LOGIC DESIGN

(Common to CSE & ISE)

Subject Code: 10CS33	**I.A. Marks : 25**
Hours/Week : 04	**Exam Hours: 03**
Total Hours : 52	**Exam Marks: 100**

PART-A

UNIT – 1 **7 Hours**

Digital Principles, Digital Logic: Definitions for Digital Signals, Digital Waveforms, Digital Logic, 7400 TTL Series, TTL Parameters The Basic Gates: NOT, OR, AND, Universal Logic Gates: NOR, NAND, Positive and Negative Logic, Introduction to HDL.

UNIT – 2 **6 Hours**

Combinational Logic Circuits

Sum-of-Products Method, Truth Table to Karnaugh Map, Pairs Quads, and Octets, Karnaugh Simplifications, Don't-care Conditions, Product-of-sums Method, Product-of-sums simplifications, Simplification by Quine-McClusky Method, Hazards and Hazard Covers, HDL Implementation Models.

UNIT – 3 **6 Hours**

Data-Processing Circuits: Multiplexers, Demultiplexers, 1-of-16 Decoder, Encoders, Exclusive-or Gates, Parity Generators and Checkers, Magnitude Comparator, Programmable Array Logic, Programmable Logic Arrays, HDL Implementation of Data Processing Circuits

UNIT – 4 **7 Hours**

Clocks, Flip-Flops: Clock Waveforms, TTL Clock, Schmitt Trigger, Clocked D FLIP-FLOP, Edge-triggered D FLIP-FLOP, Edge-triggered JK FLIP-FLOP, FLIP-FLOP Timing, JK Master-slave FLIP-FLOP, Switch Contact Bounce Circuits, Various Representation of FLIP-FLOPs, Analysis of Sequential Circuits, HDL Implementation of FLIP-FLOP

PART-B

UNIT – 5 **6 Hours**

Registers: Types of Registers, Serial In - Serial Out, Serial In - Parallel out, Parallel In - Serial Out, Parallel In - Parallel Out, Universal Shift Register, Applications of Shift Registers, Register Implementation in HDL

UNIT – 6 **7 Hours**

Counters: Asynchronous Counters, Decoding Gates, Synchronous Counters, Changing the Counter Modulus, Decade Counters, Presettable Counters, Counter Design as a Synthesis problem, A Digital Clock, Counter Design using HDL

UNIT – 7 **7 Hours**

Design of Synchronous and Asynchronous Sequential Circuits: Design of Synchronous Sequential Circuit: Model Selection, State Transition Diagram, State Synthesis Table, Design Equations and Circuit Diagram, Implementation using Read Only Memory, Algorithmic State Machine, State Reduction Technique.

Asynchronous Sequential Circuit: Analysis of Asynchronous Sequential Circuit, Problems with Asynchronous Sequential Circuits, Design of Asynchronous Sequential Circuit, FSM Implementation in HDL

UNIT – 8 **6 Hours**

D/A Conversion and A/D Conversion: Variable, Resistor Networks, Binary Ladders, D/A Converters, D/A Accuracy and Resolution, A/D Converter-Simultaneous Conversion, A/D Converter-Counter Method, Continuous A/D Conversion, A/D Techniques, Dual-slope A/D Conversion, A/D Accuracy and Resolution

Text Book:

1. Donald P Leach, Albert Paul Malvino & Goutam Saha: Digital Principles and Applications, 7th Edition, Tata McGraw Hill, 2010.

TABLE OF CONTENTS

UNIT 1: DIGITAL PRINCIPLES

ANALOG VERSUS DIGITAL

• Electronic-circuits can be divided into 2 broad categories: analog & digital (Figure: 1.1).

• *Analog-signal* is a signal whose amplitude can take any value between given limits.
On the other hand, *digital-signal* is a signal whose amplitude can have only given discrete values between defined limits.
i.e. Analog => continuous
Digital => discrete (step by step)

• Analog-circuits are generally used with small signals and are considered linear.
For e.g. an op-amp is an analog-circuit. The output signal for this circuit will be a faithfully amplified version of any signal presented at its input. This is *linear-operation*.

• Digital-circuits are generally used with large signals and are considered non-linear.
For e.g. a remote control circuit that switches the lights in a parking area after sunset.
The input signal might be a voltage representing the time of day.
The output signal is a simply on or off. This is *non-linear operation*.

• When a quantity is recorded as a series of distinct points, it is said to be sampled.

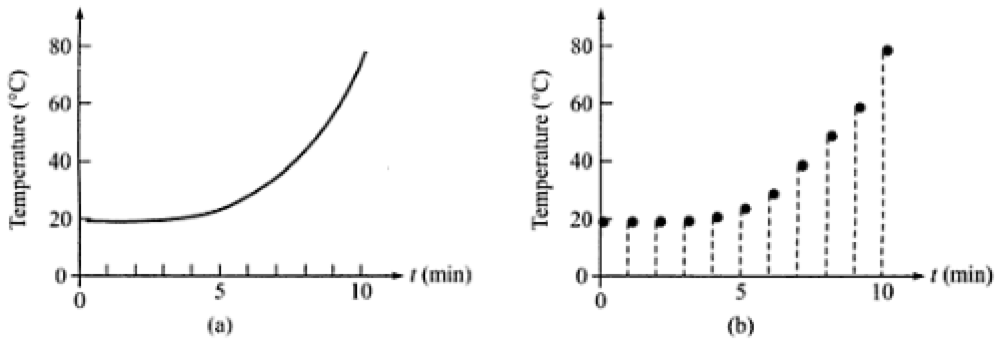


Figure 1.1: (a)analog signal (b)digital signal

BINARY SYSTEM

• The binary number system has exactly two symbols: 0 and 1.

• Choosing HIGH=1=True and LOW=0=FALSE is called *positive logic*.
Choosing HIGH=0=True and LOW=1=FALSE is called *negative logic*.

IDEAL DIGITAL SIGNALS

• The voltage levels in an ideal digital circuit will have values of either +5 V dc or 0 V dc (Figure: 1.3).

• When the voltages switch between values, they do so in *zero time*.

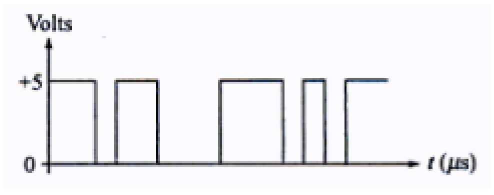


Figure 1.3: Ideal Digital Signal

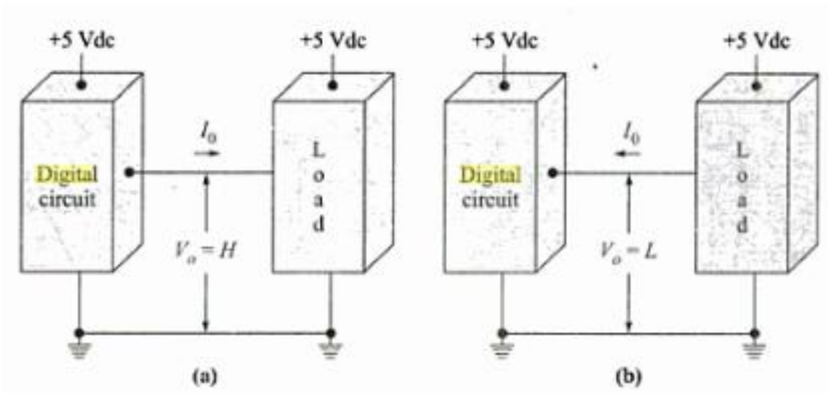
VOLTAGE LEVELS

• The output voltage level of any digital-circuit depends on its load (Figure: 1.4).

• When V_o=HIGH, the voltage should be +5 V dc. In this case, the digital circuit must act as a *current-source* to deliver the current Io to the load (Figure 1.4a).

• However, the circuit may not be capable of delivering necessary current I_o while maintaining +5 V dc. To account for this, it is agreed that any output voltage close to +5 V dc within a certain range will be considered high.

• When V_o= LOW, the voltage should be +0 V dc. In this case, the digital circuit must act as a *current-sink*. i.e. it must accept a current I_o from the load and deliver it to ground (Figure 1.4b) .



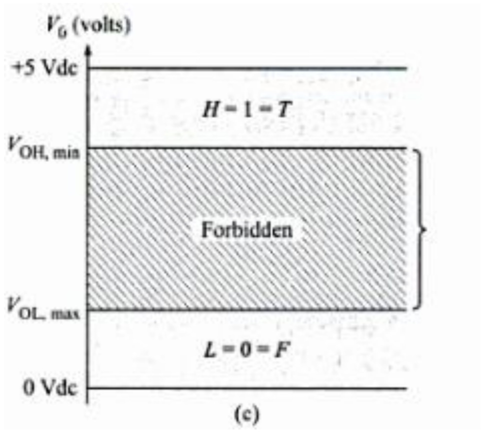


Figure 1.4: Loading of digital circuit

SWITCHING TIME

• If the digital circuit were ideal, it would change from high to low (or from low to high), in *zero time*. Thus, the output voltage would never have a value in the *forbidden range* (Figure 1.5).

• In reality, it requires a finite amount of time for V_o to make the transition between levels.

• V_o may have a value within the forbidden region only during the short time while changing from high to low (or low to high).

When not switching, V_o must either be in the high band or the low band.

• The time required for voltage to make the transition from its high level to its low level is defined as *fall time.*

The time required for voltage to make the transition from its low level to its high level is defined as *rise time.*

• Fall time is measured between 0.9H and 1.1L.

On the other hand, rise time is measured between 1.1L and 0.9H.

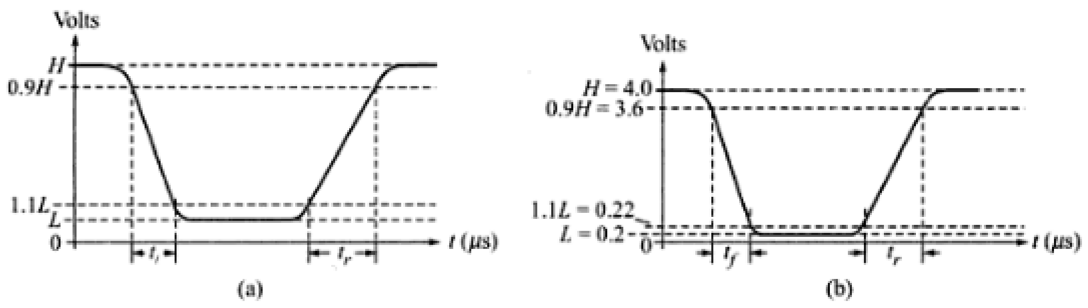


Figure 1.5: Switching in digital circuit

PERIOD & FREQUENCY
• Period(T) is the time over which the signal repeats itself (Figure: 1.6).
• Frequency (f) can be defined as f=1/T.

SYSTEM CLOCK
• This is an oscillator circuit having a very precise frequency.
• Frequency stability is provided by using a crystal as the frequency determining element.

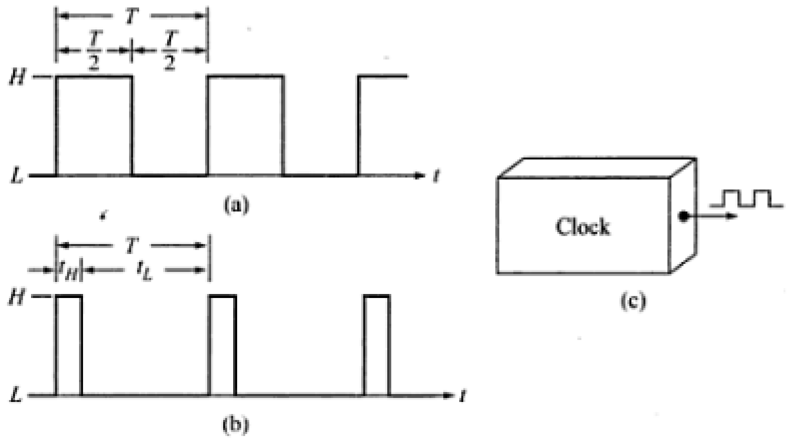


Figure 1.6: (a)Symmetrical signal with period T (b)Asymmetrical signal with period T (c)System clock.

DUTY CYCLE
• For a periodic digital signal, *duty cycle* is defined as
→ ratio of high level time to the period or
→ ratio of low level time to the period (Figure: 1.6).
i.e. Duty cycle H=$t_{H/}$T
Duty cycle L=$t_{L/}$T
• In other words, it is a convenient measure of how symmetrical or unsymmetrical a waveform is.

GENERATING LOGIC LEVELS
• The digital voltage can be produced using a *switch*.
• In figure 1.7a & 1.7b, when the switch is down, V_o=L=0=0V dc.
When the switch is up, V_o=H=1=+5V dc.
• Advantage: Switch is easy to use and easy to understand.
Disadvantage: Switch must be operated by hand (i.e. manually).
• A *relay* is a switch that is actuated by applying a voltage V_i to a coil (Figure: 1.7).
• The coil current develops a magnetic field that moves the switch-arm from one contact to the other.
• When V_i=0, V_o=L=0V dc.
When V_i=+5V dc, V_o=H=+5V dc.
• A digital integrated circuit(IC) is constructed using numerous transistors and resistors.
Each IC is designed to perform a given logic operation.
On an IC, each transistor is used as an electronic switch.

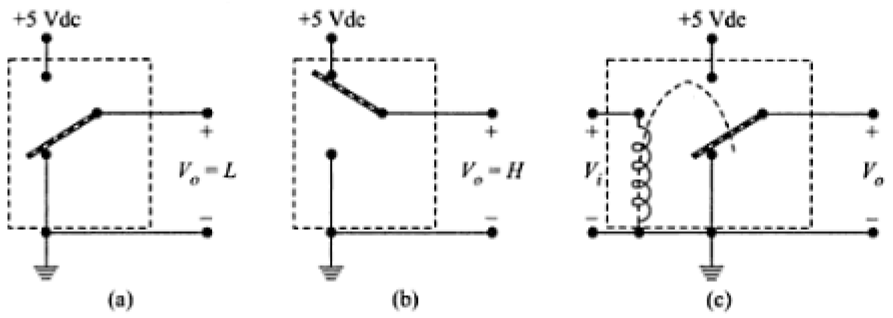
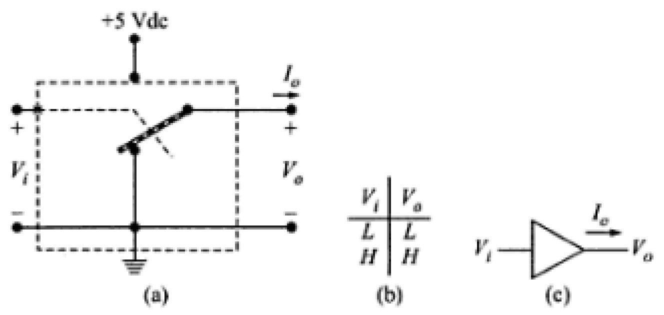


Figure 1.7: (a)Switch (b)Switch (c)Normally low relay

BUFFER

• A *Buffer* is a digital circuit which can maintain a required logic level while acting as a current source or a current sink for a given load (Figure: 1.8).
• A buffer can be thought as an *electronic switch*.
• The switch is actuated by the input voltage V_i.
• When V_i=low, the switch is down, and V_o=low.
On the other hand, when V_i=high, the switch moves up and V_o=high.
• Since buffer is capable of delivering additional current to a load, it is often called a *buffer amplifier.*

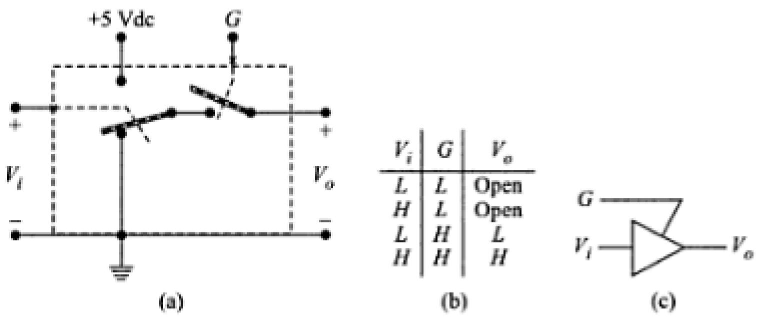


V_i	V_o
L	L
H	H

Figure 1.8: (a)Buffer amplifier model (b)Truth table (c)Symbol

TRI-STATE BUFFER

• In Figure: 1.9, when G=LOW, this switch is open and the output is disconnected from the buffer.
When G=HIGH, the switch is closed and the output follows the input.
• The control signal G connects the buffer to the load or disconnects buffer from load (Figure: 1.9).
• Since there are 3 possible states for V_o, this circuit is called a *tri-state buffer*.
• Since G controls operation of the circuit, it is often referred to as the *enable input*.

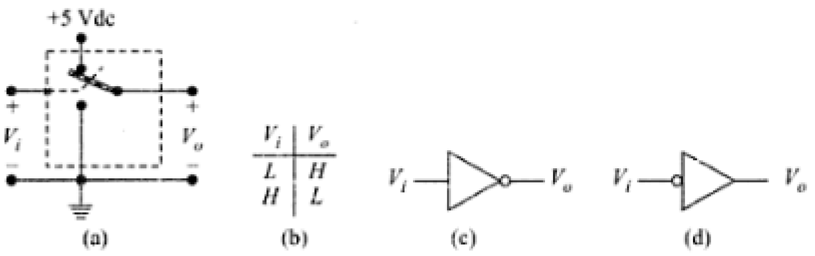


V_i	G	V_o
L	L	Open
H	L	Open
L	H	L
H	H	H

Figure 1.9: A tri-state buffer (a)Model (b)Truth table (c)Symbol

INVERTER

• Negation operation requires a circuit that will invert a digital level. This logic circuit is called an *inverter* (Figure: 1.10).
• When the input to this circuit is low, the switch remains up and the output is high.
When the input is high, the switch moves down and the output is low.

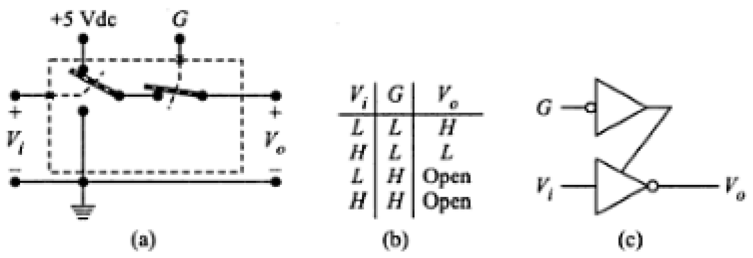


V_i	V_o
L	H
H	L

Figure 1.10: Digital inverter (a)Model (b)Truth table (c)Symbol

TRI STATE INVERTER

• When G=LOW, the inverter is connected to the output.
i.e. when G=LOW, the circuit is activated and output V_o is the inverse of the input V_i (Fig: 1.11).
• When G=HIGH, the enable switch opens, and the output is disconnected from the inverter.
• Placing a circle at the input of a logic circuit means that circuit is activated when the signal at that input is low.



V_i	G	V_o
L	L	H
H	L	L
L	H	Open
H	H	Open

Figure 1.11: Inverting tri-state buffer (a)Model (b)Truth table (c)Symbol

AND GATE

• This is a digital circuit having 2 or more inputs and a single output (Figure: 1.12 & 1.13).
• The operation of an AND gate can be expressed as follows
1. If any input is low, V_o will be low.
2. V_o will be high only when all inputs are high.
3. V_o=H only if V_1=H, and V_2=H and V_n=H.

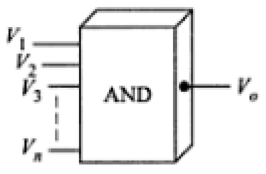
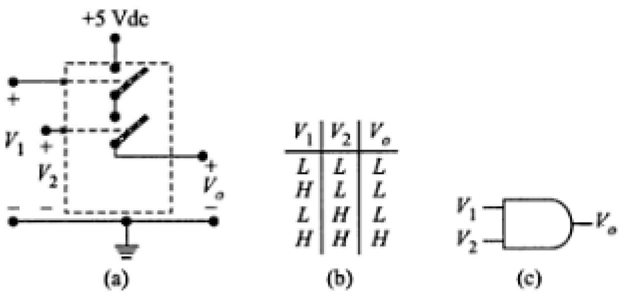


Figure 1.12: AND gate



V_1	V_2	V_o
L	L	L
H	L	L
L	H	L
H	H	H

Figure 1.13: Two input AND gate (a)Model (b)Truth table (c)Symbol

OR GATE

• It is a digital circuit having 2 or more inputs and a single output (Figure: 1.14& 1.15).
• The operation of OR gate can be expressed as follows

1. V_o will be low only when all inputs are low.
2. If any input is high, V_o will be high.
3. V_o =H if V_1 or V_2 or V_n=H

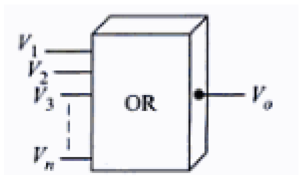
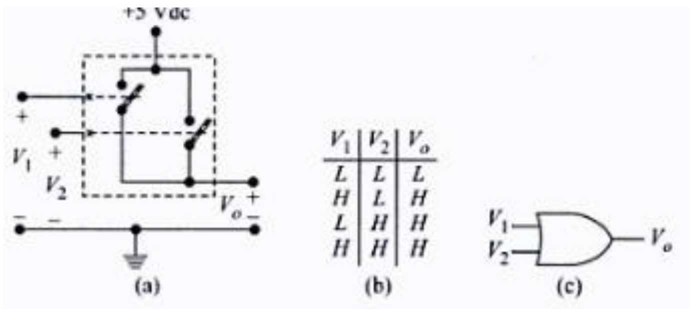


Figure 1.14: OR gate



V_1	V_2	V_o
L	L	L
H	L	H
L	H	H
H	H	H

Figure 1.15: Two input OR gate (a)Model (b)Truth table (c)Symbol

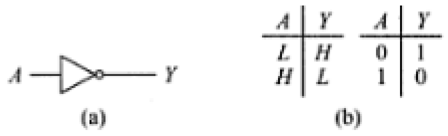
UNIT 1: DIGITAL LOGIC (CONT.)

INTRODUCTION

• A *logic gate* is a digital circuit with 1 or more input voltages but only 1 output voltage.
• Logic gates are the fundamental building blocks of digital systems.
• By connecting the different gates in different ways, we can build circuits that perform arithmetic and other functions associated with the human brain.
• Because the circuits simulate mental processes, gates are often called *logic circuits*. NOT, OR & AND gates are the basic types of gates.
• The inter-connection of gates to perform a variety of logical operations is called *logic design.*
• The operation of a logic gate can be easily understood with the help of "truth table".
• A *truth table* lists all possible combinations of inputs and the corresponding outputs.

NOT GATE (INVERTER)

• It is a gate with only 1 input and a complemented output (Figure 2.1).



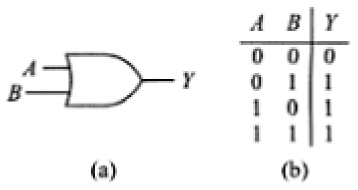
A	Y
L	H
H	L

A	Y
0	1
1	0

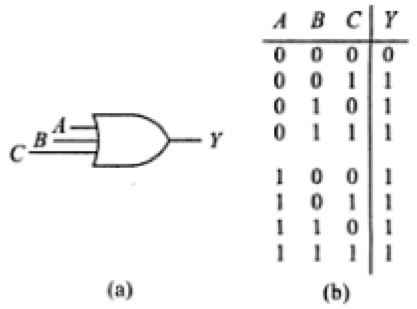
Figure 2.1: (a)Truth Table (b)Inverter Symbol

AND GATE

• This is a gate with 2 or more inputs (Figure 2.4 & 2.5).
• The output is HIGH only when all inputs are HIGH.



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

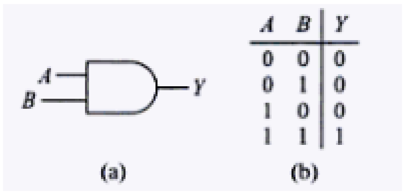


A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

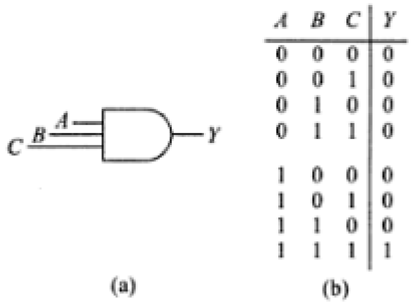
Figure 2.4: (a)Truth Table (b)2-Input AND Gate Symbol Figure 2.5 :(a)Truth Table (b)3-Input AND Gate Symbol

OR GATE

• This is a gate with 2 or more inputs (Figure 2.10 & 2.11).
• The output is HIGH when any input is HIGH.



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

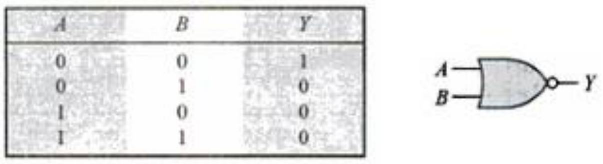


A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Figure 2.10: (a)Truth Table (b)2-Input OR Gate Symbol Figure 2.11: (a)Truth Table (b)3-Input OR Gate Symbol

NOR GATE

• This represents an OR gate followed by an inverter (Figure 2.19 & Table 2.1).



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

Figure 2.20: (a)Truth Table (b)NOR gate symbol

NAND GATE
• This represents an AND gate followed by an inverter (Figure 2.25 & Table 2.3).

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

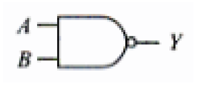


Figure 2.25: (a)Truth table (b)NAND symbol

XOR GATE
• This produces output HIGH only when odd number of 1s is present at the input (Figure: 4.30).
• This can be used as even parity bit generator.

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

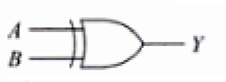


Figure 4.30: (a)Truth table (b)XOR symbol

UNIVERSAL LOGIC GATES
• Any logic function can be realized using only NAND gates or only NOR gates. For this reason, AND & NOR gates are called *universal gates*.
• Realization of other gates using only NAND gates (Figure 2.21).

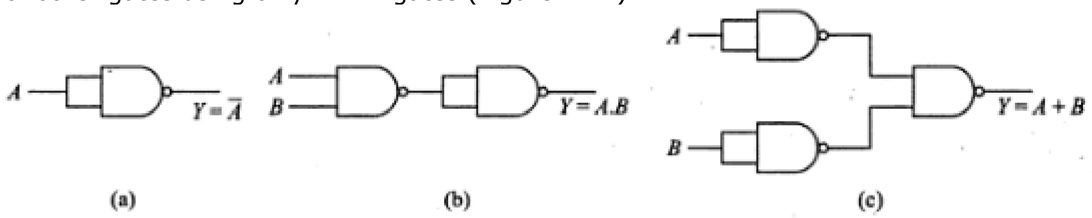


Figure 2.21: Universality of NOR gate (a)NOT from NOR (b)OR from NOR (c)AND from NOR
• Realization of other gates using only NOR gates (Figure 2.27).

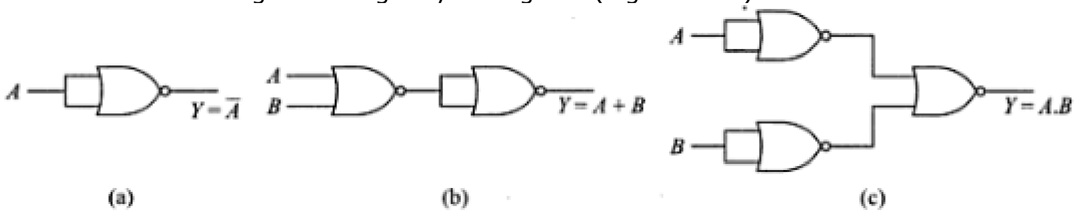


Figure 2.27: Universality of NAND gate (a)NOT from NAND (b)OR from NAND (c)AND from NAND

POSITIVE AND NEGATIVE LOGIC

• We know that, in binary logic, two voltage levels represent the two binary digits, 1 and 0.

Table 2.8

A	B	Y
Low	Low	Low
Low	High	High
High	Low	High
High	High	High

• In *positive logic*, the lower voltage level is assigned binary 0 & higher voltage level is assigned binary 1. So, we can convert table 2.8 to table 2.9.

HIGH=1
LOW=0

Table 2.9

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

• In *negative logic*, the lower voltage level is assigned binary 1 & higher voltage level is assigned binary 0. So, we can convert table 2.8 to table 2.10.

HIGH=0
LOW=1

Table 2.10

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

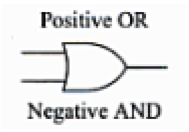


Figure 2.35: meaning of symbol depends on whether you use positive or negative logic

Table 2.9: voltage definition for basic gates

Gate	Definition
Positive OR/negative AND	Output is high if any input is high.
Positive AND/negative OR	Output is high when all inputs are high.
Positive NOR/negative NAND	Output is low if any input is high.
Positive NAND/negative NOR	Output is low when all inputs are high.

ASSERTION LEVEL

• To activate, if an input line has a bubble on it, you assert the input by making it low. If there is no bubble, you assert the input by making it high. This is called as *Assertion level.*

HDL (HARDWARE DESCRIPTION LANGUAGE)

• This is textual description of a digital circuit.

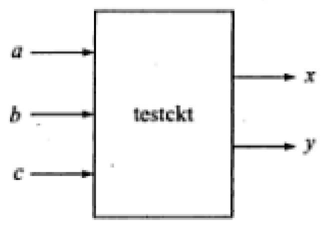
Advantages of HDL

- To describe a large complex design requiring hundreds of logic gates in a convenient manner
- To use software test-bench to detect functional error, if any, and correct it(called simulation) and
- To get hardware implementation details (called synthesis)

VERILOG HDL

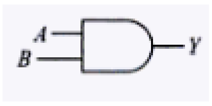
• This describes a digital system as a set of modules.

Describing input/output

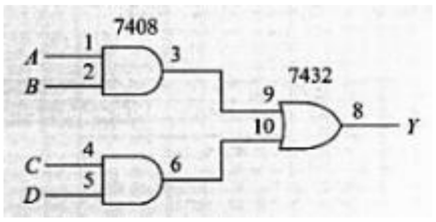


```
module testckt(x,y,z,a,b,c);  module name with port list

input a,b,c;  //defines input ports
output x,y;  //defines output ports

//module body begins next describing logic relation
.
.
//module body ends
endmodule
```

Writing module body



```
module or_gate(A,B,Y);
   input A,B;  //defines two input port
   output Y;   //define one output port
   or g1(Y,A,B); //represents OR gate
endmodule
```



```
module fig2_24(A,B,C,D,Y)
  input A,B,C,D;
  output Y;
  wire op1,op2; //internal connections
  and g1(op1,A,B); //g1 represents upper AND gate
  and g2(op2,C,D); //g2 represents lower AND gate
  or g3(Y,op1,op2); //g3 represents the OR gate
endmodule
```

PREPARATION OF TEST BENCH

- Here, we write a verilog code for simulating a OR gate
- The test bench creates an input in the form of a timing waveform and passes this to OR gate module through a function or procedural call.
- To generate timing waveform, we use time delay available in the form of #n where n=number in decimal that gives delay in nanoseconds.
- Input values to a variable can be provided through syntax m'tn

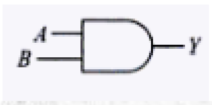
  where m=number of digits,
  t=type of number and
  n=value to be provided.
- The keyword *'reg'* is used to hold value of a data object in a procedural assignment.
- The keyword *'initial'* ensures sequential execution of codes following it, but once.
- The keyword *'always'* is used for sequential execution but for infinite time.

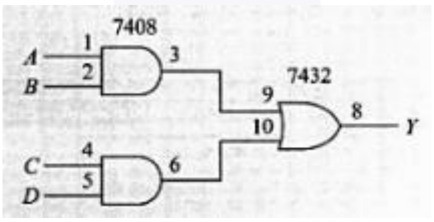
```
module testor;   //simulation module given a name tester
reg A,B;         //storage of data for passing it to module or_gate
wire x;
or_gate org(A,B,x);      //circuit is instantiated with the name,or_gate
initial                  //starts simulation
begin    // input is generated to test the circuit through following statements, simulation begins
    A=1'b0;B=1'b0;      // 1'b0 signifies on binary digit with a value 0,AB is assigned 00
    #20                  //delay of 20ns
    A=1'b0;B=1'b1;       //after 20ns AB=01
    #20                 //another delays of 20ns
    A=1'b1;B=1'b0;
    #20
    A=1'b1;B=1'b1;
    #20
end
endmodule
```

```
module or_gate(A,B,x);            //OR gate used as a procedure in simulation
  input A,B;       //defines 2 input port
  output X;       //defines 1 output port
  or #(20) g1(x,A,B);   //gate declaration with a gate delay of 20ns,output is effected after 20ns
endmodule
```

EXERCISE:

1. Compare Analog vs Digital. (4)
2. With a neat diagram, explain switching in digital circuit. (6)
3. Define duty cycle. (2)
4. With a neat diagram, explain switch & relay. (6)
5. With a neat diagram, explain buffer. (4)
6. With a neat diagram, explain tri state buffer. (4)
7. With a neat diagram, explain inverter. (4)
8. With a neat diagram, explain tri state inverter. (4)
9. With a neat diagram, two input AND gate. (4)
10. With a neat diagram, two input OR gate. (4)
11. With a neat diagram, two input OR gate. (4)
12. Explain universal gates. (4)
13. Compare positive logic vs negative logic. (4)
14. What is HDL? What are its advantages? (2)
15. Write verilog code for following circuits. (4)



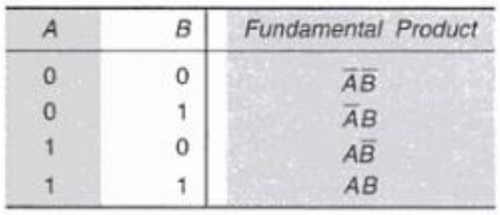


16. Explain preparation of test bench. (6)

UNIT 2: COMBINATIONAL LOGIC CIRCUITS

SUM OF PRODUCTS METHOD

- The fundamental products are also called *minterms* (Figure: 3.3 & 3.4).
- Product terms A'B, A'B, AB', AB are represented by mo, m1, m2 and m3 respectively.
- For 'n' variable, there can be 2^n number of minterms (Table: 3.1 & 3.2).



A	B	Fundamental Product
0	0	$\overline{A}\,\overline{B}$
0	1	$\overline{A}B$
1	0	$A\overline{B}$
1	1	AB

Table 3.1: Fundamental Products for two inputs

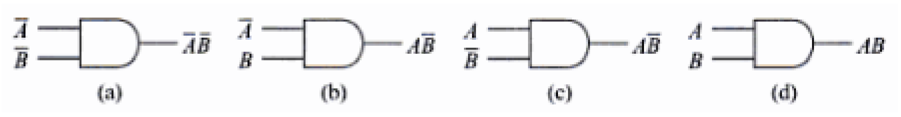
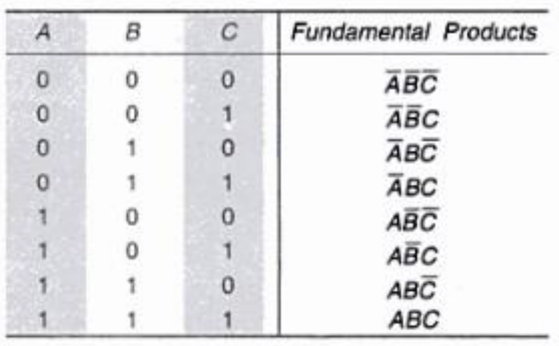


Figure 3.3: ANDing two variables and their complements



A	B	C	Fundamental Products
0	0	0	$\overline{A}\,\overline{B}\,\overline{C}$
0	0	1	$\overline{A}\,\overline{B}C$
0	1	0	$\overline{A}B\overline{C}$
0	1	1	$\overline{A}BC$
1	0	0	$A\overline{B}\,\overline{C}$
1	0	1	$A\overline{B}C$
1	1	0	$AB\overline{C}$
1	1	1	ABC

Table 3.2: Fundamental Products for Three inputs

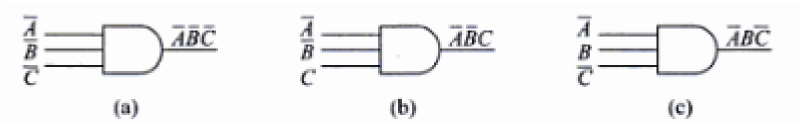


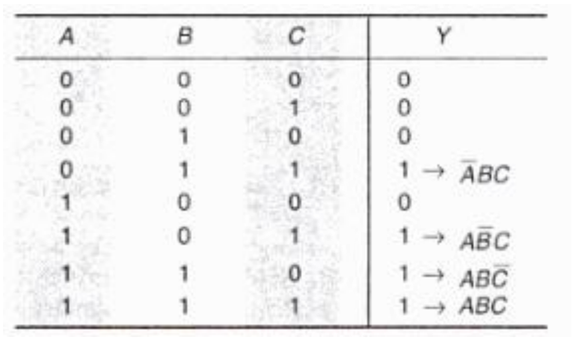
Figure 3.4: ANDing three variables and their complements

SUM OF PRODUCTS EQUATION

• Here, we have to locate output 1 in the truth table and write down the minterm (Table 3.4).
• For instance, the first output 1 appears for an input A=0, B=1 and C=1. The corresponding minterm is A'BC.
• The next output 1 appears for A=1, B=0 and C=1. The corresponding minterm is AB'C.
• To get the sum of products equation, we have to OR the minterms.

Y=A'BC+AB'C+ABC'+ABC
Y=F(A,B,C)=€m(3,5,6,7) where € denotes summation or logical OR operation.

• Y=F(A,B,C) means Y is a function of 3 boolean variables A,B and C.
• We can draw the logic circuit for Y (Figure: 3.5).



A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1 → $\bar{A}BC$
1	0	0	0
1	0	1	1 → $A\bar{B}C$
1	1	0	1 → $AB\bar{C}$
1	1	1	1 → ABC

Table 3.4: Fundamental Products

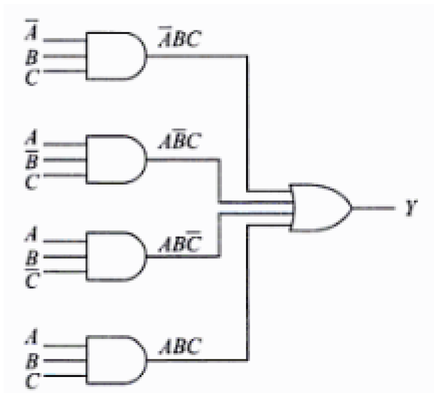
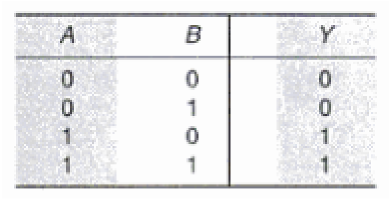


Figure 3.5: AND OR solution

TRUTH TABLE TO KARNAUGH MAP (KMAP)

• Kmap is a drawing that shows all the fundamental products (minterms) and the corresponding output values of a truth table (Table 3.5 & Figure 3.7).
• In table3.5, the first output 1 appears for A=1 and B=0. The minterm for this input condition is AB'. So, enter 1 into cell of kmap identified by row A and column B'. Similarly, enter 1 into cell identified by row A and column B.
• Finally, enter 0s in the remaining cell.



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

Table 3.5

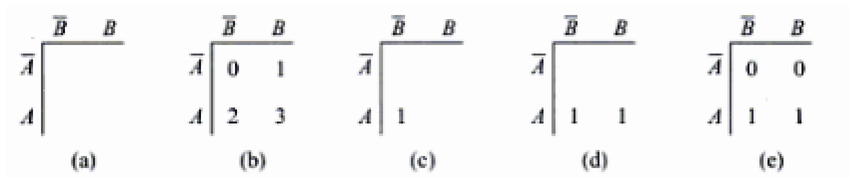
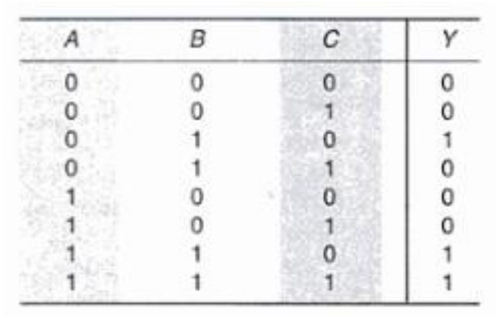


Figure 3.7: Constructing a kmap

Three-Variable Maps

• Draw a kmap for logic equation Y=F(A,B,C)=€m(2,6,7)
• Figure 3.8 shows three-variable kmap.



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

Table 3.6

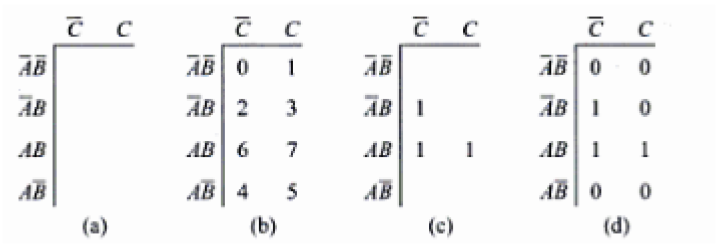


Figure 3.8: Three variable kmap

Four-Variable Maps

- Draw a kmap for logic equation Y=F(A,B,C,D)=€m(2,6,7,14)
- Figure 3.9 shows four-variable kmap (Table 3.7).

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

Table 3.7

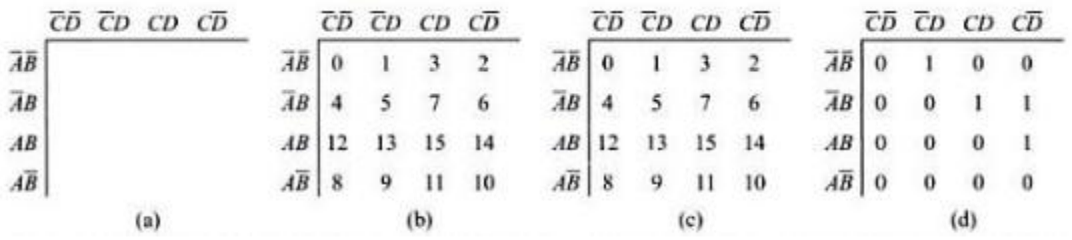


Figure 3.9: Four variable kmap

PAIRS, QUADS AND OCTETS

- A pair is a group of two 1s that are horizontally or vertically adjacent. It eliminates one variable and its complement (Figure: 3.12).
- A quad is a group of four 1s that are horizontally or vertically adjacent. It eliminates 2 variables and their complements (Figure: 3.13).
- Octet is a group of eight 1s. It eliminates three variables and their complements (Figure: 3.14).

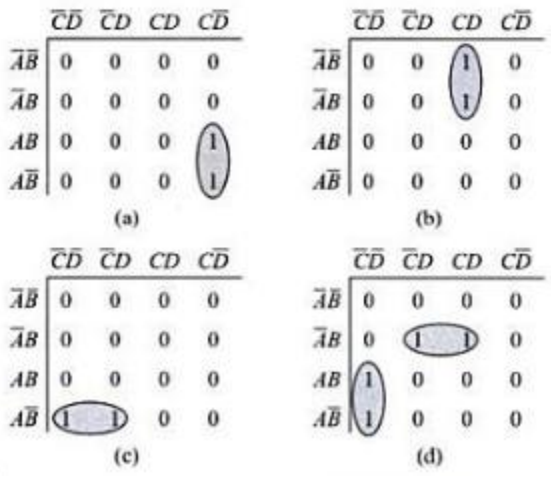


Figure 3.12: Example of pairs

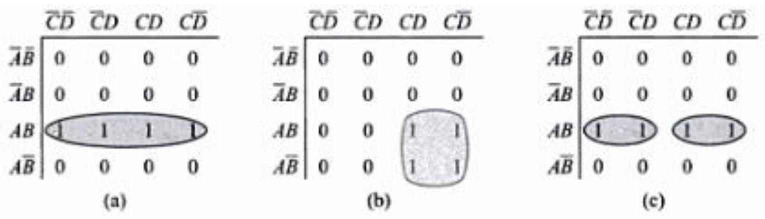


Figure 3.13: Example of quads

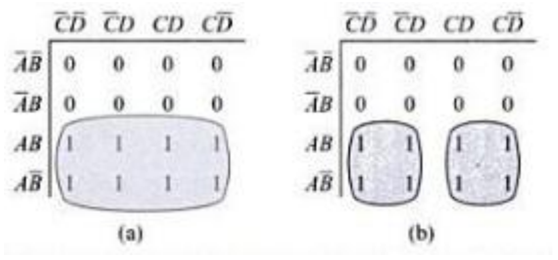


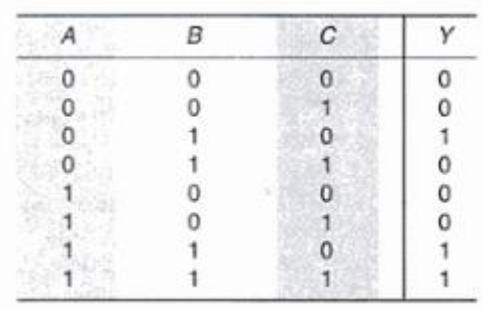
Figure 3.14: Example of octet

KMAP METHOD FOR SIMPLIFYING BOOLEAN EQUATIONS

1. Enter a 1 on map for each fundamental product that produces a 1 output in truth table. Enter 0s elsewhere.
2. Encircle the octets, quads and pairs.
3. If any isolated 1s remain, encircle each.
4. Eliminate any redundant group.
5. Write boolean equation by ORing the products corresponding to the encircled group.

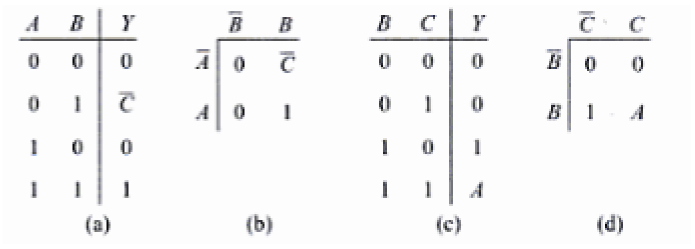
ENTERED VARIABLE MAP (EVM)

• In this, one of the input variables is placed inside kmap. This is done separately noting how it is related with output.
• This reduces the kmap size by 1 degree.
• This technique is particularly useful for mapping problems with more than 4 input variables.
• Entered variable map for Table: 3.6 is constructed as follows.
→ For AB=00, we find Y=0 and is not dependent on C (Figure: 3.10).
→ For AB=01, we find Y is complement of C thus we can write Y=C'
→ Similarly, for AB=10, Y=0 and for AB=11, Y=1.



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

Table 3.6



(a)

A	B	Y
0	0	0
0	1	$\bar{C}$
1	0	0
1	1	1

(b)

	$\bar{B}$	B
$\bar{A}$	0	$\bar{C}$
A	0	1

(c)

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

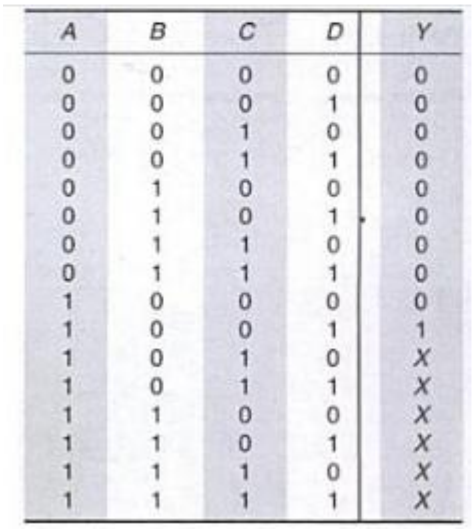
(d)

	$\bar{C}$	C
$\bar{B}$	0	0
B	1	A

Figure 3.10: Entered variable map

DON’T CARE CONDITIONS

• In some logic circuits, certain input conditions never occur; therefore the corresponding output never appears.
• In such cases, the output level is not defined; it can be either HIGH or LOW. These output levels are indicated by 'X' in the truth tables and are called *don't care conditions* (Figure: 3.23 & Table: 3.8).



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

Table 3.8: Truth table with Don't care conditions

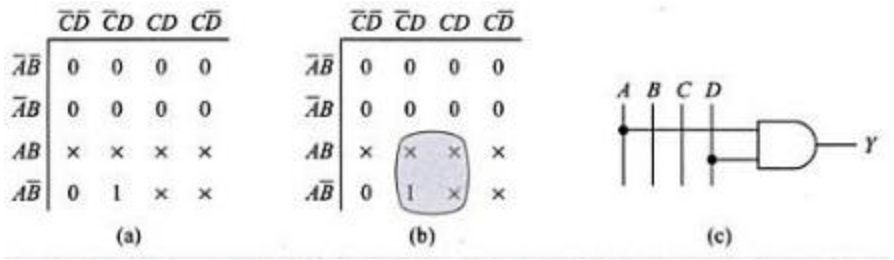


Figure 3.23:Don't care conditions

HOW TO USE DON'T CARE CONDITIONS IN KMAP SIMPLIFICATION

1>Given the truth table, draw a kmap with 0s, 1s and don't cares.
2>Encircle the actual 1s on the kmap in the largest groups you can find by treating the don't cares as 1s.
3>After the actual 1s have been included in groups, disregard the remaining don't cares by visualizing them as 0s.
Example:

Y=F(A,B,C)=€m(3,5,6,7)=   M(0,1,2,4)
Y=F(A,B,C)= M(0,3,6)= €m(1,2,4,5,7)
Y=A'BC'+AB'C+A'B'C' => Y'= (A+B'+C).(A'+B+C').(A+B+C)

PRODUCT OF SUMS METHOD

• Given a truth table, you identify the fundamental sums needed for a logic design.
• Then by ANDing these sums, you get the product-of-sums equation corresponding to the truth table.
• The fundamental sum produces an output 0 for the corresponding input condition.

CONVERTING A TRUTH TABLE TO AN EQUATION

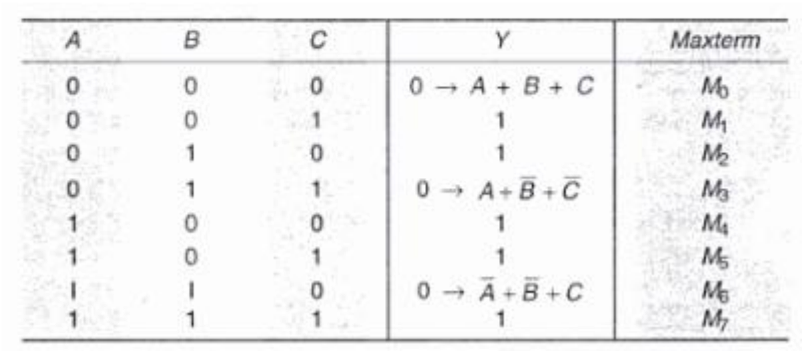
• Consider Table 3.9 and you want to get the product-of-sums equation.
• We locate each output 0 in the truth table and write down its fundamental sum.
• In Table 3.9, the first output 0 appears for A=0, B=0 and C=0. The fundamental sum for these inputs is A+B+C. Because this produces an output 0 for the corresponding input condition:

Y=A+B+C=0+0+0=0

• Table 3.9 shows all the fundamental sums needed to implement the truth table.
• Each variable is complemented when the corresponding input variable is a 1; the variable is uncomplemented when the corresponding input variable is 0.To get the product-of-sums equation, you have to AND the fundamental sums:

Y=(A+B+C)(A+B'+C')(A'+B'+C)= M(0,3,6)

• In POS, each sum term is called *maxterm* and is designated by M_i.



A	B	C	Y	Maxterm
0	0	0	$0 \rightarrow A + B + C$	M_0
0	0	1	1	M_1
0	1	0	1	M_2
0	1	1	$0 \rightarrow A + \overline{B} + \overline{C}$	M_3
1	0	0	1	M_4
1	0	1	1	M_5
1	1	0	$0 \rightarrow \overline{A} + \overline{B} + C$	M_6
1	1	1	1	M_7

Table 3.9: Fundamental Sums for Three inputs

STEPS TO CONVERT BETWEEN STANDARD SOP & POS FORM

1. Identify complementary locations
2. Changing minterm to maxterm or reverse
3. Changing summation by product or reverse.

• This is known are *conversion between canonical forms*.

LIMITATIONS OF KMAP

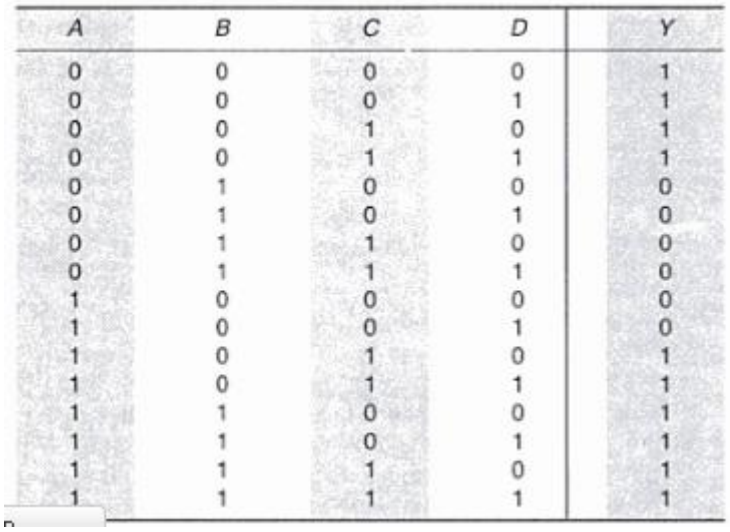
• The map method depends on the user's ability to identify patterns that gives largest size.
• The map method becomes difficult to adapt for simplification of 5 or more variables.

SIMPLIFICATION BY QUINE MCCLUSKY METHOD

• Quine McClusky method is a systematic approach for logic simplification that does not have the limitations of Kmap and also can easily be implemented in a digital computer.
• Quine McClusky method involves preparation of 2 tables:
→ one determines prime implicants and
→ other selects essential prince implicants to get minimal expression.
• *Prince implicants* are expressions with least number of literals that represents all the terms given in a truth table.
• Prime implicants are examined to get *essential prime implicants* for a particular expression that avoids any type of duplication.

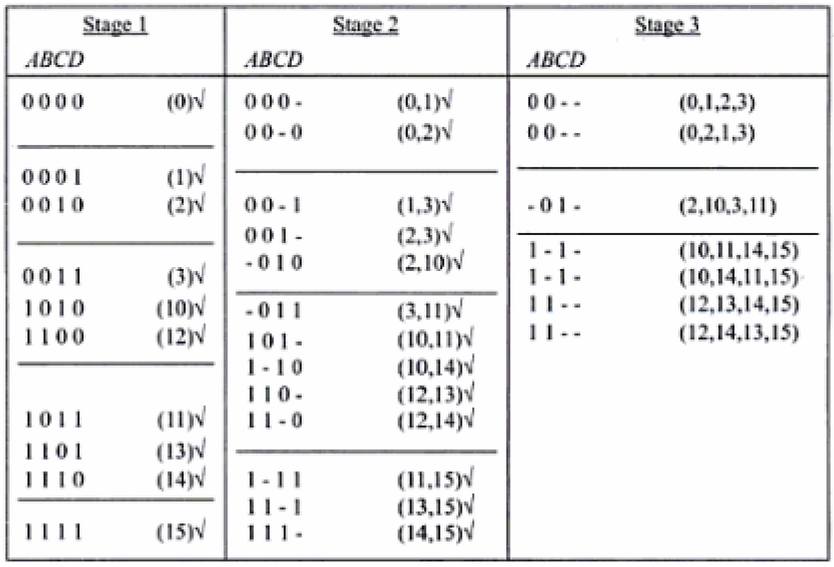
PROCEDURE USED FOR DETERMINING ESSENTIAL PRIME IMPLICANTS

• Consider a 4-variable simplification problem for Table 3.10. Figure 3.32 shows prime implicant determination table for the problem.
• In stage 1 of the process, we find out all the terms that gives output 1 from truth table (Table 3.10) and put them in different groups depending on how many 1 input variable combinations have.
For example, first group has no 1 in input combination, second group has only one 1,third two 1s,fourth three 1s and fifth four 1s. We also write decimal equivalent of each combination to their right for convenience.
• In stage 2, we first try to combine first and second group of stage 1,on a member to member basis.
• The rule is to see if only one binary digit is differing between two members and we mark that position by '-'. This means corresponding variable is not required to represent those members.
• In stage 3, we combine members of different groups of stage 2 in a similar way. Now it will have two '-' elements in each combination. This means each combination requires 2 literals to represent it.
• There is no stage 4 for this problem. This completes the process of determination of prime implicants.



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

Table 3.10



Stage 1		Stage 2		Stage 3	
ABCD		ABCD		ABCD	
0000	(0)√	000-	(0,1)√	00--	(0,1,2,3)
		00-0	(0,2)√	00--	(0,2,1,3)
0001	(1)√	00-1	(1,3)√	-01-	(2,10,3,11)
0010	(2)√	001-	(2,3)√		
		-010	(2,10)√		
0011	(3)√	-011	(3,11)√	1-1-	(10,11,14,15)
1010	(10)√	101-	(10,11)√	1-1-	(10,14,11,15)
1100	(12)√	1-10	(10,14)√	11--	(12,13,14,15)
		110-	(12,13)√	11--	(12,14,13,15)
		11-0	(12,14)√		
1011	(11)√	1-11	(11,15)√		
1101	(13)√	11-1	(13,15)√		
1110	(14)√	111-	(14,15)√		
1111	(15)√				

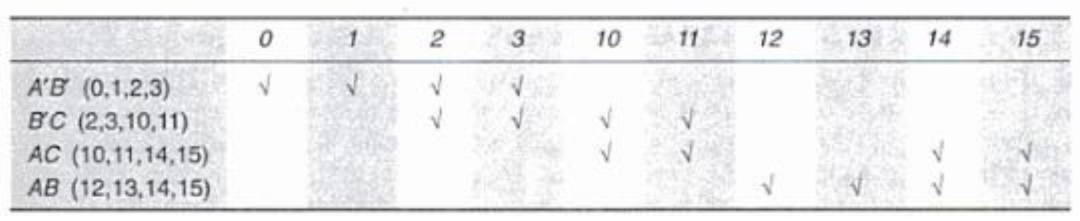
Figure 3.32: Determination of prime implicants

SELECTION OF PRIME IMPLICANTS

• Next step is to select essential prime implicants and remove redundancy or duplication among them. For this, we prepare a table as shown in Table 3.11 that along the row lists all the prime implicants and along columns lists all minterms.

• The cross-point of a row and column is ticked if the term is covered by corresponding prime implicant.

• For example, terms 0 and 1 are covered by A'B' only while 2 and 3 are covered by both A'B and B'C' and the corresponding cross-points are ticked. This way we complete the table for rest of the terms.



	0	1	2	3	10	11	12	13	14	15
A'B' (0,1,2,3)	√	√	√	√						
B'C (2,3,10,11)			√	√	√	√				
AC (10,11,14,15)					√	√			√	√
AB (12,13,14,15)							√	√	√	√

Table: 3.11

Selection of essential prime implicants from this table is done in the following way:

- We find minimum number of prime implicants that covers all the minterms.
- We find A'B' and AB cover terms that are not covered by others and they are essential prime implicants.
- B'C and AC among themselves cover 10,11 which are not covered by others. So, one of them has to be included in the list of essential prime implicants making it three. And the simplified representation of truth table given in Table3.10 is one of the following

  Y=A'B'+B'C+AB or Y=A'B'+AC+AB

HAZARDS & HAZARD COVERS

• The unwanted switching transient that may appear at the output of a circuit are called *hazards.*
• The hazards cause the circuit to malfunction.
• The main cause of hazards is the different propagation delays at different paths.

STATIC-1 HAZARD

• In a combinational circuit, if output goes momentarily 0 when it should remain a 1, the hazard is known as *static-1 hazard* (Figure: 3.34).
• This type of hazard occurs when Y=A+A' type of situation appears for a logic circuit for certain combination of other inputs and A makes a transition 1->0.
• An A+A' condition should always generate 1 at the output, i.e. static-1. But the NOT gate output takes finite time to become 1 following 1->0 transition of A. Thus for the OR gate, there are two zeros appearing at its input for that small duration, resulting a 0 at its output. The width of this zero is in nanosecond order and is called a *glitch.*

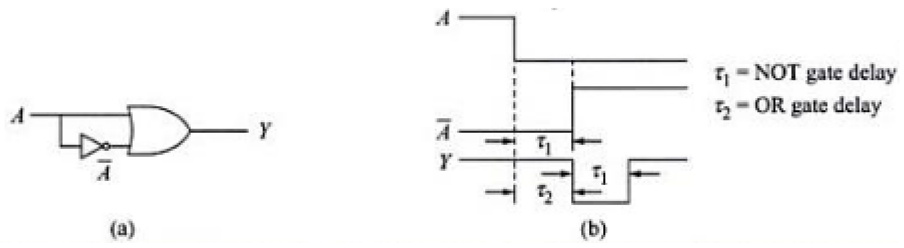


Figure 3.34: Static-1 hazard

STATIC-0 HAZARD

• If output goes momentarily 1, when it should remain a 0, the hazard is known as *static-0 hazard* (Figure: 3.36).
• This type of hazard occurs when Y=A.A' kind of situation occurs in a logic circuit for certain combination of other inputs and A makes a transition 0->1.
• An A.A' condition should always generate 0 at the output i.e. static-0. But the NOT gate output takes finite time to become 0 following a 0->1 transition of A.

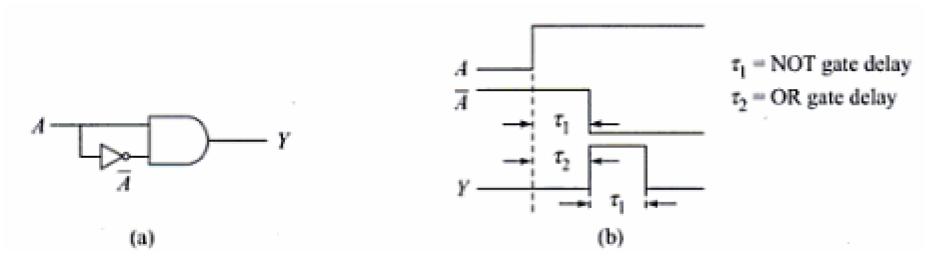


Figure 3.36: Static-0 hazard

DYNAMIC HAZARD

• Dynamic hazard occurs when circuit output makes multiple transitions before it settles to a final value while the logic equation asks for only one transition.
• When this hazard occurs, an output transition designed as
    1->0 may give 1->0->1->0
    0->1 may give 0->1->0->1

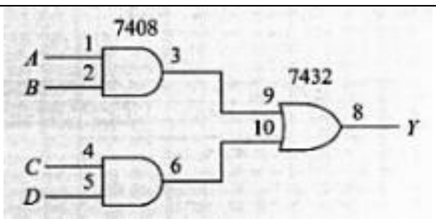
HDL IMPLEMENTATION MODEL

DATAFLOW MODELING

• Verilog provides a keyword 'assign' and a set of operators to describe a circuit through its behavior or function.
• All 'assign' statements are concurrent i.e order in which they appear do not matter and also continuous i.e any change in a variable in the right hand side will immediately effect left hand side output.

Relational Operation	*Symbol*	*Bit-wise Operation*	*Symbol*
Less than	<	Bit-wise NOT	~
Less than or equal to	<=	Bit-wise AND	&
Greater than	>	Bit-wise OR	|
Equal to	==	Bit-wise Ex-OR	^
Not equal to	!=		
Logical Operation (for expressions)	**Symbol**	**Arithmetic Operation**	**Symbol**
		Binary addition	+
Logical NOT	!	Binary subtraction	-
Logical AND	&&	Binary multiplication	*
Logical OR	||	Binary division	/

Table 3.12: List of Verilog operator



```
module Figure1(A,B,C,D,Y)
input A,B,C,D;
output Y;
assign Y=(A&B) | (C&D);
endmodule
```

BEHAVIORAL MODELING

• In a behavioral model, statements are executed sequentially following algorithmic description. It is ideally suited to describe a sequential logic circuit. However, it is also possible to describe combinational circuits with this
• It always uses 'always' keyword followed by a sensitivity list.
• The procedural assignment or output variables within 'always' must be register type, defined by 'reg'.

```
module Figure2(A,B,C,D,Y)
input A,B,C,D;
output Y;
reg Y; /* Y is a output after procedural assignment within always block, hence
defined as reg*/
always @ (A or B or C or D) //A,B,C,D form sensitivity list
if((A==1) && (B==1))
    Y=1;
else if((C==1) && (D==1))
        Y=1;
    else
        Y=0;
Endmodule
```

UNIT 3: DATA PROCESSING CIRCUITS

MULTIPLEXER

• It is a digital circuit with many inputs but only 1 output.
• By applying control-signals, we can steer any input to output.
• Thus, it is also called a *data-selector* and control inputs are termed select inputs (Figure: 4.1).

4:1 Multiplexer

• Depending on control inputs A and B, one of the 4 inputs D_0 to D_3 is steered to output Y.
• The logic equation of the circuit (Fig:4.1c) gives a SOP representation.
• Here, each AND gate generates a minterm which are finally summed by OR gate.

$Y=A'B'D_0+A'BD_1+AB'D_2+ABD_3$

If A=0, B=0; $Y=0'0'D_0+0'0D_1+00'D_2+ABD_3$

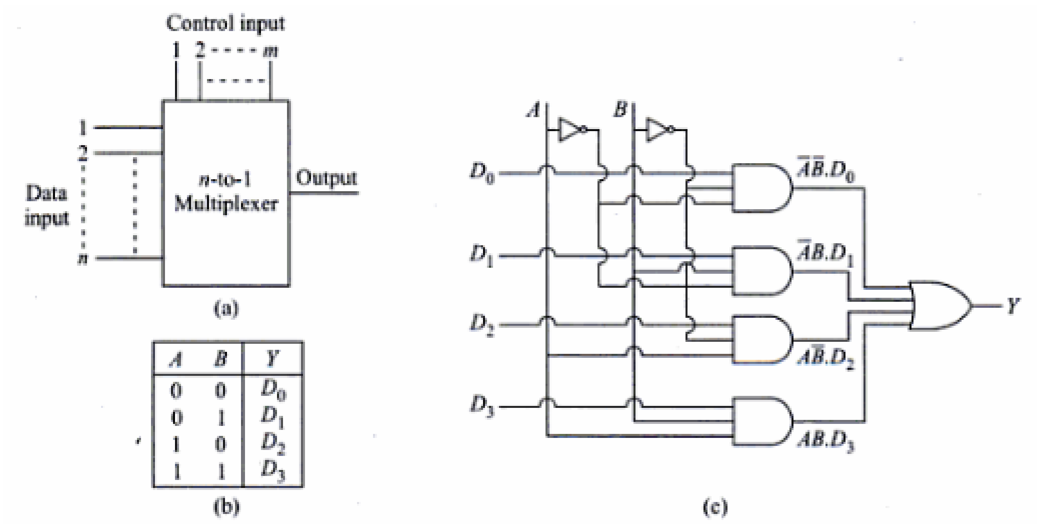
$Y=1.1.D_0+0+0+0$

$Y=D_0$

• In other words, for AB=00, the first AND gate to which D_0 is connected remains active and equal to D_0 and all other AND gate are inactive with output held at logic 0.

If $D_0=0$, Y=0 and if $D_0=1$, Y=1.

• Similarly, for AB=01, second AND gate will be active and all other AND gates remain inactive. Thus, output $Y=D_1$.



A	B	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

Figure 4.1: a)Multiplexer block diagram b)4:1 multiplexer truth table c)logic circuit

74150

- The 74150 is a 16-to-1 TTL multiplexer.
- Pin 9 is for the *STROBE* (an input signal) that disables or enables the multiplexer.
- A low strobe enables the multiplexer, so that output Y equals the complement of the input data bit:

  Y=D'n where n is the decimal equivalent of ABCD.
- On the other hand, a high strobe disables the multiplexer and forces the output into the high state. With a high strobe, the value of ABCD doesn't matter.

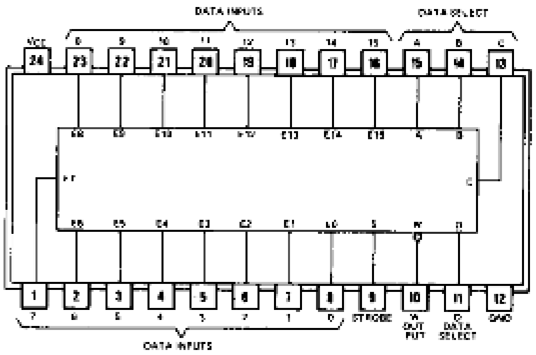


Figure 4.2a: 74150

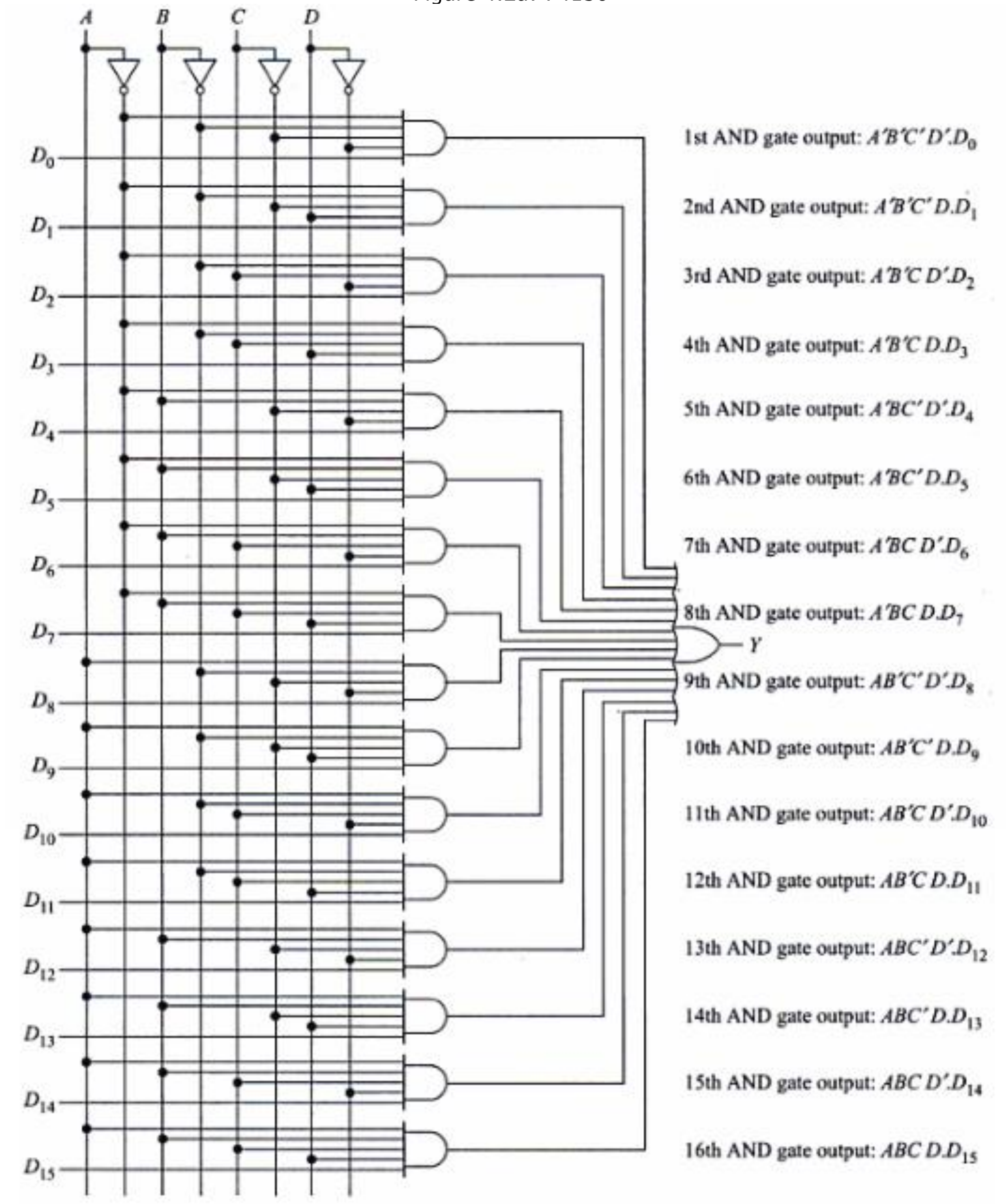

A
B
C
D
D_0
D_1
D_2
D_3
D_4
D_5
D_6
D_7
D_8
D_9
D_{10}
D_{11}
D_{12}
D_{13}
D_{14}
D_{15}
1st AND gate output: $A'B'C'D'.D_0$
2nd AND gate output: $A'B'C'D.D_1$
3rd AND gate output: $A'B'CD'.D_2$
4th AND gate output: $A'B'CD.D_3$
5th AND gate output: $A'BC'D'.D_4$
6th AND gate output: $A'BC'D.D_5$
7th AND gate output: $A'BCD'.D_6$
8th AND gate output: $A'BCD.D_7$
Y
9th AND gate output: $AB'C'D'.D_8$
10th AND gate output: $AB'C'D.D_9$
11th AND gate output: $AB'CD'.D_{10}$
12th AND gate output: $AB'CD.D_{11}$
13th AND gate output: $ABC'D'.D_{12}$
14th AND gate output: $ABC'D.D_{13}$
15th AND gate output: $ABCD'.D_{14}$
16th AND gate output: $ABCD.D_{15}$

Figure 4.2b: 16-to-1 Multiplexer

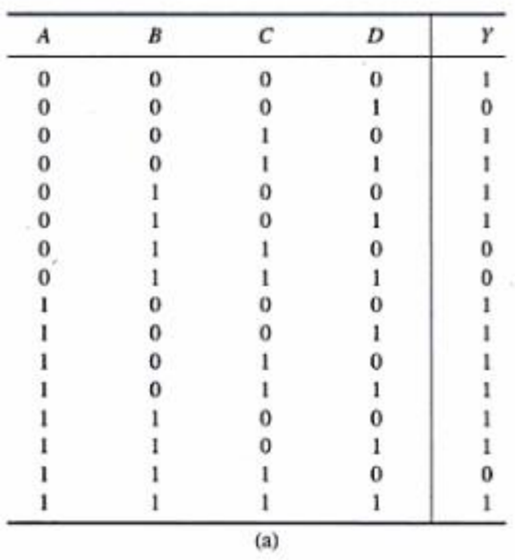
MULTIPLEXER LOGIC

• Two standard methods for implementing a truth table are SOP and POS solutions. The third method is the *multiplexer solution*. For example to use a 74150 to implement Table 4.2, complement each Y output to get the corresponding data input.

D_0=1'=0
D_1=0'=1
….
….
D_{15}=1'=0

• D_0 is grounded, D_1 is connected to +5V, D_2 is grounded and so forth (Figure 4.4).
When ABCD=0000, D_0 is the selected input. Since D_0 is low, Y is high.
When ABCD=0001, D_1 is the selected input. Since D_1 is high, Y is low.

• When the STROBE is low and inactive (disabled) when it is high. Because of this, the STROBE is called an *active-low signal*; it causes something to happen when it is low rather than when it is high.



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

(a)

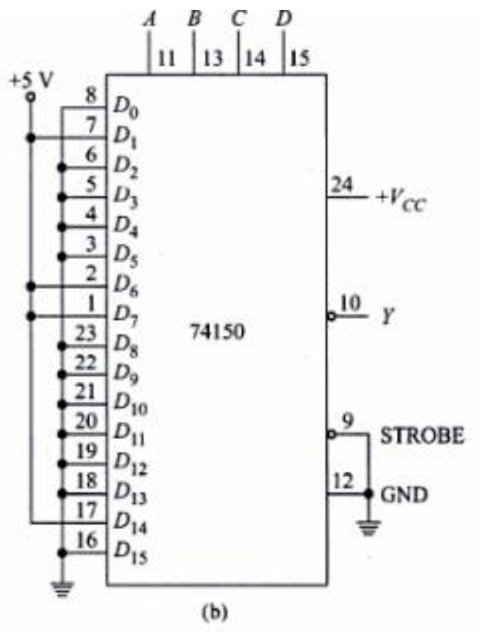


Table 4.2

Figure 4.4:Using a 74150 for multiplexer logic

Why Multiplexer is called Universal Logic Circuit?

• Because a 2^n to 1 multiplexer can be used as a design solution for any 'n' variable truth table.

• We can implement the Truth table: 4.2 using 8:1 multiplexer in following way:

Let's consider A, B and C variables to be fed as select inputs. The fourth variable D then has to be present as data input. Here, EVM method can be used to convert 4-variable truth table into 3-variable truth table.

Table 4.2a:Four variable truth table

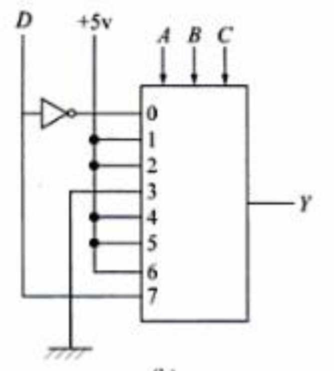
A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

(a)

Table 4.2b: Three variable truth table

ABC	000	001	010	011	100	101	110	111
$D=0$	1	1	1	0	1	1	1	0
$D=1$	0	1	1	0	1	1	1	1
Y	D'	1	1	0	1	1	1	D
8-to-1 MUX data input	$D_0=D'$	$D_1=1$	$D_2=1$	$D_3=0$	$D_4=1$	$D_5=1$	$D_6=1$	$D_7=D$

(a)



(b)

Figure 4.5: A four variable truth table realization using 8:1 multiplexer.

NIBBLE MULTIPLEXERS

• Sometimes, we want to select 1 of two input nibbles. In this case, we can use a nibble multiplexer.
• The control signal labeled SELECT determines which input nibble is transmitted to output (Fig: 4.6).

When SELECT=low, the four NAND gates on the left are activated. Therefore $Y_3Y_2Y_1Y_0=A_3A_2A_1A_0$
When SELECT=high, the four NAND gates on the right are activated. Therefore $Y_3Y_2Y_1Y_0=B_3B_2B_1B_0$

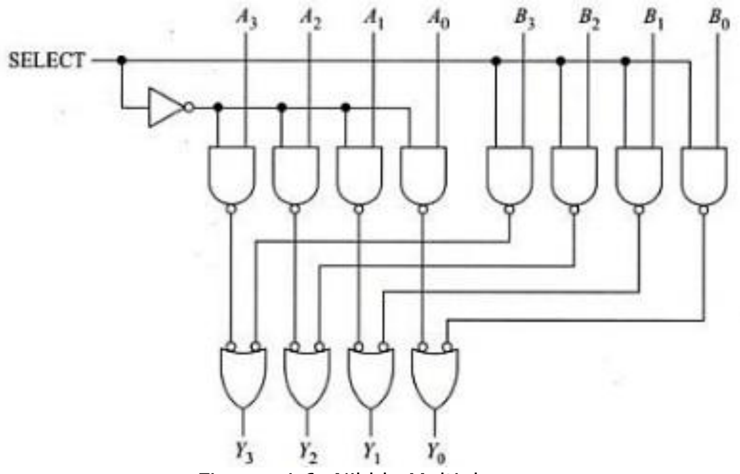


Figure: 4.6: Nibble Multiplexers

DEMULTIPLEXER

• It is a digital circuit with 1 input and many outputs (Figure: 4.9).
• By applying control signals, we can steer the input signal to one of the output lines.

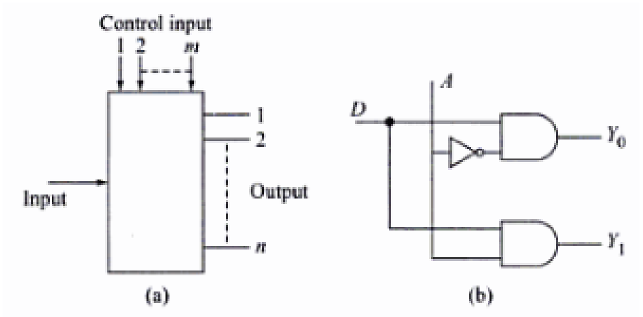


Figure 4.9: a)Demultiplexer block diagram b)Logic circuit of 1:2 demultiplexer

1:16 Demultiplexer

• The input data bit (D) is transmitted to the data bit of output lines. But which one? Again, this depends on the value of ABCD, the control input.
• When ABCD=0000, the upper AND gate is enabled while other AND gates are disabled. Therefore, data bit D is transmitted only to the Y_0 output, giving Y_0=D. If D=low, Y_0=low. If D=high, Y_0=high.
• If the control nibble is changed to ABCD=1111, all gates are disabled except the bottom AND gate. Then, D is transmitted only to the Y_{15} output and Y_{15}=D.

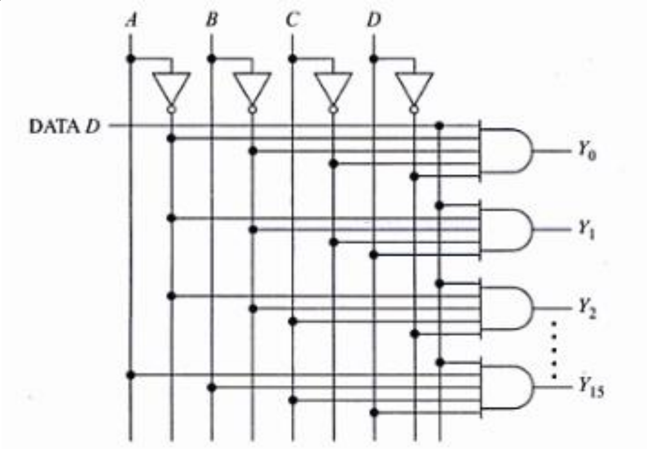


Figure 4.10: 1-to-16 demultiplexer

DECODERS

• It is a multiple-input, multiple-output logic circuit which converts coded inputs into coded outputs, where the input and output codes are different (Figure: 4.14).

• It is similar to a demultiplexer with one exception: there is no data input. The only inputs are the control bits.

1-of-16 Decoder

• 1-of-16 decoder is called so because only 1 of the 16 output lines is high.

• For instance, when ABCD=0001, only the Y_1 AND gate has all inputs high, therefore only the Y_1 output is high. If ABCD=0100, only the Y_4 AND gate has all inputs high, as a result Y_4=high.

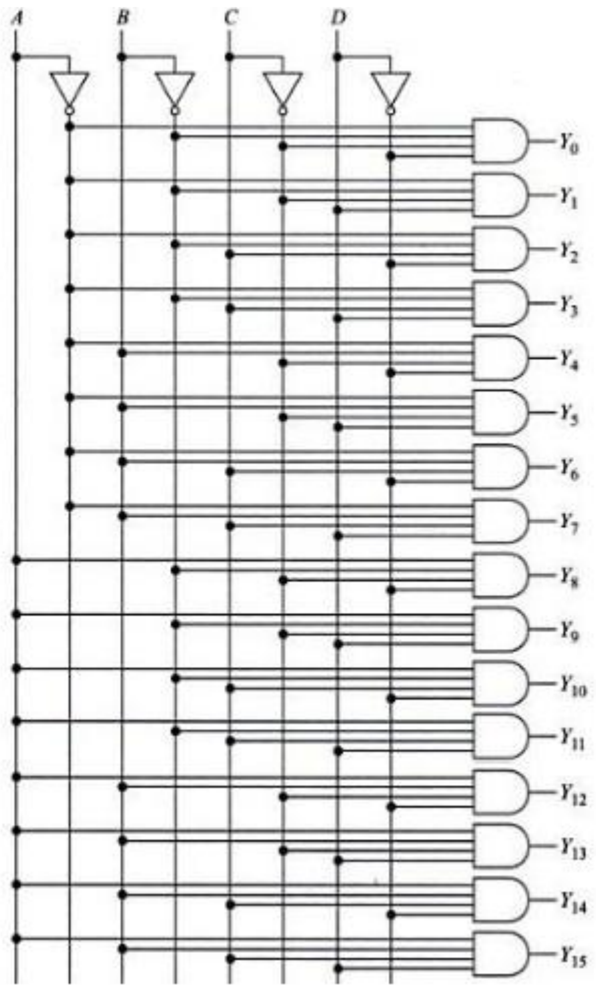


Figure 4.14: 1-of-16 decoder

ENCODERS

- It converts an active input signal into a coded signal.
- There are 'n' input lines, only one of which is active.
- Internal logic within the encoder converts this active input to a coded binary output with 'm' bits.

Decimal to BCD Encoders

- The switches are push-button switches like those of a pocket calculator (Figure: 4.24).
- When button 3 is pressed, the C and D OR gates have high inputs, therefore the output is ABCD=0011

If button 5 is pressed, output ABCD=0101
If button 9 is pressed, output ABCD=1001

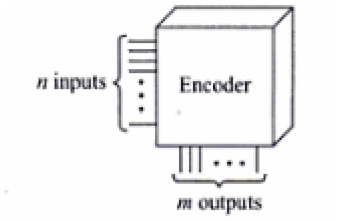


Figure: 4.24: Encoders

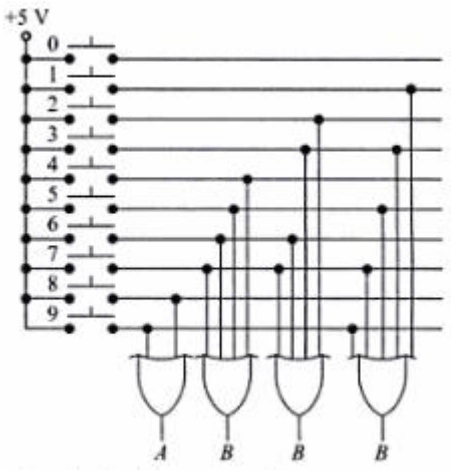


Figure 4.25: Decimal-to-BCD encoder

EXCLUSIVE OR GATES

- This has a high output only when an odd number of inputs is high (Figure: 4.29 & 4.30).
- The upper AND gate forms the product A'B, while the lower one produces AB'. Therefore, the output of the OR gate is

Y=A'B+AB'

- This gate always produces an output 1 only when n-bit input has an *odd number* of 1s (Table 4.6).

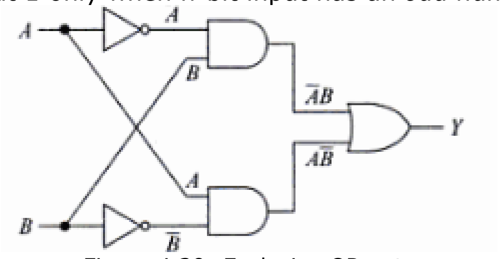


Figure 4.29: Exclusive OR gate

Table 4.6: Two input Exclusive OR gate

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

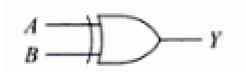


Figure 4.30: Logic symbol

PARITY GENERATORS AND CHECKERS

• Even parity means an n-bit input has an even number of 1s. For e.g. 110011 has even parity because it contains four 1s.

• Odd parity means an n-bit input has an odd number of 1s. For e.g. 110001 have odd parity because it contains three 1s.

• The circuit that generates the parity bit in the transmitter is called a *parity generator* and the circuit that checks the parity in the receiver is called a *parity checker*.

Parity Checker

• XOR gates can be used for checking the parity of a binary number because they produce an output 1 when the input has an odd number of 1s. Therefore, an even parity input to an exclusive OR gate produces a low output, while an odd parity input produces a high output (Figure: 4.33).

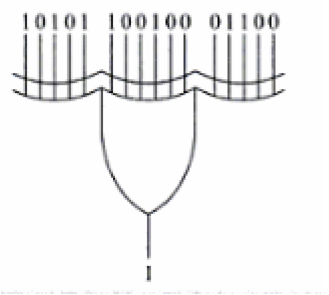


Figure 4.33: Exclusive OR gate with 16 inputs

Parity Generation

• In a computer, a binary number may represent an instruction that tells the computer to add, subtract etc. In this case, you sometimes will see an extra bit added to the original binary number to produce a new binary number with even or odd parity (Figure 4.34).

• Suppose $X_7X_6X_5X_4$ $X_3X_2X_1X_0$=0100 0001.Then, the number has even parity, which means the XOR gate produces an output 0. Because of the inverter X_8=1 and the final 9-bit output is 1 0100

• If $X_7X_6X_5X_4$ $X_3X_2X_1X_0$=0110 0001. Now, this has odd parity. In this case, the XOR gate produces an output 1. But the inverter produces a 0, so that the final 9-bit output is 0 0110 0001. Again, the final output has odd parity.

• If the 8-bit input has even parity, a 1 comes out of the inverter to produce a final output with odd parity. On the other hand, if the 8-bit input has odd parity, a 0 comes out of the inverter, and the final 9-bit output again has odd parity.

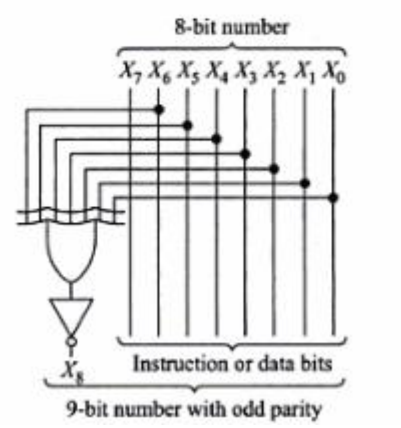

8-bit number
X_7 X_6 X_5 X_4 X_3 X_2 X_1 X_0
X_8
Instruction or data bits
9-bit number with odd parity

Figure 4.34: Odd parity generator

What is the Practical Application of Parity Generation and Checking?

• Because of transient, noise and other disturbance paths, 1-bit errors sometimes occur when binary data is transmitted over telephone lines or other communication paths.

• One way to check for errors is to use an odd parity generator at the transmitting end and an odd-parity checker at the receiving end.

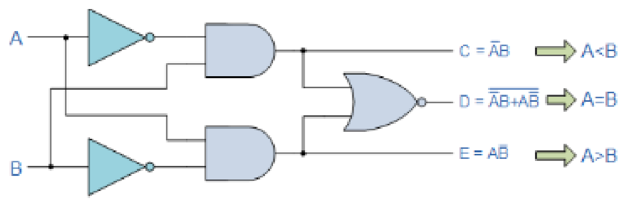
• If no 1-bit errors occur in transmission, the received data will have odd parity. But if the transmitted bits is changed by noise or any other disturbance, the received data will have even parity.

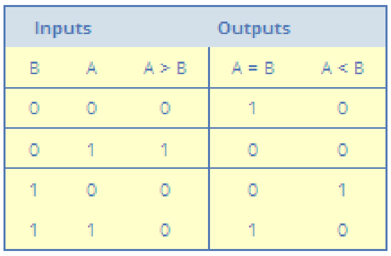
MAGNITUDE COMPARATOR

• It compares two n-bit binary numbers, say A and B and activates one of these 3 outputs: A=B, A>B and A<B (Figure: 4.37).

• The logic equations for the outputs can be written as follows, where G, L, E stand for greater than, less than and equal to respectively.

(A>B): G=AB' (A<B):L=A'B (A=B):E=A'B'+AB=(AB'+A'B)'=(G+L)'





Inputs		Outputs		
B	A	A > B	A = B	A < B
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

Figure 4.37: a)Block diagram of magnitude comparator b)Truth table c)Circuit for 1-bit comparator.

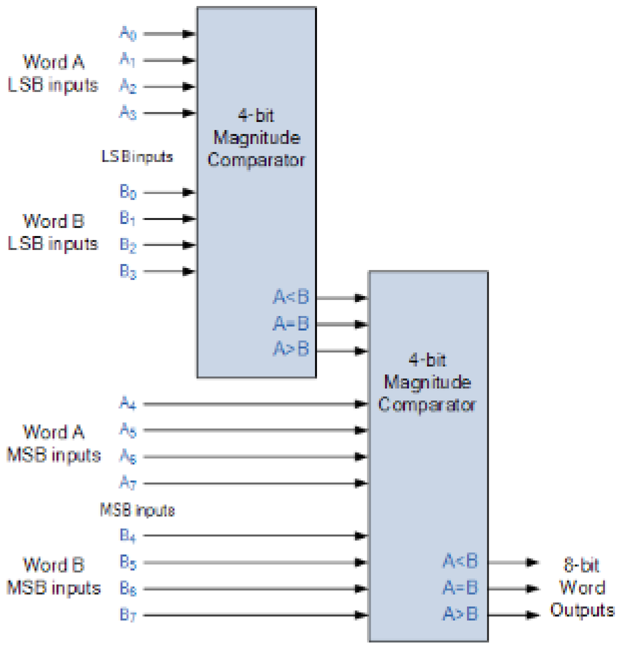
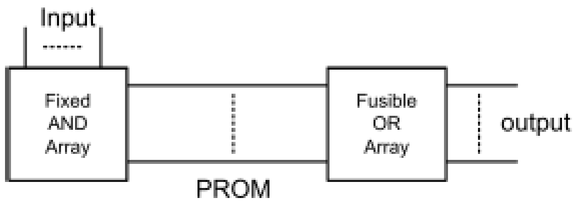


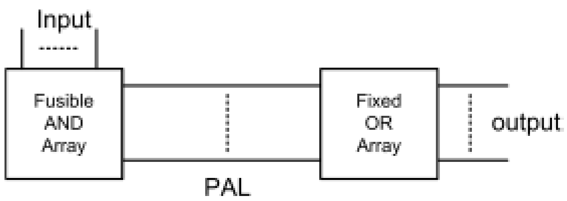
Figure 4.38: 8-bit comparator from two 4-bit comparators

READ-ONLY MEMORY

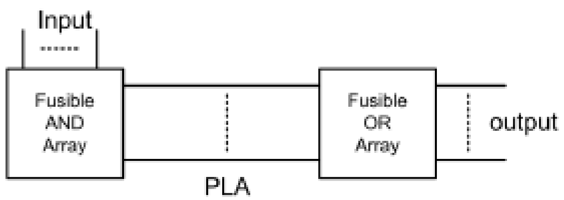
• It is an IC that can store thousands of binary numbers representing computer instructions and other fixed data.
• We can use a ROM instead of SOP circuits to generate any boolean function.
• The block diagrams of the three PLDs are as shown below:



PROM



PAL



PLA

PROGRAMMABLE ROMS

• This allows the user instead of the manufacturer to store the data. An instrument called a PROM programmer stores the words by "burning in".
• Programming like this is permanent because the data cannot be erased after it has been burned in.
Q: Why PROM is a Universal Logic Solution?
Ans: Because the AND gates generate all the fundamental products. The user can then OR these products as needed to generate any boolean output.

PROGRAMMABLE ARRAY LOGIC (PAL)

• This is a programmable array of logic gates on a single chip.
• This is different from a PROM because it has a programmable AND array and a fixed OR array.
• With a PROM programmer, we can burn in the desired fundamental products, which are then ORed by the fixed output connections.

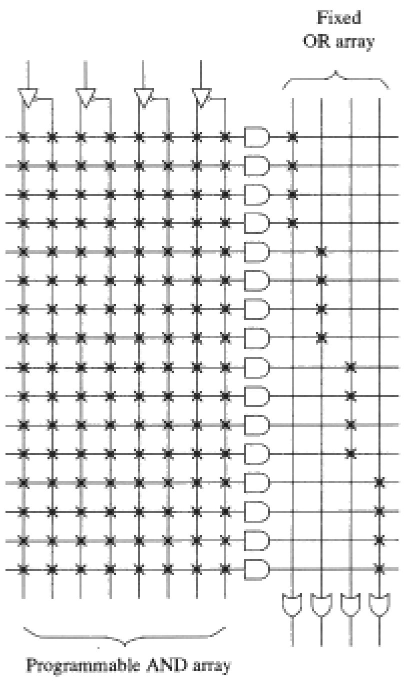


Figure 4.43: Structure of PAL

• Suppose we want to generate the following boolean functions. Show how we can program a PAL.

W = AB′C′ + CD
X = A′BC′ + A′CD + ACD′ + BCD
Y = A′C′D′ + ACD + A′BD

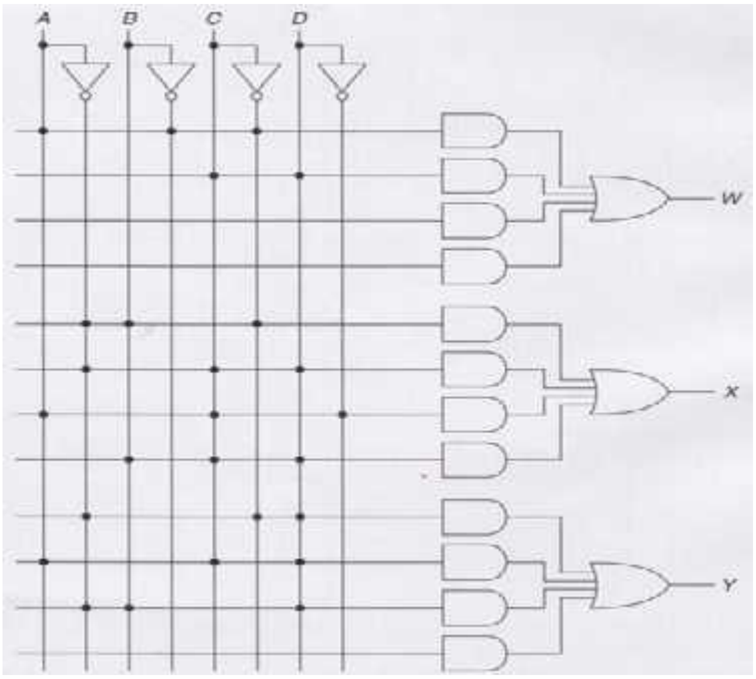


Figure 4.44: Example of programming a PAL

PROGRAMMABLE LOGIC ARRAYS (PLA)

• In this, the input signals are presented to an array of AND gates while the outputs are taken from an array of OR gates.
• In a PROM, the input AND gate array is fixed and cannot be altered, while the output OR gate array is fusible linked, and can thus be programmed. In PAL, the output OR gate array is fixed while the input AND gate array is fusible linked and thus programmable.
• The PLA is much more versatile than the PROM or the PAL, since both its AND gate array and its OR gate are fusible linked and programmable.
• PLA is also more complicated to utilize since the number of fusible links are doubled.

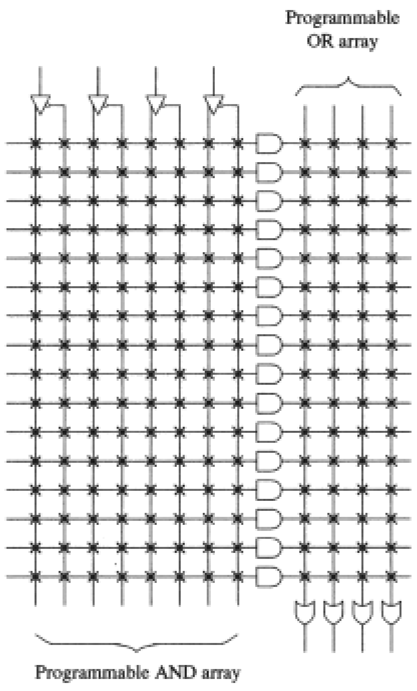


Figure 4.45: Structure of PLA

• Suppose we want to generate the following boolean functions. Show how we can program a PLA.
f(a,b,c) = a'b' + abc
g(a,b,c) = a'b'c' + ab + bc
h(a,b,c) = c

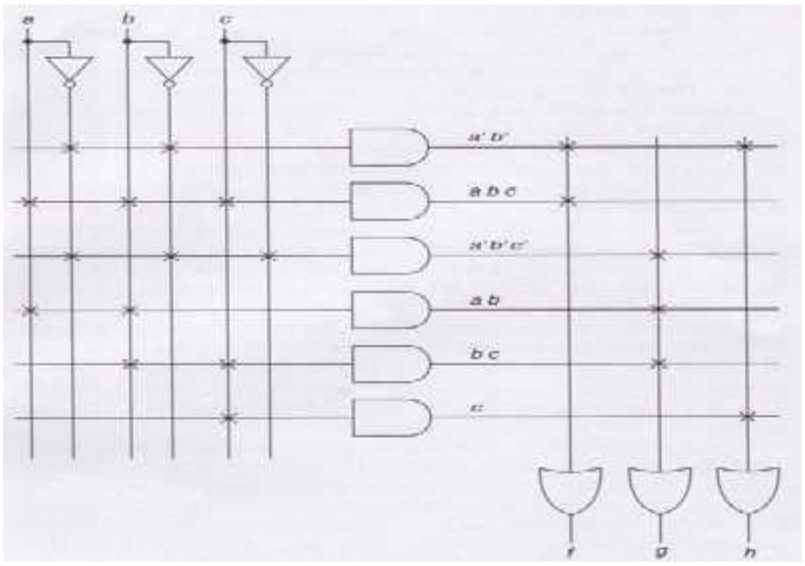


Figure 4.46: Example of programming a PLA

• Show how we can use a PLA to recognize each of the 10 decimal digits represented in binary form and to correctly drive a 7-segment display.

BCD Input				Outputs						
A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1

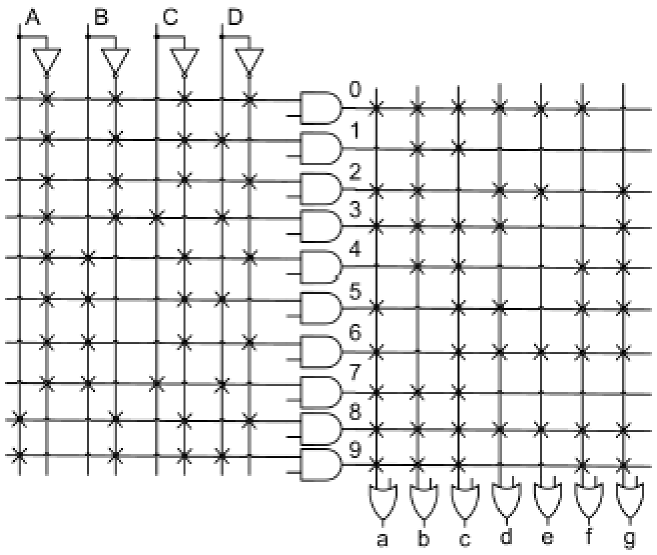


Figure 4.47: 7-segment decoder using PLA

• Write a verilog code to realize a 2:1 multiplexer for the given circuit

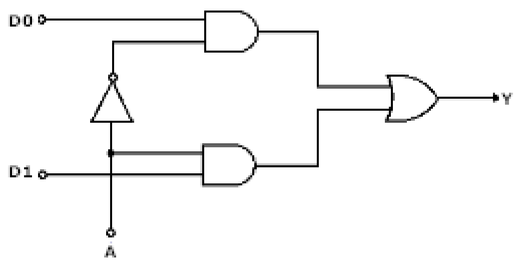


Figure 4.48:2 to 1 multiplexer

```
module mux2to1(A,D0,D1,Y);
input A,D0,D1
output Y;
assign Y=(~A&D0)|(A&D1);
endmodule
```

```
modue mux2to1(A,D0,D1,Y);
input A,D0,D1
output Y;
assign Y=A? D1: D0; //conditional assignment
endmodule
```

```
module mux2to1(A,D0,D1,Y);
input A,D0,D1
output Y;
reg Y;
always @ (A or D0 or D1)
if (A==1)
   Y=D1;
else
   Y=D0;
endmodule
```

```
module mux2to1(A,D0,D1,Y);
input A,D0,D1
output Y;
reg Y;
always @ (A or D0 or D1)
   case (A)
       0:Y=D0;
       1:Y=D1;
   endcase
endmodule
```

EXERCISE:

1. What is a multiplexer? Explain 4:1 multiplexer with logic block diagram and truth table? (4)
2. Explain 16:1 multiplexer (74150)? (4)
3. Why multiplexer is called universal logic circuit? Realize the boolean function F(A,B,C,D)=€m(0,2,3,4,5,8,9,10,13) using i) 8:1 mux ii) 4:1 mux (8)
4. What is a nibble multiplexer? Explain with block diagram? (3)
5. Design a 32:1 multiplexer using two 16:1 multiplexers and one 2:1 multiplexer? (4)
6. Construct 16:1 multiplexer using 4:1 and 2:1 multiplexer? (4)
7. Implement f(a,b,c,d)=€m(0,1,5,6,7,9,10,15) using i)8:1 mux with a,b,c as select lines
   ii)4:1 mux with a,b as select lines? (6)
8. Write a short note on demultiplexer? (5)
9. Show how two 1:16 demultiplexers can be connected to get a 1:32 demultiplexer? (5)
10. Write short note on i) decoder ii)encoder? (10)
11. Design a 4:16 decoder using two 3:8 decoder (74LS138)? (6)
12. Design a keypad interface to a digital system using 10-line BCD encoder (74LS147)? (6)
13. Give verilog HDL code for 4:1 multiplexer using conditional 'assign' and 'case' statements? (5)
14. Define an encoder? Design a priority encoder for the given truth table. The order of priority of inputs is X1>X2>X3 (6)
15. Construct a scheme to obtain a 4:16 line decoder using 74138(3:8 line decoder)? (5)
16. Describe the working principle of a 3:8 decoder? Realize the following boolean expressions using the 3:8 decoder:
    F1(A,B,C)=€m(1,2,3,4) F2(A,B,C)=€m(3,5,7) (6)
17. Write verilog code for a 4:1 Mux considering any model? (4)
18. Implement the following function using decoder
    F1(A,B,C)=€m(0,4,6) F2(A,B,C)=€m(0,5) F3(A,B,C)=€m(1,2,3,7) (6)
19. What is XOR gate? Explain 2-input XOR gate with logic symbol & truth table? (4)
20. What are even parity and odd parity? Give example for each? (3)
21. What are parity generator & parity checker? Explain both of them along with neat diagrams? What are applications of parity generator & checker? (8)
22. Explain the operation of 74180(parity generator-checker) along with truth table and logic diagram? (8)
23. For 9 bit odd-parity checker, what does Y equal for the following inputs?
    i)1111 00111 ii)1001 10111 iii)1000 11010 iv)1011 10010 (2)
24. What is magnitude comparator? Explain 1-bit magnitude comparator with logic diagram and truth table? (6)
25. Design a two bit magnitude comparator by writing the truth table, the relevant expression & logic diagram? (10)
26. Design a comparator to check if two n-bit numbers are equal. Configure this using cascaded stage of 1 bit equality comparator? (8)
27. What is PAL? Draw a 3-input & 3-output PAL circuit for following four Boolean functions
    Y2=ABC+ABC'+B'C' Y1=ABC'+AB'C+A'BC' Y0=A'BC'+ABC'+A'BC (6)
28. What is PLA? A combinational circuit is defined by the functions:
    F1=€m(3,5,7) F2=€m(4,5,7)
    Implement the circuit with a PLA having 3 inputs, 3 product terms and 2 outputs? (8)
29. Draw a 4-input & 4-output PAL circuit for following four Boolean functions
    Y3=ABCD'+A'BCD+A'BCD'+A'B'C'D Y2=ABC'D+AB'C'D'+A'B'CD
    Y1=A'B'C'D'+A'B'C'D+ABCD'+ABCD Y0=AB'C'D'+AB'C'D+ABC'D (6)
30. What is PLA? How does PLA differ from PAL? (3)
31. Explain 7-segment decoder using PLA along with block diagram? If ABCD=1001, then what segments of the in. Implement the following function using PLA
    X=A'B'C+AB'C'+B'C Y=A'B'C+AB'C' Z=B'C (8)
33. Implement the following boolean functions using an appropriate PLA:
    F1(A,B,C)=€m(0,4,7) F2(A,B,C)=€m(4,6) (4)
34. Show how data processing circuits(16:1 mux,8:1 mux, 4:16 decoder and 4-bit comparator) can be used to compare two 2-bit numbers A1A0 and B1B0 to generate two outputs, A<B and A=B? (10)

UNIT 5: REGISTERS

REGISTERS

• A register is a group of flip-flops used to momentarily store binary-information (e.g. 1101).
• Each flip-flop can store either 0 or 1.
• The flip-flops used to construct registers are usually edge-triggered JK, SR or D types.
• The register can also be used
→ to accept input-data from an alphanumeric keyboard and then present this data at the input of a microprocessor-chip.
→ to momentarily store binary-data at the output of a decoder.
→ to perform various arithmetic operations. For ex: multiplication & division.
→ to count number of pulses entering into a system as up-counter, down-counter, ring-counter or johnson-counter.
→ as serial-adder , sequence-generator and sequence-detector.
• UART(Universal Asynchronous Receiver Transmitter) is a chip used to exchange data in a microprocessor-system.

TYPES OF REGISTERS

• Shift-register is a group of flip-flops connected in such a way that a binary-number can be shifted into or out of the flip-flops.
• The bits in a binary-number can be moved from one place to another in following 2 ways:
→ Serial shifting: Data-bits are shifted one after the other in a serial fashion with one bit shifted at each clock transition (Figure: 9.1).
→ Parallel shifting: Data-bits are shifted simultaneously with a single clock transition.
• Shift-register types are
→ Serial in-Serial out (e.g. 7491, 8 bits)
→ Serial in-Parallel out (e.g. 74164, 8 bits)
→ Parallel in-Serial out (e.g. 74165, 8 bits)
→ Parallel in-Parallel out (e.g. 74198, 8 bits)

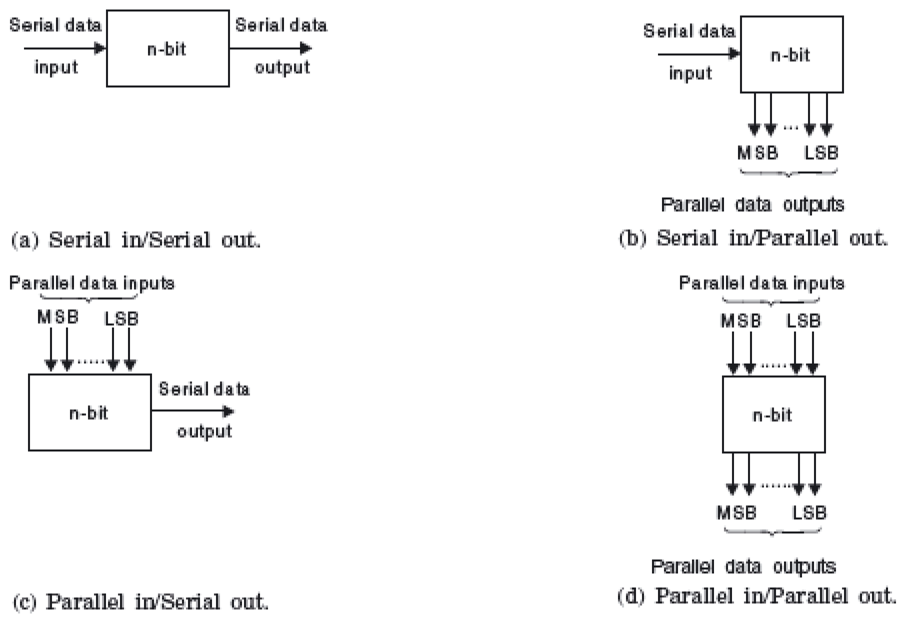


Figure: 9.1: Shift register types

SERIAL IN-SERIAL OUT (SISO)

SHIFTING BINARY-DATA INTO THE REGISTER

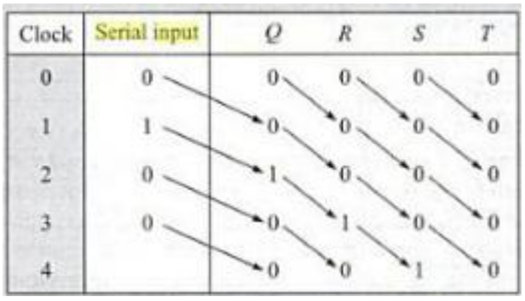
• A common clock provides trigger at its negative edge to all the flip-flops (Figure 9.2).
• The output of one D flip-flop is connected to input of the next.
• At every clock trigger, data stored in one flip-flop is transferred to the next flip-flop.
• Data-transfer takes place like this Q->R, R->S; S->T and serial data-input(D) is transferred to Q.
• Suppose that 4-bit number 0100 has to be loaded into the register. When clock signals are applied, following events happen:

At clock edge A,

→ flip-flop Q has serial data-input 0
→ flip-flop R has input 0 from output of Q
→ flip-flop S has input 0 from output of R and
→ flip-flop T has input 0 from output of S.

When clock triggers, these inputs get transferred to corresponding flip-flop outputs simultaneously so that QRST=0000.
Thus, at clock trigger, values at DQRS is transferred to QRST.
(note: before entering binary number, by default all flip-flops are cleared, that's why initial value of QRST=0000).

At clock edge B, serial data-in D=0, i.e. DQRS=0000. So after NT at B, QRST=0000.
At clock edge C, serial data-in D=1, i.e. DQRS=1000. So after NT at C, QRST=1000.
At clock edge D, serial data-in D=0, i.e. DQRS=0100. So after NT at D, QRST=0100.



Clock	Serial input	Q	R	S	T
0	0	0	0	0	0
1	1	0	0	0	0
2	0	1	0	0	0
3	0	0	1	0	0
4		0	0	1	0

Figure 9.3: Data transfer through serial input in a shift register

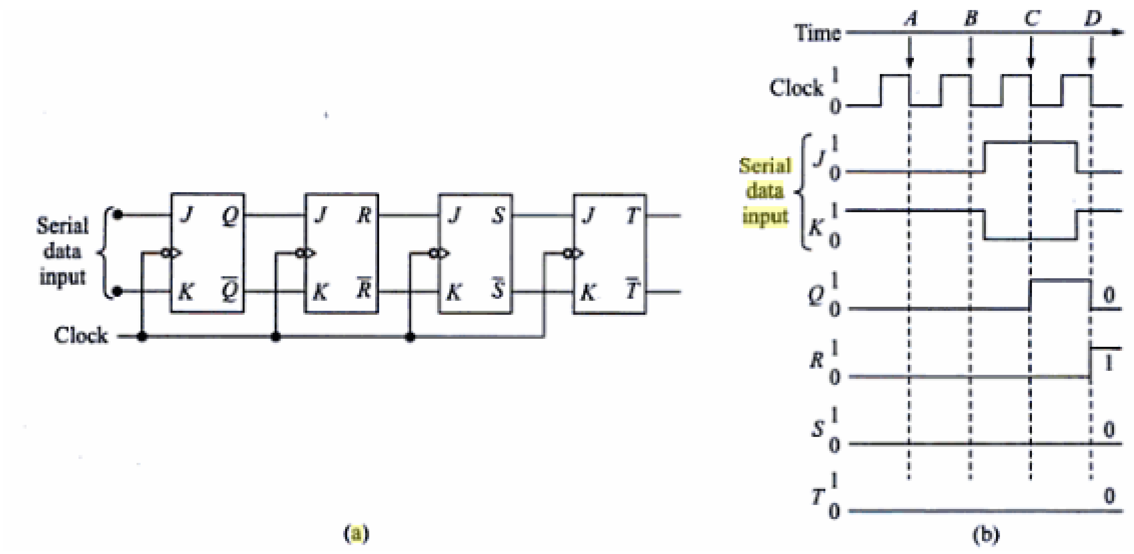


Figure 9.2: 4-bit serial input shift register

SHIFTING BINARY-DATA OUT OF THE REGISTER

• Suppose that the register has the 4-bit number QRST=1010 stored in it.
• When clock signals are applied, following events happen:

Before Time A: The register holds the bits QRST=1010.The LSB 0 appears at T.
At Time A: The bits are all shifted one flip-flop to the right. A 0 is shifted into Q and the LSB is shifted out the right-end and lost. The second LSB 1 appears at T. The register holds the bits QRST=0101.
At Time B: The bits are all shifted one flip-flop to the right. A 0 is shifted into Q. The third LSB 0 appears at T. The register holds QRST=0010.
At Time C: The bits are all shifted one flip-flop to the right. A 0 is shifted into Q. The MSB 1 appears at T. The register holds QRST=0001.
At Time D: The MSB is shifted out the right end and lost. A 0 is shifted into Q. The registers holds QRST=0000. Thus, the binary-data stored is shifted out of the right-end of the register in a serial fashion and lost after 4 clock cycles.

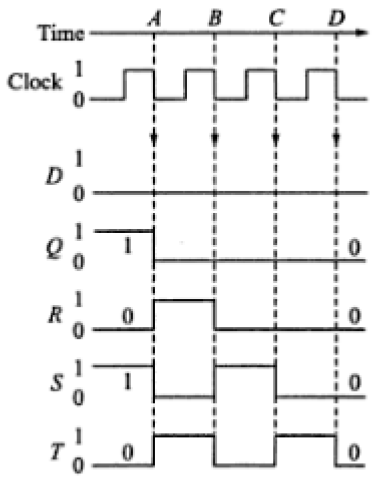


Figure 9.4: Waveform for shifting 1010 out of register

7491 8-BIT SHIFT REGISTER

• As shown in logic diagram, 7491 register has eight RS flip-flops connected to provide a serial input as well as a serial output.
• The clock input at each flip-flop is negative-edge-trigger-sensitive. However, since the applied clock signal is passed through an inverter, data will be shifted on the positive-edges of the input clock pulses.
• The inverter connected between R and S on the first flip-flop means that this circuit functions as a D-type flip-flop.
• The data input is applied at either A or B.
• A data level (0 or 1) at A is complemented by the NAND gate and then applied to the R input of the first flip-flop.
• On a positive clock transition, a 1 at input will set the flip-flop i.e. this 1 is shifted into the first flip-flop.
• The NAND gate with inputs A and B simply provides a gating function for the input data stream if desired. If gating is not desired, simply short A & C together and apply the input data stream to this connection.

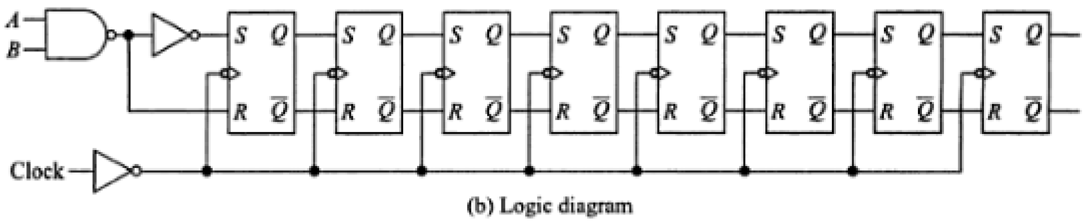


Figure 9.5:74191 8-bit shift register

SERIAL IN-PARALLEL OUT (SIPO)

• Binary-data is shifted into register serially, but shifted out in parallel.

• The 74164 is an 8-bit SIPO shift-register. It is constructed by using RS flip-flops having clock inputs that are sensitive to NTs (Figure 9.7).

• 74164 register is similar to 7491 with two exceptions:

i) All 8 bits of any number stored in the register are available simultaneously as an output.

ii) Each flip-flop has an asynchronous clear-input. A low level at the clear-input will immediately reset all flip-flops low(0000 0000). The clear-input is asynchronous i.e. it can be done at any time.

• Suppose that the serial-data is connected to A, then B can be used as a control-line. Here's how it works:

B is held high: The NAND gate is enabled and the serial input-data passes through the NAND gate inverted. The input-data is shifted serially into the register.

B is held low: The NAND gate output is forced high, the input-data stream is inhibited, and the next positive clock transition will shift a 0 into the first flip-flop. Each succeeding positive clock transition will shift another 0 into the register. After 8 clock pulses, the register will be full of zeros ($Q_AQ_BQ_CQ_D$ $Q_EQ_FQ_GQ_H$=0000 0000).

• As shown in waveform, the serial-data is input at A while the control-signal is applied at B.

• The first CLEAR pulse occurs at time A and resets all flip-flops to 0.

• The clock begins at time B, but the first PT does nothing since the control-line is low. At time C, the control-line goes high, and the first data bit(a 0) is shifted into the register at time D.

• The next 7 data bits are shifted in, in order, at times E,F,G,H,I,J & K. The clock remains high after time K, and the 8-bit number 0010 1100 now resides in the register and is available on the eight output line.

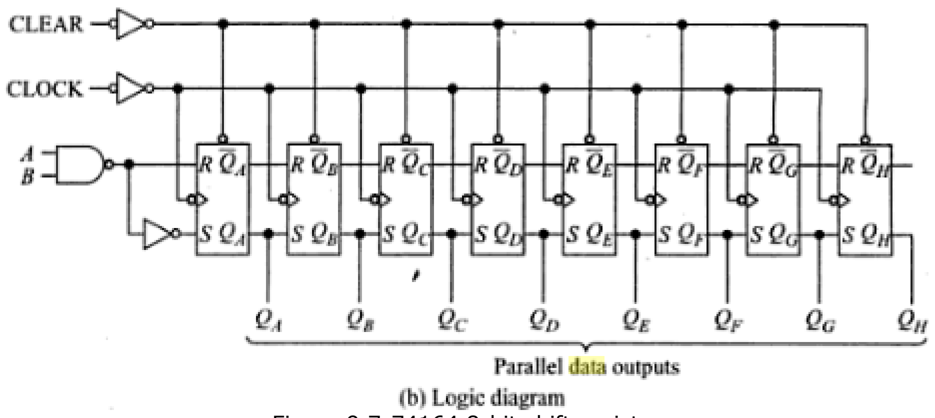

CLEAR
CLOCK
A
B
R $\overline{Q}_A$ S Q_A
R $\overline{Q}_B$ S Q_B
R $\overline{Q}_C$ S Q_C
R $\overline{Q}_D$ S Q_D
R $\overline{Q}_E$ S Q_E
R $\overline{Q}_F$ S Q_F
R $\overline{Q}_G$ S Q_G
R $\overline{Q}_H$ S Q_H
Q_A Q_B Q_C Q_D Q_E Q_F Q_G Q_H
Parallel data outputs
(b) Logic diagram

Figure 9.7:74164 8-bit shift register

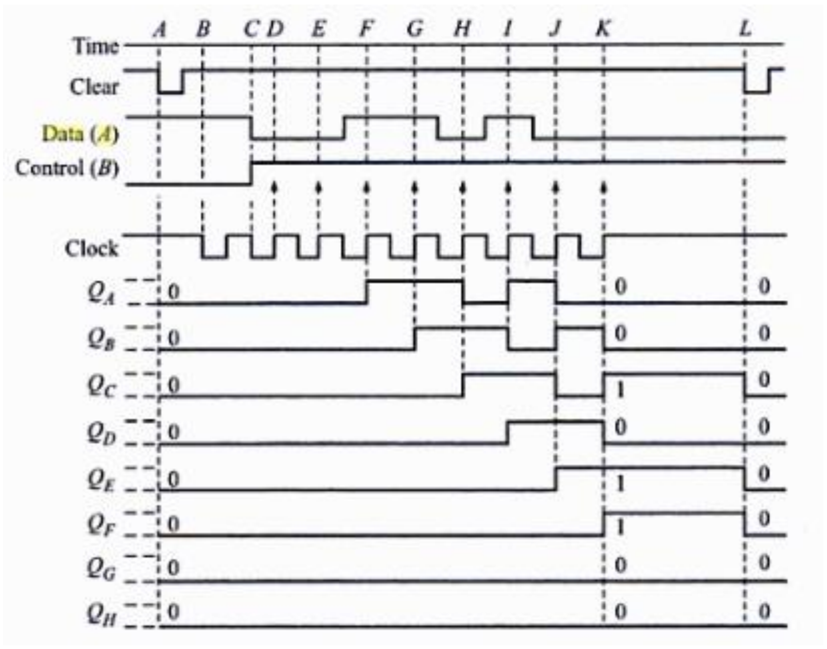


Figure 9.9:Waveform for entering data 00110100 into register

PARALLEL IN-PARALLEL OUT (PIPO)

• Here, data can be shifted either into or out of the register in parallel (Figure 9.13).
• Each flip-flop is negative-edge-triggered, and thus a PT will shift data into the register.
• The 6 data bits, D_1 through D_6 are all loaded into the register in parallel.
• The loaded-data is immediately available in parallel, at the outputs, Q_1 through Q_6.
• Since this type of register is used to store data, it is also called a data-latch.
• It is not possible to shift stored data either to the right or to the left.
• A low level at the clear input will immediately reset all flip-flops low.
• The clear input is asynchronous ie it can be done at any time.

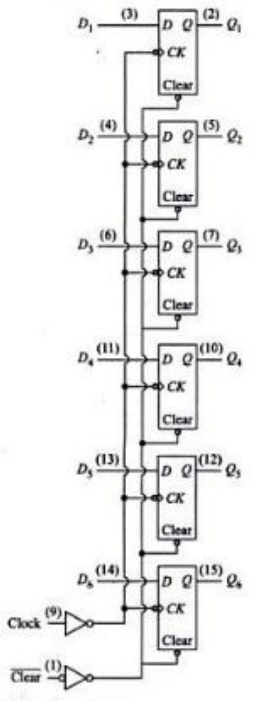


Figure 9.13: 74174

4-BIT PARALLEL ACCESS SHIFT REGISTER (54/7495A)

• It can be used for entering/removing data in parallel (Figure: 9.15).
• It can also be used to shift data to the right(from Q_A->Q_B) and to the left(from Q_B->Q_A).
• The parallel data-outputs are simply the Q sides of each of the four flip-flops in the register.
• If mode control line=HIGH, the AND gates on the right input to each NOR gate is enabled while the left AND gates are disabled. The data inputs, A, B, C and D will then be loaded into the register on a negative transition of the clock (this is parallel data input).
• If mode control line=LOW, the AND gate on the right input to each NOR gate is disabled while the left AND gate is enabled. On each clock NT, a data bit is entered serially into the register at the first flip-flop QA and each stored data bit is shifted one flip-flop to the right. This is the serial input of data, and also the right-shift operation.
• There are two clock inputs-clock 1 and clock 2.

Clock 1 is used for right-shift operation

while Clock 2 is used for left-shift operation.

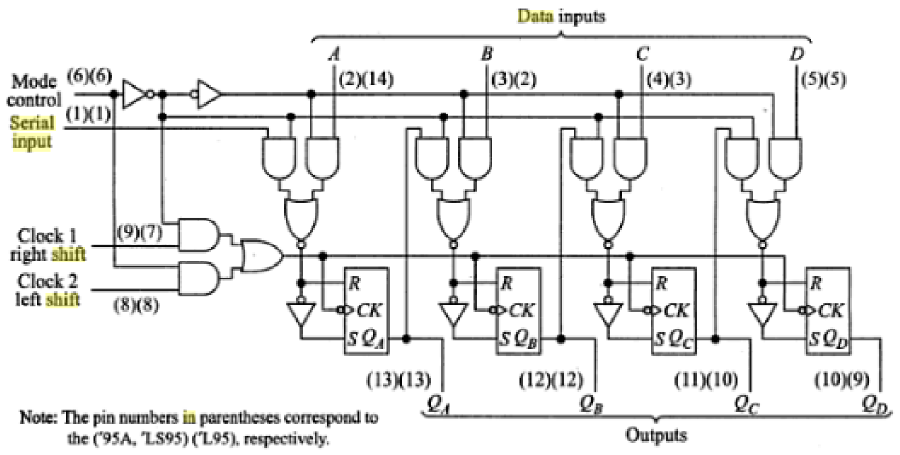


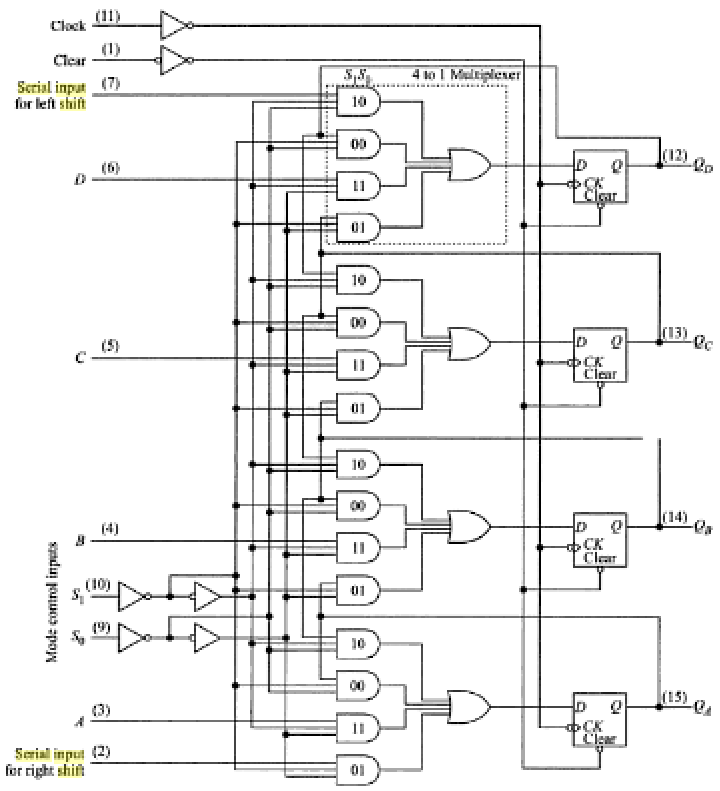
Figure 9.15:54/7495A

UNIVERSAL SHIFT REGISTER(74195)

• Four basic types of shift register are: i)serial in serial out, ii)serial in parallel out, iii)parallel in serial out and iv)parallel in parallel out. Serial in or serial out can be made possible by shifting data in any of the two directions, left shift and right shift. A universal shift register can perform all the 4 operations and is also bidirectional in nature (Figure 9.18).

• In 74194 register, there are 2 separate inputs for serial data for left and right shift.

• In addition, there are two mode control inputs which select the mode of operation for the universal shift register according to below table 9.1.



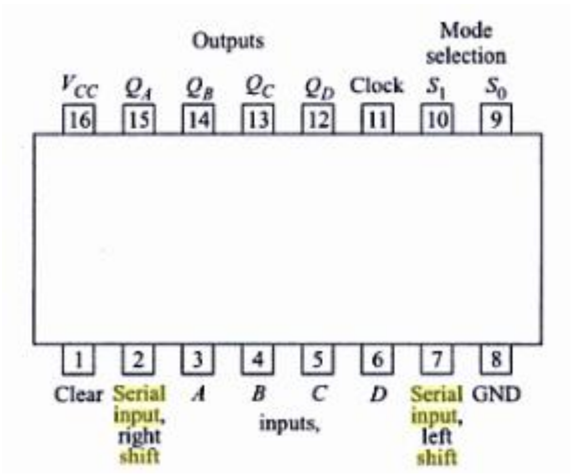


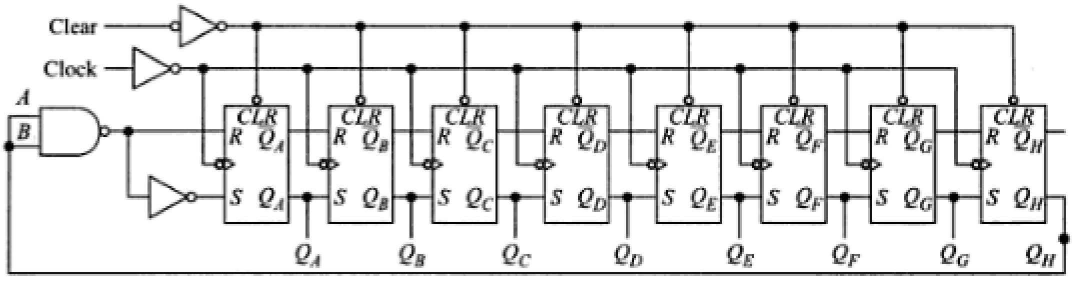
Figure 9.18:74194 pinout

Table 9.1: Function table of 74194

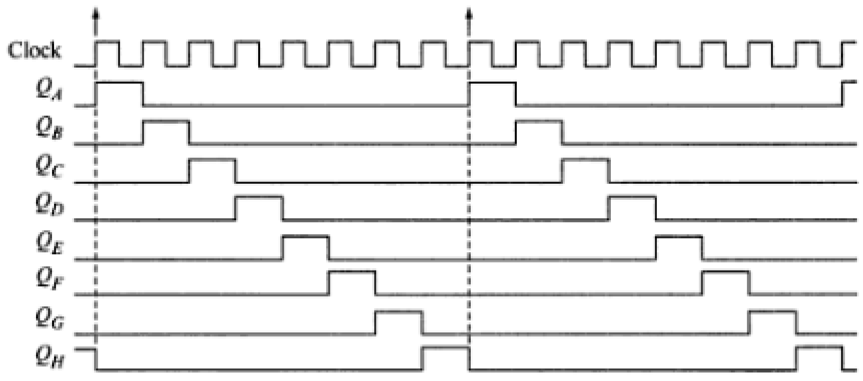
Mode	Control	Function	Next State (n+1-th state)			
S_1	S_0		$Q_{A,n+1}$	$Q_{B,n+1}$	$Q_{C,n+1}$	$Q_{D,n+1}$
0	0	Hold	$Q_{A,n}$	$Q_{B,n}$	$Q_{C,n}$	$Q_{D,n}$
0	1	Shift right	Data in (Pin 2)	$Q_{A,n}$	$Q_{B,n}$	$Q_{C,n}$
1	0	Shift left	$Q_{B,n}$ (Pin 7)	$Q_{C,n}$	$Q_{D,n}$	Data in
1	1	Load	A	B	C	D

APPLICATIONS OF SHIFT REGISTER
RING COUNTER

- It is a basic shift register with direct feedback such that
  the contents of the register simply circulate around the register when the clock is running.
- Suppose that Q_A is high and all other flip-flops are low.
  On very first clock PT, the 1 in A will shift into B and A will be reset, since the 0 in H will shift into A.
- All other flip-flops will still contain 0s.
- The second clock pulse will shift the 1 from B to C, while B resets.
- The third clock PT will shift the 1 from C to D, and so on. Thus this single 1 will shift down the register, traveling from one flip-flop to the next flip-flop each time the clock goes high. When it reaches flip-flop H, the next clock PT will shift it into flip-flop A by means of the feedback connections.
- The ping-pong balls simply circulate around the register in a clockwise direction, moving ahead one flip-flop with each clock PT. This configuration is frequently referred to as a circulating register or a ring counter.



(a) 54/74164 8-bit shift register with feedback line from Q_H to A-B

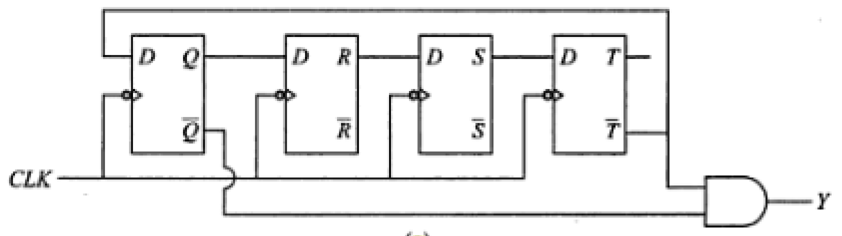


(b) Waveforms when register has a single one, and seven zeros

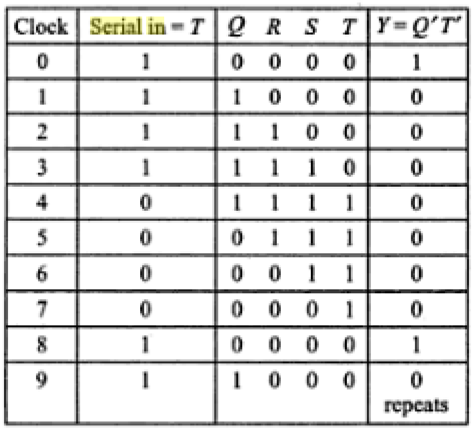
Figure 9.20: Ring Counter (a)74164 8-bit shift register with feedback line from Q_H to A-B
(b)Waveform when register has a single 1 and seven 0s

JOHNSON COUNTER (SWITCHED TAIL COUNTER)

• It is a shift register with inverting output of last flip-flop fed to first flip-flop input (Figure 9.22).
• Consider 4-bit counter. Initially, all the flip-flops are cleared. i.e. QRST=0000.
• When first clock trigger occurs, flip-flop stores QRST=1000. The output of last stage T is 0. Therefore, complement output of last stage is 1.This is connected back to the D input of first stage. So, D=1.
• When second clock trigger occurs, flip-flop stores QRST=1100.
• When third clock trigger occurs, flip-flop stores QRST=1110.
• For any N-bit shift register, johnson counter can count up to 2N number of clock pulse and gives modulo-2N counter.
• The output Y, derived by AND operation of first and last flip-flop inverting outputs gives a logic high at every 2N-th clock cycle. This two-input AND gate which decodes states repeating in the memory units to generate output that signals counting of a given number of clock pulses is called decoding gate.



(a)



Clock	Serial in = T	Q	R	S	T	Y = Q′T′
0	1	0	0	0	0	1
1	1	1	0	0	0	0
2	1	1	1	0	0	0
3	1	1	1	1	0	0
4	0	1	1	1	1	0
5	0	0	1	1	1	0
6	0	0	0	1	1	0
7	0	0	0	0	1	0
8	1	0	0	0	0	1
9	1	1	0	0	0	0 repeats

(b)

Figure 9.22: (a) 4 bit Johnson Counter b)it's state table

SEQUENCE GENERATOR AND SEQUENCE DETECTOR

Sequence Generator

• This is used to generate a prescribed-sequence repetitively (Figure 9.23a).
• Shift register can be represented as pipe full of data and each flip-flop represents one compartment of it.
• The leftmost flip-flop is connected to serial data-in and rightmost provides serial data-out.
• When clock trigger occurs, data-transfer takes place.
• The shift-register is connected like a ring-counter and with triggering of clock; the binary-word stored in the register comes out sequentially from serial out but does not get lost as it is fed back as serial-in to fill the register.

Sequence Detector

• This is used to check binary-data stream and generate a signal when a particular sequence is detected (Figure 9.23b).
• This has one register to store the binary-word, we want to detect from the data stream.
• Input-data stream enters a shift-register as serial data-in and leaves as serial-out.
• At every clocking instant, bit-wise comparisons of these 2 registers are done through Ex-NOR gate.
• 2-input Ex-NOR gives logic HIGH when both inputs are low or both of them are high.
• The final output is taken from a 4-input AND gate, which becomes 1 only when all its inputs are 1 i.e. all the bits are matched.
• This is also programmable i.e. if we want to change the binary-word to be detected, we simply load that in the bottom register.

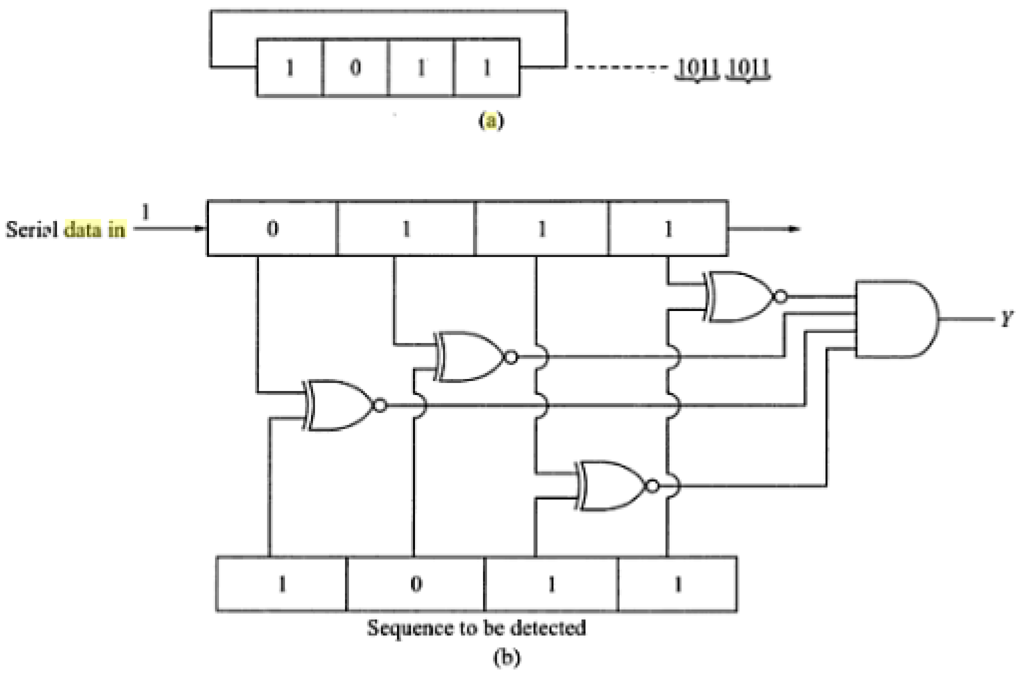


Figure 9.23: (a)4-bit sequence generator (b)4 bit sequence detector

SERIAL ADDER

• This converts parallel data to serial and uses full-adder(FA) block sequentially (Figure 9.24).
• Two 8-bit numbers to be added are loaded in two 8-bit shift registers A & B.
• The LSB of each number appears in the rightmost position in two registers.
• Serial-data out of A and B are fed to data inputs of full-adder. The carry-in is fed from its own carry output delayed by one clock period by a D flip-flop.
• The sum(S) output of FA(full adder) is fed to serial data-in of Shift Register A.
• Both registers and D flip-flop are triggered by same clock.
• The addition takes place like this:

→ The LSBs of two numbers(A0 and B0) appearing at serial out of respective registers are added by FA during 1st clock cycle and generate sum(S1) and carry(C1).
→ S1 is available at serial data input of register A and C1 at input of D flip-flop.
At NT of clock, registers shift its content to right by one unit.
→ S1 becomes MSB of register A and C1 appears at D flip-flop output.

• Ths process goes on and is stopped by inhibiting the clock after 8 clock cycles.
• Drawback: The final addition result is delayed by 8 clock cycles. Solution: Using a high frequency clock the delay factor can be reduced considerably.

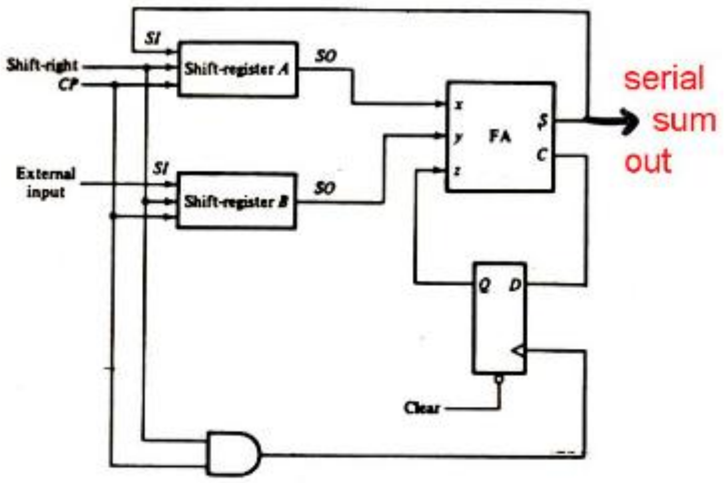


Figure 9.24: Serial Addition of two 8-bit numbers

Note: A power-on-reset circuit is used to preset flip-flops to any desired states.

Verilog code for a shift register of 5 bits constructed using D flip-flops

```
module Reg74174(D, Clock, Clear, Q)
    input Clock, Clear;
    input [5:0] D;
    output [5:0] Q;
    reg [5:0] Q;
    always @ (negedge Clock or negedge Clear)
    if(~Clear)
       Q=6'b0;              //Q stores 6 binary 0
    else
       Q=D;
endmodule
```

Verilog code for 4-bit serial input shift register

```
module ShiftReg(D, Clock, T);
    input Clock, D;
    output T;
    reg T;
    reg Q,R,S;
    always @ (negedge Clock)
    begin
      Q<=D;
      R<=Q;
      S<=R;
     T<=S;
    end
endmodule
```

Verilog code for 4-bit parallel out right shift register

```
module ShiftRegister(D, Clock D)
    input Clock, D;
    output [3:0] Q;
    always @ (negedge Clock)
    begin
      Q[0]<=D;
      Q[1]<=Q[0];
      Q[2]<=Q[1];
      Q[3]<=Q[2];
    end
endmodule
```

Verilog code for Johnson counter

```
module JohnsonC(Clock,Clear,Y);
    input Clock,Clear;
    output Y;
    reg Q,R,S,T;
    assign Y=(~Q) & (~T);
    begin
    if(~Clear)
      Q=6'b0;
    else
      begin
        Q<=~T;          //tail is switched and connected to input
        R<=Q;
        S<=R;
        T<=S;
      end
endmodule
```

EXERCISE:

1>Write a short note on register? What is UART? (4)
2>Name & explain in short the four basic types of shift registers and draw a block diagram for each? (6)
3>Explain the working of 4-bit SISO shift register with logic diagram, timing diagram & truth table? (8)
4>Draw the waveform to shift the binary number 1010 into 4-bit serial input shift register? (2)
5>Explain the working of 7491 8-bit shift register with logic diagram? (6)
6>Explain the working of 8-bit SIPO shift register (74164) with logic diagram? (10)
7>How long will it take to shift an 8-bit binary number into the 8-bit shift register (54/74164) if the clock is
i) 1 Mhz ii)5 Mhz? (2)
8>Explain the working of PIPO shift register (74174) with logic diagram? (6)
9>Explain the working of 4-bit parallel access shift register (7495A) along with logic diagram? (10)
10>What are universal shift register (74194)? Explain along with function table? (4)
11>State the applications of shift registers? (4)
12>What is a ring counter (or circulating register)? Explain working of 8-bit ring counter with logic diagram & truth table? (8)
13>What is a Johnson counter? Explain the working of 4-bit Johnson counter with logic diagram & truth table? (8)
14>Show how mod-8 Johnson counter works if it is initialized with '1001'. How to decode this counter? (6)
15>Distinguish between a ring counter and a Johnson counter? (4)
16>What are sequence generator and sequence detector? Explain both with neat diagram? (8)
17>Design a sequence detector that receives binary data stream at its input X and signals when a combination "1011" arrives at the input by masking its output Y high which otherwise remains low. Consider data is coming from left, that is, the first bit to be identified is 1, second is 1, third is 0 from input sequence? (6)
18>What is a serial adder? Explain with neat diagram? (6)
19>What is the difference between <= and = operator? (2)
20>Design an 8-bit sequence generator that generates the sequence 11000100 repetitively using shift register (i.e. ring counter, mod-8 Johnson counter)? (10)
21>Write verilog code for a shift register of 6 bits constructed using D flip-flops? (3)
22>Write verilog code for Johnson counter? (4)
23>Write verilog code for 4-bit serial input shift register? (4)
24>Write verilog code for 4-bit parallel out-right shift register? (4)

UNIT 6: COUNTERS

COUNTER

• This is a digital-circuit designed
→ to keep track of a number of events or
→ to count number of clock cycles.
• This can be constructed using a no. of flip-flops(e.g. JK, SR, D and T) & additional electronic circuits.
• This is similar to a register, since it is capable of storing a binary number (e.g. 1011).
• The input to the counter is rectangular waveform labeled CLOCK.
• Each time the clock signal changes state from low to high (0->1), the counter will add one (1) to the number stored in its flip-flops. In other words, the counter will count the number of clock transitions from low to high.
• Since the clock pulses occur at known intervals, the counter can also be used as an instrument for measuring time and therefore period(T) or frequency(f).
• There are two different types of counters-
i) Asynchronous counter: In this, each flip-flop is triggered by the output of the previous flip-flop i.e. the output of a flip-flop is used as the clock-input for the next flip-flop and
ii) Synchronous counter: In this, all flip-flops change states simultaneously since all clock inputs are driven by the same clock. ie all the flip-flops change states in synchronism.

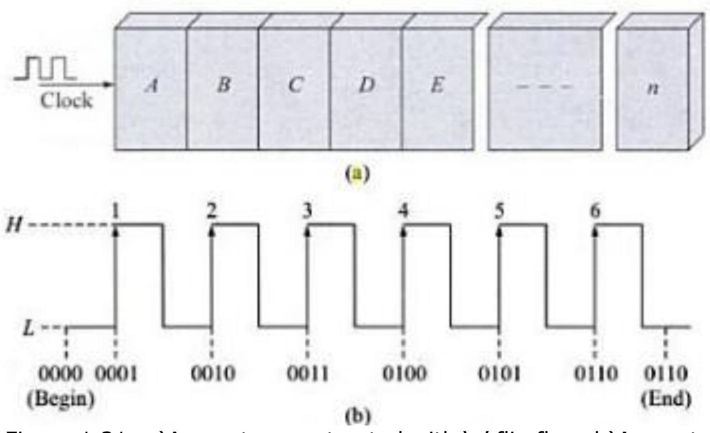


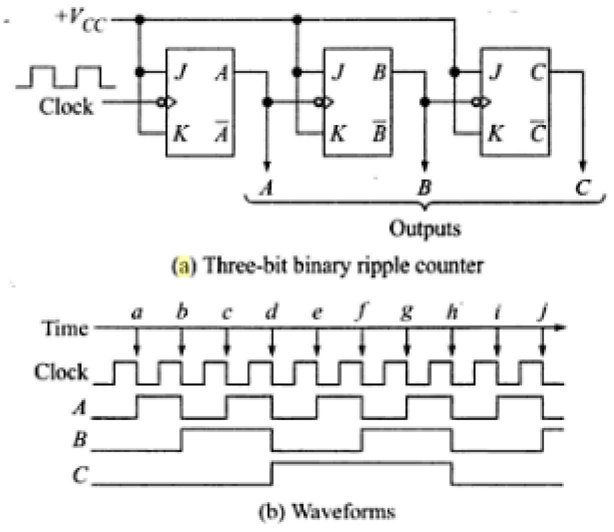
Figure 1.21: a)A counter constructed with 'n' flip flops b)A count of 6

Note:

- A counter having 'n' flip-flops will have 2^n output conditions (states).
- A 3-flip-flop counter is referred to as a mod-8 counter, since it has 8 states.
- Negative-clock-transition(NCT) means when the clock-pulse is going from 1 to 0(falling edge of the pulse). Positive-clock-transition (PCT) means when the clock-pulse is going from 0 to 1(rising edge of the pulse).
- A synchronous circuit incorporates a system-clock which generates periodic clock pulses. These pulses are distributed throughout the system such that the memory element ie flip-flops get affected only on the arrival of these pulses.
- If we consider a basic flip-flop to be a mod-2 counter, we see that a mod-4 counter is simply two mod-2 counters in series. Similarly, a mod-8 counter is simply a 2*2*2 connection.
- A higher-modulus counters can be formed by using the product of any number of lower-modulus counters. For example, mod-6 counter can be formed by using the mod-3 counter in conjunction with a flip-flop
- Present state means state of the flip-flops before the occurrence of a clock pulse. Next state means state of the flip-flops after the occurrence of the clock pulse.

ASYNCHRONOUS 3-BIT UP-COUNTER (RIPPLE-COUNTER)

• In this, each flip-flop is triggered by the output of the previous flip-flop i.e. the output of a flip-flop is used as the clock-input for the next flip-flop.
• A 3-bit ripple counter can be used to count the number of clock transitions up to a maximum of 7.
• This can be constructed using clocked JK flip-flops.
• The J and K inputs of all flip-flops are maintained HIGH by connecting them to V_{cc}. This means that each flip-flop will change state (toggle) with a negative-clock-transition(NCT).
• The system-clock is used to drive flip-flop A.
The output of flip-flop A is used to drive flip-flop B
Likewise, the output of flip-flop B is used to drive flip-flop C.
• Flip-flop A toggles with each negative-clock-transition.
Flip-flop B will toggle each time A goes *LOW.*
Likewise, flip-flop C will toggle each time B goes *LOW.*
• Overall propagation delay time=€(individual flip flops delay)



(a) Three-bit binary ripple counter

(b) Waveforms

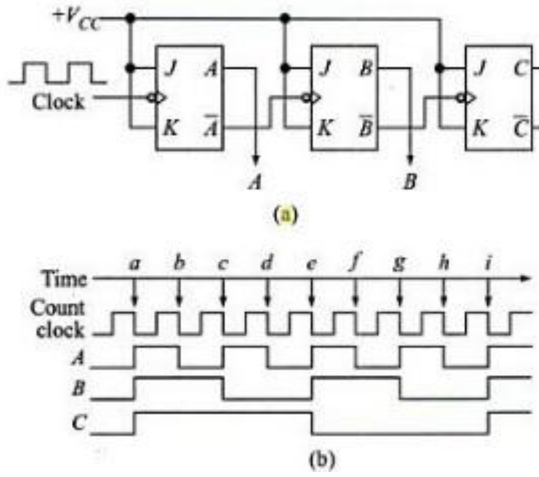
Negative clock transitions	C	B	A	State or count
---	0	0	0	0
a	0	0	1	1
b	0	1	0	2
c	0	1	1	3
d	1	0	0	4
e	1	0	1	5
f	1	1	0	6
g	1	1	1	7
h	0	0	0	0

(c) Truth table

Figure 10.1: a) 3-bit ripple counter b) waveforms c)truth table

ASYNCHRONOUS 3-BIT DOWN-COUNTER

• The down counter can be used to count downward from a maximum-count to 0 (Figure: 10.4).
• The system-clock is used to drive flip-flop A.,
But, the complement of A (i.e. A') is used to drive flip-flop B,
Likewise B' is used to drive flip-flop C.
• Flip-flop A toggles with each negative-clock-transition.
But flip-flop B will toggle each time A goes *HIGH*.
• Each time A goes high, A' goes low, and it is this negative-clock-transition on A that triggers B.
• The counter contents are reduced by one count with each clock transition.



Count	C	B	A
7	1	1	1
6	1	1	0
5	1	0	1
4	1	0	0
3	0	1	1
2	0	1	0
1	0	0	1
0	0	0	0
7	1	1	1

(c)

Figure 10.4:a) 3 bit down counter b) waveforms c)truth table

DECODING GATE

• It is a logic gate whose output is high(or low) only during one of the unique states of a counter.
• For instance, the decoding gate connected to the 3-bit ripple counter in Figure 10.6a will decode state-7(CBA=111).Thus the gate output will be HIGH only when A=1,B=1 and C=1.

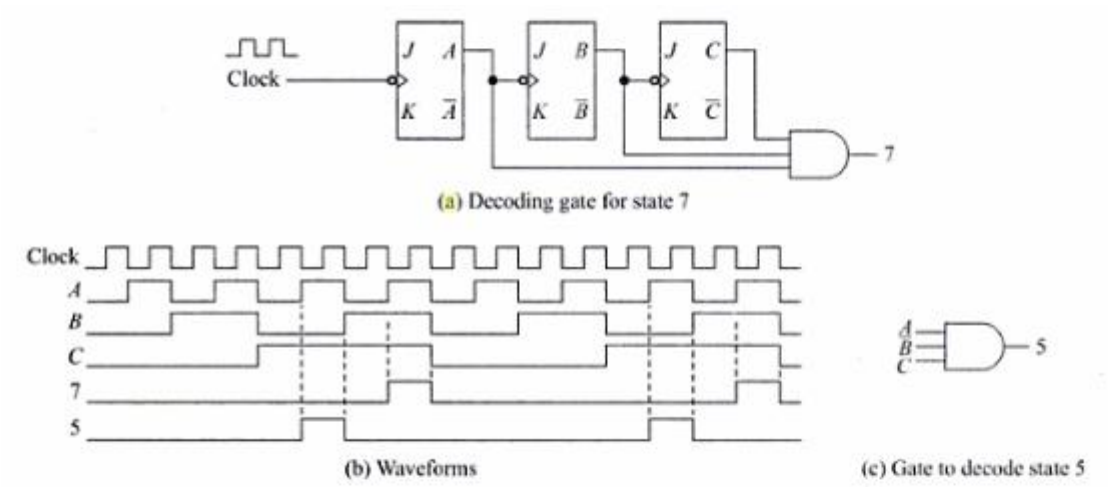


Figure 10.6: a)decoding gate for state 7 b)waveforms c)gate to decode state 5

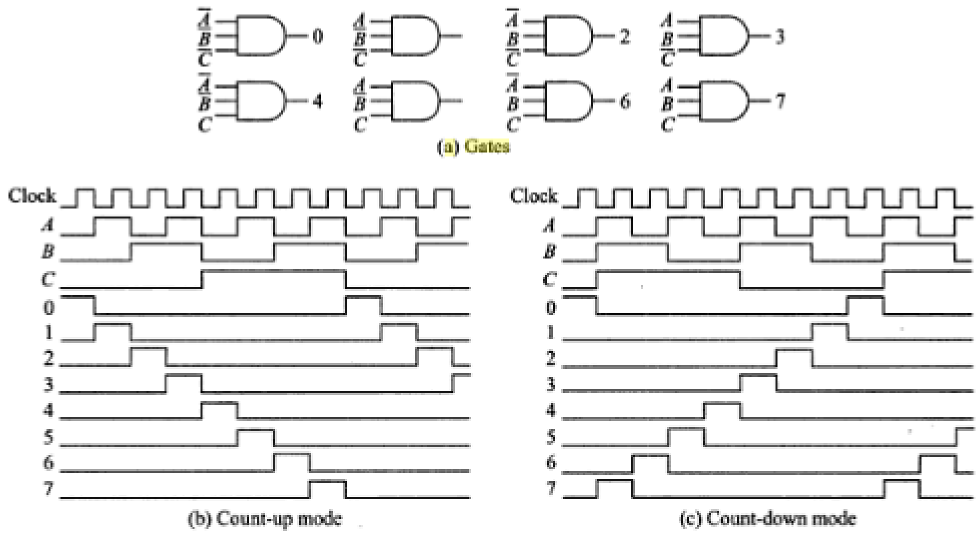


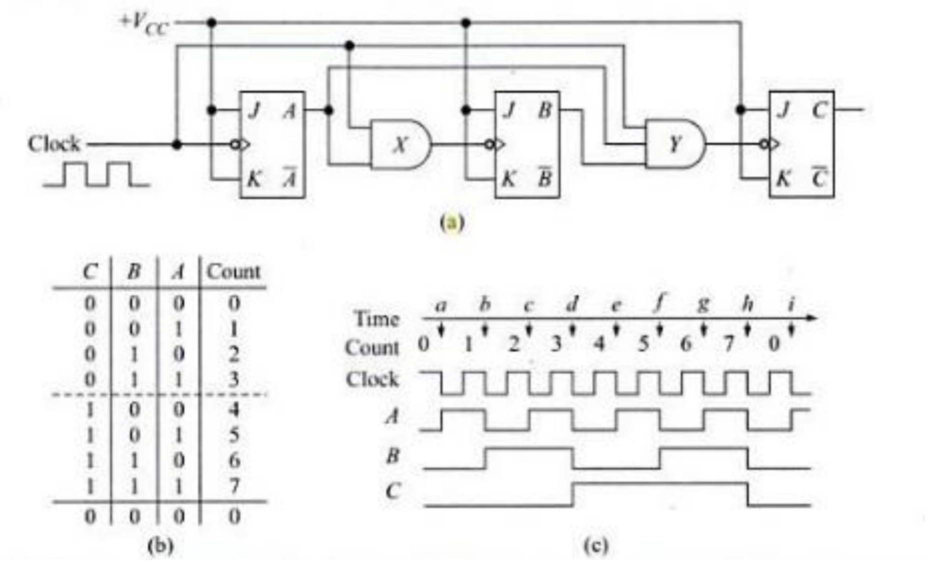
Figure 10.7: Decoding gates for a 3-bit binary ripple counter

DRAWBACKS IN ASYNCHRONOUS COUNTER

• There is a limit to its highest operating frequency.
• Each flip-flop has a delay time. These delay times are additive, and the total "settling" time for the counter is approximately the delay time times the total number of flip-flops.
• There is the possibility of glitches occurring at the output of decoding gates used with a counter.

SYNCHRONOUS 3 BIT UP-COUNTER (PARALLEL BINARY COUNTER)

• In this, all flip-flops change states simultaneously since all clock inputs are driven by the same clock. ie all the flip-flops change states in synchronism.
• A 3-bit ripple counter can be used to count the number of clock transitions up to a maximum of 7.
• This can be constructed using clocked JK flip-flops.
• The J and K inputs of all flip-flops are maintained HIGH by connecting them to V_{cc}. This means that each flip-flop will change state (toggle) with a negative-clock-transition(NCT).
• This logic configuration is referred to as *"steering logic",* since the clock pulses are steered to each individual flip-flop.
• The clock is applied directly to the flip-flop A, as result flip-flop A will change state with each negative-clock-transition.
• Whenever A is high, AND gate X is enabled and hence it transmits the clock pulse to flip-flop B.
Whenever A & B are high, AND gate Y is enabled & hence it transmits the clock pulse to flip-flop C.



C	B	A	Count
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7
0	0	0	0

Figure 10.10: a)logic diagram b)truth table c)waveforms

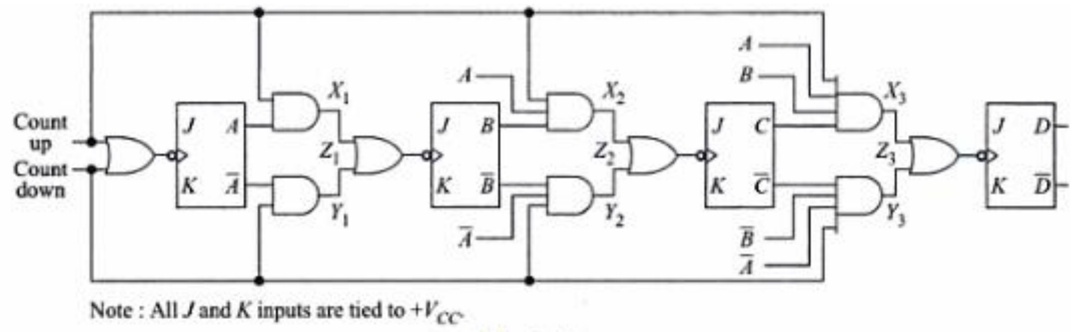
SYNCHRONOUS 4-BIT UP-DOWN COUNTER

• To operate in the count-up mode, the system-clock is applied at the count-up input, while the count-down input is held low (Figure:10.12).

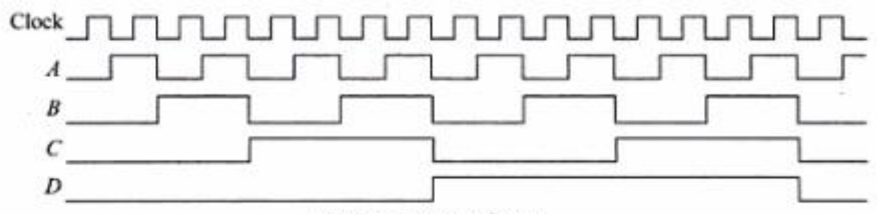
• To operate in the count-down mode, the system-clock is applied at the count-down input while holding the count-up input low.

• If the count-down line = LOW, the lower AND gates Y_1, Y_2 and Y_3 are disabled. The clock applied at count-up will then go directly into flip-flop A and will be steered into the other flip-flops by AND gates X_1, X_2 and X_3.

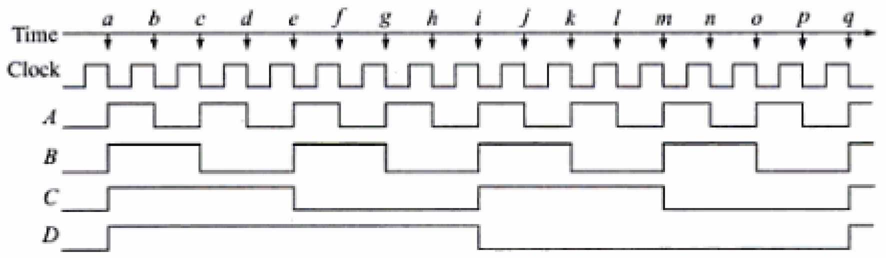
• If the count-up line = LOW, the upper AND gates X_1, X_2 and X_3 are disabled. The clock applied at input count-down will go directly into flip-flop A and will be steered into the other flip-flops by AND gates Y_1, Y_2 and Y_3.



(a) Logic diagram



(b) Count up waveforms



(c) Count down waveforms

Figure 10.12:a)logic diagram b)count up waveforms c)count down waveforms

DIFFERENCE BETWEEN ASYNCHRONOUS & SYNCHRONOUS COUNTERS

Asynchronous Counters	***Synchronous Counters***
Flip flops are connected in such a way that the output of the first flip-flop drives the second flip-flop, the output of the second flip-flop drives the third flip-flop etc.	There is no connection between output of first flip-flop and clock input of the next flip-flop.
All flip-flops are not clocked simultaneously, since all the clock inputs are not connected to the system clock. Only the first flip-flop is driven by the clock.	All flip-flops are clocked simultaneously by connecting the clocks inputs of all flip-flops in parallel to the system clock.
Logic circuit is quite simple, whatever the number of states.	As the number of states increases, the design of the logic circuit tends to become complex.
Main drawback is their low speed as the clock is propagated through number of flip-flops before it reaches last flip-flop.	Since all the flip-flops have the system clock applied simultaneously, propagation delay in minimum, it being equal to the propagation delay of one flip-flop plus propagation delay of one AND gate.

COUNTER MODULUS

• Modulus is defined as the number of states through which a counter can progress.
• The counters which progress 1 count at a time in a strict binary progression, and they all have a modulus given by 2pown, where n=number of flip-flops. Such counters are said to have a "natural count" of 2^n. For example,

→ mod2 counter consists of 1 flip-flop, and it counts two discrete states (0->1)
→ mod4 counter consists of 2 flip-flops, and it counts through 4 discrete states (00->01->10 ->11)
→ mod8 counter consists of 3 flip-flops, and it counts through 8 discrete states (000->001 ->010->011->100->101->110->111)

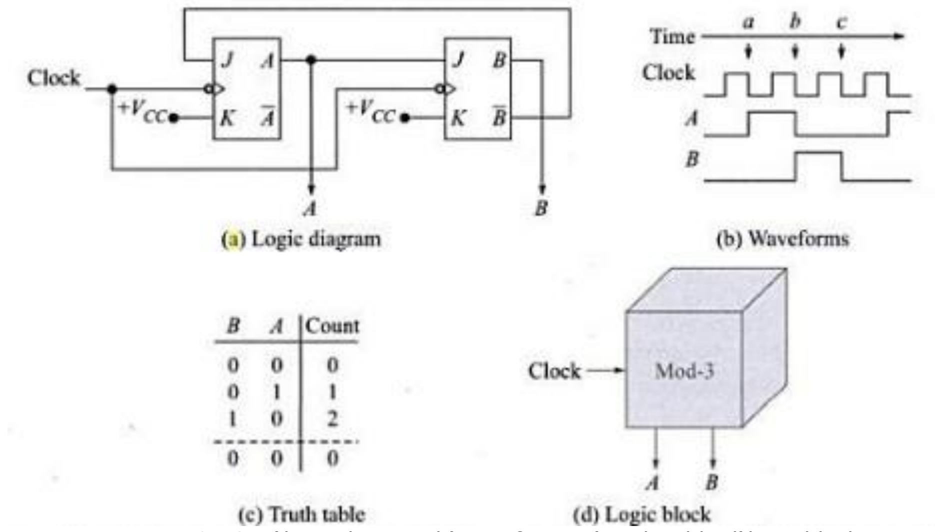
• It is often desirable to construct counters having a modulus other than 2, 4, 8 and so on. For example, a counter having a modulus of 3 or 7 would be useful.
• A small modulus counter can always be constructed from a larger modulus counter by skipping states. Such counters are said to have a *modified-count.*
• Firstly, it is necessary to determine the number of flip-flops required. The correct number of flip-flops is determined choosing the lowest natural count that is greater than the desired modified count. For example, a mod-7 counter requires 3 flip-flops, since 8 is the lowest natural count greater than the desired modified count of 7.

MOD-3 COUNTER

• A mod-3 counter can count in sequence from 00 to 11(00->01->10)
• Mod-3 counter
  → requires 2 flip-flops which has a natural count of 4
  → skips one state(i.e. 11) {Figure10.16}.
• It has a clock input and outputs at A and B.
• Here's how it works:

1. Prior to point 'a' on the time-line, A=0 and B=0. A negative-clock-transition(NCT) at 'a' will cause:
   i. 'A' to toggle to a 1, since it's inputs J=HIGH & K=HIGH.
   ii. 'B' to reset to 0, since it's inputs J=LOW & K=HIGH.
2. Prior to point 'b' on the time-line, A=1 and B=0. A negative-clock-transition at 'b' will cause:
   i. 'A' to toggle to a 0, since it's inputs J=HIGH & K=HIGH.
   ii. 'B' to toggle to a 1, since it's inputs J=HIGH & K=HIGH.
3. Prior to point 'c' on the time-line, A=0 and B=1. A negative-clock-transition at 'c' will cause:
   i. 'A' to reset to 0, since it's inputs J=LOW & K=HIGH.
   ii. 'B' to reset to 0 since it's inputs J=LOW and K=HIGH.
4. The counter has now progressed through all three of its states, advancing one count with each negative–clock-transition.

• It can be considered as a divide-by-3 block, since the output waveform at B has a period equal to three times that of the clock.



B	A	Count
0	0	0
0	1	1
1	0	2
0	0	0

Figure 10.16: a)logic diagram b)waveforms c)truth table d)logic block

MOD-6 COUNTER

• This counter can be formed by using the mod-3 counter in conjunction with a flip-flop (Figure:10.7).
• This logic circuit can no longer be considered a synchronous counter since flip-flop C is triggered by flip-flop B i.e. all flip-flops do not change status in synchronism with the clock.

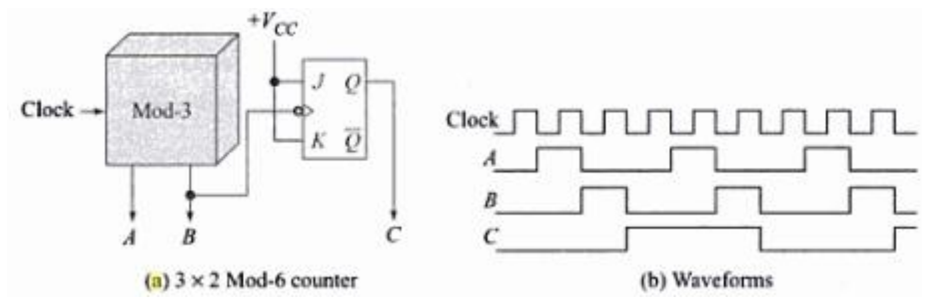


Figure 10.17: a)3*2 mod-6 counter b)waveforms

MOD-5 COUNTER

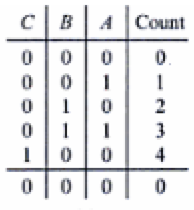
• This counter has a natural count of 8, but it is connected in such a way that it will skip over three states (101,110 & 111)

• The waveform shows that flip-flop A toggles with each negative-clock-transition except during the transition from count 4 to count 0. Thus, flip-flop A should be triggered by the clock and must have an inhibit during count 4 i.e. some signal must be provided during the transition from count 4 to count 0.

• In constructing a counter of this type, it is always necessary to examine the omitted states to make sure that the counter will not malfunction (i.e. counter should not get into invalid state e.g. 101,110,111)

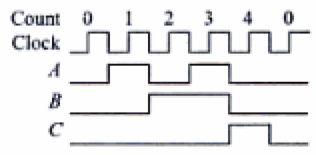
• Assuming that the counter is in state 5(CBA=101). When the next clock pulse goes low, the following events occur:

1. Since C' is low, flip-flop A resets. Thus A changes from a 1 to a 0.
2. When A goes from a 1 to a 0, flip-flop B triggers and B changes from a 0 to a 1
3. Since the J input to flip-flop C is low, flip-flop C is reset and C changes from a 1 to a 0.
4. Thus the counter progresses from the illegal state 5 to the legal state 2(CBA=0101) after one clock

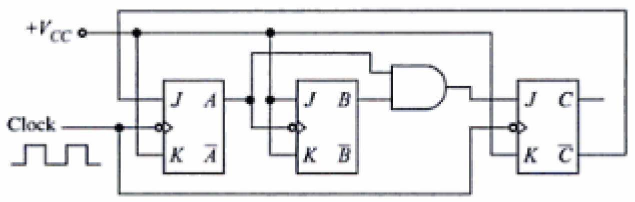


C	B	A	Count
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
0	0	0	0

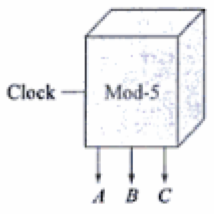
(a)



(b)



(c)

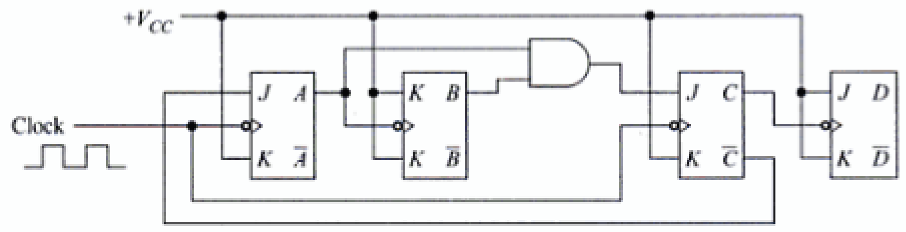


(d) Logic block

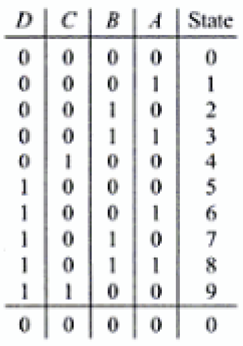
Figure 10.20:a)truth table b)waveforms c)logic diagram

MOD-10 COUNTER (DECADE COUNTER)

• A decade counter can be formed by using mod-5 counter in conjunction with a flip-flop (Fig: 10.22).

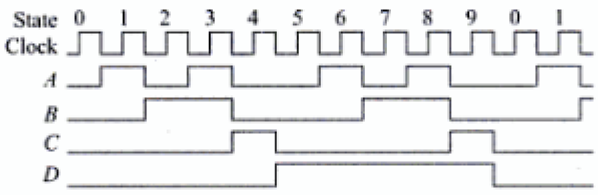


(a)



D	C	B	A	State
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
1	0	0	0	5
1	0	0	1	6
1	0	1	0	7
1	0	1	1	8
1	1	0	0	9
0	0	0	0	0

(b)



(c)

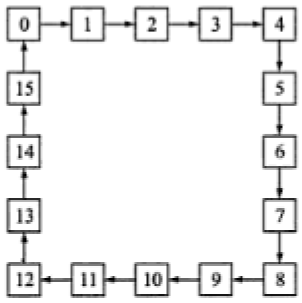
Figure 10.21:a)logic diagram b)truth table c)waveforms

PRESETTABLE COUNTER

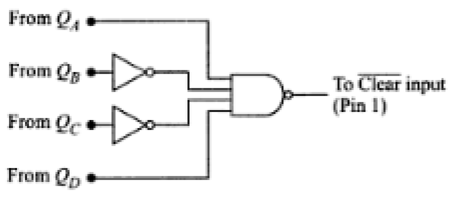
• It is a counter incorporating logic such that it can be preset to any desired state. ie it can be used to implement a counter that has any modulus (Figure:10.26).

• Nearly all the presettable counters are constructed by using four flip-flops,and they are generally referred to as 4-bit counters.

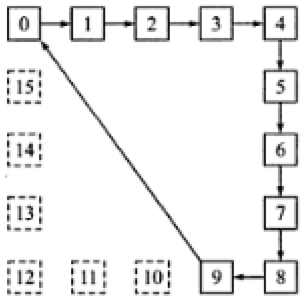
• When connected such that the count advances in a natural binary sequence from 0000 to 1111,it is simply referred to as a binary counter.



(a) Mod-16 counter state diagram



(b) Gate to decode count 9 (1001)



(c) Modified state diagram for Mod-10 counter

Figure 10.26:a)mod-16 counter state diagram b)modified state diagram for mod-10 counter

COUNTER DESIGN AS A SYNTHESIS PROBLEM

- Let us design a modulo-6 counter (Figure: 10.32).
- In mod-6 counter, there are 6 states. So, we need 3 flip-flops to design mod-6 counter.
- With 3 flip-flops, 8 different states are possible but in our design, states 110 and 111 are not used in the counting sequence.
- We use three JK flip-flops labeled A, B and C as memory element for this design.

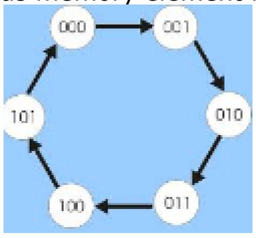


Figure 10.32: State sequence of mod 6 counter

Table 8.35: Excitation Table of flip-flops

RS

Q_t	Q_{t+1}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

JK

Q_t	Q_{t+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

D

Q_t	Q_{t+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

T

Q_t	Q_{t+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Table 10.1: State Table for Design of mod-6 counter

C_n	B_n	A_n	C_{n+1}	B_{n+1}	A_{n+1}	J_C	K_C	J_B	K_B	J_A	K_A
0	0	0	0	0	1	0	×	0	×	1	×
0	0	1	0	1	0	0	×	1	×	×	1
0	1	0	0	1	1	0	×	×	0	1	×
0	1	1	1	0	0	1	×	×	1	×	1
1	0	0	1	0	1	×	0	0	×	1	×
1	0	1	0	0	1	×	1	0	×	×	1

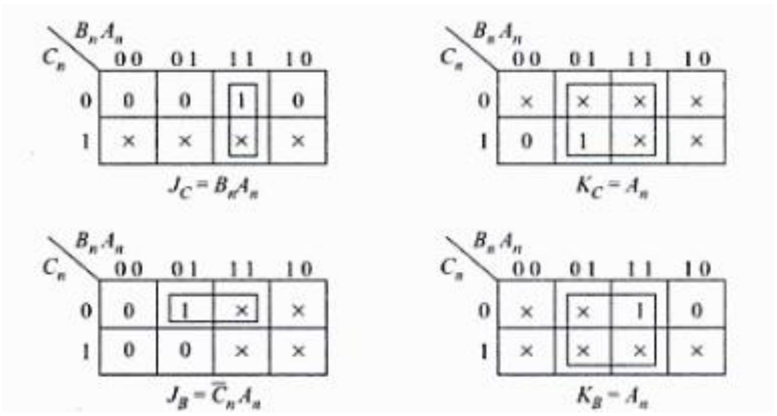


Figure 10.33: Derivation of design equations from kmap.

From the kmaps, the J, K inputs to the flip-flops A, B and C are obtained as:

$J_A=1$	$K_A= 1$	for flip-flops A
$J_B=C'_n A_n$	$K_B=A_n$	for flip-flops B
$J_C=B_nA_n$	$K_C= A_n$	for flip-flops C

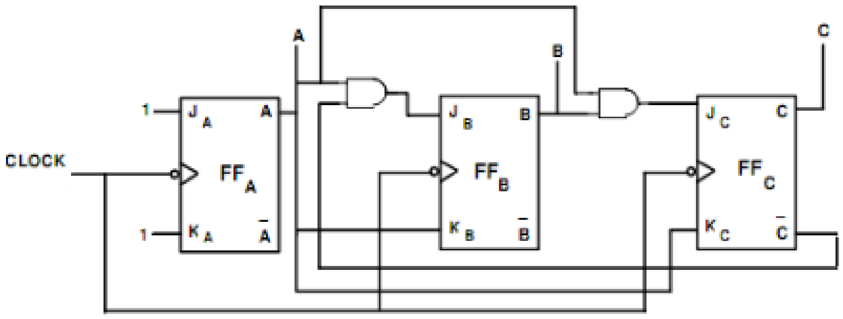


Figure 10.34: Circuit diagram of mod6 synchronous counter

DIGITAL CLOCK

• As shown in figure 10.40, the first divide-by-60 counter divides the 60Hz power signal down to a 1Hz square wave. The second divide-by-60 counter changes state once each second and has 60 discrete states. It can be decoded to provide signals to display seconds. This counter is referred to as the seconds counter.
• The third divide-by-60 counter changes state once each minute and has 60 discrete states. This counter is referred to as the minutes counter.
• The last counter changes state once each 60 minutes. Thus, if it is a divide-by-12 counter, it will have 12 states that can be decoded to provide signals to display the correct hour. This is referred to as the hours counter.

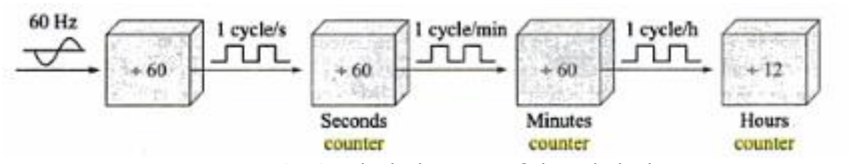


Figure 10.40: Block diagram of digital clock

DIVIDE-BY-60 COUNTER

• The divide-by-60 counter can be implemented by cascading counters(10*6=60)
• The TTL MSI 7490 decade counter can be used as a divide-by-10 counter and the TTL MSI 7492 can be used as a divide-by-6 counter. Cascading these two counters will provide a divide-by-60 counter.(Figure:10.41)
• The amplifier at the input provides a 60Hz square wave of the proper amplitude to drive the 7490.
• In figure:10.42, the 7492 is connected as a divide-by-12 counter, but only outputs QA,QB & QC are used. In this fashion, the 7492 operates essentially as a divide-by-6 counter.
• For decoding 'seconds counter' to represent each of the 60s in 1min, we construct a mod-10 counter in series with a mod-6 counter.
• The mod-10 counter can then be decoded to represent the units digit of seconds, and the mod-6 counter can be decoded to represent the tens digits of seconds.
• Since both the 7490 & 7492 count in the straight 8421,a 7447 decoder-driver can be used with each to drive two 7-segment indicators(Figure: 10.42)

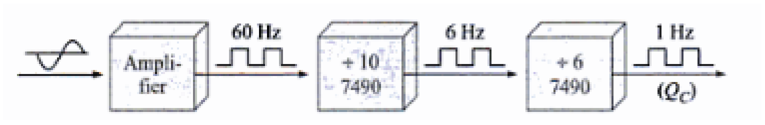


Figure 10.41:Divide-by-60 counter

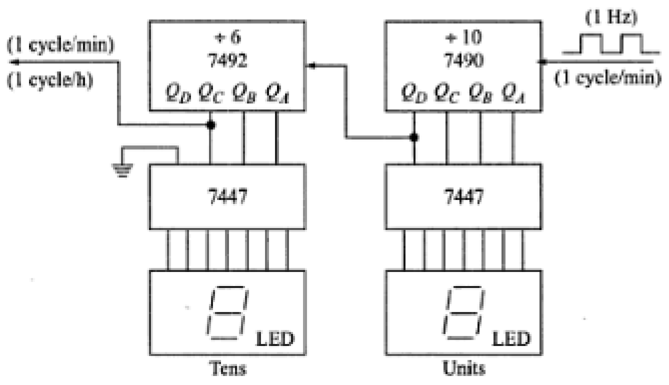


Figure 10.42: A 10*6 mod-60 counter with units and tens decoding

Verilog code for a modulo-8 up counter

```
module UpCounter(Clock, Reset, Q);
    input Clock, Reset;
    output [2:0] Q;
    always @(negedge Clock or negedge Reset)
    if(~Reset)
        Q=3'b0;              // store three 0's
    else
        Q=Q+1;
endmodule
```

Verilog code for a modulo-8 up counter using JK flip-flop

```
module UpcountJk(A, B, Clock, Reset);
    input Clock, Reset;
    output A,B;
    wire JA,JB,KA,KB;
    assign JA=~B;
    assign KA=1'b1;
    assign JB=A;
    assign KB=1'b1;
    JKFF JK1(A, JA, KA, Clock, Reset);
    JKFF JK2(B, JB, KB, Clock, Reset);
endmodule

module JKFF(Q, J, K, Clock, Reset);
    input J,K, Clock, Reset;
    output Q;
    reg Q;
    always @ (negedge Clock or negedge Reset)
    if(~Reset)
       Q=1'b0;
    else
       Q<=(J&~Q)|(~K&~Q);
endmodule
```

Verilog code for a modulo-8 up down counter which counts in upward direction if input MODE=0,else counts in downward direction. It has a parallel load facility. When PL=1,a 3-bit number D is asynchronously loaded to the counter. The counter counts at the negative edge of CLOCK and its output is represented by Q.

```
module UpCounterPL(CLOCK, PL, MODE, D ,Q);
    input CLOCK,PL,MODE;
    input [2:0] D;
    output [2:0] Q;       //modulo 8 requires 3 flip-flop
    reg  [2:0] Q;
    integer updown;     //updown will be +1 or -1 depending on MODE
    always @ (negedge CLOCK)
    begin
        if(MODE)
             updown=-1;        //if MODE=1,counts downward
        else
           updown=1;
        if(PL)             //if PL=1,parallel loading takes place
          Q=D;
        else
          Q=Q+updown;   //last else statement responds to clock
    end
endmodule
```

EXERCISE:

1> Write a short note on counters? (5)
2> Explain the working of a 3-bit asynchronous down counter? (6)
3> What is the clock frequency in mod-8 counter,
i) If the period of the waveform at C(which is MSB) is 24microsec?
ii) If the period of the waveform at A(which is LSB) is 18microsec? (2)
4> Explain the working of a 3-bit asynchronous up-counter along with truth table and waveforms? (8)
5> i) What are decoding gates? Draw waveform for state 7 in a 3-bit synchronous up counter?
ii) What is the primary cause of glitches that sometimes occur at the output of a decoding gate used with a ripple counter? What is one method to eliminate these glitches?
iii) Why are decoding gate glitches eliminated in a synchronous counter? (8)
6> Draw the logic diagram, truth table and waveforms for a two flip-flop ripple counter? (4)
7> Draw the logic diagram, truth table and waveforms for a two flip-flop ripple counter operating in the count down mode? (4)
8> Draw decoding gates and all waveforms for the mod-3 counter? (2)
9> Draw decoding gates and all waveforms for the mod-6 counter? (2)
10> Explain the working of synchronous 4-bit up counter? (8)
11> Design a synchronous mod-8 counter and sketch the output waveform? (8)
12> What is an up-down counter? Explain the working of synchronous 4-bit up-down counter? (8)
13> Differentiate between ripple and synchronous counter? (6)
14> Explain a mod-3 counter along with logic diagram, waveforms and truth table? (8)
15> Explain a 3*2 mod-6 counter along with block diagram and waveforms? (5)
16> Draw the waveforms you would expect from the mod-6 counter by connecting a single flip-flop in front of the mod-3 counter? (4)
17> When does a counter is said to have modified count? Give example? (4)
18> Explain a mod-5 counter along with truth table, waveform and logic diagram? (8)
19> Explain a mod-10 counter (decade counter) along with truth table, waveform and logic diagram? (6)
20> What are presettable counters? Give example? (4)
21> Design a mod-4 synchronous down counter using JK flip-flops and implement it? (6)
22> Design a synchronous counter using JK flip-flops to count in the sequence
0,1,2,4,5,6,0,1,2. . . . . .
Use state diagram and state table? (6)
23> Design a synchronous mod-5 up-counter using JK flip-flop. Give excitation table of JK flip-flop, state diagram and state table? (10)
24> Design a mod-4 irregular counter with following counting sequence using D flip-flop:00->10->11->01? (6)
25> Show how a mod-4 counter designed with two flip-flops can generate a repetitive sequence of binary word '1101' with minimum number of memory elements? (4)
26> Design a synchronous mod-6 counter using clocked JK flip-flop? (8)
27> Write the excitation table of SR flip-flop. Design a synchronous mod-6 counter using SR flip-flop for the following count sequence 0, 1, 3,2,6,4 and repeat. Write the transition table, logic diagram? (10)
28> Realize a 3-bit binary synchronous up counter using JK flip flop. Write the excitation table, translation table and logic diagram. Include preset, clear option? (10)
29> Design a self-correcting mod-6 counter as described in state sequence of given figure in which all the unused state leads to state CBA=000? (8)
30> Design a synchronous mod-3 counter with the following binary sequence using clocked JK flip-flops. Counter sequence 0, 1, 2, 0, 1, 2 ? (8)
31> Design a modulo-3 counter using D flip-flop that counts as 01->10->11. The unused state 00 goes to 01 at next clock trigger ? (4)
32> Design a modulo-5 counter using D flip-flop the unused states of which go to one of the valid counting state at next clock trigger ? (6)
33> Design a modulo-8 counter using SR flip-flop? (6)
34> Design a modulo-8 counter using T flip-flop? (4)
35>Design a sequence generator with minimum number of flipflops that generates sequence 110001 repeatedly? (6)
36> Design a sequence generator with minimum number of flip-flops that generates sequence '10110001' repetitively? (6)
37> Explain a 10*6 mod-60 counter with units and tens decoding? (8)
38> Write a verilog code for a modulo-8 up counter? (4)
39> Write a verilog code for a modulo-8 up counter using JK flip-flop? (6)
40> Write a verilog code for a modulo-8 up down counter which counts in upward direction if input MODE=0, else counts in downward direction. It has a parallel load (PL) facility. When PL=1, a 3-bit number D is asynchronously loaded to the counter. The counter counts at the negative edge of CLOCK and its output is represented by Q? (4)