
when G => if X='0'then
snext<= E; Unlk=0; Hint=0;
else
snext<= H; Unlk=0; Hint=1;
end if;
when D => if X='0'then
snext<= B; Unlk=1; Hint=1;
else
snext<= A; Unlk=0; Hint=0;
end if;
when others=> snext<= A; Unik=0; Hint=0;
end case;
end process;
end CM1_arch;
```

Step 7 of 11

The Verilog program for the combination-lock state machine is as follows.

```
module CLM(CLK,X,Unik,Hint) // module declaration
input CLK,X;
output Unlk,Hint;
reg Unlk,Hint;
reg[2:0] sreg,snext;
parameter[1:0] A=3'b000, // state declaration
B=3'b001,
C=3'b010,
D=3'b011,
E=3'b100,
F=3'b101,
G=3'b110,
H=3'b111;
always@( posedge CLK)
sreg<= snext;
always@(X, sreg) begin //state definition
case (sreg)
A: if (X==0)
begin
snext=B;
Unik=0;
Hint=1;
end
else
begin
snext=A;
Unik=0;
Hint=0;
end
```

Step 8 of 11

```
B: if (X==0)
begin
snext=B;
Unik=0;
Hint=0;
end
else
begin
snext=c;
Unik=0;
Hint=1;
end
C: if (X==0)
```

Step 9 of 11

```
begin
snext=B;
Unlk=0;
Hint=0;
end
else
begin
snext=D;
Unik=0;
Hint=1;
end
```

Step 10 of 11

```
D: if (X==0)
begin
snext=E;
Unik=0;
Hint=1;
end
else
begin
snext=A;
Unik=0;
Hint=0;
end
E: if (X==0)
begin
snext=B;
Unlk=0;
Hint=0;
end
else
begin
snext=F;
Unik=0;
Hint=1;
end
F: if (X==0)
begin
snext=B;
Unlk=0;
Hint=0;
end
else
begin
snext=G;
Unik=0;
Hint=1;
end
```

Step 11 of 11

```
G: if (X==0)
begin
snext=E;
Unik=0;
Hint=0;
end
else
begin
snext=H;
Unik=0;
Hint=1;
end
H: if (X==0)
begin
snext=B;
Unik=1;
Hint=1;
end
else
begin
snext=A;
Unik=0;
Hint=0;
end
default
begin
snext=A;
Unlk=0;
Hint=0;
end
endcase
end
endmodule
```

- 552 step-by-step solutions
- Solved by publishers, professors & experts
- iOS, Android, & web

Get Solutions

7.27DP

SOLUTION:

m for example of state machine is as follows.

```
tate machine'
 pins
in;
 istype 'reg';
;

 variables
ncoding





'state defining
T then S1 else S2;
T then S1
) then S3

T then S1

A) then S3

;

A.Clk=Clk; S.OE=1;
```

Chapter 7, Problem 27DP is solved.

View full solution

View a sample solution

Back to top

Textbook

Digital Design: Principles and Practices | 4th Edition

ISBN-13:	9780131733497
ISBN:	0131733494
Authors:	John F. Wakerly

Rent | Buy

This is an alternate ISBN. View the primary ISBN for: Digital Design: Principles and Practices 4th Edition Textbook Solutions

Problems in Chapter 7

2DP	3DP
5DP	6DP
8DP	9DP
11DP	12DP
14DP	15DP
17DP	18DP
20DP	21DP
23DP	24DP
26DP	27DP
29DP	30DP
32DP	33DP
35DP	36DP
38E	39E
41E	42E
44E	45E
47E	48E
50E	51E
53E	54E
56E	57E
59E	60E
62E	63E
65E	66E
68E	69E
71E	72E
74E	75E
77E	78E
80E	81E
83E	84E
86E	87E
89E	90E
92E	93E
95E	96E
98E	99E
101E	102E

Back to top

? Browse hundreds of online Electrical Engineering tutors.
]

Step 1 of 3

The ABEL program for combination lock machine state table is as follows.

```
module CLM
title 'Combination-lock machine'
"Input and Output pins
Clk,X pin;
Q1,Q2,Q3 pin istype 'reg';
Unlk,Hint pin istype 'com';
"Definitions
S=[Q1,Q2,Q3]; "state variables
A=[0,0,0]; "state encoding
B=[0,0,1];
C=[0,1,0];
D=[0,1,1];
E=[1,0,0];
F=[1,0,1];
G=[1,1,0];
H=[1,1,1];
State_diagram S "state defining
State A: if X==0 then B with {Unlk=0; Hint=1}
else A with {Unlk=0; Hint=0}
State B: if X==0 then B with {Unlk=0; Hint=0}
else C with {Unlk=0; Hint=1}
State C: if X==0 then B with {Unlk=0; Hint=0}
else D with {Unlk=0; Hint=1}
State D: if X==0 then E with {Unlk=0; Hint=1}
else A with {Unlk=0; Hint=0}
State E: if X==0 then B with {Unlk=0; Hint=0}
else F with {Unlk=0; Hint=1}
State F: if X==0 then B with {Unlk=0; Hint=0}
else G with {Unlk=0; Hint=1}
State G: if X==0 then E with {Unlk=0; Hint=0}
else H with {Unlk=0; Hint=1}
State H: if X==0 then B with {Unlk=1; Hint=1}
else A with {Unlk=0; Hint=0}
equations
S.CLK = Clk;
end CLM
```

Step 2 of 3

The test vector for the Combination lock state machine is as follows.

```
test_vectors
( [Clk, X] -> [ S, Unlk , Hint ] )
[ .C , 1] -> [.X., .X. , .X. ]
[ .C , 1] -> [.X., .X. , .X. ]
[ .C , 1] -> [.X., .X. , .X. ]
[ .C , 1] -> [ A, .X. , .X. ]
[ .C , 0] -> [ A, 0 , 1 ]
[ .C , 1] -> [ A, 0 , 0 ]
[ .C , 1] -> [ A, .X. , .X. ]
[ .C , 0] -> [ B, .X. , .X. ]
[ .C , 0] -> [ B, 0 , 0 ]
[ .C , 1] -> [ B, 0 , 1 ]
[ .C , 0] -> [ B, .X. , .X. ]
[ .C , 1] -> [ C, .X. , .X. ]
[ .C , 0] -> [ C, 0 , 0 ]
[ .C , 1] -> [ C, 0 , 1 ]
[ .C , 0] -> [ B, .X. , .X. ]
[ .C , 1] -> [ C, .X. , .X. ]
[ .C , 1] -> [ D, .X. , .X. ]
```

Step 3 of 3

The timing diagram for combination-lock state machine is as shown in Figure 1.

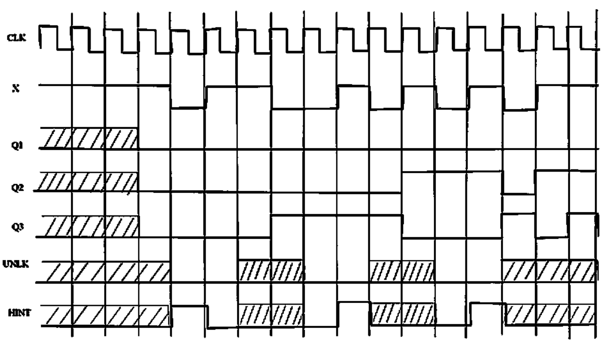


Figure 1: The timing diagram for combination lock state machine.

Step 1 of 5

The VHDL program for example of state machine is as follows.

```
Library IEEE;
Use IEEE.std_logic_1164.all;
Entity smeg is
Port (clock, reset, S1,S2: in STD_LOGIC;
Z: out STD_LOGIC);
End smeg;
Architecture sm_arch of smeg is
Type state_type is (INT, LOOK, CORRECT);
Signal sreg, snext: state_type;
Signal lastS1: std_logic;
Begin
Process(CLOCK)
Begin
If CLOCK'event and CLOCK='1' then
If RESET='1' then sreg<= INT; lastS1 <= '0';
Else sreg<=snext; lastS1<=S1; end if;
End if;
End process;
Process(S1, S2, lastS1, sreg)
Begin
Case sreg is
When INT => snext<= LOOK;
When LOOK=> if S1=lastS1 then snext<= CORRECT;
Else snext<=LOOK;
End if;
When CORRECT => if S2='1' then snext <= CORRECT;
Elseif S1=lastS1 then snext<=CORRECT;
Else snext<= LOOK;
End if;
When others => snext<=INT;
End case;
End process
With sreg select
Z<='1' when CORRECT,
'0' when others;
End sm_arch;
```

Step 2 of 5

The test vector example of state machine VHDL program is as follows.

```
Library IEEE;
Use IEEE.std_logic_1164.all;
Entity smeg_tb is
End;
Architecture smeg_tb_arch of smeg_tb is
Signal clk, reset, S1, S2, Z: STD_LOGIC;
Component smeg
Port (clock, reset, S1,S2: in STD_LOGIC;
Z: out STD_LOGIC);
End component;
Begin
UUT: smeg port map( clock => clk,
Reset => reset,
S1=> S1,
S2=> S2,
Z => Z );
Process
Begin
Clk <= '1'; wait for 5ns;
Clk <= '0'; wait for 5ns;
End process;
```

Step 3 of 5

```
Process
Begin
Reset <='1';
S1<='1'; S2<='1'; wait for 10ns;
Assert (Z='0') report "Failed test 1" severity error;
Reset <='0';
S1<='1', S2<='1'; wait for 10ns;
Assert (Z='0') report "Failed test 2" severity error;
S1<='1', S2<='0'; wait for 10ns;
Assert (Z='1') report "Failed test 3" severity error;
S1<='0', S2<='1'; wait for 10ns;
Assert (Z='1') report "Failed test 4" severity error;
S1<='1', S2<='0'; wait for 10ns;
Assert (Z='0') report "Failed test 5" severity error;
S1<='0', S2<='1'; wait for 10ns;
Assert (Z='0') report "Failed test 6" severity error;
S1<='0', S2<='0'; wait for 10ns;
Assert (Z='1') report "Failed test 7" severity error;
S1<='1', S2<='1'; wait for 10ns;
```

Step 4 of 5

```
Assert (Z='1') report "Failed test 8" severity error;
S1<='0', S2<='0'; wait for 10ns;
Assert (Z='0') report "Failed test 9" severity error;
Wait;
End process;
End smeg_tb;
```

Step 5 of 5

The timing diagram for state machine is as shown in Figure 1.

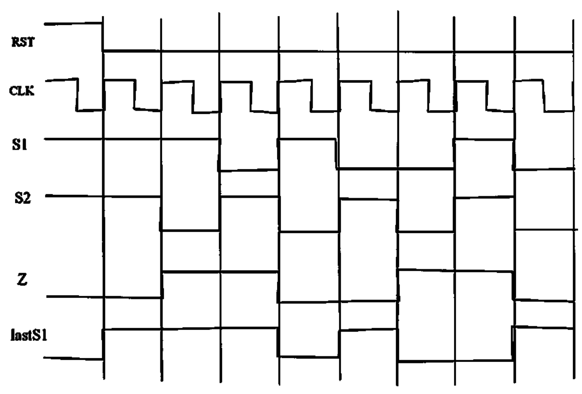


Figure 1: The timing diagram for state machine.

Step 1 of 4

7.30DP

The Verilog program for example of state machine is as follows.

```
Module smeg(clk,reset,S1,S2,Z);

Input clk,reset,S1,S2;

Output Z;

Wire Z;

Reg lastS1;

Reg[1:0] sreg,snext;

Parameter [1:0] INT = 2'b00,

LOOK = 2'b10,

CORRECT = 2'11;

Always @ (posedge clk) begin

If(reset==1) sreg<=INT;else sreg<=snext;

Lasta <=A;

End

Always @ (S1,S2,lastS1,sreg) begin

Case (sreg)

INT: snext=LOOK;

LOOK: if (S1=lastS1) snext=CORRECT;

Else snext=LOOK;

CORRECT: if(S2==1 || S1=lastS1) snext=CORRECT;

Else snext=LOOK;

Default snext=INT;

Endcase

End

Assign Z=(sreg==CORRECT)?1:0;

endmodule
```

Step 2 of 4

The test vector example of state machine verilog program is as follows.

```
Module smeg_tb();

Reg CLK, RESET, S1,S2;

Wire Z;

Smeg UUT (.clk(CLK),

.reset(RESET),

.S1(S1),

.S2(S2),

.Z(Z));

Task checkZ;

Input inpnum,expZ;

Integer inpnum; reg expZ;

Begin

If(Z!=expZ) begin

$display ($time,"Error,step%d,expected %b,

got %b", inpnum, expZ, Z);

$stop(1); end;

End

Endtask
```

Step 3 of 4

```
Always begin

#5 CLK=0;

#5 CLK=1;

End

Initial begin

$monitor("Time:%d RESET=%b CLK=%b S1=%b S2=%b Z=%b",

$time,RESET,CLK,S1,S2,Z);

RESET=1;

S1=1;S2=1;

CLK=1; #10;

checkZ(1,0);

RESET=0;

S1=1;S2=1; #10 checkZ(2,0);

S1=1;S2=0; #10 checkZ(3,1);

S1=0;S2=1; #10 checkZ(4,1);

S1=1;S2=0; #10 checkZ(5,0);

S1=0;S2=1; #10 checkZ(6,0);

S1=0;S2=0; #10 checkZ(7,1);

S1=1;S2=1; #10 checkZ(8,1);

S1=0;S2=0; #10 checkZ(9,0);

$stop(1);

End

Endmodule
```

Step 4 of 4

The timing diagram for state machine is as shown in Figure 1.

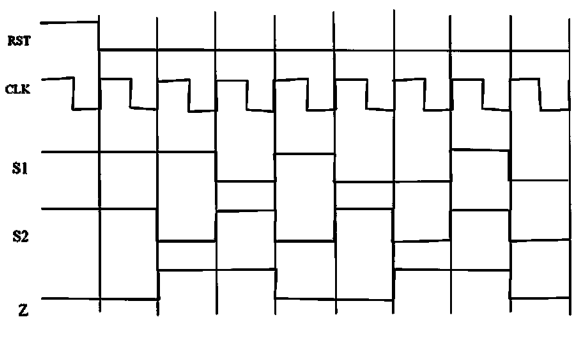


Figure 1: The timing diagram for state machine.

Step 1 of 1

7.32DP

A wired-AND function is obtained simply by tying two or more open-drain or open-collector output together, without going through another level of transistor circuitry but we need additional pull-up resistor to drive the load.

A wired-AND logic is nothing but the outputs of many open drain are connected together such that to form a AND logic with single pull-up resistor. Thus larger pull-up resistance is required to drive the load which in tum increases the RC time constant and longer rise time.

So, as a result the wired-AND logic is slower than a discrete AND logic.

Step 1 of 4 **7.33DP**

The following is the VHDL code for sticky counter.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity COUNTER is
port(
RESET, ENABLE, CLK: in STD_LOGIC;
DONE: out STD_LOGIC;
S: out UNSIGNED(2 downto 0)
);
end COUNTER;
architecture COUNTER_arch of COUNTER is
signal D: UNSIGNED(2 downto 0);
begin
process(CLK)
begin
if (CLK'event and CLK='1') then
if(RESET='1') then
D <= "000";
end if;
if(ENABLE='1' and D<7) then
D <= D+1;
end if;
if(ENABLE='1' and D=7) then
D <= "111";
DONE <= '1';
end if;
end if;
S <= D;
end process;
end COUNTER_arch;
```

Step 2 of 4

The following is the VHDL test bench to check proper operation of the sticky counter.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE STD.TEXTID.ALL;
ENTITY naaw IS
END naaw;
ARCHITECTURE testbench_arch OF naaw IS
FILE RESULTS: TEXT OPEN WRITE_MODE IS "D:
od3
aaw.ano";
COMPONENT COUNTER
PORT (
RESET : In std_logic;
ENABLE : In std_logic;
CLK : In std_logic;
DONE : Out std_logic;
S : Out UNSIGNED (2 DownTo 0)
);
END COMPONENT;
SIGNAL RESET : std_logic := '0';
SIGNAL ENABLE : std_logic := '0';
SIGNAL CLK : std_logic := '0';
SIGNAL DONE : std_logic := '0';
SIGNAL S : UNSIGNED (2 DownTo 0) := "000";
SHARED VARIABLE TX_ERROR : INTEGER := 0;
SHARED VARIABLE TX_OUT : LINE;
constant PERIOD : time := 200 ns;
constant DUTY_CYCLE : real := 0.5;
constant OFFSET : time := 0 ns;
BEGIN
UUT : COUNTER
PORT MAP (
RESET => RESET,
ENABLE => ENABLE,
CLK => CLK,
DONE => DONE,
S => S
);
PROCESS -- clock process for CLK
BEGIN
WAIT for OFFSET;
CLOCK_LOOP : LOOP
CLK <= '0';
WAIT FOR (PERIOD - (PERIOD * DUTY_CYCLE));
CLK <= '1';
WAIT FOR (PERIOD * DUTY_CYCLE);
END LOOP CLOCK_LOOP;
END PROCESS;
```

Step 3 of 4

```
PROCESS -- Annotation process for CLK
VARIABLE TX_TIME : INTEGER := 0;
PROCEDURE ANNOTATE_DONE(
TX_TIME : INTEGER
) IS
VARIABLE TX_STR : String(1 to 4096);
VARIABLE TX_LOC : LINE;
BEGIN
STD.TEXTIO.write(TX_LOC, string'("Annotate["));
STD.TEXTIO.write(TX_LOC, TX_TIME);
STD.TEXTIO.write(TX_LOC, string'(", DONE, "));
IEEE.STD_LOGIC_TEXTIO.write(TX_LOC, DONE);
STD.TEXTIO.write(TX_LOC, string'("]"));
TX_STR(TX_LOC.all'range) := TX_LOC.all;
STD.TEXTIO.writeline(RESULTS, TX_LOC);
STD.TEXTIO.Deallocate(TX_LOC);
END;
PROCEDURE ANNOTATE_S(
TX_TIME : INTEGER
) IS
VARIABLE TX_STR : String(1 to 4096);
VARIABLE TX_LOC : LINE;
BEGIN
STD.TEXTIO.write(TX_LOC, string'("Annotate["));
STD.TEXTIO.write(TX_LOC, TX_TIME);
STD.TEXTIO.write(TX_LOC, string'(", S, "));
STD.TEXTIO.write(TX_LOC, CONV_INTEGER(S));
STD.TEXTIO.write(TX_LOC, string'("]"));
TX_STR(TX_LOC.all'range) := TX_LOC.all;
STD.TEXTIO.writeline(RESULTS, TX_LOC);
STD.TEXTIO.Deallocate(TX_LOC);
END;
BEGIN
WAIT for 1 fs;
ANNOTATE_DONE(0);
ANNOTATE_S(0);
WAIT for OFFSET;
TX_TIME := TX_TIME + 0;
ANNO_LOOP : LOOP
–Rising Edge
WAIT for 115 ns;
TX_TIME := TX_TIME + 115;
ANNOTATE_DONE(TX_TIME);
ANNOTATE_S(TX_TIME);
WAIT for 85 ns;
TX_TIME := TX_TIME + 85;
END LOOP ANNO_LOOP;
END PROCESS;
PROCESS
BEGIN
– ———— Current Time: 85ns
WAIT FOR 85 ns;
ENABLE <= '1';
RESET <= '1';
– ——————————
– ———– Current Time: 285ns
WAIT FOR 200 ns;
RESET <= '0';
– ——————————
– ———– Current Time: 1885ns
WAIT FOR 1600 ns;
RESET <= '1';
– ——————————
– ———– Current Time: 2085ns
WAIT FOR 200 ns;
RESET <= '0';
– ——————————
WAIT FOR 415 ns;
STD.TEXTIO.write(TX_OUT, string'("Total[]"));
STD.TEXTIO.writeline(RESULTS, TX_OUT);
ASSERT (FALSE) REPORT
"Success! Simulation for annotation completed"
SEVERITY FAILURE;
END PROCESS;
END testbench_arch;
```

The waveform is shown in Figure 1.

Figure 1

Step 4 of 4

Figure 1 explains the behaviour in the sticky counter program.

Step 1 of 1 **7.34DP**

Write VHDL program for a desired state machine.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity StickyCounter is
port( RESET, ENABLE, CLK: in STD_LOGIC;
DONE, BACK: out STD_LOGIC;
S: out UNSIGNED(3 downto 0)
);
end StickyCounter;
architecture StickyCounter_arch of StickyCounter is
signal D: UNSIGNED(3 downto 0);
begin
process(CLK)
begin
if (CLK'event and CLK='1') then
if(RESET='1') then
D <= "0000";
end if;
if(ENABLE='1' and D<7) then
D <= D+2;
– Counts two steps forward
D <= D-1;
– Counts a step backward
BACK <= '1';
– Back is asserted
end if;
if(ENABLE='1' and D=7) then
D <= "0111";
DONE <= '1';
BACK <= '0';
end if;
end if;
S <= D;
end process;
end StickyCounter_arch;
```

Need an extra bit for the additional state as at state 6 it moves two steps ahead to state 8 which is of four bit.

Thus, desired state machine has been implimented using VHDL.

7.35DP

Step 1 of 6

The following is the VHDL code for sticky-counter:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity StickyCounter is
port(
RESET, ENABLE, CLK: in STD_LOGIC;
DONE, BACK: out STD_LOGIC;
S: out UNSIGNED(3 downto 0)
);
end StickyCounter;
architecture StickyCounter_arch of StickyCounter is
signal D: UNSIGNED(3 downto 0);
begin
process(CLK)
begin
if (CLK'event and CLK='1') then
if(RESET='1') then
D <= "0000";
end if;
if(ENABLE='1' and D<7) then
D <= D+2;
-- Counts two steps forward
D <= D-1;
– Counts a step backward
BACK <= '1';
– Back is asserted
end if;
if(ENABLE='1' and D=7) then
D <= "0111";
DONE <= '1';
BACK <= '0';
end if;
end if;
S <= D;
end process;
end StickyCounter_arch;
```

Step 2 of 6

The following is the VHDL testbench code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE STD.TEXTIO.ALL;
ENTITY neew IS
END neew;
ARCHITECTURE testbench_arch OF neew IS
COMPONENT StickyCounter
PORT (
RESET : In std_logic;
ENABLE : In std_logic;
CLK : In std_logic;
BACK : Out std_logic;
DONE : Out std_logic;
S : Out UNSIGNED (3 DownTo 0)
);
END COMPONENT;
SIGNAL RESET : std_logic := '0';
SIGNAL ENABLE : std_logic := '0';
SIGNAL CLK : std_logic := '0';
SIGNAL BACK : std_logic := '0';
SIGNAL DONE : std_logic := '0';
SIGNAL S : UNSIGNED (3 DownTo 0) := "0000";
SHARED VARIABLE TX_ERROR : INTEGER := 0;
SHARED VARIABLE TX_OUT : LINE;
constant PERIOD : time := 200 ns;
constant DUTY_CYCLE : real := 0.5;
constant OFFSET : time := 0 ns;
BEGIN
UUT : StickyCounter
PORT MAP (
RESET => RESET,
ENABLE => ENABLE,
CLK => CLK,
BACK => BACK,
DONE => DDNE,
S => S
);
PROCESS -- clock process for CLK
BEGIN
WAIT for OFFSET;
CLOCK_LOOP : LOOP
CLK <= '0';
```

Step 3 of 6

```
WAIT FOR (PERIOD - (PERIOD * DUTY_CYCLE));
CLK <= '1';
WAIT FOR (PERIOD * DUTY_CYCLE);
END LOOP CLOCK_LOOP;
END PROCESS;
PROCESS
PROCEDURE CHECK_BACK(
next_BACK : std_logic;
TX_TIME : INTEGER
) IS
VARIABLE TX_STR : String(1 to 4096);
VARIABLE TX_LOC : LINE;
BEGIN
IF (BACK /= next_BACK) THEN
STD.TEXTID.write(TX_LOC, string'("Error at time="));
STD.TEXTID.write(TX_LOC, TX_TIME);
STD.TEXTID.write(TX_LOC, string'("ns BACK="));
IEEE.STD_LOGIC_TEXTIO.write(TX_LOC, BACK);
STD.TEXTID.write(TX_LOC, string'(", Expected = "));
IEEE.STD_LOGIC_TEXTIO.write(TX_LOC, next_BACK);
STD.TEXTID.write(TX_LOC, string'(" "));
TX_STR(TX_LDC.all'range) := TX_LOC.all;
STD.TEXTID.Deallocate(TX_LOC);
ASSERT (FALSE) REPORT TX_STR SEVERITY ERROR;
TX_ERROR := TX_ERROR + 1;
END IF;
END;
PROCEDURE CHECK_DONE(
next_DONE : std_logic;
TX_TIME : INTEGER
) IS
VARIABLE TX_STR : String(1 to 4096);
VARIABLE TX_LOC : LINE;
BEGIN
IF (DDNE /= next_DONE) THEN
STD.TEXTID.write(TX_LOC, string'("Error at time="));
STD.TEXTID.write(TX_LOC, TX_TIME);
STD.TEXTID.write(TX_LOC, string'("ns DONE="));
IEEE.STD_LOGIC_TEXTIO.write(TX_LOC, DONE);
STD.TEXTID.write(TX_LOC, string'(", Expected = "));
IEEE.STD_LOGIC_TEXTIO.write(TX_LOC, next_DONE);
STD.TEXTID.write(TX_LOC, string'(" "));
TX_STR(TX_LDC.all'range) := TX_LOC.all;
STD.TEXTID.Deallocate(TX_LOC);
ASSERT (FALSE) REPORT TX_STR SEVERITY ERROR;
TX_ERROR := TX_ERROR + 1;
END IF;
END;
PROCEDURE CHECK_S(
next_S : UNSIGNED (2 DownTo 0);
TX_TIME : INTEGER
) IS
VARIABLE TX_STR : String(1 to 4096);
VARIABLE TX_LOC : LINE;
BEGIN
IF (S /= next_S) THEN
STD.TEXTID.write(TX_LOC, string'("Error at time="));
STD.TEXTID.write(TX_LOC, TX_TIME);
STD.TEXTID.write(TX_LOC, string'("ns S="));
STD.TEXTID.write(TX_LOC, CONV_INTEGER(S));
STD.TEXTID.write(TX_LOC, string'(", Expected = "));
STD.TEXTID.write(TX_LOC, CONV_INTEGER(next_S));
STD.TEXTID.write(TX_LOC, string'(" "));
TX_STR(TX_LDC.all'range) := TX_LOC.all;
STD.TEXTID.Deallocate(TX_LOC);
ASSERT (FALSE) REPORT TX_STR SEVERITY ERROR;
TX_ERROR := TX_ERROR + 1;
END IF;
END;
BEGIN
– ———— Current Time: 85ns
WAIT FOR 85 ns;
ENABLE <= '1';
RESET <= '1';
– ————————————
– ———— Current Time: 285ns
WAIT FOR 200 ns;
RESET <= '0';
– ————————————
– ———— Current Time: 515ns
WAIT FOR 230 ns;
```

Step 4 of 6

```
CHECK_S("001", 515);
– ————————————
– ———— Current Time: 715ns
WAIT FOR 200 ns;
CHECK_S("010", 715);
– ————————————
– ———— Current Time: 915ns
WAIT FOR 200 ns;
CHECK_S("011", 915);
– ————————————
– ———— Current Time: 1115ns
WAIT FOR 200 ns;
CHECK_S("100", 1115);
– ————————————
– ———— Current Time: 1315ns
WAIT FOR 200 ns;
CHECK_S("101", 1315);
– ————————————
– ———— Current Time: 1515ns
WAIT FOR 200 ns;
CHECK_S("110", 1515);
– ————————————
– ———— Current Time: 1715ns
WAIT FOR 200 ns;
CHECK_DONE('1', 1715);
CHECK_S("111", 1715);
– ————————————
– ———— Current Time: 1885ns
WAIT FOR 170 ns;
RESET <= '1';
– ————————————
– ———— Current Time: 1915ns
WAIT FOR 30 ns;
CHECK_DONE('0', 1915);
CHECK_S("000", 1915);
– ————————————
– ———— Current Time: 2085ns
WAIT FOR 170 ns;
RESET <= '0';
– ————————————
– ———— Current Time: 2115ns
WAIT FOR 30 ns;
CHECK_DONE('1', 2115);
CHECK_BACK('1', 2115);
CHECK_S("111", 2115);
– ————————————
WAIT FOR 385 ns;
```

Step 5 of 6

```
IF (TX_ERROR = 0) THEN
STD.TEXTID.write(TX_OUT, string'("No errors or warnings"));
ASSERT (FALSE) REPORT
"Simulation successful (not a failure). No problems detected."
SEVERITY FAILURE;
ELSE
STD.TEXTID.write(TX_OUT, TX_ERROR);
STD.TEXTID.write(TX_OUT,
string'(" errors found in simulation"));
ASSERT (FALSE) REPORT "Errors found during simulation"
SEVERITY FAILURE;
END IF;
END PROCESS;
END testbench_arch;
```

Step 6 of 6

The waveform is as shown in Figure 1.

Figure 1

The Figure 1 shows the behaviour of the sticky-counter program.

Step 1 of 2

The VHDL test bench of table 7-52 is

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE STD.TEXTIO.ALL;
ENTITY S IS
END S;
ARCHITECTURE testbench_arch OF S IS
COMPONENT ss
PORT (
S : In std_logic;
R : In std_logic;
Q : InOut std_logic;
QN : InOut std_logic
);
END COMPONENT;
SIGNAL S : std_logic := '0';
SIGNAL R : std_logic := '0';
SIGNAL Q : std_logic := '0';
SIGNAL QN : std_logic := '0';
SHARED VARIABLE TX_ERROR : INTEGER := 0;
SHARED VARIABLE TX_OUT : LINE;
BEGIN
UUT : ss
PORT MAP (
S => S,
R => R,
Q => Q,
QN => QN
);
PROCESS
BEGIN
– ---------- Current Time: 100ns
WAIT FOR 100 ns;
S <= '1';
– ---------------------------
– ---------- Current Time: 300ns
WAIT FOR 200 ns;
R <= '1';
– ---------------------------
– ---------- Current Time: 400ns
WAIT FOR 100 ns;
S <= '0';
– ---------------------------
– ---------- Current Time: 800ns
WAIT FOR 400 ns;
S <= '1';
– ---------------------------
– ---------- Current Time: 1100ns
WAIT FOR 300 ns;
S <= '0';
R <= '0';
WAIT FOR 400 ns;
IF (TX_ERROR = 0) THEN
STD.TEXTIO.write(TX_OUT, string'("No errors or warnings"));
ASSERT (FALSE) REPORT
"Simulation successful (not a failure). No problems detected."
SEVERITY FAILURE;
ELSE
STD.TEXTIO.write(TX_OUT, TX_ERROR);
STD.TEXTIO.write(TX_OUT,
string'(" errors found in simulation"));
ASSERT (FALSE) REPORT "Errors found during simulation"
SEVERITY FAILURE;
END IF;
END PROCESS;
END testbench_arch;
```

Step 2 of 2

The wave forms are as as shown in Figure 1.

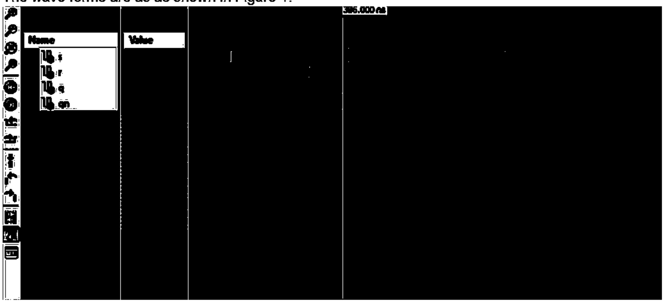


Figure 1

During the last input transaction the outputs of the flip-flop is in metastable state.

7.37DP

Step 1 of 4

The modified program using the table 7-52 is

Program:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity ss is
Port ( S : in STD_LOGIC;
R : in STD_LOGIC;
Q : inout STD_LOGIC;
QN : inout STD_LOGIC);
end ss;
architecture Behavioral of ss is
begin
QN <= S nor Q after 10 ns;
Q <= R nor QN;
end Behavioral;
```

Step 2 of 4

Test bench:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE STD.TEXTIO.ALL;
ENTITY S IS
END S;
ARCHITECTURE testbench_arch OF S IS
FILE RESULTS: TEXT OPEN WRITE_MODE IS "results.txt";
COMPONENT ss
PORT (
S : In std_logic;
R : In std_logic;
Q : InOut std_logic;
QN : InOut std_logic
);
END COMPONENT;
SIGNAL S : std_logic := '0';
SIGNAL R : std_logic := '0';
SIGNAL Q : std_logic := '0';
SIGNAL QN : std_logic := '0';
SHARED VARIABLE TX_ERROR : INTEGER := 0;
SHARED VARIABLE TX_OUT : LINE;
BEGIN
UUT : ss
PORT MAP (
S => S,
R => R,
Q => Q,
QN => QN
);
PROCESS
BEGIN
-- ------------- Current Time: 100ns
WAIT FOR 100 ns;
S <= '1';
-- -------------------------------------
-- ------------- Current Time: 300ns
WAIT FOR 200 ns;
R <= '1';
-- -------------------------------------
-- ------------- Current Time: 400ns
WAIT FOR 100 ns;
S <= '0';
-- -------------------------------------
-- ------------- Current Time: 800ns
WAIT FOR 400 ns;
S <= '1';
-- -------------------------------------
-- ------------- Current Time: 1100ns
WAIT FOR 300 ns;
S <= '0';
R <= '0';
WAIT FOR 400 ns;
IF (TX_ERROR = 0) THEN
STD.TEXTIO.write(TX_OUT, string'("No errors or warnings"));
STD.TEXTIO.writeline(RESULTS, TX_OUT);
ASSERT (FALSE) REPORT
"Simulation successful (not a failure). No problems detected."
SEVERITY FAILURE;
ELSE
```

Step 3 of 4

```
STD.TEXTIO.write(TX_OUT, TX_ERROR);
STD.TEXTIO.write(TX_OUT,
string'(" errors found in simulation"));
STD.TEXTIO.writeline(RESULTS, TX_OUT);
ASSERT (FALSE) REPORT "Errors found during simulation"
SEVERITY FAILURE;
END IF;
END PROCESS;
END testbench_arch;
```

Step 4 of 4

The waveform of the above program is as shown in figure 1.

Figure 1

The output in the last input transaction is in metastable state.

Step 1 of 2

Whenever the setup and hold times don't meet in any flip-flop, it enters into a state where its output is unpredictable: this state is known as meta-stability.

Consider a D latch and input to the D latch is obtained from the combinational circuit.

Figure 1 represents occurrence of the meta-stability with T_{su} being the setup time and T_h being the hold time.

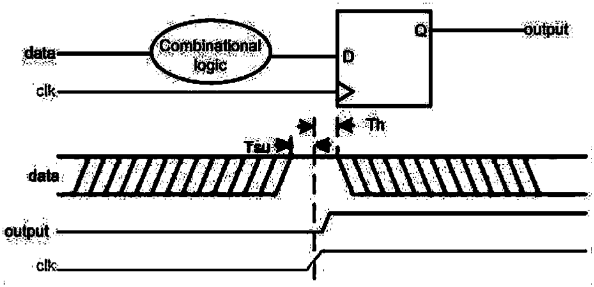


Figure 1

Step 2 of 2

The meta-stability is reached when the flip-flop setup and hold times are violated.

Assume the use of a positive edge triggered "D" flip-flop.

Whenever the rising edge of the D flip-flop occurs at the time when the input to the D flip-flop causes the master latch to transition, then the flip-flop is more likely to end up in meta-stability.

Thus, explained the occurrence of meta-stability in D latch.

Step 1 of 1

The minimum settling time of any pulse triggered flipflop depends on the duty cycle of the clock period, that is the time for which the clock stays in the high level.

The minimum duty cycle of a pulse triggered flip-flop is 50%.

Consider that the clock period is 10 ns.

The setup time is half of the time period of the clock signal, thus, the set up time is 5 ns.

Thus, the setup time of the pulse triggered flipflop depends on the duty cycle of the clock period and for a consideration of 10 ns clock period, the setup time is 5 ns.

Step 1 of 1

Metastability include the timing behaviour. It includes timing behaviour such as setup time, propagtaion delay and hold times.

Preset in 74x74 edge triggered D flip-flop is used to set the outputs states to 1 and clear is used to set the output states to 0, if both these are asserted simultaneoulsy then the outputs $\boldsymbol{Q}$ and $\boldsymbol{Q_N}$ will be same for long time.

Thus, other than metastability the outputs $\boldsymbol{Q}$ and $\boldsymbol{Q_N}$ of a 74x74 edge-triggered D flip-flop will be non-complementary for an arbitrarily long time **when preset and clear are asserted simultaneously.**

Step 1 of 3

Refer to Figure 7-12 from the text book for the logic diagram. Observe that when the C is inserted, Q output follows the D input. The path from D input to Q output is transparent when latch is open. The Q output retains its previous value when the latch is closes.

The truth table for the circuit in the Figure 7-12 is shown in Table 1.

Table 1

Inputs	**Outputs**		
C	**D**	**Q**	**QN**
1	0	0	1
1	1	1	0
0	X	Prev Q	Prev QN

Step 2 of 3

The truth table for the circuit in the Figure X7-41 is as shown in Table 2.

Table 2

Inputs	**Outputs**		
C	**D**	**Q**	**QN**
1	0	0	1
1	1	1	0
0	X	Prev Q	Prev QN

Step 3 of 3

Since both the truth tables in Table 1 and Table 2 are identical. In other words, the circuits in the Figure 7-12 and Figure X7-41 are identical.

Figure X7-41 is used where minimum number of logic gates are used.

Step 1 of 2

The design of the state machine in the Drill 7.12 using three inverting gates other than the not gate is as shown in the Figure 1.

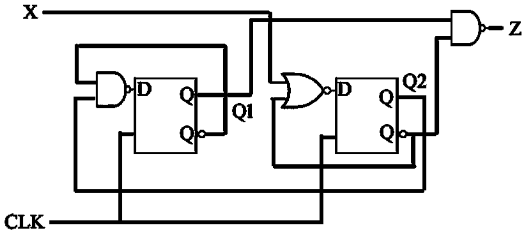


Figure 1

Step 2 of 2

Observe that the clock is applied to both D flip-flops. The output of first flipflop taken as Q_1 and output of second flipflop taken as Q_2 . The AND and OR gates are replaced with NOR and NAND gates respectively but the circuit yields the same result as the state machine in the Drill 7.12.

Step 1 of 2

The design of the state machine in the Drill 7.12 using three inverting gates other than the not gate is as shown in the Figure 1.

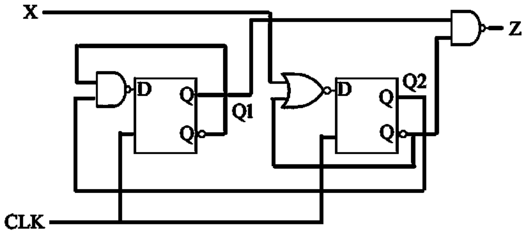


Figure 1

Step 2 of 2

Observe that the clock is applied to both D flip-flops. The output of first flipflop taken as Q_1 and output of second flipflop taken as Q_2 . The AND and OR gates are replaced with NOR and NAND gates respectively but the circuit yields the same result as the state machine in the Drill 7.12.

Step 1 of 2

Consider the states $S0, S1, SE0, SE1, SZ0, SZ1$.

$SE0$ and $S0$ are the states with $\text{INIT}=0$ and $X=0$.

$SE1$ and $S1$ are the states with $\text{INIT}=0$ and $X=1$.

$SZ0$ is the state with $\text{INIT}=0$ and twice $X=0$ or twice $X=1$.

$SZ1$ is the state with $\text{INIT}=1$ and $X=1$ or $X=0$.

Consider the state $S0$,

If INIT is 1, then it goes to next state SZ1 with output Z equal to 1.

If INIT is 0 and X is 0, then it goes to next state SE0 with output Z equal to 0.

If INIT is 0 and X is 1, then it goes to next state SE1 with output Z equal to 0.

Consider the state $S1$,

If INIT is 1, then it goes to next state SZ1 with output Z equal to 1.

If INIT is 0 and X is 0, then it goes to next state SE0 with output Z equal to 0.

If INIT is 0 and X is 1, then it goes to next state SE1 with output Z equal to 0.

Consider the state $SE0$,

If INIT is 1, then it goes to next state SZ1 with output Z equal to 1.

If INIT is 0 and X is 0, then it goes to next state SZ0 with output Z equal to 0.

Consider the state $SE1$,

If INIT is 1, then it goes to next state SZ1 with output Z equal to 1.

If INIT is 0 and X is 1, then it goes to next state SZ0 with output Z equal to 0.

Consider the state $SZ0$,

If INIT is 0, then it goes to next state SZ0 itself with output Z equal to 0.

If INIT is 1, then it goes to next state SZ1 with output Z equal to 1.

Step 2 of 2

Use the explanation and draw the state diagram for a clocked synchronous state machine is as shown in Figure 1.

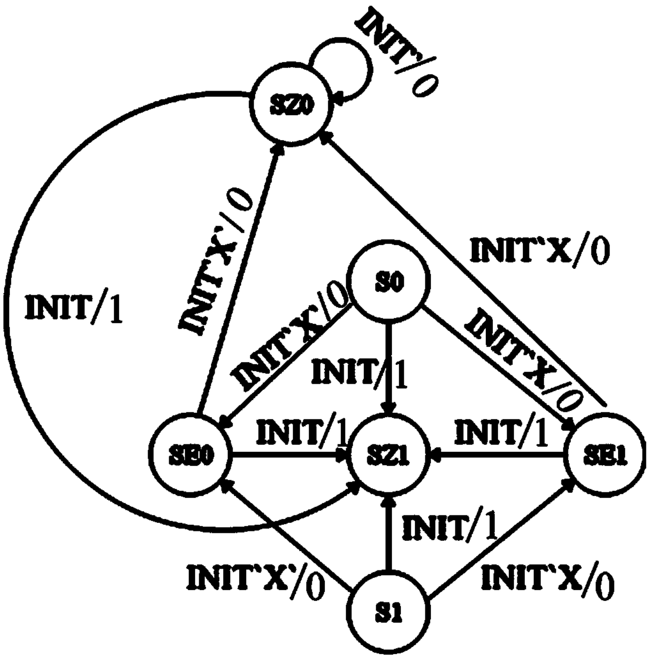


Figure 1

Thus, the state diagram is drwan for the clocked synchronous state machine.

Step 1 of 1 **7.045E**

The ABEL program for the clocked synchronous state machine of Exercise 7.44 is as follows:

```
module SMEX

title 'ABEL version of a state machine'

" Input and Outputt pins

CLOCK, RESET_L, INIT, X pin;

Q1, Q2, Q3 pin istype 'reg';

Z pin istype 'com';

" Definitions

QSTATE = [Q1, Q2, Q3];

" State variables

S0 = [0, 0, 0];

S1 = [0, 0, 1];

SE0 = [1, 0, 0];

SE1 = [1, 0, 1];

SZ0 = [0, 1, 1];

SZ1 = [1, 1, 1];

RESET = !RESET_L;

state_diagram QSTATE

state S0: if RESET then S0

else if (INIT==0) & (X==0) then SE0

else if (INIT==0) & (X==1) then SE1

else SZ1;

state S1: if RESET then S0

else if (INIT==0) & (X==0) then SE0

else if (INIT==0) & (X==1) then SE1

else SZ1;

state SE0: if RESET then S0

else if (INIT==0) & (X==0) then SZ0

else if (INIT==0) & (X==1) then SE1

else SZ1;

state SE1: if RESET then S0

else if (INIT==0) & (X==0) then SE0

else if (INIT==0) & (X==1) then SZ0

else SZ1;

state SZ0: if RESET then S0

else if (INIT==0) then SZ0

else SZ1;

state SZ1: if RESET then S0

else if (INIT==0) & (X==0) then S0

else if (INIT==0) & (X==1) then S1

else SZ1;

equations

QSTATE.CLK = CLOCK;

Z = (QSTATE==SZ1);

end SMEX
```

Step 1 of 4

The states assigned to A,B,C and D are as follows:

$A = 00$
$B = 01$
$C = 10$
$D = 11$

Refer to the Table X7.46 in the textbook.

The transition table for the given state table is as shown in Table 1.

Q1Q2	X		Z
	0	1	
00	01	10	0
01	11	01	0
11	01	00	1
10	01	11	0
	$Q1^{*}Q2^{*}$	$Q1^{*}Q2^{*}$	

Table 1

Step 2 of 4

Observe Table 1, for the present state and applied inputs, the next states are obtained.

Derive the expression for the next state $Q1^{*}$ from the transition table.

Draw the K-map for next state $Q1^{*}$ as shown in the Figure 1.

For $Q1^{*}$

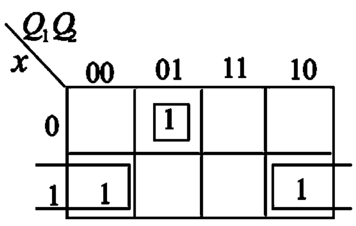


Figure 1

Obtain the boolean expression for $Q1^{*}$ from the k-map shown in Figure 1.

$$Q1^{*} = X\cdot\overline{Q2} + \overline{X}\cdot\overline{Q1}\cdot Q2$$

Step 3 of 4

Observe Table 1, for the present state and applied inputs, the next states are obtained.

Derive the expression for the next state $Q2^{*}$ from the transition table.

Draw the K-map for next state $Q2^{*}$ as shown in the Figure 2.

For $Q2^{*}$

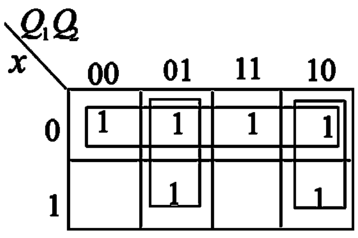


Figure 2

Obtain the boolean expression for $Q2^{*}$ from the k-map shown in Figure 2.

$$\begin{aligned} Q2^{*} &= \overline{X} + \overline{Q1}\cdot Q2 + Q1\cdot\overline{Q2} \\ &= \overline{X} + (Q1 \oplus Q2) \end{aligned}$$

Obtain the expression for Z from the transition table shown in Table 1.

$$Z = Q1\cdot Q2$$

Step 4 of 4

Use the boolean expressions of $Q1^{*}, Q2^{*}$ **and Z** and design a clocked synchronous state machine using D flip-flops as shown in the Figure 3.

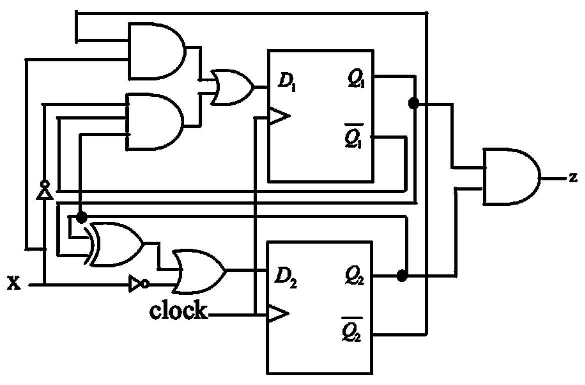


Figure 3

Thus, the clocked synchronous state machine is designed.

Step 1 of 5

Refer to the Table 7-5 and Table 7-6 from the text book.

Using the Table 7-5 and the simplest assignment in Table 7-6 the transition list is built as shown in Table 1.

Table 1

Q2Q1Q0	XY	Z			
	00	**01**	**11**	**10**	
000	001	001	010	010	0
001	011	011	010	010	0
010	001	001	100	100	0
011	011	011	100	010	1
100	001	011	100	100	1
$Q2^{*}Q1^{*}Q0^{*}$					

Step 2 of 5

Observe Table 1, for the present state and applied inputs, the next states are obtained.

Derive the expression for the next state $Q2^{*}$ from the transition table.

Draw the K-map for next state $Q2^{*}$ as shown in the Figure 1.

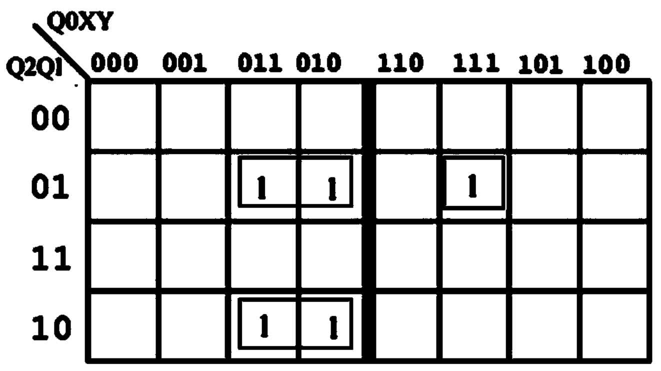


Figure 1

Obtain the boolean expression for the next state $Q2^{*}$ using the k-map shown in Figure 1.

$$Q2^{*} = Q2\cdot\overline{Q1}\cdot\overline{Q0}\cdot X + \overline{Q2}\cdot Q1\cdot\overline{Q0}\cdot X + \overline{Q2}\cdot Q1\cdot X\cdot Y$$

Step 3 of 5

Derive the expression for the next state $Q1^{*}$ from the transition table.

Draw the K-map for next state $Q1^{*}$ as shown in the Figure 2.

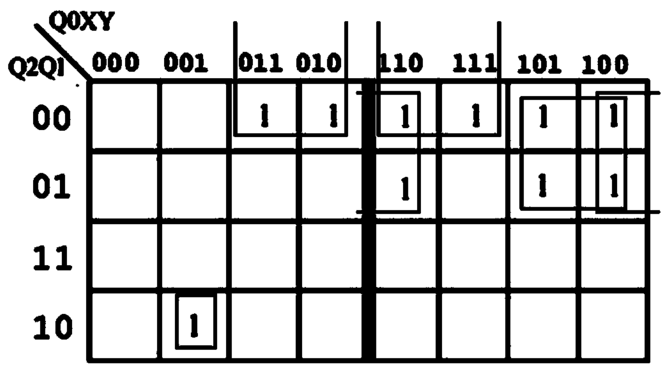


Figure 2

Obtain the boolean expression for the next state $Q1^{*}$ using the k-map shown in Figure 2.

$$Q1^{*} = Q2\cdot\overline{Q1}\cdot\overline{Q0}\cdot\overline{X}\cdot Y + \overline{Q2}\cdot\overline{Q1}\cdot X + \overline{Q2}\cdot Q0\cdot\overline{X} + \overline{Q2}\cdot Q0\cdot\overline{Y}$$

Step 4 of 5

Derive the expression for the next state $Q0^{*}$ from the transition table.

Draw the K-map for next state $Q0^{*}$ as shown in the Figure 3.

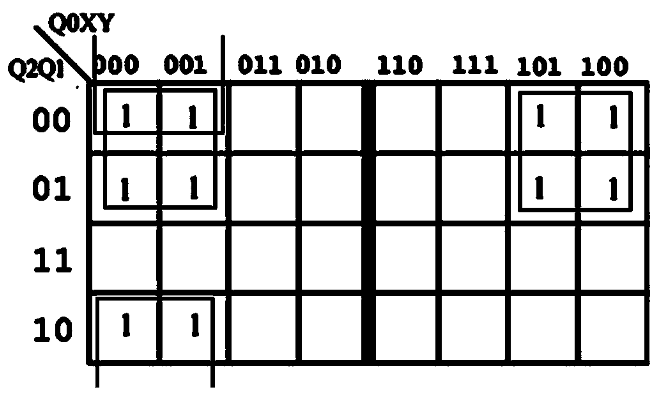


Figure 3

Obtain the boolean expression for the next state $Q0^{*}$ using the k-map shown in Figure3.

$$Q0^{*} = \overline{Q1}\cdot\overline{Q0}\cdot\overline{X} + \overline{Q2}\cdot\overline{X}$$

Obtain the boolean expression for the output Z from the transition table shown in table 1.

$$Z = \overline{Q2}\cdot Q1\cdot Q0 + Q2\cdot\overline{Q1}\cdot\overline{Q0}$$

Step 5 of 5

Since from the characteristic table of D flip-flop it is clear that input to the D flip-flop is itself the next state output.
Thus, the expressions for the inputs to the D flip-flops are,

$$D2 = Q2\cdot\overline{Q1}\cdot\overline{Q0}\cdot X + \overline{Q2}\cdot Q1\cdot\overline{Q0}\cdot X + \overline{Q2}\cdot Q1\cdot X\cdot Y$$

$$D1 = Q2\cdot\overline{Q1}\cdot\overline{Q0}\cdot\overline{X}\cdot Y + \overline{Q2}\cdot\overline{Q1}\cdot X + \overline{Q2}\cdot Q0\cdot\overline{X} + \overline{Q2}\cdot Q0\cdot\overline{Y}$$

$$D0 = \overline{Q1}\cdot\overline{Q0}\cdot\overline{X} + \overline{Q2}\cdot\overline{X}$$

$$Z = \overline{Q2}\cdot Q1\cdot Q0 + Q2\cdot\overline{Q1}\cdot\overline{Q0}$$

To realise the circuit with these expressions, neglecting the inverters the realisation need three 4-input gates, one 5-input gate, six 3-input gates and one 2-input gate.

Refer to the equations for minimal cost solution in page 566 from the text book.

$$D1 = 1$$
$$D2 = Q1.\overline{Q3}.\overline{A} + Q3.A + Q2.B$$
$$D3 = A$$

To realise the circuit with these expressions of minimal cost solution, it required relatively very less number of gates, two 2-input gates, 2 3-input gates are enough.

Thus, the cost of excitation and output logic **are more for the state table in Table 7-5** when compared to the minimal cost solution.

Step 1 of 6

7.048E

Refer to the Table 7-5 and Table 7-6 from the text book.

Using the Table 7-5 and the “almost one-hot assignment” in Table 7-6, the transition list is built as shown in Table 1.

Table 1

Q3Q2Q1Q0	XY	Z			
	00	**01**	**11**	**10**	
0000	0001	0001	0010	0010	0
0001	0100	0100	0010	0010	0
0010	0001	0001	1000	1000	0
0100	0100	0100	1000	0010	1
1000	0001	0100	1000	1000	1
$Q3^{*}Q2^{*}Q1^{*}Q0^{*}$					

Table 1

Step 2 of 6

Observe Table 1, for the present state and applied inputs, the next states are obtained.

Derive the expression for the next state $Q3^{*}$ from the transition table.

Draw the K-map for next state $Q3^{*}$ as shown in the Figure 1.

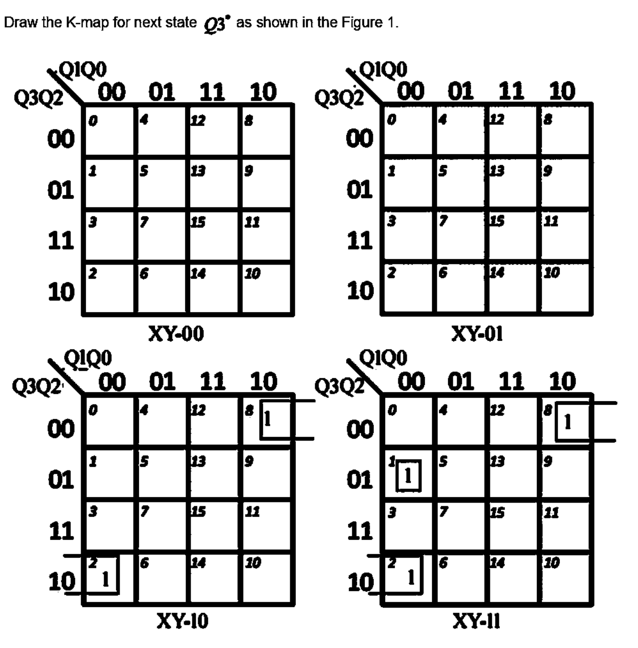


Figure 1

Obtain the boolean expression for the next state $Q3^{*}$ using the k-map shown in Figure 1.

$$Q3^{*}=\overline{Q3}\cdot Q2\cdot\overline{Q1}\cdot\overline{Q0}\cdot X\cdot Y+Q3\cdot\overline{Q2}\cdot\overline{Q1}\cdot\overline{Q0}\cdot X+\overline{Q3}\cdot\overline{Q2}\cdot Q1\cdot\overline{Q0}\cdot X$$

Step 3 of 6

Derive the expression for the next state $Q2^{*}$ from the transition table.

Draw the K-map for next state $Q2^{*}$ as shown in the Figure 2.

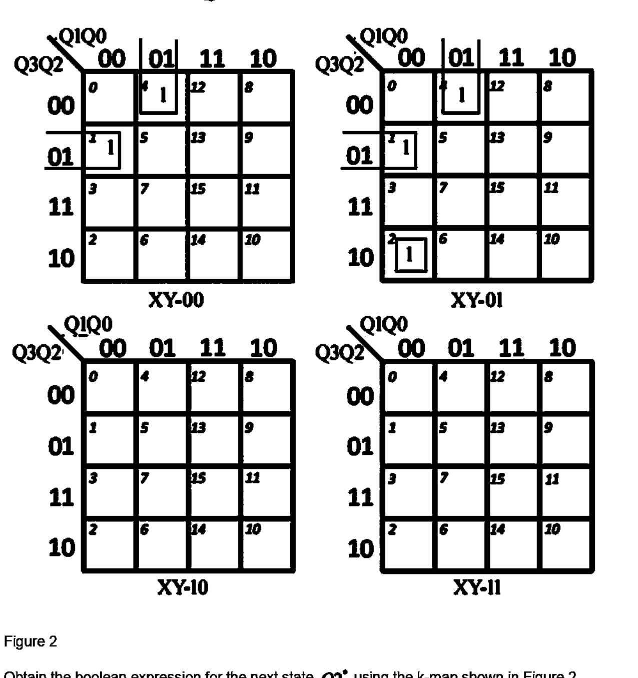


Figure 2

Obtain the boolean expression for the next state $Q2^{*}$ using the k-map shown in Figure 2.

$$Q2^{*}=\overline{Q3}\cdot Q2\cdot\overline{Q1}\cdot\overline{Q0}\cdot\overline{X}+Q3\cdot\overline{Q2}\cdot\overline{Q1}\cdot\overline{Q0}\cdot\overline{X}\cdot Y+\overline{Q3}\cdot\overline{Q2}\cdot\overline{Q1}\cdot Q0\cdot\overline{X}$$

Step 4 of 6

Derive the expression for the next state $Q1^{*}$ from the transition table.

Draw the K-map for next state $Q1^{*}$ as shown in the Figure 3.

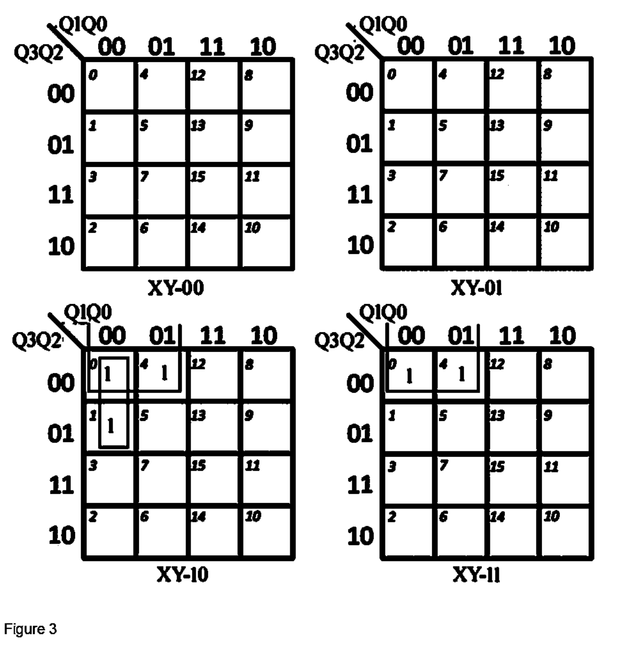


Figure 3

Obtain the boolean expression for the next state $Q1^{*}$ using the k-map shown in Figure 3.

$$Q1^{*}=\overline{Q3}\cdot\overline{Q2}\cdot\overline{Q1}\cdot X+\overline{Q3}\cdot\overline{Q1}\cdot\overline{Q0}\cdot X\cdot\overline{Y}$$

Step 5 of 6

Derive the expression for the next state $Q0^{*}$ from the transition table.

Draw the K-map for next state $Q0^{*}$ as shown in the Figure 4.

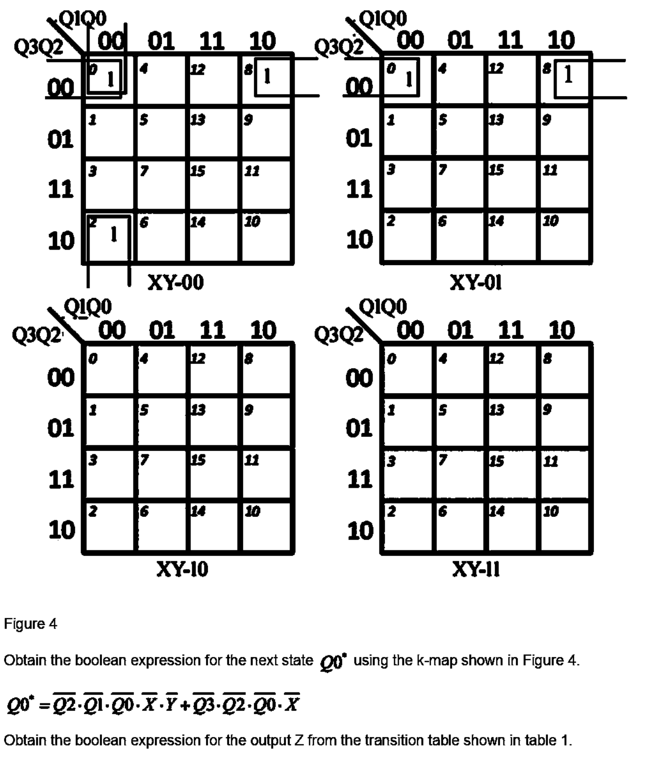


Figure 4

Obtain the boolean expression for the next state $Q0^{*}$ using the k-map shown in Figure 4.

$$Q0^{*}=\overline{Q2}\cdot\overline{Q1}\cdot\overline{Q0}\cdot\overline{X}\cdot\overline{Y}+\overline{Q3}\cdot\overline{Q2}\cdot\overline{Q0}\cdot\overline{X}$$

Obtain the boolean expression for the output Z from the transition table shown in table 1.

$$Z=\overline{Q3}\cdot Q2\cdot\overline{Q1}\cdot\overline{Q0}+Q3\cdot\overline{Q2}\cdot\overline{Q1}\cdot\overline{Q0}$$

Step 6 of 6

Since from the characteristic table of D flip-flop it is clear that input to the D flip-flop is itself the next state output.

Thus, the expressions for the inputs to the D flip-flops and output epression are,

$$D3=\overline{Q3}\cdot Q2\cdot\overline{Q1}\cdot\overline{Q0}\cdot X\cdot Y+Q3\cdot\overline{Q2}\cdot\overline{Q1}\cdot\overline{Q0}\cdot X+\overline{Q3}\cdot\overline{Q2}\cdot Q1\cdot\overline{Q0}\cdot X$$

$$D2=\overline{Q3}\cdot Q2\cdot\overline{Q1}\cdot\overline{Q0}\cdot\overline{X}+Q3\cdot\overline{Q2}\cdot\overline{Q1}\cdot\overline{Q0}\cdot\overline{X}\cdot Y+\overline{Q3}\cdot\overline{Q2}\cdot\overline{Q1}\cdot Q0\cdot\overline{X}$$

$$D1=\overline{Q2}\cdot\overline{Q1}\cdot\overline{Q0}\cdot\overline{X}\cdot\overline{Y}+\overline{Q3}\cdot\overline{Q2}\cdot\overline{Q0}\cdot\overline{X}$$

$$D0=\overline{Q2}\cdot\overline{Q1}\cdot\overline{Q0}\cdot\overline{X}\cdot\overline{Y}+\overline{Q3}\cdot\overline{Q2}\cdot\overline{Q0}\cdot\overline{X}$$

$$Z=\overline{Q3}\cdot Q2\cdot\overline{Q1}\cdot\overline{Q0}+Q3\cdot\overline{Q2}\cdot\overline{Q1}\cdot\overline{Q0}$$

To realise the circuit with these expressions, neglecting the inverters the realisation need two 6-input gates, six 5-input gate and two 4-input gates.

Refer to the equations for minimal cost solution in page 566 from the text book.

$$D1=1$$
$$D2=Q1.\overline{Q3}.\overline{A}+Q3.A+Q2.B$$
$$D3=A$$

To realise the circuit with these expressions of minimal cost solution, it required relatively very less number of gates, two 2-input gates, 2 3-input gates are enough.

Thus, the cost of excitation and output logic **are more for the state table in Table 7-5** when compared to the minimal cost solution.

Step 1 of 4

Refer to the excitation equations in the page number 566.

$$D1 = 1$$
$$D2 = Q1.\overline{Q3}.\overline{A} + Q3.A + Q2.B$$
$$D3 = A$$

Consider that the present states as $(Q3\ Q2\ Q1) = (000)$ and inputs as $AB = 00$.

Obtain the inputs to the D flip-flops using excitation equations.

$$D1 = 1$$
$$D2 = 0.1.1 + 0.0 + Q0.0$$
$$= 0$$
$$D3 = 0$$

For there are 3 flipflops, so total 2^3, that is 8 combnations are to be checked.

Using the excitation equations in the box on page 566, the transition list can be built as shown in Table 1.

Table 1

Q3Q2Q1	AB	Z			
	00	01	10	11	
000	100	100	101	101	0
001	110	110	101	101	0
010	100	110	101	111	1
011	110	110	101	111	1
100	100	100	111	111	0
101	100	100	111	111	0
110	100	110	111	111	1
111	100	110	111	111	1
D1.D2.D3					

Step 2 of 4

Consider each input combination is considered as one state and as there are 8 input combinations, so total 8 states are available.

The states are from S_0 to S_7.

The full 8-state table is obtained using transition list in Table 1 and is as shown in Table2.

Table 2

Q3Q2Q1	AB	Z			
	00	01	10	11	
S0	S4	S4	S5	S5	0
U1	S6	S6	S5	S5	0
U2	S4	S6	S5	S7	1
U3	S6	S6	S5	S7	1
S4	S4	S4	S7	S7	0
S5	S4	S4	S7	S7	0
S6	S4	S6	S7	S7	1
S7	S4	S6	S7	S7	1
D1.D2.D3					

Step 3 of 4

The state diagram is drawn using state table in the Table 2 and is as shown in Figure 1.

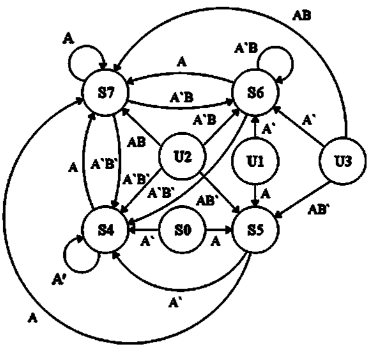


Figure 1

Step 4 of 4

U1, U2 and U3 are the unused states in the Figure 1. Figure 1 shows all the 8 states.

Thus, the state diagram is drawn.

Step 1 of 4

7.050E

Refer state table in Figure 7-49(a).

Using the state table and the counting order from 000-110 the new transition table is obtained as shown in table 1.

Q2Q1Q0	AB	Z			
	00	01	11	10	
000	001	001	010	010	0
001	011	011	010	010	0
010	001	001	100	100	0
011	011	011	110	010	1
100	001	101	100	100	1
101	011	011	110	010	1
110	001	101	100	100	1
$Q2^{*}Q1^{*}Q0^{*}$					

Table 1

Step 2 of 4

Assuming a “minimal cost” treatment of unused states, the table 1 leads to the following K-maps for the excitation logic as shown in Figure 1.

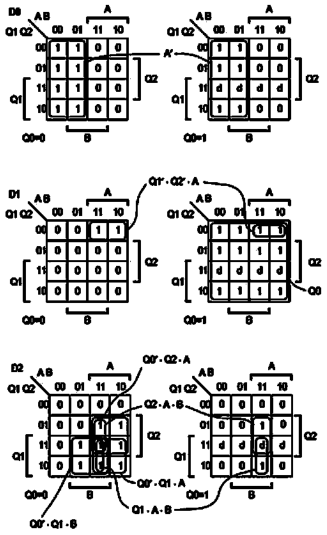


Figure 1

Step 3 of 4

The resulting excitation equations are,

$$D0 = A'$$

$$D1 = Q1' \cdot Q2' \cdot A + Q0$$

$$D2 = Q2 \cdot A \cdot B + Q0' \cdot Q2 \cdot A + Q0' \cdot Q1 \cdot A + Q1 \cdot A \cdot B + Q0' \cdot Q1 \cdot B$$

Step 4 of 4

From the excitation equations, the circuit realisation with the new equations requires one 2-input gate, one 5-input gate and six 3-input gates. So, this circuit is expensive.

Step 1 of 4

Use the Table 7-5 and build the state table as shown in Table 1.

Table 1

S	AB				Z
	00	01	11	10	
A0	OK0	OK0	A1	A1	0
A1	A0	A0	OK1	OK1	0
OK0	OK0	OK0	OK1	A1	1
OK1	A0	OK0	OK1	OK1	1
S^*					

Step 2 of 4

The states are assigned as follows:

$A0 = 00$
$A1 = 01$
$OK0 = 10$
$OK1 = 11$

Use the state table obtained in the Table 1 and build the transition list as shown in the Table 2.

Table 2

Q1Q0	AB				Z
	00	01	11	10	
00	10	10	01	01	0
01	00	00	11	11	0
10	10	10	11	01	1
11	00	10	11	11	1
$Q1^*Q0^*$					

Step 3 of 4

The transition equations are derived using the transition list in table 2 and the K-maps as shown.

Draw the k-map and obtain the expression for $Q1^*$ as shown in Figure 1.

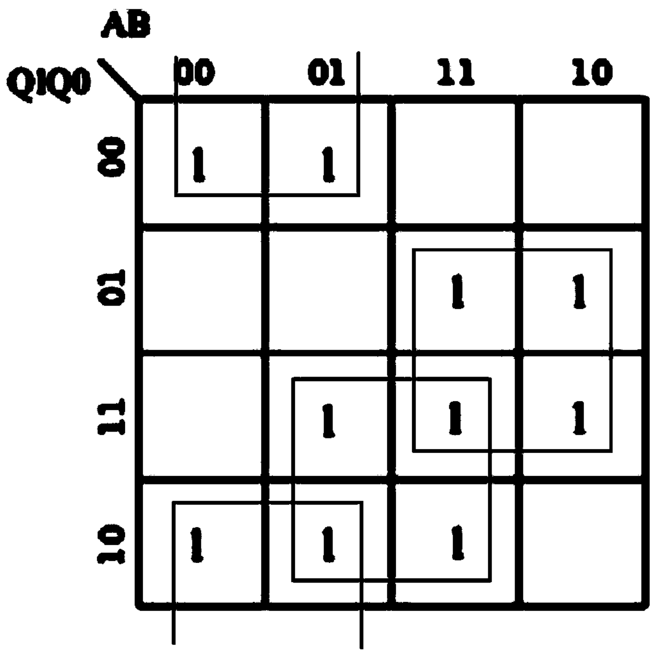


Figure 1

Obtain the boolean expression for $Q1^*$ from the Figure 1.

$$Q1^* = \overline{Q0} \cdot \overline{A} + Q1 \cdot B + Q0 \cdot A$$

Step 4 of 4

Draw the k-map and obtain the expression for $Q0^*$ as shown in Figure 2.

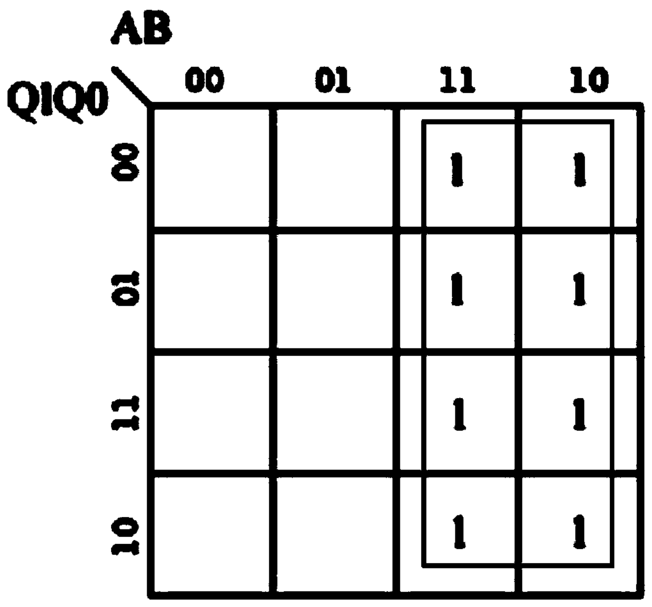


Figure 2

Obtain the boolean expression for $Q2^*$ from the Figure 2.

$$Q0^* = A$$

Obtain the expression for the output Z.

$$\begin{aligned} Z &= Q1 \cdot \overline{Q0} + Q1 \cdot Q0 \\ &= Q1 \end{aligned}$$

The cost is same as that of the the minimal risk design that was computed in the section 7.4.4. This realisation needs three 2-input gates.

Step 1 of 4

Refer Table 7-9 from the text book.

Using the table and by changing the order of the states to S0, S1, S3, S2 the new state yields the following state table as shown in Table 1.

Q1Q0	XY	Z			
	00	01	11	10	
S0	S0	S1	S3	S1	1
S1	S1	S3	S2	S3	0
S3	S3	S2	S0	S2	0
S2	S2	S0	S1	S0	0
$Q1^{*}Q0^{*}$					

Table 1

Step 2 of 4

The K-maps using the table 1 is as shown in Figure 1.

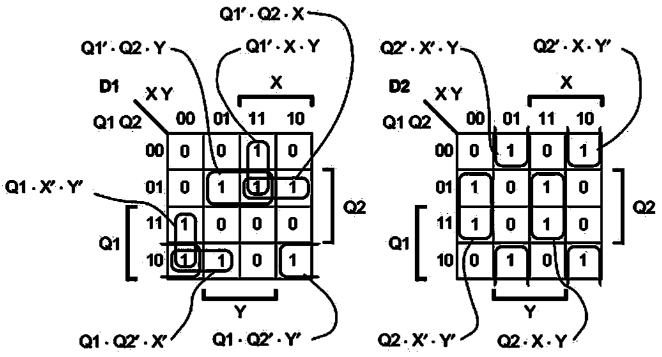


Figure 1

Step 3 of 4

The excitation equations are,

$$D1 = Q1' \cdot Q2 \cdot X + Q1' \cdot Q2 \cdot Y + Q1' \cdot X \cdot Y + Q1 \cdot Q2' \cdot X' + Q1 \cdot Q2' \cdot Y' + Q1 \cdot X' \cdot Y'$$

$$D2 = Q2 \cdot X \cdot Y + Q2' \cdot X \cdot Y' + Q2' \cdot X' \cdot Y + Q2 \cdot X' \cdot Y'$$

Step 4 of 4

From the excitation equations, the circuit realisation with the new equations requires two more 3-input AND gates and a 6-input OR gate instead of a 4-input OR gate.

Step 1 of 4

Refer Table 7-9 from the text book.

Using the table, the new state yields the following transition list as shown in Table 1.

Q1Q0	**XY**	**Z**			
	00	**01**	**11**	**10**	
00	00	01	11	01	1
01	01	11	10	11	0
11	11	10	00	10	0
10	10	00	01	00	0
$Q1^{*}Q0^{*}$					

Table 1

Step 2 of 4

The K-maps using the table 1 is shown in Figure 1.

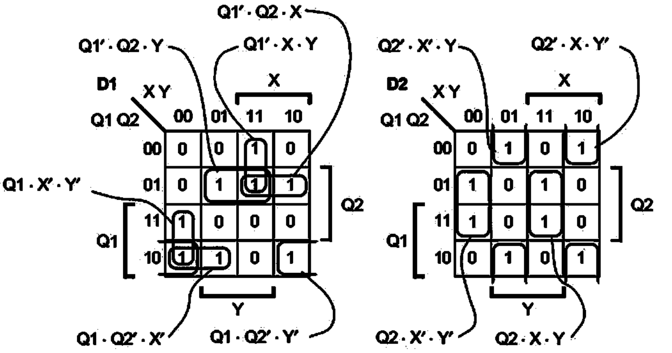


Figure 1

Step 3 of 4

The excitation equations are,

$$D1 = Q1'\cdot Q2\cdot X + Q1'\cdot Q2\cdot Y + Q1'\cdot X\cdot Y + Q1\cdot Q2'\cdot X' + Q1\cdot Q2'\cdot Y' + Q1\cdot X'\cdot Y'$$

$$D2 = Q2\cdot X\cdot Y + Q2'\cdot X\cdot Y' + Q2'\cdot X'\cdot Y + Q2\cdot X'\cdot Y'$$

Step 4 of 4

These equations require two more 3-input AND gates and a 6-input OR gate instead of a 4-input OR gate. If we are not restricted to a sum of products then $D2 = Q2 \oplus X \oplus Y$ might make the realisation less expensive.

Step 1 of 6

Using the Table 7-11 and the state order as Gray-code order, the transition list can be built as shown in Table 1.

Table 1

Q2Q1Q0	X	
	0	1
000	001,01	000,00
001	001,00	011,01
011	001,00	010,01
010	110,01	000,00
110	001,00	111,01
111	001,00	101,01
101	110,00	100,01
100	001,11	000,00
$Q2^{*}Q1^{*}Q0^{*}$,*UNLK HINT*		

Step 2 of 6

Obtain the transition equations using the transition list shown in Table 1.

Obtain the k-map for $Q2^{*}$ as shown in Figure 1.

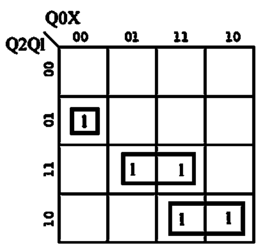


Figure 1

Obtain the boolean expression for $Q2^{*}$.

$$Q2^{*}=\overline{Q2}\cdot Q1\cdot\overline{Q0}\cdot\overline{X}+Q2\cdot Q1\cdot X+Q2\cdot\overline{Q1}\cdot Q0$$

Step 3 of 6

Obtain the k-map for $Q1^{*}$ as shown in Figure 2.

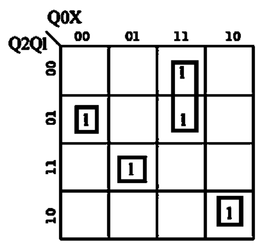


Figure 2

Obtain the boolean expression for $Q2^{*}$.

$$Q1^{*}=\overline{Q2}\cdot Q1\cdot\overline{Q0}\cdot\overline{X}+Q2\cdot Q1\cdot\overline{Q0}\cdot X+Q2\cdot\overline{Q1}\cdot Q0\cdot\overline{X}+\overline{Q2}\cdot Q0\cdot X$$

Step 4 of 6

Obtain the k-map for $Q0^{*}$ as shown in Figure 3.

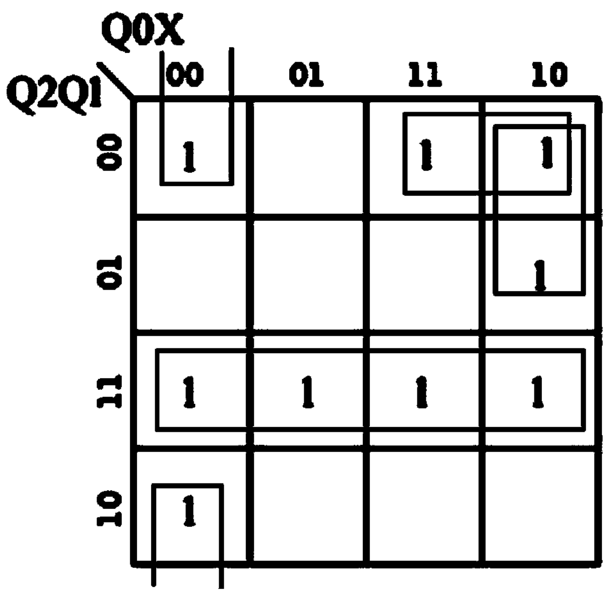


Figure 3

Obtain the boolean expression for $Q0^{*}$.

$$Q0^{*}=\overline{Q1}\cdot\overline{Q0}\cdot\overline{X}+Q2\cdot Q1+\overline{Q2}\cdot\overline{Q1}\cdot Q0+\overline{Q2}\cdot Q0\cdot\overline{X}$$

Obtain the boolean expression for *UNLK* directly obtained from the transition list.

$$UNLK=Q2\cdot\overline{Q1}\cdot\overline{Q0}\cdot\overline{X}$$

Step 5 of 6

Obtain the k-map for *HINT* as shown in Figure 4.

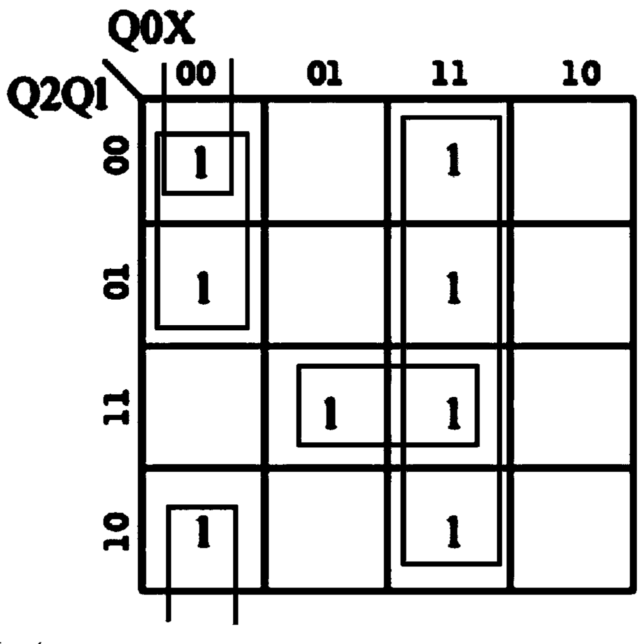


Figure 4

Obtain the boolean expression for *HINT* .

$$HINT=\overline{Q1}\cdot\overline{Q0}\cdot\overline{X}+\overline{Q2}\cdot\overline{Q0}\cdot\overline{X}+Q2\cdot Q1\cdot X+Q0\cdot X$$

Step 6 of 6

The cost is same as that of the one that is derived in the text. This realisation needs five 4-input gates, nine 3-input gates and two 2-input gates.

Step 1 of 8

Refer to Table 7-11 from the textbook for the state and output table for combination-lock machine. Consider a fact that the inputs 1 to 3 are the same as inputs 4 to 6 in the required input sequence.

Consider the following expressions for **D1, D2, D3, UNLK and HINT** from the textbook:

$$D1 = Q1 \cdot Q2' \cdot X + Q1' \cdot Q2 \cdot Q3 \cdot X' + Q1 \cdot Q2 \cdot Q3'$$
$$D2 = Q2' \cdot Q3 \cdot X + Q2 \cdot Q3' \cdot X$$
$$D3 = Q1 \cdot Q2' \cdot Q3' + Q1 \cdot Q3 \cdot X' + Q2' \cdot X' + Q3' \cdot Q1' \cdot X' + Q2 \cdot Q3' \cdot X$$

$$UNLK = Q1 \cdot Q2 \cdot Q3 \cdot X'$$
$$HINT = Q1' \cdot Q2' \cdot Q3' \cdot X' + Q1 \cdot Q2' \cdot X + Q2' \cdot Q3 \cdot X + Q2 \cdot Q3 \cdot X' + Q2 \cdot Q3' \cdot X$$

Step 2 of 8

Assign 3-bits for the states of the combination-lock machine that results in less costly excitation equations than the ones derived in the text.

Table 1

Y1Y2Y3	X	
	0	1
011	111,01	011,00
111	111,00	000,01
000	111,00	001,01
001	110,01	011,00
110	111,00	010,01
010	111,00	100,01
100	110,00	101,01
101	111,11	011,00
D1D2D3,UNLK,HINT		

Step 3 of 8

Derive the exciataion equations using the transition list from the Table 1 and Karnaugh maps.

Find the exciataion equation for D1.

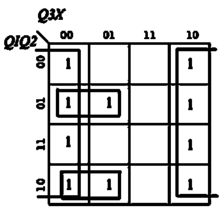


Figure 1

Step 4 of 8

Derive the expresion from Figure 1.

$$D1 = \overline{X} + Q1 \cdot \overline{Q2} \cdot \overline{Q3} + \overline{Q1} \cdot Q2 \cdot \overline{Q3}$$

Step 5 of 8

Find the exciataion equation for D2.

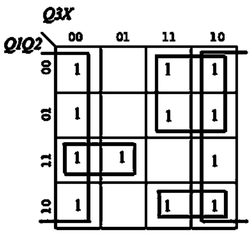


Figure 2

Step 6 of 8

Derive the expresion from Figure 2.

$$D2 = \overline{X} + Q1 \cdot Q2 \cdot \overline{Q3} + \overline{Q1} \cdot Q3 + Q1 \cdot \overline{Q2} \cdot Q3$$

Step 7 of 8

Find the exciataion equation for D3.

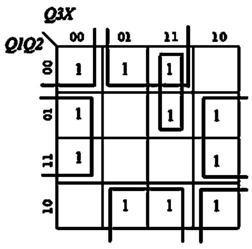


Figure 3

Step 8 of 8

Derive the expresion from Figure 3.

$$D3 = \overline{X} \cdot Q2 + \overline{Q2} \cdot X + \overline{Q2} \cdot \overline{X} + \overline{Q1} \cdot Q3 \cdot X$$

The system requires six 3-input gates, four 2-input gates and two 4-input gates.

Thus, the system is less costly than the system in the text book.

We don't have the solution to this problem yet.

Get help from a Chegg subject expert.

Ask an expert

Step 1 of 2 **7.057E**

Refer to the Table 7.47 in the textbook.

Refferd Table 7.47 consists of VHDL program for finite-memory design of combinational-lock state machine excluding the HINT output from the oroginal state-machine specifications.

Modify the Table 7.47 such that it include the HINT output from the oroginal state-machine specifications and it is shown as follows:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity Vcomblck is
port(
HINT, UNLK: out STD_LOGIC;
CLK, RESET, X: in STD_LOGIC
);
end Vcomblck;
architecture Vcomblck_arch of Vcomblck is
signal XHISTORY: STD_LOGIC_VECTOR(7 downto 1);
signal Q: UNSIGNED(2 downto 0);
constant combination: STD_LOGIC_VECTOR(7 downto 1):="0110111";
begin
process(CLK)
– State memory and next state logic
begin
Q <= "000";
if(CLK'event and CLK='1') then
if RESET = '1' then XHISTORY <= "0000000";
else XHISTORY <= XHISTORY(6 downto 1) & X;
end if;
Q <= Q + 1;
end if;
end process;
– Output logic
UNLK <= '1';
HINT <= ((not Q(2) and not Q(1) and not Q(0) and not X) or (Q(2) and not Q(1) and X) or (not Q(1) and
Q(0) and X) or (Q(1) and Q(0) and not X) or (Q(1) and not Q(0) and X)) when((XHISTORY=combination)
and (X='0')) else '0';
end Vcomblck_arch;
```

Step 2 of 2

Hence, the required VHDL program for finite-memory design of combinational-lock state machine inluding the HINT output from the oroginal state-machine specifications is done.

Step 1 of 1 7.058E

Write a Verilog program for the original state-machine along with included the HINT output.

```
module Vcomblck(HINT, UNLK, CLK, RESET, X);

output HINT;

output UNLK;

input CLK;

input RESET;

input X;

reg [7:1] XHISTORY;

reg [1:0] Q;

parameter [7:1] combination = 7'b0110111;

always @(CLK)

begin

// State memory and next state logic

Q <= 3’b000;

if (CLK == 1’b1)

begin

if (RESET == 1'b1)

XHISTORY <= 7'b0000000;

else

XHISTORY <= {XHISTORY[6:1], X};

Q <= Q + 1;

end

end

// Output logic

assign UNLK = 1'b1;

assign HINT = (((XHISTORY == combination) & (X == 1'b0))) ? (((~Q[2]) & (~Q[1]) & (~Q[0]) & (~X)) | (Q[2]
& (~Q[1]) & X) | ((~Q[1]) & Q[0] & X) | (Q[1] & Q[0] & (~X)) | (Q[1] & (~Q[0]) & X)) : 1’b0;

endmodule
```

Thus, state-machine has been implemented using Xlinx 13.2.

7.059E

Step 1 of 2

Refer to Table 7-11 from the textbook for the state and output table for combination-lock machine. The following are the expressions for **D1, D2, D3, UNLK and HINT** from the textbook:

$$D1 = Q1 \cdot Q2' \cdot X + Q1' \cdot Q2 \cdot Q3 \cdot X' + Q1 \cdot Q2 \cdot Q3'$$
$$D2 = Q2' \cdot Q3 \cdot X + Q2 \cdot Q3' \cdot X$$
$$D3 = Q1 \cdot Q2' \cdot Q3' + Q1 \cdot Q3 \cdot X' + Q2' \cdot X' + Q3' \cdot Q1' \cdot X' + Q2 \cdot Q3' \cdot X$$

$$UNLK = Q1 \cdot Q2 \cdot Q3 \cdot X'$$
$$HINT = Q1' \cdot Q2' \cdot Q3' \cdot X' + Q1 \cdot Q2' \cdot X + Q2' \cdot Q3 \cdot X + Q2 \cdot Q3 \cdot X' + Q2 \cdot Q3' \cdot X$$

Step 2 of 2

Draw the logic diagram for the circuit of combination-lock machine using the flip-flops and combinational output logic.

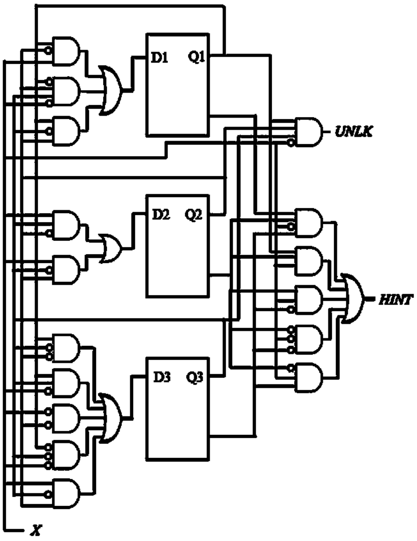


Figure 1

Step 1 of 5

The transition list is shown in the table 1.

S	$Q2Q1Q0$	Transition Expression	S^*	$Q2^*Q1^*Q0^*$
IDLE	000	$(\text{LEFT}+\text{RIGHT}+\text{HAZ})'$	IDLE	000
IDLE	000	LEFT	L1	001
IDLE	000	HAZRIGHT	LR3	100
IDLE	000	1	R1	101
L1	001	1	L2	011
L2	011	1	L3	010
L3	010	1	IDLE	000
R1	101	1	R2	111
R2	111	1	R3	110
R3	110	1	IDLE	000
LR3	100		IDLE	000

Table 1

Step 2 of 5

Derive the transition and output equations from the Table 1.

$$\begin{aligned}
D2 &= Q2^* \\
&= Q2'\cdot Q1'\cdot Q0'\cdot HAZ + Q2'\cdot Q1'\cdot Q0'\cdot RIGHT + Q2\cdot Q1'\cdot Q0 + Q2\cdot Q1\cdot Q0 \\
&= Q2'\cdot Q1'\cdot Q0'\cdot(\text{HAZ}+\text{RIGHT}) + Q2\cdot Q0(Q1'+Q1) \\
&= Q2'\cdot Q1'\cdot Q0'\cdot(\text{HAZ}+\text{RIGHT}) + Q2\cdot Q0
\end{aligned}$$

$$\begin{aligned}
D1 &= Q1^* \\
&= Q2'\cdot Q1'\cdot Q0 + Q2'\cdot Q1\cdot Q0 + Q2\cdot Q1'\cdot Q0 + Q2\cdot Q1\cdot Q0 \\
&= Q2'\cdot Q0(Q1'+Q1) + Q2\cdot Q0(Q1'+Q1) \\
&= Q2'\cdot Q0 + Q2\cdot Q0
\end{aligned}$$

$$= (Q2'+Q2)\cdot Q0$$
$$= Q0$$

$$\begin{aligned}
D0 &= Q0^* \\
&= Q2'\cdot Q1'\cdot Q0'\cdot LEFT + Q2'\cdot Q1'\cdot Q0'\cdot RIGHT + Q2'\cdot Q1'\cdot Q0 + Q2\cdot Q1'\cdot Q0 \\
&= Q2'\cdot Q1'\cdot Q0'\cdot(\text{LEFT}+\text{RIGHT}) + Q1'\cdot Q0(Q2'+Q2) \\
&= Q2'\cdot Q1'\cdot Q0'\cdot(\text{LEFT}+\text{RIGHT}) + Q1'\cdot Q0
\end{aligned}$$

Step 3 of 5

Starting from the idle state, the following transitions may be observed as shown in Table 2.

S	$Q2Q1Q0$	LEFT	RIGHT	HAZ	$Q2^*Q1^*Q0^*$	S^*
IDLE	000	1	0	1	101	R1
IDLE	000	0	1	1	101	R1
IDLE	000	1	1	0	101	R1
IDLE	000	1	1	1	101	R1

Table 2

Step 4 of 5

For each input combination, the machine goes to the R1 state, because R1's encoding is the logical OR of the encodings of the two or three next states that are specified by the ambiguous state diagram.

The behavior above is not so good and it is the result of synthesis choices, state encoding and logic synthesis method. If a different state encoding was used for R1, or if a different synthesis method was used, then the results could be different. For example, starting with the transition list as shown in Table 2, we can obtain the following set of transition equations using the product-of-sum-terms method:

$$\begin{aligned}
D2 &= Q2^* \\
&= \left\{\begin{matrix} (Q2+Q1+Q0+\text{LEFT}+\text{RIGHT}+\text{HAZ}) \\ \cdot(Q2+Q1+Q0+\text{LEFT}')\cdot(Q2+Q1+Q0')\cdot(Q2+Q1'+Q0') \\ \cdot(Q2+Q1'+Q0)\cdot(Q2'+Q1'+Q0)\cdot(Q2'+Q1+Q0) \end{matrix}\right\} \\
&= \left\{\begin{matrix} (Q2+Q1+\text{RIGHT}+\text{HAZ})\cdot(Q2+Q1+\text{LEFT}') \\ \cdot(Q2+Q0')\cdot(Q1'+Q0)\cdot(Q2'+Q0) \end{matrix}\right\}
\end{aligned}$$

$$\begin{aligned}
D1 &= Q1^* \\
&= \left\{\begin{matrix} (Q2+Q1+Q0+\text{LEFT}+\text{RIGHT}+\text{HAZ})\cdot(Q2+Q1+Q0+\text{LEFT}') \\ \cdot(Q2+Q1+Q0+\text{HAZ}')\cdot(Q2+Q1+Q0+\text{RIGHT}') \\ \cdot(Q2+Q1'+Q0)\cdot(Q2'+Q1'+Q0)\cdot(Q2'+Q1+Q0) \end{matrix}\right\} \\
&= \{(Q2+Q1+Q0)\cdot(Q1'+Q0)\cdot(Q2'+Q0)\}
\end{aligned}$$

$$\begin{aligned}
D0 &= Q0^* \\
&= \left\{\begin{matrix} (Q2+Q1+Q0+\text{LEFT}+\text{RIGHT}+\text{HAZ})\cdot(Q2+Q1+Q0+\text{HAZ}') \\ \cdot(Q2+Q1'+Q0')\cdot(Q2+Q1'+Q0)\cdot(Q2'+Q1'+Q0') \\ \cdot(Q2'+Q1'+Q0)\cdot(Q2'+Q1+Q0) \end{matrix}\right\} \\
&= \left\{\begin{matrix} (Q2+Q0+\text{LEFT}+\text{RIGHT})\cdot(Q2+Q0+\text{HAZ}') \\ \cdot(Q1')\cdot(Q2'+Q0) \end{matrix}\right\}
\end{aligned}$$

Step 5 of 5

These equations yield the following transaction table as shown in table 3.

S	$Q2Q1Q0$	LEFT	RIGHT	HAZ	$Q2^*Q1^*Q0^*$	S^*
IDLE	000	0	0	0	000	IDLE
IDLE	000	0	1	1	100	LR3
IDLE	000	1	1	0	001	L1
IDLE	000	1	1	1	000	IDLE

Table 3

The behavior is obviously different but it is still not particularly a good behavior.

7.062E

Step 1 of 1

The personalized plate in the Figure 7-54 contains OTTFFS. Actually OTTFFS is a pattern, each letter indicates the how the number is spelled. This pattern used in Moore machine, each letter or group of letters indicates the definite state. For example car left trun, the machine should cycle through four states in which the righthand lights FFS are off and OTT are one, like wise, for a right trun, it should cycle through four states in which the lefthand lights OTT are off and FFS of the righthand lights are on.

The computer engineer's version of OTTFFS is as follows:

- O – One
- T – Two
- T – Three
- F – Four
- F – Five
- S – Six

Step 1 of 2

The following VHDL code is executed for finding the fibonacci sequence for a adesired number.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity Fib is
port(
FIB: out UNSIGNED(3 downto 0);
DONE: out STD_LOGIC;
CLK, RESET, X: in STD_LOGIC;
D: in UNSIGNED(3 downto 0)
);
end Fib;
architecture Fib_arch of Fib is
signal C: UNSIGNED(2 downto 0);
signal E,F,G: UNSIGNED(3 downto 0);
begin
process(CLK)
begin
FIB <= "0000";
if (CLK'event and CLK='1') then
if(RESET='1') then
C <= "000";
FIB <= D;
C <= C + 1;
end if;
if(C=1) then
FIB <= D;
G <= D;
C <= C + 1;
end if;
if(C=2) then
FIB <= G + D;
G <= G + D;
C <= C + 1;
end if;
if(C=3) then
E <= G;
FIB <= G + D;
G <= G + D;
C <= C + 1;
end if;
if(C=4) then
F <= G;
FIB <= G + E;
G <= G + E;
C <= C + 1;
end if;
if(C=5) then
FIB <= G + F;
C <= C + 1;
end if;
end if;
end process;
end Fib_arch;
```

Step 2 of 2

here, the machnie has two additional inputs along with the clock. They are RESET and *n*-bit input bus D for loading the first two defined fibonacci sequences in the output.

The Mchnie should load the second defined Fibonacci number in sequence from the D bus after the first clock tick. This process continues for the required number and then WAITS for the DONE and need to be RESET again.

Hence, the required VHDL programm is written.

Step 1 of 3 **7.065E**

The test bench for the program in the Fibonacci sequence.

Testbench:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE STD.TEXTIO.ALL;
ENTITY Fibo IS
END Fibo;
ARCHITECTURE testbench_arch OF Fibo IS
FILE RESULTS: TEXT OPEN WRITE_MODE IS "results.txt";
COMPONENT Fi
PORT (
CLK : In std_logic;
RESET : In std_logic;
D : InOut std_logic_vector (3 DownTo 0);
FIB : InOut std_logic_vector (3 DownTo 0);
DONE : Out std_logic
);
END COMPONENT;
SIGNAL CLK : std_logic := '0';
SIGNAL RESET : std_logic := '0';
SIGNAL D : std_logic_vector (3 DownTo 0) := "0000";
SIGNAL FIB : std_logic_vector (3 DownTo 0) := "0000";
SIGNAL DONE : std_logic := '0';
SHARED VARIABLE TX_ERROR : INTEGER := 0;
SHARED VARIABLE TX_OUT : LINE;
constant PERIOD : time := 200 ns;
constant DUTY_CYCLE : real := 0.5;
constant OFFSET : time := 0 ns;
BEGIN
UUT : Fi
PORT MAP (
CLK => CLK,
RESET => RESET,
D => D,
FIB => FIB,
DONE => DONE
);
PROCESS -- clock process for CLK
BEGIN
WAIT for OFFSET;
CLOCK_LOOP : LOOP
CLK <= '0';
WAIT FOR (PERIOD - (PERIOD * DUTY_CYCLE));
CLK <= '1';
WAIT FOR (PERIOD * DUTY_CYCLE);
END LOOP CLOCK_LOOP;
END PROCESS;
PROCESS
PROCEDURE CHECK_DONE(
next_DONE : std_logic;
TX_TIME : INTEGER
) IS
VARIABLE TX_STR : String(1 to 4096);
VARIABLE TX_LOC : LINE;
BEGIN
IF (DONE /= next_DONE) THEN
STD.TEXTIO.write(TX_LOC, string'("Error at time="));
STD.TEXTIO.write(TX_LOC, TX_TIME);
STD.TEXTIO.write(TX_LOC, string'("ns DONE="));
IEEE.STD_LOGIC_TEXTIO.write(TX_LOC, DONE);
STD.TEXTIO.write(TX_LOC, string'(", Expected = "));
IEEE.STD_LOGIC_TEXTIO.write(TX_LOC, next_DONE);
STD.TEXTIO.write(TX_LOC, string'(" "));
TX_STR(TX_LOC.all'range) := TX_LOC.all;
STD.TEXTIO.writeline(RESULTS, TX_LOC);
STD.TEXTIO.Deallocate(TX_LOC);
ASSERT (FALSE) REPORT TX_STR SEVERITY ERROR;
TX_ERROR := TX_ERROR + 1;
END IF;
END;
BEGIN
-- ------------ Current Time: 85ns
WAIT FOR 85 ns;
RESET <= '1';
D <= "0001";
FIB <= "0001";
```

Step 2 of 3

```
-- ------------------------------------
-- ------------ Current Time: 285ns
WAIT FOR 200 ns;
RESET <= '0';
-- ------------------------------------
-- ------------ Current Time: 485ns
WAIT FOR 200 ns;
FIB <= "0010";
-- ------------------------------------
-- ------------ Current Time: 685ns
WAIT FOR 200 ns;
D <= "0010";
FIB <= "0011";
-- ------------------------------------
-- ------------ Current Time: 885ns
WAIT FOR 200 ns;
D <= "0011";
FIB <= "0101";
-- ------------------------------------
-- ------------ Current Time: 1085ns
WAIT FOR 200 ns;
D <= "0101";
FIB <= "1000";
-- ------------------------------------
-- ------------ Current Time: 1115ns
WAIT FOR 30 ns;
CHECK_DONE('1', 1115);
-- ------------------------------------
-- ------------ Current Time: 1285ns
WAIT FOR 170 ns;
D <= "1000";
FIB <= "1101";
-- ------------------------------------
-- ------------ Current Time: 1315ns
WAIT FOR 30 ns;
CHECK_DONE('0', 1315);
-- ------------------------------------
WAIT FOR 385 ns;
IF (TX_ERROR = 0) THEN
STD.TEXTIO.write(TX_OUT, string'("No errors or warnings"));
STD.TEXTIO.writeline(RESULTS, TX_OUT);
ASSERT (FALSE) REPORT
"Simulation successful (not a failure). No problems detected."
SEVERITY FAILURE;
ELSE
STD.TEXTIO.write(TX_OUT, TX_ERROR);
STD.TEXTIO.write(TX_OUT,
string'(" errors found in simulation"));
STD.TEXTIO.writeline(RESULTS, TX_OUT);
ASSERT (FALSE) REPORT "Errors found during simulation"
SEVERITY FAILURE;
END IF;
END PROCESS;
END testbench_arch;
```

Step 3 of 3

The waveforms are as shown in the Figure 1.

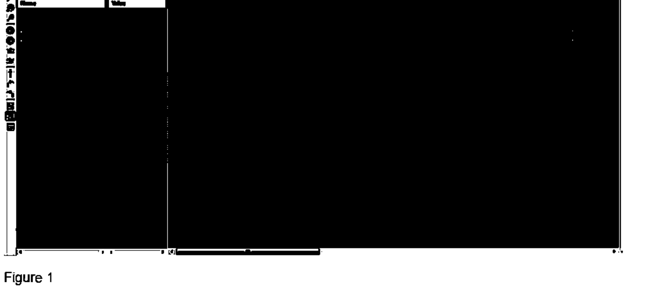


Figure 1

Assume the state up-to 13 in the Fibonacci sequence. And D holds the previous outputs. When RESET is asserted it starts counting the sequence.

Step 1 of 1

7.066E

Assume $E(\mathrm{SB})$, $E(\mathrm{SC})$ and $E(\mathrm{SD})$ be the binary encodings of states SB, SC and SD respectively. Then $E(\mathrm{SD}) = \mathrm{E(SB)} + \mathrm{E(SC)}$ is the bit-by-bit logical OR of *E*(SB) and *E*(SC). This is true because the synthesis method uses the logical OR for the next values of each state variable and by extension the logical OR of the encoded states.

Step 1 of 1

7.067E

Assume $\boldsymbol{E}(\mathbf{SB})$, $\boldsymbol{E}(\mathbf{SC})$ and $\boldsymbol{E}(\mathbf{SD})$ be the binary encodings of states SB, SC and SD respectively. Then $\boldsymbol{E}(\mathbf{SD}) = \mathbf{E}(\mathbf{SB}) \cdot \mathbf{E}(\mathbf{SC})$ is the bit-by-bit logical AND of *E*(SB) and *E*(SC). This is true because the synthesis method uses the logical AND for the next values of each state variable and by extension the logical AND of the encoded states.

Step 1 of 1

Refer Table 7-2 in the textbook.

From the table using the method $V^{*}=\sum p-terms$ where $V^{*}=1$, the initial state is SA and its coded value is,

$$SA=SD\cdot EN$$

For each state the possible states are depends on the EN = 0 and EN = 1. Here the input combination is 1, So using the synthesised method the state SD will never appear in the state diagram when $V^{*}=1$ since it is not defined in the excitation table.

Step 1 of 1

Refer Table 7-2 in the textbook

From the table using the method $V^{*} = \sum p - terms$ where $V^{*} = 0$, the initial state is SA and its coded value is,

$$SA = SA \cdot EN'$$

For each state the possible states are depends on the EN = 0 and EN = 1. Here the input combination is 0, So using the synthesised method the state SD will never appear in the state diagram when $V^{*} = 0$ since it is not defined in the excitation table.

7.070E

Step 1 of 3

Master-slave flip-flops are also called as pulse triggered because they require both logic 0-to-1 and 1-to-0 transitions on the clock input for proper operation. If in a synchronous sequential circuit, a SR flip-flop is used, an unstable oscillation cannot occur because at all the times, either the master or the slave latch is in the mode hold. The master slave SR flip-flop as shown in figure 1.

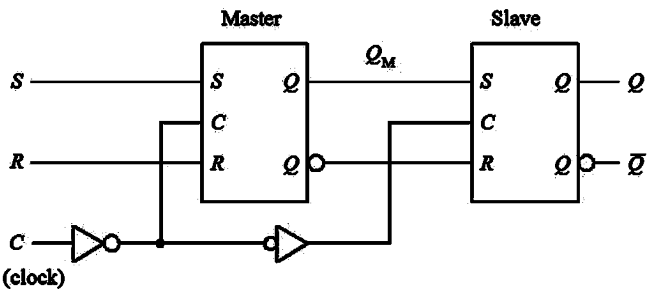
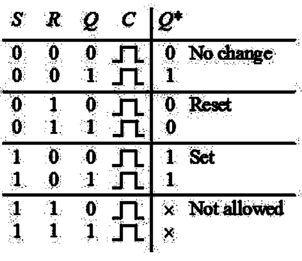


Figure 1

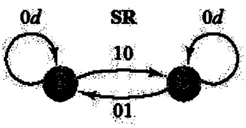
Step 2 of 3

The excitation table and state diagram for the SR master-slave flip-flop are presented in figure 2(a) and figure 2(b) respectively. The columns S, R, and Q denote the conditions on the flip-flop signals before the clock pulse is applied. The Q* column denotes the flip-flop output after the clock pulse has been applied. The state diagrams of the simple SR latch and the master-slave SR latch are identical. The difference between them is that the latch changes states immediately when S or R changes, whereas all flip-flop state changes are triggered by clock pulses.



S	R	Q	C	Q^*	
0	0	0	⎍	0	No change
0	0	1	⎍	1	
0	1	0	⎍	0	Reset
0	1	1	⎍	0	
1	0	0	⎍	1	Set
1	0	1	⎍	1	
1	1	0	⎍	×	Not allowed
1	1	1	⎍	×	

(a)



(b)

Figure 2

Step 3 of 3

Using the transition table the excitation equation is $Q^* = S + R'Q$.

The excitation equation is in the form $Qi^* = \text{expr}$, writing it in the form $Qi^* = \text{Qi}\cdot\text{expr}1 + \text{Qi}'\cdot\text{expr}2$

$$\begin{aligned} Qi^* &= \text{Q}\cdot(S + R'Q) + \text{Q}'\cdot(S + R'Q) \\ &= Q\cdot S + Q\cdot R' + Q'S \\ &= S\cdot(Q + Q') + R'\cdot Q \\ &= S + R'Q \end{aligned}$$

The derived excitation equaiton Q^* is same as Qi^* . Thus, any transititon equaton Q^* can be written as $Qi^* = \text{Qi}\cdot\text{expr}1 + \text{Qi}'\cdot\text{expr}2$.

7.071E

Step 1 of 2

7.072E

The combinational circuit with a feedback is also known as cyclic combinational circuit and a simple cyclic combinational circuit with the feedback is shown in Figure 1.

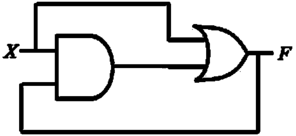


Figure 1

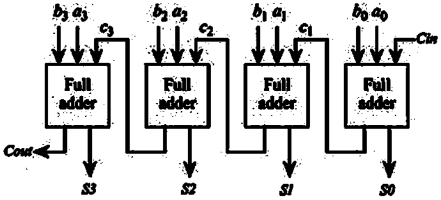
Step 2 of 2

If $x = 0$, then the output of the AND gate is fixed at 0 and it has no influence on OR gate. If $x = 1$, then the output of the OR gate is fixed at 1 and it has no influence on AND gate. Although the circuit is useless, it has a feedback loop and is combinational as the output of the circuit value depens only on the current value of *x*.

Step 1 of 4 **7.073E**

Consider the following logic diagram of a 4-bit one's-compliment adder:

Step 2 of 4



Step 3 of 4

Figure 1

Step 4 of 4

The follwing are the output equations of each and every adder:

$$S = A \oplus B \oplus Cin$$

$$Cout = Cin \cdot (\mathrm{A} \oplus \mathrm{B}) + \mathrm{A} \cdot \mathrm{B}$$

The following is the expression for the second adder *Cin*:

$$C1 = Cin \cdot (\mathrm{a}_0 \oplus \mathrm{b}_0) + \mathrm{a}_0 \cdot \mathrm{b}_0$$

Observe from the logic diagram in Figure 1 and the equations that the output of adder circuit does not only depend on the present values but also on the past values of the input signals.

Hence, it is evident that the circuit in Figure 1 is a sequential feedback circuit with carry out as its feedback signal.

Step 1 of 2

The 74x182 is a carry look-ahead circuit and the 74x381 is a 4-bit arithmetic logic unit.

The following is the purely combinational 16-bit one's-complement adder using 74x381s, a 74x182 and one 2-input gate:

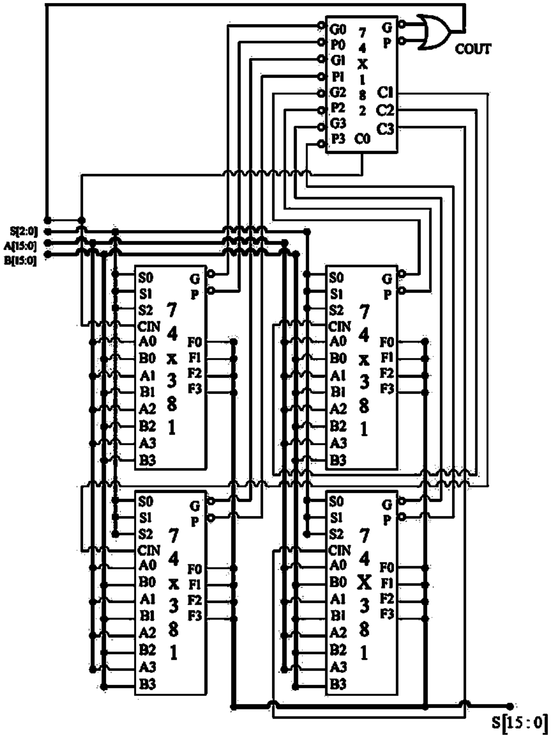


Figure 1

Step 2 of 2

The circuit in Figure 1 works as a 16 bit one's-complement adder if the selection inputs $S[2:0]$ are 011.

The output of the OR gate gives the carry out and is fed as a input to the Carry in. The selection input 011 makes the 74x381 to work as *A plus B plus CarryIn*.

Step 1 of 4

Refer to Figure 7-80 from the textbook for a positive-edge-triggered CMOS D flip-flop.

Refer to the transition/output, state/output, and flow/output tables from Figures 7-73 to 7-79 from the textbook.

Consider the following excitation equations of the circuit:

$$Y1* = CLK' \cdot D' + CLK \cdot Y1$$
$$Y2* = CLK \cdot Y1' + CLK' \cdot Y2$$
$$Q = Y2$$

Obtain the transition table using these excitation equations.

Table 1

Y1Y2	CLK D	Q			
	00	01	11	10	
00	10	00	01	01	0
01	11	01	01	01	1
10	10	00	10	10	0
11	11	01	10	10	1
$Y1*Y2*$					

Step 2 of 4

Determine the state table using the transition table.

Table 2

Y1Y2	CLK D	Q			
	00	01	11	10	
S0	S2	(S0)	S1	S1	0
S1	S3	(S1)	(S1)	(S1)	1
S2	(S2)	S0	(S2)	(S2)	0
S3	(S3)	S1	S2	S2	1
$Y1*Y2*$					

Step 3 of 4

Tabulate the flow table.

Table 3

Y1Y2	CLK D	Q			
	00	01	11	10	
S0	S2	(S0)	S1	-	0
S1	S3	(S1)	(S1)	(S1)	1
S2	(S2)	S0	(S2)	(S2)	0
S3	(S3)	S1	S2	-	1
$Y1*Y2*$					

Step 4 of 4

Compare the flow table in Table 3 with the reduced flow and output table for a positive-edge-triggered D flip-flop from Figure 7-79 in the textbook to observe that the behavior is equivalent to that of D flip-flop in Figure 7-72 from the textbook.

Hence, it is shown that the behavior of the positive-edge-triggered D flip-flop in Figure 7-80 is equivalent to that of the D flip-flop in Figure 7-72.

Step 1 of 1 **7.076E**

Consider the following form for the excitation equation of all single-loop feedback sequential circuits from Section 7.10.1 in the textbook:

$$Q* = (\text{forcing term}) + (\text{holding term}) \cdot Q$$

There are no practical circuits whose excitation equation substitutes Q' instead of Q. consider the following explanation for the same:

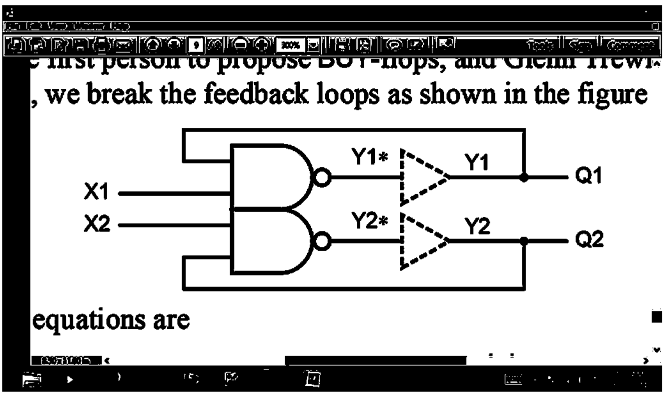
The practical circuit uses Q instead of Q' as the feedback because Q powers up first and it takes on whatever value it happens to be when the circuit is powered up. Assume that $Q = 1$ when the circuit is powered up. Since $Q = 1$ and it is also the input to the bottom invertor, it generates $Q' = 0$ which is fed to the top invertor and generates the value $Q = 1$ which is now a stable situation.

Hence Q is called the bi-stable element and is used as a feedback instead of Q' .

Step 1 of 5 **7.077E**

Refer to Figure X7.77 in the text book.

Analyze the circuit to break the feedback loops as shown in the Figure 1.


e first person to propose BUT-flops, and Glenn Trewi

, we break the feedback loops as shown in the figure

equations are

Figure 1

Step 2 of 5

A BUT' gate is defined as "Y1 is 1 if A1 and B1 are 1 but either A2 or B2 is zero; Y2 is 1 if A2 and B2 are 1 but either A1 or B1 is zero". NBUT is simply a BUT gate with inverted inputs.

Write the excitation and output equations as follows:

$$\begin{aligned} Y1 &= [(X1\cdot Y1)\cdot(X2\cdot Y2)']' \\ &= X1' + Y1' + X2\cdot Y2 \end{aligned}$$

$$\begin{aligned} Y2 &= \left[(X1\cdot Y1)'\cdot(X2\cdot Y2)\right]' \\ &= X1\cdot Y1 + X2' + Y2' \end{aligned}$$

$$Q1 = Y1$$

$$Q2 = Y2$$

Step 3 of 5

Determine the following transistion table using the excitation equations.

Y1Y2	X1X2			
	00	01	11	10
00 01 11 10	11 ○ ○ 11 11 11	11 10 10 11	11 ○ ○ 10 11 01	11 11 01 01
$Y1^{*}Y2^{*}$				

Table 1

Step 4 of 5

Using the transition table in Table 1 the corresponding flow table is as shown in Table 2.

Y1Y2	X1X2			
	00	01	11	10
11	○ ○ 11	10	○ ○ 11	01
$Y1^{*}Y2^{*}$				

Table 2

Step 5 of 5

Write the following conclusions using Table 2.

• When $X1X2 = 00(\text{or})11$, the circuit generally goes to stable state 11, with $Q1Q2 = 11$. The apparent oscillation between states 01 and 10.

• When $X1X2 = 11$ may not occur in practice, because it contains a critical race that tends to force the circuit into stable state 11.

• When $X1X2 = 01(\text{or})10$, the Q output corresponding to the HIGH input will oscillate, while the other output remains HIGH.

There are only one state which is stable is state 11 as shown in transition Table 1. But from the Table 2, the system oscillates in the states 11, 10 and 01 where 10 and 01 are unstable.

Hence, the circuit is **unstable**.

Step 1 of 6

Refer to Figure X7.78 in the text book.

Analyze the circuit to break the feedback loops as shown in the Figure 1.

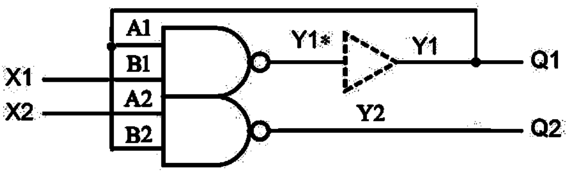


Figure 1

Step 2 of 6

A BUT' gate is defined as "Y1 is 1 if A1 and B1 are 1 but either A2 or B2 is zero; Y2 is 1 if A2 and B2 are 1 but either A1 or B1 is zero". NBUT is simply a BUT gate with inverted inputs.

Write the excitation and output equations as follows:

$$\begin{aligned} Y1 &= \left[(X1\cdot Y1)'\cdot(X2\cdot Y1)\right]' \\ &= X2'+Y1'+X1\cdot Y1 \end{aligned}$$

$$\begin{aligned} Y2 &= [(X1\cdot Y1)\cdot(X2\cdot Y1)']' \\ &= X2\cdot Y1+X1'+Y1' \end{aligned}$$

$$Q1 = Y1$$

$$Q2 = Y2$$

Step 3 of 6

Determine the following transistion table using the excitation equations.

Y1Y2	X1X2				
	00	**01**	**11**	**10**	
00 01 10 11	11 11 11 ○ ○ 11	11 11 01 01	11 11 11 ○ ○ 11	11 ○ ○ 11 10 10	
$Y1^{*}Y2^{*}$					

Table 1

Step 4 of 6

Using the transition table in Table 1 the corresponding flow table is as shown in Table 2.

Y1Y2	X1X2				
	00	**01**	**11**	**10**	
10 11	11 ○ ○ 11	01 01	11 ○ ○ 11	○ ○ 10 10	
$Y1^{*}Y2^{*}$					

Table 2

Step 5 of 6

The combine state and the output table for the flow table is asshown in Table 3.

Y1Y2	X1X2				
	00	**01**	**11**	**10**	
S2 S3	S3,10 ○ ○ S3,11	S1,10 S1,11	S3,10 ○ ○ S3,11	○ ○ S2,10 S2,11	
$Y1^{*}Y2^{*},Q1,Q2$					

Table 3

Step 6 of 6

In Table 1, the stable states are S3 and S2 but in Table 3 state S1 is unstable.

Hence, the circuit is **unstable with the input 01**.

7.079E

We don't have the solution to this problem yet.

Get help from a Chegg subject expert.

Ask an expert

Step 1 of 1

The output of a D latch is unpredictable when the input D changes at any time during the setup and hold time. Refer to Figure 7-14 from the textbook for the timing parameters of a D latch. This is called the metastability condition in which the output is either 0 or 1. Metastability includes the timing behavior such as setup time, propagation delay and hold times.

Other than metastability the outputs Q and $\mathbf{Q_N}$ of a D latch will be non-complimentary for an arbitrarily long time when preset and clear are asserted simultaneously. D latch goes to metastable state when preset and clear are asserted simultaneously.

We don't have the solution to this problem yet.

Get help from a Chegg subject expert.

Ask an expert

Step 1 of 4

The control inputs are $C1 \text{ and } C2$ and data inputs are $D1, D2 \text{ and } D3$.

Write the following function to realise the two level sum of products circuits for exciting functions.

$$z = C_1C_2 \cdot (D_1 + D_2 + D_3)$$

The circuit for the required functionality is shown in Figure 1.

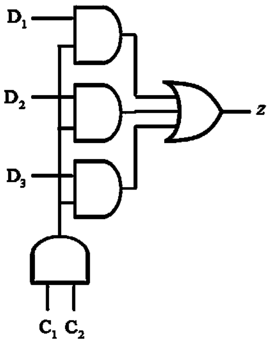


Figure 1

Step 2 of 4

The circuit in the Figure 1 is open only if both control inputs are 1, and it store 1 if any of the data inputs are 1.

The circuit is hazard free as it forms a complete sum of products.

Write the complete sum of products form.

$$z = C_1C_2 \cdot \left(\begin{array}{l} D_1D_2D_3 + \overline{D_1}\,\overline{D_2}D_3 + D_1\overline{D_2}\,\overline{D_3} + \overline{D_1}D_2\overline{D_3} + \overline{D_1}D_2D_3 \\ + D_1\overline{D_2}D_3 + D_1D_2\overline{D_3} \end{array} \right)$$

Step 3 of 4

The hazard free latch circuit is as shown in Figure 2.

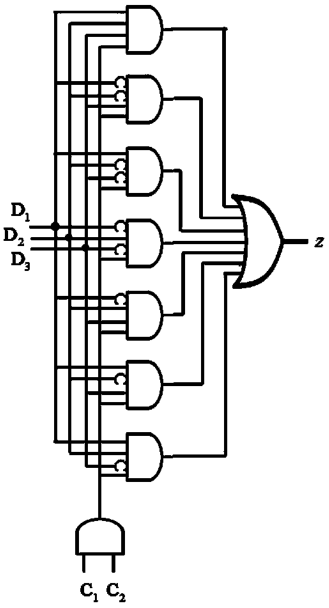


Figure 2

Step 4 of 4

Hence, the hazard free latch circuit is designed.

Step 1 of 4

Recall the output expression from the Exercise 7.82 in the text book.

$$z = C_1C_2 \cdot \begin{pmatrix} D_1D_2D_3 + \overline{D_1}\,\overline{D_2}D_3 + D_1\overline{D_2}\,\overline{D_3} + \overline{D_1}D_2\overline{D_3} + \overline{D_1}D_2D_3 + \\ +D_1\overline{D_2}D_3 + D_1D_2\overline{D_3} \end{pmatrix}$$

The multi-level circuit is developed as follows:

$$\begin{aligned} z &= C_1C_2 \cdot \left(D_1\left(D_2D_3 + \overline{D_2}\,\overline{D_3} + \overline{D_2}D_3 + D_2\overline{D_3}\right) + \overline{D_1}\left(D_2\overline{D_3} + D_2D_3 + \overline{D_2}D_3\right)\right) \\ &= C_1C_2 \cdot \left(D_1\left(D_2\left(D_3 + \overline{D_3}\right) + \overline{D_2}\left(\overline{D_3} + D_3\right)\right) + \overline{D_1}\left(D_2\overline{D_3} + D_2D_3 + \overline{D_2}D_3\right)\right) \\ &= C_1C_2 \cdot \left(D_1\left(D_2(1) + \overline{D_2}(1)\right) + \overline{D_1}\left(D_2\overline{D_3} + D_2D_3 + \overline{D_2}D_3\right)\right) \\ &= C_1C_2 \cdot \left(D_1\left(D_2 + \overline{D_2}\right) + \overline{D_1}\left(D_2\overline{D_3} + D_2D_3 + \overline{D_2}D_3\right)\right) \\ &= C_1C_2 \cdot \left(D_1(1) + \overline{D_1}\left(D_2\overline{D_3} + D_2D_3 + \overline{D_2}D_3\right)\right) \\ &= C_1C_2 \cdot \left(D_1 + \overline{D_1}\left(D_2\overline{D_3} + D_2D_3 + \overline{D_2}D_3\right)\right) \end{aligned}$$

Step 2 of 4

Recall the hazard free latch circuit from the Exercise 7.82 in the text book.

The hazard free latch circuit is as shown in Figure 1.

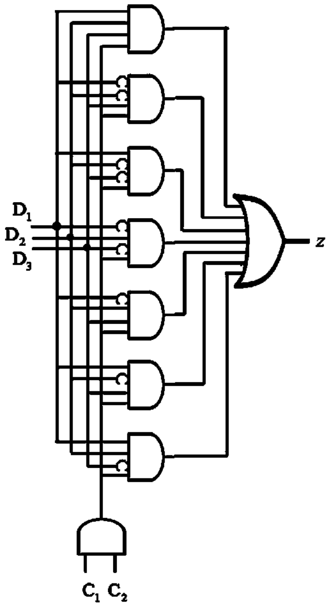


Figure 1

Step 3 of 4

The circuit with minimum number of gates is shown in Figure 2:

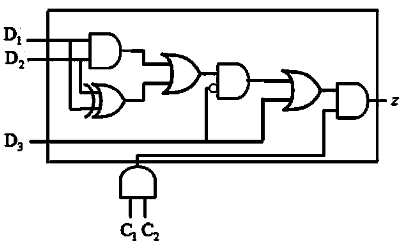


Figure 2

Step 4 of 4

The gates required to build the circuit shown in Figure 2 are less when compared to the circuit in the Figure 1. The circuit in Figure 1 has seven 4-input gates, one 2-input gate and one 7-input gate, where as the circuit in the Figure 2 requires seven 2-input gates.

We don't have the solution to this problem yet.

Get help from a Chegg subject expert.

Ask an expert

7.085E

Step 1 of 3

Refer to the Figure 7-84 from the text book.

Use the internal state variables of the pulse-catching circuit of the Figure 7-94, and draw the timing diagram as shown in the Figure 1.

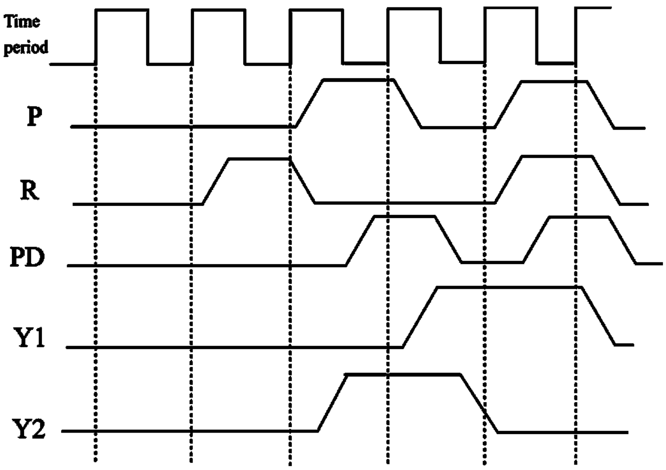


Figure 1

Step 2 of 3

The waveforms starts with the state 00 and the behaviour of internal variables are as shown in the Figure 1. The behavior of the Figure 1 is as shown in Table 1.

Table 1

P	R	Y1	Y2
0	0	0	0
0	1	0	0
1	0	0	0,1
0	0	1	1,1
1	1	1	0

Step 3 of 3

Assuming the time period to be 10 nanoseconds and the propagation delay to be 3 nanoseconds, then the transitions are as shown in the Table 1.

When PR=10 and 00 consecutively then Y2 changes its state from 0-to-1 after the propogation delay of 3 nanoseconds and Y2 remains in the same state when PR=00.

Thus, the timing diagram is drawn showing the internal state variables of the pulse-catching circuit.

Step 1 of 1

Refer to the section 7.10.2 from the text book.

Refer to the Table 7-85 from the text book.

The Table 7-85 in the text book is drawn only for the transition 0 to 1 and the transistion from 1 to 0 asserts the output Z in the state RES2.

In the concept it is given that in PLS1, "got a pulse and haven't seen a reset", this indicates a transition from 1-to-0,that is, from PLS1 to RES2, the transition changes from 1-to-0. Hence the output is asserted which makes the state to settle in RES2 after PLS1 according to the Table 1.

The fundamental flow table is obtained with both the transitions 0-to-1 and 1-to-0 is as shown in Table 1

S	**PR**	**Z**			
	00	**01**	**11**	**10**	
IDLE	IDLE	RES1	-	PLS1	0
RES1	IDLE	RES1	RES2	-	0
PLS1	PLS2	-	RES2	PLS1	1
RES2	-	RES1	RES2	PLSN	1
PLS2	PLS2	RES1	-	PLS1	1
PLSN	IDLE	-	RES2	PLSN	0
S^*					

Table 1

Thus, the fundamental flow table is obtained.

Step 1 of 3

Refer Figure X7.87 in the textbook.

Write the excitation equations for next states ($Y1*$, $Y2*$, $Y3*$) in terms of present(current) states ($Y1$, $Y2$, $Y3$) and input X since logic equations that express the excitation signals as a function of current state and input are called as excitation equations.

$$Y1* = X'Y1 + Y1Y2' + Y1Y3' + XY1'Y2Y3$$
$$Y2* = Y1'Y2 + X'Y1Y2 + XY1'Y3 + Y1Y2'Y3'$$
$$Y3* = XY2' + Y2'Y3 + X'Y2Y3'$$

Step 2 of 3

Construct the transition table from the next state equations.

Table 1

Y1Y2Y3	X	
	0	1
000	000	001
001	001	011
010	011	010
011	010	110
100	110	111
101	111	101
110	101	100
111	100	000
Y1*Y2*Y3*		

Step 3 of 3

When X = 1, the circuit was supposed to count(increment) through its eight states in Gray-code order. When X = 0, the circuit was supposed to remain in same state in the gray-code form. If this were the case, I suppose it could be used as a 3-bit random number generator or gray code generator or mod 8 counter in the gray code form.

Step 1 of 1

Refer to Figure X7.88 from the textbook for the adjacency diagram resulted from the general solution for obtaining a race-free state assignment of 2^n states using 2^{n-1} state variables for the $n=2$ case. This requires two hops for each input change.

Refer to Figure 7-91 from the textbook for the worst-case scenario of four-state adjacency diagram and the assignment using pairs of equivalent states. It is faster and requires one hop for each input change. On the other hand, Figure 7-91 cannot be generalized for $n \geq 2$.

7.089E

We don't have the solution to this problem yet.

Get help from a Chegg subject expert.

Ask an expert

7.090E

We don't have the solution to this problem yet.

Get help from a Chegg subject expert.

Ask an expert

7.091E

We don't have the solution to this problem yet.

Get help from a Chegg subject expert.

Ask an expert

7.092E

Step 1 of 2

Conventional positive edge trigger D flip-flop responds to the input at the time of clock signal is changing from 0 to 1. Similarly, negative edge trigger D flip-flop responds to the input at the clock signals is changing from 1 to 0.

Whereas, dual edge triggered D flip-flop responds to both edges implies for 0 to 1 as well as 1 to 0.

Draw the general implementation of Dual edge trigger D flip-flop.

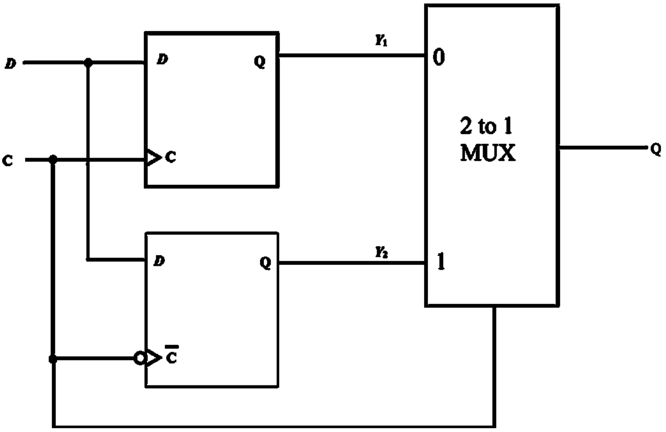


Figure 1

From the Figure 1, write the excitation equations for the double edge triggered D-flip flop.

$$Y1^{*} = C$$

$$Y2 = Dy_2 + \overline{Cy_1}y_2 + Cy_1y_2 + \overline{C}Dy_1 + CD\overline{y_1}$$

Construct a flow table for dual edge-triggered D-flip-flop using excitation equations.

State	CD			
	00	01	11	10
S1	◯	S2,0	S4,0/S5,-	S3,0
S2	◯	◯	S5,-	S3,0S6,-
S3	S1,0	◯	S4,0	◯
S4	S1,0	S2,0	◯	◯
S5	S1,0	S2,0/S7,-	◯	S3,0
S6	S1,0/S8,-	S7,1	S4,0	S3,0
S7	S1,-/S8,1	S7,1	◯	S6,1
S8	S1,0	S2,-/S7,1	◯	◯
	S8,1	◯	S5,1	◯
	◯	◯	S5,1	S6,1
	◯	S7,1	S5,1	S3,-/S6,1
	S8,1	S7,1	S4,-/S5,1	S3,-

Table 1

Here, C represents clock, D represents input.

Step 2 of 2

Table 1 is a fundamental flow table for a DET-D-FF. Note that simultaneous changes of *D* and C are treated as though one of these variables changed first, but that it does not matter *which* changed first. This is a realistic assumption as we can never rely on *exact* simultaneity for any pair of events. The option of treating a simultaneous change in either of two ways is left open for exploitation later in the design process.

Step 1 of 1

The theorem of essential hazards states that, “If a function contains an essential hazard, then any proper circuit of it must contain atleast one delay element”.

A fundamental-mode circuit must have at least three states to have an essential hazard so, latches dont have any essential hazards. Flip-flops contain more than three states and as discussed in section 7.10.6, the flow tables of all edge-triggered flip-flops contain essential hazards.

All the essential hazards can also be masked by controlling the path delays.

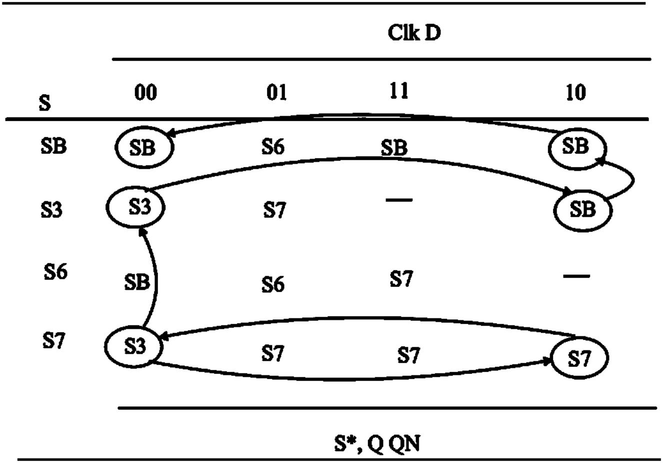
Thus, proved that the fundamental-mode flow table of any flip-flop that samples inputs and changes outputs on the rising edge only of a clock signal CLK contains an essential hazard.

Step 1 of 1

7.095E

Refer Figure 7-79 in the textbook

Repeat the figure to detect the essential hazard.



	Clk D			
S	00	01	11	10
SB	SB	S6	SB	SB
S3	S3	S7	—	SB
S6	SB	S6	S7	—
S7	S3	S7	S7	S7
	S*, Q QN			

Figure 1

The flow table for the positive-edge-triggered D flip-flop as shown in Figure 1, in stable state *S7* for $x_1x_2 = 00$ is at state $S3$; if x_1 changes to 1, the stable state reached is *S7*; but if x_1 changes from 0 to 1 then back to 0 and back to 1 again then the stable state reahed is *SB* – a different state. Hence, there is an essential hazard at S7 with 10.

We don't have the solution to this problem yet.

Get help from a Chegg subject expert.

Ask an expert

Step 1 of 3 7.097E

Conventional positive edge trigger D flip-flop responds to the input at the time of clock signal is changing from 0 to 1. Similarly, negative edge trigger D flip-flop responds to the input at the clock signals is changing from 1 to 0.

Whereas, dual edge triggered D flip-flop responds to both edges implies for 0 to 1 as well as 1 to 0.

Draw the general implementation of Dual edge trigger D flip-flop.

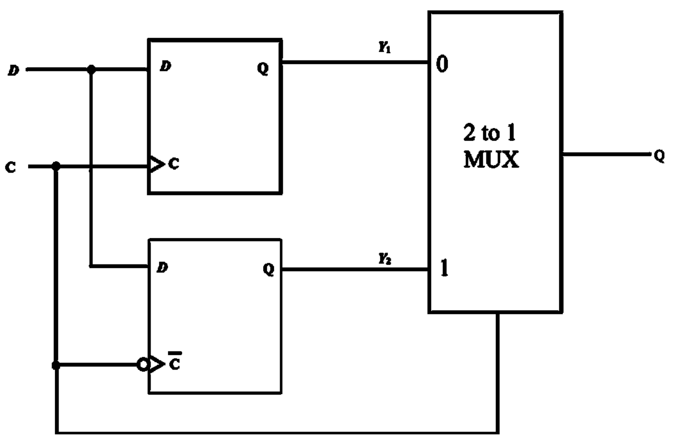


Figure 1

From the Figure 1, write the excitation equations for the double edge triggered D-flip flop.

$$Y1^* = C$$

$$Y2 = Dy_2 + \overline{Cy_1}y_2 + Cy_1y_2 + \overline{C}Dy_1 + CD\overline{y_1}$$

Step 2 of 3

Construct a flow table for dual edge-triggered D-flip-flop using excitation equations.

State	CD			
	00	01	11	10
S1	◯	S2,0	S4,0/S5,-	S3,0
S2	◯	◯	S5,-	S3,0S6,-
S3	S1,0	◯	S4,0	◯
S4	S1,0	S2,0	◯	◯
S5	S1,0	S2,0/S7,-	◯	S3,0
S6	S1,0/S8,-	S7,1	S4,0	S3,0
S7	S1,-/S8,1	S7,1	◯	S6,1
S8	S1,0	S2,-/S7,1	◯	◯
	S8,1	◯	S5,1	◯
	◯	◯	S5,1	S6,1
	◯	S7,1	S5,1	S3,-/S6,1
	S8,1	S7,1	S4,-/S5,1	S3,-

Table 1

Here, C represents clock, D represents input.

Step 3 of 3

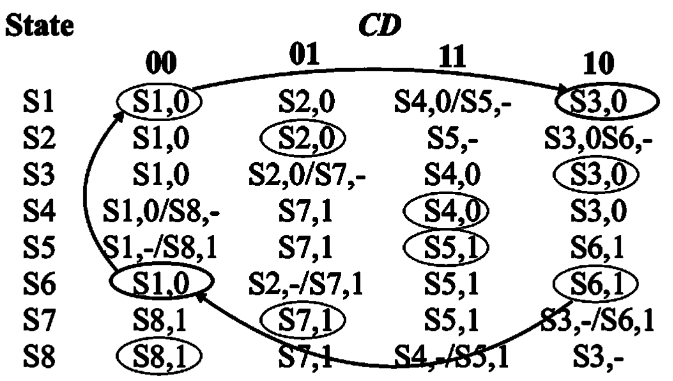
The Table 1 is a fundamental flow table for a DET-D-FF.

Note that simultaneous changes of *D* and C are treated as though one of these variables changed first, but that it does not matter *which* changed first. This is a realistic assumption as we can never rely on *exact* simultaneity for any pair of events. The option of treating a simultaneous change in either of two ways is left open for exploitation later in the design process.

Identify the essential hazard in the Table 1.

In the stable state *S6*, for $CD = 00$ if C changes 1, the stable state reached is *S6*; but if C changes from 0 to 1 then back to 0 and back to 1 again then the stable state reached is

S3 – a different state as shown in the Figure 2



State	CD			
	00	01	11	10
S1	S1,0	S2,0	S4,0/S5,-	S3,0
S2	S1,0	S2,0	S5,-	S3,0S6,-
S3	S1,0	S2,0/S7,-	S4,0	S3,0
S4	S1,0/S8,-	S7,1	S4,0	S3,0
S5	S1,-/S8,1	S7,1	S5,1	S6,1
S6	S1,0	S2,-/S7,1	S5,1	S6,1
S7	S8,1	S7,1	S5,1	S3,-/S6,1
S8	S8,1	S7,1	S4,-/S5,1	S3,-

Figure 2

Hence there is an essential hazard from S6 at clock change.

Step 1 of 3

A verbal flip flop is used produce any two complementary words such "true or false" or "accept or reject" etc. Here, the verbal flip-flop has three data inputs $(D_1, D_2 \text{ and } D_3)$ and two control inputs (C_1C_2) and it generates the word "Accepted" if *Z*=1 and generates the word "Rejected" if *Z*=0. If C_1C_2 are 11 then the verbal flip flop is in ready state and for all the remainging cases it is in rest state.

Consider a truth table for the verbal flip flop.

Table 1

C_1C_2	$D_1D_2D_3$	Output
0X	XXX	X
X0	XXX	X
11	000	Rejected
11	001	Rejected
11	010	Rejected
11	011	Accepted
11	100	Rejected
11	101	Accepted
11	110	Accepted
11	111	Accepted

From the Table 1,Write the boolean expression for the logic.

$$\textbf{Output} = D_1D_2C_1C_2 + D_2D_3C_1C_2 + D_3D_1C_1C_2$$

Step 2 of 3

Draw the verbal flip-flop with three data inputs and two control inputs is as shown in the Figure 1.

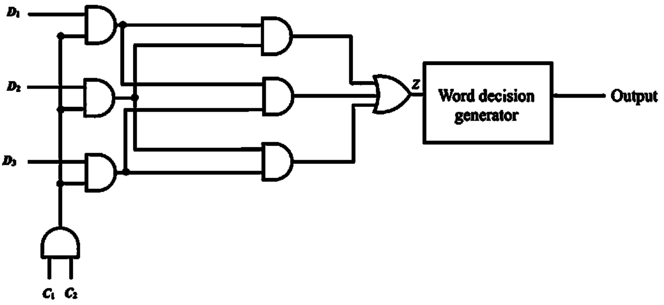


Figure 1

Step 3 of 3

If both the outputs are 1 and if the majority of outputs are 1 then the *Output* is "Accpeted" else "Rejected".

It can be adapted in the political arena for voting the and decsion making in the assembly for passing a bill or request. For example there are three members X, Y and Zin a house having buttons $(D_1, D_2 \text{ and } D_3)$ respectively, if at least two button are pressed the word generator produce the output "Accpted".

Step 1 of 1

Refer Table 7-24 from the textbook for a more natural ABEL program for a state machine. Modify the program using an output-coded state assignment in order to reduce the total number of programmable logic device (PLD) outputs required to one.

```
module SMEX2

title 'Modified Version of State Machine'

" Input and output pins

CLOCK, RESET_L, A, B pin;

LASTA, Q1, Q2 pin istype 'reg';

Z pin istype 'com';

" Definitions

QSTATE = [Q1,Q2]; " State variables

INIT = [ 0, 0]; " State encodings

LOOKING = [ 0, 1];

OK = [ 1, 0];

XTRA = [ 1, 1];

RESET = !RESET_L;

state_diagram QSTATE

state INIT: if RESET then INIT

else LOOKING;

state LOOKING: if RESET then INIT

else if (A == LASTA) then OK

else LOOKING;

state OK: if RESET then INIT

else if B then OK

else if (A == LASTA) then OK

else LOOKING;

state XTRA: goto INIT;

equations

LASTA.CLK = CLOCK;

QSTATE.CLK = CLOCK;

QSTATE.OE = 1;

LASTA := A;

Z = B # (A !$ LASTA); " Output coded state assignment

end SMEX2
```

Hence, the modified ABEL code using output-coded state assignment is provided.

7.100E

We don't have the solution to this problem yet.

Get help from a Chegg subject expert.

Ask an expert

7.101E

7.102E

Step 1 of 2

8.01DP

Consider the following circuit for switch input using debouncing.

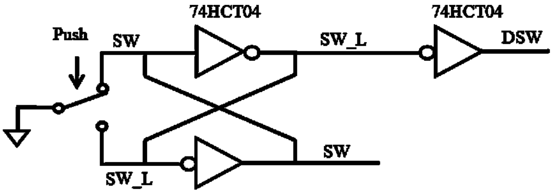


Figure 1

Step 2 of 2

When the button is at ground, the top contact holds SW at 0 V, a valid logic 0, and the top inverter produces logic 1 on SW_L and on the bottom contact. When the button is pushed and the contact is broken, feed-back in the bistable holds SW at V_{OL} (≤ 0.33 V for HCTMOS driving TTL loads), still a valid logic 0.

When wiper hits the bottom contact, the circuit operates quite unconventionally for a moment. The top inverter in the bistable is trying to maintain a logic 1 on the SW_L signal; the top transistor in its totem-pole output is "on" and connecting SW_L through a small resistance to +5V. Suddenly, the switch contact makes a metallic connection of SW_L to ground, 0.0 V.

A short time later (70 ns for the 74HCT04), the forced logic 0 on SW_L propagates through the two inverters of the bistable, so that the top inverter gives up its vain attempt to drive a 1, and instead drives a logic 0 on to SW_L.

At this point, the top inverter output is no longer shorted to ground, and feedback in the bistable maintains the logic 0 on SW_L even if the wiper bounces off the bottom contact, as it does. It does not bounce for enough to touch the top contact again.

Therefore, the inverter with output DSW is really required and the SW signal cannot be used directly as the debounced output.

Step 1 of 1

Refer to Table 8-2 in the text book for ABEL program for a latching address decoder.

The latching decoder circuit is given in Figure 8-16 in the text book.

Latching decoder operates only on the high-order bits ABUS[31:20]

The starting address of the second RAM bank is 0x01000000 and the starting address of the third RAM bank is 0x02000000.

$$\mathbf{RAMBANK1} = [0,0,0,0,0,0,0,1,X.\ ,X.\ ,X.\ ,X.]$$

ABUS is assigned with RAMBANK1.

Thus, the expression, $\left((\mathbf{ABUS} \geq \mathbf{0x010})\ \&\ (\mathbf{ABUS} < \mathbf{0x020})\right)$ gives the same result.

Hence, the second RAM bank can be decoded using the expression.

Step 1 of 1

A 16V8 has up-to 16 inputs and 8 output cells.

Each output cell is connected to AND-OR matric of size 32 columns by 8 rows. Seven of the rows are connected to the OR gate inside the output block, while the remaining row is routed separately. The number of 32 columns results from the true and complemented values of 16 inputs.

As the number of output sells is 8 and each output cell has 8 rows, the total number of rows for 16V8 device is 64.

Calculate the fuses of the whole 16V8 device.

$$\begin{aligned}\textbf{Total number of fuses} &= 32*64\\ &= 2048 \text{ fuses}\end{aligned}$$

Therefore, the fuses that are contained in 16V8 device is $\boxed{\textbf{2048 fuses}}$.

Step 1 of 1

The programmable AND array in the 22V10 device has a size of 132×44 and the number of fuses from the output pins as true and its complement is 20.

Calculate the fuses require for 22V10 device.

$$\begin{aligned}\textbf{Total number of fuses} &= (132*64)+20\\ &= 5808+20\\ &= 5828\text{ fuses}\end{aligned}$$

Therefore, the total number of fuses contained in the 22V10 are $\boxed{5828}$.

Step 1 of 9

The maximum frequency of the external feedback is the reciprocal of the minimum clock period.

$$f_{\max} = \frac{1}{t_{p\min}}$$

Write the formula for minimum clock period when the device operating with external feedback.

$$t_{p\min} = t_{CO} + t_{SU}$$

Here,

t_{CO} is the propagation delay of the first PLD.

t_{SU} is the setup time of the second PLD.

Step 2 of 9

Therefore, $\textbf{clock period } t_{p\min} \geq \left(t_{CO} \textbf{ of first PLD} + t_{SU} \textbf{ of second PLD}\right)$

Step 3 of 9

For GAL16V8D, GAL 20V8B with suffix -7

Calculate the minimum clock period of the circuit, when the devices GAL16V8D and GAL20V8B are connected in external feedback manner.

$$\begin{aligned} t_{p\min} &= t_{CO} + t_{SU} \\ &= 5\text{ ns} + 7\text{ ns} \\ &= 12\text{ ns} \end{aligned}$$

Calculate the maximum frequency with the external feedback.

$$\begin{aligned} f_{\max} &= \frac{1}{t_{p\min}} \\ &= \frac{1}{12 \times 10^{-9}} \\ &= 83.33\text{ MHz} \end{aligned}$$

Thus, the maximum frequency with the external feedback is $\boxed{83.33\text{ MHz}}$.

Step 4 of 9

For GAL16V8D, GAL 20V8B with suffix -10

Calculate the minimum clock period of the circuit, when the devices GAL16V8D and GAL20V8B are connected in external feedback manner.

$$\begin{aligned} t_{p\min} &= t_{CO} + t_{SU} \\ &= 7\text{ ns} + 10\text{ ns} \\ &= 17\text{ ns} \end{aligned}$$

Calculate the maximum frequency with the external feedback.

$$\begin{aligned} f_{\max} &= \frac{1}{t_{p\min}} \\ &= \frac{1}{17 \times 10^{-9}} \\ &= 58.82\text{ MHz} \end{aligned}$$

Thus, the maximum frequency with the external feedback is $\boxed{58.82\text{ MHz}}$.

Step 5 of 9

For GAL16V8D, GAL 20V8B with suffix -15

Calculate the minimum clock period of the circuit, when the devices GAL16V8D and GAL20V8B are connected in external feedback manner.

$$\begin{aligned} t_{p\min} &= t_{CO} + t_{SU} \\ &= 10\text{ ns} + 12\text{ ns} \\ &= 22\text{ ns} \end{aligned}$$

Calculate the maximum frequency with the external feedback.

$$\begin{aligned} f_{\max} &= \frac{1}{t_{p\min}} \\ &= \frac{1}{22 \times 10^{-9}} \\ &= 45.45\text{ MHz} \end{aligned}$$

Thus, the maximum frequency with the external feedback is $\boxed{45.45\text{ MHz}}$.

Step 6 of 9

For GAL16V8D, GAL 20V8B with suffix -25

Calculate the minimum clock period of the circuit, when the devices GAL16V8D and GAL20V8B are connected in external feedback manner.

$$\begin{aligned} t_{p\min} &= t_{CO} + t_{SU} \\ &= 12\text{ ns} + 15\text{ ns} \\ &= 27\text{ ns} \end{aligned}$$

Calculate the maximum frequency with the external feedback.

$$\begin{aligned} f_{\max} &= \frac{1}{t_{p\min}} \\ &= \frac{1}{27 \times 10^{-9}} \\ &= 58.82\text{ MHz} \end{aligned}$$

Thus, the maximum frequency with the external feedback is $\boxed{37.04\text{ MHz}}$.

For GAL22V10D with suffix -7

Calculate the minimum clock period of the circuit.

$$\begin{aligned} t_{p\min} &= t_{CO} + t_{SU} \\ &= 4.5\text{ ns} + 4.5\text{ ns} \\ &= 9\text{ ns} \end{aligned}$$

Calculate the maximum frequency with the external feedback.

$$\begin{aligned} f_{\max} &= \frac{1}{t_{p\min}} \\ &= \frac{1}{9 \times 10^{-9}} \\ &= 111.11\text{ MHz} \end{aligned}$$

Thus, the maximum frequency with the external feedback is $\boxed{111.11\text{ MHz}}$.

Step 7 of 9

For GAL22V10D with suffix -10

Calculate the minimum clock period of the circuit.

$$\begin{aligned} t_{p\min} &= t_{CO} + t_{SU} \\ &= 7\text{ ns} + 7\text{ ns} \\ &= 14\text{ ns} \end{aligned}$$

Calculate the maximum frequency with the external feedback.

$$\begin{aligned} f_{\max} &= \frac{1}{t_{p\min}} \\ &= \frac{1}{14 \times 10^{-9}} \\ &= 71.43\text{ MHz} \end{aligned}$$

Thus, the maximum frequency with the external feedback is $\boxed{71.43\text{ MHz}}$.

Step 8 of 9

For GAL22V10D with suffix -15

Calculate the minimum clock period of the circuit.

$$\begin{aligned} t_{p\min} &= t_{CO} + t_{SU} \\ &= 8\text{ ns} + 10\text{ ns} \\ &= 18\text{ ns} \end{aligned}$$

Calculate the maximum frequency with the external feedback.

$$\begin{aligned} f_{\max} &= \frac{1}{t_{p\min}} \\ &= \frac{1}{18 \times 10^{-9}} \\ &= 55.55\text{ MHz} \end{aligned}$$

Thus, the maximum frequency with the external feedback is $\boxed{55.55\text{ MHz}}$.

Step 9 of 9

For GAL22V10D with suffix -25

Calculate the minimum clock period of the circuit.

$$\begin{aligned} t_{p\min} &= t_{CO} + t_{SU} \\ &= 15\text{ ns} + 15\text{ ns} \\ &= 30\text{ ns} \end{aligned}$$

Calculate the maximum frequency with the external feedback.

$$\begin{aligned} f_{\max} &= \frac{1}{t_{p\min}} \\ &= \frac{1}{30 \times 10^{-9}} \\ &= 33.33\text{ MHz} \end{aligned}$$

Thus, the maximum frequency with the external feedback is $\boxed{33.33\text{ MHz}}$.

Step 1 of 1

The GAL 16V8 is having similar functionality as a **74×374** . It contains 8 edge-triggered D flip flop that all the inputs drive the output on the rising edge of common CLK input. The flip-flops drive their input at CLK & OE_L (output enable).

Write the ABEL program for 16V8 for a 8-bit register:

```
module eight_bit_reg

title 'eight_bit_edge-triggered Register'

" Input pins

CLK, OE pin;

I1, I2, I3, I4, I5, I6, I7, I8 pin;

" Output pins

O1, O2, O3, O4, O5, O6, O7, O8 pin istype 'reg';

" Set definitions

D = [D1, D2, D3, D4, D5, D6, D7, D8];

Q = [Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8];

Equations

Count.CLK = CLK;

When count =! OE then Q: = D;

Else Q: = Q;

end eight_bit_reg
```

Step 1 of 1

The GAL16V8, GAL20V8 and GAL22V10 has programmable output polarity nothing but polarity change at the output side instead of input and let us assume that the PLDs GAL16V8, GAL20V8 and GAL22V10 are programming for the 4-bit register as like as **74×175** .Now write the ABEL program for **74×175** a 4-bit register:

```
module Z74X175

title 'four_bit_register'

" Input pins

CLR_L, CLK pin 1, 9;

D0, D1, D2, D3 pin;

" Output pins

Q0, Q1, Q2, Q3 pin 2, 7, 10, 15 istype 'reg.buffer';

/Q0, /Q1, /Q2, /Q3 pin 3, 6, 11, 14 istype 'reg.buffer';

" Set definitions

D = [D0, D1, D2, D3];

Q = [Q0, Q1, Q2, Q3];

/Q = [/Q0, /Q1, /Q2, /Q3];

CLR =!CLR_L; "Active-level conversions

equations

if CLK then Q=D; /Q=!D;

end Z74X175
```

Step 1 of 4

8.09DP

Refer to the Figure 8.15 in the textbook.

Clearly from the referred Figure, a 32-bit latch followed by a decoder is used for memory selection and I/O devices in microprocessor systems.

Compute the propagation delay of latching decoder of referred Figure from AVALID to chip select of **latching decoder** method.

The location within the device and device selection is done based on asserting the status to “Address Valid” signal.

Refer to Table 8.1 in the textbook, the propagation delay for the input clock to the output is 8.5 ns.

Therefore, the propagation delay from AVALID to chip select is 8.5 ns from 32-bit latch.

The decoder is 16V8C of 10 ns propagation delay from input to the selected device output.

Therefore, the propagation delay from chip select input to the chip select output is 10 ns.

Hence, overall propagation delay AVALID to chip select output is the sum of 8.5 ns and 10 ns, which results the total propagation delay of $\boxed{18.5 \text{ ns}}$.

Step 2 of 4

Refer to the Figure 8.16 in the textbook.

Compute the propagation delay of latching decoder of referred Figure from AVALID to chip select of **decoder latching** method.

The location within the device and device selection is done based on asserting the status to “Address Valid” signal.

Refer to Table 8.1 in the textbook, the propagation delay for the input clock to the output is 8.5 ns.

Therefore, the propagation delay from AVALID to chip select is 8.5 ns from 20-bit latch.

The decoder is 16V8C of 10 ns propagation delay from input to the selected device output.

Hence, overall propagation delay AVALID to chip select output is $\boxed{8.5 \text{ ns}}$.

Step 3 of 4

Refer to the Figure 8.15 in the textbook.

Clearly from the referred Figure, a 32-bit latch followed by a decoder is used for memory selection and I/O devices in microprocessor systems.

Compute the propagation delay of latching decoder of referred Figure from ABUS to chip select of **latching decoder** method.

The location within the device and device selection is done based on asserting the status to “Address Valid” signal along with placing the address in the ABUS signal.

Refer to Table 8.1 in the textbook, the propagation delay for the input to the output is 5.2 ns.

Therefore, the propagation delay from ABUS to chip select is 5.2 ns from 32-bit latch.

The decoder is 16V8C of 10 ns propagation delay from input to the selected device output.

Therefore, the propagation delay from chip select input to the chip select output is 10 ns.

Hence, overall propagation delay ABUS to chip select output is the sum of 5.2 ns and 10 ns, which results the total propagation delay of $\boxed{15.2 \text{ ns}}$.

Step 4 of 4

Refer to the Figure 8.16 in the textbook.

Compute the propagation delay of latching decoder of referred Figure from ABUS to chip select of **decoder latching** method.

The location within the device and device selection is done based on asserting the status to “Address Valid” signal along with placing the address in the ABUS signal.

Therefore, the propagation delay from ABUS to chip select is 5.2 ns from 20-bit latch.

The decoder is 16V8C of 10 ns propagation delay from input to the selected device output.

Hence, overall propagation delay ABUS to chip select output is $\boxed{5.2 \text{ ns}}$.

Step 1 of 1

8.10DP

Refer Figure 7-38 in text book for clocked synchronous state machine using positive-edge-triggered D flip-flops.

The edge triggered *D* flip flop accepts data on the *D* input that are present at the active clock edge (either HIGH-to-LOW edge of C_p or the LOW-to-HIGH edge of C_p). The transitions of the level *D* before or after the active clock trigger edge are ignored.

The *D* latch accepts the *D* input data when the clock pulse has high value.

If the edge-triggered *D* flip-flops are replaced with *D* latches then the output of the circuit changes.

Step 1 of 1 **8.11DP**

The **74×162** is a decade counter. It counts from 0 to 9 where RCO signal indicates a carry when all the bit positions become 1 and ENT_LOW is asserted.

Write the ABEL program for a **74×162** decade counter.

```
module Z74X162

title 'DecadeCounter'

" Input pins

CLR_L, CLK, LD_L, ENP, ENT pin;

A, B, C, D pin;

" Output pins

QA, QB, QC, QD pin istype 'reg';

RCO pin istype 'com';

" Set definitions

INPUT = [ D, C, B, A ];

COUNT = [ QD, QC, QB, QA ];

CLR =!CLR_L; LD=!LD_L; "Active-level conversions

equations

COUNT.CLK = CLK;

COUNT:=!CLR & ( LD & INPUT # !LD & (ENT & ENP) & ( COUNT+1 ) # !LD & !(ENT & ENP) & ( COUNT );

RCO=(COUNT=[1, 0, 0, 1]) & ENT

end Z74X162
```

Step 1 of 1 **8.12DP**

The **74×161** is a modulo-16 counter. It counts from 0 to 15 where RCO signal indicates a carry when all the bit positions become 1 and ENT_LOW is asserted. The **74×161** have asynchronous clear function, so CLR_L input is connected to the asynchronous clear inputs of its flip-flops.

Write the ABEL program for a **74×162** decade counter.

```
module Z74X161

title ‘4_bit_binaryCounter’

“Input pins

RESET_L, CLK, LD_L, ENP, ENT pin1, 2, 9, 7, 10;

P0, P1, P2, P3 pin 3, 4, 5, 6;

“Output pins

Q0, Q1, Q2, Q3 pin istype ‘reg’;

RCO pin istype ‘reg’;

“ Set definitions

INPUT = [ P3, P2, P1, P0 ];

COUNT = [ Q3, Q2, Q1, Q0 ];

RESET =!RESET_L; LD=!LD_L; “Active-level conversions

equations

COUNT.CLK = CLK;

COUNT:=!CLR & ( LD & INPUT # !LD & (ENT & ENP) & ( COUNT+1 ) # !LD & !(ENT & ENP) & ( COUNT );

RCO=(COUNT=[1, 1, 1, 1]) & ENT

end Z74X161
```

Yes it still fits in a GAL16V8 as the number of pins available is enough to program the operation of 74X161.

Step 1 of 2 8.14DP

Consider the following data:

Inputs ENP, ENT, and D always HIGH

Inputs A, B, C always LOW

Input $\mathrm{LD\_L}=(\mathrm{QA}\cdot\mathrm{QC})'$ implies Gate NAND of outputs (QA, QC) is fed to input LD_L.

Similarly $\mathrm{CLR\_L}=(\mathrm{QB}\cdot\mathrm{QD})'$ implies Gate NAND of outputs (QB, QD) is fed to input $\mathrm{CLR\_L}$.

The CLK input is hooked up to a free-running clock signal.

Draw the 74×163 modulo-16 counter as per the data.

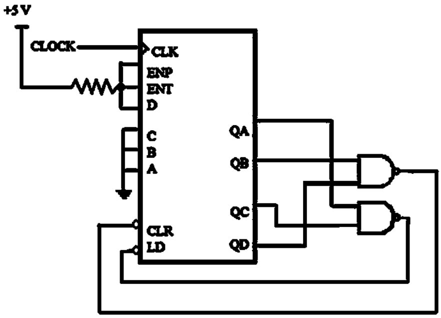


Figure 1

Step 2 of 2

The sequence starts with the state 0000 and then counts to 0001 till 0101 as shown in Table 1. Since QA and QC is fed as an input to the NAND gate, at the state 0101 the LD becomes 0 and the input is fed as the output, the state becomes 1000 when it is clocked and then counts to 1001 till 1010. Since QB and QD is fed as an input to the NAND gate, at the state 1010 the CLR becomes 0 which resets the state to 0000.

The Output sequence for the circuit is shown in table 1.

Table 1

QD	QC	QB	QA
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
1	0	0	0
1	0	0	1
1	0	1	0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1

Thus, desired sequence is tabulated.

Step 1 of 2 **8.15DP**

Consider each internal gate delay is 10ns.

The internal structure of 74X138 decoder is shown in Figure 6-35. The Propagation delay of each gate is 10 ns. From the Figure 6-35, it is clear that the total propagation delay is provided by all the three gates.

Draw the internal structure of 74X138 decoder.

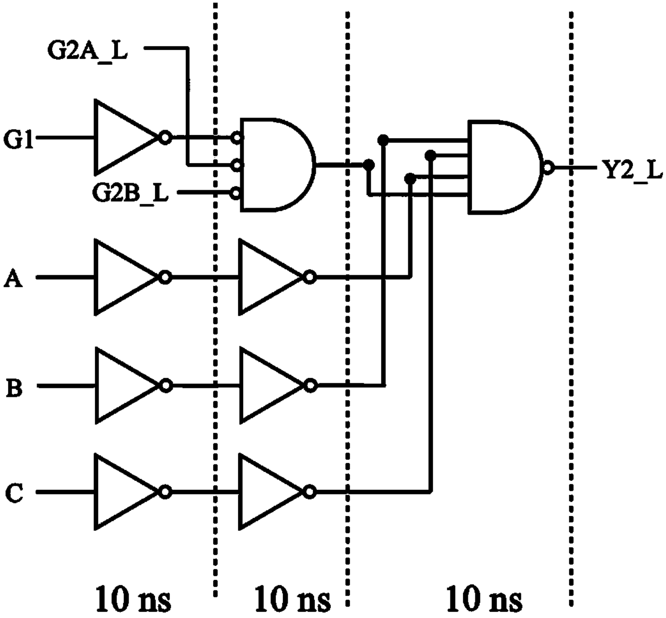


Figure 1

Step 2 of 2

The glitches are the short pulses usually occur in the decoders during the transitions. Glitches are the functions of propagation delay. Glitch width is equal to the total propagation delay. Thus

Glitch width of the Figure 8-34 is:

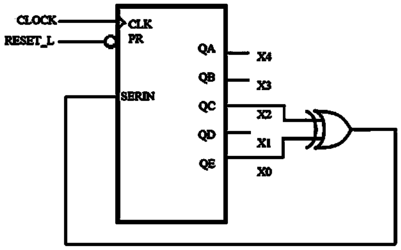
$$t_g = 10\text{ ns} + 10\text{ ns} + 10\text{ ns}$$
$$= 30\text{ ns}$$

Thus, each glitch width of Figure 8-34 for Y2_L is $\boxed{30\text{ ns}}$.

Step 1 of 4

8.16DP

Using the Figure 8-51 and Table 8-26 a 5-bit LFSR counter is shown in Figure 1.



Step 2 of 4

Figure 1

Step 3 of 4

X2 and X0 are fed to the XOR gate and the output is feedback to the SERIN. The sequence of the circuit in Figure 1 is shown in Table 1.

Table 1

X4	X3	X2	X1	X0
0	0	0	0	1
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
1	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0
0	1	1	0	1
0	0	1	1	0
1	0	0	1	1

Step 4 of 4

The sequence starts with the state 00001, at the first clock the SERIN value is 1 and the output becomes 10000. Based on the SERIN and previous outputs the sequence is as shown in Table 1.

Step 1 of 2

8.17DP

From the Table 8-43 in the textbook, the T_o and τ of TI's estimates for the 74LS74 family is:

$T_o = 4.8 \cdot 10^{-3}$ s

$\tau = 1.35$ ns

Write the formula for mean time between failures

$$\mathrm{MTBF}(t_r) = \frac{\exp\left(\frac{t_r}{\tau}\right)}{T_o \cdot f \cdot a}$$

For the first synchronizer the $t_r = 80$ ns and frequency is 10 MHz. If the asynchronous input changes 100,000 times per second, then the synchronizer MTBF is

$$\begin{aligned}\mathrm{MTBF}(80\,\mathrm{ns}) &= \frac{\exp\left(\frac{80}{1.35}\right)}{4.8 \cdot 10^{-3} \cdot 10^{7} \cdot 10^{5}} \\ &= 1.134 \cdot 10^{16}\ \mathrm{s}\end{aligned}$$

MTBF is about 360 million years between failures.

Step 2 of 2

For the second synchronizer the $t_r = 42.5$ ns and frequency is 16 MHz. If the asynchronous input changes 100,000 times per second, then the synchronizer MTBF is

$$\begin{aligned}\mathrm{MTBF}(42.5\,\mathrm{ns}) &= \frac{\exp\left(\frac{42.5}{1.35}\right)}{4.8 \cdot 10^{-3} \cdot 1.6 \cdot 10^{7} \cdot 10^{5}} \\ &= 6121.71\ \mathrm{s!}\end{aligned}$$

The synchronizer at 16 MHz is too lousy. Thus synchronizer at 10 MHz is recommended for the 74LS74 with TI's estimates.

Step 1 of 2

From the Table 8-43 in the textbook, the T_o and τ of TI's estimates for the 74LS74 family is:

$$T_o = 4.8 \cdot 10^{-3}\ \text{s}$$
$$\tau = 1.35\ \text{ns}$$

Write the formula for mean time between failures:

$$\text{MTBF}(t_r) = \frac{\exp\left(\frac{t_r}{\tau}\right)}{T_o \cdot f \cdot a}$$

For the first synchronizer the $t_r = 80$ ns and frequency is 10 MHz. If the asynchronous input change can occur on every clock tick then *f* is substituted for the parameter *a*, then the synchronizer MTBF is:

$$\text{MTBF}(80\,\text{ns}) = \frac{\exp\left(\frac{80}{1.35}\right)}{4.8 \cdot 10^{-3} \cdot 10^{7} \cdot 10^{5}}$$
$$= 1.134 \cdot 10^{16}$$

MTBF is about 3.6 million years between failures.

Step 2 of 2

For the second synchronizer the $t_r = 42.5$ ns and frequency is 16 MHz. If the asynchronous input change can occur on every clock tick then *f* is substituted for the parameter *a*, then the synchronizer MTBF is:

$$\text{MTBF}(42.5\,\text{ns}) = \frac{\exp\left(\frac{42.5}{1.35}\right)}{4.8 \cdot 10^{-3} \cdot 1.6 \cdot 10^{7} \cdot 1.6 \cdot 10^{7}}$$
$$= 38.26$$

The synchronizer at 16 MHz is too lousy. Thus synchronizer at 10 MHz is recommended for the 74LS74 with TI's estimates.

Step 1 of 1 **8.19DP**

Refer to Figure 8-76 in the text book.

The device name for the D-flip-flops used is 74F74.

Take the following parameters from Table 8-43 in the text book.

The constant, $T_o = 2.0 \cdot 10^{-4}$ s

The constant, $\tau = 0.40$ ns ns

The clock frequency, is,

$$\begin{aligned} f &= 25 \text{ MHz} \\ &= 2.5 \cdot 10^{7} \text{ Hz} \end{aligned}$$

The number of asynchronous input changes per second is,

$$\begin{aligned} a &= 1 \text{ MHz} \\ &= 10^{6} \text{ Hz} \end{aligned}$$

Write the formula for the mean time between synchronizer failures.

$$\text{MTBF}(t_r) = \frac{\exp\left(\frac{t_r}{\tau}\right)}{T_o \cdot f \cdot a}$$

Substitute 35 for t_r, 0.4 for τ $2.5 \cdot 10^{7}$ for *f*, $2.0 \cdot 10^{-4}$ for T_o and 10^{6} for *a*.

$$\begin{aligned} \text{MTBF}(36\text{ns}) &= \frac{\exp\left(\frac{35}{0.4}\right)}{\left(2.0 \cdot 10^{-4}\right)\left(2.5 \cdot 10^{7}\right)\left(10^{6}\right)} \\ &= \frac{(1.0018).10^{38}}{5.10^{9}} \\ &= 2.10^{28} \end{aligned}$$

Thus, the MTBF of the synchronizer is $\boxed{2.10^{28} \text{ s}}$.

Step 1 of 2 8.20DP

Refer to Table 8-43 from the textbook.

Observe from the table the typical 74ALS74 constants are:

$$T_o = 8.7 \cdot 10^{-6}$$
$$\tau = 1.00\ \text{ns}$$

Transition rate a is $2 \cdot 10^6$

Clock frequency f is $30\ \text{MHz}$

Set up time t_{setup} and propagation delay are equal to $10\ \text{ns}$

Write the expression for mean time between failures (MTBF).

$$\text{MTBF}(t_r) = \frac{\exp(t_r / \tau)}{T_o \cdot f \cdot a}$$

The setup time and propagations delays are 10 ns.

The clock frequency is $30\ \text{MHz}$.

The clock period is,

$$\begin{aligned} t_{\text{clk}} &= \frac{1}{f} \\ &= \frac{1}{30\ \text{MHz}} \\ &= \frac{1}{30 \times 10^6} \\ &= 33.33\ \text{ns} \end{aligned}$$

Step 2 of 2

Refer to synchronizer shown in Figure X8.20 form the textbook.

Observe from figure the propagation delay of the combinational logic and setup delay of the flip-flops is equal to $10\ \text{ns}$.

Write the expression for meta-stability resolution time t_r .

$$\begin{aligned} t_r &= t_{\text{clk}} - t_{\text{comb}} - t_{\text{setup}} \\ &= 33.33\ \text{ns} - 10\ \text{ns} - 10\ \text{ns} \\ &= 13.33\ \text{ns} \end{aligned}$$

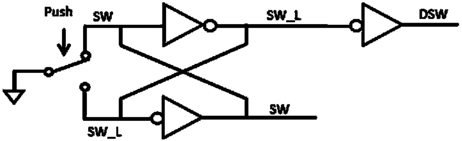
Find the mean time between failures for the synchronizer at $13.33\ \text{ns}$. $\text{MTBF}(t_r) = \frac{\exp(t_r / \tau)}{T_o \cdot f \cdot a}$

$$\begin{aligned} \text{MTBF}(13.33\ \text{ns}) &= \frac{\exp(t_r / \tau)}{T_o \cdot f \cdot a} \\ &= \frac{\exp(13.33/1.00)}{8.7 \cdot 10^{-6} \cdot 3 \cdot 10^7 \cdot 2 \cdot 10^6} \\ &= 1.178 \times 10^{-3} \end{aligned}$$

MTBF is for the synchronizer is $\boxed{1178\ \mu\text{s}}$.

Step 1 of 3

The switch debouncing circuit is given below:



Step 2 of 3

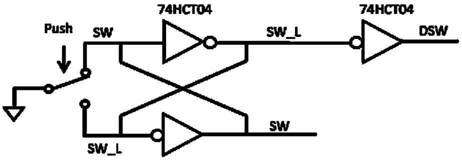
Step 3 of 3

From the TTL and CMOS data sheets it is known that the above circuit should not be used with high-speed CMOS devices, like the 74ACT04, whose outputs are capable of sourcing large amounts of current in the HIGH state. While shorting such outputs to ground momentarily will not cause any damage, it will generate a noise pulse on power and ground signals that may trigger improper operation of the circuit elsewhere. The debouncing circuit in above figure works well with families like HCT and LS-TTL.

Yes, the answer varies from slow logic family to fast logic family.

Step 1 of 2

The switch debounce circuit is given below:



Step 2 of 2

In the above circuit, with 74HCT04 inverters, before the button is pushed, the top contact holds SW at 0 V, a valid logic 0, and the top inverter produces a logic 1 on SW_L and on the bottom contact. When the button is pushed and contact is broken feedback in the bistable holds SW at V_{OL} (≤ 0.33 V for HCTCMOS), still a valid 0.

Next, when the wiper hits the bottom contact, the circuit operated quite unconventionally for a moment. The top inverter in the bistable is trying to maintain a logic 1 on the SW_L signal; the top transistor in its totem-pole output is “on” and connecting SW_L through a small resistance to +5 V. Suddenly, the switch contact makes a metallic connection of SW_L to ground, 0.0 V.

After 70 ns for 74HCT04, the forced logic 0 on SW_L propagates through the two inverters of the bistable, so that the top inverter gives up its vain attempt to drive a 1, and instead drives a logic 0 onto SW_L. At this point, the top inverter output is no longer shorted to ground, and feedback in the bistable maintains the logic on SW_L even if the wiper bounces off the bottom contact, as it does.

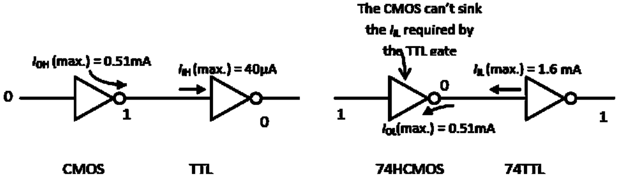
The above circuit should not be used with high-speed CMOS devices, like the 74AC04, whose outputs are capable of sourcing large amounts of current in the HIGH state. While shorting such outputs to ground momentarily will not cause any damage, it will generate a noise pulse on power and ground signals that may trigger improper operation of the circuit elsewhere. Therefore, the above debouncing circuit with 74AC04 inverters will not work properly.

Step 1 of 2

When driving TTL from CMOS, the voltage levels are no problem because the CMOS output approximately 4.95 V for HIGH and 0.05 V for a LOW, which is easily interpreted by the TTL gate.

But the current levels can be a real concern because CMOS has severe output-current limitations.

The following figures show the input/output currents that flow when interfacing CMOS to TTL:



Step 2 of 2

For the HIGH output condition, the CMOS can source a maximum current of 0.51 mA, which is enough to supply the HIGH-level input current (I_{IH}) to one TTL gate. But for the LOW output condition, the COS can also sink only 0.51 mA, which is not enough for the TTL LOW-level input current (I_{IL}).

Therefore, the CMOS bus holder circuit doesn't work well three state buses with TTL devices attached.

Step 1 of 3

The diagram of *T*-flip flop is shown in Figure 1.

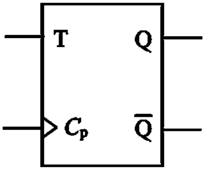


Figure 1

Step 2 of 3

To design 4-bit ripple down counter, there are four flip-flops are required. 4-bit ripple counter can be formed by cascading four *T* flip-flops together by connecting *Q* output of one flip-flop to the clock input to the next flip flop.

The circuit of the 4-bit ripple down counter using four *T* flip-flops is shown in Figure 2.

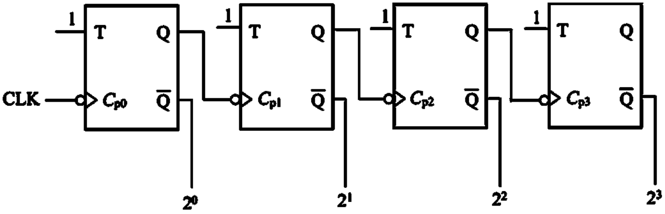


Figure 2

Step 3 of 3

With four flip-flops, we can produce 2^4 different combinations of binary outputs $2^4 = 16$, we can count from 0000 up to 1111 or from 1111 down to 0000. If the output is taken from the *Q* output of each flip-flop then the counter becomes up counter and if the output is taken from the $\overline{Q}$ output then the counter becomes a down counter.

See that Q_0 toggles at each negative edge of $\overline{C_{p0}}$, Q_1 toggles at each negative of Q_0, and Q_2 toggles at each negative edge of Q_1 and Q_3 toggles at each negative edge of Q_2. The outputs are taken from $\overline{Q}$ output of each flip-flop for down counter. The result is that the output will count repeatedly from 1111 down to 0000.

Therefore, the 4-bit ripple down counter using four *T* flip-flops with no other components is designed.

Step 1 of 2

The circuit of the 4-bit ripple down counter using four *D* flip-flops is shown in Figure 1.

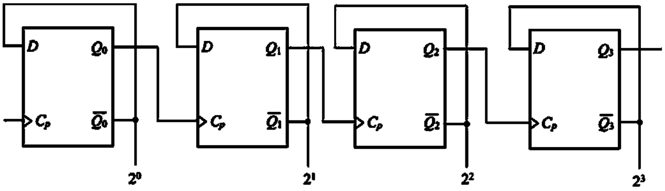


Figure 1

Considering the propagation delays, the 2^0 waveform will be delayed to the right (skewed) by the propagation of the first flip-flop. The 2^1 waveform will be skewed to the right from the 2^0 waveform, and the 2^2 waveform will be skewed to the right from the 2^1 waveform, and the 2^3 waveform will be skewed to the right from 2^2 waveform. This is a cumulative effect that causes the 2^3 waveform to be skewed to the right of the original C_{p0} waveform by four propagation delays.

Propagation delay for 74HCT *D* flip-flop:

Since, the propagation delay for 74HCT *D* flip-flop is 16 ns .

Calculate the maximum propagation delay from clock to output.

$$\begin{aligned} t_{pd} &= n(16\text{ ns}) \\ &= 4(16\text{ ns}) \\ &= 64\text{ ns} \end{aligned}$$

Therefore, the maximum propagation delay is $\boxed{64\text{ ns}}$.

Propagation delay for 74AHCT *D* flip-flop:

Since, the propagation delay for 74AHCT *D* flip-flop is 5 ns.

Calculate the maximum propagation delay from clock to output.

$$\begin{aligned} t_{pd} &= n(5\text{ ns}) \\ &= 4(5\text{ ns}) \\ &= 20\text{ ns} \end{aligned}$$

Therefore, the maximum propagation delay is $\boxed{20\text{ ns}}$.

Step 2 of 2

Propagation delay for 74LS74 *D* flip-flop:

Since, the propagation delay for 74LS74 *D* flip-flop is 40 ns.

Calculate the maximum propagation delay from clock to output.

$$\begin{aligned} t_{pd} &= n(40\text{ ns}) \\ &= 4(40\text{ ns}) \\ &= 160\text{ ns} \end{aligned}$$

Therefore, the maximum propagation delay is $\boxed{160\text{ ns}}$.

Step 1 of 1

The maximum counting speed of a ripple counter is limited by the following parameters:

• If the propagation delay of the flip-flop is in the range of the clock period then the counting speed of a ripple counter will be affected.

• The maximum counting speed of a ripple counter is affected by the numbers of flip-flops.

• It is affected by the average propagation delay of each flip-flop.

As we cascade more and more flip-flop to form higher modulus counters, the cumulative effect of the propagation delay becomes more. The approximate maximum frequency $f_{\max}$ of a ripple counter due to the accumulation of propagation delays can be determined by using the following formula.

$$f_{\max} = \frac{1}{Nt_{pd}}$$

Here,

N is the number of flip-flops,

t_{pd} is the average propagation delay of each flip-flop $\left(C_p \text{ to } Q\right)$.

$f_{\max}$ is the maximum counting speed or frequency.

The circuit of the 4-bit ripple down counter using four *D* flip-flops is shown in Figure 1.

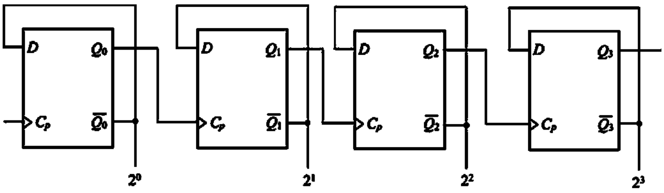


Figure 1

As we cascade more and more flip-flop to form higher modulus counters, the cumulative effect of the propagation delay becomes more. The approximate maximum frequency $f_{\max}$ of a ripple counter due to the accumulation of propagation delays can be determined by using the following formula.

$$f_{\max} = \frac{1}{Nt_{pd}} \quad \text{...... (1)}$$

Here,

N is the number of flip-flops,

t_{pd} is the average propagation delay of each flip-flop $(C_p \text{ to } Q)$.

$f_{\max}$ is the maximum counting speed or frequency.

Propagation delay for 74HCT *D* flip-flop:

Since, the propagation delay for 74HCT *D* flip-flop is 16 ns . The number of flip flops needed to design 4-bit ripple down counter are 4.

Calculate the maximum counting speed (frequency) for 74HCT.

$$\begin{aligned} f_{\max} &= \frac{1}{(4)(16\times10^{-9})} \\ &= \frac{1}{64\times10^{-9}} \\ &= 15.625 \text{ MHz} \end{aligned}$$

Therefore, the maximum counting speed (frequency) for 74HCT is $\boxed{15.625 \text{ MHz}}$.

Propagation delay for 74AHCT *D* flip-flop:

Since, the propagation delay for 74HCT *D* flip-flop is 5 ns . The number of flip flops needed to design 4-bit ripple down counter are 4.

Calculate the maximum counting speed (frequency) for 74AHCT.

$$\begin{aligned} f_{\max} &= \frac{1}{(4)(5\times10^{-9})} \\ &= \frac{1}{20\times10^{-9}} \\ &= 50 \text{ MHz} \end{aligned}$$

Therefore, the maximum counting speed (frequency) for 74AHCT is $\boxed{50 \text{ MHz}}$.

Propagation delay for 74LS74 *D* flip-flop:

Since, the propagation delay for 74LS74 *D* flip-flop is 40 ns . The number of flip flops needed to design 4-bit ripple down counter are 4.

Calculate the maximum counting speed (frequency) for 74LS74.

$$\begin{aligned} f_{\max} &= \frac{1}{(4)(40\times10^{-9})} \\ &= \frac{1}{160\times10^{-9}} \\ &= 6.25 \text{ MHz} \end{aligned}$$

Therefore, the maximum counting speed (frequency) for 74LS74 is $\boxed{6.25 \text{ MHz}}$.

Step 1 of 1 **8.031E**

Refer to the Figure 8-25 in textbook for the circuit of synchronous serial binary counter.

It is clear that four flip-flops are used in synchronous serial binary counter and they are connected serially. The propagation delay from T to Q of T-flip flop is denoted by t_{TQ} . Therefore, the propagation delay of all four flip-flops will be added from input to output.

Therefore, the total propagation delay of all flip-flops is $4t_{\mathrm{TQ}}$. Similarly, the total set-up time of the circuit is $4t_{\mathrm{setup}}$.

Three AND gates are used in this counter. The total delay of all three AND gates will also be added being connected serially. The delay of AND gate is t_{AND} . Therefore, the time delay of AND gates is $3t_{\mathrm{AND}}$.

The total time taken from input to output for synchronous serial binary counter is, $T = 4t_{\mathrm{TQ}} + 4t_{\mathrm{setup}} + 3t_{\mathrm{AND}}$

It is known that the maximum clock frequency is,

$$f_{\max} = \frac{1}{T}$$

Therefore,

$$f_{\max} = \frac{1}{4t_{\mathrm{TQ}} + 4t_{\mathrm{setup}} + 3t_{\mathrm{AND}}}$$

Therefore, the formula for maximum clock frequency, $f_{\max}$ of synchronous serial binary counter is

$$\boxed{\frac{1}{4t_{\mathrm{TQ}} + 4t_{\mathrm{setup}} + 3t_{\mathrm{AND}}}}.$$

Step 1 of 1

Refer to the Figure 8-26 in textbook for the circuit of synchronous parallel binary counter.

Four flip-flops are used in synchronous parallel binary counter and they are connected serially. The propagation delay from T to Q of T-flip flop is denoted by t_{TQ} . Therefore, the propagation delay of all four flip-flops will be added from input to output.

Therefore, the total propagation delay of all four flip-flops is $4t_{\mathrm{TQ}}$.

Now, in the circuit the operation of Enable signal and operation of AND gates are parallel.

Therefore, the propagation delay for Enable signal and for AND gate is $t_{\mathrm{setup}}+t_{\mathrm{AND}}$.

The total time is,

$$4t_{\mathrm{TQ}}+t_{\mathrm{setup}}+t_{\mathrm{AND}}$$

It is known that the maximum clock frequency is,

$$f_{\max}=\frac{1}{T}$$

Therefore,

$$f_{\max}=\frac{1}{4t_{\mathrm{TQ}}+t_{\mathrm{setup}}+t_{\mathrm{AND}}}$$

Therefore, the formula for maximum clock frequency, $f_{\max}$ of synchronous parallel binary counter is

$$\boxed{\frac{1}{4t_{\mathrm{TQ}}+t_{\mathrm{setup}}+t_{\mathrm{AND}}}}.$$

By comparing of both synchronous serial binary counter and synchronous parallel binary counter, the operation of both enable and AND gate signal are in parallel in synchronous parallel binary counter. The main comparison of both counters is based on the setup time and the AND gate time delay.

The deference in speed is that the delay encountered during the propagation of enable (*EN*) signals. Parallel carry counter is faster than the serial counter.

Refer to the Figure 8-25 in textbook for the circuit of 4-bit synchronous serial binary counter.

Draw the *n*-bit synchronous serial binary counter:

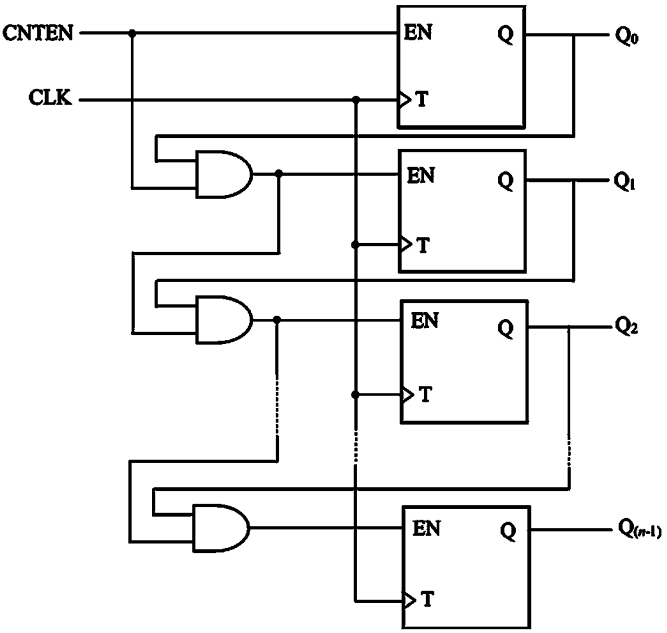


Figure 1

For *n*-bit counter, *n*-flip-flops are used in synchronous serial binary counter and they are connected serially. The propagation delay from T to Q of T-flip flop is denoted by t_{TQ}. Therefore, the propagation delay of *n*-flip-flops will be added from input to output.

Therefore, the total propagation delay of *n*-flip-flops is nt_{TQ}. Similarly, the total set-up time of the circuit is nt_{setup}.

For *n*-bit counter, $(n-1)$ AND gates are needed to design. The total delay of $(n-1)$ AND gates will also be added being connected serially. The delay of AND gate is t_{AND}. Therefore, the time delay of $(n-1)$ AND gates is $(n-1)t_{AND}$.

The total time taken from input to output for synchronous serial binary counter is,

$$T = nt_{TQ} + nt_{setup} + (n-1)t_{AND}$$

It is known that the maximum clock frequency is,

$$f_{max} = \frac{1}{T}$$

Therefore,

$$f_{max} = \frac{1}{nt_{TQ} + nt_{setup} + (n-1)t_{AND}}$$

Therefore, the formula for maximum clock frequency, f_{max} of synchronous serial binary counter is $\boxed{\frac{1}{nt_{TQ} + nt_{setup} + (n-1)t_{AND}}}$.

Step 1 of 3

The 74x163 is a high-speed synchronous MOD-16 binary counter. This is synchronously pre-settable by $\overline{LD}$ (Parallel Enable input) for application in programmable dividers. It has two count enabler inputs such as *ENP* (Count Enable Parallel input) and *ENT* (Count Enable Trickle input). The 74163 has a Terminal Count output (*RCO*) to facilitate high speed synchronous counting. It has a Synchronous Reset ($\overline{CLR}$) input that overrides counting and parallel loading and allows the outputs to be simultaneous Reset on the rising edge of the clock.

Both count enables (*ENP* and *ENT*) must be HIGH to count. The terminal count output (*RCO*) will go HIGH when the highest count is reached. *RCO* will be forced LOW, however, when *ENT* goes LOW, even though the highest count may be reached.

Step 2 of 3

Now, the count sequence is 4, 5, 6… 13, 14, 4, 5, 6 …

It is observed that the sequence is starting from number 4. Represent this number in binary form 4 (binary 0100). The count starts from 4.

Therefore, for count to start from 4 (binary 0100) we have to pre-load the counter with 0100 (*DCBA*), with $\overline{LD} = 0$.

It is also observed from the count sequence that the highest count is 14 (binary 1110). Therefore, when the count reaches 15 (binary 1111), the counter starts its counting from pre-set count that is 4 (binary 0100).

The design of the counter is shown in Figure 1.

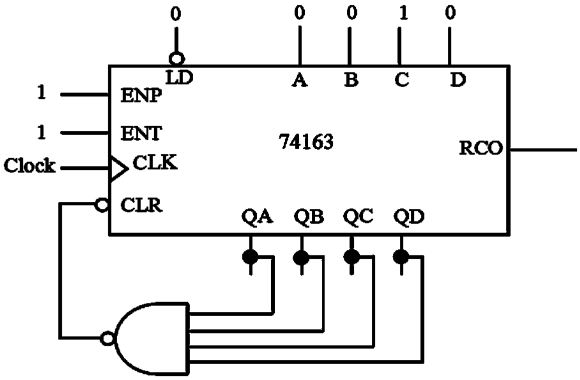


Figure 1

Step 3 of 3

Take the QA, QB, QC, and QD as outputs and connect it with the synchronous clear $\left(\overline{CLR}\right)$ through a NAND gate, then when the count reaches 15 (binary 1111) the $\left(\overline{CLR}\right)$ input becomes LOW (0) and the counter would RESET to 4 (binary 0100) and the counting starts from 4 (binary 0100) again.

Step 1 of 2

8.037E

Draw the cascading scheme of 36-bit synchronous parallel counter using nine 74x163

(4-bit binary counter).

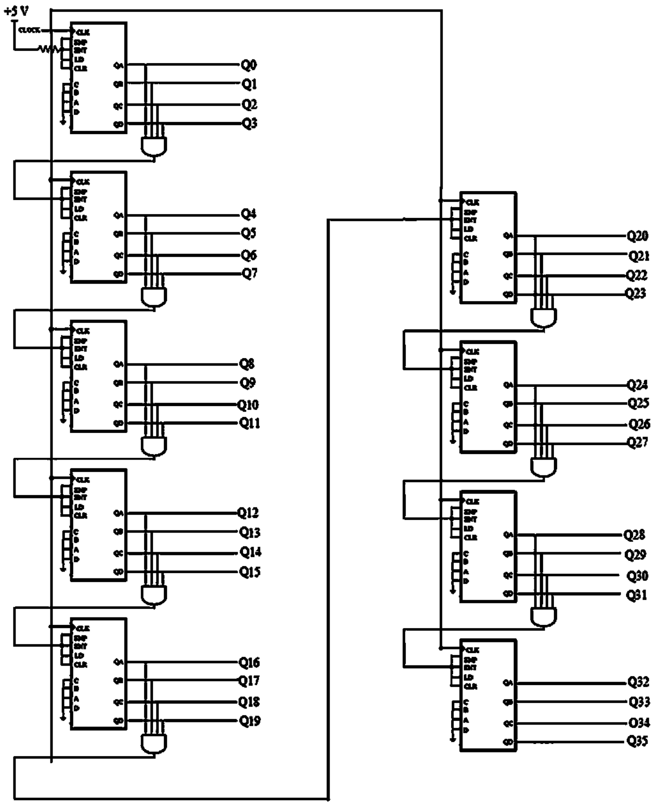


Figure 1

Step 2 of 2

In Figure 1, Q35 is the MSB and Q0 is the LSB.

The counting starts from 000000000000000000000000000000000000 and ends at 111111111111111111111111111111111111, thus covering all the states.

From the manufacturer's data sheet consider the following data:

The propagation delay of 74x163 is 14 nS.

The maximum clock frequency is 32 MHz.

Time period of one clock cycle is,

$$\frac{1}{32\text{ MHz}} = 31\text{ ns}$$

Determine the counting speed per instruction.

$$\Rightarrow \textbf{Time period + propagation delay}$$
$$\Rightarrow 31 + 14 = 45\text{ ns}$$

Thus, the maximum counting speed is $\boxed{45\text{ ns}}$.

Step 1 of 4

8.038E

The 74x163 is a high-speed synchronous MOD-16 binary counter. This is synchronously pre-settable by $\overline{LD}$ (Parallel Enable input) for application in programmable dividers. It has two count enabler inputs: *ENP* (Count Enable Parallel input) and *ENT* (Count Enable Trickle input). 74x163 has a terminal Count output (*RCO*) to facilitate high speed synchronous counting. It has a Synchronous Reset ($\overline{CLR}$) input that overrides counting and parallel loading and allows the outputs to be simultaneous Reset on the rising edge of the clock.

Both count enables (*ENP* and *ENT*) must be HIGH to count. The terminal count output (*RCO*) will go HIGH when the highest count is reached. *RCO* will be forced LOW, however, when *ENT* goes LOW, even though the highest count may be reached.

Step 2 of 4

The modulo-129 counter starts counting from 0, 1, 2, 3…128, 0, 1, 2… and so on.

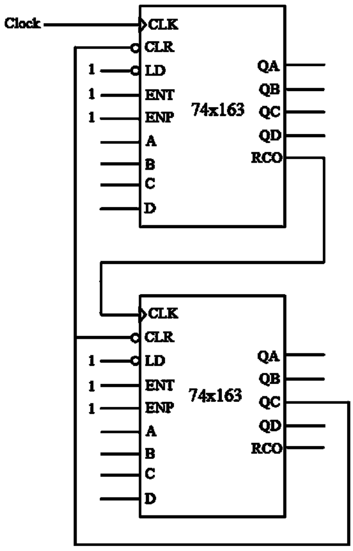
$127_{10} = 0111\ 1111_2$; and

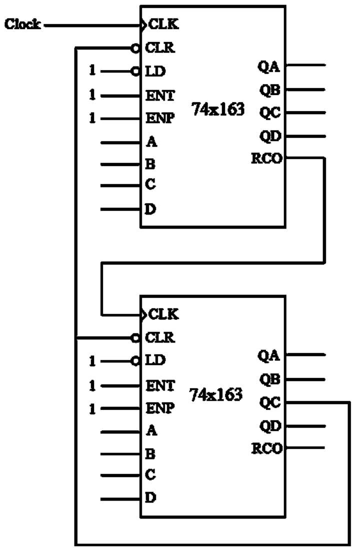
$128_{10} = 1000\ 0000_2$

When the count reaches 128, in the next clock the count must become 0 (zero) for modulo-129 counter.

Therefore, when the *QC* output of the second 74x163 becomes 0 from 1, then in the next clock the count must start its counting from 0.

The connection arrangement for module 0-129 counter using only two 74x163 is shown in Figure 1.





Step 3 of 4

Figure 1

Step 4 of 4

When the count reaches 127 (binary 0111 1111), in the next clock cycle the count becomes 128 (binary 1000 0000) and the *QC* output of the second 74x163 counter changes to 0 from 1.

The *QC* output of the second 74x163 counter is connected to the CLR input of the both the 74x163 counter. So, when the count reaches 128, in the next clock period the count reset to 0 of both the 74x163 counter and the count starts again from 0. Thus, it is a modulo-129 counter.

Step 1 of 1 8.040E

VHDL program for an 8-bit modulo-*N* counter is,

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
entity modncount is
generic (N: natural := 10);
port(
CLR,CLK,LD:in STD_LOGIC; //clear input CLR, load input LD
A:in STD_LOGIC_VECTOR(7 downto 0);
Q:out STD_LOGIC_VECTOR(7 downto 0)
);
end modncount;
architecture modncount_arch of modncount is
signal count: STD_LOGIC_VECTOR(7 downto 0);
begin
process(CLK,CLR)
begin
if CLR='1' then
COUNT<="00000000";
elsif (CLK'event and CLK='1') then
if (LD='0') then
if COUNT <= N then
COUNT <= "00000000";
else
COUNT <= COUNT + 1;
end if;
elsif(LD='1') then
COUNT<=A;
end if;
end if;
end process;
Q <= COUNT;
end modncount_arch;
```

The program has been executed in Xlinx 13.2.

Step 1 of 2

Consider four inputs **N3, N2, N1, and N0** and one output Z for a clocked synchronous circuit.

Consider that the output Z asserts only for exactly *N* clock ticks during any 16 ticks offered by the combination of inputs **N3, N2, N1, and N0** .

Consider the following circuit for the design of the clocked synchronous circuit:

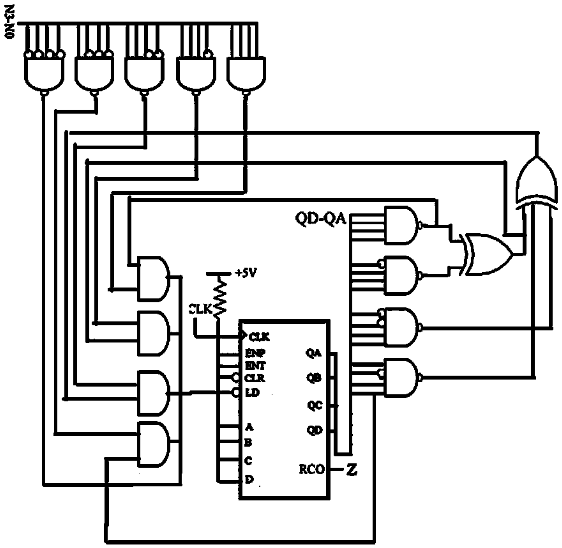


Figure 1

Step 2 of 2

Consider the following data for the operation of the circuit in Figure 1:

If $N = 15$, the output Z asserts when the count is 16. If $N = 8$, the output asserts when the count is 8 and 16. If $N = 4$, the output asserts when the count is 4, 8, 12 and 16. If $N = 2$, the output asserts when the count is 2, 4, 6, 8, 10, 12, 14 and 16. If $N = 0$, the output asserts continuously.

Step 1 of 2

The clocked synchronous circuit has eight inputs N7-N0. That represents an integer *N* in the range of 0-255. If $N = 255$, then *Z* will be asserted when the count is 255. If $N = 128$, then Z will be asserted when the count is 128 and 256. If $N = 0$, then *Z* will be asserted continuously.

Write the ABEL program to realize the circuit.

```
module counter
title 'Eight-input counter'
COUNTER device 'PAL22V10'
" Input and Output pins
CLK, CLR, LD, ENP, ENT pin;
N1, N2, N3,N4, N5, N6, N7, N8, Z pin istype 'reg';
" Definitions
N = [N1, N2, N3, N4, N5, N6, N7, N8];
I = [I1, I2, I3, I4, I5, I6, I7, I8];
" Equations
Z.CLK = CLK;
when CLR then I:=0;
when LD then I := N;
when (ENP== 1 & ENT := 1) then I:=I+1;
when N==1 then Z:=1;
when (N==2 & I8==0) then Z:=1;
when (N==4 & I==[.X., .X., .X., .X., .X., .X., 0, 0]) then Z:=1;
when (N==8 & I==[.X., .X., .X., .X., .X., 0, 0, 0]) then Z:=1;
when (N==16 & I==[.X., .X., .X., .X., 0, 0, 0, 0]) then Z:=1;
when (N==32 & I==[.X., .X., .X., 0, 0, 0, 0, 0]) then Z:=1;
when (N==64 & I==[.X., .X., 0, 0, 0, 0, 0, 0]) then Z:=1;
when (N==128 & I==[.X., 0, 0, 0, 0, 0, 0, 0]) then Z:=1;
end counter
```

Step 2 of 2

```
module counter
title 'Eight-input counter'
COUNTER device 'PAL22V10'
" Input and Output pins
CLK, CLR, LD, ENP, ENT pin;
N1, N2, N3,N4, N5, N6, N7, N8, Z pin istype 'reg';
" Definitions
N = [N1, N2, N3, N4, N5, N6, N7, N8];
I = [I1, I2, I3, I4, I5, I6, I7, I8];
COUNT = [0, 0, 0, 0, 0, 0, 0, 0];
" Equations
Z.CLK = CLK;
when COUNT < N then
when CLR then I:=0;
when LD then I := N;
when (ENP== 1 & ENT := 1) then I:=I+1;
when N==1 then Z:=1 COUNT = COUNT + 1;
when (N==2 & I8==0) then Z:=1 COUNT = COUNT + 1;
when (N==4 & I==[.X., .X., .X., .X., .X., .X., 0, 0]) then Z:=1 COUNT = COUNT + 1;
when (N==8 & I==[.X., .X., .X., .X., .X., 0, 0, 0]) then Z:=1 COUNT = COUNT + 1;
when (N==16 & I==[.X., .X., .X., .X., 0, 0, 0, 0]) then Z:=1 COUNT = COUNT + 1;
when (N==32 & I==[.X., .X., .X., 0, 0, 0, 0, 0]) then Z:=1 COUNT = COUNT + 1;
when (N==64 & I==[.X., .X., 0, 0, 0, 0, 0, 0]) then Z:=1 COUNT = COUNT + 1;
when (N==128 & I==[.X., 0, 0, 0, 0, 0, 0, 0]) then Z:=1 COUNT = COUNT + 1;
else Z:=0;
end counter
```

Therefore, the clocked synchronous circuit is realized using ABEL program.

Step 1 of 6 **8.045E**

Consider a synchronous sequential circuit clocked having four inputs $N3, N2, N1, N0$ which represents an integer in the range of 0-15 with single output Z having asserted to N clock ticks in the 16-tick interval.

This is implemented by using free running divide by 16 counter and the ticks of Z are evenly spaced for as possible.

The VHDL code for implementation of this circuit is as follows:

Step 2 of 6

Step 3 of 6

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.NUMERIC_STD.all;
entity counter is
port (CLK, CLR_L, LD_L, ENP, ENT: in STD_LOGIC; N: in unsigned(7 downto 0); Z: out STD_LOGIC);
end counter;
architecture counter_arch of counter is
signal IQ: unsigned(7 downto 0);
begin
process(CLK, ENT, IQ)
begin
if(CLK'event and CLK = '1') then
if CLR_L = '0' then IQ <= (others => '0');
elsif LD_L = '0' then IQ <= N;
end if;
if (ENP = '1' and ENT = '1') then
IQ <= IQ+1;
end if;
if (N = "00000001") then
Z <= '1';
end if;
if(N = "00000010" and IQ(0) = '0') then
Z <= '1';
end if;
if(N = "00000100" and IQ = "XXXXXX00") then
Z <= '1';
end if;
if(N = "00001000" and IQ = "XXXXX000") then
Z <= '1';
end if;
if(N = "00010000" and IQ = "XXXX0000") then
Z <= '1';
end if;
if(N = "00100000" and IQ = "XXX00000") then
Z <= '1';
end if;
if(N = "01000000" and IQ = "XX000000") then
Z <= '1';
end if;
if(N = "10000000" and IQ = "X0000000") then
Z <= '1';
end if;
end if;
end process;
end counter_arch;
```

Step 4 of 6

thus, the execution of this code results a synchronous sequential circuit clocked having four inputs $N3, N2, N1, N0$ which represents an integer in the range of 0-15 with single output Z having asserted to N clock ticks in the 16-tick interval.

Step 5 of 6

Consider a synchronous sequential circuit clocked having four inputs $N3, N2, N1, N0$ which represents an integer in the range of 0-15 with single output Z having asserted to N transitions in the 16-tick interval known as binary rate multiplier.
This is implemented by using free running divide by 16 counter and the ticks of Z are connected such that the level output is clocked from the previous input.

The VHDL code for implementation of this circuit is as follows:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.NUMERIC_STD.all;
entity counter is
port (CLK, CLR_L, LD_L, ENP, ENT: in STD_LOGIC; N: in unsigned(7 downto 0); Z: out STD_LOGIC);
end counter;
architecture counter_arch of counter is
signal IQ,c: unsigned(7 downto 0);
begin
process(CLK, ENT, IQ)
begin
L1:loop
if(CLK'event and CLK = '1') then
if CLR_L = '0' then IQ <= (others => '0');
elsif LD_L = '0' then IQ <= N;
end if;
if (ENP = '1' and ENT = '1') then
IQ <= IQ+1;
end if;
if (IQ = "00000001" or IQ = "00000010" or IQ = "00000100" or IQ = "00001000" or IQ = "00010000" or IQ = "00100000" or IQ = "01000000" or IQ = "10000000") then
z <= '1';
c <= c + 1;
end if;
exit when c = N;
end if;
end loop L1;
end process;
end counter_arch;
```

Step 6 of 6

Hence, the required codes for the circuit realizations are done.

Step 1 of 2

74169 uses edge triggered *J-K* flip-flops which constitute to make it 4-bit binary counters. When Parallel Enable $\overline{PE}$ is LOW, the data on the P_0-P_3 inputs enters the flip-flops on the next rising edge of the clock. In order for counting to occur both $\overline{CEP}$ and $\overline{CET}$ must be LOW and $\overline{PE}$ must be HIGH; $U/\overline{D}$ input then determines the direction of counting. For up counting $U/\overline{D}=1$ and for down counting $U/\overline{D}=0$. The terminal count output is normally HIGH and goes LOW, provided that $\overline{CET}$ is LOW, when a counter reaches zero in the count-down mode or reaches 15 in the count-up mode. The $\overline{TC}$ output state is not a function of the Count Enable parallel ($\overline{CEP}$) input level.

Step 2 of 2

The counting sequence is 7, 6, 5, 4, 3, 2, 1, 0, 8, 9, 10, 11, 12, 13, 14, 15, 7, ...

So, the count is initially preloaded to 7 (binary 0111) and the $U/\overline{D}$ is made 0 for down counting. When the count reaches 0 (binary 0000) the parallel preloaded inputs becomes 8 (binary 1000) and now the up counting starts by making $U/\overline{D}=1$. Now, when the count reaches to 15 (binary 1111) the count starts from 7 (binary 0111) by loading the parallel inputs to 0111, and the desired count sequence is achieved.

The circuit connection to the 74x169 is shown in Figure 1.

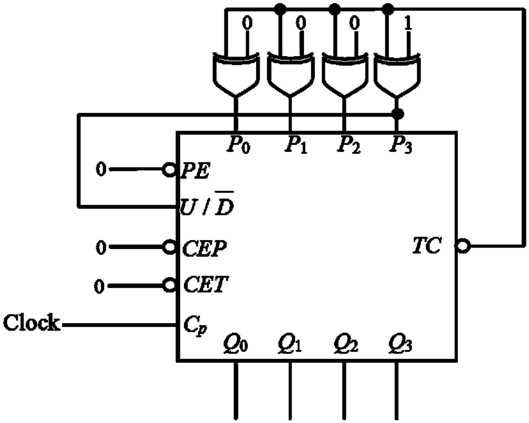


Figure 1

The parallel loading of 7 (binary 0111) and then 8 (binary 1000) is achieved by the XOR SSI IC package.

Step 1 of 1

Write an ABEL program for an *n*-bit counter with counting sequence, 7,6,5,4,3,2,1,0,8,9,10,11,12,13,14,15,7,...

```
module count

title '8-bit Counter'

" Input pins

CLK, LD_L, CLR_L, ENP, ENT pin;

A, B, C, D, E, F, G, H pin;

" Output pins

QA, QB, QC, QD, QE, QF, QG, QH pin istype 'reg';

RCO pin istype 'com';

" Set definitions

INPUT = [H, G, F, E, D, C, B, A];

COUNT = [QH, QG, QF, QE, QD, QC, QB, QA];

LD=!LD_L;

CLR=!CLR_L;

equations

COUNT.CLK = CLK;

when (COUNT<=7 & COUNT>0) then

[COUNT := !CLR & ( LD & INPUT # !LD & (ENT & ENP) & (COUNT-1) # !LD & !(ENT & ENP) & (COUNT);}

else when (COUNT==0) then

[COUNT == [0, 0, 0, 0, 1, 0, 0, 0];}

else when (COUNT>=8 & COUNT<15) then

[COUNT := !CLR & ( LD & INPUT # !LD & (ENT & ENP) & (COUNT+1) # !LD & !(ENT & ENP) & (COUNT);}

else when (COUNT==15) then

[COUNT == [0, 0, 0, 0, 0, 1, 1, 1];}

end count
```

During the clock event, the count is checked if it is less than or equal to 7 and also greater than 0. Then the count is decremented. If the count is 0, then the output is set to 8. Then the count is incremented until it reaches 15. When the count reaches 15, it is reset to 7.

Thus, the ABEL code for the counter is written.

Step 1 of 1 **8.048E**

Write a VHDL code for an *n*-bit counter with counting sequence, 7,6,5,4,3,2,1,0,8,9,10,11,12,13,14,15,7,...

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity counter is

generic (N: natural := 2);

port ( CLK: in STD_LOGIC; COUNT: in STD_LOGIC_VECTOR(3 downto 0); Q: out
STD_LOGIC_VECTOR(N-1 downto 0) );

end counter;

architecture counter_arch of counter is

signal Q1: STD_LOGIC_VECTOR(N-1 downto 0);

begin

process(CLK, COUNT)

begin

if (CLK'event and CLK = '1') then

if(COUNT<="0111" and COUNT>"0000") then

Q1<=COUNT-1;

end if;

if (COUNT="0000") then

Q1<=(3=> '1', others =>'0');

end if;

if(COUNT>="1000" and COUNT<"1111") then

Q1<=COUNT+1;

end if;

if (COUNT="1111") then

Q1<=(0=> '1', 1=>'1', 2=>'1', others =>'0');

end if;

end if;

Q<=Q1;

end process;

end counter_arch;
```

The size of the counter is determined by a constant *N* and can be changed by changing this constant.

During the clock event, the count is checked if it is less than or equal to 7 and also greater than 0. Then the count is decremented. If the count is 0, then the output is set to 8. Then the count is incremented until it reaches 15. When the count reaches 15, it is reset to 7.

Thus, the VHDL code for the counter is written.

Step 1 of 3

8.049E

The binary up/down counter for the elevator controller in a 20- storey building is shown in Figure 1. The ENT and ENP are pulled up to enable the GAL16V8R. From the state 12, “01011” and the state 14, “01101”, the input E is pulled down and A & D are pulled up respectively. B and C are decision bits and they are connected logically in order to omit the state 13, “01100”. QB is connected to C and B in direct and inverted form respectively. The load must be asserted at the state 12 and 14 to omit the state 13 and hence the QA, XOR of QB and QC and QD are given to the AND gate.

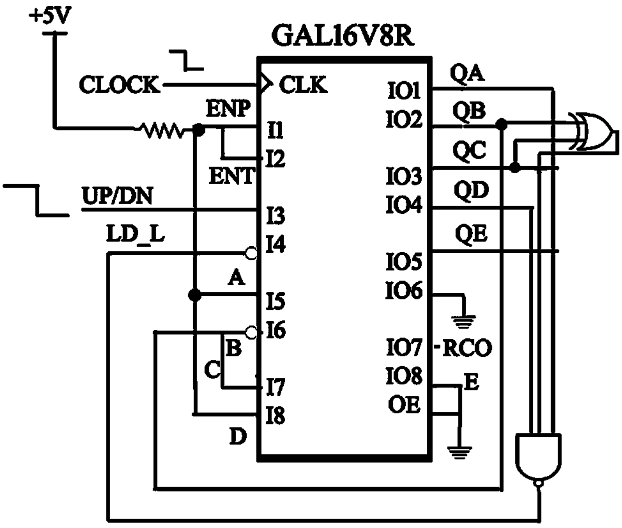


Figure 1

Step 2 of 3

CLOCK signal and UP/DN must be square waves and the time per of UP/DN is 20 times the time period of the CLOCK signal. The sequence starts with 00000 then the UP/DN is 1, LD_L is 1 and hence the next states are 00001, 00010, 00011, 00100, 00101, 00110, 00111, 01000, 01001, 01010 and 01011. At 01011 LD_L becomes 0 and the next state is 01101. Since the UP/DN is still 1 the next states are 01110, 01111, 10000, 10001, 10010, 10011 and 10100.

Now at 10100, UP/DN is 0 and hence the count decrements and the next states are 10011, 10010, 10001, 10000, 01111, 01110 and 01101. Now at 01101 the LD_L is 0 and the next state is 01011, since the UP/DN is still 0 the count decrements and the next states are 01010, 01001, 01000, 00111, 00110, 00101, 00100, 00011, 00010, 00001 and 0000.

Step 3 of 3

Write ABEL equations for the design.

```
COUNT.CLK = CLK;

when (UP/DN==1 & COUNT!=[0,1,1,0,0]) then

[COUNT := !CLR & ( LD & INPUT # !LD & (ENT & ENP) & (COUNT+1) # !LD & !(ENT & ENP) & (COUNT);}

else when (UP/DN==1 & COUNT==[0,1,1,0,0]) then {COUNT = [0,1,1,1,0];}

else when (UP/DN==0 & COUNT!=[0,1,1,1,0]) then

{COUNT := !CLR & ( LD & INPUT # !LD & (ENT & ENP) & (COUNT+1) # !LD & !(ENT & ENP) & (COUNT);}

else when (UP/DN==0 & COUNT==[0,1,1,1,0]) then {COUNT = [0,1,1,0,0];}
```

Thus, the logic circuit is drawn and the ABEL equations are written.

Step 1 of 1

Refer to Table 8-16 for the VHDL program for a 74x163-like 4-bit binary counter.

Modify the VHDL program by changing the type of ports D and Q to STD_LOGIC_VECTOR.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity V74x163 is
port ( CLK, CLR_L, LD_L, ENP, ENT: in STD_LOGIC;
D, Q: out STD_LOGIC_VECTOR(3 downto 0);
RCO: out STD_LOGIC );
end V74x163;
architecture V74x163_arch of V74x163 is
signal Q1: STD_LOGIC_VECTOR(3 downto 0);
begin
process(CLK, ENT, IQ)
begin
if (CLK= '1' and CLK'event) then
if CLR_L= '0' then IQ<= (others => '0');
elsif LD_L= '0' then IQ <= D;
elsif (ENT and ENP)= '1' then IQ <= IQ+1;
end if;
end if;
if (IQ=15) and (ENT= '1') then RCO <= '1';
else RCO <= '0';
end if;
Q<=IQ;
end process;
end V74x163_arch;
```

Thus, the VHDL code is written.

Step 1 of 1 **8.052E**

Refer to Figure 8-36 write a VHDL code for One bit-cell of a synchronous serial counter in structural style as follows:

```
library IEEE;

use IEEE.STD_LOGIC_1164.All;

use IEEE.NUMERIC_STD.all;

entity one_bit_cell is

port(

clk, LDNOCLR, Di, NOCLRORLD, CNTENP, CNTENi : in bit;

Qi, CNTENi1 : out bit

);

end one_bit_cell;

architecture Stru of one_bit_cell is

component AND2

port(x,y:in bit; z:out bit);

end component;

component VDFFQQN

port(d,clock:in bit;q,qbar:out bit);

end component;

component OR2

port(da,db:in bit; dz:out bit);

end component;

component INV

port(dx:in bit;dy:out bit);

end component;

component NOR2

port(a,b:in bit;c:out bit);

end component;

component XNOR2

port(d,e:in bit;f:out bit);

end component;

signal CEi,CNTEN_Li,CDi,Q_Li,CDATi,LDATi,DINi:bit;

begin

A1:AND2 port map(LDNOCLR,Di,LDATi);

A2:AND2 port map(CNTENP,CNTENi,CEi);

A3:AND2 port map(NOCLRORLD,CDi,CDATi);

XN1:XNOR2 port map(CEi,Q_Li,CDi);

O1:OR2 port map(LDATi,CDATi,DINi);

D1:VDFFQQN port map(DINi,clk,Qi,Q_Li);

I1:INV port map(CNTENi,CNTEN_Li);

N1:NOR2 port map(Q_Li,CNTEN_Li);

end Stru;
```

The program has been executed in Xlinx 13.2.

Step 1 of 1 **8.053E**

Write a VHDL program for synchronous serial counter and provide counter size changing facility at the user end using generic statement.

```
library IEEE;

use IEEE.std_logic_1164.all;

use IEEE.std_logic_arith.all;

use IEEE.std_logic_unsigned.all;

entity v74x163s is

generic(n : integer := 8); --added generic

port (

CLK,CLR_L,LD_L,ENP,ENT : in STD_LOGIC;

D : in STD_LOGIC_VECTOR (n-1 downto 0);

Q : out STD_LOGIC_VECTOR (n-1 downto 0);

RCO : out STD_LOGIC

);

end v74x163s;

architecture v74x163_arch of v74x163s is

component synsercell is

port (

CLK, LDNOCLR, Di, NOCLRORLD, CNTENP, CNTEN : in STD_LOGIC;

Q,CNTEN0 : out STD_LOGIC

);

end component;

signal LDNOCLR,NOCLRORLD : STD_LOGIC;

signal SCNTEN : STD_LOGIC_VECTOR (n downto 0);

begin

LDNOCLR <= (not LD_L) and CLR_L;

NOCLRORLD <= LD_L and CLR_L;

SCNTEN(0) <= ENT;

RCO <= SCNTEN(n);

gi: for i in 0 to n-1 generate

U1: synsercell port map (CLK, LDNOCLR, NOCLRORLD, ENP, D(i), SCNTEN(i), SCNTEN(i+1), Q(i));

end generate;

end v74x163_arch;
```

Step 1 of 2 8.055E

The 8 bit self-correcting ring counter with the states 11111110, 11111101, 11111011, 11110111, 11101111, 11011111, 10111111 and 0111111 can be designed using two 74x194 packages. In this design the two packages share the same clock. S_0 , the $\overline{CLR}$ and S_1 are pulled up in both the packages. Now if S_0 is zero, the package works as a left shift register similarly if S_0 is 1, the package is reset by loading the input values. For the 8 bit ring counter the Q_D of the second package is feedback to the *DSL* of the first package and the Q_D of the first package is feedback to the *DSL* of the second package. The 8 bit self-correcting ring counter is as shown in the Figure 1.

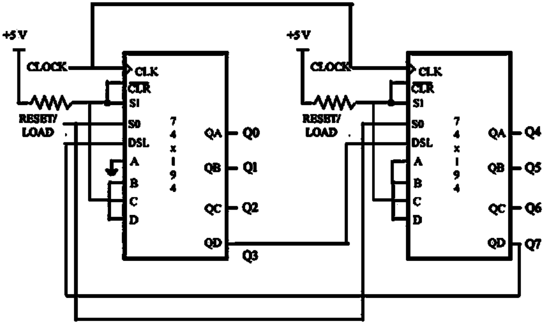


Figure 1

Step 2 of 2

The counter starts with the state 11111110. When the clock has an event and S_0 = 0 then the bit 0 is shifted left and the state becomes 11111101. And with the CLK and S_0 = 0 the consecutive states are 11111011, 11110111, 11101111, 11011111, 10111111 and 01111111.

Step 1 of 1

VHDL program for an 8-bit self-correcting ring counter as follows:

```
library IEEE;
use IEEE.STD_LOGIC_1164.All;
use IEEE.NUMERIC_STD.all;
entity Vshftreg is
port(
clk, RES, EN : in STD_LOGIC;
Q : out STD_LOGIC_VECTOR( 7 downto 0)
);
end Vshftreg;
architecture Vshftreg_arch of Vshftreg is
signal s_count : STD_LOGIC_VECTOR(7 downto 0):=(others => '0');
begin
Q <= s_count;
process(clk)
begin
if(RES='1')then Q <= "11111110";
end if;
if (EN = '1') then
if (clk'event and clk='1') then
s_count(1) <= s_count(0);
s_count(2) <= s_count(1);
s_count(3) <= s_count(2);
s_count(4) <= s_count(3);
s_count(5) <= s_count(4);
s_count(6) <= s_count(5);
s_count(7) <= s_count(6);
s_count(0) <= s_count(7);
end if;
end if;
end process;
end Vshftreg_arch;
```

Resultant VHDL code has been verified using Xlinx 13.2.

Step 1 of 4

Before the first clock the Johnson counter is cleared. At the clock the $\overline{Q_D}$ at the last stage is feedback to input at the first stage. The four bit Johnson counter circulates four zeroes and four ones to form all the eight states. Thus using four D flip-flops, eight NOR gates construct the 4 bit Johnson counter which decode all eight states as shown in Table 1.

Table 1

Bits Considered	G
Q_A and Q_D	G_0
$\overline{Q_A}$ and Q_B	G_1
$\overline{Q_B}$ and Q_C	G_2
$\overline{Q_C}$ and Q_D	G_3
$\overline{Q_D}$ and $\overline{Q_A}$	G_4
Q_A and $\overline{Q_B}$	G_5
Q_B and $\overline{Q_C}$	G_6
Q_C and $\overline{Q_D}$	G_7

Step 2 of 4

4 bit Johnson counter which decodes all the eight states is as shown in Figure 1.

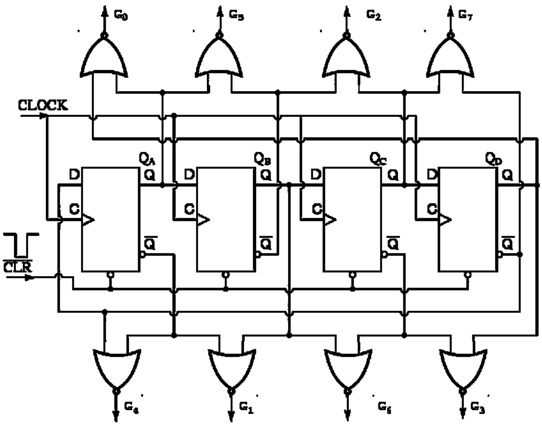


Figure 1

Step 3 of 4

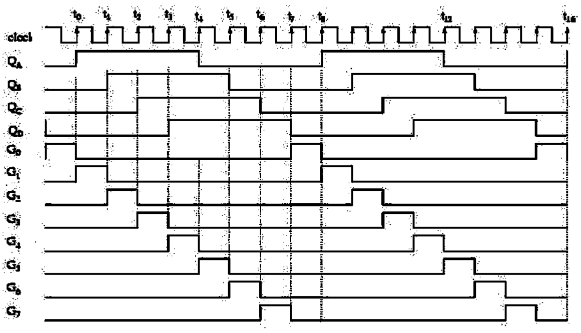
Write the truth table for Figure 1 is as shown in Table 2.

Table 2

$Q_AQ_BQ_CQ_D$	G
0000	G_0
1000	G_1
1100	G_2
1110	G_3
1111	G_4
0111	G_5
0011	G_6
0001	G_7

Step 4 of 4

Draw the waveforms for the Figure 1 is as shown in Figure 2.



Therefore, desired 4-bit Johnson counter decoding all eight states.

Step 1 of 1

VHDL program for an 8-bit Johnson counter as follows:

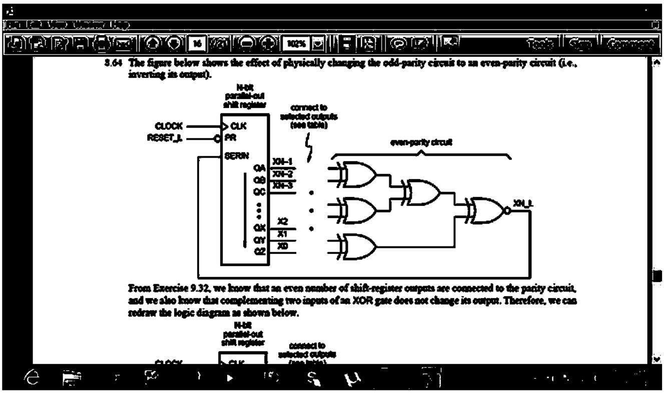
```
library IEEE;
use IEEE.STD_LOGIC_1164.All;
use IEEE.NUMERIC_STD.all;
entity johnson_counter is
port(
clk, reset_n, enable : in STD_LOGIC;
counter : out STD_LOGIC_VECTOR( 7 downto 0)
);
end johnson_counter;
architecture JCArch of johnson_counter is
signal s_count : STD_LOGIC_VECTOR(7 downto 0):=(others => '0');
begin
counter <= s_count;
process(clk)
begin
if( rising_edge(clk) )then
if(reset_n = '0') then
s_count <= (others => '0');
elsif (enable = '1') then
s_count(1) <= s_count(0);
s_count(2) <= s_count(1);
s_count(3) <= s_count(2);
s_count(4) <= s_count(3);
s_count(5) <= s_count(4);
s_count(6) <= s_count(5);
s_count(7) <= s_count(6);
s_count(0) <= not s_count(7);
end if;
end if;
end process;
end JCArch;
```

Resultant VHDL code has been verified using Xlinx 13.2.

Step 1 of 3

Refer to the Figure 8.51 from the text book.

Redraw the linear feedback shift-register counter with odd parity circuit in Figure 8-51 to change the odd parity circuit to even parity circuit as shown in Figure 1.


8.64 The figure below shows the effect of physically changing the odd-parity circuit to an even-parity circuit (i.e., inverting its output).

From Exercise 9.32, we know that an even number of shift-register outputs are connected to the parity circuit, and we also know that complementing two inputs of an XOR gate does not change its output. Therefore, we can redraw the logic diagram as shown below.

Figure 1

Step 2 of 3

Complementing the even number of inputs of the XOR gate does not change the output. Redraw the Figure 1 as shown in Figure 2.

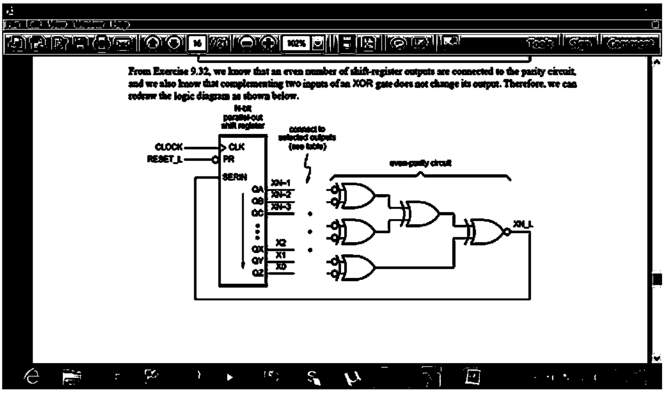

From Exercise 9.32, we know that an even number of shift-register outputs are connected to the parity circuit, and we also know that complementing two inputs of an XOR gate does not change its output. Therefore, we can redraw the logic diagram as shown below.

Figure 2

Step 3 of 3

Write the expression for the feedback equation of the linear feedback shift-register counter for selected values of n as follows:

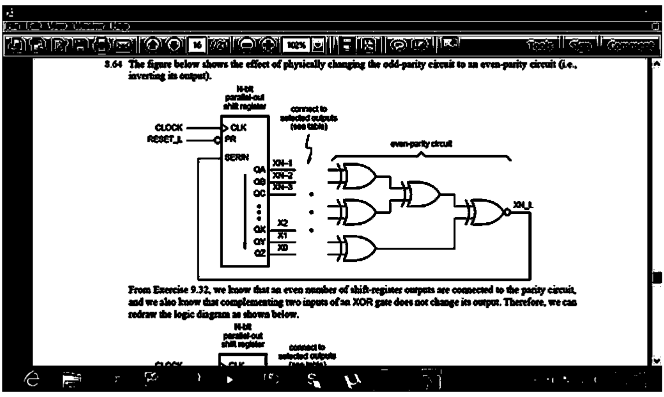
n	Feedback Equation
2	$X2 = X1 \oplus X0$
4	$X2 = X1 \oplus X0$
6	$X2 = X1 \oplus X0$
8	$X8 = X4 \oplus X3 \oplus X2 \oplus X0$
12	$X12 = X6 \oplus X4 \oplus X1 \oplus X0$
16	$X16 = X5 \oplus X4 \oplus X3 \oplus X0$

Therefore, from the expression for the feedback equation of the linear feedback shift-register LFSR counter for selected values of n the bit $X0$ appears on the right side of all the equations. Hence, proved that the bit $X0$ appears on the right side of all the equations of LFSR counter.

Step 1 of 4

Refer to the Figure 8.51 from the text book.

Redraw the linear feedback shift-register counter with odd parity circuit in Figure 8-51 to change the odd parity circuit to even parity circuit as shown in Figure 1.



8.64 The figure below shows the effect of physically changing the odd-parity circuit to an even-parity circuit (i.e., inverting its output).

From Exercise 9.32, we know that an even number of shift-register outputs are connected to the parity circuit, and we also know that complementing two inputs of an XOR gate does not change its output. Therefore, we can redraw the logic diagram as shown below.

Figure 1

Step 2 of 4

Write the expression for the feedback equation of the linear feedback shift-register counter for selected values of n as follows:

n	Feedback Equation
2	$X2 = X1 \oplus X0$
4	$X2 = X1 \oplus X0$
6	$X2 = X1 \oplus X0$
8	$X8 = X4 \oplus X3 \oplus X2 \oplus X0$
12	$X12 = X6 \oplus X4 \oplus X1 \oplus X0$
16	$X16 = X5 \oplus X4 \oplus X3 \oplus X0$

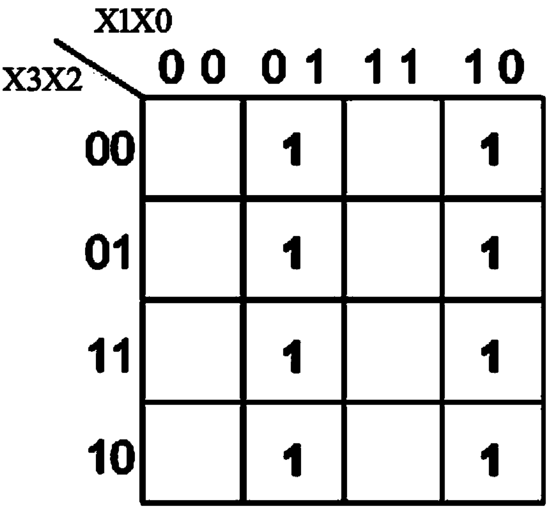
From the Table-26 the sequence of a 4-bit LFSR counter that produces a maximum length sequence is as shown in Table 1.

Table 1

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

Step 3 of 4

Applying K-map to find the equation for the feedback as shown in Figure 1,



X3X2 \ X1X0	00	01	11	10
00		1		1
01		1		1
11		1		1
10		1		1

Figure 1

Step 4 of 4

Applying K-map equation for zeroes as shown in Figure 1,

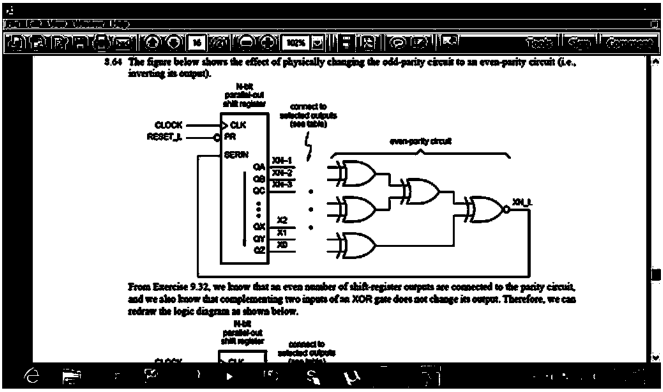
$$X4 = \overline{X1X0 + \overline{X1}\,\overline{X0}}$$
$$X4 = X1 \odot X0$$

Therefore, the expression for the feedback equation of the linear feedback shift-register counter for maximum length sequence is $\boxed{X4 = X1 \odot X0}$.

Step 1 of 2 **8.064E**

Refer to the Figure 8.52 from the text book.

Redraw the linear feedback shift-register counter with odd parity circuit in Figure 8-51 to change the odd parity circuit to even parity circuit as shown in Figure 1.



8.64 The figure below shows the effect of physically changing the odd-parity circuit to an even-parity circuit (i.e., inverting its output).

From Exercise 9.32, we know that an even number of shift-register outputs are connected to the parity circuit, and we also know that complementing two inputs of an XOR gate does not change its output. Therefore, we can redraw the logic diagram as shown below.

Figure 1

Step 2 of 2

Write the expression for the feedback equation of the linear feedback shift-register counter for selected values of n as follows:

n	Feedback Equation
2	$X2 = X1 \oplus X0$
4	$X2 = X1 \oplus X0$
6	$X2 = X1 \oplus X0$
8	$X8 = X4 \oplus X3 \oplus X2 \oplus X0$
12	$X12 = X6 \oplus X4 \oplus X1 \oplus X0$
16	$X16 = X5 \oplus X4 \oplus X3 \oplus X0$

From the Figure 8-52 and Table 8-27 an LFSR counter can be modified to have all the 2^n states.

The resulting sequence also contains 000 which is omitted in the original sequence. The state 0 in a n-bit LSFR can be attained by adding a $n-1$ input NOR gate and a XOR gate with 2 inputs one from the NOR gate and the other from the feedback equation realized by XOR gates as shown in Figure 8-52 using Table 8-26.

State 0 is skipped after state 1 and hence the NOR is applied to all the inputs except the LSB bit of the output and is XORED with feedback equation to bring the state 0 and helps the LFSR to attain all the 2^n states.

Hence, the required realization of 2^n states is done by using a XOR gate with 2 inputs one from the NOR gate.

Step 1 of 4

Write a VHDL code for an 8-bit LFSR counter.

```
library IEEE;
use IEEE.STD_LOGIC_1164.All;
use IEEE.NUMERIC_STD.all;
entity LFSR is
port(
clk, RES, EN : in STD_LOGIC;
Q : out STD_LOGIC_VECTOR( 7 downto 0)
);
end LFSR;
architecture LFSR_arch of LFSR is
signal s_count : STD_LOGIC_VECTOR(7 downto 0):=(7 => '1', others => '0');
signal poly : STD_LOGIC;
begin
Q <= s_count;
process(clk)
begin
if(RES='1')then Q <= "10000000";
end if;
if (EN = '1') then
if (clk'event and clk='1') then
poly <= s_count(0) xor s_count(2) xor s_count(3) xor s_count(4);
s_count(1) <= s_count(0);
s_count(2) <= s_count(1);
s_count(3) <= s_count(2);
s_count(4) <= s_count(3);
s_count(5) <= s_count(4);
s_count(6) <= s_count(5);
s_count(7) <= s_count(6);
s_count(0) <= poly;
end if;
end if;
end process;
end LFSR_arch;
```

Step 2 of 4

Write a VHDL test bench for the program of the 8-bit LFSR counter.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY LFST_tb IS
END LFST_tb;
ARCHITECTURE behavior OF LFST_tb IS
-- Component Declaration for the Unit Under Test (UUT)
COMPONENT LFSR
PORT(
clk : IN std_logic;
RES : IN std_logic;
EN : IN std_logic;
Q : OUT std_logic_vector(7 downto 0)
);
END COMPONENT;
--Inputs
signal clk_tb : std_logic := '0';
signal RES_tb : std_logic := '0';
signal EN_tb : std_logic := '0';
--Outputs
signal Q_tb : std_logic_vector(7 downto 0);
-- Clock period definitions
constant clk_period : time := 10 ns;
BEGIN
-- Instantiate the Unit Under Test (UUT)
uut: LFSR PORT MAP (
clk => clk_tb,
RES => RES_tb,
EN => EN_tb,
Q => Q_tb
);
-- Clock process definitions
clk_process :process
begin
clk_tb <= '0';
wait for clk_period/2;
clk_tb <= '1';
wait for clk_period/2;
end process;
```

Step 3 of 4

```
-- Stimulus process
stim_proc: process
begin
RES_tb <= '1';
clk_tb <= '1';
wait for 100 ns;
RES_tb <= '0';
EN_tb <= '1';
clk_tb <= '1';
wait for 100 ns;
RES_tb <= '0';
EN_tb <= '1';
clk_tb <= '1';
wait for 100 ns;
wait;
end process;
END;
```

Step 4 of 4

Draw the waveforms result from the test bench:

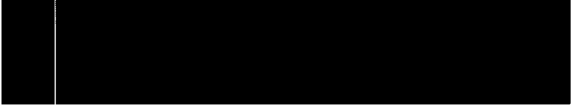


Figure 1

8.067E

Step 1 of 3

The polynomial that yields a maximum length sequence for a 8-bit LFSR is $X^8 + X^6 + X^5 + X^4 + 1$ where are [8, 6, 5, 4] are the positions of the taps. The bit positions that affect the next state are called the taps.

Step 2 of 3

```
library IEEE;

use IEEE.std_logic_1164.all;

use IEEE.NUMERIC_STD.all;

entity counter is

port (CLK, CLR_L, LD_L, ENP, ENT: in STD_LOGIC; N: in unsigned(7 downto 0); Z: out unsigned(7
downto 0));

end counter;

architecture counter_arch of counter is

signal a: STD_LOGIC;

Signal IQ: unsigned(7 downto 0);

begin

process(CLK, ENT, N)

begin

if(CLK'event and CLK = '1') then

if CLR_L = ’0’ then IQ <= (others => '0');

elsif LD_L = '0' then IQ <= N;

end if;

if (ENP = '1' and ENT = '1') then

a <= IQ(0) xor IQ(2) xor IQ(3) xor IQ(4);

Z <= a & IQ(7 downto 1);

end if;

end if;

end process;

end counter_arch;
```

Step 3 of 3

Hence, the required VHDL code is described.

Step 1 of 2

8.068E

The following VHDL code is used for generating the polynomial of form $X^8+X^6+X^5+X^4+1$.

```
library IEEE;

use IEEE.std_logic_1164.all;

use IEEE.NUMERIC_STD.all;

entity counter is

port (CLK, CLR_L, LD_L, ENP, ENT: in STD_LOGIC; N: in unsigned(7 downto 0); Z: out unsigned(7
downto 0));

end counter;

architecture counter_arch of counter is

signal a,b,c: STD_LOGIC;

Signal IQ: unsigned(7 downto 0);

begin

process(CLK, ENT, N)

begin

if(CLK'event and CLK = '1') then

if CLR_L = '0' then IQ <= (others => '0');

elsif LD_L = '0' then IQ <= N;

end if;

if (ENP = '1' and ENT = '1') then

a <= IQ(0) xor IQ(2) xor IQ(3) xor IQ(4);

b <= ((((((IQ(7) nor IQ(6)) nor IQ(5)) nor IQ(4)) nor IQ(3)) nor IQ(2)) nor IQ(1));

c <= a xor b;

Z <= c & IQ(7 downto 1);

end if;

end if;

end process;

end counter_arch;
```

Step 2 of 2

After execution of this code, the required polynomial generated is $\boxed{X^8+X^6+X^5+X^4+1}$.

Step 1 of 3

VHDL program for the iterative circuit that checks the parity of 16 bit data word with single parity bit is as follows.

```
library IEEE;

use IEEE.std_logic_1164.all;

entity Iter_Mod is

port (

carry_in, pri_in: in STD_LOGIC;

carry_out: out STD_LOGIC

);

end Iter_Mod;

architecture PC of Iter_Mod is

begin

carry_out <= carry_in xor pri_in;

end PC;
```

Step 2 of 3

```
library IEEE;

use IEEE.std_logic_1164.all;

entity P_16bit is

port (

d_word: in STD_LOGIC_VECTOR (15 downto 0);

parity: out STD_LOGIC

);

end P_16bit;

architecture P_16bit_arch of P_16bit is

component Iter_Mod

port (

carry_in, pri_in: in STD_LOGIC;

carry_out: out STD_LOGIC

);

end component;

signal carry: STD_LOGIC_VECTOR (15 downto 0);

signal cin0: STD_LOGIC;

begin

cin0 <= '0';

P0: Iter_Mod port map (cin0, d_word(0), carry(0));

P1: Iter_Mod port map (carry(0), d_word(1), carry(1));

P2: Iter_Mod port map (carry(1), d_word(2), carry(2));

P3: Iter_Mod port map (carry(2), d_word(3), carry(3));

P4: Iter_Mod port map (carry(3), d_word(4), carry(4));

P5: Iter_Mod port map (carry(4), d_word(5), carry(5));

P6: Iter_Mod port map (carry(5), d_word(6), carry(6));

P7: Iter_Mod port map (carry(6), d_word(7), carry(7));

P8: Iter_Mod port map (carry(7), d_word(8), carry(8));

P9: Iter_Mod port map (carry(8), d_word(9), carry(9));

P10: Iter_Mod port map (carry(9), d_word(10), carry(10));

P11: Iter_Mod port map (carry(10), d_word(11), carry(11));

P12: Iter_Mod port map (carry(11), d_word(12), carry(12));

P13: Iter_Mod port map (carry(12), d_word(13), carry(13));

P14: Iter_Mod port map (carry(13), d_word(14), carry(14));

P15: Iter_Mod port map (carry(14), d_word(15), carry(15));

parity <= carry(15); --parity = 0 is even parity and 1 if odd parity

end P_16bit_arch;
```

Step 3 of 3

The code is cascaded 16 times with Iter_Mod and connects together.

For checking the parity of a 16 bit word the order of transmission doesn't matter.

Step 1 of 1

Refer to Table 8-28 from the textbook.

Use a 22V10 to modify the program to provide an asynchronous clear input.

```
module Z74x194

title '4-bit Universal Shift Register'

Z74x194 device '22V10';

" Input and output pins

CLK, RIN, A, B, C, D, LIN pin 1, 2, 3, 4, 5, 6, 7;

S1, S0, CLR pin 8, 9, 12;

QA, QB, QC, QD pin 19, 18, 17, 16 istype 'reg';

" Set definitions

INPUT = [A, B, C, D];

LEFTIN = [QB, QC, QD, LIN];

RIGHTIN = [RIN, QA, QB, QC];

OUT = [QA, QB, QC, QD];

CTRL = [S1, S0];

HOLD = (CTRL == [0, 0]);

RIGHT = (CTRL == [0, 1]);

LEFT = (CTRL == [1, 0]);

LOAD = (CTRL == [1, 1]);

equations

OUT.CLK=CLK;

[QA, QB, QC, QD].AR = !CLR;

OUT := !CLR & ( HOLD & OUT # RIGHT & RIGHTIN # LEFT & LEFTIN # LOAD & INPUT);

end Z74x194;
```

Step 1 of 1

8.072E

Refer to Tables 8-31 and 8-32 from the textbook.

In all the situations the ABEL programs in the Tables 8-31 and 8-32 gives same operational result. But in Table 8-32, TSTATE is assigned a value in the equations of the program, as well as being used in the state_diagram section. This is done to ensure that the program goes to SRESET from any undefined state.

Step 1 of 1

Refer to the Table 8-31 from the text book.

Write the following ABEL program in which the phases are always at least two ticks long, even if the RESTART is asserted at the beginning of the phase and RESET should still take the effect immediately.

```
module TIMEGEN6

title 'Six-phase Master Timing Generator'

"Input and Output pins

MCLK, RESET, RUN, RESTART pin;

T1, P1_L, P2_L, P3_L, P4_L, P5_L, P6_L pin istype 'reg';

"State definitions

PHASES = [P1_L, P2_L, P3_L, P4_L, P5_L, P6_L];

NEXTPH = [P6_L, P1_L, P2_L, P3_L, P4_L, P5_L];

SRESET = [1, 1, 1, 1, 1, 1]

P1 = [0, 0, 1, 1, 1, 1];

equations

T1.CLK = MCLK; PHASES.CLK = MCLK;

when RESET then {T1 := 1; PHASES := SRESET;}

else when (PHASES == SRESET) # RESTART then {T1 := 1; PHASES := P1;}

else when RUN & T1 then { T1 := 0; PHASES := PHASES;}

else when RUN & !T1 then { T1 := 1; PHASES := NEXTPH;}

else { T1 := T1; PHASES := PHASES;}

end TIMEGEN6
```

Step 1 of 1

Refer to the ABEL program from Table 8-32 in the textbook.

Consider that the phases are always at least 2 clock ticks long, even if RESTART asserts at the beginning of phase.

Modify the program in Table 8-32:

```
module TIMEGEN6

title 'Six-phase Master Timing Generator'

" Input and Output pins

MCLK, RESET, RUN, RESTART pin;

T1, P1_L, P2_L, P3_L, P4_L, P5_L, P6_L pin istype 'reg';

" State definitions

TSTATE = [T1, P1_L, P2_L, P3_L, P4_L, P5_L, P6_L];

SRESET = [1, 1, 1, 1, 1, 1,1]

P1F = [1, 0, 0, 1, 1, 1, 1];

P1S = [0, 0, 1, 1, 1, 1, 1];

P2F = [1, 1, 0, 0, 1, 1, 1];

P2S = [1, 0, 0, 1, 1, 1, 1];

P3F = [1, 1, 1, 0, 0, 1, 1];

P3S = [1, 1, 0, 0, 1, 1, 1];

P4F = [1, 1, 1, 1, 0, 0, 1];

P4S = [1, 1, 1, 0, 0, 1, 1];

P5F = [1, 1, 1, 1, 1, 0, 0];

P5S = [1, 1, 1, 1, 0, 0, 1];

P6F = [0, 1, 1, 1, 1, 1, 0];

P6S = [1, 1, 1, 1, 1, 0, 0];

equations

TSTATE.CLK = MCLK;

When RESET then TSTATE :=SRESET;

state_diagram TSTATE

state SRESET: if RESET then SRESET else P1F;

state P1F: if RESET then SRESET else if RESTART then P1F else if RUN then P1S else P1F;

state P1S: if RESET then SRESET else if RESTART then P1F else if RUN then P2F else P1S;

state P2F: if RESET then SRESET else if RESTART then P1F else if RUN then P2S else P2F;

state P2S: if RESET then SRESET else if RESTART then P1F else if RUN then P3F else P2S;

state P3F: if RESET then SRESET else if RESTART then P1F else if RUN then P3S else P3F;

state P3S: if RESET then SRESET else if RESTART then P1F else if RUN then P4F else P3S;

state P4F: if RESET then SRESET else if RESTART then P1F else if RUN then P4S else P4F;

state P4S: if RESET then SRESET else if RESTART then P1F else if RUN then P5F else P4S;

state P5F: if RESET then SRESET else if RESTART then P1F else if RUN then P5S else P5F;

state P5S: if RESET then SRESET else if RESTART then P1F else if RUN then P6F else P5S;

state P6F: if RESET then SRESET else if RESTART then P1F else if RUN then P6S else P6F;

state P6S: if RESET then SRESET else if RESTART then P1F else if RUN then P1F else P6S;

end TIMEGEN6
```

Thus, the program in Table 8-32 is modified for the specified requirement.

Step 1 of 5

The VHDL program to have the phases always at least two ticks long, even if the RESTART is asserted at the beginning of the phase is as follows:

```
library IEEE;
use IEEE.std_logic_1164.all;
entity Vtimegn6 is
port (
MCLK,RESET,RUN,RESTART: in STD_LOGIC;
P_L: out STD_LOGIC_VECTOR(1 to 6)
);
end Vtimegn6;
architecture Vtimegn6_arch of Vtimegn6 is
signal IP: STD_LOGIC_VECTOR (1 to 6);
signal T1: STD_LOGIC;
begin
process(MCLK,IP)
begin
if(MCLK'event and MCLK='1') then
if(RESET='1') then
T1 <= '1';IP <= ('0','0','0','0','0','0');
elsif((IP=('0','0','0','0','0','0')) or (RESTART='1')) then
T1<='1'; IP<=('1','1','0','0','0','0');
elsif(RUN='1') then
T1<= not T1;
if(T1='0') then
IP <= IP(6) & IP(1 to 5);
end if;
end if;
end if;
P_L<=not IP;
end process;
end Vtimegn6_arch;
```

Step 2 of 5

The test bench generation program is as follows:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;
ENTITY fegfdsg IS
END fegfdsg;
ARCHITECTURE behavior OF fegfdsg IS
-- Component Declaration for the Unit Under Test (UUT)
COMPONENT Vtimegn6
PORT(
MCLK : IN std_logic;
RESET : IN std_logic;
RUN : IN std_logic;
RESTART : IN std_logic;
P_L : OUT std_logic_vector(1 to 6)
);
END COMPONENT;
--Inputs
signal MCLK_tb : std_logic := '0';
signal RESET_tb : std_logic := '0';
signal RUN_tb : std_logic := '0';
signal RESTART_tb : std_logic := '0';
--Outputs
signal P_L_tb : std_logic_vector(1 to 6);
-- Clock period definitions
constant MCLK_period : time := 100 ns;
BEGIN
-- Instantiate the Unit Under Test (UUT)
uut: Vtimegn6 PORT MAP (
MCLK => MCLK_tb,
RESET => RESET_tb,
RUN => RUN_tb,
RESTART => RESTART_tb,
P_L => P_L_tb
);
-- Clock process definitions
MCLK_process :process
begin
MCLK_tb <= '0';
wait for MCLK_period/2;
MCLK_tb <= '1';
wait for MCLK_period/2;
end process;
-- Stimulus process
```

Step 3 of 5

```
stim_proc: process
begin
RESTART_tb <= '1';
wait for 100 ns;
wait for MCLK_period*10;
RUN_tb <= '1';
wait for 100 ns;
wait for MCLK_period*10;
-- insert stimulus here
wait;
end process;
END;
```

Step 4 of 5

The test bench waveform is obtained as shown in Figure 1.

Figure 1

Step 5 of 5

Thus, the VHDL program is modified and the test bench generation program is written.

Step 1 of 1 **8.076E**

Refer to the ABEL program for a timing generator from Table 8-31 in the textbook.

Consider that this timing generator is used to control a dynamic memory system.

Consider also that the memory contents corrupt, if the timing generator resets without completing all six phases.

Modify the program in Table 8-31 to overcome this situation:

```
module TIMING6

title 'Six-phase Master Timing Generator'

" Input and Output pins

MCLK, RESET, RUN, RESTART pin;

T1, P1_L, P2_L, P3_L, P4_L, P5_L, P6_L pin istype 'reg';

"State definitions

PHASES = [P1_L, P2_L, P3_L, P4_L, P5_L, P6_L];

NEXTPH = [P6_L, P1_L, P2_L, P3_L, P4_L, P5_L];

equations

T1.CLK = MCLK; PHASES.CLK = MCLK;

when RESET and PHASES = [1, 1, 1, 1, 1, 0] then [T1 := 1; PHASES := SRESET;}

else when (PHASES = SRESET) # RESTART then {T1 := 1; PHASES := P1;}

else when RUN & T1 then {T1 := 0; PHASES := PHASES;}

else when RUN & !T1 then {T1 := 1; PHASES := NEXTPH;}

else {T1 := T1; PHASES := PHASES;}

end TIMING6
```

Step 1 of 2 8.077E

Refer to Table 8-32 from the textbook for an ABEL program for the waveform generator.

Modify the program to generate the timing waveforms of Figure 8-55 from the textbook such that the phase output is 0 only during the second tick of each phase.

```
module TIMEGEN6

title 'Six-phase Master Timing Generator'

" Input and Output pins

MCLK, RESET, RUN, RESTART pin;

T1, P1_L, P2_L, P3_L, P4_L, P5_L, P6_L pin istype 'reg';

" State definitions

TSTATE = [T1, P1_L, P2_L, P3_L, P4_L, P5_L, P6_L];

SRESET = [1, 1, 1, 1, 1, 1, 1];

P1F = [0, 1, 0, 0, 0, 0, 0];

P1S = [1, 1, 0, 0, 0, 0, 0];

P2F = [0, 0, 1, 0, 0, 0, 0];

P2S = [1, 0, 1, 0, 0, 0, 0];

P3F = [0, 0, 0, 1, 0, 0, 0];

P3S = [1, 0, 0, 1, 0, 0, 0];

P4F = [0, 0, 0, 0, 1, 0, 0];

P4S = [1, 0, 0, 0, 1, 0, 0];

P5F = [0, 0, 0, 0, 0, 1, 0];

P5S = [1, 0, 0, 0, 0, 1, 0];

P6F = [0, 0, 0, 0, 0, 0, 1];

P6S = [1, 0, 0, 0, 0, 0, 1];

"Now the phase outputs are active high signals

equations

TSTATE.CLK = MCLK;

When RESET then TSTATE:=SRESET;

state_diagram TSTATE

state SRESET: if RESET then SRESET else P1F;

state P1F: if RESET then SRESET else if RESTART then P1F else if RUN then P1S else P1F;

state P1S: if RESET then SRESET else if RESTART then P1F else if RUN then P2F else P1S;

state P2F: if RESET then SRESET else if RESTART then P1F else if RUN then P2S else P2F;

state P2S: if RESET then SRESET else if RESTART then P1F else if RUN then P3F else P2S;

state P3F: if RESET then SRESET else if RESTART then P1F else if RUN then P3S else P3F;

state P3S: if RESET then SRESET else if RESTART then P1F else if RUN then P4F else P3S;

state P4F: if RESET then SRESET else if RESTART then P1F else if RUN then P4S else P4F;

state P4S: if RESET then SRESET else if RESTART then P1F else if RUN then P5F else P4S;

state P5F: if RESET then SRESET else if RESTART then P1F else if RUN then P5S else P5F;

state P5S: if RESET then SRESET else if RESTART then P1F else if RUN then P6F else P5S;

state P6F: if RESET then SRESET else if RESTART then P1F else if RUN then P6S else P6F;

state P6S: if RESET then SRESET else if RESTART then P1F else if RUN then P1F else P6S;

end TIMEGEN6
```

Step 2 of 2

Changing phase output of each of states P1F, P2F,…, P6F to 1 instead of 0, so that the phase output is 0 only during the second of each phase is a good approach where the power consumption is less. But random glitches occur for a very short time that causes unpredictable values in the data bus. Hence, this approach can be used in the glitch free zones.

Step 1 of 2 8.078E

Refer Tables 8-34 and 8-35 from the textbook. The output waveforms produced by the ABEL programs are not similar though both the programs account for a one-tick decoding delay between each transition. The states are different for the both. The state difference is shown in table 1.

Table 1

States	
Table 8-34	**Table 8-35**
P1_L, P2_L, P3_L, P4_L, P5_L, P6_L	**P1_L, P2_L, P3_L, P4_L, P5_L, P6_L**
111111	011111
011111	101111
101111	110111
110111	111011
111011	111101
111101	111110
111110	111111

Step 2 of 2

Modify the ABEL program in Table 8-35 to match the behavior of program in Table 8-34

```
module TIMEG12A

title 'Counter-based Six-phase Master Timing Generator'

" Input and Output pins

MCLK, RESET, RUN, RESTART pin;

P1_L, P2_L, P3_L, P4_L, P5_L, P6_L pin istype 'reg';

CNT3, CNT2, CNT1, CNT0 pin istype 'reg';

" State definitions

P_L = [P1_L, P2_L, P3_L, P4_L, P5_L, P6_L];

CNT = [CNT3, CNT2, CNT1, CNT0];

equations

CNT.CLK = MCLK;

P_L.CLK = MCLK;

when RESET then CNT := 15

else when RESTART then CNT := 0

else when (RUN & (CNT < 12)) then CNT := CNT+1

else when RUN then CNT := 0

else CNT := CNT;

P1_L := !(CNT == 1);

P2_L := !(CNT == 3);

P3_L := !(CNT == 5);

P4_L := !(CNT == 7);

P5_L := !(CNT == 9);

P6_L := !(CNT == 11);

end TIMEG12A
```

Hence, the modified program of Table 8-35 matches the program in Table 8-34.

Step 1 of 2

Refer to Table 8-31 from the textbook for the ABEL ring-counter implementation. The design is not the self-synchronizing.

If the outputs [P1_L, P2_L, P3_L, P4_L, P5_L, P6_L] are initially 0 and if the RUN input is asserted without ever asserting the RESET and RESTART then the all the states will stick to [0, 0, 0, 0, 0, 0]. It is not possible to generate the six-phases as there is no chance of assigning P1 to the PHASES. To make the design self-synchronizing, assert RUN after asserting RESTART so that the initial state P1 is assigned to the PHASES.

Step 2 of 2

Modify the program in Table 8-31 from the textbook to make the design self-synchronizing.

```
module TIMEGEN6

title 'Six-phase Master Timing Generator'

" Input and Output pins

MCLK, RESET, RUN, RESTART pin;

T1, P1_L, P2_L, P3_L, P4_L, P5_L, P6_L pin istype 'reg';

" State definitions

PHASES = [P1_L, P2_L, P3_L, P4_L, P5_L, P6_L];

NEXTPH = [P6_L, P1_L, P2_L, P3_L, P4_L, P5_L];

SRESET = [1, 1, 1, 1, 1, 1];

P1 = [0, 1, 1, 1, 1, 1];

equations

T1.CLK = MCLK;

PHASES.CLK = MCLK;

when RESET then {T1 := 1; PHASES := SRESET;}

else when (PHASES == SRESET ) # RESTART then {T1 := 1; PHASES :=P1;}

else when

if (RESTART & RUN & T1) then

T1 := 0;

PHASES := PHASES;

else if (RESTART & RUN & !T1) then

T1 := 1;

PHASES := NEXTPH;

else {T1 := T1; PHASES := PHASES;};

end TIMEGEN6
```

Hence, the modified program is provided.

Step 1 of 12

Consider the following truth table for the input and output sequence of a sequence detector:

Table 1

B_i	0 1	0 1	1 0	1 1	1 0
X	0 0	0 1	1 1	1 1	1 1
Y	0 0	0 0	1 0	0 1	1 0

Step 2 of 12

Draw the state diagram for X.

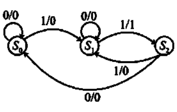


Figure 1

Step 3 of 12

Tabulate the state assignment table for the sequential circuit.

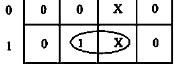
Table 2

Present state	Next state	Output		
	$B_i=0$	$B_i=1$	$B_i=0$	$B_i=1$
y_2y_1	y_2y_1	y_2y_1	X	X
00	00	01	0	0
01	01	10	0	1
10	00	01	0	0
11	XX	XX	X	X

Step 4 of 12

Apply Karnaugh's maps for y_2, y_1 and X.

The following is the Karnaugh's map for y_2.

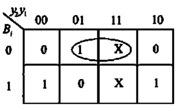


Find the expression for y_2.

$$y_2 = B_i y_1$$

Step 5 of 12

The following is the Karnaugh's map for y_1.



Find the expression for y_1.

$$\begin{aligned} y_1 &= \overline{B}_i y_1 + B_i \overline{y}_1 \\ &= B_i \oplus y_1 \end{aligned}$$

Find the expression for X from truth table.

$$\mathrm{X} = B_i \overline{y}_2 y_1$$

Step 6 of 12

The following is the state diagram for detecting consecutives ones:

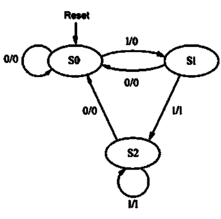


Figure 2

Step 7 of 12

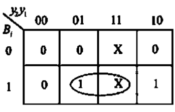
Tabulate the state assignment table for the sequential circuit.
Table 3

Present state	Next state	Output		
	$B_i=0$	$B_i=1$	$B_i=0$	$B_i=1$
y_2y_1	y_2y_1	y_2y_1	Y	Y
00	00	01	0	0
01	00	10	0	1
10	00	10	0	1
11	XX	XX	X	X

Step 8 of 12

Apply Karnaugh's maps for y_2, y_1 and Y.

The following is the Karnaugh's map for y_2.

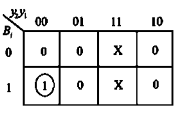


Find the expression for y_2.

$$y_2 = B_i y_1 + B_i y_2$$

Step 9 of 12

The following is the Karnaugh's map for y_1.

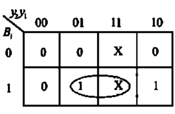


Find the expression for y_1.

$$y_1 = B_i \overline{y}_1 \overline{y}_2$$

Step 10 of 12

The following is the Karnaugh's map for Y.



Find the expression for Y.

$$\begin{aligned} \mathrm{Y} &= B_i y_1 + B_i y_2 \\ &= y_2 \end{aligned}$$

Step 11 of 12

Use the all equations to design the logic diagram.

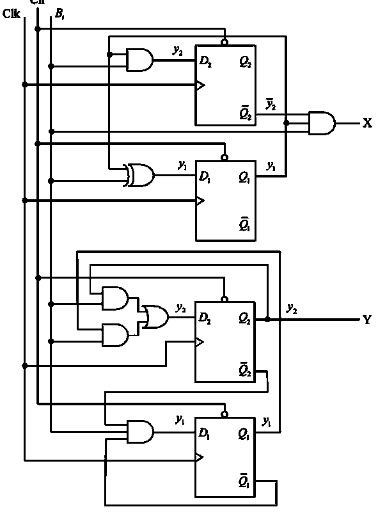


Figure 3

Step 12 of 12

Observe from the logic diagram in Figure **3** that the output X asserts if there appear two ones in the input and the output Y asserts if there appear two consecutive ones in the input.

Step 1 of 1

8.082E

Refer to the Figure X8.20 from the text book.

It is observed from the Figure that the asynchronous inputs and the clock are given to the D-flip flops, so the transitions on SYNCIN occur at maximum of 20 ns after the CLOCK.

Consider that the clock period is 40 ns and a 10 ns setup-time is required for the other 74ALS74, therefore 10 ns is the maximum propagation delay of the combinational logic.

Thus, the maximum propagation delay is $\boxed{10\text{ ns}}$.

Step 1 of 2

8.083E

Refer to Figure X8.83 from the textbook for the circuit.

Observe that the circuit has three flipflops (74AC74). Therefore,

$$n = 3$$
$$t_{\text{setup}} = 4.5\text{ ns}$$
$$t_{\text{pd}} = 10.5\text{ns}$$

Calculate the rise time.

$$\begin{aligned} t_r &= \frac{t_{\text{setup}} + t_{\text{pd}}}{n} \\ &= \frac{4.5\text{ ns} + 10.5\text{ ns}}{3} \\ &= \frac{15\text{ ns}}{3} \\ &= 5\text{ ns} \end{aligned}$$

Step 2 of 2

Calculate the clock period.

$$\begin{aligned} t_{\text{clk}} &= t_r + t_{\text{setup}} \\ &= 5\text{ ns} + 4.5\text{ ns} \\ &= 9.5\text{ ns} \end{aligned}$$

The maximum clock frequency is,

$$\begin{aligned} f_{\text{clk}} &= \frac{1}{t_{\text{clk}}} \\ &= \frac{1}{9.5 \times 10^{-9}} \\ &= \left(\frac{1000}{9.5}\right)10^6 \\ &= 105.26\text{ MHz} \end{aligned}$$

Therefore, the maximum frequency of CLOCK is $\boxed{105.26\text{ MHz}}$.

Step 1 of 3

Refer to Figure X8.83 from the textbook for the circuit.

Observe that the circuit has three flipflops (74AC74). Therefore,

$$n = 3$$
$$t_{\text{setup}} = 4.5\text{ ns}$$
$$t_{\text{pd}} = 10.5\text{ns}$$

Calculate the rise time.

$$\begin{aligned} t_r &= \frac{t_{\text{setup}} + t_{\text{pd}}}{n} \\ &= \frac{4.5\text{ ns} + 10.5\text{ ns}}{3} \\ &= \frac{15\text{ ns}}{3} \\ &= 5\text{ ns} \end{aligned}$$

Step 2 of 3

Calculate the clock period.

$$\begin{aligned} t_{\text{clk}} &= t_r + t_{\text{setup}} \\ &= 5\text{ ns} + 4.5\text{ ns} \\ &= 9.5\text{ ns} \end{aligned}$$

The maximum clock frequency is,

$$\begin{aligned} f_{\text{clk}} &= \frac{1}{t_{\text{clk}}} \\ &= \frac{1}{9.5\times10^{-9}} \\ &= \left(\frac{1000}{9.5}\right)10^6 \\ &= 105.26\text{ MHz} \end{aligned}$$

Step 3 of 3

Refer to Table 8-43 from the text book. For 74ACxx (74AC74),

$$T_o = 1.1\cdot10^{-4}\text{ s}$$

$$\tau = 0.36\text{ ns}$$

Write the formyla for mean time between failures is (MTBF).

$$\text{MTBF} = \frac{\exp\left(\frac{t_r}{\tau}\right)}{T_o\cdot f\cdot a}$$

If the asynchronous transition rate, $a = 4\text{ MHz}$ then the synchronizer MTBF is,

$$\begin{aligned} \text{MTBF}(5\text{ ns}) &= \frac{\exp\left(\frac{5\text{ ns}}{0.36\text{ ns}}\right)}{1.1\cdot10^{-4}\cdot105.26\cdot10^6\cdot4\cdot10^6} \\ &= \frac{1.076\times10^6}{463.144\cdot10^8} \\ &= 2.323\cdot10^{-5} \\ &= 0.02323\text{ ms} \end{aligned}$$

Therefore, the synchronizer's MTFB is $\boxed{0.02323\text{ ms}}$.

Step 1 of 2

Refer to Figure X8.83 from the textbook.

Refer to Table 8-43 from the textbook for typical values of D flip-flop 74ACXX (74AC74).

The typical values are,

$$T_o = 1.1 \cdot 10^{-4}\ \text{s}$$
$$\tau = 0.36\ \text{ns}$$

The clock frequency is $f = 40\ \text{MHz}$.

Refer to Figure X8.83 of the exercise 83 from the textbook. The circuit has three flipflops. Therefore,

$$n = 3$$
$$t_{\text{setup}} = 4.5\ \text{ns}$$
$$t_{\text{pd}} = 10.5\text{ns}$$

Calculate the rise time.

$$\begin{aligned} t_r &= \frac{t_{\text{setup}} + t_{\text{pd}}}{n} \\ &= \frac{4.5\ \text{ns} + 10.5\ \text{ns}}{3} \\ &= \frac{15\ \text{ns}}{3} \\ &= 5\ \text{ns} \end{aligned}$$

Step 2 of 2

Write the formyla for mean time between failures is (MTBF).

$$\text{MTBF} = \frac{\exp\left(\frac{t_r}{\tau}\right)}{T_o \cdot f \cdot a}$$

If the asynchronous input change, $a = 4\ \text{MHz}$ then the synchronizer MTBF is,

$$\begin{aligned} \text{MTBF}(5\ \text{ns}) &= \frac{\exp\left(\frac{5\ \text{ns}}{0.36\ \text{ns}}\right)}{1.1 \cdot 10^{-4} \cdot 40 \cdot 10^6 \cdot 4 \cdot 10^6} \\ &= \frac{10.66 \times 10^5}{1.1 \cdot 10^{-4} \cdot 40 \cdot 10^6 \cdot 4 \cdot 10^6} \\ &= 6.66 \cdot 10^{-5}\ \text{s} \end{aligned}$$

Therefore, the MTBF of the synchronizer is $\boxed{6.66 \cdot 10^{-5}\ \text{s}}$.

Step 1 of 1

Refer to Figure X8.86 (a) from the textbook for the circuit which eliminates the metastability with in one period of a system clock. The output of a D latch is unpredictable when the input D changes at any time during the setup and hold time. This is called the metastability condition in which the output is either 0 or 1.

Refer Figure X8.86 (b) from the textbook for the excepted waveforms. These expected waveforms of the circuit are not the actual waveforms. Asynchronous inputs are often requests for services like interrupts or status flags. These inputs change slowly when compared to the clock frequency and they need to be recognized at a particular clock tick and if it is missed they will be detected at the next clock tick. D flip-flop samples the asynchronous input at each tick of the system clock and produces a synchronous output that is valid during the next clock period. The synchronous output is cleared by the NAND gate which results in asynchronous outputs.

Hence, the circuit and the timing diagrams are failed.

8.087E

We don't have the solution to this problem yet.

Get help from a Chegg subject expert.

Ask an expert

8.088E

We don't have the solution to this problem yet.

Get help from a Chegg subject expert.

Ask an expert

Step 1 of 2

Refer to Figure 8-85 from the textbook for a synchronization control circuit using a latch. The circuit modifies to the circuit shown in Figure 8-86 from the textbook if the edge-triggered flip flop is used instead of latch.

The maximum delay takes 3SCLK periods to load SBYTE and assert SLOAD, 2CQ periods for two flip flops, 2SU periods to setup the two flip flops.

$t_{\text{maxd}} = 3t_{\text{SCLK}} + 2t_{\text{CQ}} + 2t_{\text{su}} - t_{\text{RCLK}}$ ...... (a)

The SBYTE must be remain valid until at least time $t_{\text{end}} + t_{\text{h}}$ to be loaded successfully into SREG. The point at which SBYTE changes and becomes invalid is 8RCLK periods after t_{start}. Thus consider the following for the proper circuit operation:

$t_{\text{end}} + t_{\text{h}} \le t_{\text{start}} + 8t_{\text{RCLK}}$ ...... (b)

For the maximum delay case, substitute $t_{\text{end}} = t_{\text{start}} + t_{\text{maxd}}$ into this relation and subtract t_{start} from both sides to obtain the following expression:

$t_{\text{maxd}} + t_{\text{h}} \le 8t_{\text{RCLK}}$ ...... (c)

Substitute the value of t_{maxd} and rearrange the equation.

$3t_{\text{SCLK}} + 2t_{\text{CQ}} + 2t_{\text{SU}} + t_{\text{h}} \le 9t_{\text{RCLK}}$ ...... (1)

Ensure that when the SYNC pulse for the next byte occurs, it is not negated until time t_{rec} after SLOAD for the previous byte was negated. So, the following is the another condition for proper operation:

$t_{\text{end}} + 2t_{\text{CQ}} + t_{\text{rec}} \le t_{\text{start}} + 8t_{\text{RCLK}} + t_{\text{CQ(min)}}$ ...... (d)

Use equations (b) and (c) to simplify the equation (d).

$3t_{\text{SCLK}} + 2t_{\text{CQ}} + 2t_{\text{SU}} + t_{\text{rec}} \le 9t_{\text{RCLK}} + t_{\text{CQ(min)}}$ ...... (2)

Step 2 of 2

Consider the following expression for minimum delay:

$t_{\text{mind}} = 2t_{\text{SCLK}} - t_{\text{h}} - nt_{\text{RCLK}}$ ...... (e)

Ensure that the new byte has propagated to the output of HREG when the setup time window of SREG begins, so consider the following expression:

$t_{\text{end}} - 2t_{\text{su}} \ge t_{\text{start}} + t_{\text{co}}$ ...... (f)

Here, t_{co} is the minimum clock-to-output delay of HREG. Substitute $t_{\text{end}} = t_{\text{start}} + t_{\text{mind}}$ and subtract t_{start} from both sides.

$t_{\text{mind}} - 2t_{\text{su}} \ge t_{\text{co}}$ ...... (g)

Substitute the value of t_{mind} in equation (e) and rearrange.

$2t_{\text{SCLK}} - t_{\text{h}} - t_{\text{co}} - 2t_{\text{su}} \ge nt_{\text{RCLK}}$ ...... (3)

Assume that t_{h} , t_{su} and t_{co} are 10 ns each then the maximum value of *n* is 2 which eases the delay requirements when D flip-flop is used instead of an SR latch.

9.01DP

Step 1 of 8

Consider the expression for the ROM size.

$$\textbf{ROM size} = 2^n \times m$$

Where,

n is the number of input lines

m is the number of output lines

Step 2 of 8

74x49 is a BCD to 7-segment Decoder which has 4 BCD inputs B_0, B_1, B_2, B_3 and 7 outputs a, b, c, d, e, f, g.

The number of inputs n is 4

The number of inputs m is 7

Calculate the ROM size.

$$\begin{aligned}\textbf{ROM size} &= 2^n \times m\\ &= 2^4 \times 7\\ &= 16 \times 7\end{aligned}$$

Therefore, the ROM size for 74x49 is $\boxed{16\times 7}$.

Step 3 of 8

74x139 is a Dual 1-of-4 Decoder which has 3 inputs E, A_0, A_1 and 4 outputs B_0, B_1, B_2, B_3.

The number of inputs n is 3

The number of inputs m is 4

Calculate the ROM size.

$$\begin{aligned}\textbf{ROM size} &= 2^n \times m\\ &= 2^3 \times 4\\ &= 8 \times 4\end{aligned}$$

Therefore, the ROM size for 74x139 is $\boxed{8\times 4}$.

Step 4 of 8

74x153 is a Dual 4 Line – 1 Line Data selector / Multiplexer which has 10 inputs $1\bar{G}, 1C_0, 1C_1, 1C_2, 1C_3, 2\bar{G}, 2C_0, 2C_1, 2C_2, 2C_3$ and 2 outputs Y_1, Y_2.

The number of inputs n is 10

The number of inputs m is 2

Calculate the ROM size.

$$\begin{aligned}\textbf{ROM size} &= 2^n \times m\\ &= 2^{10} \times 2\\ &= 1024 \times 2\end{aligned}$$

Therefore, the ROM size for 74x153 is $\boxed{1024\times 2}$.

Step 5 of 8

74x257 is a 2 input–4 bit Multiplexer which has 9 inputs $I_{0a}, I_{0b}, I_{0c}, I_{0d}, I_{1a}, I_{1b}, I_{1c}, I_{1d}, E_0$ and 4 outputs Z_a, Z_b, Z_c, Z_d.

The number of inputs n is 9

The number of inputs m is 4

Calculate the ROM size.

$$\begin{aligned}\textbf{ROM size} &= 2^n \times m\\ &= 2^9 \times 4\\ &= 512 \times 4\end{aligned}$$

Therefore, the ROM size for 74x257 is $\boxed{512\times 4}$.

Step 6 of 8

74x381 is a 4-bit Arithmetic logic unit (ALU) which has 12 inputs $(A_0 - A_3), (B_0 - B_3), (S_0 - S_2), C_n$ and 6 outputs $(F_0 - F_3), \bar{G}, \bar{P}$.

The number of inputs n is 12

The number of inputs m is 6

Calculate the ROM size.

$$\begin{aligned}\textbf{ROM size} &= 2^n \times m\\ &= 2^{12} \times 6\\ &= 4096 \times 6\end{aligned}$$

Therefore, the ROM size for 74x381 is $\boxed{4096\times 6}$.

Step 7 of 8

74x682 is an 8-bit comparator which has 16 inputs $(A_0 - A_7), (B_0 - B_7)$ and 2 outputs $(A = B), (A > B)$.

The number of inputs n is 8

The number of inputs m is 2

Calculate the ROM size.

$$\begin{aligned}\textbf{ROM size} &= 2^n \times m\\ &= 2^{16} \times 2\\ &= 65536 \times 2\end{aligned}$$

Therefore, the ROM size for 74x682 is $\boxed{65536\times 2}$.

Step 8 of 8

Consider the following tabular form:
Table 1

MSI parts	**Description**	**ROM Size**
74x49	BCD to 7-segment Decoder	16x7
74x139	Dual 1-of-4 Decoder	8x8
74x153	Dual 4 Line –1 Line Data selector / Multiplexer	1024x2
74x257	2 input – 4 bit multiplexer	512x4
74x381	4-bit Arithmetic logic unit(ALU)	4096x6
74x682	8 bit Comparator	65536x2

Step 1 of 3

74x381 is 4-bit Arithmetic logic unit (ALU) or function generator with generate and propagate outputs which has 12 inputs $(A_0 - A_3)$, $(B_0 - B_3)$, $(S_0 - S_2)$, C_n and 6 outputs $(F_0 - F_3)$, $\overline{G}$, $\overline{P}$.

74x382 is 4-bit Arithmetic logic unit (ALU) or function generator with ripple carry and overflow outputs which has 12 inputs $(A_0 - A_3)$, $(B_0 - B_3)$, $(S_0 - S_2)$, C_n and 6 outputs $(F_0 - F_3)$, OVR, COUT.

Step 2 of 3

Draw the following logic circuit.

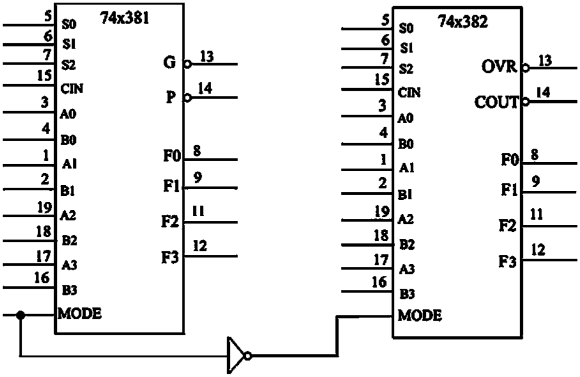


Figure 1

Step 3 of 3

Here MODE input is used to enable one of the IC.

The number of inputs n is 12

The number of inputs m is 6

Calculate the ROM size.

$$\begin{aligned}\text{ROM size} &= 2^n \times m \\ &= 2^{12} \times 6 \\ &= 4096 \times 6 \\ &= 8K \times 6\end{aligned}$$

Therefore, the ROM size for combinational logic function is $\boxed{8K \times 6}$.

Step 1 of 2

9.04DP

Consider the two 8-bit numbers i.e. $(A_0 - A_7)$ **and** $(B_0 - B_7)$ which has 16 input lines and the output of the multiplier as $(P_0 - P_{15})$ which has 16 output lines.

The number of inputs n is 16

The number of inputs m is 16

Calculate the ROM size.

$$\text{ROM size} = 2^n \times m$$

Where,

n is the number of input lines

m is the number of output lines

Substitute 16 for n and 16 for m in $\text{ROM size} = 2^n \times m$.

$$\begin{aligned}\text{ROM size} &= 2^n \times m \\ &= 2^{16} \times 16 \\ &= 65536 \times 16 \\ &= 64\text{K} \times 16\end{aligned}$$

Therefore, the ROM size for 74x49 is $\boxed{64\text{K} \times 16}$.

Step 2 of 2

Draw the logic symbol for 8x8 Multiplier.

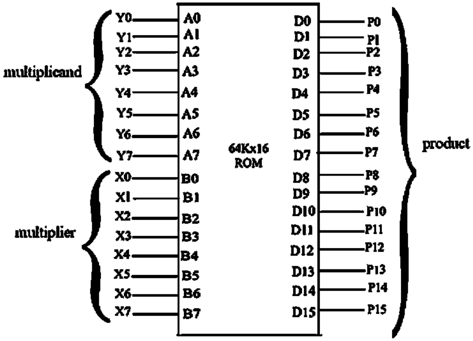


Figure 1

Step 1 of 2

Refer the Logic symbol for HM628512 in the Figure 9-24 from the textbook.

HM628512 is a $512\text{K}\times 8$ SRAM.

Convert $512\text{K}\times 8$ SRAM to binary format.

$$\begin{aligned}512\text{K}\times 8 &= 512\left(2^{10}\right)\times 8\\ &= 2^{9}\left(2^{10}\right)\times 8\\ &= 2^{19}\times 8\end{aligned}$$

Step 2 of 2

Convert $2\text{M}\times 8$ SRAM to binary format.

$$\begin{aligned}2\text{M}\times 8 &= 2\left(2^{20}\right)\times 8\\ &= 2^{21}\times 8\\ &= 2^{2}\left(2^{19}\times 8\right)\\ &= 4\left(2^{19}\times 8\right)\end{aligned}$$

Replace $512\text{K}\times 8$ by $2^{19}\times 8$ in $2\text{M}\times 8 = 4\left(2^{19}\times 8\right)$.

$$\begin{aligned}2\text{M}\times 8 &= 4\left(512\text{K}\times 8\right)\\ &= 4\left(\text{HM628512}\right)\end{aligned}$$

Therefore, to build a $2\text{M}\times 8$ SRAM, we need $\boxed{\text{four HM628512}}$, one 2 to 4 decoder at input side and eight 4 x 1 multiplexer at output side.

Step 1 of 1 **9.006E**

Refer the Sneak paths in a ROM in the Figure 9-6 from the text book.

The open collector output without resistor is treated as 1 (HIGH) when output transistor is OFF, though this HIGH state may be changed to 0 (LOW) by any small noise. In the Figure 9-6 direct connection produces low output at bit line with open collector resistor. If there was no resistor, direct connection produces high output at bit line. Thus D2_L and D0_L are pulled HIGH and correct outputs are produced.

Thus, the sneak path incident happens of direct connection can change the output only in case of open-collector resistor outputs.

9.007E

Step 1 of 4

There are total 128 possible combinations of 64 product terms are possible and the final output will be the canonical sum of those terms. Here only 8 of them would be demonstrated. In Figure 9-6 according to the high order inputs (A4, A5, and A6) 74x138 asserted row (word) line will be low which will pull the corresponding column (bit) line low. Now according to the low order inputs (A3, A2, A1, and A0) the corresponding bit line will be selected and transmitted to the output of multiplexer.

Step 2 of 4

Consider the following truth table.

Table 1

High order address bits			Low order address bits				Y
A6	A5	A4	A3	A2	A1	A0	D0
0	0	1	0	1	1	1	B7(0)
0	1	0	0	1	1	1	B7(0)
0	1	0	0	1	0	1	B5(1)
0	1	1	0	1	0	1	B5(0)
0	1	1	1	0	0	0	B8(1)
1	1	1	1	0	0	0	B8(0)
0	0	0	1	1	1	1	B15(0)
1	0	0	0	1	0	1	B5(1)

Step 3 of 4

Consider the following truth table.

Table 2

Output bits															
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12	B13	B14	B15
1	1	1	1	1	1	1	0	1	1	1	0	1	0	0	0
1	1	1	1	1	1	1	0	1	1	1	0	1	0	0	0
1	1	1	1	1	1	1	0	1	1	1	0	1	0	0	0
1	1	1	0	1	0	0	0	1	0	0	0	0	0	0	0
1	1	1	0	1	0	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
1	0	0	0	0	0	0	1	0	0	0	1	0	1	1	1

Step 4 of 4

The first case is explained as follows.

When high order input bits are selected as 010, the 5th row is asserted as 0(low). Now if there are any diodes connected between row and column the corresponding column (bit) line will be pulled low. Here 7th, 11th, 13th, 14th, 15th column (bit) line will be pulled low.

Finally, depending on the value of low order input bits as 0101, the corresponding column (bit) line which is B5 will be transmitted to the output.

We don't have the solution to this problem yet.

Get help from a Chegg subject expert.

Ask an expert

Step 1 of 1

Refer the circuit in the Figure 9-17 from the textbook.

The attenuation amounts can be doubled by keeping the ROM size of digital attenuator 28C64 as same with addition of one 1 x 2 decoder. If decoder output bit is D0 the MSB (6th) bit will be taken as 0, otherwise 1. This decoder output bit will be taken into account in the implementation of the attenuator logic. Only decoder output bits are to be included in logical expression. When decoder output bit is 1, the attenuator can measure the double the amount of the previous version.

Step 1 of 1

9.010E

Consider the following c program.

```
int pow(int,int );
float frac(int , int);
UlawToLinear( pcmIN ) ;
{
int i,k,l,t,b,q,g[8]={0,0,0,0,0,0,0,0};
int u[14]={ };
int E=0,S;
float M=0,V=0;
clrscr();
q=pcmIN;
for(i=0;q>=2;i++)
{
g[i]=q%2;
q=q/2;
}
g[i]=q;
while(i<=7)
{
i++;
g[i]=0;
}
for(b=7;b>=0;b--)
printf("%d",g[b]);
for(k=3;k>=0;k--)
{
l=1;
M+=g[k]*frac(2,-l);
l++;
}
for(k=4;k<=6;k++)
{
l=0;
E+=g[k]*pow(2,l);
l++;
}
S=g[7];
V= (1-2*S)*((pow(2,E))*(2*M+33)-33);
printf(" the analog value of V is %f",V);
for(i=0;V>=2;i++)
{
u[i]=V%2;
V=V/2;
}
u[i]=V;
if(i>13)
printf (“the ulawtoLinear code is out of range”)
else
{
while(i<=13)
{
i++;
g[i]=0;
}
}
for(b=13;b>=0;b--)
printf("%d",g[b]);
return(V);
}
int pow(int p,int q)
{
int j,r=1;
for(j=1;j<=q;j++)
r=r*p;
return(r);
}
float frac(int p,int q)
{
int r=1;
while(q<0)
{
r/=p;
q++;
}
return(r);
}
```

Step 1 of 1 **9.012E**

Consider the following c program.

```
extem int UlawToLinear(int in );

extem int LinearTo Ulaw(int x);

void main()

{

int i, j , position ;

int pcmIN, linearOUT, pcmOUT ;

double atten, attenDB , fpcmOUT ;

for (position=0 ; position<=31 ; position++)

{

printf("%i attenuation (dB) :" ) ;

scanf("%f
",attenDB) ;

atten = exp(log(10)*attenDB/10);

for(i=0; i<15 ; i++)

{

printf("%04x:",position*256+i*16);

for(j=0 ; j<=15 ; j++)

{

pcmIN=i*16+j;

fpcmOUT=atten*UlawToLinear(pcmIN);

if (fpcmOUT>=0) linearOUT=floor(fpcmOUT + 0.5);

else linearOUT = ceil(fpcmOUT – 0.5) ;

pcmOUT = LinearToUlaw(linearOUT);

printf("%2x,pcmOUT");

}

for(i=0; pcmOUT >=2;i++)

{

g[i]= pcmOUT %2;

pcmOUT = pcmOUT /2;

g[i]= pcmOUT;

if(i>7)

printf ("the ulawtoLinear code is out of range")

else

{

while(i<=7)

{

i++;

g[i]=0;

}

}

for(b=7;b>=0;b–)

printf("%d",g[b]);

}

printf("
");

}

}

}
```

Step 1 of 1

The C program to generate the contents of a 256×8 ROM that converts 8-bit Grey to 8- bit Binary code is as follows:

```
#include

#include

#include

void main()

{

int b[8];

int i,g[8];

clrscr();

printf("enter 8 bits(0/1) grey code starting from MSB:
");

for(i=7;i>=0;i--)

scanf("%d",&g[i]);

b[7]=g[7];

for(i=6;i>=0;i--)

b[i]=b[i+1]^g[i];

printf("
The binary code is");

for(i=7;i>=0;i--)

printf("%d",b[i]);

getch();

}
```

The output for the program is as follows:

```
Enter 8 bits(0/1) grey code starting from MSB:11011001

The binary code is 10010001.
```

Step 1 of 3

The communication system is designed to transmit ASCII characters serially on a medium as 5 out of 10 code.

The 5 out of 10 code is obtained by making the total no of 1's in 5 out of 10 code as five. The extra three appended codes that choose to make the total number of 1's of 10 bits as five.

The appended code is obtained by changing the value of three bits to 1 starting from left.

The value in Table 1 shows some examples of 5 out of 10 code.

Table 1

ASCII code(7 bits)	Appended code(3 bits)	5 out of 10 code(10 bits)
1000001	111	1000001111
0110000	111	0110000111
0101110	100	0101110100

Step 2 of 3

The 'C' program to generate the contents of a 128×10 ROM that converts ASCII code to 5 out of 10 code is as follows:

```
#include

#include

#include

void main()

{

int i,t=0,bc[7],fc[10],a,ac[3]={0,0,0};

clrscr();

printf("enter the ASCII code of 7 bits in 0/1 only starting from MSB
");

for(i=6;i>=0;i--)

{

scanf("%d",&bc[i]);

if(bc[i]>1)

{

printf("
invalid input
");

scanf("%d",&bc[i]);

}

}

printf("The input ASCII code is ");

for(i=6;i>=0;i--)

{

if(bc[i]==1)

t++;

printf("%d",bc[i]);

}

a=5-t;

printf("
The no of 1's needed to add to make it 5 out of 10 code is %d",a);

for(i=2;i>=3-a;i--)

ac[i]=1;

for(i=9;i>=3;i--)

fc[i]=bc[i-3];

for(i=2;i>=0;i--)

fc[i]=ac[i];

printf("
The 5 out of code is ");

for(i=9;i>=0;i--)

printf("%d",fc[i]);

getch();

}
```

Step 3 of 3

The output is as follows:

```
The input ASCII code is 1000001.The no of 1's needed to add to make it 5 out of 10 code is 3.The 5 out of code is 1000001111.
```

Step 1 of 3

9.015E

The 5 out of 10 code is obtained by making the total no of 1's in 5 out of 10 code as five. The extra three appended codes choose to make the total number of 1's of 10 bits as five.

The appended code is made by changing the value of three bits to 1 starting from left.

To convert the 5 out of 10 bit code to 7 bit ASCII code, discard the first three bits from LSB side of the received 10 bit code.

If there are more or less than five 1's in the 5 out of 10 bit code.

The received code is an error code.

The values in Table 1 shows some examples to understand the logic.

Table 1

ASCII code(7 bits)	Appended code(3 bits)	5 out of 10 code(10 bits)
1000001	111	10000001111
0110000	111	0110000111
0101110	100	0101110100

Step 2 of 3

The C program to generate the contents of a $1K \times 8$ ROM that converts words into ASCII characters is as follows:

```
#include

#include

#include

void main()

{

int i,bc[7],fc[10],a=0,ec[8]={0,0,0},e=0;

clrscr();

printf("enter the received code of 10 bits in 0/1 only starting from MSB
");

for(i=9;i>=0;i--)

{

scanf("%d",&fc[i]);

if(fc[i]>1)

{

printf("
invalid input
");

scanf("%d",&fc[i]);

}

}

printf("The input '5 out of 10 code'is ");

for(i=9;i>=0;i--)

printf("%d",fc[i]);

for(i=9;i>=3;i--)

bc[i-3]=fc[i];

for(i=9;i>=0;i--)

{

if(fc[i]==1)

a++;

}

if(a!=5)

{

e=1;

printf("
The received code is not'5 out of 10'code");

}

printf("
The input ASCII code is ");

for(i=6;i>=0;i--)

printf("%d",bc[i]);

for(i=7;i>=1;i--)

ec[i]=bc[i-1];

ec[0]=e;

printf("
The 8 bit code is ");

for(i=7;i>=0;i--)

printf("%d",ec[i]);

getch();

}
```

Step 3 of 3

The output is as follows:

The input '5 out of 10 code'is 1000001111.The input ASCII code is 1000001.The 8 bit code is 10000010.

Step 1 of 3

Consider the design of a 16 bit full adder/subtracter with mode control, carry input, carry output, and two's complement overflow output.

The circuit consists of two 16 bit inputs along with carry input and mode control input.

The total number of input bits required is shown in Table 1 as follows:

Table 1

Augend/ Minuend bits	**16**
Addend/ Subtrahend bits	**16**
Carry input bit	**1**
Mode control bit	**1**
Total bits	**34**

Total number of input bits required is $\mathbf{34}$ bits.

Step 2 of 3

The output consists of 16 bit adder/subtracter result, carry output and two's complement overflow output.

Consider carry output and overflow output as a single common bit.

During the circuit operated as adder, the common bit is treated as carry output bit.

During the circuit operated as subtracter, the common bit is treated as two's complement over flow output.

The subtraction operation is performed as an addition operation of minuend and the two's complement of subtrahend.

The total number of output bits required is shown in Table 2 as follows:

Table 2

Result	**16**
Carry/Two's complement Overflow output	**1**
Total bits	**17**

The total number of output bits required is $\mathbf{17}$ bits.

Step 3 of 3

The Read only memory is designated as $\mathbf{2^n \times m}$.

Here $\mathbf{n}$ is number of input bits, $\mathbf{m}$ is number of output bits.

The input bits are address bits and output bits are data bits.

The address bits are called ROM bits.

Therefore, the number of address bits required for the circuit are $\mathbf{34}$.

The number of data bits are $\mathbf{17}$.

The size of the ROM required is as follows:

$$\textbf{Size of the ROM} = 2^{34} \times 17$$

Therefore, the number of ROM bits are $\boxed{34}$.

Step 1 of 4

The circuit is a 16 bit full adder/subtracter, it contains 32 input bits with carry input and mode control.

Therefore, the total number of input bits is the sum of 32 input bits, carry bit and mode control.

The sum gives 34 bits..

The realization of the full adder/subtracter requires 2 same size ROM's.

The 2 same size ROM's are used to implement the 16 bit adder/ subtracter operation.

The carry output bit of one ROM is carry input bit of other ROM.

Mode control input bit is common to both the ROM's.

The number of input bits for the each ROM is shown in Table 1 and is as follows:

Table 1

Augend/ Minuend bits	8
Addend/ Subtrahend bits	8
Carry input bit	1
Mode control bit	1
Total bits	18

Step 2 of 4

There are 8 output data bits and one overflow/ carry output bit.

Overflow bit and carry output bit are same.

The both bits are equal to C_{OUT} .

The number of output bits for each ROM is shown in Table 2 as follows:

Table 2

Outout data bits	8
Carry/Two's complement Overflow output	1
Total bits	9

Step 3 of 4

The number of input bits and output bits required by each ROM is 18 and 9 .

The size of each ROM required to design the circuit is as follows:

$$\begin{aligned}\textbf{Size of each ROM} &= 2^{18} \times 9 \\ &= 256\text{ k} \times 9\end{aligned}$$

By this design approach, the size of each ROM is minimized, but the overall size of the circuit is not minimized.

The design of the 16 bit adder/subtracter with two ROM's of equal size is shown in Figure 1.

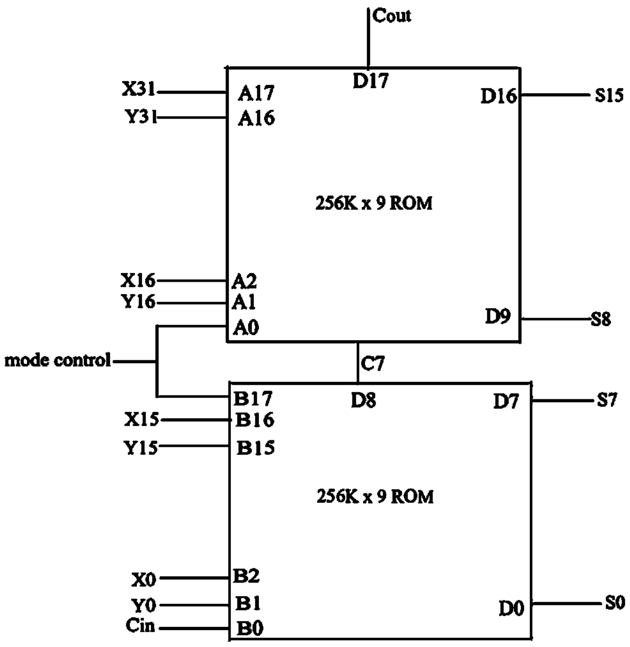


Figure 1

Step 4 of 4

The number of ROM bits can be reduced by using two different size ROM's.

The reduction in the ROM bits is only for the individual ROM.

The total number of ROM bits required in the design of 16 bit adder/subtracter is not reduced.

Therefore, the number ROM bits $\boxed{\text{cannot be reduced}}$.

9.018E

Step 1 of 4

The μ law adder circuit with a $32K \times 8$ ROM and two XOR gates is shown in Figure 1.

In this circuit, out of 16 adder bits inputs one pair of inputs is applied to one ex-or gate, which performs the operation of addition of those operand bits.

The second ex-or gate is used to generate common active low chip select and output enable signal.

Thus, total number of input bits to the ROM is 15.

Therefore, the address bits are as follows:

$$\begin{aligned} 2^{15} &= 2^{5} \times 2^{10} \\ &= 32K \end{aligned}$$

The total number of out bits is 8.

Step 2 of 4

The design of the 28C512, 64K x 8 ROM as 32K x 8 ROM is shown in Figure 1.

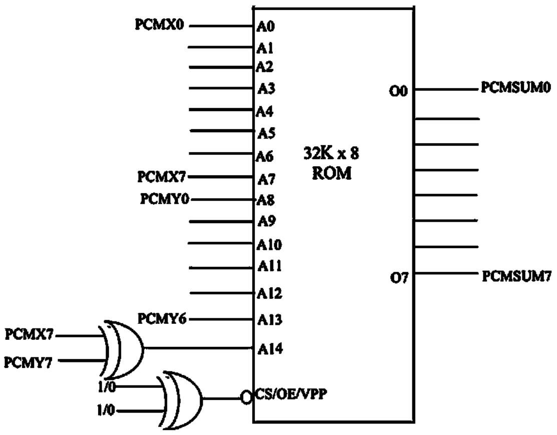


Figure 1

Step 3 of 4

The 1st and 7th elements are shown here.

The remaining intermediate elements are assumed to be there did not showed due to space constraint.

Step 4 of 4

The C program to generate the ROM contents as follows:

```
#include

#include

extern int UlawToLinear(int in);

extern int LinearToUlaw(int x);

#define MINLINEAR -8159

#define MAXLINEAR 8159

void main()

{

int i , j,e , linearSum;

int pcmINX, pcmINY ;

for(e=0 ; e<=1; e++)

{

for(pcmINY=0; pcmINY<=127; pcmINY++)

{

for (i=0 ; i<=15 ; i++)

{

printf("%04x:" , pcmINY*256 + i*16) ;

for ( j=0 ; j<=7 ; j++)

{

pcmINX = i*16 + j ;

linearSUM= UlawToLinear(pcmINX) + UlawToLinear(pcmINY);

if (linearSUM < MINLINEAR ) linearSum = MINLINEAR ;

if (linearSUM > MAXINEAR ) linearSum = MAXLINEAR ;

printf ("%02x" , LinearToUlaw(linearSum)) ;

}

printf ("
")

}

}

}
```

Step 1 of 3

9.021E

Refer to Figure XCbb-3 in section XCbb.2 at DDPPonline.

The ROM required to build the fixed point to floating point number is as follows:

The input are B_L[10:0] and A,B,C.

The output are M0_L-M3_L,EO_L-E1_L.

The A,B,C inputs are common to all three 74x151.

All other input are from B_L[10:0] and is applied to 74x148 encoder and three 74x151 multiplexer.

Thus the total numbers of input bits required are as follows:

$$11+3=14$$

The number of input bits are 14 .

The total no of required output bits are as follows:

$$4+3=7$$

The number of input bits are 7 .

Therefore the number of address input bits of fixed point to floating point encoder is as follows:

$$\begin{aligned} 2^{14} &= 2^{4}\times 2^{10} \\ &= 16\text{K} \end{aligned}$$

Therefore, the ROM size required for this fixed point to floating point encoder is $16\text{K} \times 7$.

Step 2 of 3

The logic diagram with commercially available ROM is shown in Figure 1:

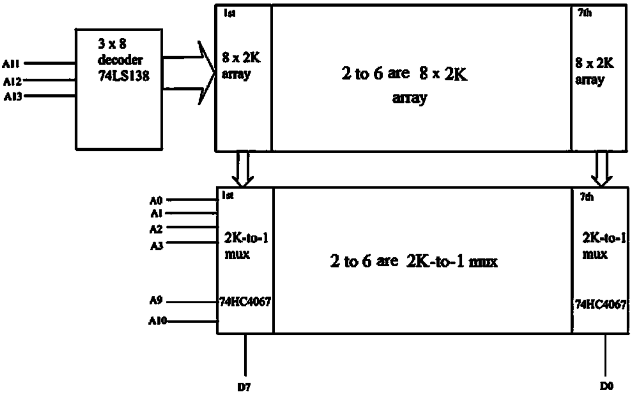


Figure 1

Step 3 of 3

Here only 1st and 7th elements were shown.

The intermediate elements were assumed to be there.

The IC 74LS138 is a 3 to 8 decoder and 74HC4067 is a 16 Channel Multiplexer.

Step 1 of 2

The fixed point number is having fixed number of digits after decimal point.

Therefore, the highest and lowest number represented in this format is as follows:

Highest number = 99999.99999
lowest number = 00000.00001

The floating point number is having variable number of digits after decimal point.

It is represented as two parts. One is mantissa and the other is exponent.

Therefore, the highest and lowest number represented in this format is as follows:

9.999999e+50 and 0.000001e-49

Here different floating type format is used to see how a floating point format differs from fixed.

Step 2 of 2

The C programming is as follows,

```
#include

#include

#include

void main()

{

double y;

clrscr();

printf("enter the fixed point value: ");

scanf("%lf",&y);

printf("
the input fixed point value is %f
%10.2f
%0.8lf",y,y,y);

printf("
the floating point value: %e",y);

getch();

}
```

Step 1 of 3

Binary multiplication is just like decimal multiplication. It is actually shifted summation. Here, eight 28C64 chips are used and in 7 chips 9 inputs lines and eight output lines are used. 10 input lines and eight output lines are used, one extra input is for SIGNED input.

Out of these 9 inputs, 8 input lines are used to represent the 8 bits of multiplicand and 9th input line is used for one bit of 8 bits multiplier.

By using full adder circuit, OR gates with output bits of different 28C64 chips, shifted addition is performed.

SIGNED input is taken at 8th 28C64 chip to designate it as signed or unsigned multiplication.

If it is signed multiplication, MSB is considered as signed bit and the magnitude is calculated out of the rest 15 bits.

Step 2 of 3

Draw the logic diagram for combinational multiplication using ROM-based circuit.

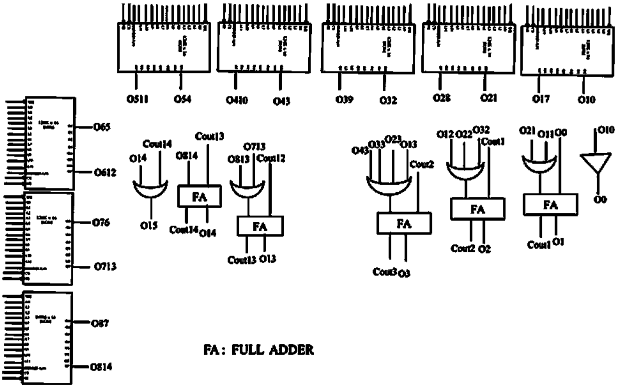


Figure 1

Step 3 of 3

The pin diagram for 28C64 chip is shown in Figure 2.

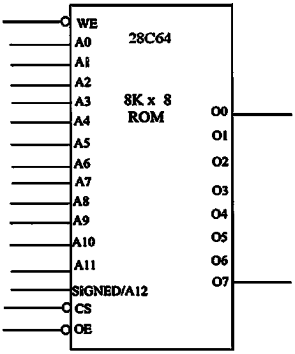


Figure 2

Thus, the logic diagram using ROM is drawn.

Step 1 of 1 9.024E

Here the C program is divided into two segments. One for unsigned operation, and other for signed operation. For signed operation, take MSB (8th) bit as sign value and rest 7 bits are taken as of magnitude value.

Write the C program to generate the ROM contents:

```
#include

#include

void main()

{

int i , j,e = 0, product;

int A, B;

printf("enter the value of e =0 and 1for signed and unsigned operation respectively");

scanf("%d",&e);

if(e==0)

{

for(A=0; A<=255; A++)

{

for (i=0 ; i<=16 ; i++)

{

for ( j=0 ; j<=6 ; j++)

{

B = i*16 + j ;

product = A*B ;

printf ("the product of % 04x row and %04x column element is %04x", A, B , product) ;

)

printf ("
")

)

)

)

if(e==1)

{

for(A=0; A<=127; A++)

{

for (i=0 ; i<=15 ; i++)

{

for ( j=0 ; j<=7 ; j++)

{

B = i*16 + j ;

product = A*B ;

printf ("the product of % 04x row and %04x column element is %04x", A, B, product) ;

)

printf ("
")

)

)

)

)
```

Hence, written and tested the c program.

9.025E

We don't have the solution to this problem yet.

Get help from a Chegg subject expert.

Ask an expert

9.030E

9.031E

9.034E

We don't have the solution to this problem yet.

Get help from a Chegg subject expert.

Ask an expert

9.035E

We don't have the solution to this problem yet.

Get help from a Chegg subject expert.

Ask an expert