UNIT-2

PART-1

Digital Logic Circuits-I:

2.1.1 Basic Logic Functions,

2.1.2Logic gates, 2.1.3universal logic gates,
2.1.4Minimization of Logic expressions. 2.1.5Flip-flops,

2.1.1 **Boolean algebra:**

Boolean algebra, like any other deductive mathematical system, may be defined with aset of elements, a set of operators, and a number of unproved axioms or postulates. A *set* of elements is anycollection of objects having a common property. If **S** is a set and ***x*** and ***y*** are certain objects, then **x Î S**denotes that ***x*** is a member of the set **S**, and ***y*** Ï**S** denotes that ***y*** is not an element of **S**. A set with adenumerable number of elements is specified by braces: **A** = {1,2,3,4}, *i.e.* the elements of set **A** are thenumbers 1, 2, 3, and 4. A *binary operator* defined on a set S of elements is a rule that assigns to each pair ofelements from S a unique element from S._ Example: In *a*b=c*, we say that * is a binary operator if it specifies a rule for finding *c* from the pair (*a*,*b*)and also if *a*, *b*, *c* Î S.

CLOSURE: The Boolean system is *closed* with respect to a binary operator if for every pair of Boolean values,it produces a Boolean result. For example, logical AND is closed in the Boolean system because it accepts only Boolean operands and produces only Boolean results.

_ A set *S* is closed with respect to a binary operator if, for every pair of elements of *S*, the binary operator specifies a rule for obtaining a unique element of *S*.

_ For example, the set of natural numbers N = {1, 2, 3, 4, … 9} is closed with respect to the binary operator plus (+) by the rule of arithmetic addition, since for any *a*, *b* Î N we obtain a unique *c* Î N by the operation *a* + *b* = c.

ASSOCIATIVE LAW:

A binary operator * on a set *S* is said to be associative whenever (*x* * *y*) * *z* = *x* * (*y* * *z*) for all *x*, *y*, *z* Î S, forall Boolean values x, y and z.

COMMUTATIVE LAW:

A binary operator * on a set *S* is said to be commutative whenever x * *y* = *y* * *x* for all *x*, *y*, *z* ϵ S

IDENTITY ELEMENT:

A set ***S*** is said to have an identity element with respect to a binary operation * on S if there exists an element $e \epsilon$ S with the property $e * x = x * e = x$ for every $x \epsilon$ S

BASIC IDENTITIES OF BOOLEAN ALGEBRA

- *Postulate 1 (Definition)*: A Boolean algebra is a closed algebraic system containing a set *K* of two or more elements and the two operators · and + which refer to logical AND and logical OR

- $x + 0 = x$
- $x \cdot 0 = 0$
- $x + 1 = 1$
- $x \cdot 1 = 1$
- $x + x = x$
- $x \cdot x = x$
- $x + x' = x$
- $x \cdot x' = 0$
- $x + y = y + x$
- $xy = yx$
- $x + (y + z) = (x + y) + z$
- $x (yz) = (xy) z$
- $x (y + z) = xy + xz$
- $x + yz = (x + y)(x + z)$
- $(x + y)' = x' y'$
- $(xy)' = x' + y'$
- $(x')' = x$

DeMorgan's Theorem

(a) $(a + b)' = a'b'$ (b) $(ab)' = a' + b'$

Generalized DeMorgan's Theorem (a) $(a + b +$

$\ldots z)' = a'b' \ldots z'$ (b) $(a.b \ldots z)' = a' + b' + \ldots z'$

LOGIC GATES

Formal logic: In formal logic, a statement (proposition) is a declarative sentence that is either true(1) or false (0). It is easier to communicate with computers using formal logic.

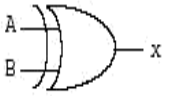
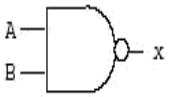
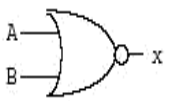
- Boolean variable: Takes only two values – either true (1) or false (0). They are used as basic units of formal logic.

- Boolean algebra: Deals with binary variables and logic operations operating on those variables.

- Logic diagram: Composed of graphic symbols for logic gates. A simple circuit sketch that represents inputs and outputs of Boolean functions.

2.1.1. UNIVERSAL GATES: NAND and NOR are universal gates.

- Other common gates include:

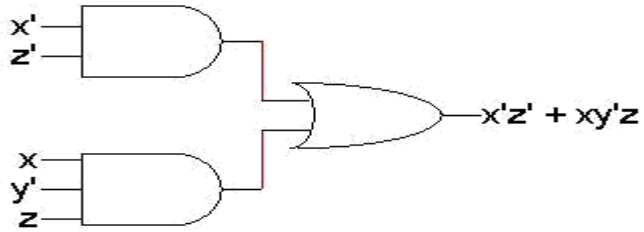
Name	Graphic symbol	Algebraic function	Truth table	
Exclusive-OR (XOR)		x = A ⊕ B = A'B + AB'	A B | x 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0	Parity check: True if only one is true.
NAND		x = (AB)'	A B | x 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0	Inversion of AND.
NOR		x = A + B	A B | x 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0	Inversion of OR.

2.1.2. **MINIMIZATION OF BOOLEAN FUNCTIONS**

Minimization of switching functions is to obtain logic circuits with least circuit complexity. This goal is very difficult since how a minimal function relates to the implementation technology is important. For

example, If we are building a logic circuit that uses discrete logic made of small scale Integration ICs(SSIs) like 7400 series, in which basic building block are constructed and are available for use. The goal of minimization would be to reduce the number of ICs and not the logic gates. For example, If we require two 6 and gates and 5 Or gates,we would require 2 AND ICs(each has 4 AND gates) and one OR IC. (4 gates). On the other hand if the same logic could be implemented with only 10 nand gates, we require only 3 ICs. Similarly when we design logic on Programmable device, we may implement the design with certain number of gates and remaining gates may not be used. Whatever may be the criteria of minimization we would be guided by the following:

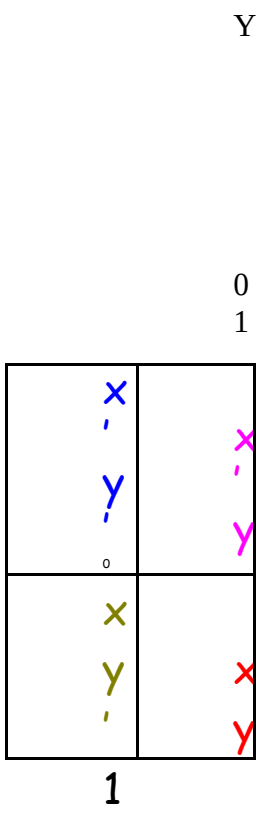
- Boolean algebra helps us simplify expressions and circuits
- Karnaugh Map: A graphical technique for simplifying a Boolean expression into either form:
  - o minimal sum of products (MSP)
  - o minimal product of sums (MPS)
- Goal of the simplification: There are a minimal number of product/sum terms and Each term has a minimal number of literals
- Circuit-wise, this leads to a *minimal* two-level implementation



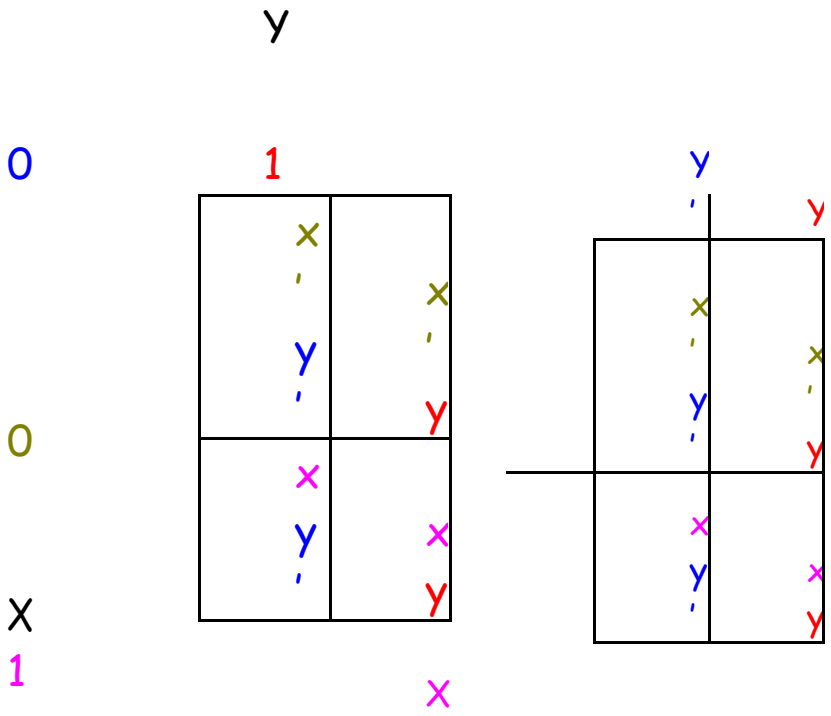
• A two-variable function has four possible minterms. We can re-arrange these minterms into a

Karnaugh map

	minterm
	$x'y'$
	$x'y$
	xy'
	xy



- Now we can easily see which minterms contain common literals
  - Minterms on the left and right sides contain y' and y respectively
  - Minterms in the top and bottom rows contain x' and x respectively

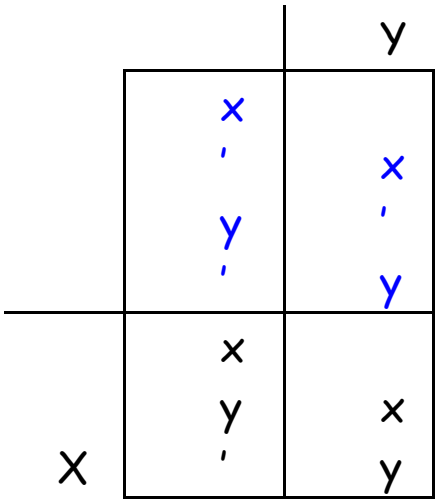


k-map Simplification

- Imagine a two-variable sum of minterms

  x'y' + x'y

- Both of these minterms appear in the top row of a Karnaugh map, which means that they both contain the literal x'



- What happens if you simplify this expression using Boolean algebra?

x'y' + [x'y = x'(y' + y) Distributive]

[

y

+

= y’

x’ =

• 1

1]

[

x

•

1

=

=

x

x’]

A Three-Variable Karnaugh Map

- For a three-variable expression with inputs x, y, z, the arrangement of minterms is more tricky:

X	YZ: 00	01	11	10
0	x'y'z'	x'y'z	x'yz	x'yz'
1	xy'z'	xy'z	xyz	xyz'

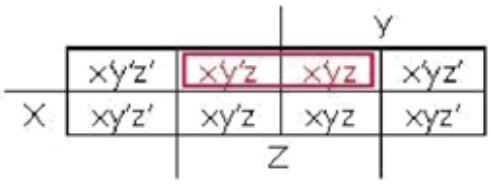
X	YZ: 00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

- Another way to label the K-map (use whichever you like):

		Y			
		x'y'z'	x'y'z	x'yz	x'yz'
X		xy'z'	xy'z	xyz	xyz'
			Z		

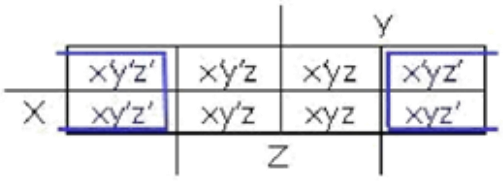
		Y			
		m_0	m_1	m_3	m_2
X		m_4	m_5	m_7	m_6
			Z		

- With this ordering, any group of 2, 4 or 8 adjacent squares on the map contains common literals that can be factored out



$x'y'z + x'yz$
$= x'z(y' + y)$
$= x'z \bullet 1$
$= x'z$

- "Adjacency" includes wrapping around the left and right sides:

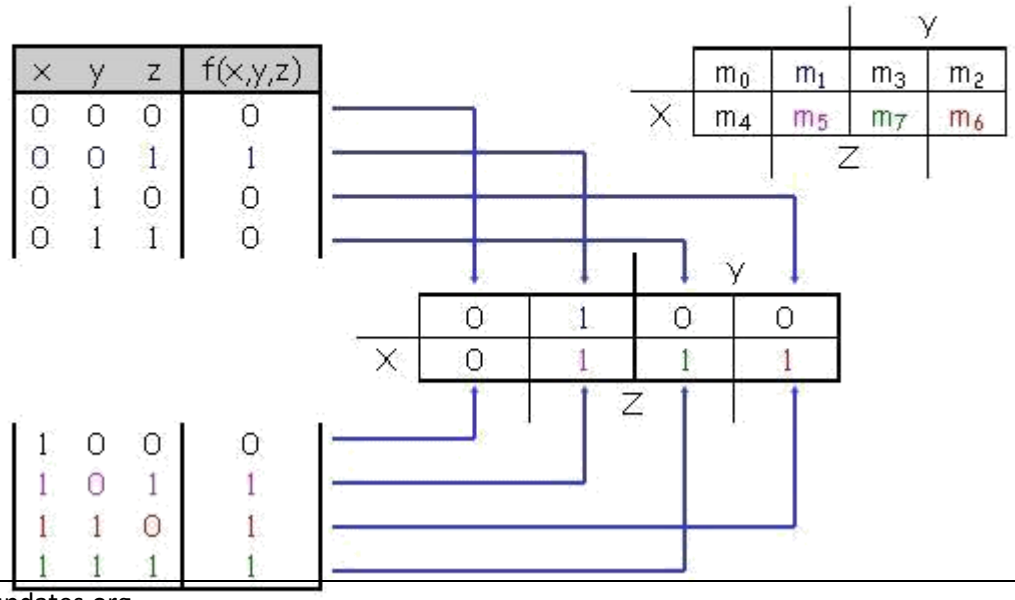


$x'y'z' + xy'z' + x'yz' + xyz'$
$= z'(x'y' + xy' + x'y + xy)$
$= z'(y'(x' + x) + y(x' + x))$
$= z'(y'+y)$
$= z'$

- We'll use this property of adjacent squares to do our simplifications.

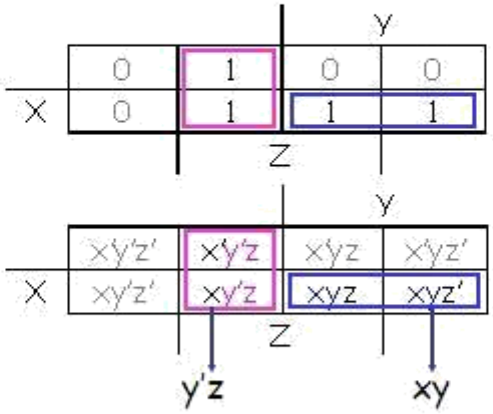
K-maps
From
Truth
Tables

- We can fill in the K-map directly from a truth table
  - The output in row *i* of the table goes into square m_i of the K-map
  - Remember that the rightmost columns of the K-map are "switched"



Reading the MSP from the K-map

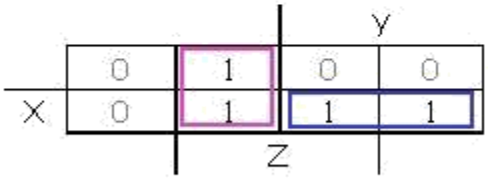
- You can find the minimal SoP expression
  - Each rectangle corresponds to one product term
  - The product is determined by finding the common literals in that rectangle



F(x,y,z)= y'z + xy

Grouping the Minterms Together

- The most difficult step is grouping together all the 1s in the K-map
  - Make rectangles around groups of one, two, four or eight 1s
  - All of the 1s in the map should be included in at least one rectangle
  - Do *not* include any of the 0s
  - Each group corresponds to one product term



K-map Simplification of SoP Expressions

- Let's consider simplifying $f(x,y,z) = xy + y'z + xz$
- You should convert the expression into a sum of minterms form,
  - The easiest way to do this is to make a truth table for the function, and then read off the minterms
  - You can either write out the literals or use the minterm shorthand
- Here is the truth table and sum of minterms for our example:

x	y	z	f(x,y,z)
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

$$f(x,y,z) = x'y'z + xy'z + xyz' + xyz$$
$$= m_1 + m_5 + m_6 + m_7$$

Unsimplifying Expressions

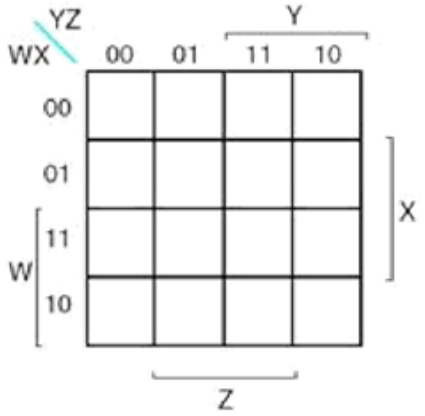
- You can also convert the expression to a sum of minterms with Boolean algebra
  - Apply the distributive law in reverse to add in missing variables.
  - Very few people actually do this, but it's occasionally useful.

$$\begin{aligned} xy + y'z + xz &= (xy \bullet 1) + (y'z \bullet 1) + (xz \bullet 1) \\ &= (xy \bullet (z' + z)) + (y'z \bullet (x' + x)) + (xz \bullet (y' + y)) \\ &= (xyz' + xyz) + (x'y'z + xy'z) + (xy'z + xyz) \\ &= xyz' + xyz + x'y'z + xy'z \\ &= m_1 + m_5 + m_6 + m_7 \end{aligned}$$

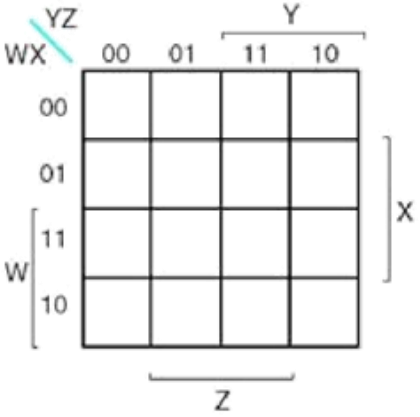
- In both cases, we're actually "unsimplifying" our example expression
  - The resulting expression is larger than the original one!
  - But having all the individual minterms makes it easy to combine them together with the K-map

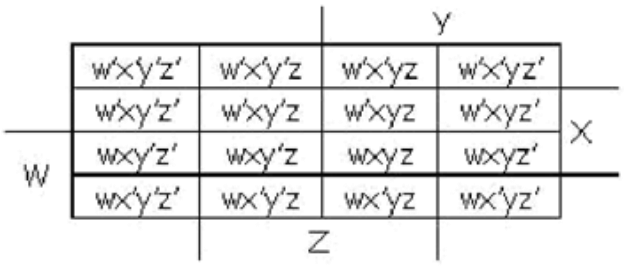
Four-variable K-maps – f(W,X,Y,Z)

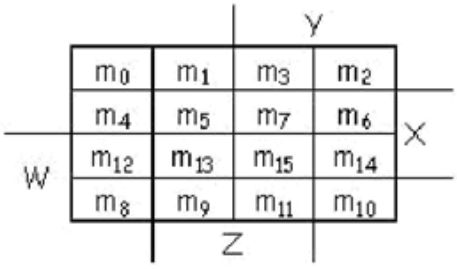
- We can do four-variable expressions too!
  - The minterms in the third and fourth columns, *and* in the third and fourth rows, are switched around.
  - Again, this ensures that adjacent squares have common literals



- Grouping minterms is similar to the three-variable case, but:
  - You can have rectangular groups of 1, 2, 4, 8 or 16 minterms
  - You can wrap around *all four* sides

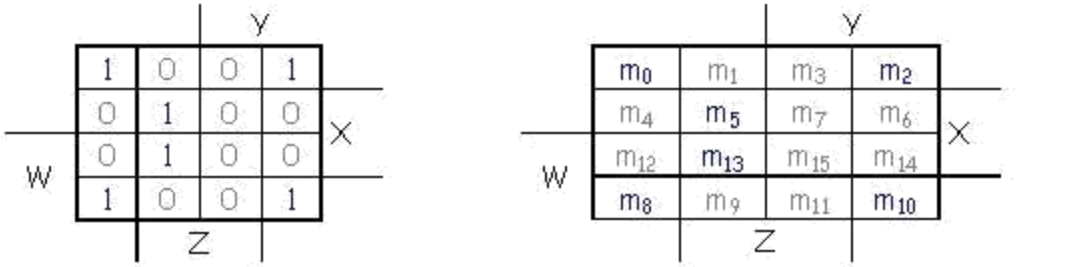




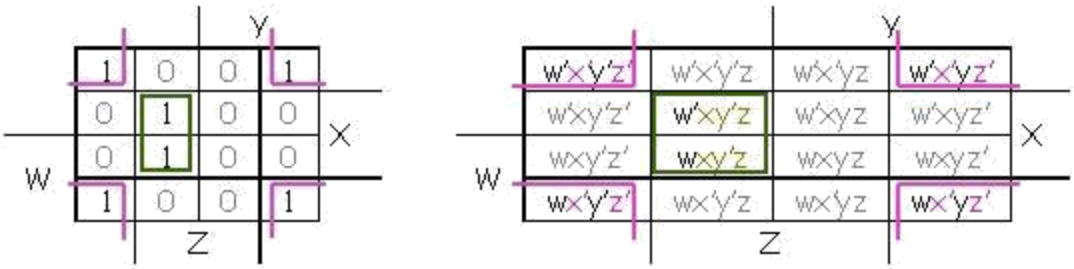


Simplify
$m_0+m_2+m_5+m_8+$
$m_{10}+m_{13}$

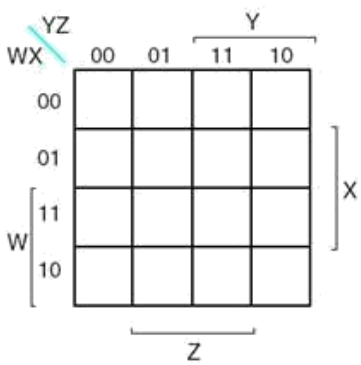
- The expression is already a sum of minterms, so here's the K-map:



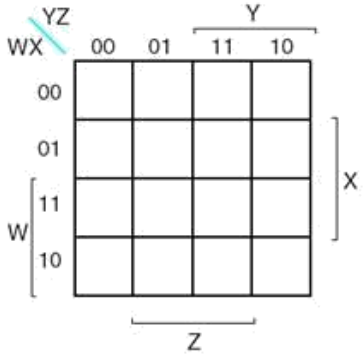
- We can make the following groups, resulting in the MSP x'z' + xy'z



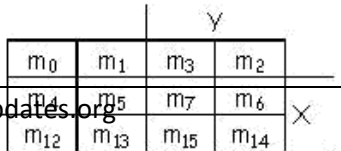
Five-variable K-maps –
f(V,W,X,Y,Z)

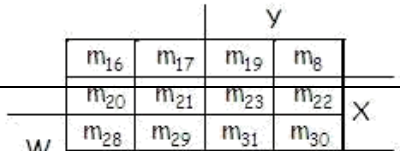


V= 0

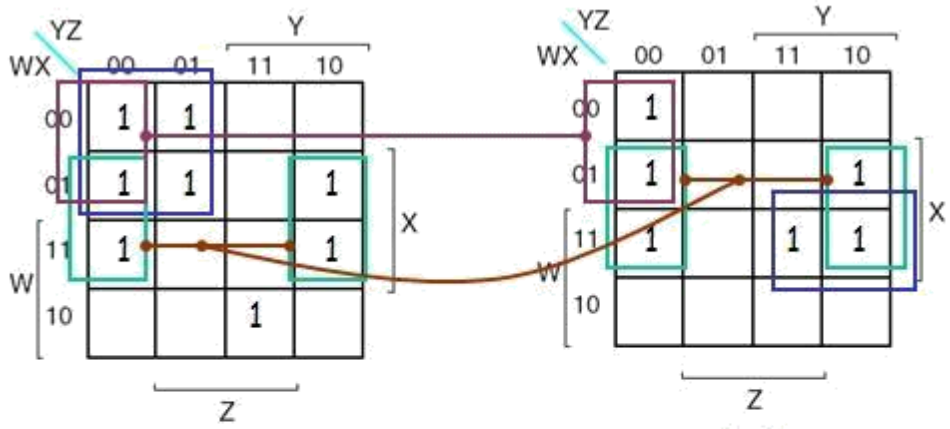


V= 1





Simplify
f(V,W,X,Y,Z)=Σm(0,1,4,5,6,11,12,14,16,20,22, 28,30,31)



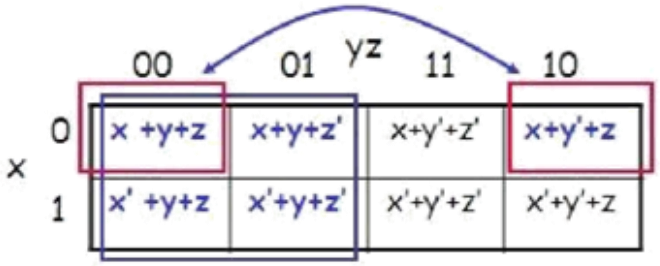
f = XZ'	Σm(4,6,12,14,20,22,28,30)
+ V'W'Y'	Σm(0,1,4,5)
+ W'Y'Z'	Σm(0,4,16,20)
+ VWXY	Σm(30,31)
+ V'WX'YZ	m11

PoS Optimization

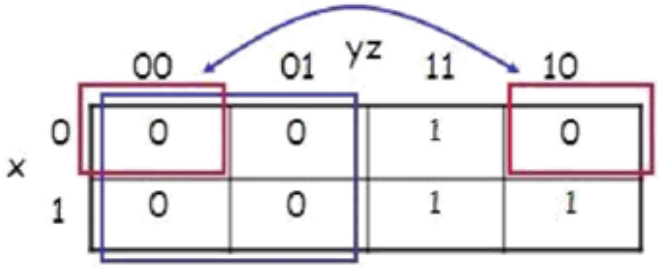
- Maxterms are grouped to find minimal PoS expression

x \ yz	00	01	11	10
0	x +y+z	x+y+z'	x+y'+z'	x+y'+z
1	x' +y+z	x'+y+z'	x'+y'+z'	x'+y'+z

- $F(W,X,Y,Z)= \Pi M(0,1,2,4,5)$



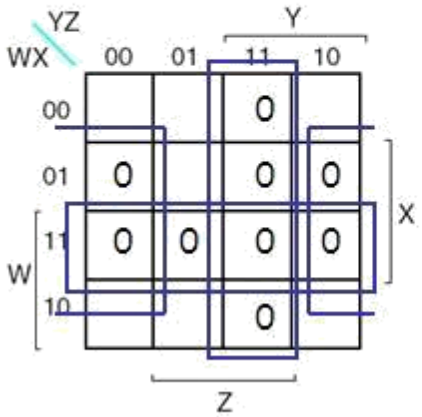
$F(W,X,Y,Z)= Y . (X + Z)$



PoS Optimization from SoP

$F(W,X,Y,Z)= \Sigma m(0,1,2,5,8,9,10)$

$= \Pi M(3,4,6,7,11,12,13,14,15)$



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

Or,

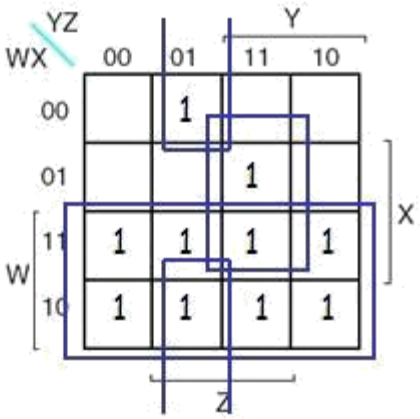
$F(W,X,Y,Z)= X'Y' + X'Z' + W'Y'Z$

Which one is the minimal one?

SoP Optimization from PoS

$F(W,X,Y,Z) = \prod M(0,2,3,4,5,6)$

$= \Sigma m(1,7,8,9,10,11,12,13,14,15)$



$F(W,X,Y,Z) = W + XYZ + X'Y'Z$

Don't care

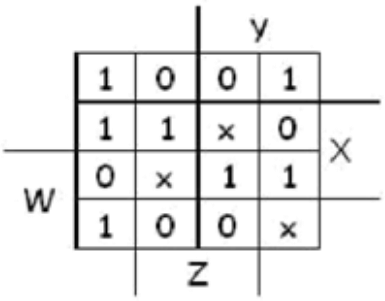
- You don't always need all 2^n input combinations in an n-variable function
  - If you can guarantee that certain input combinations never occur
  - If some outputs aren't used in the rest of the circuit
- We mark don't-care outputs in truth tables and K-maps with Xs.

x	y	z	f(x,y,z)
0	0	0	0
0	0	1	1
0	1	0	X
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	X
1	1	1	1

- Find a MSP for

$$f(w,x,y,z) = \Sigma m(0,2,4,5,8,14,15), d(w,x,y,z) = \Sigma m(7,10,13)$$

This notation means that input combinations wxyz = 0111, 1010 and 1101 (corresponding to minterms m_7, m_{10} and m_{13}) are unused.



K-map Summary

- K-maps are an alternative to Boolean algebra for simplifying expressions
  - The result is a MSP/MPS, which leads to a minimal two-level circuit
  - It's easy to handle don't-care conditions
  - K-maps are really only good for manual simplification of small expressions...
  - Things to keep in mind:
  - Remember the correct order of minterms/maxterms on the K-map
  - When grouping, you can wrap around all sides of the K-map, and your groups can overlap

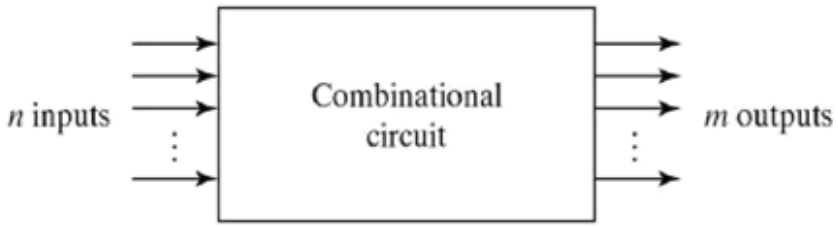
– Make as few rectangles as possible, but make each of them as large as possible. This leads to fewer, but simpler, product terms

  - There may be more than one valid solution

2.1.3. **Combinational Logic**

- Logic circuits for digital systems may be combinational or sequential.

- A combinational circuit consists of input variables, logic gates, and output variables.



Analysis procedure

To obtain the output Boolean functions from a logic diagram, proceed as follows:

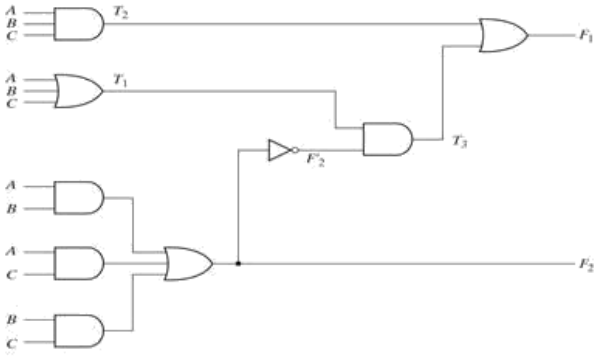
1. Label all gate outputs that are a function of input variables with arbitrary symbols. Determine the Boolean functions for each gate output.
2. Label the gates that are a function of input variables and previously labeled gates with other arbitrary symbols. Find the Boolean functions for these gates.
3. Repeat the process outlined in step 2 until the outputs of the circuit are obtained.
4. By repeated substitution of previously defined functions, obtain the output Boolean functions in terms of input variables.

Example

$F_2 = AB + AC + BC$; $T_1 = A + B + C$; $T_2 = ABC$; $T_3 = F_2'T_1$;

$F_1 = T_3 + T_2$

$F_1 = T_3 + T_2 = F_2'T_1 + ABC = A'BC' + A'B'C + AB'C' + ABC$



Derive truth table from logic diagram

We can derive the truth table by using the above circuit

A	B	C	F_2	F_2	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

2.1.4. FLIP FLOPS

There are a variety of flip-flops, all of which share two properties:

1. The flip-flop is a bistable device either 0 or 1. It exists in one of two states and, in the absence of input, remains in that state. Thus, the flip-flop can function as a 1-bit memory.
2. The flip-flop has two outputs, which are always the complements of each other. These are generally labeled

   Q and $\overline{Q}$.

Table 1 shows symbolic graphic and feature table for three types of flip-flop that are S-R, J-K and D flip-flops. Flip-flop is a form of memory element used to construct sequential circuits that are more complex, such as registers etc. Sequential circuits can be divided into

1. Synchronous
2. Asynchronous

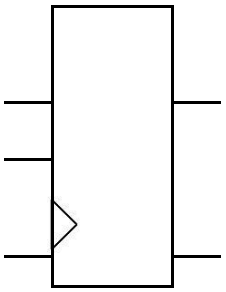
In synchronous sequential circuit, all flip-flops are moved by the same clock pulse so that all flip-flops involved change simultaneously.

In asynchronous circuit, the change of flip-flop condition depends on the change that

occurs on the input and the late time that is in the circuit.

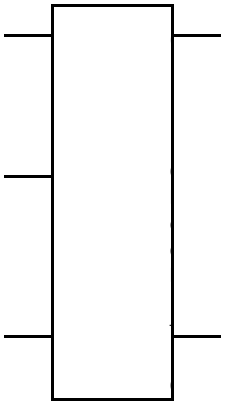
Name	Graphical Symbol	Feature Table

S - R



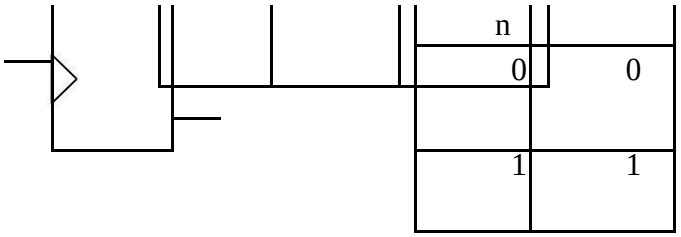
S	R	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	-

J - K

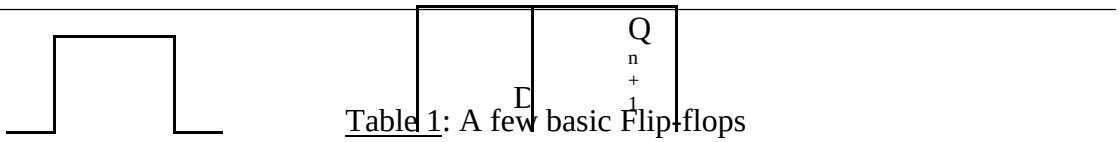


J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	Change condition

D



	n	
	0	0
	1	1



	Q_{n+1}
D	

Table 1: A few basic Flip-flops

S-R Flip-flop

S-R flip-flop has 2 inputs, S (set) and R (reset) like Diagram 3 below. In the diagram below, (also for JK and D flip-flops), there use another input called clock. It is to control the movement of input that is input will only occur when given a clock pulse (synchronous circuit)

The features of S-R flip-flop can be depicted in Table 2 below. It can be summarized that:

1. If the value of both S and R are 0, the flip-flop will remain in its present condition (either 0 or 1).
2. If S = 0 and R = 1 *(*reset*)*, then the flip-flop condition will change to 0 (its output, Q = 0).
3. If S = 1 (set) and R = 0, then the flip-flop condition will change to 1 (output, Q = 1).
4. This circuit does not allow combinational input of input S = 1 and R = 1.

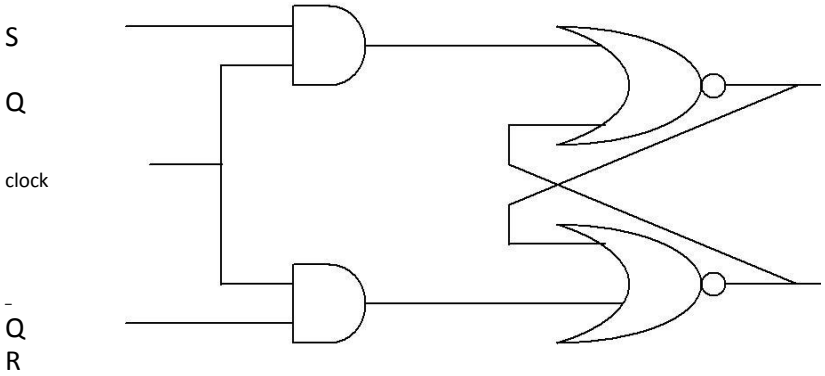


Diagram 3 : S-R Flip-flop

S	R	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	-

1	1	1	-

Table 2 : Feature table of S-R Flip-flop

J-K Flip-flop

J-K flip-flop also has 2 inputs, J and K. The function of clock is same as S-R flip-flop. Unlike S-R flip-flop, J-K flip-flop allows all combination of inputs. The logic circuit for J-K flip-flop is shown in Diagram 2 below. Table 3 shows the features of J-K flip-flop. From the table, it can be summarized that:

1. If J = 0 and K = 0, it will maintain the flip-flop condition like before
2. If J = 0 and K = 1, it will cause flip-flop to change to condition 0 (reset).
3. If J = 1 and K = 0, it will cause flip-flop to change to condition 1 (set).
4. If J = 1 and K = 1, it will change the flip-flop condition, that is it will become complementary to the

   initial or prior condition

It can be observed that J-K flip-flop is built to address the input problem of S = R = 1 in S-R flip-flop. Features 1 till 3 are same as S-R flip-flop.

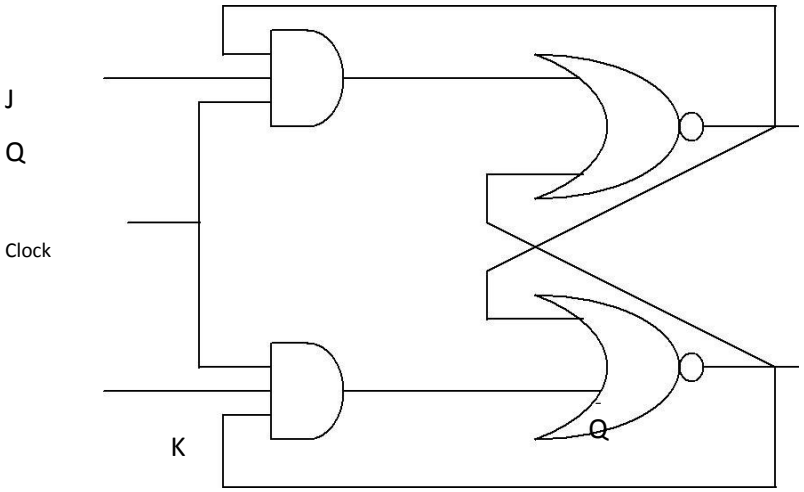


Diagram 2: J-K Flip-flop

J	K	Q_n	Q_{n+1}
0	0	0	0

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

Table 3: Features table of J-K flip-flop

D Flip-flop

Logic circuit for D flip-flop is shown in Diagram 5 below. This flip-flop only has one input that is D. The clock function is same as S-R and J-K flip-flops. The features of D flip-flop can be illustrated by Table 2. From the table, it can be seen that this flip-flop produces the same output as its input regardless of the condition of the stated flip-flop. This feature is very suitable to be used as memory element and this flip-flop is mostly used to make registers and computer memory (RAM)

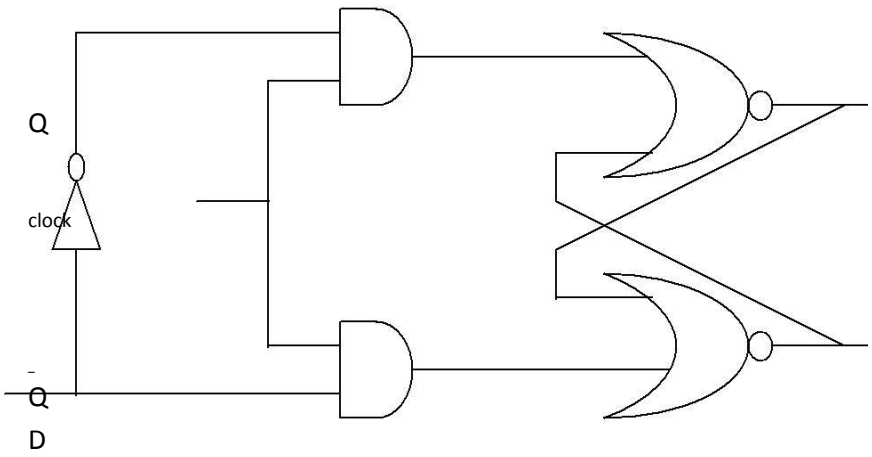


Diagram 5 : D Flip-flop

D	Q_n	Q_{n+1}
0	0	0

0	1	0
1	0	1
1	1	1

Table 2 : Feature table of D Flip-flop

UNIT-2

PART-2

Combinational and Sequential Circuit

Registers,
Shift Registers,
Binary counters
Decoders,
Multiplexers,
Programmable Logic Devices.

Combinational Circuit and Sequential Circuit Criterions

Logic
Circuits can
be divided
into :

1. Combinational Logic Circuit
2. Sequential Logic Circuit

Combinational Logic Circuit

Combinational Logic circuit contains logic gates where its output is determined by the combination of the current input, regardless of the output or the prior combination of input. Basically, combinational circuit can be depicted by Diagram 1 below:

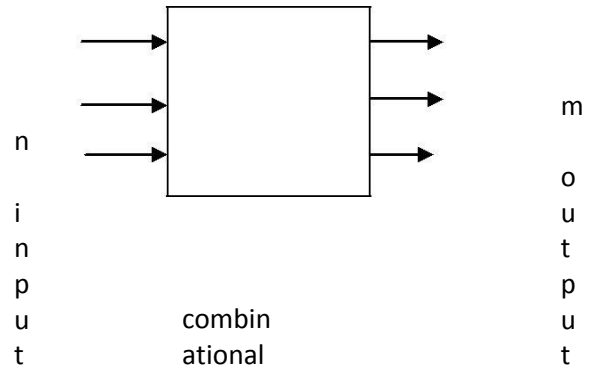


Diagram 1

Examples of Combinational circuits in the computer system are decoder, parallel adder, and multiplexer

(Note: Students are encouraged to obtain examples of combinational circuits stated above)

Sequential Logic Circuit

Sequential Logic Circuit contains logic gates arranged in parallel and its output is not only determined by the combination of the current input, but also the prior output. The circuit also contains memory elements that enable it to store the information of the prior output.

Examples of sequential circuits in the computer system are like registers, counters and serial adders

Some Examples of Combinational Circuit: Parallel Adder, Decoder, etc

The circuits learnt in chapter 3 are combinational circuits. The steps to design combinational circuits are as the following: