Implementation of Combinational Logic Functions - Part 1

Encoders & Decoders

Half Adder, & Full Adder

MUX and deMUX

DIGITAL CIRCUITS

The classification of digital circuitsis shown in Fig. 1.8.

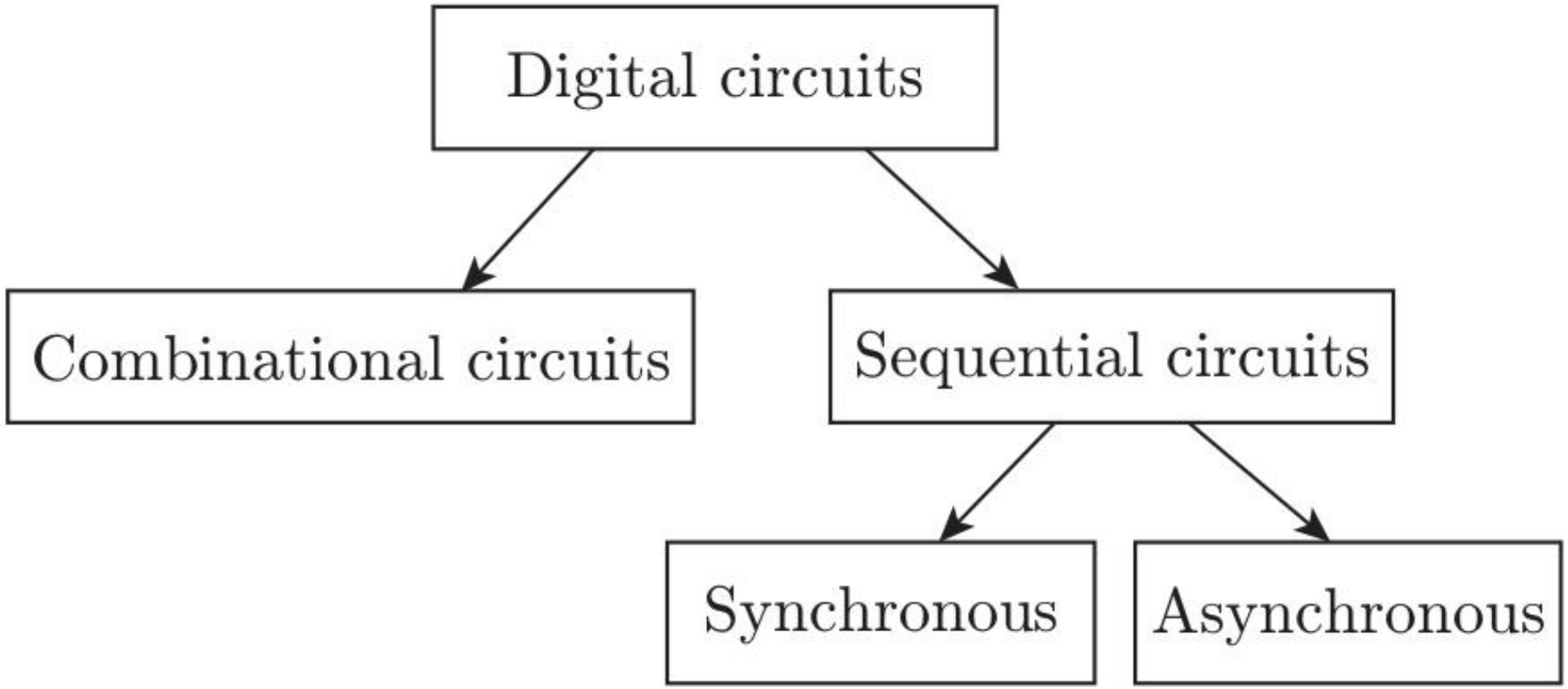


Figure 1.8 | Types of digital circuits.

Combinational Circuits

A logic circuit whose output at any instant of time depends only on the present input is called combinational circuit. It contains no memory element.

HALF ADDER

Recall the basic rules for binary addition as stated in Chapter 2.

$$
\begin{aligned}
0 + 0 &= 0 \\
0 + 1 &= 1 \\
1 + 0 &= 1 \\
1 + 1 &= 10
\end{aligned}
$$

The operations are performed by a logic circuit called a **half-adder**.

HALF ADDER

The half-adder accepts two binary digits on its inputs and produces two binary digits on its outputs—a sum bit and a carry bit.

A half-adder is represented by the logic symbol in Figure 6–1.

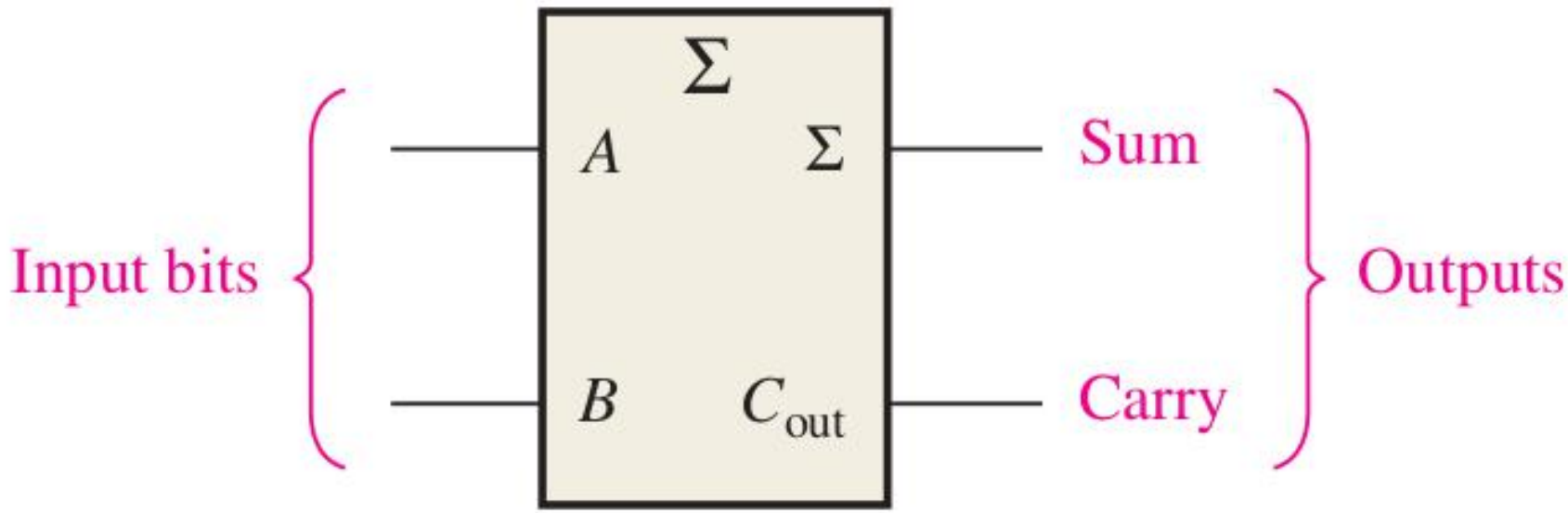


FIGURE 6–1 Logic symbol for a half-adder.

Half-Adder Logic

Half-Adder Logic

From the operation of the half-adder as stated in Table 6–1, expressions can be derived for the sum and the output carry as functions of the inputs. Notice that the output carry (C_{out}) is a 1 only when both A and B are 1s; therefore, C_{out} can be expressed as the AND of the input variables.

$$C_{out} = AB \qquad \text{Equation 6–1}$$

TABLE 6–1

Half-adder truth table.

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

Σ = sum

C_{out} = output carry

A and B = input variables (operands)

Half-Adder Logic

TABLE 6–1

Half-adder truth table.

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

Σ = sum
C_{out} = output carry
A and B = input variables (operands)

From the operation of the half-adder as stated in Table 6–1, expressions can be derived for the sum and the output carry as functions of the inputs. Notice that the output carry (C_{out}) is a 1 only when both A and B are 1s; therefore, C_{out} can be expressed as the AND of the input variables.

$$C_{out} = AB \qquad \text{Equation 6–1}$$

Now observe that the sum output (Σ) is a 1 only if the input variables, A and B, are not equal. The sum can therefore be expressed as the exclusive-OR of the input variables.

$$\Sigma = A \oplus B \qquad \text{Equation 6–2}$$

Half-Adder Logic

$$C_{out} = AB \qquad \text{Equation 6–1}$$

$$\Sigma = A \oplus B \qquad \text{Equation 6–2}$$

TABLE 6–1

Half-adder truth table.

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

Σ = sum
C_{out} = output carry
A and B = input variables (operands)

From Equations 6–1 and 6–2, the logic implementation required for the half-adder function can be developed. The output carry is produced with an AND gate with A and B on the inputs, and the sum output is generated with an exclusive-OR gate, as shown in Figure 6–2. Remember that the exclusive-OR can be implemented with AND gates, an OR gate, and inverters.

Half-Adder Logic

TABLE 6–1

Half-adder truth table.

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

Σ = sum
C_{out} = output carry
A and B = input variables (operands)

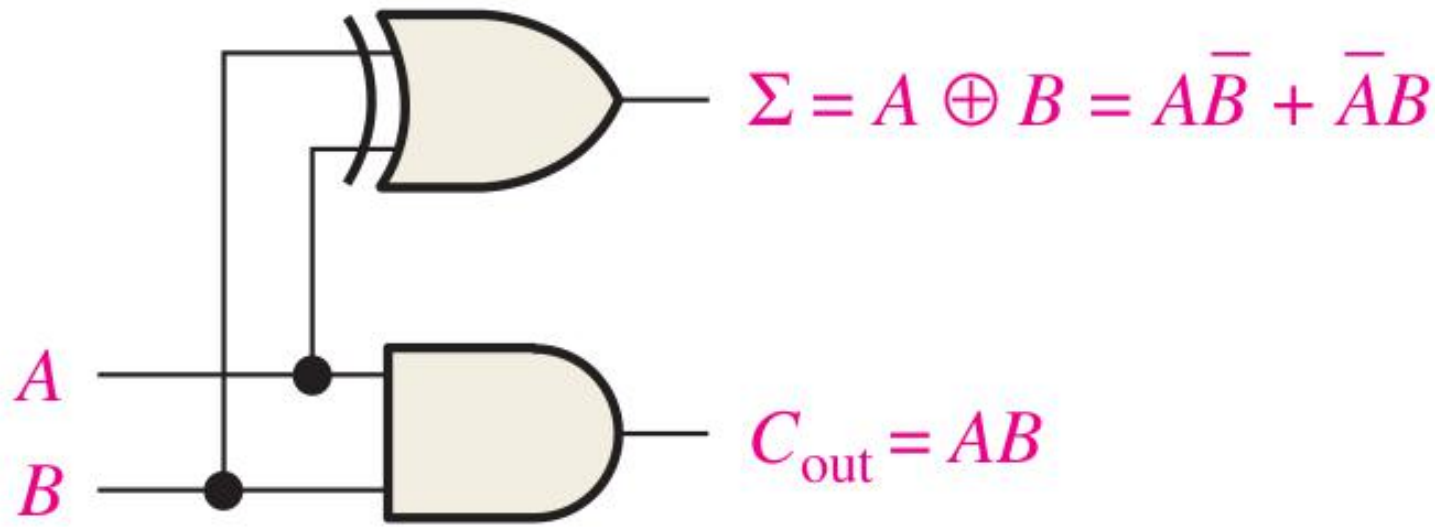
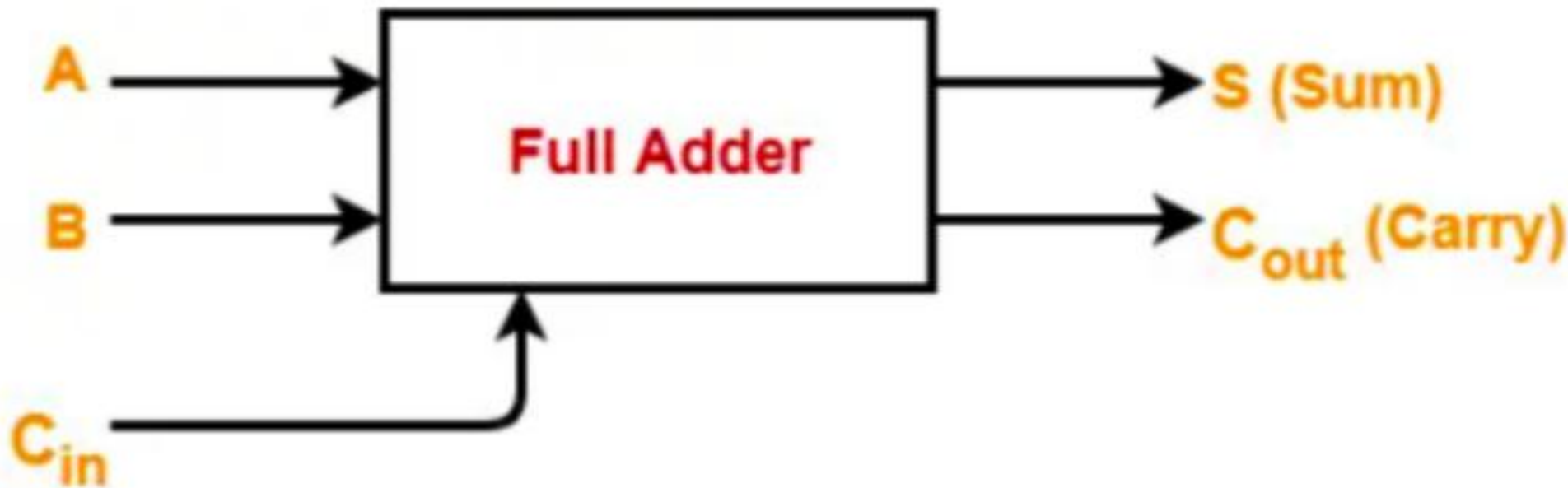


FIGURE 6–2 Half-adder logic diagram.

The Full-Adder

The Full-Adder

The full-adder accepts two input bits and an input carry and generates a sum output and an output carry.



Block diagram of a full adder

The Full-Adder

FULL ADDER

The full-adder accepts two input bits and an input carry and generates a sum output and an output carry.

The basic difference between a full-adder and a half-adder is that the full-adder accepts an input carry.

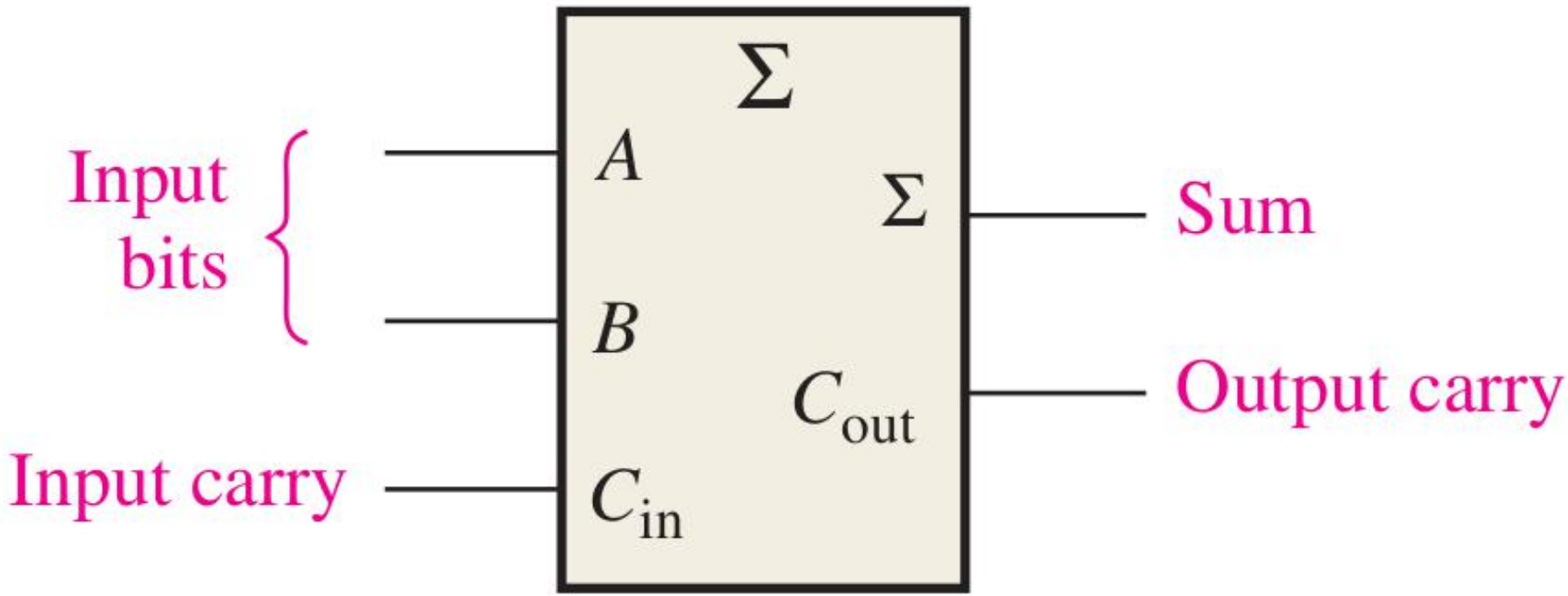


FIGURE 6–3 Logic symbol for a full-adder.

The Full-Adder

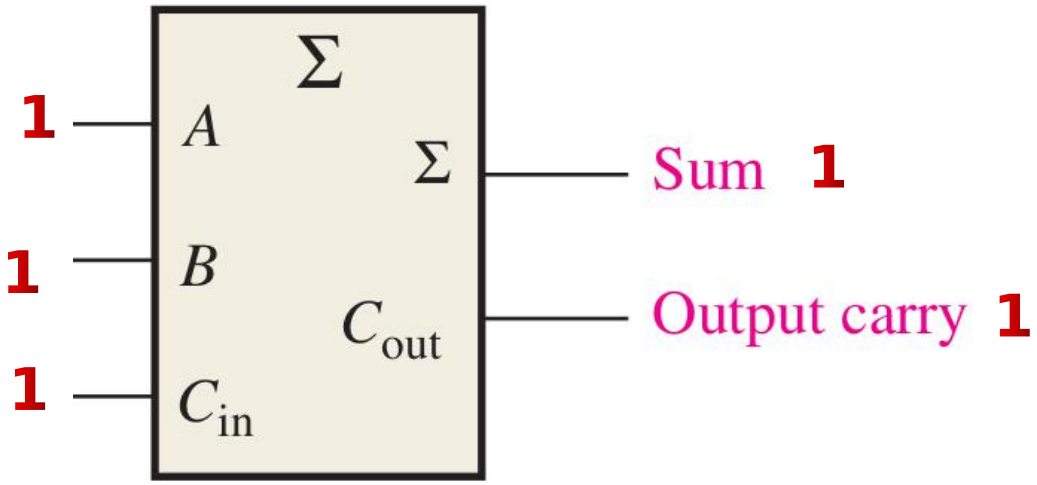


TABLE 6–2

Full-adder truth table.

A	B	C_{in}	C_{out}	Σ
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

C_{in} = input carry, sometimes designated as CI
C_{out} = output carry, sometimes designated as CO
Σ = sum
A and B = input variables (operands)

The Full-Adder

Full-Adder Logic

The full-adder must add the two input bits and the input carry. From the half-adder you know that the sum of the input bits A and B is the exclusive-OR of those two variables, $A \oplus B$. For the input carry (C_{in}) to be added to the input bits, it must be exclusive-ORed with $A \oplus B$, yielding the equation for the sum output of the full-adder.

$$\Sigma = (A \oplus B) \oplus C_{in} \qquad \text{Equation 6–3}$$

$$C_{out} = AB + (A \oplus B)C_{in} \qquad \text{Equation 6–4}$$

The Full-Adder

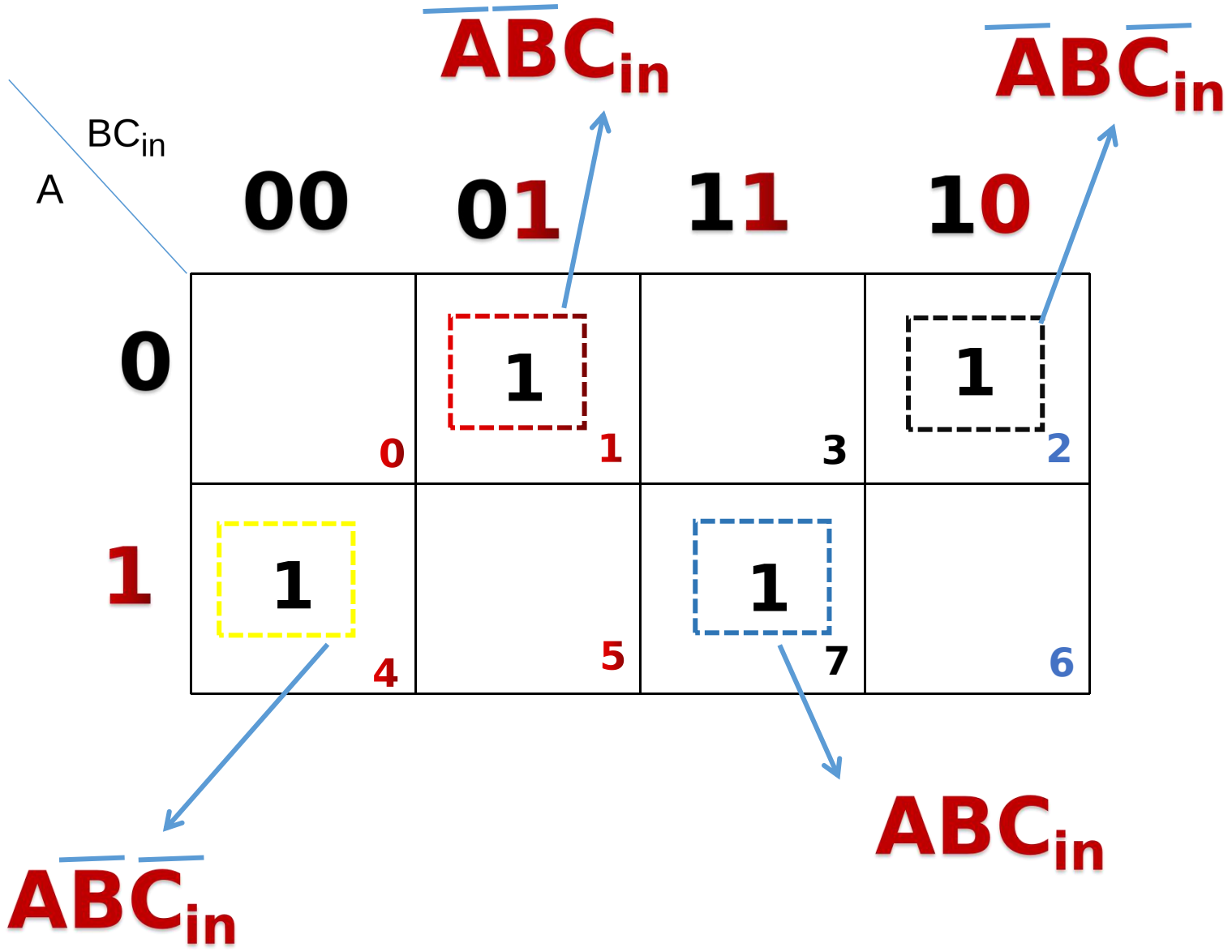
TABLE 6–2

Full-adder truth table.

A	B	C_{in}	C_{out}	Σ
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

C_{in} = input carry, sometimes designated as *CI*
C_{out} = output carry, sometimes designated as *CO*
Σ = sum
A and B = input variables (operands)

k-Map For Sum(Σ)



The Full-Adder

$$S = \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + ABC + A\overline{B}\,\overline{C}$$

$$S = \overline{A}\,\overline{B}C + ABC + \overline{A}B\overline{C} + A\overline{B}\,\overline{C}$$

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

$$Let \quad \overline{A}B + A\overline{B} = X$$

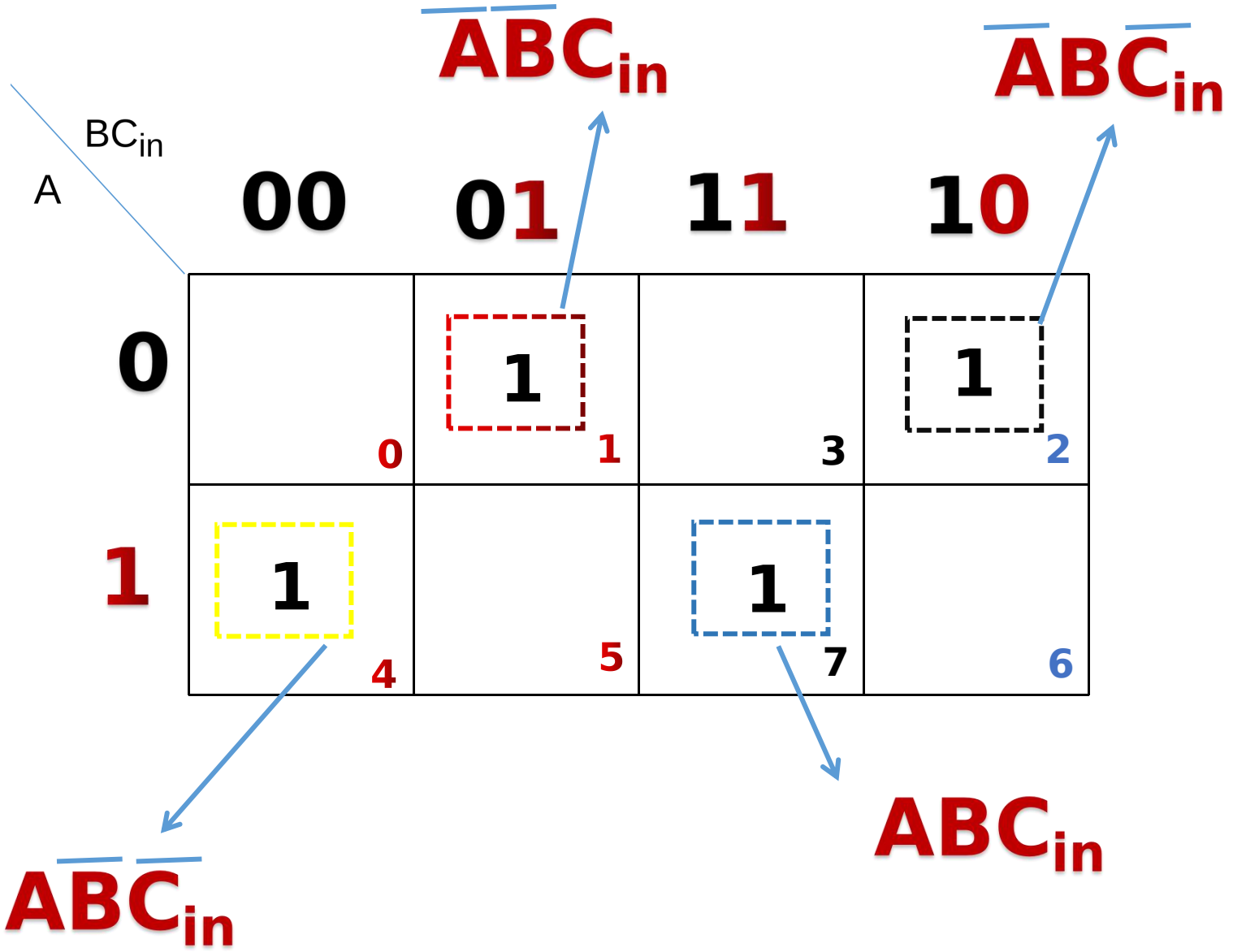
$$\therefore S = C(\overline{X}) + \overline{C}(X)$$

$$S = C \oplus X$$

$$Let \quad X = A \oplus B$$

$$\therefore S = C \oplus A \oplus B$$

k-Map For Sum(Σ)



The Full-Adder

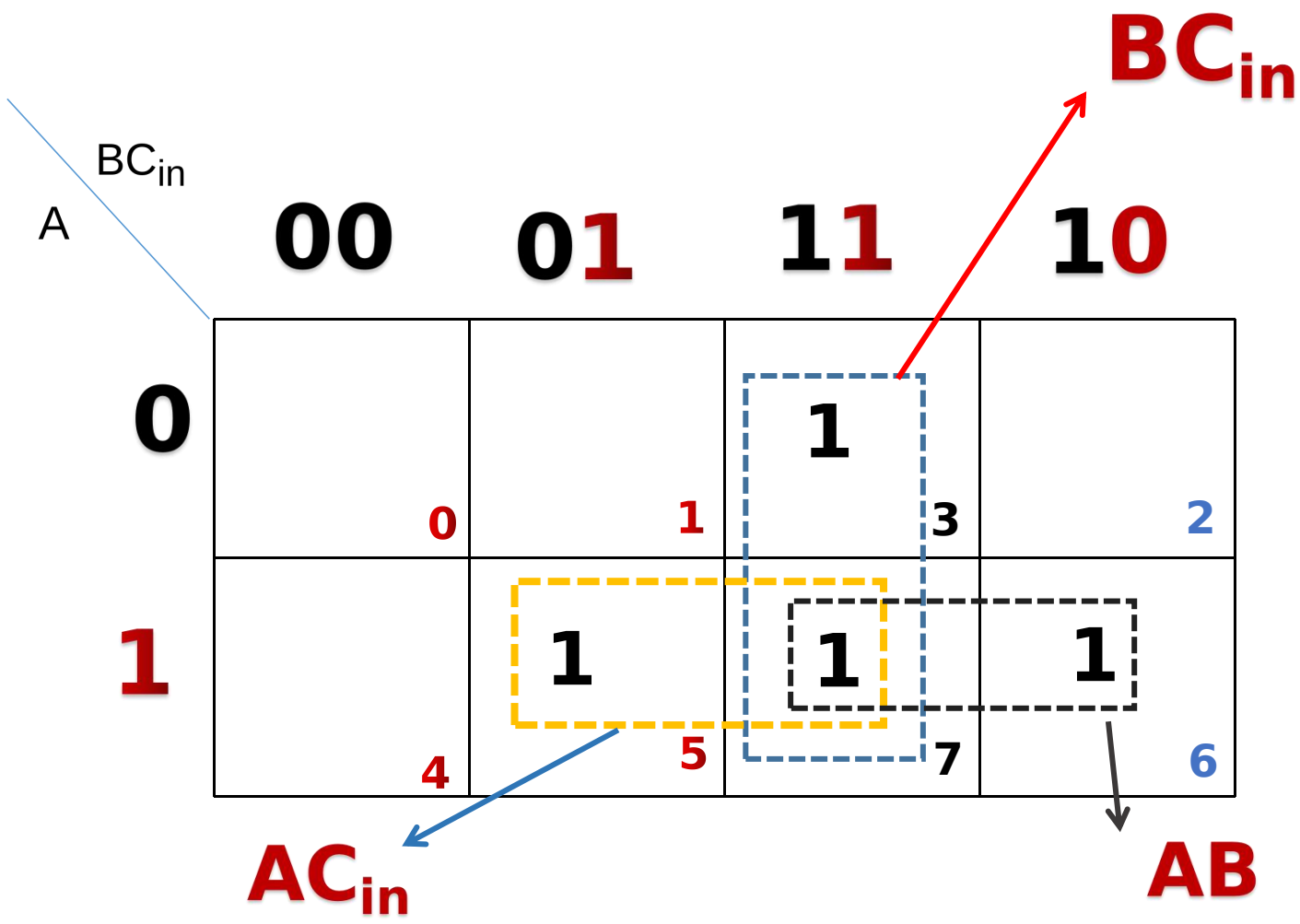
TABLE 6–2

Full-adder truth table.

A	B	C_{in}	C_{out}	Σ
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

C_{in} = input carry, sometimes designated as *CI*
C_{out} = output carry, sometimes designated as *CO*
Σ = sum
A and B = input variables (operands)

k-Map For CARRY



$$= AB + AC_{in} + BC_{in}$$

The Full-Adder

TABLE 6–2

Full-adder truth table.

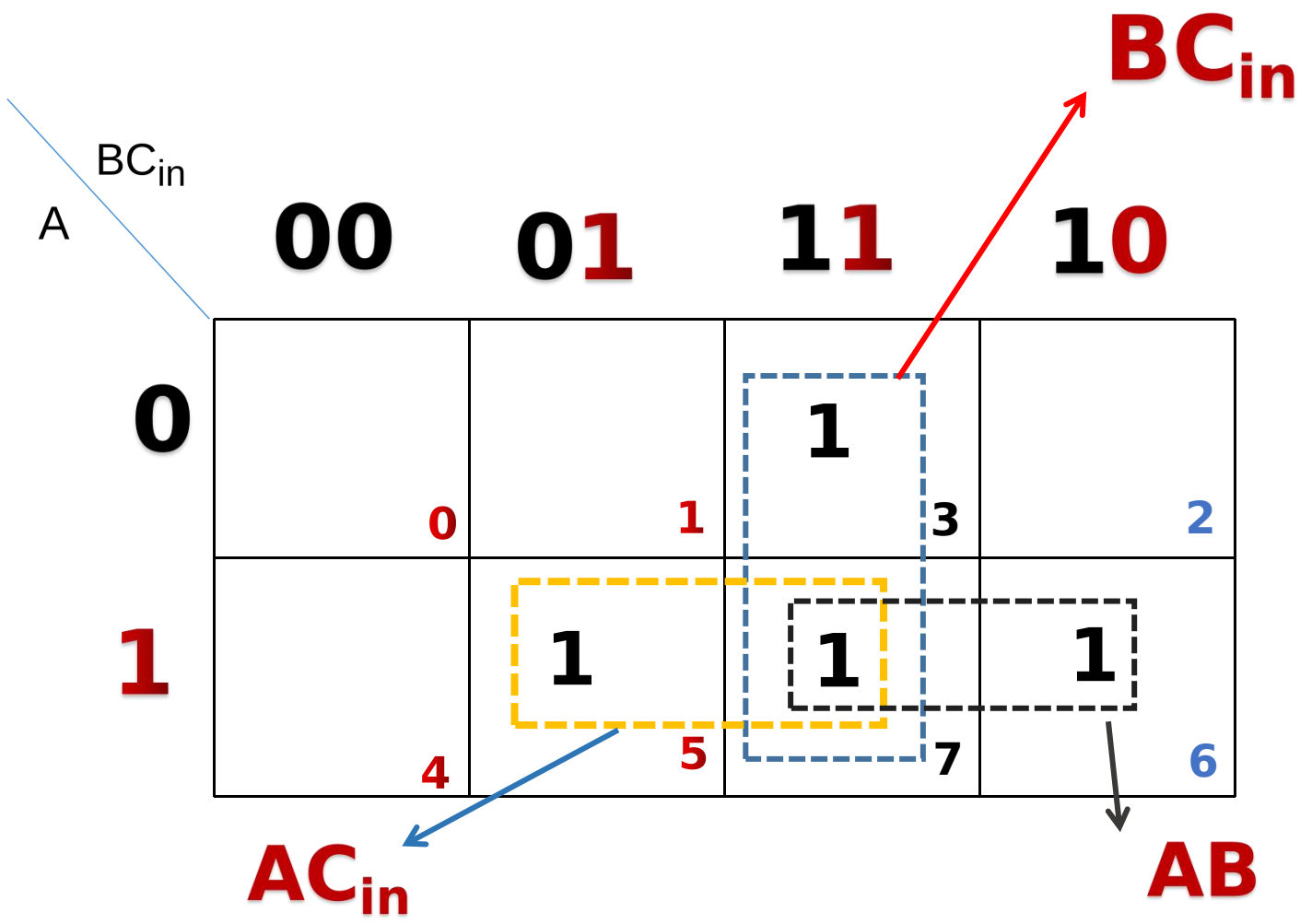
A	B	C_{in}	C_{out}	Σ
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

C_{in} = input carry, sometimes designated as CI
C_{out} = output carry, sometimes designated as CO
Σ = sum
A and B = input variables (operands)

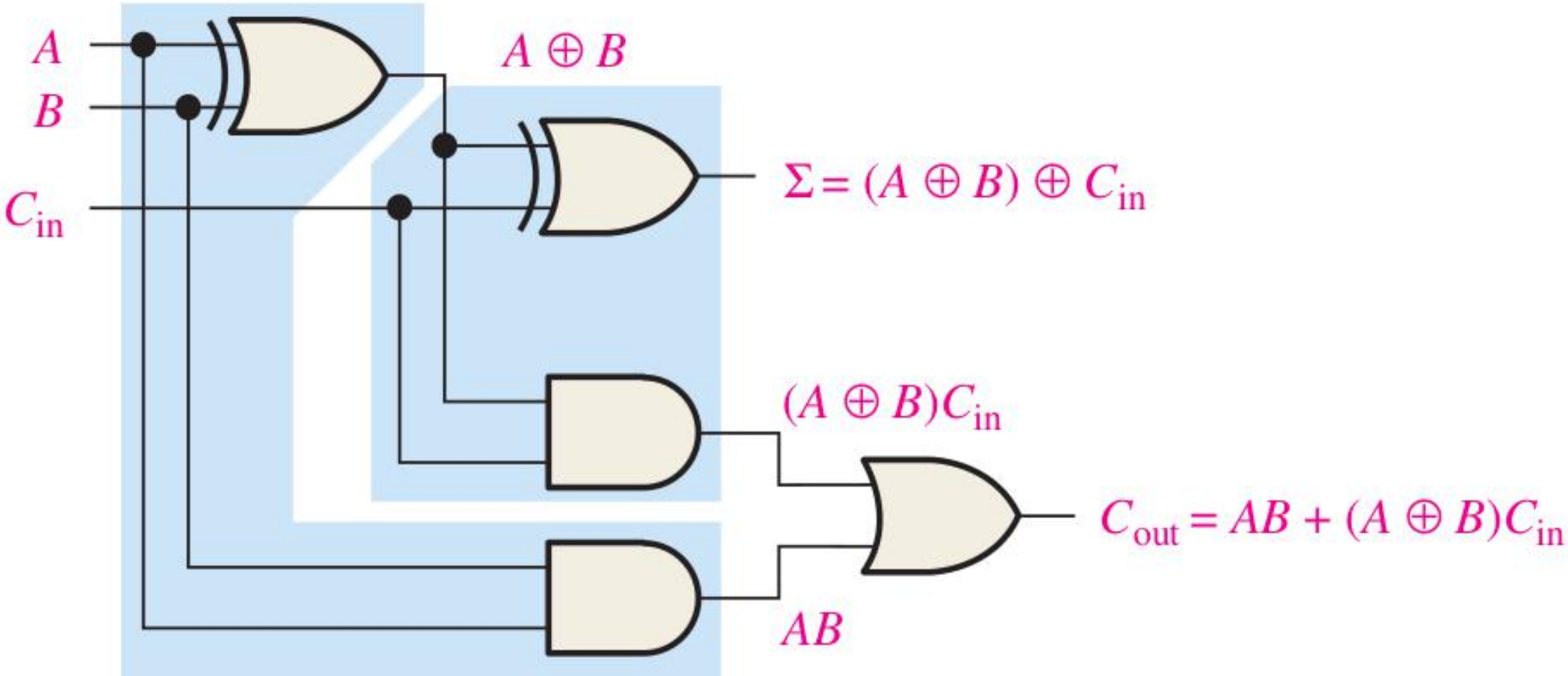
$$= AB + AC_{in} + BC_{in}$$

$$= AB + C_{in}(A + B)$$

k-Map For CARRY



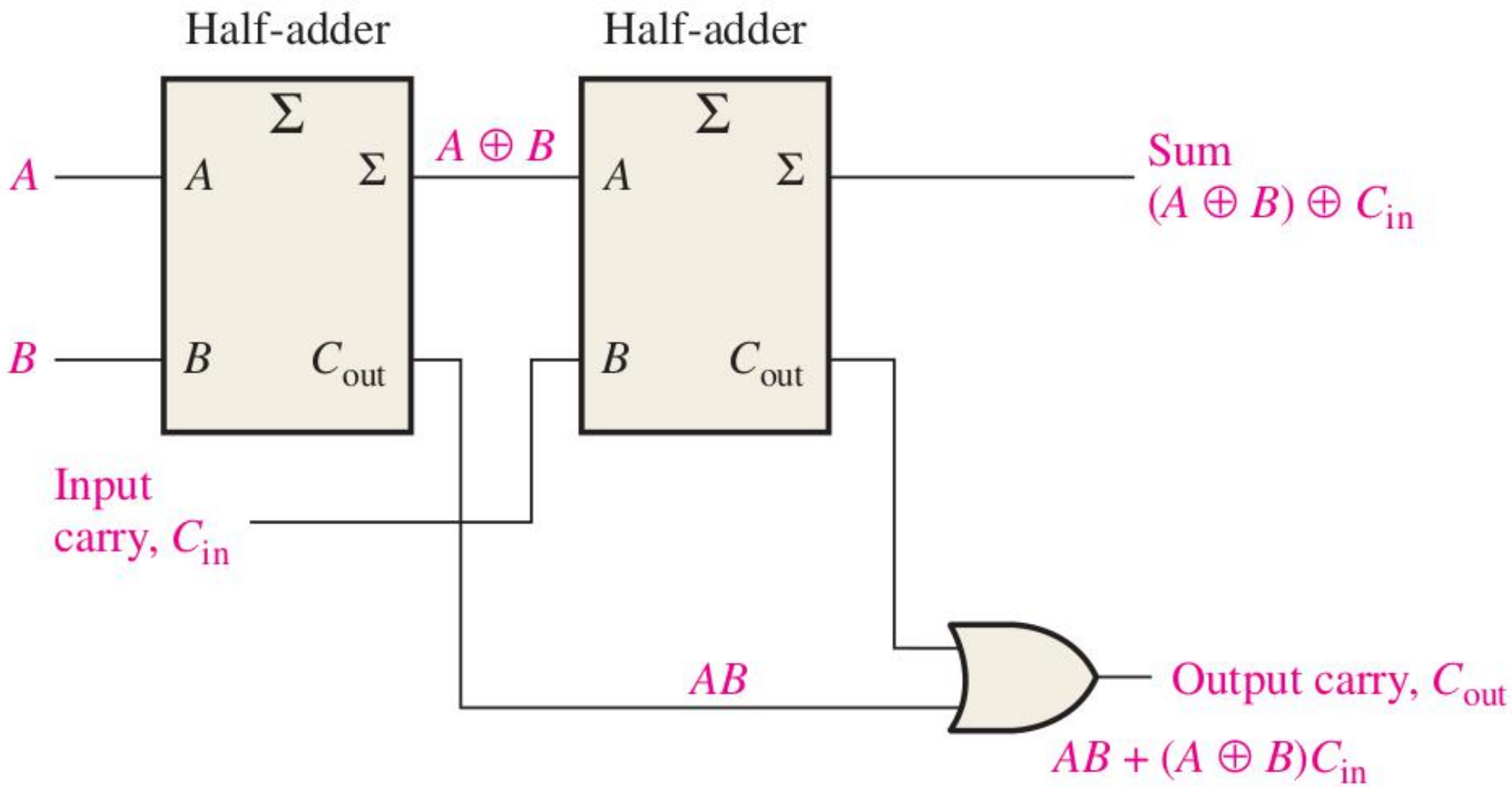
The Full-Adder



(b) Complete logic circuit for a full-adder (each half-adder is enclosed by a shaded area)

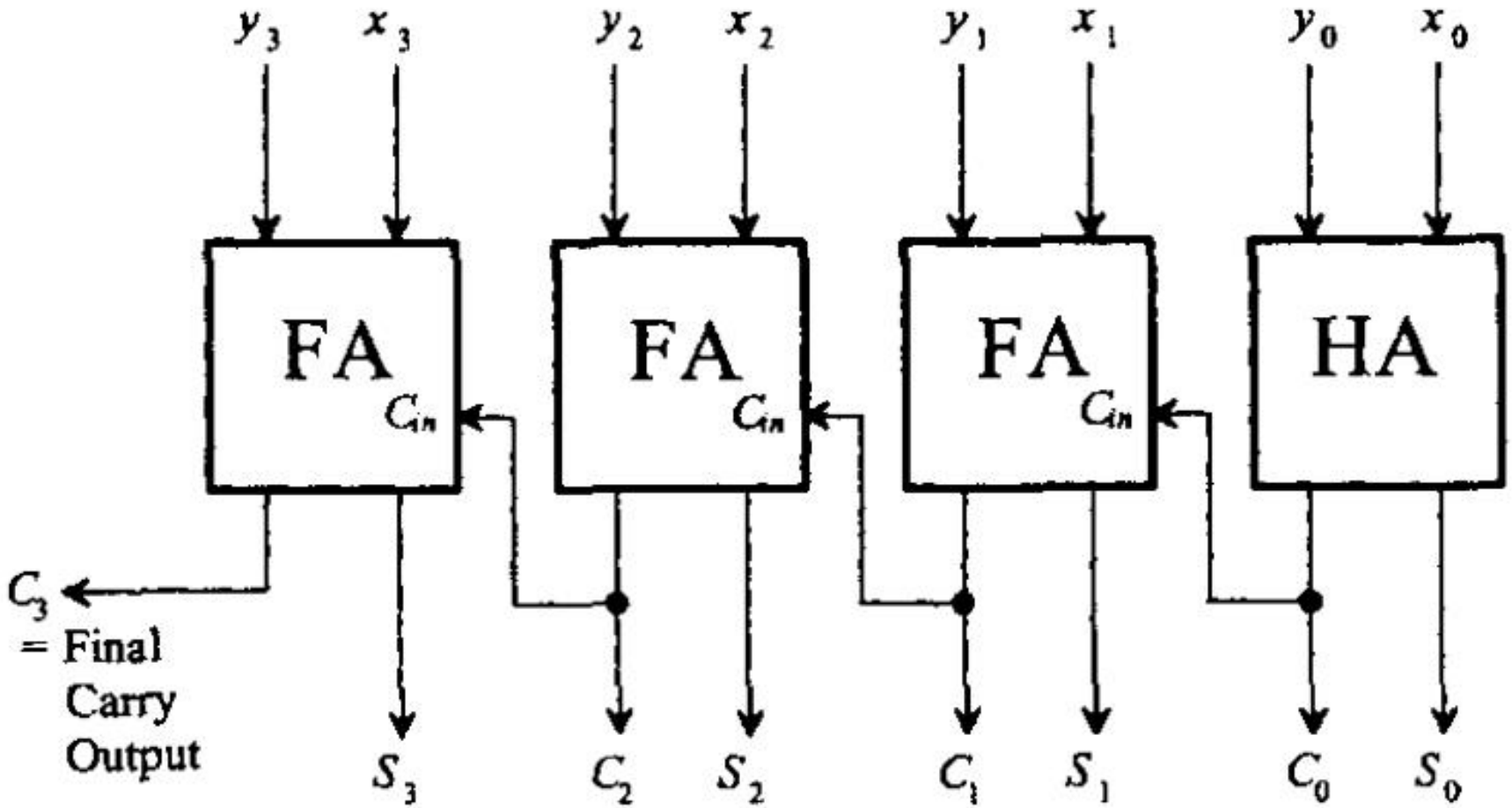
The Full-Adder

FULL ADDER



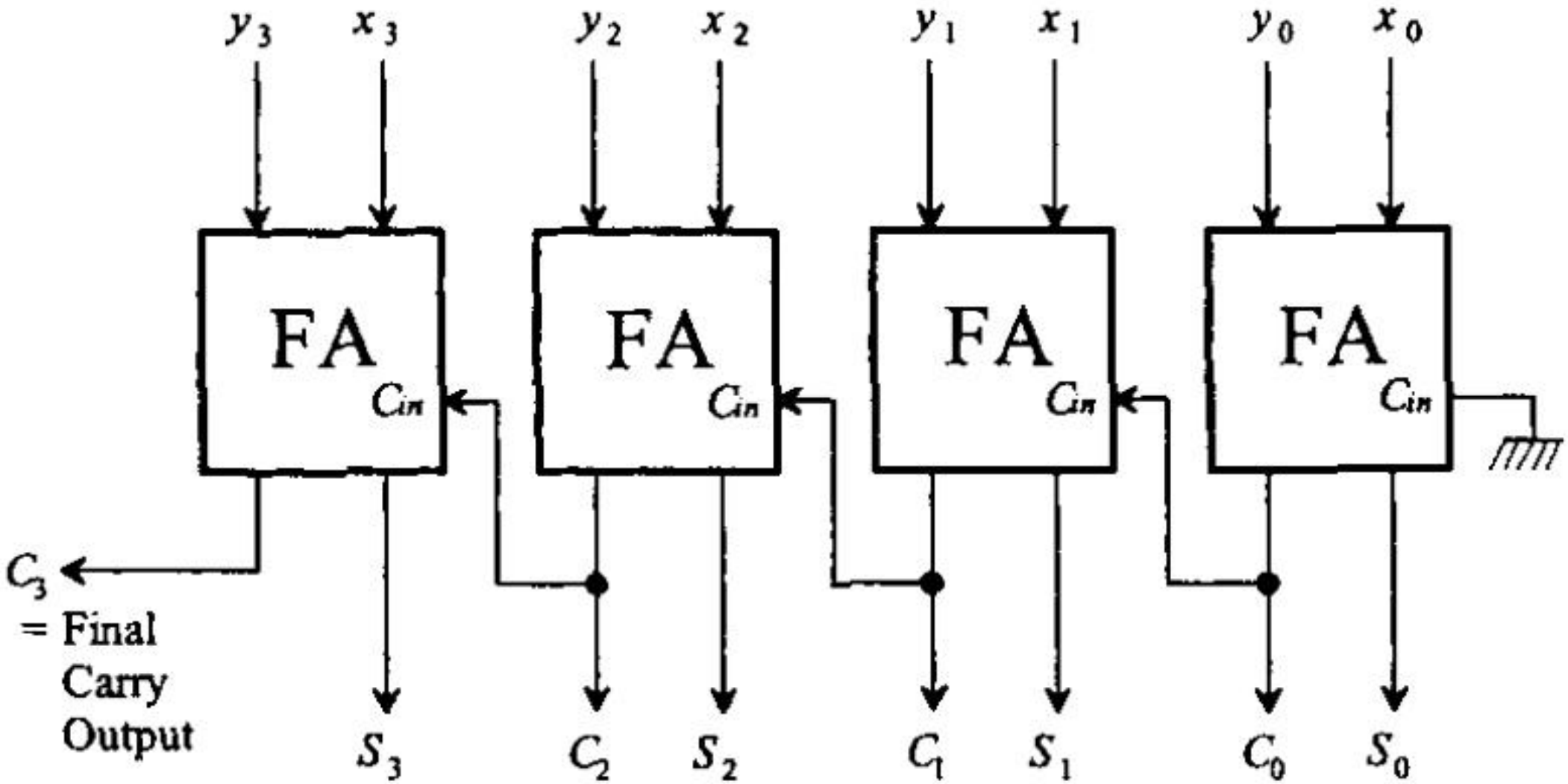
(a) Arrangement of two half-adders to form a full-adder

The Full-Adder



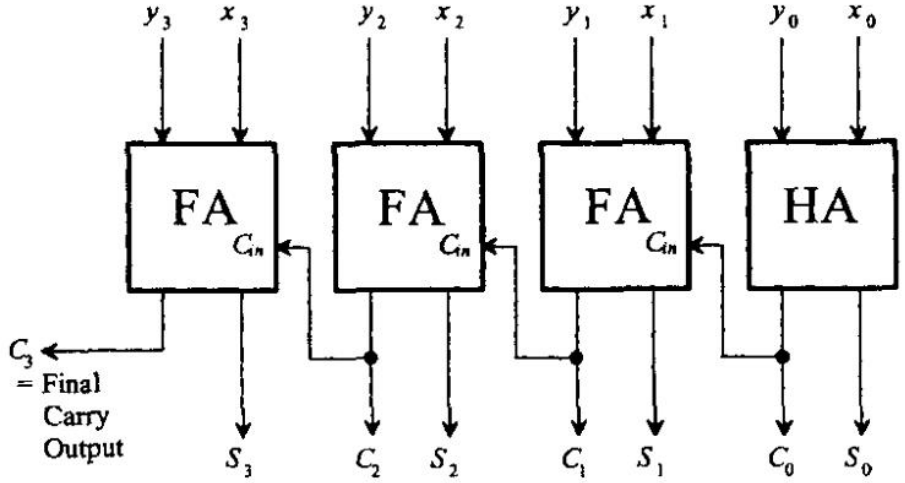
4-bit binary adder using one half-adder and three full adders

The Full-Adder



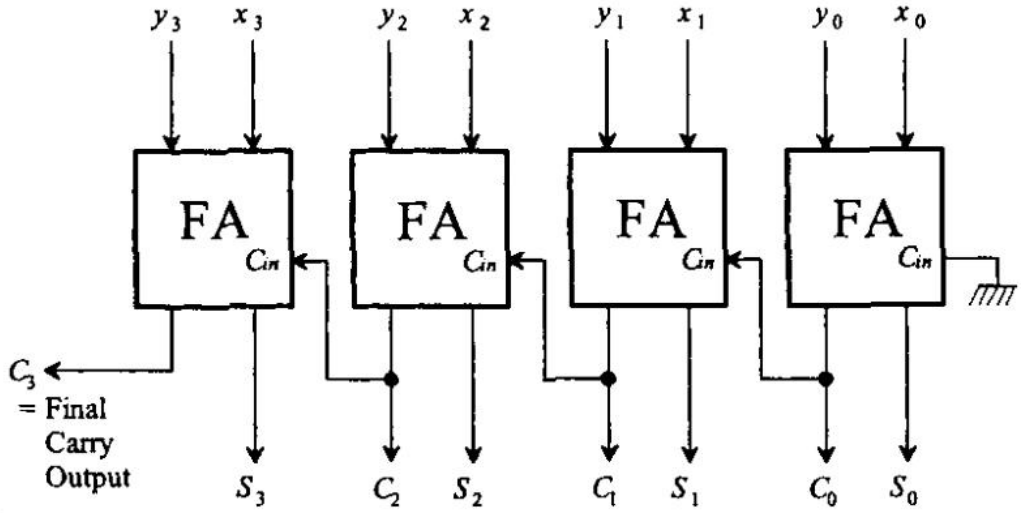
Four-bit binary adder using full adders

The Full-Adder



4.10

4-bit binary adder using one half-adder and three full adders



4.11

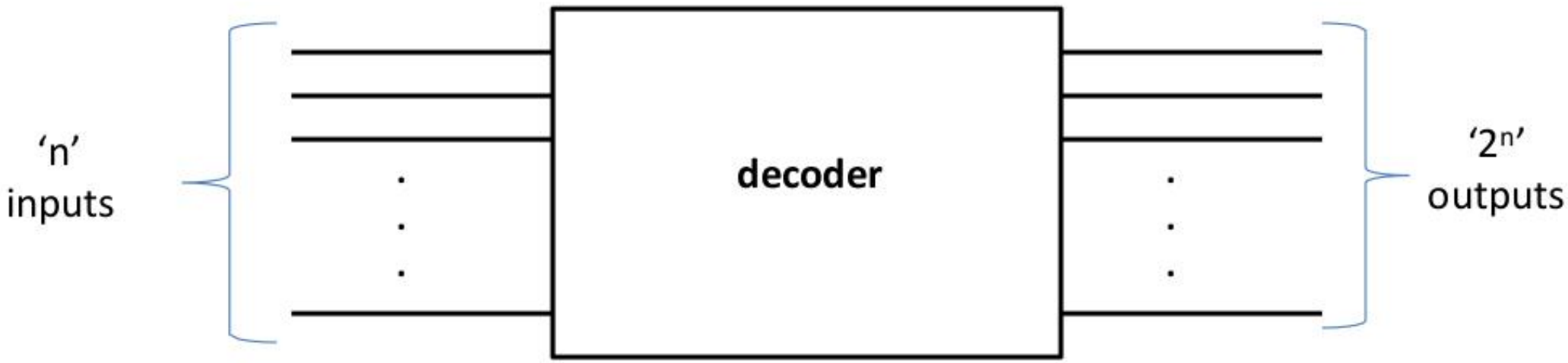
Four-bit binary adder using full adders

A 4-bit binary adder (also called "Ripple Carry Adder") for adding two 4-bit numbers $x_3\,x_2\,x_1\,x_0$ and $y_3\,y_2\,y_1\,y_0$ can be implemented using one half-adder and three full adders as shown in Figure 4.10. A full adder adds two bits if one of its inputs $C_{in} = 0$. This means that the half-adder in Figure 4.10 can be replaced by a full adder with its C_{in} connected to ground. Figure 4.11 shows implementation of a 4-bit binary adder using four full adders.

DECODER

The decoder is a *combinatorial logic circuit(CLC).*

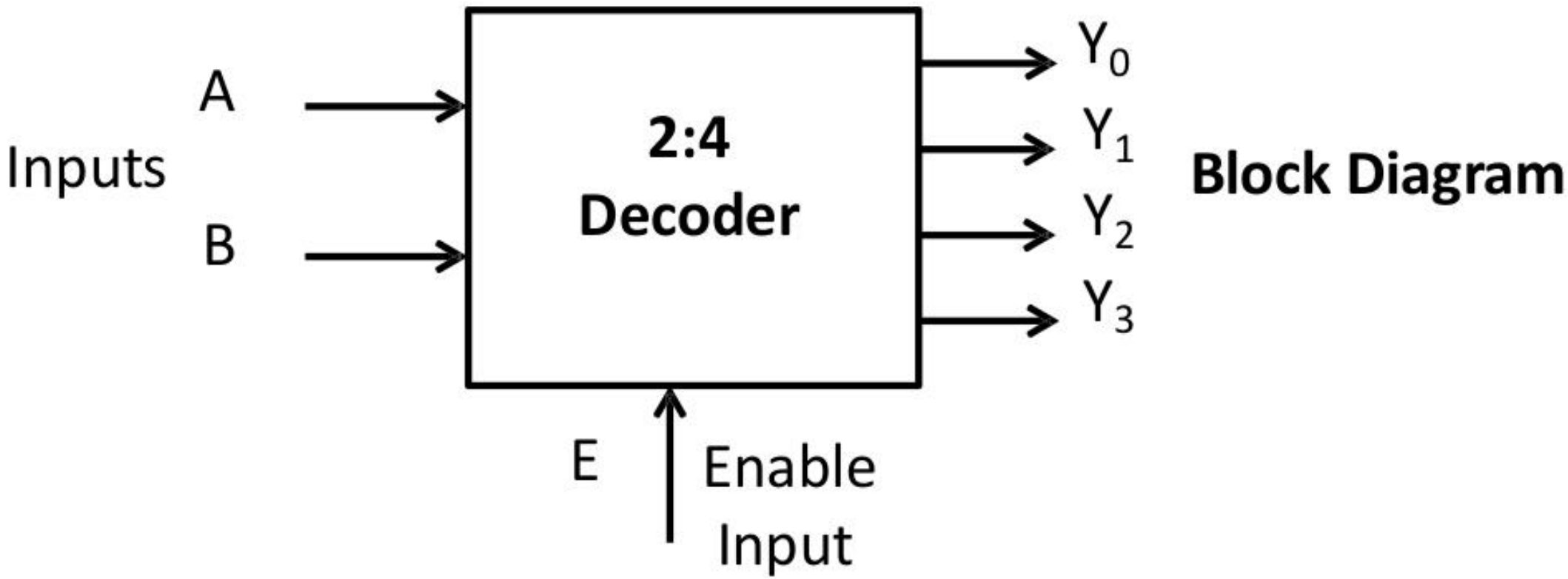
If there are *n input terminals*, then a complete binary decoder has *2^n output terminals.*



Types of Decoders

- ✓ 2 to 4 line Decoder
- ✓ 3 to 8 line Decoder
- ✓ BCD to 7 Segment Decoder

2 to 4 Line Decoder

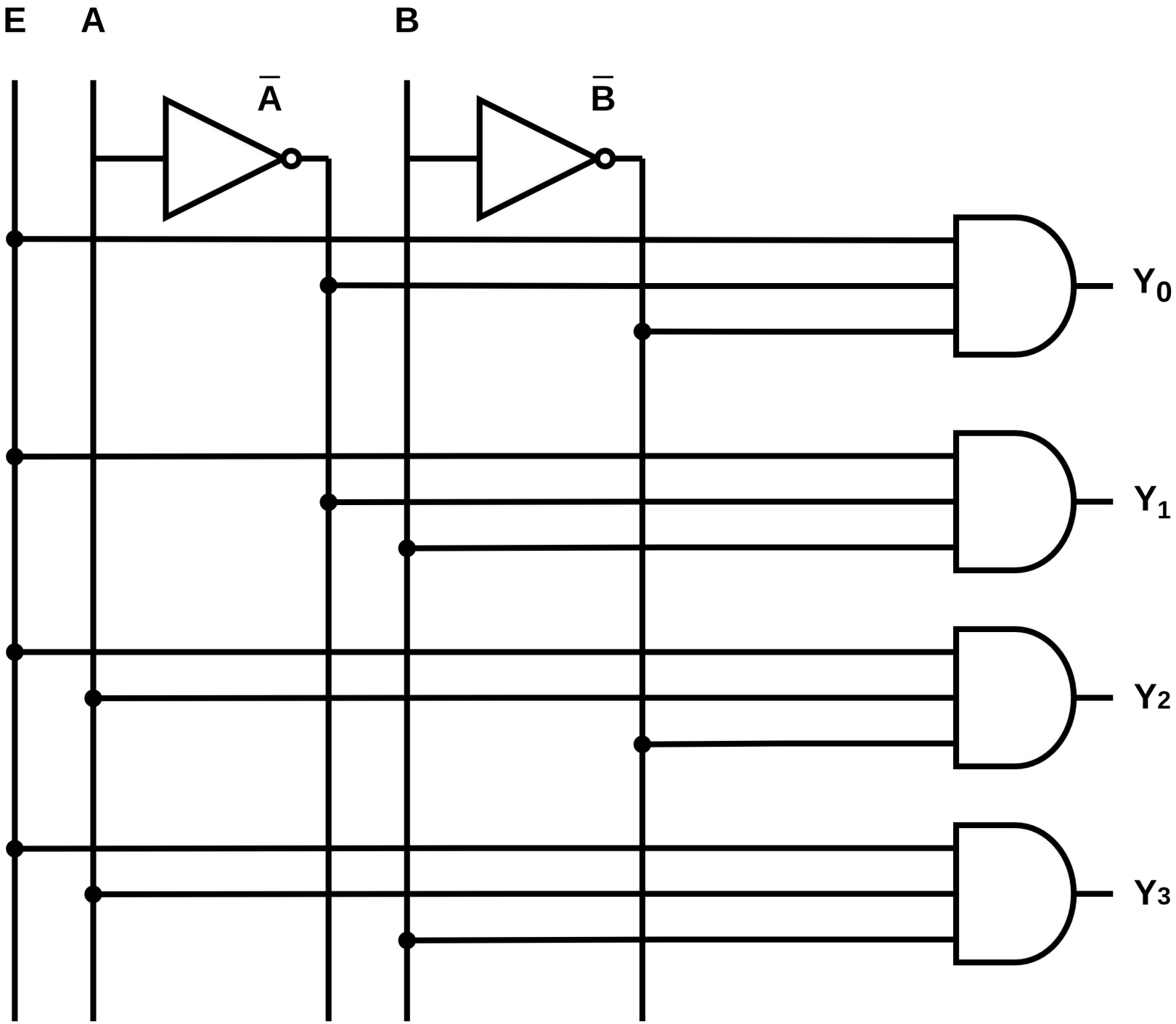


Block Diagram

2 to 4 Line Decoder

Truth Table

Enable i/p	Data Inputs		Outputs			
E	A	B	Y_0	Y_1	Y_2	Y_3
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

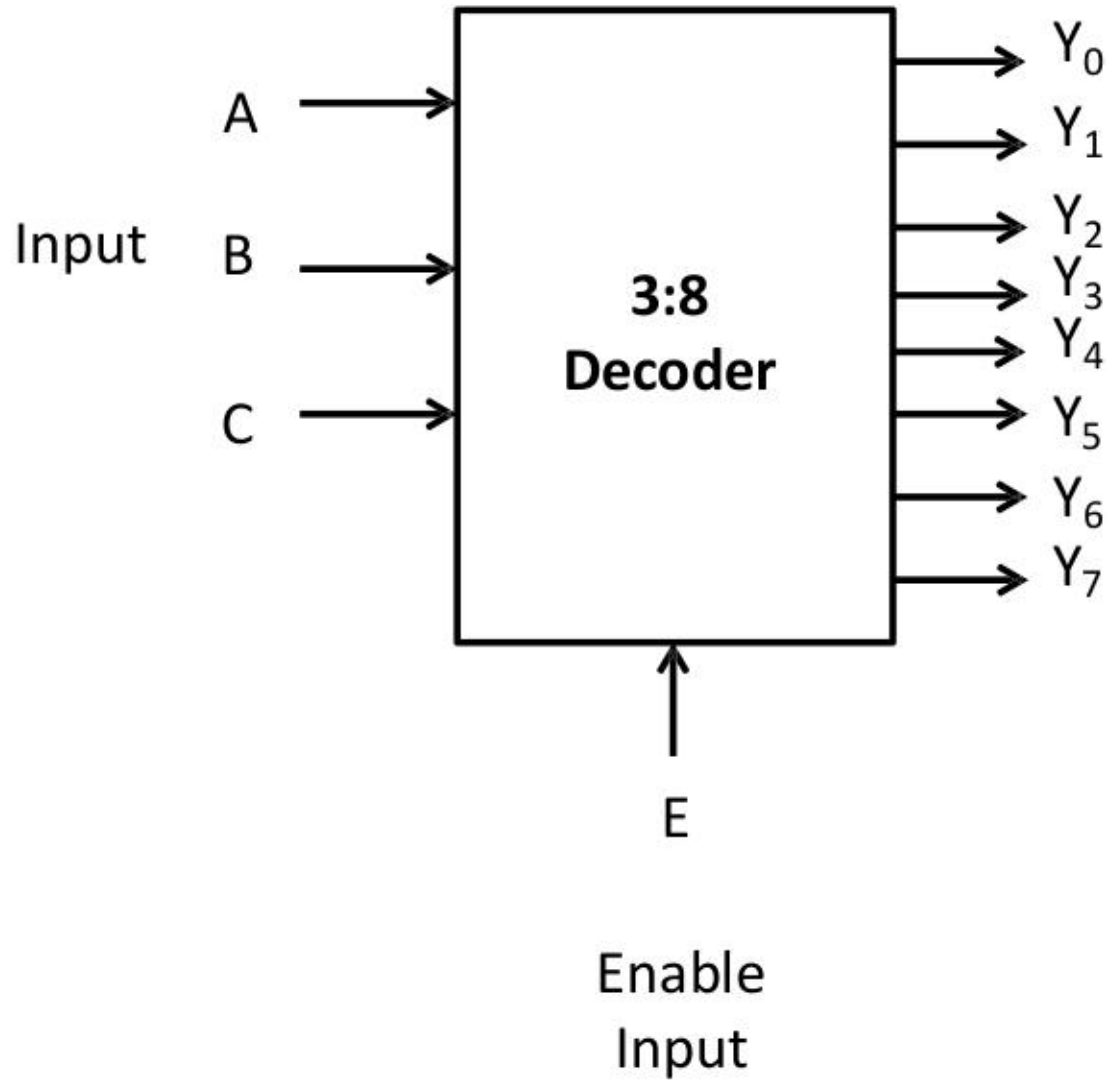


Enable i/p	Data Inputs		Outputs			
E	A	B	Y_0	Y_1	Y_2	Y_3
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

3 to 8 Line Decoder

3 to 8 Line Decoder

Block Diagram



3 to 8 Line Decoder

Truth Table

Enable i/p	Inputs			Outputs							
E	A	B	C	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	X	X	X	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

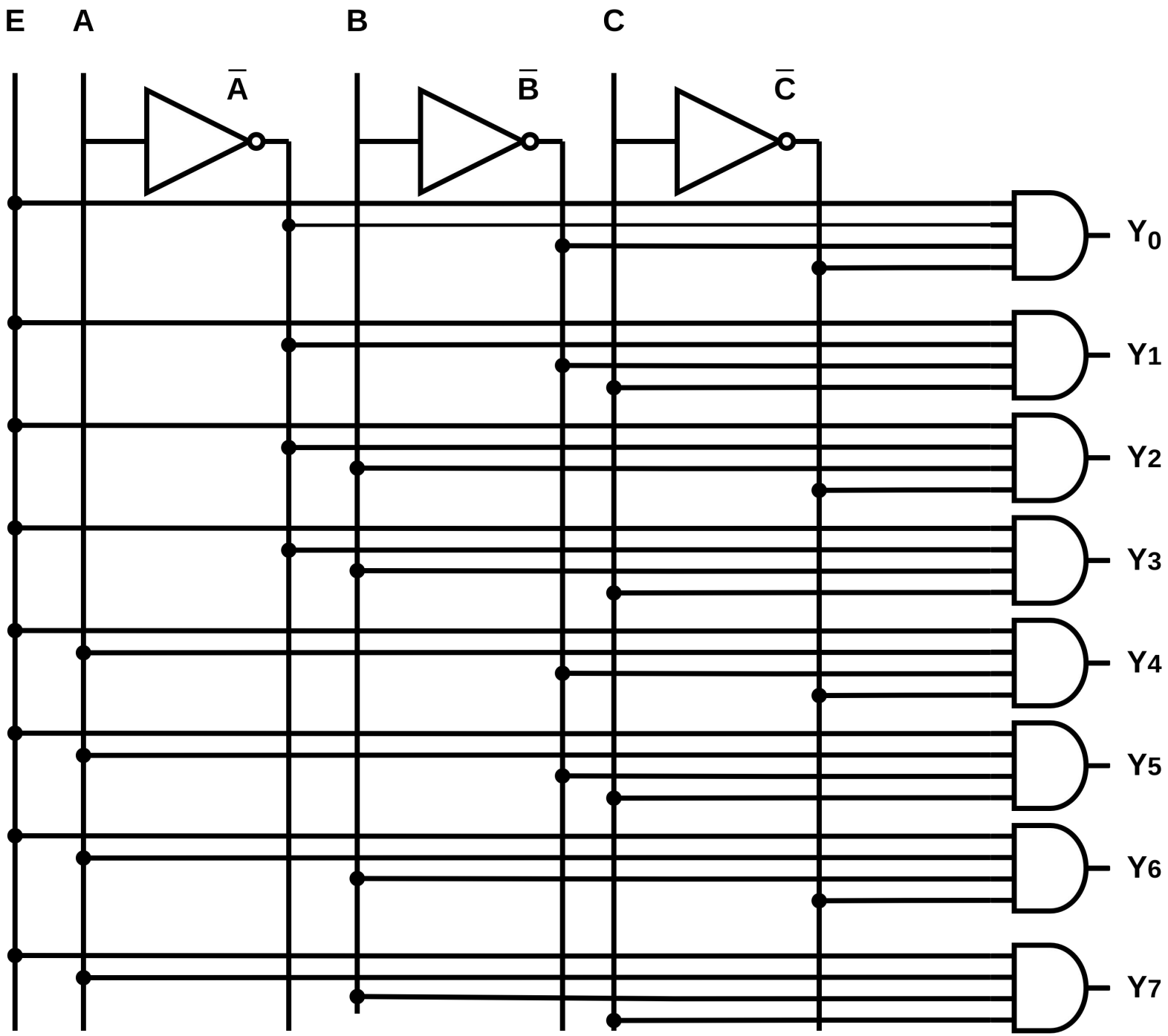
3 to 8 Line Decoder

Truth Table

Enable i/p	Inputs			Outputs							
E	A	B	C	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	X	X	X	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

3 to 8 Line Decoder

Inputs			
A	B	C	
X	X	X	
0	0	0	$\overline{A} \cdot \overline{B} \cdot \overline{C}$
0	0	1	$\overline{A} \cdot \overline{B} \cdot C$
0	1	0	$\overline{A} \cdot B \cdot \overline{C}$
0	1	1	$\overline{A} \cdot B \cdot C$
1	0	0	$A \cdot \overline{B} \cdot \overline{C}$
1	0	1	$A \cdot \overline{B} \cdot C$
1	1	0	$A \cdot B \cdot \overline{C}$
1	1	1	$A \cdot B \cdot C$



$$\overline{A} \cdot \overline{B} \cdot \overline{C}$$

$$\overline{A} \cdot \overline{B} \cdot C$$

$$\overline{A} \cdot B \cdot \overline{C}$$

$$\overline{A} \cdot B \cdot C$$

$$A \cdot \overline{B} \cdot \overline{C}$$

$$A \cdot \overline{B} \cdot C$$

$$A \cdot B \cdot \overline{C}$$

$$A \cdot B \cdot C$$

Implement Full Adder Using Decoder

Implement Full Adder Using Decoder

For Full Adder, Number of inputs = 3 (A,B,C)

Decoder Size = 3:8

Number Of Outputs = 2 (Sum, Carry)

$$\boldsymbol{Sum = \Sigma m(1,2,4,7)}$$

$$\boldsymbol{Carry = \Sigma m(3,5,6,7)}$$

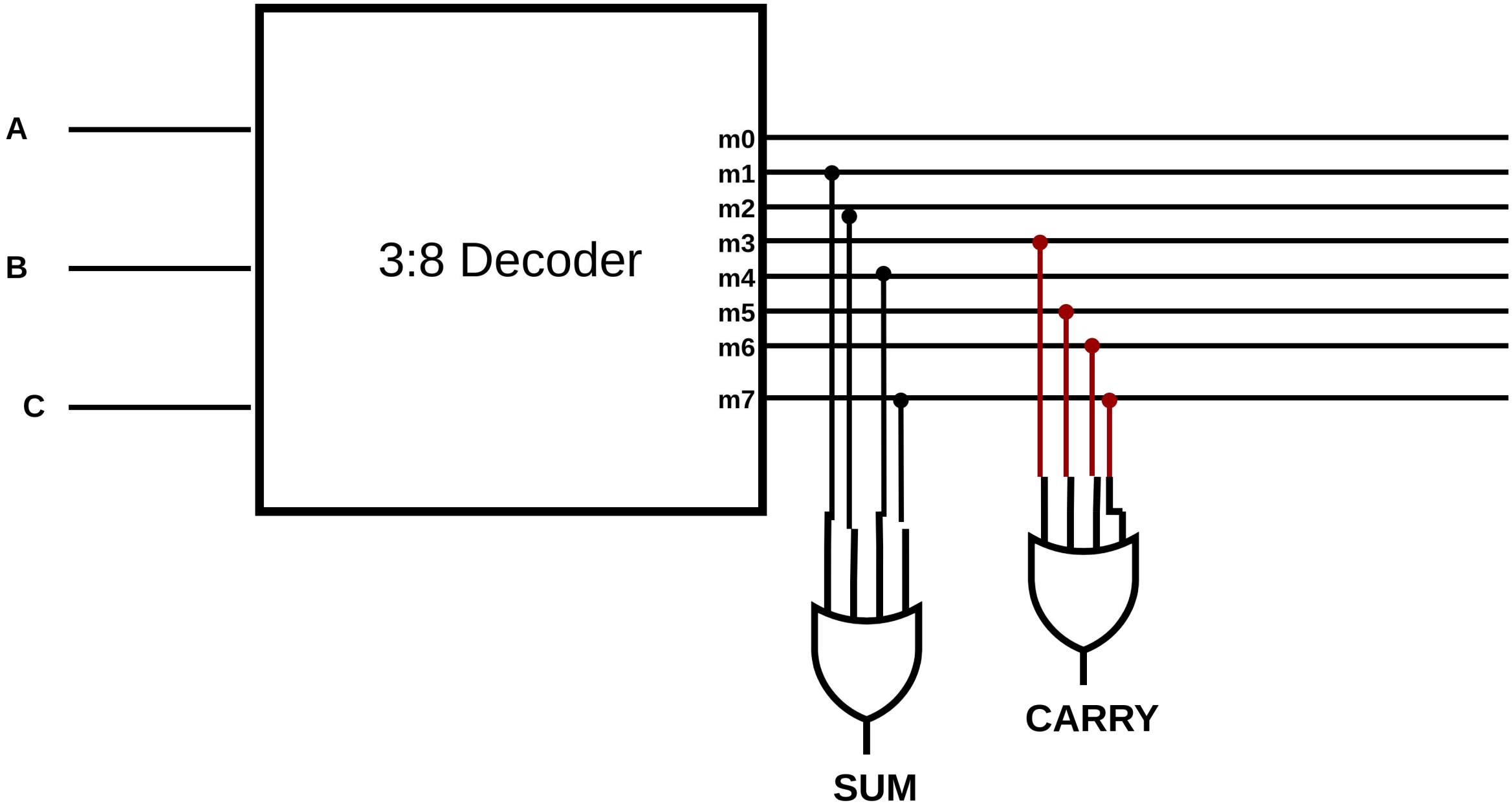
TABLE 6–2

Full-adder truth table.

A	B	C_{in}	C_{out}	Σ
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

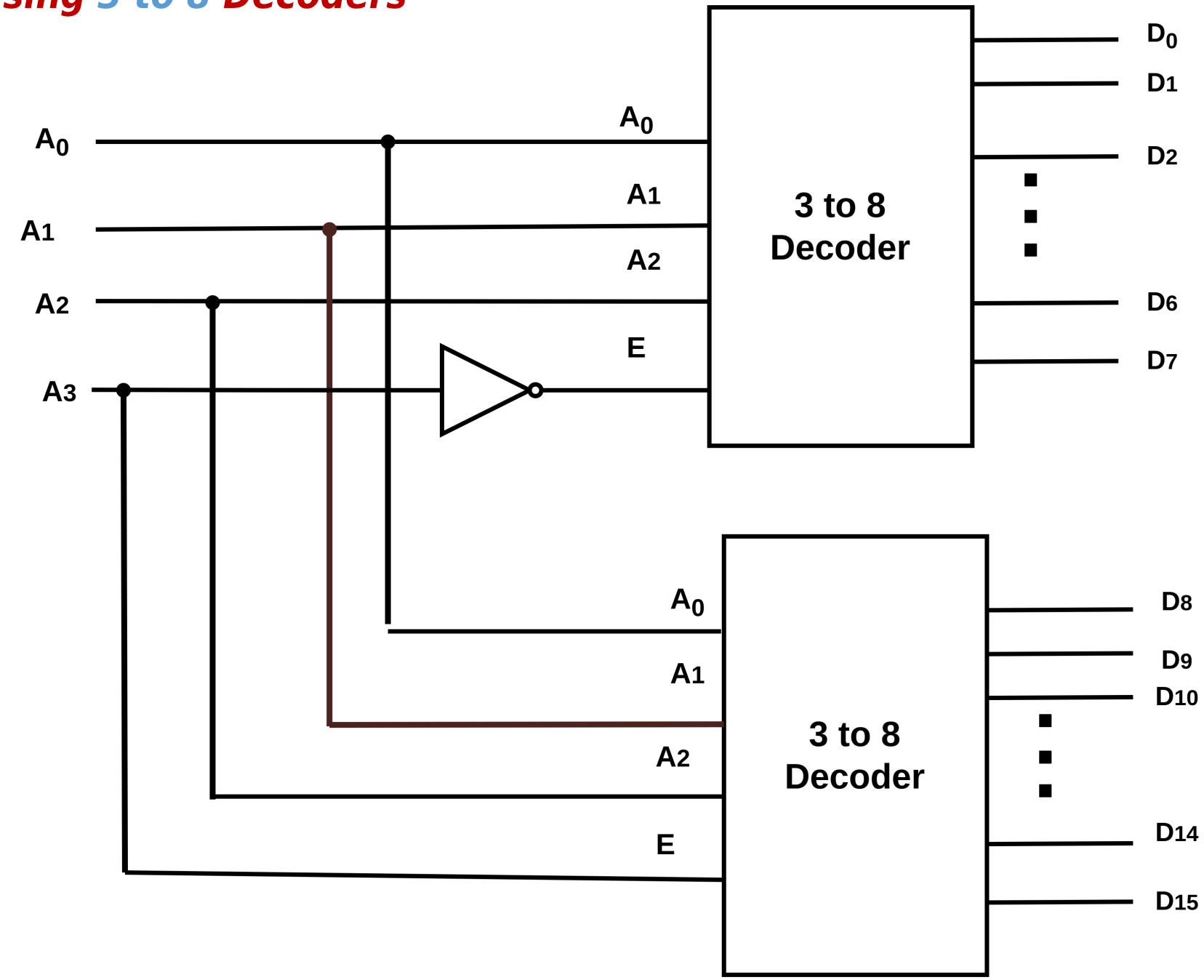
C_{in} = input carry, sometimes designated as CI
C_{out} = output carry, sometimes designated as CO
Σ = sum
A and B = input variables (operands)

Implement Full Adder Using Decoder



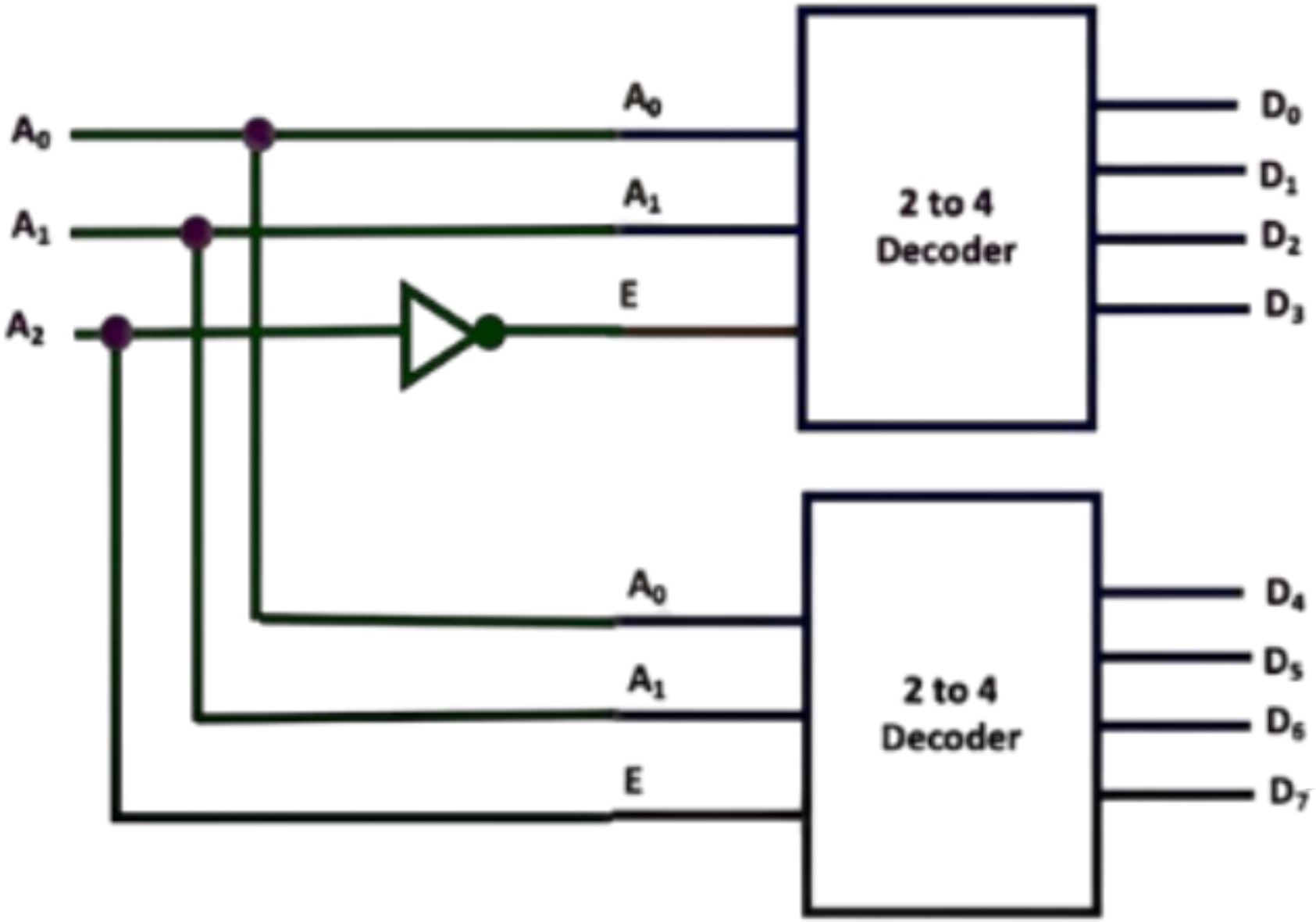
Implement 4 to 16 Decoder Using 3 to 8 Decoders

Implement 4 to 16 Decoder Using 3 to 8 Decoders

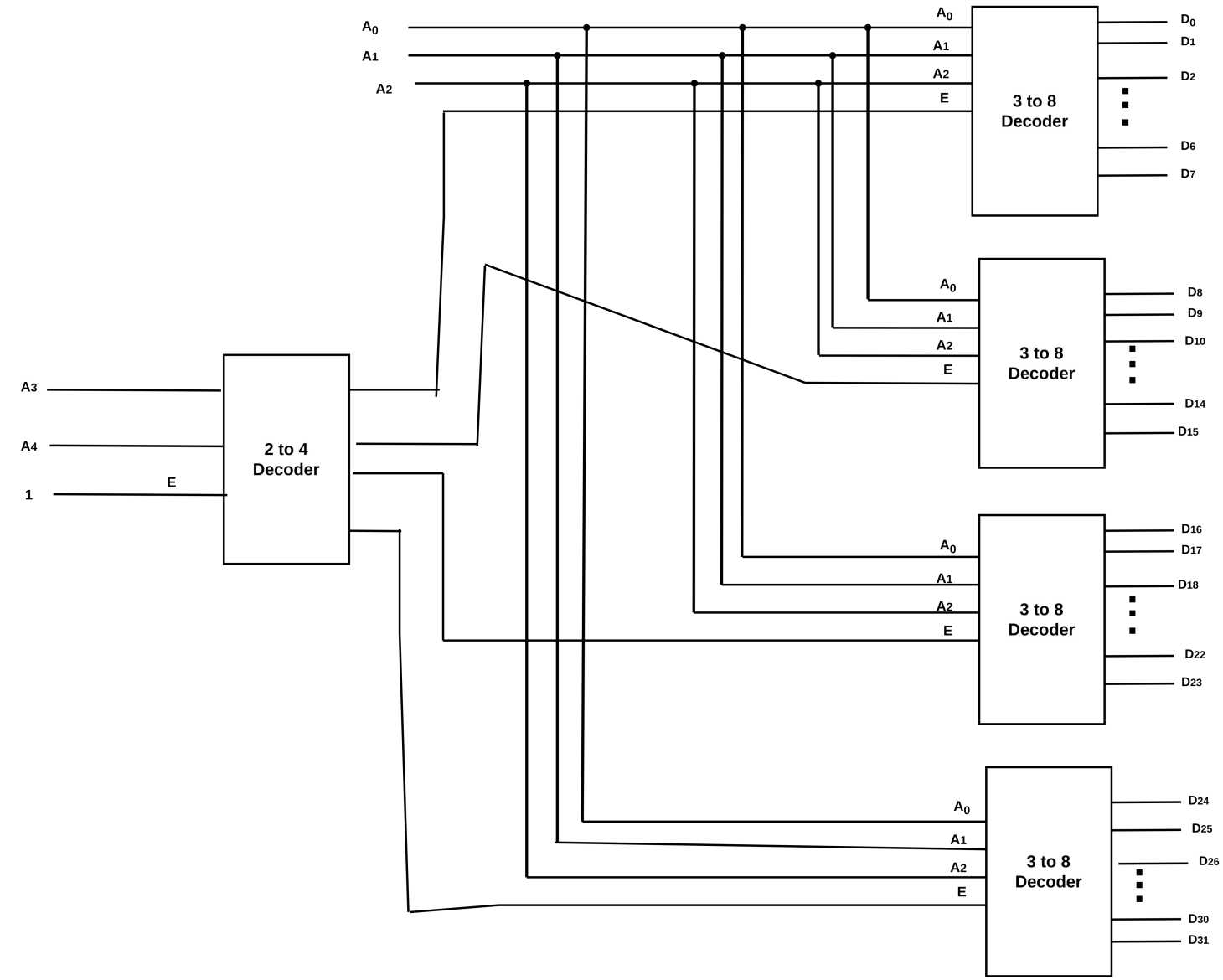


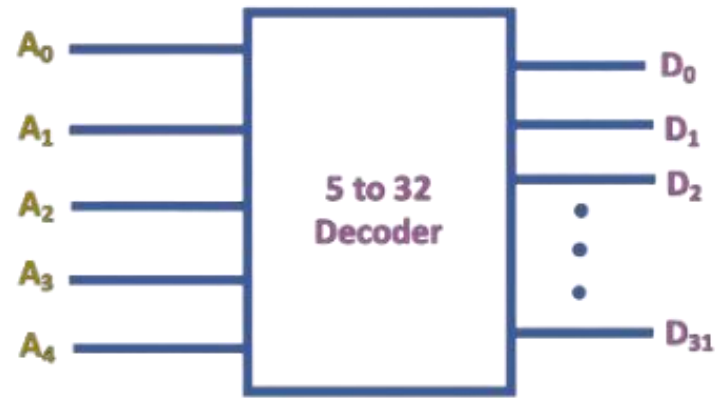
Implement 3 to 8 Decoder Using 2 to 4 Decoders

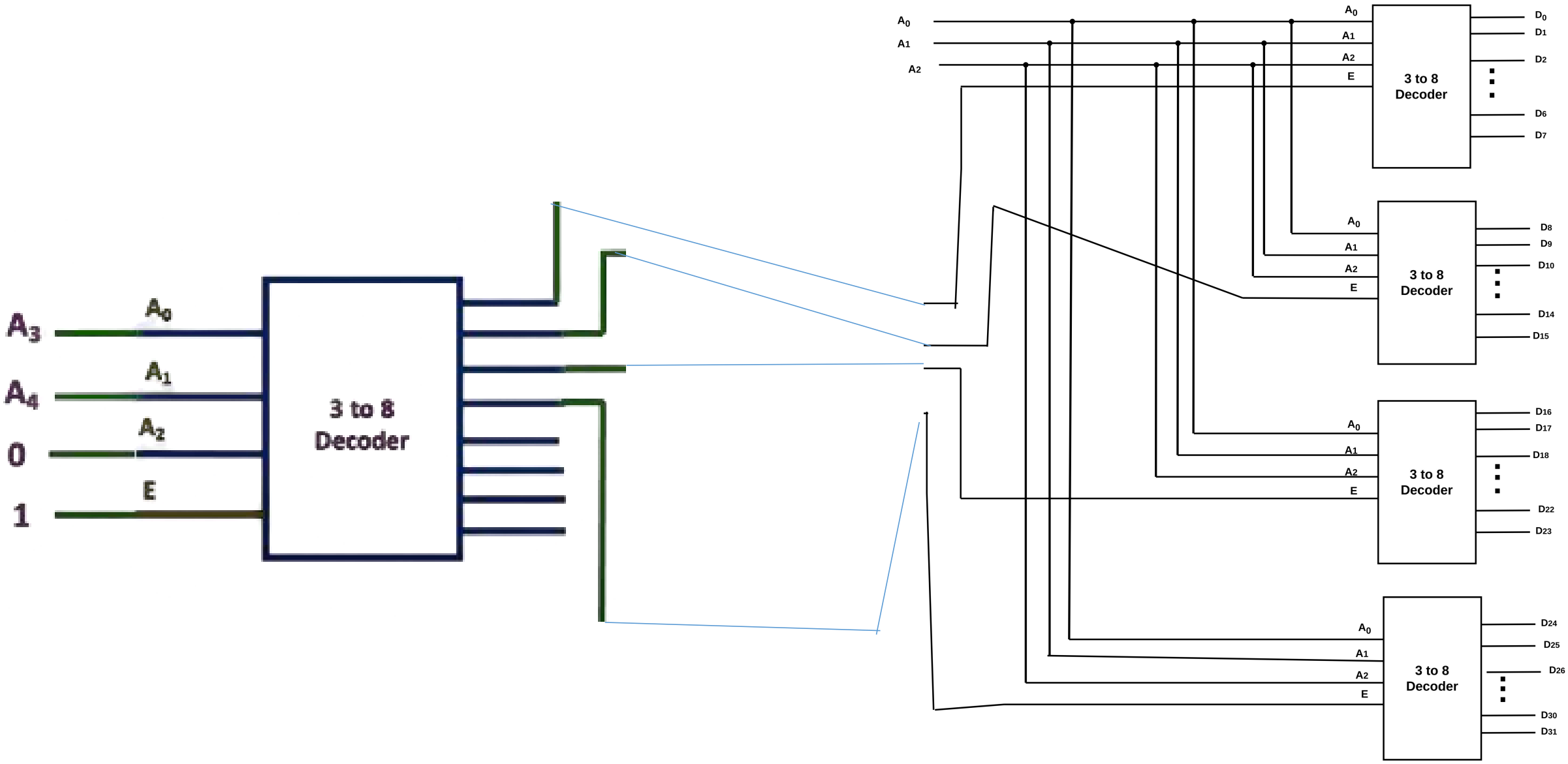
Implement 3 to 8 Decoder Using 2 to 4 Decoders



Implement 5 to 32 Decoder Using 3 to 8 Decoders

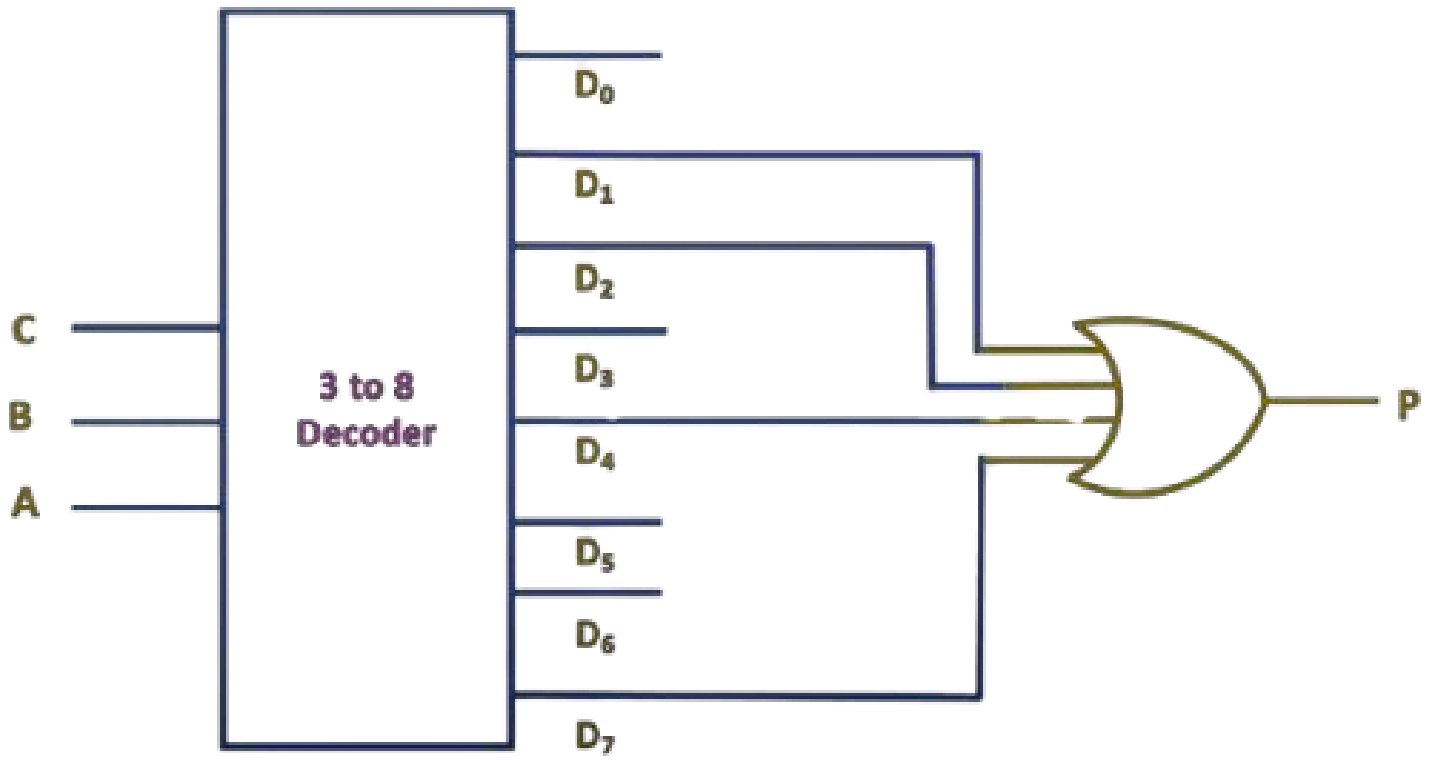

A0
A1
A2
A3
A4
1
E
2 to 4 Decoder
3 to 8 Decoder
3 to 8 Decoder
3 to 8 Decoder
3 to 8 Decoder
D0
D1
D2
D6
D7
D8
D9
D10
D14
D15
D16
D17
D18
D22
D23
D24
D25
D26
D30
D31


A0
A1
A2
A3
A4
5 to 32 Decoder
D0
D1
D2
D31


A3
A4
0
1
A0
A1
A2
E
3 to 8 Decoder
A0
A1
A2
E
3 to 8 Decoder
D0
D1
D2
D6
D7
A0
A1
A2
E
3 to 8 Decoder
D8
D9
D10
D14
D15
A0
A1
A2
E
3 to 8 Decoder
D16
D17
D18
D22
D23
A0
A1
A2
E
3 to 8 Decoder
D24
D25
D26
D30
D31

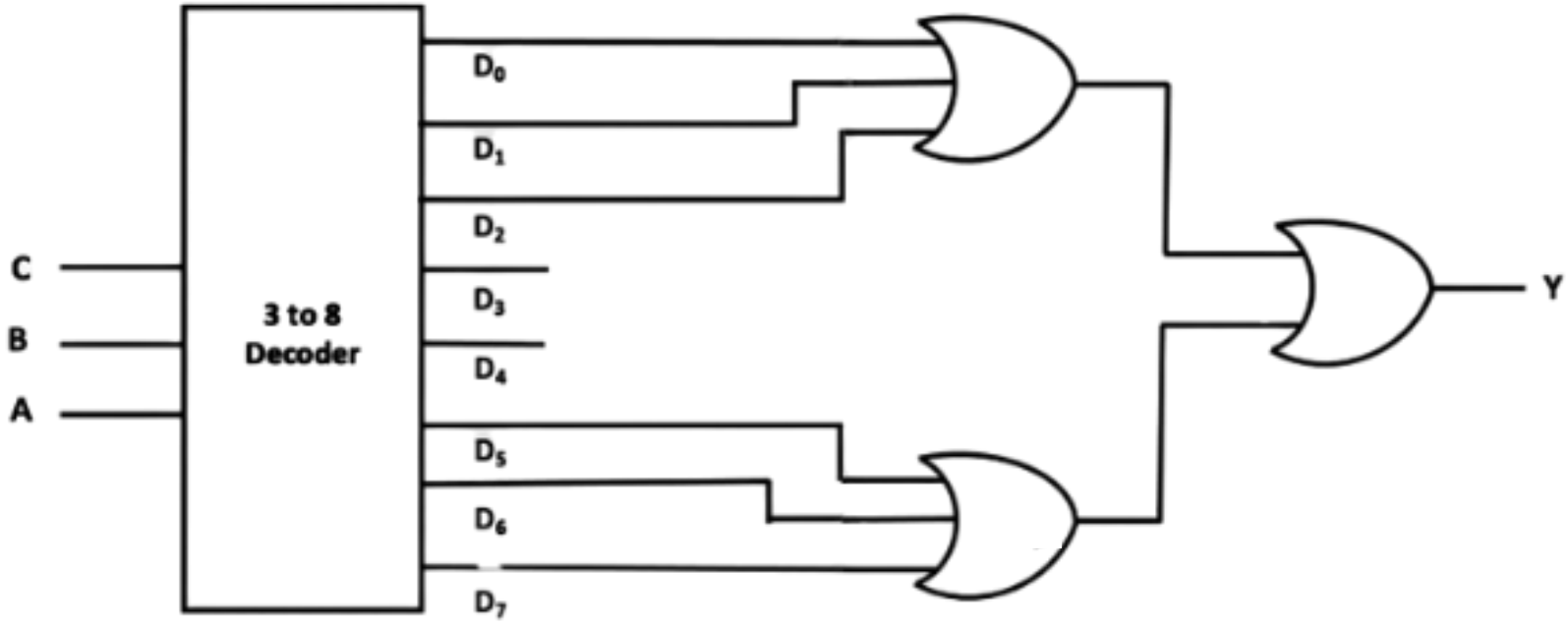
Even Parity Checker

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



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

Consider Following Truth Table(*O/P is randomly selected*)



ENCODER

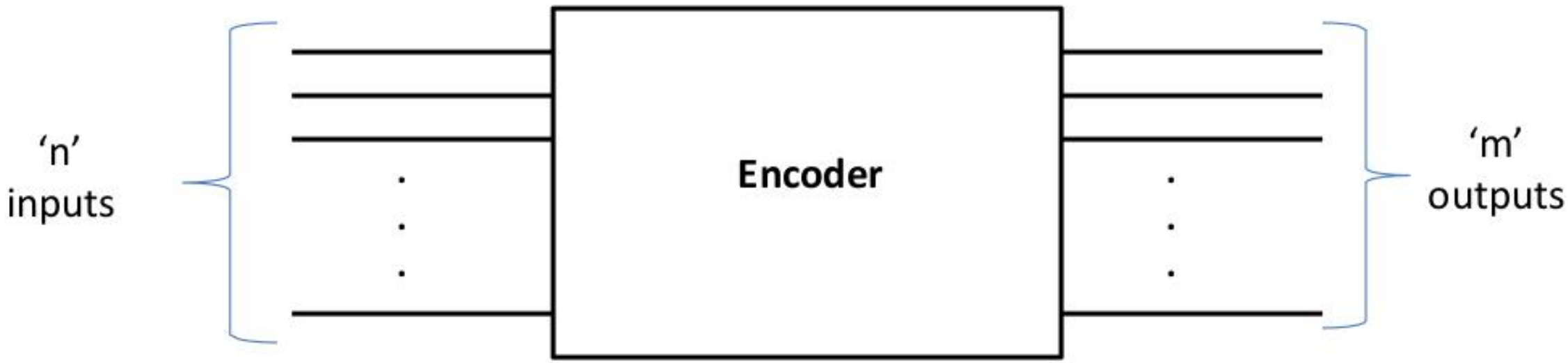
ENCODER

- Encoder is a combinational circuit which is designed to compress/reduce the bits.
- An encoder has **‘n’** number of input lines and **‘m’** number of output lines.
- An encoder produces an **m** bit binary code corresponding to the digital input number.
- The encoder accepts an n input digital word and converts it into m bit another digital word

ENCODER

- An encoder in digital logic is a device that converts multiple input signals into a smaller set of output signals. It assigns a unique code to each possible combination of inputs, enabling efficient data representation.

ENCODER



Types of Encoders

- ✓ Priority Encoder
- ✓ Decimal to BCD Encoder
- ✓ Octal to BCD Encoder
- ✓ Hexadecimal to Binary Encoder

4 Line to 2 Line Encoder

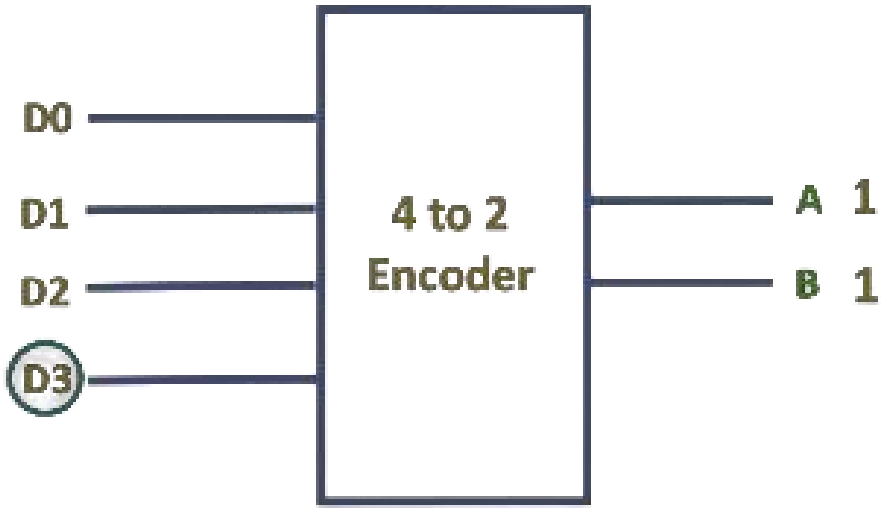
4 Line to 2 Line Encoder

A 4-to-2 encoder is a digital circuit that takes *four input lines* and produces a *two-bit binary output*.

It generates a binary code representing the active input line, providing a compact representation of the input state.

This type of encoder is commonly used in digital systems for tasks like address decoding and data multiplexing.

4 Line to 2 Line Encoder



Truth Table

D0	D1	D2	D3	A	B
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

4 Line to 2 Line Encoder

So , Based On the Truth Table, the o/p A is *HIGH* when the input D2 is *HIGH* or D3 is *HIGH*

That means, the logical expression, for output A is

$$A = D2 + D3$$

LikeWise, the output **B** is ***HIGH*** *when* ***D1*** *is* ***HIGH or*** *when* ***D3*** *is* ***HIGH***

That means, the logical expression, for output **B** is

$$B = D1 + D3$$

Truth Table

D0	D1	D2	D3	A	B
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

4 Line to 2 Line Encoder

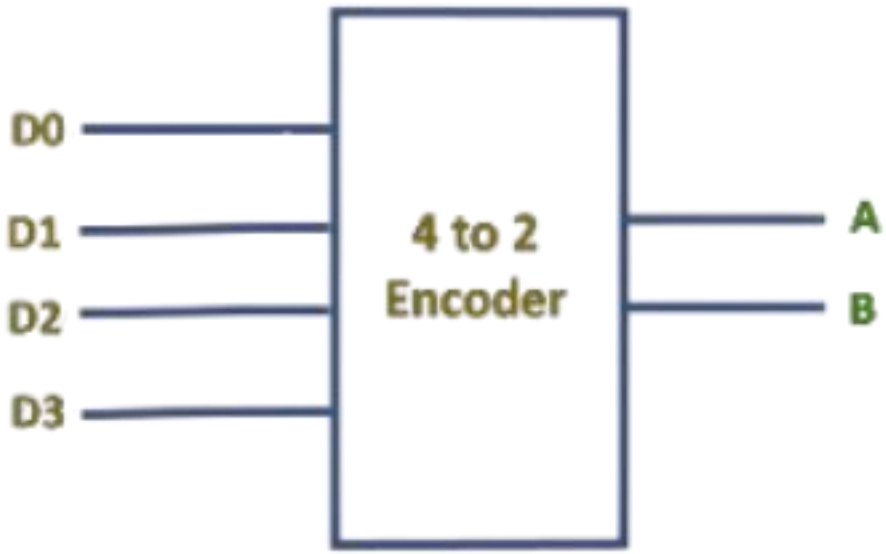
Truth Table

D0	D1	D2	D3	A	B
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

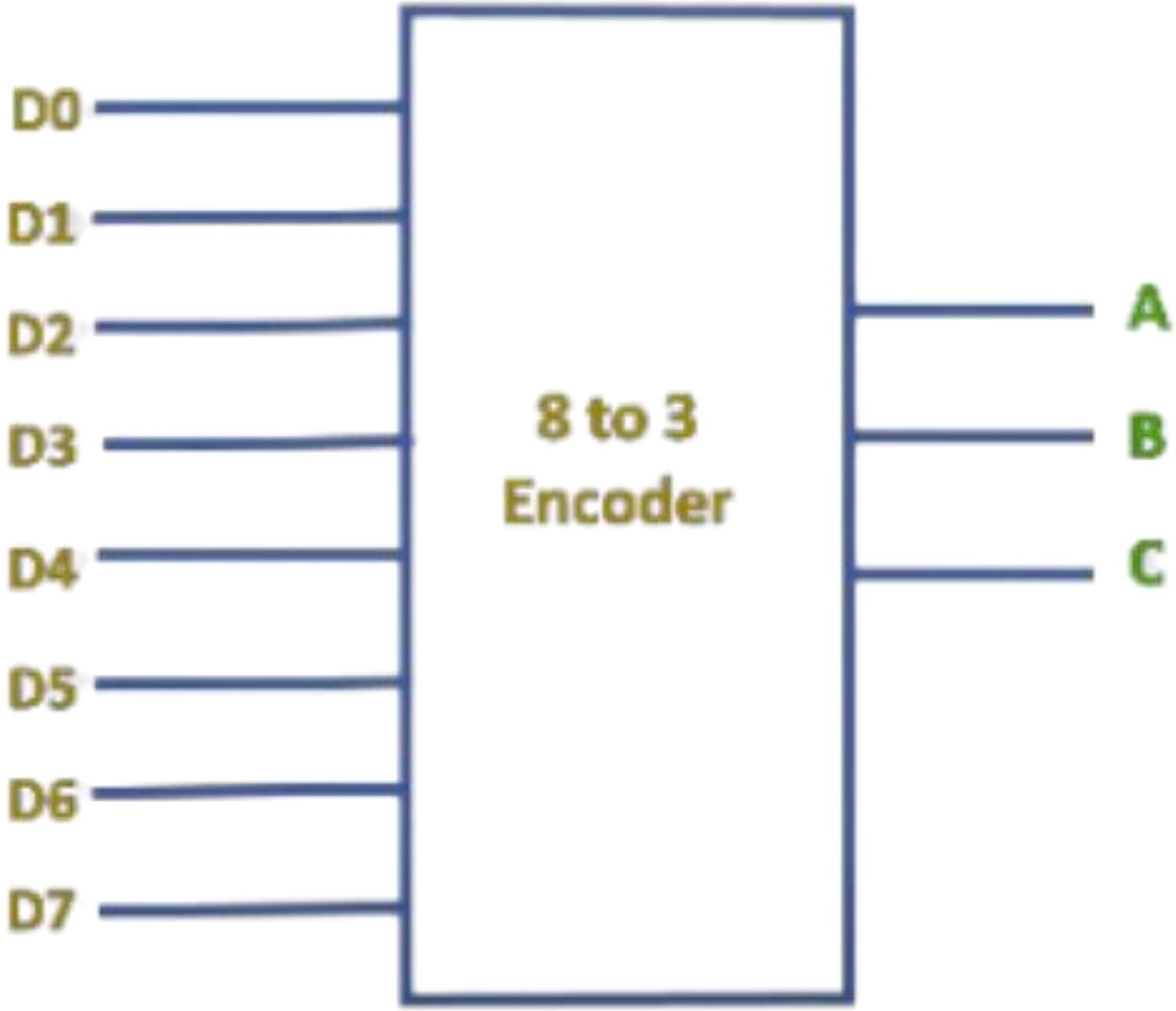
$$A = D2 + D3$$

$$B = D1 + D3$$

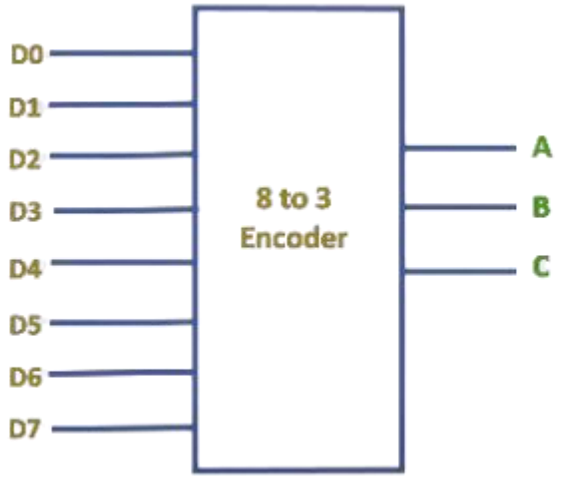
4 Line to 2 Line Encoder



8 Line to 3 Line Encoder



8 Line to 3 Line Encoder/Octal to Binary Encoder



Truth Table

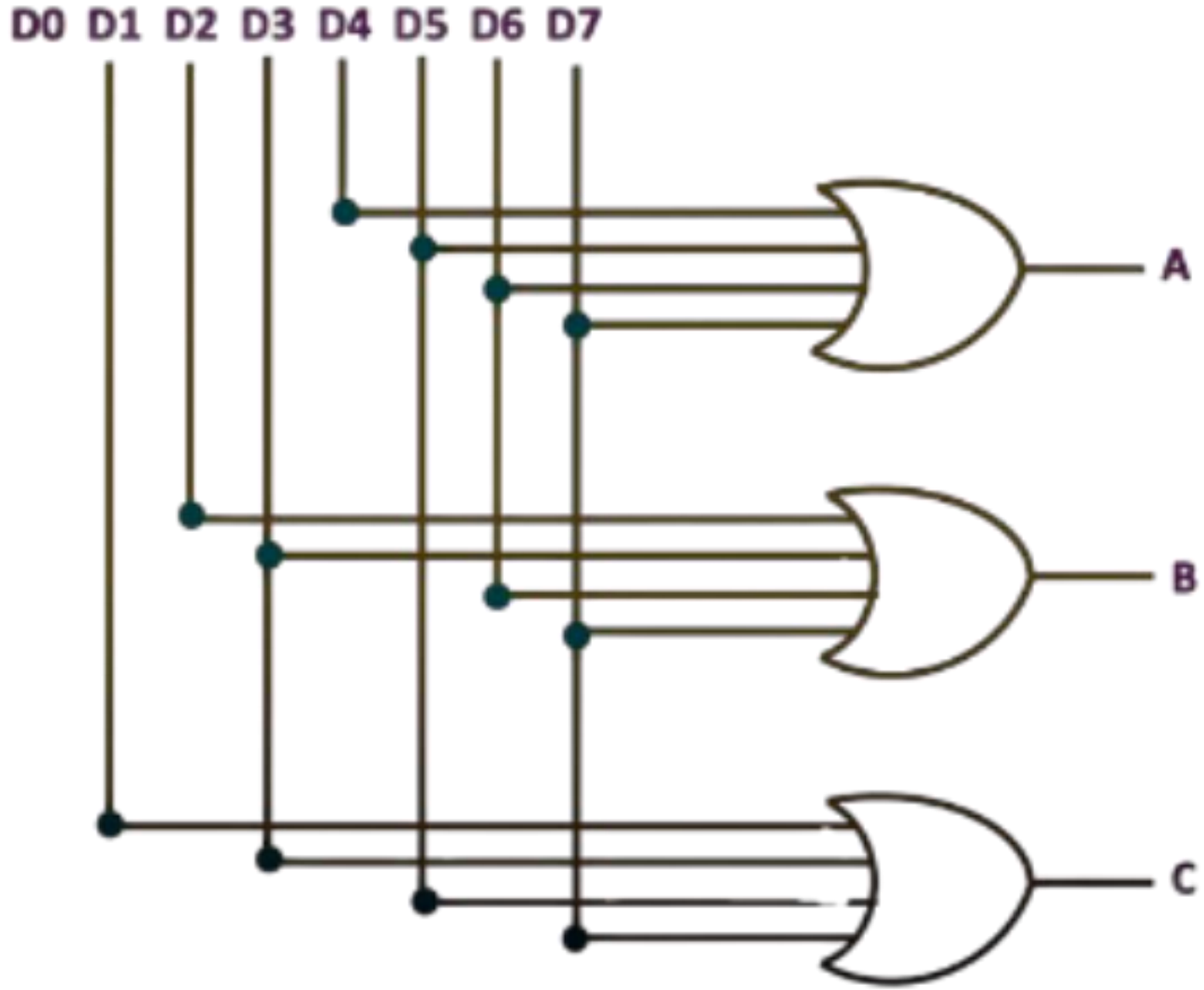
D0	D1	D2	D3	D4	D5	D6	D7	A	B	C
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

8 Line to 3 Line Encoder/Octal to Binary Encoder

A = D4 + D5 + D6 + D7

B = D2 + D3 + D6 + D7

C = D1 + D3 + D5 + D7



Assignments

Assignments

- ✓ Decimal to BCD Encoder
- ✓ Octal to BCD Encoder
- ✓ Hexadecimal to Binary Encoder

✓Decimal to BCD Encoder

Truth Table

D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	A	B	C	D
1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	0	0	0	1	1
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	1	0	1
0	0	0	0	0	0	1	0	0	0	0	1	1	0
0	0	0	0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	0	1	1	0	0	1

✓Decimal to BCD Encoder

Truth Table

D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	A	B	C	D
1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	0	0	0	1	1
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	1	0	1
0	0	0	0	0	0	1	0	0	0	0	1	1	0
0	0	0	0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	0	1	1	0	0	1

$A = D8 + D9$

$B = D4 + D5 + D6 + D7$

$C = D2 + D3 + D6 + D7$

$D = D1 + D3 + D5 + D7 + D9$

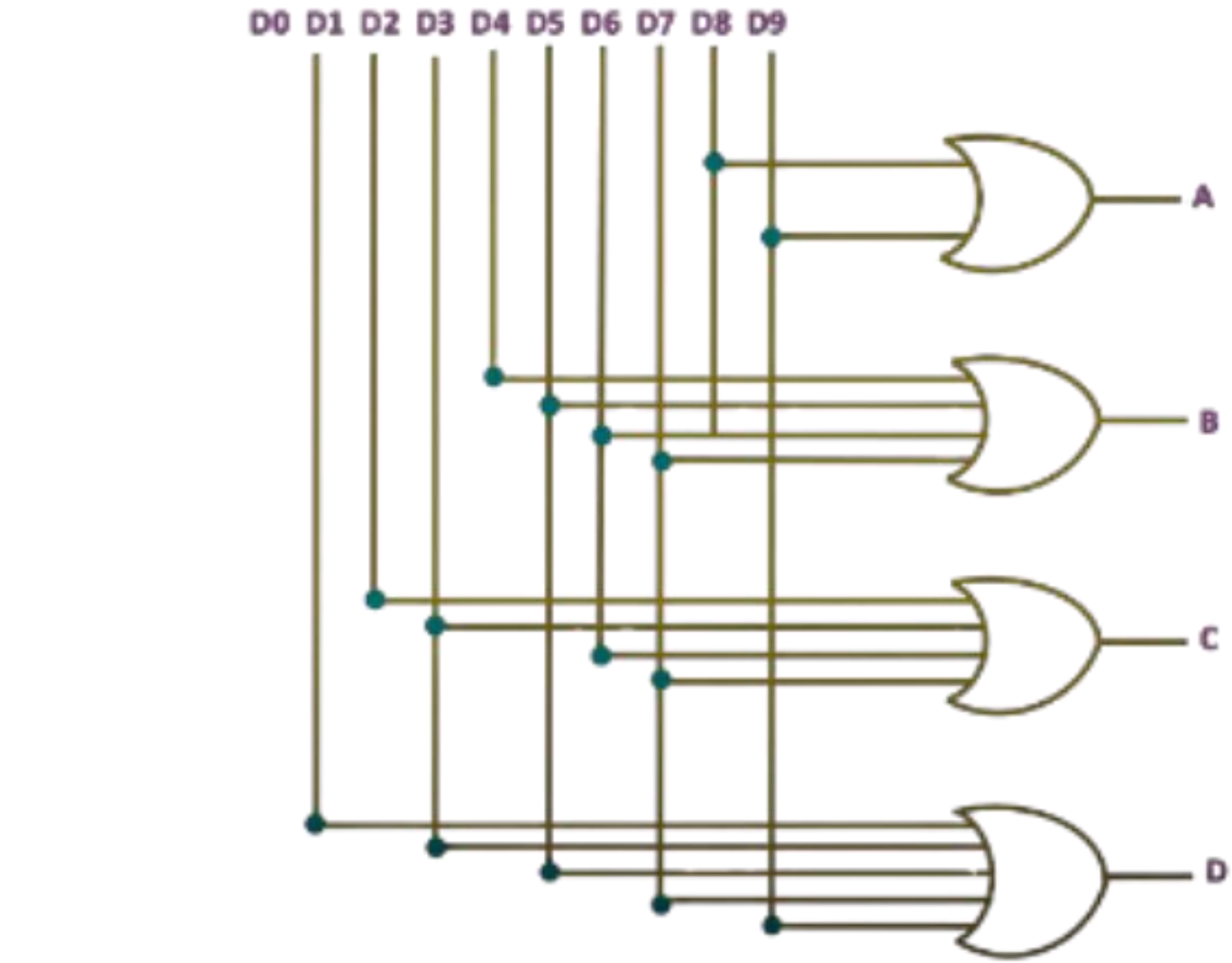
✓Decimal to BCD Encoder

A = D8 + D9

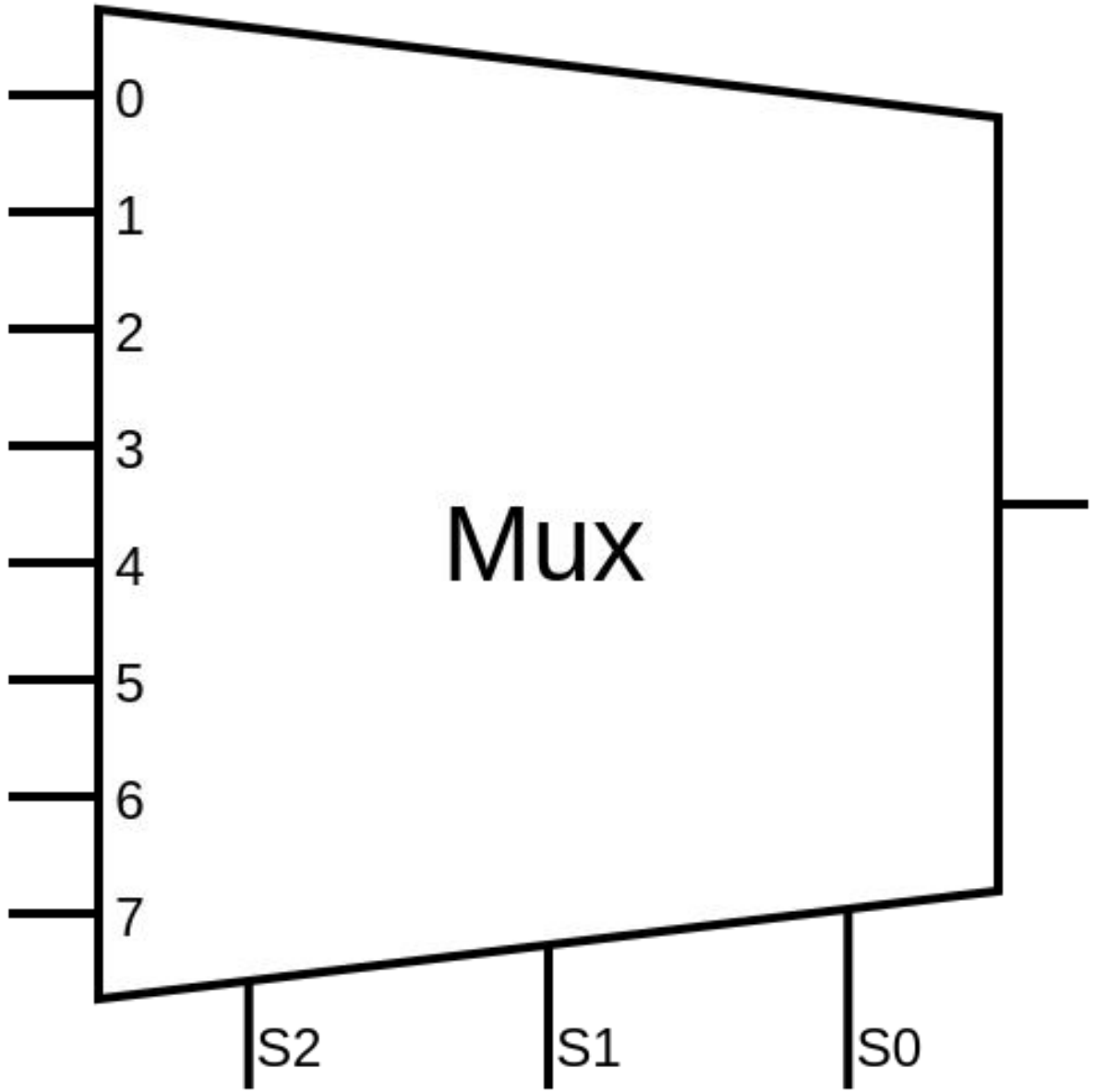
B = D4 + D5 + D6 + D7

C = D2 + D3 + D6 + D7

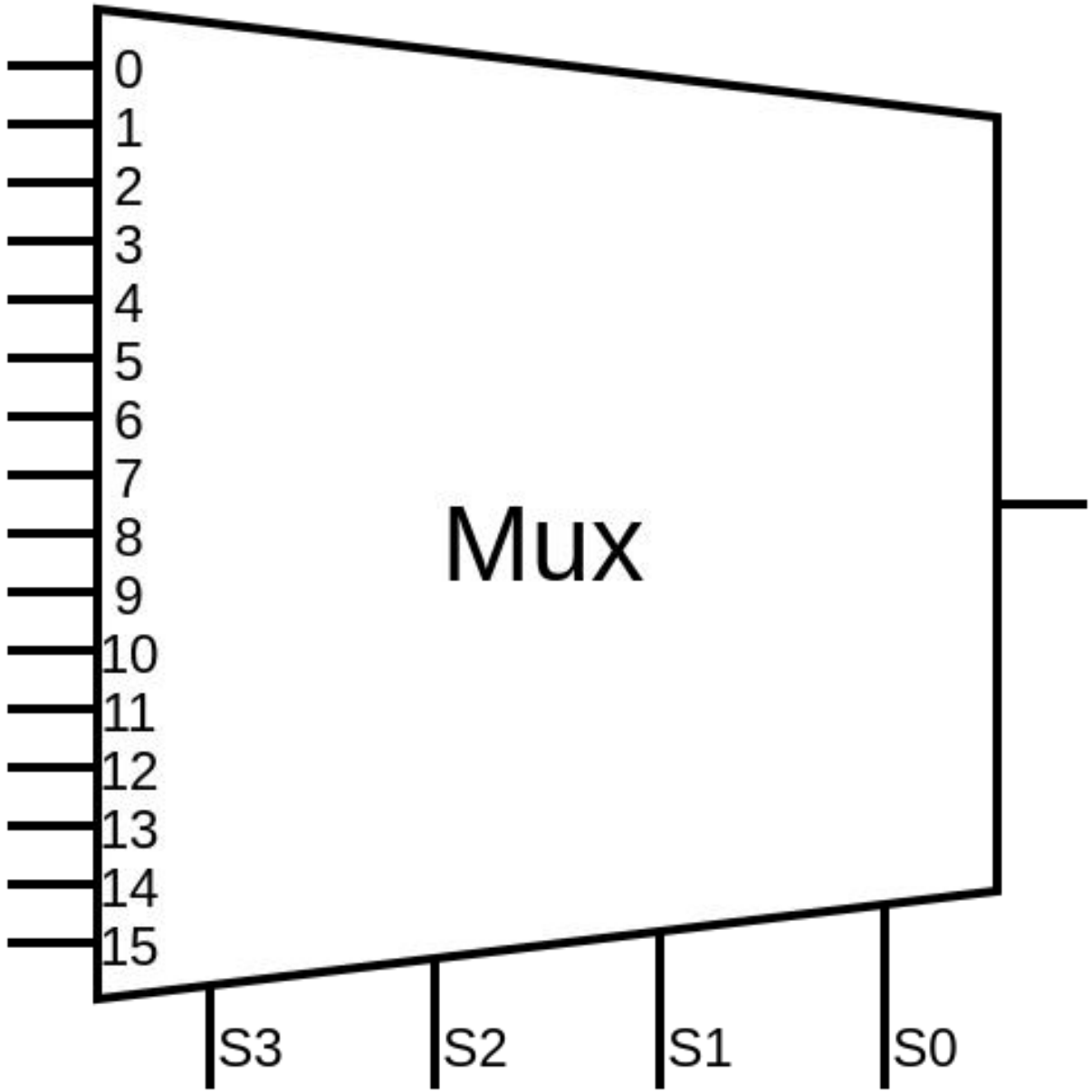
D = D1 + D3 + D5 + D7 + D9



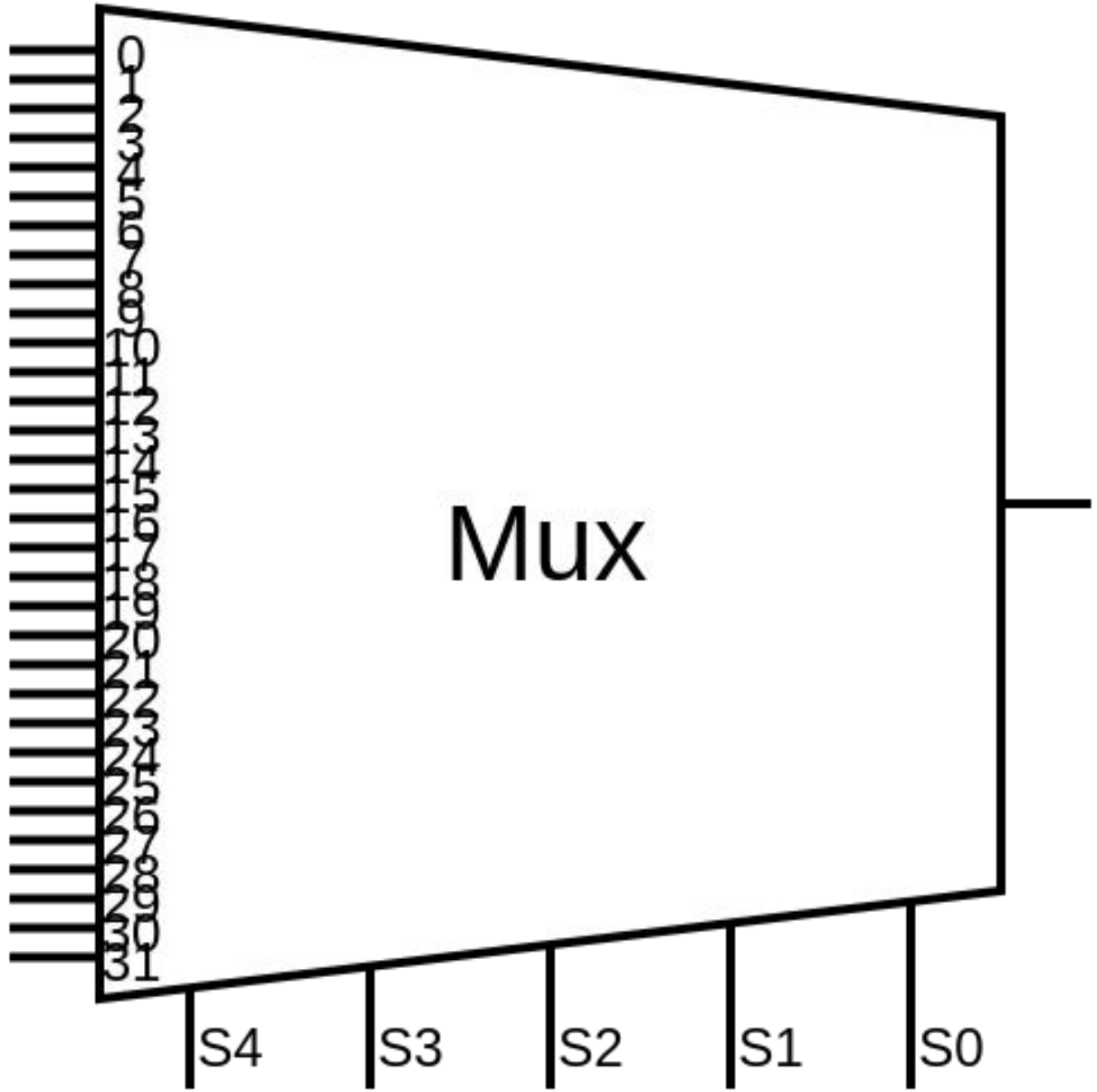
MultiPlexer



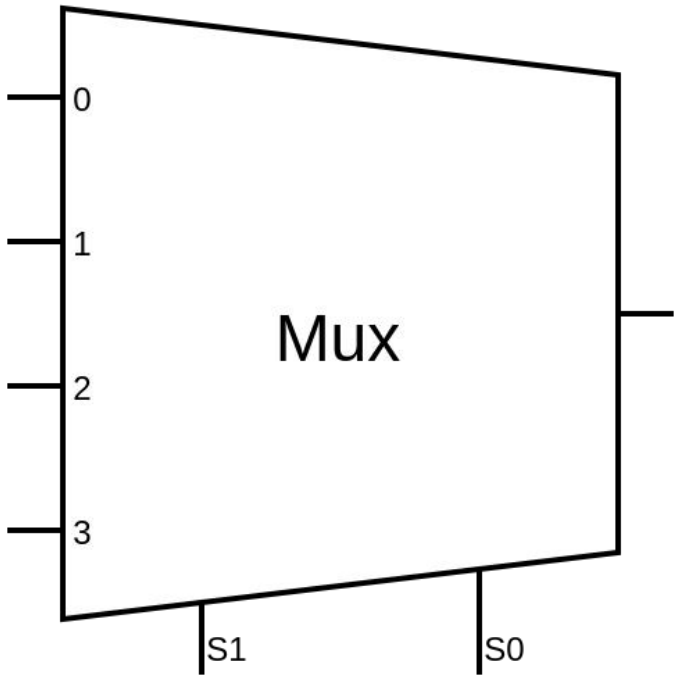
MultiPlexer



MultiPlexer



Multiplexer

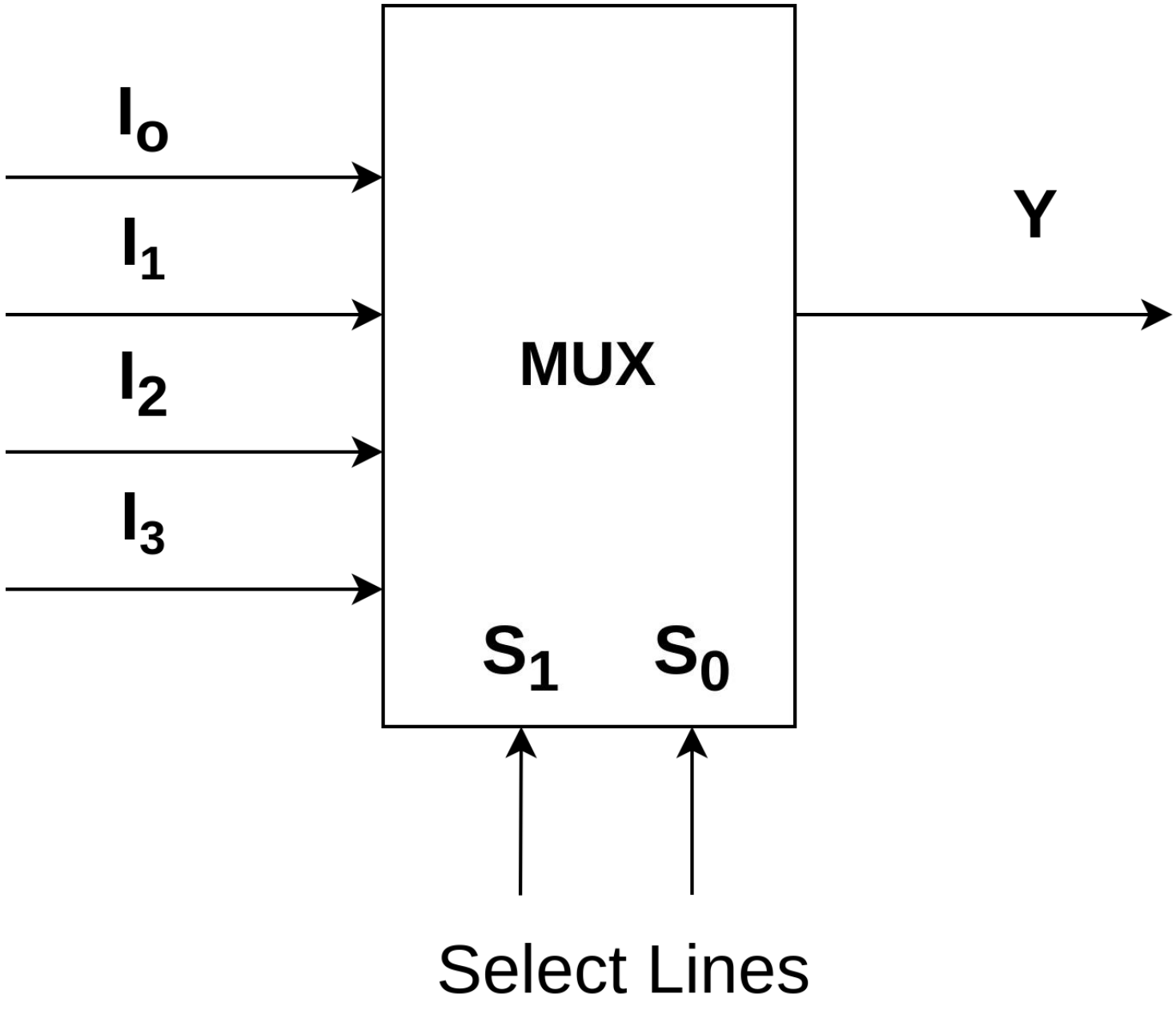


- Multiplexer is a combinational circuit which accepts data from *multiple input line* , but allows data to pass only through *single output line*, with the help of some extra input lines which is called as ***select lines*.**

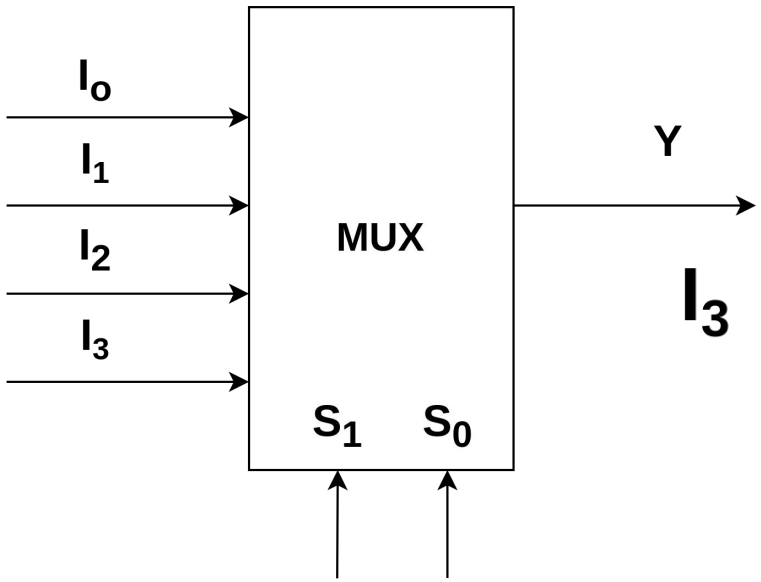
- *Select Lines* will **decide** in which condition which line will pass in output.

4 to 1 MUX

MultiPlexer



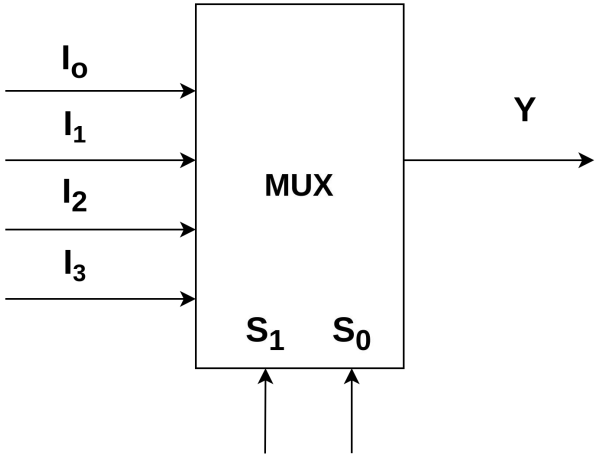
MultiPlexer



Select Lines		Output Line
S_1	S_0	Y
0	0	I_o
0	1	I_1
1	0	I_2
1	1	I_3

4 to 1 MUX

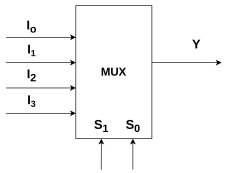
MultiPlexer



	Select Lines		Output Line
	S_1	S_0	Y
$\overline{S_1}\ \overline{S_0}$	0	0	I_o
$\overline{S_1}\ S_0$	0	1	I_1
$S_1\ \overline{S_0}$	1	0	I_2
$S_1\ S_0$	1	1	I_3

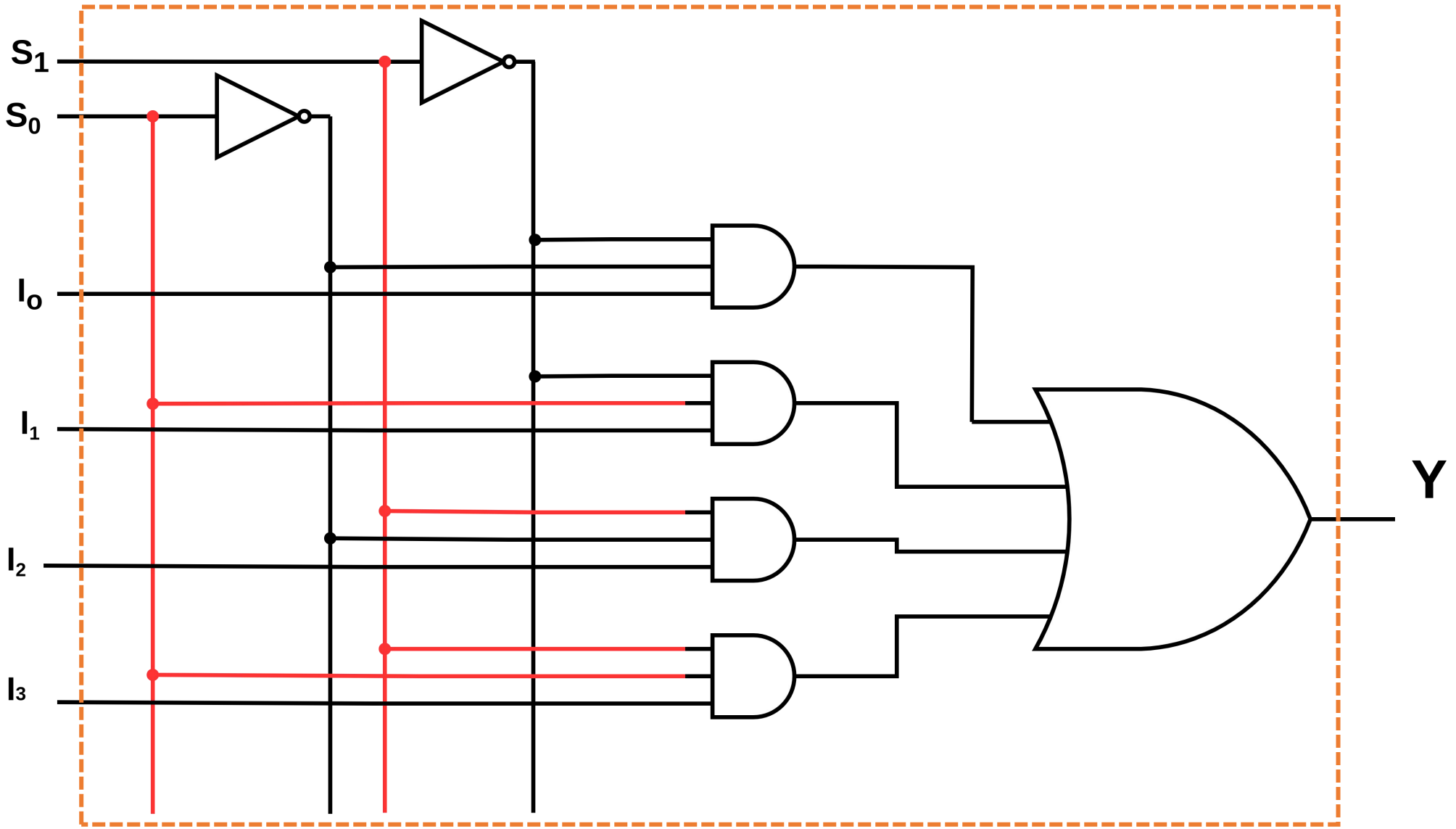
Output Expression:

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



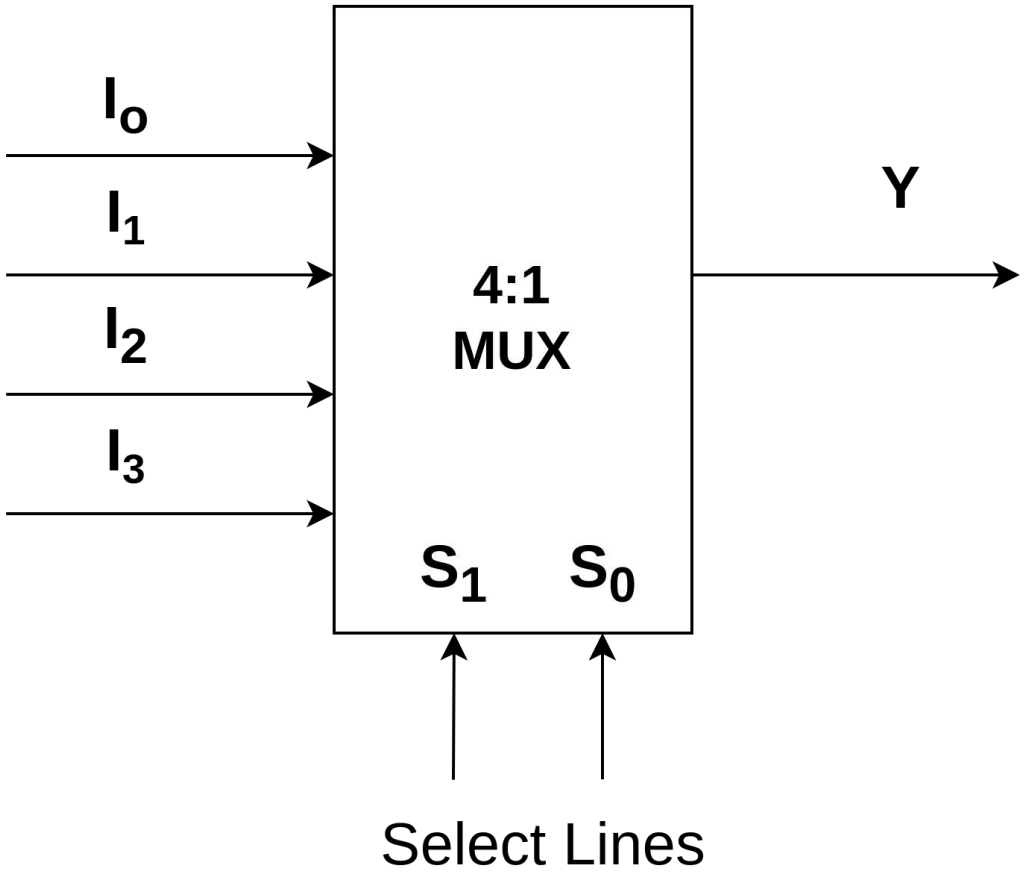
Select Lines		Output Line
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

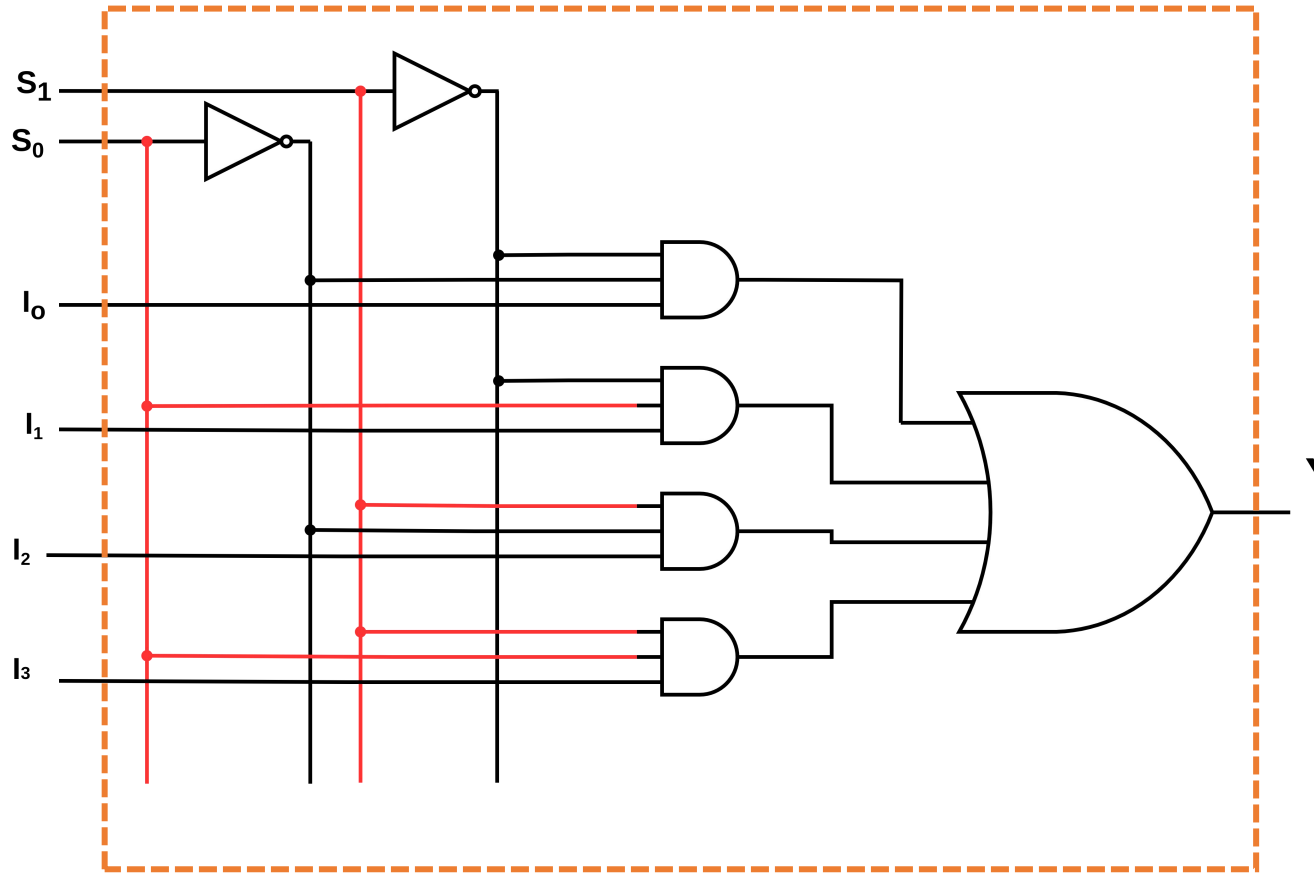
$$Y = \overline{S_1}\,\overline{S_0} I_0 + \overline{S_1} S_0 I_1 + S_1 \overline{S_0} I_2 + S_1 S_0 I_3$$



Select Lines		Output Line
S_1	S_0	Y
0	0	I_o
0	1	I_1
1	0	I_2
1	1	I_3

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

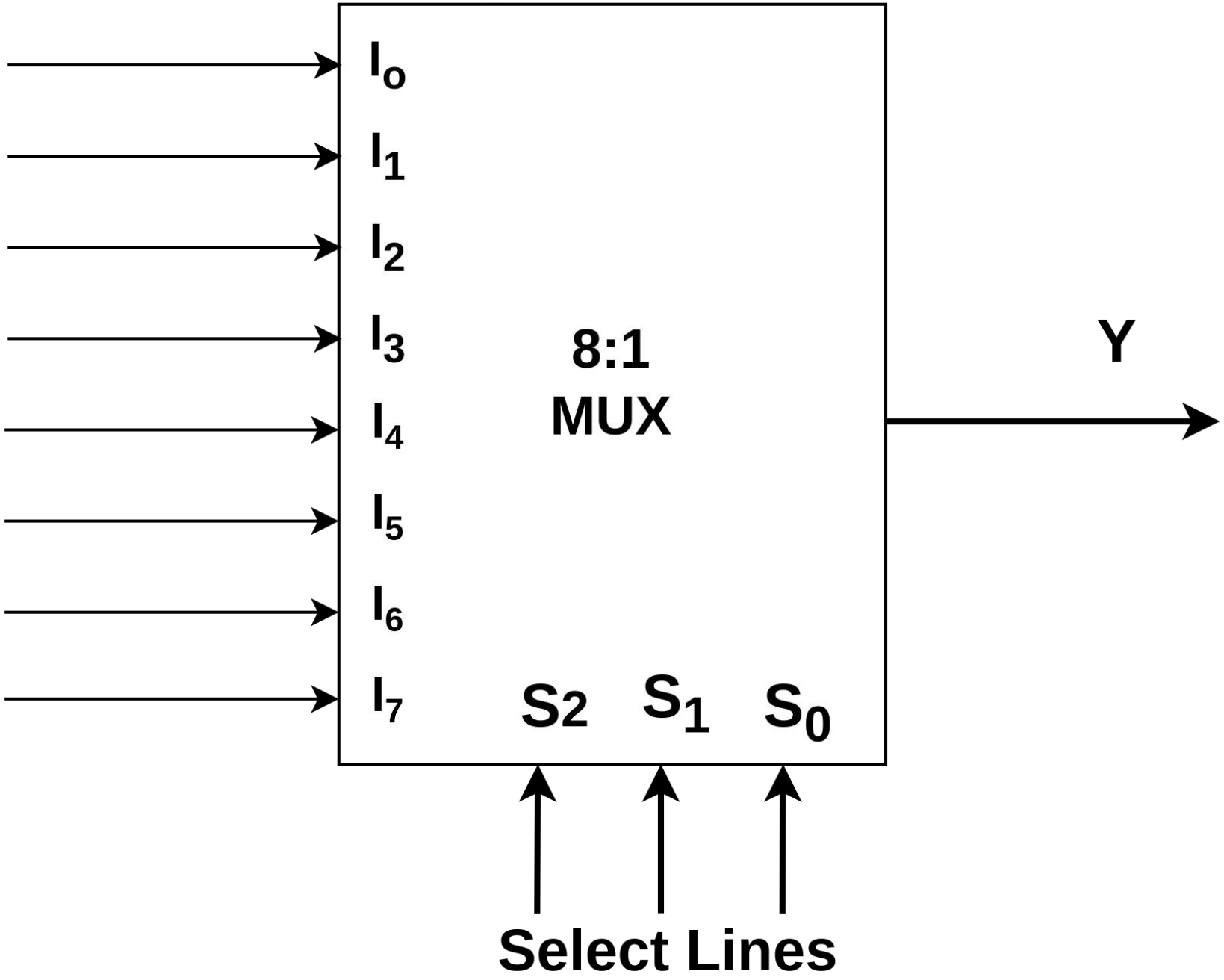




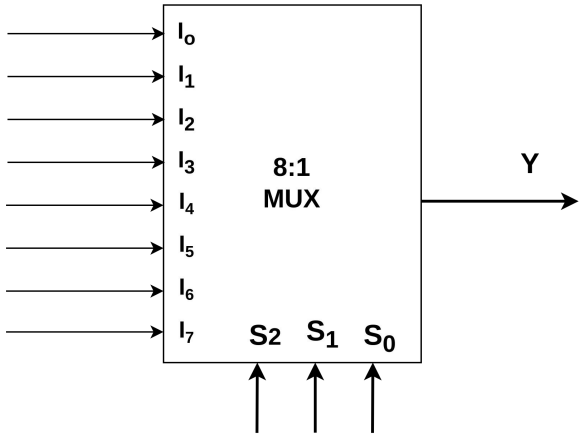
8 to 1 MUX

8 to 1 MUX

MultiPlexer

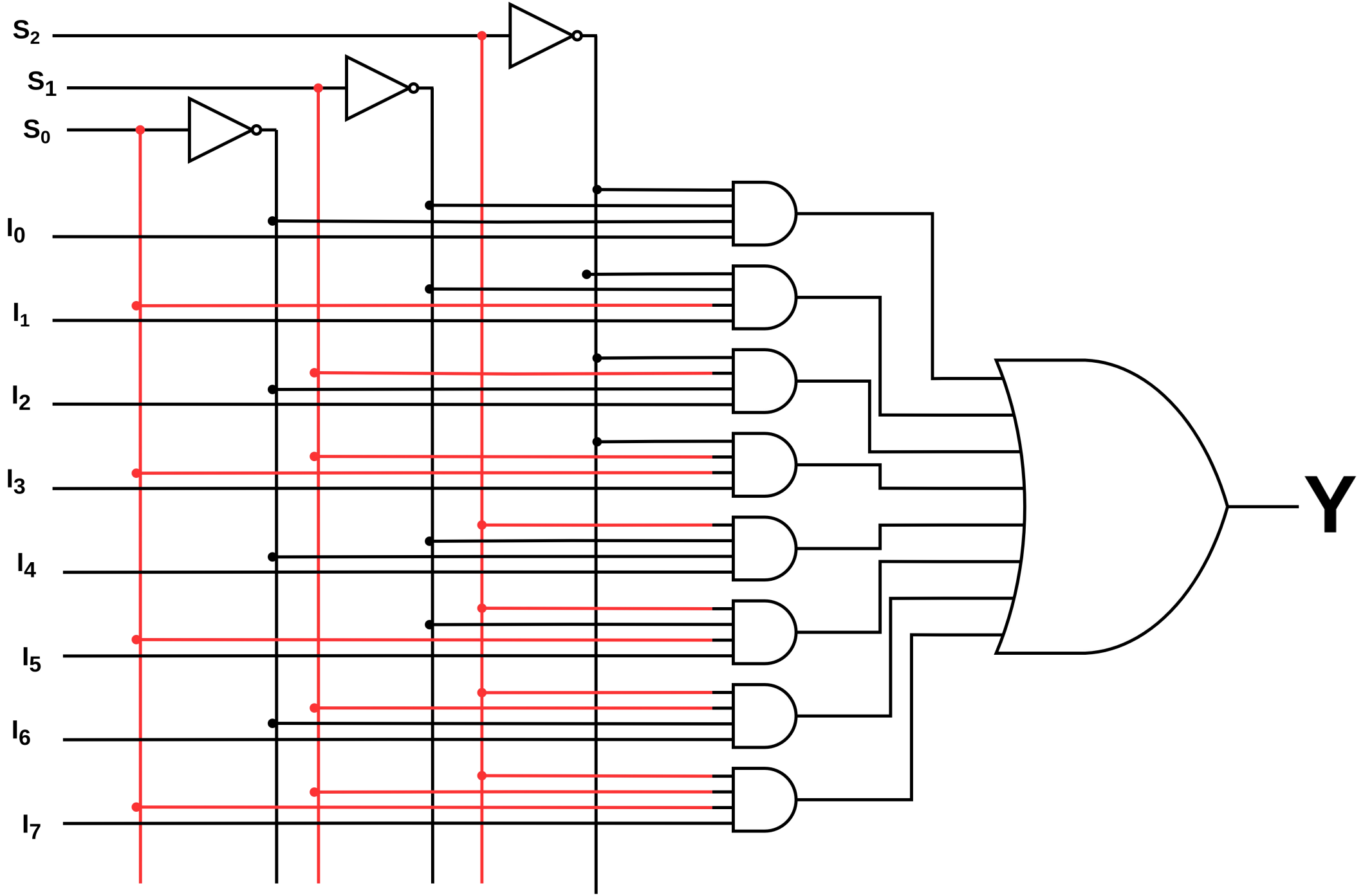


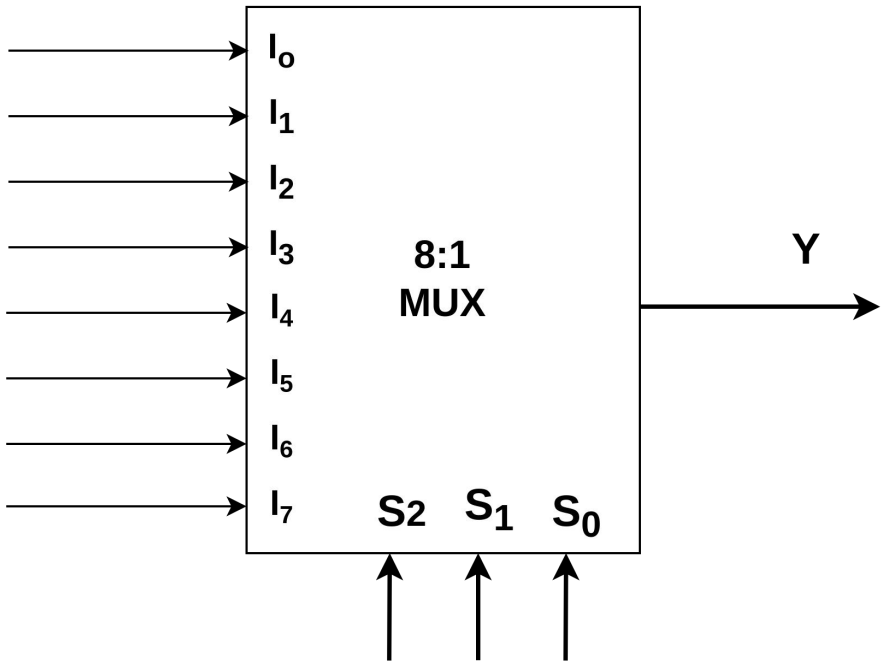
8 to 1 MUX

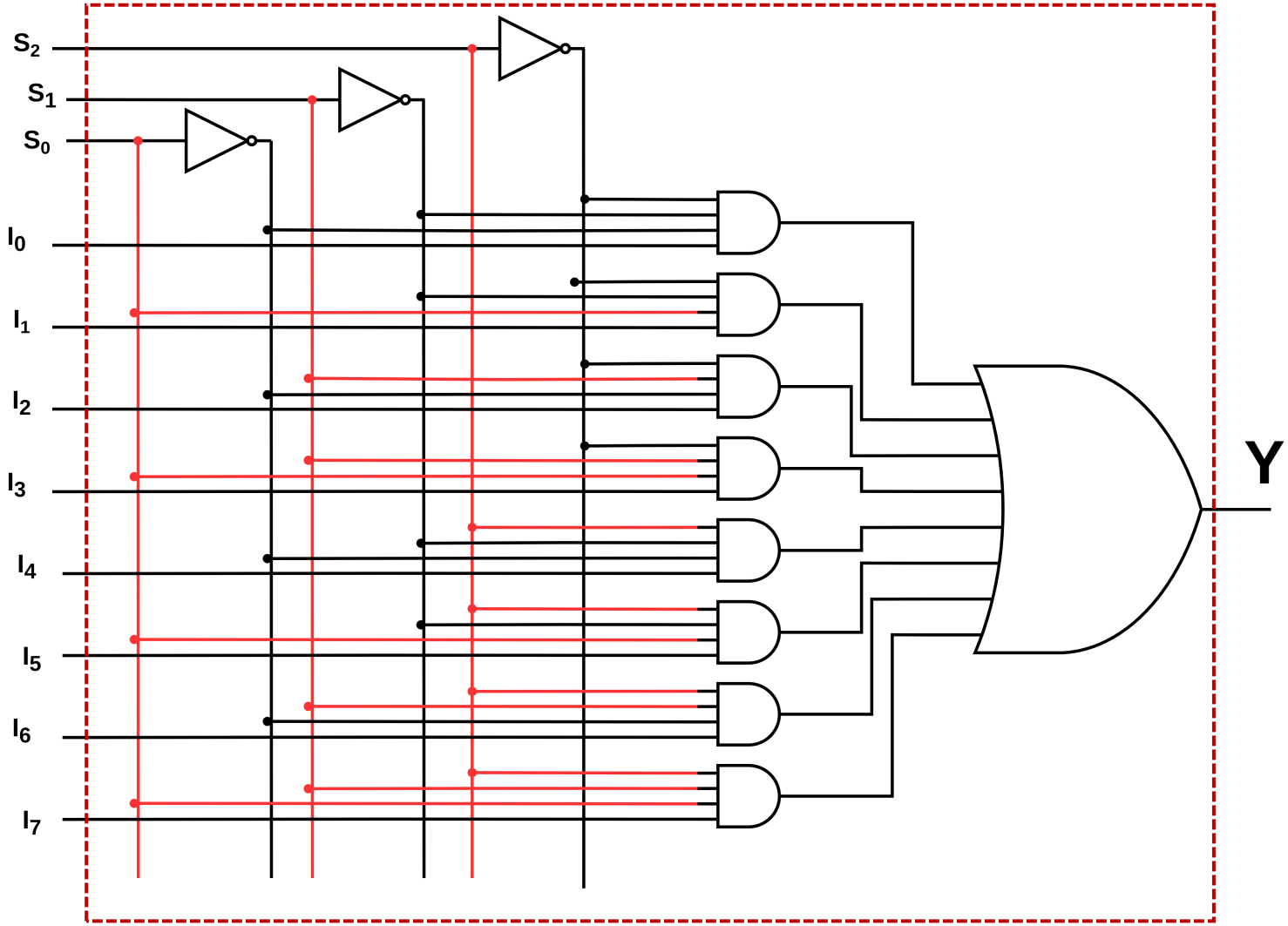


Select Lines			Output Line
S_2	S_1	S_0	Y
0	0	0	I_o
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

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


S2
S1
S0
I0
I1
I2
I3
I4
I5
I6
I7
Y


I0
I1
I2
I3
I4
I5
I6
I7
8:1
MUX
S2
S1
S0
Y

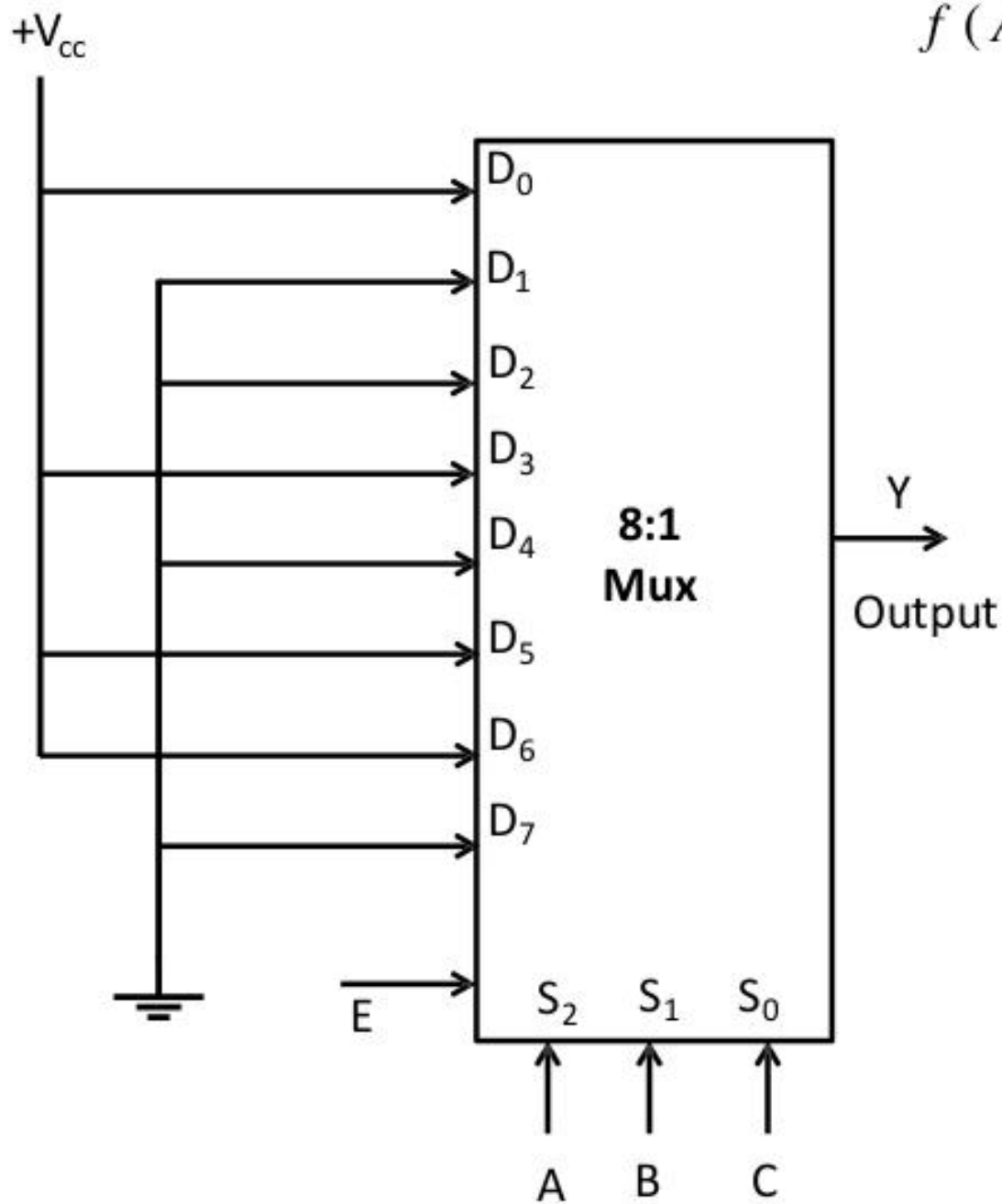

S2
S1
S0
I0
I1
I2
I3
I4
I5
I6
I7
Y

Implement following Boolean expression using multiplexer

$$f(A, B, C) = \sum m(0, 3, 5, 6)$$

- ✓ Since there are three variables, therefore a multiplexer with three select input is required i.e. 8:1 multiplexer is required
- ✓ The 8:1 multiplexer is configured as below to implement given Boolean expression

$$f(A, B, C) = \sum m(0, 3, 5, 6)$$

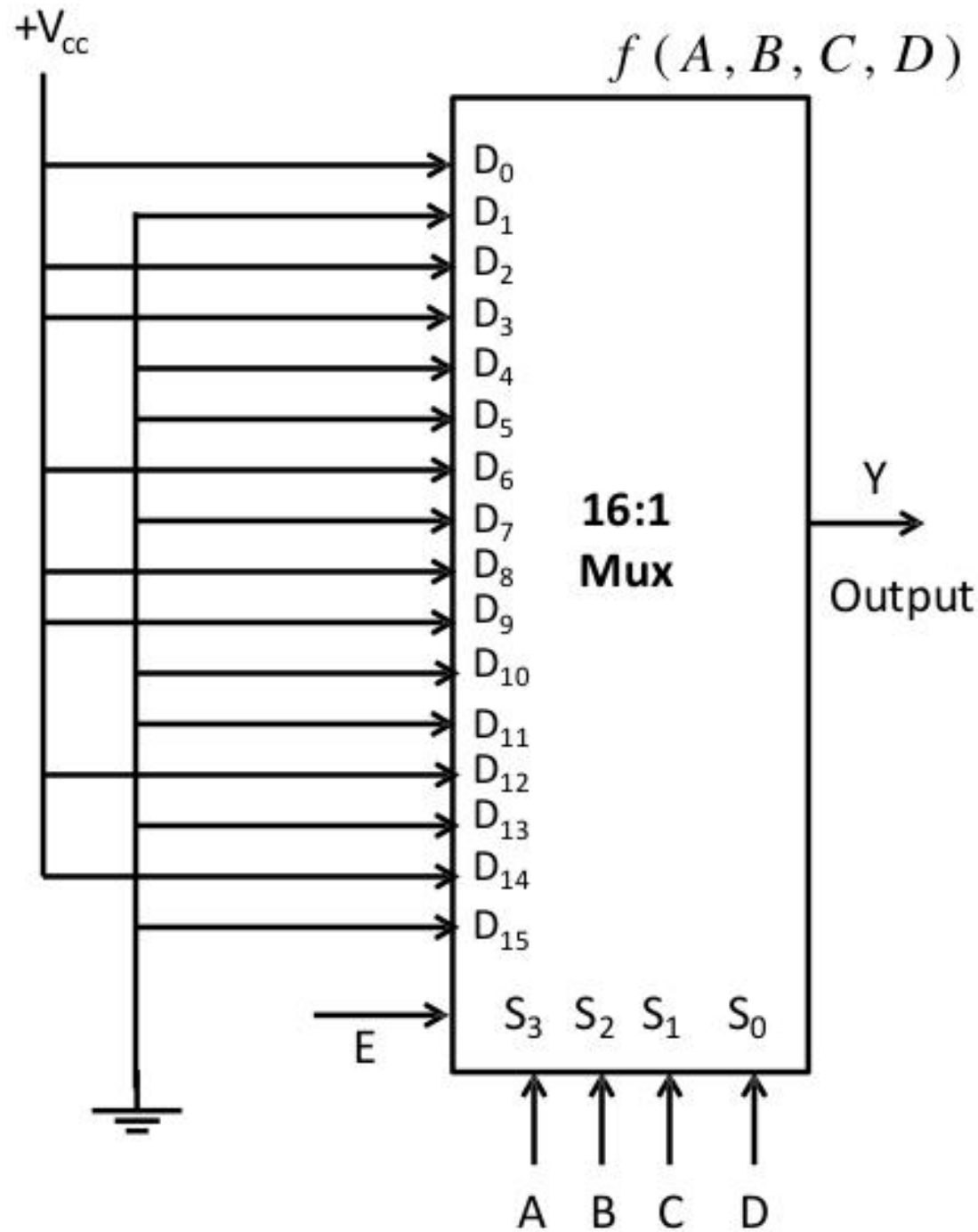


Implement following Boolean expression using multiplexer

$$f(A, B, C, D) = \sum m(0, 2, 3, 6, 8, 9, 12, 14)$$

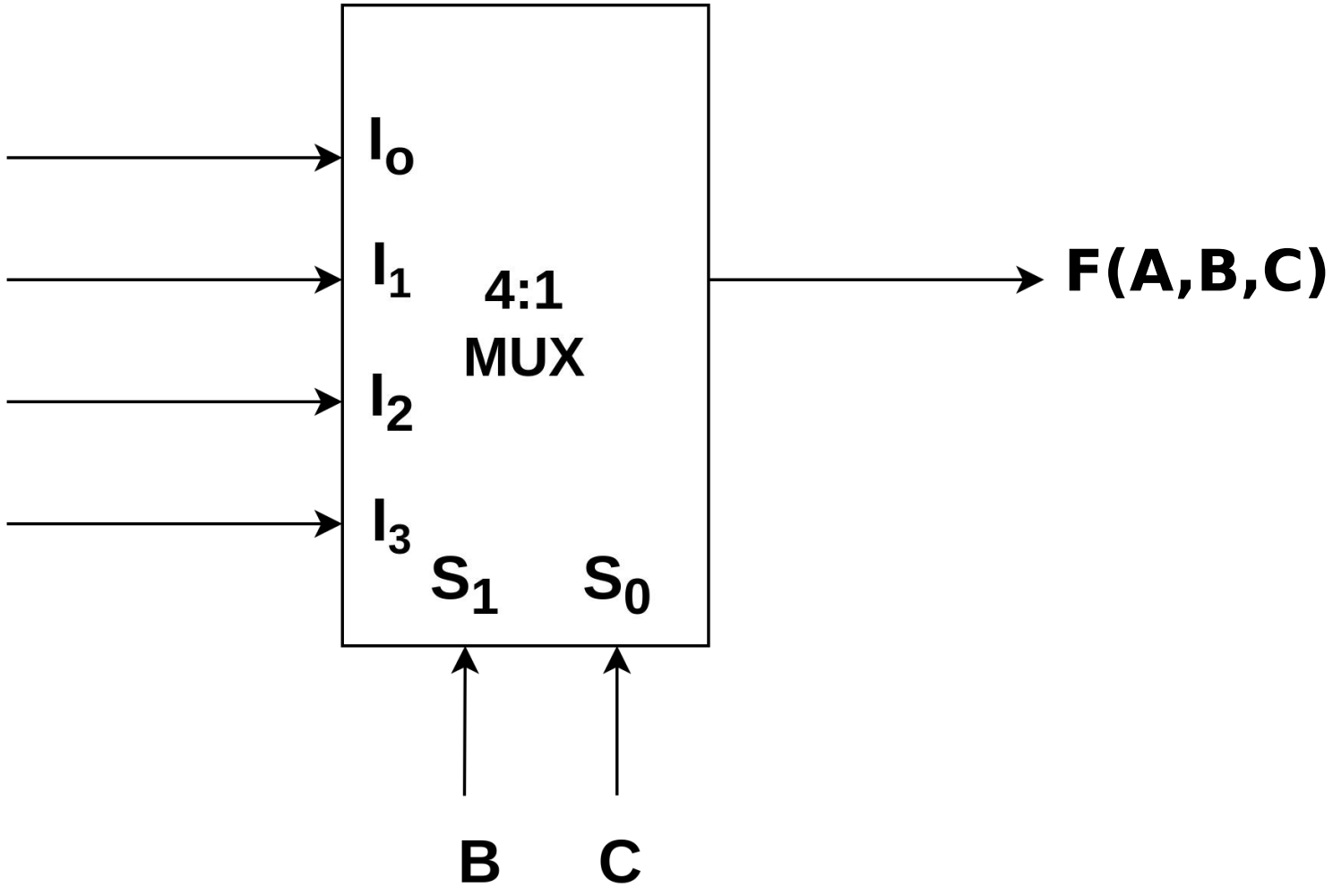
- ✓ Since there are four variables, therefore a multiplexer with four select input is required i.e. 16:1 multiplexer is required
- ✓ The 16:1 multiplexer is configured as below to implement given Boolean expression

$$f(A, B, C, D) = \sum m(0, 2, 3, 6, 8, 9, 12, 14)$$

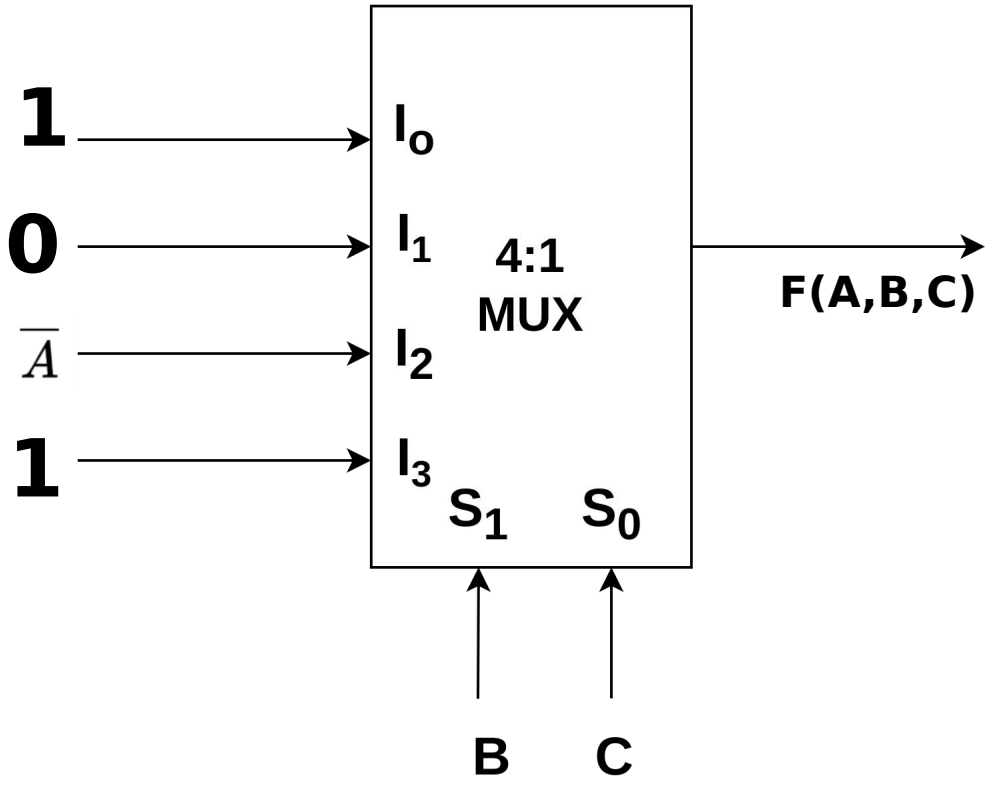


Implement the given boolean expression using 4 X 1 MUX :$F(A,B,C) = \Sigma m(0,2,3,4,7)$

Implement the given boolean expression using 4 X 1 MUX :F(A,B,C) = $\Sigma m(0,2,3,4,7)$



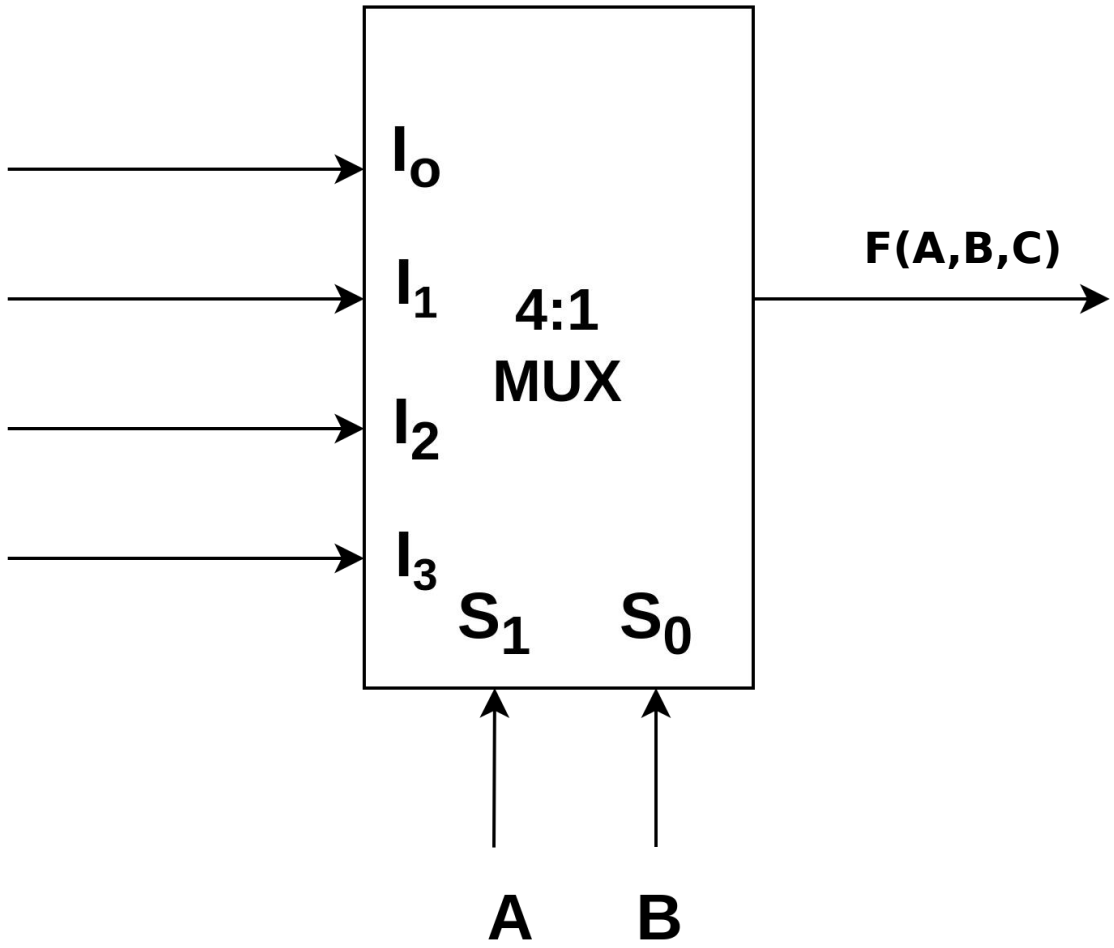
Implement the given boolean expression using 4 X 1 MUX :F(A,B,C) = $\Sigma m(0,2,3,4,7)$



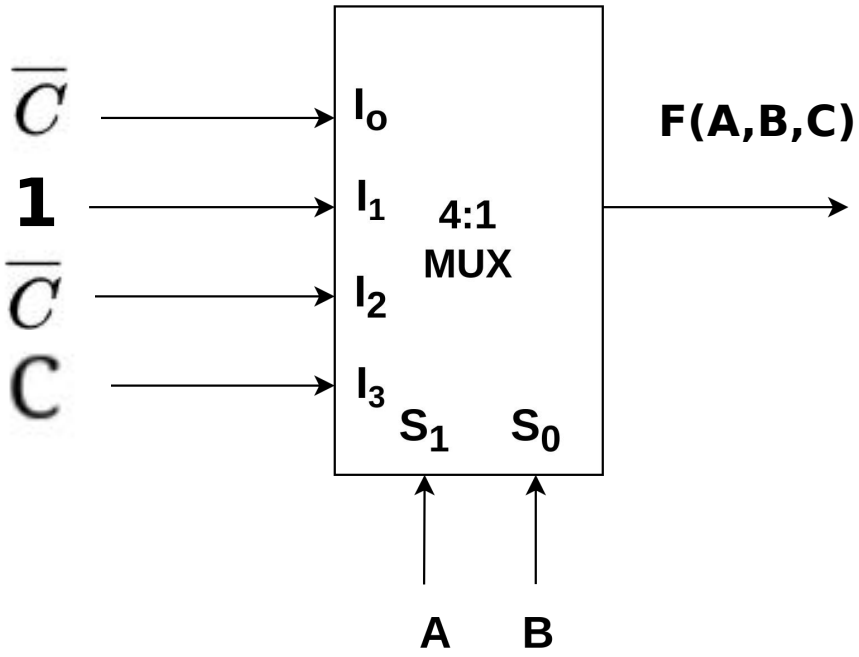
	00 I_o	01 I_1	10 I_2	11 I_3
$\overline{A}$	000 0	001 1	010 2	011 3
A	100 4	101 5	110 6	111 7
	$\overline{A}$+ A	0	$\overline{A}$	$\overline{A}$+ A
	1			1

Implement the given boolean expression using 4 X 1 MUX :$F(A,B,C) = \Sigma m(0,2,3,4,7)$

Use A and B as Selection Lines.



Implement the given boolean expression using 4 X 1 MUX :F(A,B,C) = $\Sigma m(0,2,3,4,7)$
Use A and B as Selection Lines.



	00 I_0	01 I_1	10 I_2	11 I_3
$\overline{C}$	000 (0)	010 (2)	100 (4)	110 (6)
C	001 (1)	011 (3)	101 (5)	111 (7)
	$\overline{C}$	1	$\overline{C}$	C

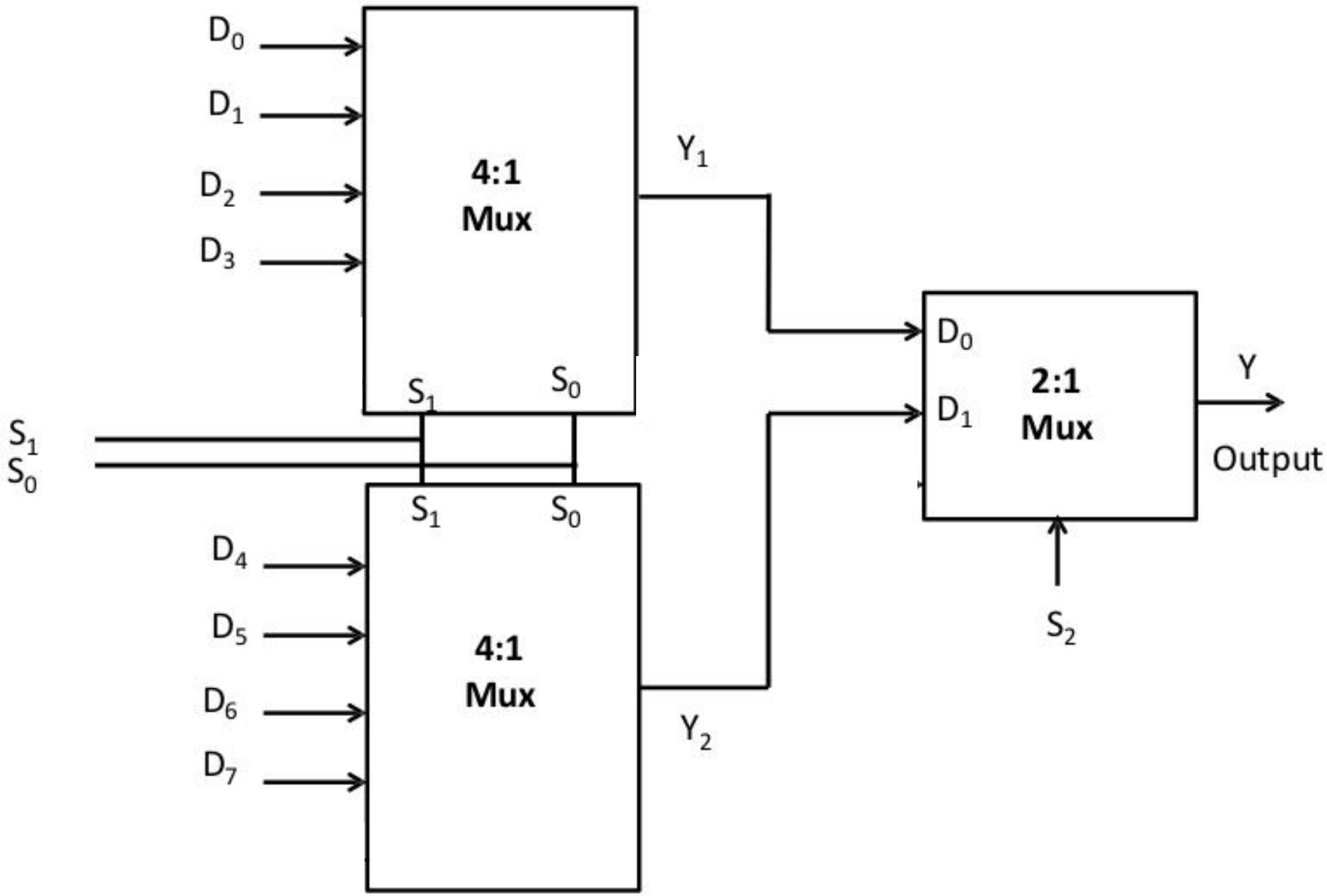
Mux Tree

- ✓ The multiplexers having more number of inputs can be obtained by cascading two or more multiplexers with less number of inputs. This is called as Multiplexer Tree.

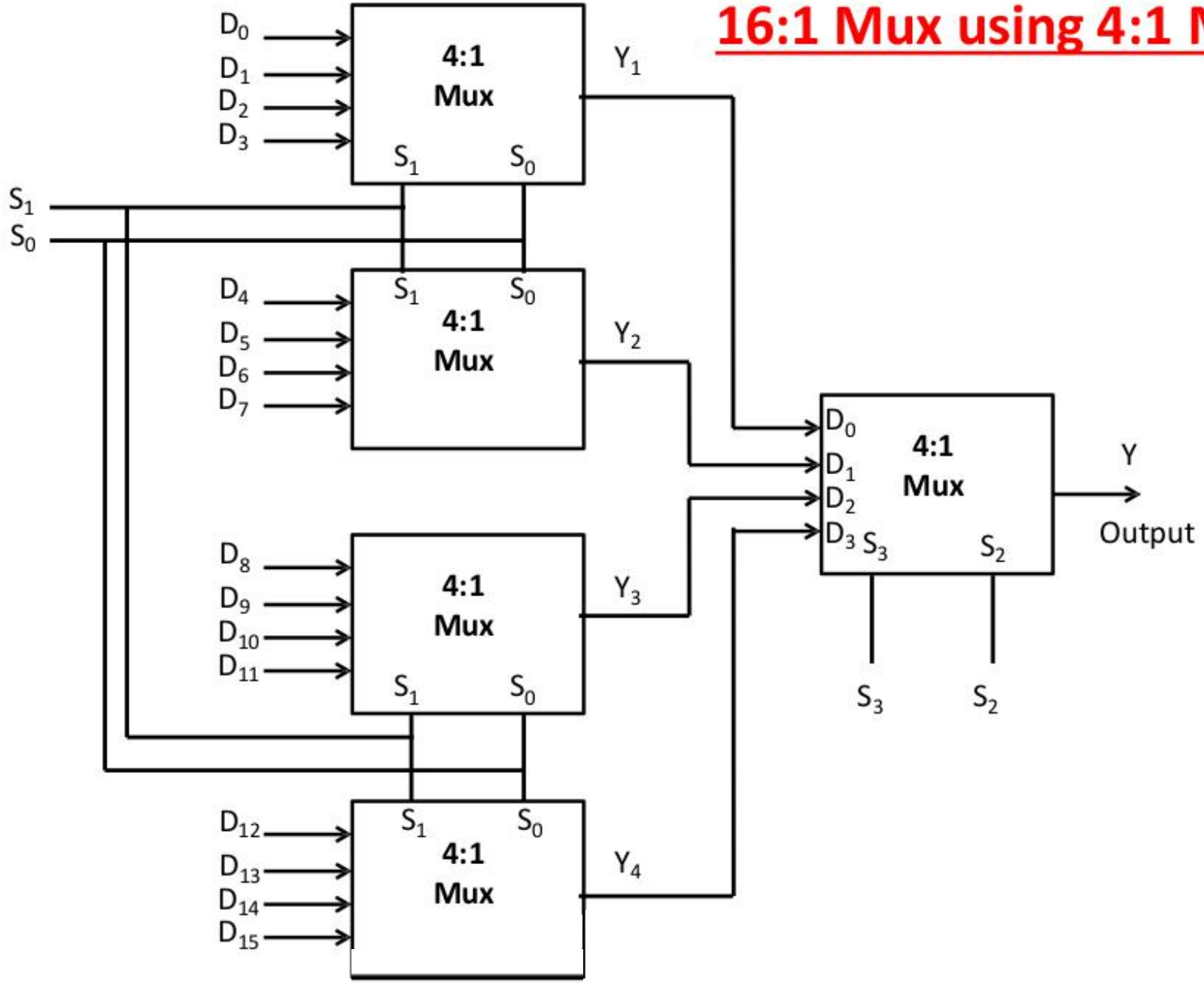
- ✓ For example, 32:1 mux can be realized using two 16:1 mux and one 2:1 mux.

8:1 Multiplexer using 4:1 Multiplexer

8:1 Multiplexer using 4:1 Multiplexer



16:1 Mux using 4:1 Mux



DeMultiplexer

DeMultiplexer

- ✓A de-multiplexer performs the reverse operation of a multiplexer i.e. it receives one input and distributes it over several outputs.
- ✓At a time only one output line is selected by the select lines and the input is transmitted to the selected output line.
- ✓It has only one input line, n number of output lines and m number of select lines.

DeMultiplexer

Block Diagram of De-multiplexer

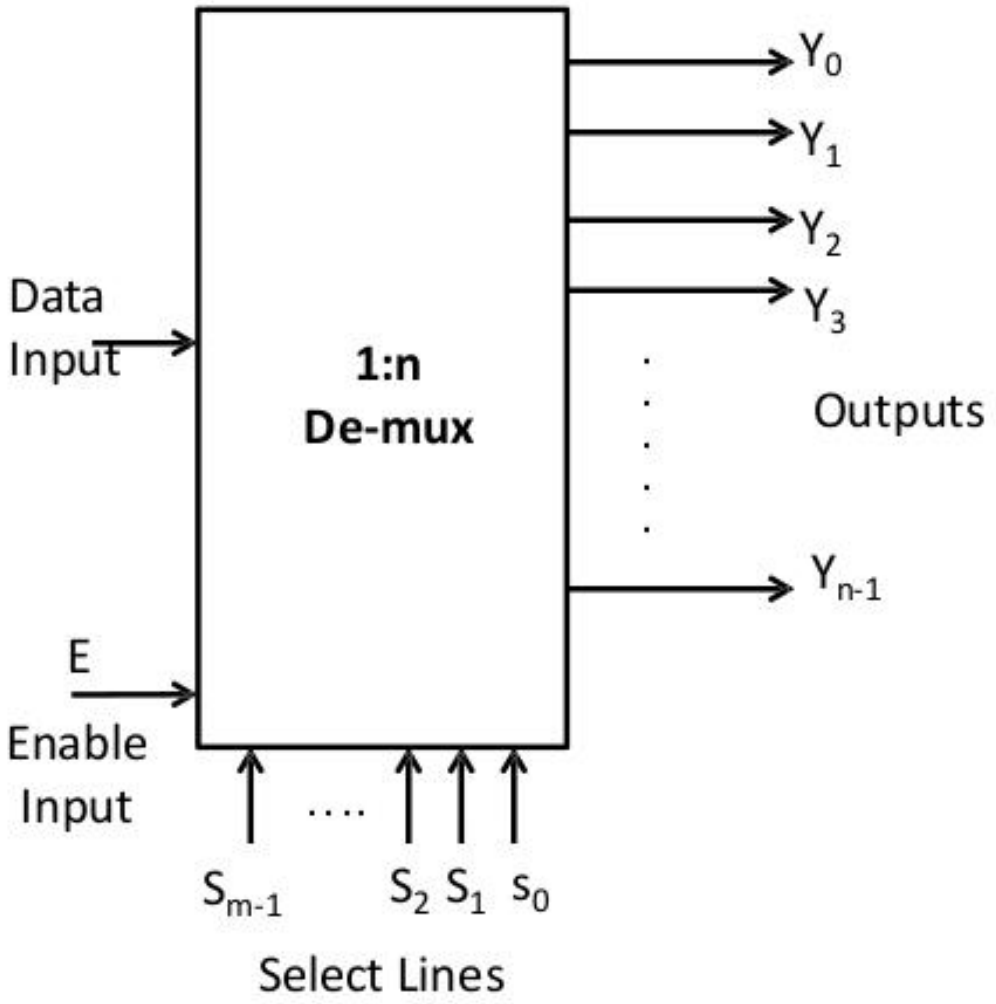


Fig. General Block Diagram

DeMultiplexer

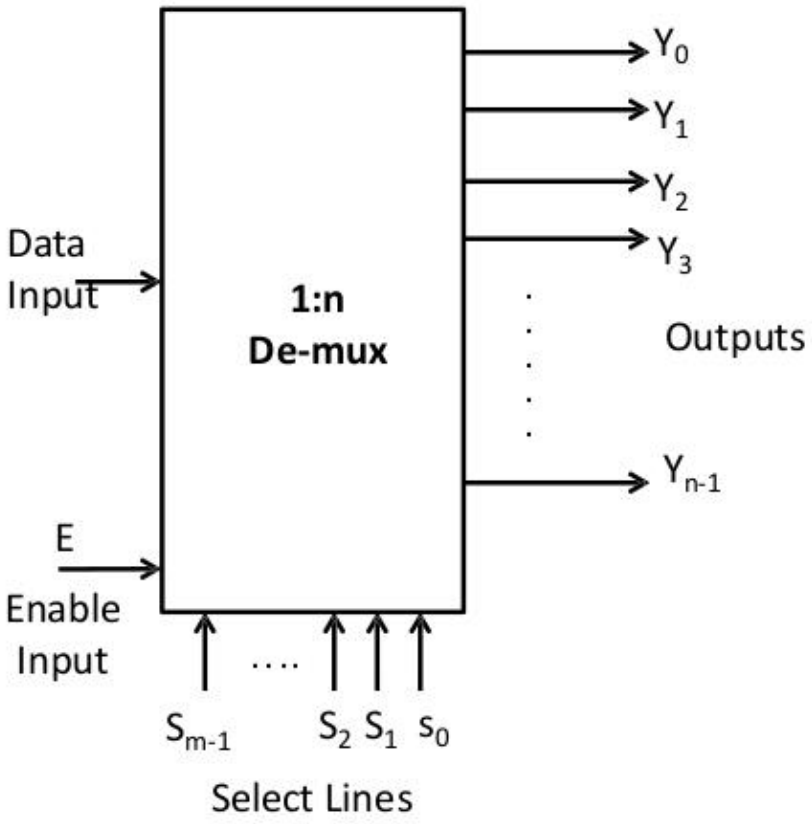


Fig. General Block Diagram

Relation between Data Output Lines & Select Lines

- ✓ In general de-multiplexer contains , n output lines, one input line and m select lines.
- ✓ To select n outputs we need m select lines such that $n=2^m$.

DeMultiplexer

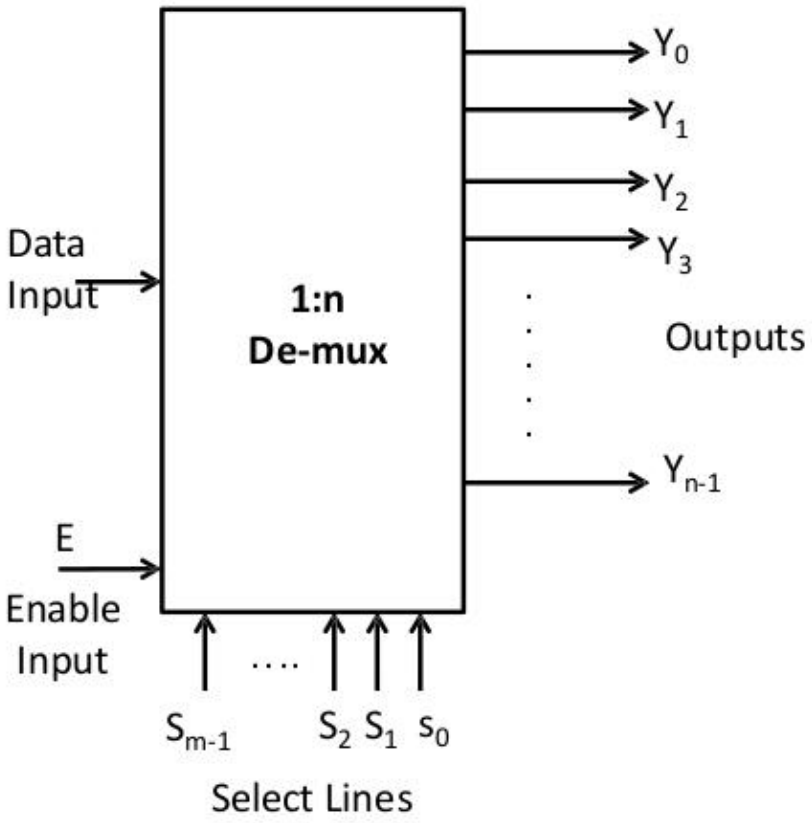


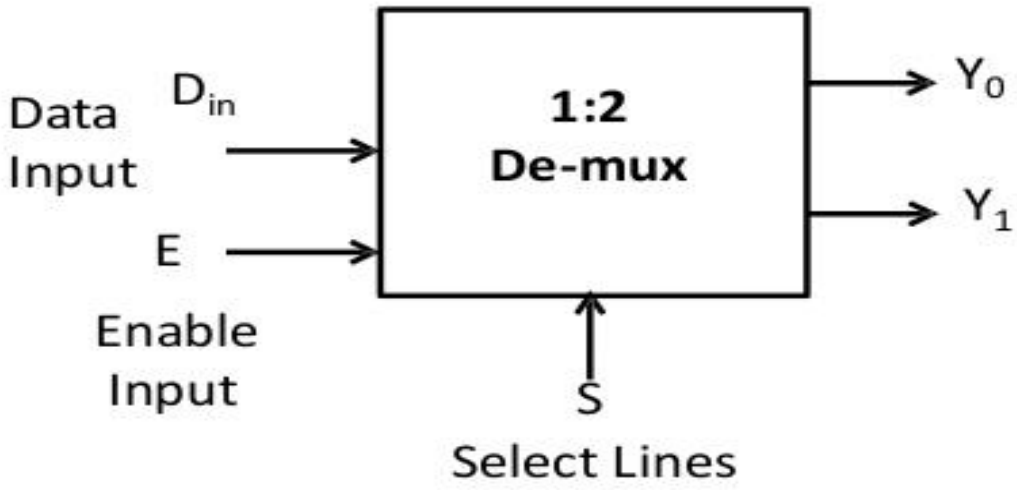
Fig. General Block Diagram

Types of De-multiplexers

- ✓1:2 De-multiplexer
- ✓1:4 De-multiplexer
- ✓1:8 De-multiplexer
- ✓1:16 De-multiplexer
- ✓1:32 De-multiplexer
- ✓1:64 De-multiplexer

and so on.....

1: 2 De-multiplexer

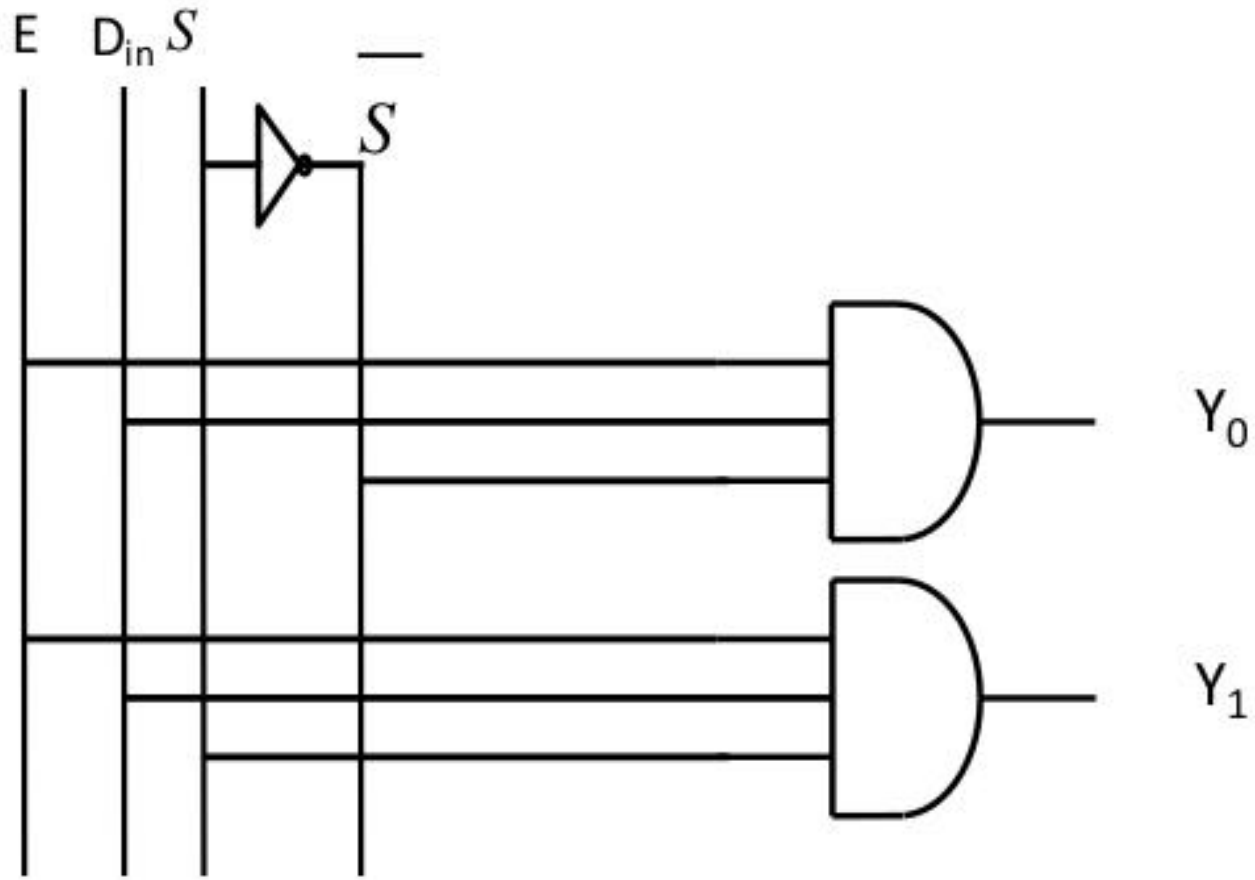


Block Diagram

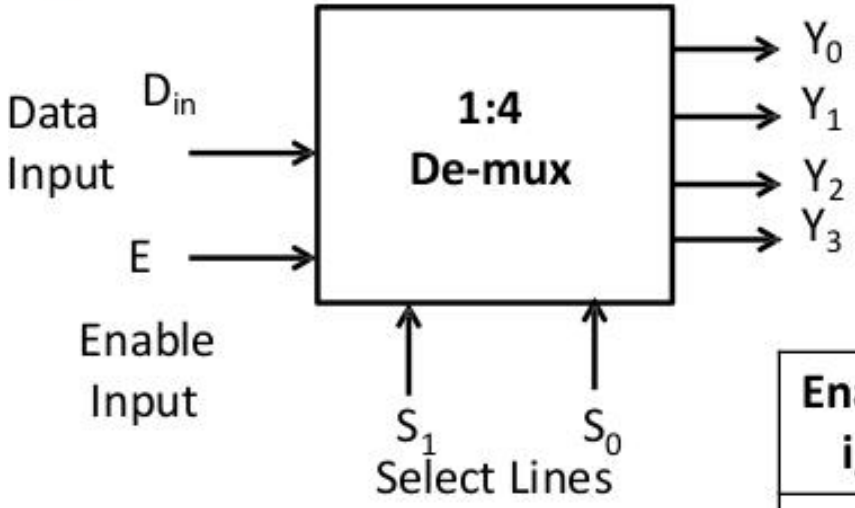
Truth Table

Enable i/p	Select i/p	Outputs	
E	**S**	**Y_0**	**Y_1**
0	X	0	0
1	0	D_{in}	0
1	1	0	D_{in}

1:2 De-mux using basic gates



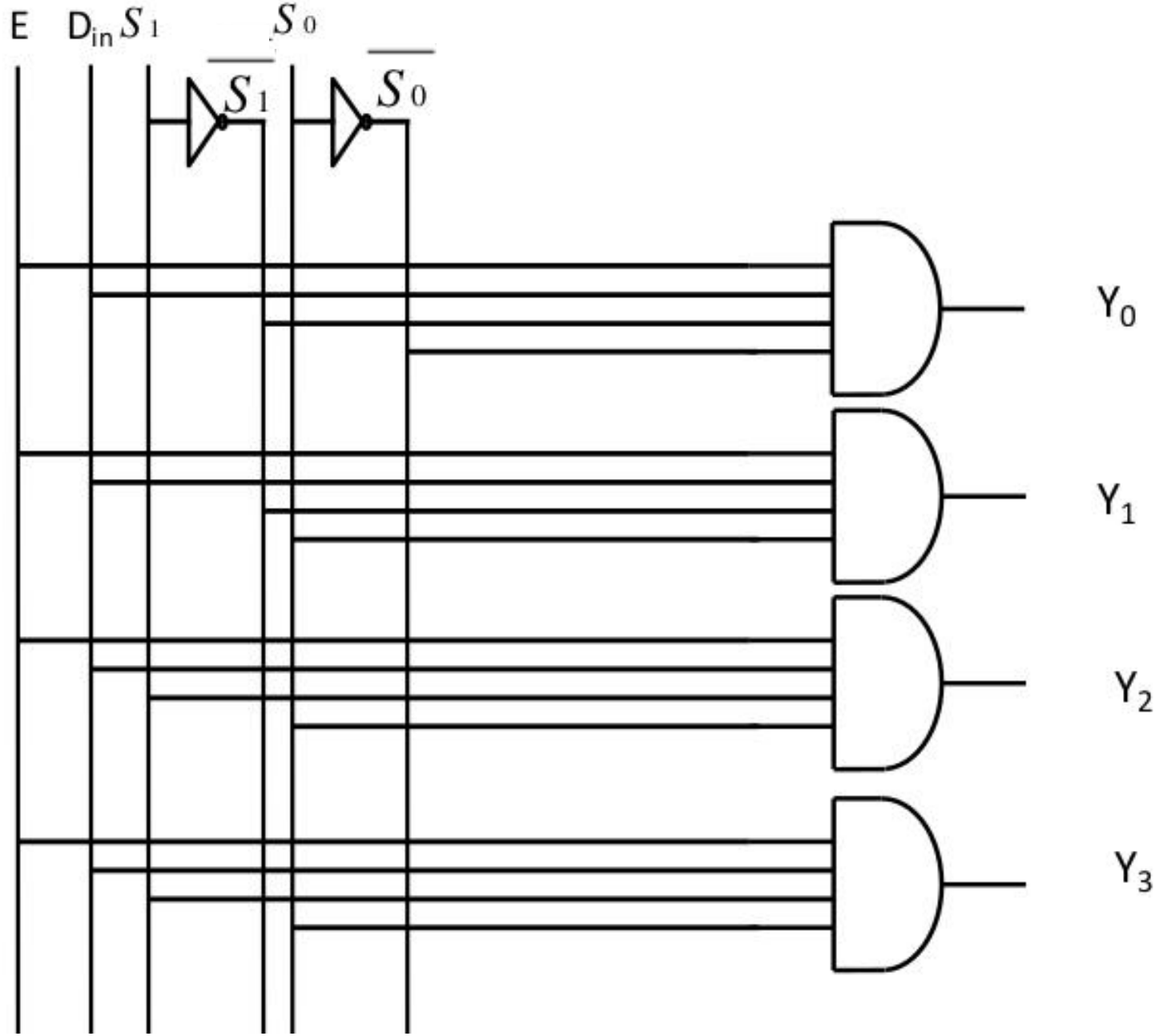
1: 4 De-multiplexer



Block Diagram

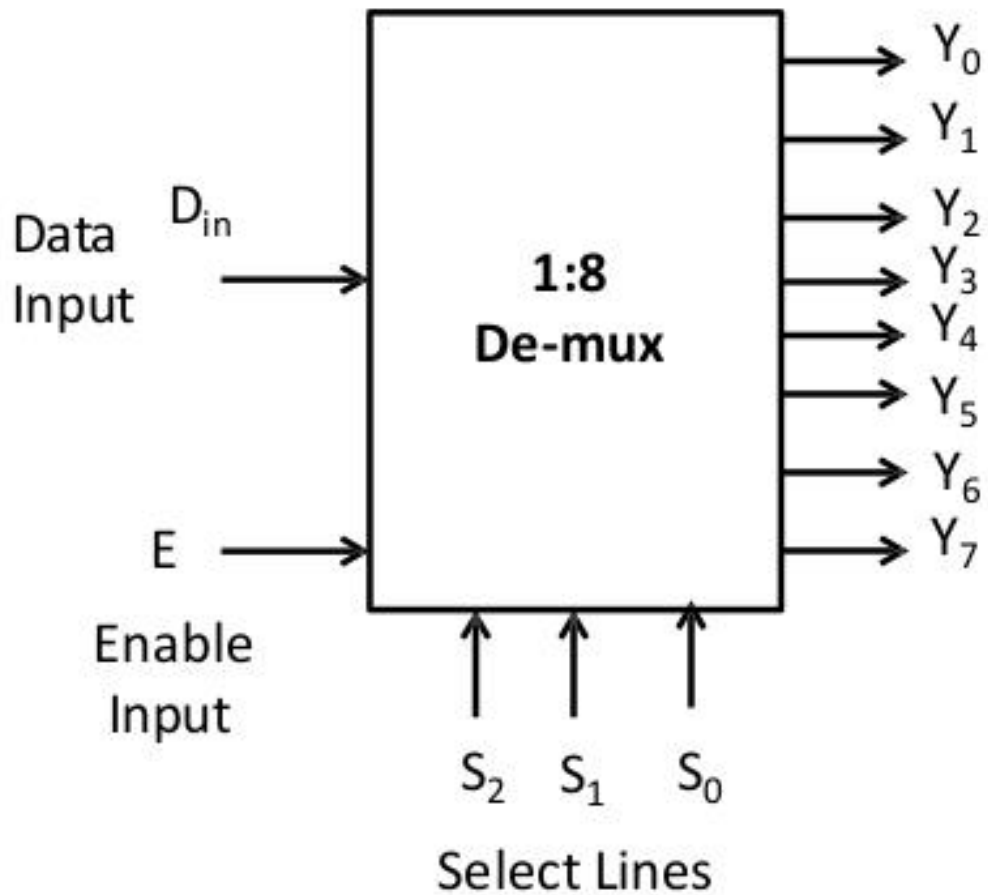
Truth Table

Enable i/p	Select i/p		Outputs			
E	**S_1**	**S_0**	**Y_0**	**Y_1**	**Y_2**	**Y_3**
0	X	X	0	0	0	0
1	0	0	D_{in}	0	0	0
1	0	1	0	D_{in}	0	0
1	1	0	0	0	D_{in}	0
1	1	1	0	0	0	D_{in}


E
D_{in}
S_1
S_0
$\overline{S_1}$
$\overline{S_0}$
Y_0
Y_1
Y_2
Y_3

1: 8 De-multiplexer

Block Diagram



Truth Table

Enable i/p	Select i/p			Outputs							
E	S_2	S_1	S_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	X	X	X	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	D_{in}
1	0	0	1	0	0	0	0	0	0	D_{in}	0
1	0	1	0	0	0	0	0	0	D_{in}	0	0
1	0	1	1	0	0	0	0	D_{in}	0	0	0
1	1	0	0	0	0	0	D_{in}	0	0	0	0
1	1	0	1	0	0	D_{in}	0	0	0	0	0
1	1	1	0	0	D_{in}	0	0	0	0	0	0
1	1	1	1	D_{in}	0	0	0	0	0	0	0

- Assignments:

  --Complete Logic Diagram For 1:8 de-Multiplexer