

DIGITAL LOGIC CIRCUITS

Digital Computers

Logic Gates

Boolean Algebra

Map Specification

Combinational Circuits

Flip-Flops

Sequential Circuits

Memory Components

Integrated Circuits

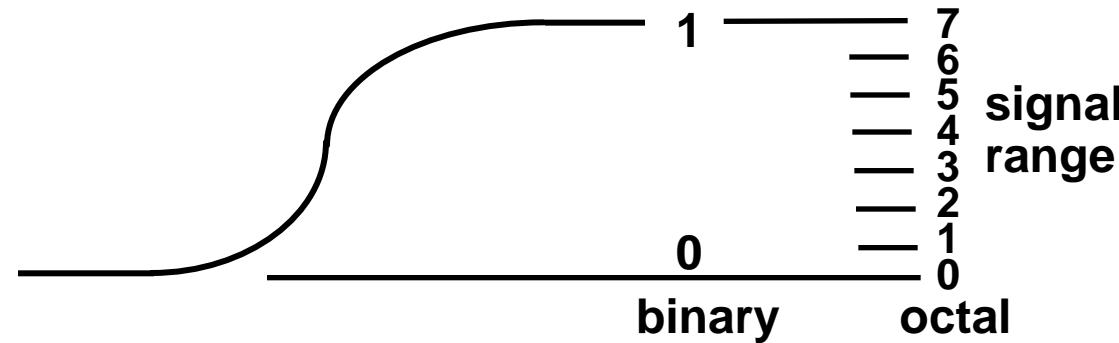
Digital Computer

- **Hardware / Software**
- **Hardware**
 - CPU, RAM, IOP
- **Software**
 - System Software, programs, database
- **Computer Organization**
 - The way H/W operate and the way they are connected togather
- **Computer Design**
 - What h/w should be used and how the parts are to be connected.
- **Computer Architecture**
 - Structure and behavior of computer as seen by the user.

LOGIC GATES

Digital Computers

- Imply that the computer deals with digital information, i.e., it deals with the information that is represented by binary digits
- Why **BINARY**? instead of Decimal or other number system?
- * Consider electronic signal

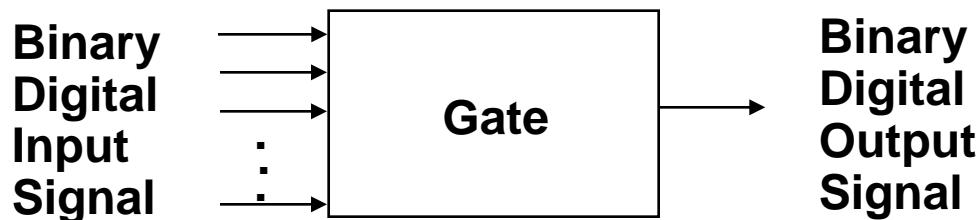


- * Consider the calculation cost - Add

	0	1
0	0	1
1	1	10

	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	10
2	2	3	4	5	6	7	8	9	10	11
3	3	4	5	6	7	8	9	10	11	12
4	4	5	6	7	8	9	10	11	12	13
5	5	6	7	8	9	10	11	12	13	14
6	6	7	8	9	10	11	12	13	14	15
7	7	8	9	10	11	12	13	14	15	16
8	8	9	10	11	12	13	14	15	16	17
9	9	10	11	12	13	14	15	16	17	18

BASIC LOGIC BLOCK - GATE -



Types of Basic Logic Blocks

- **Combinational Logic Block**

Logic Blocks whose output logic value depends only on the input logic values

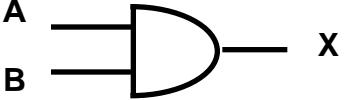
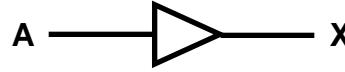
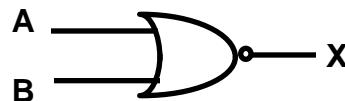
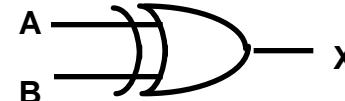
- **Sequential Logic Block**

Logic Blocks whose output logic value depends on the input values and the state (stored information) of the blocks

Functions of Gates can be described by

- Truth Table
- Boolean Function
- Karnaugh Map

COMBINATIONAL GATES

Name	Symbol	Function	Truth Table															
AND		$X = A \cdot B$ or $X = AB$	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>A</td><td>B</td><td>X</td></tr> <tr> <td>0</td><td>0</td><td>0</td></tr> <tr> <td>0</td><td>1</td><td>0</td></tr> <tr> <td>1</td><td>0</td><td>0</td></tr> <tr> <td>1</td><td>1</td><td>1</td></tr> </table>	A	B	X	0	0	0	0	1	0	1	0	0	1	1	1
A	B	X																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$X = A + B$	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>A</td><td>B</td><td>X</td></tr> <tr> <td>0</td><td>0</td><td>0</td></tr> <tr> <td>0</td><td>1</td><td>1</td></tr> <tr> <td>1</td><td>0</td><td>1</td></tr> <tr> <td>1</td><td>1</td><td>1</td></tr> </table>	A	B	X	0	0	0	0	1	1	1	0	1	1	1	1
A	B	X																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
I		$X = A'$	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>A</td><td>X</td></tr> <tr> <td>0</td><td>1</td></tr> <tr> <td>1</td><td>0</td></tr> </table>	A	X	0	1	1	0									
A	X																	
0	1																	
1	0																	
Buffer		$X = A$	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>A</td><td>X</td></tr> <tr> <td>0</td><td>0</td></tr> <tr> <td>1</td><td>1</td></tr> </table>	A	X	0	0	1	1									
A	X																	
0	0																	
1	1																	
NAND		$X = (AB)'$	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>A</td><td>B</td><td>X</td></tr> <tr> <td>0</td><td>0</td><td>1</td></tr> <tr> <td>0</td><td>1</td><td>1</td></tr> <tr> <td>1</td><td>0</td><td>1</td></tr> <tr> <td>1</td><td>1</td><td>0</td></tr> </table>	A	B	X	0	0	1	0	1	1	1	0	1	1	1	0
A	B	X																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$X = (A + B)'$	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>A</td><td>B</td><td>X</td></tr> <tr> <td>0</td><td>0</td><td>1</td></tr> <tr> <td>0</td><td>1</td><td>0</td></tr> <tr> <td>1</td><td>0</td><td>0</td></tr> <tr> <td>1</td><td>1</td><td>0</td></tr> </table>	A	B	X	0	0	1	0	1	0	1	0	0	1	1	0
A	B	X																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
XOR Exclusive OR		$X = A \oplus B$ or $X = A'B + AB'$	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>A</td><td>B</td><td>X</td></tr> <tr> <td>0</td><td>0</td><td>0</td></tr> <tr> <td>0</td><td>1</td><td>1</td></tr> <tr> <td>1</td><td>0</td><td>1</td></tr> <tr> <td>1</td><td>1</td><td>0</td></tr> </table>	A	B	X	0	0	0	0	1	1	1	0	1	1	1	0
A	B	X																
0	0	0																
0	1	1																
1	0	1																
1	1	0																
XNOR Exclusive NOR or Equivalence		$X = (A \oplus B)'$ or $X = A'B' + AB$	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>A</td><td>B</td><td>X</td></tr> <tr> <td>0</td><td>0</td><td>1</td></tr> <tr> <td>0</td><td>1</td><td>0</td></tr> <tr> <td>1</td><td>0</td><td>0</td></tr> <tr> <td>1</td><td>1</td><td>1</td></tr> </table>	A	B	X	0	0	1	0	1	0	1	0	0	1	1	1
A	B	X																
0	0	1																
0	1	0																
1	0	0																
1	1	1																

BOOLEAN ALGEBRA

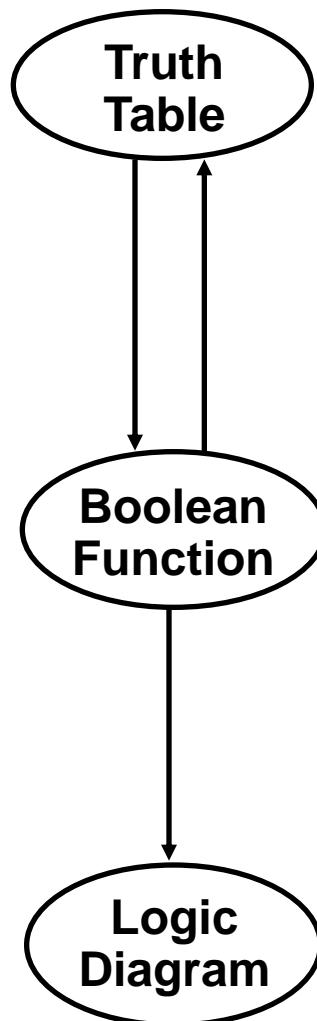
Boolean Algebra

- * Algebra with Binary(Boolean) Variable and Logic Operations
- * Boolean Algebra is useful in Analysis and Synthesis of Digital Logic Circuits
 - Input and Output signals can be represented by Boolean Variables, and
 - Function of the Digital Logic Circuits can be represented by Logic Operations, i.e., Boolean Function(s)
 - From a Boolean function, a logic diagram can be constructed using AND, OR, and I

Truth Table

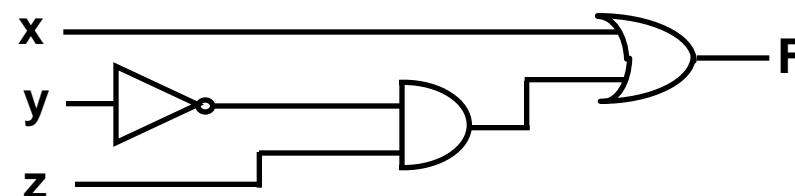
- * The most elementary specification of the function of a Digital Logic Circuit is the Truth Table
 - Table that describes the Output Values for all the combinations of the Input Values, called *MINTERMS*
 - n input variables → 2^n minterms

LOGIC CIRCUIT DESIGN



x	y	z	F
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	1

$$F = x + y'z$$



BASIC IDENTITIES OF BOOLEAN ALGEBRA

$$[1] \quad x + 0 = x$$

$$[3] \quad x + 1 = 1$$

$$[5] \quad x + x = x$$

$$[7] \quad x + x' = 1$$

$$[9] \quad x + y = y + x$$

$$[11] \quad x + (y + z) = (x + y) + z$$

$$[13] \quad x(y + z) = xy + xz$$

$$[15] \quad (x + y)' = x'y'$$

$$[17] \quad (x')' = x$$

$$[2] \quad x \cdot 0 = 0$$

$$[4] \quad x \cdot 1 = x$$

$$[6] \quad x \cdot x = x$$

$$[8] \quad x \cdot x' = 0$$

$$[10] \quad xy = yx$$

$$[12] \quad x(yz) = (xy)z$$

$$[14] \quad x + yz = (x + y)(x + z)$$

$$[16] \quad (xy)' = x' + y'$$

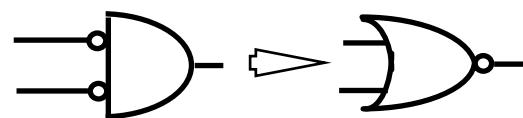
[15] and [16] : De Morgan's Theorem

Usefulness of this Table

- Simplification of the Boolean function
- Derivation of equivalent Boolean functions
to obtain logic diagrams utilizing different logic gates
 - Ordinarily ANDs, ORs, and Inverters
 - But a certain different form of Boolean function may be convenient
to obtain circuits with NANDs or NORs
 - Applications of De Morgans Theorem

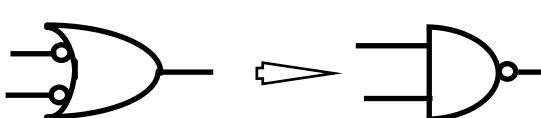
$$x'y' = (x + y)'$$

I, AND → NOR



$$x' + y' = (xy)'$$

I, OR → NAND



De Morgan Theorem Proof

- Informal

- Negation of Disjunction

- » It is False that either A or B is true

- $(A+B)' \rightarrow$ neither A nor B is true \rightarrow A is not true AND B is not true,

lemma 1: if $A \cdot B' = 0$ and $A + B' = 1$, then $A=B$

proof by contrapositive: suppose $A \neq B$, then $A = B'$ therefore

$A \cdot B' = A \cdot A = 0$ and by id 6,

$A = 0$, but likewise

$A + B' = A+A = 1$ and by id 6

$A+A = A = 1$, thus we have $A = 1$ and $A = 0$, a contradiction.

Thus the lemma is proved.

$$\begin{array}{ll} [1] & x + 0 = x \\ [3] & x + 1 = 1 \\ [5] & x + x = x \\ [7] & x + x' = 1 \\ [9] & x + y = y + x \\ [11] & x + (y + z) = (x + y) + z \\ [13] & x(y + z) = xy + xz \\ [15] & (x + y)' = x'y' \\ [17] & (x')' = x \end{array}$$

$$\begin{array}{ll} [2] & x \cdot 0 = 0 \\ [4] & x \cdot 1 = x \\ [6] & x \cdot x = x \\ [8] & x \cdot x' = 0 \\ [10] & xy = yx \\ [12] & x(yz) = (xy)z \\ [14] & x + yz = (x + y)(x + z) \\ [16] & (xy)' = x' + y' \end{array}$$

De Morgan Theorem Proof Contd.

- Now we use this to prove $(a + b)' = a' \cdot b'$

we use the lemma, if $A \cdot B' = 0$ and $A + B' = 1$, then $A=B$

let $A = a' \cdot b'$ and let $B = (a + b)'$

first we show $A \cdot B' = 0$

$A \cdot B' = (a' \cdot b') \cdot (a + b)$, using the distributive and commutative postulates,

$$= a' \cdot a \cdot b' + a' \cdot b' \cdot b \text{ now using id13}$$

$$= 0 \cdot b' + a' \cdot 0, \text{ now by id 8}$$

$$= 0 \cdot 0 = 0$$

now we show $A + B' = 1$

$$A + B' = a' \cdot b' + a + b$$

suppose $a = b$, then

$$a' \cdot b' + a + b = a' \cdot a' + a + a.$$

$$= a' + a.$$

$$= 1$$

De Morgan Theorem Proof Contd.

- suppose now that $a \neq b$, then (and so $a = b'$ and $b = a'$) then

$$\begin{aligned} a' \cdot b' + a + b &= a' \cdot a + a + a' \\ &= 0 + 1 = 1 \end{aligned}$$

therefore the lemma applies so $A = B$, or more specifically, $(a + b)' = a' \cdot b'$

EQUIVALENT CIRCUITS

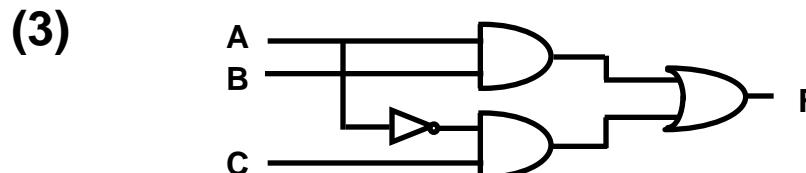
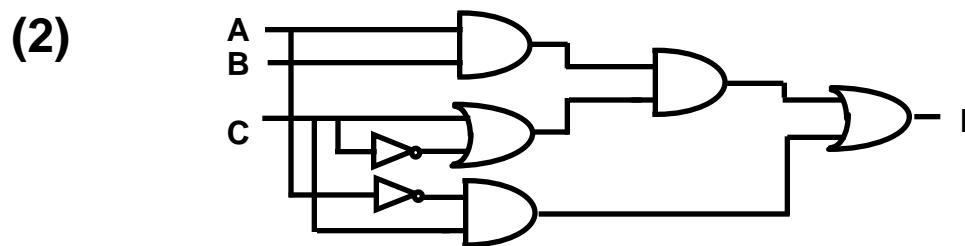
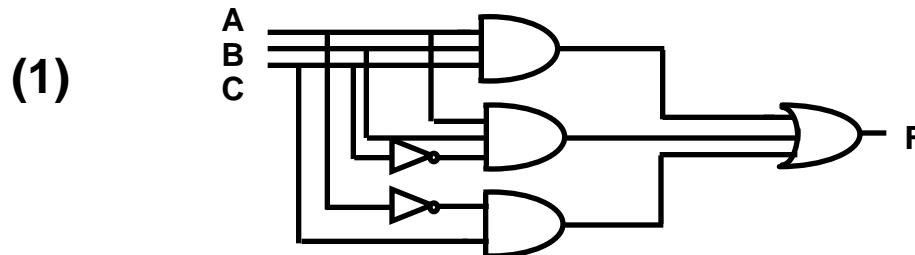
Many different logic diagrams are possible for a given Function

$$F = ABC + ABC' + A'C \quad \dots \quad (1)$$

$$= AB(C + C') + A'C \quad [13] \dots (2)$$

$$= AB \bullet 1 + A'C \quad [7]$$

$$= AB + A'C \quad [4] \dots (3)$$



COMPLEMENT OF FUNCTIONS

A Boolean function of a digital logic circuit is represented by only using logical variables and AND, OR, and Invert operators.

→ Complement of a Boolean function

- Replace all the variables and subexpressions in the parentheses appearing in the function expression with their respective complements

$$\begin{aligned} A, B, \dots, Z, a, b, \dots, z &\Rightarrow A', B', \dots, Z', a', b', \dots, z' \\ (p + q) &\Rightarrow (p + q)' \end{aligned}$$

- Replace all the operators with their respective complementary operators

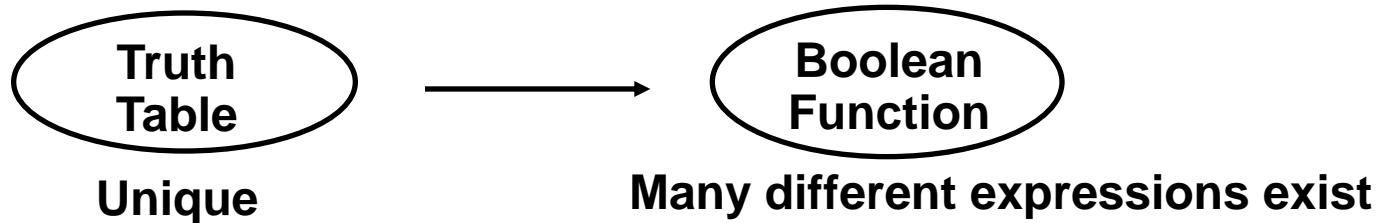
$$\begin{aligned} \text{AND} &\Rightarrow \text{OR} \\ \text{OR} &\Rightarrow \text{AND} \end{aligned}$$

- Basically, extensive applications of the De Morgan's theorem

$$(x_1 + x_2 + \dots + x_n)' \Rightarrow x_1' x_2' \dots x_n'$$

$$(x_1 x_2 \dots x_n)' \Rightarrow x_1' + x_2' + \dots + x_n'$$

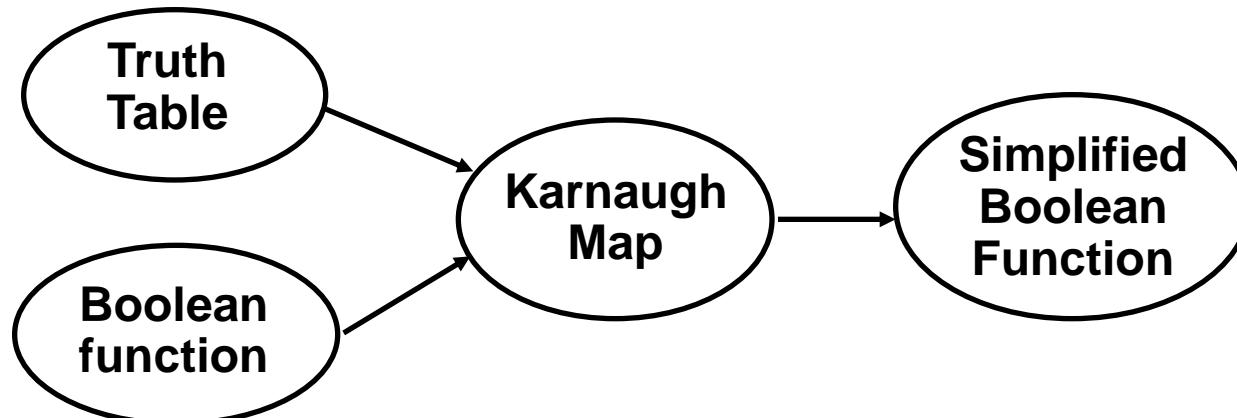
SIMPLIFICATION



Simplification from Boolean function

- Finding an equivalent expression that is least expensive to implement
- For a simple function, it is possible to obtain a simple expression for low cost implementation
- But, with complex functions, it is a very difficult task

Karnaugh Map (K-map) is a simple procedure for simplifying Boolean expressions.

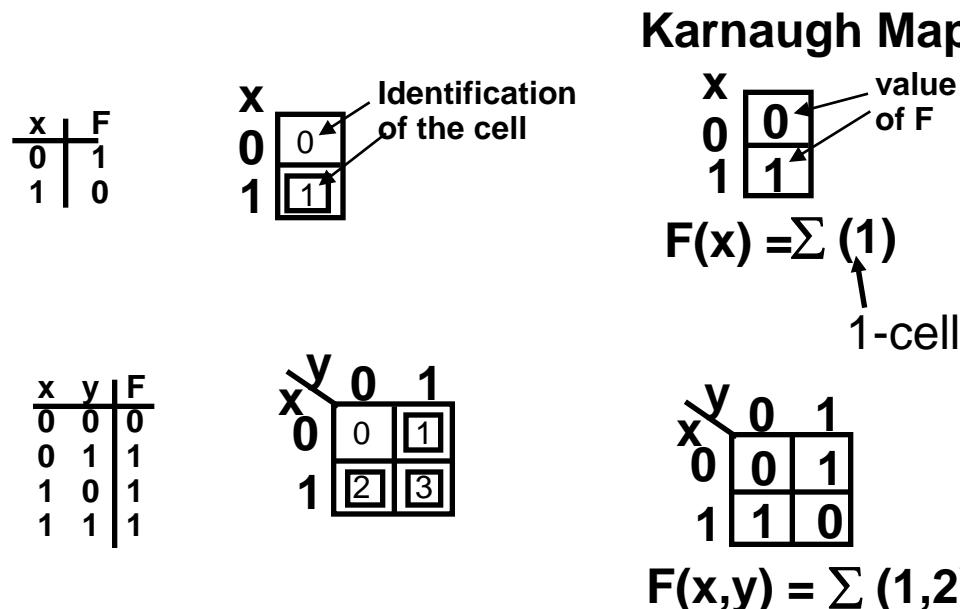


KARNAUGH MAP

Karnaugh Map for an n-input digital logic circuit (n-variable sum-of-products form of Boolean Function, or Truth Table) is

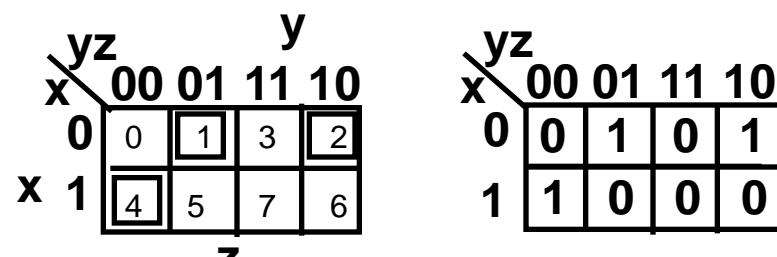
- Rectangle divided into 2^n cells
- Each cell is associated with a *Minterm*
- An output(function) value for each input value associated with a minterm is written in the cell representing the minterm
→ 1-cell, 0-cell

Each Minterm is identified by a decimal number whose binary representation is identical to the binary interpretation of the input values of the minterm.



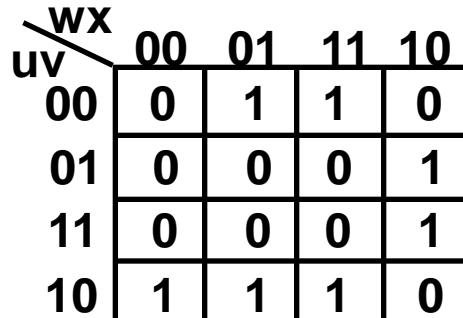
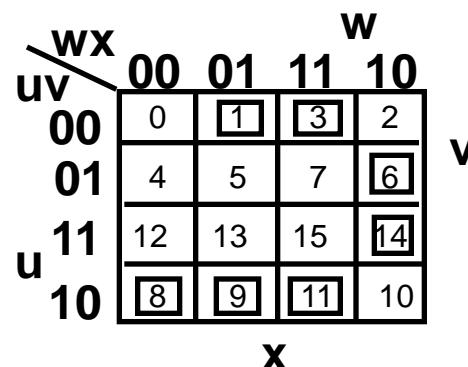
KARNAUGH MAP

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0



$$F(x,y,z) = \sum (1,2,4)$$

u	v	w	x	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0



$$F(u,v,w,x) = \sum (1,3,6,8,9,11,14)$$

MAP SIMPLIFICATION - 2 ADJACENT CELLS -

Rule: $xy' + xy = x(y+y') = x$

Adjacent cells

- binary identifications are different in one bit
 - minterms associated with the adjacent cells have one variable complemented each other

Cells (1,0) and (1,1) are adjacent
 Minterms for (1,0) and (1,1) are

$$\begin{aligned}x \cdot y' &\rightarrow x=1, y=0 \\x \cdot y &\rightarrow x=1, y=1\end{aligned}$$

$F = xy' + xy$ can be reduced to $F = x$

From the map

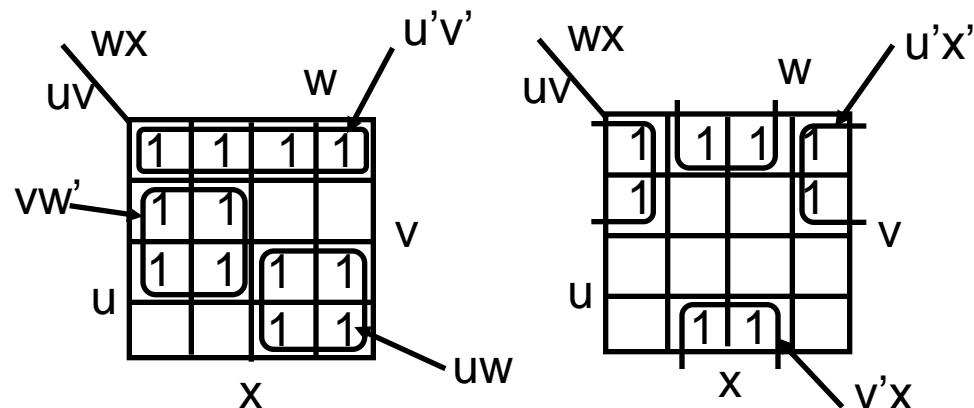
	x	0	1
0		0	0
1		1	1

2 adjacent cells xy' and xy
 → merge them to a larger cell x

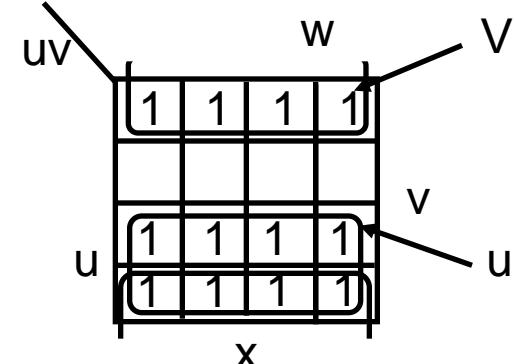
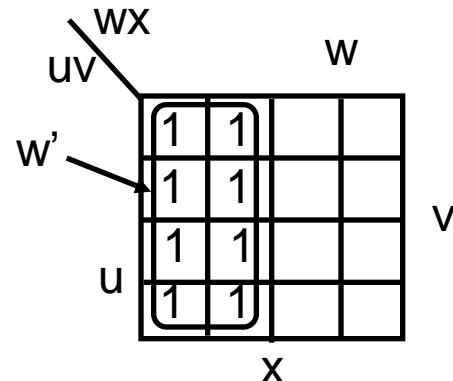
$$\begin{aligned}F(x,y) &= \sum (2,3) \\&= xy' + xy \\&= x\end{aligned}$$

MAP SIMPLIFICATION - MORE THAN 2 CELLS -

$$\begin{aligned}
 & u'v'w'x' + u'v'w'x + u'v'wx + u'v'wx' \\
 & = u'v'w'(x'+x) + u'v'w(x+x') \\
 & = u'v'w' + u'v'w \\
 & = u'v'(w'+w) \\
 & = u'v'
 \end{aligned}$$



$$\begin{aligned}
 & u'v'w'x' + u'v'w'x + u'vw'x' + u'vw'x + uvw'x' + uvw'x + uv'w'x' + uv'w'x \\
 & = u'v'w'(x'+x) + u'vw'(x'+x) + uvw'(x'+x) + uv'w'(x'+x) \\
 & = u'(v'+v)w' + u(v'+v)w' \\
 & = (u'+u)w' = w'
 \end{aligned}$$



MAP SIMPLIFICATION

	wx	00	01	11	10
uv	00	1	1	0	1
	01	0	0	0	0
	11	0	1	1	0
	10	0	1	0	0



	w	0	1
u	1	1	0
	0	0	0
	0	1	1
	0	1	0

x

$$F(u,v,w,x) = \sum (0,1,2,9,13,15)$$

(0,1), (0,2), (0,4), (0,8)

Adjacent Cells of 1

Adjacent Cells of 0

(1,0), (1,3), (1,5), (1,9)

...

...

Adjacent Cells of 15

(15,7), (15,11), (15,13), (15,14)

Merge (0,1) and (0,2)

$$\rightarrow u'v'w' + u'v'x'$$

Merge (1,9)

$$\rightarrow v'w'x$$

Merge (9,13)

$$\rightarrow uw'x$$

Merge (13,15)

$$\rightarrow uvx$$

$$F = u'v'w' + u'v'x' + v'w'x + uw'x + uvx$$

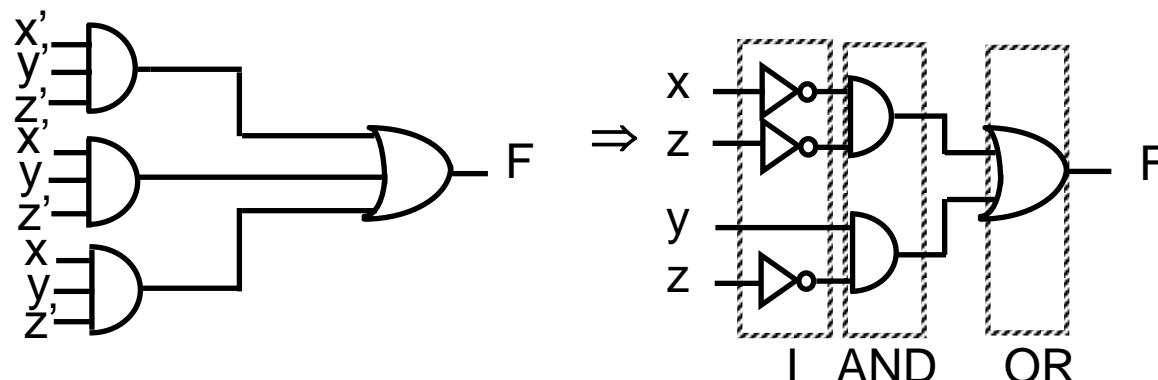
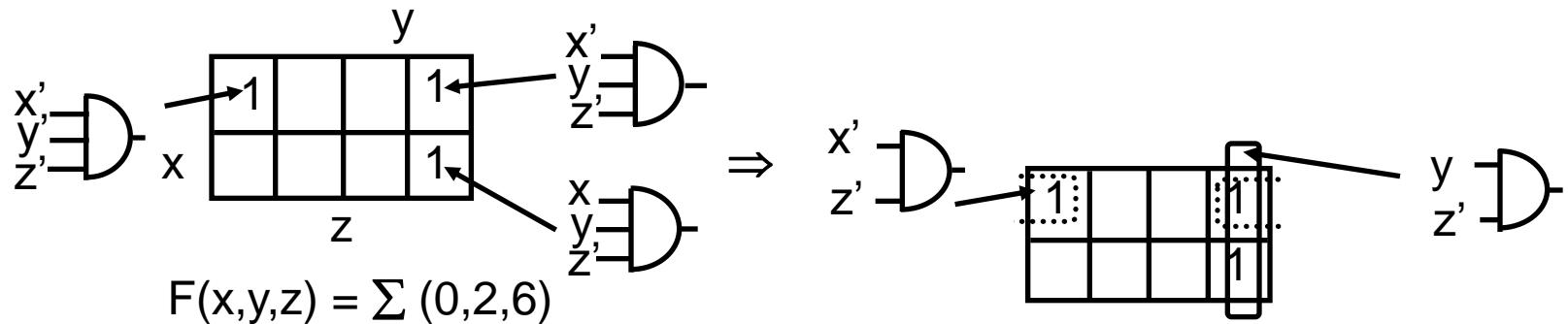
But (9,13) is covered by (1,9) and (13,15)

$$F = u'v'w' + u'v'x' + v'w'x + uvx$$

IMPLEMENTATION OF K-MAPS - Sum-of-Products Form -

Logic function represented by a Karnaugh map can be implemented in the form of I-AND-OR

A cell or a collection of the adjacent 1-cells can be realized by an AND gate, with some inversion of the input variables.

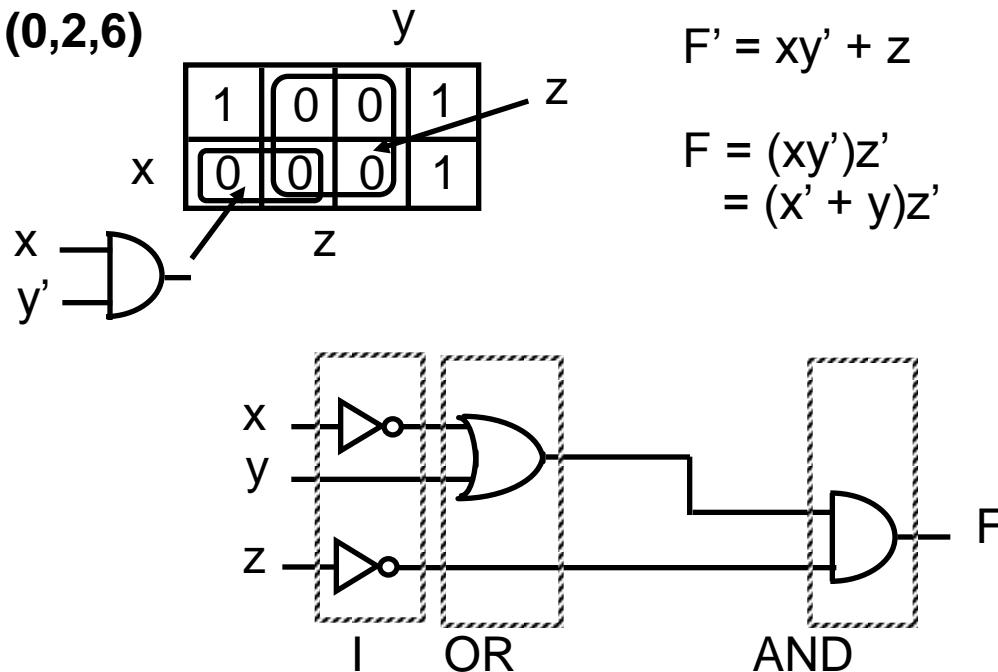


IMPLEMENTATION OF K-MAPS - Product-of-Sums Form -

Logic function represented by a Karnaugh map can be implemented in the form of I-OR-AND

If we implement a Karnaugh map using 0-cells, the complement of F, i.e., F' , can be obtained. Thus, by complementing F' using DeMorgan's theorem F can be obtained

$$F(x,y,z) = (0,2,6)$$



IMPLEMENTATION OF K-MAPS

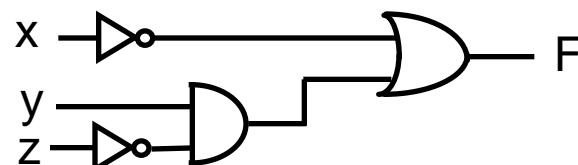
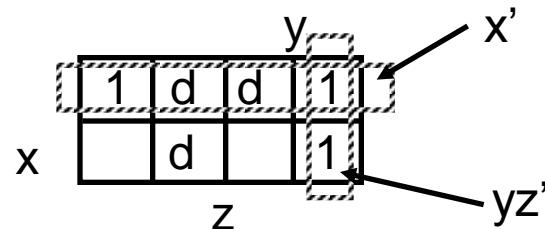
- Don't-Care Conditions -

In some logic circuits, the output responses for some input conditions are don't care whether they are 1 or 0.

In K-maps, don't-care conditions are represented by d's in the corresponding cells.

Don't-care conditions are useful in minimizing the logic functions using K-map.

- Can be considered either 1 or 0
- Thus increases the chances of merging cells into the larger cells
--> Reduce the number of variables in the product terms



COMBINATIONAL LOGIC CIRCUITS

Half Adder

x	y	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

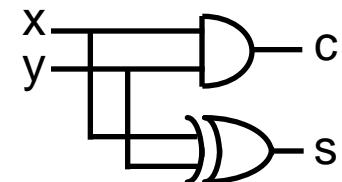
$$x \begin{array}{|c|} \hline y \\ \hline 0 & 0 \\ \hline 0 & 1 \\ \hline \end{array}$$

$$c = xy$$

$$x \begin{array}{|c|} \hline y \\ \hline 0 & 1 \\ \hline 1 & 0 \\ \hline \end{array}$$

$$s = xy' + x'y$$

$$= x \oplus y$$



Full Adder

x	y	C_{n-1}	C_n	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$x \begin{array}{|c|c|c|} \hline y & & \\ \hline 0 & 0 & \\ \hline 0 & 1 & \\ \hline 1 & 1 & \\ \hline 0 & 1 & \\ \hline \end{array} C_{n-1}$$

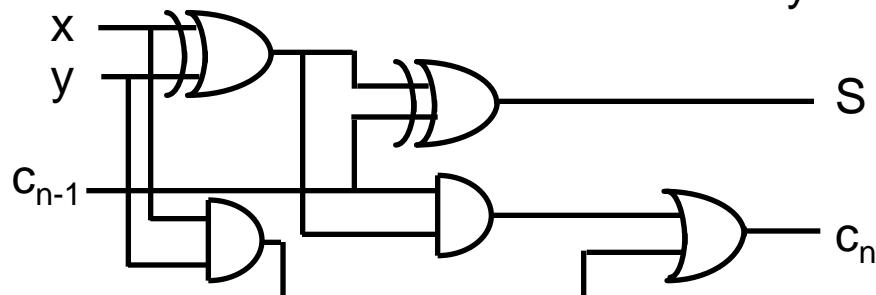
$$x \begin{array}{|c|c|c|} \hline y & & \\ \hline 0 & 1 & \\ \hline 1 & 0 & \\ \hline 0 & 1 & \\ \hline 1 & 0 & \\ \hline \end{array} C_{n-1}$$

$$C_n = xy + xc_{n-1} + yc_{n-1}$$

$$= xy + (x \oplus y)c_{n-1}$$

$$S = x'y'c_{n-1} + x'yc'_{n-1} + xy'c'_{n-1} + xyc_{n-1}$$

$$= x \oplus y \oplus c_{n-1} = (x \oplus y) \oplus c_{n-1}$$



COMBINATIONAL LOGIC CIRCUITS

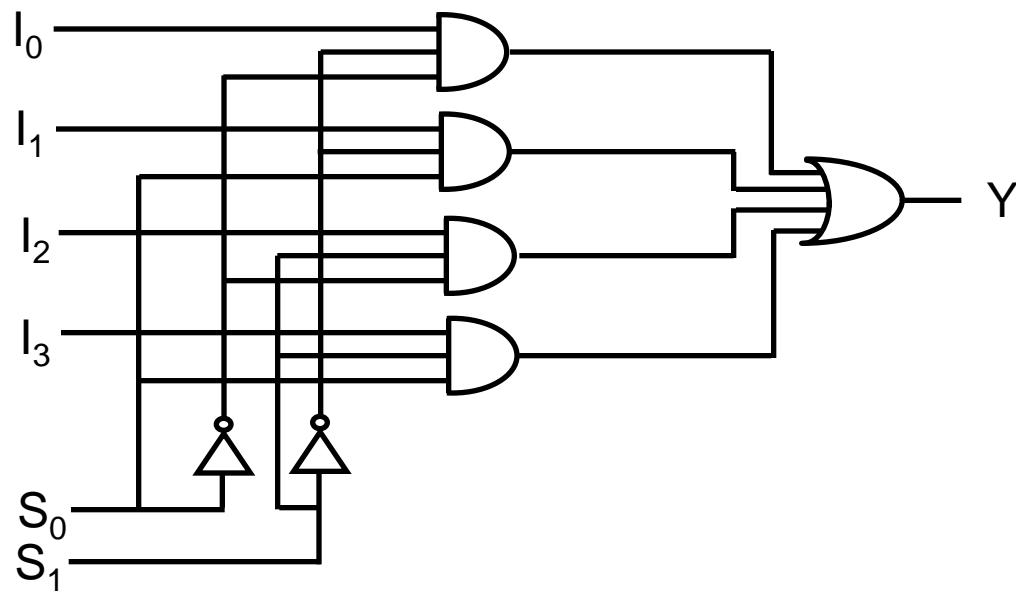
Other Combinational Circuits

**Multiplexer
Encoder
Decoder
Parity Checker
Parity Generator
etc**

MULTIPLEXER

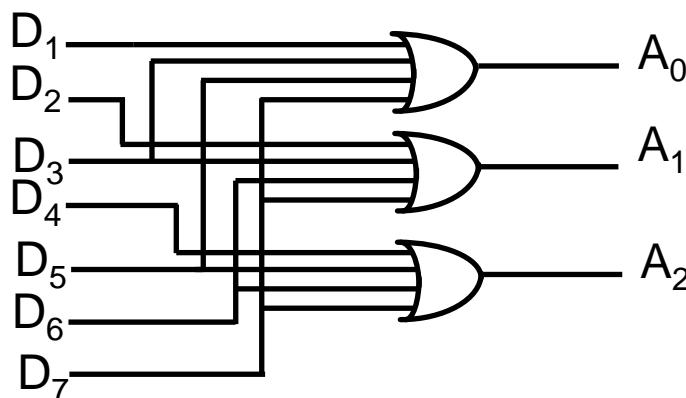
4-to-1 Multiplexer

Select		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3



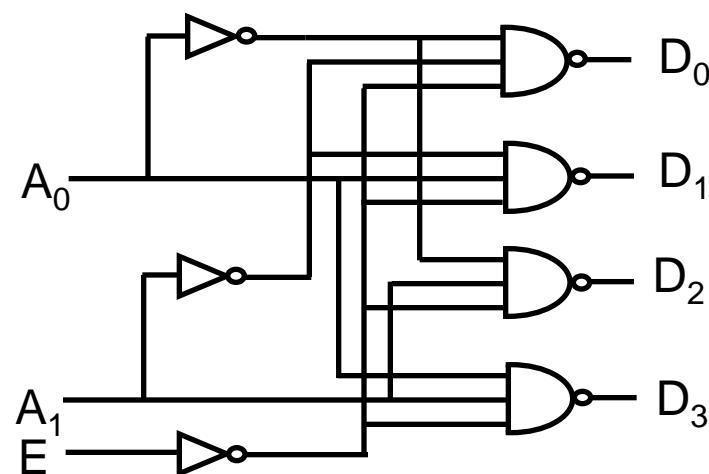
ENCODER/DECODER

Octal-to-Binary Encoder



2-to-4 Decoder

E	A_1	A_0	D_0	D_1	D_2	D_3
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0
1	d	d	1	1	1	1



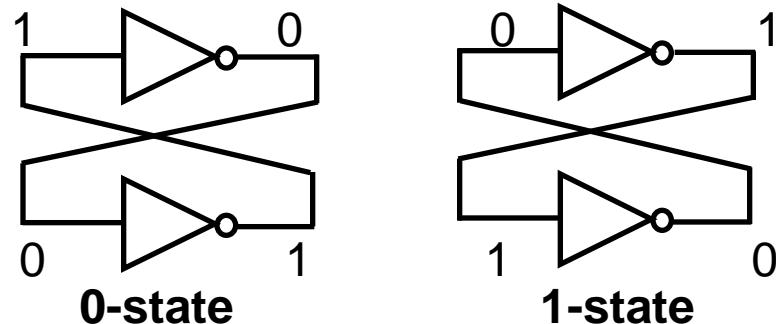
Sequential Circuits

- In Combinational Circuits, outputs at any time depends upon inputs that are presented at that time.
- Storage elements are also required, which requires system to be defined in terms of sequential circuits.
- Synchronization is achieved through Clock pulses.
- Clocked Synchronous sequential circuits

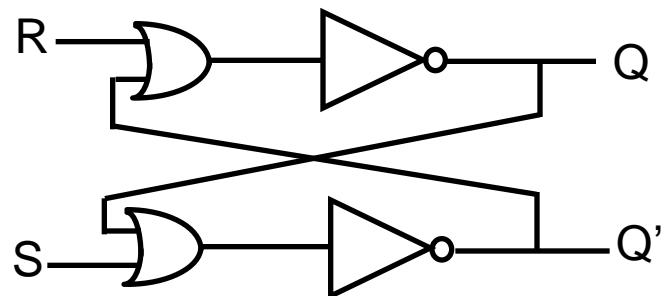
FLIP FLOPS

Characteristics

- 2 stable states
- Memory capability
- Operation is specified by a Characteristic Table



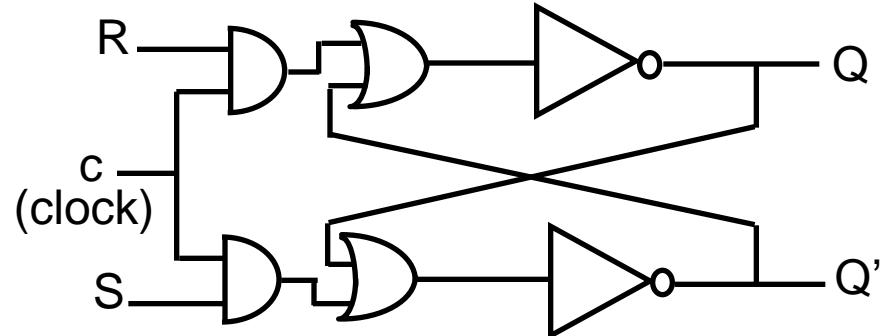
In order to be used in the computer circuits, state of the flip flop should have input terminals and output terminals so that it can be set to a certain state, and its state can be read externally.



S	R	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	indeterminate (forbidden)

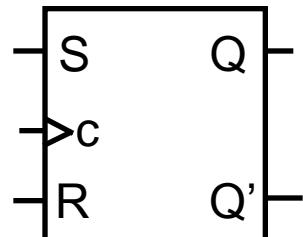
CLOCKED FLIP FLOPS

In a large digital system with many flip flops, operations of individual flip flops are required to be synchronized to a clock pulse. Otherwise, the operations of the system may be unpredictable.

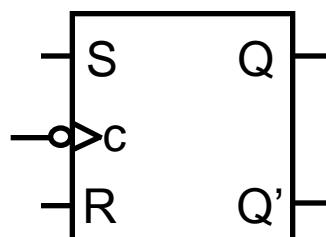


Clock pulse allows the flip flop to change state only when there is a clock pulse appearing at the c terminal.

We call above flip flop a Clocked RS Latch, and symbolically as



operates when
clock is high

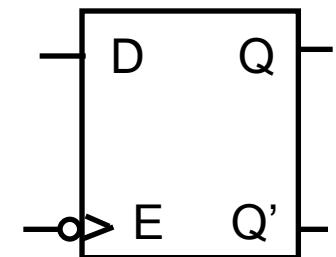
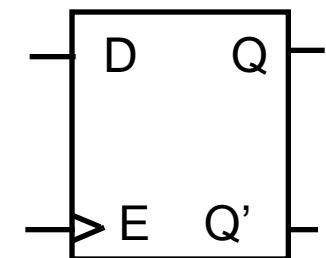
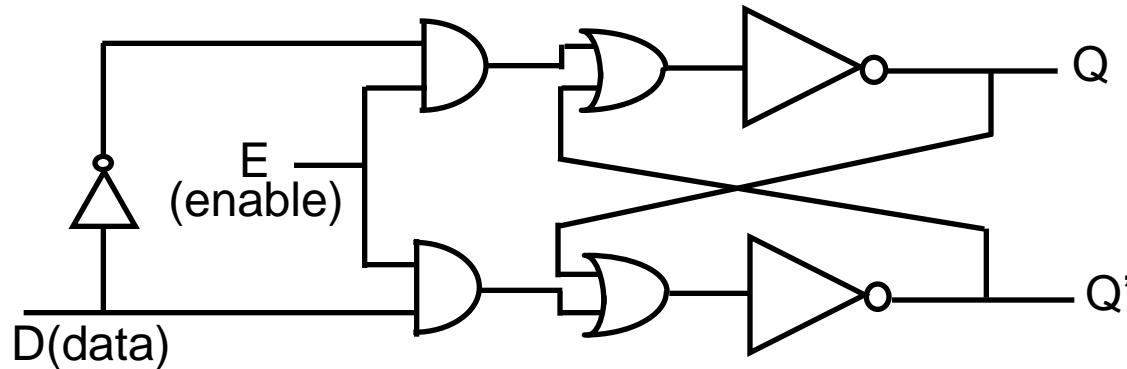


operates when
clock is low

D-LATCH

D-Latch

Forbidden input values are forced not to occur by using an inverter between the inputs



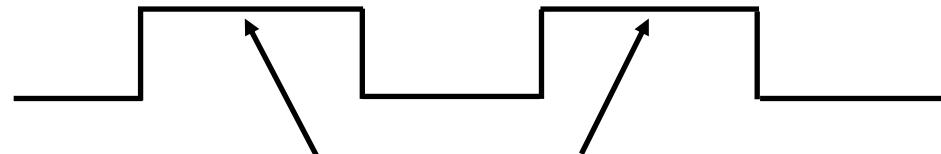
D	Q(t+1)
0	0
1	1

EDGE-TRIGGERED FLIP FLOPS

Characteristics

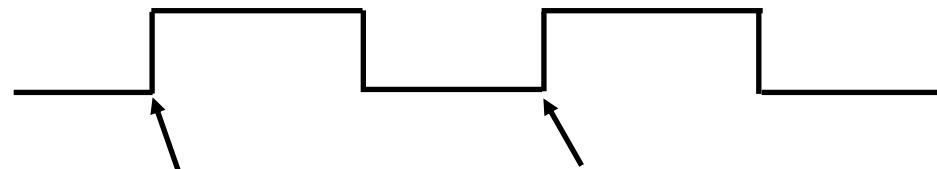
- State transition occurs at the rising edge or falling edge of the clock pulse

Latches



respond to the input only during these periods

Edge-triggered Flip Flops (positive)



respond to the input only at this time

Excitation Tables

an **excitation table** shows the minimum inputs that are necessary to generate a particular next state when the current state is known.

SR flip flop			
Q(t)	Q(t+1)	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

D flip-flop		
Q(t)	Q(t+1)	D
0	0	0
0	1	1
1	0	0
1	1	1

JK flip flop			
Q(t)	Q(t+1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

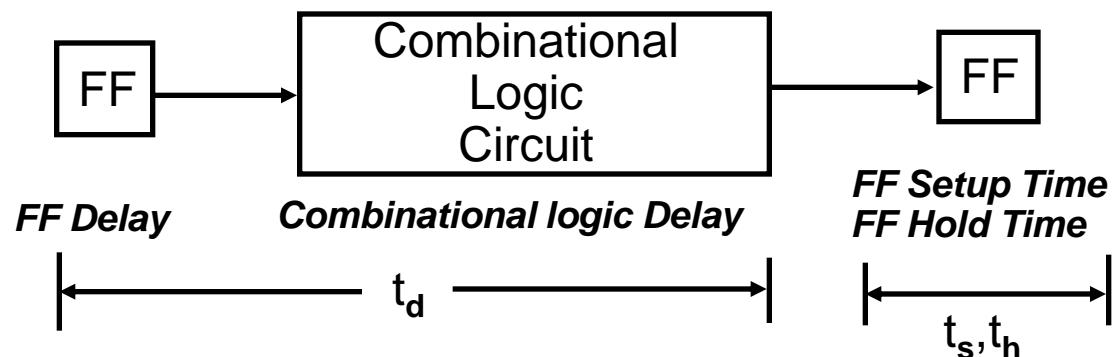
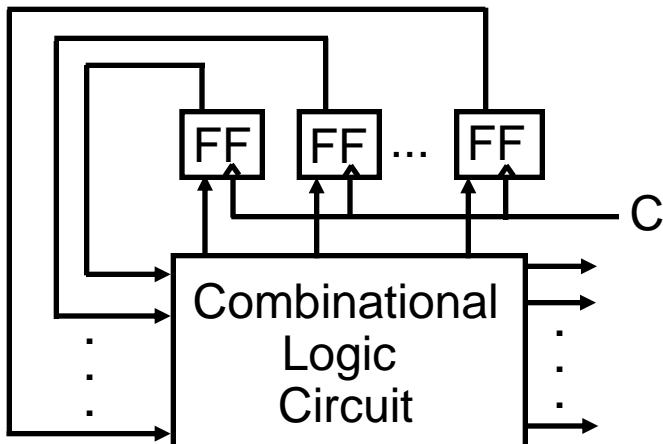
T flip-flop		
Q(t)	Q(t+1)	D
0	0	0
0	1	1
1	0	1
1	1	0



CLOCK PERIOD

Clock period determines how fast the digital circuit operates.
How can we determine the clock period ?

Usually, digital circuits are sequential circuits which has some flip flops



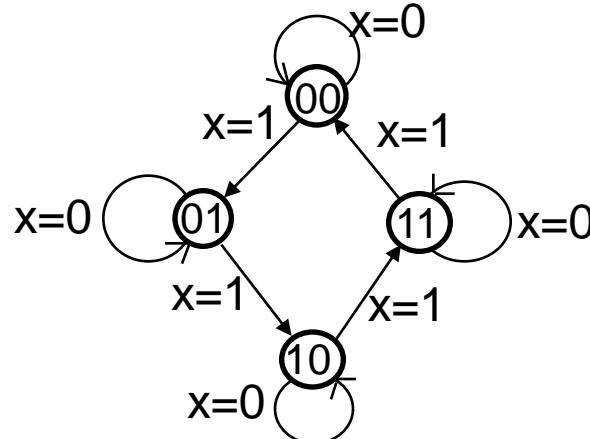
$$\text{clock period } T = t_d + t_s + t_h$$

DESIGN EXAMPLE

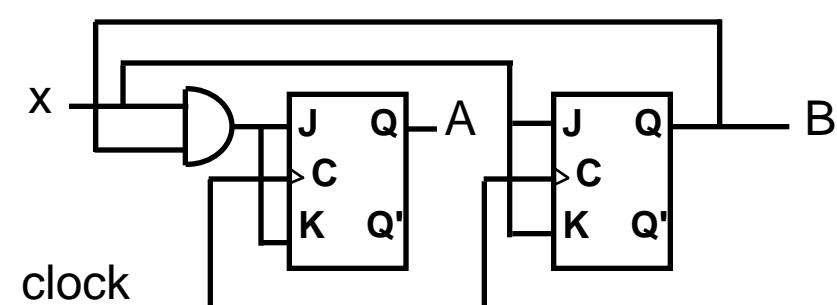
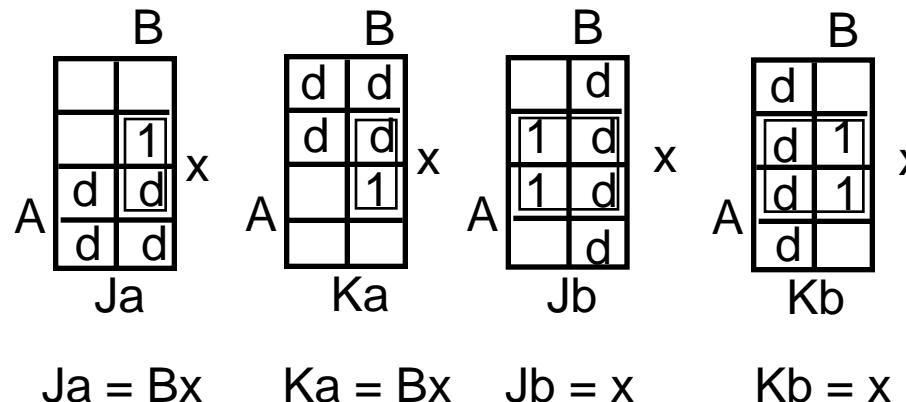
Design Procedure:

Specification \Rightarrow State Diagram \Rightarrow State Table \Rightarrow
 Excitation Table \Rightarrow Karnaugh Map \Rightarrow Circuit Diagram

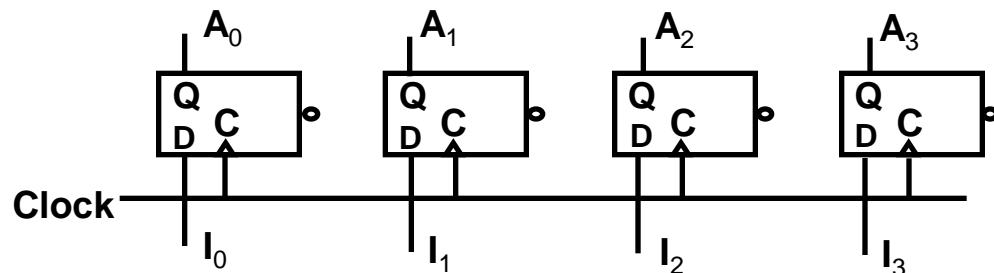
Example: 2-bit Counter \rightarrow 2 FF's



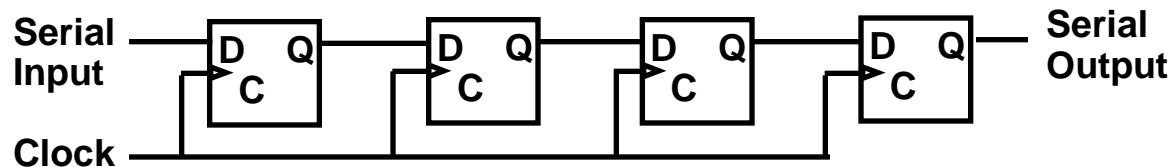
current state	input	next state		FF inputs			
		A	B	Ja	Ka	Jb	Kb
0 0	0	0	0	0	d	0	d
0 0	1	0	1	0	d	1	d
0 1	0	0	1	0	d	d	0
0 1	1	1	0	1	d	d	1
1 0	0	1	0	d	0	0	d
1 0	1	1	1	d	0	1	d
1 1	0	1	1	d	0	d	0
1 1	1	0	0	d	1	d	1



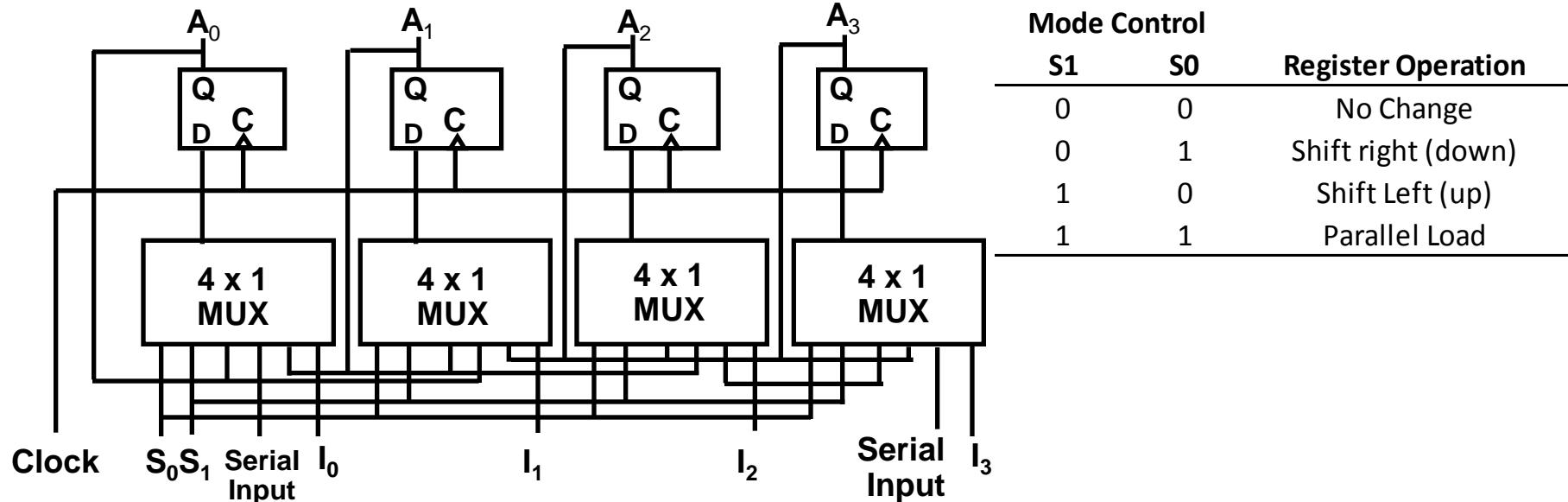
SEQUENTIAL CIRCUITS - Registers



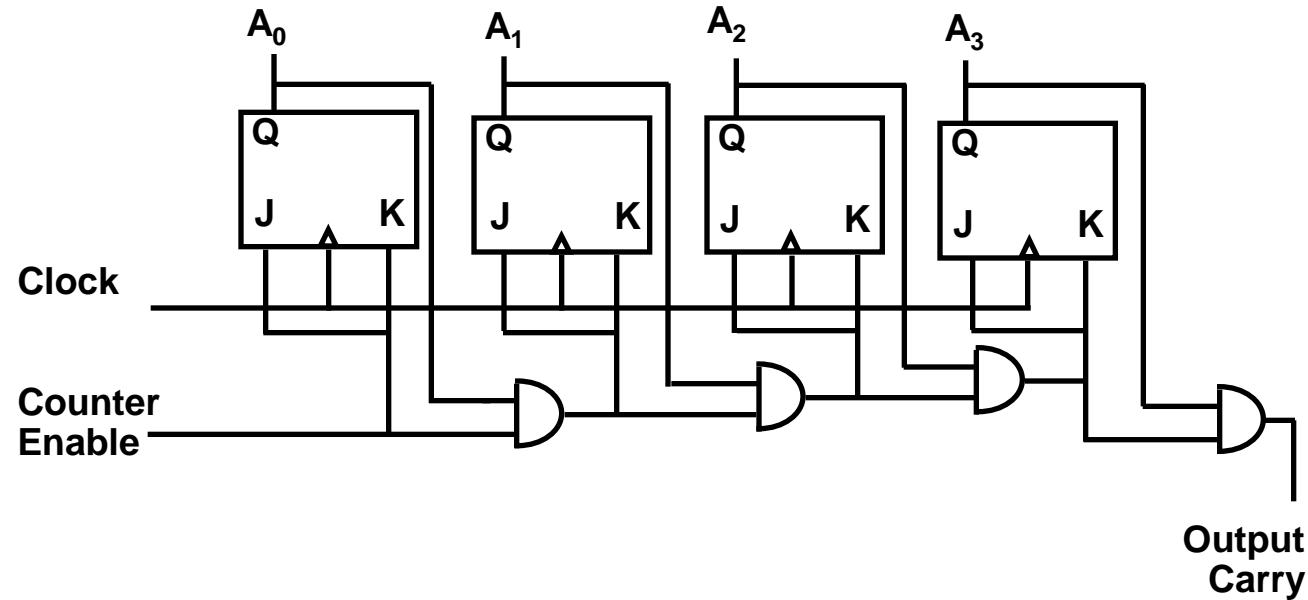
Shift Registers



Bidirectional Shift Register with Parallel Load



SEQUENTIAL CIRCUITS - Counters



Binary Counter with Parallel Load and Synchronous Clear

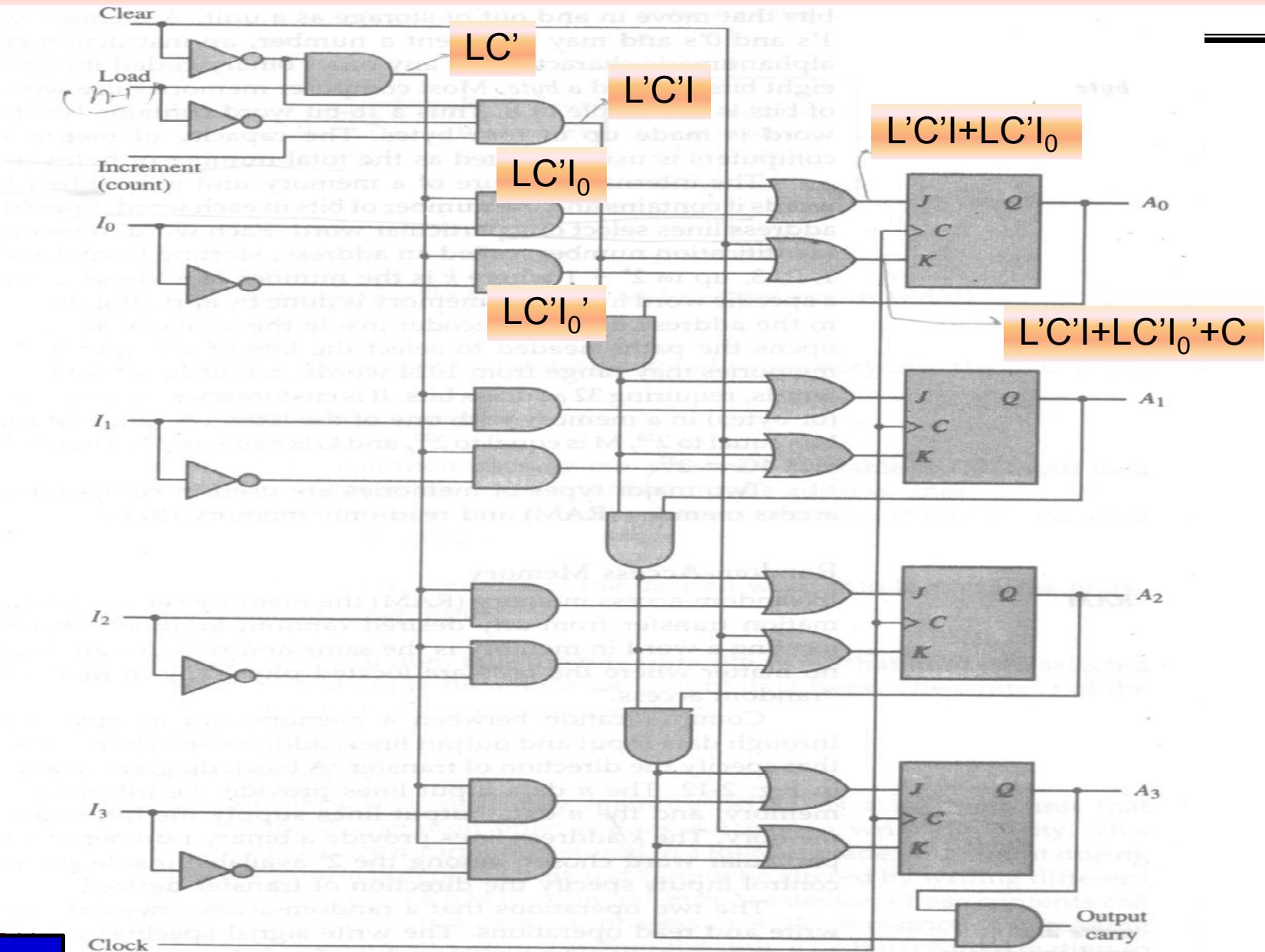
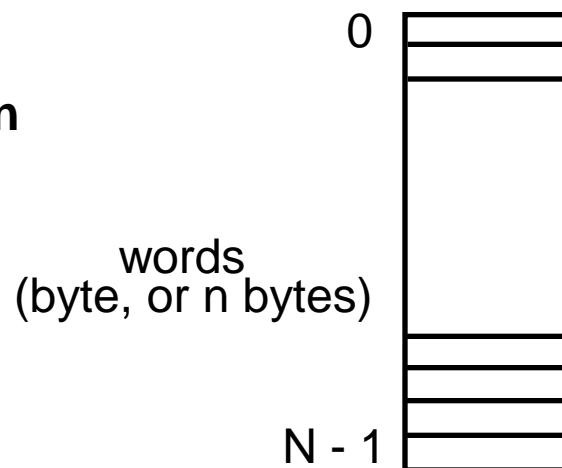


Figure 2-11 4-bit binary counter with parallel load and synchronous clear.

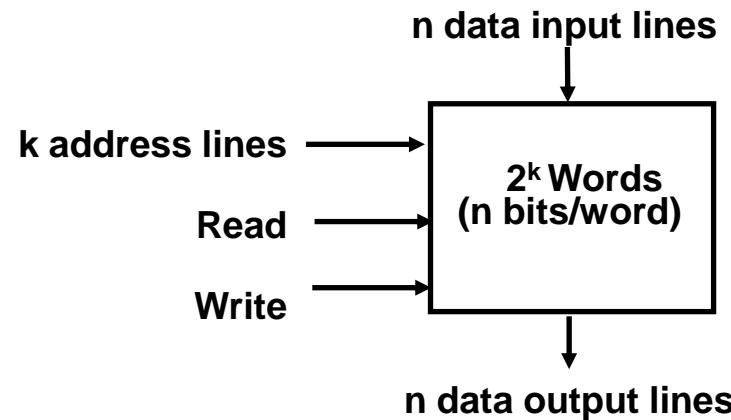
MEMORY COMPONENTS

Logical Organization



Random Access Memory

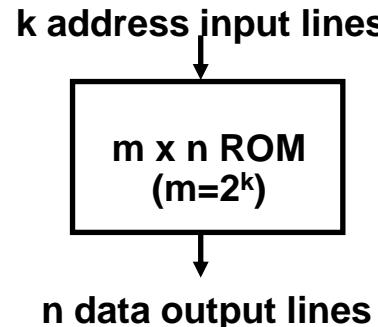
- Each word has a unique address
- Access to a word requires the same time independent of the location of the word
- Organization



READ ONLY MEMORY(ROM)

Characteristics

- Perform read operation only, write operation is not possible
- Information stored in a ROM is made permanent during production, and cannot be changed
- Organization



Information on the data output line depends only on the information on the address input lines.

--> Combinational Logic Circuit

$$\begin{aligned} X_0 &= A'B' + B'C \\ X_1 &= A'B'C + A'BC' \\ X_2 &= BC + AB'C' \\ X_3 &= A'BC' + AB' \\ X_4 &= AB \end{aligned}$$

$$\begin{aligned} X_0 &= A'B'C' + A'B'C + AB'C \\ X_1 &= A'B'C + A'BC' \\ X_2 &= A'BC + AB'C' + ABC \\ X_3 &= A'BC' + AB'C' + AB'C \\ X_4 &= ABC' + ABC \end{aligned}$$

Canonical minterms

address	Output				
	ABC	X ₀	X ₁	X ₂	X ₃
000	1	0	0	0	0
001	1	1	0	0	0
010	0	1	0	1	0
011	0	0	1	0	0
100	0	0	1	1	0
101	1	0	0	1	0
110	0	0	0	0	1
111	0	0	1	0	1

TYPES OF ROM

ROM

- Store information (function) during production
- Mask is used in the production process
- Unalterable
- Low cost for large quantity production --> used in the final products

PROM (Programmable ROM)

- Store info electrically using PROM programmer at the user's site
- Unalterable
- Higher cost than ROM -> used in the system development phase
-> Can be used in small quantity system

EPROM (Erasable PROM)

- Store info electrically using PROM programmer at the user's site
- Stored info is erasable (alterable) using UV light (electrically in some devices) and rewriteable
- Higher cost than PROM but reusable --> used in the system development phase. Not used in the system production due to erasability

INTEGRATED CIRCUITS

Classification by the Circuit Density

- SSI** - several (less than 10) independent gates
- MSI** - 10 to 200 gates; Perform elementary digital functions;
Decoder, adder, register, parity checker, etc
- LSI** - 200 to few thousand gates; Digital subsystem
Processor, memory, etc
- VLSI** - Thousands of gates; Digital system
Microprocessor, memory module

Classification by Technology

- TTL** - Transistor-Transistor Logic
Bipolar transistors
NAND
- ECL** - Emitter-coupled Logic
Bipolar transistor
NOR
- MOS** - Metal-Oxide Semiconductor
Unipolar transistor
High density
- CMOS** - Complementary MOS
Low power consumption