Lecture -1
Logic Gates & Boolean Algebra

Prepared By: Asif Mahfuz

Logic Gates

- Logic Gates are the basic building blocks of any digital system.
- A logic gate can have one or more than one input but only one output.
- The relationship between the input/s and the output is based on a certain logic.
- The gates are named based on the logic.

Basic Logic Gates

- NOT gate
- AND gate
- OR gate

Universal Logic Gates

- NAND gate
- NOR gate

Exclusive Logic Gates

- XOR gate
- XNOR gate

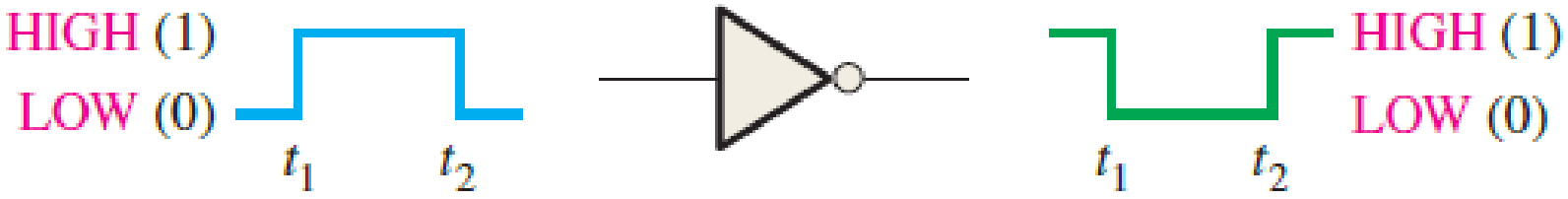
NOT Gate (Inverter)

- The output an inverter (NOT gate) is the opposite of its input.

Inverter truth table.

Input	Output
LOW (0)	HIGH (1)
HIGH (1)	LOW (0)

$X = \overline{X}$

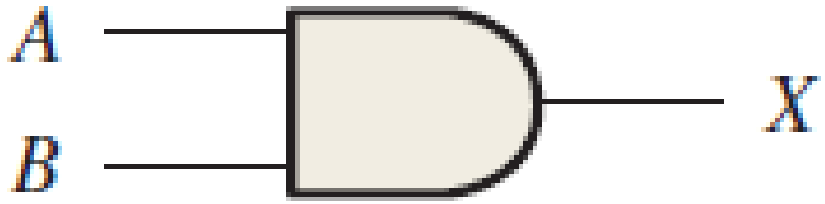


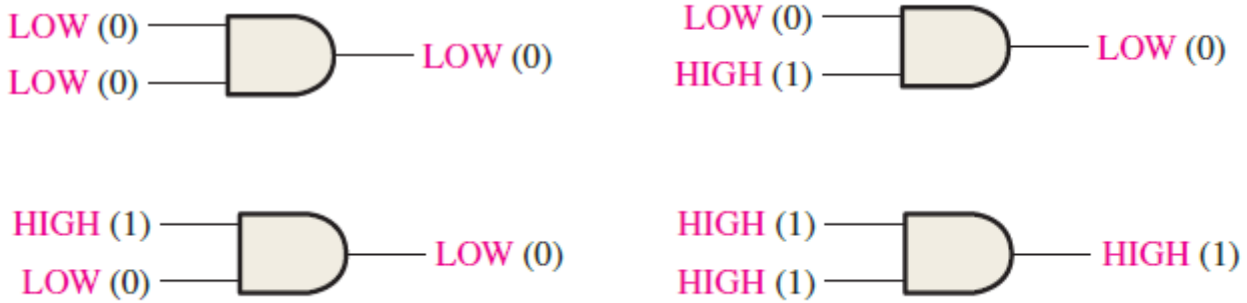
Input pulse

Output pulse

AND Gate

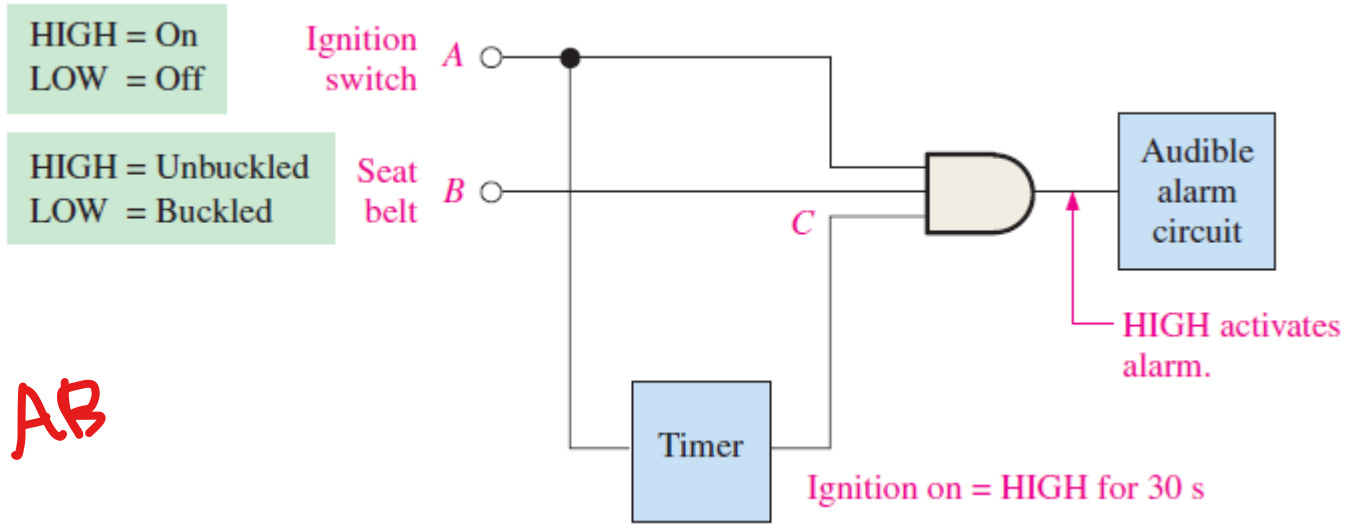
- The output an AND gate is HIGH only when both the inputs are HIGH.





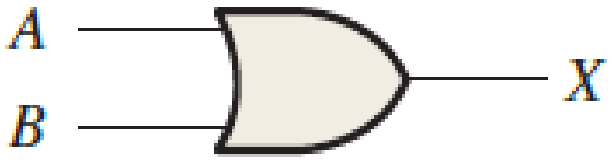
Inputs		Output
A	*B*	*X*
0	0	0
0	1	0
1	0	0
1	1	1

$X = AB$



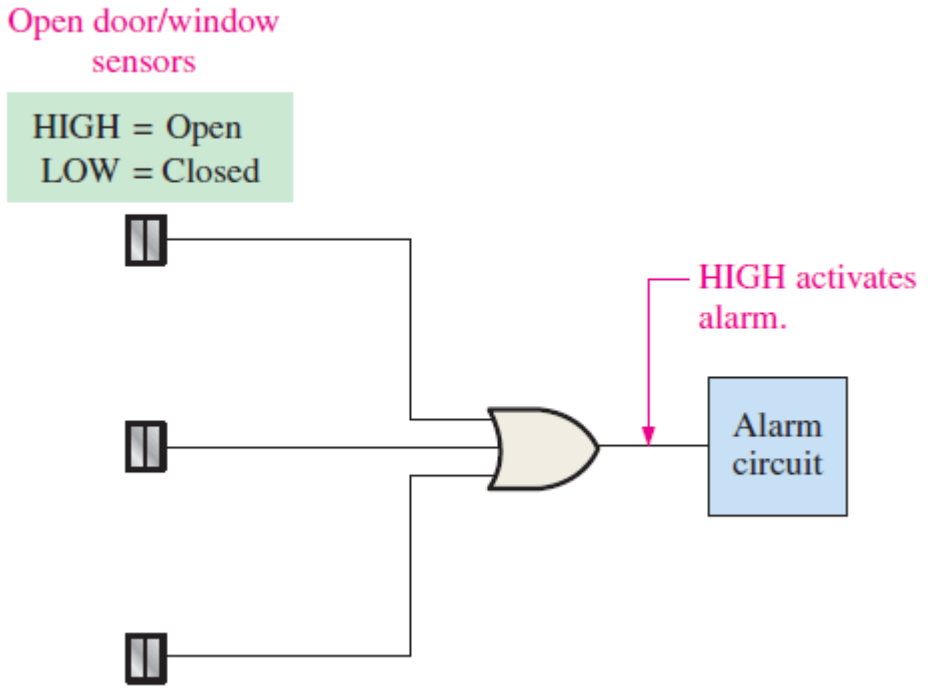
OR Gate

- The output an OR gate is HIGH when anyone or both the inputs are HIGH.



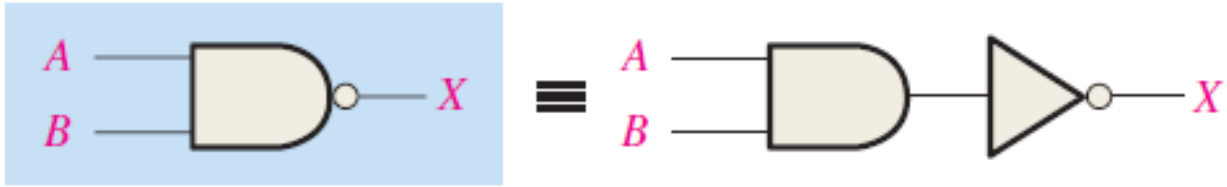
Inputs		Output
A	*B*	*X*
0	0	0
0	1	1
1	0	1
1	1	1

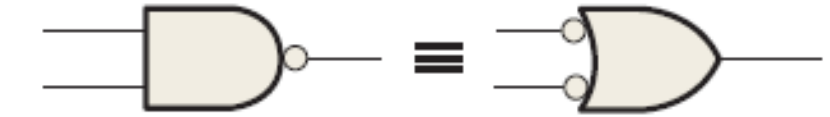
$x = A + B$



NAND Gate

- The output of a NAND gate is HIGH whenever one or more inputs are LOW.

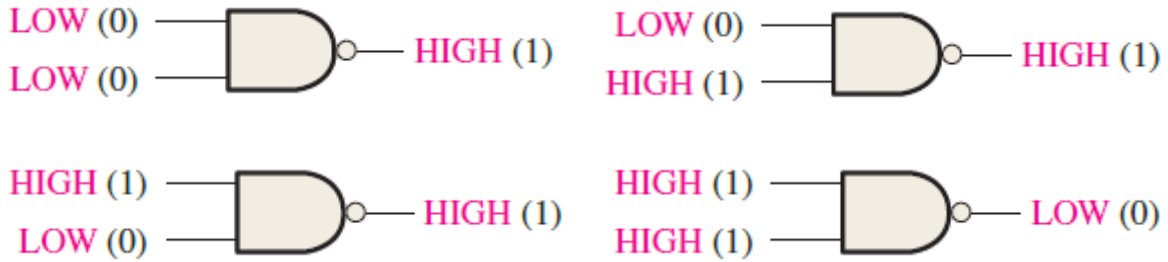


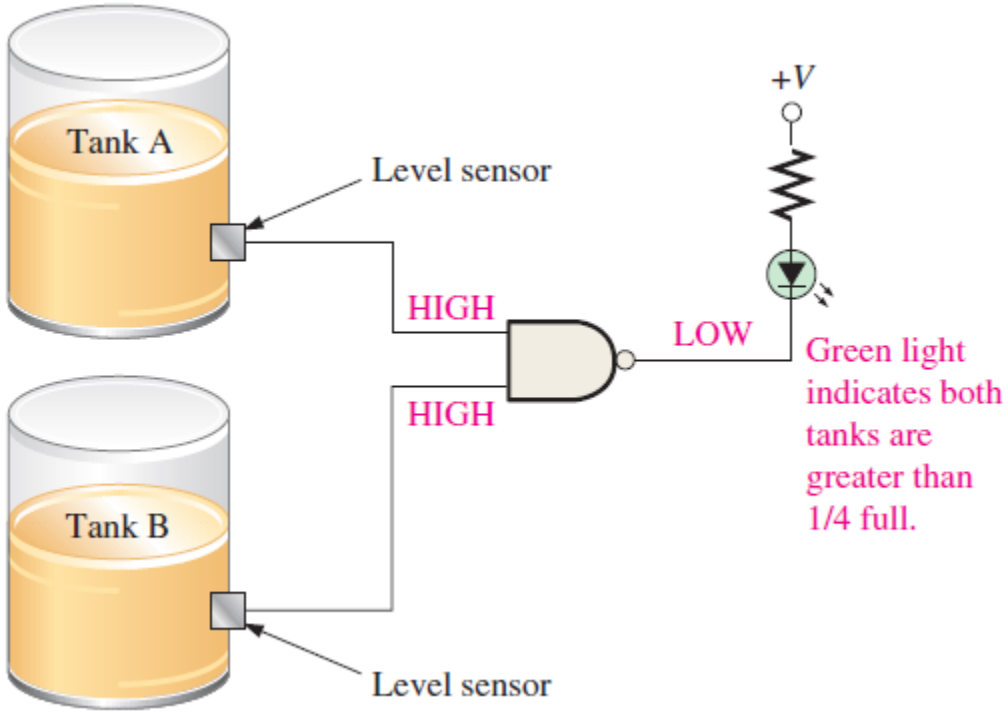


NAND ≡ Negative-OR

Inputs		Output
A	*B*	*X*
0	0	1
0	1	1
1	0	1
1	1	0

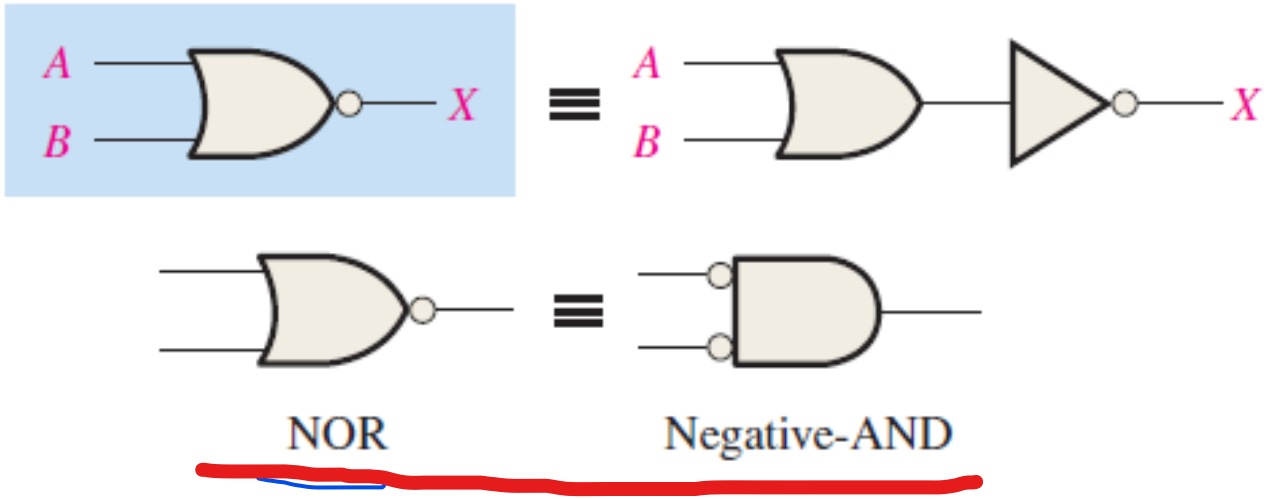
$X = \overline{AB}$

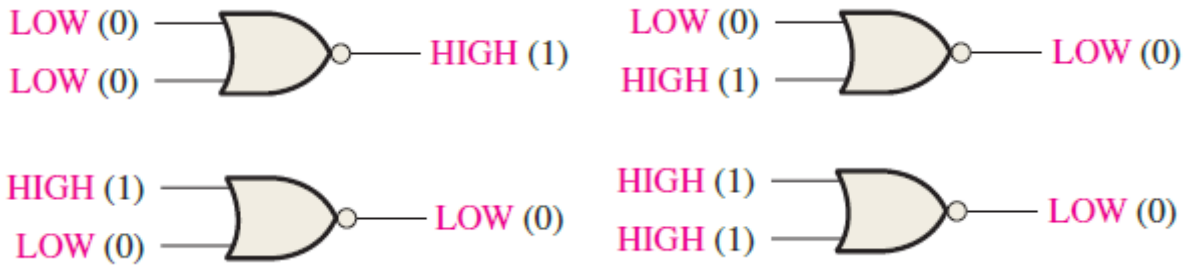




NOR Gate

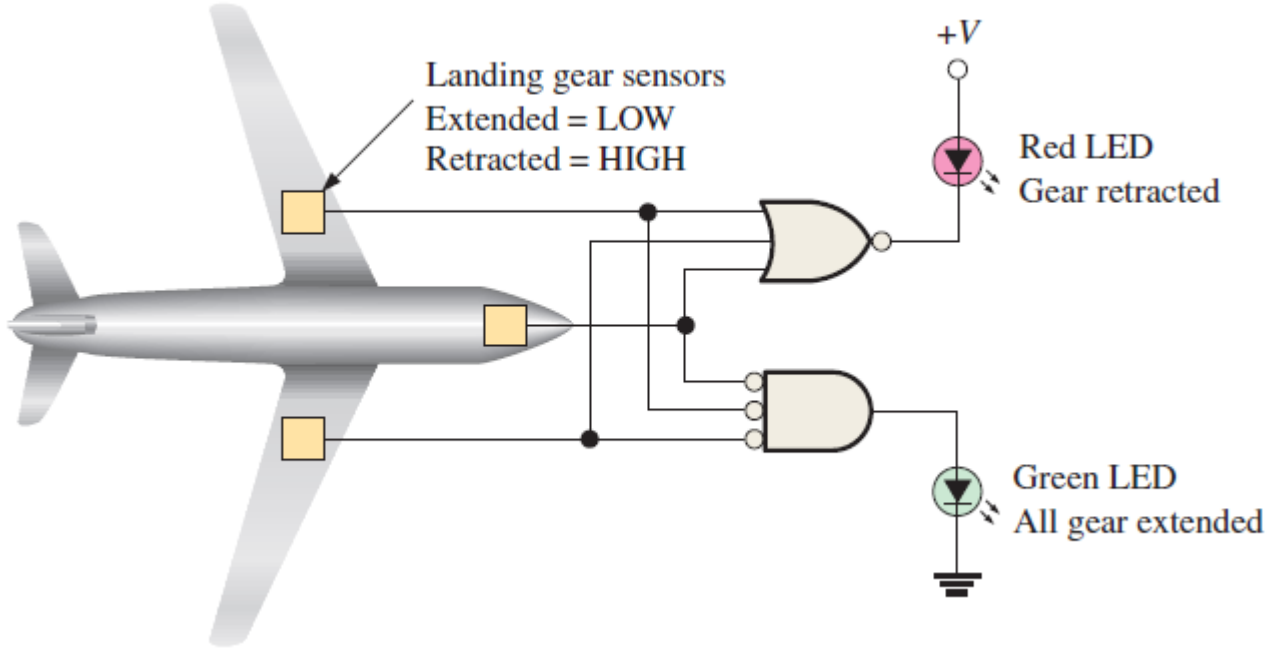
- The output of a NOR gate is LOW whenever one or more inputs are HIGH.





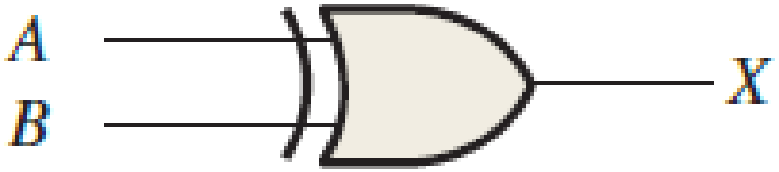
Inputs		Output
A	*B*	*X*
0	0	1
0	1	0
1	0	0
1	1	0

$X = \overline{A + B}$



XOR Gate

- The output of a XOR gate is HIGH whenever the two inputs are different.

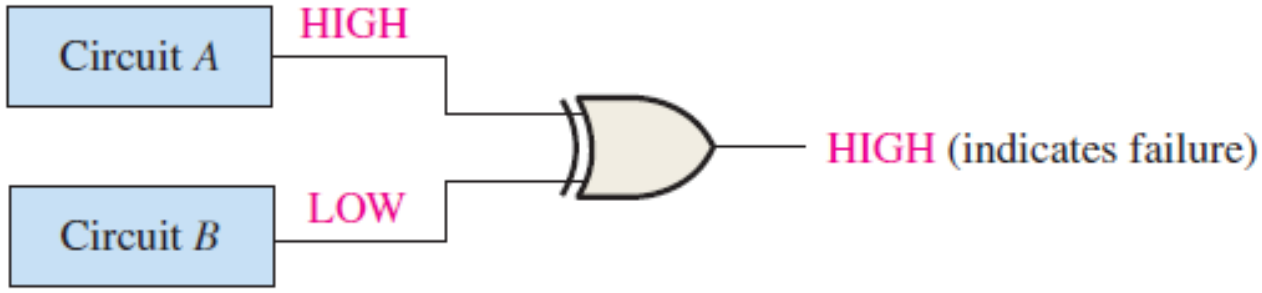


Truth table for an exclusive-OR gate.

Inputs		Output
A	*B*	*X*
0	0	0
0	1	1
1	0	1
1	1	0

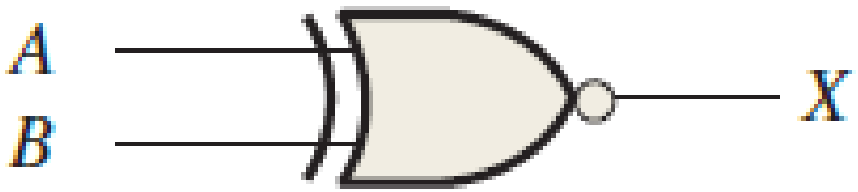
$X = A \oplus B$

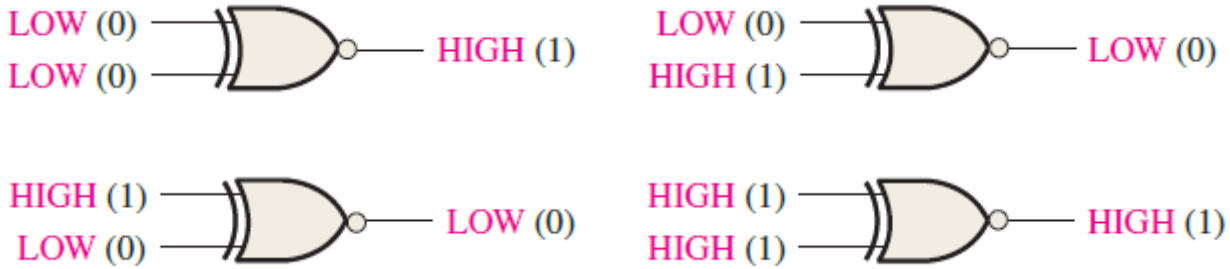
Two circuits are supposed to work parallelly in a process. If any of the circuit fails an indicator is activated.



XNOR Gate

- The output of a XNOR gate is gate is HIGH whenever the two inputs are same.





Inputs		Output
A	*B*	*X*
0	0	1
0	1	0
1	0	0
1	1	1

$$X = \overline{A \oplus B}$$

Boolean Algebra

- Boolean Variable: These are variables which can either take the value 1 or 0.

- Boolean Logic Expression: A Boolean logic expression is an expression constituted of only Boolean variables. The output of a Boolean logic expression is a Boolean value i.e. either True/False.

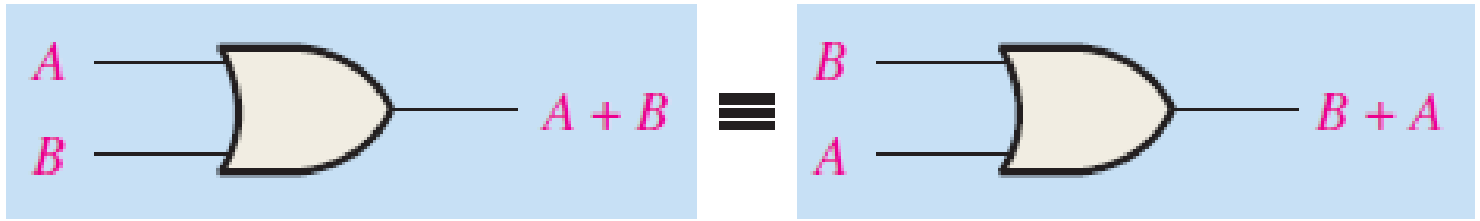
$$AB + A(B + C) + B(B + C)$$

- Boolean Algebra: It is the mathematics of digital logic. Usually Boolean algebra is used to simplify Boolean expressions or Boolean Function.

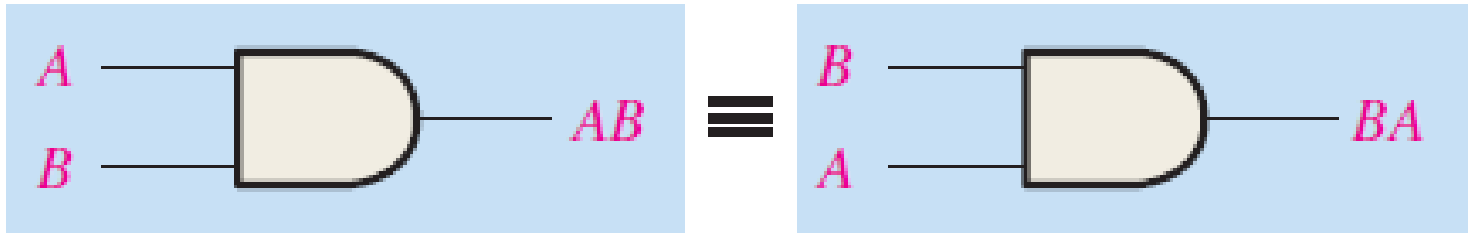
Laws of Boolean Algebra

Commutative Law:

- The commutative law for addition can be written as **A+B=B+A**



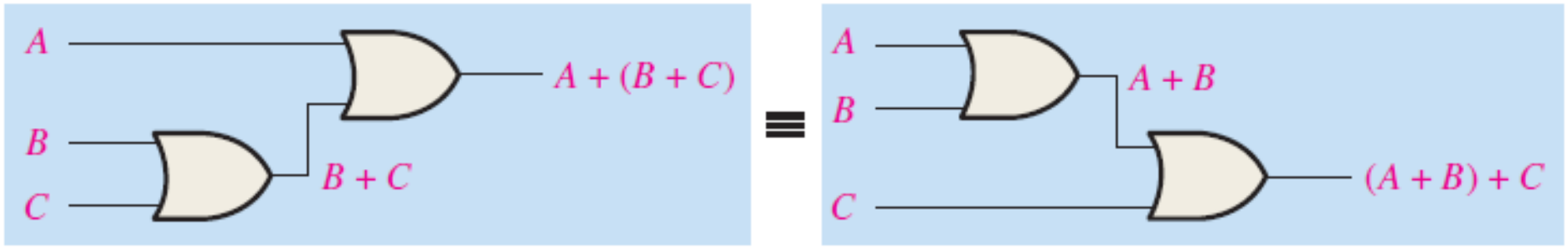
- The commutative law for multiplication can be written as **AB=BA**



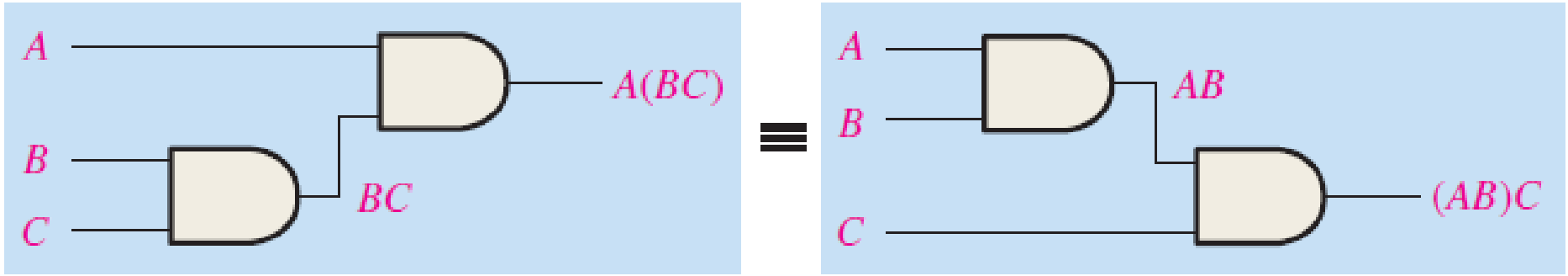
Laws of Boolean Algebra

Associative Law:

- The associative law of addition for three variables is written as $\mathbf{A+(B+C)=(A+B)+C}$



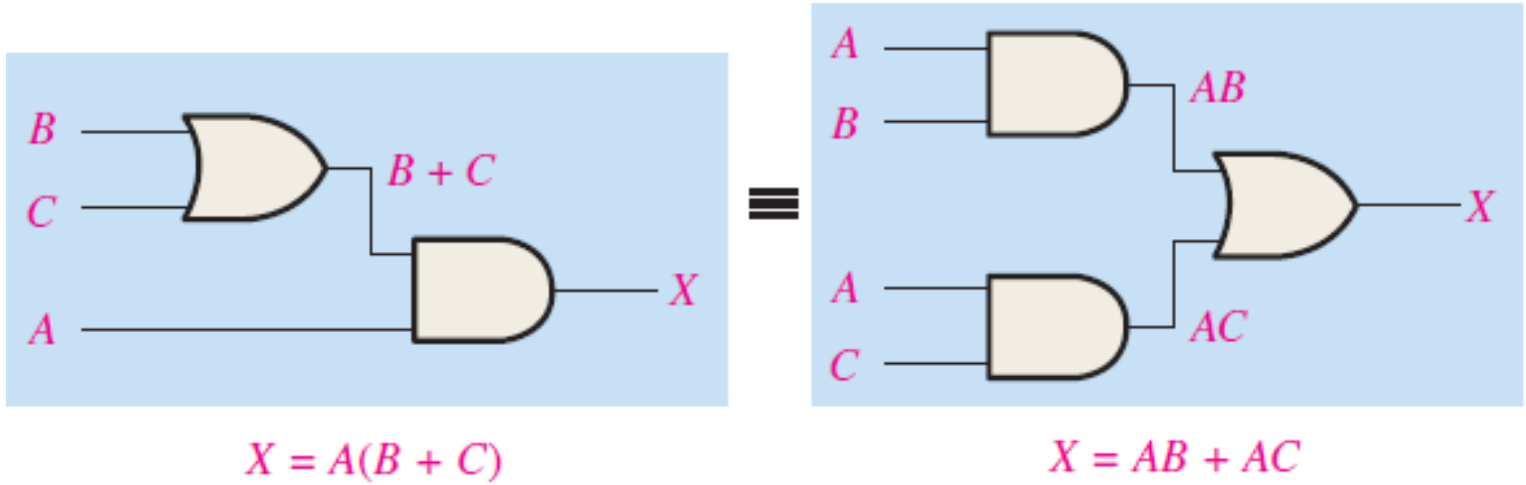
- The associative law of multiplication for three variables is written as $\mathbf{A(BC)=(AB)C}$



Laws of Boolean Algebra

Distributive Law:

- The distributive law for three variables is written as $\mathbf{A(B+C)=AB+AC}$



Rules of Boolean Algebra

Basic rules of Boolean algebra.

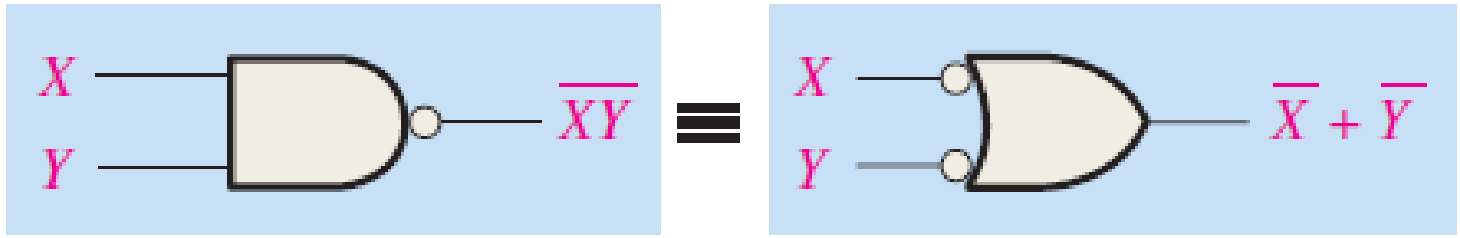
1. $A + 0 = A$
2. $A + 1 = 1$
3. $A \cdot 0 = 0$
4. $A \cdot 1 = A$
5. $A + A = A$
6. $A + \overline{A} = 1$
7. $A \cdot A = A$
8. $A \cdot \overline{A} = 0$
9. $\overline{\overline{A}} = A$
10. $A + AB = A$
11. $A + \overline{A}B = A + B$
12. $(A + B)(A + C) = A + BC$

De Morgan's Theorem

The first theorem is stated as follows:

- The complement of a product of variables is equal to the sum of the complements of complements of the variable.
- The formula of this theorem for two variables is written as:

$$\overline{XY} = \overline{X} + \overline{Y}$$



NAND Negative-OR

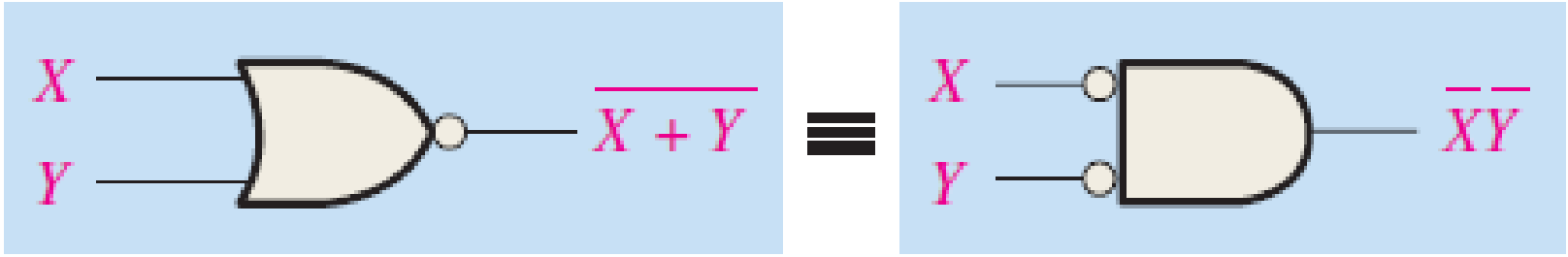
Inputs		Output	
X	Y	$\overline{XY}$	$\overline{X} + \overline{Y}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

De Morgan's Theorem

The second theorem is stated as follows:

- The complement of a sum of variables is equal to the product of the complements of the variables.
- The formula of this theorem for two variables is written as:

$$\overline{X + Y} = \overline{X}.\overline{Y}$$



NOR Negative-AND

Inputs		Output	
X	Y	$\overline{X + Y}$	$\overline{X}\,\overline{Y}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

Application of De Morgan's Theorem

Apply DeMorgan's theorems to the expressions $\overline{XYZ}$ and $\overline{X + Y + Z}$.

Solution

$$\overline{XYZ} = \overline{X} + \overline{Y} + \overline{Z}$$
$$\overline{X + Y + Z} = \overline{X}\,\overline{Y}\,\overline{Z}$$

Apply DeMorgan's theorems to the expressions $\overline{WXYZ}$ and $\overline{W + X + Y + Z}$.

Solution

$$\overline{WXYZ} = \overline{W} + \overline{X} + \overline{Y} + \overline{Z}$$
$$\overline{W + X + Y + Z} = \overline{W}\,\overline{X}\,\overline{Y}\,\overline{Z}$$

Apply DeMorgan's theorems to each expression:

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

(b) $\overline{(\overline{A} + B) + CD}$

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

Solution

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

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

(c) $\overline{(A + B)\overline{C}\,\overline{D} + E + \overline{F}} = \overline{((A + B)\overline{C}\,\overline{D})}\,\overline{(E + \overline{F})} = (\overline{A}\,\overline{B} + C + D)\overline{E}F$

Application of De Morgan's Theorem

Apply DeMorgan's theorem to the expression $\overline{\overline{X} + \overline{Y} + \overline{Z}}$.

Apply DeMorgan's theorem to the expression $\overline{\overline{W}\,\overline{X}\,\overline{Y}\,\overline{Z}}$.

Apply DeMorgan's theorems to each of the following expressions:

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

(b) $\overline{ABC + DEF}$

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

The Boolean expression for an exclusive-OR gate is $A\overline{B} + \overline{A}B$. With this as a starting point, use DeMorgan's theorems and any other rules or laws that are applicable to develop an expression for the exclusive-NOR gate.

Starting with the expression for a 4-input NAND gate, use DeMorgan's theorems to develop an expression for a 4-input negative-OR gate.

Apply DeMorgan's theorems to the following expressions:

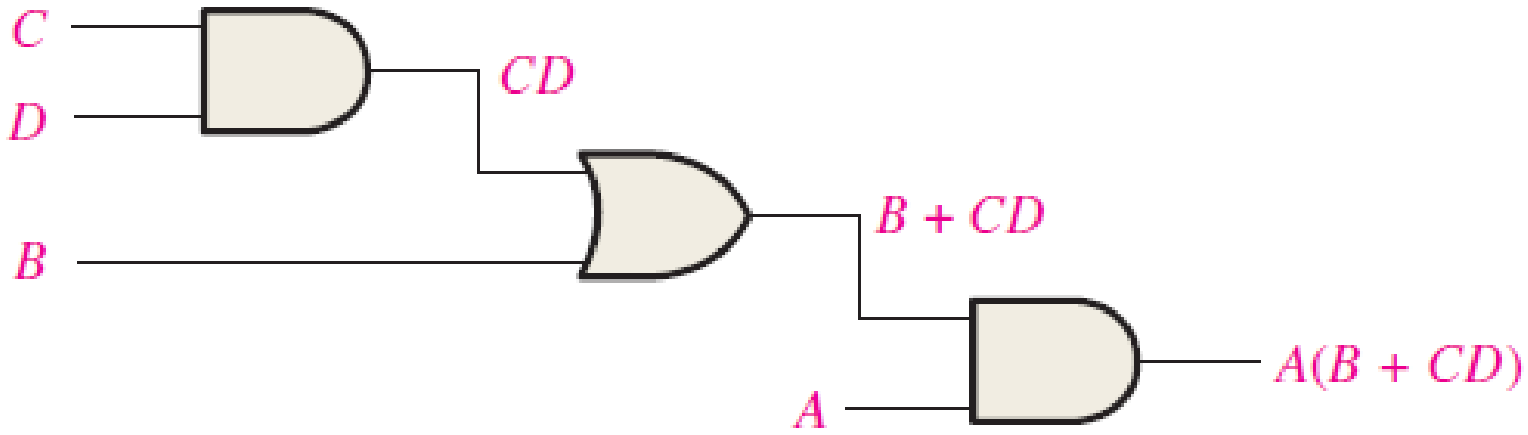
(a) $\overline{ABC} + (\overline{\overline{D} + E})$ **(b)** $\overline{(A + B)C}$ **(c)** $\overline{\overline{A + B + C} + \overline{D}E}$

Boolean Analysis of Logic Circuit

Boolean algebra provides a concise way to express the operation of a logic circuit formed by a combination of logic gates so that the output can be determined for various combinations of input values.

- **A logic circuit can be expressed by Boolean expression and Boolean expression can be implemented by a logic circuit.**
- **The following Boolean expression can be implemented by the logic circuit below:**

$$\mathbf{A(B + CD)}$$



Constructing a Truth-table from a Boolean Expression

- Once we have the Boolean expression describing a process or a logical circuit, a truth-table to show the operation for all possible combination can be constructed.
- First, we need to determine the number of inputs in the expression.
- Then, we need to note down all possible combination of the inputs.
- Lastly, we will evaluation the expression for all possible combination.

$$\mathbf{A(B + CD)}$$

Inputs				Output
A	*B*	*C*	*D*	*A*(*B* + *CD*)
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

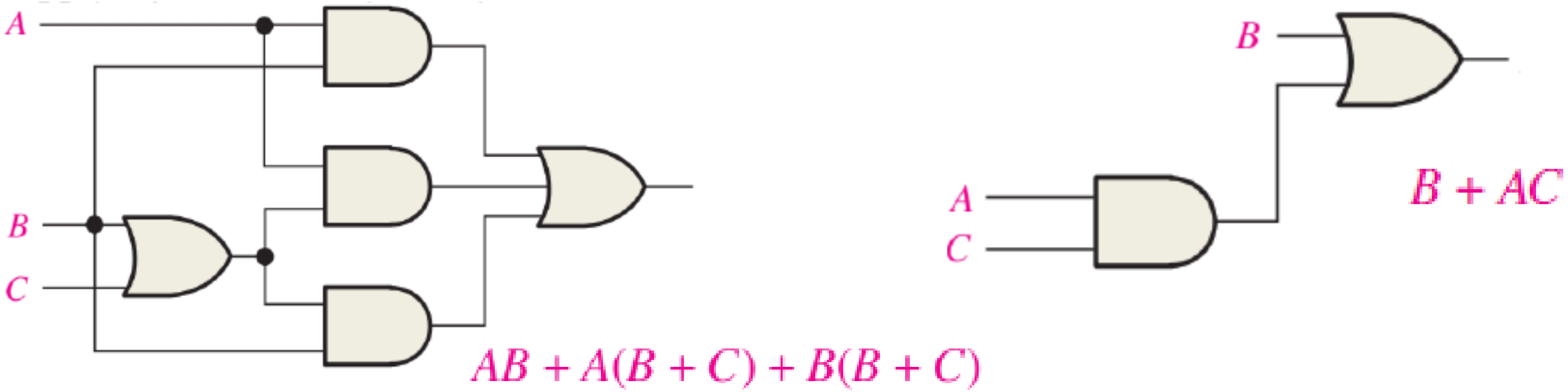
Logic Simplification

- We know that a Boolean expression can be implemented by a logical circuit.
- A large Boolean expression can often be simplified to a simpler and shorter expression.
- This is done by applying the laws and rules of Boolean algebra.
- Simplifying makes implementation simpler and thus requires lesser number of gates.
- Boolean algebra can be used to simplify the following expression:

$$\mathbf{AB + A(B + C) + B(B + C)}$$

- The simplified expression is:

$$\mathbf{B + AC}$$



Logic Simplification

Simplify the Boolean expression $A\overline{B} + A(\overline{B + C}) + B(\overline{B + C})$.

Simplify the following Boolean expression:

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

Simplify the following Boolean expression:

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

Simplify the following Boolean expression:

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

Simplify the Boolean expression $\overline{AB} + \overline{AC} + \overline{A}\,\overline{B}\,\overline{C}$.

***Applying rules of Boolean algebra and DeMorgan's Theorem show that:

i) $\mathbf{MN + \overline{MO} + M\overline{N}O(MN + O) = 1}$

ii) $\mathbf{\overline{A}B\overline{C} + \overline{A}BC + AB\overline{C} + ABC = B}$

1. Simplify the following Boolean expressions:

   (a) $A + AB + A\overline{B}C$ (b) $(\overline{A} + B)C + ABC$ (c) $A\overline{B}C(BD + CDE) + A\overline{C}$

2. Implement each expression in Question 1 as originally stated with the appropriate logic gates. Then implement the simplified expression, and compare the number of gates.

References

1. Thomas L. Floyd, “Digital Fundamentals” 11th edition, Prentice Hall – Pearson Education.

Thank You

Lecture -2
Sum-of-Products, Product-of-Sum

Prepared By: Asif Mahfuz

Standardization

- All Boolean expression, regardless of their form, can be converted into either of the two standard forms.
- The two standard forms are: Sum-of-Products (SOP) and Product-of-Sum (POS).
- Standardization makes evaluation, simplification and implementation of Boolean expression much more systematic and easier.

Domain of a Boolean Expression

- The domain of a general Boolean expression is the set of variable contained in the Boolean expression in either complemented or uncomplemented form.
- For example, the domain of the expression $A\overline{B} + \overline{A}C$ is the set of variable A,B and C.
- And for example, the domain of the expression $A\overline{B}C + \overline{C}E + \overline{A}\overline{D}$ is the set of variables A,B,C,D and E.

Sum-of-Product (SOP)

- Sum of Products: A Product is defined as a term consisting of products of the literals. When two or more products are summed in a Boolean expression, it is called the Sum-of-Products (SOP).

$$\mathbf{A + \bar{A}B + B\bar{C}}$$

- Standard SOP: It is an expression where all the variables are present in each product terms. The products in a SSOP are called min terms ($\mathbf{m_i}$).

$$\mathbf{AB\bar{C} + \bar{A}BC + A\bar{B}C}$$

- Conversion of SOP to Standard SOP
  - Step1: Multiply each of the non-standard terms with a term made up of the sum of the missing variable and its complement. This do not change the function as we are just multiplying by 1.
  - Step2: Repeat step 1 until all the non-standard terms become standard terms.
- **Sum-of-Products is used to describe when the function is 1.**

Product-of-Sum (POS)

- Product of Sum: A Sum is defined as a term consisting of sum of the literals. When two or more sum terms are multiplied in a Boolean Expression, it is called the Product-of-Sum (POS).

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

- Standard POS: It is an expression where all the variables are present in each sum terms. Each sum terms are called max terms ($\mathbf{M_i}$).

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

- Conversion of POS to Standard POS
  - Step1: Add to each non-standard product terms a term made up of the product of the missing variable and its complement. This does not change the expression as we are just adding a 0.
  - Step2: Apply rule 12: $\mathbf{A} + \mathbf{BC} = (\mathbf{A} + \mathbf{B})(\mathbf{A} + \mathbf{C})$
  - Step3: Repeat step 1 until all the sum terms contain all the variable in the domain.
- **Product-of-Sum is used to describe when the function is 0.**

SOP & POS

Convert the following Boolean expression into standard SOP form:

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

Convert the expression $W\overline{X}Y + \overline{X}Y\overline{Z} + WX\overline{Y}$ to standard SOP form.

Convert the following Boolean expression into standard POS form:

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

Convert the expression $(A + \overline{B})(B + C)$ to standard POS form.

Min Terms and Max Terms

- Each variable in a Boolean expression is a literal.
- Boolean variable can appear in normal (A) or complemented ($\overline{A}$) form.
- Each product of all variables in the domain is called Min-Term.
- Each sum of all variables in the domain is called Max-Term.

A	B	C	Min-Terms	
0	0	0	$\overline{A}\overline{B}\overline{C}$	m_0
0	0	1	$\overline{A}\overline{B}C$	m_1
0	1	0	$\overline{A}B\overline{C}$	m_2
0	1	1	$\overline{A}BC$	m_3
1	0	0	$A\overline{B}\overline{C}$	m_4
1	0	1	$A\overline{B}C$	m_5
1	1	0	$AB\overline{C}$	m_6
1	1	1	ABC	m_7

A	B	C	Max-Terms	
0	0	0	$A + B + C$	M_0
0	0	1	$A + B + \overline{C}$	M_1
0	1	0	$A + \overline{B} + C$	M_2
0	1	1	$A + \overline{B} + \overline{C}$	M_3
1	0	0	$\overline{A} + B + C$	M_4
1	0	1	$\overline{A} + B + \overline{C}$	M_5
1	1	0	$\overline{A} + \overline{B} + C$	M_6
1	1	1	$\overline{A} + \overline{B} + \overline{C}$	M_7

For Min-Terms:
When 0→ Complemented Form
When 1→ Normal Form

For Max-Terms:
When 1→ Complemented Form
When 0→ Normal Form

Boolean Expression(SOP) to Truth-Table

- Truth-table can be formed for any Boolean expression.
- Converting a Boolean Expression to SSOP can make this task a lot easier.
- Find the truth-table for the following Boolean expression:

$$F(A, B, C) = AB + \bar{B}\bar{C}$$

$$F(A, B, C) = AB(C + \bar{C}) + (A + \bar{A})\bar{B}\bar{C}$$

$$F(A, B, C) = ABC + AB\bar{C} + A\bar{B}\bar{C} + \bar{A}\bar{B}\bar{C}$$

A	B	C	Min-Terms
0	0	0	$\bar{A}\bar{B}\bar{C}$
0	0	1	$\bar{A}\bar{B}C$
0	1	0	$\bar{A}B\bar{C}$
0	1	1	$\bar{A}BC$
1	0	0	$A\bar{B}\bar{C}$
1	0	1	$A\bar{B}C$
1	1	0	$AB\bar{C}$
1	1	1	ABC

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

Truth-Table to Function Implementation

- Find and implement the function from the following truth-table.

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

A	B	C	F
0	0	0	$\bar{A}\bar{B}\bar{C}$
0	0	1	
0	1	0	$\bar{A}B\bar{C}$
0	1	1	
1	0	0	
1	0	1	
1	1	0	$AB\bar{C}$
1	1	1	ABC

$$\mathbf{F(A,B,C) = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + AB\bar{C} + ABC}$$
$$\mathbf{F(A,B,C) = \bar{A}\bar{C} + AB}$$

- For the function F $= \sum(2,3,5,7)$:
  a) Construct the truth table.
  b) Implement the function.

Boolean Expression(POS) to Truth-Table

- Form a truth-table for the following Boolean expression.

$$\mathbf{F(A, B, C) = (A + B) \cdot (\bar{B} + \bar{C})}$$

- In the first step, we will convert the POS to SPOS:

$$\mathbf{F(A, B, C) = (A + B + C\bar{C}) \cdot (\bar{B} + \bar{C} + A\bar{A})}$$

$$\mathbf{F(A, B, C) = (A + B + C) \cdot (A + B + \bar{C}) \cdot (\bar{B} + \bar{C} + A) \cdot (\bar{B} + \bar{C} + \bar{A})}$$

- Once we have converted the expression to SPOS, now we can directly form the truth-table.

A	B	C	Max-Terms
0	0	0	$A + B + C$
0	0	1	$A + B + \bar{C}$
0	1	0	$A + \bar{B} + C$
0	1	1	$A + \bar{B} + \bar{C}$
1	0	0	$\bar{A} + B + C$
1	0	1	$\bar{A} + B + \bar{C}$
1	1	0	$\bar{A} + \bar{B} + C$
1	1	1	$\bar{A} + \bar{B} + \bar{C}$

A	B	C	Max-Terms
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Function Implementation using POS

- For the function $F = \prod(1,2,5,6)$:
  a) Construct the truth table.
  b) Implement the function.

Solution:

Step 1

A	B	C	Max-Terms
0	0	0	
0	0	1	0
0	1	0	0
0	1	1	
1	0	0	
1	0	1	0
1	1	0	0
1	1	1	

Step 2

A	B	C	Max-Terms
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Step 3

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

- **Now that we have the Boolean expression, we can simplify it and implement it.**

Connecting the Dots between SOP and POS

- From the truth table determine the standard SOP expression and the equivalent POS expression.

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

Standard SOP expression:

- Write down the sum of min terms of the combinations for which the function is 1.

$$F = \bar{A}BC + A\bar{B}\bar{C} + A\bar{B}C + AB\bar{C} + ABC$$

Equivalent POS expression:

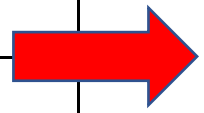
- Write down the product of max terms of the combinations for which the function is 0.

$$F = (A + B + C) \cdot (A + B + \bar{C}) \cdot (A + \bar{B} + C)$$

HOW DOES THIS EVEN WORK!!!!!!!!

Connecting the Dots between SOP and POS

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



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

$$F = \bar{A}BC + A\bar{B}\bar{C} + A\bar{B}C + AB\bar{C} + ABC$$

$$\bar{F} = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}B\bar{C}$$

- Now we have the SOP expression of the complement of the function:

$$\bar{F} = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}B\bar{C}$$

- This means we have converted the 0s of the function 1s and vice versa.
- What if we turn the 0s back to 1, that is we convert $\bar{F}$ to F

Connecting the Dots between SOP and POS

$$\bar{F} = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}B\bar{C}$$

- Now to convert the 0s back to 1s we complement $\bar{F}$ that is $\bar{\bar{F}}$.

$$\bar{\bar{F}} = \overline{\bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}B\bar{C}}$$

$$F = \overline{\bar{A}\bar{B}\bar{C}} \cdot \overline{\bar{A}\bar{B}C} \cdot \overline{\bar{A}B\bar{C}}$$

$$F = (A + B + C) \cdot (A + B + \bar{C}) \cdot (A + \bar{B} + C)$$

- So we have reached the same expression of POS as we did earlier.
- SOP is the positive logic definition of the function.
- POS is the negative logic definition of the function.

What is positive logic and what is negative logic????

References

1. Thomas L. Floyd, “Digital Fundamentals” 11th edition, Prentice Hall – Pearson Education.

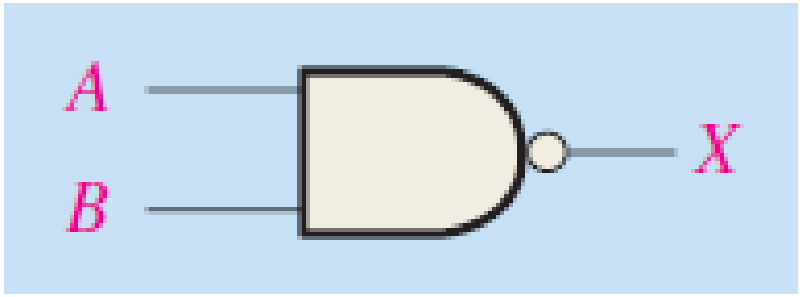
Thank You

Lecture -3
Universal Gates

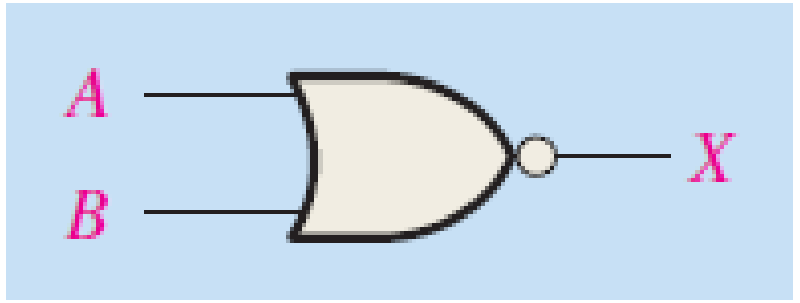
Prepared By: Asif Mahfuz

Universal Gates

- Although we can implement any circuit with AND/OR/NOT, we can also implement any circuit with only NAND or NOR gates.
- We might want to do this because of technology considerations, that is, these gates might be cheaper to implement in silicon or they might be the only type of gates we have available.
- Since we can always use only NAND or NOR gates, these gates are sometimes called universal gates.
- The "trick" (if you want to call it that) is to see that we can implement the three basic gates (AND, OR, NOT) in terms of NAND or NOR gates.



$$\mathbf{X = \overline{AB} = \overline{A} + \overline{B}}$$

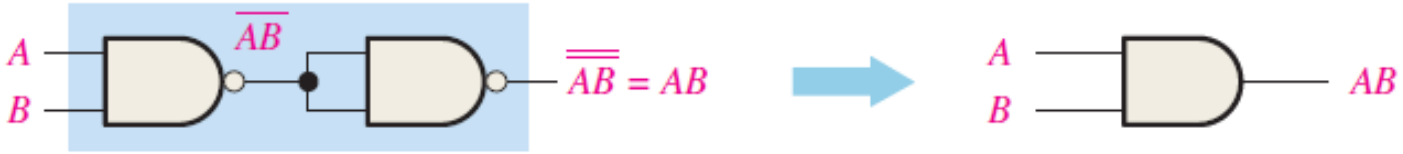


$$\mathbf{X = \overline{A + B} = \overline{A} \cdot \overline{B}}$$

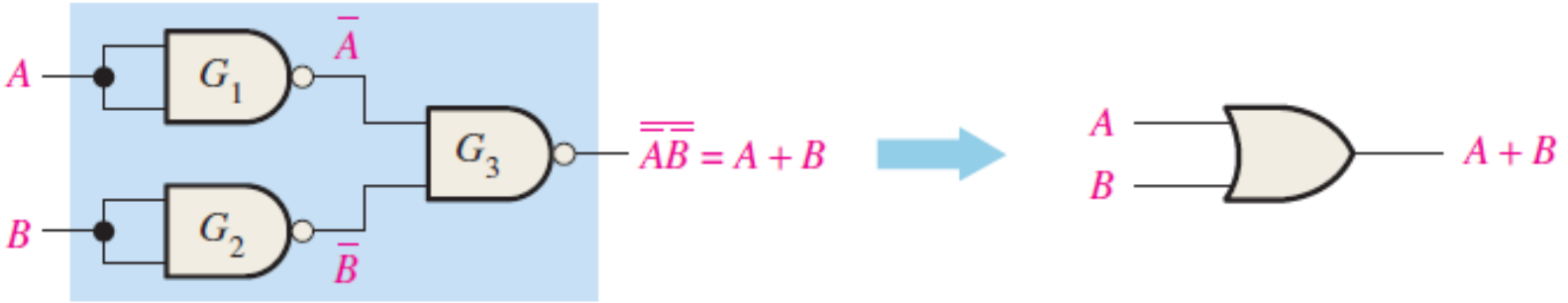
NAND Gate Implementation of Basic Gates

- NOT Gate

- AND Gate



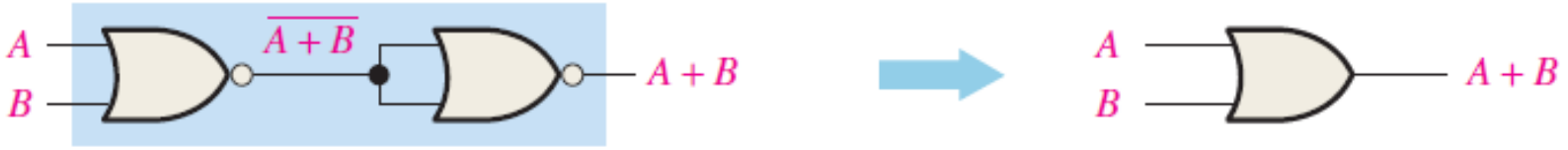
- OR Gate



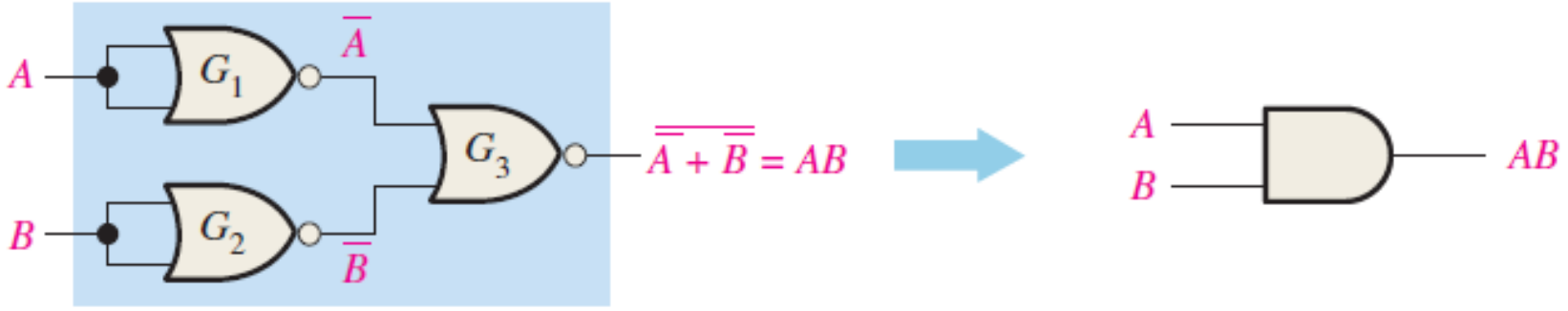
NOR Gate Implementation of Basic Gates

- NOT Gate

- OR Gate



- AND Gate



NAND Gate Implementation of Boolean Logic

- There are three basic tricks/ways to convert any Boolean Logic Circuit to its NAND equivalent:
  1. First implement the Boolean Logic with basic AND, OR and NOT gates and then replace the basic gates with its NAND equivalent.
  2. Convert the Boolean Logic expression to its NAND form using De Morgan's theorem and then implement with NAND gates directly.
  3. Bubble Pushing Technique.
- Implement the following Boolean expression with NAND logic:
  a) $F = \overline{AB} + CD$
  b) $F = \overline{\overline{AB}} + CD$
  c) $F = A(\overline{B + CD})$
  d) $F = C(\overline{A + BD})$
  e) $F = A\overline{B} + \overline{A}B$

- Implementation will be shown in class!!!!

NOR Gate Implementation of Boolean Logic

- There are three basic tricks/ways to convert any Boolean Logic Circuit to its NOR equivalent:
  1. First implement the Boolean Logic with basic AND, OR and NOT gates and then replace the basic gates with its NOR equivalent.
  2. Convert the Boolean Logic expression to its NOR form using De Morgan's theorem and then implement with NOR gates directly.
  3. Bubble Pushing Technique.
- Implement the following Boolean expression with NOR logic:
  a) $F = \overline{AB} + CD$
  b) $F = \overline{\overline{AB}} + CD$
  c) $F = A(\overline{B + CD})$
  d) $F = C(\overline{A + BD})$
  e) $F = A\overline{B} + \overline{A}B$

- Implementation will be shown in class!!!!

References

1. Thomas L. Floyd, “Digital Fundamentals” 11th edition, Prentice Hall – Pearson Education.

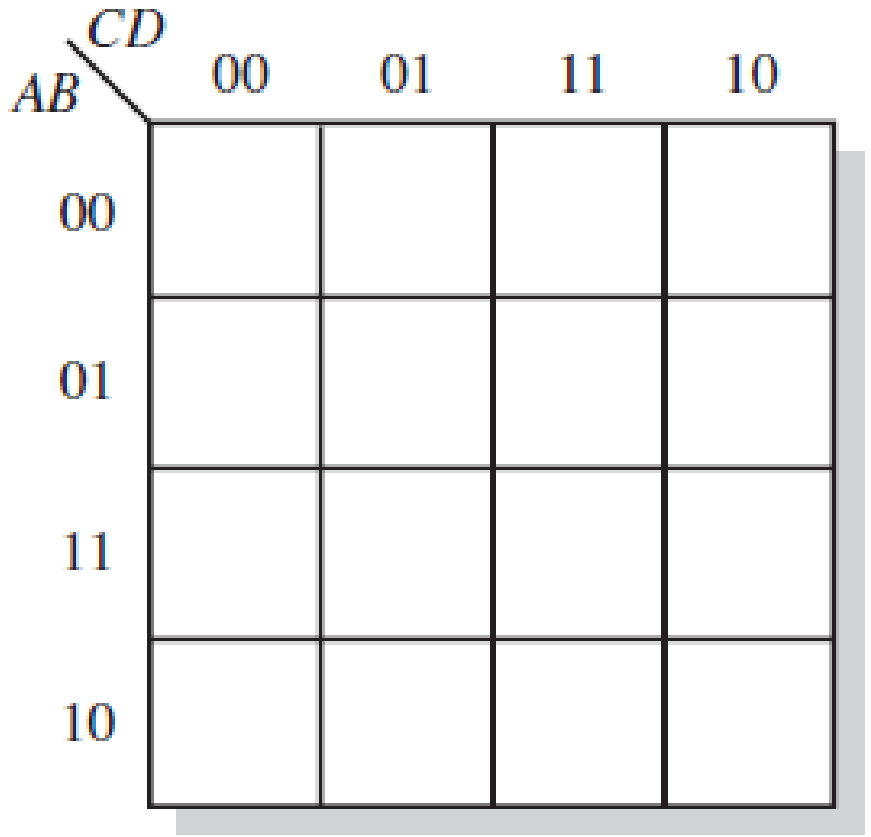
Thank You

Lecture -4
Karnaugh Map

Prepared By: Asif Mahfuz

The Karnaugh Map (K-Map)

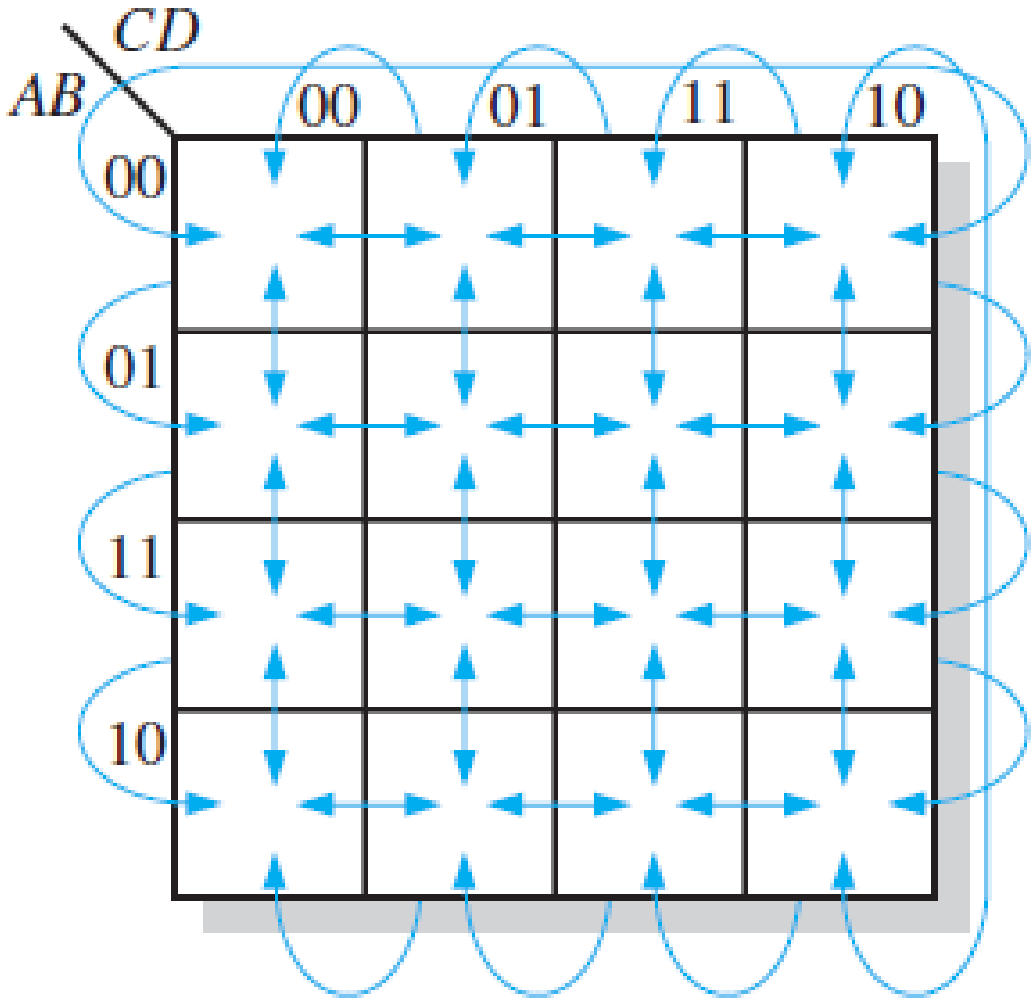
- A Karnaugh map is similar to a truth table because it presents all of the possible values of input variables and the resulting output for each value.
- The main purpose of K-Map is to simplify a Boolean expression.
- A Karnaugh map provides a systematic method for simplifying Boolean expressions and, if properly used, will produce the simplest SOP or POS expression possible, known as the minimum expression.
- The effectiveness of algebraic simplification depends on your familiarity with all the laws, rules, and theorems of Boolean algebra and on your ability to apply them.
- .The Karnaugh map, on the other hand, provides a "cookbook" method for simplification.



A 4-variable K-Map

Cell Adjacency

- The cells in a Karnaugh map are arranged so that there is only a single-variable change between adjacent cells.
- Adjacency is defined by a single-variable change.
- Physically, each cell is adjacent to the cells that are immediately next to it on any of the four sides.
- A cell is not adjacent to the cells that diagonally touch any of its corners.
- The cells in the top row are adjacent to the cells in the bottom row.
- The cells in the left column is adjacent to the cells in the right column.
- This is called “wrap-around” adjacency.



Truth Table to K-Map

- From the following truth table form a K-MAP

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

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

Truth Table to K-Map

- From the following truth table form a K-MAP

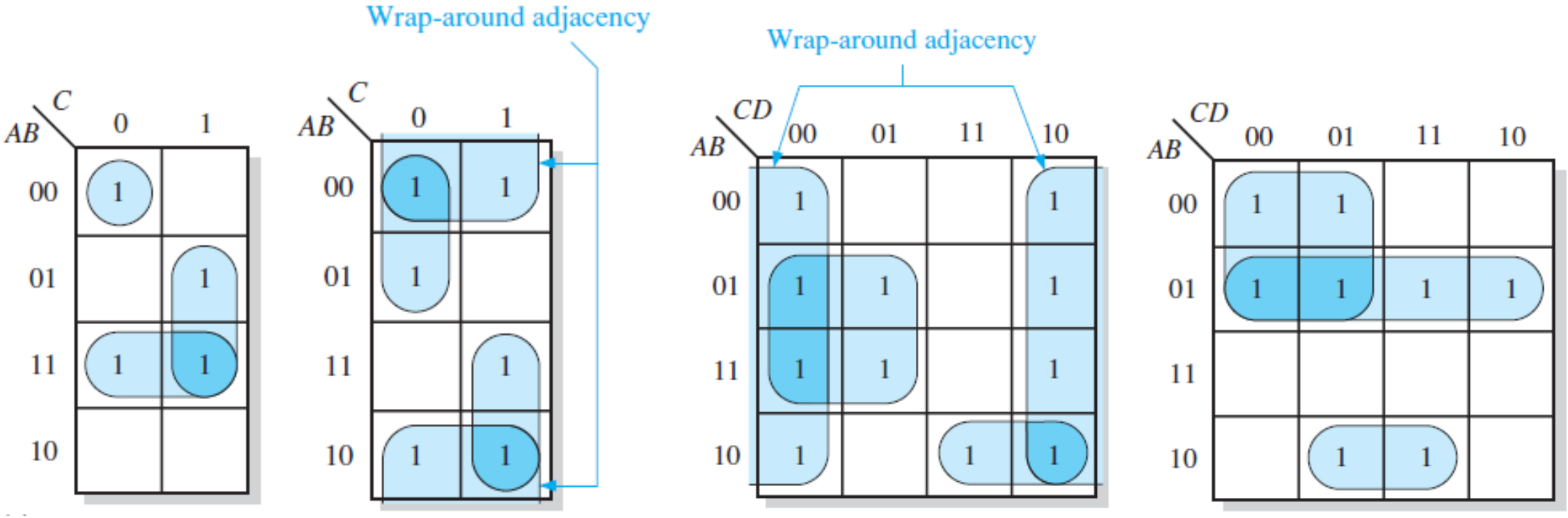
A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

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

Forming the Groups

- To find the minimal SOP/ simplified SOP expression we must first group the 1s that are adjacent.
- To group the 1s we should follow these rules:
  1. A group can contain number of cells that is only a power of 2 (2^n), i.e. 1,2,4,8 and 16.
  2. Each cell in a group must be adjacent to some other cell in the group, but not all cells need to be adjacent.
  3. Priority is to find the group containing maximum number of adjacent cells and to find the minimum no. of groups possible keeping in accordance to rule 1and 2.
  4. Each 1 in the map must be included in at least one group keeping accordance to the rules above.
  5. The 1s included in a group can be included in another group if the overlapping group includes ungrouped 1s keeping accordance to rules 1 to 4.

Forming the Groups



Forming the Groups

- Find the optimal groups for the following K-MAPs

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

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

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

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

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

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

1 - group

Finding the minimal SOP Form

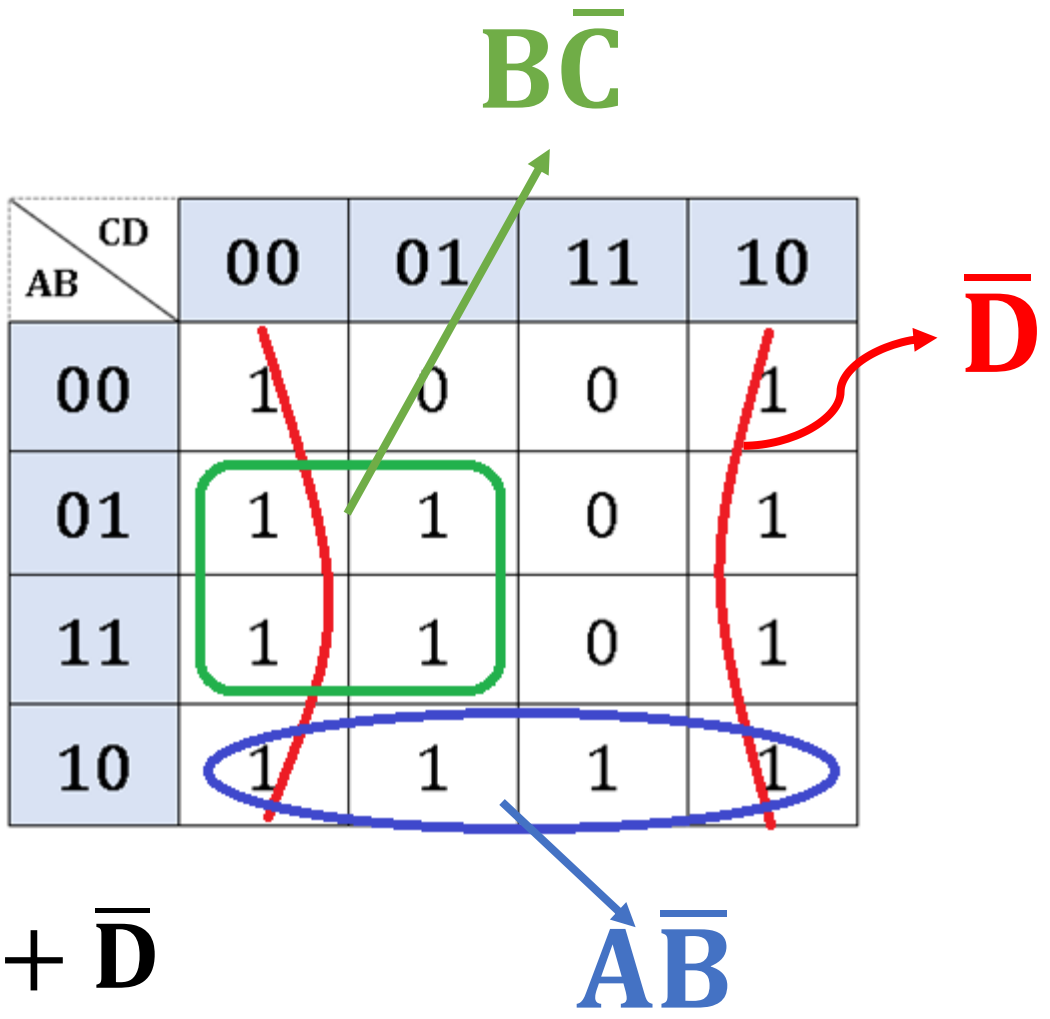
- After the 1s in the K-Maps are grouped, the process of minimizing the expression begins.
- For each group containing the 1s, there are variables which occur only in 1 form (complemented or uncomplemented form) form a product.
- The variables that occur in both complemented and uncomplemented forms are called contradictory variables.
- These variables are eliminated.
- For a 3 variable K-Map:
  - A 1-cell group yields a 3-variable product form.
  - A 2-cell group yields a 2-variable product form.
  - A 4-cell group yields a 1-variable product form.
  - An 8-cell group yields a value of 1 for the expression.
- For a 4 variable K-Map:
  - A 1-cell group yields a 4-variable product form.
  - A 2-cell group yields a 3-variable product form.
  - A 4-cell group yields a 2-variable product form.
  - An 8-cell group yields a 1-variable product form.
  - A 16-cell group yields a value of 1 for the expression.

Finding the minimal SOP Form

Example 1:

- First, we do the grouping.
- Then we find the products for each of the groups

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



$$F = A\bar{B} + B\bar{C} + \bar{D}$$

Finding the minimal SOP Form

- Find the functions for the following K-MAPs

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

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

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

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

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

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

K-Map with Don't Care

- Sometimes situation arises where we do not require all the input combinations, or they are simply not allowed.
- These are some combination that will never occur.
- These combinations can be treated as don't care.
- A don't care combination can be treated as 1 or 0, as per our simplification requirements.

A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

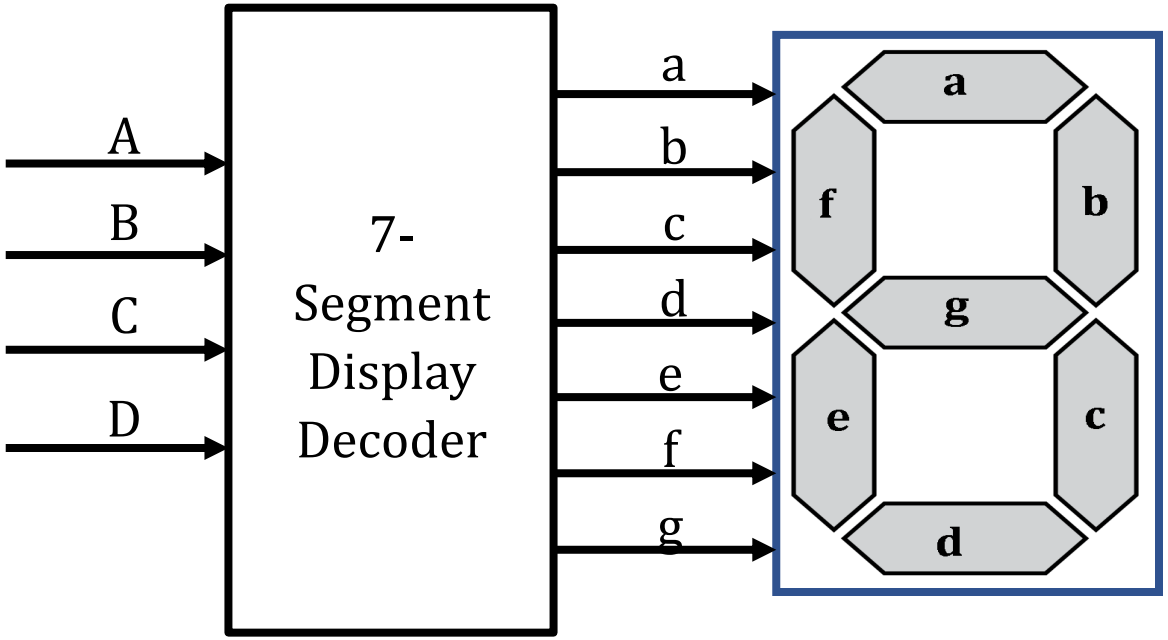
Required

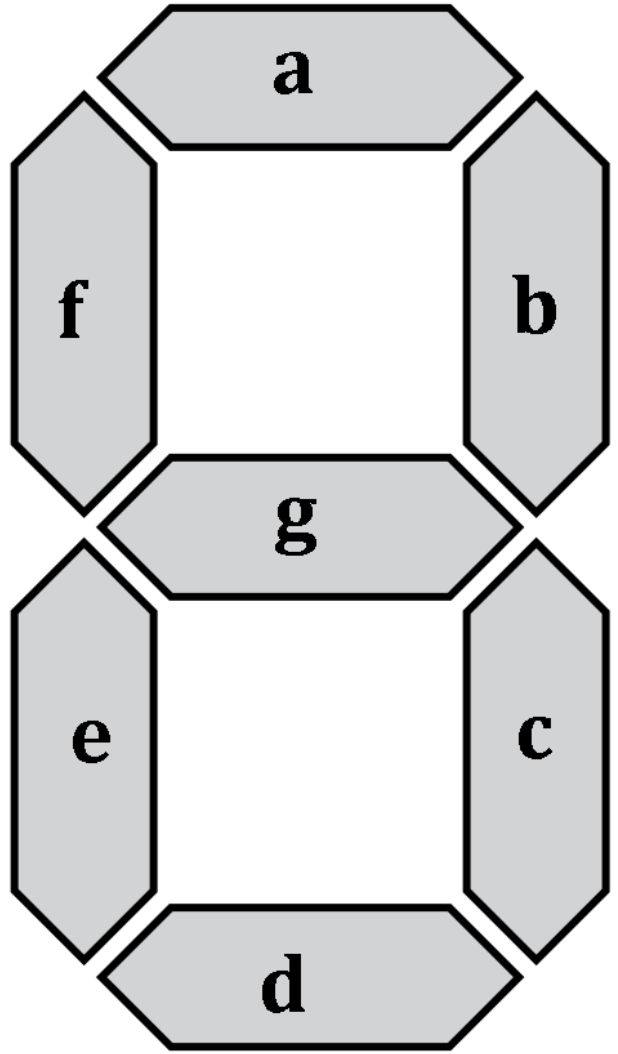
Not Required

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

Application of K-Map with Don't Care

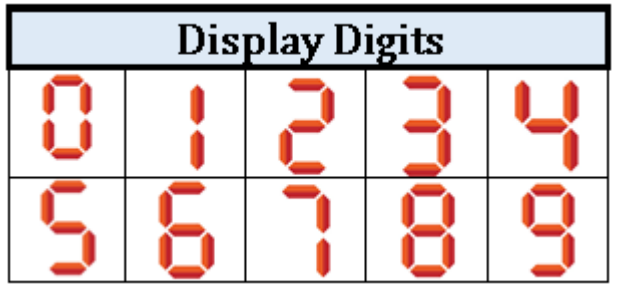
- An application where we need to use don't cares is the design of decoder for 7 segment displays.
- There are two types of 7-segment displays, namely Common Cathode and Common Anode.
- A decoder is needed to display the digits on a 7-segment display.
- We will design the decoder based on a common cathode 7-segment display.

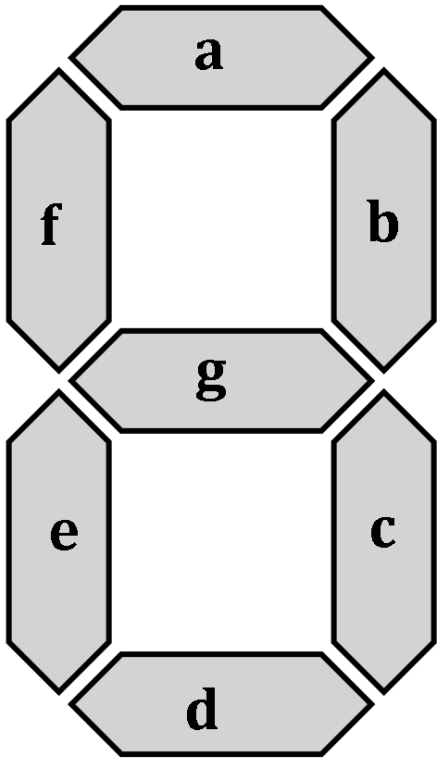




Application of K-Map with Don't Care

Truth-table for 7-segment display decoder logical circuit.





A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
1	0	1	0	X	X	X	X	X	X	X
1	0	1	1	X	X	X	X	X	X	X
1	1	0	0	X	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X

Problem Set for K-Map

1. For the function $F(A, B, C) = \sum(1,2,3,6)$
   a) Construct the truth table.
   b) Find the standard POS.
   c) Find the simplified SOP using KMAP.
   d) Find the simplified POS using KMAP.
   e) Implement the simplified SOP using basic logic gates.
   f) Implement the simplified SOP using universal NAND gates only.
2. For the function $F(A, B, C) = \prod(2,3,5,7)$
   a) Construct the truth table.
   b) Find the standard SOP.
   c) Find the simplified SOP using KMAP.
   d) Find the simplified POS using KMAP.
   e) Implement the simplified POS using basic logic gates.
   f) Implement the simplified SOP using universal NOR gates only.
3. For the function $F(A, B, C, D) = \sum(1,3,8,10)$ and $d(A, B, C, D) = (11,12,13,14,15)$ where, d(A,B,C,D) represents the don't care condition.
   a) Construct the truth table.
   b) Find the simplified SOP using KMAP.
   c) Find the simplified POS using KMAP.
   d) Implement the simplified POS using basic logic gates.
   e) Implement the simplified SOP using universal NAND gates only.

Problem Set for K-Map

4. For the following function $F(A, B, C, D) = \prod(2,4,7,9,11)$ and $d(A, B, C, D) = (1,3,12,13,14,15)$ where d(A,B,C,D) represents the don't care conditions.
   a) Construct the truth table.
   b) Find the simplified SOP using KMAP.
   c) Find the simplified POS using KMAP.
   d) Implement the simplified SOP using basic logic gates.
   e) Implement the simplified POS using universal NOR gates only.

References

1. Thomas L. Floyd, “Digital Fundamentals” 11th edition, Prentice Hall – Pearson Education.

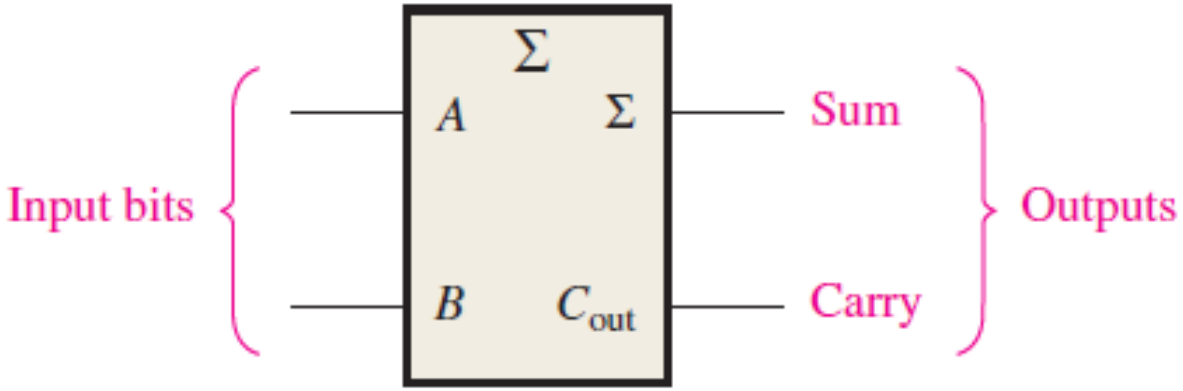
Thank You

Lecture -5
Combination Circuits-1

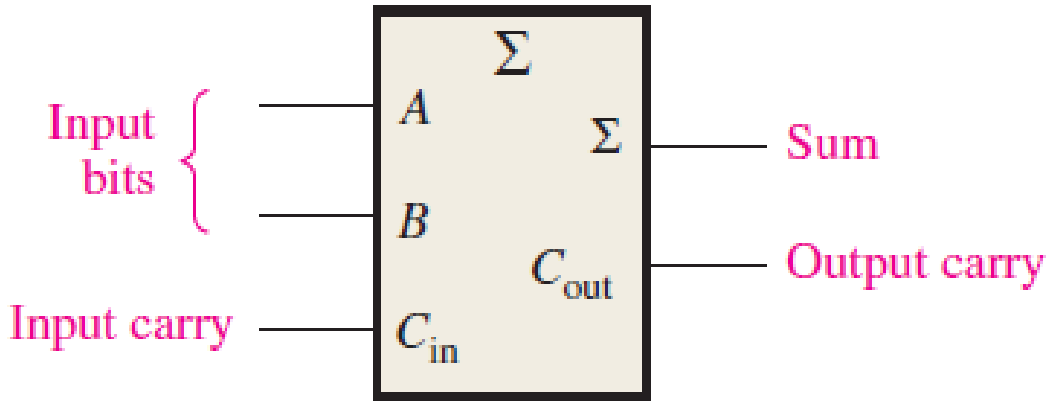
Prepared By: Asif Mahfuz

Adder

- Adders are important in computers and in other digital systems.
- To design an adder we first need to understand the basic adder operation.
- Thus we first need to know the rules of binary addition.
- The rules of binary addition are as follows:
  - $0 + 0 = 0$
  - $0 + 1 = 1$
  - $1 + 0 = 1$
  - $1 + 1 = 10$
- There are two types of adders namely, **Half-adder** and a **Full-adder**.



Block diagram of a Half-Adder



Block diagram of a Full-Adder

Half-Adder

- A half-adder accepts two binary digits on its inputs and produces two binary digits on its outputs- a sum bit and a carry bit.
- So a half-adder has two inputs and two outputs.
- Now to design a digital system we need to know its outputs for all the input combinations.
- The truth-table for the half adder can constructed following the binary addition rule.

A	B	C_{out}	Σ
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

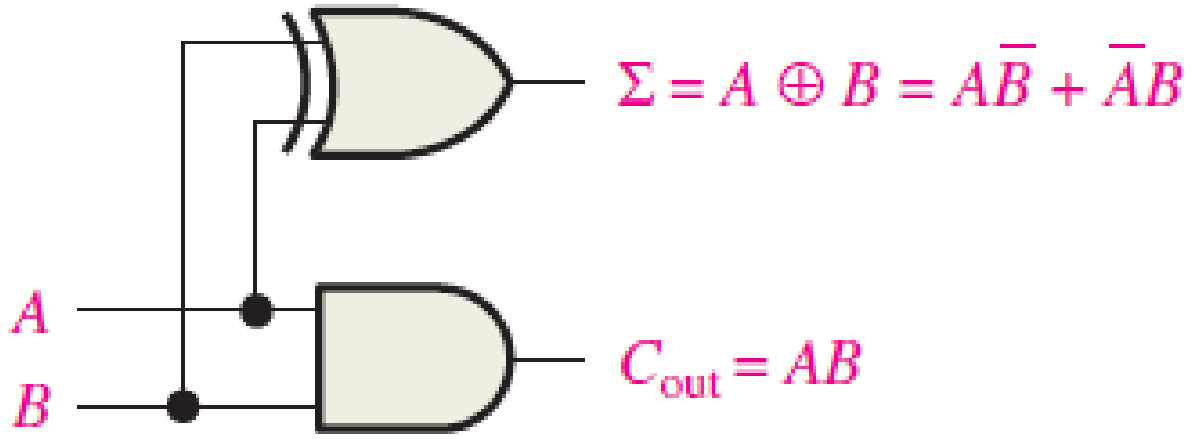
- The Boolean expression of the sum bit can be found as:

$$\mathbf{S = \bar{A}B + A\bar{B} = A \oplus B}$$

- The Boolean expression of carry bit can be found as:

$$\mathbf{C_{IN} = AB}$$

- So now a logical circuit can be drawn for a half-adder.



Full-Adder

- A full-adder accepts two input bits and an input carry and generates a sum output and a carry output.
- So a full adder has three inputs and two outputs.
- The truth table can be constructed the same way as we constructed for a half-adder.

A	B	C_{IN}	C_{out}	$\sum$
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

Full-Adder

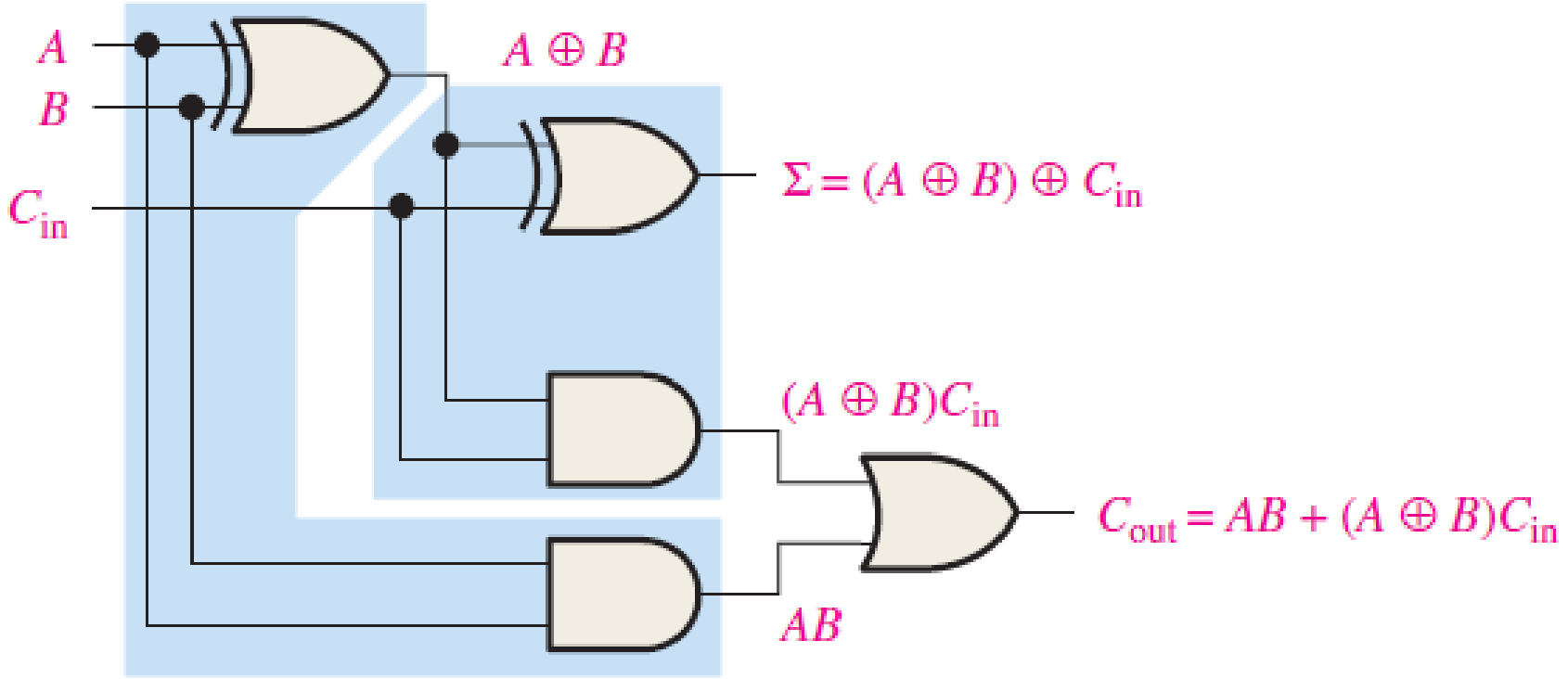
- After constructing the truth, now we can find the Boolean expressions

$$\mathbf{S = \bar{A}\bar{B}C_{IN} + \bar{A}B\overline{C_{IN}} + A\bar{B}\,\overline{C_{IN}} + ABC_{IN}}$$

$$\mathbf{S = A \oplus B \oplus C}$$

$$\mathbf{C_{OUT} = \bar{A}BC_{IN} + A\bar{B}C_{IN} + AB\overline{C_{IN}} + ABC_{IN}}$$

$$\mathbf{C_{OUT} = AB + (A \oplus B)C_{IN}}$$



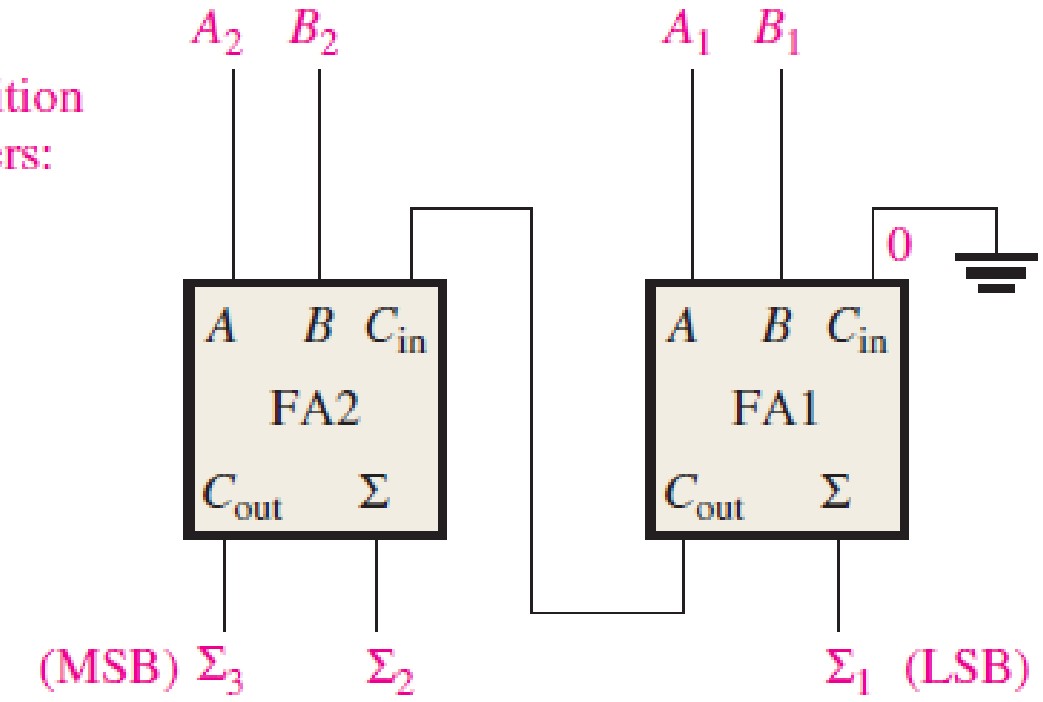
Parallel Adder

- Till now we have learned how add two numbers of 1 bit.
- So how do we add two number of two bits.
- The principle is same as we add two decimal numbers of two digits.
- We sum the first digits and below the first digits we write the sum and the generated carry is summed with the next digits.

Carry	1	
Number 1	2	9
Number 2	3	5
Sum	6	4

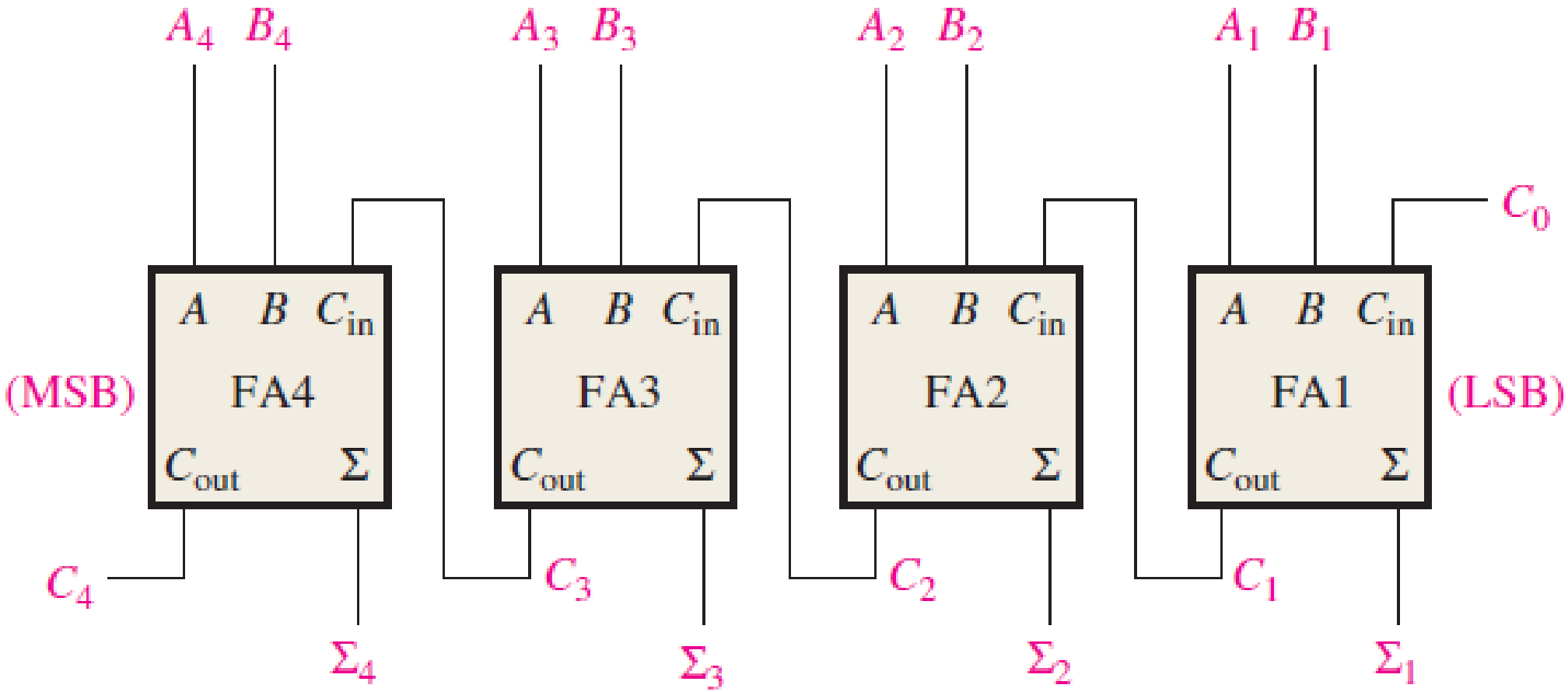
General format, addition of two 2-bit numbers:

$$\begin{array}{r} A_2A_1 \\ +\,B_2B_1 \\ \hline \Sigma_3\Sigma_2\Sigma_1 \end{array}$$



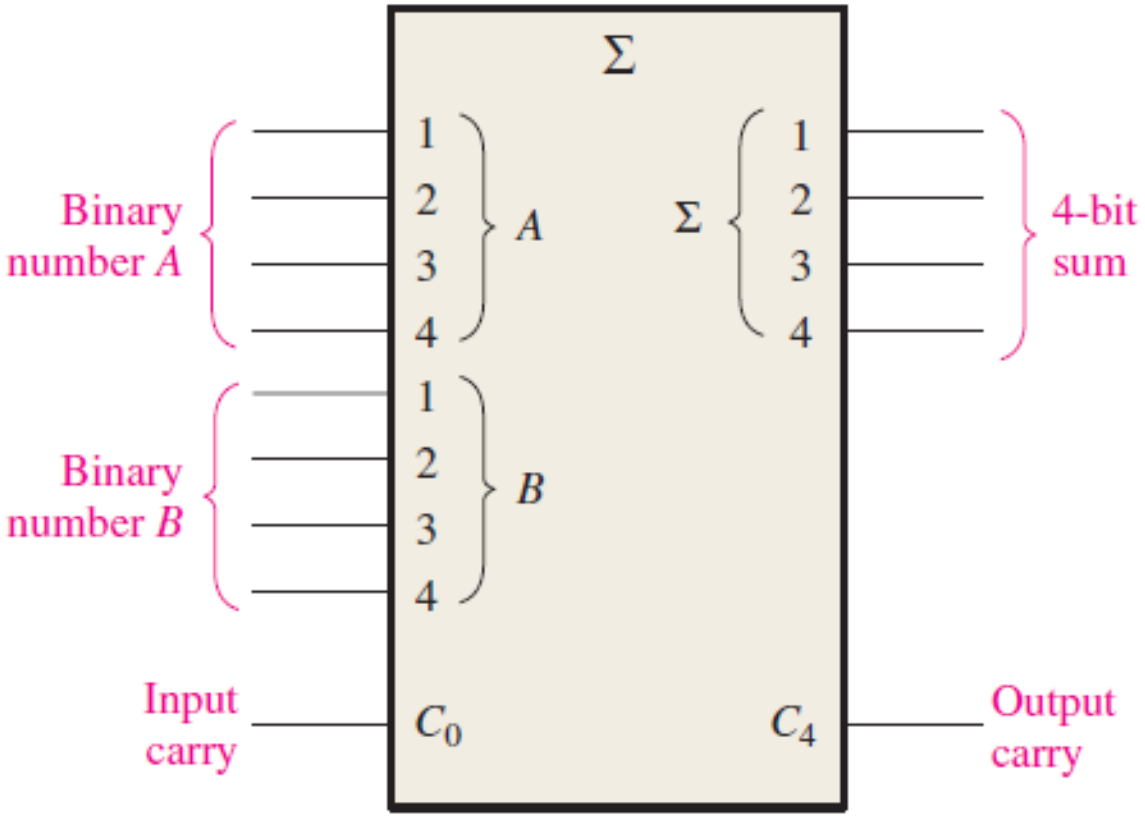
Parallel Adder

- A 4-bit Parallel adder to add two 4bit numbers can be designed using the same principle.



Adder Expansion

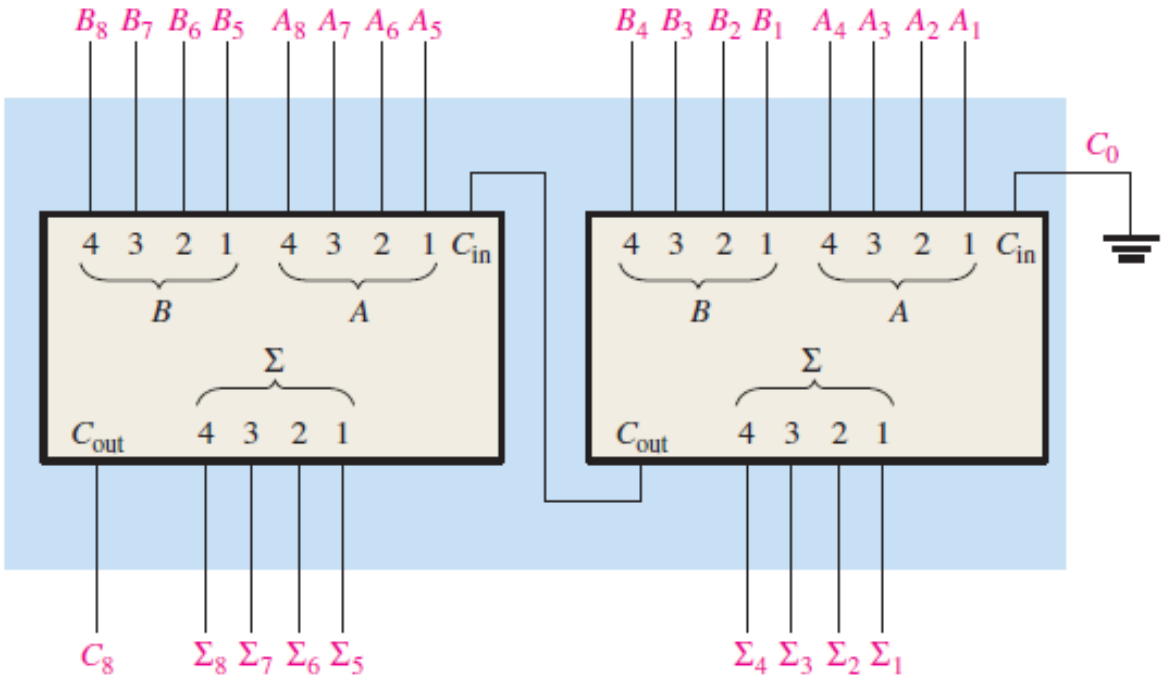
- We have learned how to add two 4-bit binary numbers.
- A 4-bit is called a nibble.
- So how do we add two numbers of 8-bits or even higher.
- That is when we use the concept of adder expansion.
- The following is the logic symbol used to represent a 4-bit parallel adder.



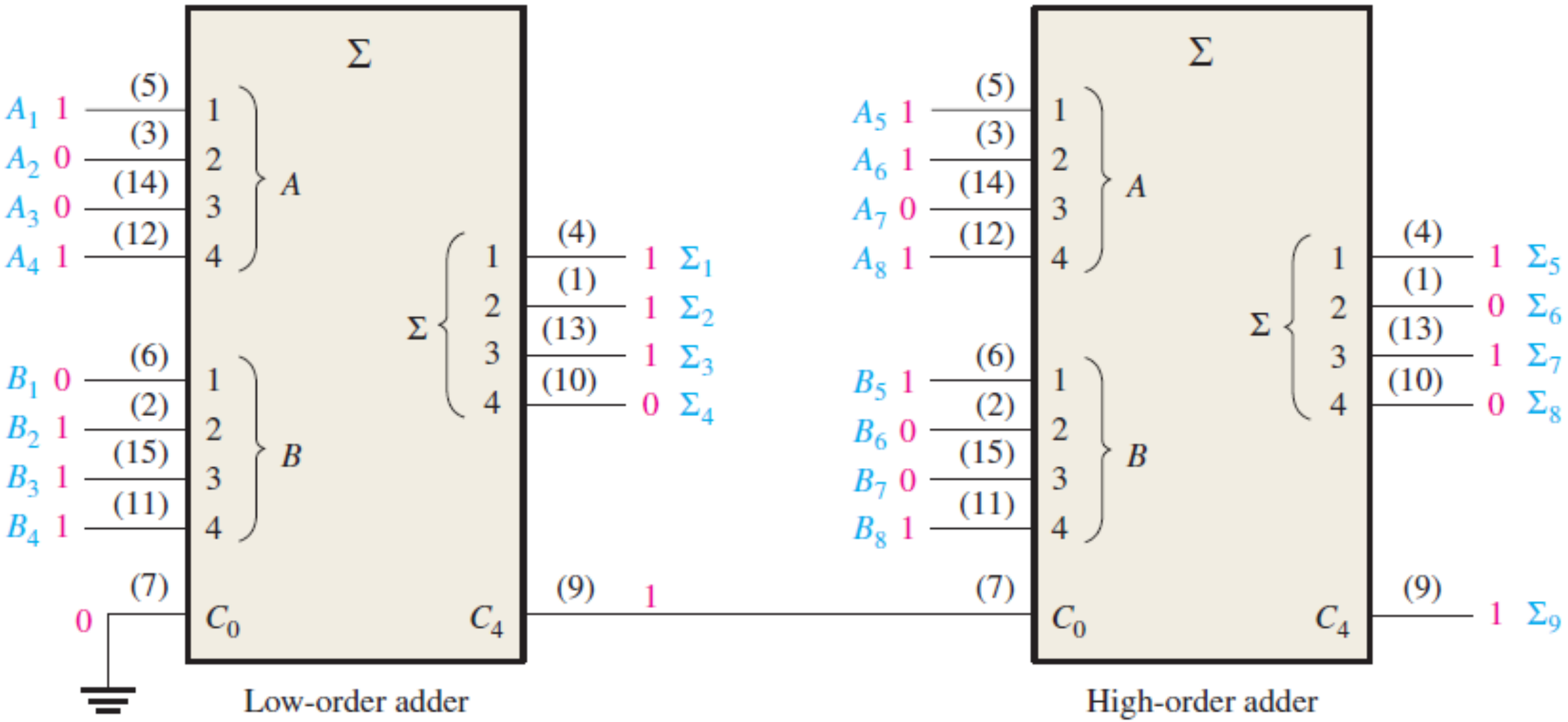
Adder Expansion

- We can cascade two blocks of 4-bit parallel adder to add two numbers of 8-bits.

		1	1	1					
	1	0	1	1		1	0	0	1
	1	0	0	1		1	1	1	0
1	0	1	0	1		0	1	1	1
C	S_8	S_7	S_6	S_5		S_4	S_3	S_2	S_1



Example of Adder Expansion



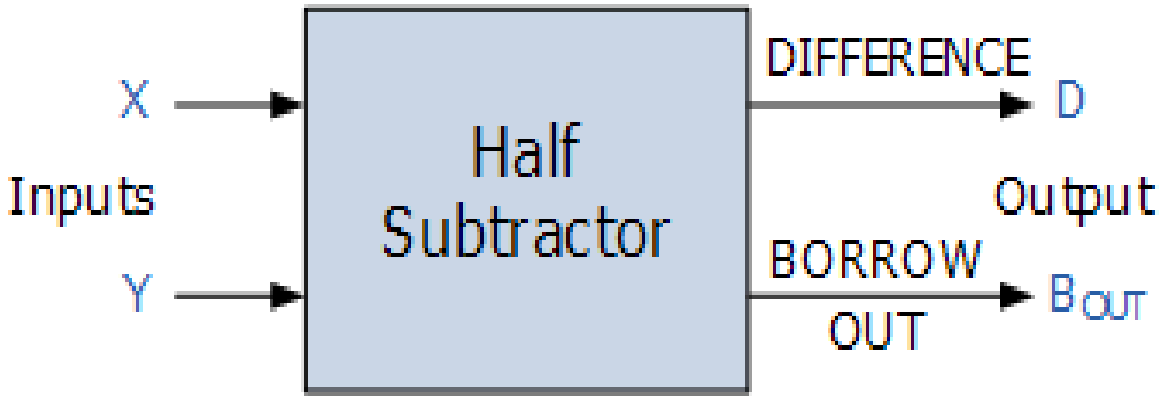
Subtractor

- A subtractor is a combinational circuit that take two numbers as input and produces their difference.
- It also has an output to specify if a 1 has been borrowed.
- The minuend is designated by **x** and the subtrahend is designated by **y**.
- Like adders, subtractors are also of two types namely, Half-subtractor and Full-subtractor.
- The rules of binary subtraction are:

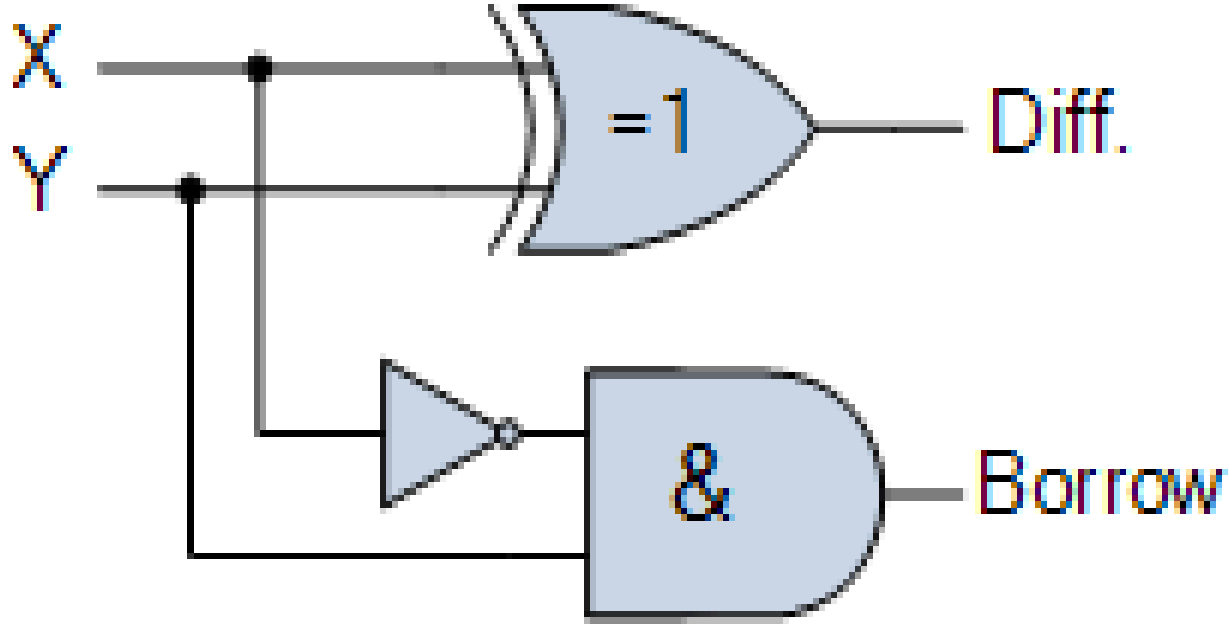
X	Y	B	D
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

Half-Subtractor

- A half-subtractor take two binary bits as input and outputs the difference and borrow.
- The truth table for a half-subtractor is as follows:



X	Y	B	D
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

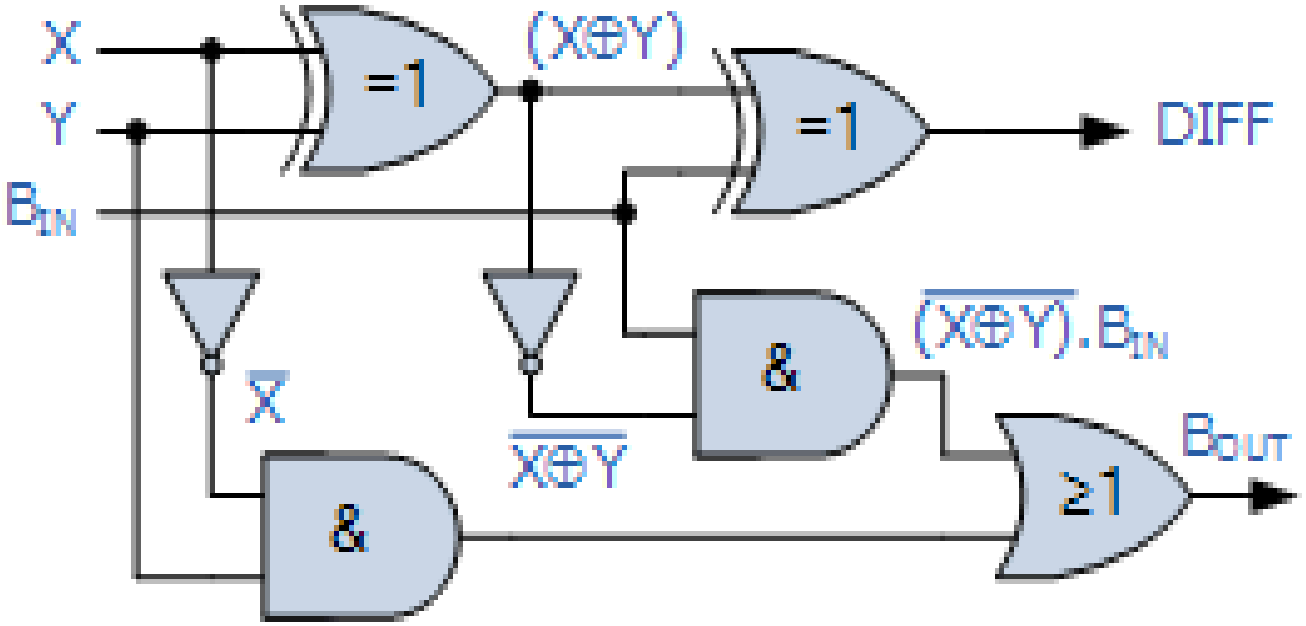


$$D = \bar{X}Y + X\bar{Y} = X \oplus Y$$

$$B = \bar{X}Y$$

Full-Subtractor

- A full-subtractor take the binary bits as input and outputs the difference and borrow.
- The truth table for a full-subtractor is as follows:



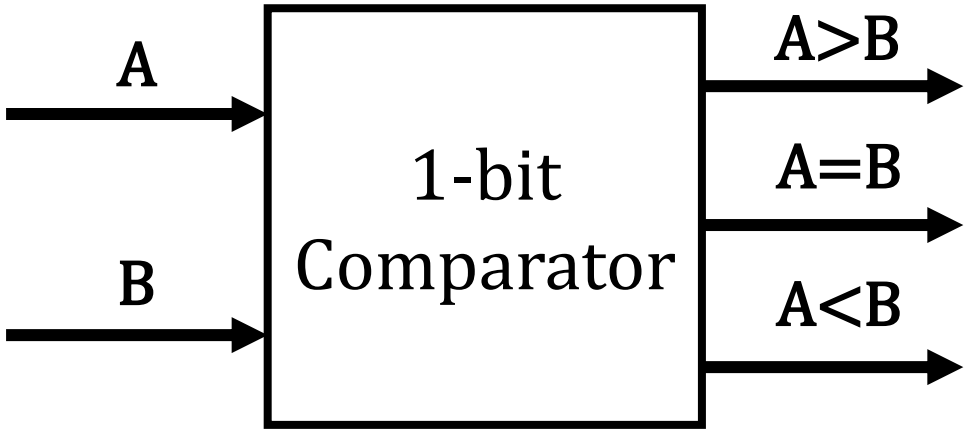
X	Y	B_{IN}	B	D
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

$$D = X \oplus Y \oplus B_{IN}$$

$$B = \overline{X}Y + \overline{(X \oplus Y)}B_{IN}$$

Digital Magnitude Comparator

- A comparator is a combinational circuit which can be used to compare between two number.
- A magnitude comparator has three possible outputs; A is greater than B, A is equal to B and A is less than B.
- The truth-table for a 1-bit comparator can be constructed as:



A	B	A>B	A=B	A<B
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

The Boolean expression for the outputs:

$$F_{A=B} = \overline{A}\,\overline{B} + AB$$

$$F_{A<B} = \overline{A}B$$

$$F_{A>B} = A\overline{B}$$

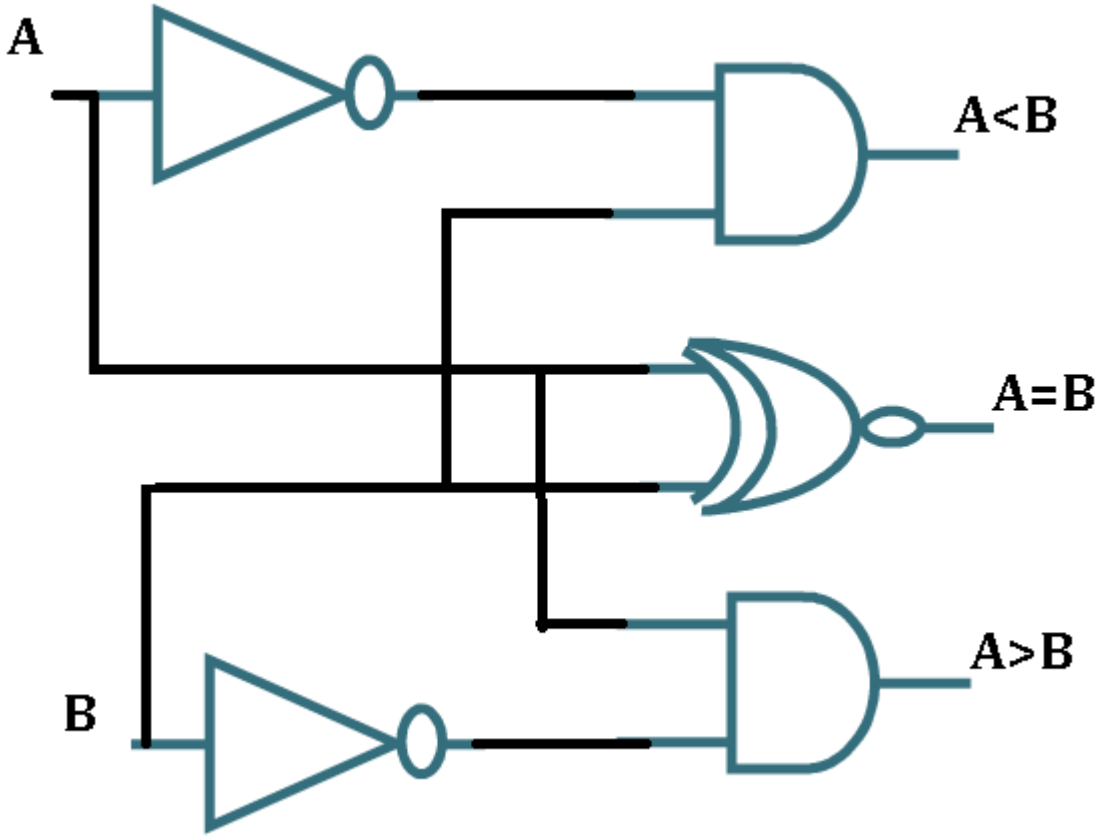
Digital Magnitude Comparator

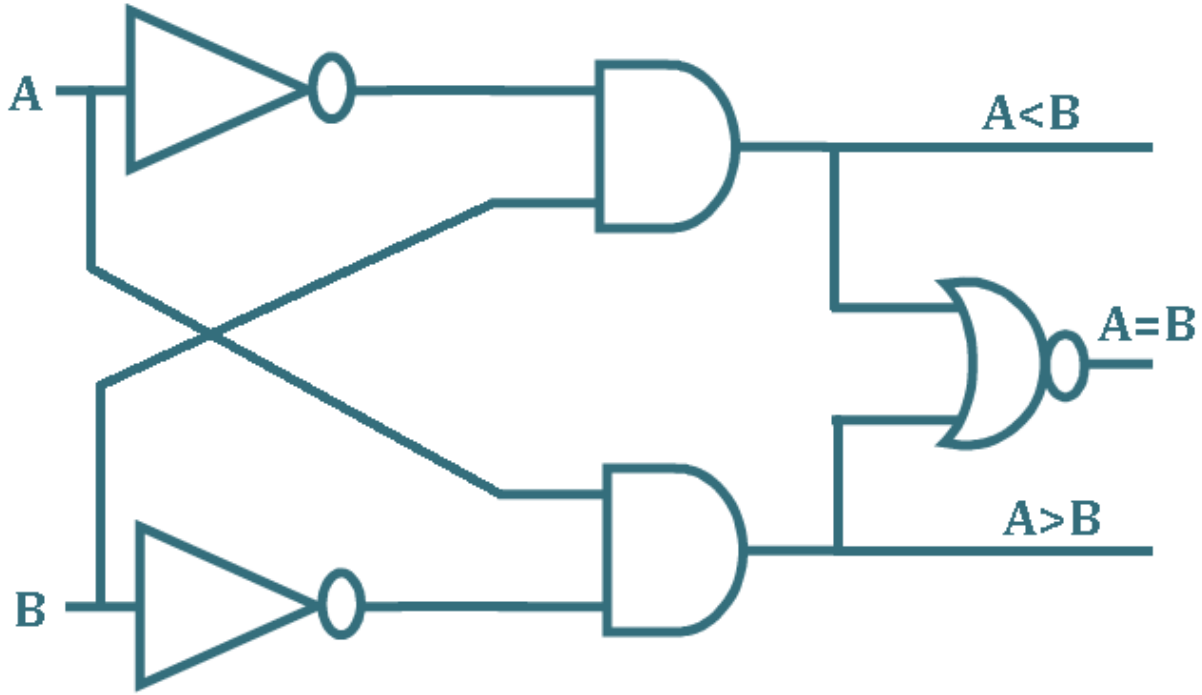
The Boolean expression for the outputs:

$$F_{A=B} = \overline{A}\,\overline{B} + AB = \overline{A \oplus B}$$

$$F_{A<B} = \overline{A}B$$

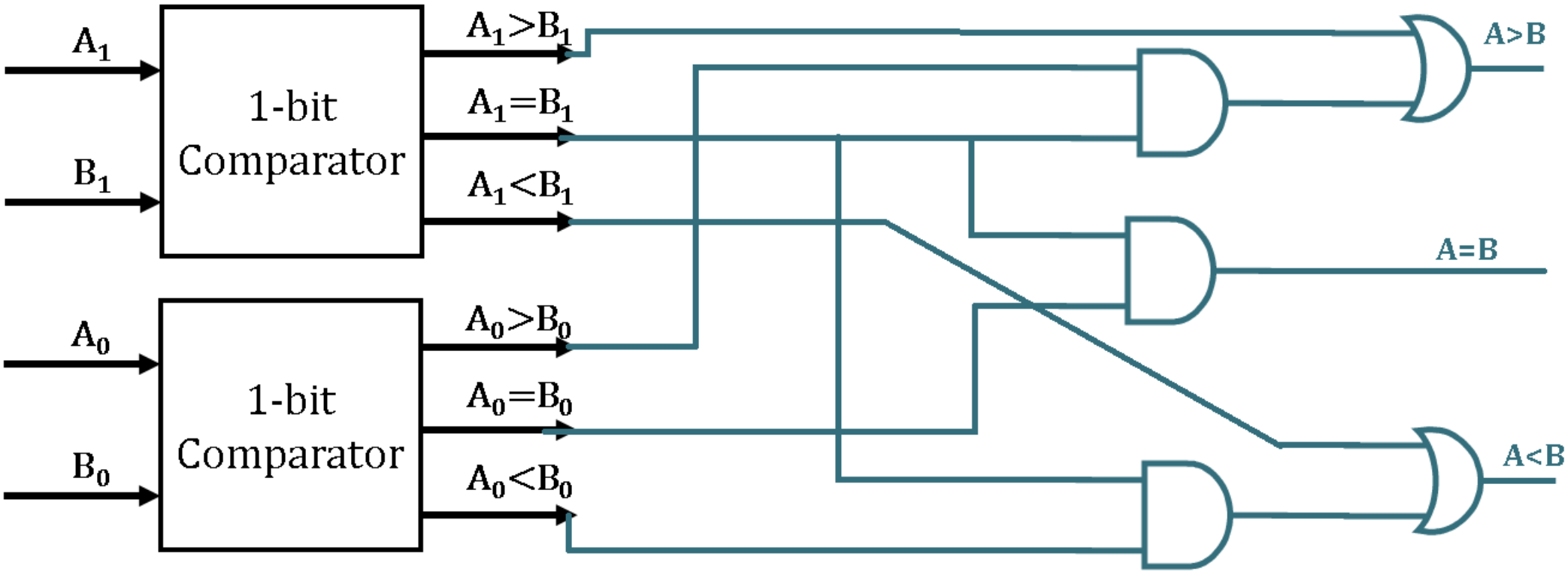
$$F_{A>B} = A\overline{B}$$





Digital Magnitude Comparator

- Two 1-bit comparators can be logically connected to make a 2-bit number comparator.
- The logic behind the connections are:
  - Let $\mathbf{X_0 = (A_0 == B_0)}$ and $\mathbf{X_1 = (A_1 == B_1)}$
  - For $\mathbf{A = B}$, the Boolean expression is X_1X_0
  - For $\mathbf{A > B}$, the Boolean expression is $\mathbf{A_1\overline{B_1} + X_1(A_0\overline{B_0})}$
  - For $\mathbf{A < B}$, the Boolean expression is $\mathbf{\overline{A_1}B_1 + X_1(\overline{A_0}B_0)}$



References

1. Thomas L. Floyd, “Digital Fundamentals” 11th edition, Prentice Hall – Pearson Education.

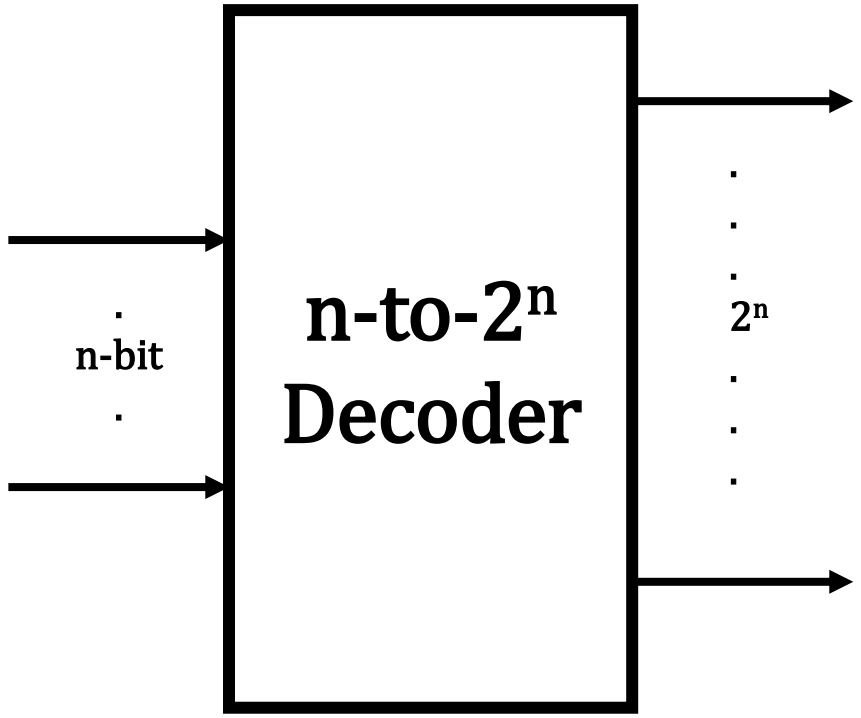
Thank You

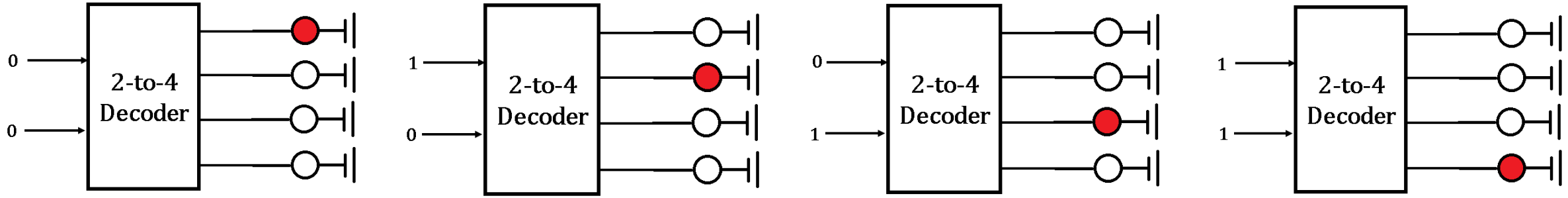
Lecture -6
Combination Circuits-2

Prepared By: Asif Mahfuz

Decoder

- A n-to-2^n decoder takes an n-bit input and produces 2^n outputs.
- The n-bit inputs represent a binary number that determines which of the 2^n outputs is **uniquely true**.
- As the name suggest the job of the decoder is to decode.
- For instance, if you ask the guard of a building, which floor does Mr. Y lives? The guard gives a specific answer based on your question.
- A decoder is commonly used in address decoding.





Decoder Design

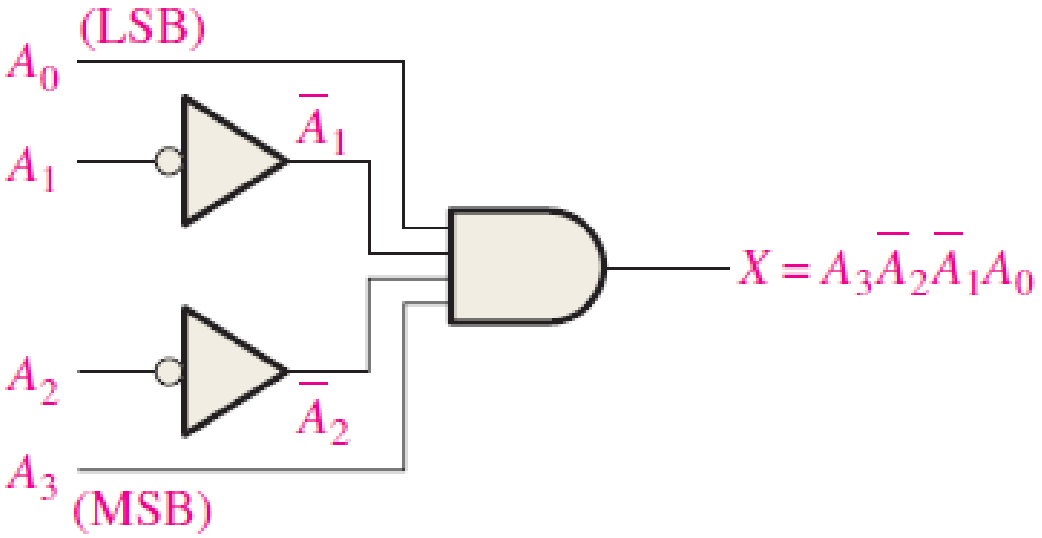
- We have now a superficial idea of how a decoder works.
- A 2-to-4 decoder has two inputs namely, **S0** and **S1** and 4 outputs namely **Q0**, **Q1**, **Q2** and **Q3**.
- The 2-to-4 decoder operates according to the following truth table.
- So how is this decoding done?
- Implementing the min-term for a specific combination is the answer.

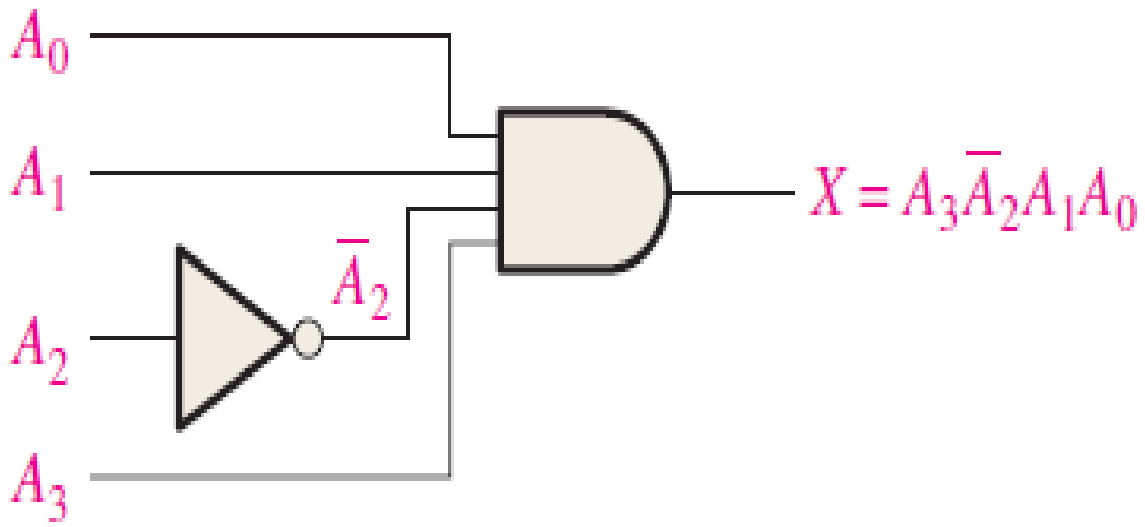
S1	S0	Q0	Q1	Q2	Q3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

S1	S0	Q0	Q1	Q2	Q3
0	0	$\overline{S1}\,\overline{S0}$	0	0	0
0	1	0	$\overline{S1}\,S0$	0	0
1	0	0	0	$S1\,\overline{S0}$	0
1	1	0	0	0	$S1\,S0$

Decoding Operation

- So how do we implement the decoding function.
- Say we want to decode a combination **1001**.
- The min-term of this combination is $\mathbf{A_3\overline{A_2}\,\overline{A_1}A_0}$.
- Now we implement this min-term.
- Now say we want to decode **1011**.
- We need to implement the min-term $\mathbf{A_3\overline{A_2}A_1A_0}$.

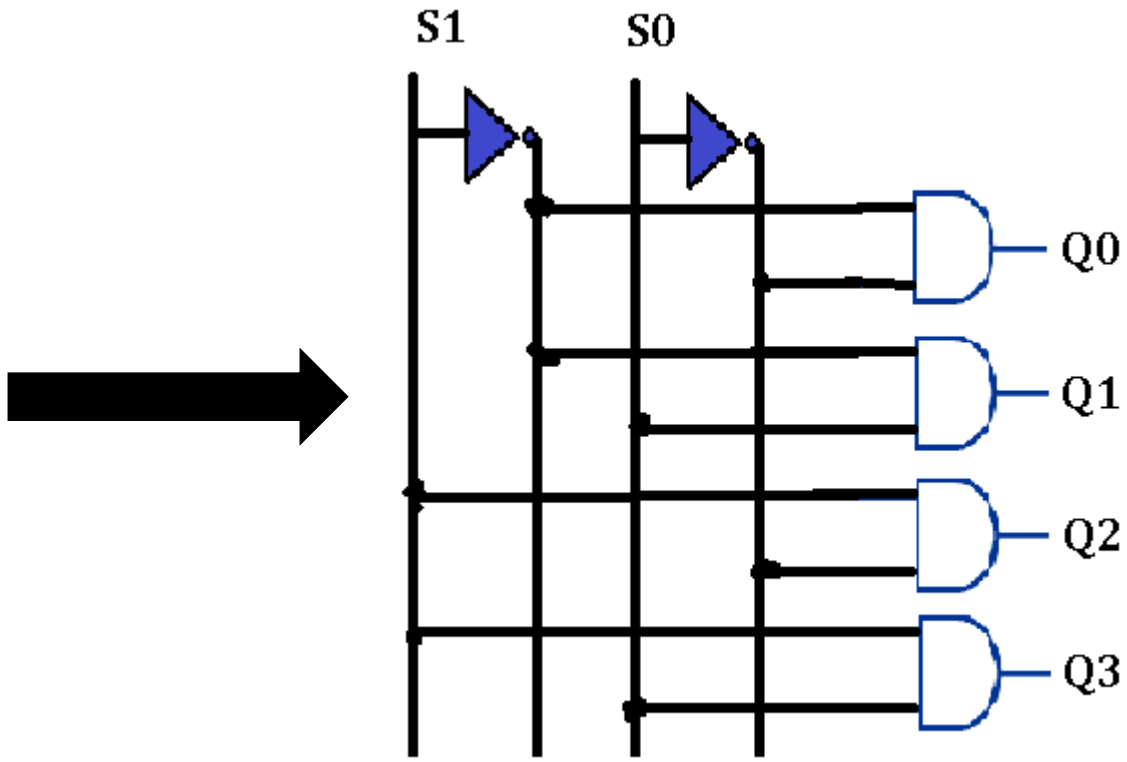




Logical Circuit of 2-to-4 Decoder

- So now that we know how to decode a combination, we can draw the logical circuit of the a 2-to-4 Decoder.
- The truth table of the decoder is as follows.
- From the truth table we can draw the logical circuit.
- As each output is implemented from a min-term, an **Active HIGH** decoder is also called a min term generator.

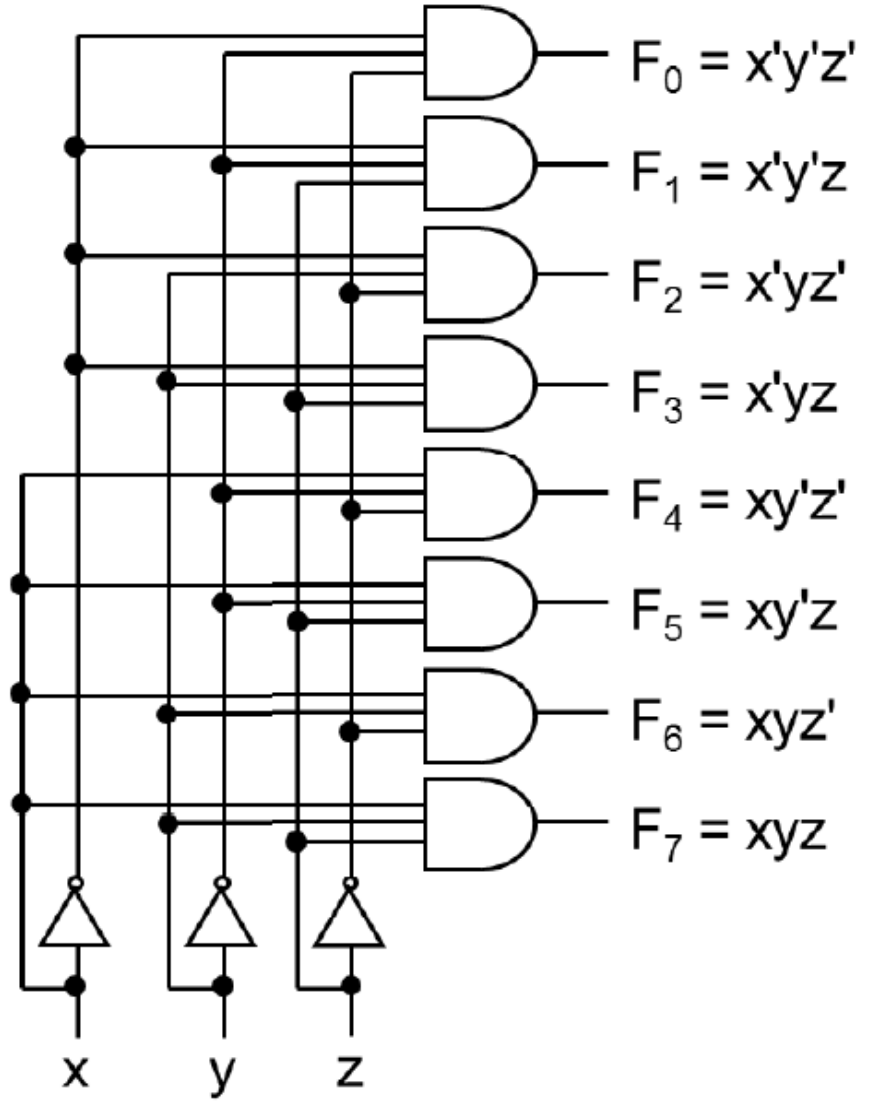
S1	S0	Q0	Q1	Q2	Q3
0	0	$\overline{S1}\,\overline{S0}$	0	0	0
0	1	0	$\overline{S1}\,S0$	0	0
1	0	0	0	$S1\,\overline{S0}$	0
1	1	0	0	0	$S1\,S0$



Design of a 3-to-8 Decoder

- We can design a 3-to-8 decoder following the same procedure.

X	Y	Z	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



Variation of Decoder

- Till now we have only seen the active-HIGH decoders.
- That is, when a combination is given in the input the corresponding output is HIGH and the other outputs are LOW.
- A very common variation of decoders is the active-LOW decoders.
- This means when a combination is given in the input, the corresponding output is LOW and the other outputs are HIGH.
- The same procedure can be followed to design an active-LOW decoder, with the only difference of using a NAND gate instead of the AND gate for decoding.
- Thus active-LOW decoders are also called Max-Term generators.

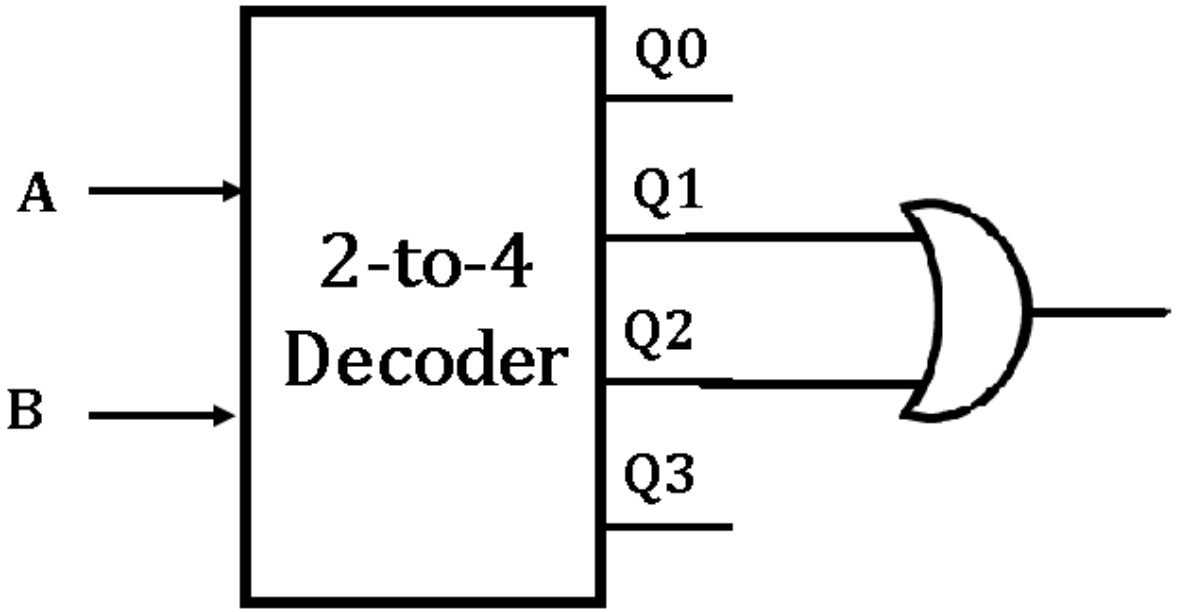
S1	S0	Q0	Q1	Q2	Q3
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

S1	S0	Q0	Q1	Q2	Q3
0	0	(S1+S0)	1	1	1
0	1	1	$(S1+\overline{S0})$	1	1
1	0	1	1	$(\overline{S1}+S0)$	1
1	1	1	1	1	$(\overline{S1}+\overline{S0})$

Application of Decoders

- As we have already studied, one of the application of decoders is addressing or selecting memory locations.
- Another very important use of decoders is to implement functions.
- As active-HIGH decoders are min-term generators and active-LOW decoders are max-term generators the SOP form and POS form can be easily implemented with decoders.
- The procedure is very simple, if we have the truth-table of the function or can somehow determine the truth-table, function implementation will be very easy.

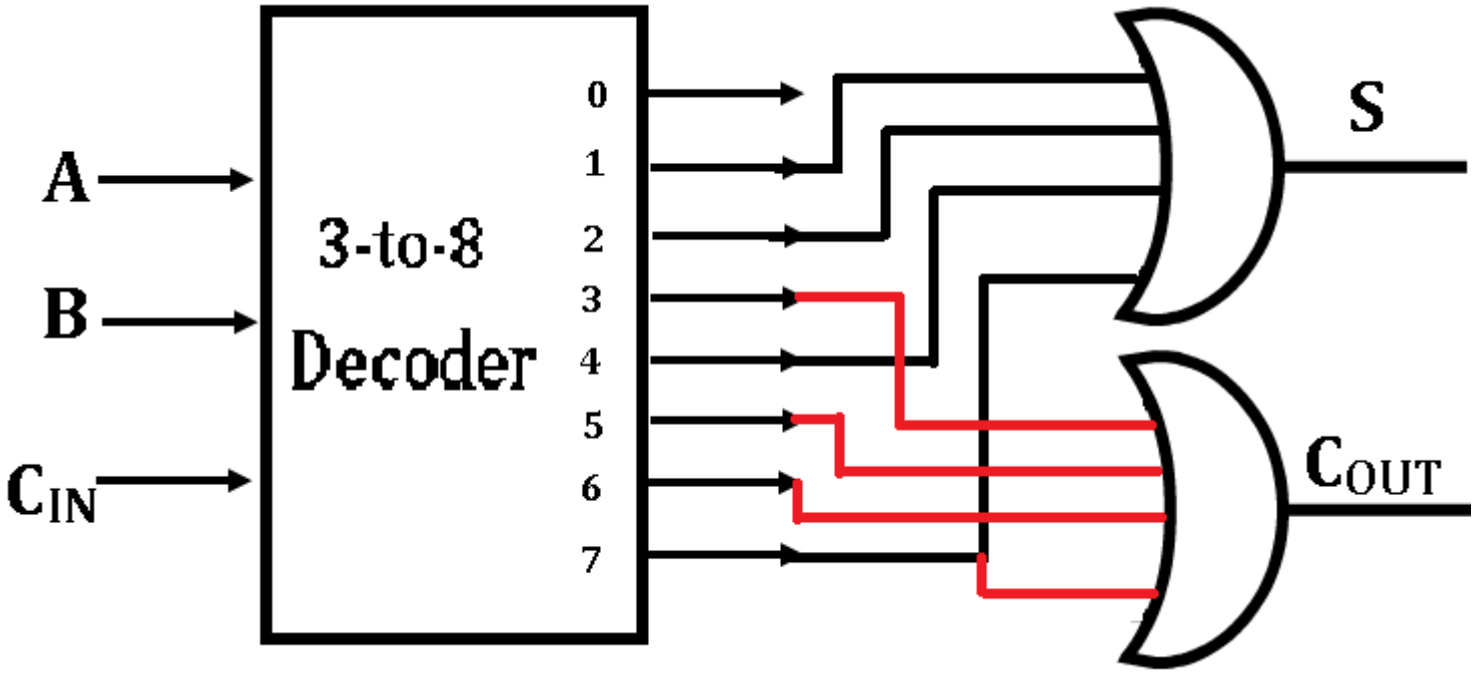
A	B	F
0	0	0
0	1	1
1	0	1
1	1	0



Application of Decoders

- A full-adder can be very easily implemented with the help of a decoder.
- The truth-table of a full adder is given below:
- The full adder can be implemented as shown in the diagram.

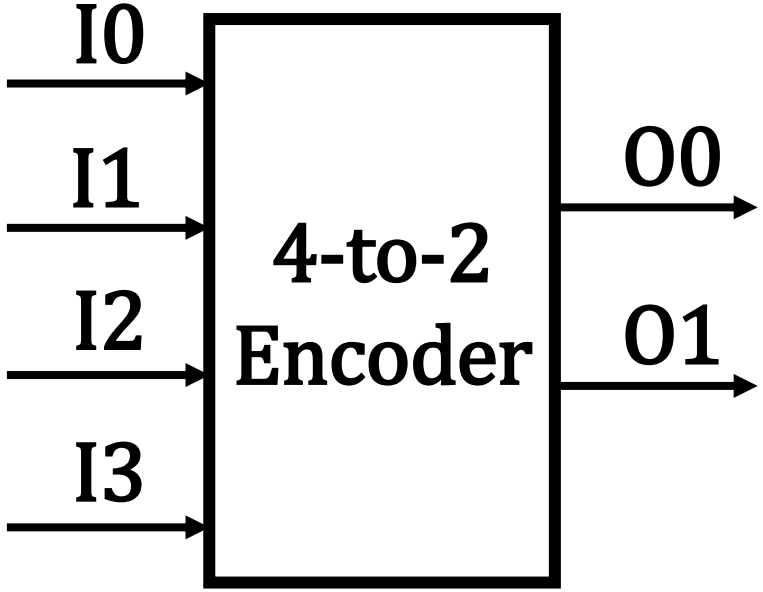
A	B	C_{IN}	C_{out}	$\sum$
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

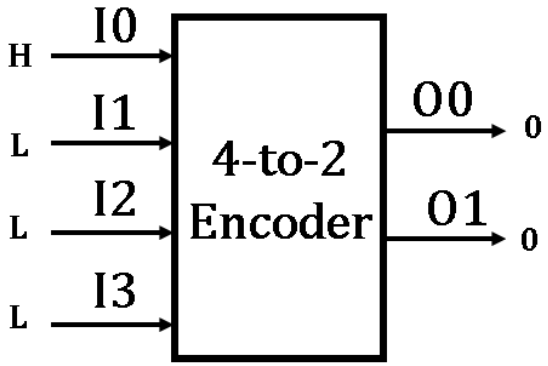


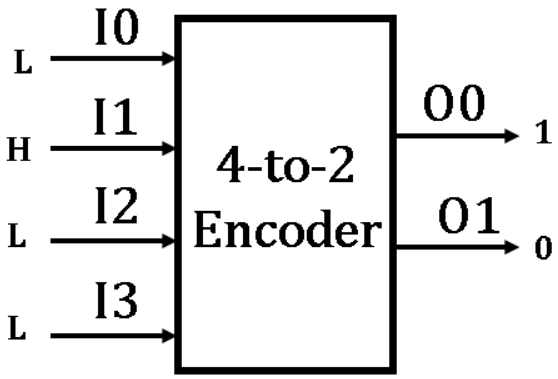
- Implement the function $F = \sum(1,3,4,7)$ using a 3-to-8 decoder.

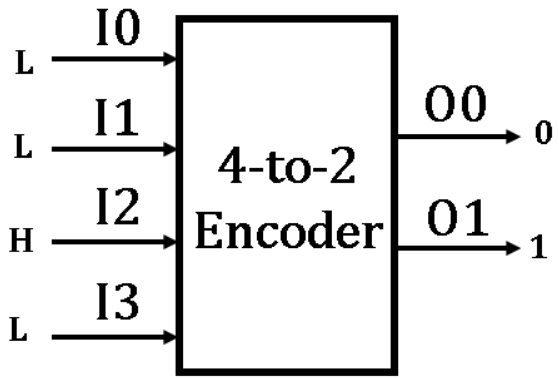
Encoders

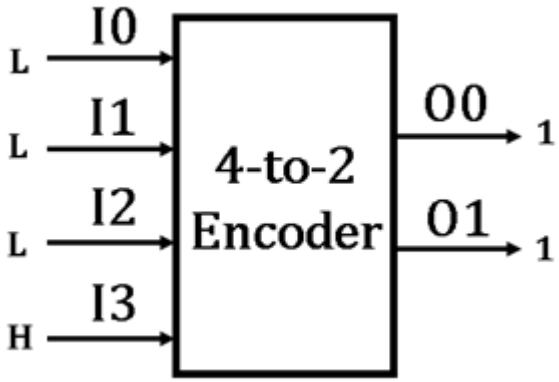
- An Encoder is a combinational logic circuit that performs a “reverse” decoder functions.
- As the name suggests its function is to encode an input signal.
- An encoder accepts an active level on one of its input and outputs the coded output for the input.
- Generally an encoder has 2^n inputs and **n**-outputs.
- There can be a variety of encoders, but we will only keep ourselves to binary encoders.











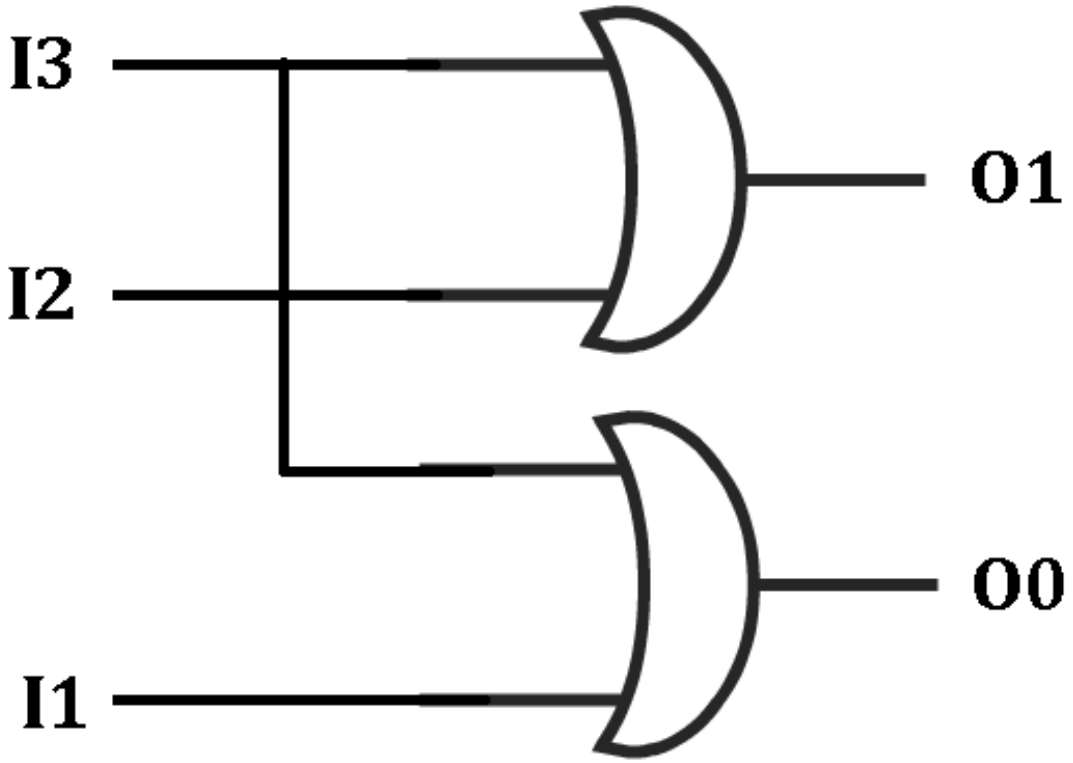
4-to-2 Encoder Design

- We now have a superficial idea of how the encoder works.
- So now we can form the truth table and finally design the encoder.

I3	I2	I1	I0	O1	O0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

- Since we have only these 4-combinations in the ideal case, the Boolean expressions are:

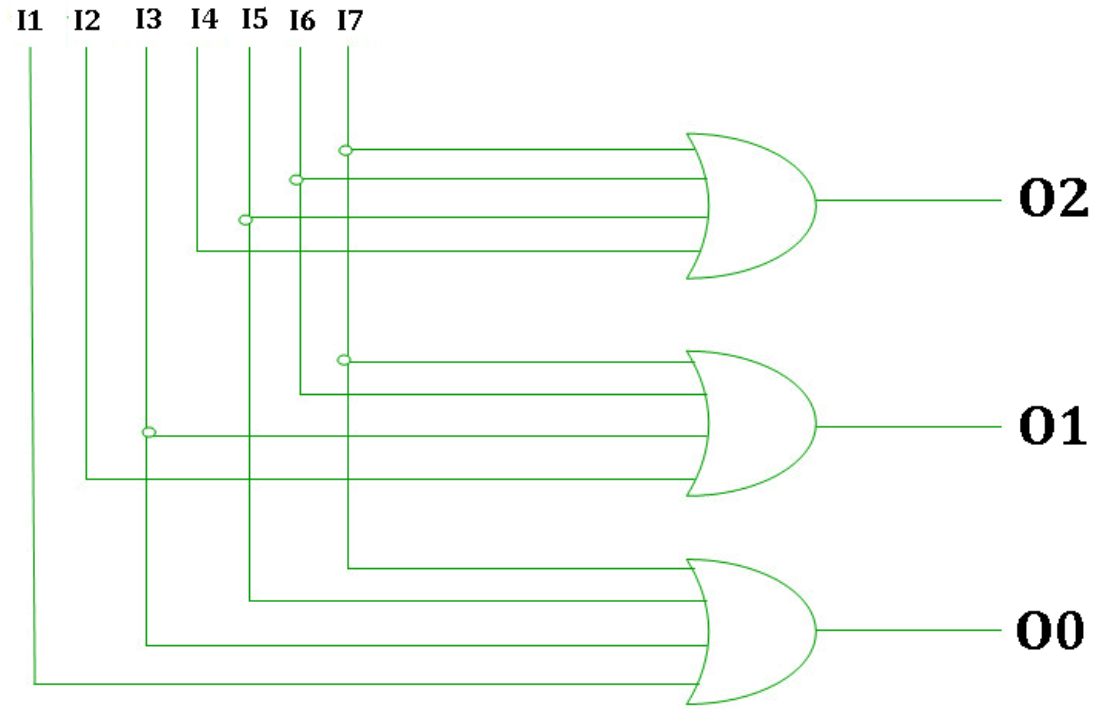
$$O0 = I1 + I3$$
$$O1 = I2 + I3$$



8-to-3 Encoder Design

- An 8-to-3 encoder can be designed in the same way.

I7	I6	I5	I4	I3	I2	I1	I0	O2	O1	O0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

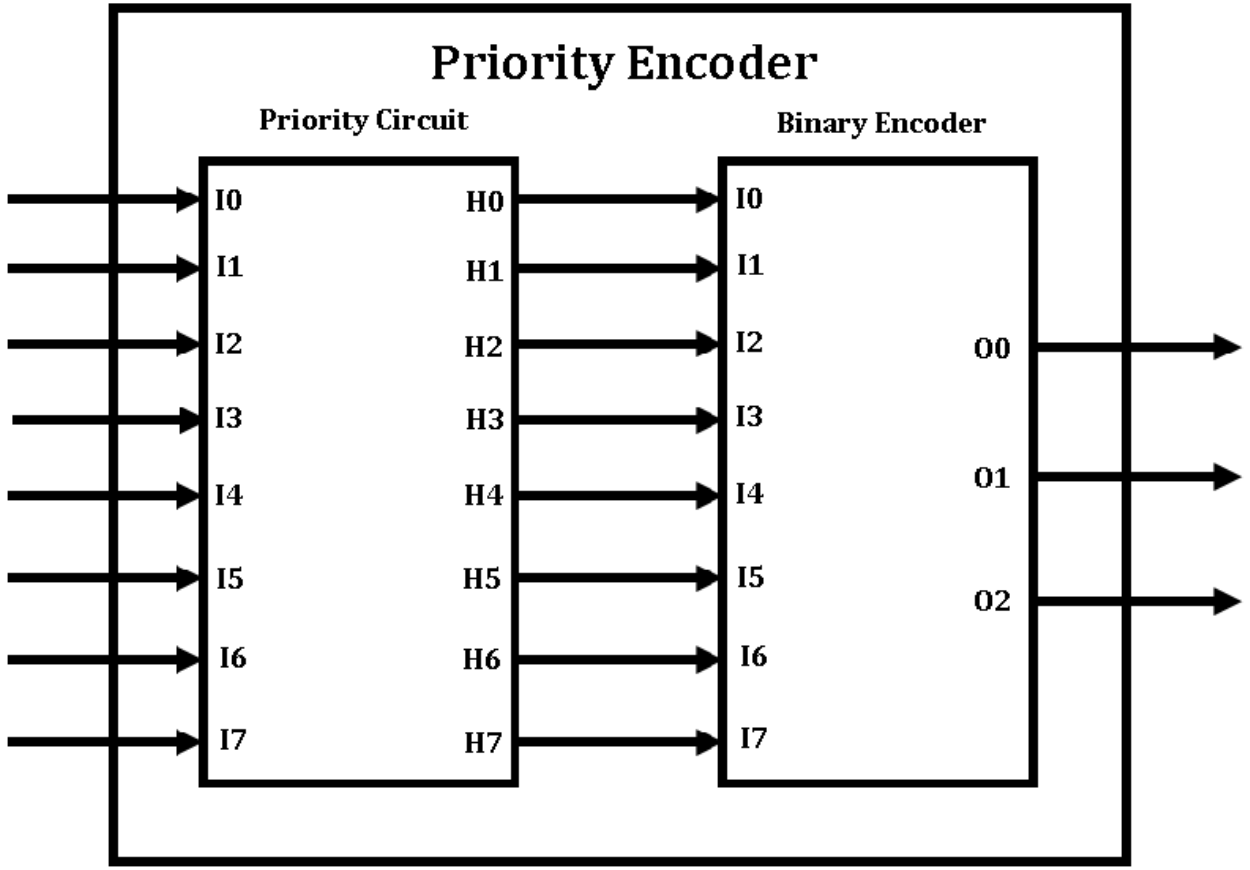


- Boolean expression:

$$O0 = I1 + I3 + I5 + I7$$
$$O1 = I2 + I3 + I6 + I7$$
$$O2 = I4 + I5 + I6 + I7$$

Priority Encoders

- We have designed the encoders for an ideal scenario, where only 1 input is active at time.
- However, in real world scenario, this will always not be the case.
- Hence, we need some modifications to be made.
- The solution to this problem is to define priority amongst the inputs.
- The priority circuit will detect the input with highest priority and output that signal only to the binary encoder.
- The encoder will then encode the received input.
- So lets us learn how to design a priority circuit.



Designing the Priority Circuit

- We now know the purpose of the priority circuit.
- So how to design the priority circuit.
- First, we need to know the priority of the inputs. So for our designing example let the priority be, I0>I1>I2>I3>I4>I5>I6>I7.
- Now, we need to have the truth table of the priority circuit.

I7	I6	I5	I4	I3	I2	I1	I0	H7	H6	H5	H4	H3	H2	H1	H0
X	X	X	X	X	X	X	1	0	0	0	0	0	0	0	1
X	X	X	X	X	X	1	0	0	0	0	0	0	0	1	0
X	X	X	X	X	1	0	0	0	0	0	0	0	1	0	0
X	X	X	X	1	0	0	0	0	0	0	0	1	0	0	0
X	X	X	1	0	0	0	0	0	0	0	1	0	0	0	0
X	X	1	0	0	0	0	0	0	0	1	0	0	0	0	0
X	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

Designing the Priority Circuit

I7	I6	I5	I4	I3	I2	I1	I0	H7	H6	H5	H4	H3	H2	H1	H0
X	X	X	X	X	X	X	1	0	0	0	0	0	0	0	1
X	X	X	X	X	X	1	0	0	0	0	0	0	0	1	0
X	X	X	X	X	1	0	0	0	0	0	0	0	1	0	0
X	X	X	X	1	0	0	0	0	0	0	0	1	0	0	0
X	X	X	1	0	0	0	0	0	0	0	1	0	0	0	0
X	X	1	0	0	0	0	0	0	0	1	0	0	0	0	0
X	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

Boolean Expression for the priority circuit:

- $H0 = I0$
- $H1 = I1\overline{I0}$
- $H2 = I2\ \overline{I1}\ \overline{I0}$
- $H3 = I3\ \overline{I2}\ \overline{I1}\ \overline{I0}$
- $H4 = I4\ \overline{I3}\ \overline{I2}\ \overline{I1}\ \overline{I0}$
- $H5 = I5\ \overline{I4}\ \overline{I3}\ \overline{I2}\ \overline{I1}\ \overline{I0}$
- $H6 = I6\ \overline{I5}\ \overline{I4}\ \overline{I3}\ \overline{I2}\ \overline{I1}\ \overline{I0}$
- $H7 = I7\overline{I6}\ \overline{I5}\ \overline{I4}\ \overline{I3}\ \overline{I2}\ \overline{I1}\ \overline{I0}$

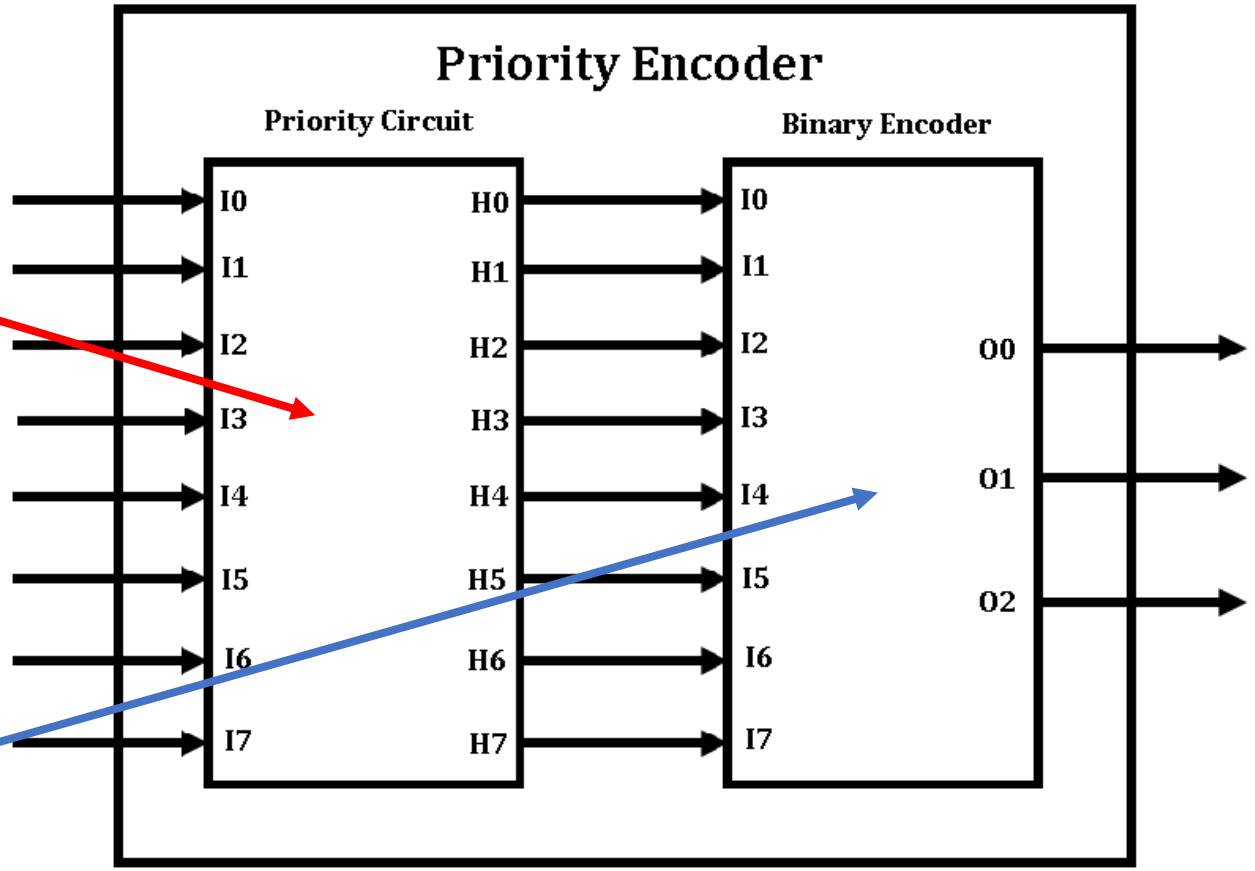
Now that we have assured the ideal scenario, we can just use the same logic for the encoder design.

- $O0 = I1 + I3 + I5 + I7$
- $O1 = I2 + I3 + I6 + I7$
- $O2 = I4 + I5 + I6 + I7$

Designing the Priority Circuit

Boolean Expression for the priority circuit:

- $H0 = I0$
- $H1 = I1\overline{I0}$
- $H2 = I2\ \overline{I1}\ \overline{I0}$
- $H3 = I3\ \overline{I2}\ \overline{I1}\ \overline{I0}$
- $H4 = I4\ \overline{I3}\ \overline{I2}\ \overline{I1}\ \overline{I0}$
- $H5 = I5\ \overline{I4}\ \overline{I3}\ \overline{I2}\ \overline{I1}\ \overline{I0}$
- $H6 = I6\ \overline{I5}\ \overline{I4}\ \overline{I3}\ \overline{I2}\ \overline{I1}\ \overline{I0}$
- $H7 = I7\overline{I6}\ \overline{I5}\ \overline{I4}\ \overline{I3}\ \overline{I2}\ \overline{I1}\ \overline{I0}$



Boolean Expression for the priority circuit:

- $O0 = I1 + I3 + I5 + I7$
- $O1 = I2 + I3 + I6 + I7$
- $O2 = I4 + I5 + I6 + I7$

Priority Encoders with Irregular Sequence

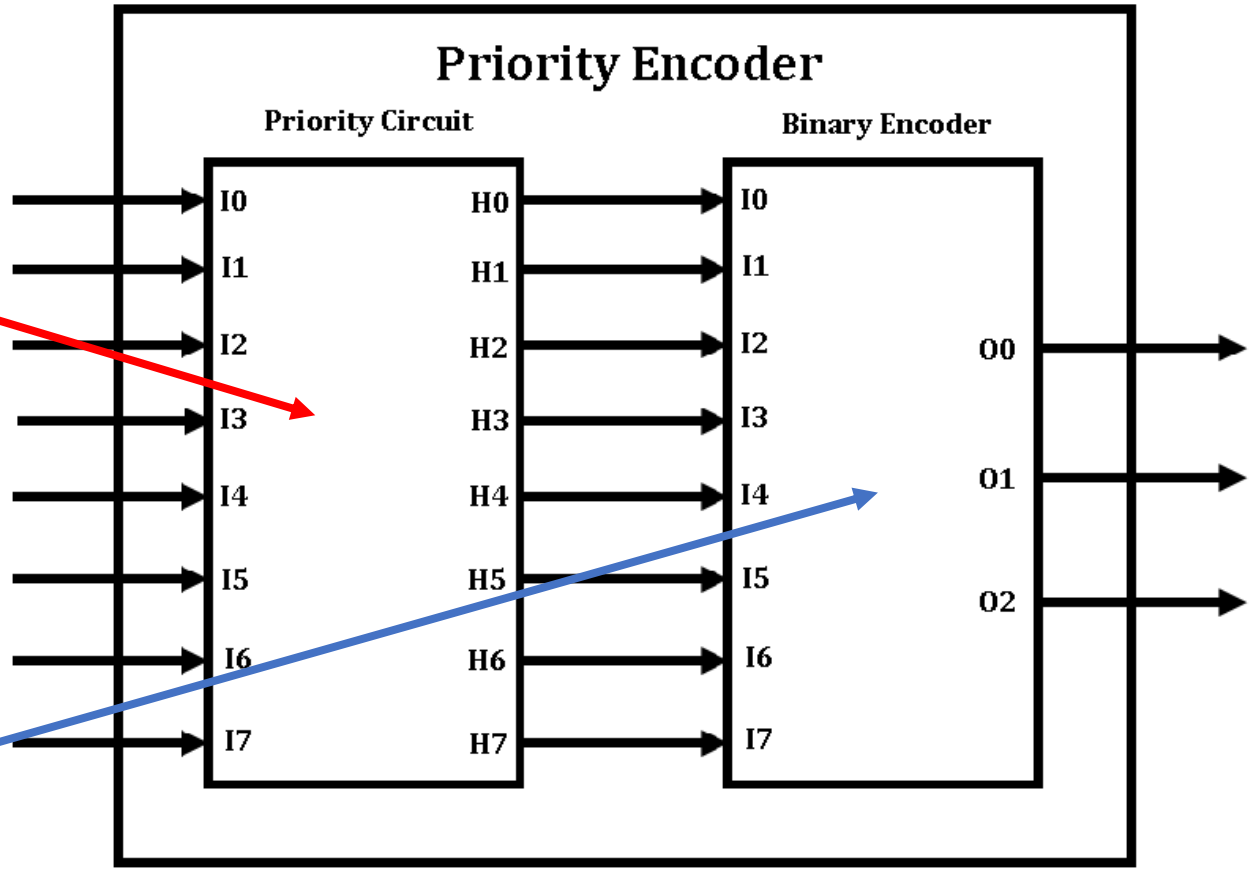
- Priority sequence of an encoder might not always be with a regular priority sequence such as ascending priority or descending priority.
- The priority for sequence might be irregular.
- An example can be: **I0>I1>I5>I6>I7>I3>I2>I4.**
- Designing procedure for this will be the same.
- So first we need the truth-table.

I7	I6	I5	I4	I3	I2	I1	I0	H7	H6	H5	H4	H3	H2	H1	H0
X	X	X	X	X	X	X	1	0	0	0	0	0	0	0	1
X	X	X	X	X	X	1	0	0	0	0	0	0	0	1	0
0	0	0	X	0	1	0	0	0	0	0	0	0	1	0	0
0	0	0	X	1	X	0	0	0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0
X	X	1	X	X	X	0	0	0	0	1	0	0	0	0	0
X	1	0	X	X	X	0	0	0	1	0	0	0	0	0	0
1	0	0	X	X	X	0	0	1	0	0	0	0	0	0	0

Priority Encoders with Irregular Sequence

Boolean Expression for the priority circuit:

- $H0 = I0$
- $H1 = I1\overline{I0}$
- $H2 = I2\ \overline{I5}\ \overline{I6}\ \overline{I7}\ \overline{I3}\ \overline{I1}\ \overline{I0}$
- $H3 = I3\ \overline{I7}\ \overline{I6}\ \overline{I5}\ \overline{I1}\ \overline{I0}$
- $H4 = I4\ \overline{I2}\ \overline{I3}\ \overline{I7}\ \overline{I6}\ \overline{I5}\ \overline{I1}\ \overline{I0}$
- $H5 = I5\ \overline{I1}\ \overline{I0}$
- $H6 = I6\ \overline{I5}\ \overline{I1}\ \overline{I0}$
- $H7 = I7\overline{I6}\ \overline{I5}\ \overline{I1}\ \overline{I0}$

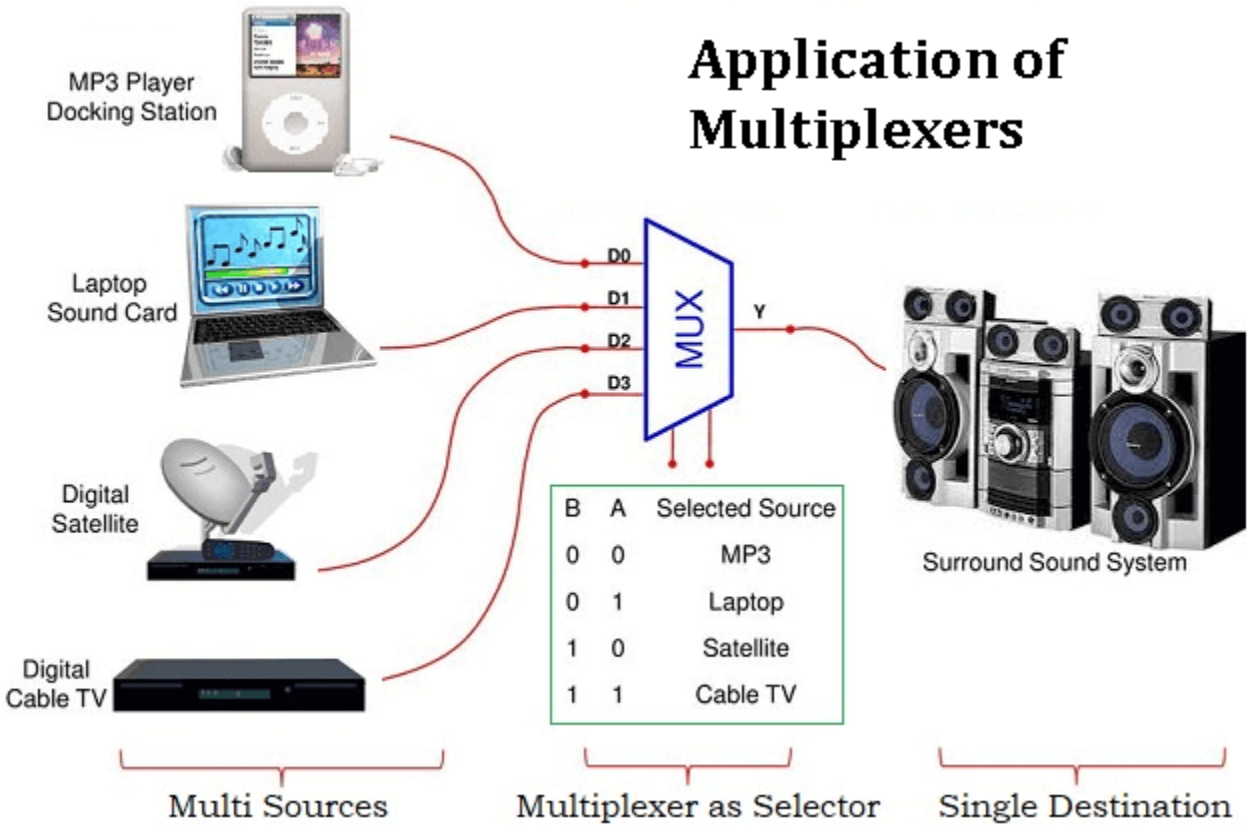


Boolean Expression for the priority circuit:

- $O0 = I1 + I3 + I5 + I7$
- $O1 = I2 + I3 + I6 + I7$
- $O2 = I4 + I5 + I6 + I7$

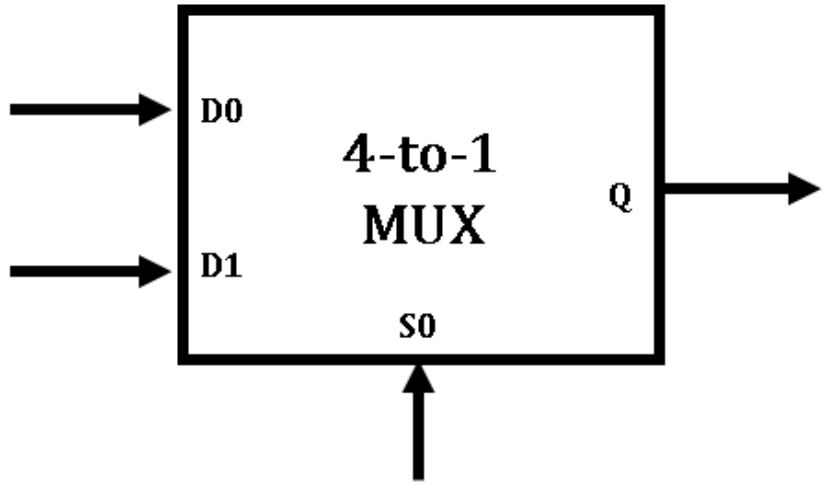
Multiplexers (MUX)

- In old days, multiple devices could be connected to a single device with the help of a switch.
- For example, multiple sources of input could be used as an input to a single home theater.
- The device that acts as switch is known as the multiplexer.
- What we see in the picture is a 4-to-1MUX.
- A 4-to-1 MUX has 2 selector pins.
- Therefore a 2^n-to-1 MUX will have n selector pins.
- As for our example we can see:
  - When selector pins are **00** the MP3 will be selected.
  - When the selector pins are **01** the laptop will be selected.
  - When the selector pin are 10 the digital satellite is selected and so on.



Designing a Multiplexers (MUX)

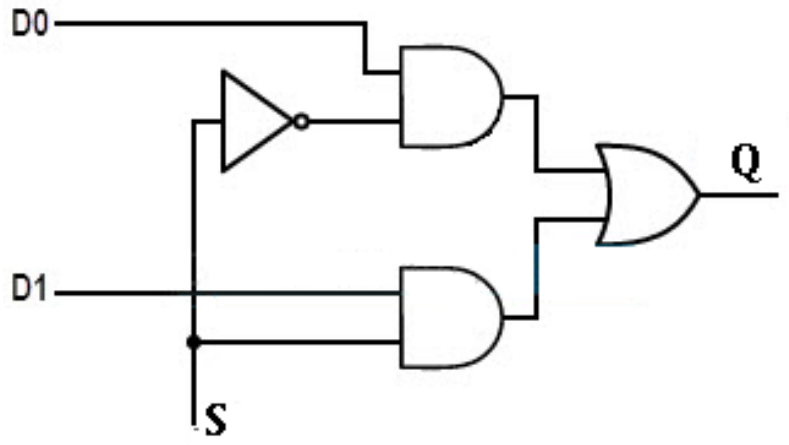
- So how is a MUX designed?
- MUX is a digital combinational circuit.
- Thus we need to first find its truth table.
- Let us for example take a 2-to-1 MUX.
- A 2-to-1 MUX will be 2 inputs to select from using a 1 selector pin.



S	D1	D0	Q
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Boolean expression for a MUX:

- $Q = \bar{S}\,\overline{D1}D0 + \bar{S}D1D0 + SD1\overline{D0} + SD1D0$
- $Q = \bar{S}D0 + SD1$



Designing a Multiplexers (MUX)

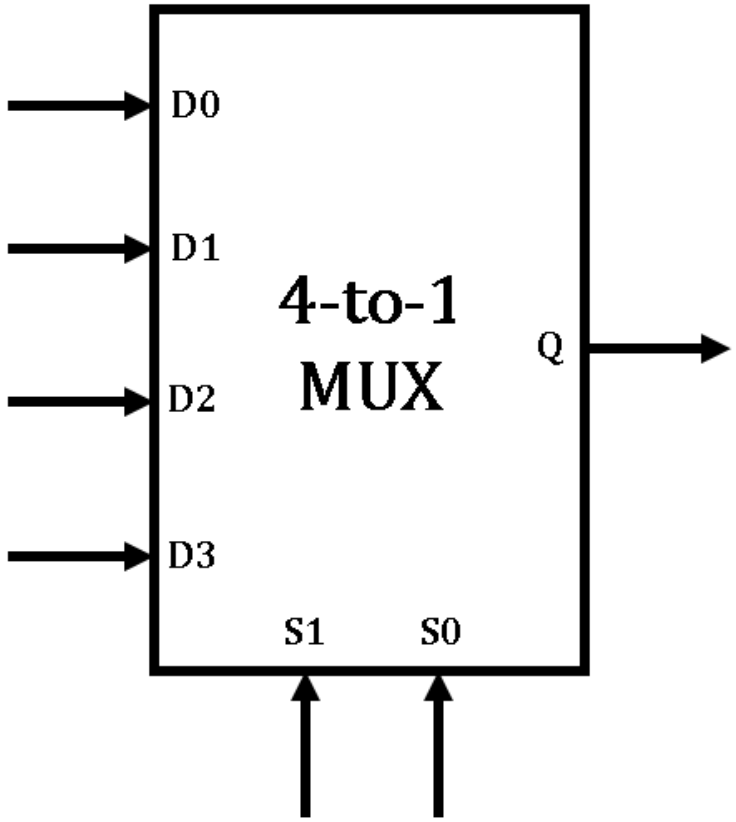
- Let us now design a 4-to-1 MUX.
- Since there are 6 inputs in total the truth table will become very big with 64 combinations.
- We can have the truth table, then find the min terms as our previous example, but this will surely become very tedious.
- So we can probably find a trick if we look at Boolean expression of our last design

$$\mathbf{Q = \bar{S}D0 + SD1}$$

- Seeing this equation we can find a shortcut to write the equation for 4-to-1 MUX as:

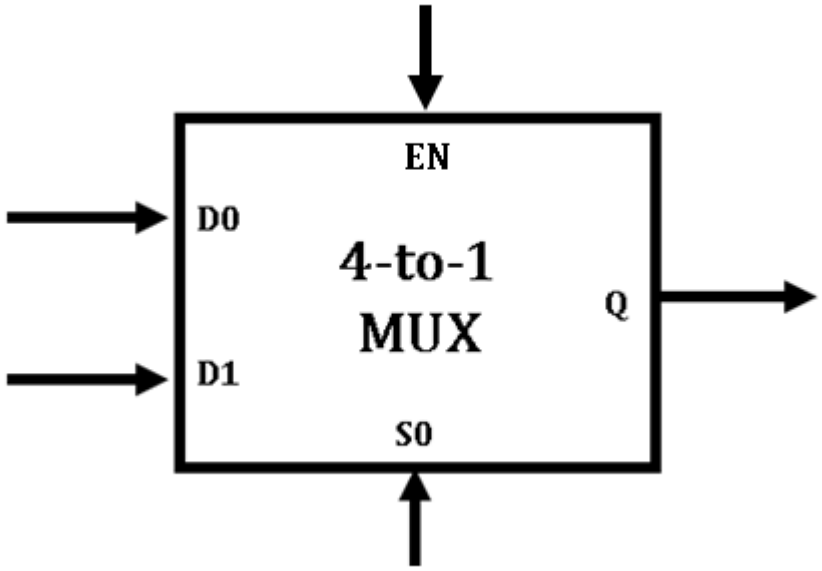
$$\mathbf{Q = \overline{S1S0}D0 + \overline{S1}S0D1 + S1\overline{S0}D2 + S1S2D3}$$

- After finding the Boolean expression we can draw the logical circuit of the 4-to-1 MUX



Enable Input

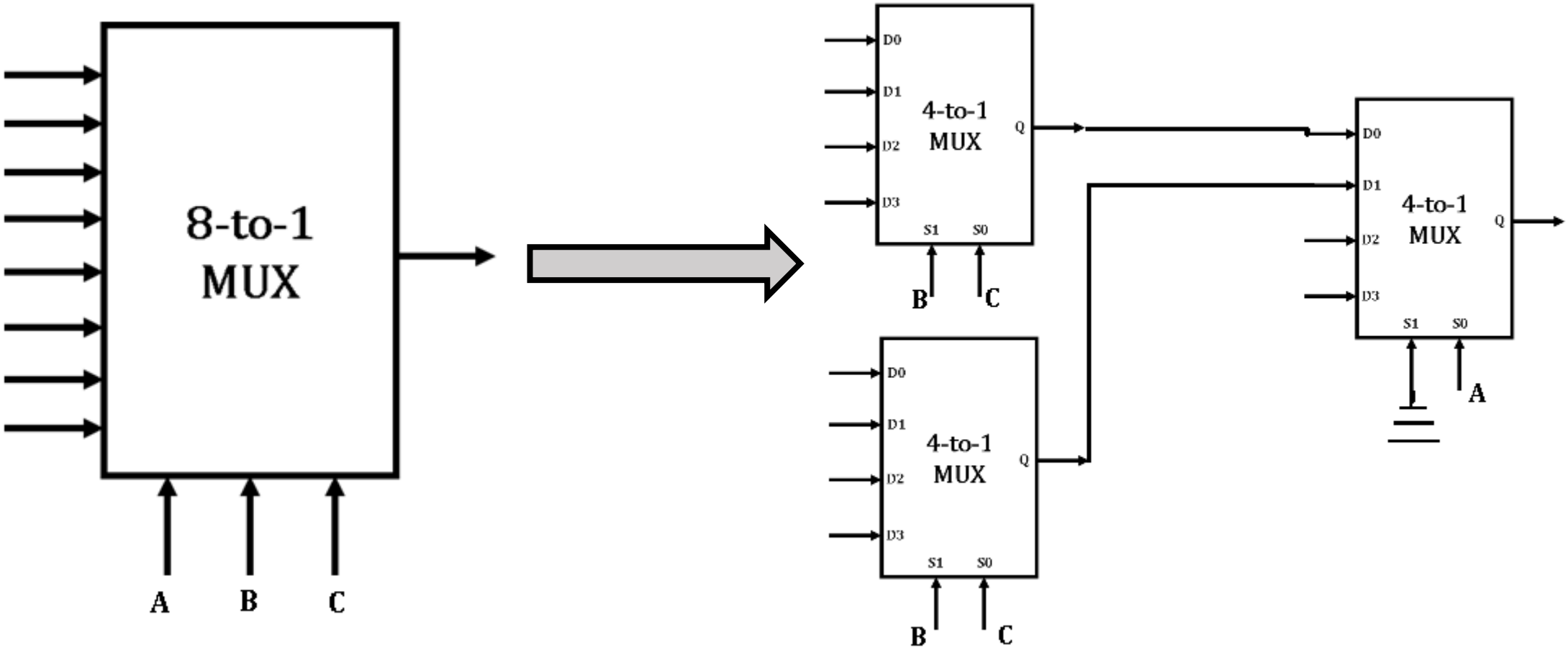
- Many devices have an extra input known as the “enable” input.
- This is used to activate or deactivate a device.
- Enables are especially useful in combining smaller MUXs to make larger MUXs.
- So how does the truth table vary with inclusion go enable.
- When enable is 0, the output is 0 irrespective of all other inputs.
- And when enable is 1, the MUX operates as specified earlier.



En	S	D1	D0	Q
0	X	X	X	0
1	0	0	0	0
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	1

Modularity

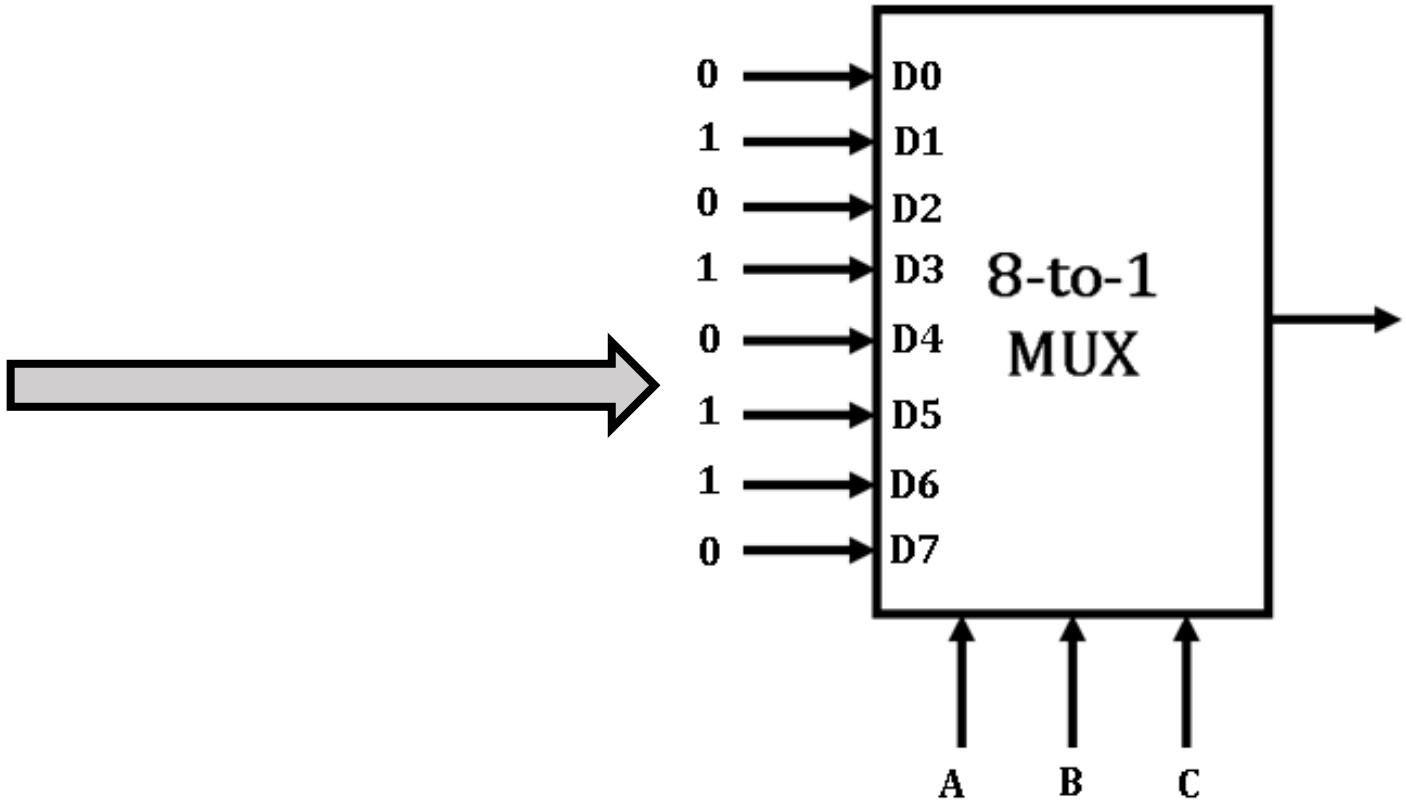
- It is not always possible to get a MUX with the exact number of required inputs.
- Say we need to connect 6 devices and we have only access to 4-to-1 MUXs.
- In such cases we can use smaller MUXs and combine them make larger MUXs.
- So as an example we will see how to use 4-to-1 MUX to make 8-to-1 MUX.



Implementing Functions with MUX

- MUX can also be used to implement Boolean functions.
- A n-variable function can be implemented using a 2^n-to-1 MUX (very easy).
- However, we can also implement a (n+1) variable function with 2^n-to-1 MUX (advanced).
- Let us see how we can implement the function $F(A, B, C) = \sum(1,3,5,6)$ using an 8-to-1 MUX.

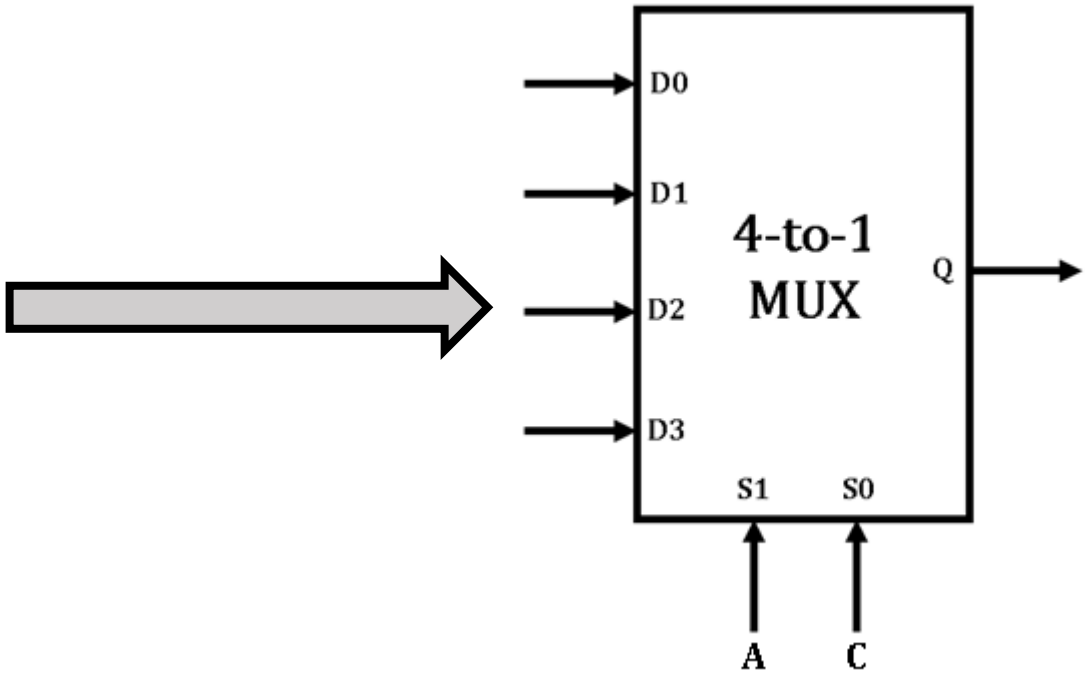
A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0



Implementing Functions with MUX

- Now let us implement the same function using a 4-to-1 MUX.
- First, we need to construct the truth table.
- Then we need to select the variables of the function we want to use as selector pin.
- Let us take A and C as selector pin.
- The remaining variable of B will be used in a different way.

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



Implementing Functions with MUX

- Now we will construct a special table for variable B from the truth-table.
- First, we need to find the min-terms for which B is 0. The min-terms are 0,1,4 and 5.
- Then, we need to find the min-terms for which B is 1. The min-terms are 2,3,6 and 7.
- Now list the input of the 4-to-1 MUX. The inputs are D0, D1, D2 and D3.
- Now create a table as shown

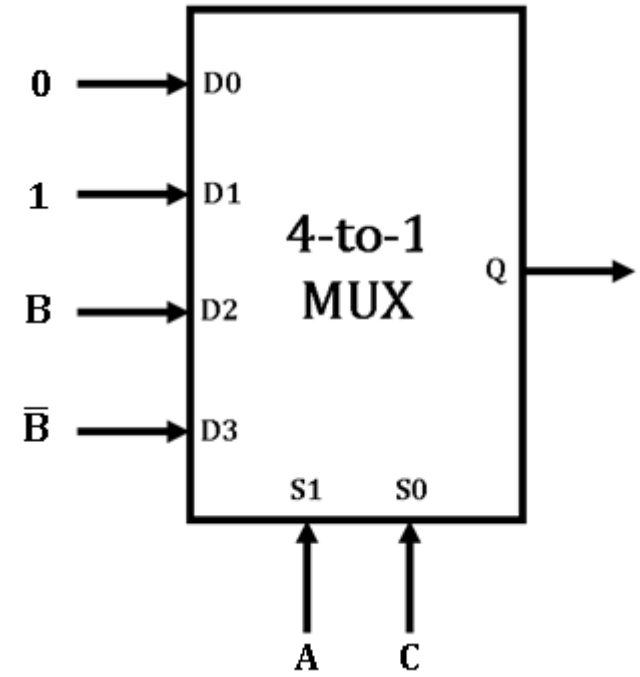
Minterms	A	B	C	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

	D0	D1	D2	D3
$\overline{B}$	0	1	4	5
B	2	3	6	7

Implementing Functions with MUX

- Now circle all the min-terms for which the function is 1.
- Now if two min-terms in a column is not circled, we apply a 0 for the input.
- Now if two min-terms in a column are circled, we apply 1 for the input.
- If there is one circle in a column, and the circle is the row of $\overline{B}$, we apply $\overline{B}$ for the input. And if the only circle is in the row of B, we apply B for the input.

	D0	D1	D2	D3
$\overline{B}$	0	(1)	4	(5)
B	2	(3)	(6)	7
	0	1	B	$\overline{B}$



Implementing Functions with MUX

- Implement the following function with an 8-to-1 MUX with B,C and D as selector pins.

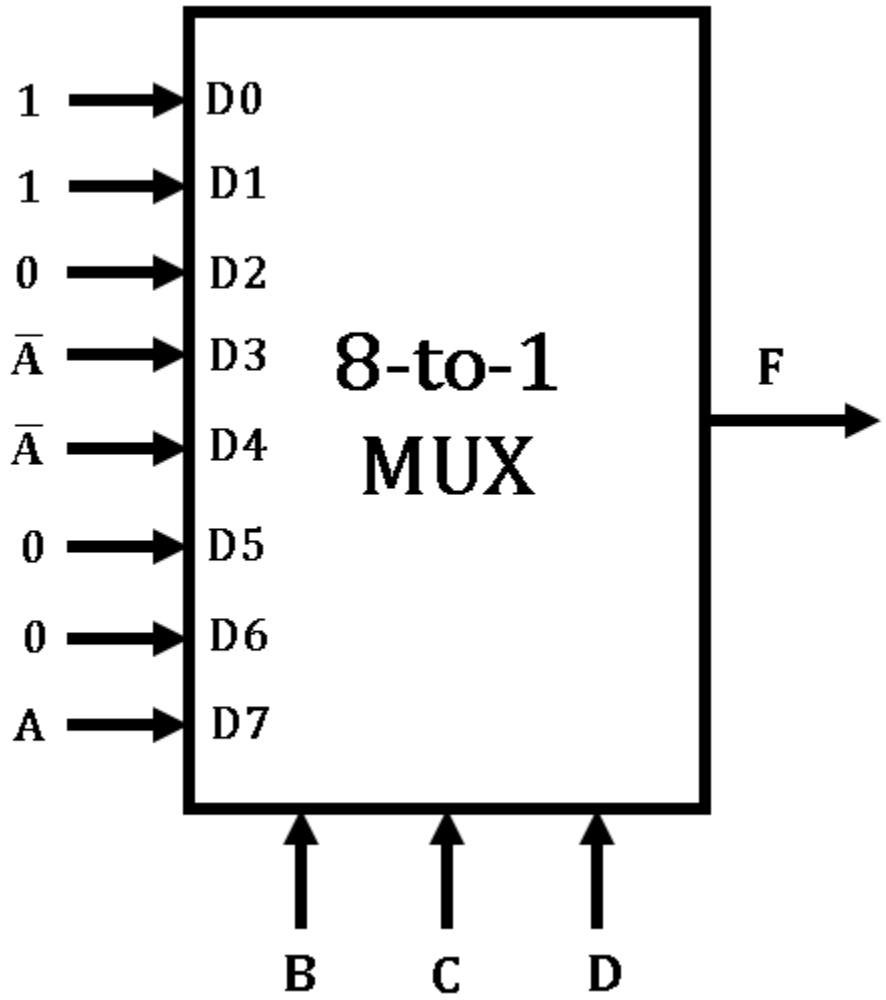
$$F(A, B, C, D) = \sum(0, 1, 3, 4, 8, 9, 15)$$

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

	D0	D1	D2	D3	D4	D5	D6	D7
$\bar{A}$	0	1	2	3	4	5	6	7
A	8	9	10	11	12	13	14	15
	1	1	0	$\bar{A}$	$\bar{A}$	0	0	A

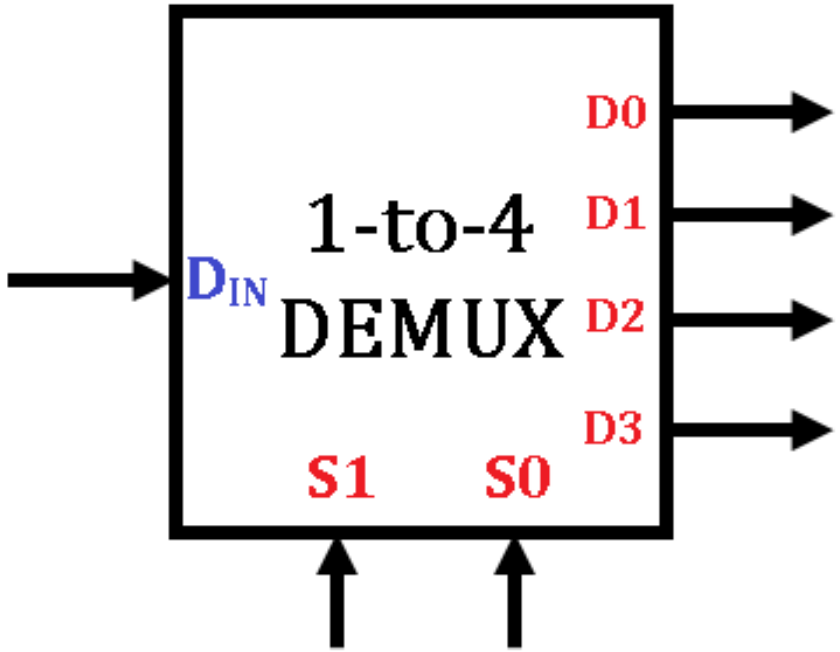
Implementing Functions with MUX

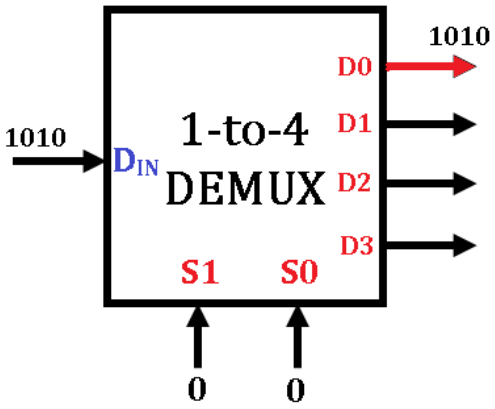
	D0	D1	D2	D3	D4	D5	D6	D7
$\bar{A}$	0	1	2	3	4	5	6	7
A	8	9	10	11	12	13	14	15
	1	1	0	$\bar{A}$	$\bar{A}$	0	0	A

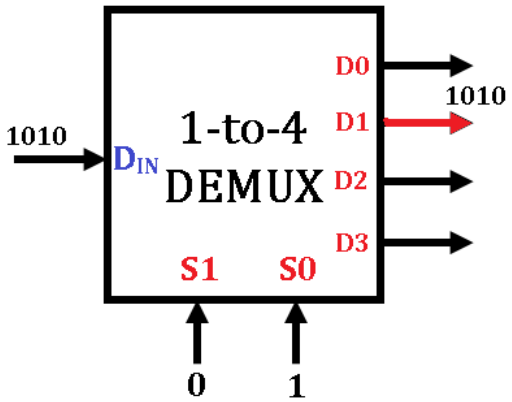


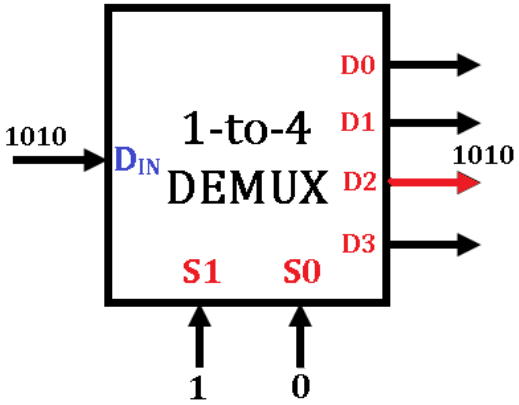
Demultiplexer (DEMUX)

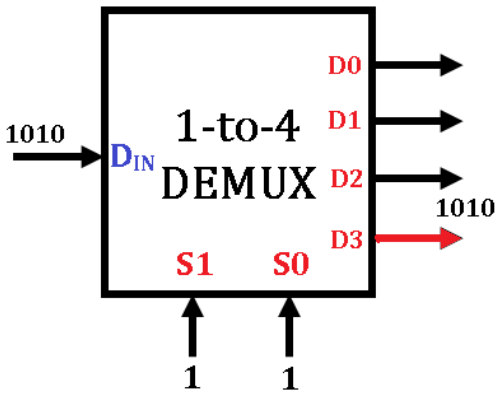
- A demultiplexer is a combinational circuit.
- It just reverses the operation of a multiplexer.
- It basically takes digital information from one line and distribute it to a given number of output lines.
- For this reason, the demultiplexer is also know as a data distributor.
- A 1-to-2^n DEMUX has n selector pins.
- The selector pin are used to select the output lines where the data is to be distributed.
- The figure shows a 1-to-4 demultiplexer.









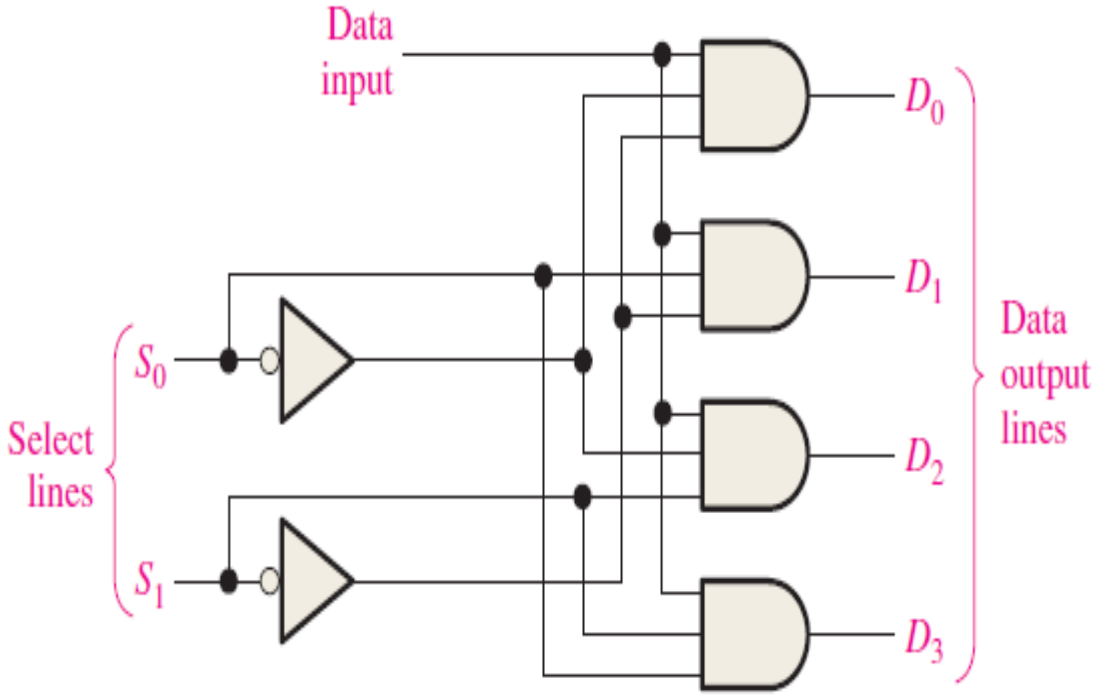


Demultiplexer Design

- How to design a 1-to-4 demultiplexer.
- First, according to our understanding of the operation we make the truth-table.
- The from the truth table we will form the Boolean expressions.
- And from the Boolean expression we will draw the logic circuit.

D_{IN}	S1	S0	D0	D1	D2	D3
D_{IN}	0	0	D_{IN}	0	0	0
D_{IN}	0	1	0	D_{IN}	0	0
D_{IN}	1	0	0	0	D_{IN}	0
D_{IN}	1	1	0	0	0	D_{IN}

- $D0 = \overline{S1}.\overline{S0}.D_{IN}$
- $D1 = \overline{S1}.S0.D_{IN}$
- $D2 = S1.\overline{S0}.D_{IN}$
- $D3 = S1.S2.D_{IN}$



References

1. Thomas L. Floyd, “Digital Fundamentals” 11th edition, Prentice Hall – Pearson Education.

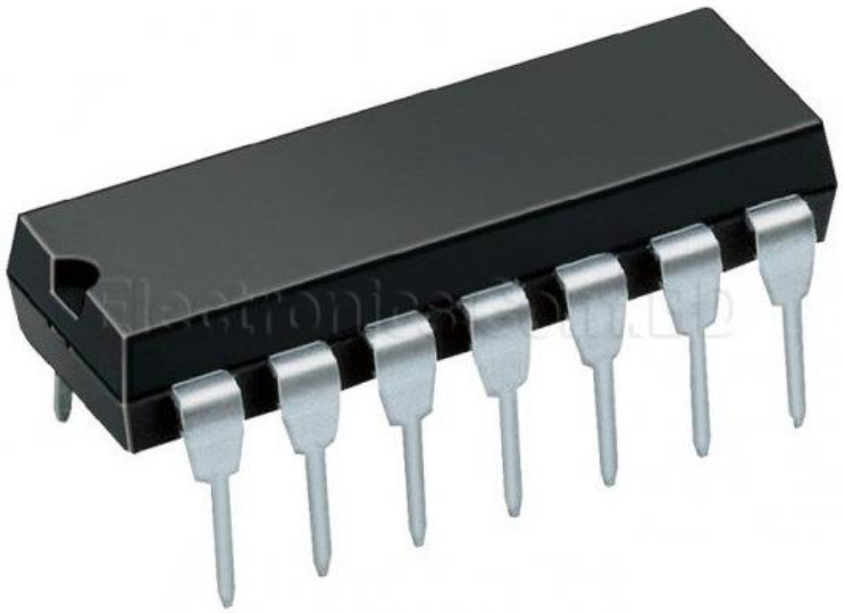
Thank You

Lecture -7
Integrated Circuit Technology-1

Prepared By: Asif Mahfuz

Introduction

- In practice digital circuits are not constructed with individual gates.
- Rather digital circuits are constructed with integrated circuits.
- An integrated circuit (abbreviated as IC) is a small silicon semiconductor, crystal called a chip, containing all the electronic components for the digital gates.
- The various gates are interconnected inside the chip to form the required circuit.
- This cheap is mounted in a ceramic or plastic container, and connections are welded to external pins to form the IC.
- As an engineering when working with an IC we must be aware of the following factors.
  - **The integration level**
  - **The logic family or technology**
  - **The logic level parameters.**
  - **The performance parameters.**



Integration Level

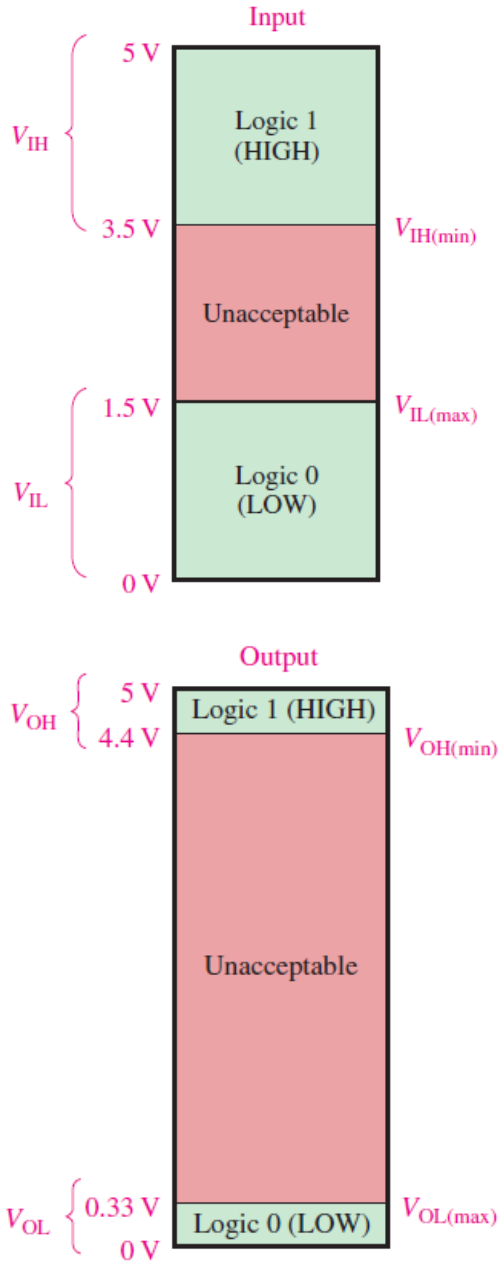
- The integration level of an IC or a semiconductor chip gives us an idea about the number of gates that are present in the chip.
- Integration Level for Integrated Circuits.
  - Small-Scale Integration (SSI) (<12 gates/chip).
  - Medium-Scale Integration (MSI) (<100 gates/chip).
  - Large-Scale Integration (LSI) (…1K gates/chip).
  - Very Large-Scale Integration (VLSI) (…10K gates/chip).
  - Ultra Large-Scale Integration (ULSI) (…100K gates/chip).
  - Giga Scale Integration (GSI) (…1M gates/chip).
- Examples:
  - Pentium III Coppermine(32-bit, large cache): 21,000,000 gates
  - Pentium 4 Willamette (32-bit, large cache): 42,000,000 gates
  - Core 2 Duo Conroe (dual-core 64-bit, large caches): 291,000,000 gates
  - ARM Cortex-A9 (32-bit, (optional) SIMD, caches):26,000,000 gates
  - Atom (32-bit, large cache): 47,000,000

Digital Logic Family

- Digital integrated circuits are classified not only by their complexity or logical operation, but also by the specific circuit technology to which they belong.
- The circuit technology is referred to as a digital logic family.
- The basic circuits in each technology is a NAND, NOR or an inverter gate.
- The logic families are:
  - RTL → Resistor-Transistor Logic
  - DTL → Diode- Transistor Logic
  - I^2L → Integrated injection Logic
  - ECL → Emitter Collector Logic
  - TTL → Transistor-Transistor Logic
  - MOS → Metal Oxide Semiconductor
  - CMOS → Complementary MOS

Logic Level Parameters

- Different Logic Families usually operate at different voltage and current levels.
- **Voltage Parameters**
  - **High-Level Output Voltage, $V_{OH(MIN)}$:** This is the minimum output voltage available at the output under stated loaded condition which corresponds to logic '1'.
  - **Low-Level Output Voltage, $V_{OL(MAX)}$:** This is the maximum output voltage available at the output under stated loaded condition which corresponds to logic '0'
  - **High-Level Input Voltage, $V_{IH(MIN)}$:** This is the minimum voltage required at an input to be recognized as a logic '1'.
  - **Low-Level Input Voltage, $V_{IL(MAX)}$:** This is the maximum voltage at an input which will be recognized as a logic '0'.

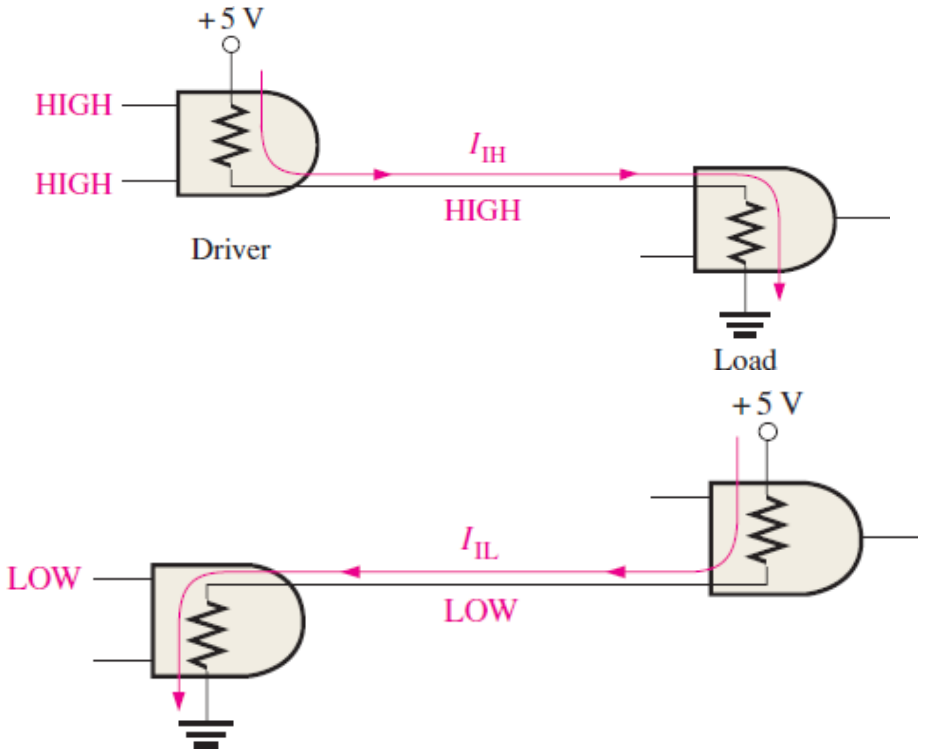


Performance Parameters

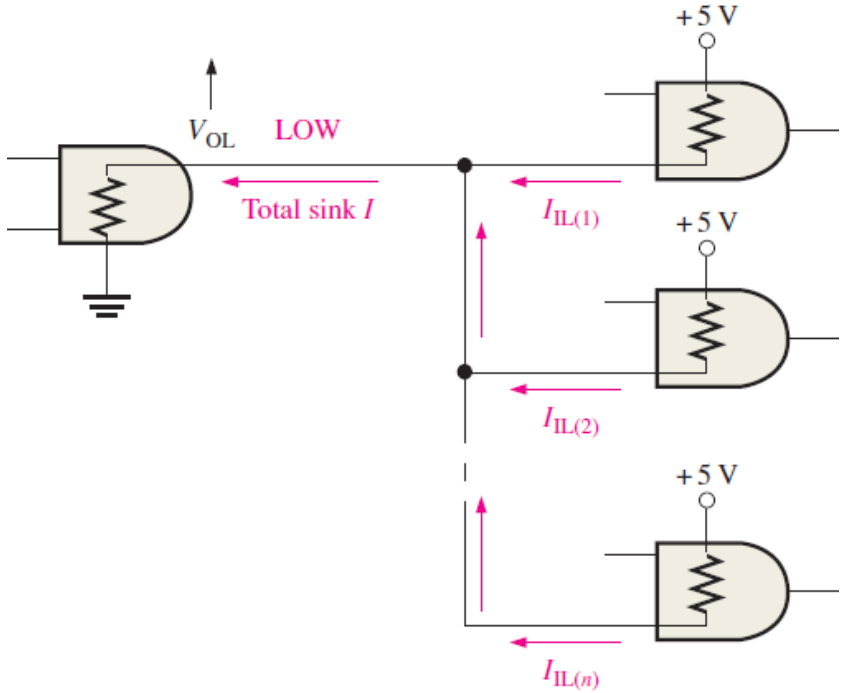
- The performance parameters are also known as the special characteristics of a logic family.
- The characteristics of IC digital logic families are usually compared by analyzing the circuit of the basic gate in each family.
- The most important parameters that are evaluated and compared are
  - Fan-Out
  - Fan-In
  - Power Dissipation
  - Propagation Delay
  - Speed-Power Product
  - Noise Margin

Loading

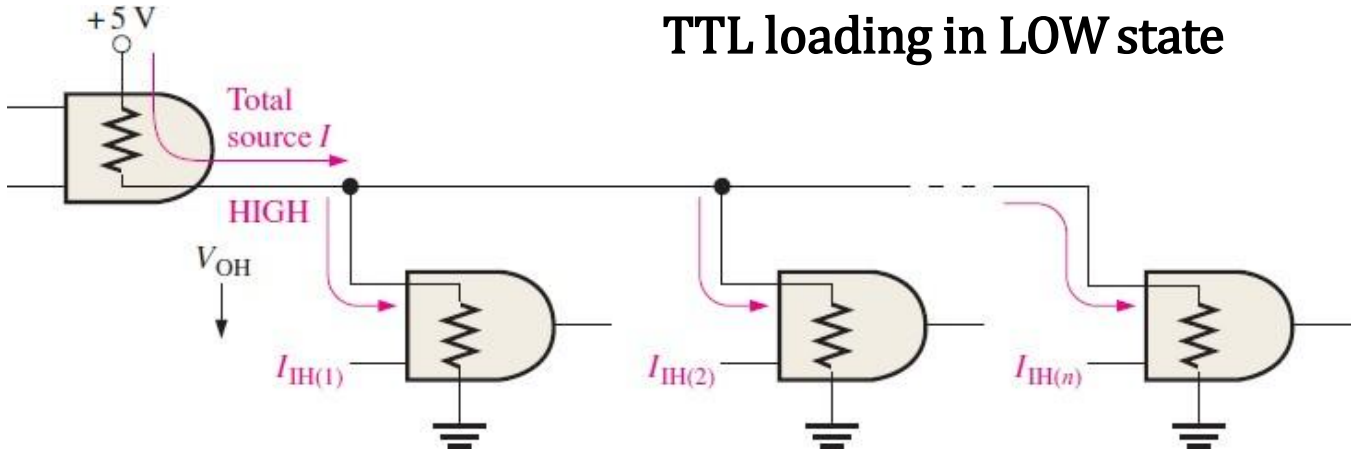
- When the output of a gate is connected to the input of one or more gates, a load is created on the driver gate. There is a limit to the number of input that the output a gate can drive. This is determined by the fan-out.



TTL loading in HIGH and LOW state



TTL loading in LOW state



TTL loading in HIGH state

Performance Parameters: Fan-Out

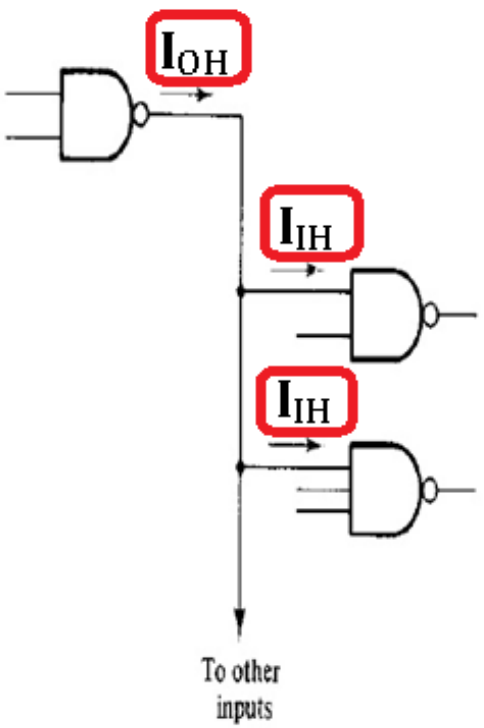
- The **fan-out** of a gate specifies the number of standard loads that can be connected to the output of the gate without degrading its normal operation.
- A **standard load** is usually defined as the amount of current needed by an input of another gate in the same logic family. Sometimes the term loading is used instead of fan-out.
- This term is derived because the output of a gate can supply a limited amount of current, above which it ceases to operate properly and is said to be overloaded.
- Each input consumes a certain amount of current from the gate output, so that each additional connection adds to the load of the gate.
- Exceeding the specified maximum load may cause a malfunction because the circuit cannot supply the power demanded from it.
- Also the propagation delay increases with fan-out.
- The fan-out is the maximum number of inputs that can be connected to the output of a gate and is expressed by a number.

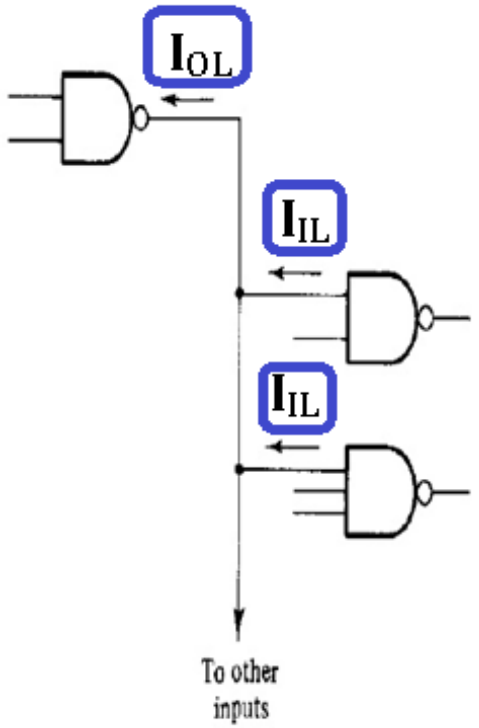
$$\textbf{DC Fanout} = \min\left(\frac{I_{OH}}{I_{IH}}, \frac{I_{OL}}{I_{IL}}\right)$$

Performance Parameters: Fan-Out

- Calculating the Fan-out of a driving gate with the following parameters:
  - $I_{OH} = 400\mu A$ and $I_{IH} = 40\mu A$
  - $I_{IH} = 16mA$ and $I_{IL} = 2mA$
- Solution

$$FO = \min\left(\frac{400\mu A}{40\mu A}, \frac{16mA}{2mA}\right) = 8units$$

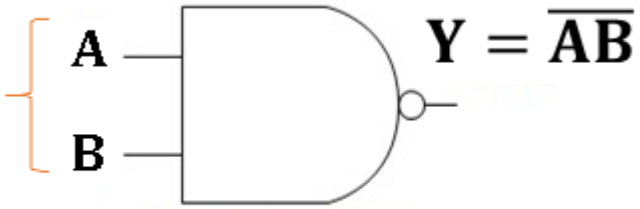




Performance Parameters: Fan-In

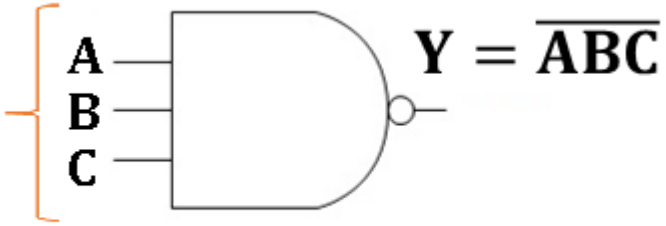
- The fan in defined as the maximum number of inputs that a logic gate can accept.
- If number of input exceeds, the output will be undefined or incorrect.
- It is specified by manufacturer and is provided in the data sheet.
- Delay approximately has a quadratic relation with Fan-In.

Fan In = 2

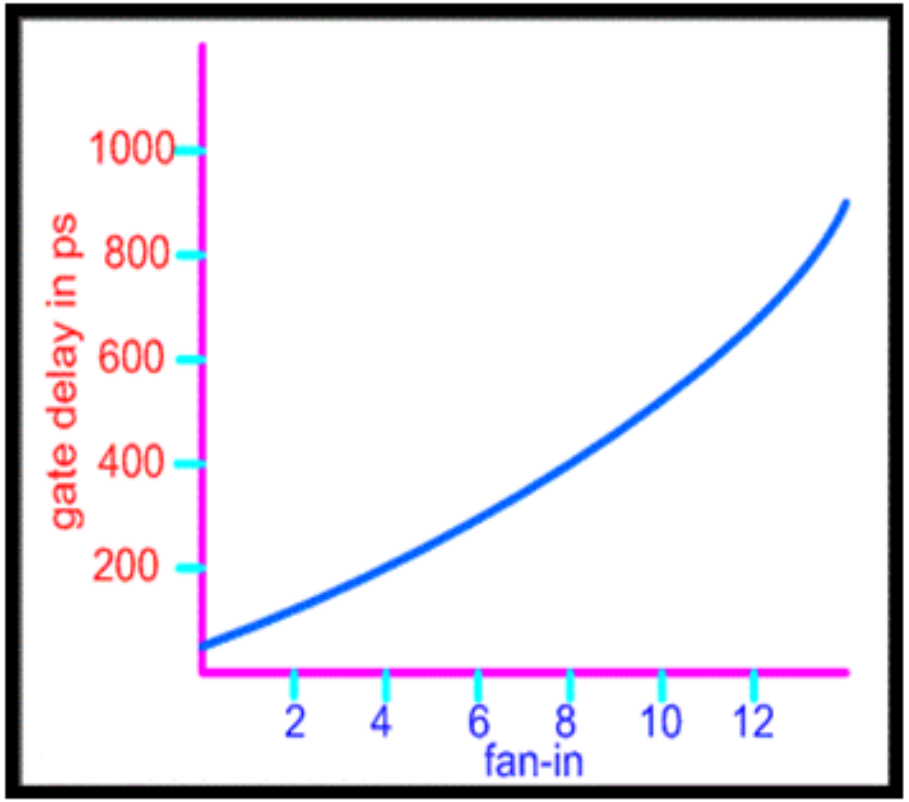


Two Input NAND Gate

Fan In = 3

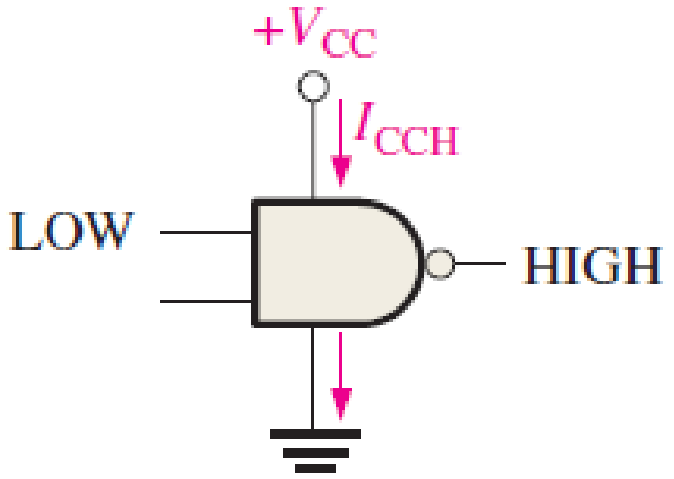


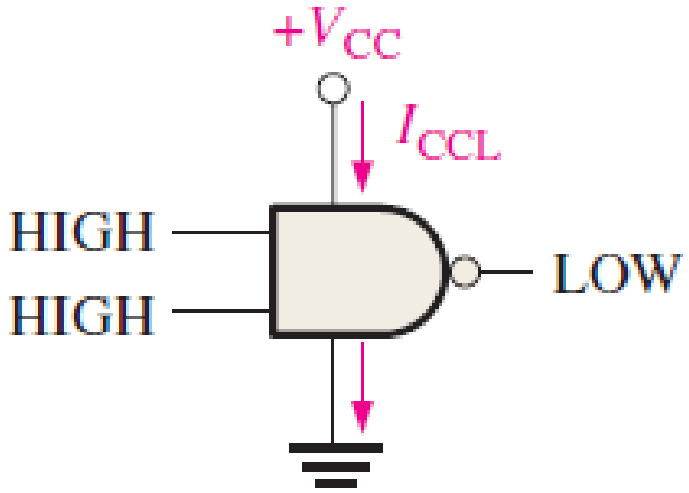
Three Input NAND Gate



Performance Parameters: Power Dissipation

- A gate draws in current both in HIGH and LOW states.
- Therefore, in both states a gate dissipates power.
- The power dissipation is a parameter expressed in milliwatts (mW) and represents the amount of power needed by the gate.
- The number that represents this parameter does not include the power delivered from another gate; rather, it represents the power delivered to the gate from the power supply.
- An IC with four gates will require, from its power supply, four times the power dissipated in each gate.
- The current associated with HIGH state is named I_{CCH}.
- The current associated with LOW state is named I_{CCL}. Therefore,
- Power Dissipation in HIGH state, $P_{DH} = V_{CC}I_{CCH}$
- Power Dissipation in LOW state, $P_{DL} = V_{CC}I_{CCL}$





Performance Parameters: Power Dissipation

- When gate is pulsed, its output switches back and forth between HIGH and Low
- The amount of supply current also varies between ICCH and ICCL.
- The average power is dissipated when the duty cycle is 50%, the output is HIGH half the time and LOW the other half.
- The average power dissipated in a cycle is,

$$\mathbf{PD} = \frac{\mathbf{V_{CC}(I_{CCH} + I_{CCL})}}{\mathbf{2}}$$

- So the power dissipated in a cycle with duty cycle of X%is,

$$\mathbf{PD} = \frac{\mathbf{V_{CC}((X \times I_{CCH}) + ((100 - X)I_{CCL}))}}{\mathbf{100}}$$

- Find the average power dissipated for a NAND gate with $V_{CC} = 5V$, $I_{CCH} = 4mA$ and $I_{CCL} = 2mA$ when the duty cycle is:
  a) 40%
  b) 75%
  c) 20%

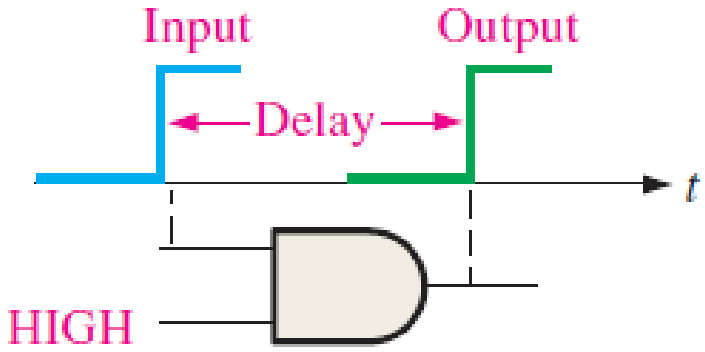
Performance Parameters: Propagation Delay

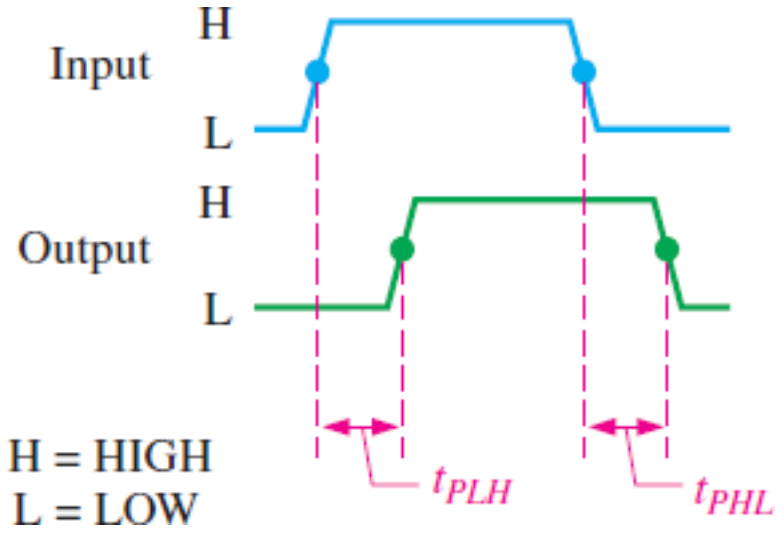
- When a signal passes through a logic circuit, it always experiences a time delay.
- A change in the output level always occurs after a short time, called the propagation delay.
- $\mathbf{T_{PHL}}$: Propagation delay for High to Low.
- $\mathbf{T_{PLH}}$: Propagation delay for Low to High.

$$\mathbf{t_{PD} = \frac{1}{2}(t_{PHL} + t_{PLH})}$$

- As an example, the delays for a standard TTL gate are $t_{PHL} = 7$ ns and $t_{PLH} = 11$ ns. These quantities are given in the TTL data book and are measured with a load resistance of 400 ohms and a load capacitance of 15 pF. The average propagation delay of the TTL gate is:

$$t_{PD} = \frac{11 + 7}{2} = 9 \text{ ns}$$





Performance Parameters: Speed Power Product

- The speed power product provides the basis for the comparison of logic circuits when both propagation delay time and power dissipation are important considerations in the selection of the type of logic to be used in a certain application.
- The lower the speed-power the better.
- The unit of speed-power product is Pico joule(pJ).
- The speed power product can be calculated as:

$$\textbf{SPP} = \textbf{Power Disspation} \times \textbf{Propagation Delay}$$

- The table below lists the parameters for three types of gates. Basing your decision on the speed-power product, which one would you select for best performance?

	t_{PLH}	t_{PHL}	P_D
Gate *A*	1 ns	1.2 ns	15 mW
Gate *B*	5 ns	4 ns	8 mW
Gate *C*	10 ns	10 ns	0.5 mW

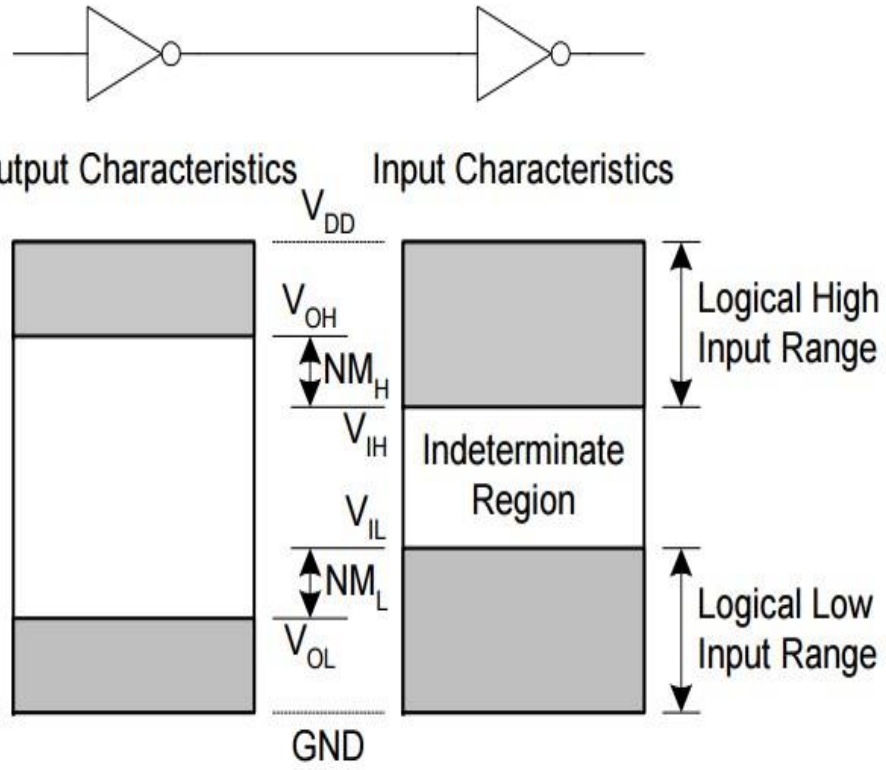
Performance Parameters: Noise Margin

- All electrical circuits are susceptible to noise.
- Unwanted signals are referred to as *noise.*
- This unwanted induced voltage can disrupt the operation of a digital circuit.
- In order to not get adversely affected by noise the circuit should have some amount of noise immunity.
- Noise Immunity is the ability to tolerate unwanted voltage fluctuation.
- There are two types of noise to be considered :
  - **DC noise is** caused by a drift in the voltage levels of a signal.
  - AC noise is a random pulse that may be created by other switching signals.
- Noise margin is the maximum noise voltage added to an input signal of a digital circuit that does not cause an undesirable change in the circuit output.
- Noise margin is expressed in volts and represents the maximum noise signal that can be tolerated by the gate.

Performance Parameters: Noise Margin

- There are two values of noise margin specified for a given logic circuit:
  - the High-level noise margin (V_{NH}) and
  - the Low-level noise margin (V_{NL}).
- There parameters are defined by the following equations:

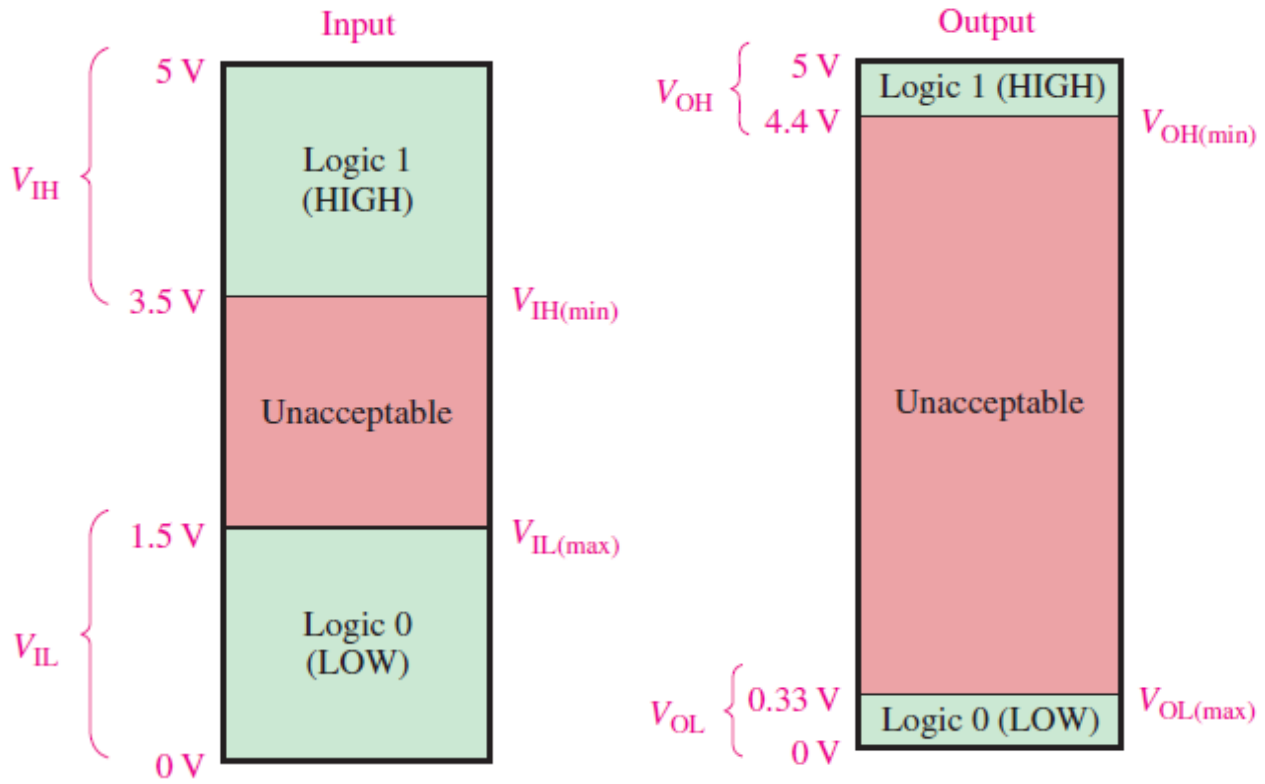
$$V_{NL} = V_{IL} - V_{OL}$$
$$V_{NH} = V_{OH} - V_{IH}$$



Performance Parameters: Noise Margin

- The figure below shows the different voltage level parameters for a +5V CMOS.
- The noise margin for +5V CMOS are:

$$V_{NH} = 4.4V - 3.5V = 0.9V$$
$$V_{NL} = 1.5V - 0.33V = 1.17$$

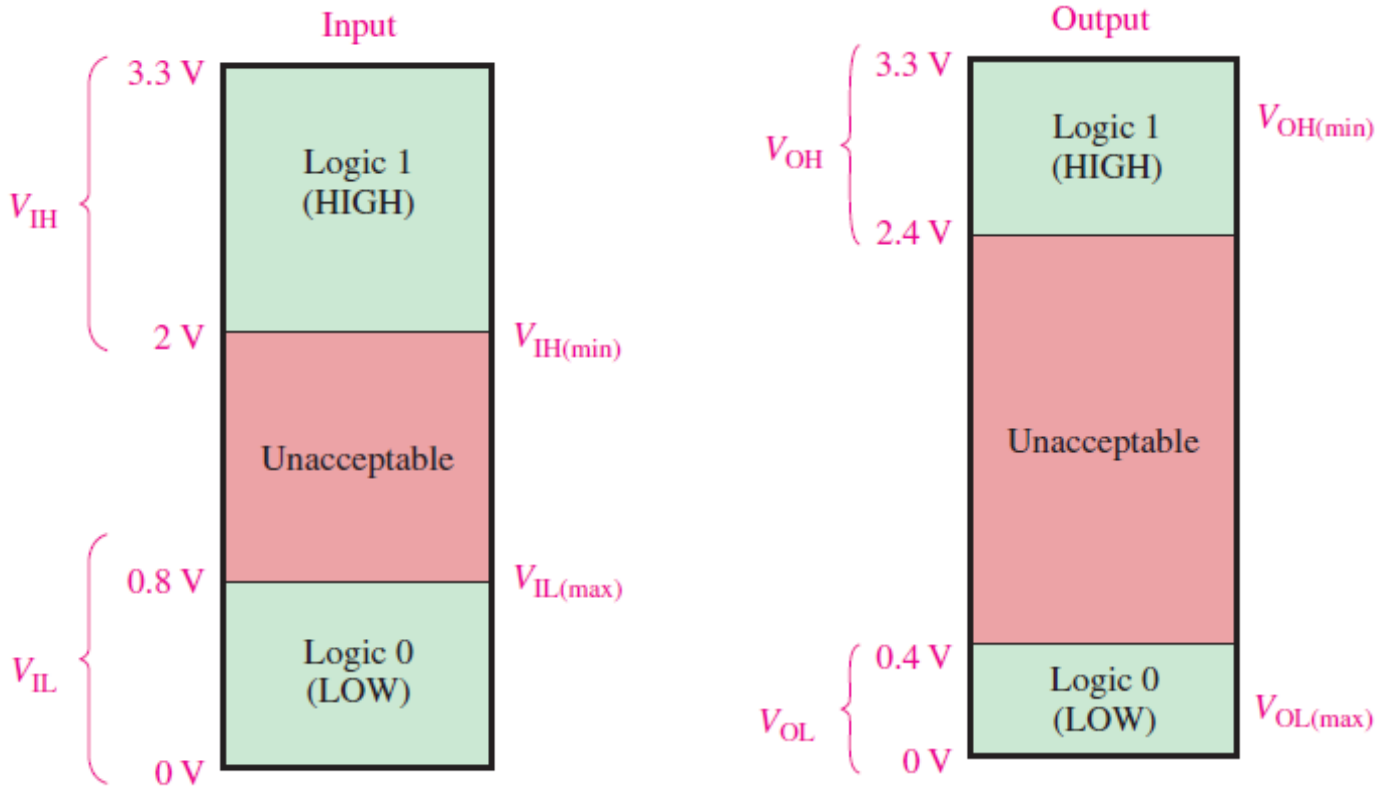


Logic Levels for +5V CMOS

Performance Parameters: Noise Margin

- The figure below shows the different voltage level parameters for a +3.3V CMOS.
- The noise margin for +3.3V CMOS are:

$$V_{NH} = 2.4V - 2V = 0.4V$$
$$V_{NL} = 0.8V - 0.4V = 0.4V$$

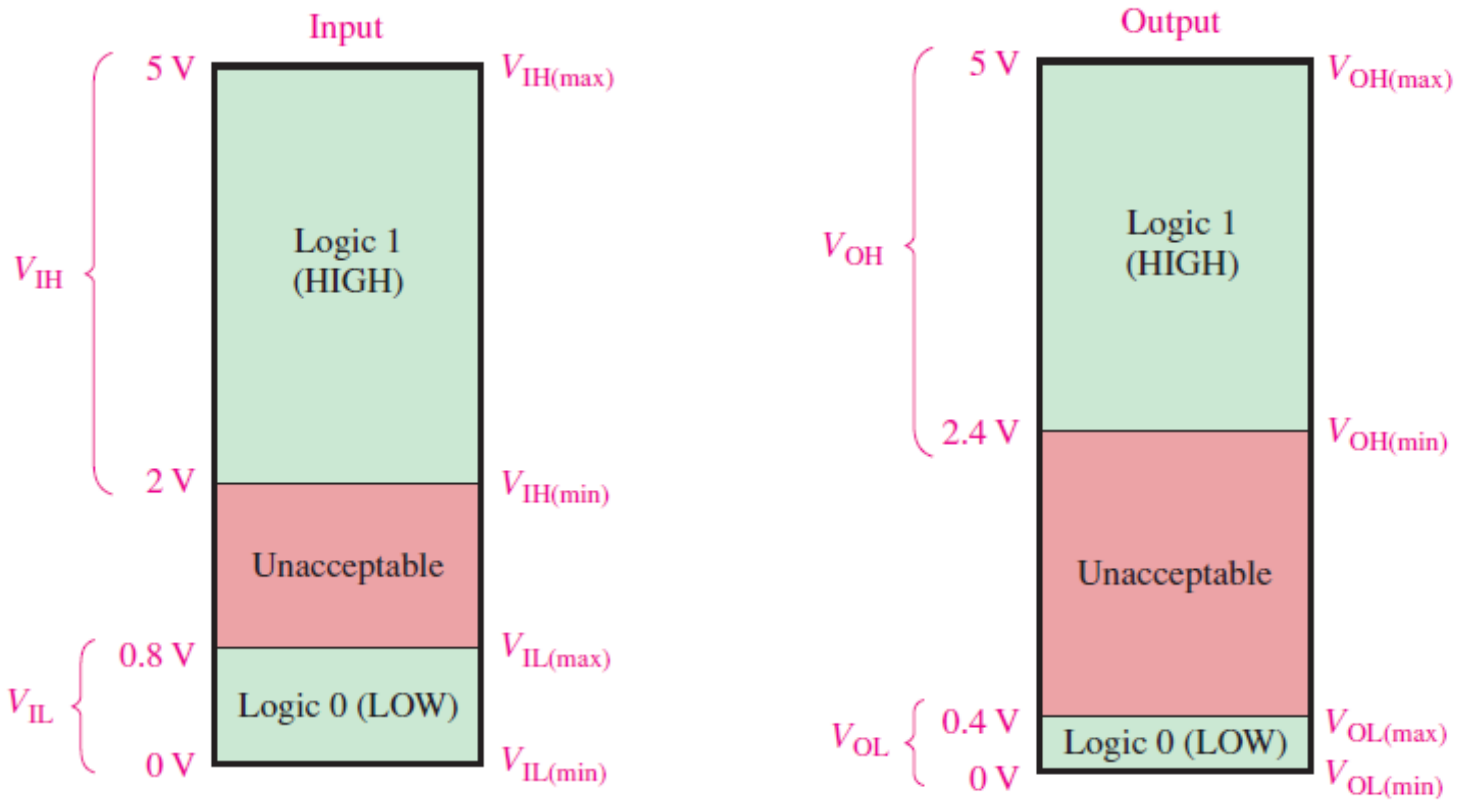


Logic Levels for +3.3 V CMOS

Performance Parameters: Noise Margin

- The figure below shows the different voltage level parameters for TTL.
- The noise margin for TTL are:

$$V_{NH} = 2.4V - 2V = 0.4V$$
$$V_{NL} = 0.8V - 0.4V = 0.4V$$



Logic Levels for TTL

Performance Parameters: Noise Margin

- Voltage specifications for three types of logic gates are given in the Table below. Select the gate that you would use in a high-noise environment.

	$V_{OH(min)}$	$V_{OL(max)}$	$V_{IH(min)}$	$V_{IL(max)}$
Gate *A*	2.4 V	0.4 V	2 V	0.8 V
Gate *B*	3.5 V	0.2 V	2.5 V	0.6 V
Gate *C*	4.2 V	0.2 V	3.2 V	0.8 V

Solution:

- Gate A: $N_{MH} = 2.4V - 2V = 0.4V$; $N_{ML} = 0.8V - 0.4V = 0.4V$
- Gate B: $N_{MH} = 3.5V - 2.5V = 1V$; $N_{ML} = 0.6V - 0.2V = 0.4V$
- Gate C: $N_{MH} = 4.2V - 3.2V = 1V$; $N_{ML} = 0.8V - 0.2V = 0.6V$

References

1. Thomas L. Floyd, “Digital Fundamentals” 11th edition, Prentice Hall – Pearson Education.

Thank You

Lecture -8
Integrated Circuit Technology-2

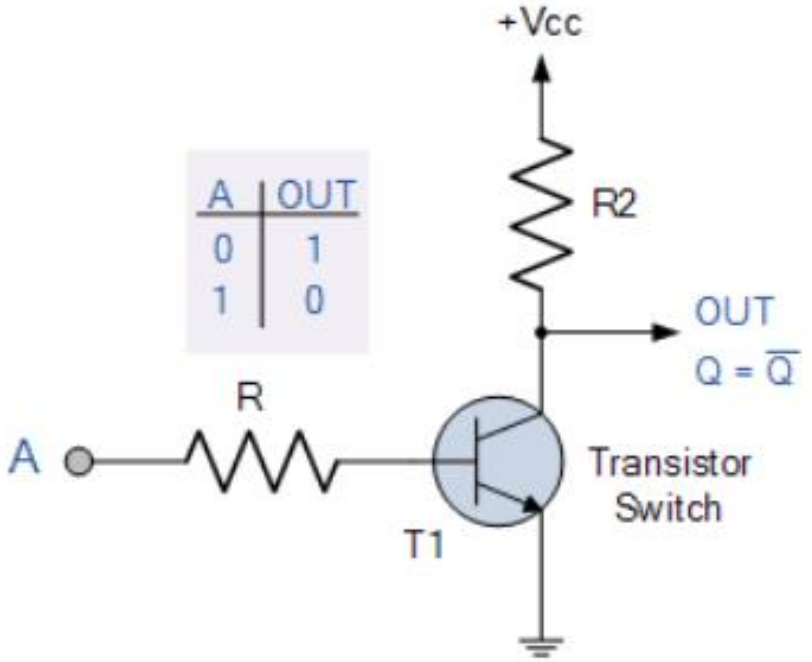
Prepared By: Asif Mahfuz

Resistor-Transistor Logic (RTL)

- It is the first the first logic family to have been commercially used but now it is obsolete.
- The basic circuit of the RTL digital logic family is the NOR gate.
- The voltage levels for the circuit are 0.2 V for the low level and from 1 to 3.6 V for the high level.
- The fan-out of the RTL gate is limited by the value of the output voltage when high.
- As the output is loaded with inputs of other gates, more current is consumed by the load.
- Any voltage below 1 V in the output may not drive the next transistor into saturation as required.
- The power dissipation of the RTL gate is about 12 mW.
- The propagation delay averages 25 ns.
- **Disadvantages of RTL Logic are as follows:**
  - It's relatively slow.
  - Low noise immunity and noise margin.
  - Low Fan-out (Approx. 3~5)
  - Expensive due to fabrication of resistor.
  - It can not operate above 4MHz.

NOT Gate Using RTL

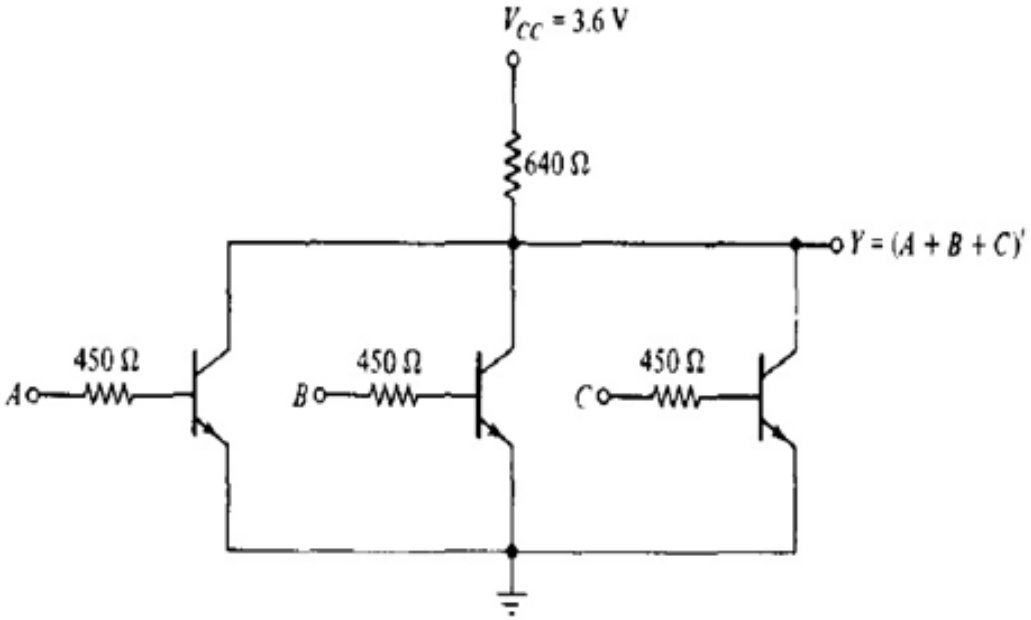
- The figure in the left shows an RTL NOT gate.
- The operation of the RTL gate is as follows:
  - When the input is LOW, the transistor T1 is at cutoff region. The output is $+V_{CC}$.
  - When the input is HIGH, the transistor T1 is ON and in saturation region. The output is LOW. The voltage across the output terminal is the saturation voltage 0.2V
  - The table below summarizes the operation.



Input	Transistor Status	Output
A=Low	T1=OFF, Cut-off Region	$Q=+V_{CC}$, High
A=High	T1=ON, Saturation Region	$Q=V_{CE(SAT)}=0.2V$, Low

NOR Gate Using RTL

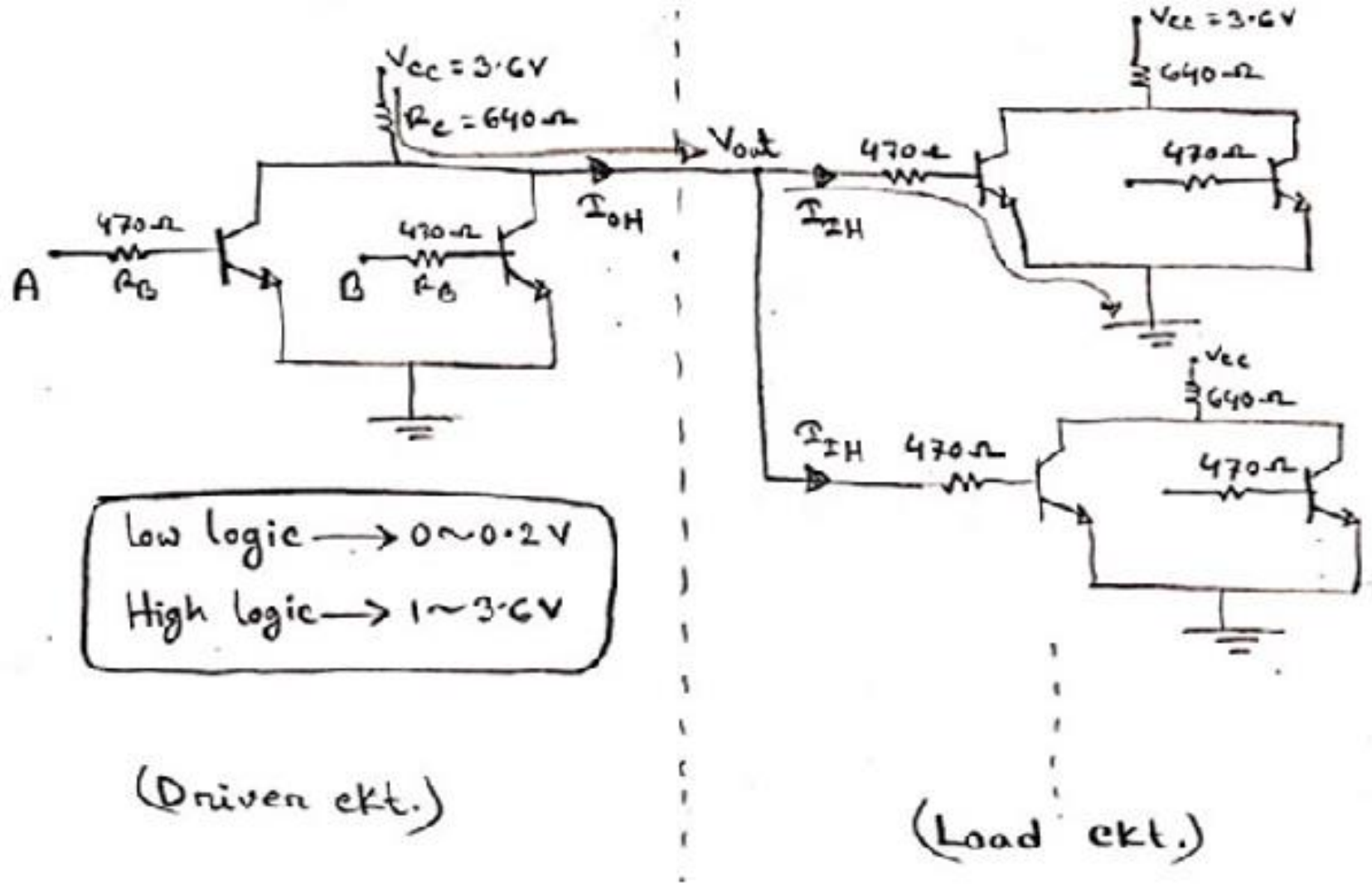
- If any input of the RTL gate is high, the corresponding transistor is driven into saturation. This causes the output to be low, regardless of the states of the other transistors.
- If all inputs are low at 0.2 V, all transistors are cut off because VBE < 0.6 V. This causes the output of the circuit to be high, approaching the value of supply voltage VCC.
- The table below summarizes the operation.



Input	Transistor Status	Output
All inputs are LOW	All transistors are OFF and in cutoff region	$Q = +V_{CC}$, High
Any one input/ Two input/ all inputs is HIGH	Corresponding transistor turns ON and is in saturated region	$Q = V_{CE(SAT)} = 0.2V$, Low

Fanout Calculation for RTL NOR

- For the given circuit calculate the Fan-out of RTL NOR ?



Fanout Calculation for RTL NOR

$$\text{Fan-out, } N = \frac{I_{OH}}{I_{IH}}$$

A	B	Y_{NOR}
0	0	1
0	1	0
1	0	0
1	1	0

KVL in driven ckt,

$$3.6 - 640 \times I_{OH} - 1 = 0$$

$$\Rightarrow I_{OH} = \frac{3.6-1}{640} = 0.00406\ A$$

KVL in load ckt,

$$1 - 470 \times I_{IH} - 0.7 = 0$$

$$\Rightarrow I_{IH} = \frac{1-0.7}{470} = 0.000638\ A$$

$$\therefore N = \frac{I_{OH}}{I_{IH}} = \frac{0.00406}{0.000638} = 6.36 \approx 6$$

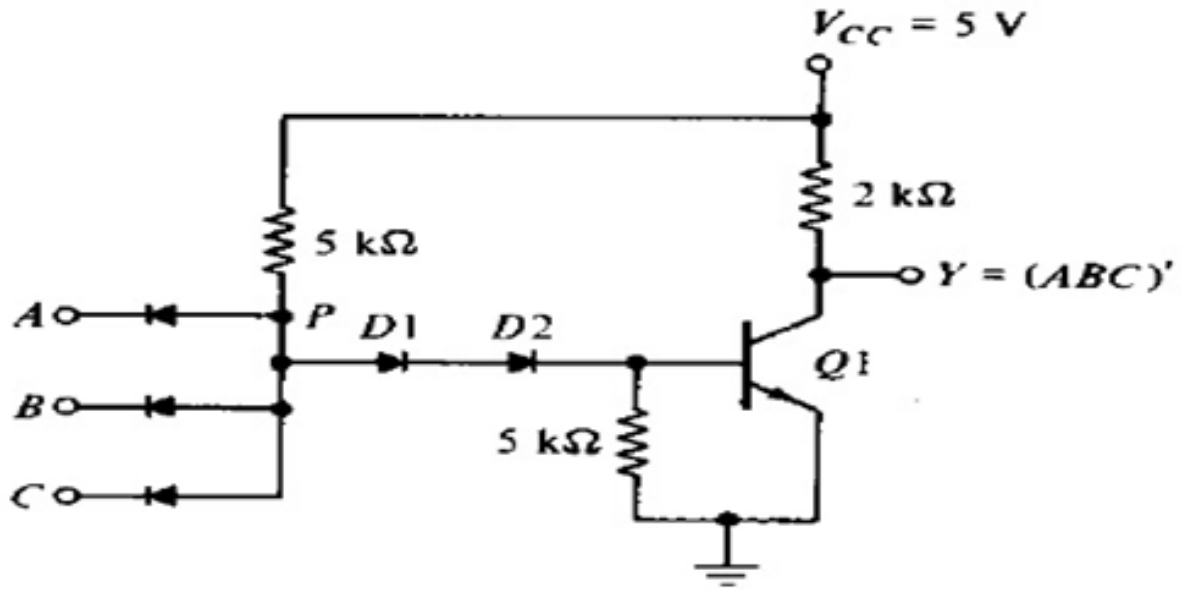
∴ Fan-out is 6.

Diode-Transistor Logic (DTL)

- It is currently an obsolete technology.
- The basic circuit in the DTL digital logic family is the NAND gate.
- Each input is associated with one diode.
- The diodes and the 5-kΩ resistor form an AND gate.
- The transistor serves as a current amplifier while inverting the digital signal.
- The two voltage levels are 0.2 V for the low level and between 4 and 5 V for the high level.
- The power dissipation of a DTL gate is about 12 mW.
- The propagation delay averages 30 ns.
- The noise margin is about I V and a fan-out as high as 8 is possible.
- **Disadvantages of DTL logic family are as follows:**
  - Relatively lower speed.
  - Propagation delay is higher than RTL.

NAND Gate Using DTL

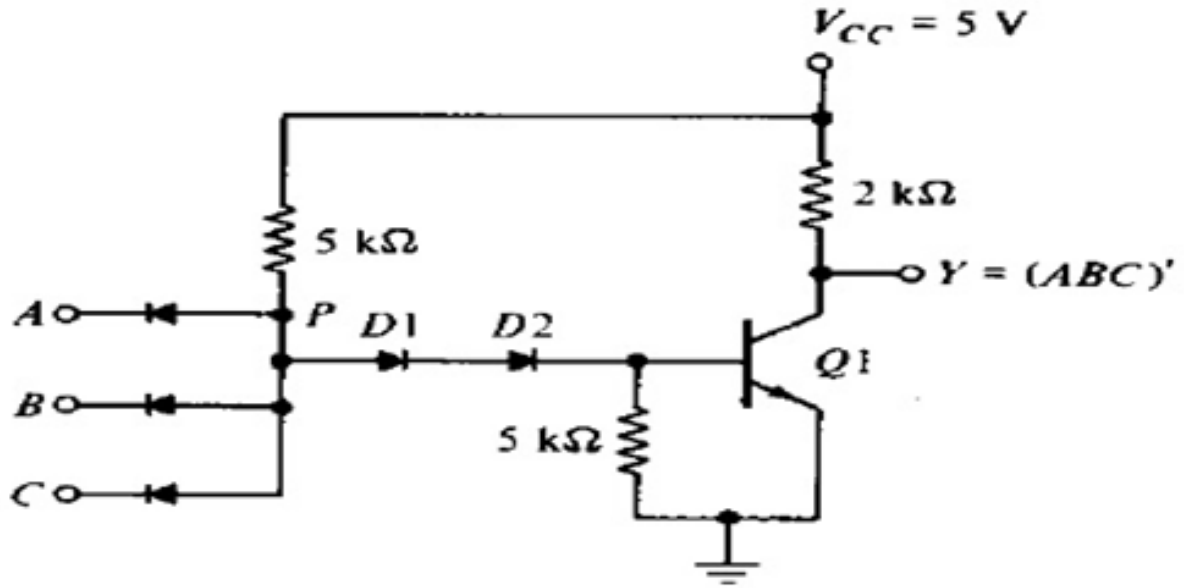
- If any input of the gate is low at 0.2 V, the corresponding input diode conducts current through Vcc and the 5-kΩ resistor into the input node.
- The voltage at point P is equal to the input voltage of 0.2 V plus a diode drop of 0.7 V, for a total of 0.9 V.
- For the transistor to start conducting, the voltage at point P must overcome a potential of one VBE drop in Q 1 plus two diode drops across D 1 and D 2, or 3 X 0.6 = 1.8 V.
- Since the voltage at P is maintained at 0.9 V by the input conducting diode, the transistor is cut off and the output voltage is high at 5 V.



NAND Gate Using DTL

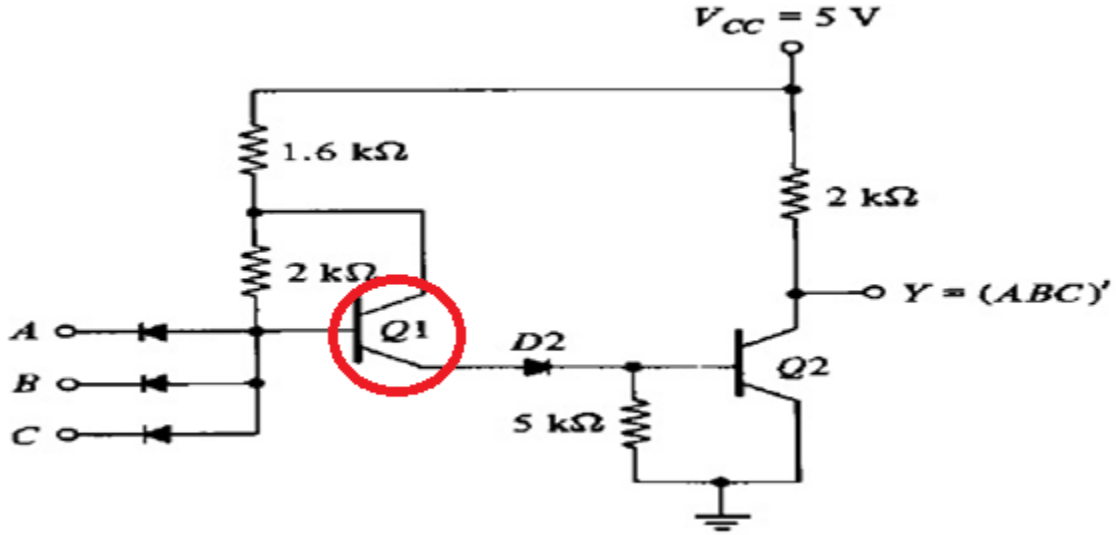
- If all inputs of the gate are high, the transistor is driven into the saturation region.
- The voltage at P now is equal to V_{BE} plus the two diode drops across D I and D 2, or 0.7 x 3 = 2.1 V.
- Since all inputs are high at 5 V and $V_P = 2.1$ V, the input diodes are reverse biased and off.
- The base current is equal to the difference of currents flowing in the two 5-kΩ resistors and is sufficient to drive the transistor into saturation.
- With the transistor saturated, the output drops to VCE of 0.2 V, which is the low level for the gate.

Input	Transistor and Diode Status	Output
Any input is Low (0.2 V)	Corresponding input diode is forward biased. Q1 is off and in cutoff region. $V_P = (0.2 + 0.7)V$	Y=+V_{CC}, High
All inputs are High (5V)	All input diode is reverse biased. D1 & D2 is forward biased. Q1 is ON and in saturation region.	Y=$V_{CE(SAT)}$=0.2V, Low



NAND Gate Using Modified DTL

- The fan-out of the DTL gate is limited by the maximum current that can flow in the collector of the saturated transistor.
- The fan-out of a DTL gate may be increased by replacing one of the diodes in the base circuit with a transistor.



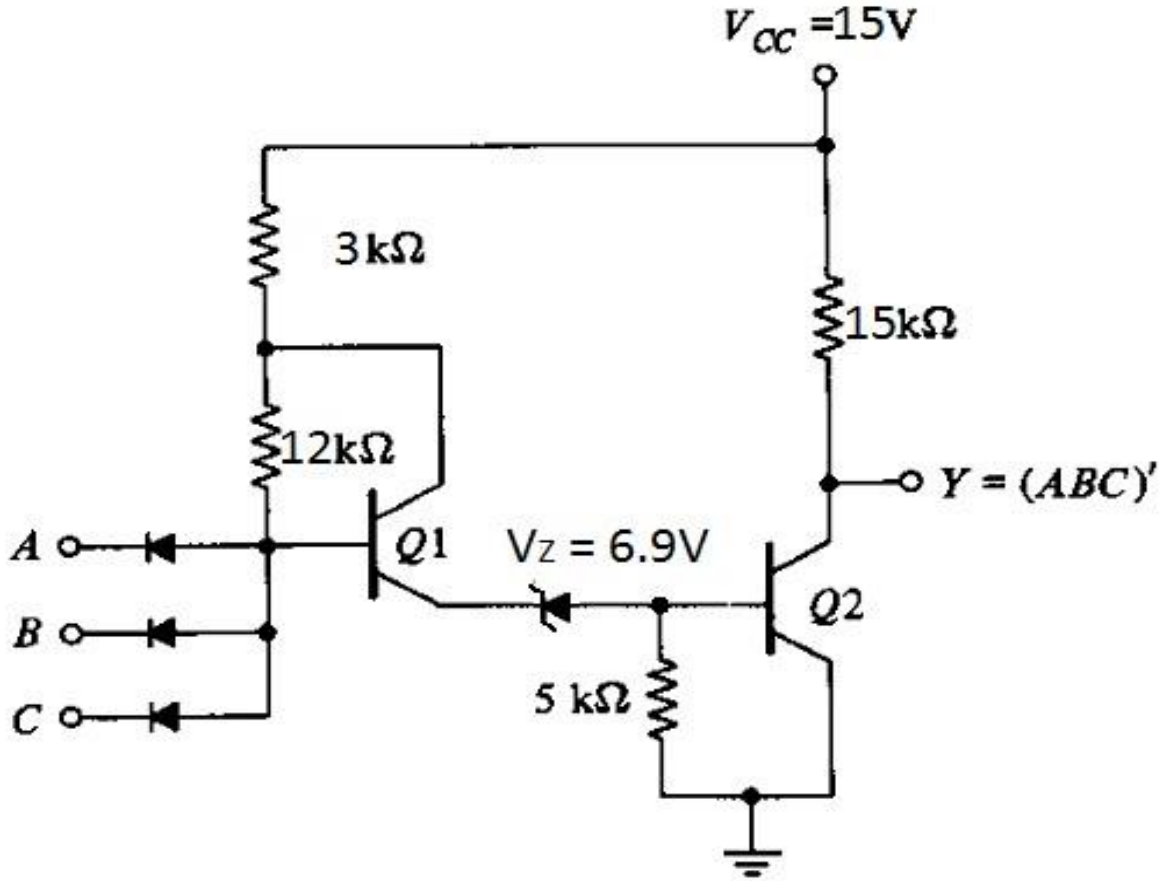
- Transistor Q1 is maintained in the **active region** when output transistor Q2 is **saturated**.
- Therefore, the modified circuit can supply a larger amount of base current to the output transistor.
- The output transistor can now draw a larger amount of collector current before it goes out of saturation.
- Part of the collector current comes from the conducting diodes in the loading gates when Q 2 is saturated.
- Thus, an increase in allowable collector saturated current allows more loads to be connected to the output, which increases the fan-out capability of the gate.

NAND Gate Using HTL

- Due to presence of electrical motor's on-off control circuits, High voltage switches etc. are used in industrial environment, Thereby noise level is very high.
- So DTL redesigned with a power supply of 15V and the D2 is replaced by a Zener diode with a breakdown voltage of 6.9 V.
- So HTL posses a high threshold of noise immunity.
- To conduct Q2, the emitter of Q1 must rise to a potential,

$$V_{BE}\,(Q2) + V_Z = 0.7 + 6.9 = 7.6\,V$$

- So, Only if the noise signal is greater than 7.6 V , then it will be able to change the state. Higher noise immunity.



Transistor-Transistor Logic

- The original basic TTL gate was a slight improvement over the DTL gate.
- As the TTL technology progressed, additional improvements were added to the point where this logic family became the most widely used family in the design of digital systems.
- The propagation delay of a transistor circuit that goes into saturation depends mostly on two factors: storage time and *RC* time constants.
- Reducing the storage time decreases the propagation delay. [RC T_P]
- Reducing resistor values in the circuit reduces the *RC* time constants and decreases the propagation delay.
- Of course, the trade-off is higher power dissipation because lower resistances draw more current from the power supply. The speed of the gate is inversely proportional to the propagation delay.
- In the low-power TTL gate, the resistor values are higher than in the standard gate to reduce the power dissipation, but the propagation delay is increased. [R↑ so P_D↓ but T_P↑]
- In the high-speed TTL gate, resistor values are lowered to reduce the propagation delay, but the power dissipation is increased. . [R↓ so T_P↓ but P_D↑]

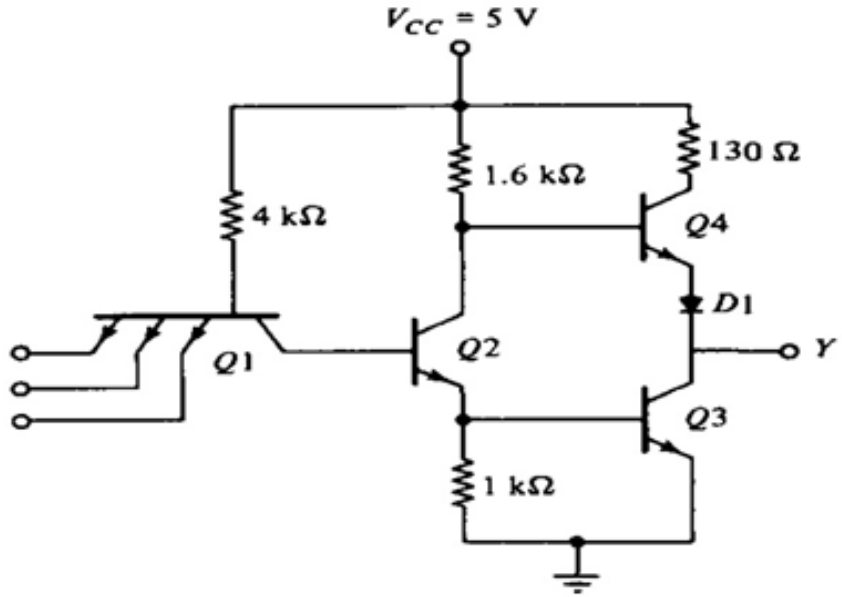
Transistor-Transistor Logic

- The Schottky TTL gate was the next improvement in the technology.
- The effect of the Schottky transistor is to remove the storage time delay by preventing the transistor from going into saturation.
- The low-power Schottky TTL sacrifices some speed for reduced power dissipation.
- It is equal to the standard TTL in propagation delay, but has only one fifth the power dissipation
- Recent innovations have led to the development of the advanced Schottky series.
- The advanced low-power Schottky has the lowest speed-power product and is the most efficient series. It is replacing all other low· power versions in new designs.
- TTL gates in all the available series come in three different types of output configuration:

1. **Open -collector output**
2. **Totem-pole output**
3. **Three-state (or tristate) output**

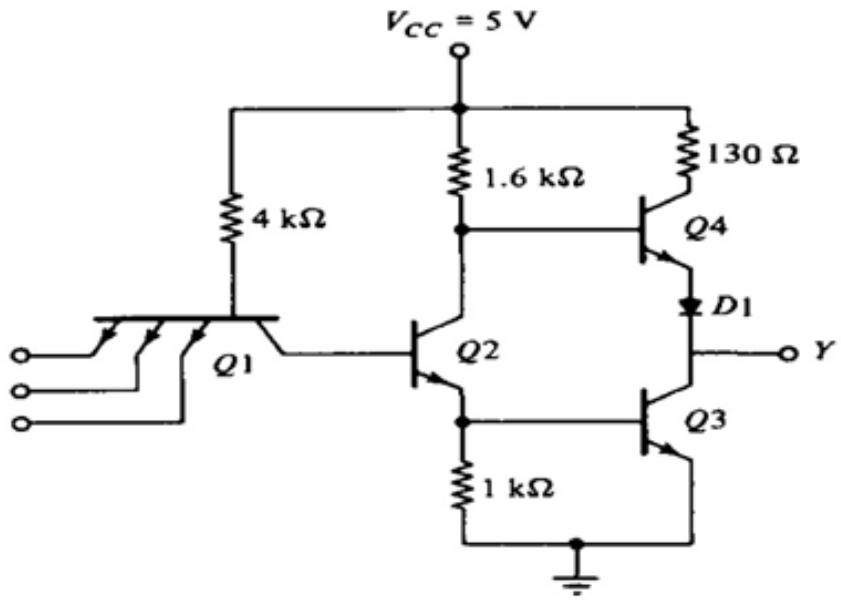
TTL with Totem Pole

- The output impedance of a gate is normally a resistive plus a capacitive load.
- The capacitive load consists of the capacitance of the output transistor, the capacitance of the fan-out gates, and any stray wiring capacitance.
- When the output changes from the low to the high state, the output transistor of the gate goes from saturation to cutoff and the total load capacitance, C, charges exponentially from the low to the high voltage level with a time constant equal to RC.
- For the open-collector gate, R is the external resistor marked RL.
- Typical operating value C = 15 pF and RL = 4 kΩ. Propagation delay, tP = 35ns.
- With an active pull-up circuit replacing the passive pull-up resistor RL , the propagation delay is reduced to 10 ns.



NAND using TTL with Totem Pole

Input	Transistor and Diode Status	Output
Any input is Low (0.2 V)	• Corresponding BE junction of Q1 is forward biased. • Voltage at the base of Q1 is: $0.2V + 0.7V = 0.9V$ • Q2 and Q3 is in cutoff region. • Q4 is ON and in D1 is in forward bias.	Y= High
All inputs are High (5V)	• All BE junctions of Q1 is revered biased. • Potential at base of Q1 is 2.1V. • Q2 and Q3 are in saturation region. • Q4 is in cutoff region.	$Y=V_{CE(SAT)}=0.2V$, Low



- The reason for placing the diode in the circuit is to provide a diode drop in the output path and thus ensure that Q4 is cut off when Q3 is saturated.
- For Q3 to start conducting, the path from Q1 to Q3 must be overcame.
- One diode drop in B-C junction of Q1 + two VBE drop in Q2 & Q3 = 3 × 0.7 = 2.1 V
- To conduct Q4, base voltage must have = 0.7 (VBE of Q4)+ 0.7 (D1) = 1.4 V.

NAND using TTL with Totem Pole

Advantage of totem pole output:

- When the output changes to the high state because one of the inputs drops to the low state, transistors Q2 and Q3 go into cutoff. However, the output remains momentarily low because the voltages across the load capacitance cannot change instantaneously.
- The current needed to charge the load capacitance causes Q4 to momentarily saturate, and the output voltage rises with a time constant RC.
- But R in this case is equal to 130Ω, plus the saturation resistance of Q4, plus the resistance of the diode, for a total of approximately 150Ω.
- This value of R is much smaller than the passive pull-up resistance used in the open-collector circuit.
- Therefore, the transition from the low to high level is much faster.

References

1. Thomas L. Floyd, “Digital Fundamentals” 11th edition, Prentice Hall – Pearson Education.

Thank You

Lecture -9
CMOS Logic, Digital System Design

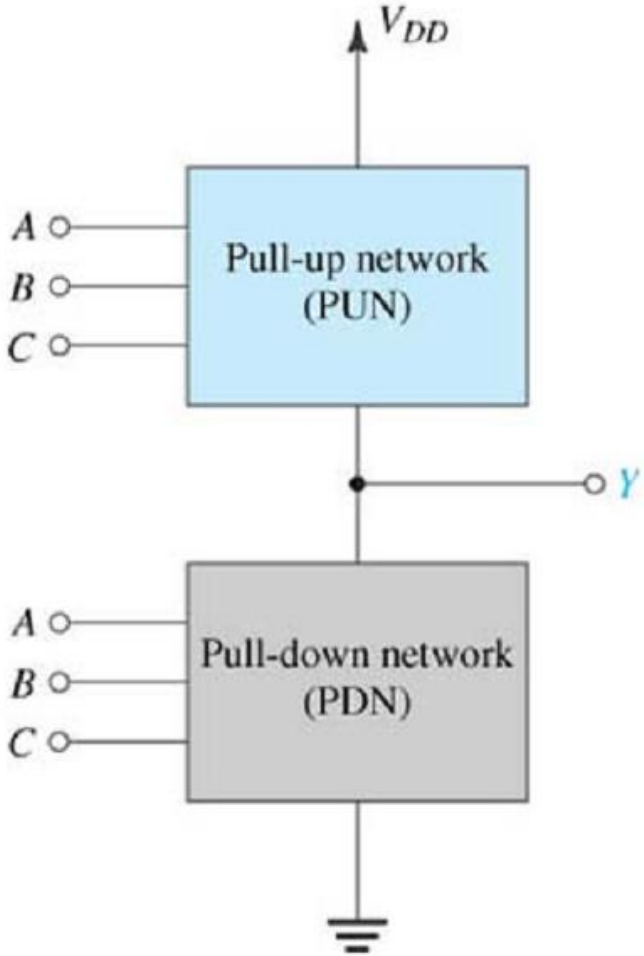
Prepared By: Asif Mahfuz

CMOS Logic

- Static CMOS is the most widely used logic family.
- The primary advantage of CMOS structure is robustness, good performance, and low power consumption.
- It falls under the category of static circuit, as at every point of time (except for switching) each gate is connected to either V_{DD} or ground via low resistance path.
- CMOS allows a much higher density of logic functions in a single chip compared to TTL.
- TTL circuits consume more power in comparison to CMOS circuits at rest.
- A CMOS network is a combination of pull-up network (PUN) and pull-down network (PDN).
- The function of the PUN is to provide a connection between V_{DD} and output when the function needs to be 1.
- Similarly the task of the PDN is to provide a connection between output and ground when function needs to be 0.
- The PUN and PDN are constructed in a mutually exclusive fashion such that one and only one of the networks is conducting in any state.

PUN and PDN

- The pull-up network (PUN) provides a connection between V_{DD} and output.
- The PUN network is constructed from PMOS.
- The PUN is constructed from PMOS as they can give strong “ones”.
- On the other hand, the pull-down network provides a connection between output and ground.
- The PDN network is constructed from NMOS.
- The PDN is constructed from NMOS as they can give strong “zeros”.
- At a time, only one of the network is connected to the output.
- The network provides a low resistance path.

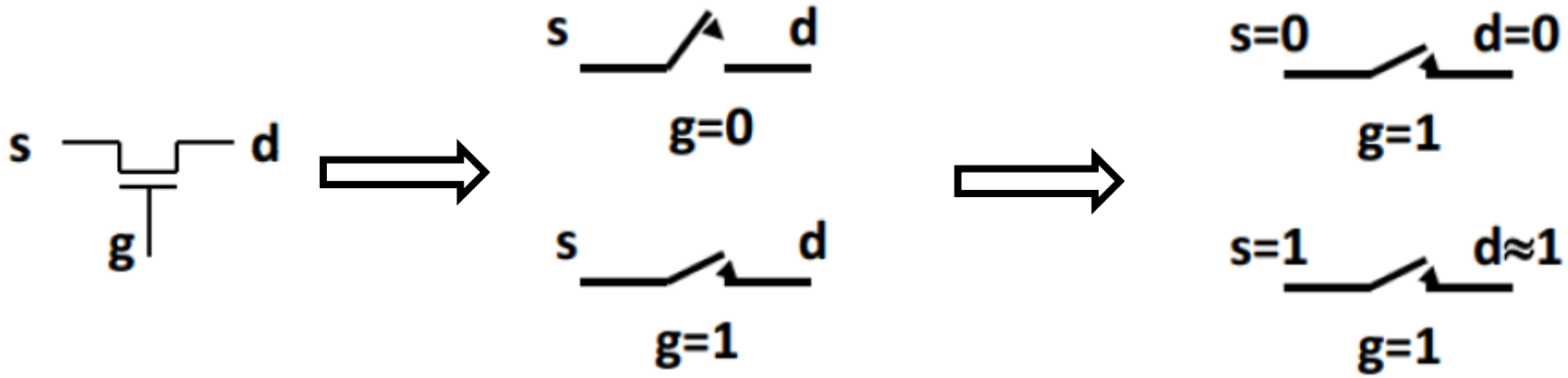


Network of PMOS transistors

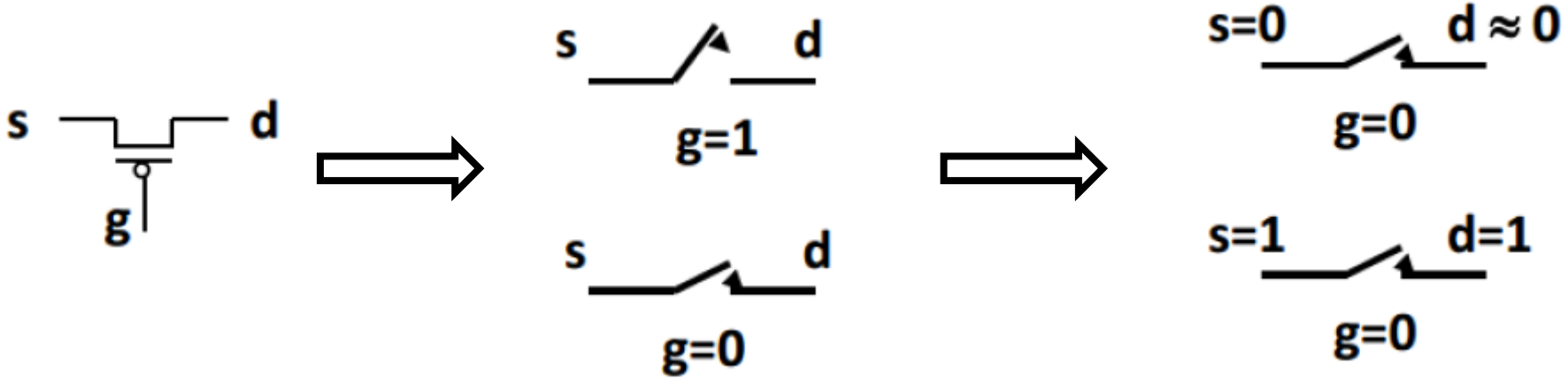
Network of NMOS transistors

PMOS and NMOS as Switches

NMOS



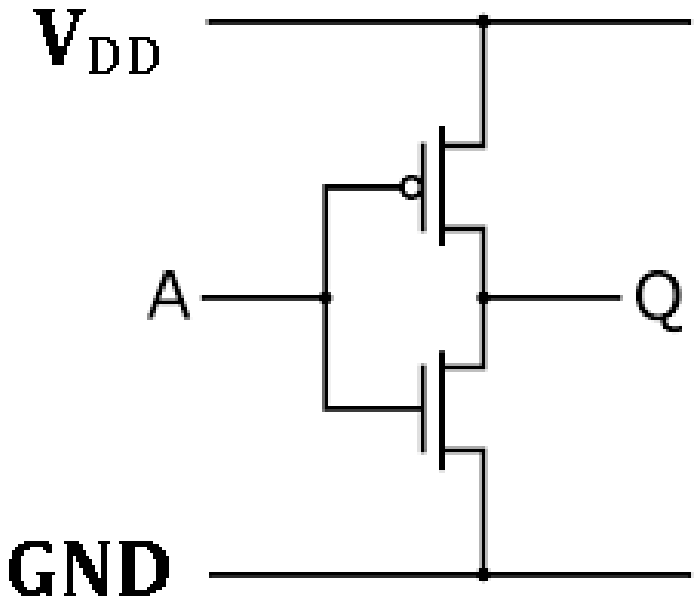
PMOS



NOT Gate using CMOS (CMOS Inverter)

Operation:

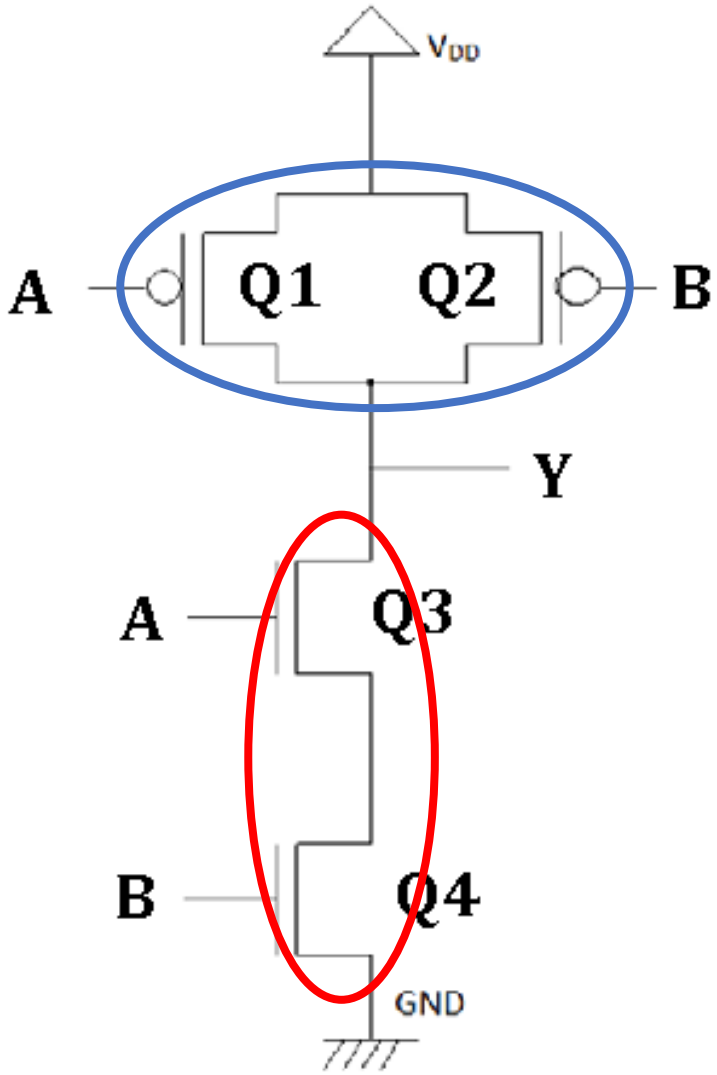
- When A is Low, the PMOS is ON and NMOS is OFF.
- Hence the output is $+V_{DD}$.
- When A is High, the PMOS is OFF an NMOS is ON.
- As a result, the output is Low.
- The table below summarizes the operation.



A	PMOS	NMOS	Q
Low	ON	OFF	$+V_{DD}$
High	OFF	ON	Low

NAND Gate using CMOS (CMOS Inverter)

A	B	Q1 PMOS	Q2 PMOS	Q3 NMOS	Q4 NMOS	Y
0	0	ON	ON	OFF	OFF	$+V_{DD}$
0	1	ON	OFF	OFF	ON	$+V_{DD}$
1	0	OFF	ON	ON	OFF	$+V_{DD}$
1	1	OFF	OFF	ON	ON	Low

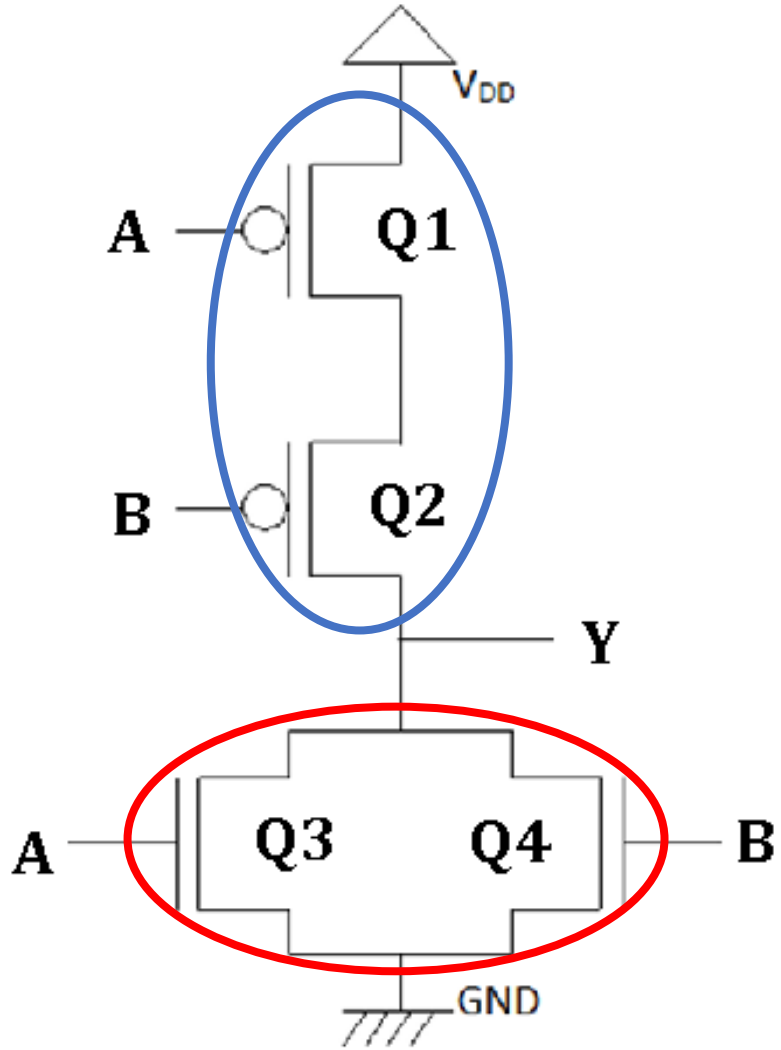


Takeaways:

- When any of the input in **LOW** the output is **HIGH**.
- The Boolean expression for a NAND gate is $\mathbf{Y = \overline{AB}}$.
- The whole circuit can be divided in to two parts, namely PDN and the PUN.
- **PDN/NMOS structure is designed in a series structure.**
- **PUN/PMOS structure is designed in a parallel structure.**

NOR Gate using CMOS (CMOS Inverter)

A	B	Q1 PMOS	Q2 PMOS	Q3 NMOS	Q4 NMOS	Y
0	0	ON	ON	OFF	OFF	$+V_{DD}$
0	1	ON	OFF	OFF	ON	Low
1	0	OFF	ON	ON	OFF	Low
1	1	OFF	OFF	ON	ON	Low



Takeaways:

- When any of the input is HIGH the output is LOW.
- The Boolean expression is $\mathbf{Y = \overline{A + B}}$.
- Similarly, the circuit can be divided in to two parts namely, PDN and PUN.
- **PDN/NMOS structure is designed in parallel structure.**
- **PUN/PMOS structure is designed in series structure.**

Function Implementation with CMOS Logic

- Till now we have seen how to implement two know Boolean expression.
- We have learnt that, for each expression we need to separately have a PUN and a PDN.
- Furthermore, we have seen that, for PUN network a product is implemented using the parallel combination where as, a sum is implemented with a series combination.
- And for the PDN network a product is implemented using series combination and a sum is implemented using parallel combination.
- So can we use these observations to help build some more complex function such as;

$$F = \overline{A + BC}$$

Function design will be shown in class….

Digital System Design

Designing a digital system requires following the steps to be performed:

- Look for the problem statement.
- Find out the inputs and outputs of the system.
- Related the input of your system with the outputs.
- Develop a truth table/ behavior table for your system using input and output relationship that you have established.
- From your truth-table generate a standard output expression, relating the inputs to the outputs.
- Reduce the output expression using either Boolean Algebra or K-MAP.
- Write down your reduced expression (minimized expression)
- Design the system circuit with combinational logic gates.
- Convert the combinational logic gates to transistor level schematic (CMOS schematic)

References

1. Thomas L. Floyd, “Digital Fundamentals” 11th edition, Prentice Hall – Pearson Education.

Thank You