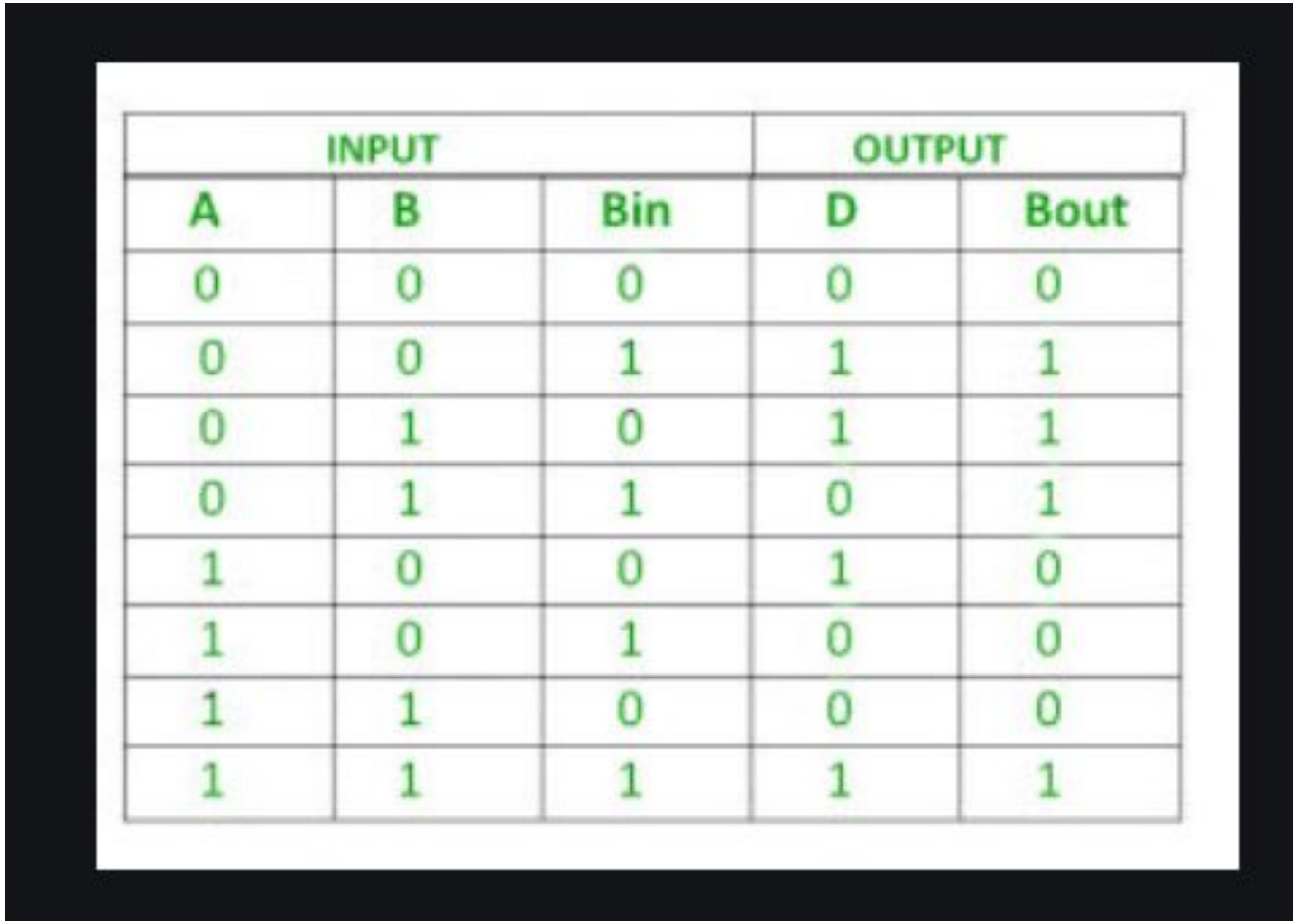
Digital Design and Computer Organization

Module 2

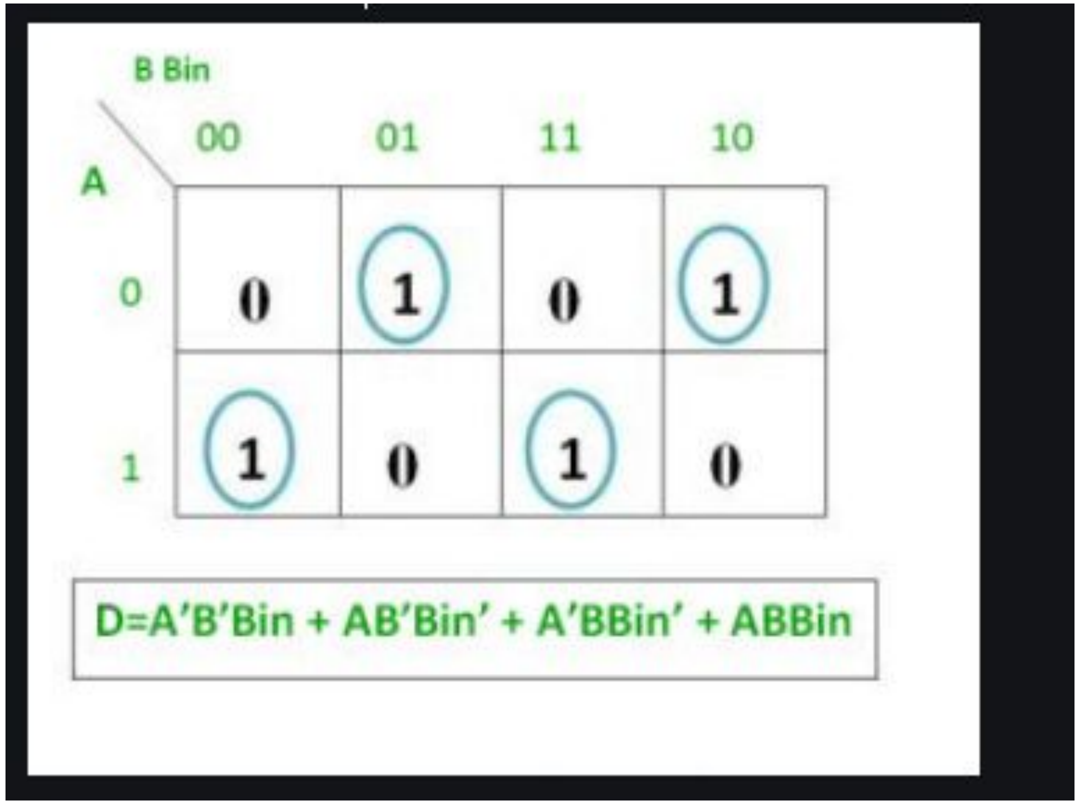
Topics to be covered….

- Combinational Logic
  - Combinational circuits
  - Design Procedure
  - Binary Adder – Subtractor
  - Decoders
  - Encoders
  - Multiplexers
- Sequential Logic
  - Sequential Circuits
  - Latches
  - Flip Flops

Full Subtractor

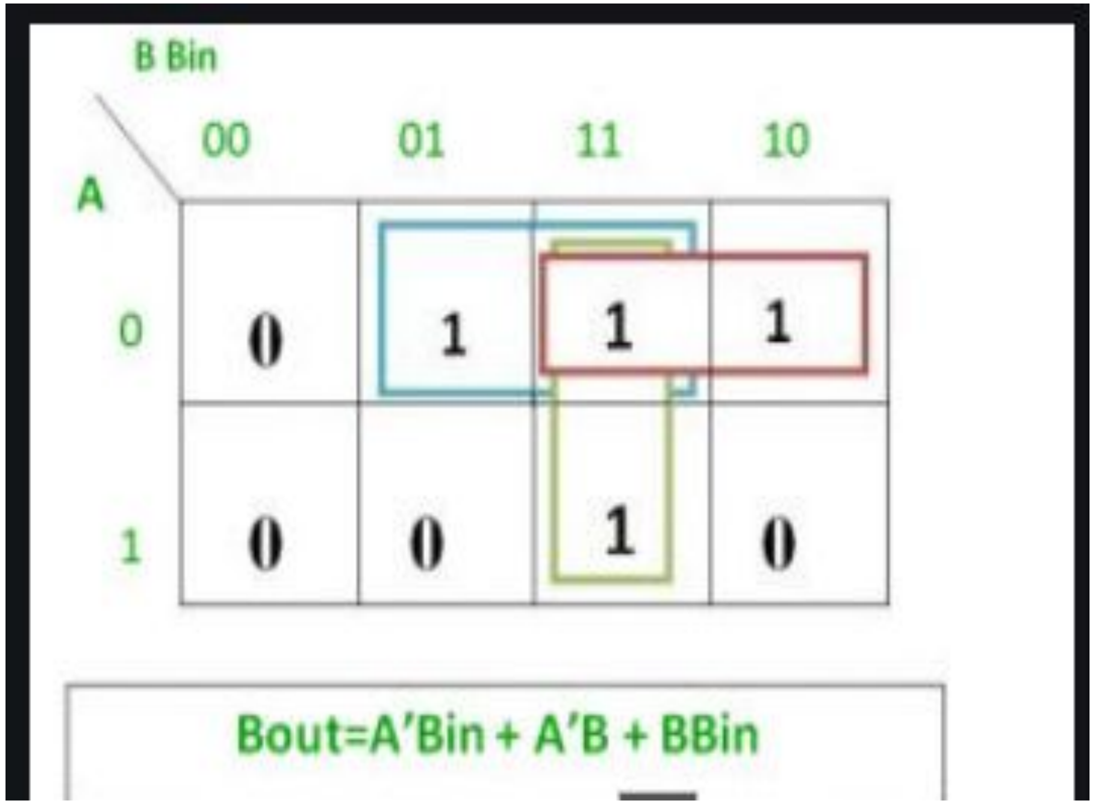


INPUT			OUTPUT	
A	B	Bin	D	Bout
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1



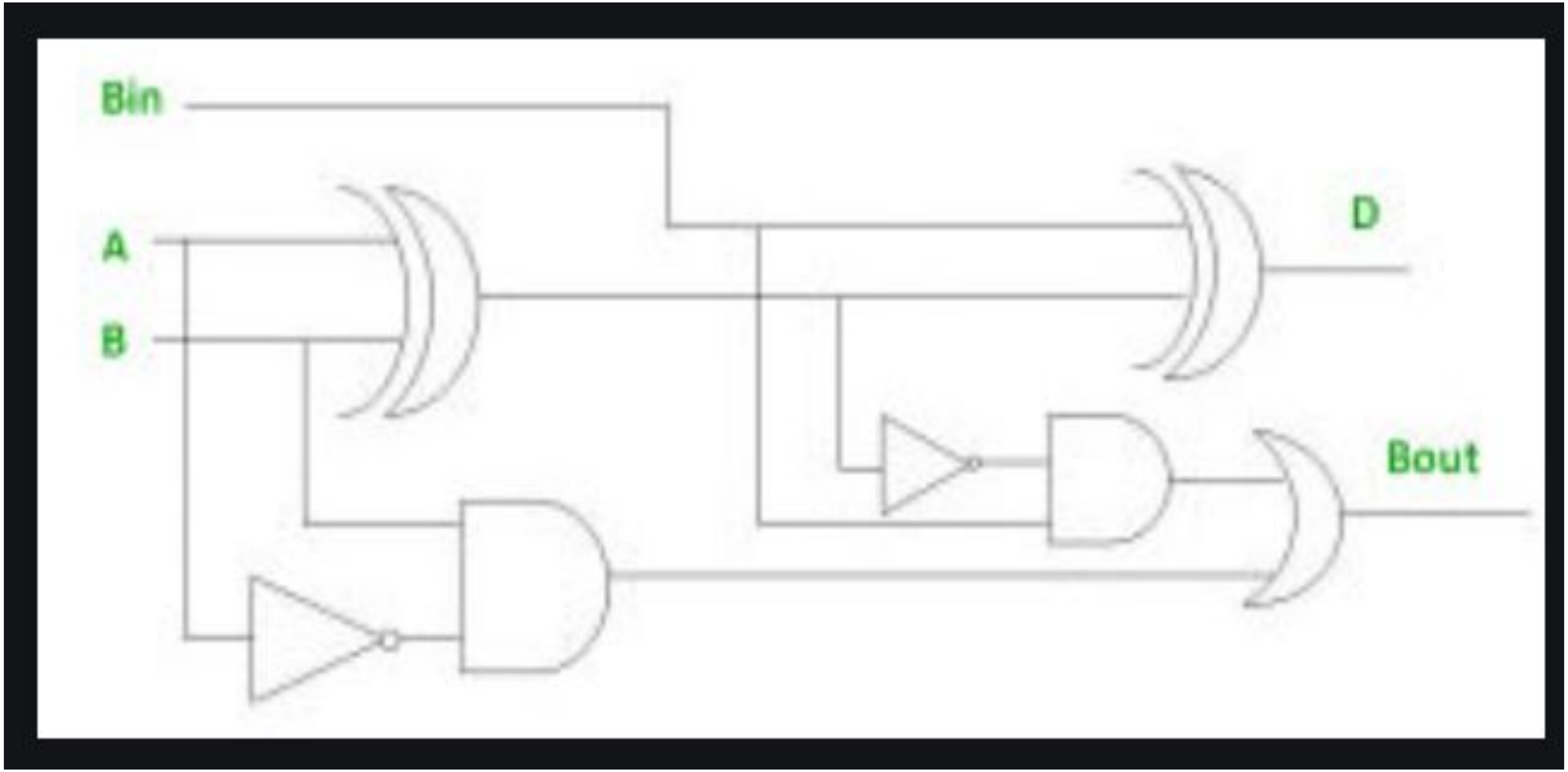
A \ B Bin	00	01	11	10
0	0	1	0	1
1	1	0	1	0

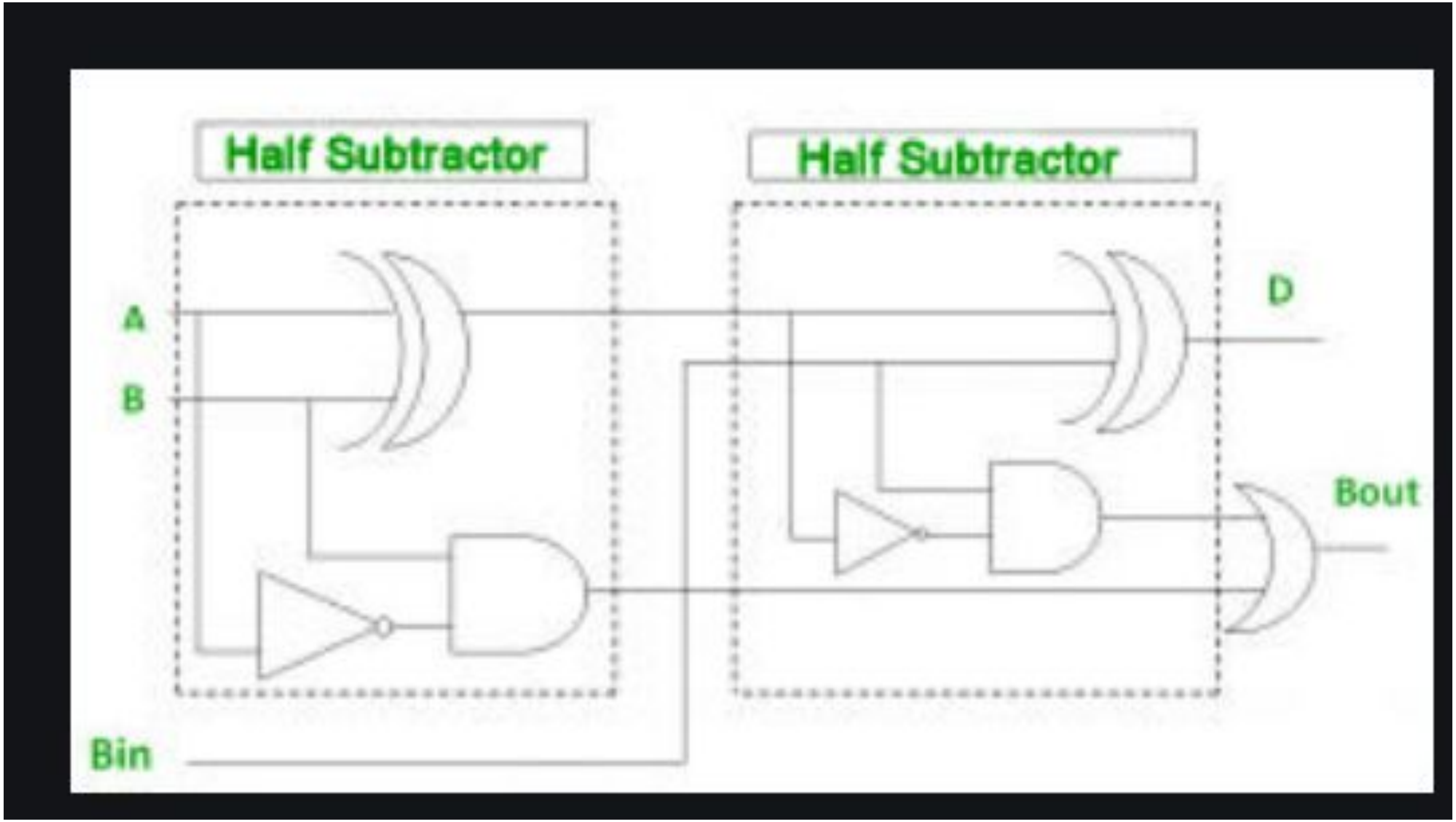
D=A’B’Bin + AB’Bin’ + A’BBin’ + ABBin



A \ B Bin	00	01	11	10
0	0	1	1	1
1	0	0	1	0

Bout=A’Bin + A’B + BBin


Bin
A
B
D
Bout


Half Subtractor
Half Subtractor
A
B
D
Bout
Bin

Binary Adder

- A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers.
- It can be constructed with full adders connected in cascade, with the output carry from each full adder connected to the input carry of the next full adder in the chain.
- Addition of n-bit numbers requires a chain of n full adders or a chain of one-half adder and n - 1 full adders.
- The input carry to the least significant position is fixed at 0.

Subscript *i*:	**3**	**2**	**1**	**0**	
Input carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output carry	0	0	1	1	C_{i+1}

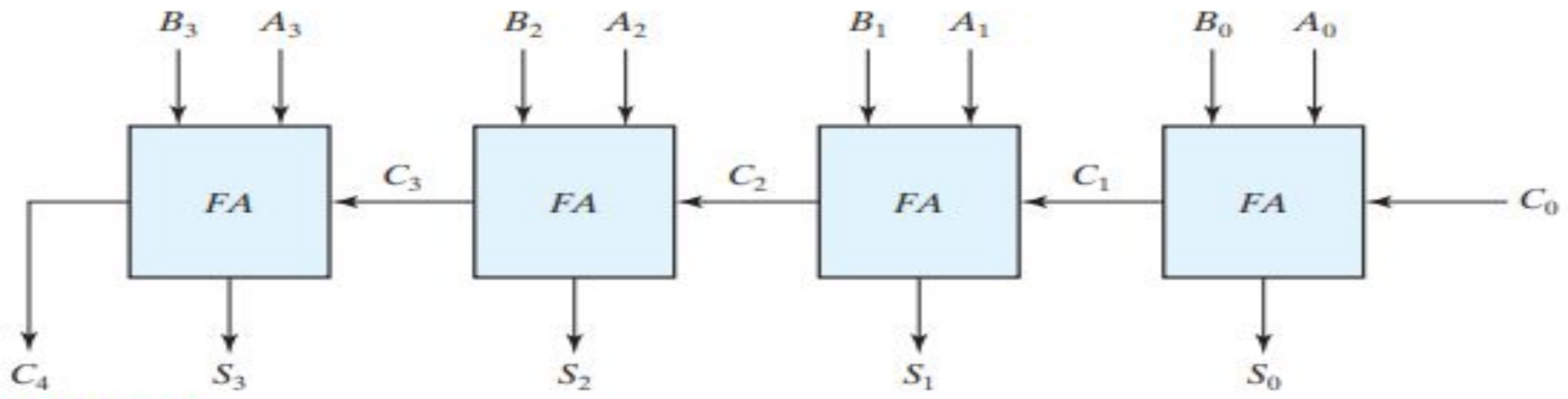


FIGURE 4.9
Four-bit adder

Carry Propagation

- The addition of two binary numbers in parallel implies that all the bits of the augend and addend are available for computation at the same time.
- In any combinational circuit, the signal must propagate through the gates before the correct output sum is available in the output terminals.
- The total propagation time is equal to the propagation delay of a typical gate, times the number of gate levels in the circuit.
- The longest propagation delay time in an adder is the time it takes the carry to propagate through the full adders.
- Inputs A3 and B3 are available as soon as input signals are applied to the adder. However, input carry C3 does not settle to its final value until C2 is available from the previous stage. Similarly, C2 has to wait for C1 and so on down to C0. Thus, only after the carry propagates and ripples through all stages will the last output S3 and carry C4 settle to their final correct value.

- The signals at Pi and Gi settle to their steady-state values after they propagate through their respective gates. These two signals are common to all half adders and depend on only the input augend and addend bits.
- The signal from the input carry Ci to the output carry Ci+1 propagates through an AND gate and an OR gate, which constitute two gate levels.
- If there are four full adders in the adder, the output carry C4 would have 2 * 4 = 8 gate levels from C0 to C4.
- For an n -bit adder, there are 2 n gate levels for the carry to propagate from input to output.

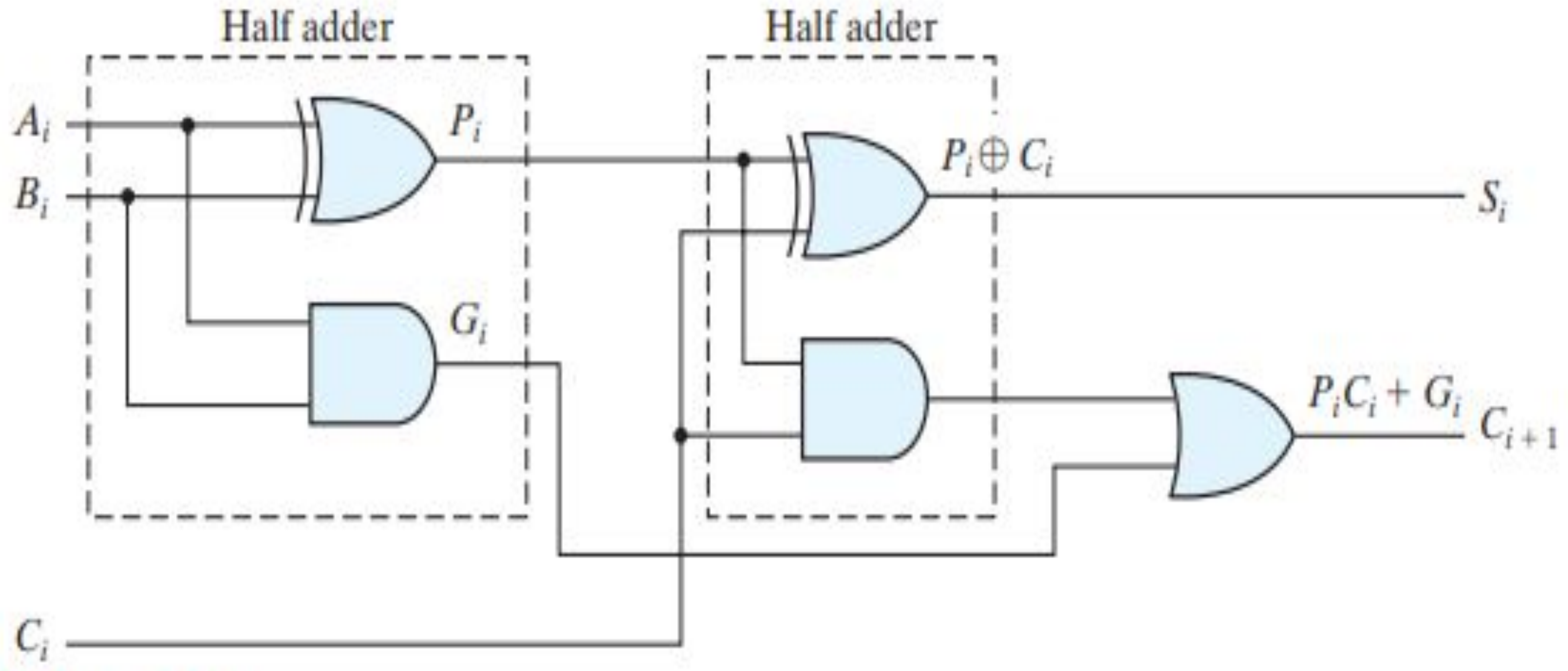
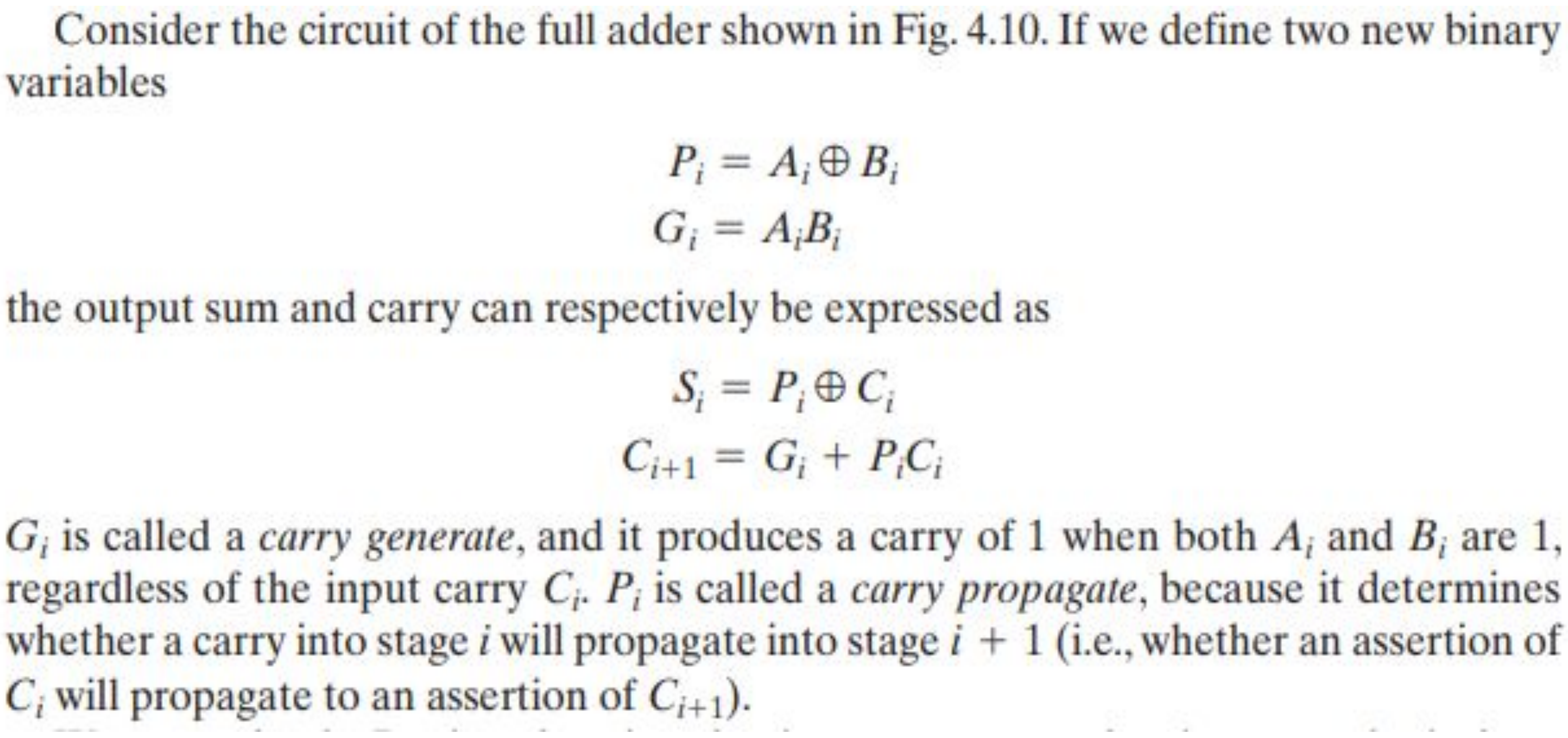


FIGURE 4.10
Full adder with *P* and *G* shown

- There are several techniques for reducing the carry propagation time in a parallel adder. The most widely used technique employs the principle of carry lookahead logic .

-



Consider the circuit of the full adder shown in Fig. 4.10. If we define two new binary variables

$$P_i = A_i \oplus B_i$$
$$G_i = A_i B_i$$

the output sum and carry can respectively be expressed as

$$S_i = P_i \oplus C_i$$
$$C_{i+1} = G_i + P_i C_i$$

G_i is called a *carry generate*, and it produces a carry of 1 when both A_i and B_i are 1, regardless of the input carry C_i. P_i is called a *carry propagate*, because it determines whether a carry into stage i will propagate into stage $i + 1$ (i.e., whether an assertion of C_i will propagate to an assertion of C_{i+1}).

$$C_1 = G_0 + P_0 C_{in}$$

$$C_2 = G_1 + P_1 C_1 = G_1 + P_1 G_0 + P_1 P_0 C_{in}$$

$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{in}$$

$$C_4 = G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{in}$$

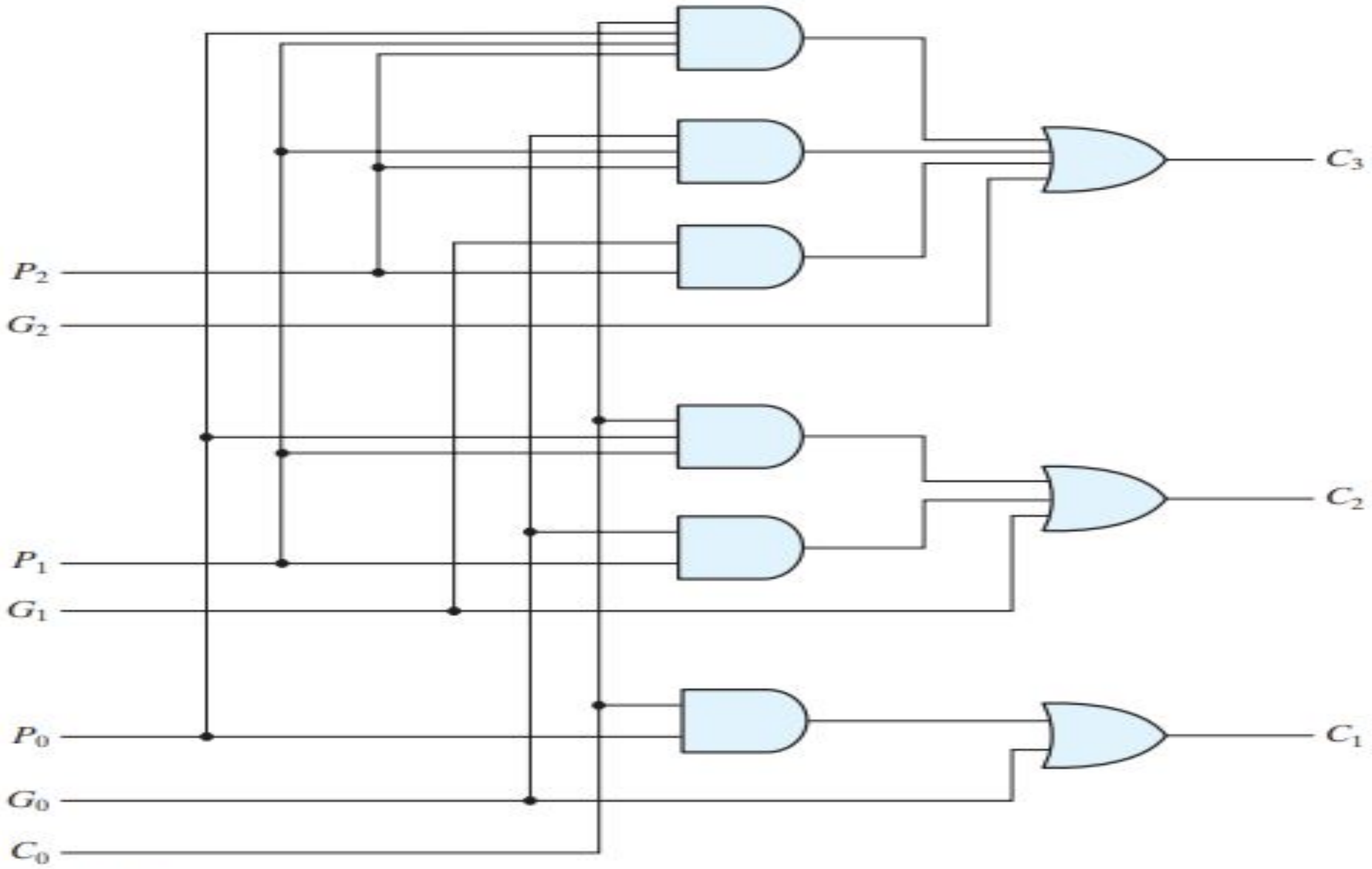


FIGURE 4.11
Logic diagram of carry lookahead generator

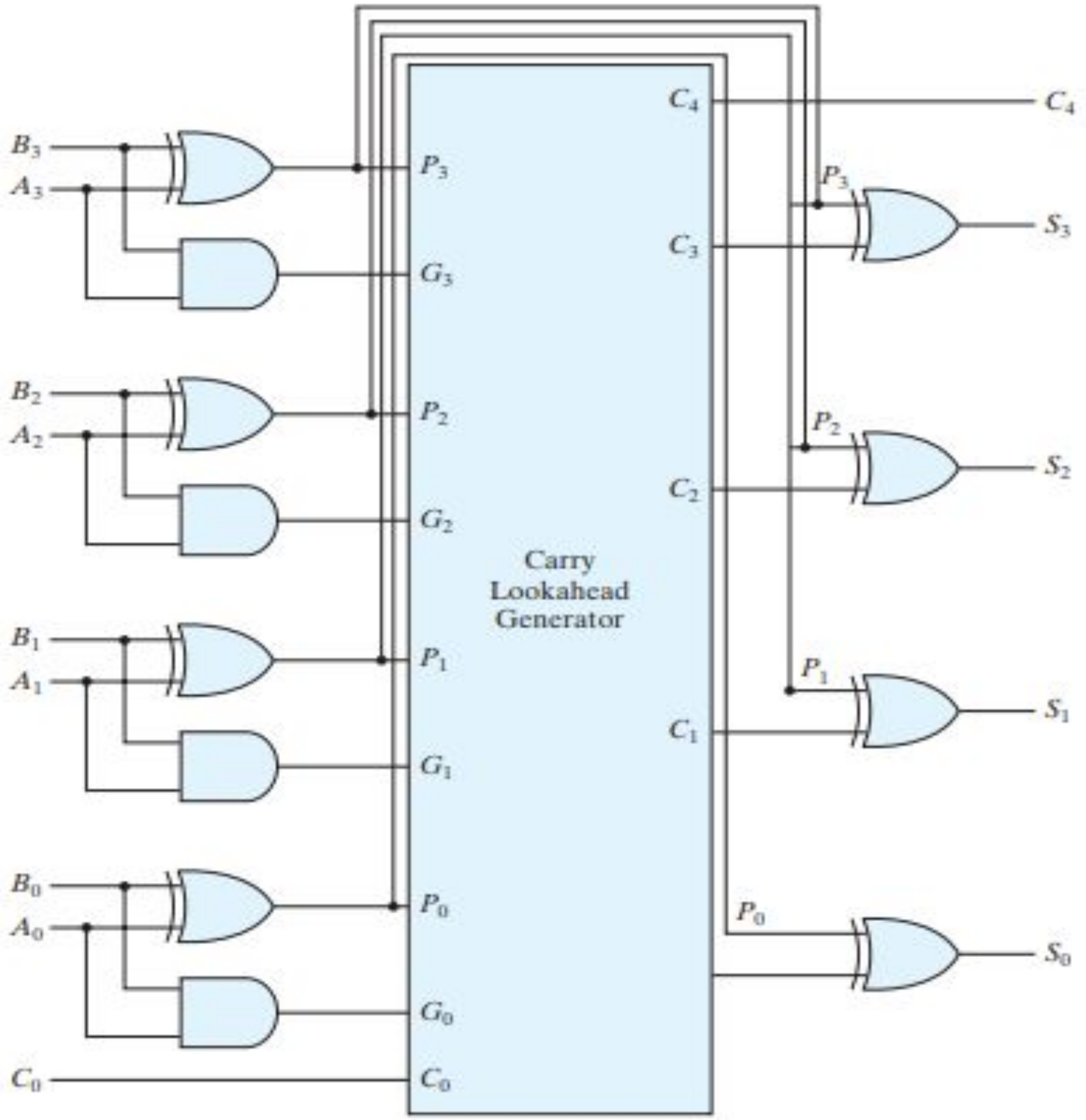


FIGURE 4.12
Four-bit adder with carry lookahead

Binary Subtractor

- The subtraction of unsigned binary numbers can be done most conveniently by means of complement.
- Remember that the subtraction A - B can be done by taking the 2’s complement of B and adding it to A .
- The 2’s complement can be obtained by taking the 1’s complement and adding 1 to the least significant pair of bits.
- The 1’s complement can be implemented with inverters, and a 1 can be added to the sum through the input carry.

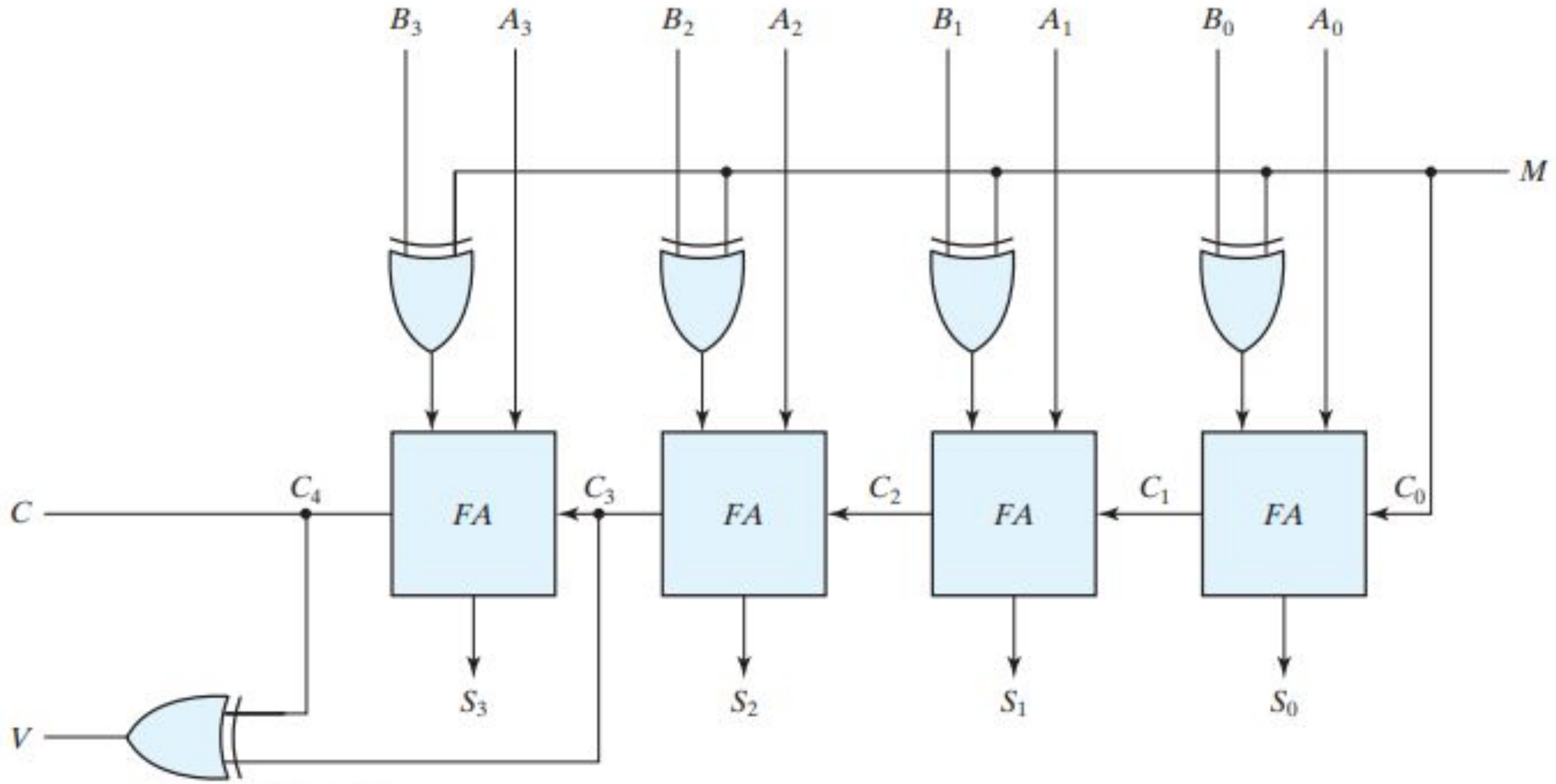


FIGURE 4.13
Four-bit adder–subtractor (with overflow detection)

Decoders

- Discrete quantities of information are represented in digital systems by binary codes.
- A binary code of n bits is capable of representing up to 2^n distinct elements of coded information.
- A decoder is a combinational circuit that converts binary information from

n input lines to a maximum of 2^n unique output lines.

- The decoders presented here are called n -to- m -line decoders, where

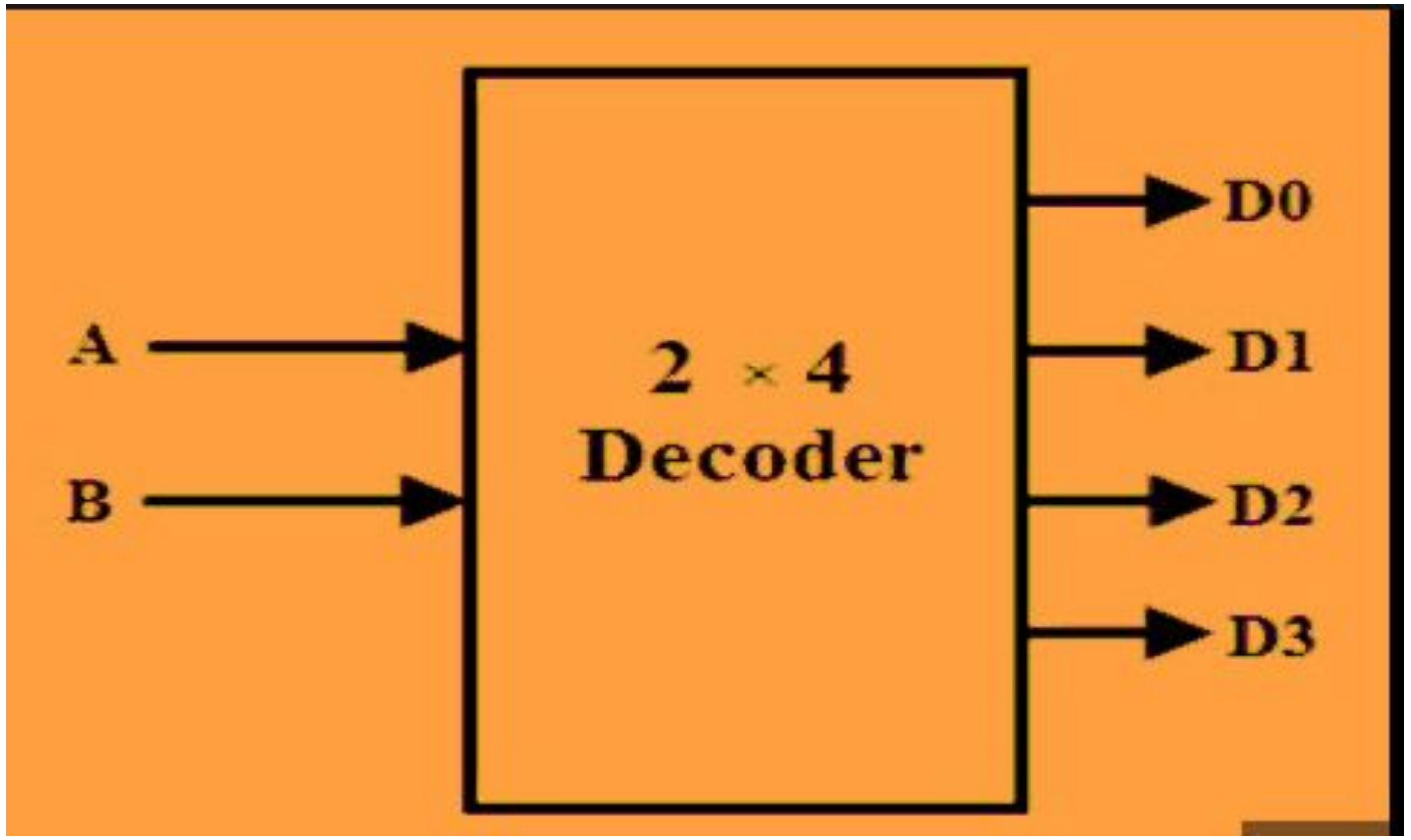
$M <= 2^n$

Their purpose is to generate the 2^n minterms of n input variables. Each combination of inputs will assert a unique output.

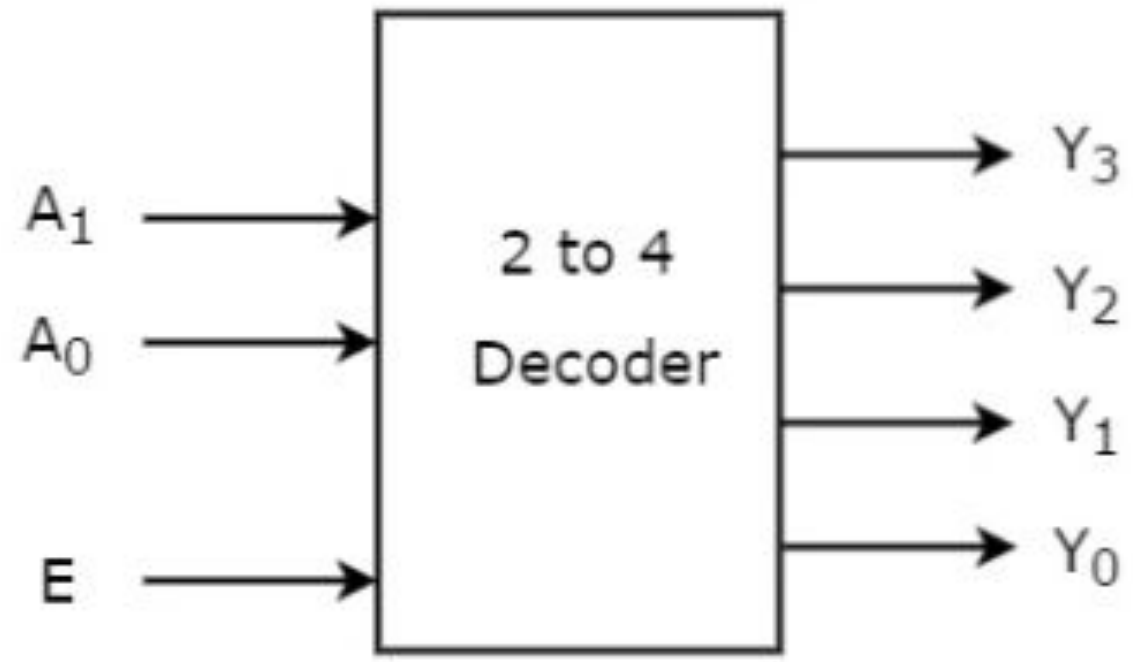
Applications of Decoders

- Decoders are crucial in selecting specific memory locations in RAM, ROM, and other storage devices.
- Decoders are used to control seven-segment displays, which are commonly found in digital clocks, calculators, and electronic meters.
- Decoders are used in computer keyboards and other input devices to detect which key has been pressed.
- Decoders are essential in multiplexers (MUX) and demultiplexers (DEMUX), which are used for routing data in communication systems, such as data buses or network systems.

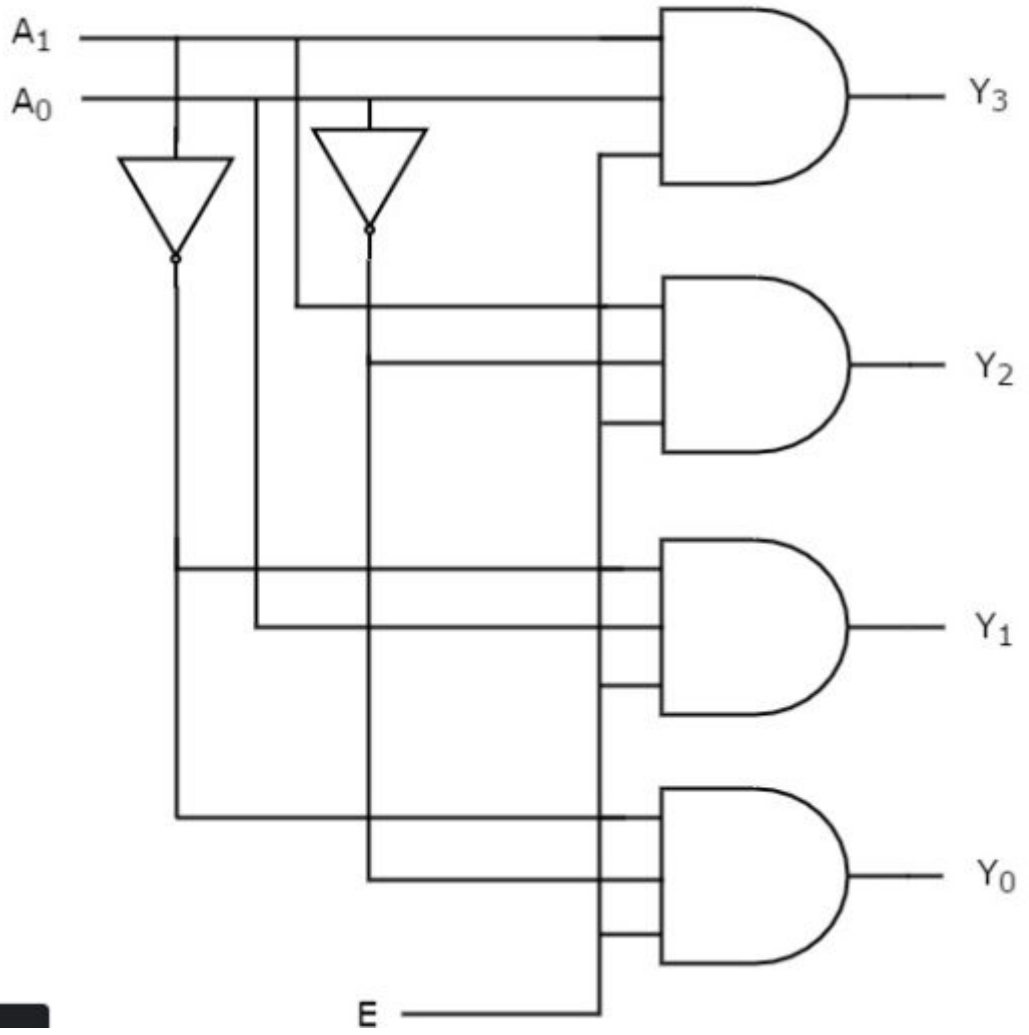
Example of 2 to 4 Decoder



2 to 4 Decoder with enable input



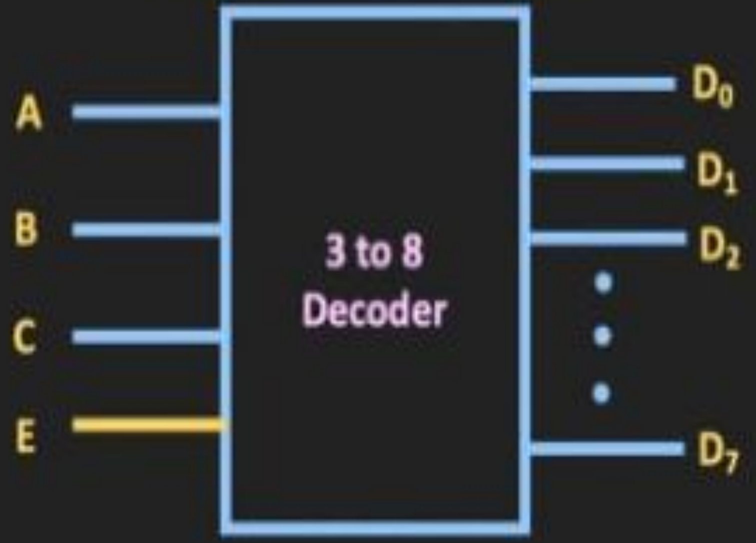
Enable	Inputs		Outputs			
E	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0



[redacted]

3 to 8 Decoder

Decoder with Enable Input

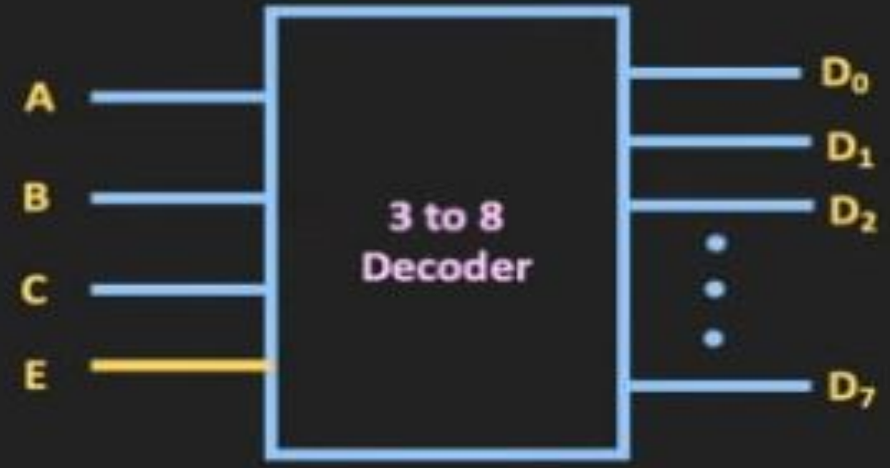


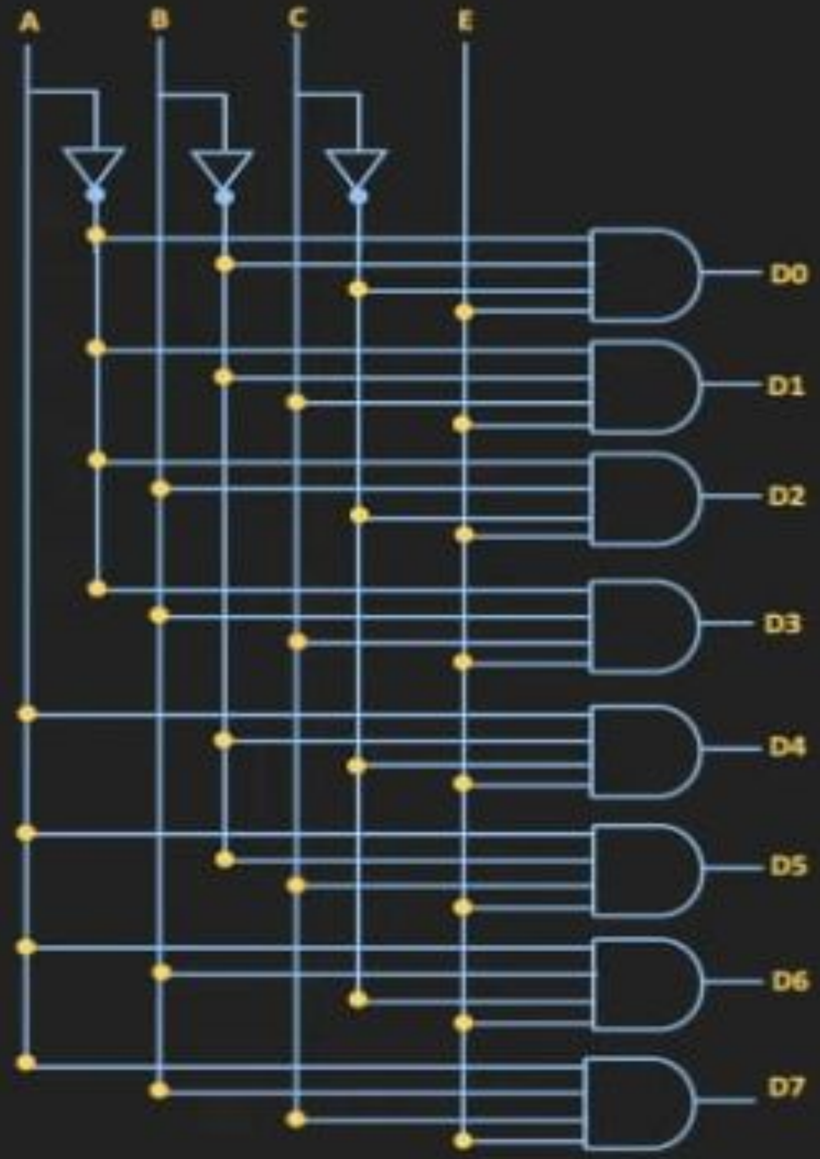
Truth Table

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

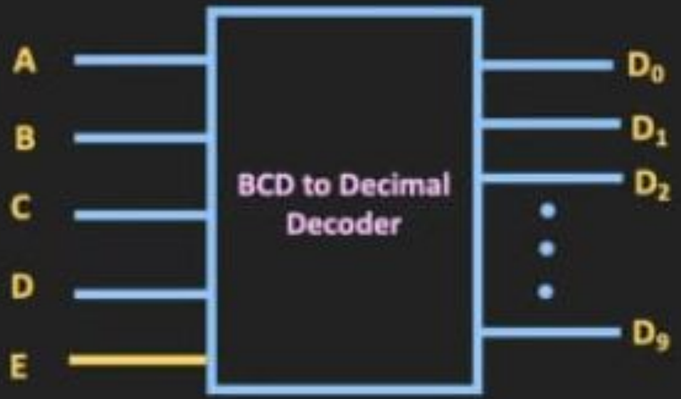
3 to 8 Decoder

Decoder with Enable Input



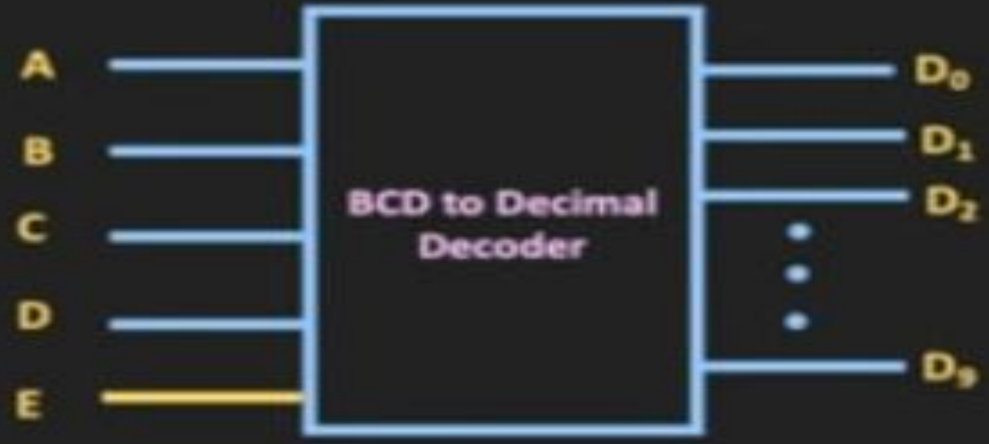


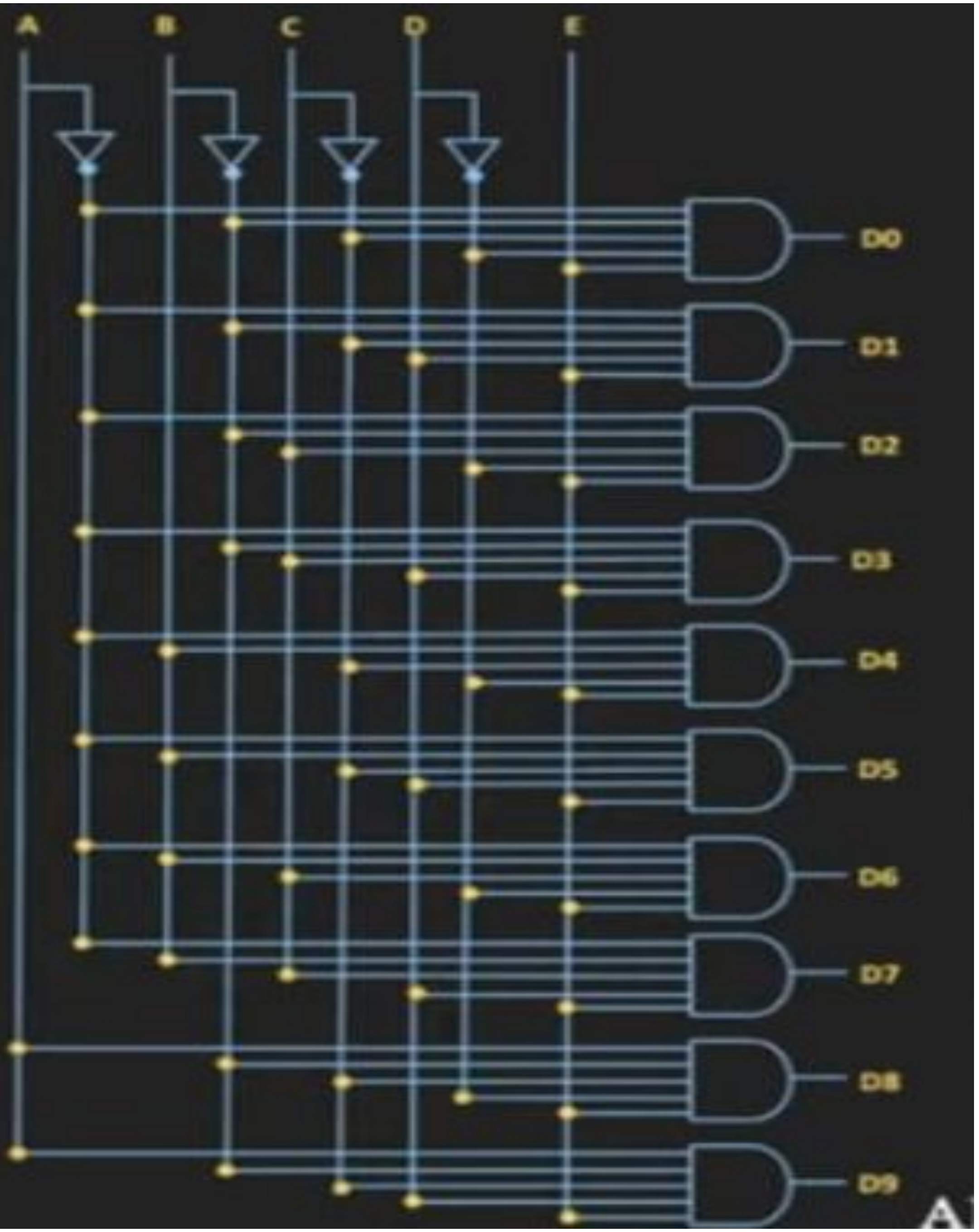
BCD to Decimal Decoder



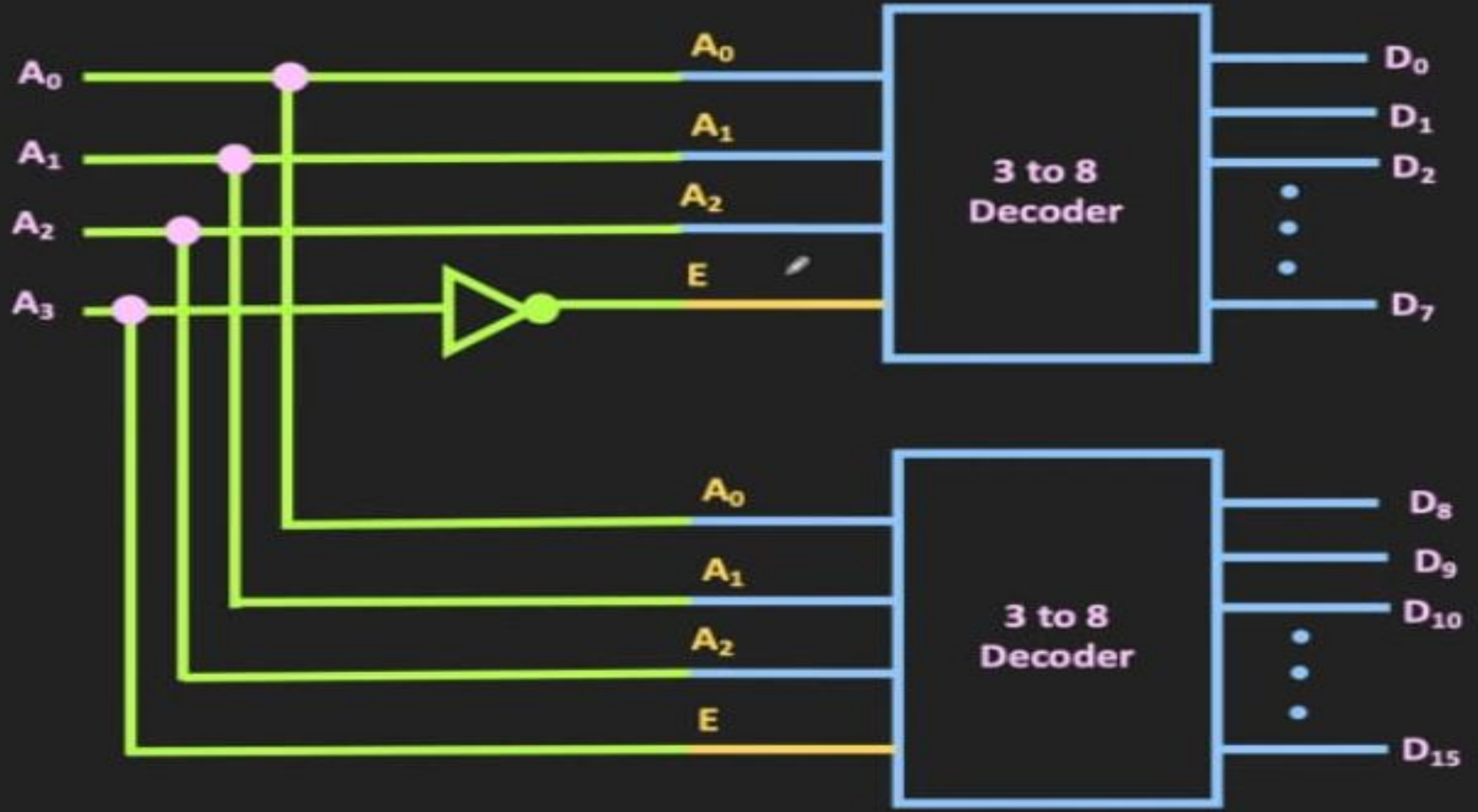
Truth Table

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

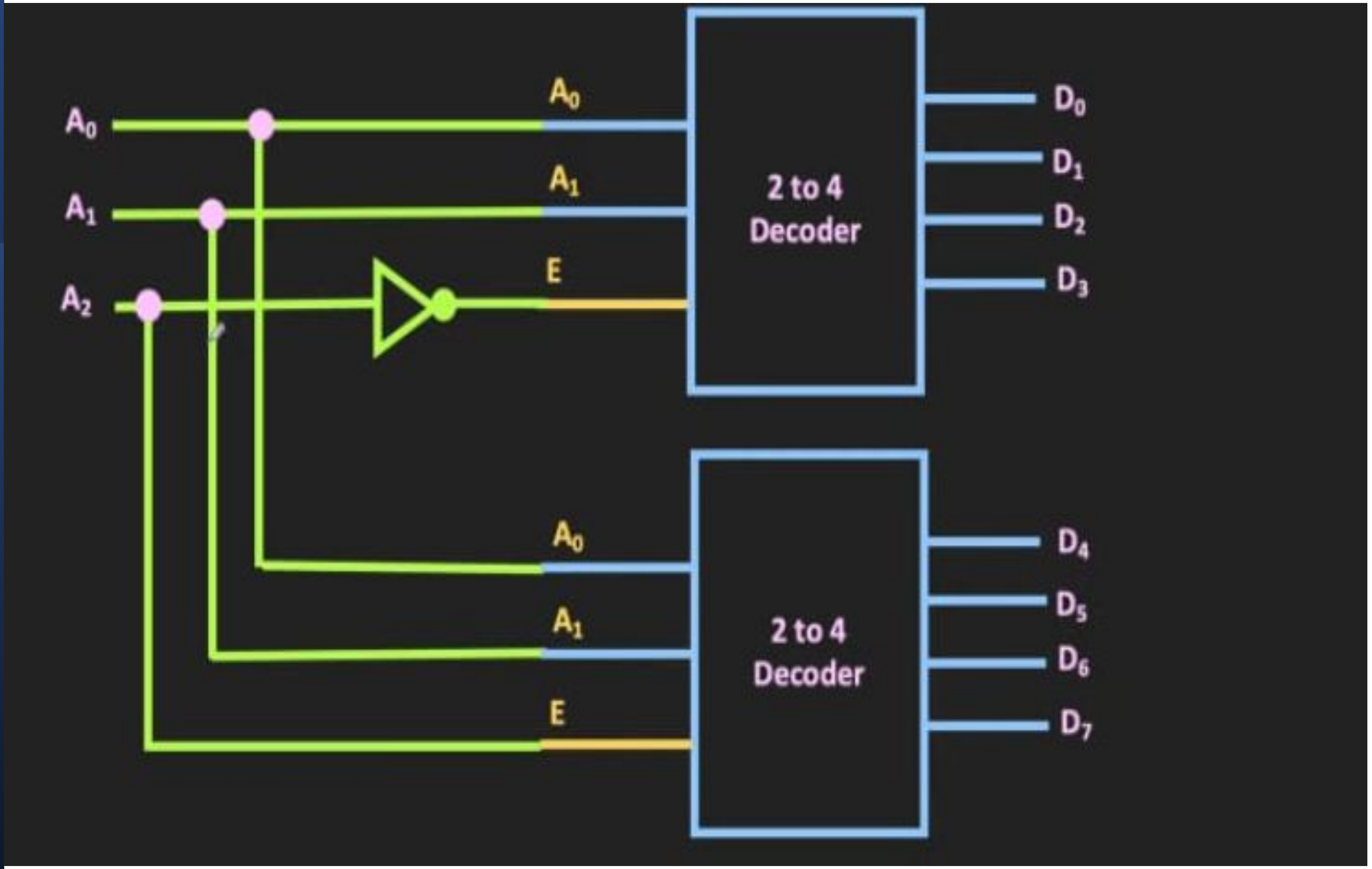

A
B
C
D
E
BCD to Decimal Decoder
D_0
D_1
D_2
D_9


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

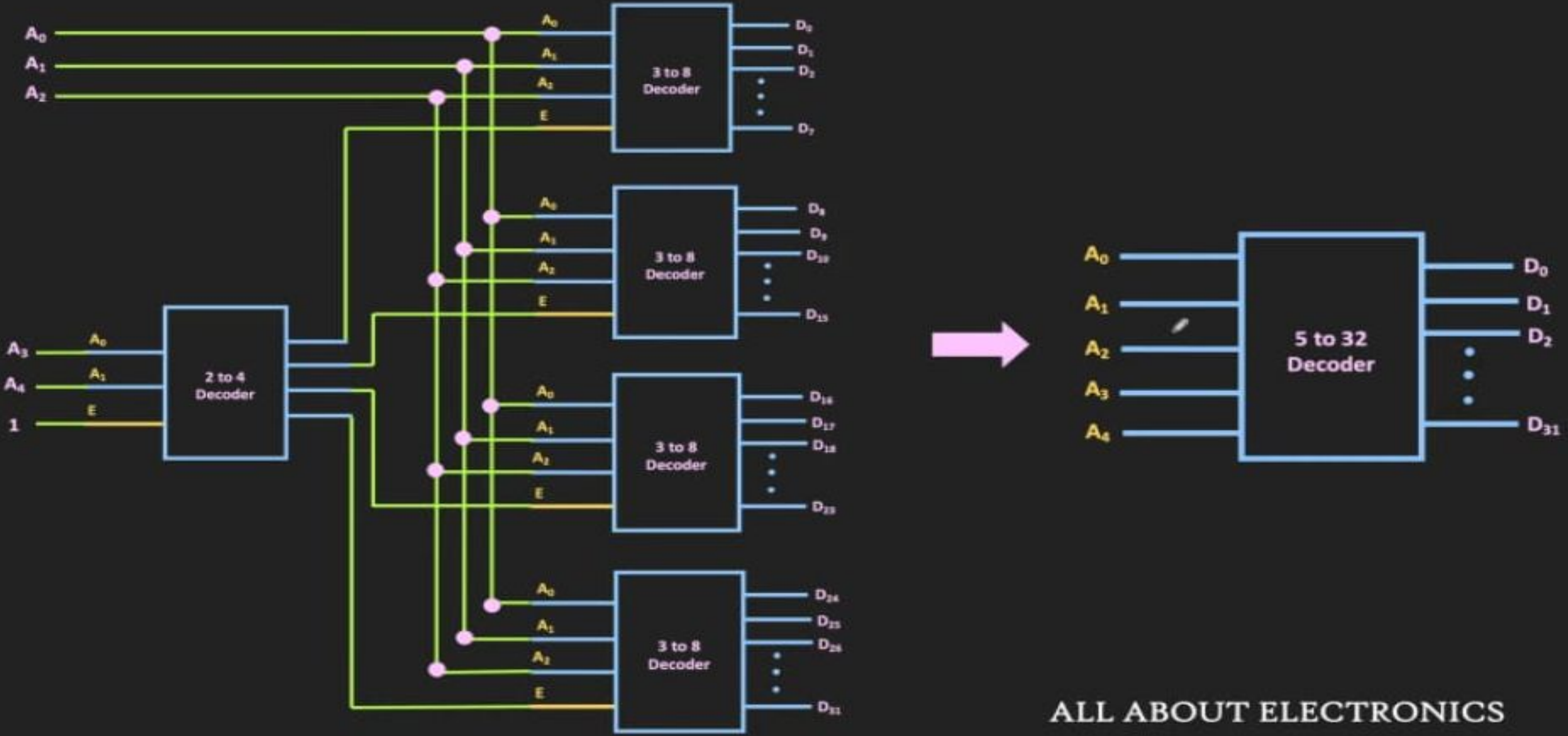
4 to 16 Decoder using 3 to 8 Decoders



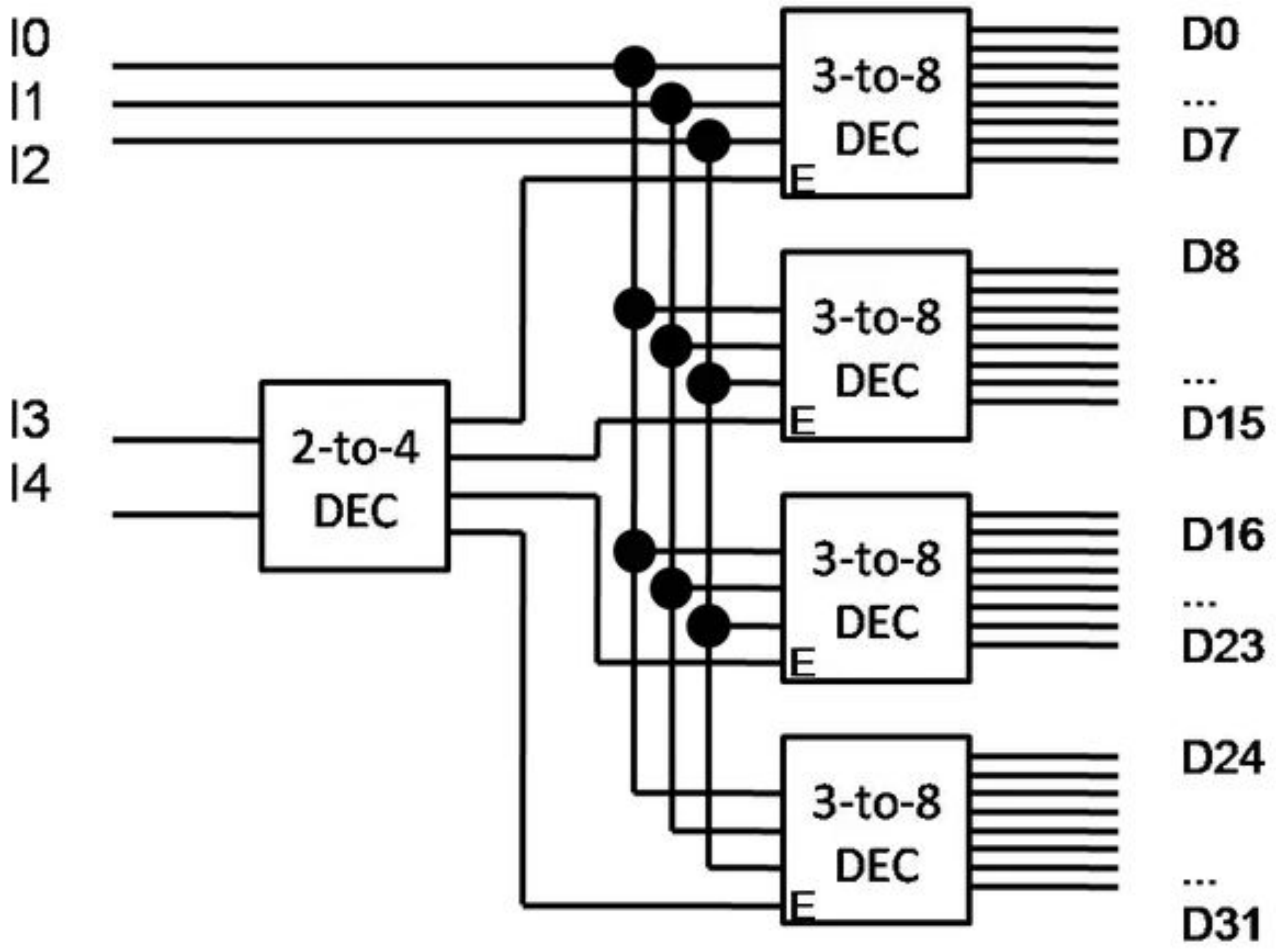
3 to 8 using 2 to 4 Decoder



5 to 32 Decoder using 3 to 8 Decoders



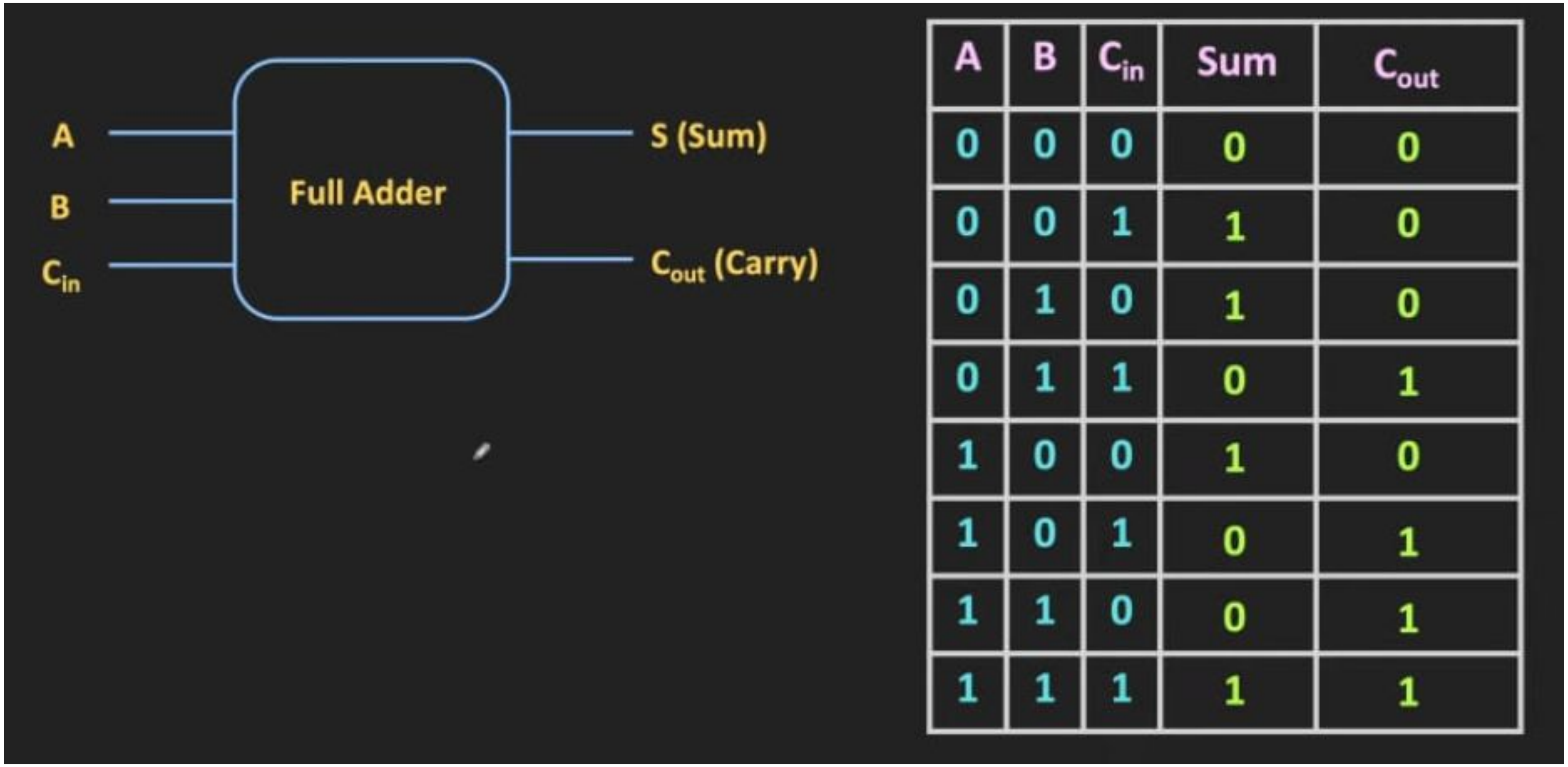
ALL ABOUT ELECTRONICS


I0
I1
I2
I3
I4
2-to-4
DEC
3-to-8
DEC
E
D0
...
D7
D8
...
D15
D16
...
D23
D24
...
D31

Combinational Logic Implementation

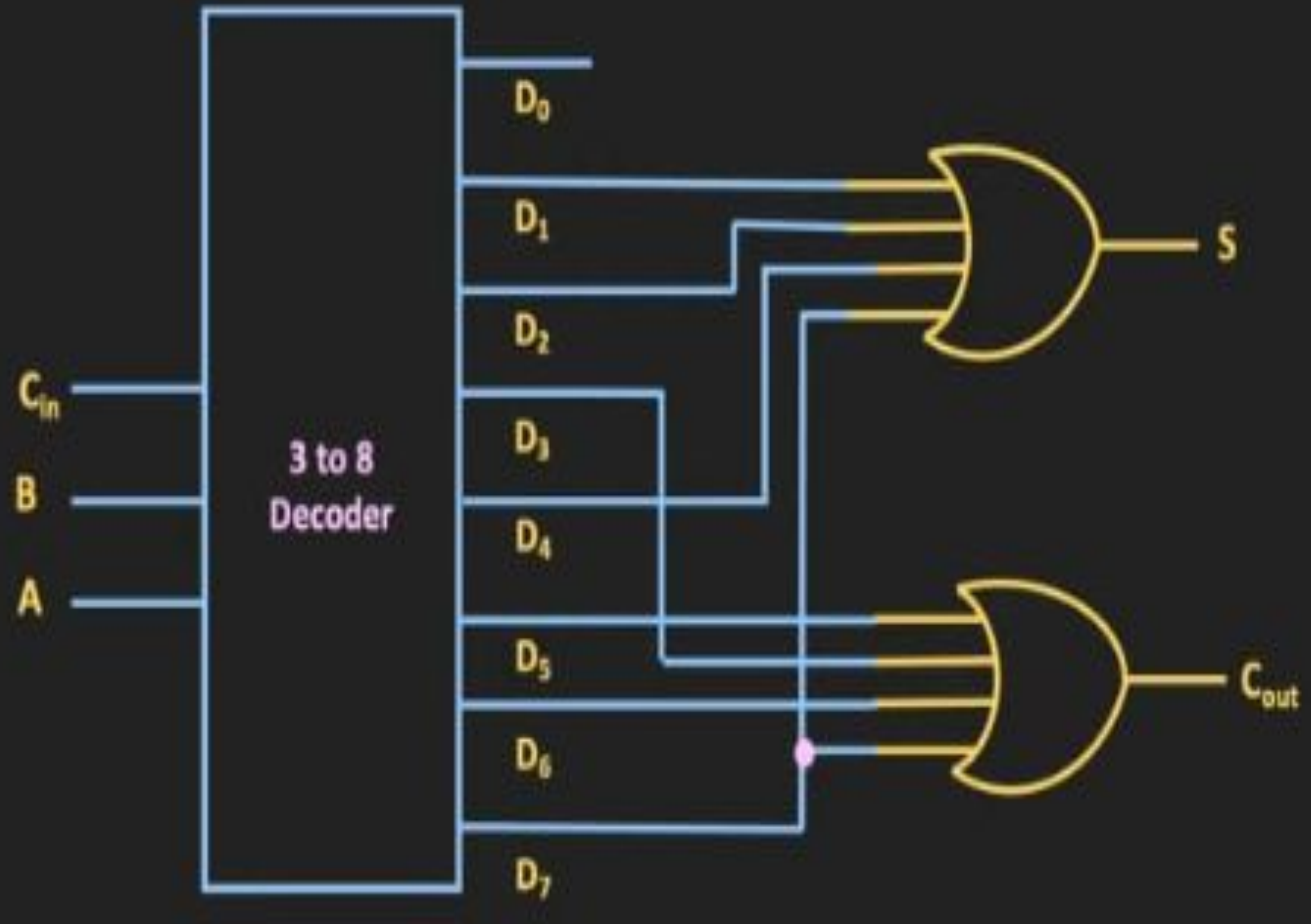
- Since any Boolean function is expressed in terms of minterms, one can use a decoder to generate minterms and an external OR gate to form a logical sum.
- The procedure for implementing a combinational circuit by means of a decoder and OR gates requires that the Boolean function for circuit is expressed in sum of minterms.
- The inputs to each decoder is then chosen that generates all the minterms of the input variables.
- The inputs to each OR gate are selected from decoder outputs according to the list of minterms of each functions.
- Consider an example : Full Adder

Combinational Logic Implementation

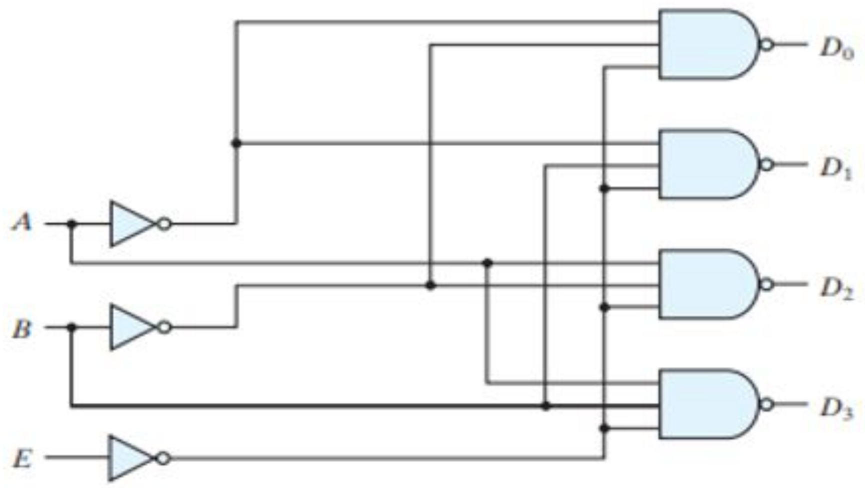


A	B	C_{in}	Sum	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

A	B	C_{in}	Sum	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



Decoders are constructed with NAND gates



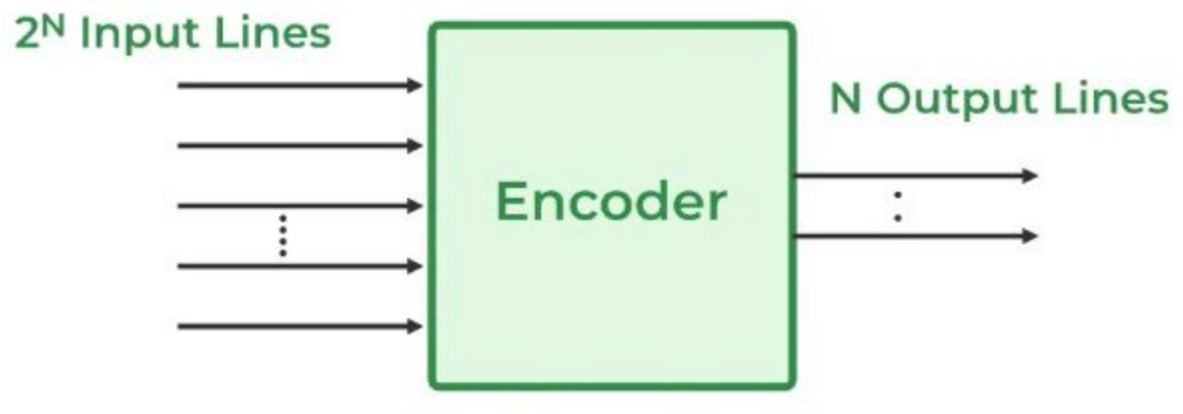
(a) Logic diagram

FIGURE 4.19
Two-to-four-line decoder with enable input

E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

Encoder

- An encoder is a digital circuit that performs the inverse operation of a decoder.
- An encoder has 2^n input lines and n output lines. The output lines, as an aggregate, generate the binary code corresponding to the input value.
- Example :



- Encoders are used in industrial automation to monitor and control machinery, conveyor belts, and other equipment.
- Encoders are used in sensors and measurement systems to convert physical phenomena (such as position, speed, or pressure) into a digital or analog signal.

Examples

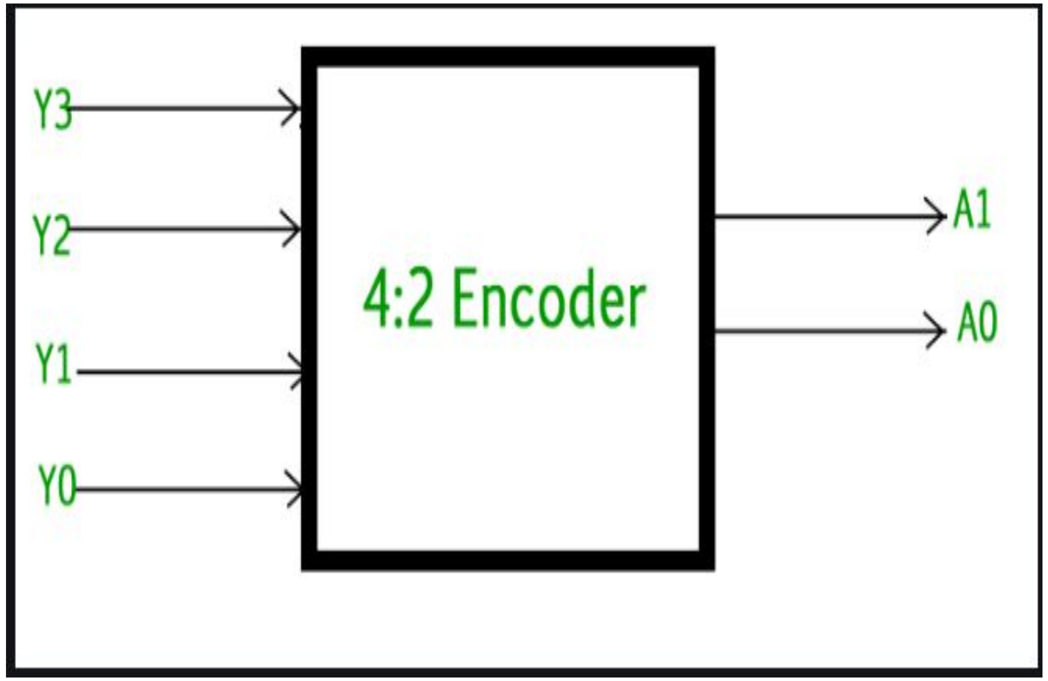
There are different types of Encoders which are mentioned below.

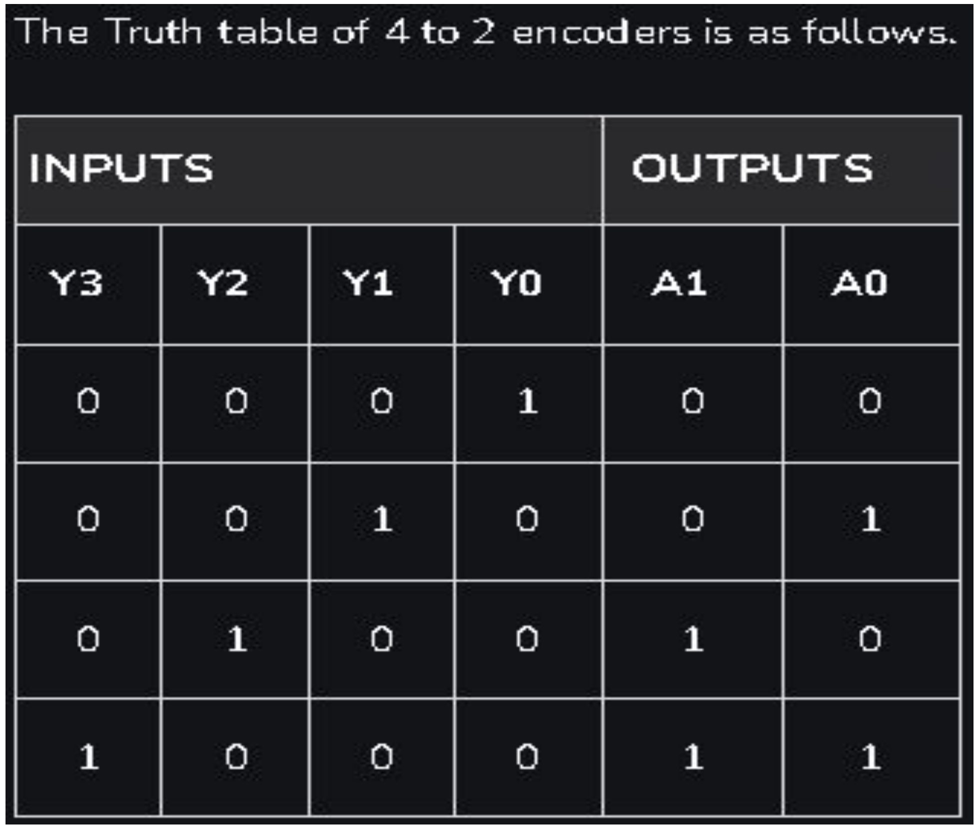
- 4 to 2 Encoder.
- Octal to Binary Encoder (8 to 3 Encoder).
- Decimal to BCD Encoder.
- Priority Encoder.

- **4 to 2 Encoder**

4 to 2 Encoder consists of **four inputs Y3, Y2, Y1 & Y0, and two outputs A1 & A0**.

At any time, only one of these 4 inputs can be ‘1’ in order to get the respective binary code at the output.



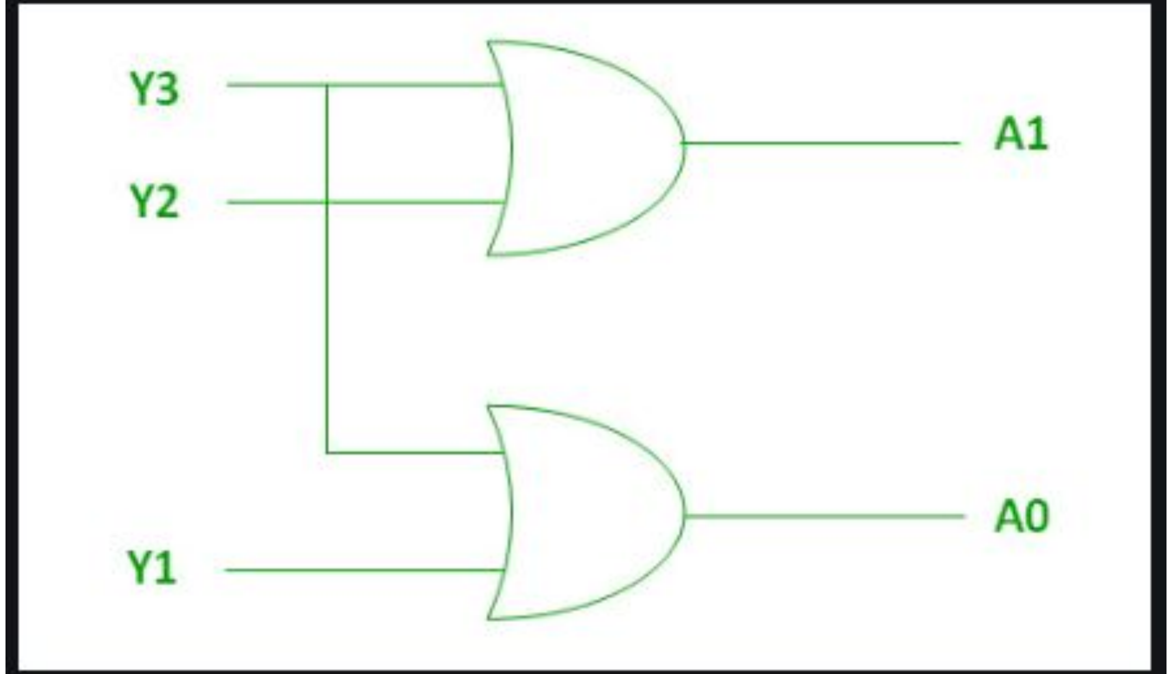

The Truth table of 4 to 2 encoders is as follows.

INPUTS				OUTPUTS	
Y3	Y2	Y1	Y0	A1	A0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

Logical expression for A1 and A0:

A0 = Y3 + Y1

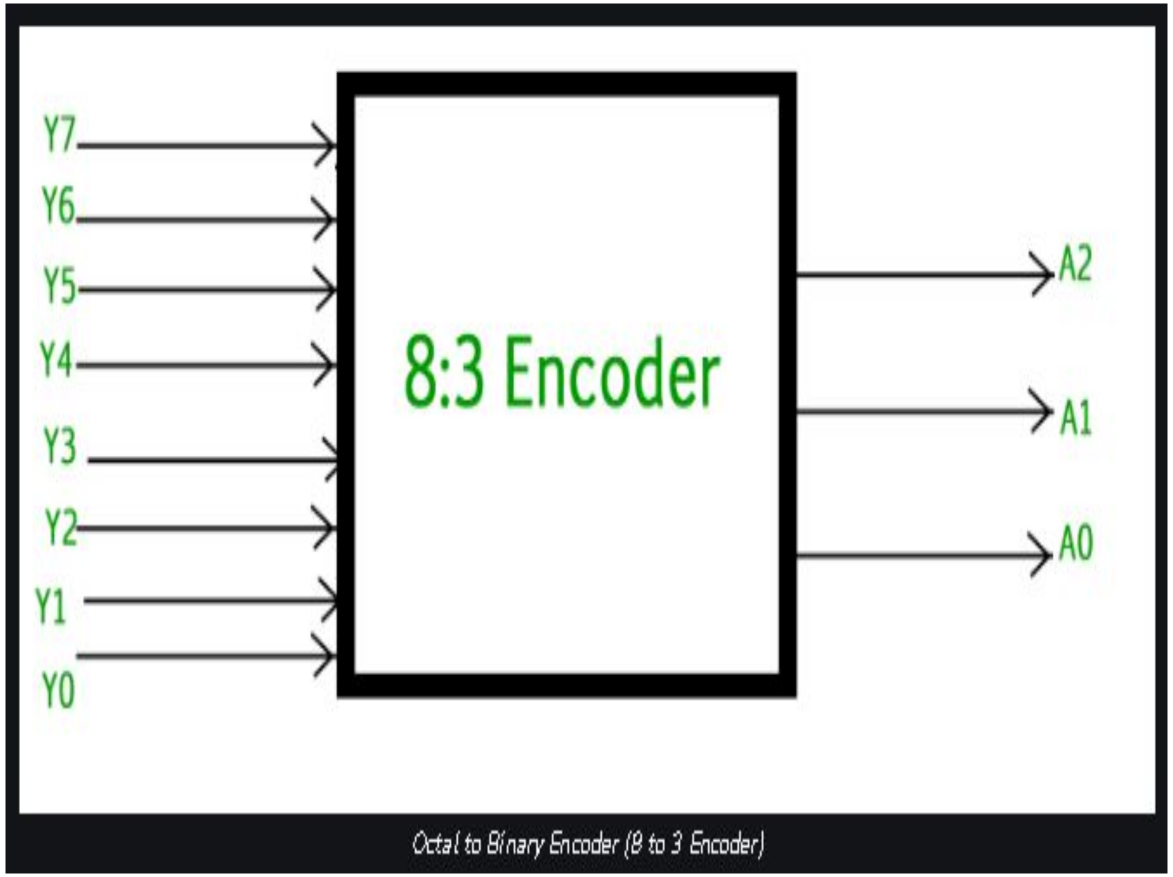
A1 = Y3 + Y2



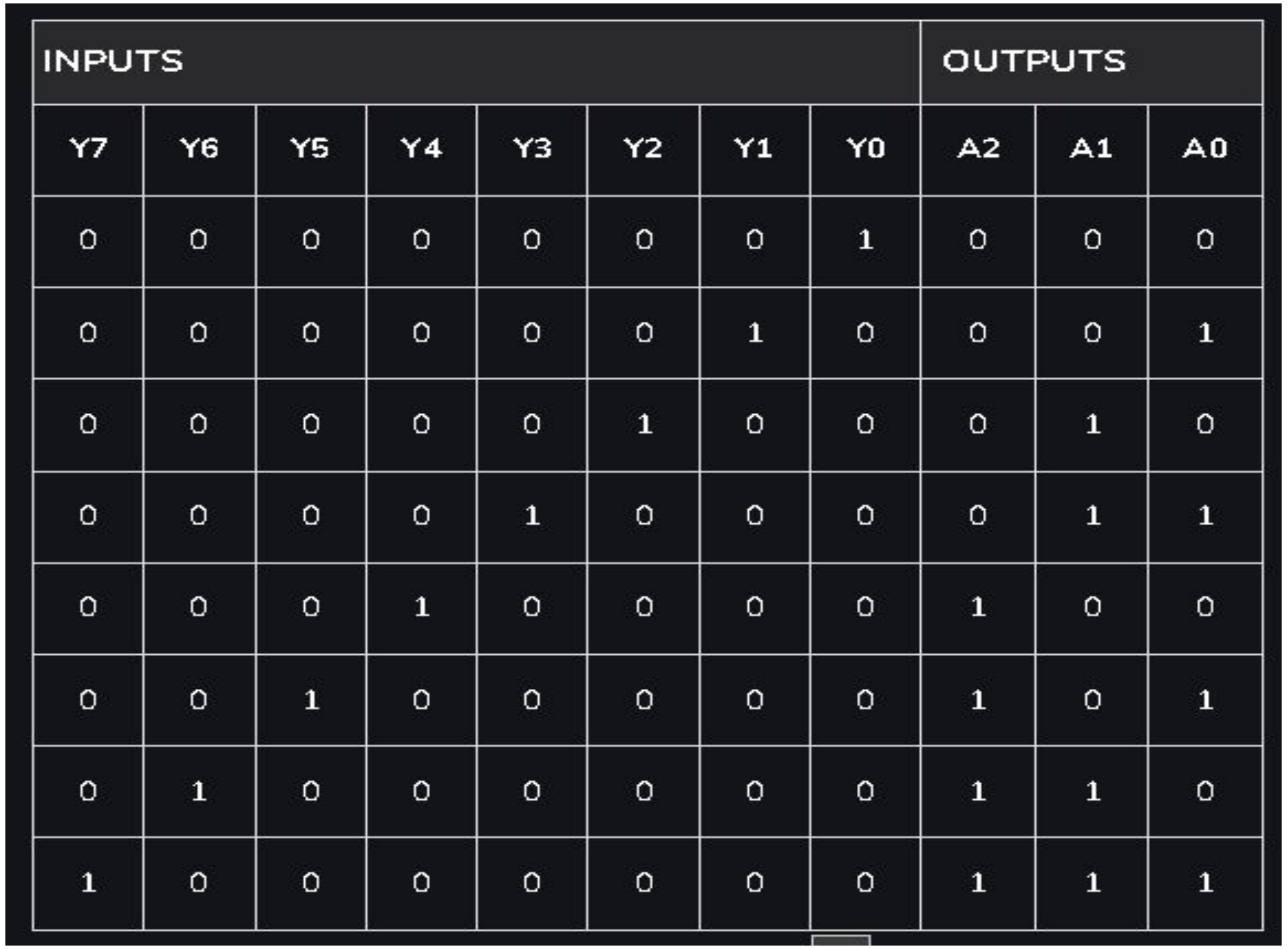
Implementation using OR Gate

Octal to Binary Encoder (8 to 3 Encoder)

- The 8 to 3 Encoder or octal to Binary encoder consists of **8 inputs**: Y7 to Y0 and **3 outputs**: A2, A1 & A0.
- Each input line corresponds to each octal digit and three outputs generate corresponding binary code.



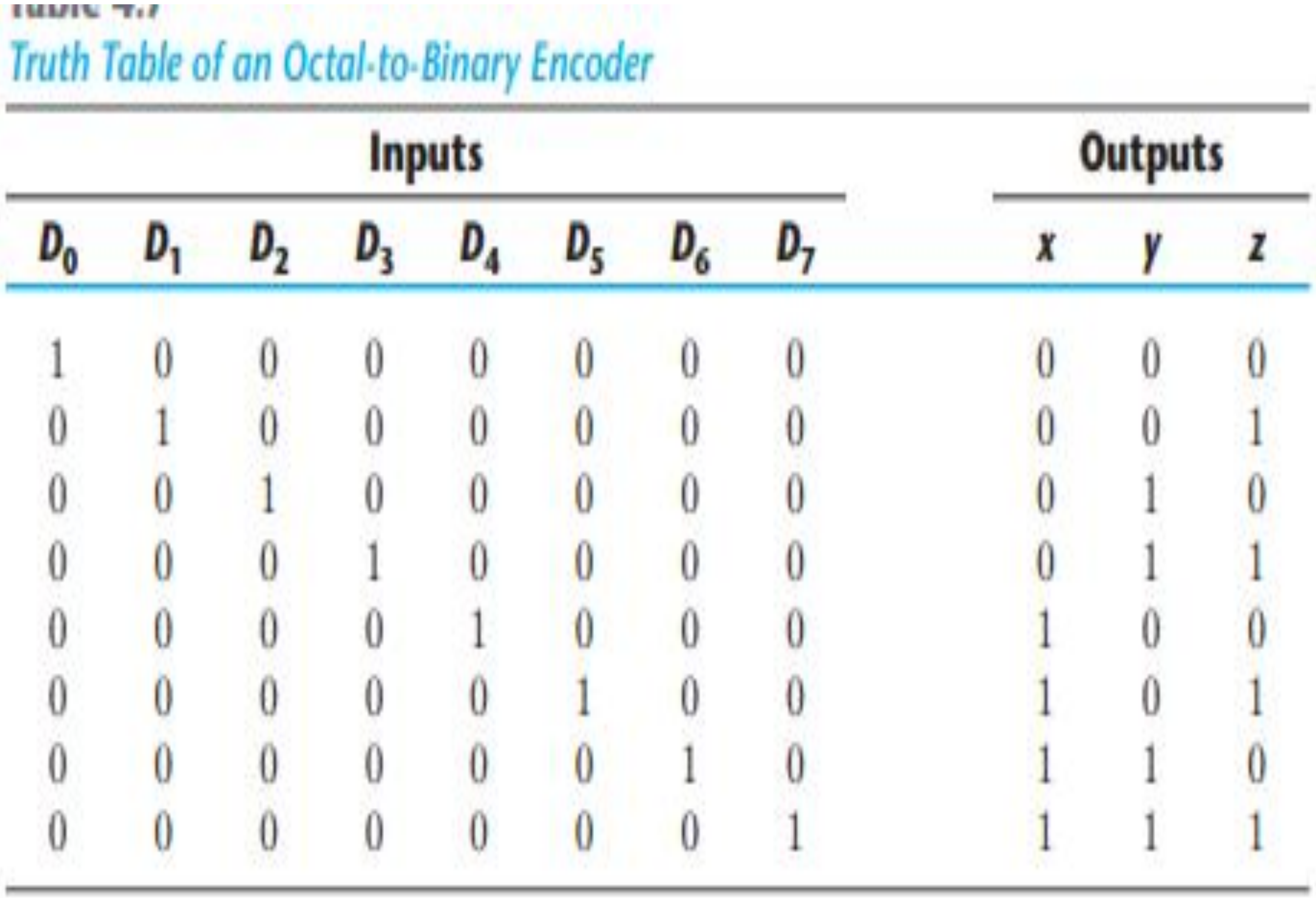
Octal to Binary Encoder (8 to 3 Encoder)



INPUTS								OUTPUTS		
Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	A2	A1	A0
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

Logical expression for A2, A1, and A0:

Octal-to-Binary encoder



Truth Table of an Octal-to-Binary Encoder

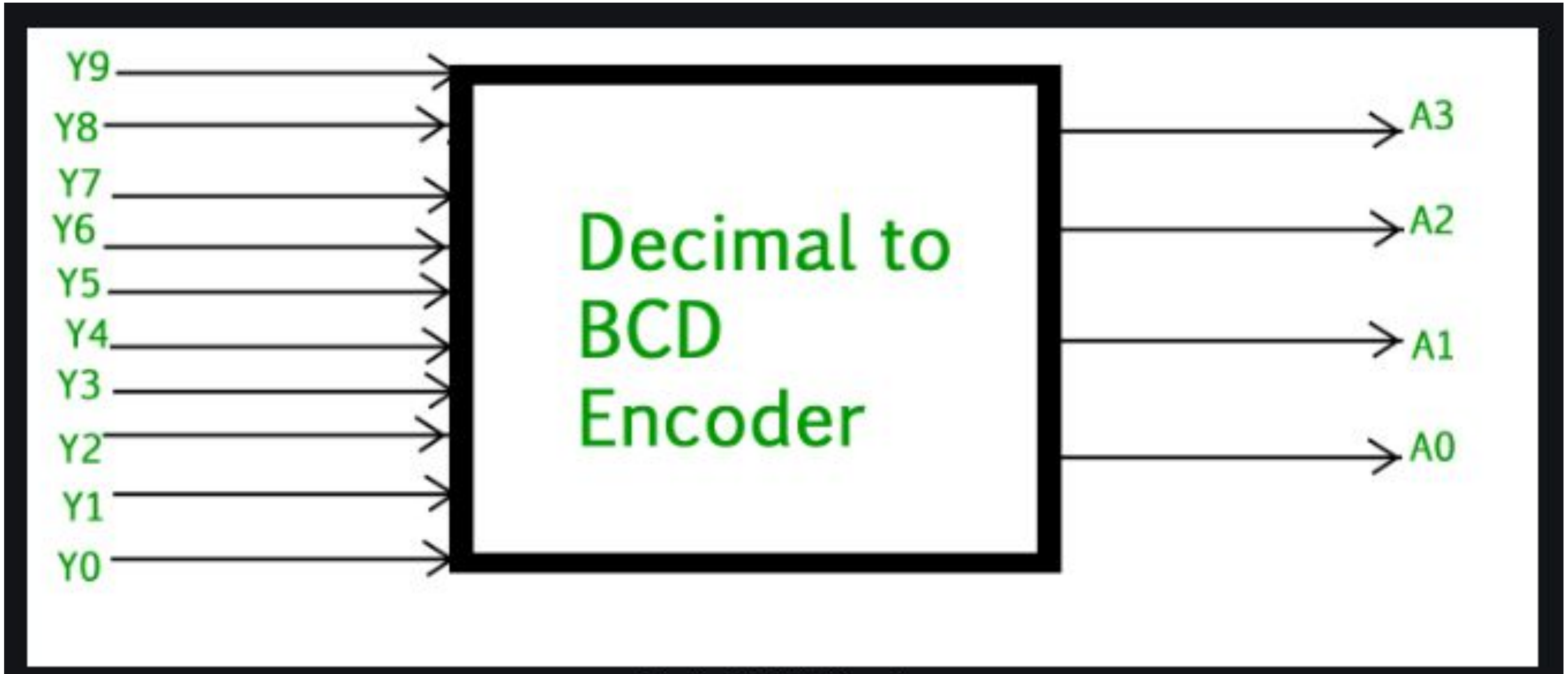
Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

$$z = D_1 + D_3 + D_5 + D_7$$

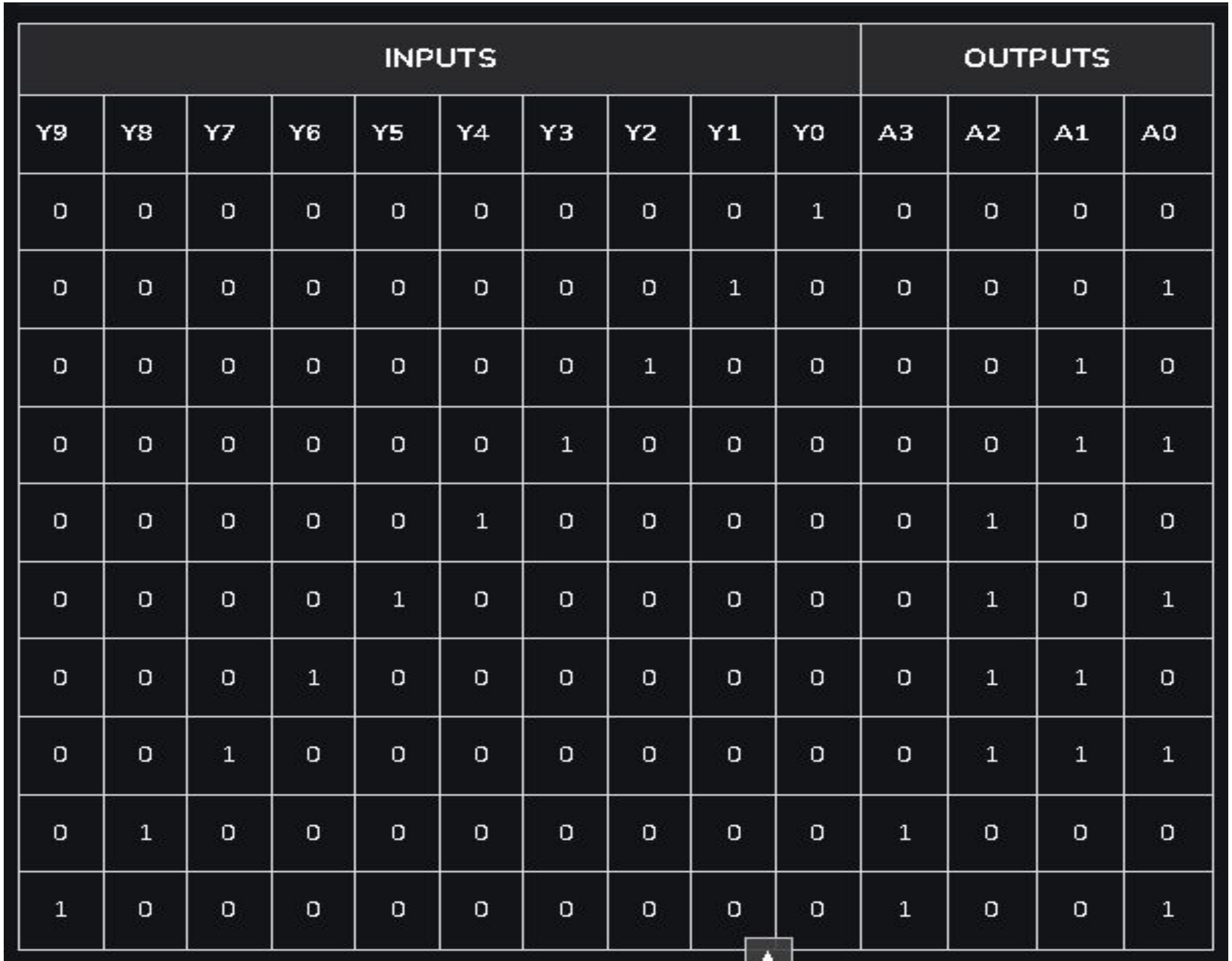
$$y = D_2 + D_3 + D_6 + D_7$$

$$x = D_4 + D_5 + D_6 + D_7$$

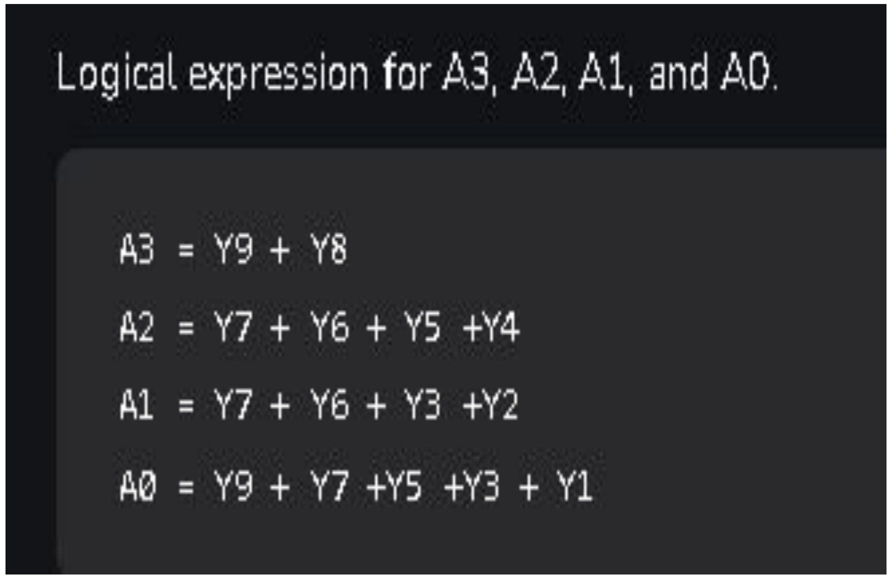
Decimal to BCD



Decimal to BCD Encoder



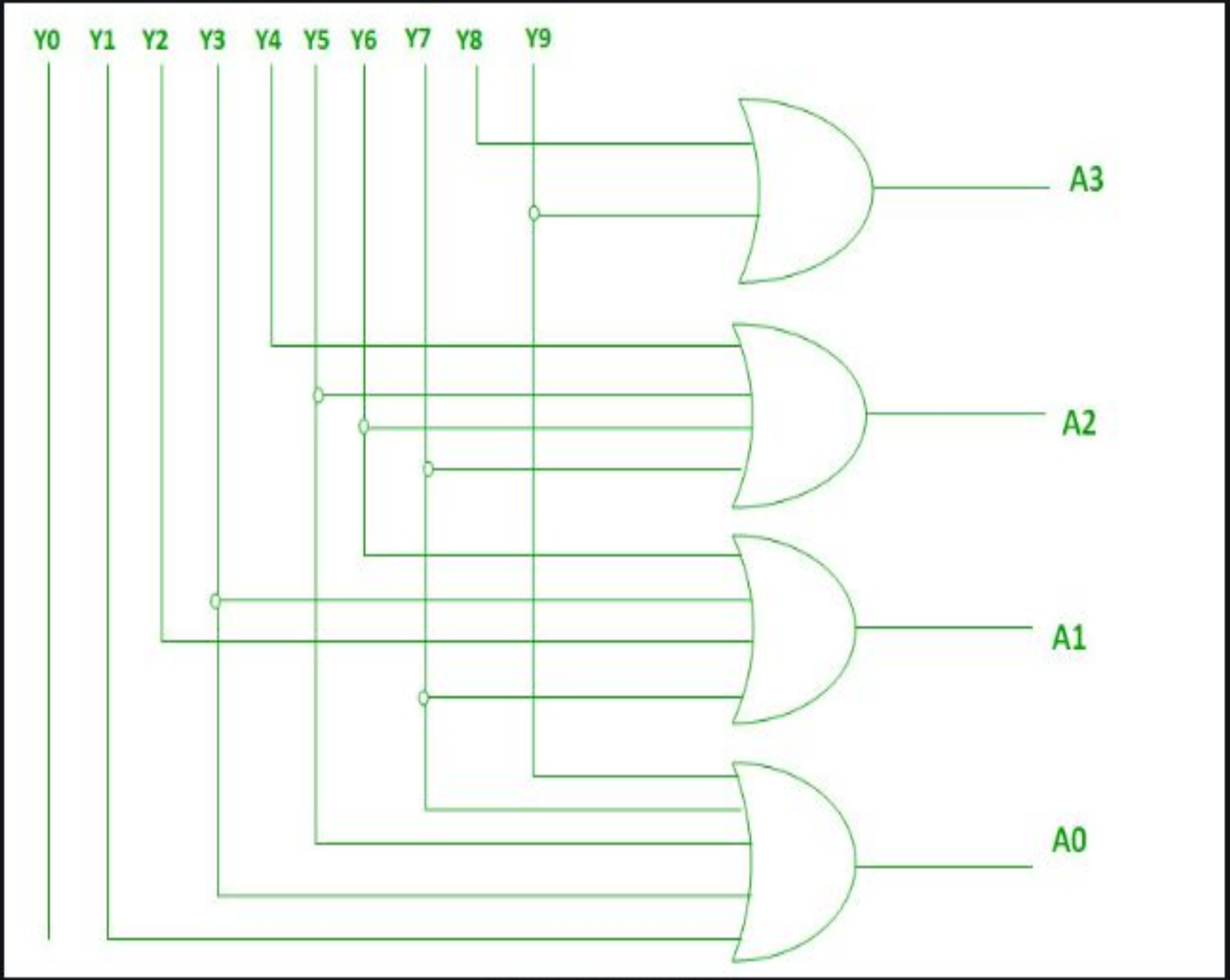
INPUTS										OUTPUTS			
Y9	Y8	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	A3	A2	A1	A0
0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	1	1	0
0	0	1	0	0	0	0	0	0	0	0	1	1	1
0	1	0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0	0	1	0	0	1



Logical expression for A3, A2, A1, and A0.

```
A3 = Y9 + Y8
A2 = Y7 + Y6 + Y5 +Y4
A1 = Y7 + Y6 + Y3 +Y2
A0 = Y9 + Y7 +Y5 +Y3 + Y1
```

The above two Boolean functions can be implemented using OR gates.



Implementation using OR Gate

Priority Encoders

- A priority encoder is an encoder circuit that includes the priority function.
- The operation of the priority encoder is such that if two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence.
- Consider a truth table.
- In addition to the two outputs x and y , the circuit has a third output designated by V ;
- this is a valid bit indicator that is set to 1 when one or more inputs are equal to 1.
- If all inputs are 0, there is no valid input and V is equal to 0.
- The other two outputs are not inspected when V equals 0 and are specified as don’t-care conditions.

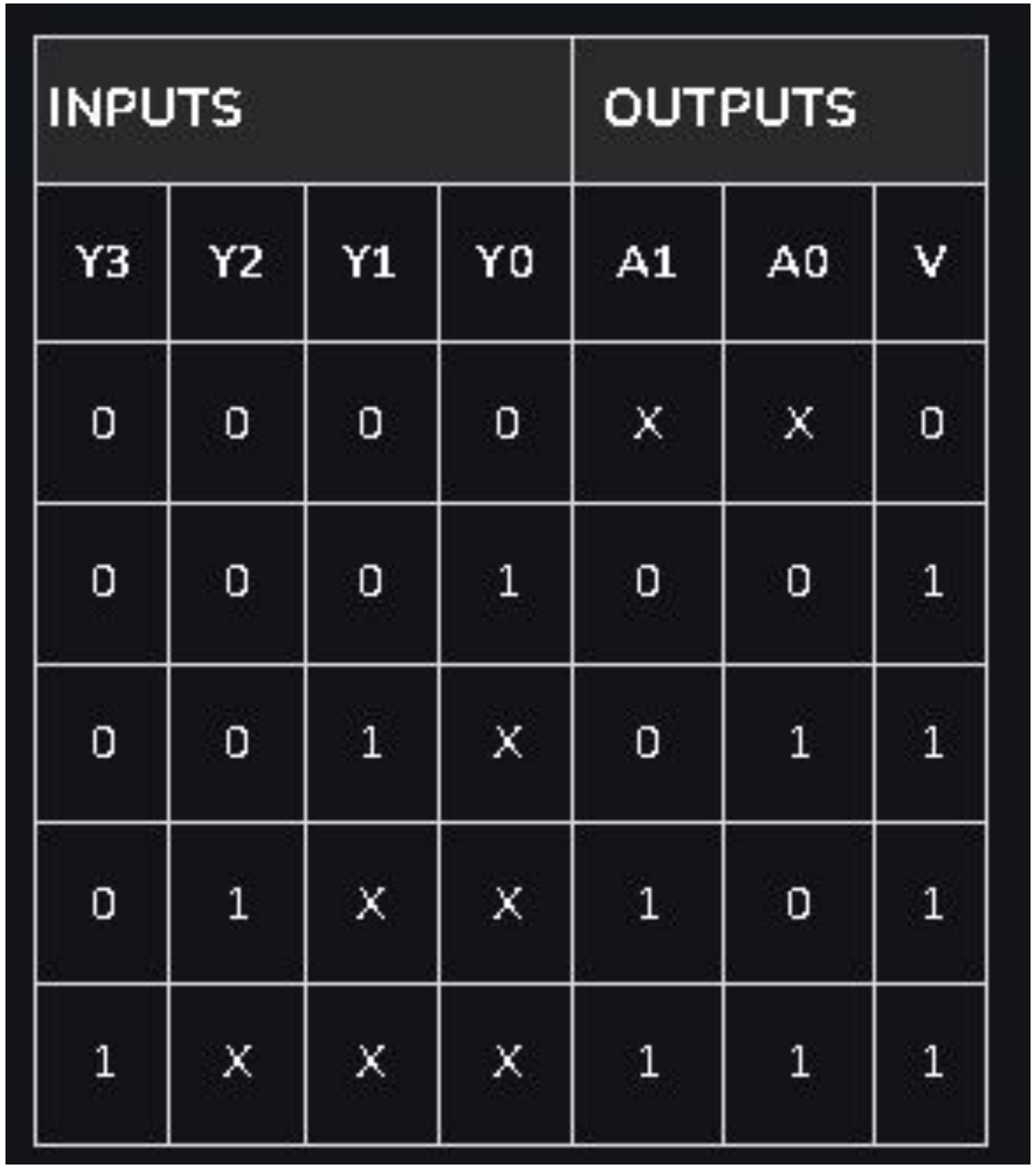
Priority Encoder:

A 4 to 2 priority encoder has **4 inputs**: Y3, Y2, Y1 & Y0, and **2 outputs**: A1 & A0.

Here, the input, Y3 has the **highest priority**, whereas the input, Y0 has the **lowest priority**.

In this case, even if more than one input is ‘1’ at the same time, the output will be the (binary) code corresponding to the input, which is having **higher priority**.

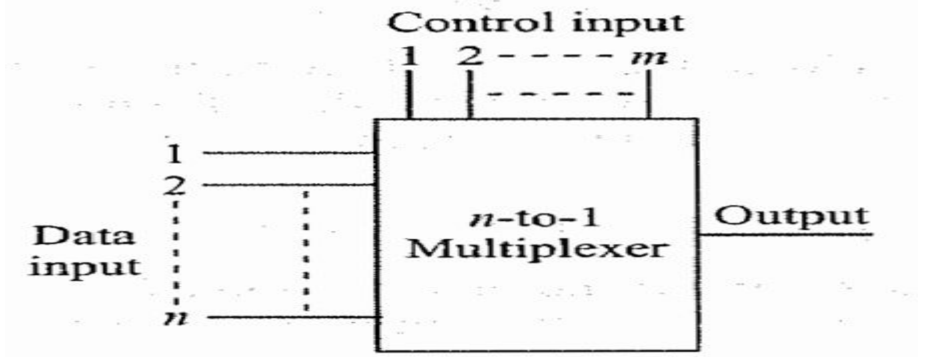
The truth table for the priority encoder is as follows.



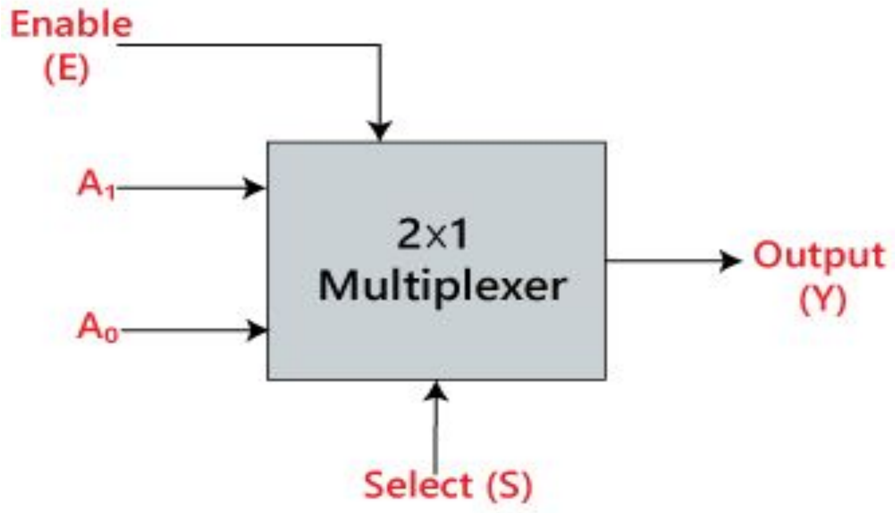
INPUTS				OUTPUTS		
Y3	Y2	Y1	Y0	A1	A0	V
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

Multiplexers

- A multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line.
- i.e., Binary information received from the input lines are directed to output lines based on the selection inputs.
- The selection of a particular input line is controlled by a set of selection lines.
- Normally, there are 2n input lines and n selection lines whose bit combinations determine which input is selected.
- A multiplexer is also treated as **Mux**.

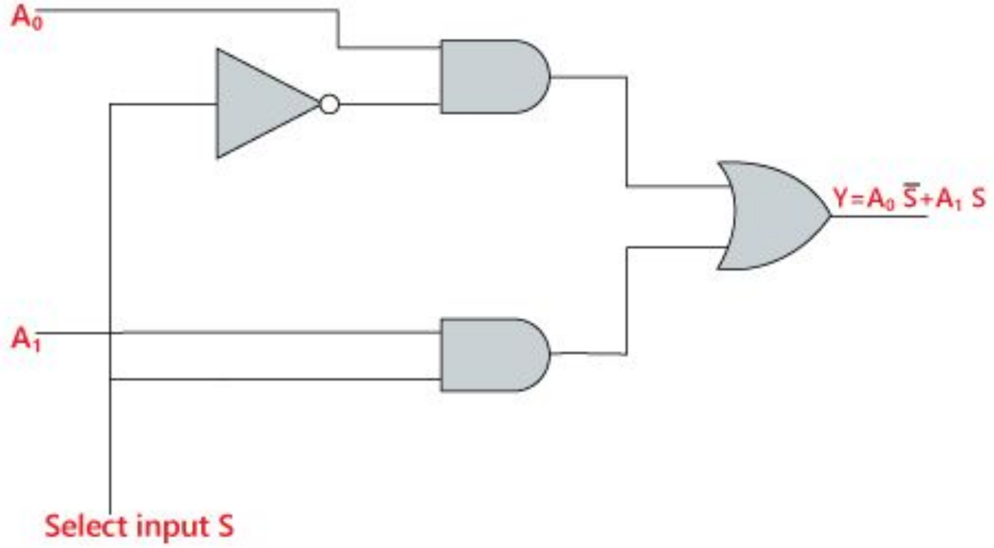


2×1 Multiplexer:



Truth Table:

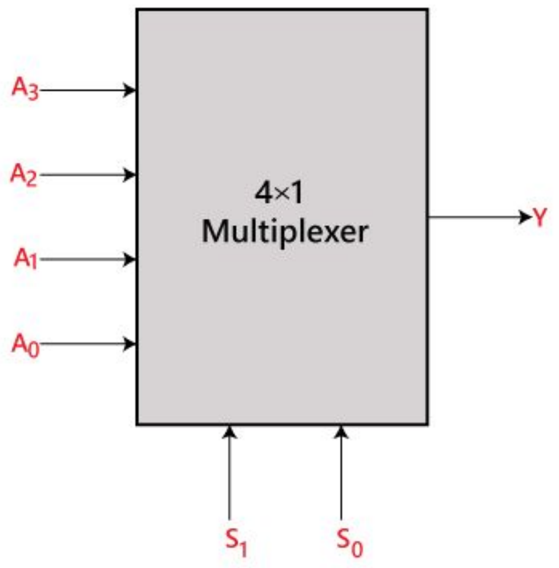
INPUTS	Output
S_0	Y
0	A_0
1	A_1



Logical Expression :

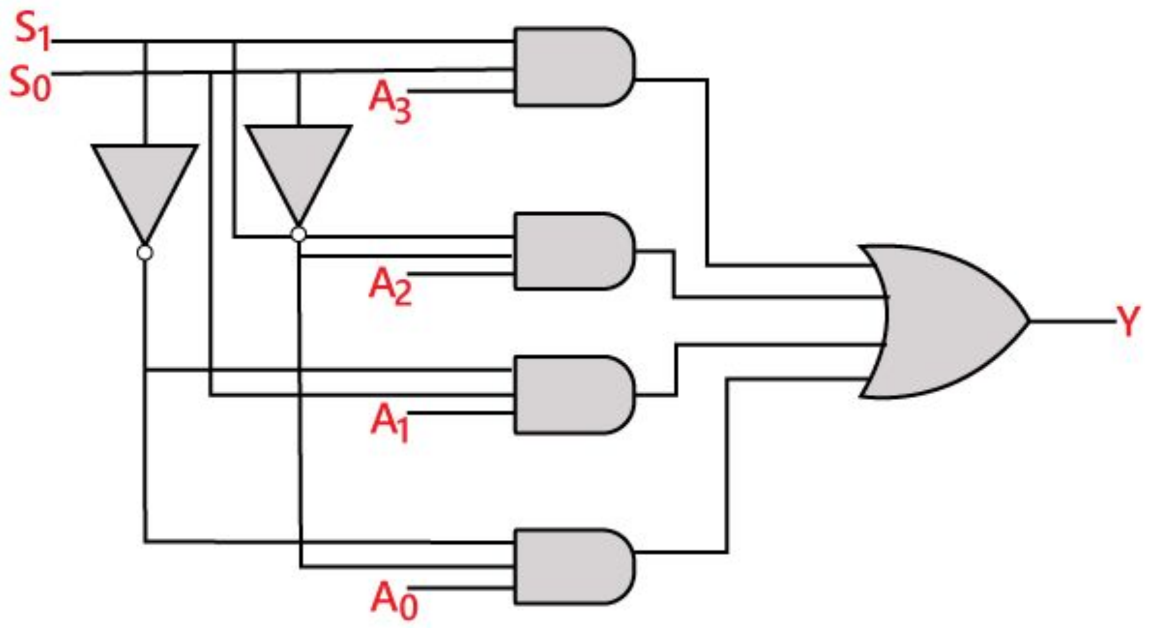
$$Y=S_0'.A_0+S_0.A_1$$

4×1 Multiplexer:



Truth Table:

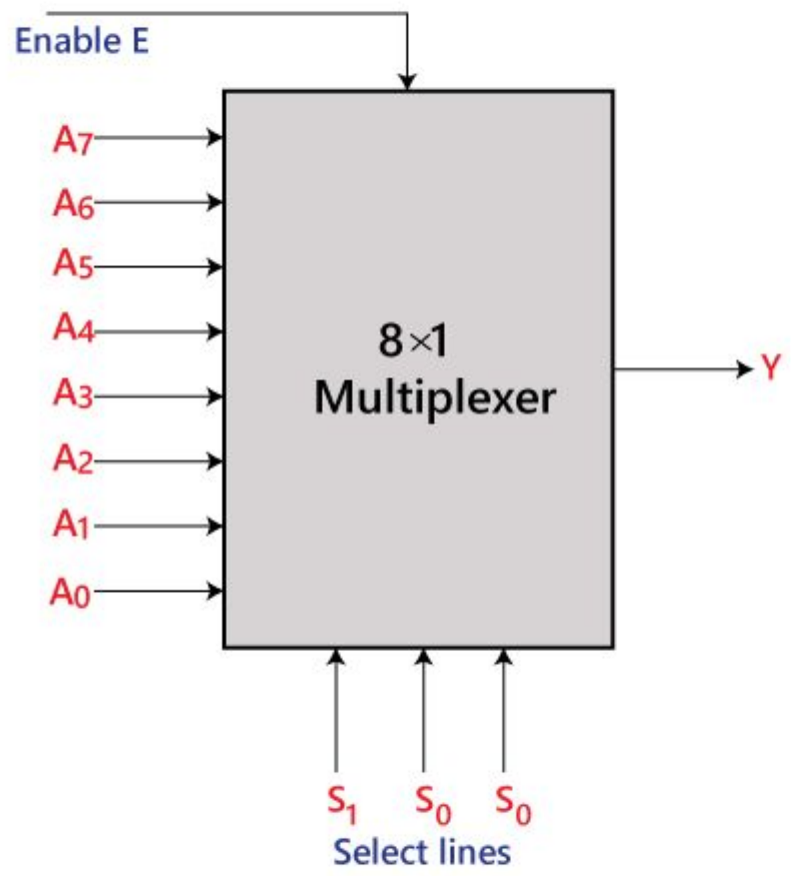
INPUTS		Output
S_1	S_0	Y
0	0	A_0
0	1	A_1
1	0	A_2
1	1	A_3



The logical expression of the term Y is as follows:

$$Y=S_1' S_0' A_0+S_1' S_0 A_1+S_1 S_0' A_2+S_1 S_0 A_3$$

8 to 1 Multiplexer

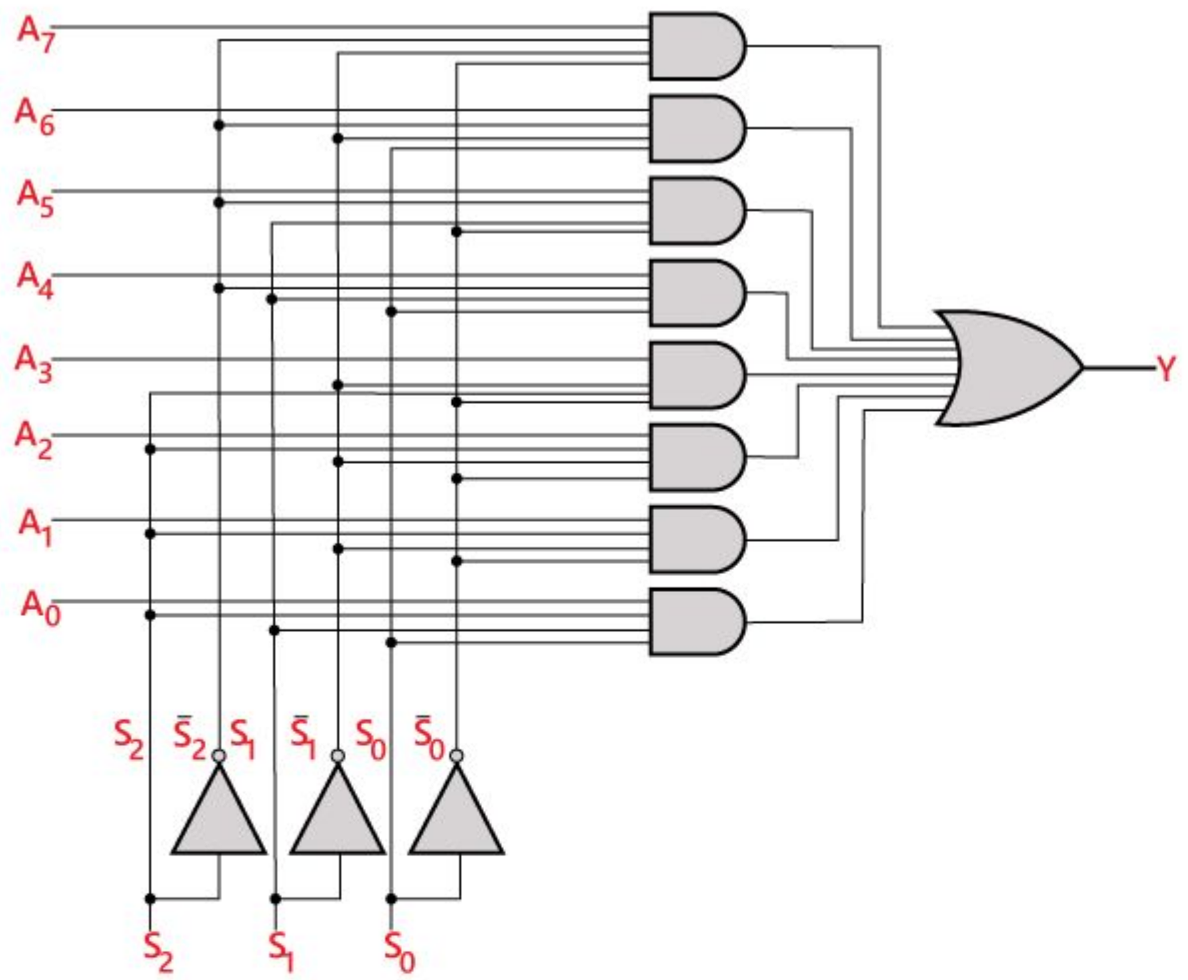


Truth Table:

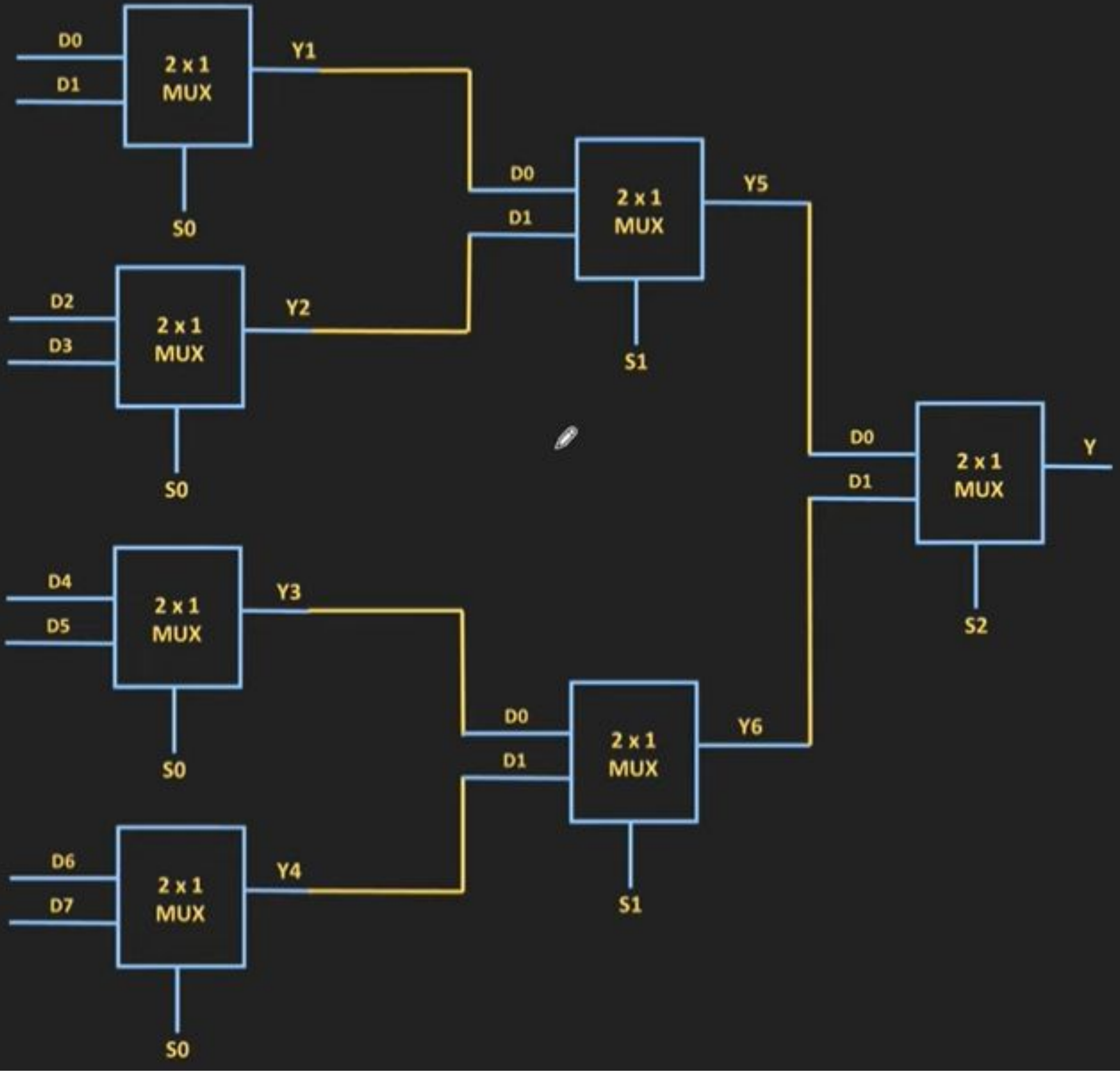
INPUTS			Output
S_2	S_1	S_0	Y
0	0	0	A_0
0	0	1	A_1
0	1	0	A_2
0	1	1	A_3
1	0	0	A_4
1	0	1	A_5
1	1	0	A_6
1	1	1	A_7

- **The logical expression of the term Y is as follows:**

$Y=S_0'.S_1'.S_2'.A_0+S_0.S_1'.S_2'.A_1+S_0'.S_1.S_2'.A_2+S_0.S_1.S_2'.A_3+S_0'.S_1'.S_2\ A_4+S_0.S_1'.S_2\ A_5+S_0'.S_1.S_2\ .A_6+S_0.S_1.S_3.A_7$

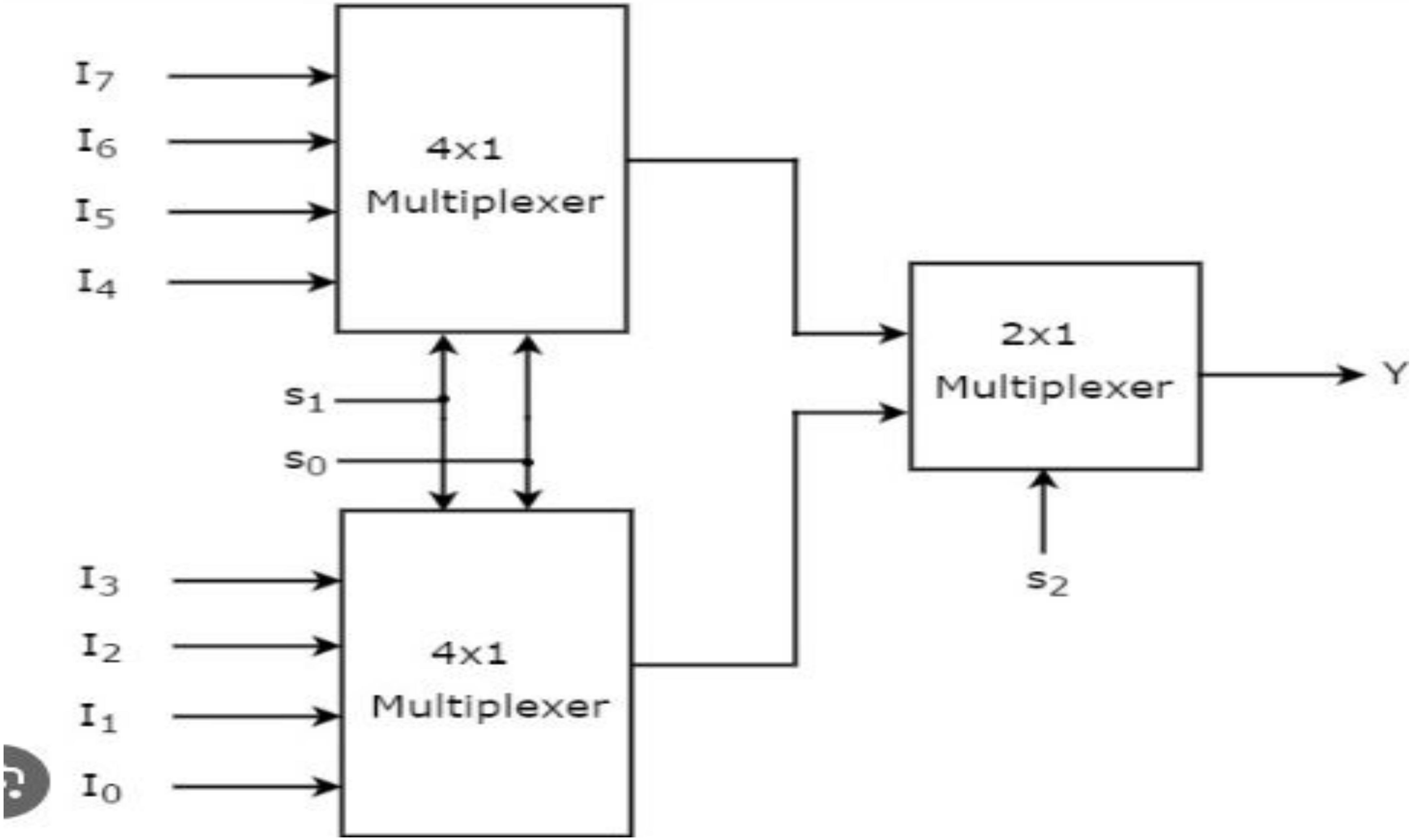


8 x 1 Multiplexer using 2 x 1 Multiplexer

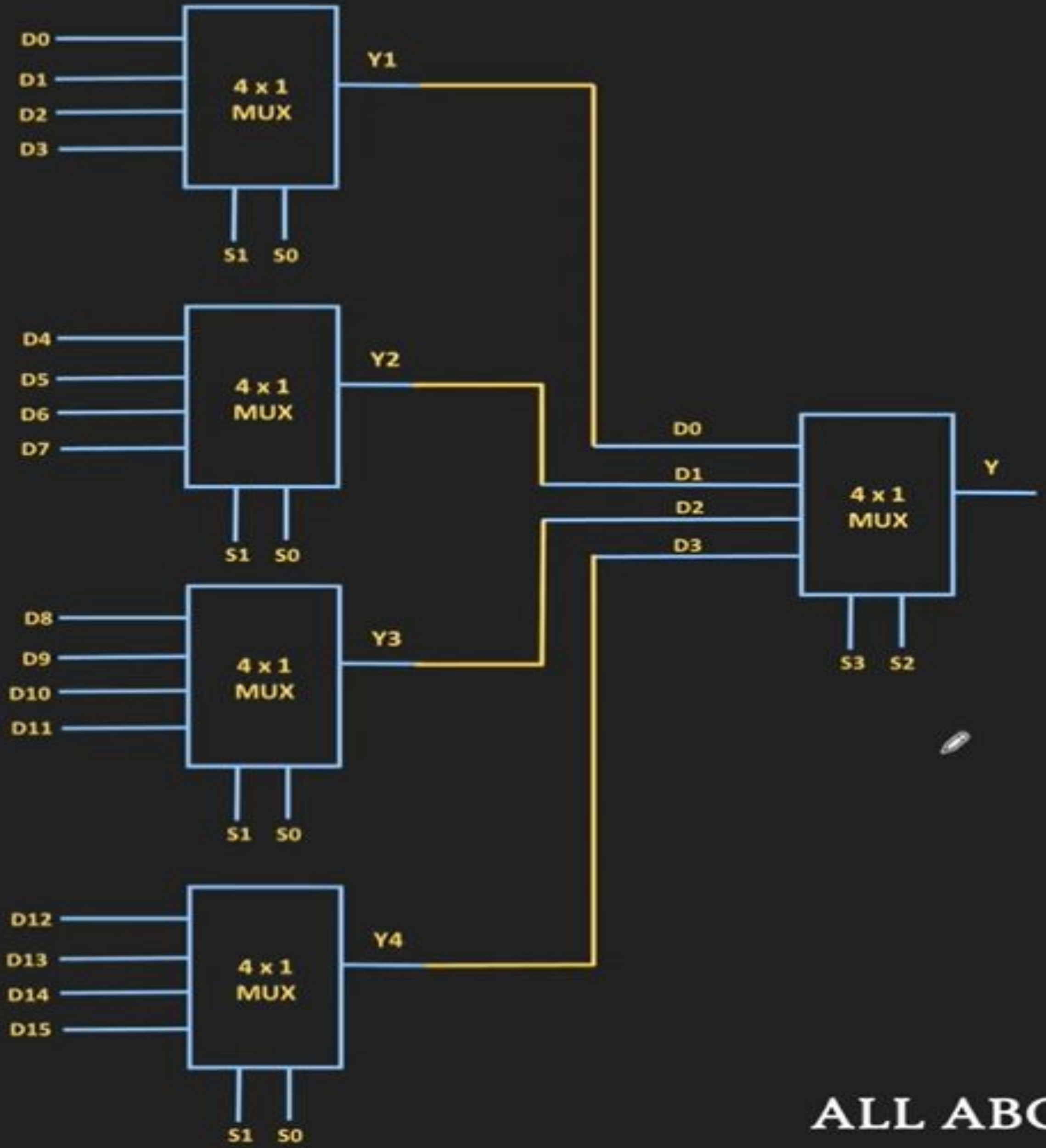


Truth Table

S2	S1	S0	Y
0	0	0	D0
0	0	1	D1
0	1	0	D2
0	1	1	D3
1	0	0	D4
1	0	1	D5
1	1	0	D6
1	1	1	D7


I_7
I_6
I_5
I_4
4x1 Multiplexer
s_1
s_0
I_3
I_2
I_1
I_0
4x1 Multiplexer
2x1 Multiplexer
Y
s_2

16 x 1 Multiplexer using 4 x 1 Multiplexer



ALL ABOUT ELECTRONICS

Boolean Function Implementation

- A Boolean function with n variables can be implemented with a MUX with

(n-1) selection lines.

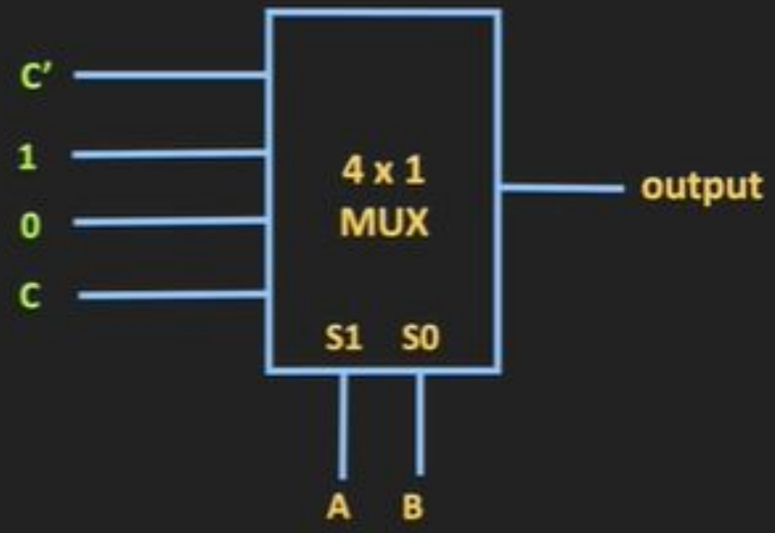
- e.g., 3 variable Boolean function can be implemented with a 4 X 1 MUX with 2 selection lines.
- During the implementation , starting from MSB , the first (n – 1) variables are connected to the selection lines and the last variable is connected to the data lines.

Implementation of Boolean Function using MUX

$F(A, B, C) = \sum m(0, 2, 3, 7)$

Truth Table

A	B	C	F	
0	0	0	1	F = C’
0	0	1	0	
0	1	0	1	F = 1
0	1	1	1	
1	0	0	0	F = 0
1	0	1	0	
1	1	0	0	F = C
1	1	1	1	

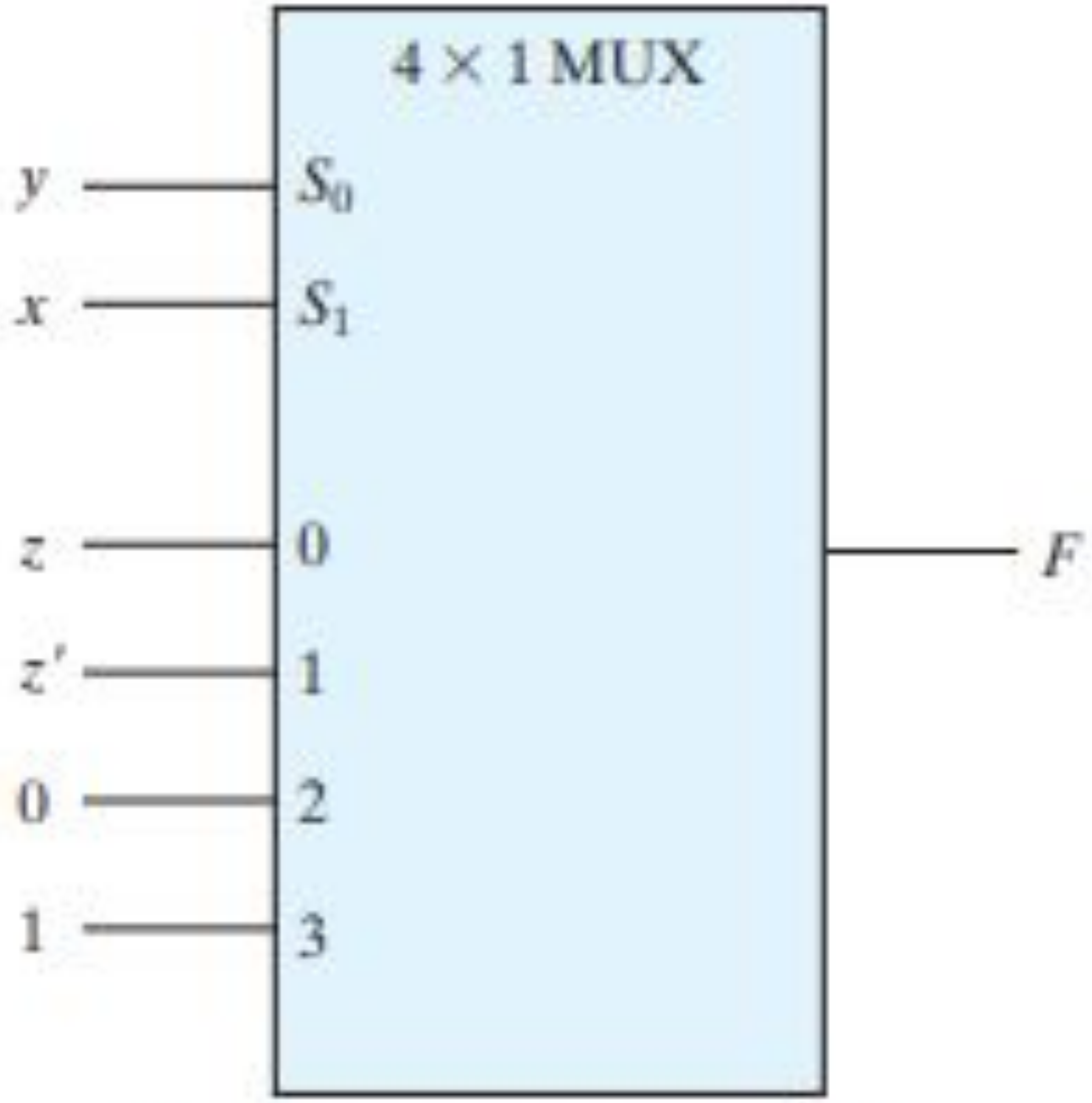


Problems

1. F (x, y, z) = (1, 2, 6, 7)

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

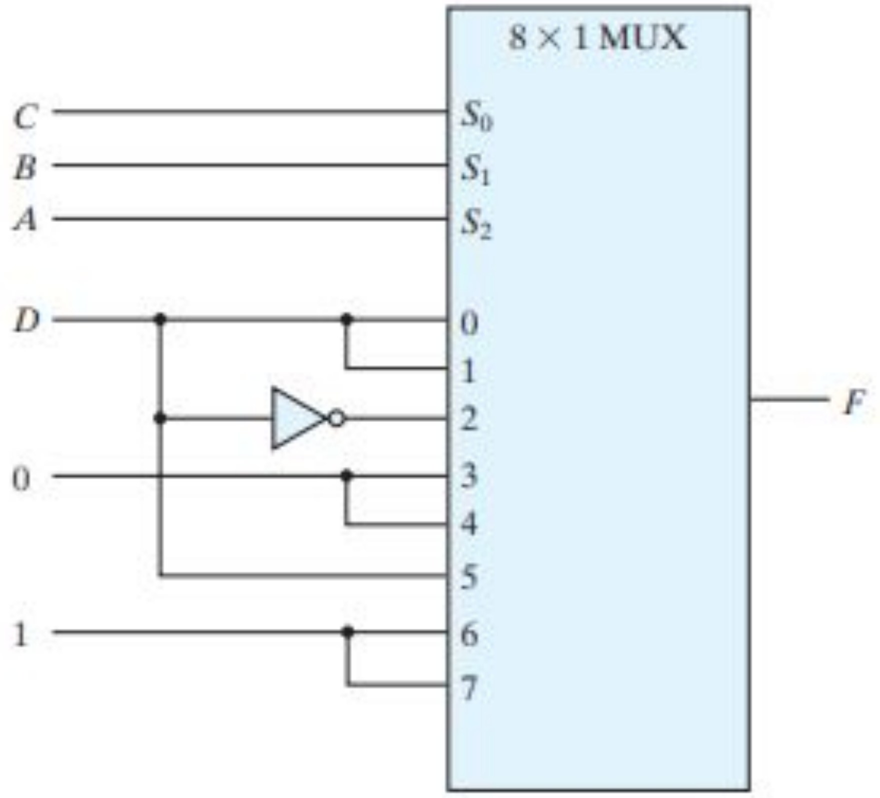
(a) Truth table



(b) Multiplexer implementation

2. F (A, B, C, D) = (1, 3, 4, 11, 12, 13, 14, 15)

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



Sequential Circuits

- A sequential circuit refers to a special type of circuit, whose output dependent on the combinations of both present inputs as well as the previous output.

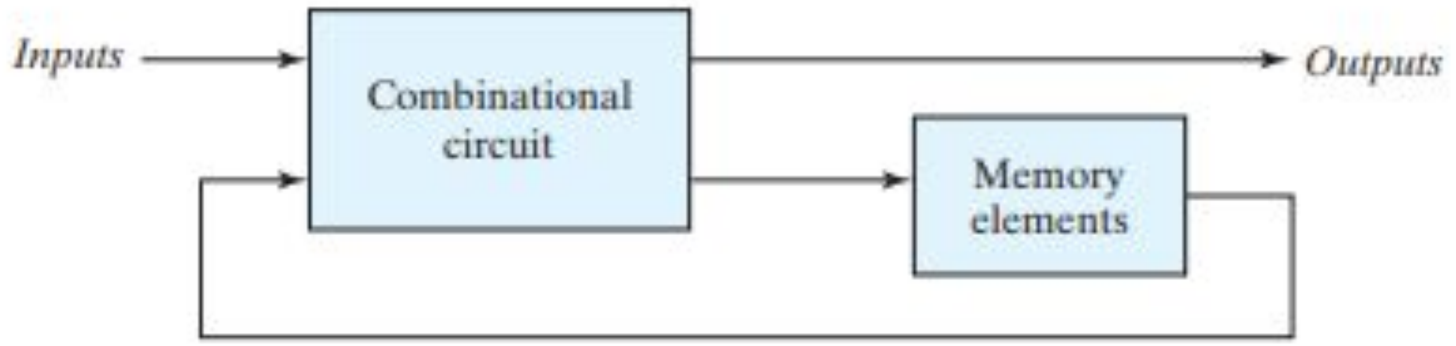


FIGURE 5.1
Block diagram of sequential circuit

- It consists of a combinational circuit to which storage elements are connected to form a feedback path.
- The storage elements are devices capable of storing binary information. The binary information stored in these elements at any given time defines the state of the sequential circuit at that time.
- The sequential circuit receives binary information from external inputs that, together with the present state of the storage elements, determine the binary value of the outputs.
- These external inputs also determine the condition for changing the state in the storage elements.

- The block diagram demonstrates that the outputs in a sequential circuit are a function not only of the inputs, but also of the present state of the storage elements.
- The next state of the storage elements is also a function of external inputs and the present state.
- Thus, a sequential circuit is specified by a time sequence of inputs, outputs, and internal states .

Differences between Combinational circuit and Sequential Circuit

Combinational circuit	Sequential Circuit
Output present on current input state	Output present on previous output and present input
Don’t have a memory element	It has a memory elements
Clock signals are not necessary	Clock signals are present
Adders, subtractors , Multiplexers	Flipflops , registers , counters.

Types of Sequential Circuits

- **Asynchronous sequential circuits:**

- The clock signals are not used by the **Asynchronous sequential circuits**.
- The asynchronous circuits do not use clock pulses.
- The internal state is changed when the input variable is changed.

- **Synchronous sequential circuits :**

- Synchronization of the memory element's state is done by the clock signal.
- The output is stored in either flip-flops or latches(memory devices).
- The synchronization of the outputs is done with either only negative edges of the clock signal or only positive edges.

Clock Pulse

- The clock pulses are distributed throughout the system in such a way that storage elements are affected only with the arrival of each pulse.
- In practice, the clock pulses determine when computational activity will occur within the circuit.
- For example, a circuit that is to add and store two binary numbers would compute their sum from the values of the numbers and store the sum at the occurrence of a clock pulse.

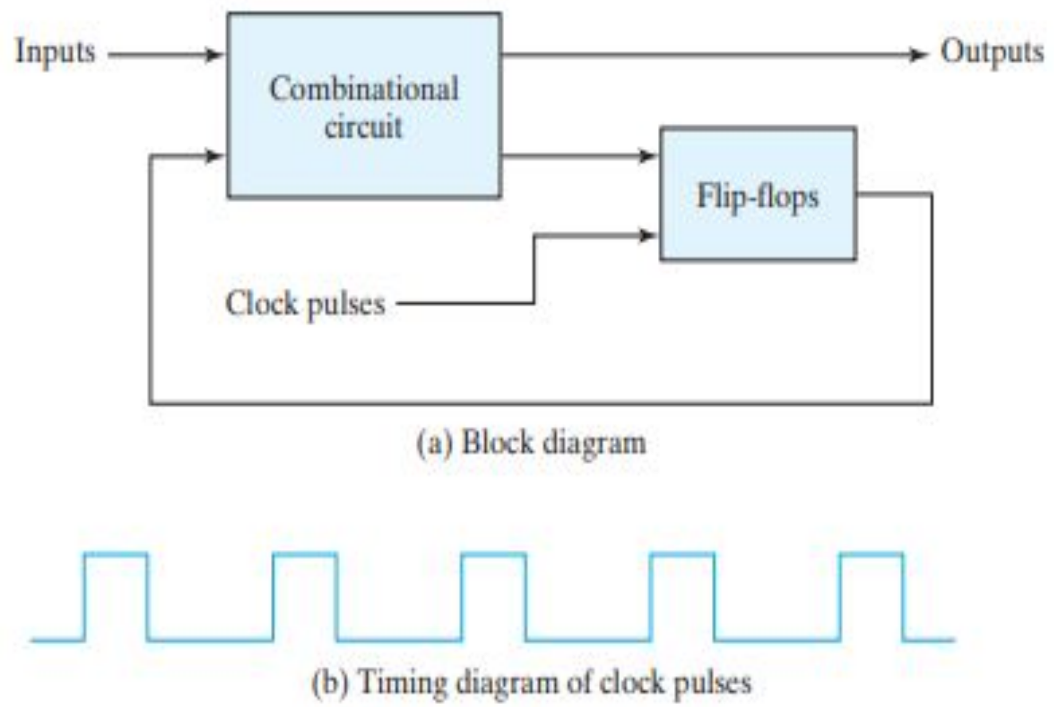


FIGURE 5.2
Synchronous clocked sequential circuit

Storage elements

Digital circuits that store a single bit of information

- Latch
- Flip flops

Differences between Latches and Flipflops

Latches	Flipflops
Latches continuously checks its input and changes its output correspondingly	Flipflop continuously checks its inputs and changes its output based on the clock pulse
Latch is a level triggered device	Flipflop is Edge triggered device.
Bistable Asynchronous device	Bistable synchronous device

Storage elements : Latches

- Latches are digital circuits that store a single bit of information and hold its value until it is updated by new input signals.
- They are used in digital systems as temporary storage elements to store binary information.
- Latches can be implemented using various digital logic gates, such as AND, OR, NOT, NAND, and NOR gates.
- **There are two types of latches:**

1. S-R (Set-Reset) Latches:
2. D Latches

- **S-R (Set-Reset) Latches:**

S-R latches are the simplest form of latches and are implemented using two inputs: S (Set) and R (Reset).

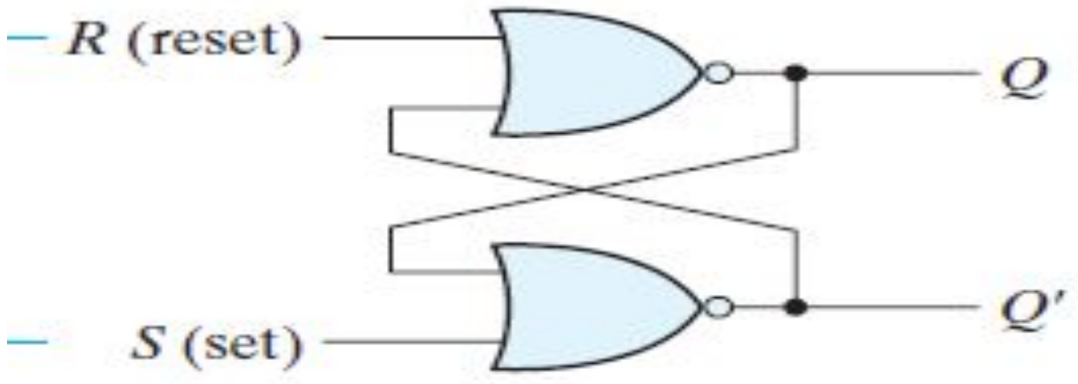
The S input sets the output to 1, while the R input resets the output to 0. When both S and R are at 1, the latch is said to be in an “undefined” state.

Two types of S-R Latch

1. NOR S-R Latch and
2. NAND S-R Latch

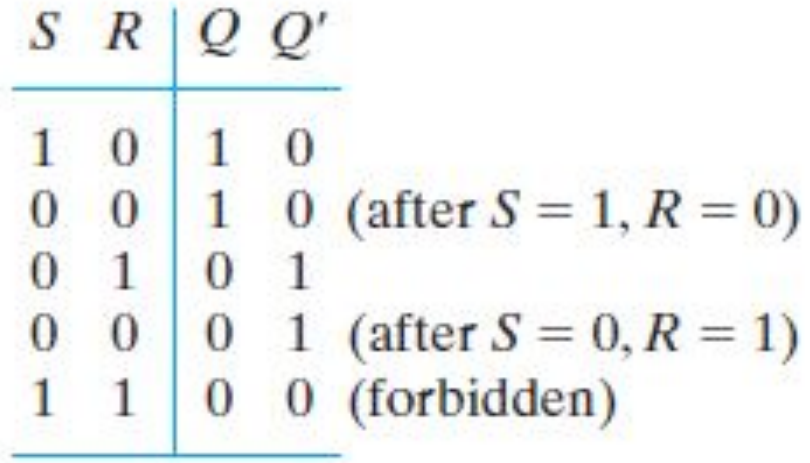
SR latch with NOR gates

S	R	Q	Q’
0	0	Memory	
0	1	0	1
1	0	1	0
1	1	Forbidden state	



(a) Logic diagram

A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

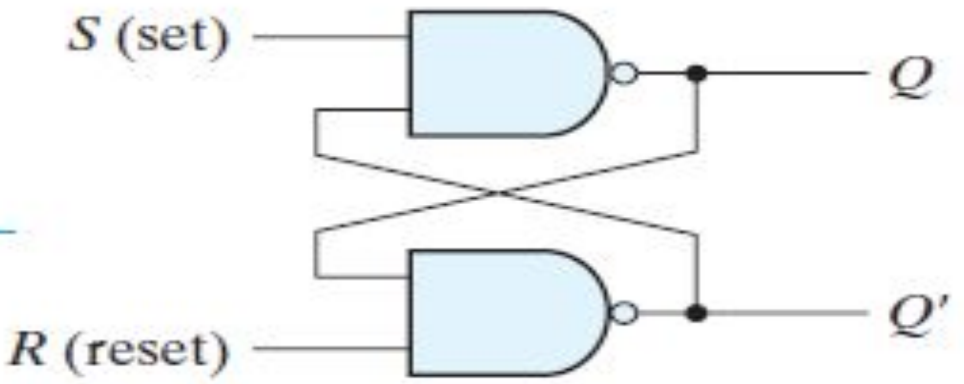


S	R	Q	Q'	
1	0	1	0	
0	0	1	0	(after $S = 1, R = 0$)
0	1	0	1	
0	0	0	1	(after $S = 0, R = 1$)
1	1	0	0	(forbidden)

(b) Function table

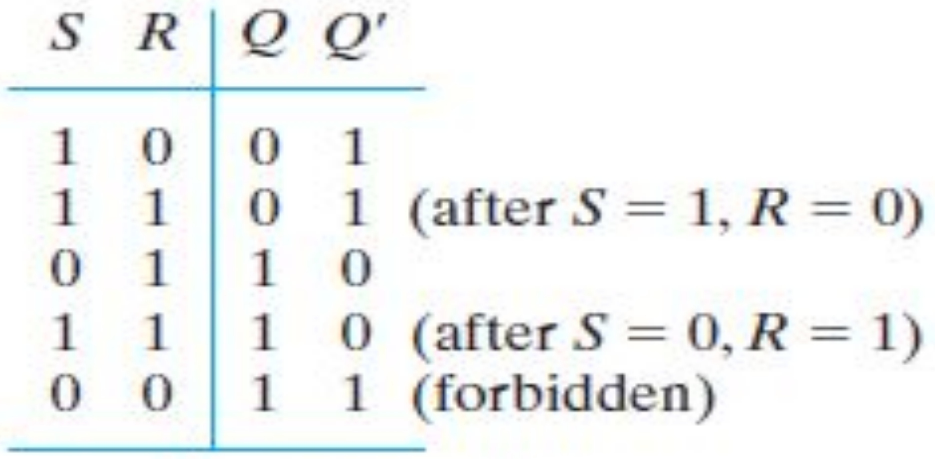
SR latch with NAND gates

S	R	Q	Q’
0	0	Not used state	
0	1	1	0
1	0	0	1
1	1	Memory state	



(a) Logic diagram

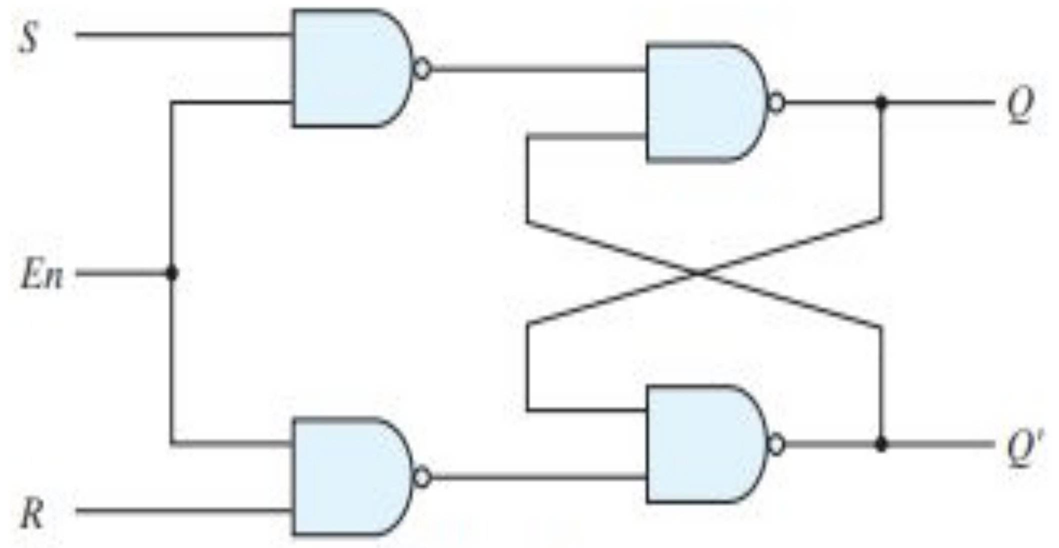
A	B	NAND
0	0	1
0	1	1
1	0	1
1	1	0



S	R	Q	Q'	
1	0	0	1	
1	1	0	1	(after $S = 1, R = 0$)
0	1	1	0	
1	1	1	0	(after $S = 0, R = 1$)
0	0	1	1	(forbidden)

(b) Function table

SR latch with Enable input



(a) Logic diagram

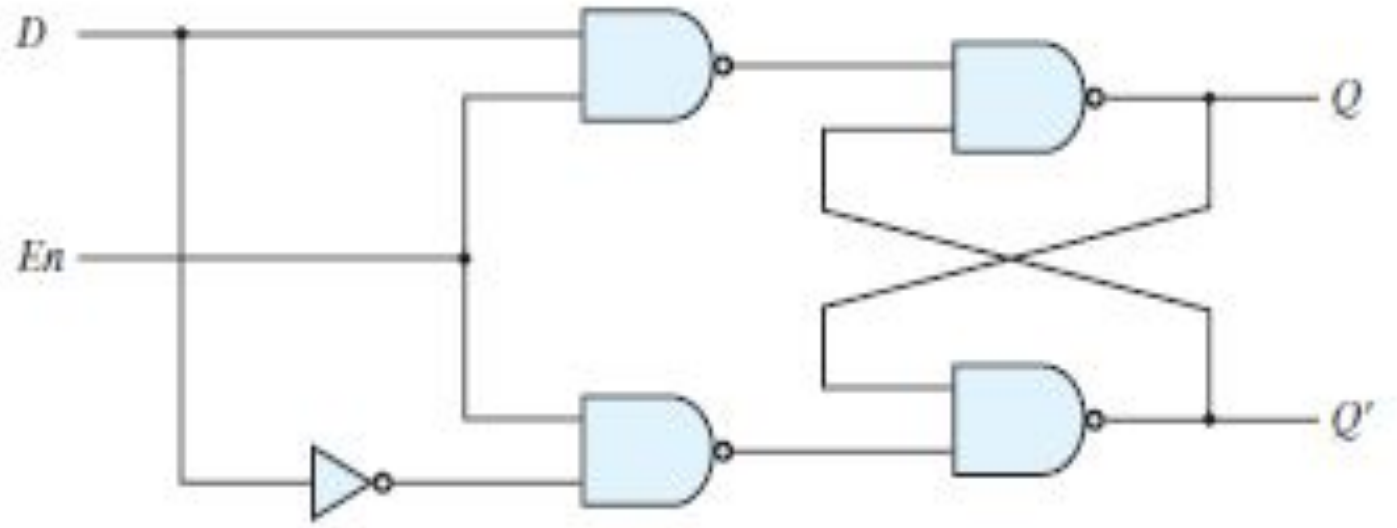
FIGURE 5.5
SR latch with control input

En	*S*	*R*	Next state of *Q*
0	X	X	No change
1	0	0	No change
1	0	1	$Q = 0$; reset state
1	1	0	$Q = 1$; set state
1	1	1	Indeterminate

(b) Function table

D (Data) Latches:

- One way to eliminate the undesirable condition of the indeterminate state in the *SR* latch is to ensure that inputs *S* and *R* are never equal to 1 at the same time. This is done in the *D* latch,



(a) Logic diagram

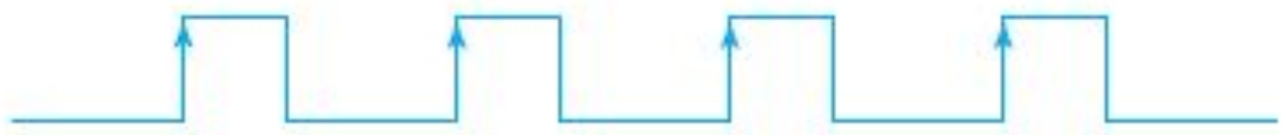
En *D*	Next state of *Q*
0 X	No change
1 0	$Q = 0$; reset state
1 1	$Q = 1$; set state

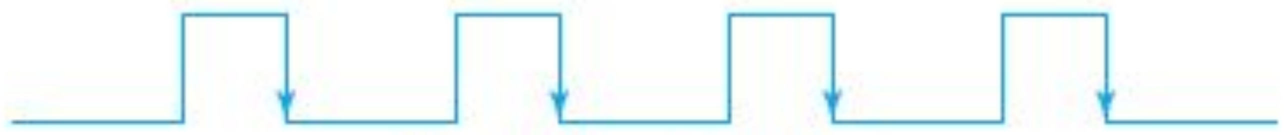
(b) Function table

FIGURE 5.6
D latch

Storage Element : Flip-Flops

- Flip-flop circuits are constructed in such a way as to make them operate properly when they are part of a sequential circuit that employs a common clock.
- A clock pulse goes through two transitions: from 0 to 1 and the return from 1 to 0.


(b) Positive-edge response


(c) Negative-edge response

Edge-Triggered *D* Flip-Flop

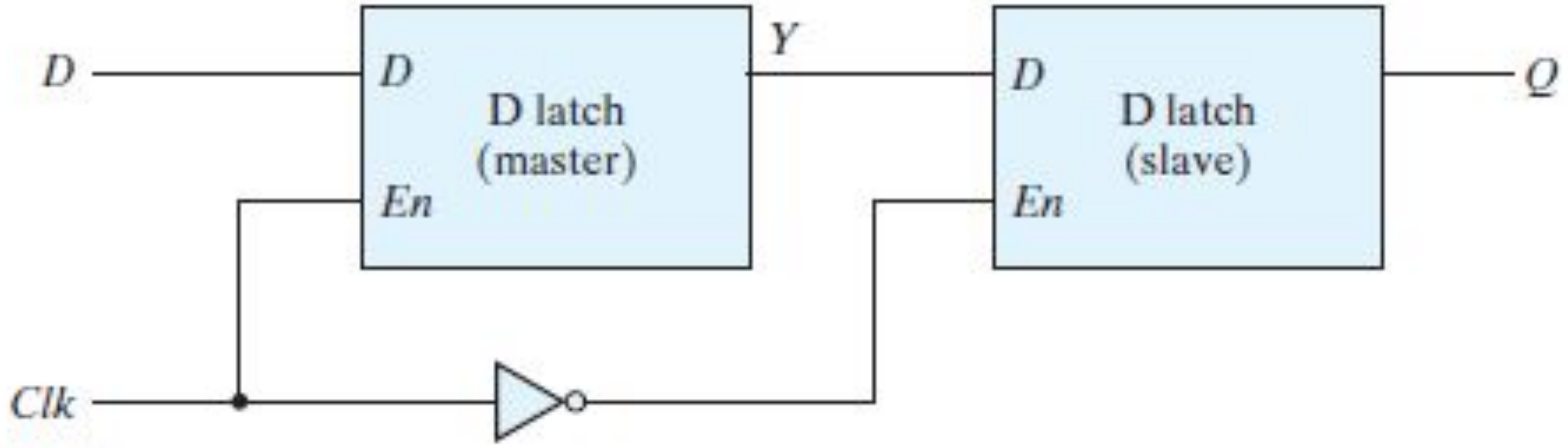


FIGURE 5.9
Master–slave *D* flip-flop

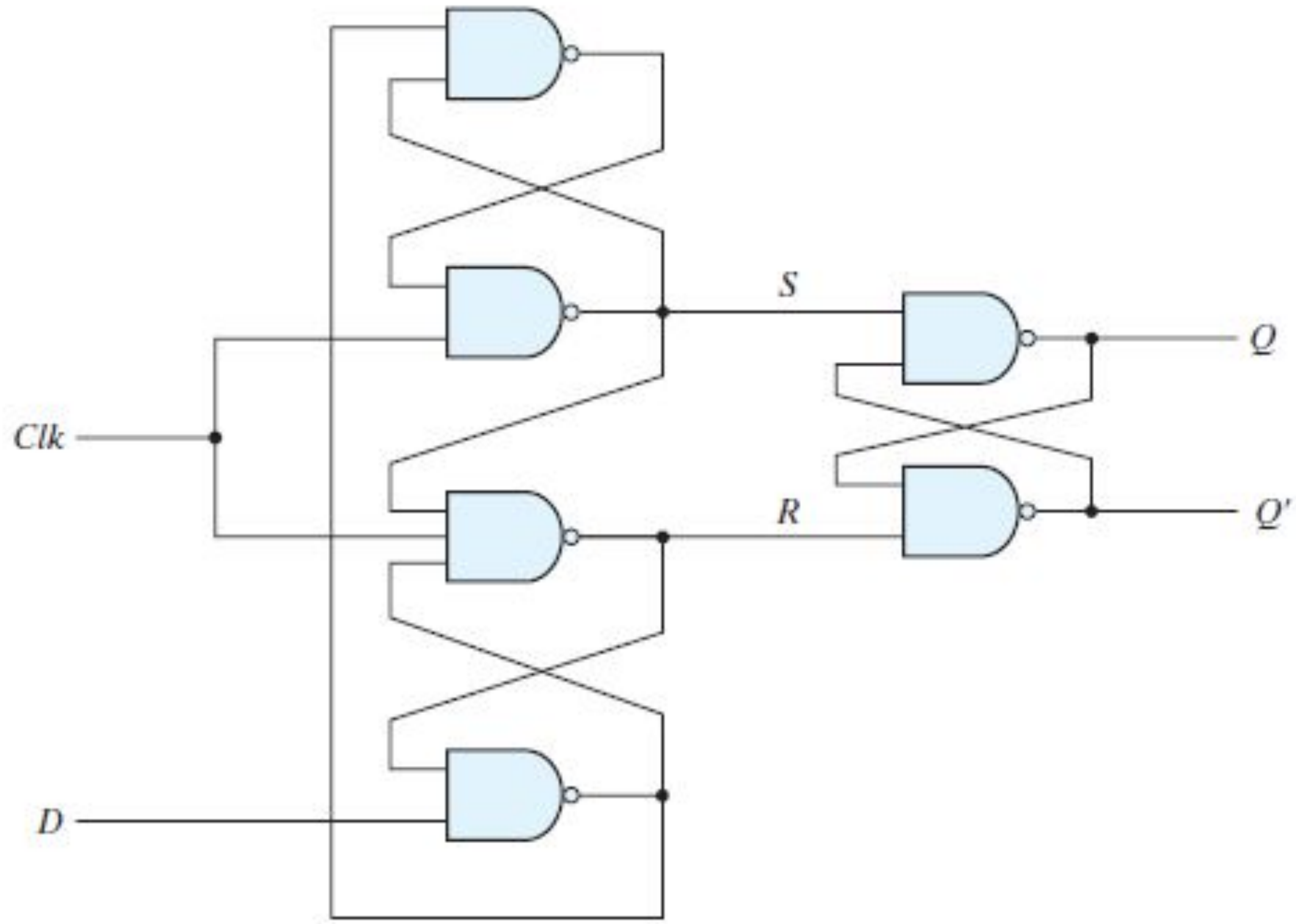
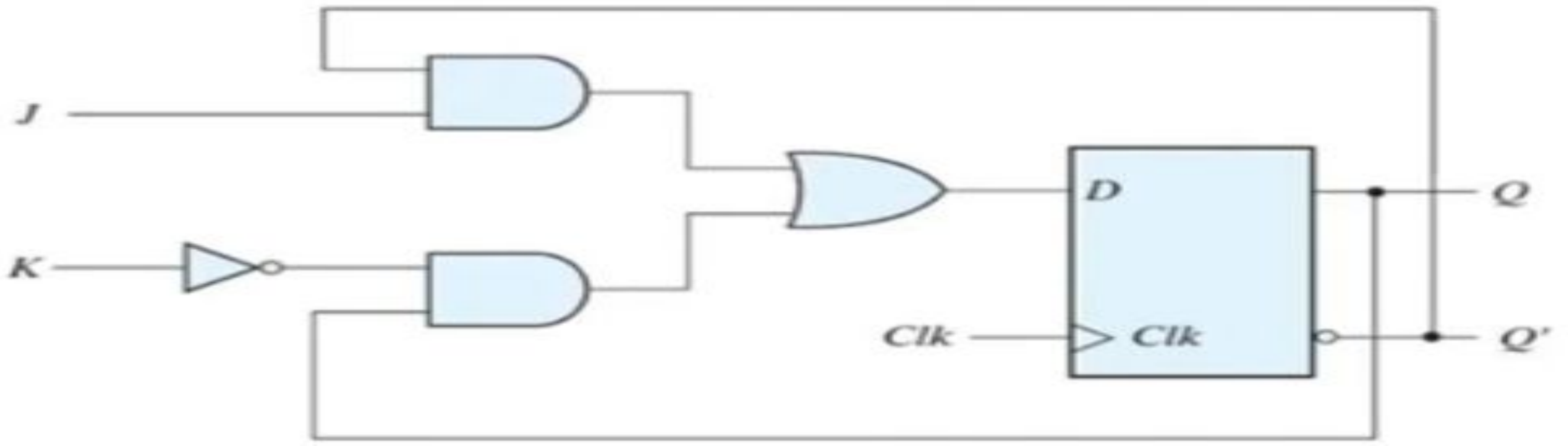


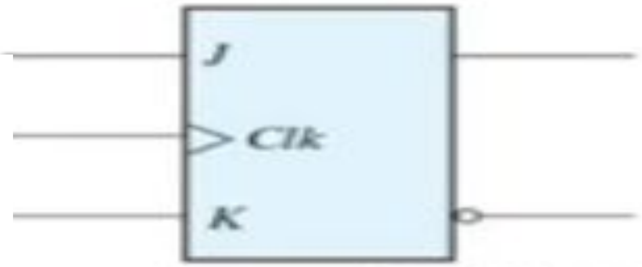
FIGURE 5.10
D-type positive-edge-triggered flip-flop

Other kind of Flip-Flops

JK Flip Flop



$$Q(t+1) = JQ' + K'Q$$



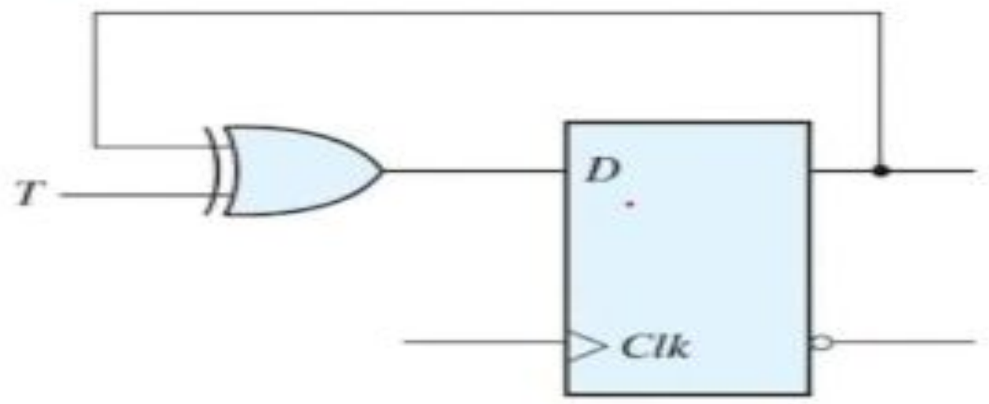
JK Flip-Flop

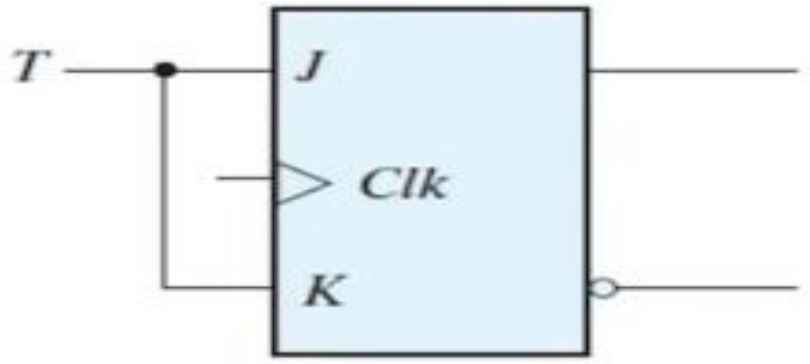
J	K	Q(t + 1)	
0	0	$Q(t)$	No change
0	1	0	Reset
1	0	1	Set
1	1	$Q'(t)$	Complement

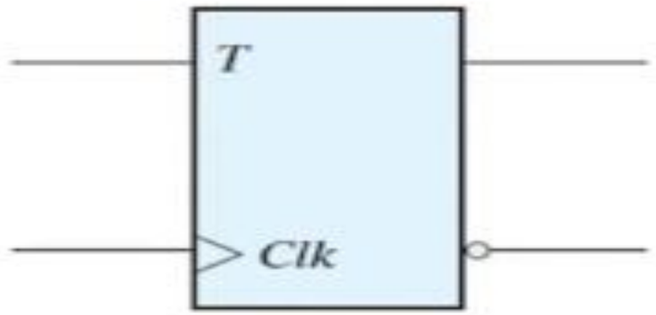
T- Flip Flop

$$Q(t+1) = T \oplus Q = TQ' + T'Q$$

T flip-flop







T Flip-Flop

T	$Q(t+1)$	
0	$Q(t)$	No change
1	$Q'(t)$	Complement

Characteristic Tables

- A characteristic table defines the logical properties of a flip-flop by describing its operation in tabular form.

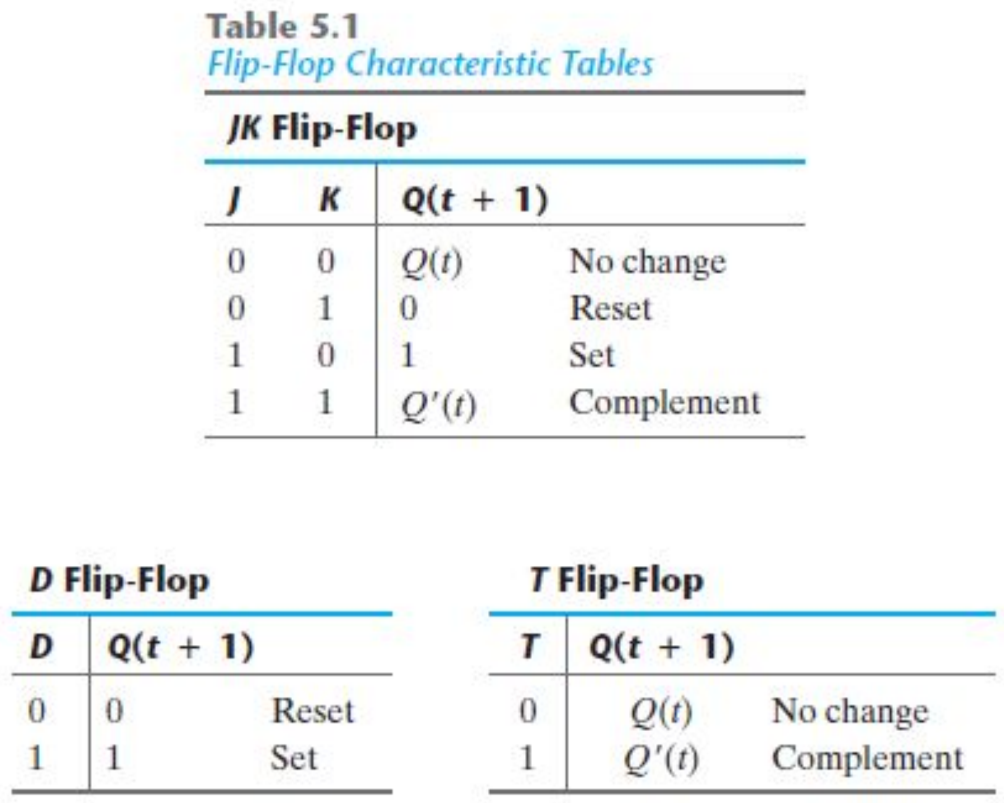


Table 5.1
Flip-Flop Characteristic Tables

***JK* Flip-Flop**

J	K	$Q(t+1)$	
0	0	$Q(t)$	No change
0	1	0	Reset
1	0	1	Set
1	1	$Q'(t)$	Complement

***D* Flip-Flop**

D	$Q(t+1)$	
0	0	Reset
1	1	Set

***T* Flip-Flop**

T	$Q(t+1)$	
0	$Q(t)$	No change
1	$Q'(t)$	Complement

Characteristic Equations

The logical properties of a flip-flop, as described in the characteristic table, can be expressed algebraically with a characteristic equation. For the D flip-flop, we have the characteristic equation

$$Q(t + 1) = D$$

which states that the next state of the output will be equal to the value of input D in the present state. The characteristic equation for the JK flip-flop can be derived from the characteristic table or from the circuit of Fig. 5.12. We obtain

$$Q(t + 1) = JQ' + K'Q$$

where Q is the value of the flip-flop output prior to the application of a clock edge. The characteristic equation for the T flip-flop is obtained from the circuit of Fig. 5.13:

$$Q(t + 1) = T \oplus Q = TQ' + T'Q$$