ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Electronics for Information Technology

(Điện tử cho Công nghệ Thông tin)

IT3420E

Đỗ Công Thuần

Department of Computer Engineering

Email: thuandc@soict.hust.edu.vn

General Information

- Course: **Electronics for Information Technology**
- ID Number: IT3420
- Credits: 2 (2-1-0-4)
- Lecture/Exercise: 32/16 hours (48 hours, 16 weeks)
- Evaluation:
  - Midterm examination and weekly assignment: **50%**
  - Final examination: **50%**
- Learning Materials:
  - Lecture slides
  - Textbooks
    - ***Introductory Circuit Analysis*** (2015), 10th – 13th ed., Robert L. Boylestad
    - ***Electronic Device and Circuit Theory*** (2013), 11th ed., Robert L. Boylestad, Louis Nashelsky
    - ***Microelectronics Circuit Analysis and Design*** (2006), 4th ed., Donald A. Neamen
    - ***Digital Electronics: Principles, Devices and Applications*** (2007), Anil K. Maini

Contact Your Instructor

- You can reach me through office in **Room 802, B1 Building**, HUST.
  - You should make an appointment by email before coming.
  - If you have urgent things, just come and meet me!
- You can also reach me at the following **email** any time. This is the best way to reach me!
  - thuandc@soict.hust.edu.vn

Course Contents

- The Concepts of Electronics for IT
- **Chapter 1**: Passive Electronic Components and Applications
- **Chapter 2**: Semiconductor Components and Applications
- **Chapter 3**: Operational Amplifiers
- **Chapter 4**: Fundamentals of Digital Circuits
- **Chapter 5**: Logic Gates
- **Chapter 6**: Combinational Logic
- **Chapter 7**: Sequential Logic

Chapter 6: **Combinational Logic**

1. Concepts
2. Combinational Circuits
   - Arithmetic Circuits
   - Multiplexers
   - Encoders
   - Demultiplexers
   - Decoders

References:

Digital electronics: Principles, Devices, and Applications, Anil Kumar Maini 2007 John Wiley & Sons

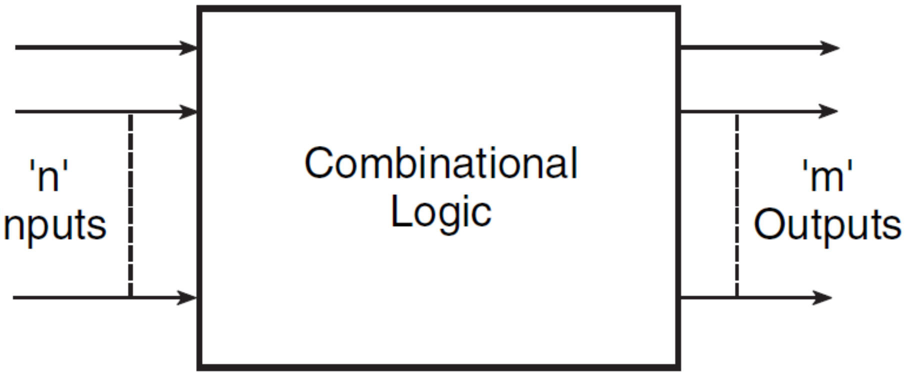
Fundamentals of Logic Design, Seventh Edition, Charles H. Roth, Jr. and Larry L. Kinney

Digital Fundamentals, Thomas L. Floyd, Eleventh Edition, Pearson Education Limited 2015

Contents

Definition

- A combinational circuit is one where the output at any time **depends only on the present combination of inputs** at that point of time with total **disregard to the past state of the inputs**.



Implementing Combinational Logic

1. Statement of the problem.
2. Identification of input and output variables.
3. Expressing the relationship between the input and output variables.
4. Construction of a truth table to meet input–output requirements.
5. Writing Boolean expressions for various output variables in terms of input variables.
6. Minimization of Boolean expressions.
7. Implementation of minimized Boolean expressions

Guidelines for Hardware Implementation

1. The implementation should have the minimum number of gates, with the gates used having the minimum number of inputs.
2. There should be a minimum number of interconnections, and the propagation time should be the shortest.
3. Limitation on the driving capability of the gates should not be ignored.

Combinational Circuits

- Arithmetic Circuits
- Multiplexers
- Encoders
- Demultiplexers
- Decoders

Arithmetic Circuits

a. Adder
b. Subtractor
c. Comparator

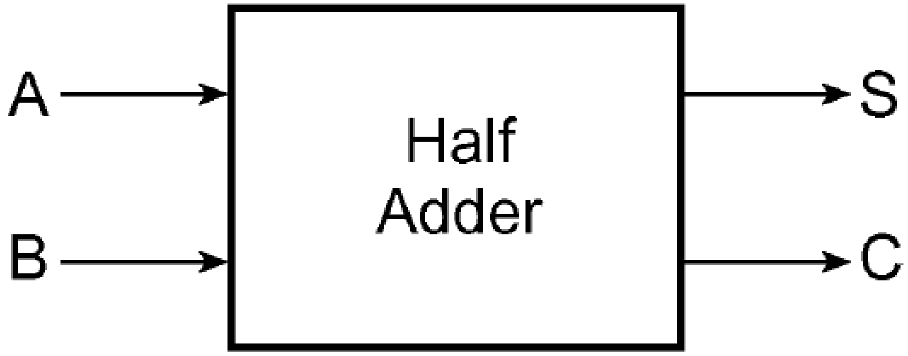
a. Adder

- Used to perform addition on binary numbers
- Basic building blocks:
    - Half-Adder
    - Full-Adder
    - N-Bit Adder

a. Adder → Half-Adder

- A half-adder is an arithmetic circuit block that can be used to add 2 bits.

- Block diagram:



a. Adder → Half-Adder

- Truth table:

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

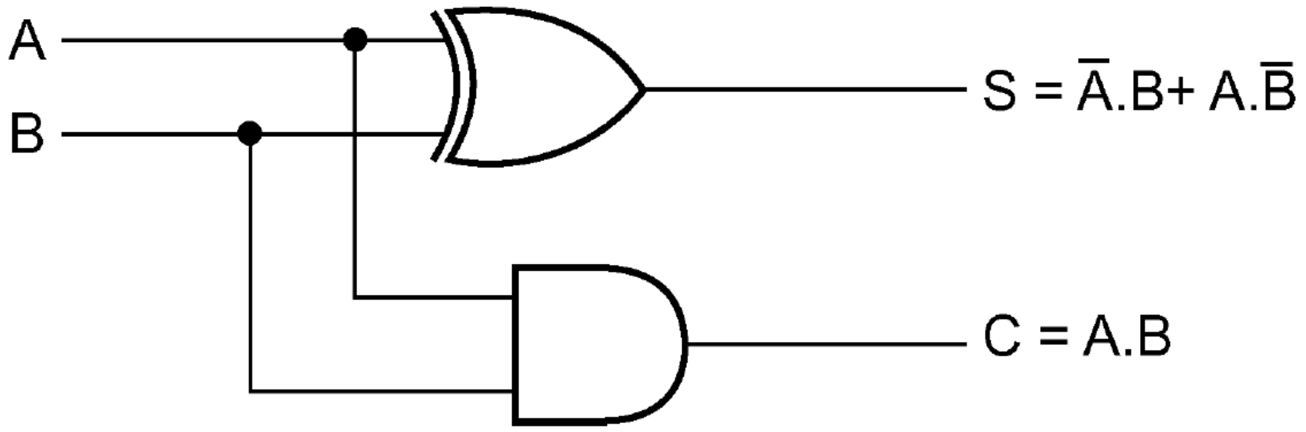
- The Boolean expressions for the SUM and CARRY outputs:

$$\text{SUM } S = A.\overline{B} + \overline{A}.B$$

$$\text{CARRY } C = A.B$$

a. Adder → Half-Adder

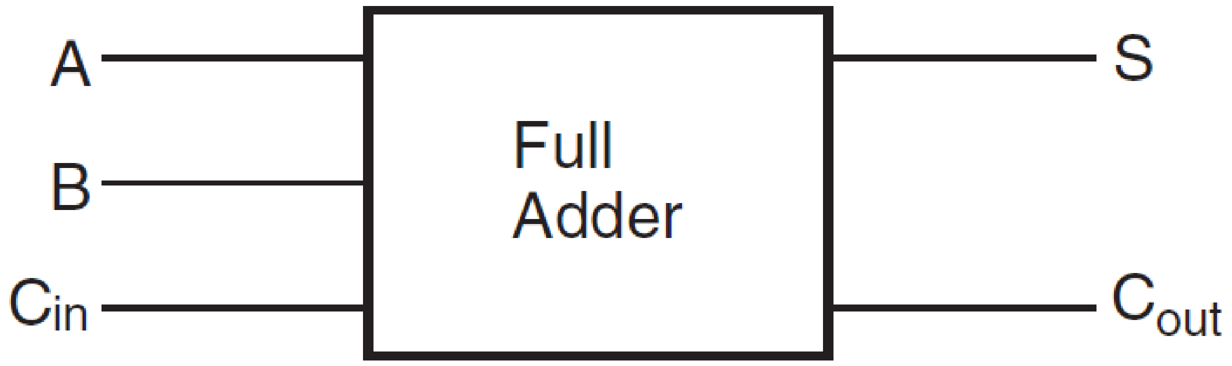
- Logic implementation of a half-adder by using XOR and AND gates.
- Could also be implemented by using an appropriate arrangement of either NAND or NOR gates.



a. Adder → Full-Adder

- A full adder circuit is an arithmetic circuit block that can be used to add three bits to produce a SUM and a CARRY output.

- Block diagram:



a. Adder → Full-Adder

- Truth table:

A	B	C_{in}	SUM (S)	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

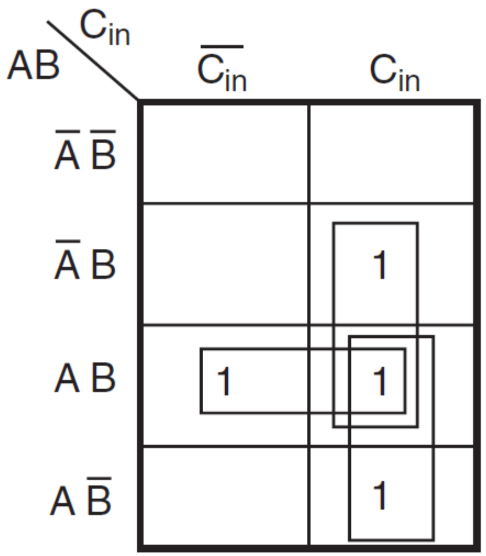
- The Boolean expressions for the SUM and CARRY outputs:

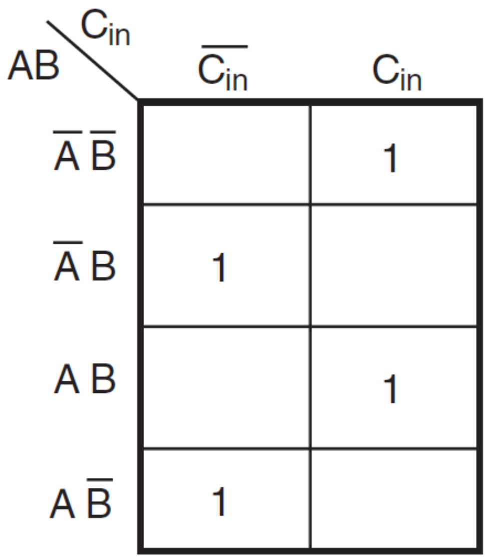
$$S = \overline{A}.\overline{B}.C_{in} + \overline{A}.B.\overline{C}_{in} + A.\overline{B}.\overline{C}_{in} + A.B.C_{in}$$

$$C_{out} = \overline{A}.B.C_{in} + A.\overline{B}.C_{in} + A.B.\overline{C}_{in} + A.B.C_{in}$$

a. Adder → Full-Adder

- Karnaugh maps for the sum and carry-out (C_{out}):





- Simplified expressions:

$$C_{out} = A.B + C_{in}.(\overline{A}.B + A.\overline{B})$$

$$S = \overline{C}_{in}.(\overline{A}.B + A.\overline{B}) + C_{in}.(A.B + \overline{A}.\overline{B})$$

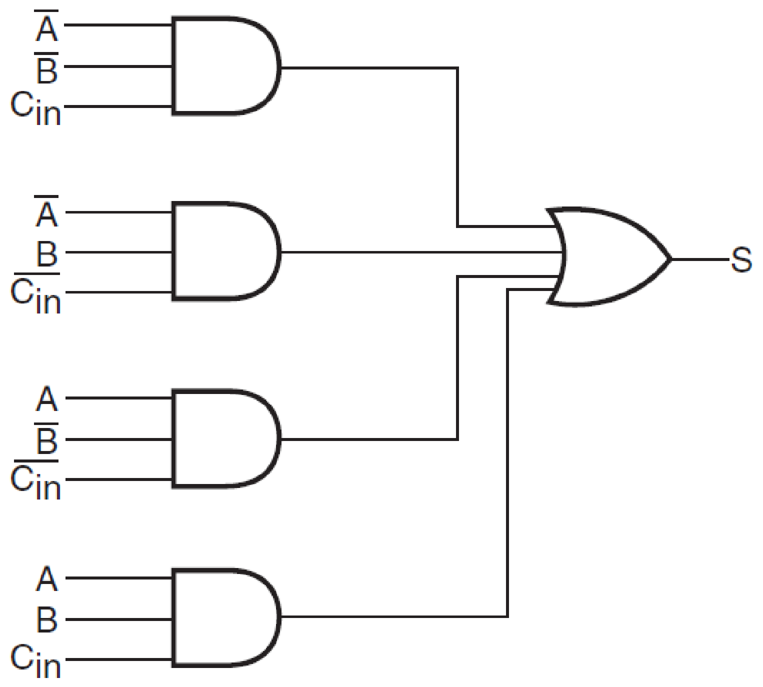
$$S = \overline{C}_{in}.(\overline{A}.B + A.\overline{B}) + C_{in}.(\overline{\overline{A}.B + A.\overline{B}})$$

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

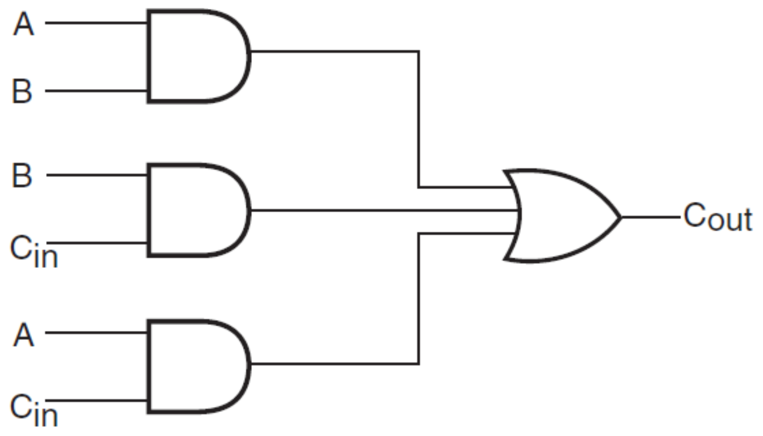
a. Adder → Full-Adder

- Logic circuit diagram:

$$S = \overline{C}_{in}.(\overline{A}.B + A.\overline{B}) + C_{in}.(A.B + \overline{A}.\overline{B})$$



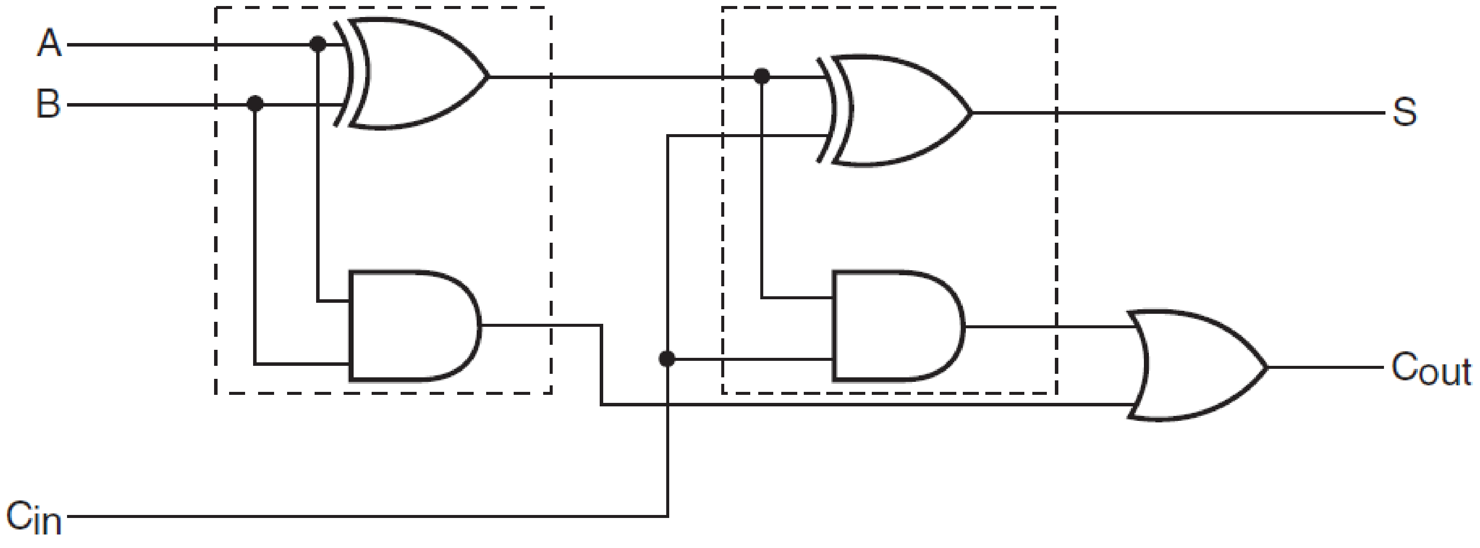
$$C_{out} = A.B + C_{in}.(\overline{A}.B + A.\overline{B})$$



a. Adder → Full-Adder

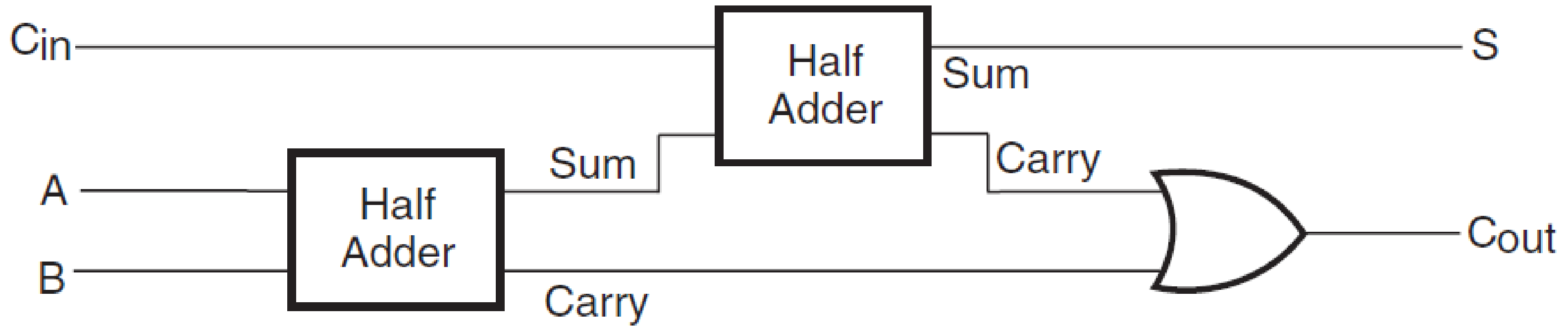
$$S = \overline{C}_{in}.(\overline{A}.B + A.\overline{B}) + C_{in}.(\overline{\overline{A}.B + A.\overline{B}})$$

$$C_{out} = A.B + C_{in}.(\overline{A}.B + A.\overline{B})$$



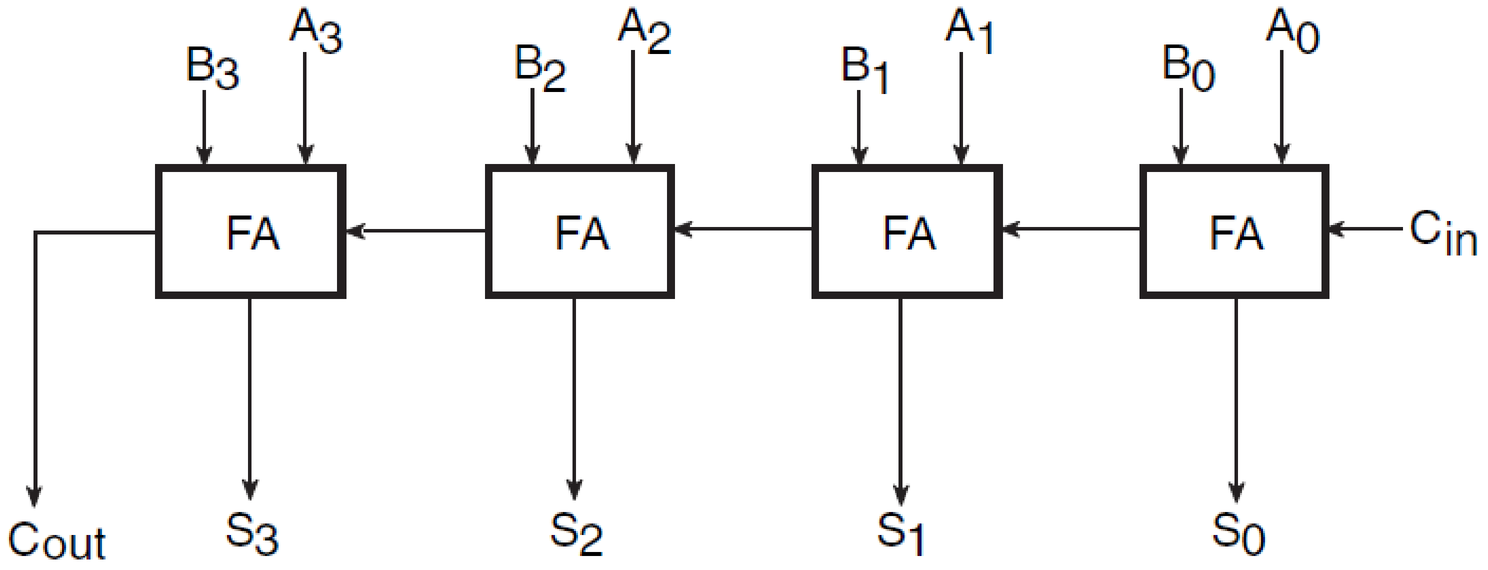
a. Adder → Full-Adder

- Logic implementation of a full adder with half-adders



a. Adder → N-bit Adder

- A cascade arrangement of full adders can be used to construct adders capable of adding binary numbers with a larger number of bits.
- Example: 4-bit binary adder



Arithmetic Circuits

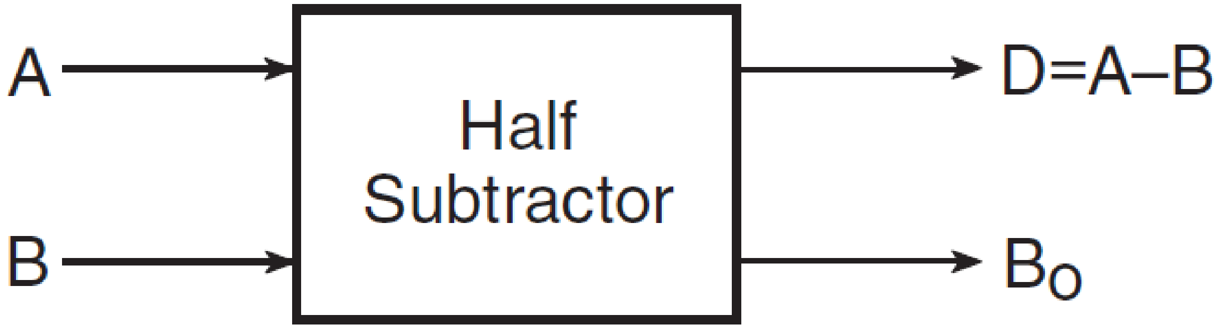
a. Adder
b. Subtractor
c. Comparator

b. Subtractor

- Used to substract one binary number from another
- Results: a difference output and a borrow output
    - Half-Subtractor
    - Full Subtractor
    - Adder–Subtractor

b. Subtractor → Half-Subtractor

- A half-subtractor can be used to subtract one binary **digit** from another to produce a difference output and a borrow output.
- Block diagram:
  - D: Difference
  - B_o: Borrow



b. Subtractor → Half-Subtractor

- Truth Table:

A	B	D	B_O
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

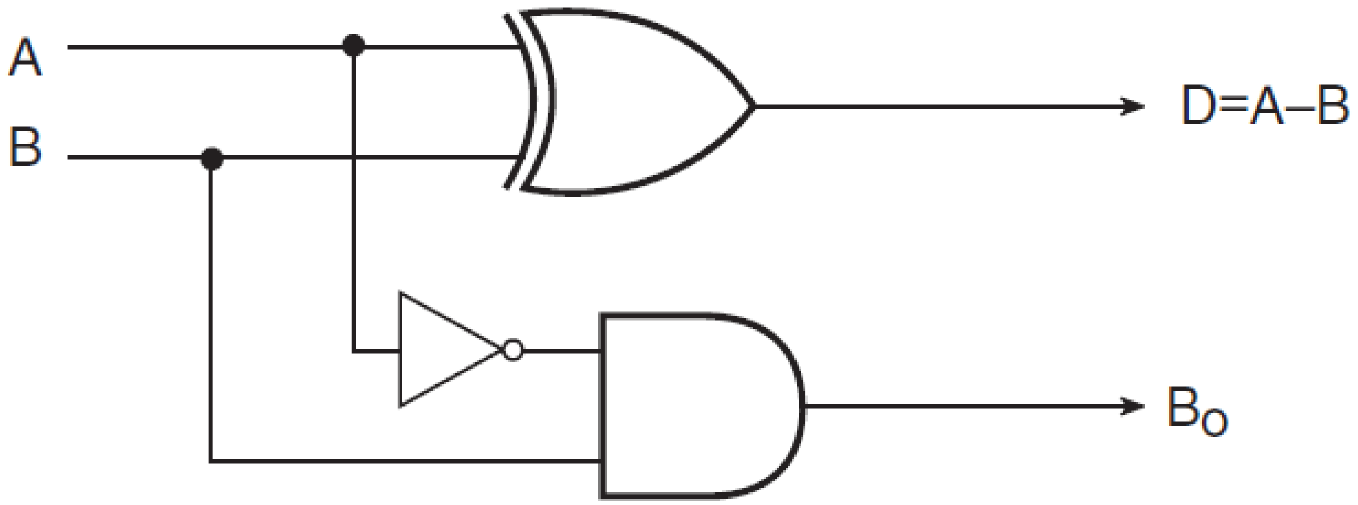
- The Boolean expressions for the D and B_o outputs:

$$D = \overline{A}.B + A.\overline{B}$$

$$B_o = \overline{A}.B$$

b. Subtractor → Half-Subtractor

- Logic diagram of a half-subtractor by using XOR, NOT, and AND gates.

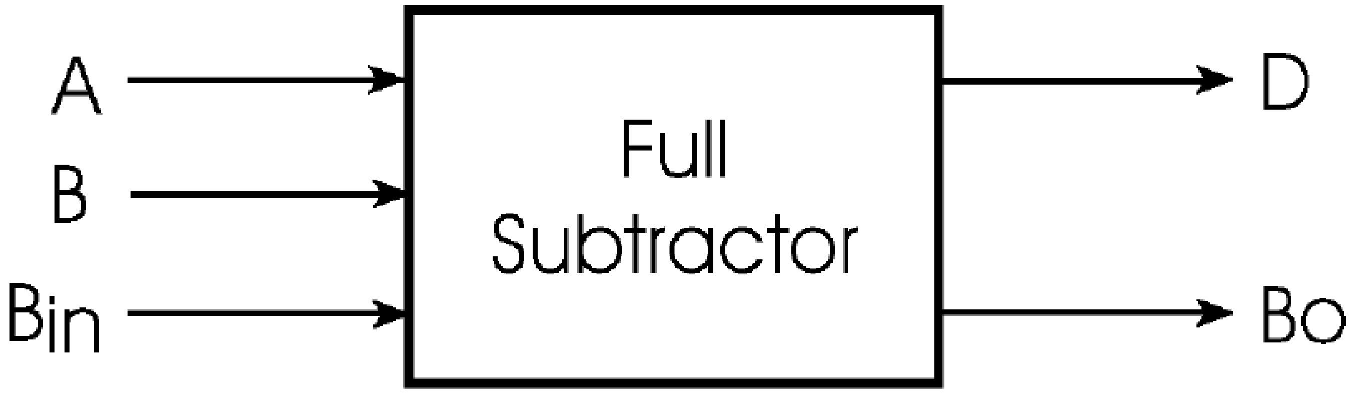


$$D = \overline{A}.B + A.\overline{B}$$

$$B_o = \overline{A}.B$$

b. Subtractor → Full Subtractor

- Used to perform subtraction operation on two bits.
- Also considers whether a ‘1’ has already been borrowed by the previous adjacent lower minuend bit or not.
- Block diagram:



b. Subtractor → Full Subtractor

- Truth Table:

Minuend (A)	Subtrahend (B)	Borrow In (B_{in})	Difference (D)	Borrow Out (B_o)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

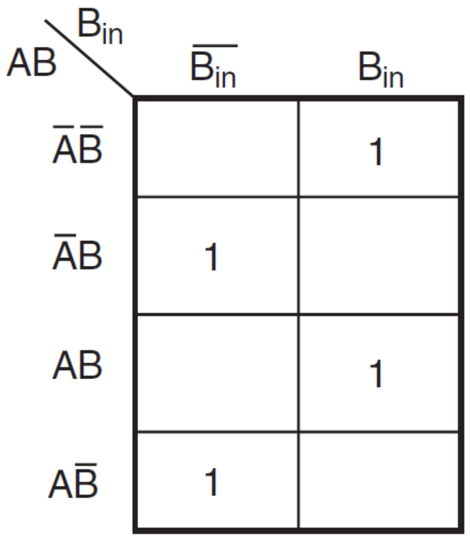
- The Boolean expressions for the D and B_o outputs:

$$D = \overline{A}.\overline{B}.B_{\text{in}} + \overline{A}.B.\overline{B}_{\text{in}} + A.\overline{B}.\overline{B}_{\text{in}} + A.B.B_{\text{in}}$$

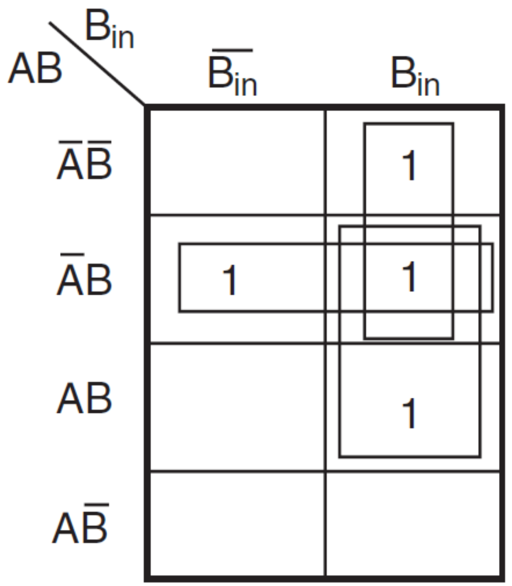
$$B_{\text{o}} = \overline{A}.\overline{B}.B_{\text{in}} + \overline{A}.B.\overline{B}_{\text{in}} + \overline{A}.B.B_{\text{in}} + A.B.B_{\text{in}}$$

b. Subtractor → Full Subtractor

- Logic simplification:
  - K-maps for D and B_{out}:



$AB \backslash B_{in}$	$\overline{B_{in}}$	B_{in}
$\overline{A}\overline{B}$		1
$\overline{A}B$	1	
AB		1
$A\overline{B}$	1	



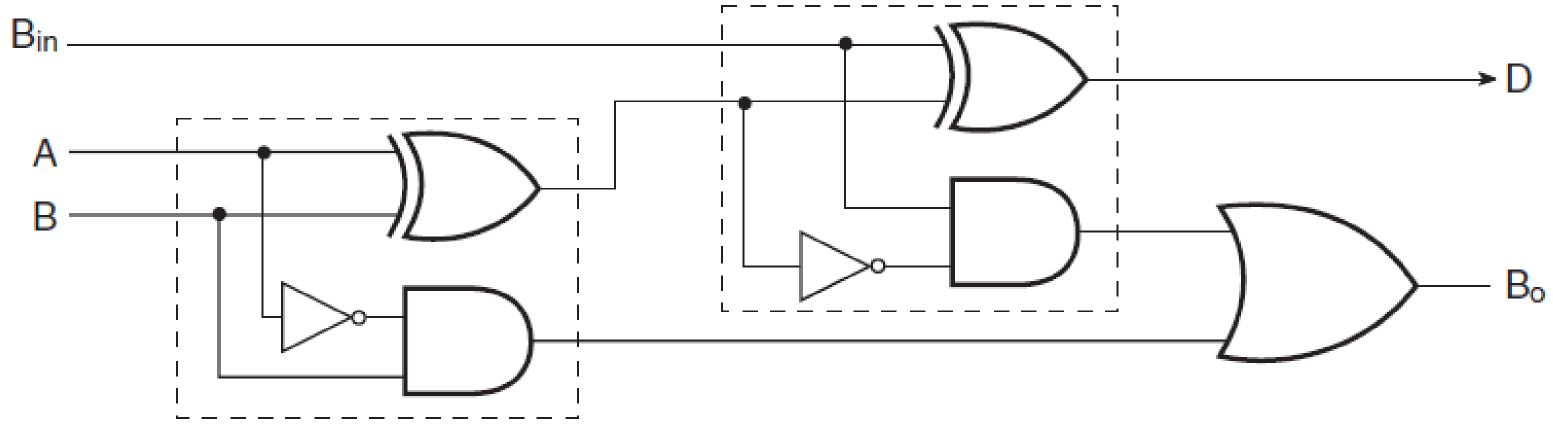
$AB \backslash B_{in}$	$\overline{B_{in}}$	B_{in}
$\overline{A}\overline{B}$		1
$\overline{A}B$	1	1
AB		1
$A\overline{B}$		

$$B_o = \overline{A}.B + \overline{A}.B_{in} + B.B_{in}$$

$$D = \overline{A}.\overline{B}.B_{in} + \overline{A}.B.\overline{B}_{in} + A.\overline{B}.\overline{B}_{in} + A.B.B_{in}$$

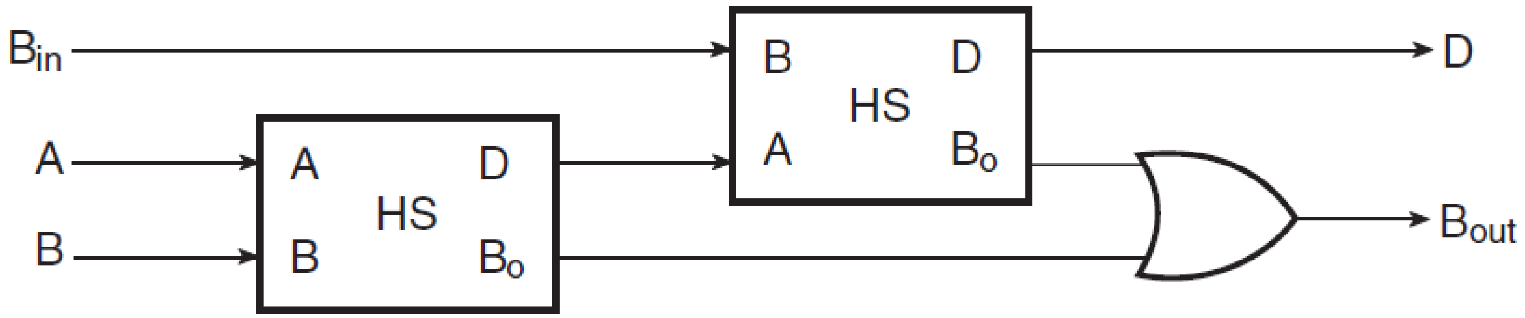
b. Subtractor → Full Subtractor

- Logic implementation of a full subtractor with half-subtractors:



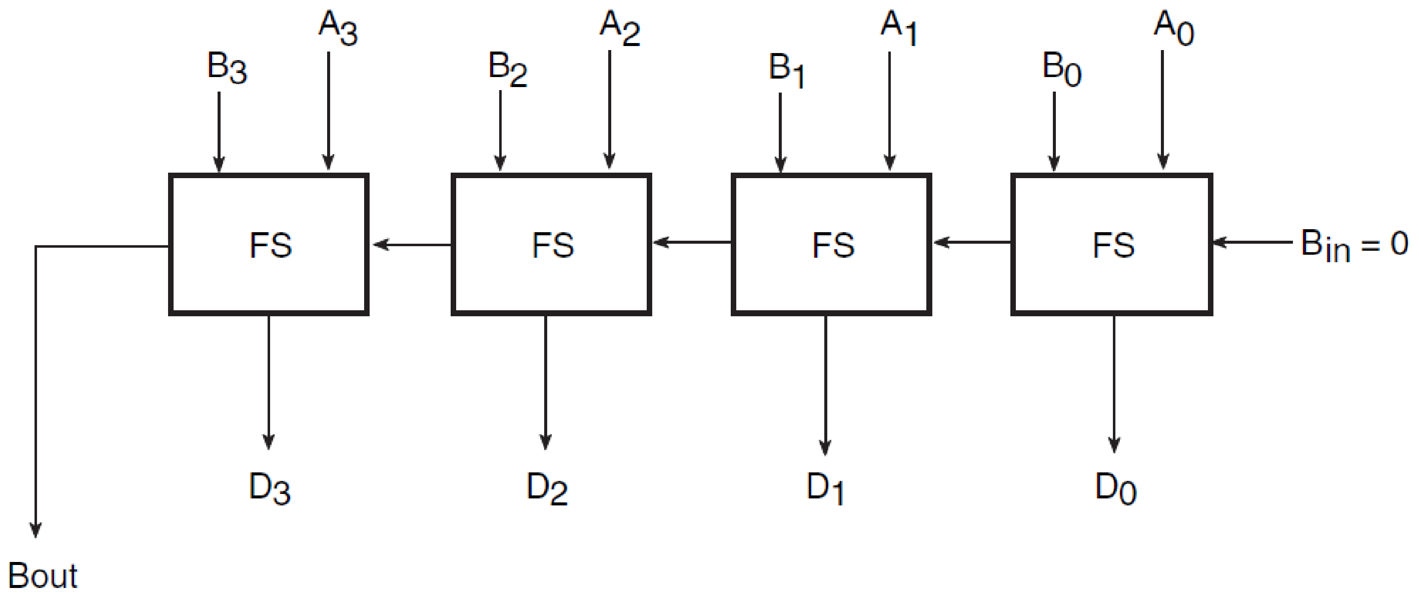
b. Subtractor → Full Subtractor

- Logic implementation of a full subtractor with half-subtractors.



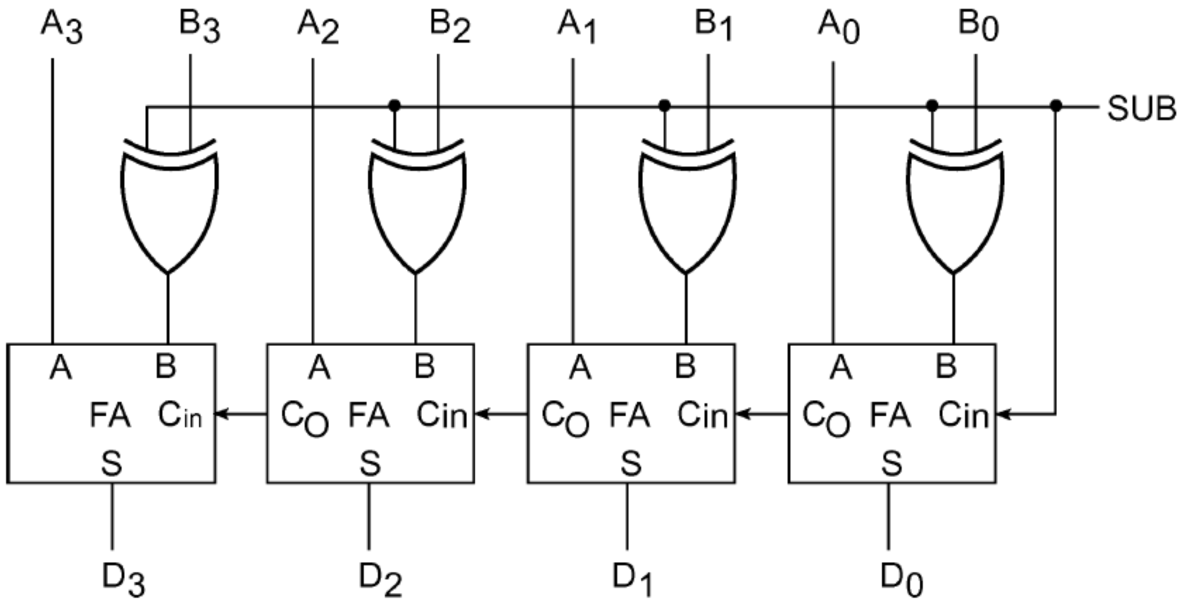
b. Subtractor → N-bit Subtractor

- More than one full subtractor can be connected in cascade to perform subtraction on two larger binary numbers.
- Example: 4-bit Subtractor



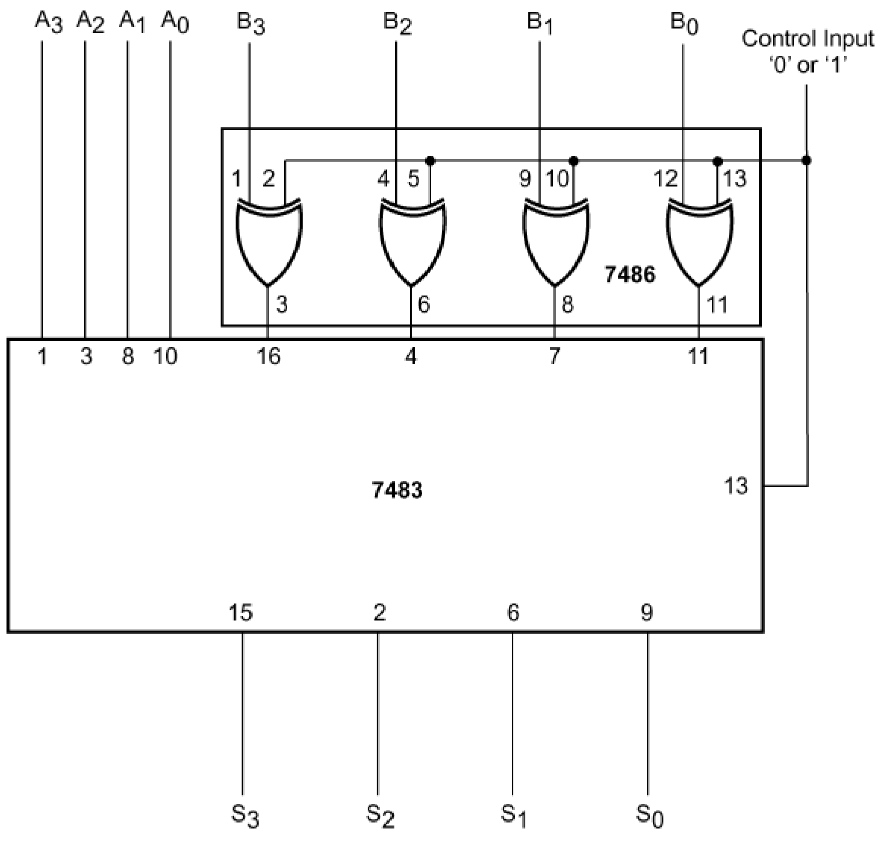
b. Subtractor → Adder–Subtractor

- 4-bit adder-subtractor:
  - SUB = 0
    - A+B
  - SUB=1
  - C_{in}=1
    - A-B



b. Subtractor → Adder–Subtractor

- 4-bit adder-subtractor:

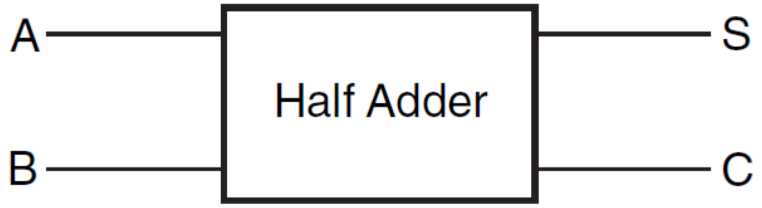


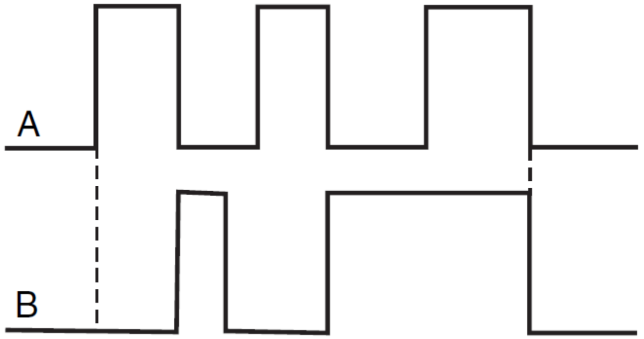
- IC 74LS83
- 4 2-input XOR gates

  (IC 74LS86)

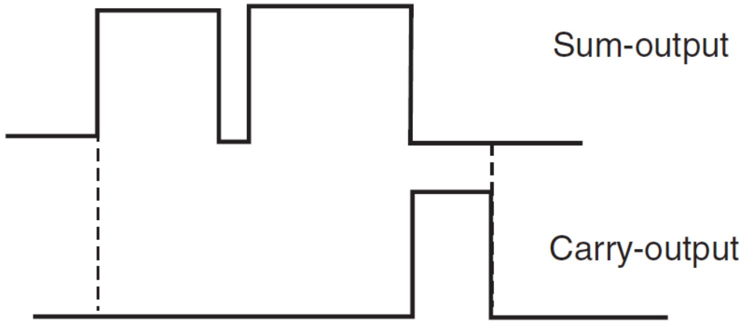
Example 1

- For the half-adder circuit, the inputs applied at A & B.



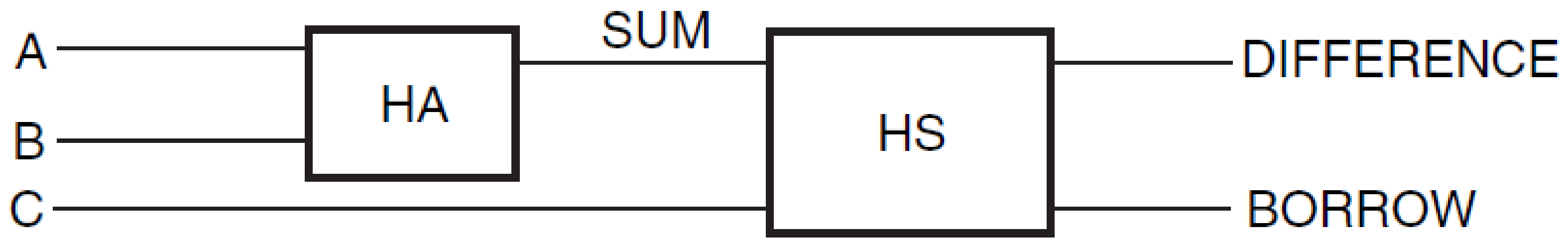


- Plot the corresponding SUM and CARRY outputs on the same scale.



Example 2

- For the circuit:



- Write the simplified Boolean expressions for DIFFERENCE and BORROW outputs.

Example 2

- The simplified Boolean expressions for D and B_o:

$$\text{DIFFERENCE output} = X \oplus Y = \overline{X}.Y + X.\overline{Y}$$

$$\text{BORROW output} = \overline{X}.Y$$

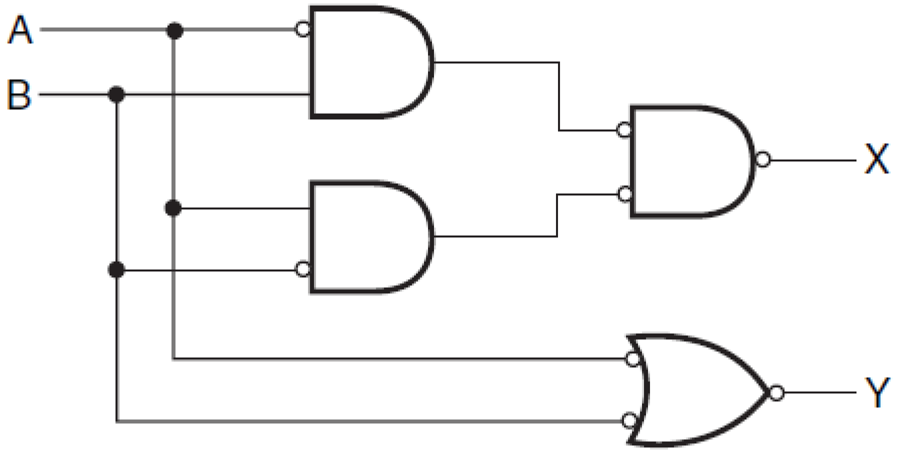
$$X = \overline{A}.B + A.\overline{B} \text{ and } Y = C$$

$$\begin{aligned}\text{DIFFERENCE output} &= (\overline{\overline{A}.B + A.\overline{B}}).C + (\overline{A}.B + A.\overline{B}).\overline{C} \\ &= (A.B + \overline{A}.\overline{B}).C + (\overline{A}.B + A.\overline{B}).\overline{C} \\ &= A.B.C + \overline{A}.\overline{B}.C + \overline{A}.B.\overline{C} + A.\overline{B}.\overline{C}\end{aligned}$$

$$\begin{aligned}\text{BORROW output} = \overline{X}.Y &= (\overline{\overline{A}.B + A.\overline{B}}).C \\ &= (A.B + \overline{A}.\overline{B}).C = A.B.C + \overline{A}.\overline{B}.C\end{aligned}$$

Example 3

- For the logic diagram:



- Identify the logic function.

$$X = \overline{\overline{(\overline{A}.B)}.\overline{(A.\overline{B})}} = (\overline{\overline{\overline{A}.B}} + \overline{\overline{A.\overline{B}}}) = \overline{A}.B + A.\overline{B}$$

$$Y = \overline{(\overline{A} + \overline{B})} = A.B$$

Arithmetic Circuits

a. Adder
b. Subtractor
c. Comparator

c. Comparator

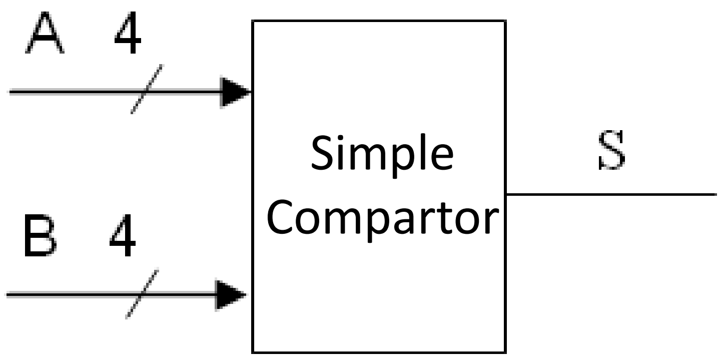
- Used to compare the magnitude of 2 givien binary numbers.
- 2 types of comparators:
  - Simple comparator: the output is in the form of 1 binary variable representing the condition: *equal to* or *different from*.
  - Full comparator: the output is in the form of 3 binary variables representing the conditions: *less than*, *equal to* or *greater than*.

c. Comparator → Simple Comparator

- 2 giving binary numbers, A and B:

A	$a_3\ a_2\ a_1\ a_0$
B	$b_3\ b_2\ b_1\ b_0$

- Output S:
    - S = 1 <=> A = B
    - S = 0 <=> A ≠ B



c. Comparator → Simple Comparator

- We have:

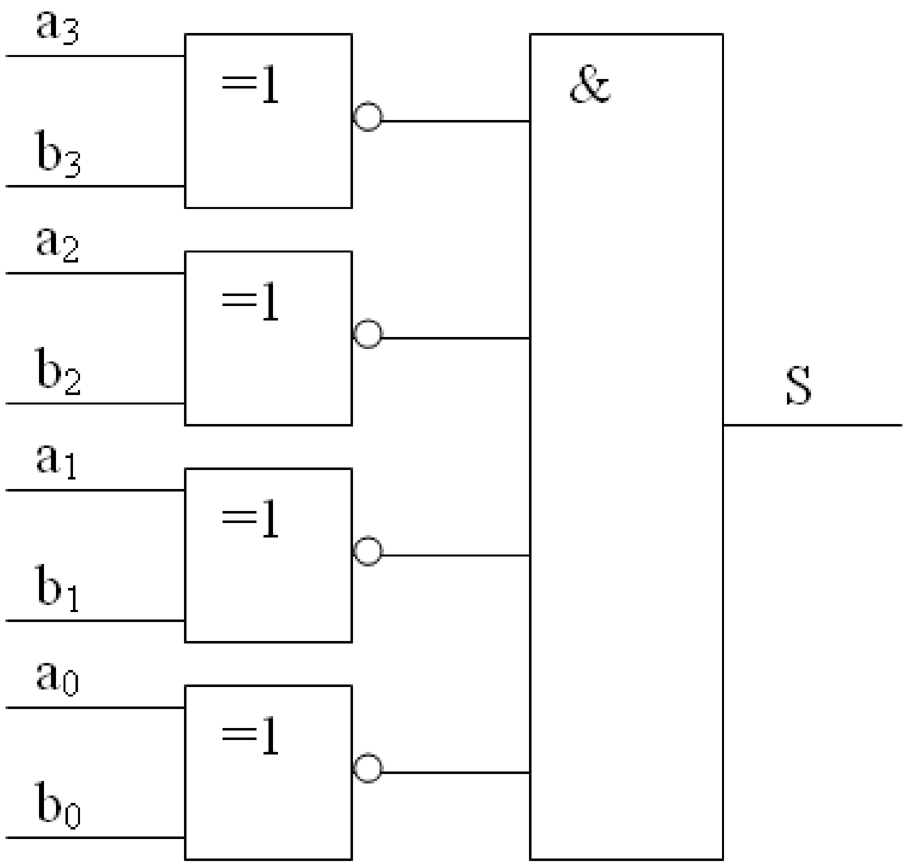
$$A = B \leftrightarrow \begin{cases} a_3 = b_3 \\ a_2 = b_2 \\ a_1 = b_1 \\ a_0 = b_0 \end{cases} \leftrightarrow \begin{cases} a_3 \oplus b_3 = 0 \\ a_2 \oplus b_2 = 0 \\ a_1 \oplus b_1 = 0 \\ a_0 \oplus b_0 = 0 \end{cases} \leftrightarrow \begin{cases} \overline{a_3 \oplus b_3} = 1 \\ \overline{a_2 \oplus b_2} = 1 \\ \overline{a_1 \oplus b_1} = 1 \\ \overline{a_0 \oplus b_0} = 1 \end{cases}$$

- The Boolean expression for the S output:

$$S = (\overline{a_3 \oplus b_3})(\overline{a_2 \oplus b_2})(\overline{a_1 \oplus b_1})(\overline{a_0 \oplus b_0})$$

c. Comparator → Simple Comparator

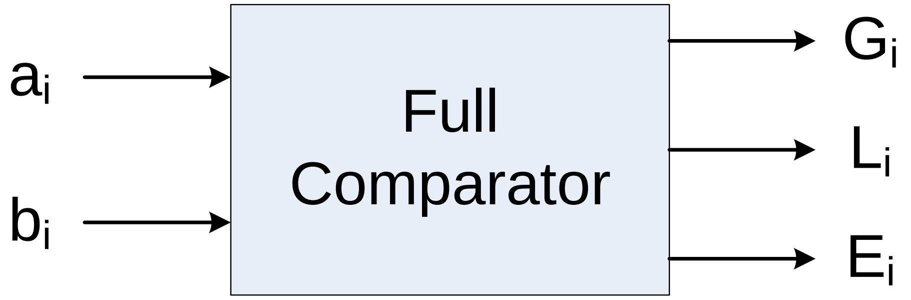
- Logic implementation:



c. Comparator → Full Comparator

- 2-bit full comparator:
  - Input (1-bit inputs): a_i and b_i
  - Output: G_i, L_i, E_i
    - $a_i > b_i$ <=> $G_i = 1$ & $E_i, L_i = 0$
    - $a_i < b_i$ <=> $L_i = 1$ & $E_i, G_i = 0$
    - $a_i = b_i$ <=> $E_i = 1$ & $G_i, L_i = 0$

  - Block diagram:



c. Comparator → 2-bit Full Comparator

- Truth table:

a_i	b_i	G_i	L_i	E_i
0	0	0	0	1
0	1	0	1	0
1	0	1	0	0
1	1	0	0	1

- The Boolean expressions for the 3 outputs:

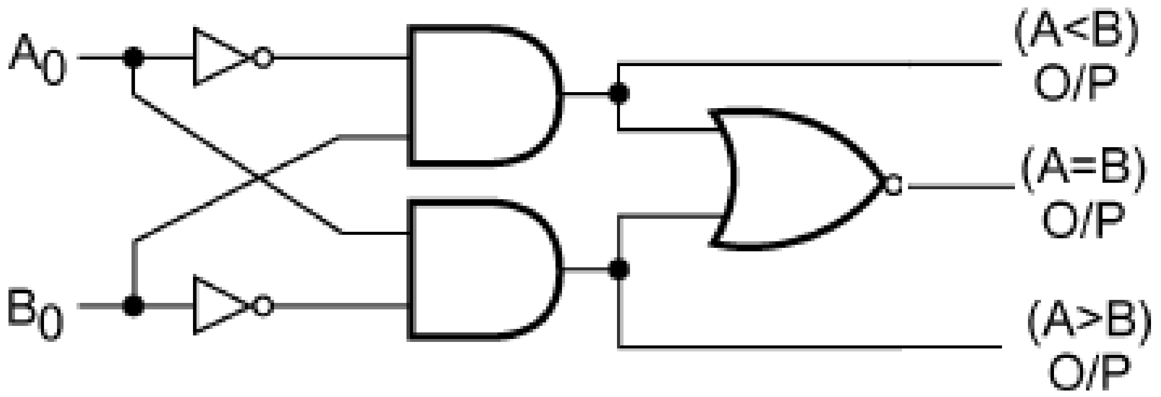
$$G_i = a_i.\overline{b_i}$$

$$L_i = \overline{a_i}.b_i$$

$$E_i = \overline{a_i \oplus b_i}$$

c. Comparator → 2-bit Full Comparator

- Logic implementation:



c. Comparator → 2-bit Full Comparator

- Input (2-bit inputs): $A\,(A_1A_0)$
  $B\,(B_1B_0)$

- Output: $X, Y, Z \Rightarrow A = B, A > B, A < B$

$$X = 1,\ Y = 0,\ Z = 0 \qquad A = B$$

$$X = 0,\ Y = 1,\ Z = 0 \qquad A > B$$

$$X = 0,\ Y = 0,\ Z = 1 \qquad A < B$$

c. Comparator → 2-bit Full Comparator

- The Boolean expressions for the outputs:

$$X = x_1 . x_0$$

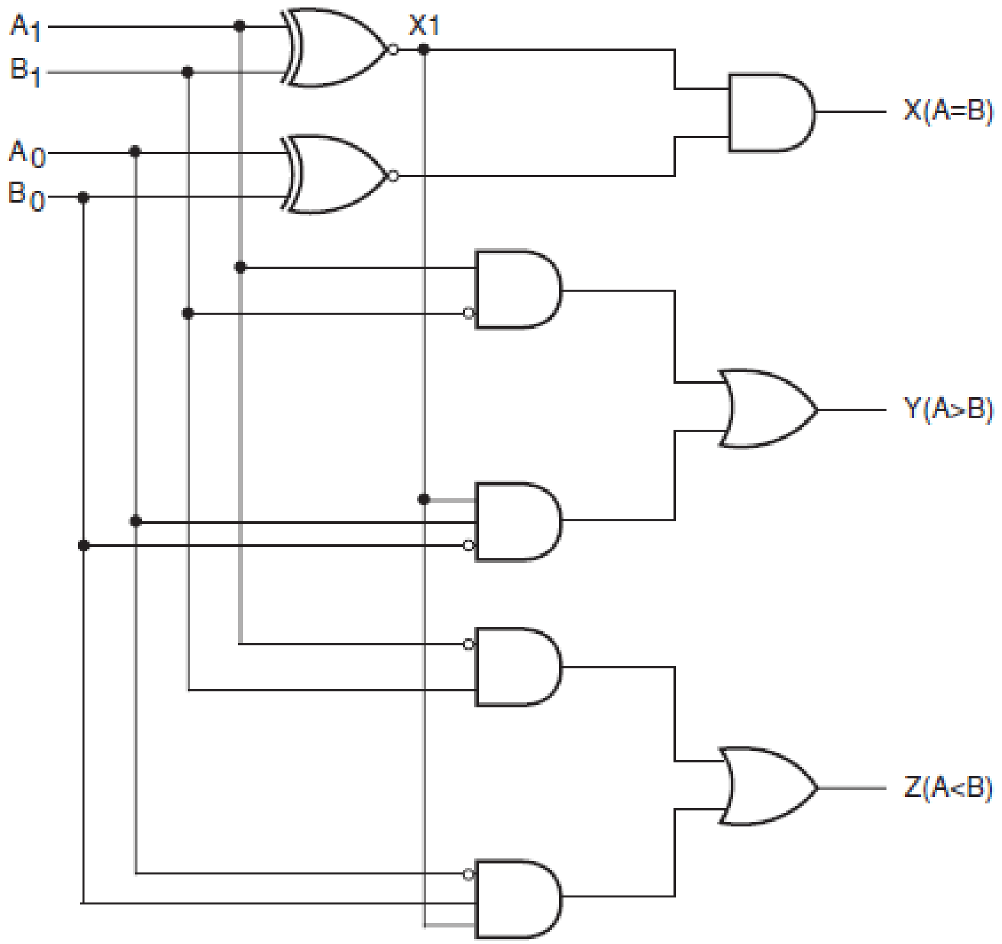
$$x_0 = A_0 . B_0 + \overline{A_0} . \overline{B_0}$$
$$x_1 = A_1 . B_1 + A_1 . \overline{B_1}$$

$$Y = A_1 . \overline{B_1} + x_1 . A_0 . \overline{B_0}$$

$$Z = \overline{A_1} . B_1 + x_1 . \overline{A_0} . B_0$$

c. Comparator → 2-bit Full Comparator

- Logic implemention:



Combinational Circuits

- Arithmetic Circuits
- Multiplexers
- Encoders
- Demultiplexers
- Decoders

Multiplexers

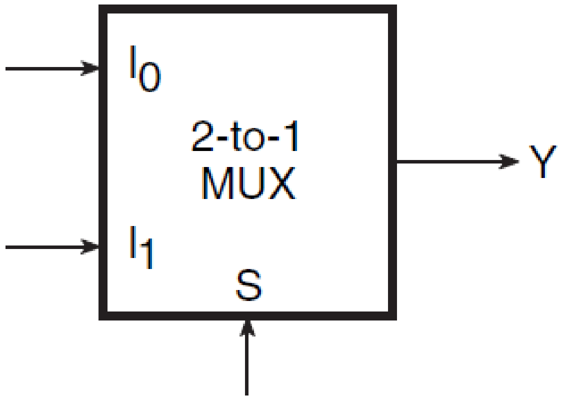
- A multiplexer or MUX is also called data selector.
- More than one input line, one output line, and more than one selection line.
- Used to select binary information present on any one of the input lines, depending upon the logic status of the selection inputs, and routes it to the output line.
- Examples:
  - MUX 2-1
  - MUX 4-1
  - MUX 8-1
  - MUX 16-1
  - MUX 2^n – 1, where n is the number of selection lines

Multiplexers

- Applications of MUX:
  - Implementing an n-variable Boolean function with 2^n–to–1 MUX.
  - Implementing an (n+1)-variable Boolean function with 2^n–to–1 MUX
  - Parallel-to-Serial Data Conversion
  - Cascading Multiplexer Circuits

2-to-1 MUX

- Block diagram:



- Truth table:

S	Y
0	I_0
1	I_1

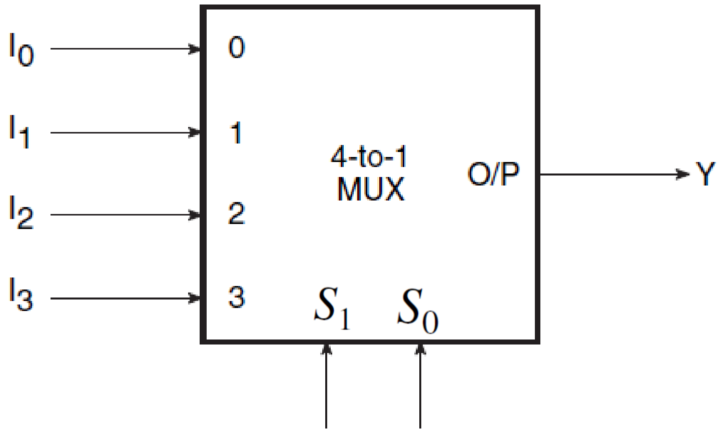
- Output:

$$S = 0 \quad Y = I_0$$
$$S = 1 \quad Y = I_1$$

$$Y = \overline{S} I_0 + S I_1$$

4-to-1 MUX

- Block diagram:



- Truth table:

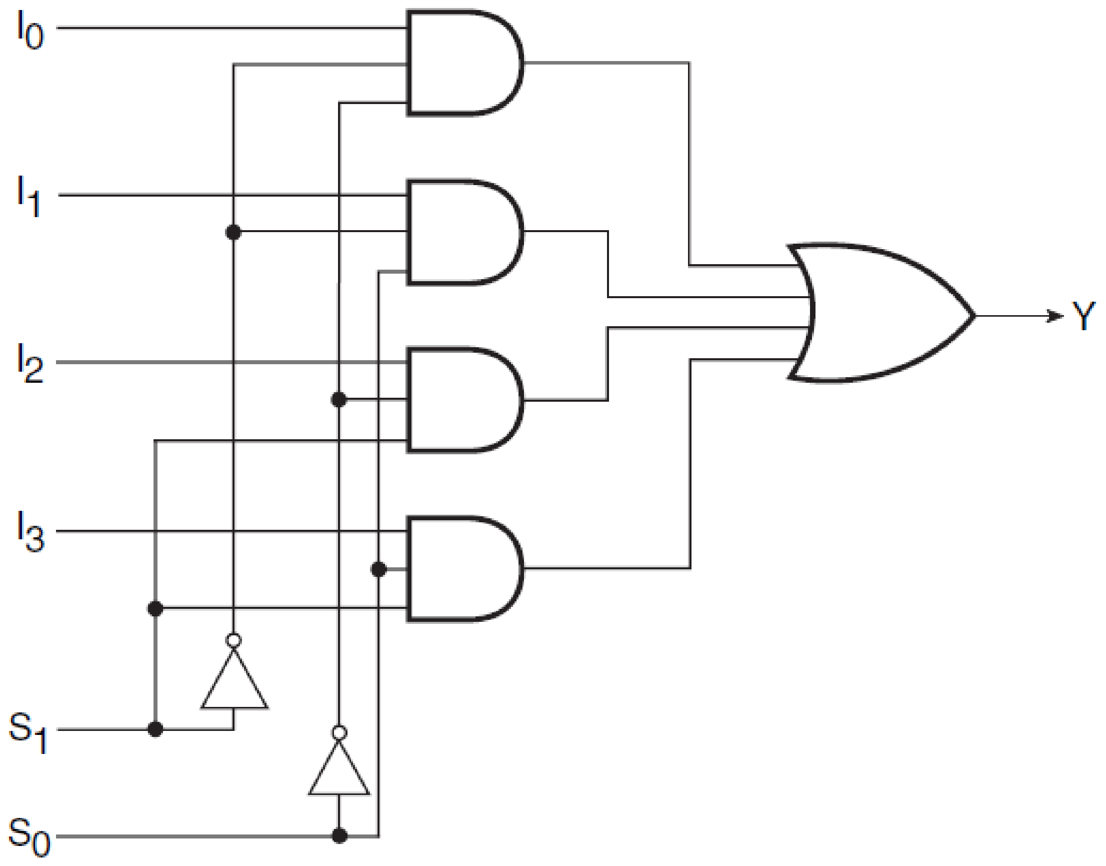
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

- Output:

$$Y = I_0.\overline{S_1}.\overline{S_0} + I_1.\overline{S_1}.S_0 + I_2.S_1.\overline{S_0} + I_3.S_1.S_0$$

4-to-1 MUX

- Logic diagram:



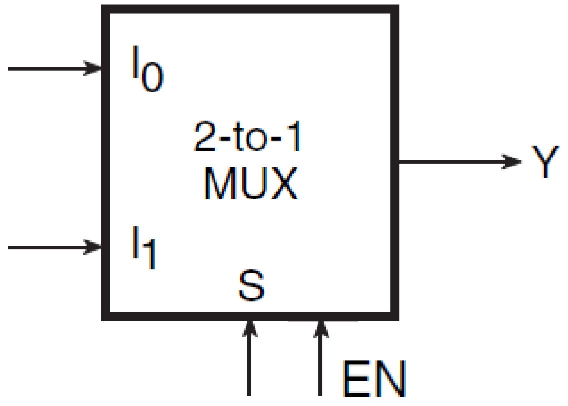
8-to-1 MUX

- Output:

$$Y = I_0.\overline{S_2}.\overline{S_1}.\overline{S_0} + I_1.\overline{S_2}.\overline{S_1}.S_0 + I_2.\overline{S_2}.S_1.\overline{S_0}$$
$$+ I_3.\overline{S_2}.S_1.S_0 + I_4.S_2.\overline{S_1}.\overline{S_0} + I_5.S_2.\overline{S_1}.S_0$$
$$+ I_6.S_2.S_1.\overline{S_0} + I_7.S_2.S_1.S_0$$

2-to-1 MUX with an ENABLE Input

- Block diagram:



- Truth table:

S	EN	Y
X	0	0
0	1	I_0
1	1	I_1

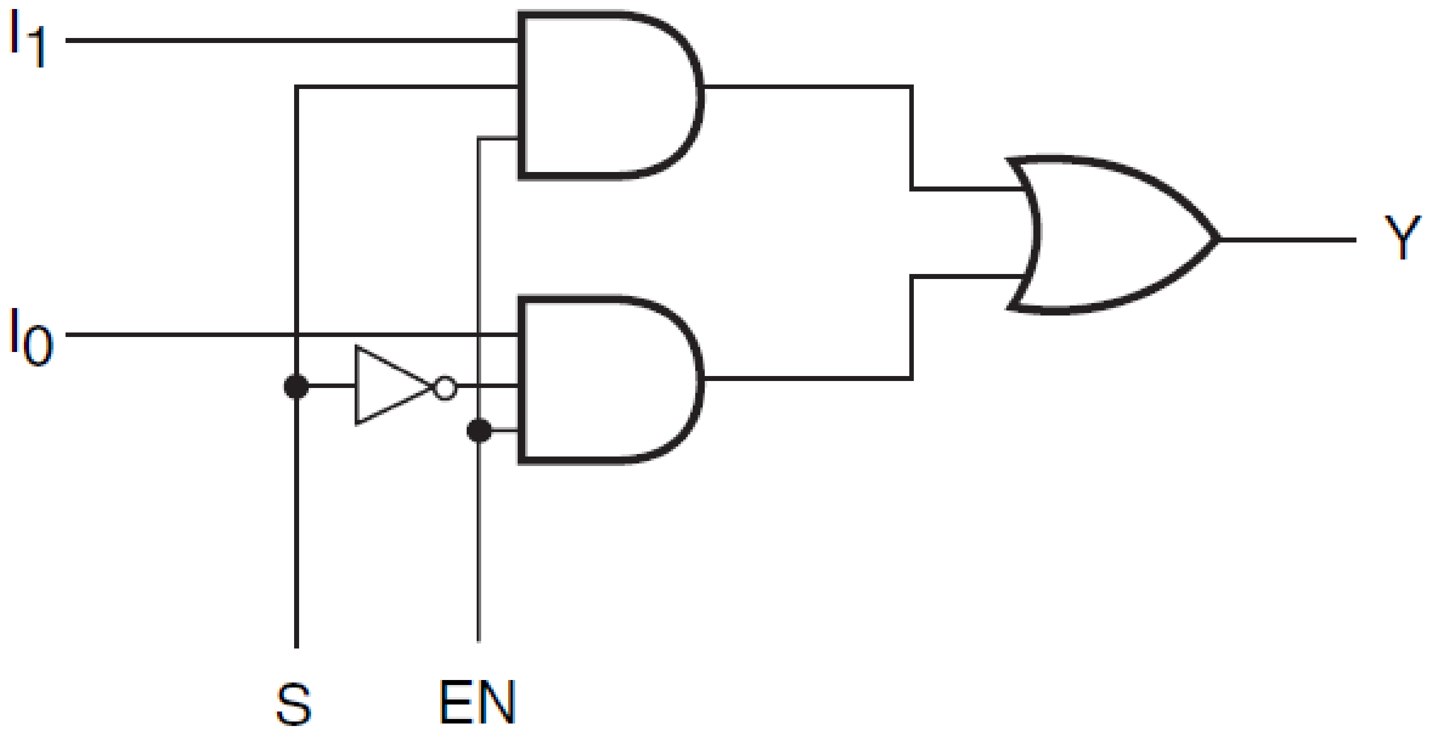
- Output:

$$\text{EN} = 1 \quad S = 0 \quad Y = I_0 \qquad Y = EN(\overline{S}I_0 + SI_1)$$

$$S = 1 \quad Y = I_1$$

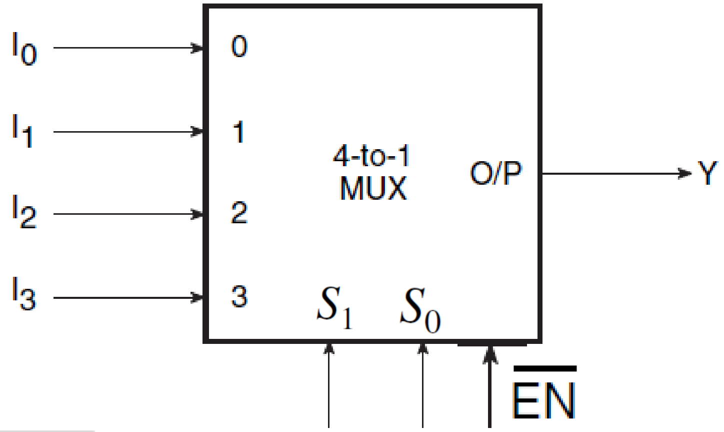
2-to-1 MUX with an ENABLE Input

- Logic diagram:



4-to-1 MUX with an ENABLE Input

- Block diagram:



- Truth table:

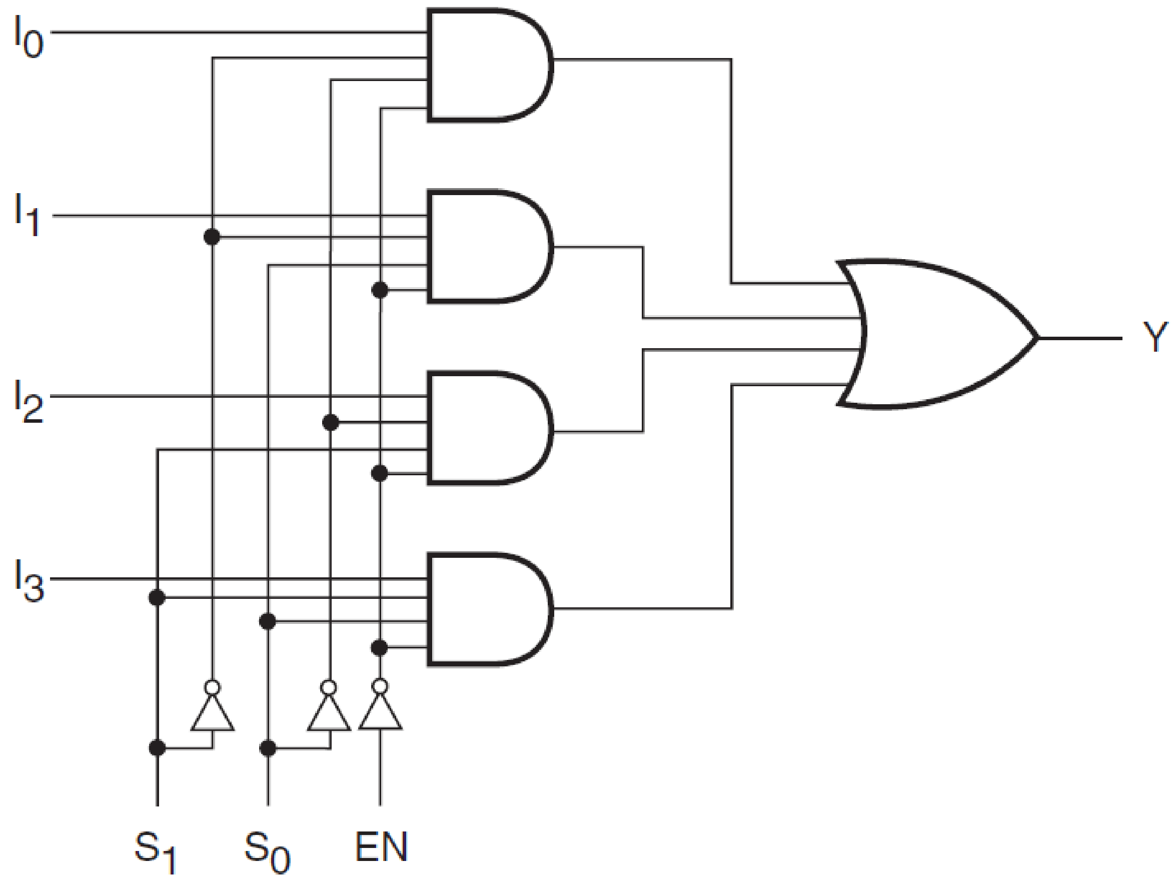
S_1	S_0	EN	Y
X	X	1	0
0	0	0	I_0
0	1	0	I_1
1	0	0	I_2
1	1	0	I_3

- Output:

$$Y = \left(I_0.\overline{S_1}.\overline{S_0} + I_1.\overline{S_1}.S_0 + I_2.S_1.\overline{S_0} + I_3.S_1.S_0\right)\overline{EN}$$

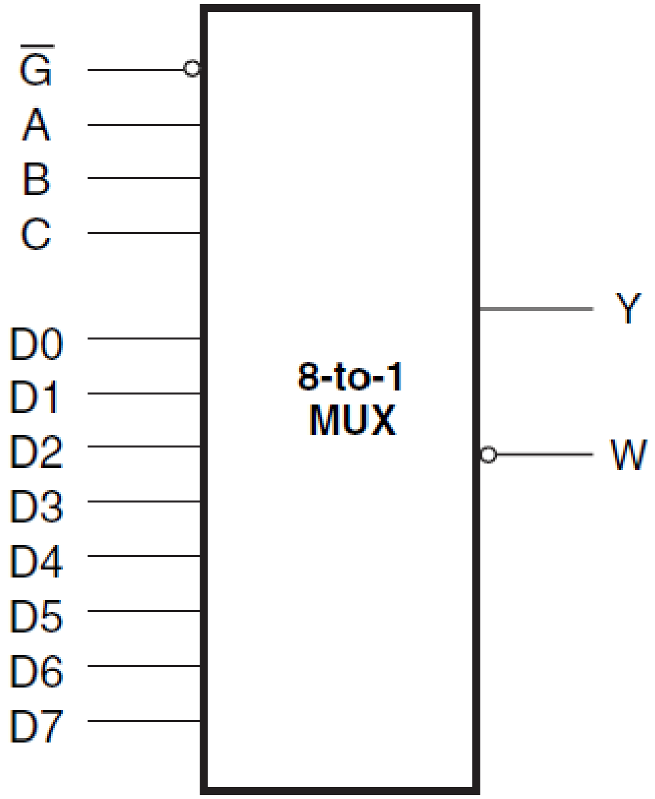
4-to-1 MUX with an ENABLE Input

- Logic diagram:



8-to-1 MUX with an ENABLE Input

- Block diagram:



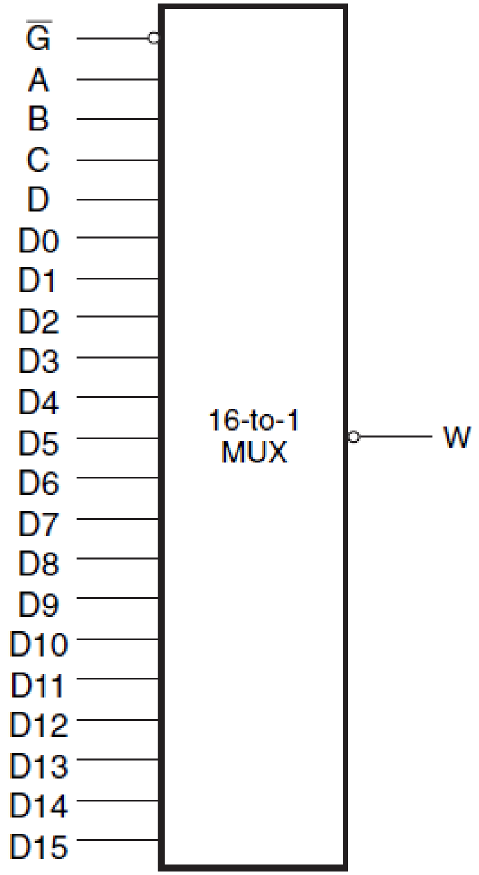
- Truth table:

Inputs				Output	
Select			Enable		
C	B	A	$\overline{G}$	Y	W
X	X	X	H	L	H
L	L	L	L	D0	$\overline{D0}$
L	L	H	L	D1	$\overline{D1}$
L	H	L	L	D2	$\overline{D2}$
L	H	H	L	D3	$\overline{D3}$
H	L	L	L	D4	$\overline{D4}$
H	L	H	L	D5	$\overline{D5}$
H	H	L	L	D6	$\overline{D6}$
H	H	H	L	D7	$\overline{D7}$

$\overline{G}$: ENABLE input
A, B, C : Select inputs
D0-D7 : Data inputs
Y,W : outputs

16-to-1 MUX with an ENABLE Input

- Block diagram:



- Truth table:

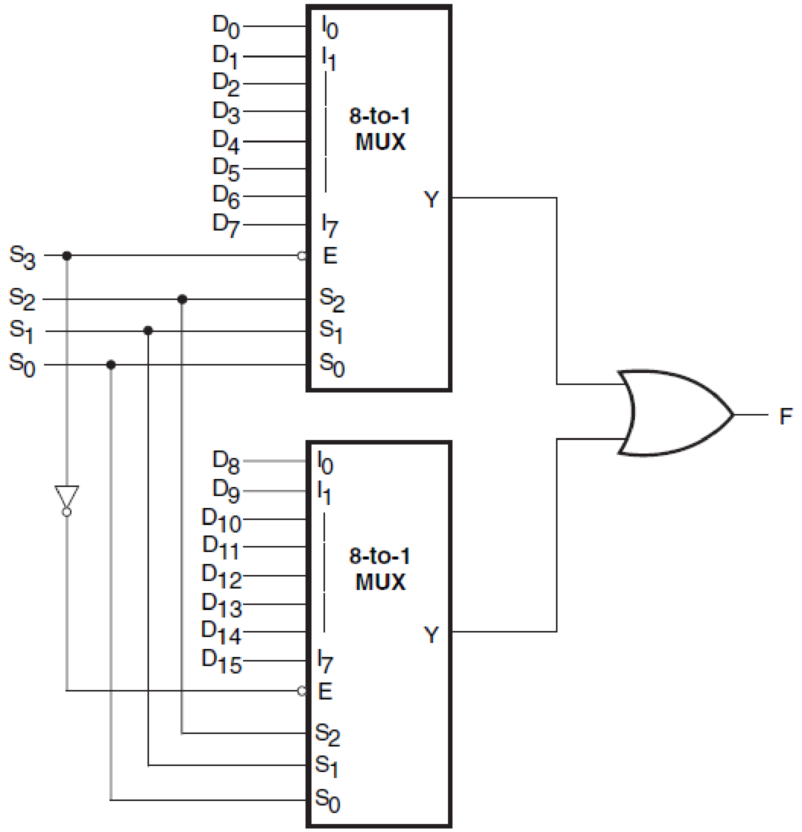
Inputs					Output W
Select				Enable	
D	C	B	A	$\overline{G}$	
X	X	X	X	H	H
L	L	L	L	L	$\overline{D0}$
L	L	L	H	L	$\overline{D1}$
L	L	H	L	L	$\overline{D2}$
L	L	H	H	L	$\overline{D3}$
L	H	L	L	L	$\overline{D4}$
L	H	L	H	L	$\overline{D5}$
L	H	H	L	L	$\overline{D6}$
L	H	H	H	L	$\overline{D7}$
H	L	L	L	L	$\overline{D8}$
H	L	L	H	L	$\overline{D9}$
H	L	H	L	L	$\overline{D10}$
H	L	H	H	L	$\overline{D11}$
H	H	L	L	L	$\overline{D12}$
H	H	L	H	L	$\overline{D13}$
H	H	H	L	L	$\overline{D14}$
H	H	H	H	L	$\overline{D15}$

Cascading Multiplexer Circuits

- **Possible problem**: the desired number of input channels is not available in IC multiplexers.
- **Solution**:
    - If the available MUX has 2^n input lines and the desired MUX has 2^N input lines, then it requires 2^{N-n} individual MUXs to construct the desired MUX.
    - Connect the LSBs of the selection inputs of the desired MUX to the selection inputs of the available MUX.
    - The left-over bits of the selection inputs of the desired MUX are used to enable or disable the individual MUXs so that their outputs when ORed produce the final output.

Example 4

- Design a 16-to-1 MUX using two 8-to-1 MUXs having an active LOW ENABLE input.



Implementing Boolean Functions with MUXs

- One of the most common applications of a MUX is its use for implementation of combinational logic Boolean functions.
- Employing a 2^n-to-1 MUX to implement an n-variable Boolean function:
    - The **input lines corresponding to** each of the **minterms** present in the Boolean function are made equal to **logic ‘1’** state.
    - The **remaining minterms** that are **absent** in the Boolean function are disabled by making their corresponding input lines equal to **logic ‘0’**.

Example 5

- Use an 8-to-1 MUX to emplement the Boolean function given by:

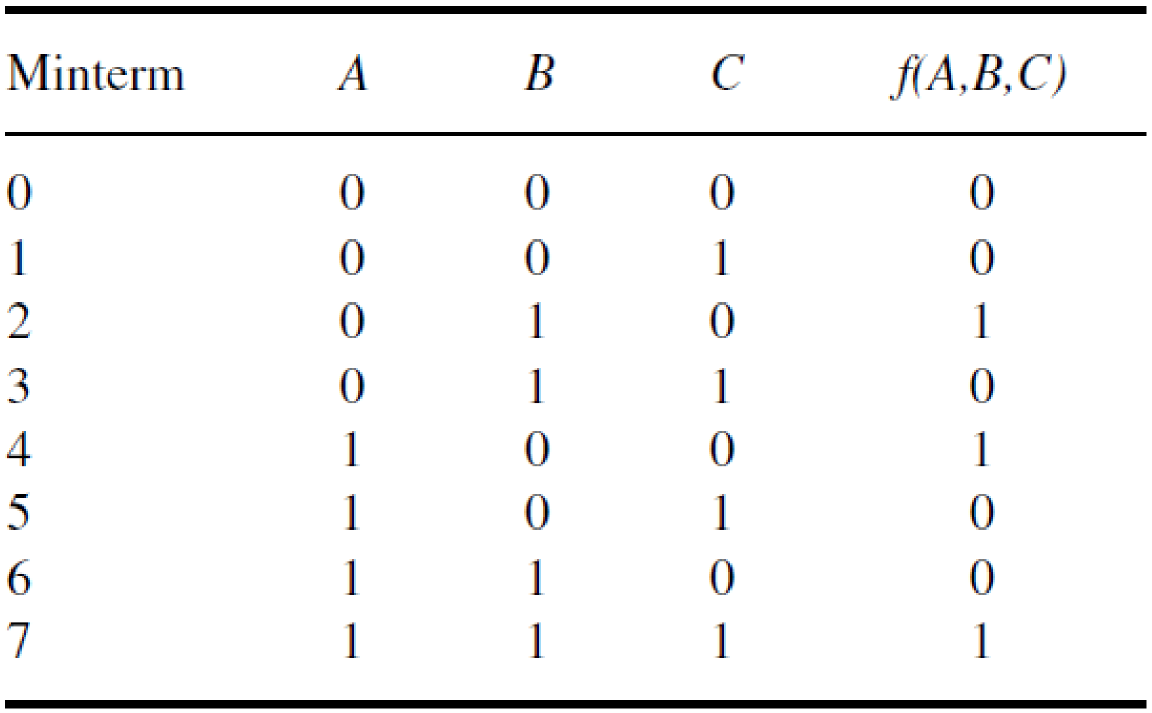
$$f(A, B, C) = \sum 2, 4, 7$$

- The equation can be written as follows:

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

Example 5

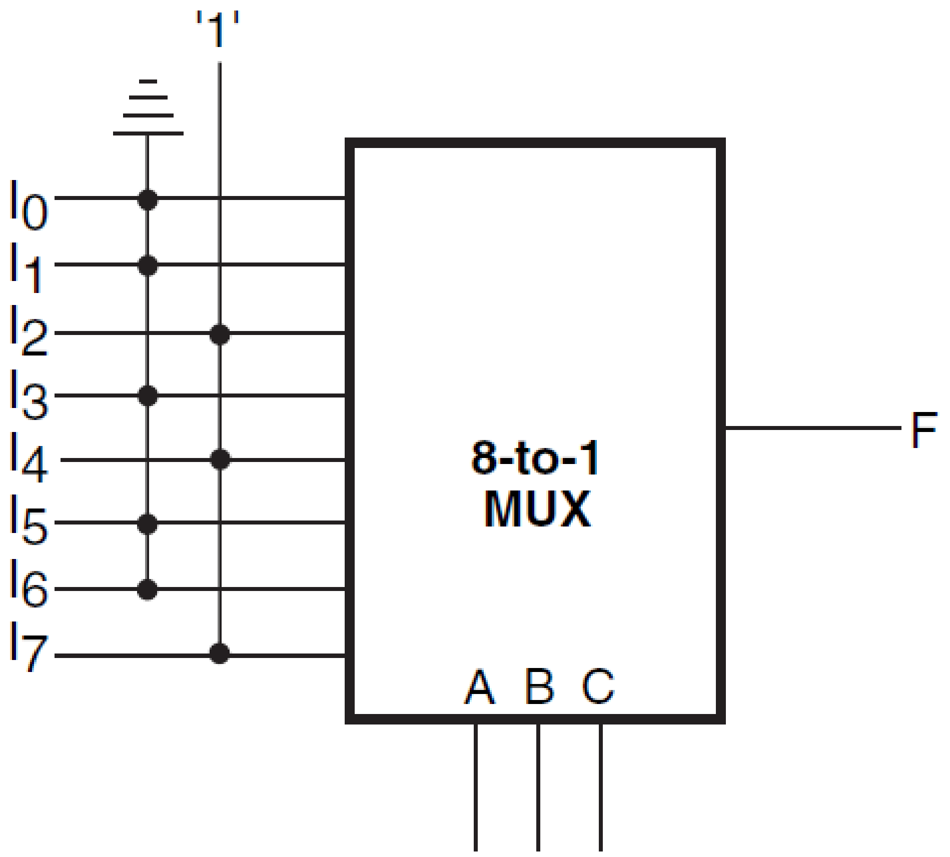
- Truth table:



Minterm	A	B	C	$f(A,B,C)$
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	0
6	1	1	0	0
7	1	1	1	1

Example 5

- Hardware implementation:



Implementing Boolean Functions with MUXs

- Employing a 2^n-to-1 MUX to implement an (n+1)-variable Boolean function.
  - Out of n +1 variables, n are connected to the n selection lines of the 2^n-to-1 multiplexer.
  - The left-over variable is used with the input lines.
  - Various input lines are tied to one of the following: ‘0’, ‘1’, the left-over variable and the complement of the left-over variable.

Example 6

- Use an 4-to-1 MUX to emplement the Boolean function given by:

$$f(A, B, C) = \sum 2, 4, 7$$

- The equation can be written as follows:

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

Example 6

- Truth table:

Minterm	A	B	C	$f(A,B,C)$
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	0
6	1	1	0	0
7	1	1	1	1

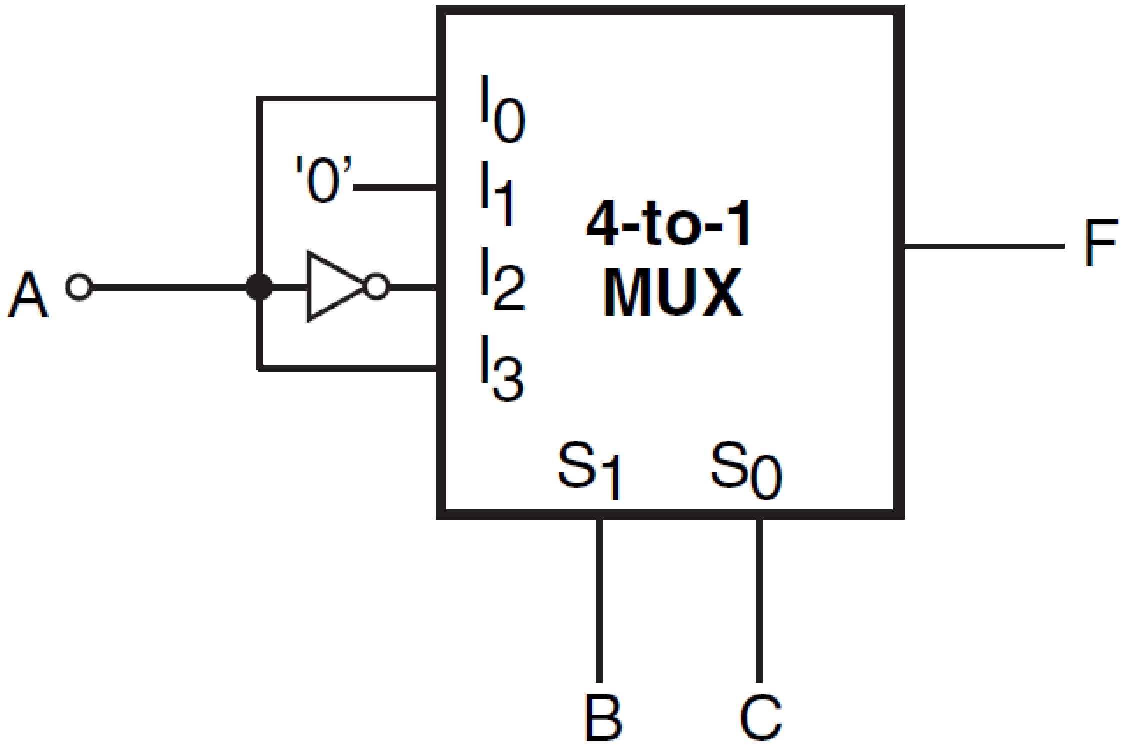
Example 6

- A is connected to the input lines, B and C are connected to the 2 selection lines.
- Construct the table:
  - If neither minterm of a certain column is highlighted, a ‘0’ is written below that.
  - If both are highlighted, a ‘1’ is written.
  - If only one is highlighted, the corresponding variable (complemented or uncomplemented) is written.

	I_0	I_1	I_2	I_3
$\overline{A}$	0	1	**2**	3
A	**4**	5	6	**7**
	A	0	$\overline{A}$	A

Example 6

- Hardware implementation:

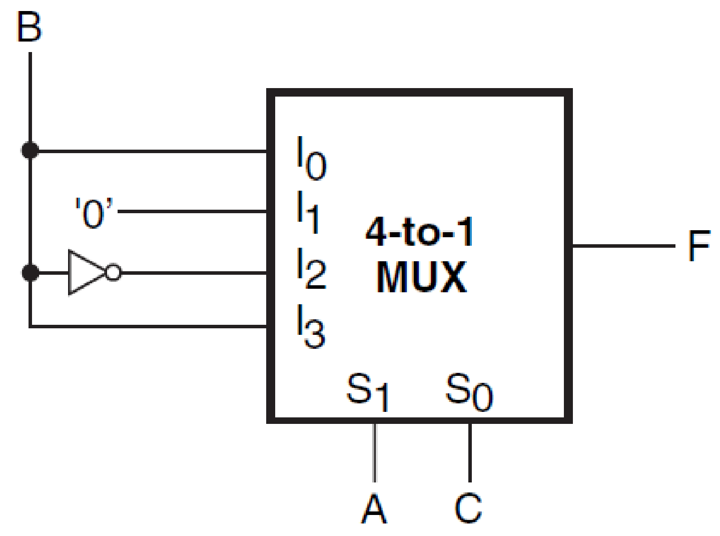


Example 6

- For the case of B being the left-out variable:

	I_0	I_1	I_2	I_3
$\overline{B}$	0	1	**4**	5
B	**2**	3	6	**7**
	B	0	$\overline{B}$	B

- Hardware implementation:

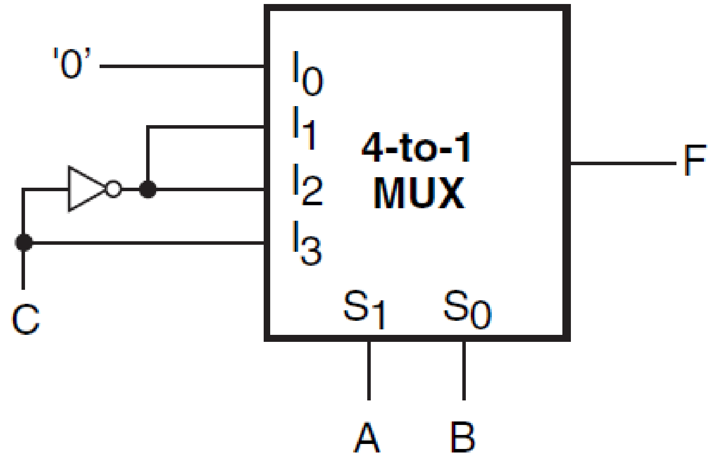


Example 6

- For the case of C being the left-out variable:

	I_0	I_1	I_2	I_3
$\overline{C}$	0	**2**	**4**	6
C	1	3	5	**7**
	0	$\overline{C}$	$\overline{C}$	C

- Hardware implementation :



Combinational Circuits

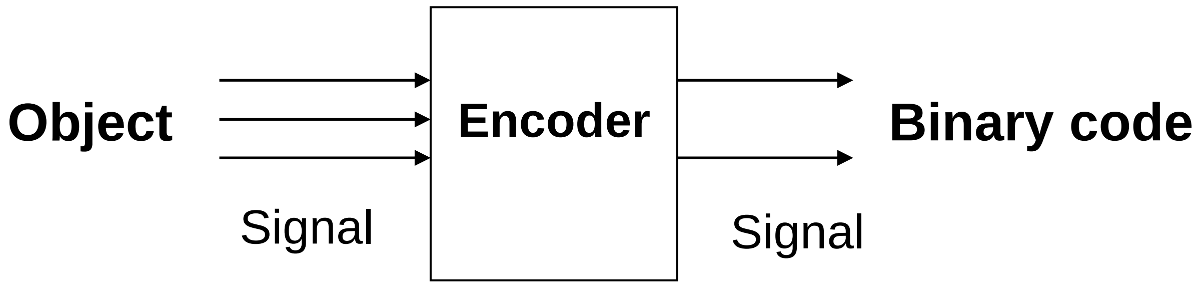
- Arithmetic Circuits
- Multiplexers
- Encoders
- Demultiplexers
- Decoders

Encoders

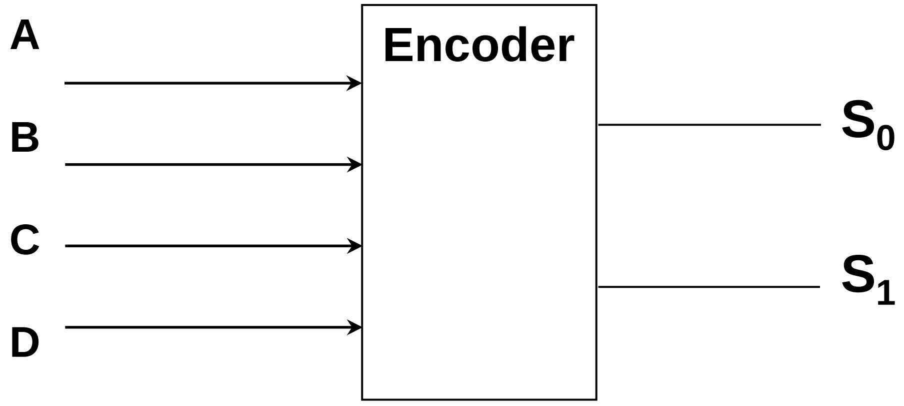
- An encoder is a multiplexer without its single output line.
- It has 2^n (or fewer) input lines and n output lines, which correspond to n selection lines in a multiplexer.
- Example:

Object	Decimal Code	Binary Code
A	0	00
B	1	01
C	2	10
D	3	11

Encoders



- Example:



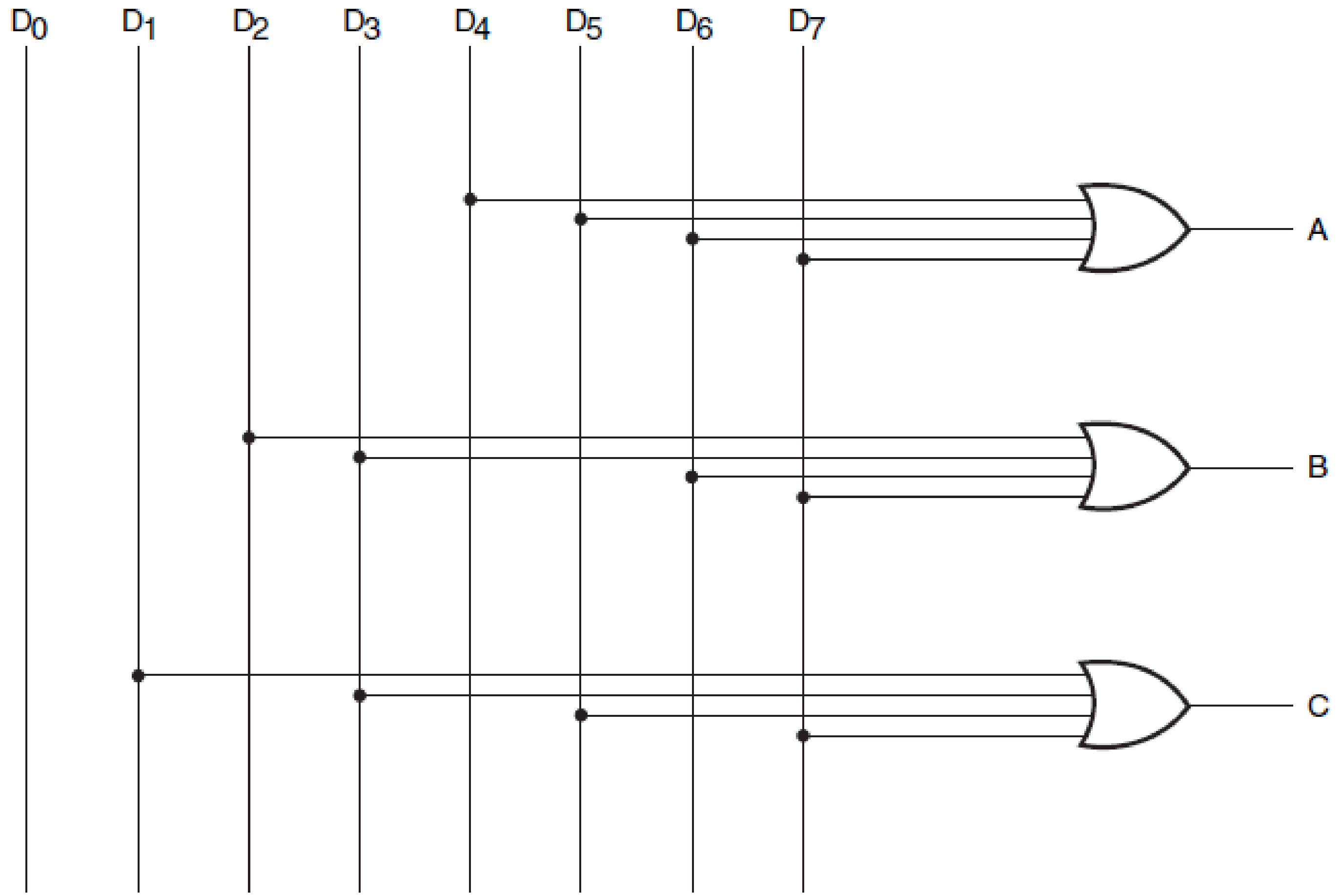
Example 8

- 8 to 3 encoder
- Truth table:

D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	A	B	C
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Example 8

- Logic diagram:

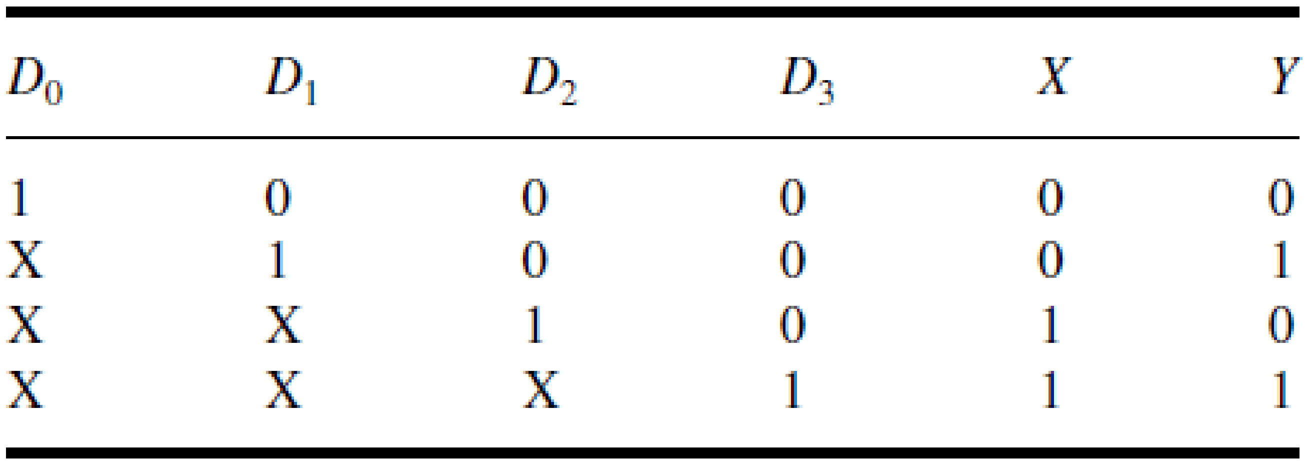


Priority Encoder

- A priority encoder is a practical form of an encoder.
- A priority is assigned to each input so that, when more than one input is simultaneously active, the input with the highest priority is encoded.

Example 9

- 4 to 2 priority encoder
  - All inputs and outputs are active when HIGH
  - Higher priority encoding for higher-order digits.
- Truth table:



D_0	D_1	D_2	D_3	X	Y
1	0	0	0	0	0
X	1	0	0	0	1
X	X	1	0	1	0
X	X	X	1	1	1

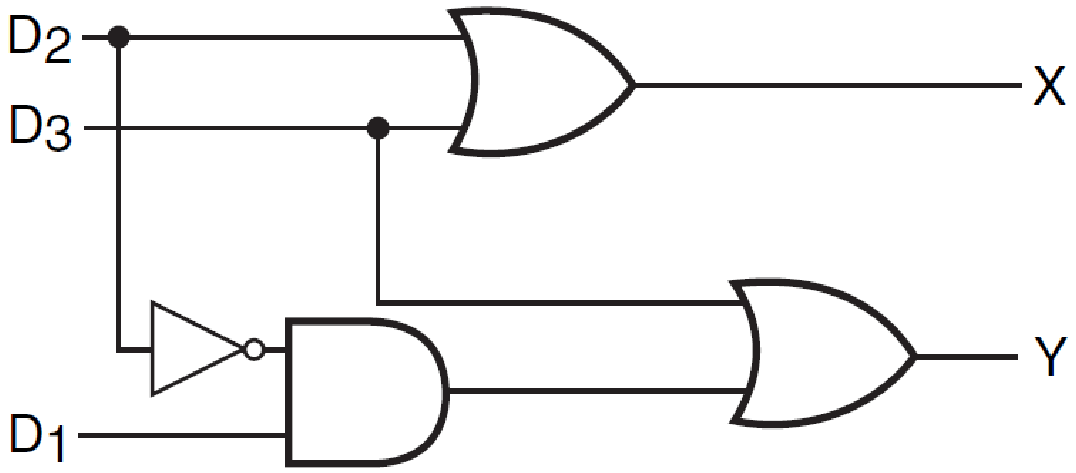
Example 9

- The Boolean expressions:

$$X = D_2.\overline{D_3} + D_3 = D_2 + D_3$$

$$Y = D_1.\overline{D_2}.\overline{D_3} + D_3 = D_1.\overline{D_2} + D_3$$

- Logic diagram:



Combinational Circuits

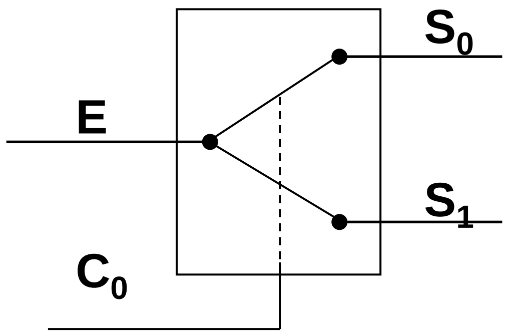
- Arithmetic Circuits
- Multiplexers
- Encoders
- Demultiplexers
- Decoders

Demultiplexers

- DeMultiPlexor – DeMUX
- A DeMux has an input line, 2^n output lines and n select lines.
- It routes the information present on the input line to any of the output lines, depending on the bit status of the selection lines.

1-to-2 DeMUX

- Block diagram:

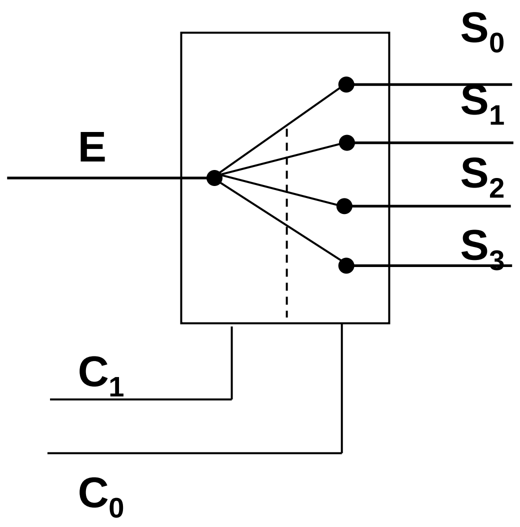


- Truth table:

C_0	S_0	S_1
0	E	0
1	0	E

1-to-4 DeMUX

- Block diagram:



- Truth table:

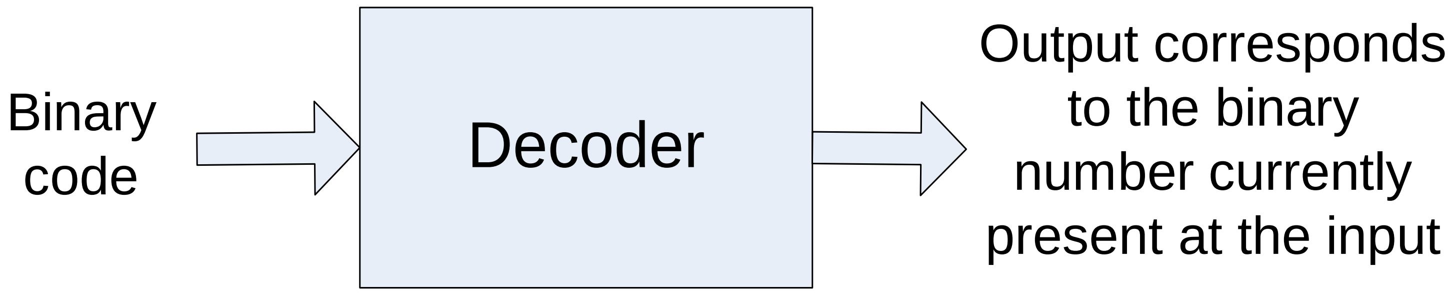
C_1	C_0	S_0	S_1	S_2	S_3
0	0	E	0	0	0
0	1	0	E	0	0
1	0	0	0	E	0
1	1	0	0	0	E

Combinational Circuits

- Arithmetic Circuits
- Multiplexers
- Encoders
- Demultiplexers
- Decoders

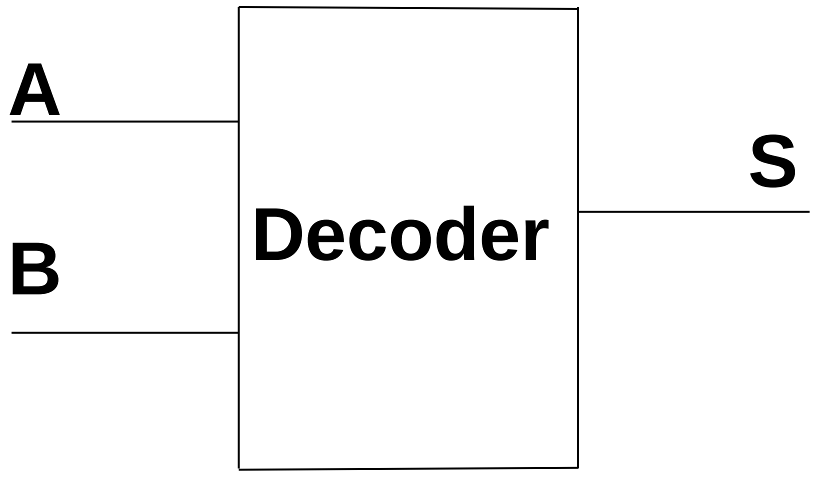
Decoders

- A decoder is a combinational circuit that decodes the information on n input lines to a maximum of 2^n unique output lines.


Binary code
Decoder
Output corresponds to the binary number currently present at the input

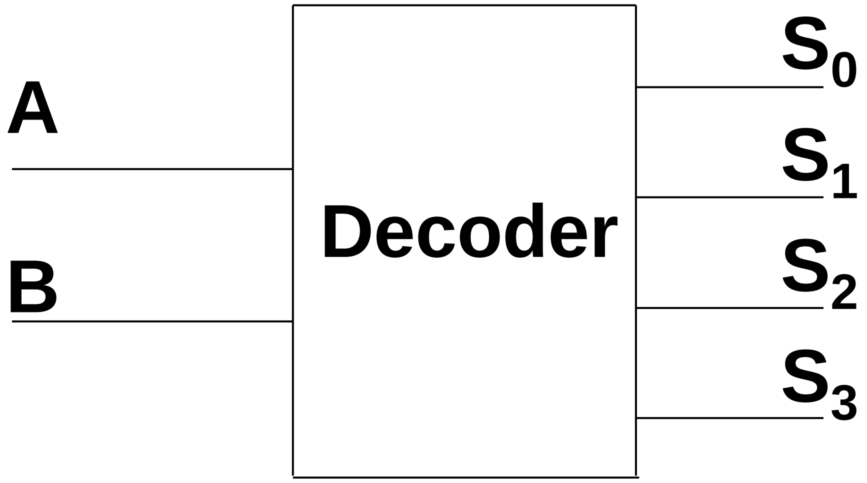
Decoders

- Partial decoder:
    - A decoder with n inputs may have fewer than 2^n outputs.
    - Example: S = 1 if (AB) = (10), S = 0 if (AB) $\neq$ (10)



Decoders

- Full decoder:
  - A decoder with n inputs may have enough 2^n outputs.



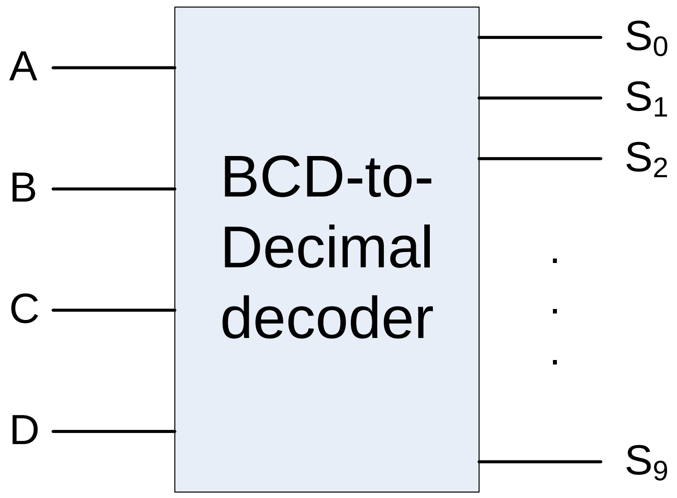
Example 10

- Design a BCD-to-Decimal decoder

Decimal numeral	BCD code
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Example 10

- Determine the input and output:
  - Input: BCD code (4 bits ⇒ 16 input combinations)
  - Output: output line, which corresponds to the BCD code currently present at the input
- 10 input combinations are used, 6 input combinations are unused (i.e. don't care combinations).



Example 10

- Truth table:

ABCD	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9
0000	1	0	0	0	0	0	0	0	0	0
0001	0	1	0	0	0	0	0	0	0	0
0010	0	0	1	0	0	0	0	0	0	0
0011	0	0	0	1	0	0	0	0	0	0
0100	0	0	0	0	1	0	0	0	0	0
0101	0	0	0	0	0	1	0	0	0	0
0110	0	0	0	0	0	0	1	0	0	0
0111	0	0	0	0	0	0	0	1	0	0
1000	0	0	0	0	0	0	0	0	1	0
1001	0	0	0	0	0	0	0	0	0	1
1010	-	-	-	-	-	-	-	-	-	-
1011	-	-	-	-	-	-	-	-	-	-
1100	-	-	-	-	-	-	-	-	-	-
1101	-	-	-	-	-	-	-	-	-	-
1110	-	-	-	-	-	-	-	-	-	-
1111	-	-	-	-	-	-	-	-	-	-

Example 10

- The simplified Boolean expressions for S_0 and S_1

$$S_0(A,B,C,D) = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} \qquad S_1(A,B,C,D) = \overline{A}\,\overline{B}\,\overline{C}\,D$$

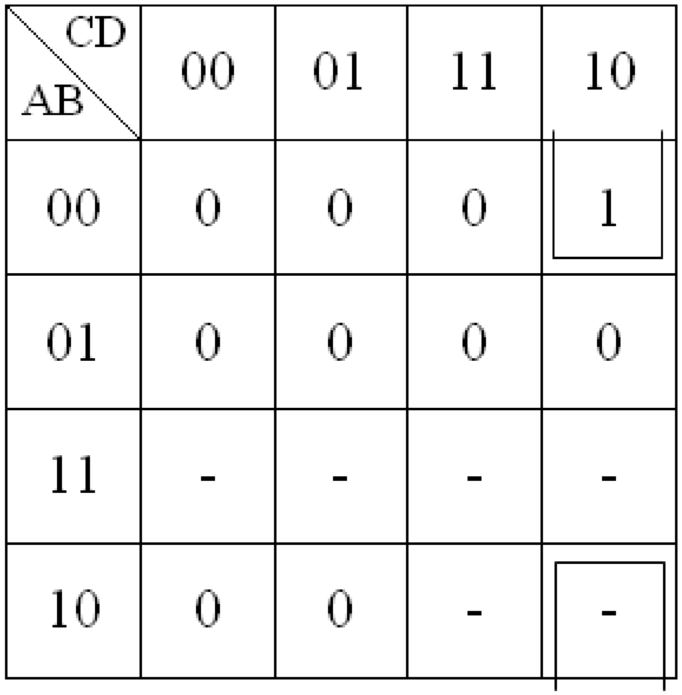
AB \ CD	00	01	11	10
00	$\boxed{1}$	0	0	0
01	0	0	0	0
11	-	-	-	-
10	0	0	-	-

AB \ CD	00	01	11	10
00	0	$\boxed{1}$	0	0
01	0	0	0	0
11	-	-	-	-
10	0	0	-	-

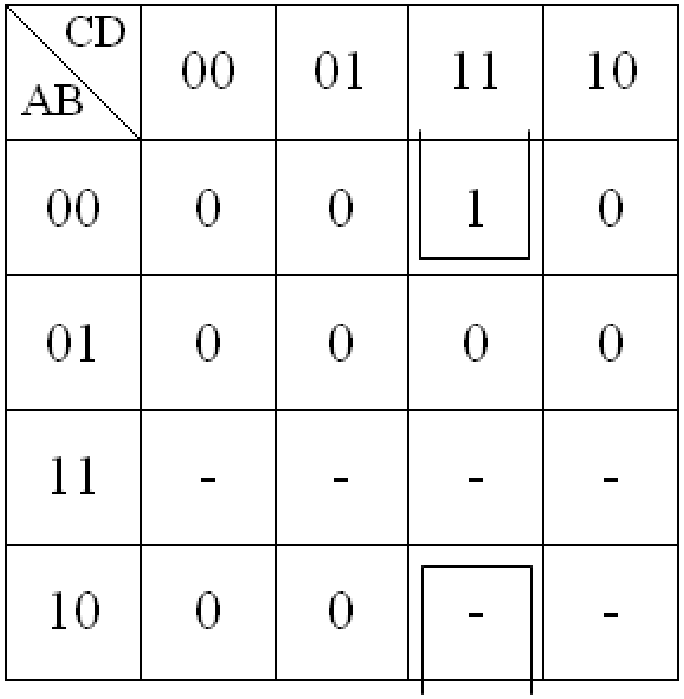
Example 10

- The simplified Boolean expressions for S_2 and S_3

$S_2(A,B,C,D) = \overline{B}\,C\,\overline{D}$ $S_3(A,B,C,D) = \overline{B}\,CD$



CD / AB	00	01	11	10
00	0	0	0	1
01	0	0	0	0
11	-	-	-	-
10	0	0	-	-



CD / AB	00	01	11	10
00	0	0	1	0
01	0	0	0	0
11	-	-	-	-
10	0	0	-	-

Example 10

- The simplified Boolean expressions for S_4 and S_5

$$S_4(A,B,C,D) = B\,\overline{C}\,\overline{D}$$

$$S_5(A,B,C,D) = B\,\overline{C}\,D$$

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

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

Example 10

- The simplified Boolean expressions for S_6 and S_7

$$S_6(A,B,C,D) = BC\overline{D}$$

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

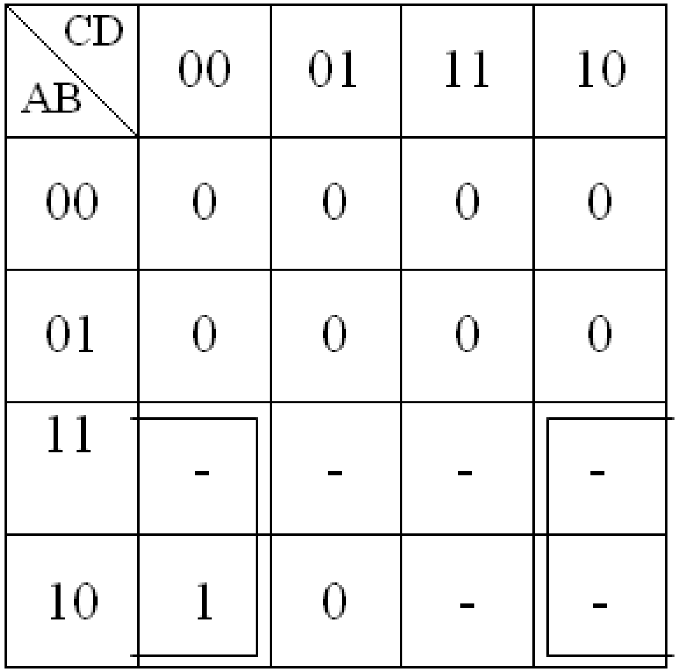
$$S_7(A,B,C,D) = BCD$$

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

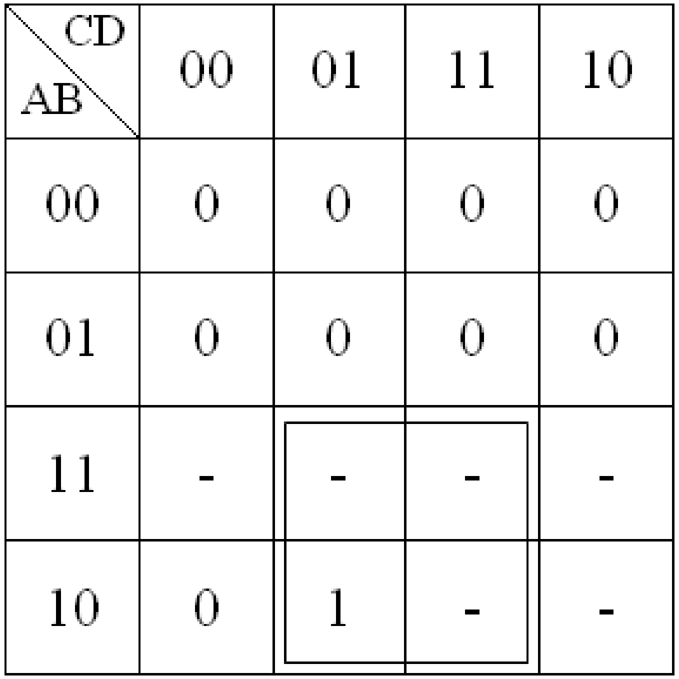
Example 10

- The simplified Boolean expressions for S_8 and S_9

$S_8(A,B,C,D) = A\,\overline{D}$ $\quad\quad$ $S_9(A,B,C,D) = AD$



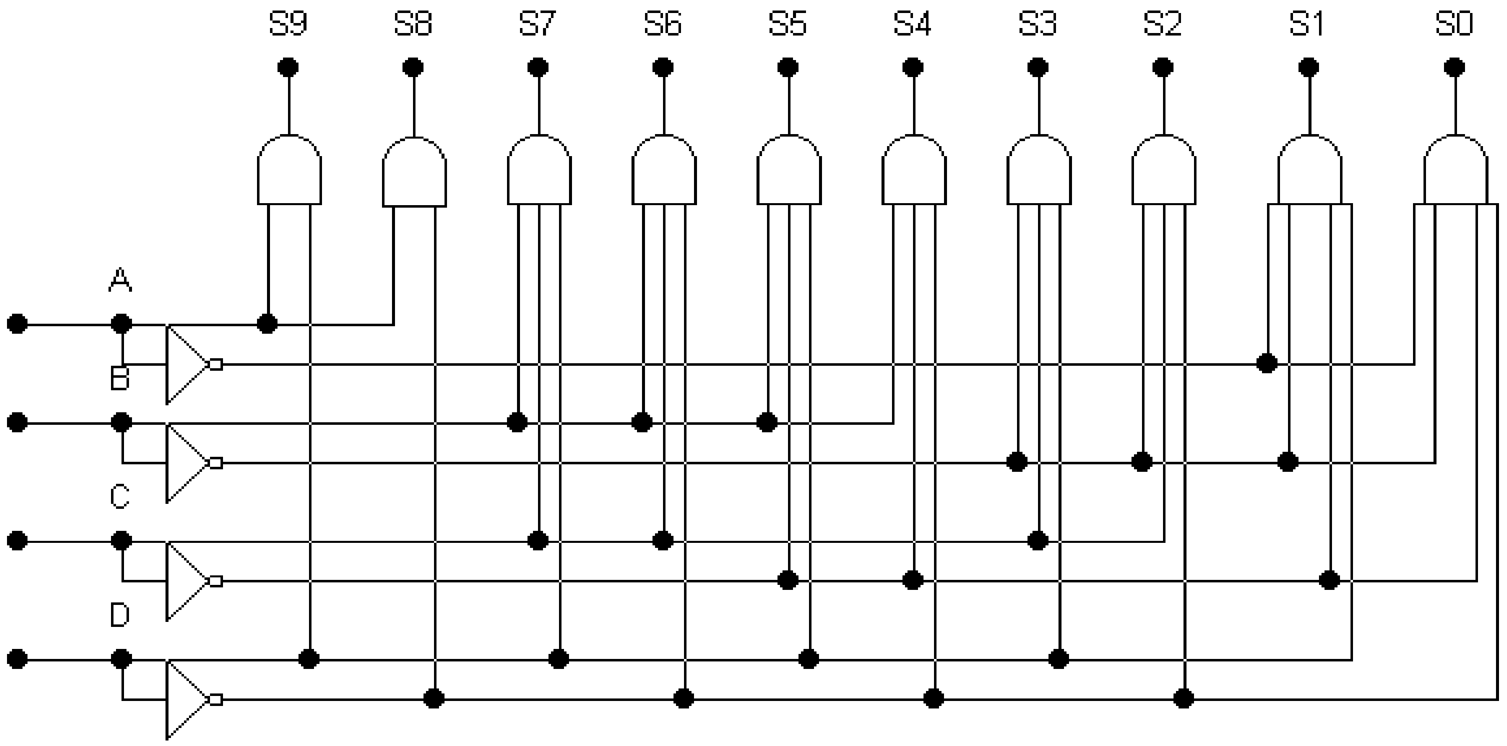
AB \ CD	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	-	-	-	-
10	1	0	-	-



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

Example 10

- Logic diagram:



Implementing Boolean Functions with Decoders

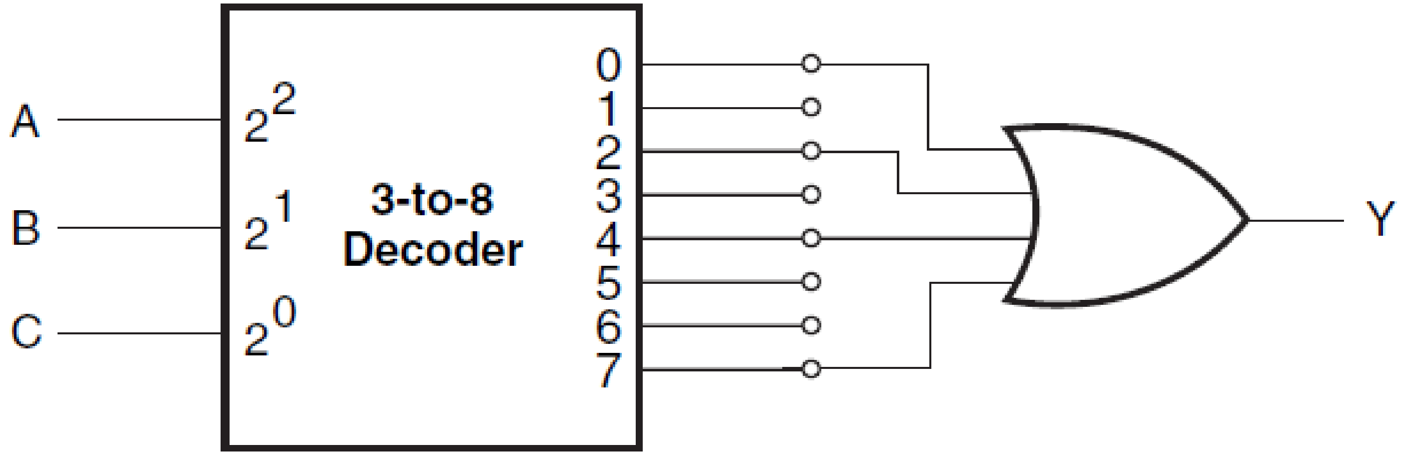
- A decoder can implement a given Boolean function by generating the required minterms and the an external OR gate is used to produce the sum of minterms.

Example 11

- Implement the following Boolean function using a 3-to-8 decoder.:

$$Y = A.\overline{B}.\overline{C} + \overline{A}.B.\overline{C} + A.B.C + \overline{A}.\overline{B}.C$$

- Logic diagram:

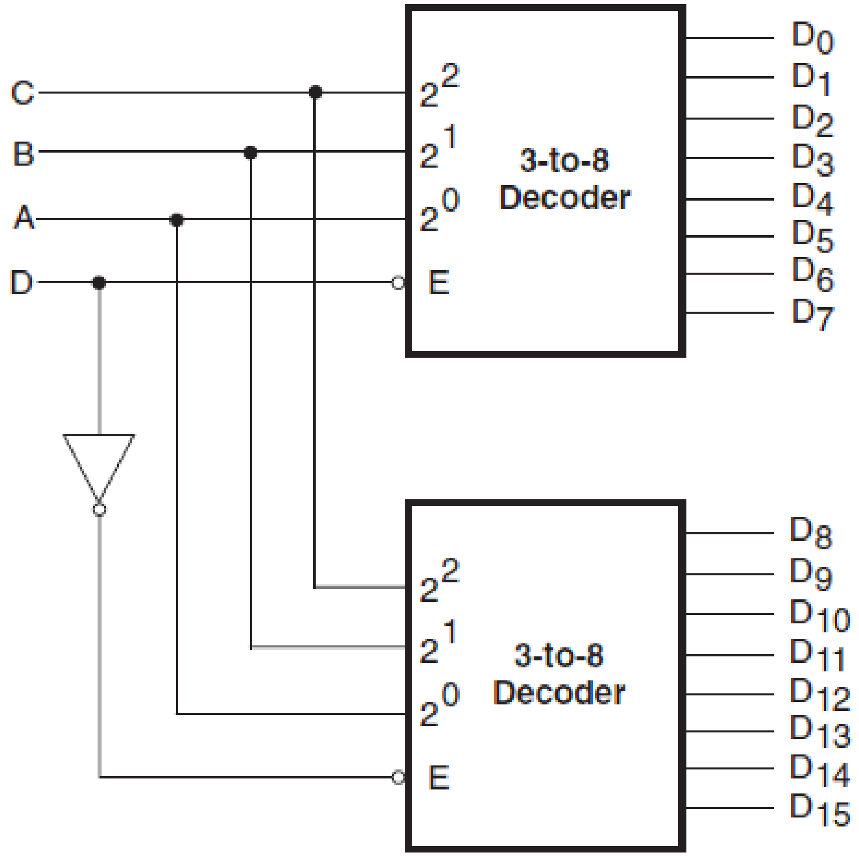


Cascading Decoder Circuits

- If the available decoder has n input lines and the desired decode has N input line → it requires 2^{N-n} individual decoders to construct the desired decoder.
- Connect the LSBs of the input lines of the desired decoder to the input lines of the available decoder.
- The left-over bits of the input lines of the desired decoder circuit are used to enable or disable the individual decoders.
- The output lines of the individual decoders together constitute the output lines.

Example 12

- Implement a 4-to-16 decoder using 2 3-to-8 decoders. All inputs are active when LOW.



Example 13

- Implement a full adder using a 3-to-8 decoder.
- Truth table:

A	B	C	S	C_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

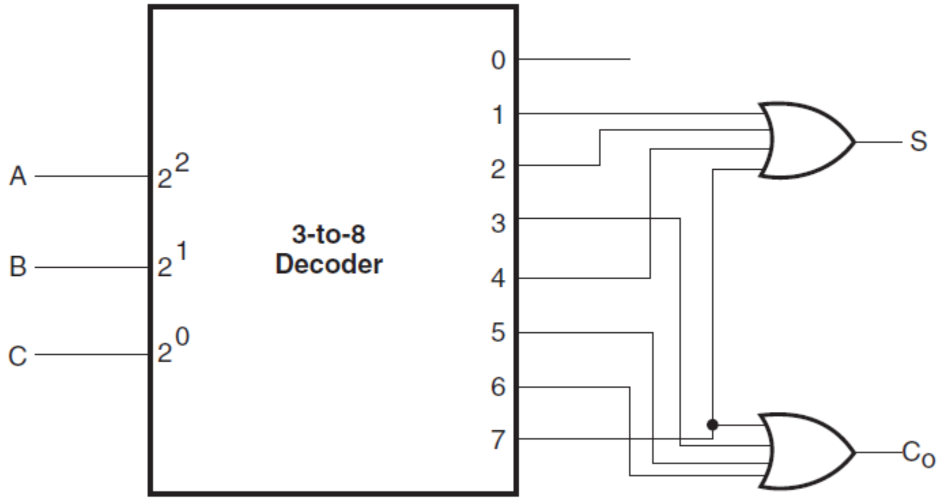
Example 13

- The Boolean expressions for:

$$\text{Sum output } S = \Sigma\ 1, 2, 4, 7$$

$$\text{Carry output } C_o = \Sigma\ 3, 5, 6, 7$$

- Logic diagram:


A
B
C
2^2
2^1
2^0
3-to-8 Decoder
0
1
2
3
4
5
6
7
S
C_o