UNIT-II

Fundamentals of Boolean Algebra

- ***Basic Postulates***
- ***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 +.
- ***Postulate 2 (Existence of 1 and 0 element)***:
  (a) $a + 0 = a$ (identity for +), (b) $a \bullet 1 = a$ (identity for ●)
- ***Postulate 3 (Commutativity)***:
  (a) $a + b = b + a$, (b) $a \bullet b = b \bullet a$
- ***Postulate 4 (Associativity)***:
  (a) $a + (b + c) = (a + b) + c$ (b) $a\bullet (b\bullet c) = (a\bullet b) \bullet c$
- ***Postulate 5 (Distributivity)***:
  (a) $a + (b\bullet c) = (a + b) \bullet (a + c)$ (b) $a\bullet (b + c) = a\bullet b + a\bullet c$
- ***Postulate 6 (Existence of complement)***:
  (a) $A + \overline{A} = 1$ (b) $A \bullet \overline{A} = 0$
- Normally ● is omitted.

Fundamentals of Boolean Algebra

- ***Fundamental Theorems of Boolean Algebra***

- ***Theorem 1 (Idempotency)***:
  (a) $a + a = a$ (b) $aa = a$
- ***Theorem 2 (Null element)***:
  (a) $a + 1 = 1$ (b) $a0 = 0$
- ***Theorem 3 (Involution)***

  $\overline{\overline{A}} = A$

- ***Properties of 0 and 1 elements*** (Table 2.1):

OR	AND	Complement
$a + 0 = 0$	$a0 = 0$	0' = 1
$a + 1 = 1$	$a1 = a$	1' = 0

Fundamentals of Boolean Algebra (3)

- ***Theorem 4 (Absorption)***

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

- ***Examples***:
  - $(X + Y) + (X + Y)Z = X + Y$
  - $AB'(AB' + B'C) = AB'$
- ***Theorem 5***

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

- ***Examples***:
  - $B + AB'C'D = B + AC'D$
  - $(X + Y)((X + Y)' + Z) = (X + Y)Z$

Fundamentals of Boolean Algebra (4)

- ***Theorem 6***

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

- ***Examples***:
  - $ABC + AB'C = AC$
  - $(W' + X' + Y' + Z')(W' + X' + Y' + Z)(W' + X' + Y + Z')(W' + X' + Y + Z)$

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

    $= (W' + X' + Y')(W' + X' + Y)$

    $= (W' + X')$

Fundamentals of Boolean Algebra (5)

- ***Theorem 7***

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

- ***Examples***:
  - $wy' + wx'y + wxyz + wxz' = wy' + wx'y + wxy + wxz$
    $= wy' + wy + wxz'$
    $= w + wxz'$
    $= w$
  - $(x'y' + z)(w + x'y' + z') = (x'y' + z)(w + x'y')$

Fundamentals of Boolean Algebra (6)

- ***Theorem 8 (DeMorgan's Theorem)***

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

- Generalized DeMorgan's Theorem

  (a) $(\boldsymbol{a + b + \ldots z})' = a'b' \boldsymbol{\ldots z'}$ (b) $(ab \boldsymbol{\ldots z})' = a' + b' \boldsymbol{+ \ldots z'}$

- ***Examples***:
  - $(a + bc)' = (a + (bc))'$
    $= a'(bc)'$
    $= a'(b' + c')$
    $= a'b' + a'c'$

  Note: $(a + bc)' \neq a'b' + c'$

Logic Gates

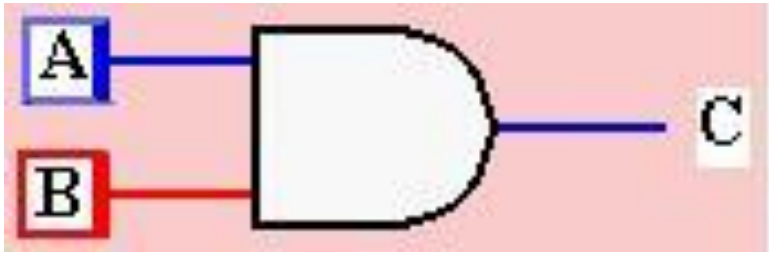
- ***Electrical Signals and Logic Values***

Electric Signal	Logic Value	
	Positive Logic	Negative Logic
High Voltage (H)	1	0
Low Voltage (L)	0	1

  - A signal that is set to logic 1 is said to be *asserted*, *active*, or *true*.
  - An *active-high* signal is asserted when it is high (positive logic).
  - An *active-low* signal is asserted when it is low (negative logic).

AND

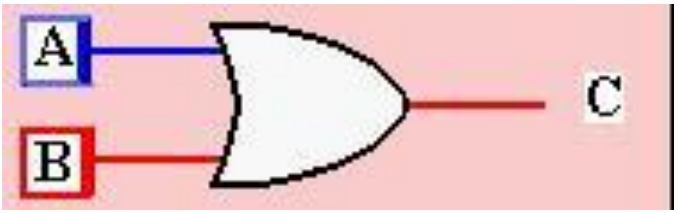
– Logic notation A•B = C
(Sometimes AB = C)



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

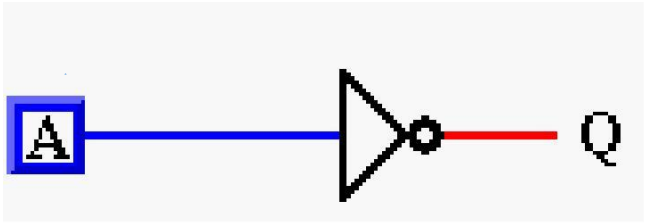
OR

- Logic notation A + B = C



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

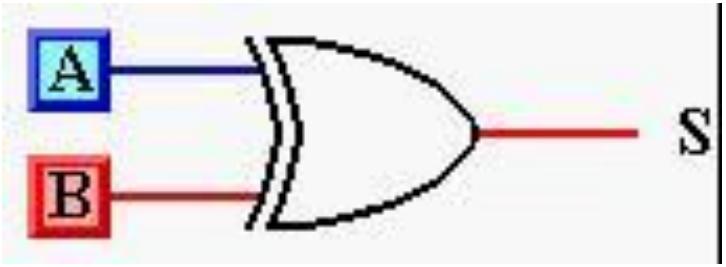
Inversion (NOT)



Logic: $Q = \overline{A}$

0	1
1	0

Exclusive OR (XOR)



Either A or B, but not both

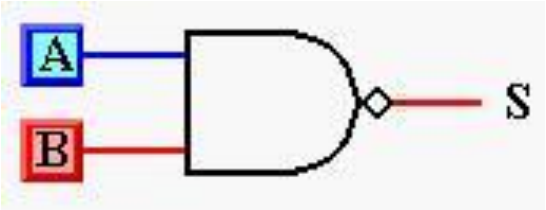
This is sometimes called the **inequality detector**, because the result will be 0 when the inputs are the same and 1 when they are different.

The truth table is the same as for S on Binary Addition. S = A ⊕ B

A	B	S
0	0	0
1	0	1
0	1	1
1	1	0

UNIVERSAL GATES

NAND (NOT AND)

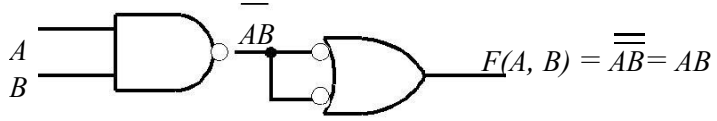


$$Q = \overline{A \cdot B}$$

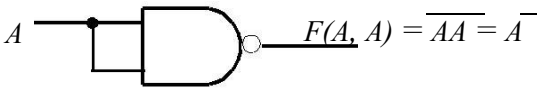
A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0

Basic Functional Components

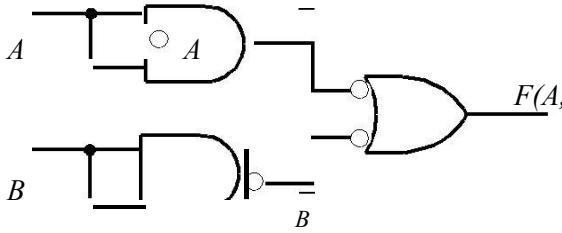
- AND, OR, and NOT gates constructed exclusively from NAND gates



AND gate



NOT gate



OR gate

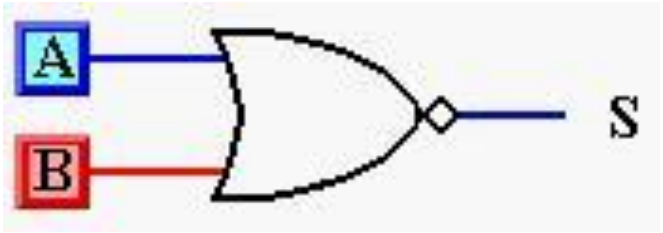
$$F_{NAND}(A, A) = A \cdot A = A = F_{NOT}(A)$$

$$\bar{F}_{NAND}(A, B) = A \cdot B = A \cdot B = F_{AND}(A, B)$$

$$F_{NAND}(A, \bar{B}) = A \cdot \bar{B} = A + B = F_{OR}(A, B)$$

Hence, NAND gate may be used to implement all three elementary operators.

NOR (NOT OR)

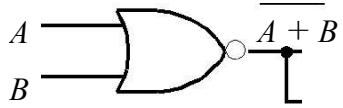


$$Q = \overline{A + B}$$

A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0

Basic Functional Components

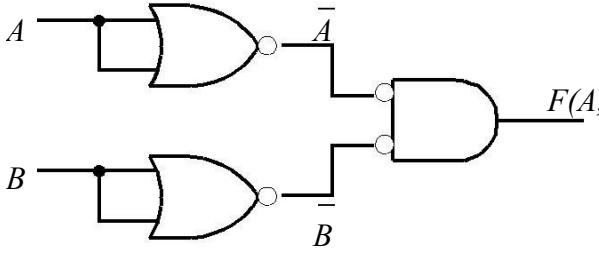
- AND, OR, and NOT gates constructed exclusively from NOR gates.



OR gate

$F(A, B) = A + B\,A$

NOT gate



AND gate

$$F_{NOR}(A, A) = \overline{A + A} = \bar{A} = F_{NOT}(A)$$

$$\bar{F}_{NOR}(A, B) = \overline{\overline{A + B}} = A + B = F_{OR}(A, B)$$

$$F_{NOR}(\bar{A}, \bar{B}) = \overline{\bar{A} + \bar{B}} = A \cdot B = F_{AND}(A, B)$$

Hence, NOR gate may be used to implement all three elementary operators.

Summary

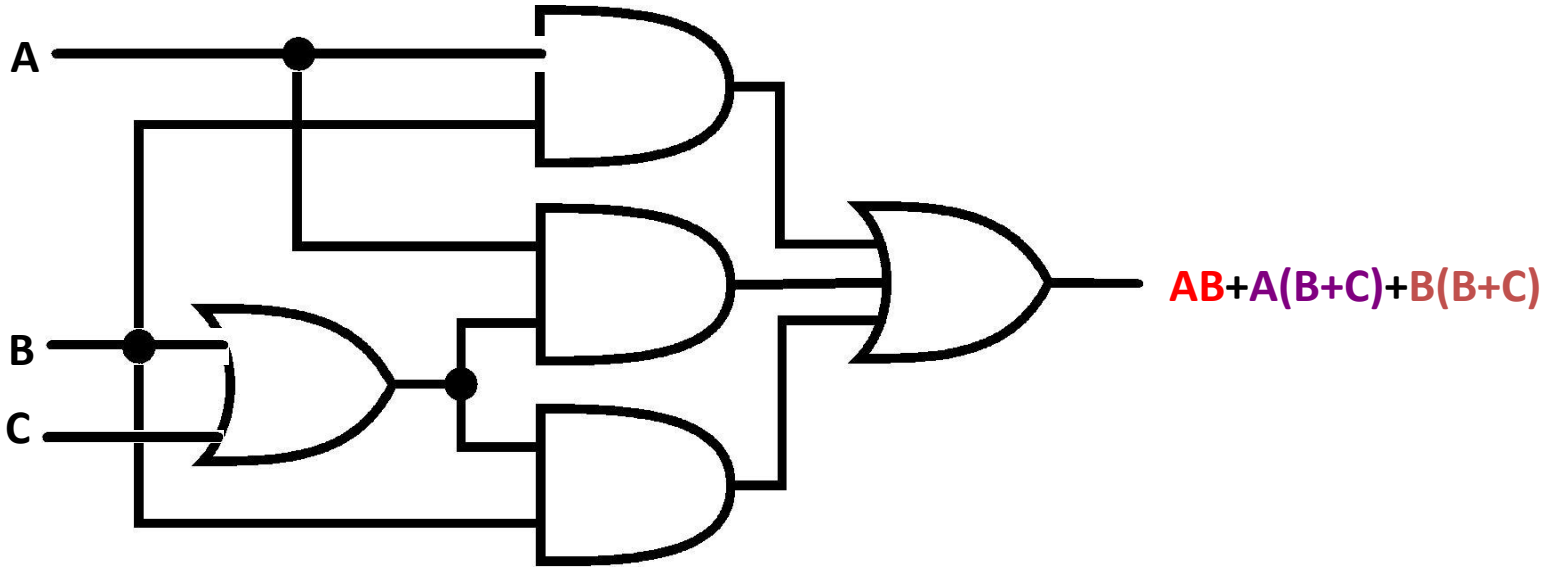
Summary for all 2-input gates							
Inputs		Output of each gate					
A	B	AND	NAND	OR	NOR	XOR	XNOR
0	0	0	1	0	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	0	1	0	0	1

MINIMIZATON OF LOGIC EXPRESSION

- Goal -- minimize the cost of realizing a switching function
- Cost measures and other considerations
  - Number of gates
  - Number of levels
  - Gate fan in and/or fan out
  - Interconnection complexity
  - Preventing hazards
- Two-level realizations
  - Minimize the number of gates (terms in switching function)
  - Minimize the fan in (literals in switching function)
- Commonly used techniques
  - Boolean algebra postulates and theorems
  - Karnaugh maps

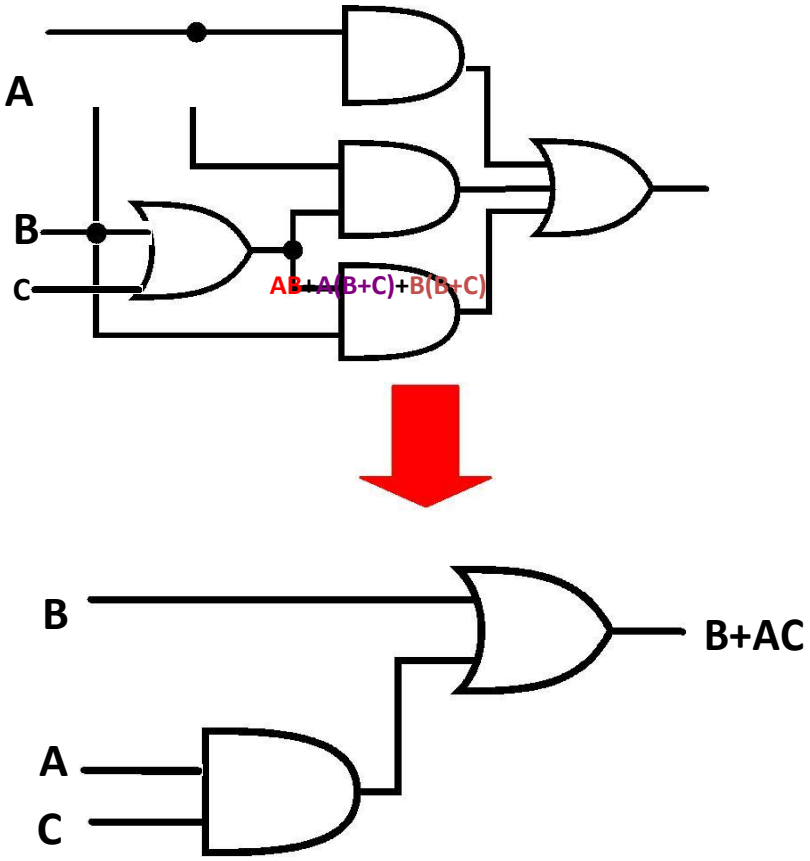
Simplification Using Boolean Algebra

- A simplified Boolean expression uses the fewest gates possible to implement a given expression.



Simplification Using Boolean Algebra

- AB+A(B+C)+B(B+C)
  - (distributive law)
    - AB+AB+AC+BB+BC
  - (BB=B)
    - AB+AB+AC+**B**+BC
  - (AB+AB=AB)
    - **AB**+AC+B+BC
  - (B+BC=B)
    - AB+AC+**B**
  - (AB+B=B)
    - **B**+AC



Simplification Using Boolean Algebra

- Try these:

$$[\, A\overline{B}\,(C + BD) + \overline{A}\,\overline{B}\,]C$$

$$\overline{A}\,BC + A\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\overline{B}\,C +$$

$$\overline{ABC\,AB} + \overline{A}\overline{C} + A\,B\,C$$

8

Standard Forms of Boolean Expressions

- All Boolean expressions, regardless of their form, can be converted into either of two standard forms:
  - The sum-of-products (SOP) form
  - The product-of-sums (POS) form
- Standardization makes the evaluation, simplification, and implementation of Boolean expressions much more systematic and easier.

The Sum-of-Products (SOP) Form

- An SOP expression → when two or more product terms are summed by Boolean addition.
  - Examples:

$$AB + ABC$$

$$ABC + CDE + \overline{B}\,C\overline{D}$$

$$\overline{A}\,B + \overline{A}\,B\overline{C} + AC$$

  - Also:

$$A + \overline{A}\,\overline{B}C + BC\overline{D}$$

- In an SOP form, a single overbar cannot extend over more than one variable; however, more than one variable in a term can have an overbar.
  - example: $A\,\overline{B}\,\overline{C}$ is OK
  - BUT NOT: $\overline{ABC}$

Converting Product Terms to Standard SOP

- **Step 1:** Multiply each nonstandard product term by a term made up of the sum of a missing variable and its complement. This results in two product terms.
  - As you know, you can multiply anything by 1 without changing its value.
- **Step 2:** Repeat step 1 until all resulting product term contains all variables in the domain in either complemented or uncomplemented form. In converting a product term to standard form, the number of product terms is doubled for each missing variable.

Converting Product Terms to Standard SOP (example)

- Convert the following Boolean expression into standard SOP form:

$$A\overline{B}C + \overline{A}\overline{B} + AB\overline{C}\,D$$

$$A\overline{B}\,C = A\overline{B}\,C\,(D + \overline{D}) = A\overline{B}\,CD + A\overline{B}\,C\overline{D}$$

$$\overline{A}\,\overline{B} = \overline{A}\,\overline{B}\,(C + \overline{C}) = \overline{A}\,\overline{B}\,C + \overline{A}\,\overline{B}\,C$$

$$\overline{A}\,\overline{B}\,C\,(D + \overline{D}) + \overline{A}\,\overline{B}\,\overline{C}\,(D + \overline{D}) = \overline{A}\,\overline{B}\,CD + \overline{A}\,\overline{B}\,C\overline{D} + \overline{A}\,\overline{B}\,\overline{C}\,D + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$$

$$A\overline{B}\,C + \overline{A}\,\overline{B} + AB\overline{C}\,D = A\overline{B}\,CD + A\overline{B}\,C\overline{D} + \overline{A}\,\overline{B}\,CD + \overline{A}\,B\,\overline{C}D + \overline{A}\,\overline{B}\,C\,D + \overline{A}\,\overline{B}\,\overline{C}\,D + AB\overline{C}\,D$$

8

The Product-of-Sums (POS) Form

- When two or more sum terms are multiplied, the result expression is a product-of-sums (POS):
  - Examples:

$$(\overline{A}+B)(A+\overline{B}+C)$$

$$(\overline{A}+\overline{B}+\overline{C})(C+\overline{D}+E)(\overline{B}+C+D)$$

$$(A+B)(A+\overline{B}+C)(\overline{A}+C)$$

  - ~~Also:~~

$$A(A+B+C)(B+C+D)$$

- In a POS form, a single overbar cannot extend over more than one variable; however, more than one variable in a term can have an overbar
  - example: $\overline{A}+\overline{B}+\overline{C}$ is OK
  - BUT NOT $\overline{A+B+C}$

Converting a Sum Term to Standard POS

- **Step 1:** Add to each nonstandard product term a term made up of the product of the missing variable and its complement. This results in two sum terms.
  - As you know, you can add 0 to anything without changing its value.
- **Step 2:** Apply rule → A+BC=(A+B)(A+C).
- **Step 3:** Repeat step 1 until all resulting sum terms contain all variable in the domain in either complemented or uncomplemented form.

Converting a Sum Term to Standard POS (example)

- Convert the following Boolean expression into standard POS form:

$$(A+\overline{B}+C)(\overline{B}+C+\overline{D})(A+\overline{B}+\overline{C}+D)$$

$$A+\overline{B}+C = A+\overline{B}+C+D\overline{D} = (A+\overline{B}+C+D)(A+\overline{B}+C+\overline{D})$$

$$\overline{B}+C+\overline{D} = \overline{B}+C+\overline{D}+A\overline{A} = (A+\overline{B}+C+\overline{D})(\overline{A}+\overline{B}+C+\overline{D})$$

$$(A+\overline{B}+C)(\overline{B}+C+\overline{D})(A+\overline{B}+\overline{C}+D) =$$

$$(A+\overline{B}+C+D)(A+\overline{B}+C+\overline{D})(A+\overline{B}+C+\overline{D})(\overline{A}+\overline{B}+C+\overline{D})(A+\overline{B}+\overline{C}+D)$$

Boolean Expressions & Truth Tables

- All standard Boolean expression can be easily converted into truth table format using binary values for each term in the expression.
- Also, standard SOP or POS expression can be determined from the truth table.

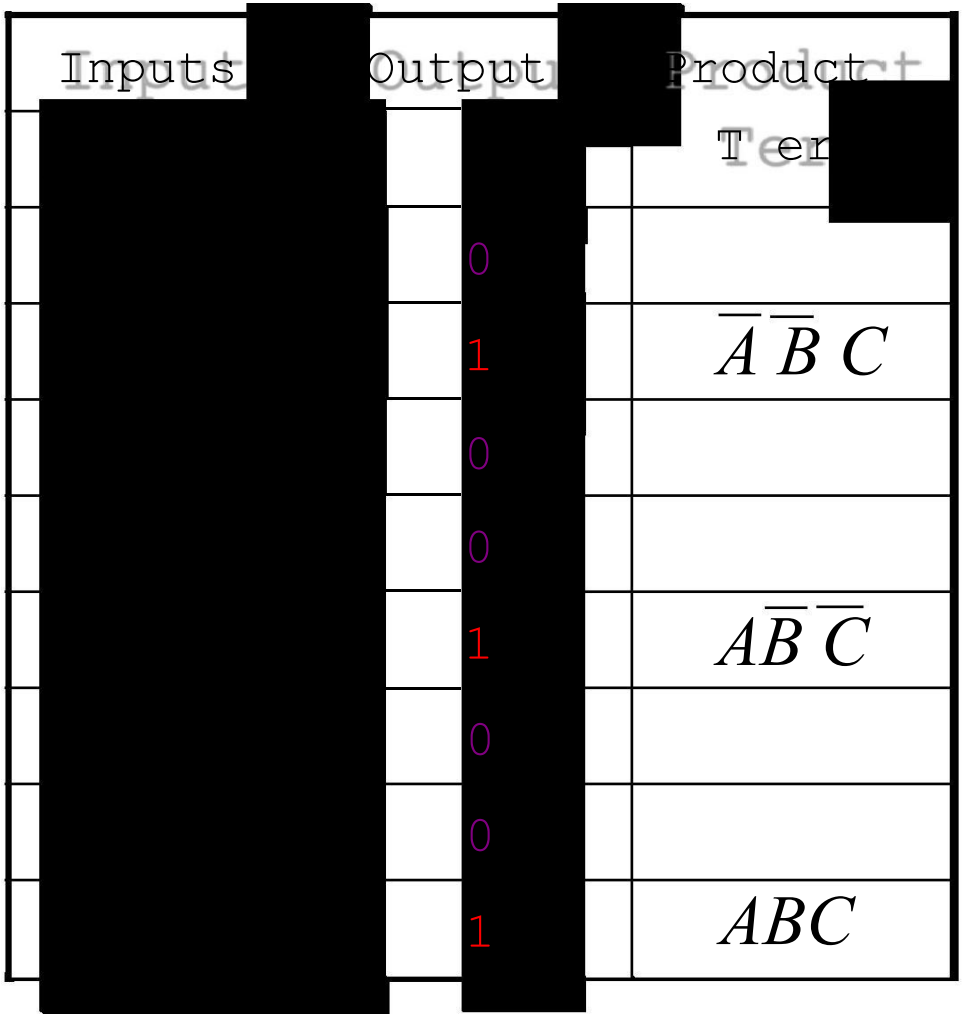
Converting SOP Expressions to Truth Table Format

- Recall the fact:
  - An SOP expression is equal to 1 only if at least one of the product term is equal to 1.
- Constructing a truth table:
  - **Step 1:** List all possible combinations of binary values of the variables in the expression.
  - **Step 2:** Convert the SOP expression to standard form if it is not already.
  - **Step 3:** Place a 1 in the output column (X) for each binary value that makes the standard SOP expression a 1 and place 0 for all the remaining binary values.

Converting SOP Expressions to Truth Table Format (example)

- Develop a truth table for the standard SOP expression

$$\overline{A}\overline{B}C + A\overline{B}\overline{C} + ABC$$



Converting POS Expressions to Truth Table Format

- Recall the fact:
  - A POS expression is equal to 0 only if at least one of the product term is equal to 0.
- Constructing a truth table:
  - **Step 1:** List all possible combinations of binary values of the variables in the expression.
  - **Step 2:** Convert the POS expression to standard form if it is not already.
  - **Step 3:** Place a 0 in the output column (X) for each binary value that makes the standard POS expression a 0 and place 1 for all the remaining binary values.

Converting POS Expressions to Truth Table Format (example)

- Develop a truth table for the standard SOP expression

$$(A+B+C)(A+\overline{B}+C)(A+\overline{B}+\overline{C})(A+\overline{B}+\overline{C})(\overline{A}+B+C)$$

Input	Output	Product Term
	0	$(A+B+C)$
	1	
	0	$(A+\overline{B}+C)$
	0	$(A+\overline{B}+\overline{C})$
	1	
	0	$(\overline{A}+B+\overline{C})$
	0	$(\overline{A}+\overline{B}+C)$
	1	

Determining Standard Expression from a Truth Table

- To determine the standard **SOP expression** represented by a truth table.
- Instructions:
  - **Step 1:** List the binary values of the input variables for which the output is 1.
  - **Step 2:** Convert each binary value to the corresponding product term by replacing:
    - each 1 with the corresponding variable, and
    - each 0 with the corresponding variable complement.
- Example: 1010 → $A\overline{B}C\overline{D}$

Determining Standard Expression from a Truth Table

- To determine the standard **POS expression** represented by a truth table.
- Instructions:
  - **Step 1:** List the binary values of the input variables for which the output is 0.
  - **Step 2:** Convert each binary value to the corresponding product term by replacing:
    - each 1 with the corresponding variable complement, and
    - each 0 with the corresponding variable.
- Example: 1001 → $\overline{A} + B + C + \overline{D}$

The Karnaugh Map

- Feel a little difficult using Boolean algebra laws, rules, and theorems to simplify logic?
- A K-map provides a systematic method for simplifying Boolean expressions and, if properly used, will produce the simplest SOP or POS expression possible, known as the minimum expression.

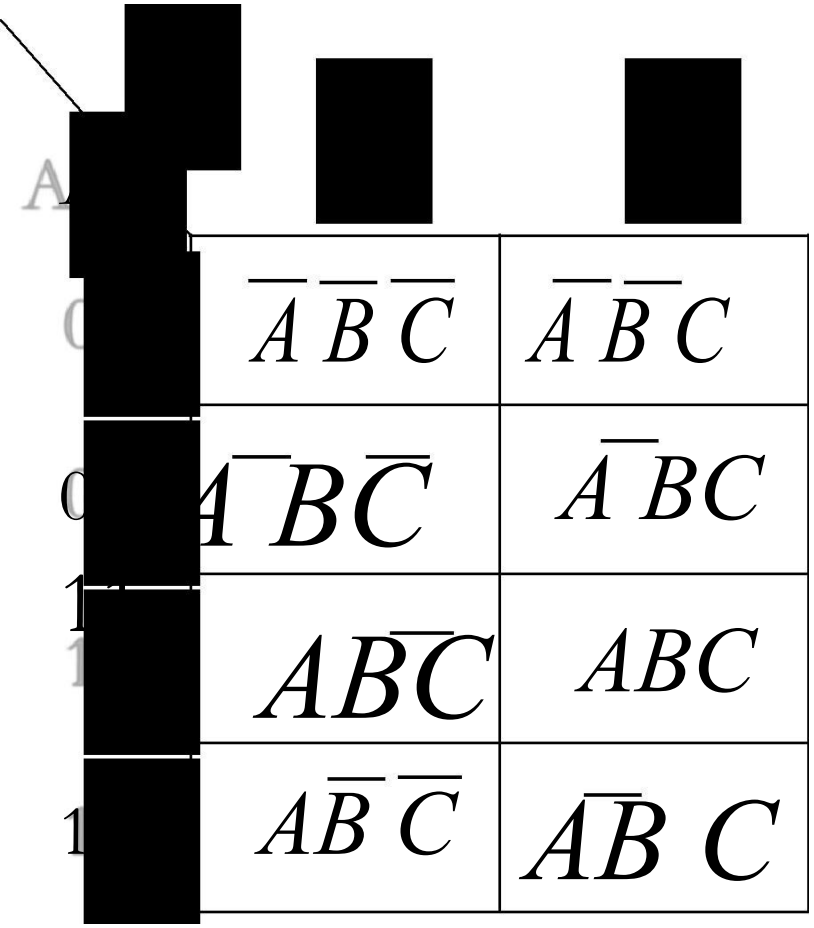
What is K-Map

- **It s si ilar to truth ta le; i stead of ei g orga ized**

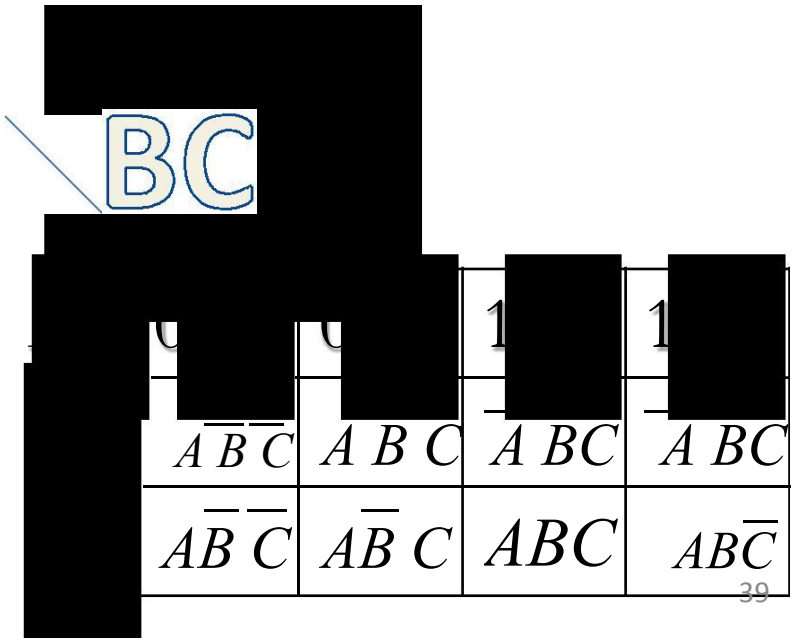
  (i/p and o/p) into columns and rows, the K-map is an array of cells in which each cell represents a binary value of the input variables.
- The cells are arranged in a way so that simplification of a given expression is simply a matter of properly grouping the cells.
- K-maps can be used for expressions with 2, 3, 4, and 5 variables.

The 3 Variable K-Map

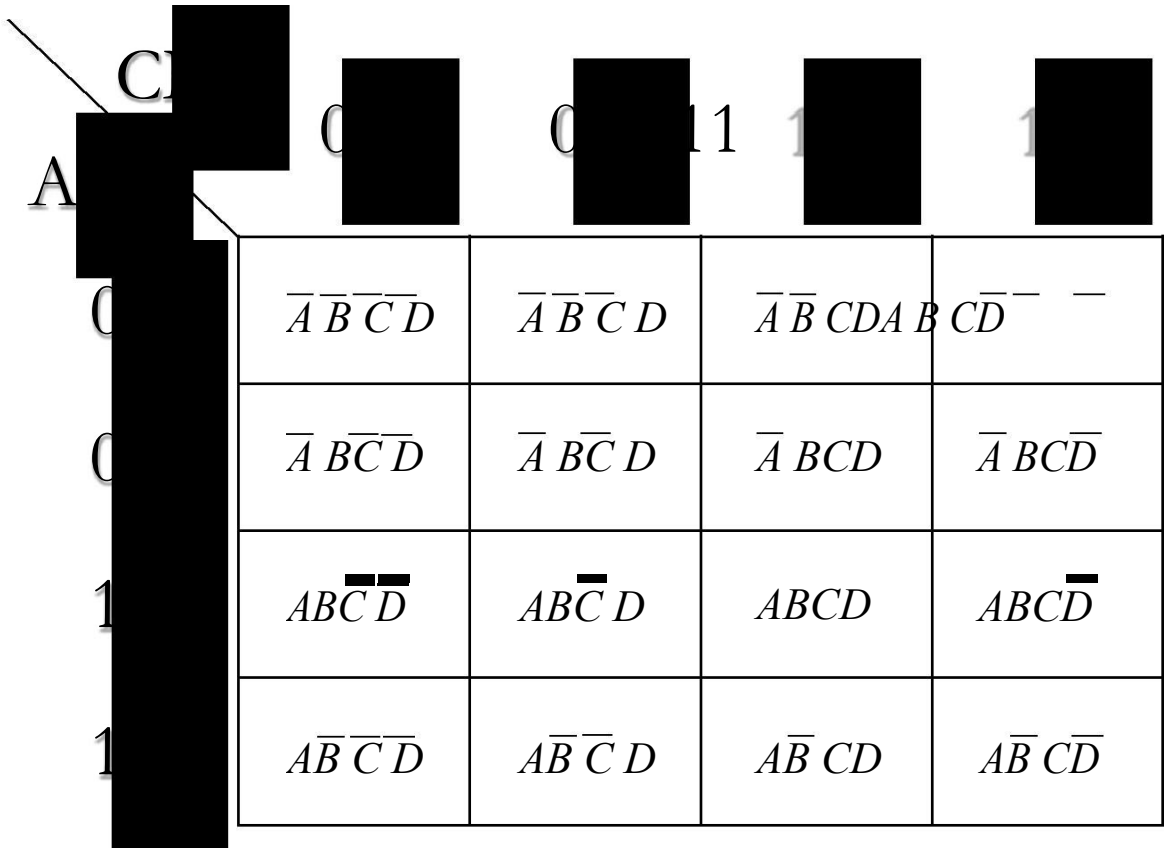
- There are 8 cells as shown:



Or



The 4-Variable K-Map



			11	
	$\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$	$\overline{A}\,\overline{B}\,\overline{C}\,D$	$\overline{A}\,\overline{B}\,CDA\,B\,C\overline{D}$	
	$\overline{A}\,B\overline{C}\,\overline{D}$	$\overline{A}\,B\overline{C}\,D$	$\overline{A}\,BCD$	$\overline{A}\,BC\overline{D}$
1	$AB\overline{C}\,\overline{D}$	$AB\overline{C}\,D$	$ABCD$	$ABC\overline{D}$
1	$A\overline{B}\,\overline{C}\,\overline{D}$	$A\overline{B}\,\overline{C}\,D$	$A\overline{B}\,CD$	$A\overline{B}\,C\overline{D}$

K-Map SOP Minimization

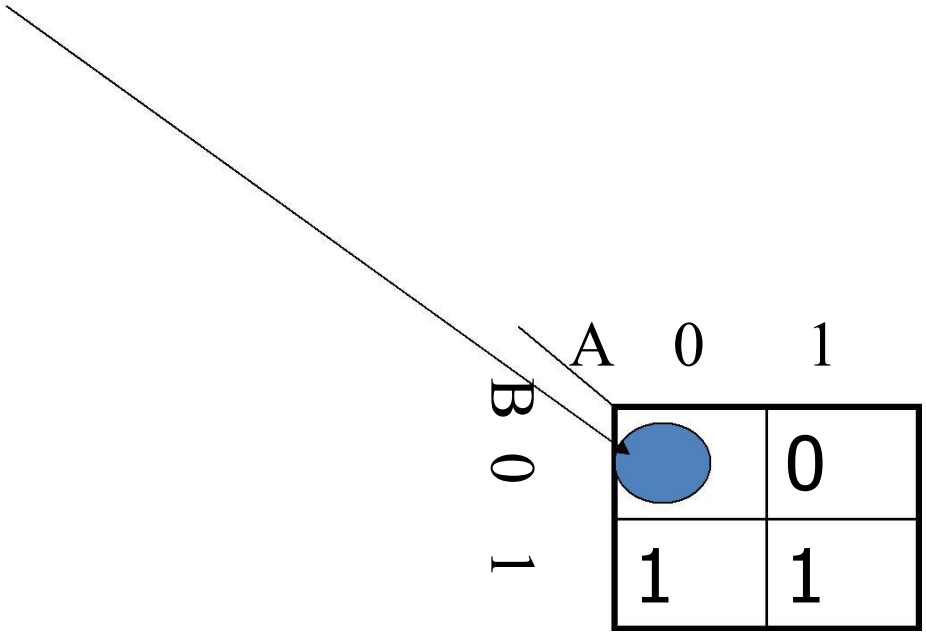
- The K-Map is used for simplifying Boolean expressions to their minimal form.
- A minimized SOP expression contains the fewest possible terms with fewest possible variables per term.
- Generally, a minimum SOP expression can be implemented with fewer logic gates than a standard expression.

Karnaugh Maps (K-maps)

- If m_i is a minterm of *f,* then place a 1 in cell *i* of the K-map.
- If M_i is a maxterm of *f*, then place a 0 in cell *i.*
- If d_i **is a do t are of** *f,* then place a *d or x* in cell *i.*

Examples

- *Two variable K-map*
  f(A,B)=$\sum$*m*(0,1,3)=A`B`+A`B+AB



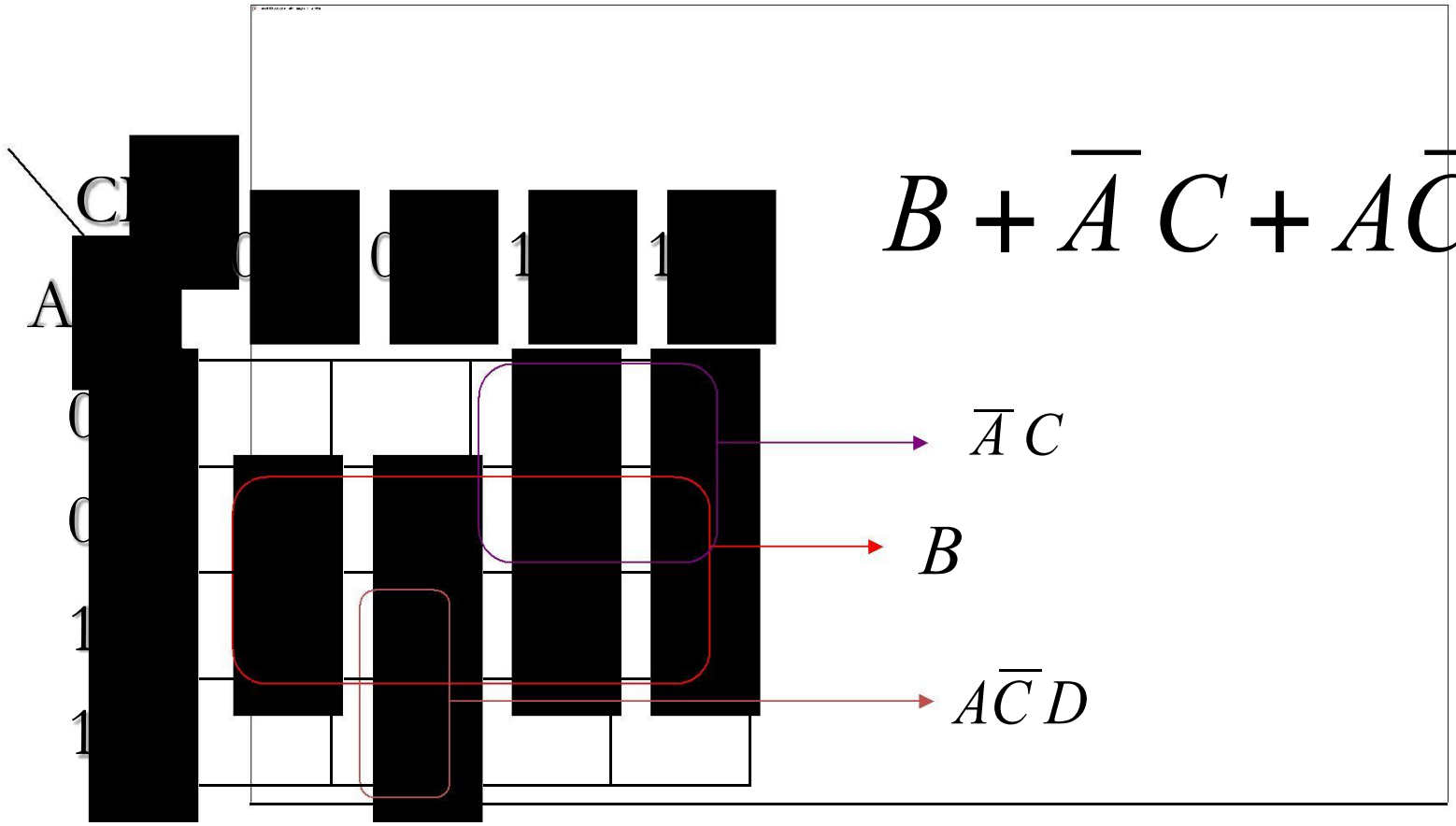
Grouping the 1s (rules)

1. A group must contain either 1,2,4,8,or 16 cells (depending on number of variables in the expression)
2. Each cell in a group must be adjacent to one or more cells in that same group, but all cells in the group do not have to be adjacent to each other.
3. Always include the largest possible number of 1s in a group in accordance with rule 1.
4. Each 1 on the map must be included in at least one group. The 1s already in a group can be included in another group as long as the overlapping groups include noncommon 1s.

Determining the Minimum SOP Expression from the Map

2. Determine the minimum product term for each group.
   - For a 3-variable map:
     1. A 1-cell group yields a 3-variable product term
     2. A 2-cell group yields a 2-variable product term
     3. A 4-cell group yields a 1-variable product term
     4. An 8-cell group yields a value of 1 for the expression.
   - For a 4-variable map:
     1. A 1-cell group yields a 4-variable product term
     2. A 2-cell group yields a 3-variable product term
     3. A 4-cell group yields a 2-variable product term
     4. An 8-cell group yields a a 1-variable product term
     5. A 16-cell group yields a value of 1 for the expression.

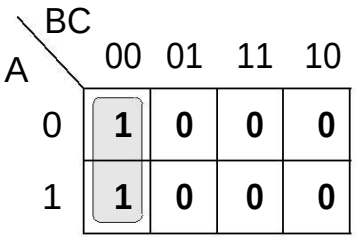
Determining the Minimum SOP Expression from the Map (example)



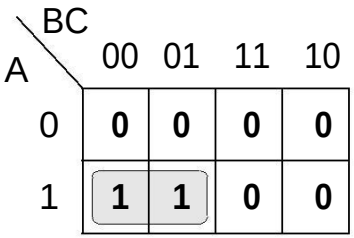
$$B + \overline{A}\,C + A\overline{C}\,D$$

Three-Variable K-Maps

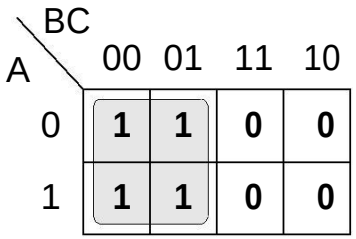
$f = \sum(0,4) = \overline{B}\,\overline{C}$



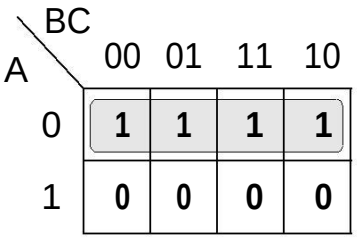
$f = \sum(4,5) = A\,\overline{B}$



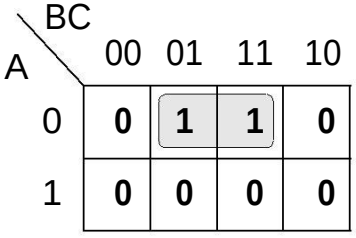
$f = \sum(0,1,4,5) = \overline{B}$



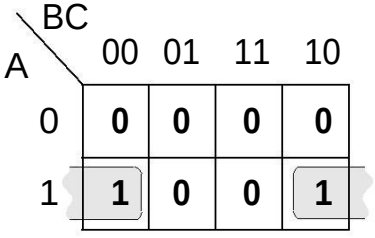
$f = \sum(0,1,2,3) = \overline{A}$



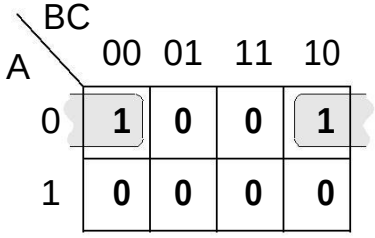
$f = \sum(0,4) = \overline{A}\,C$



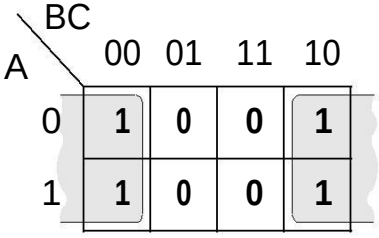
$f = \sum(4,6) = A\,\overline{C}$



$f = \sum(0,2) = \overline{A}\,\overline{C}$

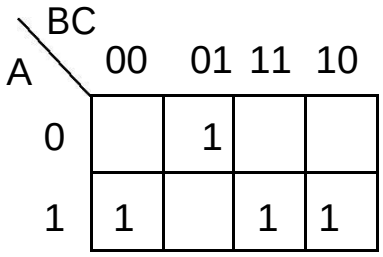


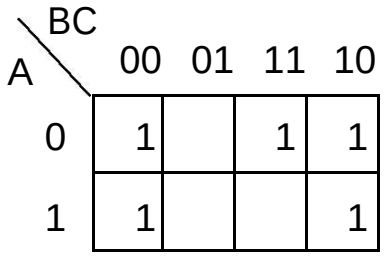
$f = \sum(0,2,4,6) = \overline{C}$

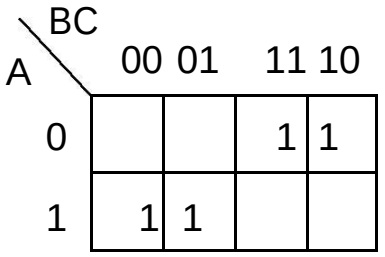


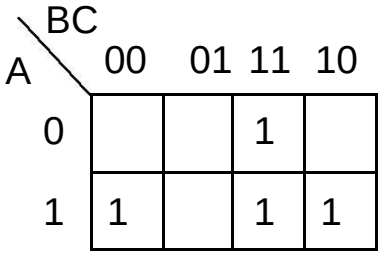
Three-Variable K-Map Examples

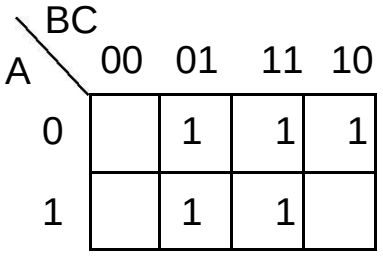
- We can write any way either AB and C or A BC

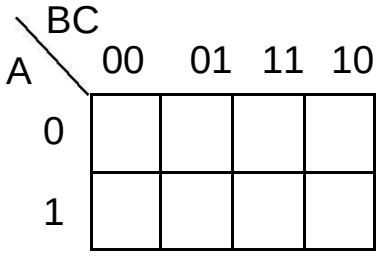




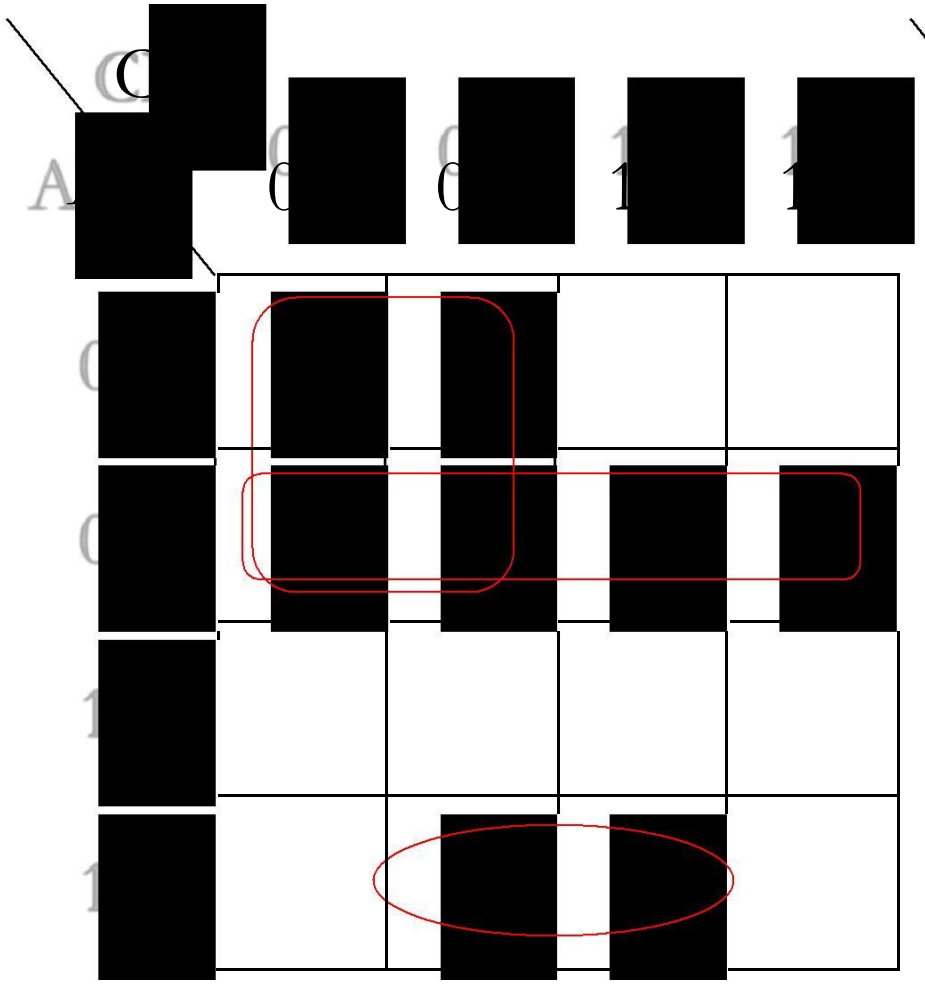




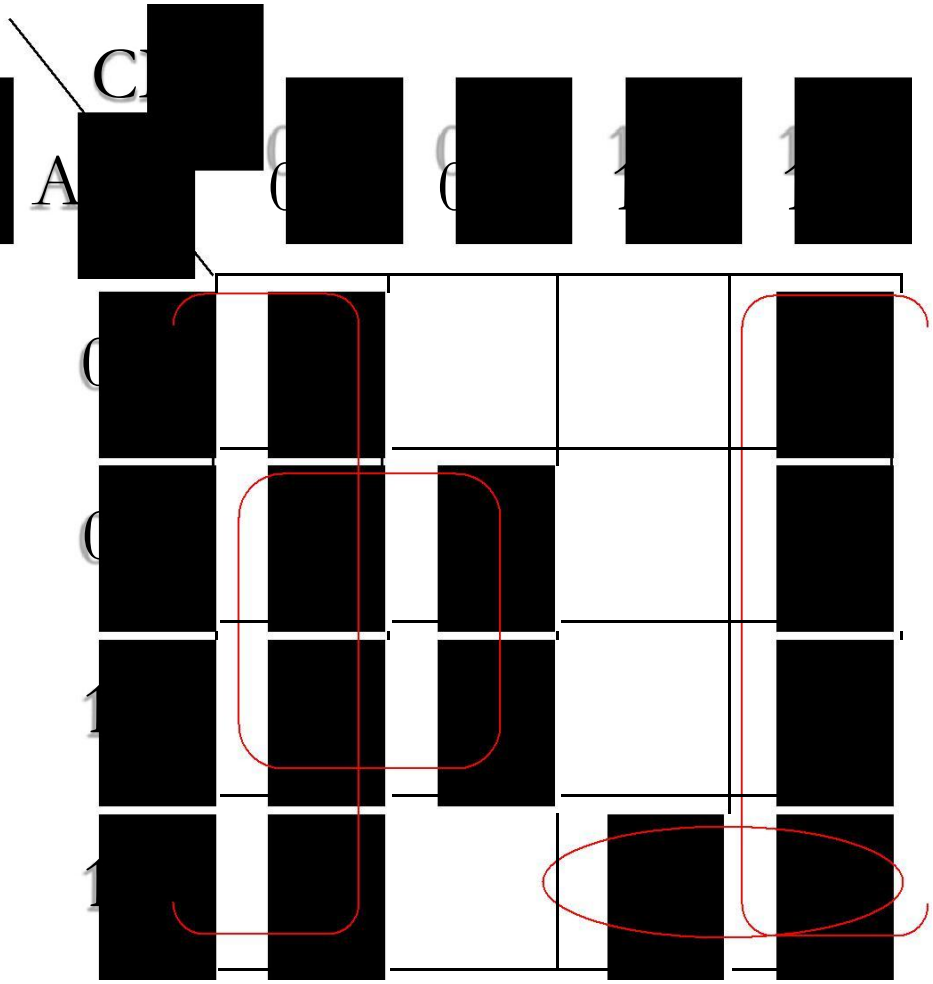




Determining the Minimum SOP Expression from the Map (exercises)



$$\overline{A}B + \overline{A}\overline{C} + A\overline{B}D$$



$$\overline{D} + A\overline{B}C + B\overline{C}$$

Four-Variable K-Maps

CD \ AB	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	0	0	0	0

$f = \sum (4, 5, 6, 7) = \overline{A} \bullet B$

CD \ AB	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	0	0	1	0
10	0	0	1	0

$f = \sum (3, 7, 11, 15) = C \bullet D$

CD \ AB	00	01	11	10
00	1	0	1	0
01	0	1	0	1
11	1	0	1	0
10	0	1	0	1

$f = \sum (0, 3, 5, 6, 9, 10, 12, 15)$

$f = A \otimes B \otimes C \otimes D$

CD \ AB	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

$f = \sum (1, 2, 4, 7, 8, 11, 13, 14)$

$f = A \oplus B \oplus C \oplus D$

CD \ AB	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	1	1	0
10	0	1	1	0

$f = \sum (1, 3, 5, 7, 9, 11, 13, 15)$

$f = D$

CD \ AB	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	1	0	0	1
10	1	0	0	1

$f = \sum (0, 2, 4, 6, 8, 10, 12, 14)$

$f = \overline{D}$

CD \ AB	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	1	1	1
10	0	0	0	0

$f = \sum (4, 5, 6, 7, 12, 13, 14, 15)$

$f = B$

CD \ AB	00	01	11	10
00	1	1	1	1
01	0	0	0	0
11	0	0	0	0
10	1	1	1	1

$f = \sum (0, 1, 2, 3, 8, 9, 10, 11)$

$f = \overline{B}$

Practicing K-Map (SOP)

$$A\overline{B}\,C + \overline{A}\,BC + \overline{A}\,\overline{B}\,C + \overline{A}\,\overline{B}\,C + \overline{A}\overline{B}\,C$$

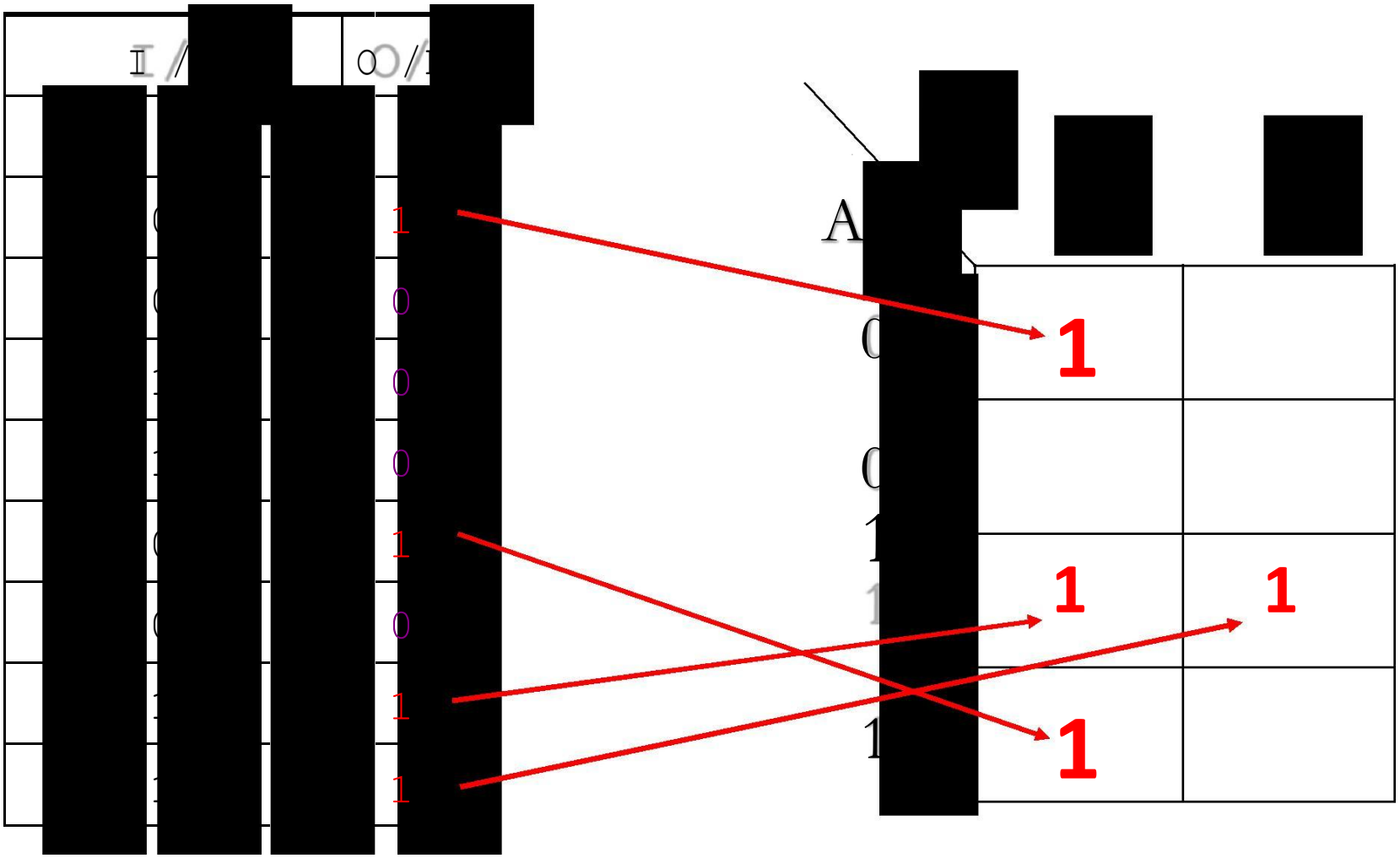
$$\overline{B} + \overline{A}\,C$$

$$\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\overline{C}\,D + A\overline{B}\overline{C}\,D + \overline{A}\,\overline{B}\,CD + \overline{A}\overline{B}$$

$$\overline{C}\overline{D} + \overline{A}\,B\,\overline{C}D + \overline{A}\,BCD + AB\overline{C}\overline{D} + \overline{A}\overline{B}\,CD$$

$$\overline{D} + \overline{B}\,C$$

Mapping Directly from a Truth Table

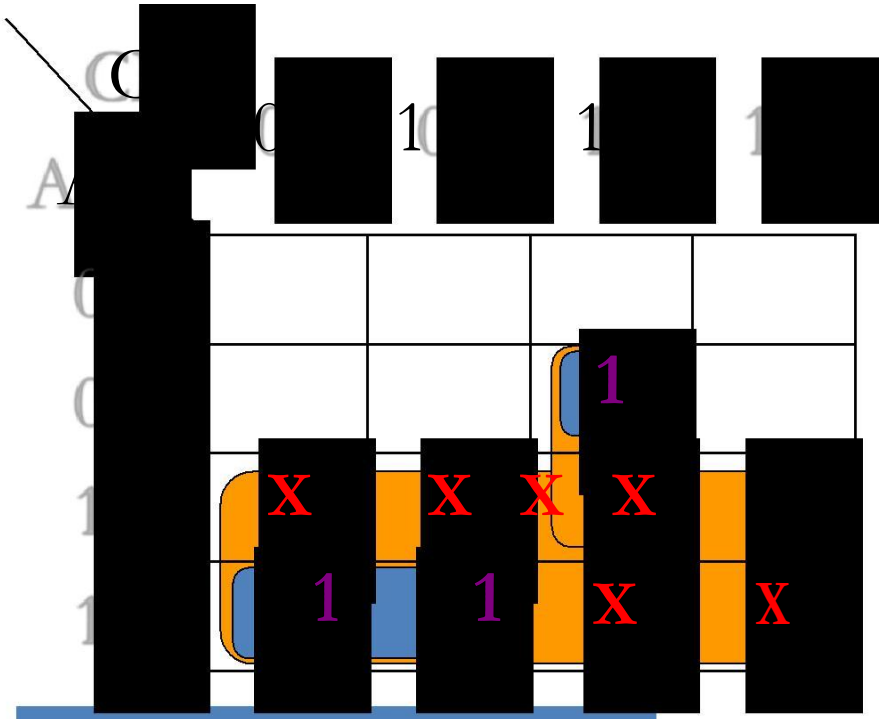


Do t Care Co ditio s

- Sometimes a situation arises in which some input variable combinations are not allowed, i.e. BCD code:
  - There are six invalid combinations: 1010, 1011, 1100, 1101, 1110, and 1111.
- Since these unallowed states will never occur in an application involving the BCD code → they can be **treated as do t are ter s ith respe t to their** effect on the output.
- **The do t are ter s a e used to ad a tage o** the K-map (how? see the next slide).

Do t Care Co ditio s

INPUTS				O/P
A	B	C	D	Y
[redacted]	[redacted]	[redacted]	[redacted]	0
0	0	0	1	0
[redacted]	[redacted]	[redacted]	[redacted]	0
0	0	1	1	0
[redacted]	[redacted]	[redacted]	[redacted]	0
0	1	0	1	0
[redacted]	[redacted]	[redacted]	[redacted]	0
0	1	1	1	1
[redacted]	[redacted]	[redacted]	[redacted]	1
1	0	0	1	1
[redacted]	[redacted]	[redacted]	[redacted]	X
1	0	1	1	X
[redacted]	[redacted]	[redacted]	[redacted]	X
1	1	0	1	X
[redacted]	[redacted]	[redacted]	[redacted]	X
1	1	1	1	X



Without do t are

$$Y = ABC + ABCD$$

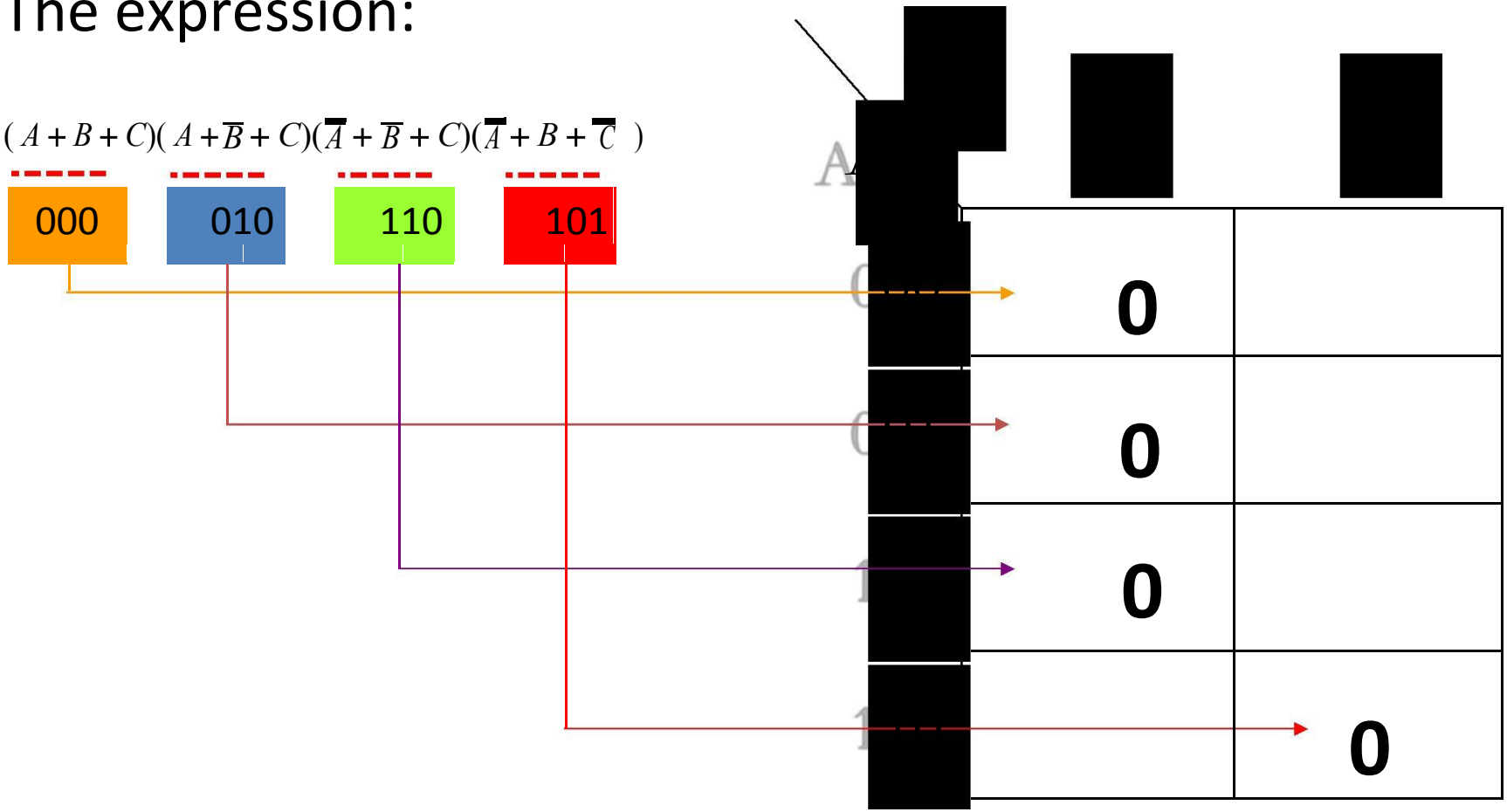
With do t are

$$Y = A + BCD$$

Mapping a Standard POS Expression (full example)

The expression:

$$(A+B+C)(A+\overline{B}+C)(\overline{A}+\overline{B}+C)(\overline{A}+B+\overline{C})$$



Combinational Circuits

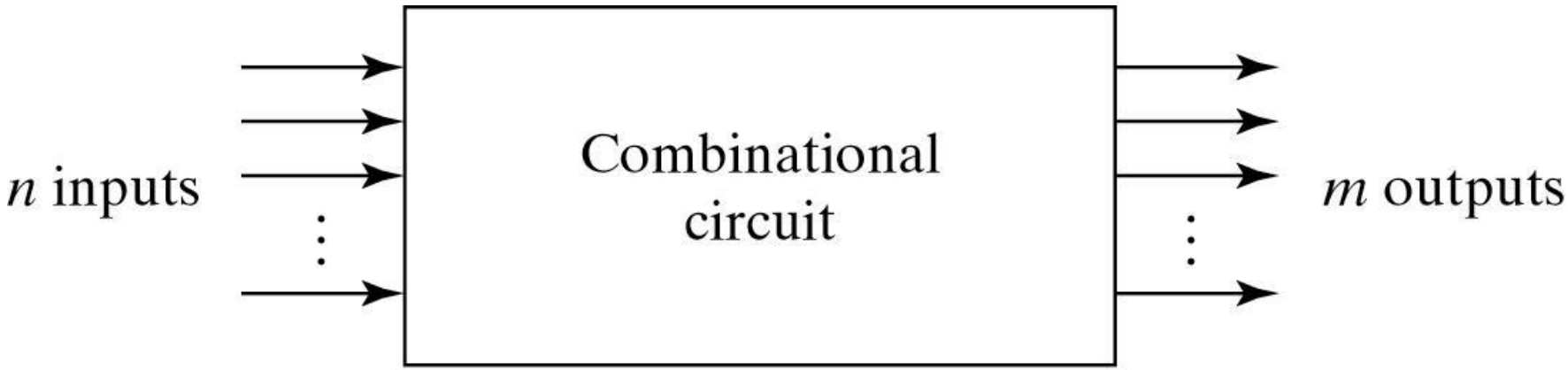


Fig. 4-1 Block Diagram of Combinational Circuit

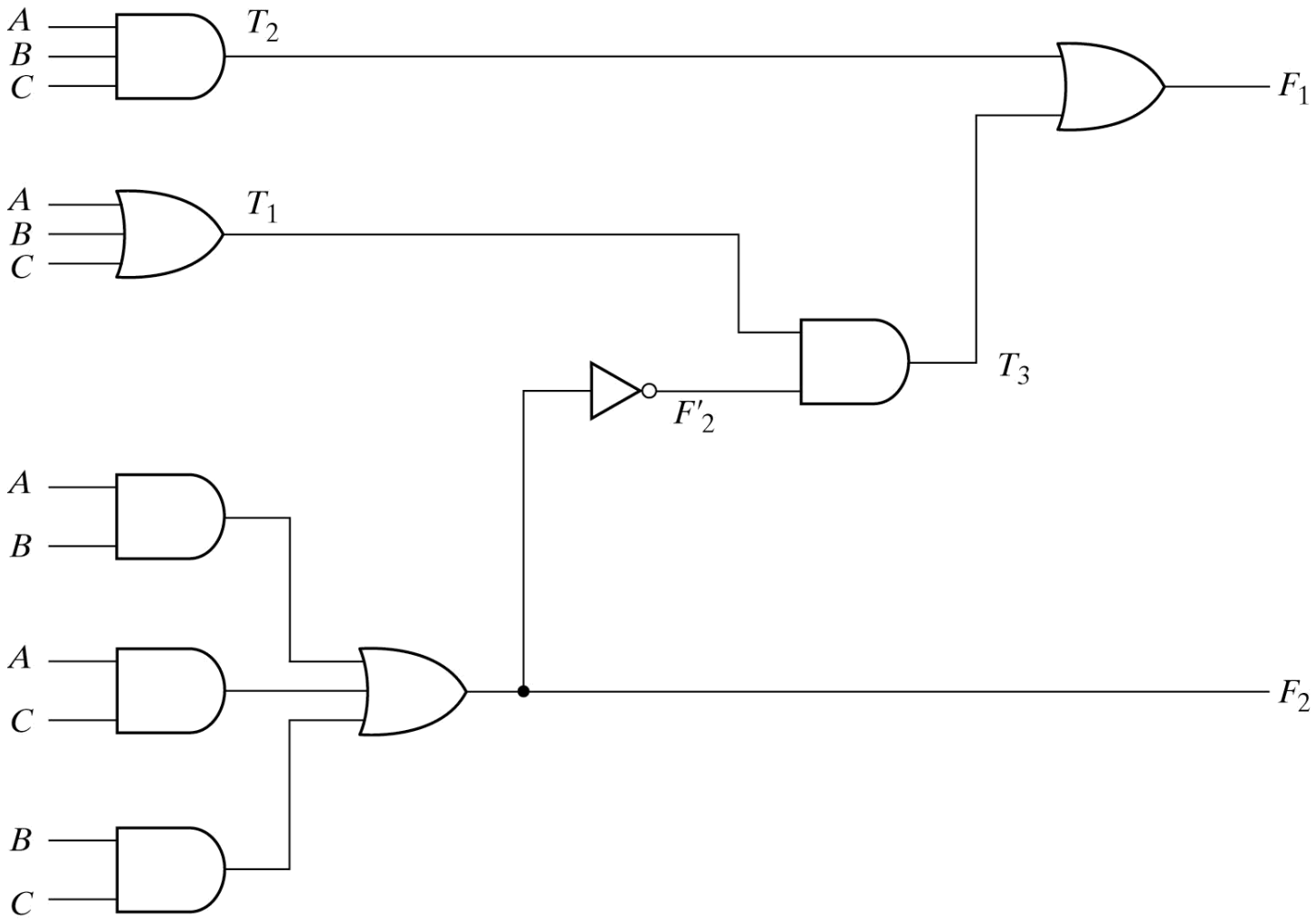


Fig. 4-2 Logic Diagram for Analysis Example

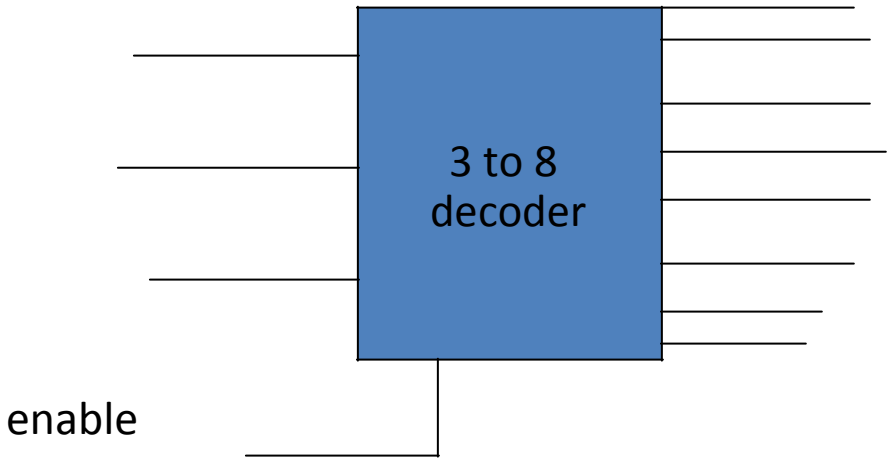
Designing Combinational Circuits

In general we have to do following steps:

1. Problem description
2. Input/output of the circuit
3. Define truth table
4. Simplification for each output
5. Draw the circuit

Decoder

- Is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines For example if the number of input is n=3 the number of output lines can be m=2^3 . It is also known as 1 of 8 because one output line is selected out of 8 available lines:



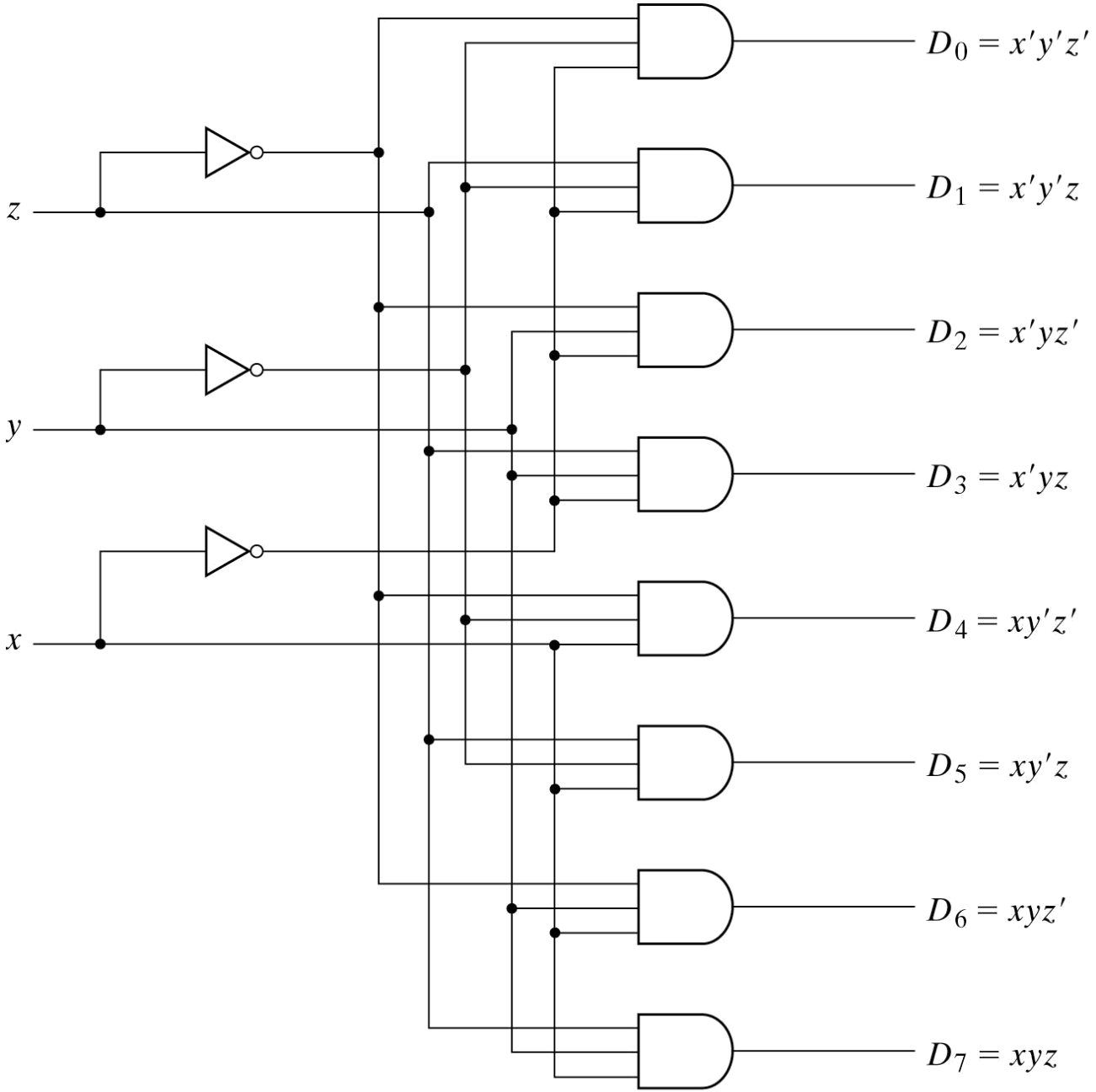


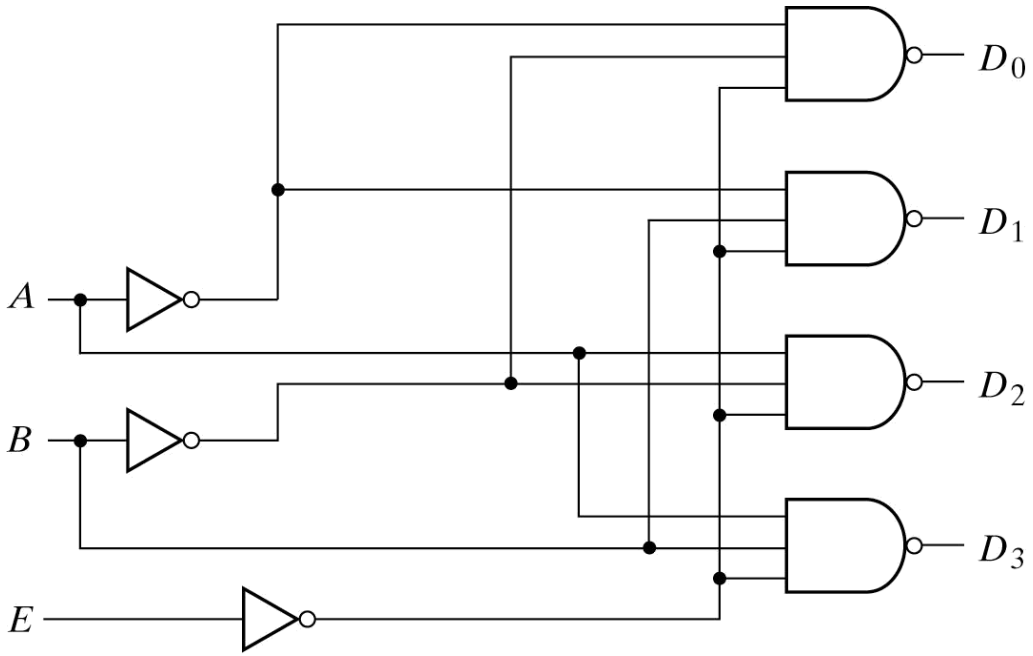
Fig. 4-18 3-to-8-Line Decoder

Decoder with Enable Line

- Decoders usually have an enable line,
- If enable=0 , decoder is off. It means all output lines are zero
- If enable=1, decoder is on and depending on input, the corresponding output line is 1, all other lines are 0
- See the truth table in next slide

Truth table for decoder

E	a2	a1	a0	D7	D6	D5	D4	D3	D2	D1	D0
0	x	x	x	0			0	0	0	0	0	0	0
1	0		0	0		0	0	0	0	0	0	0	1
1	0		0	1		0	0	0	0	0	0	1	0
1													
1													
1													
1	1		1	1		1	1	0	0	0	0	0	0



(a) Logic diagram

E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
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

(b) Truth table

Fig. 4-19 2-to-4-Line Decoder with Enable Input

Major application of Decoder

- Decoder is use to implement any combinational cicuits (f^n)

For e a ple the truth ta le for full adder is s , ,z = ∑ , , ,

a d C , ,z = ∑ , , , . The i ple e tatio ith de oder is:

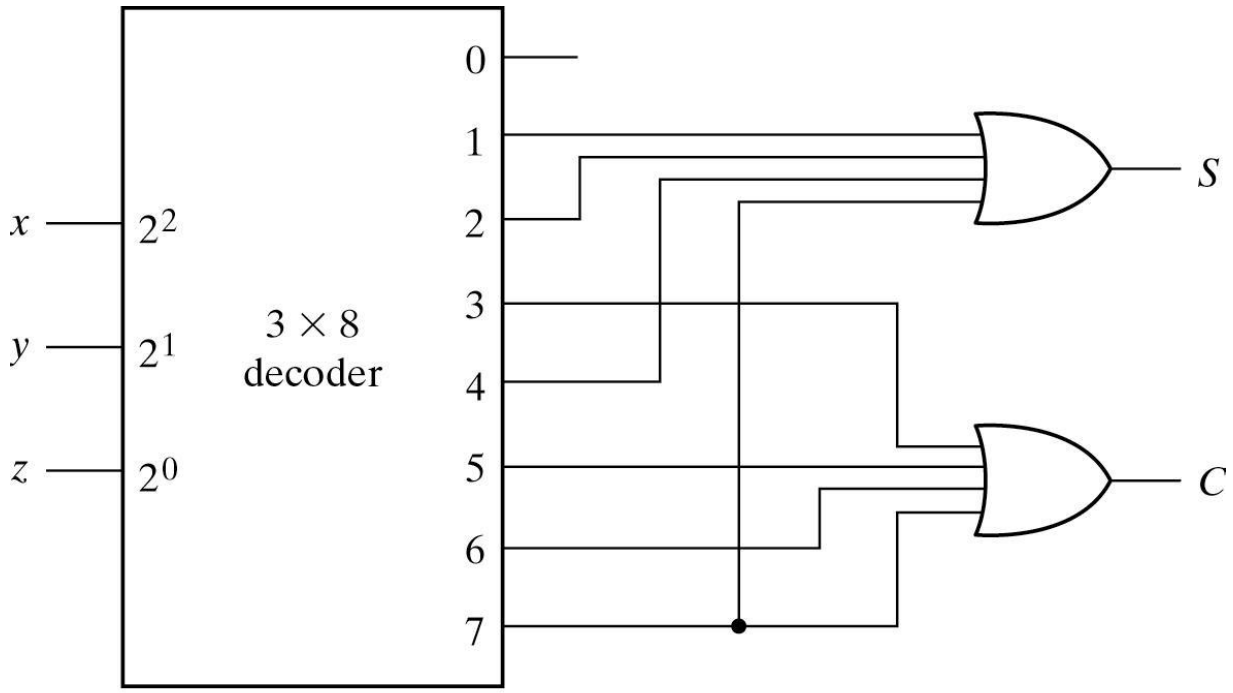
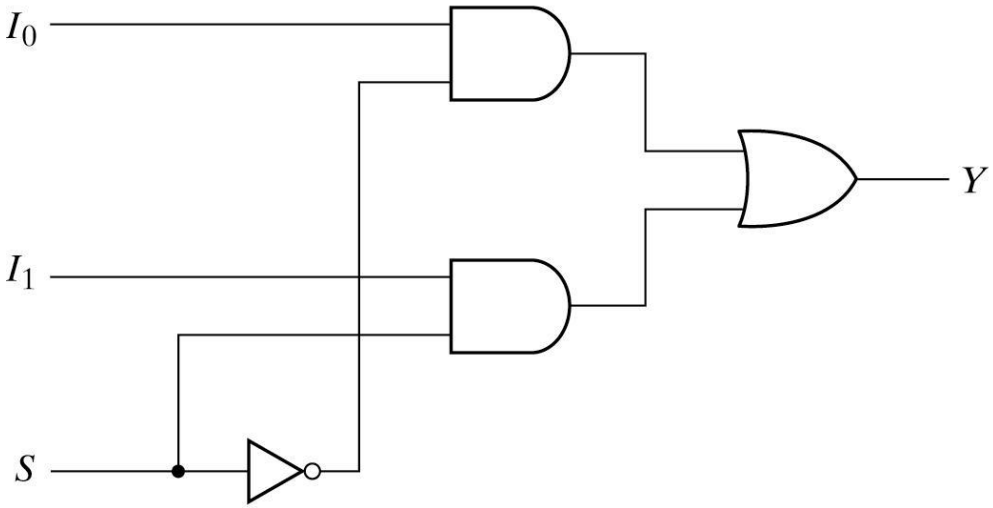


Fig. 4-21 Implementation of a Full Adder with a Decoder

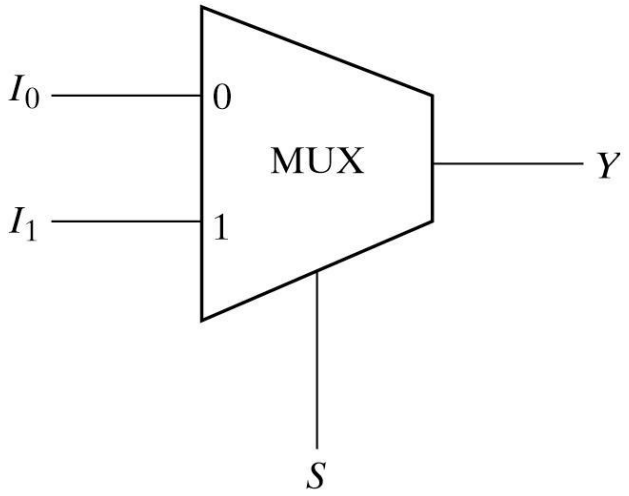
Multiplexer

- It is a combinational circuit that selects binary information from one of the input lines and directs it to a single output line
- Usually there are 2^n input lines and n selection lines whose bit combinations determine which input line is selected
- For example for 2-to-1 multiplexer if selection S is zero then I_0 has the path to output and if S is one I_1 has the path to output (see the next slide)

2-to-1 multiplexer

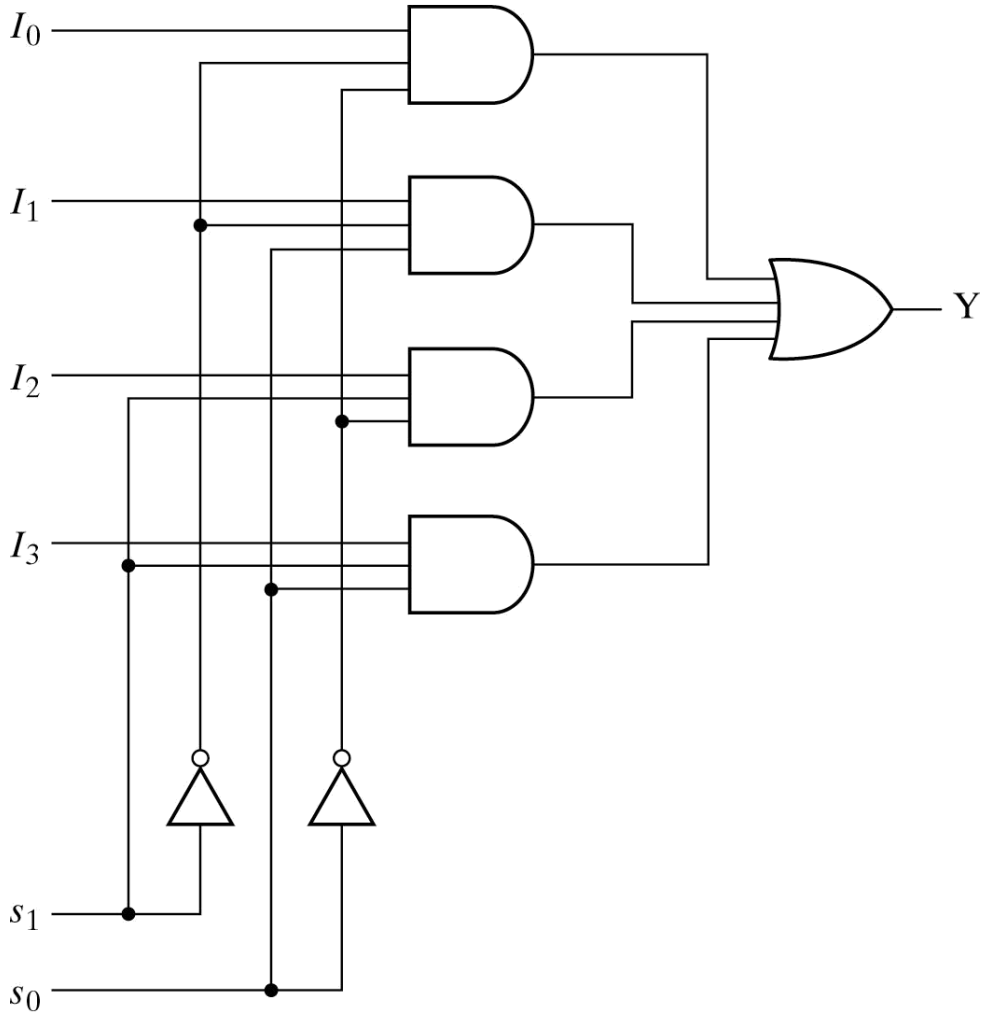


(a) Logic diagram



(b) Block diagram

Fig. 4-24 2-to-1-Line Multiplexer



(a) Logic diagram

s_1	s_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

(b) Function table

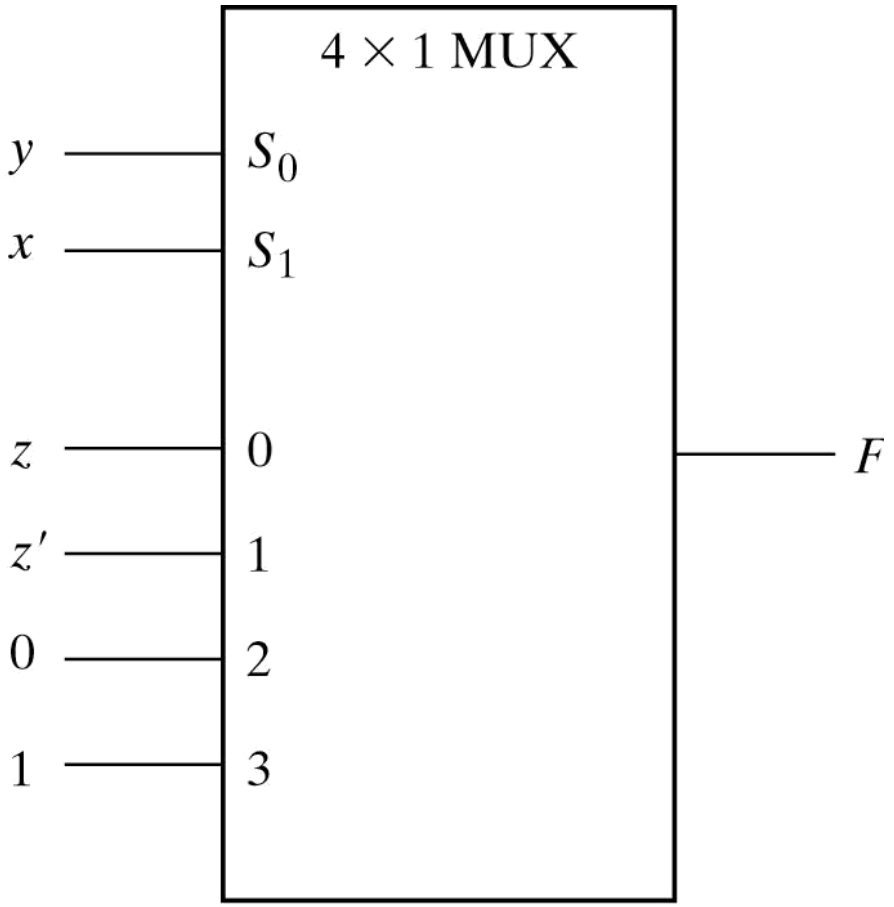
Fig. 4-25 4-to-1-Line Multiplexer

Boolean function Implementation

- Another method for implementing boolean function is using multiplexer
- For doing that assume boolean function has n variables. We have to use multiplexer with n-1 selection lines and
- 1- first n-1 variables of function is used for data input
- 2- the remaining single variable (named z)is used **for data i put. Ea h data i put a e z, z , or .** From truth table we have to find the relation of F and z to be able to design input lines. For example : **f , ,z = ∑ , , ,**

x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = z'$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	

(a) Truth table



(b) Multiplexer implementation

Fig. 4-27 Implementing a Boolean Function with a Multiplexer

F A,B,C,D = ∑ , , , , , , , ,

A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = D'$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	

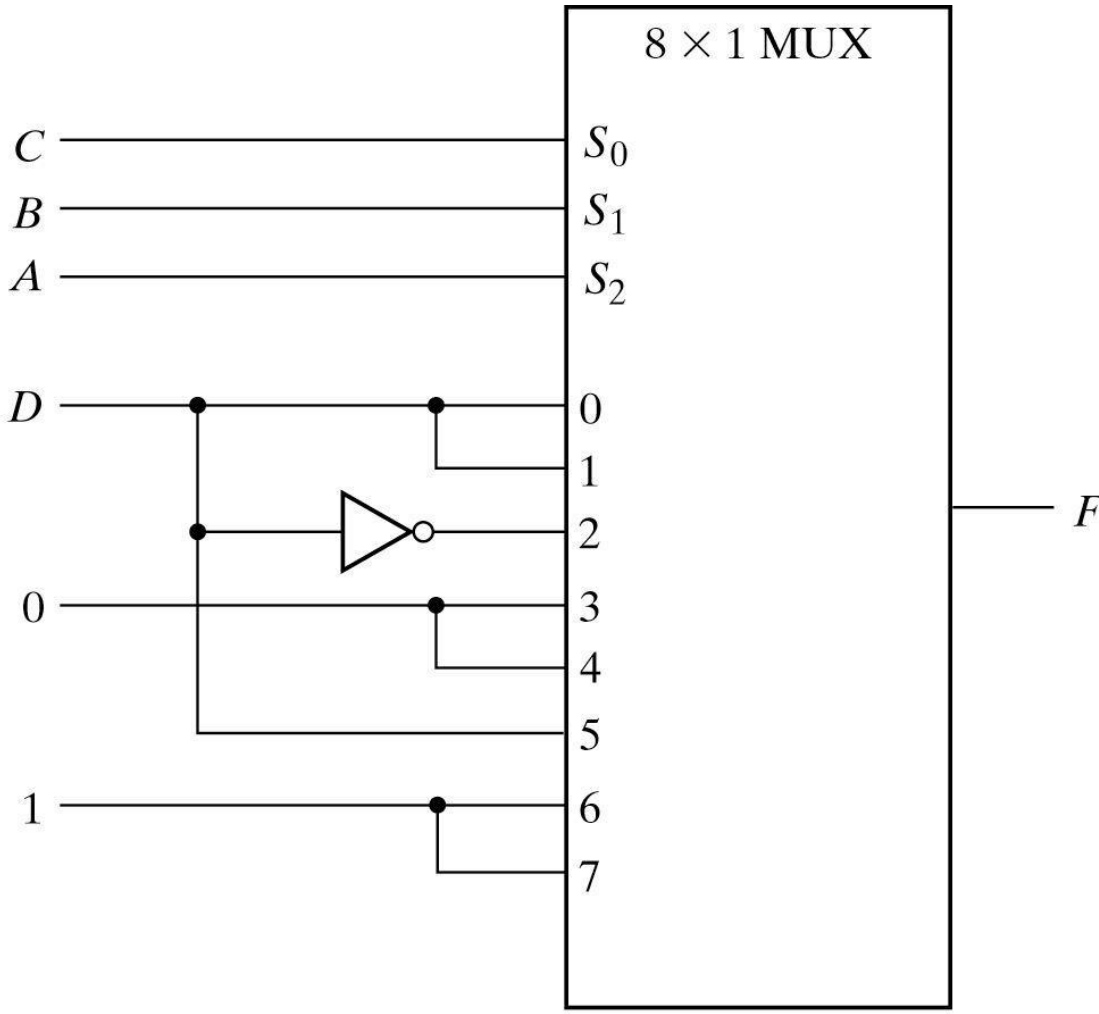
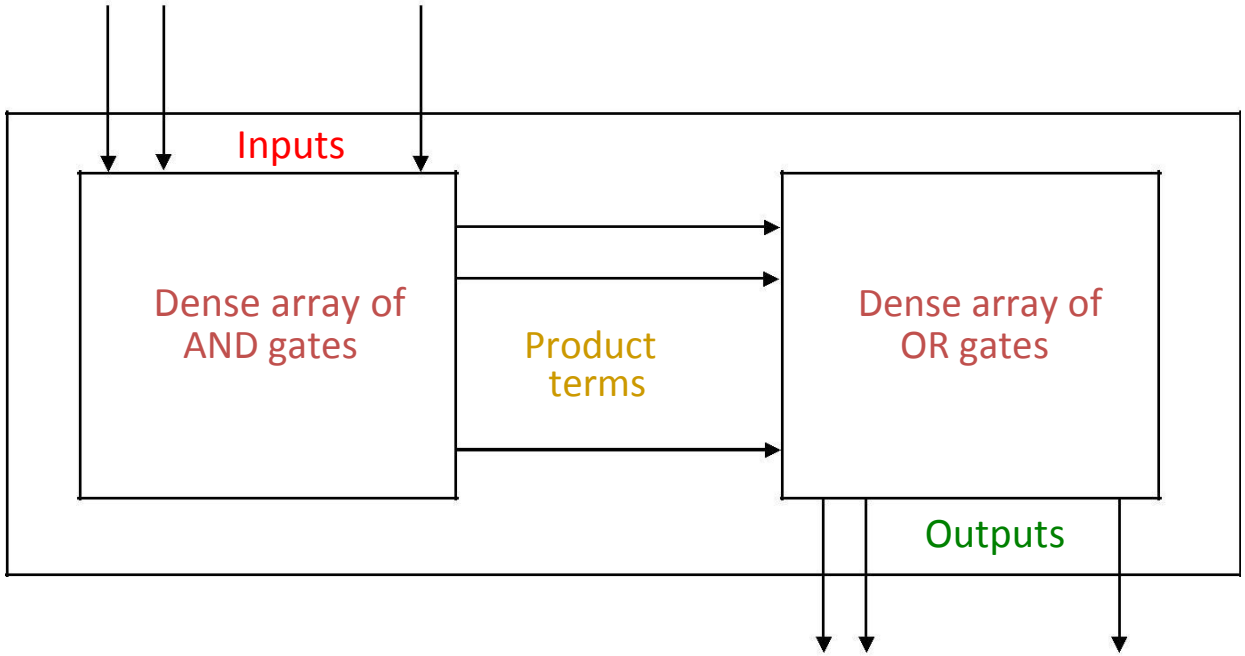


Fig. 4-28 Implementing a 4-Input Function with a Multiplexer

Prgrammable Logic Organization

- **Pre-fabricated building block of many AND/OR gates (or NOR, NAND)**
- **"Personalized" by making or breaking connections among the gates**



Programmable Array Block Diagram for Sum of Products Form

Basic Programmable Logic Organizations

- Depending on which of the AND/OR logic arrays is programmable, we have three basic organizations

ORGANIZATION	AND ARRAY	OR ARRAY
PAL	PROG.	FIXED
PROM	FIXED	PROG.
PLA	PROG.	PROG.

PLA Logic Implementation

Key to Success: Shared Product Terms

Equations

Example:

$F0 = A + B\,\overline{C}\;\overline{}$

$F1 = A\,\overline{C} + A\,B$

$F2 = B\,\overline{C} + A\,B$

$F3 = B\,\overline{C} + A$

Personality Matrix

Product term	Inputs A	B	C	Outputs F_0	F_1	F_2	F_3
A B	1	1	-	0	(1)	(1)	0
$\overline{B}$ C	-	0	1	0	0	0	1
A $\overline{C}$	1	-	0	0	1	0	0
$\overline{B}$ $\overline{C}$	-	0	0	(1)	0	(1)	0
A	1	-	-	(1)	0	0	(1)

Reuse of terms

Input Side:

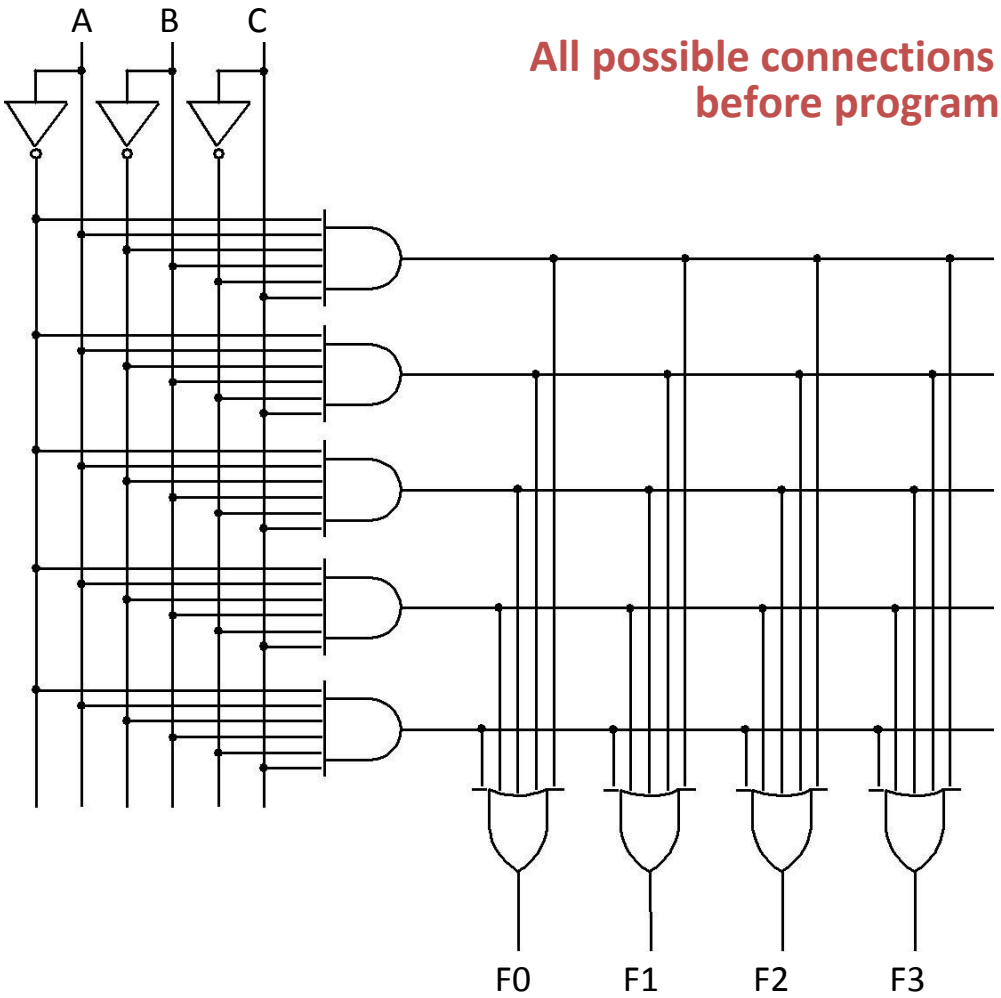
1 = asserted in term
0 = negated in term
- = does not participate

Output Side:

1 = term connected to output
0 = no connection to output

PLA Logic Implementation

Example Continued - Unprogrammed device



All possible connections are available before programming

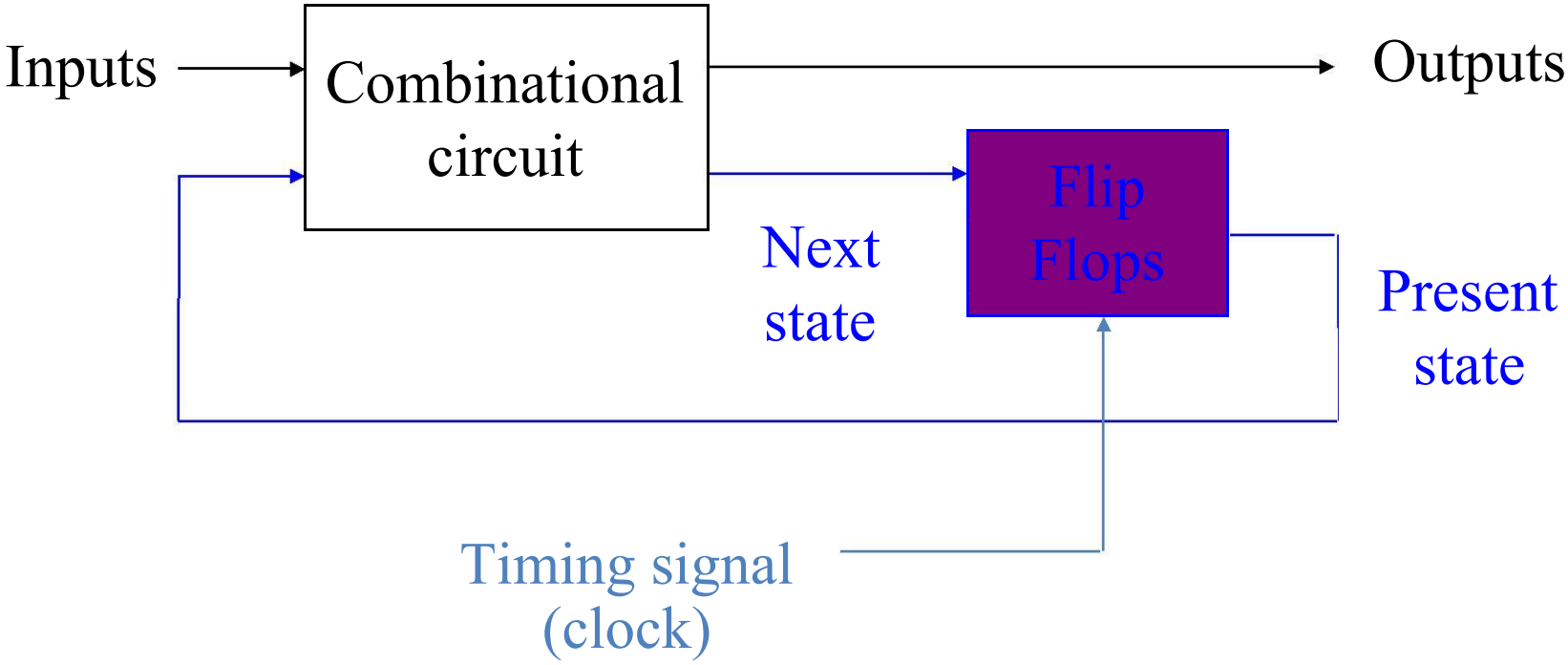
Sequential Circuits

- Circuits require memory to store intermediate data
- Sequential circuits use a periodic signal to determine when to store values.
  - A clock signal can determine storage times
  - Clock signals are periodic
- Single bit storage element is a flip flop
- A basic type of flip flop is a latch
- Latches are made from logic gates
  - NAND, NOR, AND, OR, Inverter

The story so far ...

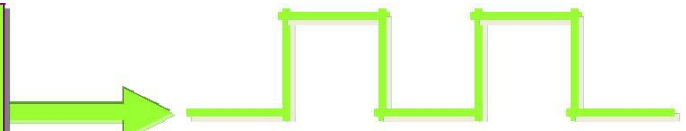
- Logical operations which respond to **combinations** of inputs to produce an output.
  - Call these **combinational logic** circuits.
- For example, can add two numbers. But:
  - No way of adding two numbers, then adding a third (a **sequential** operation);
  - No way of remembering or storing information after inputs have been removed.
- To handle this, we need **sequential logic** capable of storing intermediate (and final) results.

Sequential Circuits



Clock

a periodic external event (input)



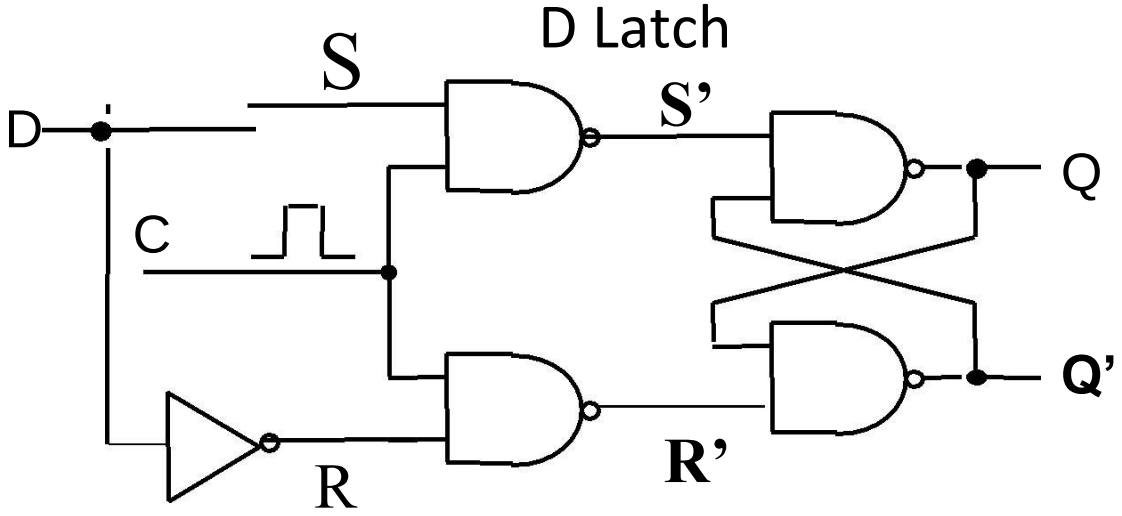
Clock

synchronizes when current state changes
happen keeps system well-behaved
makes it easier to design and build large systems

Sequential Circuits: Flip flops

Overview

- Latches respond to trigger levels on control inputs
  - Example: If G = 1, input reflected at output
- Difficult to precisely time when to store data with latches
- Flip flips store data on a rising or falling trigger edge.
  - Example: control input transitions from 0 -> 1, data input appears at output
  - Data remains stable in the flip flop until until next rising edge.
- Different types of flip flops serve different functions
- Flip flops can be defined with characteristic functions.



D	C	Q	Q'
0	1	0	1
1	1	0	X 0
Q_0		Q_0'	

S	R	C	Q	Q'	
0	0	1	Q_0		Q_0' Store
0	1	1	0	1	Reset
1	0	1	1	0	Set
1	1	1	1	1	Disallowed
X	X	0	Q_0		Q_0' Store

- When C is high, D passes from input to output (Q)

Master-Slave D Flip Flop

- Consider two latches combined together
- Only one *C* value active at a time
- Output changes on falling edge of the clock

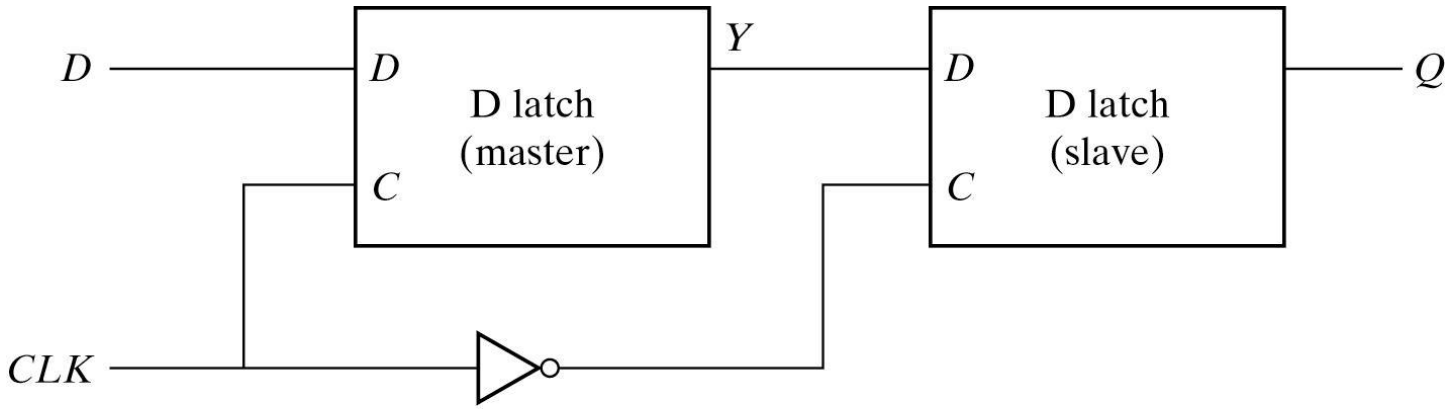
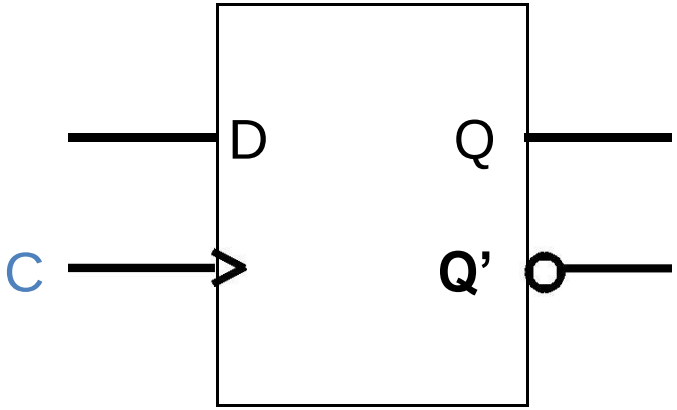


Fig. 5-9 Master-Slave *D* Flip-Flop

D Flip-Flop

- Stores a value on the positive edge of *C*
- Input changes at other times have no effect on output

Positive edge triggered

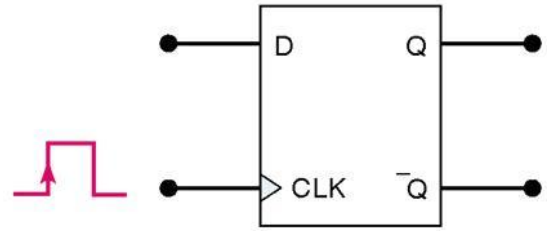


D	C	Q	Q'
0	↑	0	1
1	↑	1	0
X	0	Q_0	Q_0'

D gets latched to Q on the rising edge of the clock.

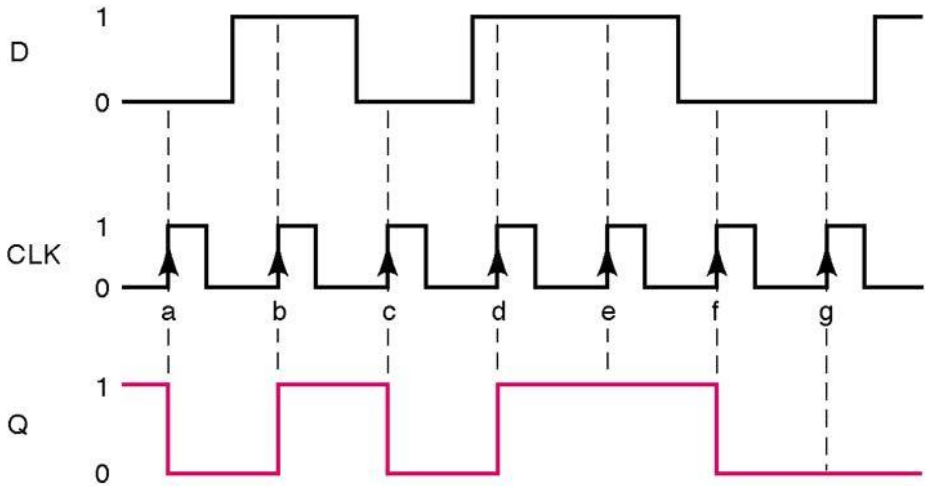
Clocked D Flip-Flop

- Stores a value on the positive edge of *C*
- Input changes at other times have no effect on output



D	CLK	Q
0	↑	0
1	↑	1

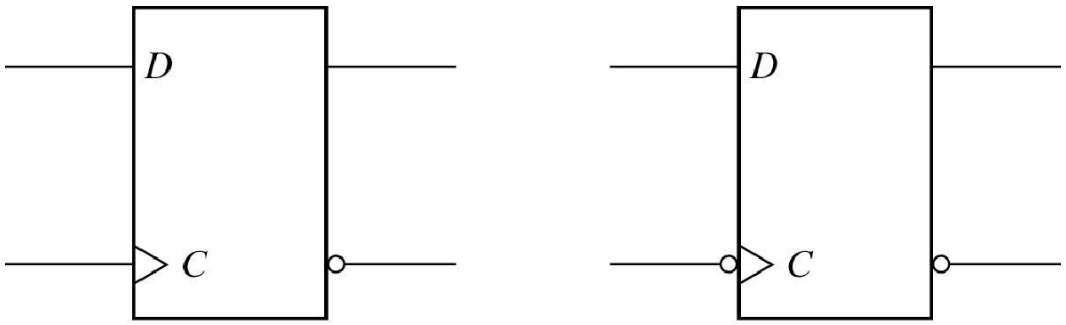
(a)



(b)

Positive and Negative Edge D Flip-Flop

- D flops can be triggered on positive or negative edge
- Bubble before *Clock (C)* input indicates negative edge trigger

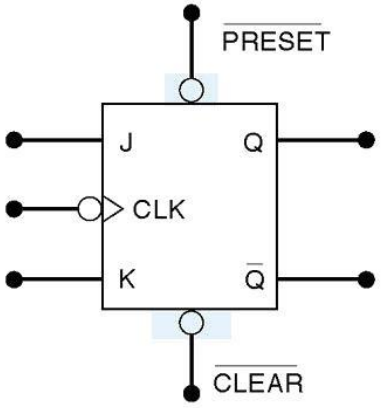


(a) Positive-edge (a) Negative-edge

Fig. 5-11 Graphic Symbol for Edge-Triggered *D* Flip-Flop

Lo-Hi edge **Hi-Lo edge**

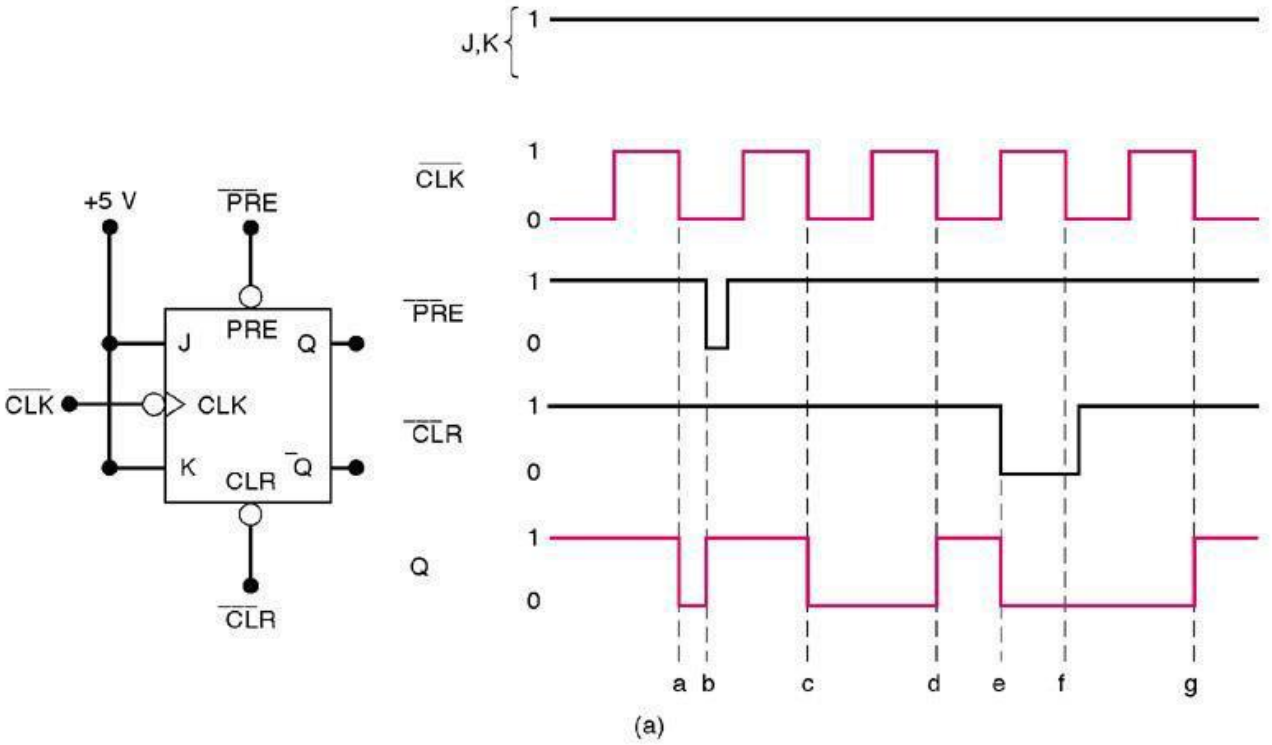
- J, K are **synchronous inputs**
  - o Effects on the output are synchronized with the *CLK* input.
- **Asynchronous inputs** operate independently of the synchronous inputs and clock
  - o Set the FF to 1/0 states *at any time.*



PRESET	CLEAR	FF response
1	1	Clocked operation*
0	1	Q = 1 (regardless of CLK)
1	0	Q = 0 (regardless of CLK)
0	0	Not used

*Q will respond to J, K, and CLK

Asynchronous Inputs

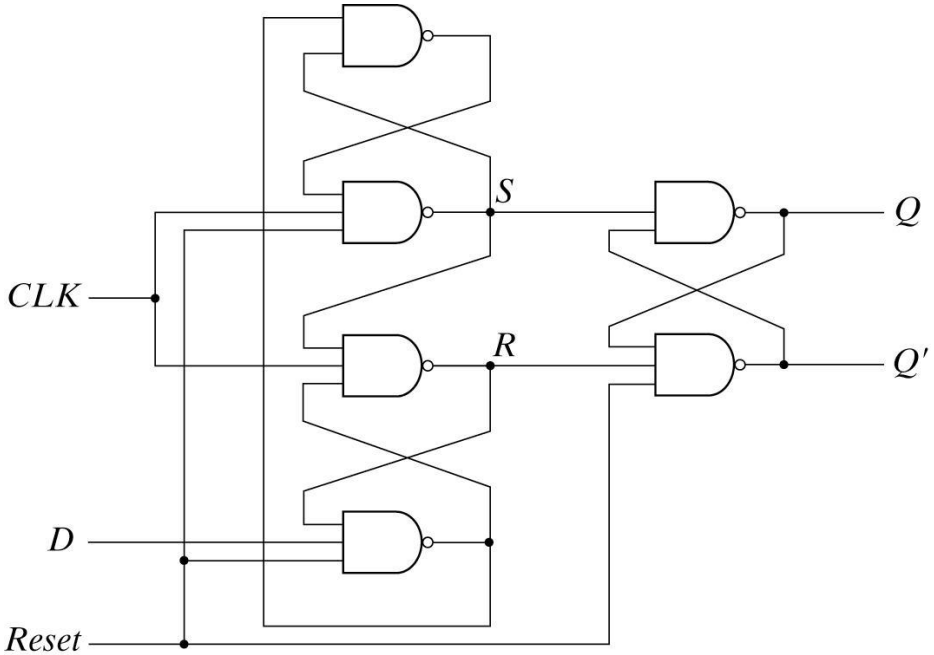


(a)

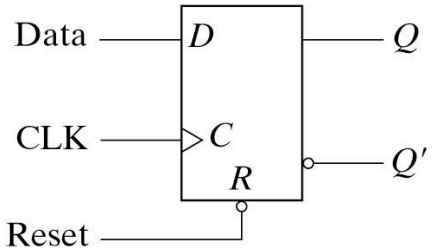
Point	Operation
a	Synchronous toggle on NGT of $\overline{CLK}$
b	Asynchronous set on $\overline{PRE}$ = 0
c	Synchronous toggle
d	Synchronous toggle
e	Asynchronous clear on $\overline{CLR}$ = 0
f	$\overline{CLR}$ over-rides the NGT of $\overline{CLK}$
g	Synchronous toggle

(b)

Asynchronous Inputs



(a) Circuit diagram



(b) Graphic symbol

R	C	D	Q	Q′
0	X	X	0	1
1	↑	0	0	1
1	↑	1	1	0

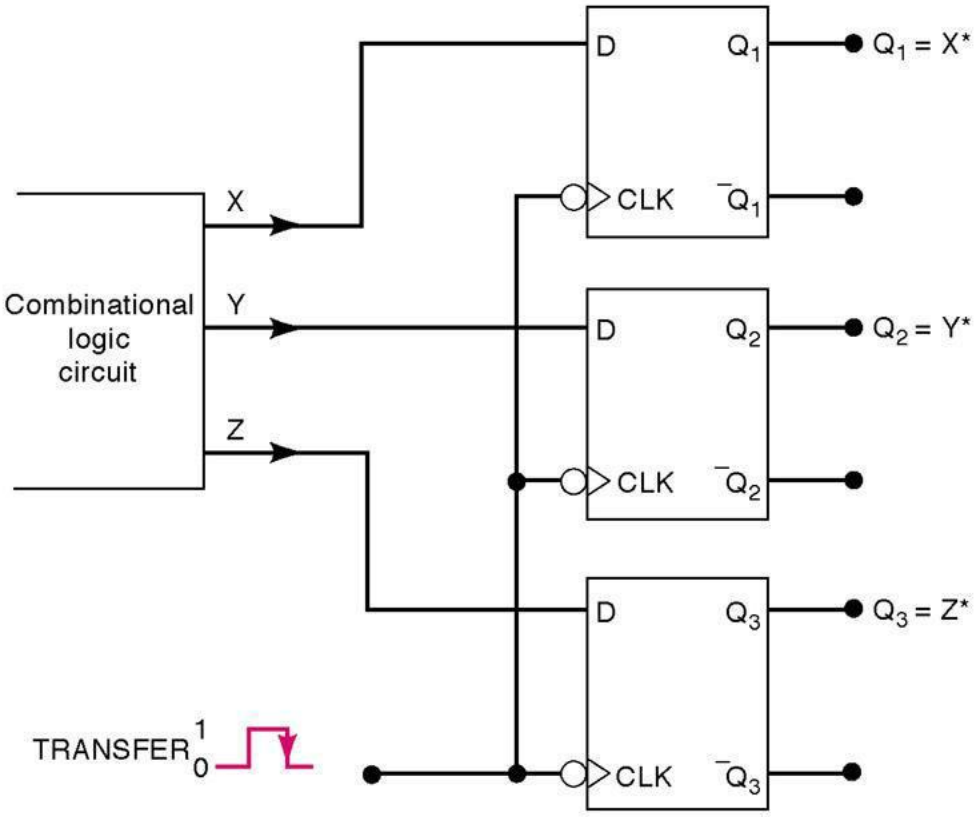
(b) Function table

Fig. 5-14 *D* Flip-Flop with Asynchronous Reset

- Note reset signal (R) for D flip flop
- If R = 0, the output Q is cleared
- This event can occur at any time, regardless of the value of the CLK

Parallel Data Transfer

- Flip flops store outputs from combinational logic
- Multiple flops can store a collection of data



*After occurrence of NGT

Summary

- Flip flops are powerful storage elements
  - They can be constructed from gates and latches!
- D flip flop is simplest and most widely used
- Asynchronous inputs allow for clearing and presetting the flip flop output
- Multiple flops allow for data storage
  - The basis of computer memory!
- Combine storage and logic to make a computation circuit
- Next time: Analyzing sequential circuits.

Counters

- Counters are important components in computers
  - The increment or decrement by one in response to input
- Two main types of counters
  - Ripple (asynchronous) counters
  - Synchronous counters
- Ripple counters
  - Flip flop output serves as a source for triggering other flip flops
- Synchronous counters
  - All flip flops triggered by a clock signal
- Synchronous counters are more widely used in industry.

Counters

- Counter: A register that goes through a prescribed series of states
- Binary counter
  - Counter that follows a binary sequence
  - N bit binary counter counts in binary from n to 2^{n-1}
- Ripple counters triggered by initial Count signal
- Applications:
  - Watches
  - Clocks
  - Alarms
  - Web browser refresh

Binary Ripple Counter

- Reset signal sets all outputs to 0
- Count signal toggles output of low-order flip flop
- Low-order flip flop provides trigger for adjacent flip flop
- Not all flops change value simultaneously
  - Lower-order flops change first
- Focus on D flip flop implementation

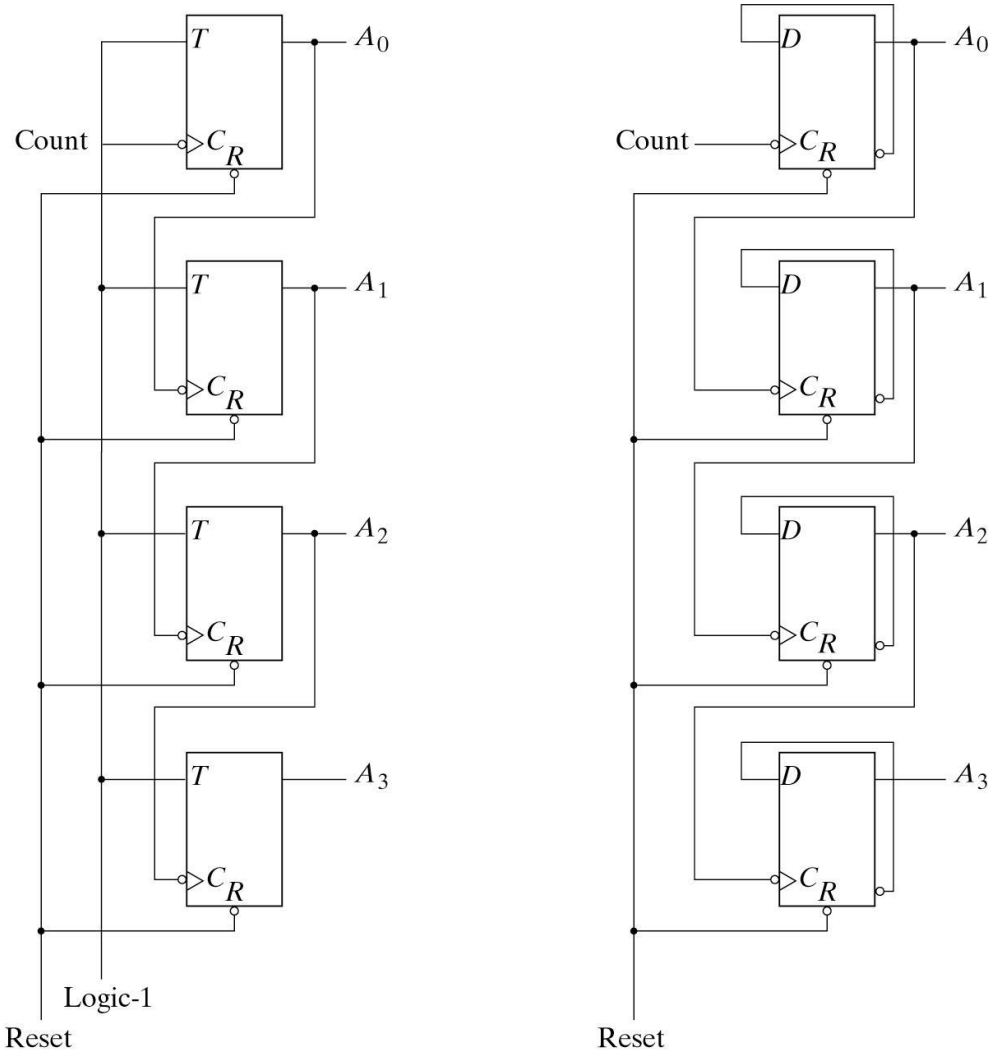


Fig. 6-8 4-Bit Binary Ripple Counter

Asynchronous Counters

- Each FF output drives the CLK input of the next FF.
- FFs do not change states in exact synchronism with the applied clock pulses.
- *There is delay between the responses of successive FFs.*
- *Ripple counter* due to the way the FFs respond one after another in a kind of rippling effect.

A_3	A_2	A_1	A_0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
1	0	0	0
1	0	0	1

Synchronous counters

- Synchronous(parallel) counters
  - All of the FFs are triggered simultaneously by the clock input pulses.
  - All FFs change at same time
- Remember
  - If J=K=0, flop maintains value
  - If J=K=1, flop toggles
- Most counters are synchronous in computer systems.
- Can also be made from D flops
- Value increments on positive edge

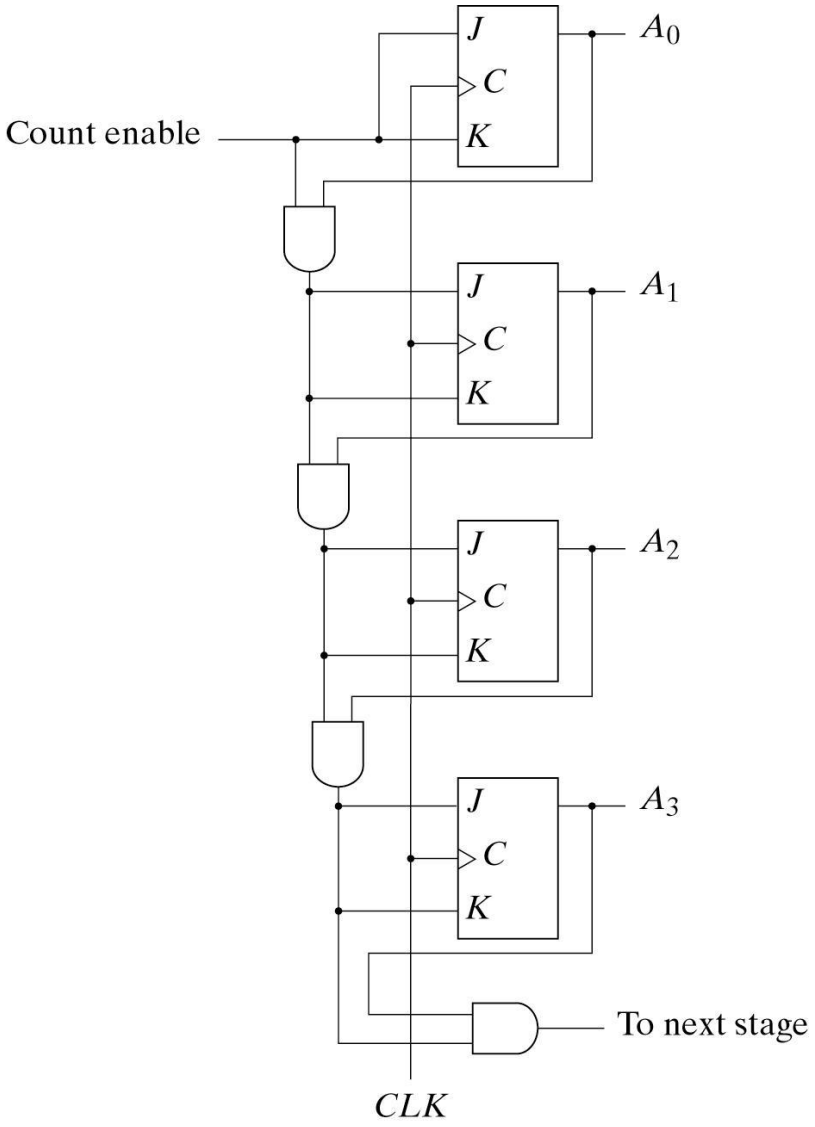
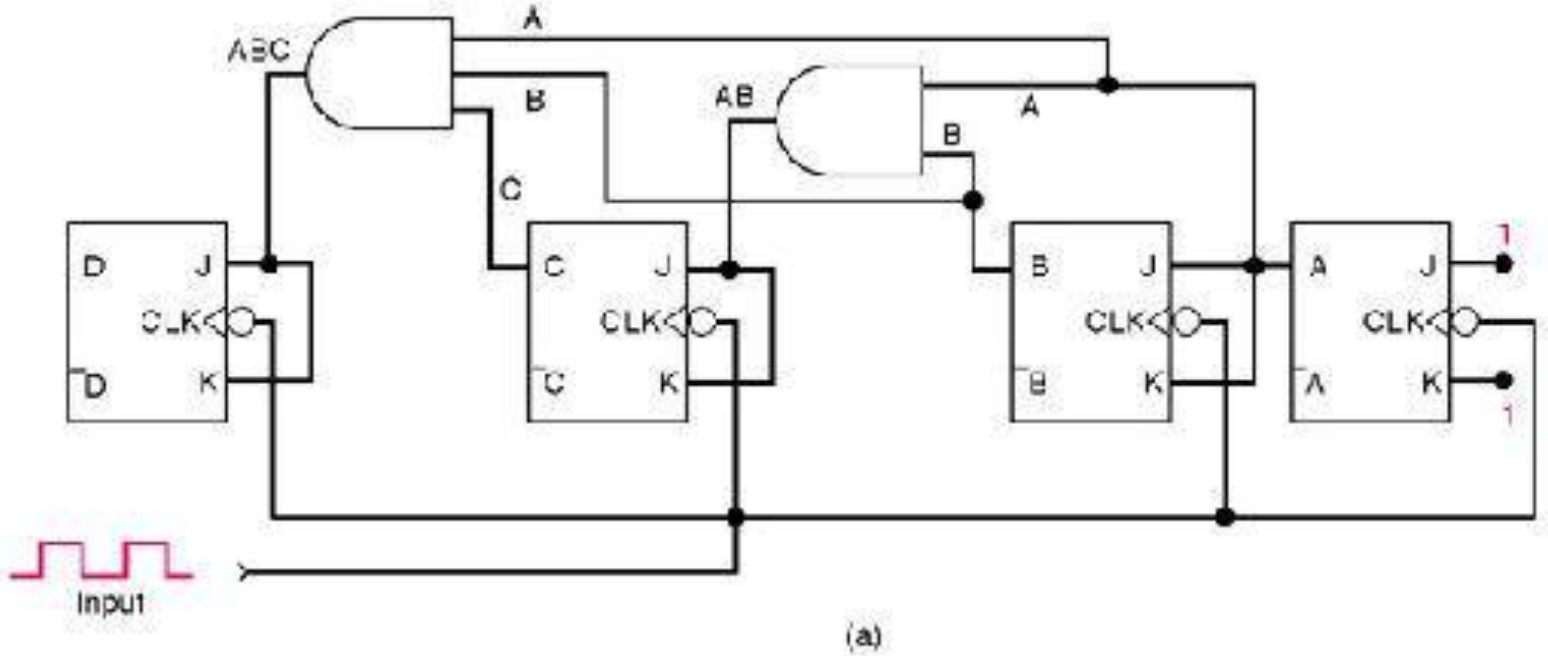


Fig. 6-12 4-Bit Synchronous Binary Counter

Synchronous counters

- Synchronous counters
  - Same counter as previous slide except Count enable replaced by J=K=1
  - Note that clock signal is a square wave
  - Clock fans out to all clock inputs



Circuit operation

Count	D	C	B	A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
0	0	0	0	0
.	.	.	.	.
.	.	.	.	.
.	.	etc.	.	.

(b)

- Count value increments on each negative edge
- Note that low-order bit (A) toggles on each clock cycle

Registers

- Register
  - Consists of N Flip-Flops
  - Stores N bits
  - Common clock used for all Flip-Flops
- Shift Register
  - A register that provides the ability to shift its contents (either left or right).
  - Must use Flip-Flops
    - Either edge-triggered or master-slave
  - Cannot use Level-sensitive Gated Latches

Overview of Shift Registers

- **A shift register is a sequential logic device made up of flip-flops that allows parallel or serial loading and serial or parallel outputs as well as shifting bit by bit.**
- **Common tasks of shift registers:**
  - **Serial/parallel data conversion**
  - **Time delay**
  - **Ring counter**
  - **Twisted-ring counter or Johnson counter**
  - **Memory device**

Characteristics of Shift Registers

- **Number of bits (4-bit, 8-bit, etc.)**
- **Loading**
  - **Serial**
  - **Parallel (asynchronous or synchronous)**
- **Common modes of operation.**
  - **Parallel load**
  - **Shift right-serial load**
  - **Shift left-serial load**
  - **Hold**
  - **Clear**
- **Recirculating or non-recirculating**

Serial/Parallel Data Conversion

Shift registers can be used to convert from serial-to-parallel or the reverse from parallel-to-serial.