K.N.Govt. Arts College for Women(A), Thanjavur-7

Department of Computer Science

Subject : Digital Design

Subject Code: 18K5CS08

Subject Incharge:

Unit III: A.Thirumalai Raj, Asst. Prof. in Computer Science.

Unit IV: G.Umarani, Guest Lecturer in Computer Science.

Unit V : N.Baby Kala, Guest Lecturer in Computer Science.

Course Content

Unit-III : Logic Gates and Logic Circuits: Introduction – Analog and Digital Signals – Basic Logic Gates, NOT, OR, AND – Logic Circuit and Logic Expressions – Sum of Product(SOP) – Product of Sum(POS) – NAND and NOR Gates – EX-OR and EX-NOR Gates – Boolean Algebra: Laws of Boolean Algebra – DeMorgan’s Theorem – NAND as Universal Gate – NAND- NAND Network – NOR as Universal Gate.

Unit-IV: Karnaugh Map: Minterms and Maxterms – Relationship between k Map and truth table -2-variable K-map using minterms – 3-Variable K-map using minterms – 4-variable K-map using minterms – Don’t Care Conditions –
Arithmetic Circuits: Half Adder and Full Adder.

Unit-V: Combinational Logic: Combinational Circuits – Analysis Procedure – Decimal Adder – Encoders – Multiplexer. Synchronous Sequential Logic: Sequential Circuits – Storage Elements: Flip-Flops. Registers and Counters: Registers – Synchronous Counter.

References

“Digital Fundamentals” V.Vijayendran – S.Viswanathan(Printers & Publishers), Pvt. Ltd. – Reprint 2011.
“Digital Design” – M.Morris Mano Michael D.Ciletti – Fifth Edition –Pearson India Education Services Pvt. Ltd., Second Impression 2016.

UNIT III-LOGIC GATES AND CIRCUITS

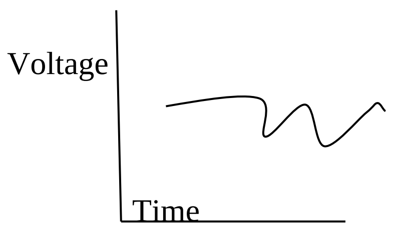
Introduction

- The mathematical theory of binary system called Boolean Alegebra
- The binary number system uses only two bits 0 and 1
- The digital circuits use of two states as ON and OFF or HIGH and LOW or True and Flase
- Each logic gates can have one or more inputs and only one output
- Two type voltages low voltage(0),high voltage(1)
- All the gates are available in integrated circuit (IC)
- Diode logic,Resistor-Transistor(RTL),Transistor-Transistor logic,Resistor-Transistor Logic

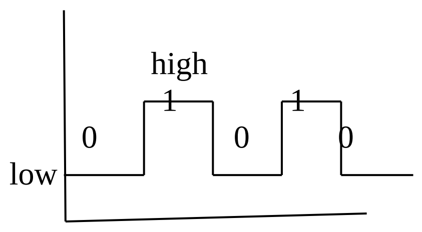
ANALOG AND DIGITAL SIGNAL

Analog electric signal is continuous in nature and takes all possible value with in a limit

A Graph is continuous and takes all the possible values,its include the both positive and negative



Analog Signal



Digital Signal

A digital signal Low (0) and High 1.The voltage level of 0 represents the 0 state and a voltage level 5 volts represent the 1 state fig 2

BASIC LOGIC GATES – NOT, OR, AND

Gates is an electronic circuits with one or more inputs but only one outputs

All the gates are digital states LOW and HIGH

Three basic gates are AND ,OR, NOR gates

NOT GATE or INVERTER

Not gate has only one input and one output. The input LOW(logic 0) and the output is HIGH(logic1)

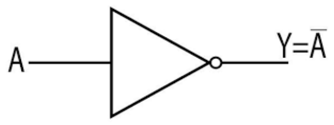
The input is HIGH(logic 1)and the output is LOW(logic0)

$Y=\overline{A}$

Truth Table

Logic Diagram

A	$Y=\overline{A}$
0	1
1	0



A is the input variable and Y is the output variable. The Binary variable logic variable

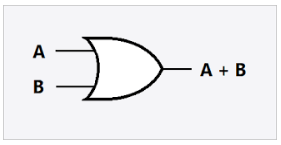
OR GATE

The gate perform the logical addition. A OR gates has two or more inputs and one output.

The one of the input is HIGH the output is HIGH

Both inputs are LOW the output is LOW

Y=A+B

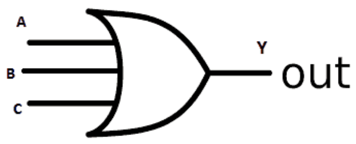


A	B	Y=A+B
0	0	0
0	1	1
1	0	1
1	1	1

A and B are input logical variables and Y is the output .The two inputs there are four possible

Combination 00,01,10 and 11

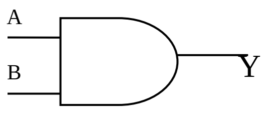
There inputs of OR GATES



A	B	C	Y=A+B+C
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

AND GATE

And gate performs multiplication. AND gate has two or more inputs and one output.If input is low the output goes LOW.The both inputs is high the outputs is high.

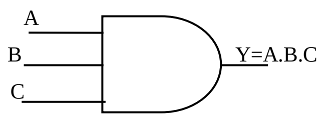


A	B	Y=A.B
0	0	0
0	1	0
1	0	0
1	1	1

A 3 inputs AND GATE three inputs and one output.

Y=A.B.C=ABC

A	B	C	Y=A.B.C
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



LOGIC CIRCUITS AND LOGIC EXPRESSIONS

To connect the different gates in different ways and form of logic expression and a truth table for the entire circuit. The logic expression is called Boolean expression. To construct the digital circuits

1. A set of statements or
2. Boolean Expression or
3. Truth table

The given logic circuit actually represents a new gate ,EX-OR(Exclusive OR,XOR
The truth table consist of different combination of input variables and the output for each combination
The input variables and the output must be connected by single expression. The two methods widely
Used to convert the truth table to logical expression

1.Sum of Product (SOP)

2.Product of Sum(POS)

Sum of Product

Sum of Product the expression consists of number of terms. Each term is a logical product of input variables. All the term are logically summed up to give the output and hence the sum of product
To consider three variables A,B and C .

Y = $\bar{A}\bar{B}C + \bar{A}B\bar{C} + ABC$

This is sum of product .The first term is logical product of $\bar{A}$ and $\bar{B}$ and C. The second term is logical Product of $\bar{A}$ and B and $\bar{C}$. The third term is a logical product of A and B and C. The summed up to the product expression. The variable can appear in normal form its complement forms .The three variable
A,B and C 8=(=2^3) combination and 8 standard products. These fundamental products are also called is minterms.

For two variables A and b there are four possible combinations $\bar{A}\bar{B}$(00), $\bar{A}$B(01),A$\bar{B}$ (10),AB(11) and 4 standard products or minterm

Two variable A and B

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

SOP expression for the truth table ,Select the rows whose output is 1

First row Y=1 when A=0 and B=0 are complement to give $\bar{A}$ = 1 and $\bar{B}$ =1 .the output is 1

Y=1, A=1 and B= 0 B needs to be complemented and A$\bar{B}$=1

The two standard products are $\bar{A}\bar{B}$ and A$\bar{B}$

Y=$\bar{A}\bar{B}$+A $\bar{B}$

The truth table of three variable A, B and C. The output Y is given values 0 and 1 at random.The Sum of product

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

Y=1 A=0,B=0and C=1 Standard product is $\bar{A}\bar{B}$C;

A=1,B=0and C=1;Standard product is A$\bar{B}$C;
A=1.B=1 and C=1; Standard product is AB$\bar{C}$
A=1,B=1 and C=1 Standard product is ABC

The standard SOP expression is

Y=$\bar{A}\bar{B}$C+A$\bar{B}$C+AB$\bar{C}$

PRODUCT OF SUM (POS)

Each term is a logical sum(OR) of input variables. All the terms are logically multiplied to give the output and the product of sum(POS)
each term is a logical product and gives a logical 1.POS each term is a logical sum and produces a logical 0

F(A,B,C) = (A + $\bar{B}$ + $\bar{C}$)(A + $\bar{B}$ + C)($\bar{A}$ + B + C) OR F(A,B,C)=($\bar{A}$ + $\bar{B}$ +C) ($\bar{A}$ + B + $\bar{C}$) (A +B+C)
This product of sum form.
The Sums (A+B+C),(A+B+$\bar{C}$)
The term ($\bar{A}$ + B + C) to A=1,B=1and C=0 that ($\bar{A}$ + $\bar{B}$ +C)=0
The term ($\bar{A}$ + B + C) to A=1,B=1 and C=1 that ($\bar{A}$ + B + $\bar{C}$)=0
The term (A + B+C) to A=0,B=0and C=0 that (A+B+C)=0
The terms put together gives Y=0.0.0=0.These fundamental sums are called as Maxterm.

Y=

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

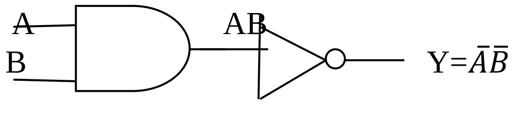
Y=0when A=0,B=0,and C=0 the standard sum is (A+B+C)
A=0,B=1,and C=0 the standard sum is (A+$\bar{B}$+C)
A=0,B=1,and C=1 the standard sum is (A+$\bar{B}$+$\bar{C}$)
A=1,B=0,and C=0 the standard sum is ($\bar{A}$+B+C)

Therefore the standard POS Expression is
(A + B + C) (A + $\bar{B}$ +C) (A + $\bar{B}$ + $\bar{C}$)($\bar{A}$ + B +C)

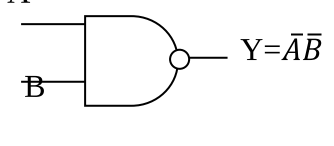
NAND and NOR GATES

NAND GATE

The AND gate and the NOT as a NAND Gate



The symbol for the NAND gate the and its truth table

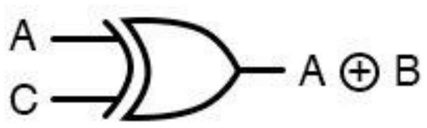


A	B	Y=$\bar{A}\bar{B}$
0	0	1
0	1	1
1	0	1
1	1	0

The output is 1 when any input goes to O. The output is 0 only when all the input are in 1 state

EX-OR GATE

In a two input Exclusive OR (EX-OR or XOR) the output is high(1) only when one of inputs is high(1).The truth table for a two input EX-OR gate



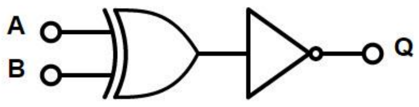
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

This is different from the regular OR gate in the last combination

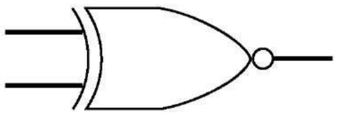
EX-NOR GATE

In a two input Exclusive OR(EX-OR or XOR), the output is high(1) when of inputs is high(1).The truth table for a two input EX-OR gate is

Truth Table



Y=AʘB



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

POSTIVE and NEGATIVE LOGIC

The low voltage to 0 and the high voltage to 1 state. The low stands for 0 and high stands for 1 knows as positive logic

Ideal Condition

Positive Logic	Negative Logic
HIGH = 1 state = 5 Volts	HIGH = 0 State = 0 volts
LOW = 0 state = 0 Volts	LOW = 1 State = 5 volts

The change of positive and negative logic .the action of the logic gates also change.The truth table

A	B	Y
LOW	LOW	LOW
LOW	HIGH	HIGH
HIGH	LOW	HIGH
HIGH	HIGH	HIGH

This table has inputs and outputs represented in LOW and HIGH format. They convert into 1's and 0's

The positive logic and Negative logic

In positive logic LOW =0 and HIGH=1and the truth table

The

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

truth table corresponds to OR gate

In negative logic LOW=1 and HIGH =1 and the new truth table

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

The truth table corresponds to AND gate

OR gate under positive logic functions as an AND gate under negative logic

Karnaugh Map

Minterms

A *minterm* is a Boolean expression resulting in 1 for the output of a single cell, and 0s for all other cells in a Karnaugh map, or truth table. If a minterm has a single 1 and the remaining cells as 0s, it would appear to cover a minimum area of 1s.

The illustration above left shows the minterm **ABC**, a single product term, as a single **1** in a map that is otherwise **0**s. We have not shown the **0**s in our Karnaugh maps up to this point, as it is customary to omit them unless specifically needed. Another minterm **A'BC'** is shown above right.

The point to review is that the address of the cell corresponds directly to the minterm being mapped. That is, the cell **111** corresponds to the minterm **ABC** above left.

Above right we see that the minterm **A'BC'** corresponds directly to the cell **010**. A Boolean expression or map may have multiple minterms.

Referring to the above figure, Let's summarize the procedure for placing a minterm in a K-map:

- Identify the minterm (product term) term to be mapped.
- Write the corresponding binary numeric value.
- Use binary value as an address to place a **1** in the K-map

Example Convert the non standard SOP function F = x y + x z + y z
Sol:
F = x y + x z + y z
= x y (z + z') + x (y + y') z + (x + x') y z
= x y z + x y z' + x y z + x y' z + x y z + x' y z
= x y z + x y z' + x y' z + x' y z
The standard SOP form is F = x y z + x y z' + x y' z + x' y z

Maxterms

Maxterm

A *maxterm* is a Boolean expression resulting in a **0** for the output of a single cell expression, and **1**s for all other cells in the Karnaugh map, or truth table. The illustration above left shows the maxterm **(A+B+C)**, a single sum term, as a single **0** in a map that is otherwise **1**s.

If a maxterm has a single **0** and the remaining cells as **1**s, it would appear to cover a maximum area of **1**s.

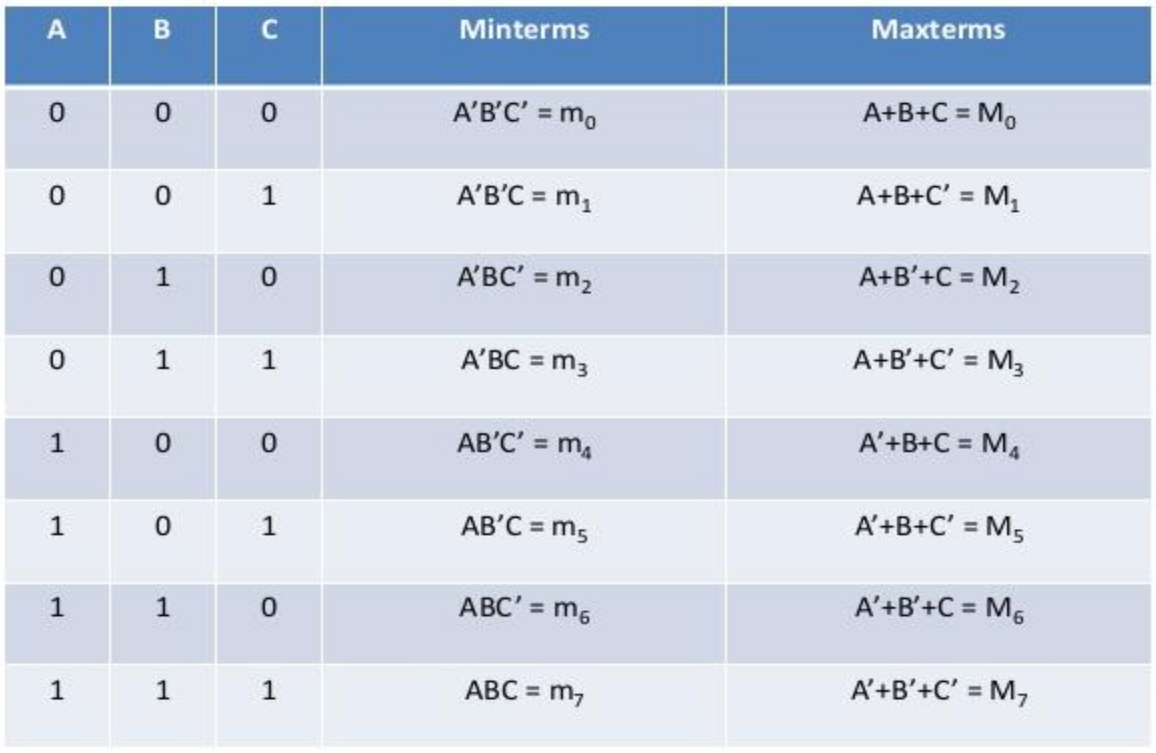
There are some differences now that we are dealing with something new, maxterms. The maxterm is a **0**, not a **1** in the Karnaugh map. A maxterm is a sum term, **(A+B+C)** in our example, not a product term. It also looks strange that **(A+B+C)** is mapped into the cell **000**.

For the equation **Out=(A+B+C)=0**, all three variables **(A, B, C)** must individually be equal to **0**. Only **(0+0+0)=0** will equal **0**. Thus we place our sole **0** for minterm **(A+B+C)** in cell **A,B,C=000** in the K-map, where the inputs are all **0** .

This is the only case which will give us a **0** for our maxterm. All other cells contain **1**s because any input values other than (**(0,0,0)** for **(A+B+C)** yields **1**s upon evaluation.

Referring to the above figure, the procedure for placing a maxterm in the K-map is:

- Identify the Sum term to be mapped.
- Write corresponding binary numeric value.
- Form the complement
- Use the complement as an address to place a **0** in the K-map



A	B	C	Minterms	Maxterms
0	0	0	$A'B'C' = m_0$	$A+B+C = M_0$
0	0	1	$A'B'C = m_1$	$A+B+C' = M_1$
0	1	0	$A'BC' = m_2$	$A+B'+C = M_2$
0	1	1	$A'BC = m_3$	$A+B'+C' = M_3$
1	0	0	$AB'C' = m_4$	$A'+B+C = M_4$
1	0	1	$AB'C = m_5$	$A'+B+C' = M_5$
1	1	0	$ABC' = m_6$	$A'+B'+C = M_6$
1	1	1	$ABC = m_7$	$A'+B'+C' = M_7$

Example:
Convert the **F = (A' + B + C) * (B' + C + D') * (A + B' + C' + D) into Standard POS form**
Solution
In the first term, the variable D or D' is missing, so we add D*D' = 1 to it. Then

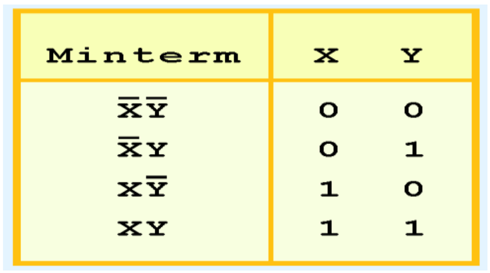
(A’ + B + C + D*D’) = (A’ + B + C + D) * (A’ + B + C + D’)
Similarly, in the second term, the variable A or A’ is missing, so we add A*A’ = 1 to it. Then
(B’ + C + D’ + A*A’) = (A + B’ + C + D’) * (A’ + B’ + C + D’)
The third term is already in the standard form, as it has all the variables. Now the standard POS form equation of the function is
F = (A’ + B + C + D) * (A’ + B + C + D’) * (A + B’ + C + D’) * (A’ + B’ + C + D’) * (A + B’ + C’ + D)

K-Map

- Simplification of Boolean functions leads to simpler (and usually faster) digital circuits.
- Simplifying Boolean functions using identities is time-consuming and error-prone.
- This special section presents an easy, systematic method for reducing Boolean expressions
- In 1953, Maurice Karnaugh was a telecommunications engineer at Bell Labs.
- While exploring the new field of digital logic and its application to the design of telephone circuits, he invented a graphical way of visualizing and then simplifying Boolean expressions.
- This graphical representation, now known as a Karnaugh map, or Kmap, is named in his honor.

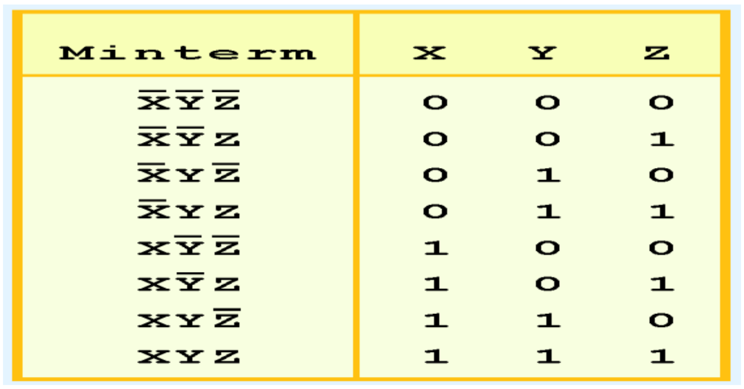
Description of K-maps and Terminology

- A Kmap is a matrix consisting of rows and columns that represent the output values of a Boolean function.
- The output values placed in each cell are derived from the minterms of a Boolean function.
- A *minterm* is a product term that contains all of the function’s variables exactly once, either complemented or not complemented.
- For example, the minterms for a function having the inputs *x* and *y* are:
- $\overline{x}\overline{y}, \overline{x}y, x\overline{y},$ and xy
- Consider the Boolean function, $F(x,y) = xy + x\overline{y}$
- Its minterms are:



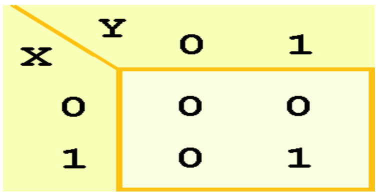
Minterm	X	Y
$\overline{X}\,\overline{Y}$	0	0
$\overline{X}Y$	0	1
$X\overline{Y}$	1	0
XY	1	1

- Similarly, a function having three inputs, has the minterms that are shown in this diagram.



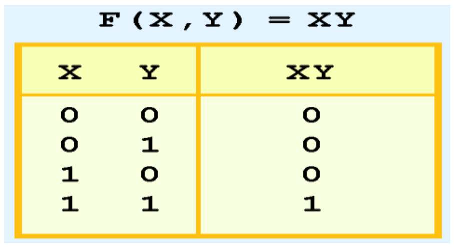
Minterm	X	Y	Z
$\overline{X}\,\overline{Y}\,\overline{Z}$	0	0	0
$\overline{X}\,\overline{Y}Z$	0	0	1
$\overline{X}Y\overline{Z}$	0	1	0
$\overline{X}YZ$	0	1	1
$X\overline{Y}\,\overline{Z}$	1	0	0
$X\overline{Y}Z$	1	0	1
$XY\overline{Z}$	1	1	0
XYZ	1	1	1

- A Kmap has a cell for each minterm.
- This means that it has a cell for each line for the truth table of a function.



X \ Y	0	1
0	0	0
1	0	1

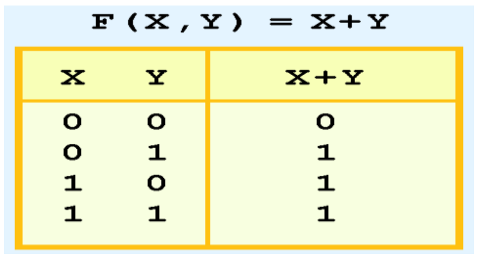
- The truth table for the function *F(x,y)* = *xy* is shown below at the along with its corresponding K-map.



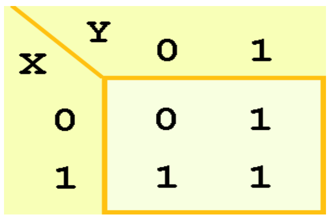
F(X,Y) = XY

X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

- As another example, we give the truth table and K-Map for the function, *F(x,y)* = *x* + *y*


F(X,Y) = X+Y

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1



X \ Y	0	1
0	0	1
1	1	1

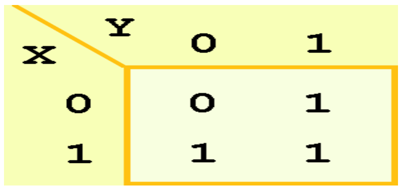
- This function is equivalent to the OR of all of the minterms that have a value of 1. Thus:

$$F(x,y) = x + y = \bar{x}y + x\bar{y} + xy$$

K-map Simplification for Two Variables

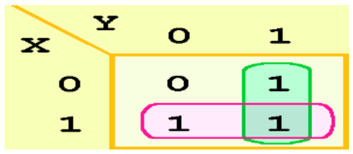
- Of course, the minterm function that we derived from our Kmap was not in simplest terms.
  - That's what we started with in this example.
- We can, however, reduce our complicated expression to its simplest terms by finding adjacent 1s in the K-map that can be collected into groups that are powers of two.
- In our example, we have two such groups.

  Can you find them?



X \ Y	0	1
0	0	1
1	1	1

- The best way of selecting two groups of 1s form our simple K-map is shown below.
- We see that both groups are powers of two and that the groups overlap.



X \ Y	0	1
0	0	1
1	1	1

F(X,Y)=X'Y+XY+XY+XY'

=Y(X+X')+X(Y+Y")

=(X+Y)

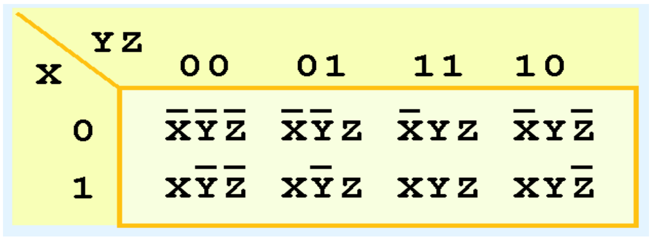
Rules for K-Map Simplification

The rules of K-map simplification are:

- Groupings can contain only 1s; no 0s.
- Groups can be formed only at right angles; diagonal groups are not allowed.
- The number of 1s in a group must be a power of 2 – even if it contains a single 1.
- The groups must be made as large as possible.
- Groups can overlap and wrap around the sides of the K-map.

K-map Simplification for Three Variables

- A K-map for three variables is constructed as shown in the diagram below.
- We have placed each minterm in the cell that will hold its value.
  - Notice that the values for the *yz* combination at the top of the matrix form a pattern that is not a normal binary sequence.



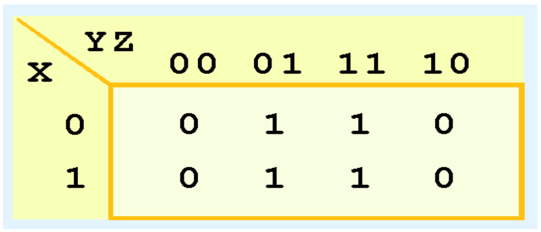
X \ YZ	00	01	11	10
0	$\bar{X}\bar{Y}\bar{Z}$	$\bar{X}\bar{Y}Z$	$\bar{X}YZ$	$\bar{X}Y\bar{Z}$
1	$X\bar{Y}\bar{Z}$	$X\bar{Y}Z$	XYZ	$XY\bar{Z}$

- Thus, the first row of the K-map contains all minterms where *x* has a value of zero.
- The first column contains all minterms where *y* and *z* both have a value of zero.

Consider the function:

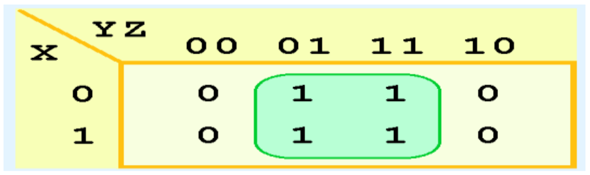
$$F(X,Y) = \bar{X}\bar{Y}Z + \bar{X}YZ + X\bar{Y}Z + XYZ$$

- Its -Kmap is given below.
  - What is the largest group of 1s that is a power of 2?



X \ YZ	00	01	11	10
0	0	1	1	0
1	0	1	1	0

- This grouping tells us that changes in the variables *x* and *y* have no influence upon the value of the function: They are irrelevant.
- This means that the function,



X \ YZ	00	01	11	10
0	0	1	1	0
1	0	1	1	0

F(X.Y)=X’Y’Z+X’YZ+XY’Z+XYZ

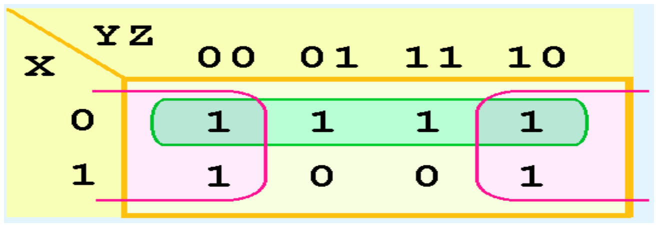
=X’Z(Y+Y’)+XZ(Y+Y’)

= X’Z+XZ

= Z(X+X’)

=Z

- reduces to *F(x) =Z.*
- The green group in the top row tells us that only the value of *x* is significant in that group.
- We see that it is complemented in that row, so the other term of the reduced function is . $\overline{X}$
- Our reduced function is:



X \ YZ	00	01	11	10
0	1	1	1	1
1	1	0	0	1

- $F(X,Y,Z) = \overline{X}+\overline{Z}$

For Group I

F(X,Y,Z)=X’Y’Z’+X’Y’Z+X’YZ+X’YZ’

=X’Y’(Z+Z’)+X’Y(Z+Z’)

=X’Y’+X’Y

=X’(Y+Y’)

=X’

For Group II

F(X,Y,Z)=X’Y’Z’+XY’Z’+X’YZ’+XYZ’

= Y’Z’(Y+Y’)+YZ’(X+X’)

=Y’Z’+YZ’

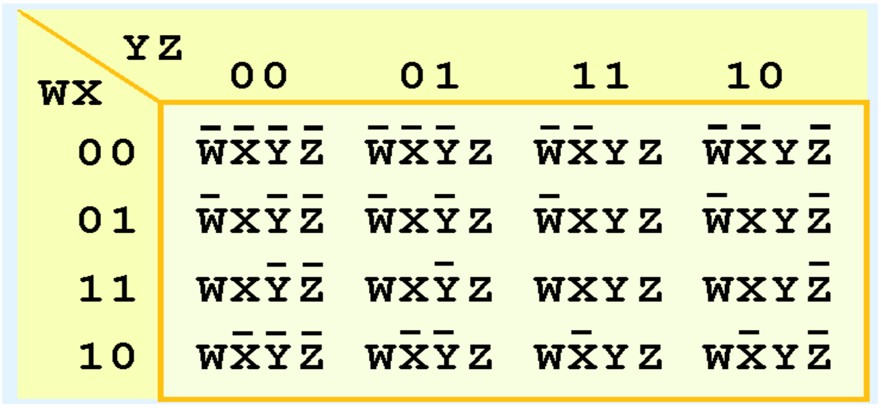
=Z’(Y+Y’)

Combining group I and group II we get

F(X,Y,Z) = X’+Z’

K-map Simplification for Four Variables

- Our model can be extended to accommodate the 16 minterms that are produced by a four-input function.
- This is the format for a 16-minterm K-map.

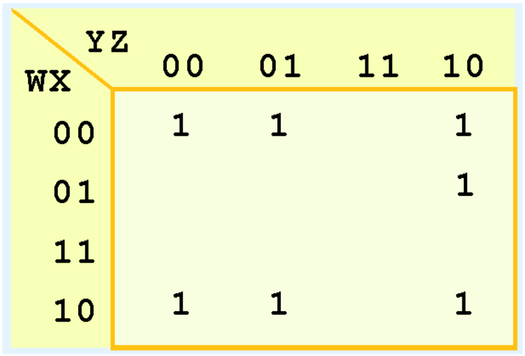


WX \ YZ	00	01	11	10
00	$\overline{W}\overline{X}\overline{Y}\overline{Z}$	$\overline{W}\overline{X}\overline{Y}Z$	$\overline{W}\overline{X}YZ$	$\overline{W}\overline{X}Y\overline{Z}$
01	$\overline{W}X\overline{Y}\overline{Z}$	$\overline{W}X\overline{Y}Z$	$\overline{W}XYZ$	$\overline{W}XY\overline{Z}$
11	$WX\overline{Y}\overline{Z}$	$WX\overline{Y}Z$	$WXYZ$	$WXY\overline{Z}$
10	$W\overline{X}\overline{Y}\overline{Z}$	$W\overline{X}\overline{Y}Z$	$W\overline{X}YZ$	$W\overline{X}Y\overline{Z}$

- We have populated the K-map shown below with the nonzero minterms from the function:
  - Can you identify (only) three groups in this Kmap?

$$F(W,X,Y,Z)=\overline{W}\overline{X}\overline{Y}\overline{Z}+\overline{W}\overline{X}\overline{Y}Z+\overline{W}\overline{X}Y\overline{Z}+\overline{W}XY\overline{Z}+W\overline{X}\overline{Y}\overline{Z}+W\overline{X}\overline{Y}Z+W\overline{X}Y\overline{Z}$$

-

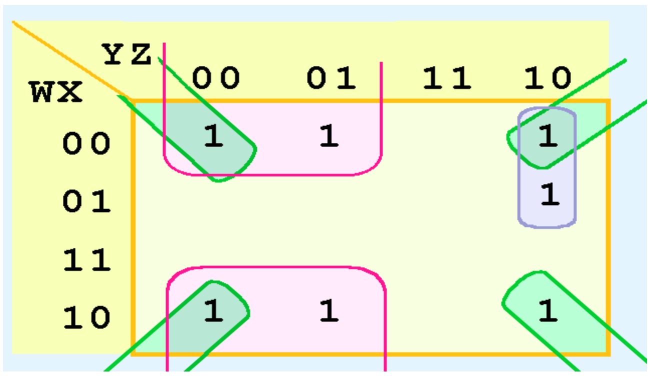


WX \ YZ	00	01	11	10
00	1	1		1
01				1
11				
10	1	1		1

Our three groups consist of:

- A purple group entirely within the Kmap at the right.

- – A pink group that wraps the top and bottom.
- – A green group that spans the corners.
- Thus we have three terms in our final function:



$$F(W,X,Y,Z) = \overline{W}\,\overline{Y} + \overline{X}\,\overline{Z} + \overline{W}Y\overline{Z}$$

For Group 1

F(W,X,Y,Z)=W’X’Y’Z’+W’X’Y’Z+WX’Y’Z’+WX’Y’Z

=W’X’Y’(Z+Z’)+WX’Y’(Z+Z’)

=W’X’Y’+WX’Y’

=X’Y’(W+W’)

=X’Y’

For Group II

F(W,X,Y,Z)=W’X’Y’Z’+W;X’YZ’+WX’Y’Z’+WX’YZ’

=W’X’Z’(Y+Y’)+WX’Z’(Y+Y’)

=W’X’Z’+WX’Z’

=X’Z’(W+W’)

=X’Z’

For Group III

F(W,X,Y,Z)=W’X’YZ’+W’XYZ’

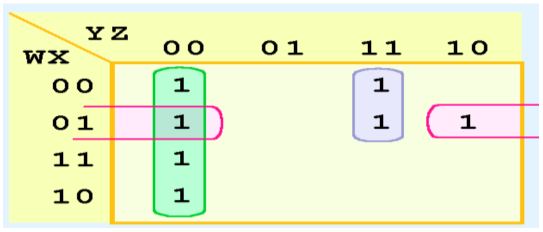
=W’YZ’(X+X’)

=WYZ’

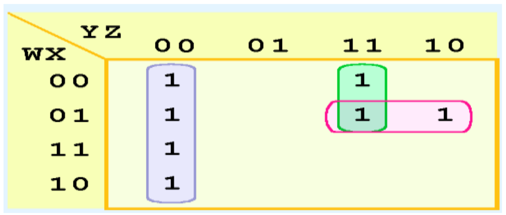
Combining all the three we get

F(W,X,Y,Z)=WYZ'+X'Y'+X'Z'

- It is possible to have a choice as to how to pick groups within a K-map, while keeping the groups as large as possible.
- The (different) functions that result from the groupings below are logically equivalent.



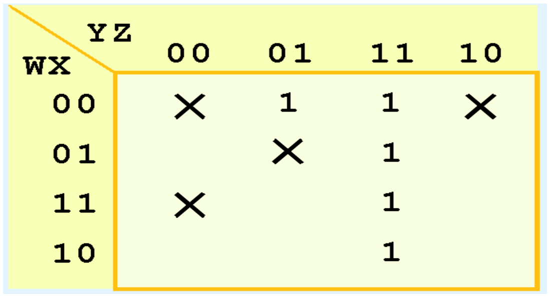
WX \ YZ	00	01	11	10
00	1		1	
01	1		1	1
11	1			
10	1			



WX \ YZ	00	01	11	10
00	1		1	
01	1		1	1
11	1			
10	1			

Don't Care Conditions

- Real circuits don't always need to have an output defined for every possible input.
  - For example, some calculator displays consist of 7-segment LEDs. These LEDs can display $2^7 - 1$ patterns, but only ten of them are useful.
- If a circuit is designed so that a particular set of inputs can never happen, we call this set of inputs a *don't care* condition.
- They are very helpful to us in K-map circuit simplification.
- In a K-map, a don't care condition is identified by an *X* in the cell of the minterm(s) for the don't care inputs, as shown below.
- In performing the simplification, we are free to include or ignore the *X*'s when creating our groups.



WX \ YZ	00	01	11	10
00	×	1	1	×
01		×	1	
11	×		1	
10			1	

- The truth table of:

$$\mathbf{F(W,X,Y,Z) = \overline{W}\,\overline{Y} + YZ}$$

differs from the truth table of:

$$F(W,X,Y,Z) = \bar{W}Z + YZ$$

- K-maps provide an easy graphical method of simplifying Boolean expressions.
- A K-map is a matrix consisting of the outputs of the minterms of a Boolean function.
- In this section, we have discussed 2- 3- and 4-input K-maps. This method can be extended to any number of inputs through the use of multiple tables.
- Recapping the rules of K-map simplification:
- Groupings can contain only 1s; no 0s.
- Groups can be formed only at right angles; diagonal groups are not allowed.
- The number of 1s in a group must be a power of 2 – even if it contains a single 1.
- The groups must be made as large as possible.
- Groups can overlap and wrap around the sides of the Kmap.
- Uses don't care conditions when you can.

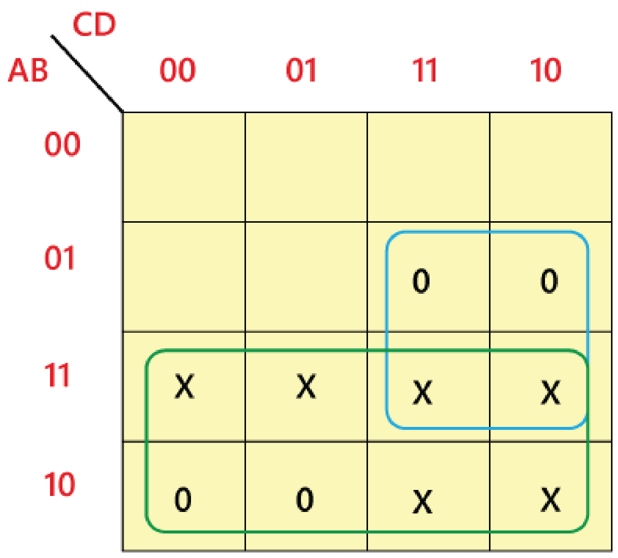
Example 1: Minimize F(A,B,C,D) = m(0,1,2,3,4,5) + d(10,11,12,13,14,15) in SOP minimal form

Solution:

The POS form of the given function is:

F(A,B,C,D) = M(6,7,8,9) + d(10,11,12,13,14,15)

The POS K-map for the given expression is:



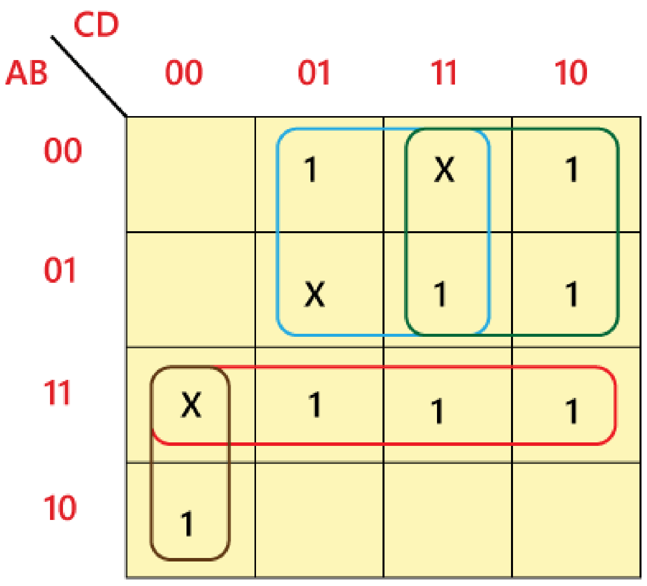
So, the minimized POS form of the function is:

F = A'(B' + C')

Example-2:
Minimize the following function in SOP minimal form using K-Maps: F(A, B, C, D) = m(1, 2, 6, 7, 8, 13, 14, 15) + d(3, 5, 12)

Explanation:

The SOP K-map for the given expression is:

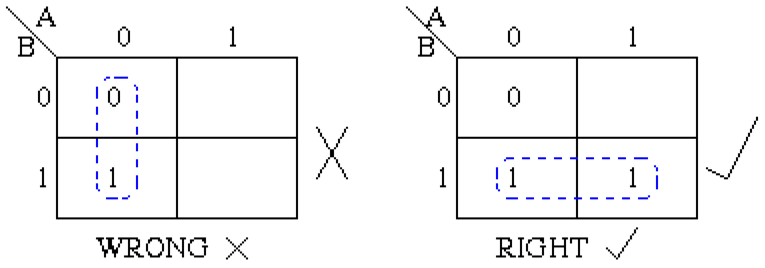


Therefore,

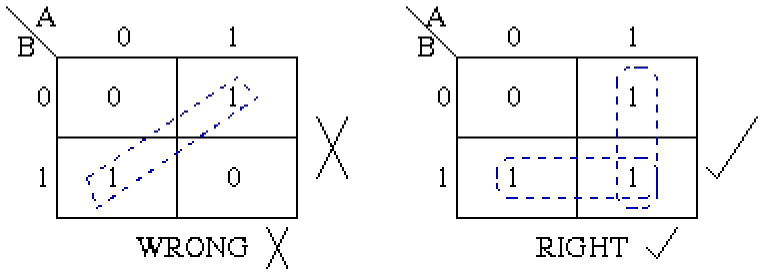
f = AC'D' + A'D + A'C + AB

Rules for k-Map Simplification

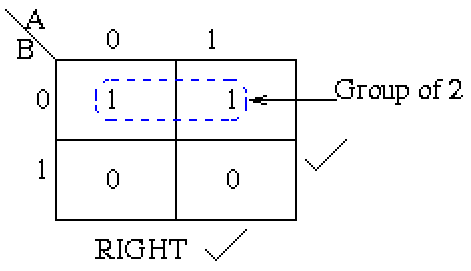
- **Groups may not include any cell containing a zero**

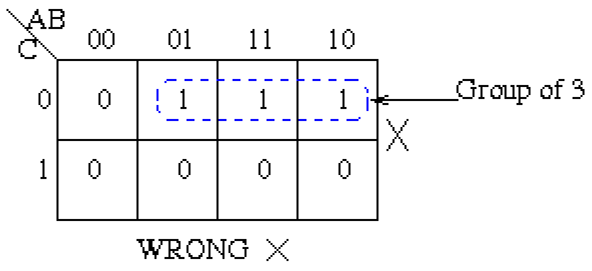


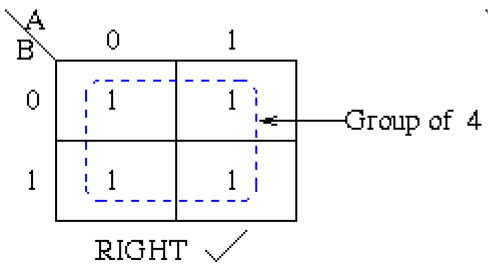
- **Groups may be horizontal or vertical, but not diagonal.**

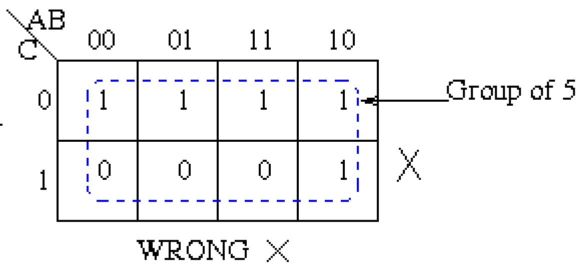


- **Groups must contain 1, 2, 4, 8, or in general 2^n cells.
  That is if n = 1, a group will contain two 1's since $2^1 = 2$.
  If n = 2, a group will contain four 1's since $2^2 = 4$.**

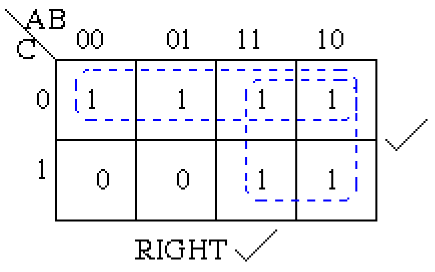


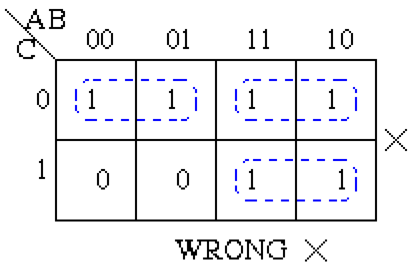






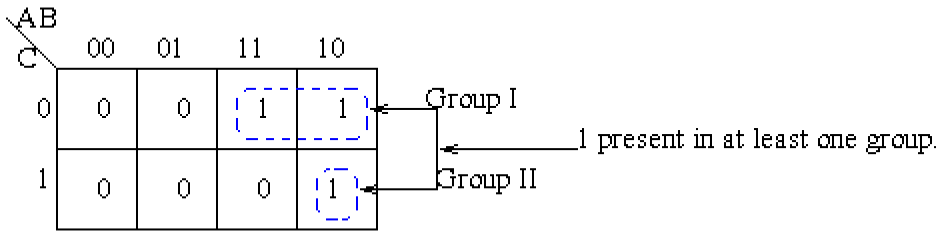
- **Each group should be as large as possible.**



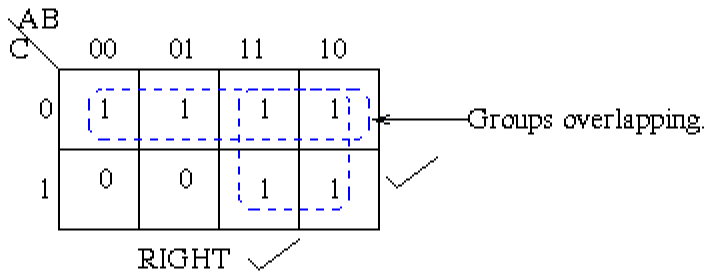


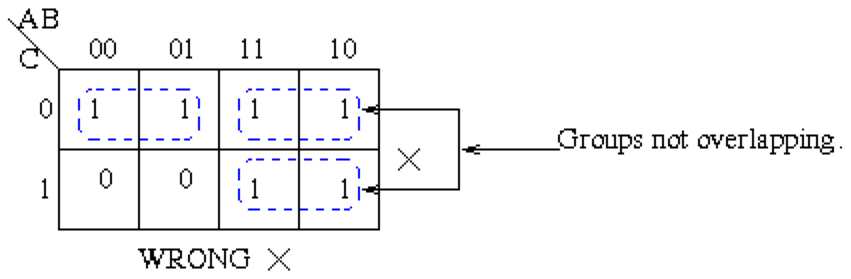
(Note that no Boolean laws broken, but not sufficiently minimal)

- **Each cell containing a *one* must be in at least one group.**

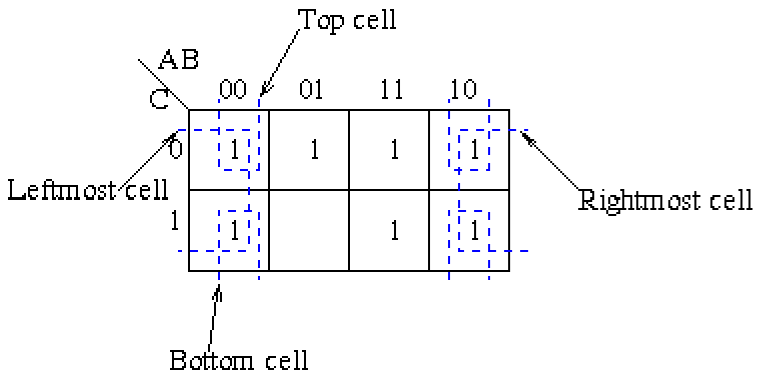


- **Groups may overlap.**

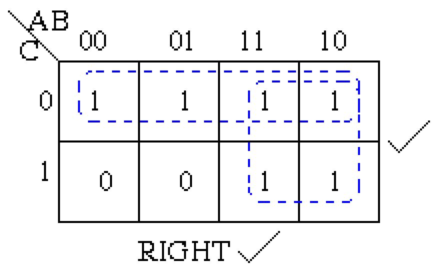


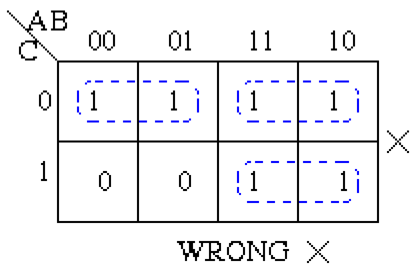


- **Groups may wrap around the table. The leftmost cell in a row may be grouped with the rightmost cell and the top cell in a column may be grouped with the bottom cell.**



- **There should be as few groups as possible, as long as this does not contradict any of the previous rules.**





Arithmetic Circuits

By combining logic circuits is a right way, we can build circuits that add and subtract binary numbers. Adder circuits are used to perform addition as well as subtraction.

Binary Addition

The binary addition operation works similarly to the base 10 decimal system, except that it is a base 2 system. The binary system consists of only two digits, 1 and 0. Most of the functionalities of the computer system use the binary number system. The binary code uses the digits 1's and 0's to make certain processes turn off or on. The process of the addition operation is very familiar to the decimal system by adjusting to the base 2.

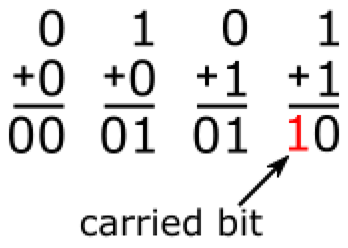
Rules of Binary Addition

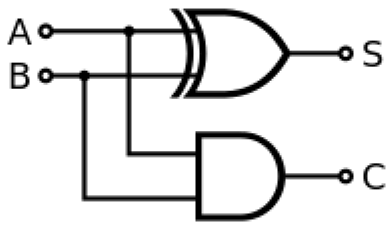
Binary addition is much easier than the decimal addition when you remember the following tricks or rules. Using these rules, any binary number can be easily added. The four rules of binary addition are:

- 0 + 0 = 0
- 0 + 1 = 1
- 1 + 0 = 1
- 1 + 1 =10

Half Adder and Full Adder

When adding two separate bits together there are four possible combinations. Each of these is shown





It can easily be seen that the bit in the right-hand column (the "ones" column) is a 1 only when the addends are different. XORing the addends together can therefore give us the right-hand bit. This bit is called the sum and is the modulo-2 sum of the addends (i.e. the solution if you loop round to zero again once you pass one).

The left-hand bit reads 1 only when both addends are 1, so an AND gate can be used to generate this bit, called the carried bit. As a summary:

The diagram to the left shows the complete half adder with the addends being represented by A and B, the sum represented by S and the carried bit represented by C.

The truth table is as follows (the number in the brackets are the weights of the bits - each addend is one, as is the sum - the carried bit is two)

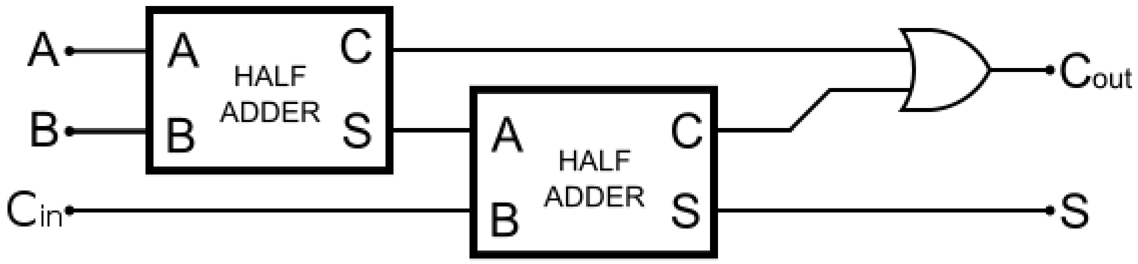
A (1)	B (1)		S (1)	C (2)
0	0		0	0
0	1		1	0
1	0		1	0
1	1		0	1

Full Adder

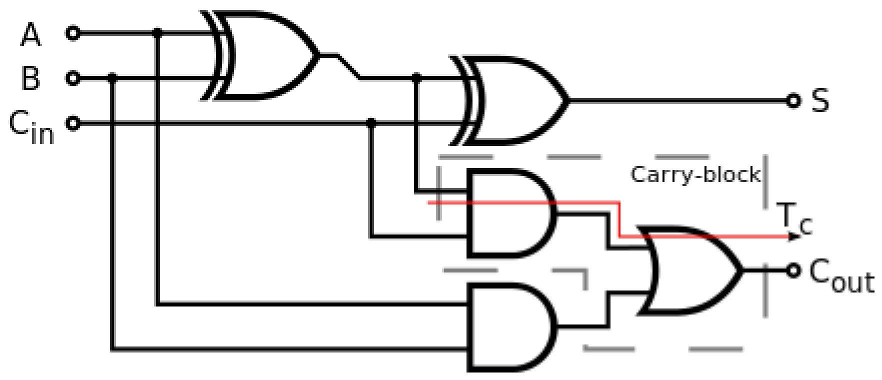
The downfall of half adders is that while they can generate a carry out output, they cannot deal with a carry in signal. This means that they can only ever be stand-alone units, and cited to add multiple bit numbers.

A full adder solves this problem by adding three numbers together - the two addends as in the half adder, and a carry in input.

A full adder can be constructed from two half adders by connecting A and B to the input of one half adder, connecting the sum from that to an input to the second adder, connecting the carry in, C_{in}, to the other input and ORing the two half adder carry outputs to give the final carry output, C_{out}.



The digram below shows a full adder at the gate level.

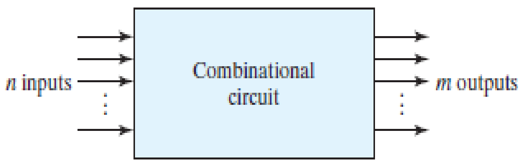


A version of this diagram showing the two half adder modules outlined is available here. The output of the full adder is the two-bit arithmetic sum of three one-bit numbers.

UNIT V: COMBINATIONAL CIRCUIT

COBINATIONAL CIRCUIT:

- A combinational circuit consists of logic gates whose outputs at any time are determined from only the present combination of inputs.
- A combinational circuit consists of an interconnection of logic gates.
- For *n* input variables, there are 2*n* possible combinations of the binary inputs.
- The diagram of a combinational circuit has logic gates with no feedback paths or memory elements.



ANALYSIS PROCEDURE

1. Determine the number of input variables in the circuit. For *n* inputs, form the 2*n* possible input combinations and list the binary numbers from 0 to (2*n* - 1) in a table.
2. Label the outputs of selected gates with arbitrary symbols.
3. Obtain the truth table for the outputs of those gates which are a function of the input variables only.
4. Proceed to obtain the truth table for the outputs of those gates which are a function of previously defined values until the columns for all outputs are determined.

DESIGN PROCEDURE:

1) From the specifications of the circuit, determine the required number of inputs and outputs and assign a symbol to each.
2) Derive the truth table that defines the required relationship between inputs and outputs.
3) Obtain the simplified Boolean functions for each output as a function of the input variables
4) Draw the logic diagram and verify the correctness of the design (manually or by simulation).

DECIMAL ADDER

A decimal adder requires a minimum of nine inputs and five outputs, since four bits are required to code each decimal digit and the circuit must have an input and output carry

BCD Adder :

- The adder will form the sum in *binary* and produce a result that ranges from 0 through 19.
- When the binary sum is equal to or less than 1001 to obtain an valid BCD representation.

  < 9 - Valid BCD

  > 9 - Invalid BCD

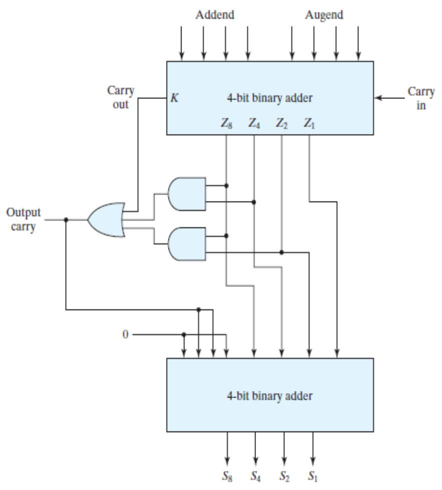
  (Add 06)

When the binary sum is greater than 1001, we obtain an invalid BCD representation. The addition of binary 6 (0110) to the binary sum converts it to the correct BCD representation

Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

Block diagram of a BCD adder

$$C = K + Z8Z4 + Z8Z2$$

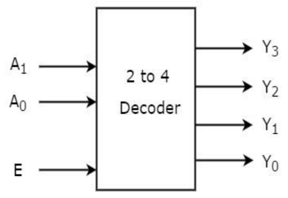


DECODERS:

- A *decoder* is a combinational circuit that converts binary information from *n* input lines to a maximum of 2*n* unique output lines. If the *n* -bit coded information has unused combinations, the decoder may have fewer than 2*n* outputs.

- The decoders presented here are called *n* -to- *m* -line decoders, where *m* … 2*n*. Their n purpose is to generate the 2*n* (or fewer)n minterms of *n* input variables. Each combination of inputs will assert a unique output. The name *decoder* is also used in conjunction with n other code converters, such as a BCD-to-seven-segment decoder.

2 to 4 Decoder : Let 2 to 4 Decoder has two inputs A_1 & A_0 and four outputs Y_3, Y_2, Y_1 & Y_0. The **block diagram** of 2 to 4 decoder is shown in the following figure.



One of these four outputs will be ‘1’ for each combination of inputs when enable, E is ‘1’. The **Truth table** of 2 to 4 decoder is shown below.

Enable	**Inputs**		**Outputs**			
E	**A_1**	**A_0**	**Y_3**	**Y_2**	**Y_1**	**Y_0**
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

From Truth table, we can write the **Boolean functions** for each output as

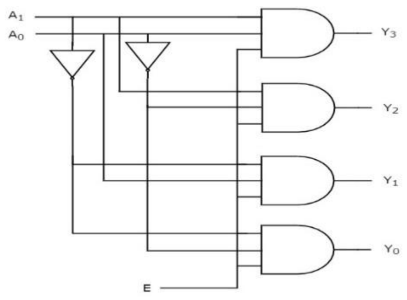
Y3=E.A1.A0Y3=E.A1.A0

Y2=E.A1.A0′Y2=E.A1.A0′

Y1=E.A1′.A0Y1=E.A1′.A0

Y0=E.A1′.A0′Y0=E.A1′.A0′

Each output is having one product term. So, there are four product terms in total. We can implement these four product terms by using four AND gates having three inputs each & two inverters. The **circuit diagram** of 2 to 4 decoder is shown in the following figure.



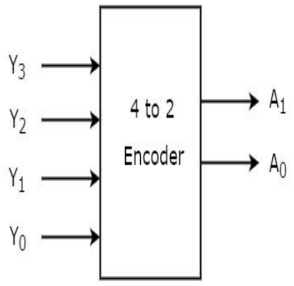
- Therefore, the outputs of 2 to 4 decoder are nothing but the **min terms** of two input variables A_1 & A_0, when enable, E is equal to one. If enable, E is zero, then all the outputs of decoder will be equal to zero.
- Similarly, 3 to 8 decoder produces eight min terms of three input variables A_2, A_1 & A_0 and 4 to 16 decoder produces sixteen min terms of four input variables A_3, A_2, A_1 & A_0.

ENCODERS

An **Encoder** is a combinational circuit that performs the reverse operation of Decoder. It has maximum of 2^n input lines and 'n' output lines. It will produce a binary code equivalent to the input, which is active High. Therefore, the encoder encodes 2^n input lines with 'n' bits. It is optional to represent the enable signal in encoders.

4 to 2 Encoder

Let 4 to 2 Encoder has four inputs Y_3, Y_2, Y_1 & Y_0 and two outputs A_1 & A_0. The **block diagram** of 4 to 2 Encoder is shown in the following figure.



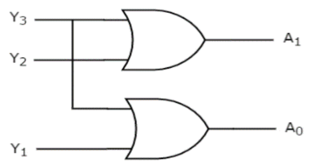
At any time, only one of these 4 inputs can be '1' in order to get the respective binary code at the output. The **Truth table** of 4 to 2 encoder is shown below.

Inputs				Outputs	
Y_3	**Y_2**	**Y_1**	**Y_0**	**A_1**	**A_0**
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

From Truth table, we can write the **Boolean functions** for each output as

A1=Y3+Y2A1=Y3+Y2

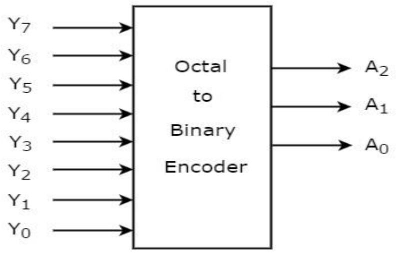
A0=Y3+Y1A0=Y3+Y1



The above circuit diagram contains two OR gates. These OR gates encode the four inputs with two bits

Octal to Binary Encoder

Octal to binary Encoder has eight inputs, Y_7 to Y_0 and three outputs A_2, A_1 & A_0. Octal to binary encoder is nothing but 8 to 3 encoder. The **block diagram** of octal to binary Encoder is shown in the following figure.



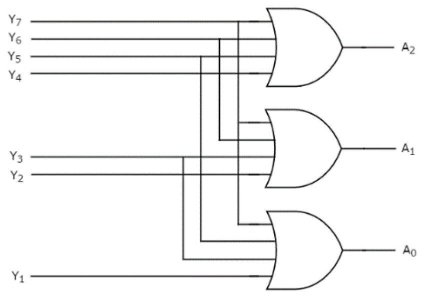
- At any time, only one of these eight inputs can be '1' in order to get the respective binary code. The **Truth table** of octal to binary encoder is shown below.
- From Truth table, we can write the **Boolean functions** for each output as

A2=Y7+Y6+Y5+Y4A2=Y7+Y6+Y5+Y4

A1=Y7+Y6+Y3+Y2A1=Y7+Y6+Y3+Y2

A0=Y7+Y5+Y3+Y1A0=Y7+Y5+Y3+Y1

- We can implement the above Boolean functions by using four input OR gates. The **circuit diagram** of octal to binary encoder is shown in the following figure.



The above circuit diagram contains three 4-input OR gates. These OR gates encode the eight inputs with three bits.

Drawbacks of Encoder

Following are the drawbacks of normal encoder.

- There is an ambiguity, when all outputs of encoder are equal to zero. Because, it could be the code corresponding to the inputs, when only least significant input is one or when all inputs are zero.
- If more than one input is active High, then the encoder produces an output, which may not be the correct code. For **example**, if both Y_3 and Y_6 are '1', then the encoder produces 111 at the output. This is neither equivalent code corresponding to Y_3, when it is '1' nor the equivalent code corresponding to Y_6, when it is '1'.

So, to overcome these difficulties, we should assign priorities to each input of encoder. Then, the output of encoder will be the binary code corresponding to the active High inputs, which has higher priority. This encoder is called as **priority encoder**.

Priority Encoder

A 4 to 2 priority encoder has four inputs Y_3, Y_2, Y_1 & Y_0 and two outputs A_1 & A_0. Here, the input, Y_3 has the highest priority, whereas the input, Y_0 has the lowest priority. In this case, even if more than one input is '1' at the same time, the output will be the binarybinary code corresponding to the input, which is having **higher priority**.

We considered one more **output, V** in order to know, whether the code available at outputs is valid or not.

- If at least one input of the encoder is '1', then the code available at outputs is a valid one. In this case, the output, V will be equal to 1.
- If all the inputs of encoder are '0', then the code available at outputs is not a valid one. In this case, the output, V will be equal to 0.

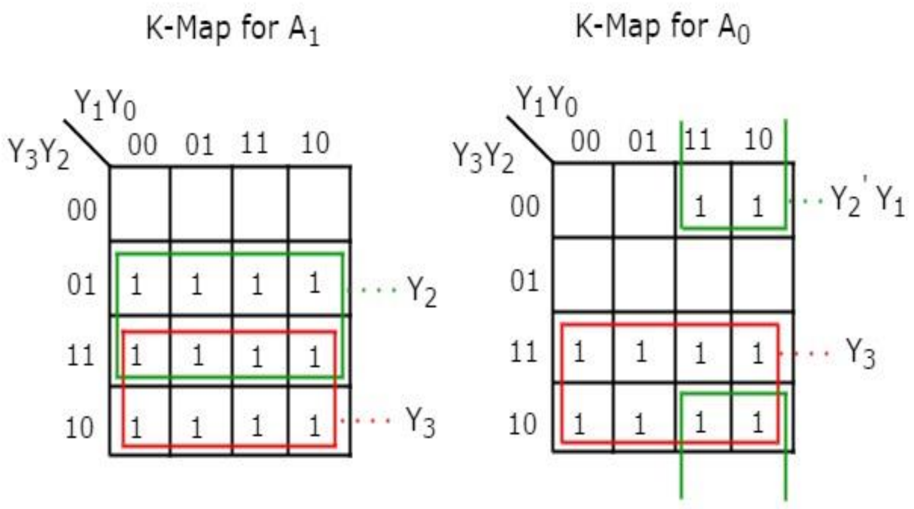
The **Truth table** of 4 to 2 priority encoder is shown below.

Y_3	**Y_2**	**Y_1**	**Y_0**	**A_1**	**A_0**	**V**
0	0	0	0	0	0	0

N. Baby Kala

0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

Use **4 variable K-maps** for getting simplified expressions for each output.



The simplified **Boolean functions** are

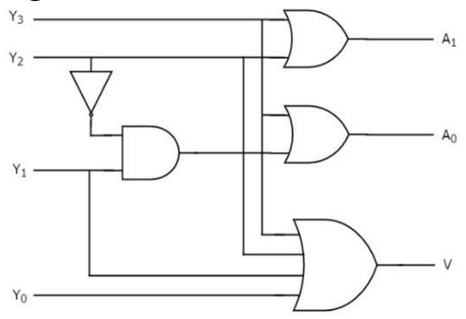
A1=Y3+Y2A1=Y3+Y2

A0=Y3+Y2′Y1A0=Y3+Y2′Y1

Similarly, we will get the Boolean function of output, V as

V=Y3+Y2+Y1+Y0V=Y3+Y2+Y1+Y0

- We can implement the above Boolean functions using logic gates. The **circuit diagram** of 4 to 2 priority encoder is shown in the following figure.



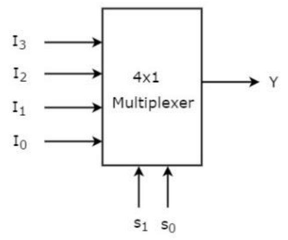
- The above circuit diagram contains two 2-input OR gates, one 4-input OR gate, one 2input AND gate & an inverter.
- Here AND gate & inverter combination are used for producing a valid code at the outputs, even when multiple inputs are equal to ‘1’ at the same time.
- Hence, this circuit encodes the four inputs with two bits based on the **priority** assigned to each input.

MULTIPLEXERS

- A multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line.
- The selection of a particular input line is controlled by a set of selection lines. Normally, there are 2*n* input lines and *n* selection lines whose bit combinations determine which input is selected.
- Since there are ‘n’ selection lines, there will be 2^n possible combinations of zeros and ones. So, each combination will select only one data input. Multiplexer is also called as **Mux**.

4x1 Multiplexer

4x1 Multiplexer has four data inputs I_3, I_2, I_1 & I_0, two selection lines s_1 & s_0 and one output Y. The **block diagram** of 4x1 Multiplexer is shown in the following figure.



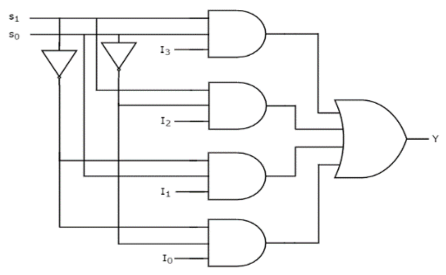
One of these 4 inputs will be connected to the output based on the combination of inputs present at these two selection lines. **Truth table** of 4x1 Multiplexer is shown below.

Selection Lines		Output
S_1	**S_0**	**Y**
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

From Truth table, we can directly write the **Boolean function** for output, Y as

$$Y=S1'S0'I0+S1'S0I1+S1S0'I2+S1S0I3Y=S1'S0'I0+S1'S0I1+S1S0'I2+S1S0I3$$

We can implement this Boolean function using Inverters, AND gates & OR gate. The **circuit diagram** of 4x1 multiplexer is shown in the following figure.



We can easily understand the operation of the above circuit. Similarly, you can implement 8x1 Multiplexer and 16x1 multiplexer by following the same procedure.

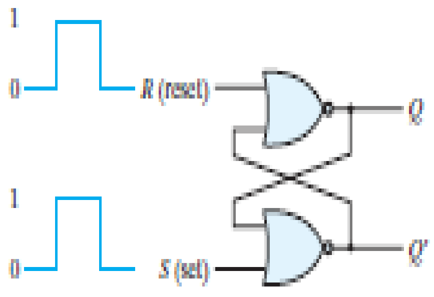
STORAGE ELEMENTS: LATCHES

A storage element in a digital circuit can maintain a binary state indefinitely until directed by an input signal to switch states.

- *Storage elements that operate with signal levels (rather than signal transitions) are referred to as latches; those controlled by a clock transition are flip-flops.* Latches are said to be level sensitive devices; flip-flops are edge-sensitive devices.
- The two types of storage elements are related because latches are the basic circuits from which all flip-flops are constructed.

SR Latch:

The *SR* latch is a circuit with two cross-coupled NOR gates or two cross-coupled NAND gates, and two inputs labelled *S* for set and *R* for reset. The *SR* latch constructed with two cross-coupled NOR gates is shown in Fig



(a) Logic diagram

S	R	Q	Q'	
1	0	1	0	
0	0	1	0	(after $S = 1, R = 0$)
0	1	0	1	
0	0	0	1	(after $S = 0, R = 1$)
1	1	0	0	(forbidden)

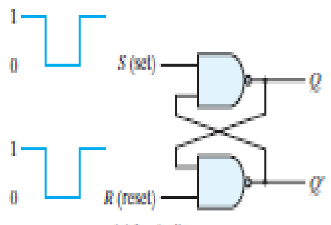
(b) Function table

The latch has two useful states.

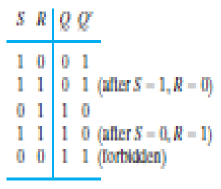
(1) When output $Q = 1$ and $\overline{Q} = 0$, the latch is said to be in the *set state* .
i
(2) When $Q = 0$ and $\overline{Q} = 1$, it is in the *reset state*.

(3) Outputs Q and $\overline{Q}$ are normally the complement of each other.

However, when both inputs are equal to 1 at the same time, a condition in which both outputs are equal to 0 (rather than be mutually complementary) occurs. If both inputs are then switched to 0 simultaneously, the device will enter an unpredictable or undefined state or a metastable state. Consequently, in practical applications, setting both inputs to 1 is forbidden.



(a) Logic diagram



S	*R*	*Q*	*Q'*	
1	0	0	1	
1	1	0	1	(after *S* = 1, *R* = 0)
0	1	1	0	
1	1	1	0	(after *S* = 0, *R* = 1)
0	0	1	1	(forbidden)

(b) Function table

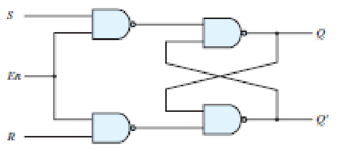
- It operates with both inputs normally at 1, unless the state of the latch has to be changed. The application of 0 to the *S* input causes output *Q* to go to 1, putting the latch in the set state.
- When the *S* input goes back to 1, the circuit remains in the set state.
- This action causes the circuit to go to the reset state and stay there even after both inputs return to 1.

- The condition that is forbidden for the NAND latch is both inputs being equal to 0 at the same time, an input combination that should be avoided.
- NAND require the complement of those values used for the NOR latch. Because the NAND latch requires a 0 signal to change its state, it is sometimes referred to as an *S*_*R*_ latch.
- The primes (or, sometimes, bars over the letters) designate the fact that the inputs must be in their complement form to activate the circuit.

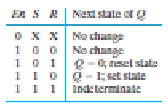
The operation of the basic *SR* latch can be modified by providing an additional input signal that determines (controls) *when* the state of the latch can be changed by determining whether *S* and *R* (or *S* _ and *R* _) can affect the circuit. It consists of the basic *SR* latch and two additional NAND gates. The control input *En* acts as an *enable* signal for the other two inputs. **The outputs of the NAND gates stay at the logic-1 level as long as the enable signal remains at 0.**

- This is the quiescent condition for the *SR* latch. When the enable input goes to 1, information from the *S* or *R* input is allowed to affect the latch.
- The set state is reached with *S* = 1, *R* = 0, and *En* = 1 (active-high enabled).
- To change to the reset state, the inputs must be *S* = 0, *R* = 1, and *En* = 1.
- In either case, when *En* returns to 0, the circuit remains in its current state.

The control input disables the circuit by applying 0 to *En*, so that the state of the output does not change regardless of the values of *S* and *R*. Moreover, when *En* = 1 and both the *S* and *R* inputs are equal to 0, the state of the circuit does not change. These conditions are listed in the function table accompanying the diagram.



(a) Logic diagram



En	*S*	*R*	Next state of *Q*
0	X	X	No change
1	0	0	No change
1	0	1	*Q* = 0; reset state
1	1	0	*Q* = 1; set state
1	1	1	Indeterminate

(b) Function table

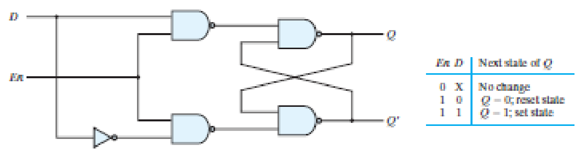
- An indeterminate condition occurs when all three inputs are equal to 1.
- This condition places 0's on both inputs of the basic *SR* latch, which puts it in the undefined state. When the enable input goes back to 0, one cannot conclusively determine the next state, because it depends on whether the *S* or *R* input goes to 0 first.

- This indeterminate condition makes this circuit difficult to manage.

D Latch (Transparent Latch)

One way to eliminate the undesirable condition of the indeterminate state in the *SR* latch is to ensure that inputs *S* and *R* are never equal to 1 at the same time. This is done in the *D* latch.

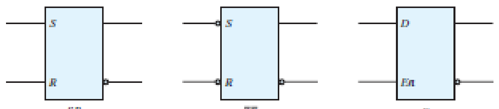
- This latch has only two inputs: *D* (data) and *En* (enable).
- The *D* input goes directly to the *S* input, and its complement is applied to the *R* input. As long as the enable input is at 0, the cross-coupled *SR* latch has both inputs at the 1 level and the circuit cannot change state regardless of the value of *D*.
- The *D* input is sampled when *En* = 1. If *D* = 1, the *Q* output goes to 1, placing the circuit in the set state.
- If *D* = 0, output *Q* goes to 0, placing the circuit in the reset state.
- The *D* latch receives that designation from its ability to hold *data* in its internal storage. It is suited for use as a temporary storage for binary information between a unit and its environment.



En	*D*	Next state of *Q*
0	X	No change
1	0	*Q* = 0; reset state
1	1	*Q* = 1; set state

(a) Logic diagram (b) Function table

The graphic symbols for the various latches are shown in Fig



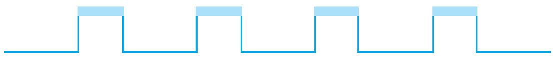
SR $\overline{SR}$ *D*

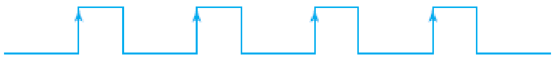
STORAGE ELEMENTS: FLIP-FLOPS

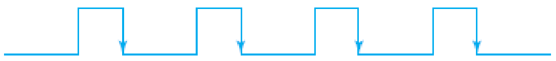
The state of a latch or flip-flop is switched by a change in the control input. This momentary change is called a *trigger*, and the transition it causes is said to trigger the flip-flop.

As long as the pulse input remains at this level, any changes in the data input will change the output and the state of the latch.

- When latches are used for the storage elements, a serious difficulty arises. If the inputs applied to the latches change while the clock pulse is still at the logic-1 level, the latches will respond to new values and a new output state may occur.
- The result is an unpredictable situation, Because of this unreliable operation, the output of a latch cannot be applied directly.
- Flip-flop circuits are to operate properly. The operation of a flip-flop is to trigger it only during a signal *transition*.
- A clock pulse goes through **two transitions**: from 0 to 1 and the return from 1 to 0. The positive transition is defined as the positive edge and the negative transition as the negative edge

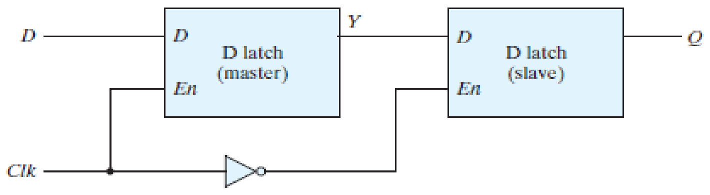

(a) Response to positive level


(b) Positive-edge response


(c) Negative-edge response

Edge-Triggered *D* Flip-Flop:

The construction of a *D* flip-flop with two *D* latches and an inverter is shown in Fig.

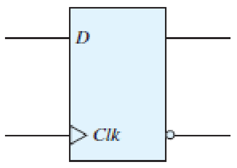


The first latch is called the master and the second the slave.

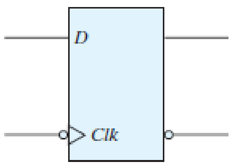
(i) When the clock is 1 the Master Latch is enable.
(ii) When the clock is 0, the output of the inverter is 1. The slave latch is enabled, and its output *Q* is equal to the master output *Y*.

- Any change in the input changes the master output at *Y*, but cannot affect the slave output.
- When the clock pulse returns to 0, the master is disabled and is isolated from the *D* input.
- At the same time, the slave is enabled and the value of *Y* is transferred to the output of the flip-flop at *Q* .
- Thus, *a change in the output of the flip-flop can be triggered only by and during the transition of the clock from 1* to 0.
- The output of the flip-flop is the value that was *stored in the master stage immediately before the negative edge occurred.*
- The negative edge of the clock affects the master and the positive edge affects the slave and the output terminal.

Graphic symbol for edge-triggered *D* flip-flop



(a) Positive-edge



(a) Negative-edge

It is similar to the symbol used for the *D* latch, except for the arrowhead-like symbol in front of the letter *Clk*, designating a *dynamic* input. The *dynamic indicator* (>) denotes the fact that the flip-flop responds to the edge transition of the clock. A bubble outside the block adjacent to the dynamic indicator designates a negative edge for triggering the circuit. The absence of a bubble designates a positive-edge response.

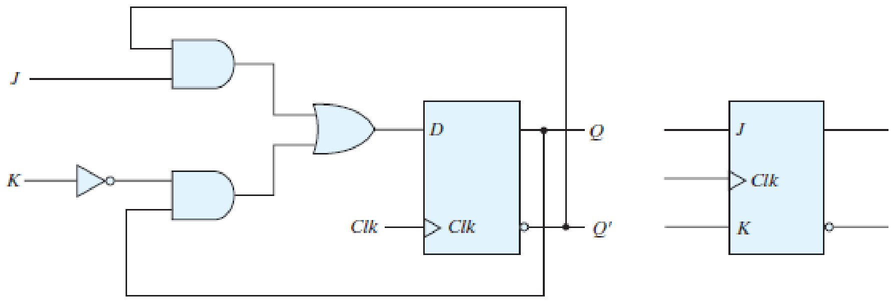
Other Flip-Flops

- Very large-scale integration circuits contain several thousands of gates within one package.
- Circuits are constructed by interconnecting the various gates to provide a digital system.
- Each flip-flop is constructed from an interconnection of gates.
- The most economical and efficient flip-flop constructed in this manner is the *edge-triggered D flip-flop*, because it requires the smallest number of gates.
- Other types of flip-flops can be constructed by using the *D* flip-flop and external logic. Two flip-flops less widely used in the design of digital systems are the *JK* and *T* flip-flops.

There are three operations that can be performed with a flip-flop:

(i) Set it to 1,
(ii) reset it to 0, or
(iii) complement its output.

With only a single input, the *D* flip-flop can set or reset the output, depending on the value of the *D* input immediately before the clock transition. Synchronized by a clock signal, the *JK* flip-flop has two inputs and performs all three operations. The circuit diagram of a *JK* flip-flop constructed with a *D* flip-flop and gates is shown in Fig.



(a) Circuit diagram (b) Graphic symbol

The *J* input sets the flip-flop to 1, the *K* input resets it to 0, and when both inputs are enabled, the output is complemented. This can be verified by investigating the circuit applied to the *D* input:

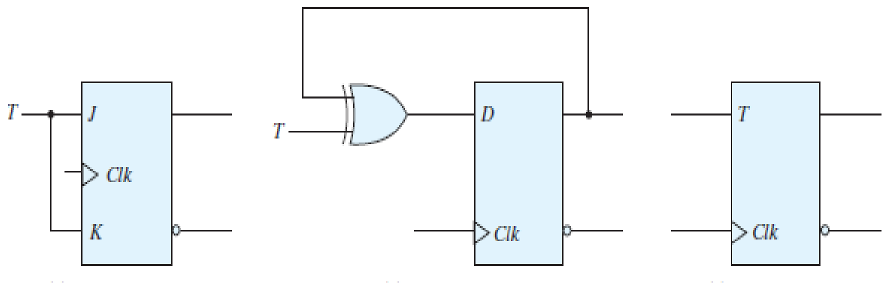
$$D = JQ\_ + K\_Q$$

- ➢ When $J = 1$ and $K = 0$, $D = Q\_ + Q = 1$, so the next clock edge sets the output to 1.
- ➢ When $J = 0$ and $K = 1$, $D = 0$, so the next clock edge resets the output to 0.
- ➢ When both $J = K = 1$ and $D = Q\_$, the next clock edge complements the output.
- ➢ When both $J = K = 0$ and $D = Q$, the clock edge leaves the output unchanged.

The *T* (toggle) flip-flop is a complementing flip-flop and can be obtained from a *JK* flip-flop when inputs *J* and *K* are tied together.

- When $T = 0$ ($J = K = 0$), a clock edge does not change the output.
- When $T = 1$ ($J = K = 1$), a clock edge complements the output.

The complementing flip-flop is useful for designing binary counters. The *T* flip-flop can be constructed with a *D* flip-flop and an exclusive-OR gate



(a) From *JK* flip-flop (b) From *D* flip-flop (c) Graphic symbol

The expression for the *D* input is

$$D = T \{ Q = TQ\_ + T\_Q$$

- When $T = 0$, $D = Q$ and there is no change in the output.
- When $T = 1$, $D = Q\_$ and the output complements.

Characteristic Tables

Table 5.1
Flip-Flop Characteristic Tables

***JK* Flip-Flop**

J	*K*	Q(*t* + 1)	
0	0	$Q(t)$	No change
0	1	0	Reset
1	0	1	Set
1	1	$Q'(t)$	Complement

***D* Flip-Flop**

D	Q(*t* + 1)	
0	0	Reset
1	1	Set

***T* Flip-Flop**

T	Q(*t* + 1)	
0	$Q(t)$	No change
1	$Q'(t)$	Complement

REGISTERS:

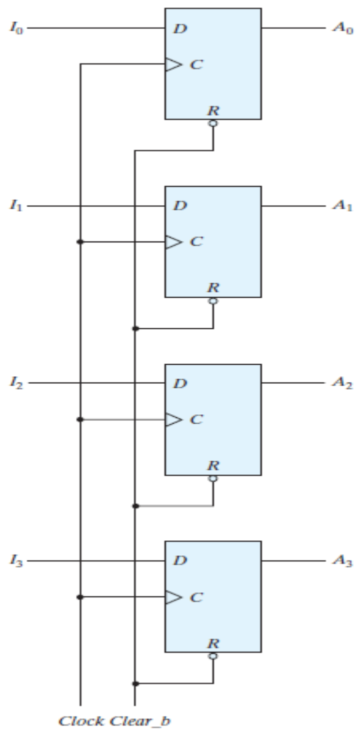
A clocked sequential circuit consists of a group of flip-flops.A *register* is a group of flip-flops, each one of which shares a common clock and is capable of storing one bit of information.

- An *n* -bit register consists of a group of *n* flip-flops capable of storing *n* bits of binary information.

- **Definition**: A register consists of a group of flip-flops together with gates that affect their operation. The flip-flops hold the binary information, and the gates determine how the information is transferred into the register.

- A *counter* is essentially a register that goes through a predetermined sequence of n binary states. The gates in the counter are connected in such a way as to produce the prescribed sequence of states.

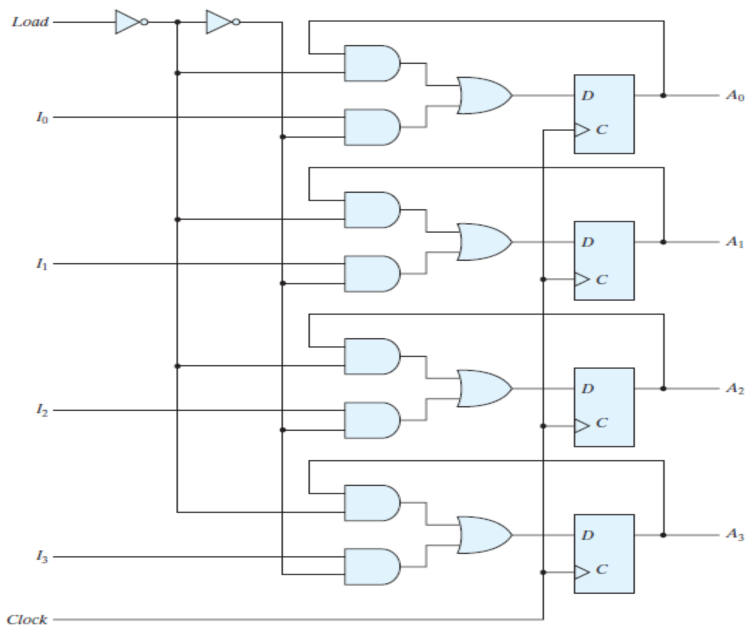
- Various types of registers are available commercially. The simplest register is one that consists of only flip-flops, without any gates. A register constructed with four *D* -type flip-flops to form a four-bit data storage register.



- The common clock input triggers all flip-flops on the positive edge of each pulse, and the binary data available at the four inputs are transferred into the register.
- The value of (*I* 3 , *I* 2 , *I* 1 , *I* 0) immediately before the clock edge determines the value of (*A* 3 , *A* 2 , *A* 1 , *A* 0) after the clock edge.
- The four outputs can be sampled at any time to obtain the binary information stored in the register.
- The input *Clear_b* goes to the active-low *R* (reset) input of all four flip-flops.
- When this input goes to 0, all flip-flops are reset asynchronously.
- The *Clear_b* input is useful for clearing the register to all 0's prior to its clocked operation.
- The *R* inputs must be maintained at logic 1 (i.e., de-asserted) during normal clocked operation.
- Note that, depending on the flip-flop, either *Clear*, *Clear_b*, *reset*, or *reset_b* can be used to indicate the transfer of the register to an all 0's state.

Register with Parallel Load:

- Registers with parallel load are a fundamental building block in digital systems.
- Synchronous digital systems have a master clock generator that supplies a continuous train of clock pulses.
- The pulses are applied to all flip-flops and registers in the system.
- The master clock acts like a drum that supplies a constant beat to all parts of the system.
- A separate control signal must be used to decide which register operation will execute at each clock pulse.
- The transfer of new information into a register is referred to as *loading* or *updating* the register.
- If all the bits of the register are loaded simultaneously with a common clock pulse, this loading is done *in parallel*. A clock edge applied to the *C* inputs of the register will load all four inputs in parallel.
- In this configuration, if the contents of the register must be left unchanged, the inputs must be held constant or the clock must be inhibited from the circuit.
- A four-bit data-storage register with a load control input that is directed through gates and into the *D* inputs of the flip-flops.



The additional **gates** implement a two-channel mux whose output drives the input to the register with either the data bus or the output of the register.

The **load input** to the register determines the action to be taken with each clock pulse.

i. When the load input is 1, the data at the four external inputs are transferred into the register with the next positive edge of the clock.
ii. When the load input is 0, the outputs of the flip-flops are connected to their respective inputs.

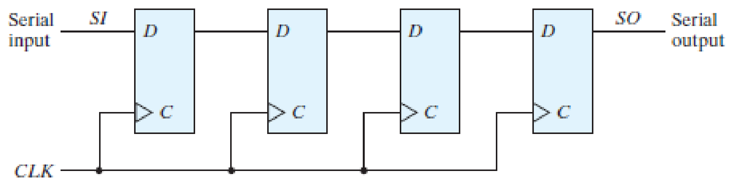
- The **feedback connection** from output to input is necessary because a *D* flip-flop does not have a “no change” condition. With each clock edge, the *D* input determines the next state of the register.

- The transfer of information from the data inputs or the outputs of the register is done simultaneously with all four bits in response to a clock edge.

SHIFT REGISTERS

- A register capable of shifting the binary information held in each cell to its neighbouring cell, in a selected direction, is called a *shift register*.
- The output of one flip-flop connected to the input of the next flip-flop.
- All flip-flops receive common clock pulses, which activate the shift of data from one stage to the next.

Simple Shift Register:



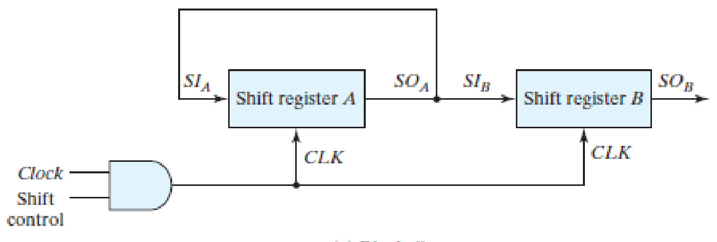
- The output of a given flip-flop is connected to the *D* input of the flip-flop at its right.
- This shift register is unidirectional (left-to-right). Each clock pulse shifts the contents of the register one bit position to the right. The configuration does not support a left shift.
- The *serial input* determines what goes into the leftmost flip-flop during the shift. The *serial output* is taken from the output of the rightmost flip-flop.

Types of Flip Flop: There are four types of Flip Flop transfer

i) SISO (Serial In Serial Out)
ii) SIPO (Serial In Parallel Out)
iii) PISO (Parallel In Serial Out)
iv) PIPO (Parallel In Parallel Out)

Serial Transfer

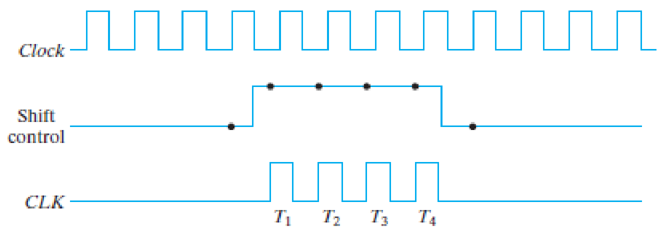
- The data path of a digital system is said to operate in serial mode when information is transferred and manipulated one bit at a time.
- Information is transferred one bit at a time by shifting the bits out of the source register and into the destination register.
- This type of transfer is in contrast to parallel transfer.
- The serial transfer of information from register *A* to register *B* is done with shift registers.



- The serial output (*SO*) of register *A* is connected to the serial input (*SI*) of register *B.*
- To prevent the loss of information stored in the source register, the information in register *A* is made to circulate by connecting the serial output to its serial input.
- The initial content of register *B* is shifted out through its serial output and is lost unless it is transferred to a third shift register.

The **shift control input** determines when and how many times the registers are shifted.

- The shift control signal is synchronized with the clock and changes value just after the negative edge of the clock. The next four clock pulses find the shift control signal in the active state, so the output of the AND gate connected to the *CLK* inputs produces four pulses: *T*1, *T*2, *T*3, and *T*4.
- Each rising edge of the pulse causes a shift in both registers. The fourth pulse changes the shift control to 0, and the shift registers are disabled.



(b) Timing diagram

- Assume that the binary content of *A* before the shift is 1011 and that of *B* is 0010.
- The serial transfer from *A* to *B* occurs in four steps, as shown in Table. With the first pulse, *T*1, the rightmost bit of *A* is shifted into the leftmost bit of *B* and is also circulated into the leftmost position of *A.*
- The contents of *A* are copied into *B*, so that the contents of *A* remain unchanged i.e., the contents of *A* are restored to their original value.

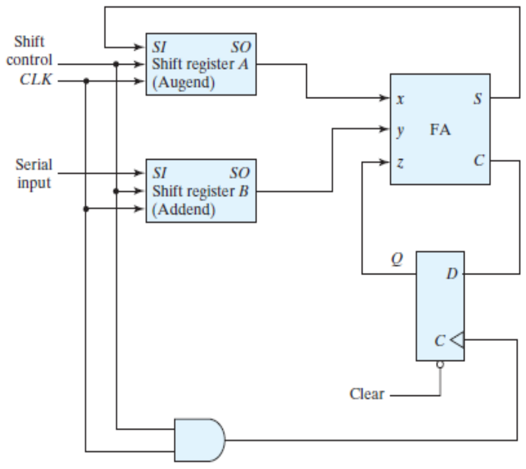
Serial-Transfer Example

Timing Pulse	Shift Register *A*	Shift Register *B*
Initial value	1 0 1 1	0 0 1 0
After T_1	1 1 0 1	1 0 0 1
After T_2	1 1 1 0	1 1 0 0
After T_3	0 1 1 1	0 1 1 0
After T_4	1 0 1 1	1 0 1 1

- The difference between the serial and the parallel mode of operation should be apparent from this example.
- In the parallel mode, information is available from all bits of a register and all bits can be transferred simultaneously during one clock pulse.
- The registers have a single serial input and a single serial output. The information is transferred one bit at a time while the registers are shifted in the same direction.

Serial Addition

Operations in digital computers to perform parallel operation are faster and Serial operations are slower because a datapath operation takes several clock cycle. But serial operations have the advantage of requiring fewer hardware components.



- The operation of the serial adder is as follows: Initially, register *A* holds the augend, register *B* holds the addend, and the carry flip-flop is cleared to 0.
- The outputs (*SO*) of *A* and *B* provide a pair of significant bits for the full adder at *x* and *y*. Output *Q* of the flip-flop provides the input carry at *z*.
- The shift control enables both registers and the carry flip-flop, so at the next clock pulse, both registers are shifted once to the right, the sum bit from *S* enters the leftmost flip-flop of *A*, and the output carry is transferred into flip-flop *Q*.
- The shift control enables the registers for a number of clock pulses equal to the number of bits in the registers. For each succeeding clock pulse, a new sum bit is transferred to *A*, a new carry is transferred to *Q*, and both registers are shifted once to the right. This process continues until the shift control is disabled.

Comparing the serial adder with the parallel adder:

- The parallel adder uses registers with a parallel load, whereas the serial adder uses shift registers.
- The number of full-adder circuits in the parallel adder is equal to the number of bits in the binary numbers, whereas the serial adder requires only one full-adder circuit and a carry flip-flop.
- The parallel adder is a combinational circuit, whereas the serial adder is a sequential circuit which consists of a full adder and a flip-flop that stores the output carry.

Universal Shift Register

If a parallel load capability is added to a shift register, then data entered in parallel can be taken out in serial fashion by shifting the data stored in the register.

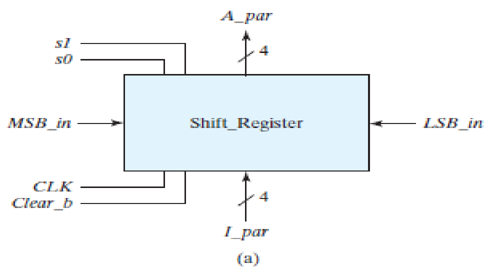
Shift register has the following capabilities:

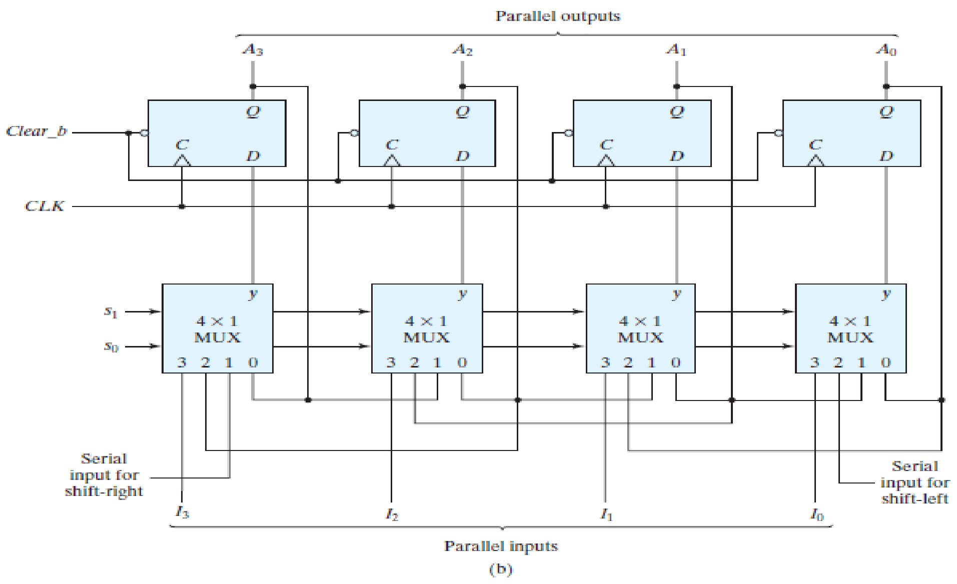
1. A *clear* control to clear the register to 0.
2. A *clock* input to synchronize the operations.

3. A *shift-right* control to enable the shift-right operation and the *serial input* and *output* lines associated with the shift right.
4. A *shift-left* control to enable the shift-left operation and the *serial input* and *output* lines associated with the shift left.
5. A *parallel-load* control to enable a parallel transfer and the *n* input lines associated with the parallel transfer.
6. *n* parallel output lines.
7. A control state that leaves the information in the register unchanged in response to the clock. Other shift registers may have only some of the preceding functions, with at least one shift operation.

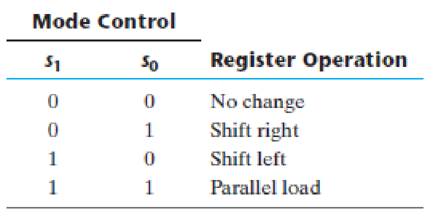
A register capable of shifting in one direction only is a *unidirectional* shift register. One that can shift in both directions is a *bidirectional* shift register. If the register has both shifts and parallel-load capabilities, it is referred to as a *universal shift register.*

The block diagram *for Universal Shift Register:*





- The circuit consists of four *D* flip-flops and four multiplexers.
- The four multiplexers have two common selection inputs *s*1 and *s*0.
- Input 0 in each multiplexer is selected when *s*1*s*0 = 00, input 1 is selected when *s*1*s*0 = 01, and similarly for the other two inputs.
- The selection inputs control the mode of operation of the register according to the function entries



Mode Control		
s_1	s_0	**Register Operation**
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load

- The transmitter accepts the *n* -bit data in parallel into a shift register and then transmits the data serially along the common line. The receiver accepts the data serially into a shift register.
- When all *n* bits are received, they can be taken from the outputs of the register in parallel.
- Thus, the transmitter performs a parallel-to-serial conversion of data and the receiver does a serial-to-parallel conversion.

Counters:

A register that goes through a prescribed sequence of states upon the application of input pulses is called a *counter*. The input pulses may be clock pulses, or they may originate from some external source and may occur at a fixed interval of time or at random.

Binary Counters:

The sequence of states may follow the binary number sequence or any other sequence of states. A counter that follows the binary number sequence is called a *binary counter*. An *n*-bit binary counter consists of *n* flip-flops and can count in binary from 0 through 2^n - 1.
Counters are available in two categories:

- Ripple counters
- Synchronous counters

Ripple Counters

In a ripple counter, a flip-flop output transition serves as a source for triggering other flip-flops. In other words, the *C* input of some or all flip-flops are triggered, not by the common clock pulses, but rather by the transition that occurs in other flip-flop outputs.
Two Categories of Ripple Counters are:

- ❖ Binary Ripple Counters
- ❖ BCD Ripple Counters

Binary Ripple Counter

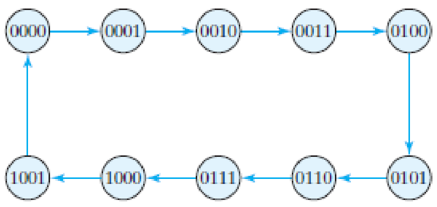
A binary ripple counter consists of a series connection of complementing flip-flops, with the output of each flip-flop connected to the *C* input of the next higher order flip-flop. The flip-flop holding the least significant bit receives the incoming count pulses. A complementing flip-flop can be obtained from a *JK* flip-flop with the *J* and *K* inputs tied together or from a *T* flip-flop. A third possibility is to use a *D* flip-flop with the complement output connected to the *D* input. In this way, the *D* input is always the complement of the present state, and the next clock pulse will cause the flip-flop to complement.

- The output of each flip-flop is connected to the *C* input of the next flip-flop in sequence. The flip-flop holding the least significant bit receives the incoming count pulses.

- The *T* inputs of all the flip-flops in (a) are connected to a permanent logic 1, making each flip- flop complement if the signal in its *C* input goes through a negative transition.
- The bub- ble in front of the dynamic indicator symbol next to *C* indicates that the flip-flops respond to the negative-edge transition of the input.
- The negative transition occurs when the output of the previous flip-flop to which *C* is connected goes from 1 to 0.
- The count starts with binary 0 and increments by 1 with each count pulse input. After the count of 15, the counter goes back to 0 to repeat the count. The least significant bit, A_0, is complemented with each count pulse input. Every time that A_0 goes from 1 to 0, it complements A_1.
- Every time that A_1 goes from 1 to 0, it complements A_2. Every time that A_2 goes from 1 to 0, it complements A_3, and so on for any other higher order bits of a ripple counter.

BCD Ripple Counter

- A decimal counter follows a sequence of 10 states and returns to 0 after the count of 9. Such a counter must have at least four flip-flops to represent each decimal digit, since a decimal digit is represented by a binary code with at least four bits.
- The sequence of states in a decimal counter is dictated by the binary code used to represent a decimal digit. If BCD is used, the sequence of states is as shown in the state diagram .
- A decimal counter is similar to a binary counter, except that the state after 1001 (the code for decimal digit 9) is 0000 (the code for decimal digit 0).

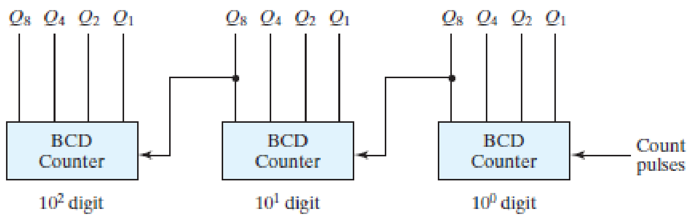


BCD Ripple Counter

Q_1 changes state after each clock pulse. Q_2 complements every time Q_1 goes from 1 to 0, as long as $Q_8 = 0$. When Q_8 becomes 1, Q_2 remains at 0. Q_4 complements every time Q_2 goes from 1 to 0. Q_8 remains at 0 as long as Q_2 or Q_4 is 0. When both Q_2 and Q_4 become 1, Q_8 complements when Q_1 goes from 1 to 0. Q_8 is cleared on the next transition of Q_1.

Three-decade BCD Counter

The BCD counter is a *decade* counter, since it counts from 0 to 9. To count in decimal from 0 to 99, we need a two-decade counter. To count from 0 to 999, we need a three-decade counter. Multiple decade counters can be constructed by connecting BCD counters in cascade, one for each decade. The inputs to the second and third decades come from Q_8 of the previous decade.When Q_8 in one decade goes from 1 to 0, it triggers the count for the next higher order decade while its own decade goes from 9 to 0.



Synchronous Counters

- Synchronous counters are different from ripple counters in that clock pulses are applied to the inputs of all flip-flops.
- A common clock triggers all flip-flops simultaneously, rather than one at a time in succession as in a ripple counter.

The decision whether a flip-flop is to be complemented is determined from the values of the data inputs, such as T or J and K at the time of the clock edge. If $T = 0$ or $J = K = 0$, the flip-flop does not change state. If $T = 1$ or $J = K = 1$, the flip-flop complements

Binary Counter

- The design of a synchronous binary counter is so simple that there is no need to go through a sequential logic design process.
- In a synchronous binary counter, the flip-flop in the least significant position is complemented with every pulse.
- *A flip-flop in any other position is complemented when all the bits in the lower significant positions are equal to 1*

For example, if the present state of a four-bit counter is $A_3A_2A_1A_0 = 0011$, the next count is 0100. A_0 is always complemented. A_1 is complemented because the present state of $A_0 = 1$. A_2 is complemented because the present state of $A_1A_0 = 11$. However, A_3 is not complemented, because the present state of $A_2A_1A_0 = 011$, which does not give an all-1's condition

- The C inputs of all flip-flops are connected to a common clock. The counter is enabled by *Count_enable*.
- If the enable input is 0, all J and K inputs are equal to 0 and the clock does not change the state of the counter.

- The first stage, A_0, has its J and K equal to 1 if the counter is enabled. The other J and K inputs are equal to 1 if all previous least significant stages are equal to 1 and the count is enabled.
- The chain of AND gates generates the required logic for the J and K inputs in each stage.
- The counter can be extended to any number of stages, with each stage having an additional flip-flop and an AND gate that gives an output of 1 if all previous flip-flop outputs are 1.

Up-Down Binary Counter:

- A synchronous countdown binary counter goes through the binary states in reverse order, from 1111 down to 0000 and back to 1111 to repeat the count.
- It is possible to design a countdown counter in the usual manner, but the result is predictable by inspection of the downward binary count.
- The bit in the least significant position is complemented with each pulse.
- *A bit in any other position is complemented if all lower significant bits are equal to 0.*
- For example, the next state after the present state of 0100 is 0011. The least significant bit is always complemented. The second significant bit is complemented because the first bit is 0. The third significant bit is complemented because the first two bits are equal to 0. But the fourth bit does not change, because not all lower significant bits are equal to 0.

  FIG

BCD Counter

- A BCD counter counts in binary-coded decimal from 0000 to 1001 and back to 0000. Because of the return to 0 after a count of 9, a BCD counter does not have a regular pattern, unlike a straight binary count.
- To derive the circuit of a BCD synchronous counter, it is necessary to go through a sequential circuit design procedure.
- The input conditions for the T flip-flops are obtained from the present- and next-state conditions. Also shown in the table is an output y, which is equal to 1 when the present state is 1001. In this way, y can enable the count of the next-higher significant decade while the same pulse switches the present decade from 1001 to 0000.
- The flip-flop input equations can be simplified by means of maps. The unused states for minterms 10 to 15 are taken as don't-care terms. The simplified functions are
- $T_{Q1} = 1$
- $T_{Q2} = Q'_8Q_1$ $T_{Q4} = Q_2Q_1$
- $T_{Q8} = Q_8Q_1 + Q_4Q_2Q_1$ $y = Q_8Q_1$

BCD Counter with parallel load

- Counters employed in digital systems quite often require a parallel-load capability for transferring an initial binary number into the counter prior to the count operation.
- When equal to 1, the input load control disables the count operation and causes a transfer of data from the four data inputs into the four flip-flops. If both control inputs are 0, clock pulses do not change the state of the register.

Load input of 1 causes a transfer from inputs I_0 - I_3 into the register during a positive edge of *CLK*. The input data are loaded into the register regardless of the value of the *Count* input, because the *Count* input is inhibited when the *Load* input is enabled. The *Load* input must be 0 for the *Count* input to control the operation of the counter.

BCD Counter with parallel load

- *The Count* control is set to 1 to enable the count through the *CLK* input. Also, recall that the *Load* control inhibits the count and that the clear operation is independent of other control inputs.

Other Counters

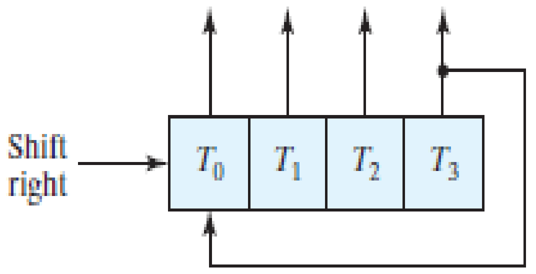
- Counters can be designed to generate any desired sequence of states.
- A divide-by-*N* counter (also known as a modulo-*N* counter) is a counter that goes through a repeated sequence of *N* states.
- The sequence may follow the binary count or may be any other arbitrary sequence.
- Counters are used to generate timing signals to control the sequence of operations in a digital system.
- Counters can also be constructed by means of shift registers.

Counter with Unused States

- A circuit with n flip-flops has 2^n binary states. There are occasions when a sequential circuit uses fewer than this maximum possible number of states. States that are not used
- In simplifying the input equations, the unused states may be treated as don't-care conditions or may be assigned specific next states.
- The count has a repeated sequence of six states, with flip-flops *B* and *C* repeating the binary count 00, 01, 10, and flip-flop *A* alternating between 0 and 1 every three counts.
- Two states, 011 and 111, are not included in the count.
- If the circuit ever goes to one of the unused states because of outside interference, the next count pulse transfers it to one of the valid states and the circuit continues to count correctly.
- Thus, the counter is self-correcting. In a self-correcting counter, if the counter happens to be in one of the unused states, it eventually reaches the normal count sequence after one or more clock pulses.
- An alternative design could use additional logic to direct every unused state to a specific next state.

Ring Counter

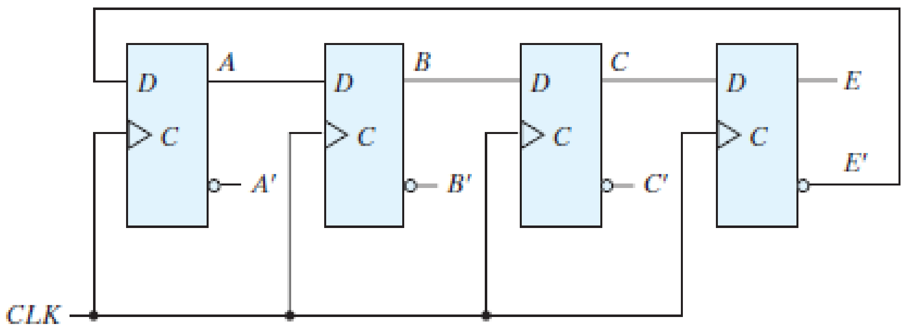
- A *ring counter* is a circular shift register with only one flip-flop being set at any particular time; all others are cleared.
- The single bit is shifted from one flip-flop to the next to produce the sequence of timing signals.
- The initial value of the register is 1000 and requires Preset/Clear flip-flops.
- The single bit is shifted right with every clock pulse and circulates back from T_3 to T_0. Each flip-flop is in the 1 state once every four clock cycles and produces one of the four timing signals.



(a) Ring-counter (initial value = 1000)

Johnson Counter

- A *k*-bit ring counter circulates a single bit among the flip-flops to provide *k* distinguish- able states.
- The number of states can be doubled if the shift register is connected as a *switch-tail* ring counter.
- A switch-tail ring counter is a circular shift register with the complemented output of the last flip-flop connected to the input of the first flip-flop.



(a) Four-stage switch-tail ring counter