MODULE – 02

Chapter-01: Combinational Logic

Introduction

- Logic circuits for digital systems may be combinational or sequential.
- A combinational circuit consists of logic gates whose outputs at any time are determined from only the present combination of inputs.
- A combinational circuit performs an operation that can be specified logically by a set of Boolean functions. In contrast, sequential circuits employ storage elements in addition to logic gates.
- Their outputs are a function of the inputs and the state of the storage elements. Because the state of the storage elements is a function of previous inputs.
- The outputs of a sequential circuit depend not only on present values of inputs, but also on past inputs, and the circuit behaviour must be specified by a time sequence of inputs and internal states.
- Sequential circuits are the building blocks of digital systems and are discussed in Chapters 5 and 8.

Combinational circuit

- A combinational circuit consists of an interconnection of logic gates.
- Combinational logic gates react to the values of the signals at their inputs and produce the value of the output signal, transforming binary information from the given input data to a required output data.
- A block diagram of a combinational circuit is shown in Fig. 4.1. The n input binary variables come from an external source; the m output variables are produced by the internal combinational logic circuit and go to an external destination.
- Each input and output variable exists physically as an analog signal whose values are interpreted to be a binary signal that represents logic 1 and logic 0.
- **"(Note: Logic simulators show only 0's and 1's, not the actual analog signals.)."**

- In many applications, the source and destination are storage registers. If the registers are included with the combinational gates, then the total circuit must be considered to be a sequential circuit.

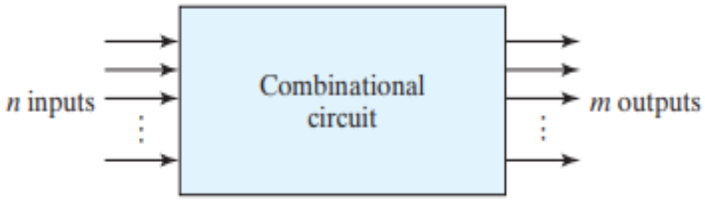


FIGURE 4.1
Block diagram of combinational circuit

- For n input variables, there are 2n possible combinations of the binary inputs. For each possible input combination, there is one possible value for each output variable.
- A combinational circuit also can be described by m Boolean functions, one for each output variable. Each output function is expressed in terms of the n input variables.
- There are several combinational circuits that are employed extensively in the design of digital systems.
- These components are available in integrated circuits as medium-scale integration (MSI) circuits.
- They are also used as standard cells in complex very large scale integrated (VLSI) circuits such as application-specific integrated circuits (ASICs).
- The standard cell functions are interconnected within the VLSI circuit in the same way that they are used in multiple-IC MSI design.
- The diagram of a combinational circuit has logic gates with no feedback paths or memory elements.

Design Procedure

The design of combinational circuits starts from the specification of the design objective and culminates in a logic circuit diagram or a set of Boolean functions from which the logic diagram can be obtained.

The procedure involves the following steps:

- From the specifications of the circuit, determine the required number of inputs and outputs and assign a symbol to each.
- Derive the truth table that defines the required relationship between inputs and Outputs.
- Obtain the simplified Boolean functions for each output as a function of the input variables.

- Draw the logic diagram and verify the correctness of the design (manually or by simulation).
- A truth table for a combinational circuit consists of input columns and output Columns.
- The input columns are obtained from the 2n binary numbers for the n input variables.
- The binary values for the outputs are determined from the stated specifications.
- The output functions specified in the truth table give the exact definition of the Combinational circuit.
- It is important that the verbal specifications be interpreted correctly in the truth table, as they are often incomplete, and any wrong interpretation may result in an incorrect truth table.
- The output binary functions listed in the truth table are simplified by any available method, such as algebraic manipulation, the map method, or a computer-based simplification program.
- Frequently, there is a variety of simplified expressions from which to choose. In a particular application, certain criteria will serve as a guide in the process of choosing an implementation.

Binary Adder–Subtractor

- A binary adder–Subtractor is a combinational circuit that performs the arithmetic operations of addition and subtraction with binary numbers.
- We will develop this circuit by means of a hierarchical design.
- The half adder design is carried out first, from which we develop the full adder. Connecting n full adders in cascade produces a binary adder for two n -bit numbers.
- The subtraction circuit is included in a complementing circuit.
- Digital computers perform a variety of information-processing tasks. Among the functions encountered are the various arithmetic operations.
- The most basic arithmetic operation is the addition of two binary digits. This simple addition consists of four possible elementary operations: 0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1, and 1 + 1 = 10.
- The first three operations produce a sum of one digit, but when both augend and addend bits are equal to 1, the binary sum consists of two digits. The higher significant bit of this result is called a carry.
- When the augend and addend numbers contain more significant digits, the carry obtained from the addition of two bits is added to the next higher order pair of significant bits.
- A combinational circuit that performs the addition of two bits is called a half adder.
- One that performs the addition of three bits (two significant bits and a previous carry) is a full adder.
- The names of the circuits stem from the fact that two half adders can be employed to implement a full adder.

HALF - ADDER

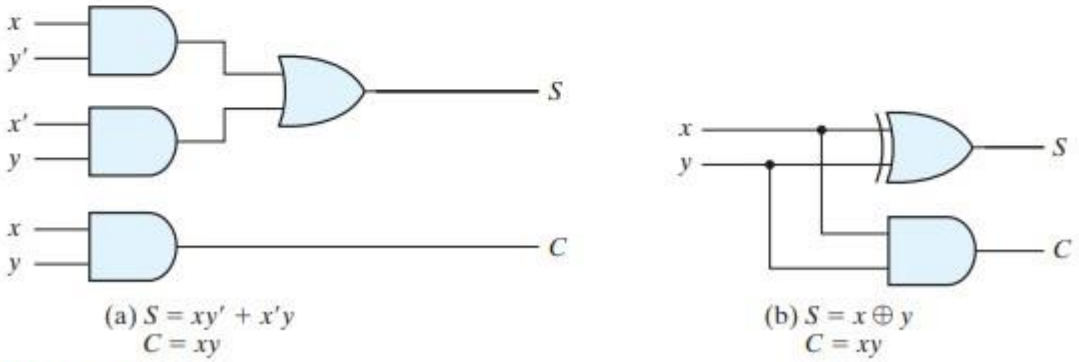
The simplified Boolean functions for the two outputs can be obtained directly from the truth table. The simplified sum-of-products expressions are

$$S = x'y + xy'$$
$$C = xy$$

The logic diagram of the half adder implemented in sum of products is shown in Fig. 4.5(a). It can be also implemented with an exclusive-OR and an AND gate as shown in Fig. 4.5(b). This form is used to show that two half adders can be used to construct a full adder.

Table 4.3
Half Adder

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



(a) $S = xy' + x'y$
$C = xy$

(b) $S = x \oplus y$
$C = xy$

FIGURE 4.5
Implementation of half adder

FULL - ADDER

A full adder is a combinational circuit that forms the arithmetic sum of three bits. It consists of three inputs and two outputs.

Two of the input variables, denoted by x and y, represent the two significant bits to be added.

The third input, z, represents the carry from the previous lower significant position.

The maps for the outputs of the full adder are shown in Fig. 4.6. The simplified expressions are

$$S = x'y'z + x'yz' + xy'z' + xyz$$
$$C = xy + xz + yz$$

The logic diagram for the full adder implemented in sum-of-products form is shown in Fig. 4.7. It can also be implemented with two half adders and one OR gate, as shown

Table 4.4
Full Adder

x	y	z	C	S
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

DECODER

- Discrete quantities of information are represented in digital systems by binary codes.
- A binary code of n bits is capable of representing up to 2n distinct elements of coded information.
- A decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2n unique output lines.
- If the n -bit coded information has unused combinations, the decoder may have fewer than 2n outputs.
- The decoders presented here are called n -to- m -line decoders, where m … 2n.
- Their purpose is to generate the 2n (or fewer) minterms of n input variables. Each combination of inputs will assert a unique output.
- The name decoder is also used in conjunction with other code converters, such as a BCD-to-seven-segment decoder.
- A particular application of this decoder is binary-to-octal conversion.
- The input variables represent a binary number, and the outputs represent the eight digits of a number in the octal number system.
- However, a three-to-eight-line decoder can be used for decoding any three-bit code to provide eight outputs, one for each element of the code.
- The three inverters provide the complement of the inputs, and each one of the eight AND gates generates one of the minterms.

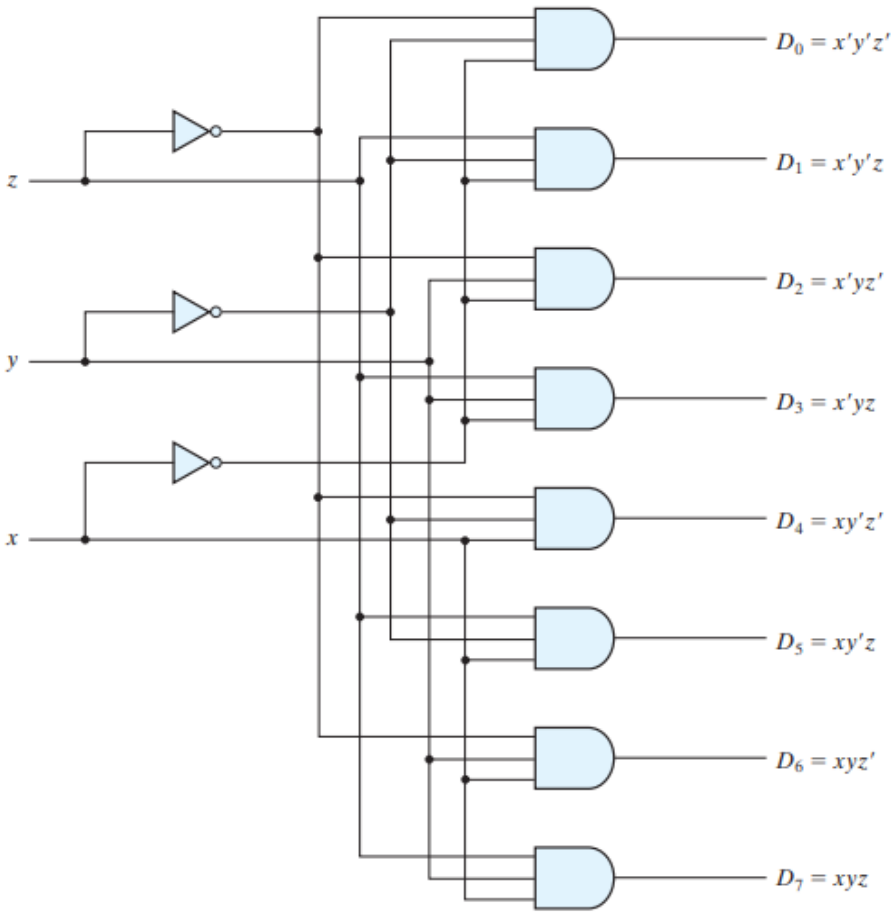
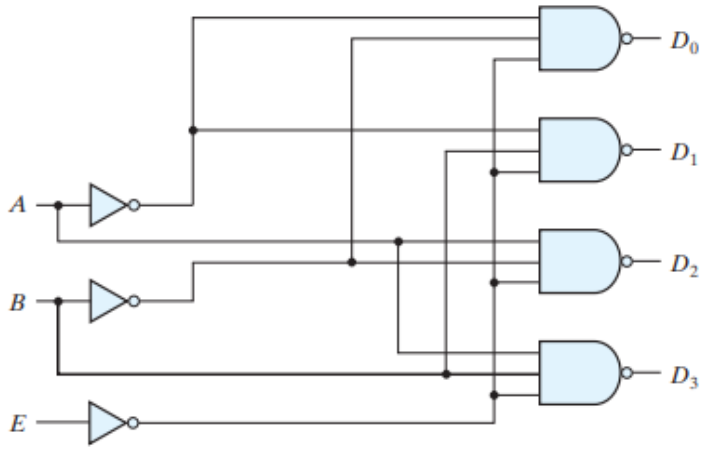


FIGURE 4.18
Three-to-eight-line decoder

Truth Table of a Three-to-Eight-Line Decoder

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

- The operation of the decoder may be clarified by the truth table listed in Table.
- For each possible input combination, there are seven outputs that are equal to 0 and only one that is equal to 1.
- The output whose value is equal to 1 represents the minterm equivalent of the binary number currently available in the input lines.
- Some decoders are constructed with NAND gates.
- Since a NAND gate produces the AND operation with an inverted output, it becomes more economical to generate the decoder minterms in their complemented form.
- Furthermore, decoders include one or more enable inputs to control the circuit operation.



(a) Logic diagram

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

(b) Truth table

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

- In general, a decoder may operate with complemented or un complemented outputs.
- The enable input may be activated with a 0 or with a 1 signal. Some decoders have two or more enable inputs that must satisfy a given logic condition in order to enable the circuit.
- A decoder with enable input can function as a demultiplexer— a circuit that receives information from a single line and directs it to one of 2n possible output lines.
- The selection of a specific output is controlled by the bit combination of n selection lines.

ENCODER

- An encoder is a digital circuit that performs the inverse operation of a decoder. An encoder has 2n (or fewer) input lines and n output lines.
- The output lines, as an aggregate, generate the binary code corresponding to the input value.
- An example of an encoder is the octal-to-binary encoder whose truth table is given in Table 4.7.
- It has eight inputs (one for each of the octal digits) and three outputs that generate the corresponding binary number.
- It is assumed that only one input has a value of 1 at any given time.
- The encoder can be implemented with OR gates whose inputs are determined directly from the truth table. Output z is equal to 1 when the input octal digit is 1, 3, 5, or 7.
- Output y is 1 for octal digits 2, 3, 6, or 7, and output x is 1 for digits 4, 5, 6, or 7.
- These conditions can be expressed by the following Boolean output functions:

$$z = D_1 + D_3 + D_5 + D_7$$
$$y = D_2 + D_3 + D_6 + D_7$$
$$x = D_4 + D_5 + D_6 + D_7$$

The encoder can be implemented with three OR gates.

Table 4.7
Truth Table of an Octal-to-Binary Encoder

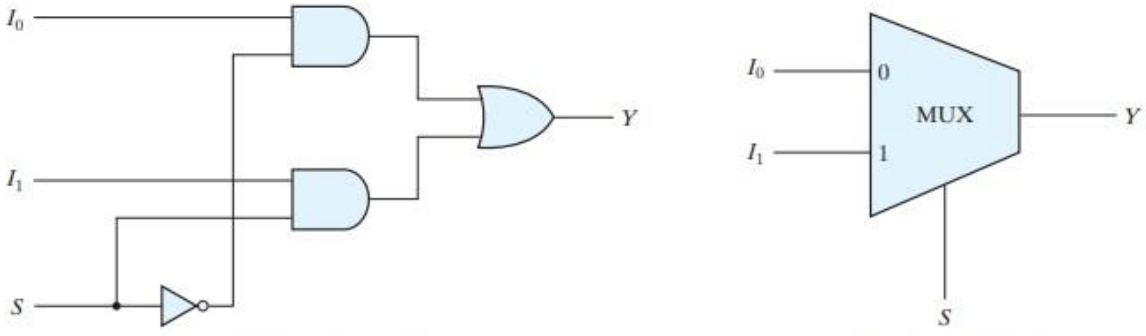
Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

- The encoder defined in Table has the limitation that only one input can be active at any given time.
- If two inputs are active simultaneously, the output produces an undefined combination.
- For example, if D3 and D6 are 1 simultaneously, the output of the encoder will be 111 because all three outputs are equal to 1.
- The discrepancy can be resolved by providing one more output to indicate whether at least one input is equal to 1.

MULTIPLEXER

- A multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line.
- 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 two-to-one-line multiplexer connects one of two 1-bit sources to a common destination, as shown in Figure.
- The circuit has two data input lines, one output line, and one selection line S.
- When S = 1, the lower AND gate is enabled and I1 has a path to the output.

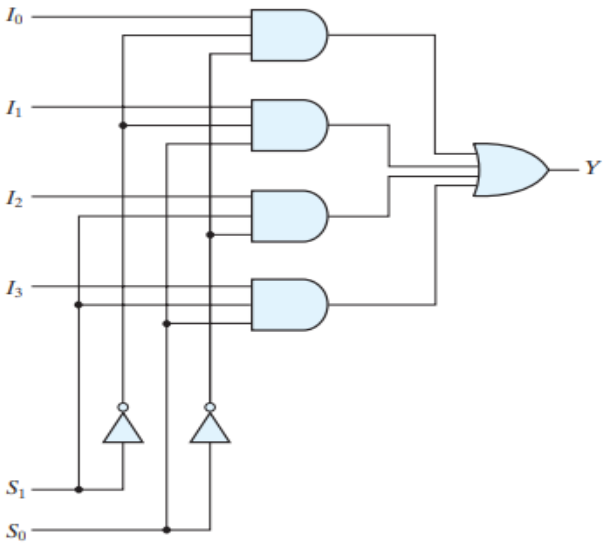
- When S = 0, the upper AND gate is enabled and I0 has a path to the output.
- The multiplexer acts like an electronic switch that selects one of two sources. The block diagram of a multiplexer is sometimes depicted by a wedge-shaped symbol, as shown in Figure.



(a) Logic diagram (b) Block diagram

FIGURE 4.24
Two-to-one-line multiplexer

- It suggests visually how a selected one of multiple data sources is directed into a single destination. The multiplexer is often labeled "MUX" in block diagrams.
- A four-to-one-line multiplexer is shown in Figure each of the four inputs, I0through I3, is applied to one input of an AND gate.



(a) Logic diagram

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

(b) Function table

FIGURE 4.25
Four-to-one-line multiplexer

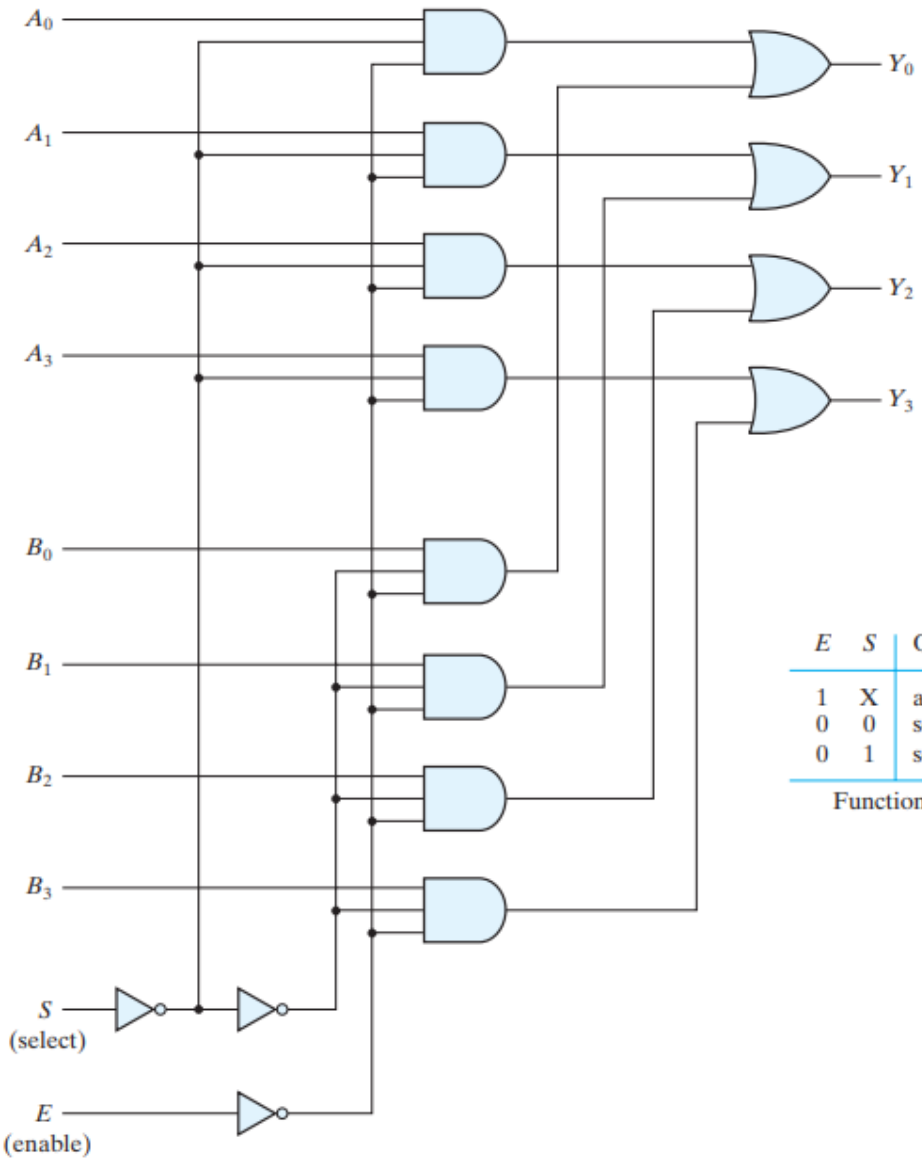
- Selection lines S1 and S0 are decoded to select a particular AND gate. The outputs of the AND gates are applied to a single OR gate that provides the one-line output.
- The function table lists the input that is passed to the output for each combination of the binary selection values.
- To demonstrate the operation of the circuit, consider the case when S1S0 = 10.
- Multiplexer circuits can be combined with common selection inputs to provide multiple-bit selection logic. As an illustration, a quadruple 2-to-1-line multiplexer is shown in Figure.
- The circuit has four multiplexers, each capable of selecting one of two input lines.
- Output Y0 can be selected to come from either input A0 or input B0. Similarly, output Y1 may have the value of A1 or B1, and so on. Input selection line S selects one of the lines in each of the four multiplexers.

Boolean Function Implementation

- It was shown that a decoder can be used to implement Boolean functions by employing external OR gates.
- An examination of the logic diagram of a multiplexer reveals that it is essentially a decoder that includes the OR gate within the unit.
- The minterms of a function are generated in a multiplexer by the circuit associated with the selection inputs.
- The individual minterms can be selected by the data inputs, thereby providing a method of implementing a Boolean function of n variables with a multiplexer that has n selection inputs and 2n data inputs, one for each minterm.
- Implementing a Boolean function of n variables with a multiplexer that has n - 1 selection inputs. The first n - 1 variables of the function are connected to the selection inputs of the multiplexer.
- The remaining single variable of the function is used for the data inputs. If the single variable is denoted y z, each data input of the multiplexer will be z , z, 1, or 0.

- To demonstrate this procedure, consider the Boolean function.

$$F(x, y, z) = \Sigma(1, 2, 6, 7)$$



E	S	Output Y
1	X	all 0's
0	0	select A
0	1	select B

Function table

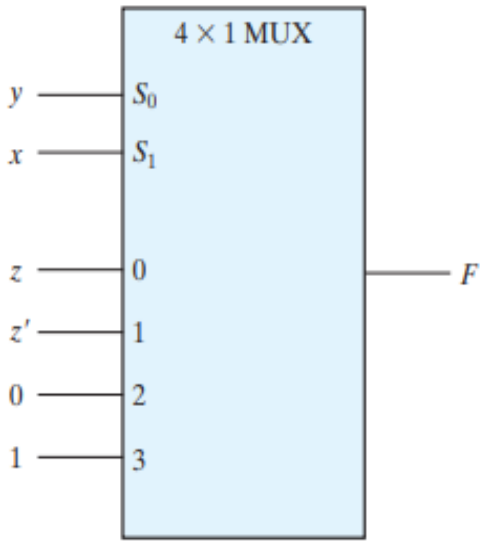
FIGURE 4.26
Quadruple two-to-one-line multiplexer

- This function of three variables can be implemented with a four-to-one-line multiplexer as shown in Figure. 4.27.

- The two variables x and y are applied to the selection lines in that order; x is connected to the S1 input and y to the S0 input.
- The values for the data input lines are determined from the truth table of the function.
- When xy = 00, output F is equal to z because F = 0 when z = 0 and F = 1 when z = 1. This requires that variable z be applied to data input 0.
- The general procedure for implementing any Boolean function of n variables with a multiplexer with n - 1 selection inputs and 2n-1data inputs follows from the previous example.
- To begin with, Boolean function is listed in a truth table.

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

FIGURE 4.27
Implementing a Boolean function with a multiplexer

- This function is implemented with a multiplexer with three selection inputs as shown in Figure. 4.28.

- Note that the first variable A must be connected to selection input S2 so that A , B, and C correspond to selection inputs S2, S1, and S0, respectively.

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	

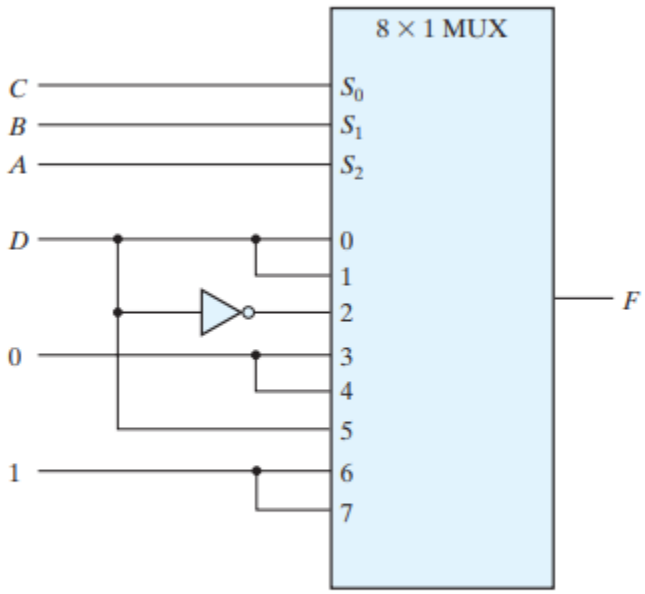


FIGURE 4.28
Implementing a four-input function with a multiplexer

- The values for the data inputs are determined from the truth table listed in the figure. The corresponding data line number is determined from the binary combination of ABC.

Three-State Gates

- A multiplexer can be constructed with three-state gates—digital circuits that exhibit three states.
- Two of the states are signals equivalent to logic 1 and logic 0 as in a conventional gate.
- The third state is a high-impedance state in which (1) the logic behaves like an open circuit, which means that the output appears to be disconnected, (2) the circuit has no logic significance, and (3) the circuit connected to the output of the three-state gate is not affected by the inputs to the gate.
- Three-state gates may perform any conventional logic, such as AND or NAND. However, the one most commonly used is the buffer gate.
- The graphic symbol for a three-state buffer gate is shown in Fig. 4.29. It is distinguished from a normal buffer by an input control line entering the bottom of the symbol.
- The buffer has a normal input, an output, and a control input that determines the state of the output.
- The graphic symbol for a three-state buffer gate is shown in Fig. 4.29. It is distinguished from a normal buffer by an input control line entering the bottom of the symbol.

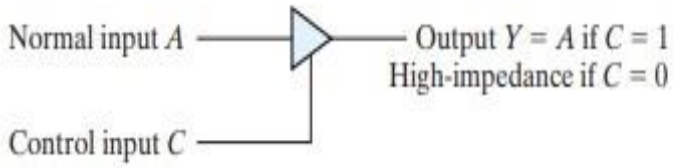


FIGURE 4.29
Graphic symbol for a three-state buffer

- The high-impedance state of a three-state gate provides a special feature not available in other gates.

- Because of this feature, a large number of three-state gate outputs can be connected with wires to form a common line without endangering loading effects.
- The construction of multiplexers with three-state buffers is demonstrated in Fig. 4.30.
- Figure 4.30(a) shows the construction of a two-to-one-line multiplexer with 2 three-state buffers and an inverter.
- The two outputs are connected together to form a single output line. (Note that this type of connection cannot be made with gates that do not have Three-state outputs.)

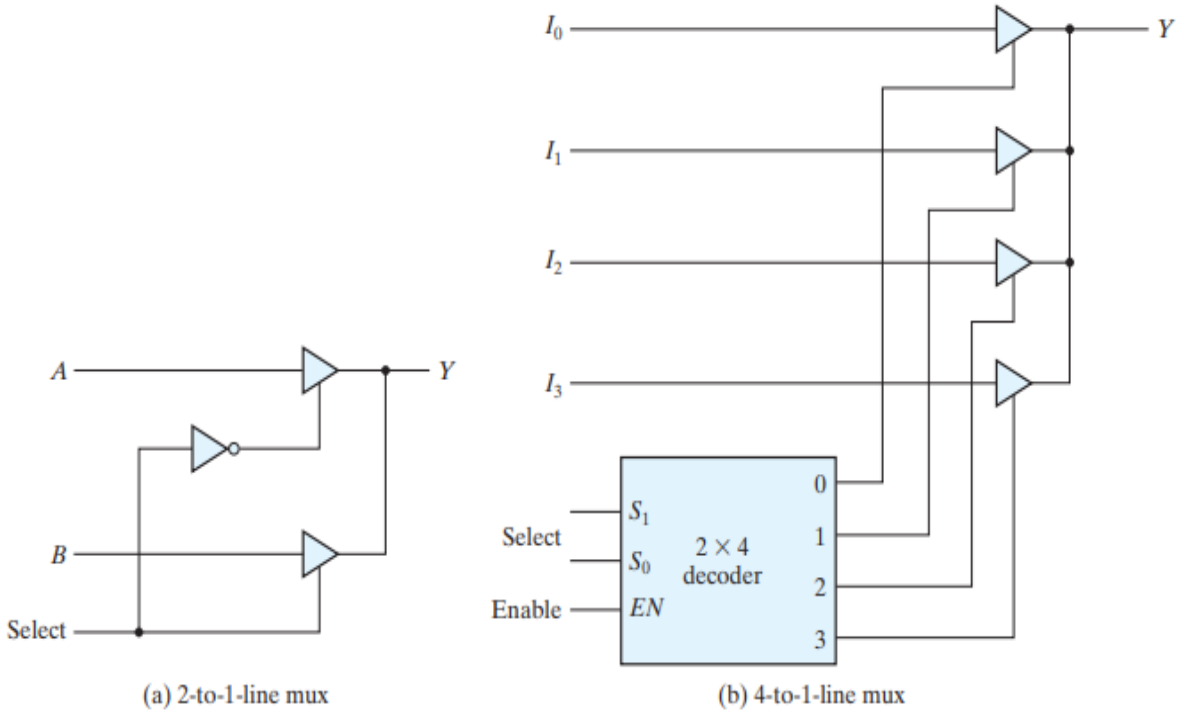


FIGURE 4.30
Multiplexers with three-state gates

- The construction of a four-to-one-line multiplexer is shown in Fig. 4.30(b). The outputs of 4 three-state buffers are connected together to form a single output line.
- The control inputs to the buffers determine which one of the four normal inputs I0 through I3 will be connected to the output line.

- No more than one buffer may be in the active state at any given time. The connected buffers must be controlled so that only 1 three state buffer has access to the output while all other buffers are maintained in a high impedance state.

HDL MODELS OF COMBINATIONAL CIRCUIT

The Verilog HDL was introduced in Section 3.10. In the current section, we introduce additional features of Verilog, present more elaborate examples, and compare alternative descriptions of combinational circuits in Verilog.

Sequential circuits are presented in as mentioned previously, the module is the basic building block for modelling hardware with the Verilog HDL.

The logic of a module can be described in any one (or a combination) of the following modelling styles:

- **Gate-level modelling using instantiations of predefined and user-defined primitive gates.**
- **Dataflow modelling using continuous assignment statements with the keyword assign.**
- **Behavioural modelling using procedural assignment statements with the keyword always.**

- Gate-level (structural) modelling describes a circuit by specifying its gates and how they are connected with each other.
- Dataflow modelling is used mostly for describing the Boolean equations of combinational logic.
- We'll also consider here behavioural modelling that is used to describe combinational and sequential circuits at a higher level of abstraction.
- Combinational logic can be designed with truth tables, Boolean equations, and
- Schematics; Verilog has a construct corresponding to each of these "classical" approaches to design: user-defined primitives, continuous assignments, and primitives, as shown in Fig 4.31.
- There is one other modelling style, called switch-level modelling.
- It is sometimes used in the simulation of MOS transistor circuit models, but not in logic synthesis. We will not consider switch-level modelling.

Gate-Level Modelling

- Gate-level modelling was introduced in Section 3.10 with a simple example. In this type of representation, a circuit is specified by its logic gates and their interconnections.
- Gate level modelling provides a textual description of a schematic diagram.
- The Verilog HDL includes 12 basic gates as predefined primitives. Four of these primitive gates are of the three-state type.
- The other eight are the same as the ones listed in Section 2.8. They are all declared with the lowercase keywords and, nand, or, nor, xor, xnor, not, and buf.
- Primitives such as and are n -input primitives. They can have any number of scalar inputs (e.g., a three-input and primitive).
- The buf and not primitives are n -output primitives.
- A single input can drive multiple output lines distinguished by their identifiers.
- The Verilog language includes a functional description of each type of gate, too. The logic of each gate is based on a four-valued system.

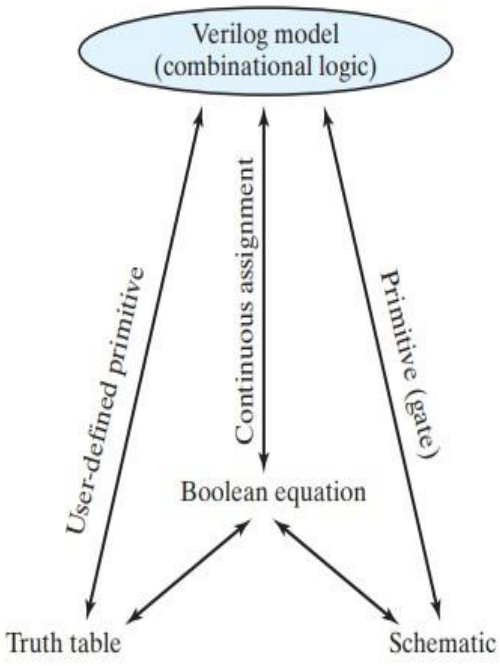


FIGURE 4.31
Relationship of Verilog constructs to truth tables, Boolean equations, and schematics

- The truth table for the other four gates is the same, except that the outputs are complemented.
- Note that for the and gate, the output is 1 only when both inputs are 1 and the output is 0 if any input is 0. Otherwise, if one input is x or z, the output is x.
- The output of the or gate is 0 if both inputs are 0, is 1 if any input is 1, and is x otherwise.
- When a primitive gate is listed in a module, we say that it is instantiated in the module.
- In general, component instantiations are statements that reference lower level components in the design, essentially creating unique copies (or instances) of those components in the higher level module.
- Thus, a module that uses a gate in its description is said to instantiate the gate. Think of instantiation as the HDL counterpart of placing and connecting parts on a circuit board.

- We now present two examples of gate-level modelling. Both examples use identifiers having multiple bit widths, called vectors .
- The syntax specifying a vector includes within square brackets two numbers separated with a colon. The following Verilog statements
  Specify two vectors:
  - output [0: 3] D;
  - wire [7: 0] SUM;
- The first statement declares an output vector D with four bits, 0 through 3. The second declares a wire vector SUM with eight bits numbered 7 through 0.

Truth Table for Predefined Primitive Gates

and	0	1	**x**	**z**
0	0	0	0	0
1	0	1	**x**	**x**
x	0	**x**	**x**	**x**
z	0	**x**	**x**	**x**

or	0	1	**x**	**z**
0	0	1	**x**	**x**
1	1	1	1	1
x	**x**	1	**x**	**x**
z	**x**	1	**x**	**x**

xor	0	1	**x**	**z**
0	0	1	**x**	**x**
1	1	0	**x**	**x**
x	**x**	**x**	**x**	**x**
z	**x**	**x**	**x**	**x**

not	input	output
	0	1
	1	0
	x	**x**
	z	**x**

Chapter-02: Sequential Logic

Sequential Logic

Introductions

- Hand-held devices, cell phones, navigation receivers, personal computers, digital cameras, personal media players, and virtually all electronic consumer products have the ability to send, receive, store, retrieve, and process information represented in a binary format.
- The technology enabling and supporting these devices is critically dependent on electronic components that can store information, i.e., have memory.
- This chapter examines the operation and control of these devices and their use in circuits and enables you to better understand what is happening in these devices when you interact with them.
- The digital circuits considered thus far have been combinational—their output depends only and immediately on their inputs they have no memory, i.e., dependence on past values of their inputs.
- Sequential circuits, however, act as storage elements and have memory.
- They can store, retain, and then retrieve information when needed at a later time. Our treatment will distinguish sequential logic from combinational logic.

Sequential circuit

- A block diagram of a sequential circuit is shown in Fig. 5.1.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.
- In contrast, the outputs of combinational logic depend only on the present values of the inputs.

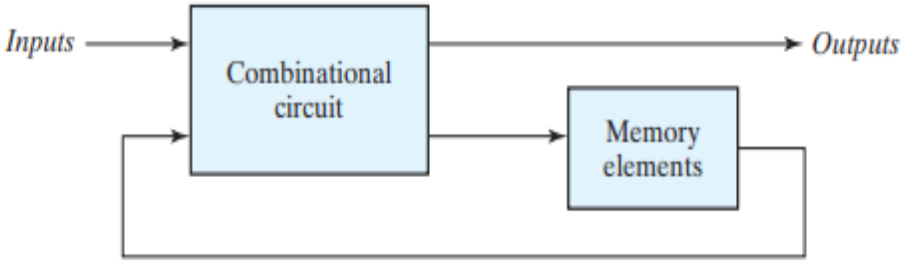
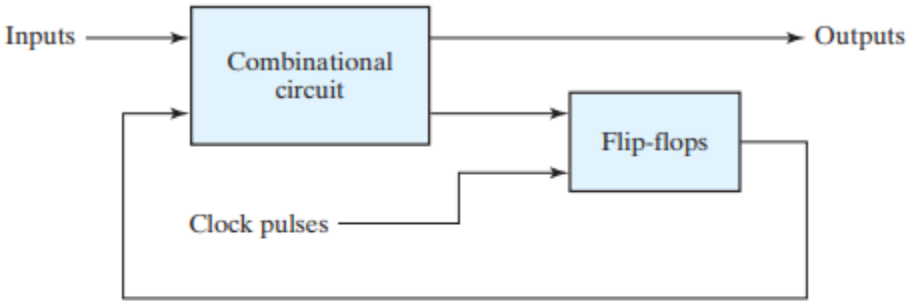


FIGURE 5.1
Block diagram of sequential circuit

- There are two main types of sequential circuits, and their classification is a function of the timing of their signals.
- A synchronous sequential circuit is a system whose behaviour can be defined from the knowledge of its signals at discrete instants of time.
- The behaviour of an asynchronous sequential circuit depends upon the input signals at any instant of time and the order in which the inputs change.
- The storage elements commonly used in asynchronous sequential circuits are time-delay devices.
- The storage capability of a time-delay device varies with the time it takes for the signal to propagate through the device.
- A synchronous sequential circuit employs signals that affect the storage elements at only discrete instants of time.
- Synchronization is achieved by a timing device called a clock generator, which provides a clock signal having the form of a periodic train of clock pulses.
- The clock signal is commonly denoted by the identifiers clock and clk.

- The storage elements (memory) used in clocked sequential circuits are called flip-flops.
- A flip-flop is a binary storage device capable of storing one bit of information. In a stable state, the output of a flip-flop is either 0 or 1.
- A sequential circuit may use many flip-flops to store as many bits as necessary.
- The block diagram of a synchronous clocked sequential circuit is shown in Fig. 5.2.



(a) Block diagram

(b) Timing diagram of clock pulses

FIGURE 5.2
Synchronous clocked sequential circuit

Storage Elements: LATCHES

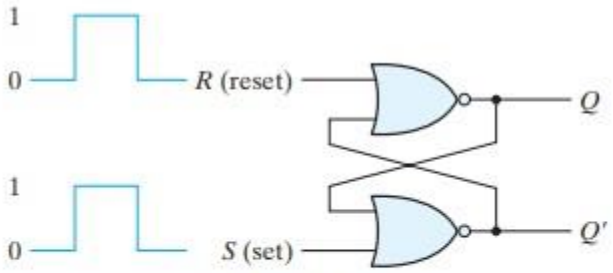
- Storage elements that operate with signal levels (rather than signal transitions) are referred to as latches; those controlled by a clock transition are flip-flops.
- Latches are said to be level sensitive devices; flip-flops are edge-sensitive devices.
- The two types of storage elements are related because latches are the basic circuits from which all flip-flops are constructed.
- Although latches are useful for storing binary information and for the design of asynchronous sequential circuits, they are not practical for use as storage elements in synchronous sequential circuits.

- Because they are the building blocks of flip-flops, however, we will consider the fundamental storage mechanism used in latches before considering flip-flops in the next section.

SR – LATCHES

The *SR* latch is a circuit with two cross-coupled NOR gates or two cross-coupled NAND gates, and two inputs labeled *S* for set and *R* for reset. The *SR* latch constructed with two cross-coupled NOR gates is shown in Fig. 5.3. The latch has two useful states. When output $Q = 1$ and $Q' = 0$, the latch is said to be in the *set state*. When $Q = 0$ and $Q' = 1$, it is in the *reset state*. Outputs Q and Q' are normally the complement of each other.

- Under normal conditions, both inputs of the latch remain at 0 unless the state has to be changed.
- The application of a momentary 1 to the S input causes the latch to go to the set state.
- The S input must go back to 0 before any other changes take place, in order to avoid the occurrence of an undefined next state that results from the forbidden input condition.
- As shown in the function table of Fig. 5.3 (b), two input conditions cause the circuit to be in the set state. The first condition (S = 1,R = 0) is the action that must be taken by input S to bring the circuit to the set state.
- Removing the active input from S leaves the circuit in the same state. After both inputs return to 0, it is then possible to shift to the reset state by momentary applying a 1 to the R input.
- The 1 can then be removed from R, whereupon the circuit remains in the reset state.



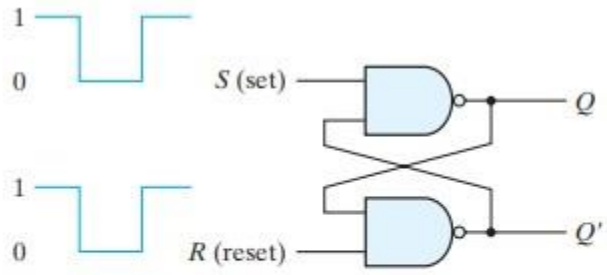
(a) Logic diagram

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

FIGURE 5.3
***SR* latch with NOR gates**

- The SR latch with two cross-coupled NAND gates is shown in Fig. 5.4 .
- It operates with both inputs normally at 1, unless the state of the latch has to be changed.
- The application of 0 to the S input causes output Q to go to 1, putting the latch in the set state.
- When the S input goes back to 1, the circuit remains in the set state. After both inputs go back to 1, we are allowed to change the state of the latch by placing a 0 in the R input.



(a) Logic diagram

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

FIGURE 5.4
***SR* latch with NAND gates**

FLIP - FLOPS

- The state of a latch or flip-flop is switched by a change in the control input. This momentary change is called a trigger, and the transition it causes is said to trigger the flip-flop.
- The D latch with pulses in its control input is essentially a flip-flop that is triggered every time the pulse goes to the logic-1 level.
- As long as the pulse input remains at this level, any changes in the data input will change the output and the state of the latch.
- As seen from the block diagram of Fig. 5.2, a sequential circuit has a feedback path from the outputs of the flip-flops to the input of the combinational circuit.
- Consequently, the inputs of the flip-flops are derived in part from the outputs of the same and other 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.
- The problem with the latch is that it responds to a change in the level of a clock pulse. As shown in Fig. 5.8 (a), a positive level response in the enable input allows changes in the output.
- When the D input changes while the clock pulse stays at logic 1. The key to the proper operation of a flip-flop is to trigger it only during a signal transition.
- This can be accomplished by eliminating the feedback path that is inherent in the operation of the sequential circuit using latches.
- A clock pulse goes through two transitions: from 0 to 1 and the return from 1 to 0.
- There are two ways that a latch can be modified to form a flip-flop. One way is to employ two latches in a special configuration that isolates.
- The output of the flip-flop and prevents it from being affected while the input to the flip-flop is changing. Another way is to produce a flip-flop that triggers only.
- During a signal transition (from 0 to 1 or from 1 to 0) of the synchronizing signal (clock) and is disabled during the rest of the clock pulse. We will now proceed to show the implementation of both types of flip-flops.

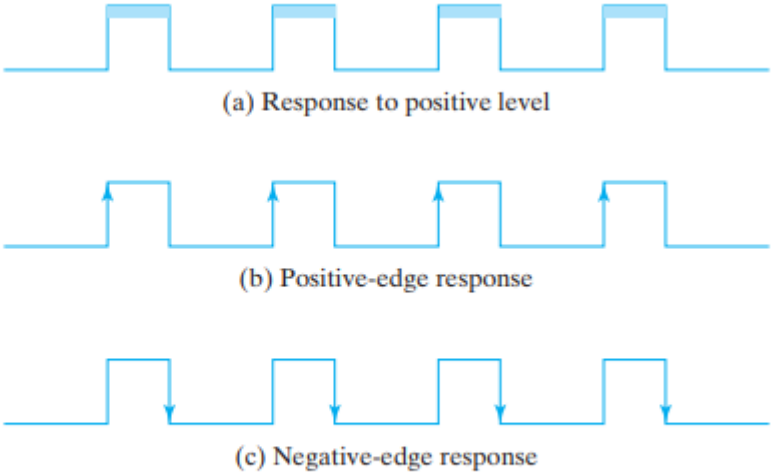


FIGURE 5.8
Clock response in latch and flip-flop

- 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.
- The problem with the latch is that it responds to a change in the level of a clock pulse.

Edge-Triggered D Flip-Flop

- The construction of a D flip-flop with two D latches and an inverter is shown in Figure 5.9.
- The first latch is called the master and the second the slave. The circuit samples the D input and changes its output Q only at the negative edge of the synchronizing or controlling clock (designated as Clk).

- The behaviour of the master–slave flip-flop just described dictates that (1) the output may change only once, (2) a change in the output is triggered by the negative edge of the clock, and (3) the change may occur only during the clock's negative level.

- When the clock is 0, the output of the inverter is 1. The slave latch is enabled, and its output Q is equal to the master output Y.
- The master latch is disabled because Clk = 0. When the input pulse changes to the logic-1 level, the data from the external D input are transferred to the master.
- Thus, a change in the output of the flip-flop can be triggered only by and during the transition of the clock from 1 to 0.
- It is also possible to design the circuit so that the flip-flop output changes on the positive edge of the clock.
- This happens in a flip-flop that has an additional inverter between the Clk terminal and the junction between the other inverter and input En of the master latch.
- Such a flip-flop is triggered with a negative pulse, so that the negative edge of the clock affects the master and the positive edge affects the slave and the output terminal.

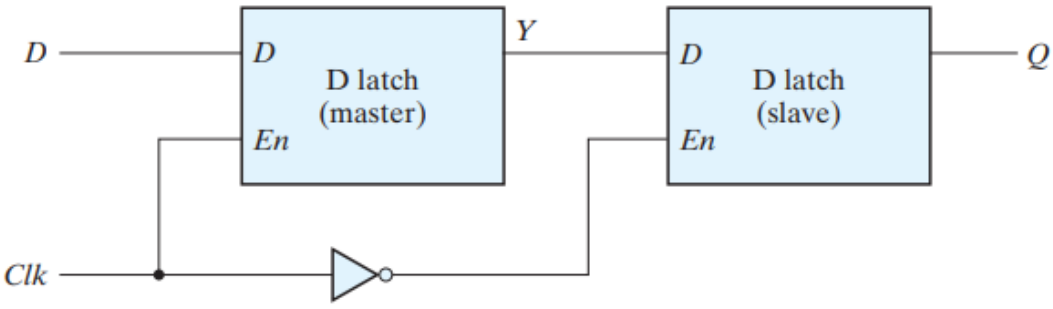


FIGURE 5.9
Master–slave *D* flip-flop

- Another construction of an edge-triggered D flip-flop uses three SR latches as shown in Figure 5.10.
- Two latches respond to the external D (data) and Clk (clock) inputs. The third latch provides the outputs for the flip-flop.

- The S and R inputs of the output latch are maintained at the logic-1 level when Clk = 0. This causes the output to remain in its present state. Input D may be equal to 0 or 1.
- If D = 0 when Clk becomes 1, R changes to 0. This causes the flip-flop to go to the reset state, making Q = 0. If there is a change in the D input while Clk = 1, terminal R remains at 0 because Q is 0.
- Thus, the flip-flop is locked out and is unresponsive to further changes in the input. When the clock returns to 0, R goes to 1, placing the output latch in the quiescent condition without changing the output.
- Similarly, if D = 1 when Clk goes from 0 to 1, S changes to 0. This causes the circuit to go to the set state, making Q = 1. Any change in D while Clk = 1 does not affect the output.

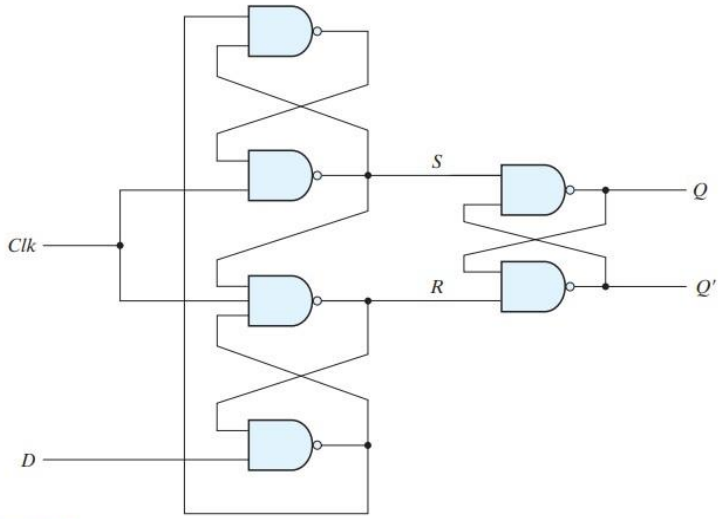


FIGURE 5.10
D-type positive-edge-triggered flip-flop

- The graphic symbol for the edge-triggered D flip-flop is shown in Figure 5.11.
- It is similar to the symbol used for the D latch, except for the arrowhead-like symbol in front of the letter Clk, designating a dynamic input.
- The dynamic indicator (>) denotes the fact that the flip-flop responds to the edge transition of the clock.

- A bubble outside the block adjacent to the dynamic indicator designates a negative edge for triggering the circuit. The absence of a bubble designates a positive-edge response.

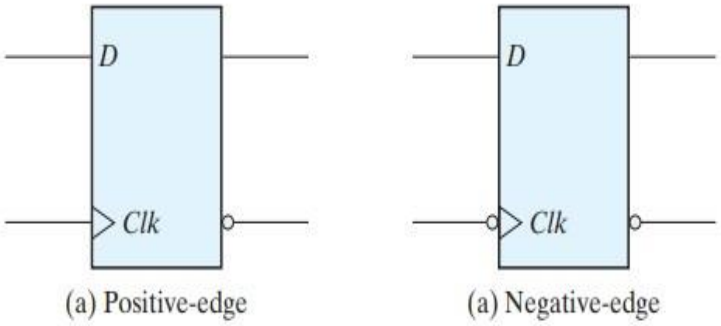
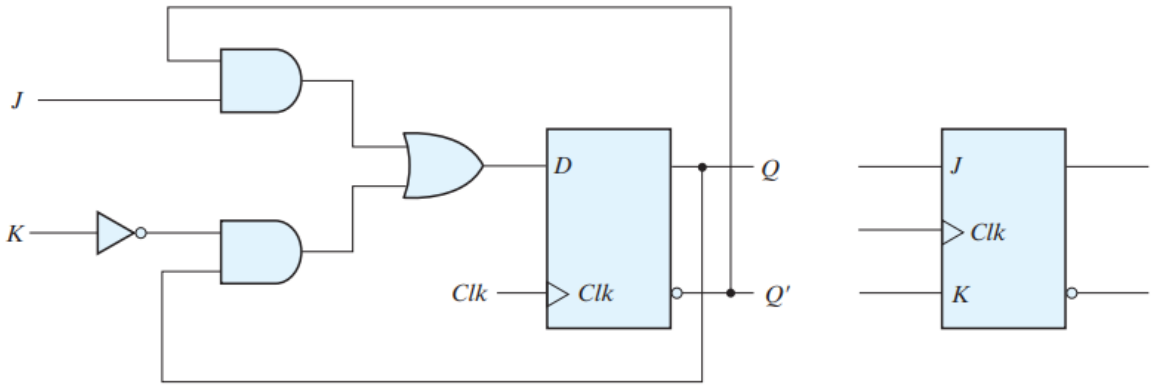


FIGURE 5.11
Graphic symbol for edge-triggered *D* flip-flop

Other Flip-Flops

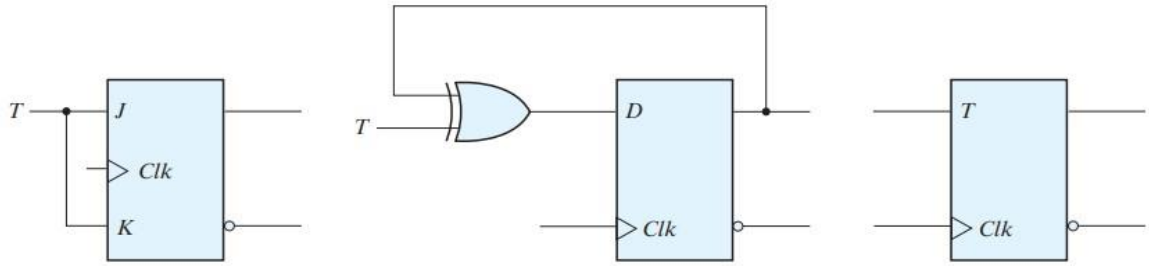
- Very large-scale integration circuits contain several thousands of gates within one package.
- Circuits are constructed by interconnecting the various gates to provide a digital System.
- Each flip-flop is constructed from an interconnection of gates.
- The most economical and efficient flip-flop constructed in this manner is the edge-triggered D flip flop, because it requires the smallest number of gates. Other types of flip-flops can be
- Constructed by using the D flip-flop and external logic. Two flip-flops less widely used in the design of digital systems are the JK and T flip-flops.



(a) Circuit diagram (b) Graphic symbol

FIGURE 5.12
JK flip-flop

The T (toggle) flip-flop is a complementing flip-flop and can be obtained from a JK flip-flop when inputs J and K are tied together. This is shown in Fig. 5.13 (a).



(a) From *JK* flip-flop (b) From *D* flip-flop (c) Graphic symbol

FIGURE 5.13
T flip-flop

$T = 0$ ($J = K = 0$), a clock edge does not change the output. When $T = 1$ ($J = K = 1$), a clock edge complements the output. The complementing flip-flop is useful for designing binary counters.

The T flip-flop can be constructed with a D flip-flop and an exclusive-OR gate as shown in Fig. 5.13(b). The expression for the D input is

$$D = T \oplus Q = TQ' + T'Q$$

When $T = 0$, $D = Q$ and there is no change in the output. When $T = 1$, $D = Q'$ and the output complements. The graphic symbol for this flip-flop has a T symbol in the input.

Characteristic Tables

- A characteristic table defines the logical properties of a flip-flop by describing its operation in tabular form. The characteristic tables of three types of flip-flops are presented in Table 5.1.
- They define the next state (i.e., the state that results from a clock transition) as a function of the inputs and the present state.
- Q (t) refers to the present state (i.e., the state present prior to the application of a clock edge).
- Q(t + 1) is the next state one clock period later. Note that the clock edge input is not included in the characteristic table, but is implied to occur between times t and t + 1.
- Thus Q(t) denotes the state of the flip-flop immediately before the clock edge, and Q(t + 1) denotes the state that results from the clock transition.

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

- The characteristic table for the JK flip-flop shows that the next state is equal to the present state when inputs J and K are both equal to 0.
- This condition can be expressed as Q(t + 1) = Q(t), indicating that the clock produces no change of state.
- When K = 1 and J = 0, the clock resets the flip-flop and Q(t + 1) = 0. With J = 1 and K = 0, the flip-flop sets and Q(t + 1) = 1.
- When both J and K are equal to 1, the next state changes to the complement of the present state, a transition that can be expressed as Q(t + 1) = Q'(t).
- The characteristic table of the T flip-flop has only two conditions: When T = 0, the clock edge does not change the state;
- When T = 1, the clock edge complements the state of the flip-flop.

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:

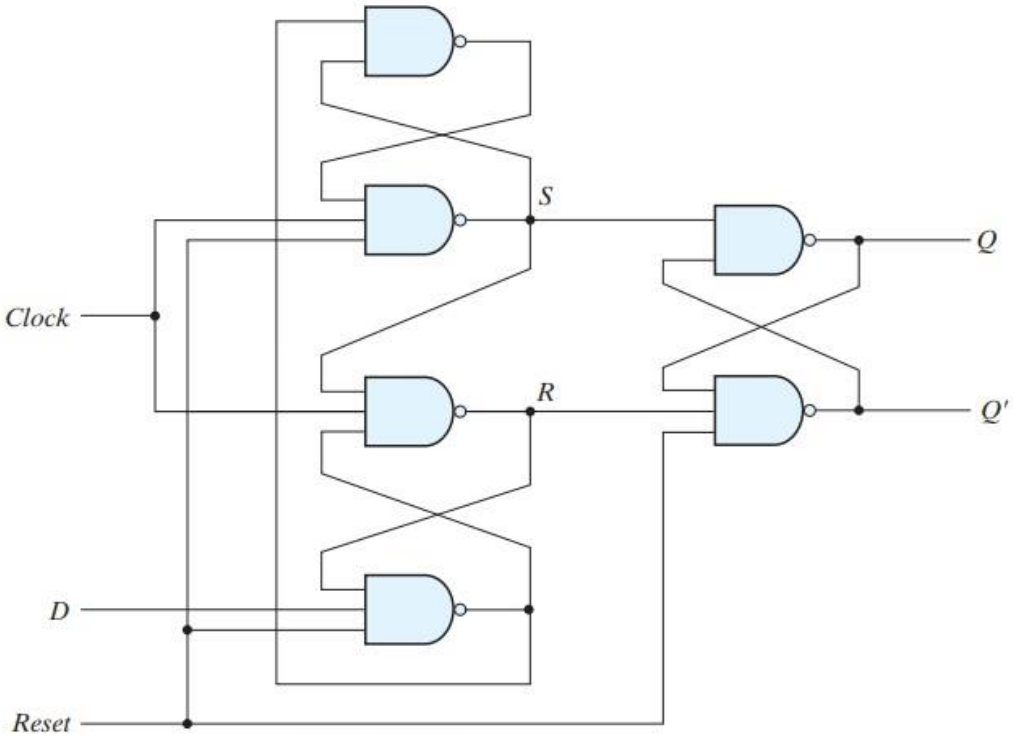
$$Q(t + 1) = T \oplus Q = TQ' + T'Q$$

Direct Inputs

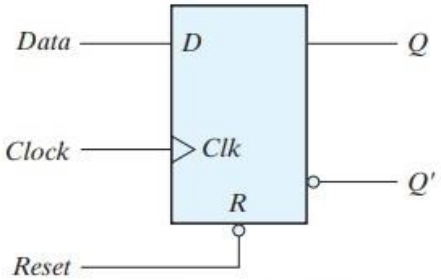
Some flip-flops have asynchronous inputs that are used to force the flip-flop to a particular state independently of the clock. The input that sets the flip-flop to 1 is called preset or direct set.

The input that clears the flip-flop to 0 is called clear or direct reset.

- When power is turned on in a digital system, the state of the flip-flops is unknown.
- The direct inputs are useful for bringing all flip-flops in the system to a known starting state prior to the clocked operation.
- A positive-edge-triggered D flip-flop with active-low asynchronous reset is shown in Fig. 5.14.
- The circuit diagram is the same as the one in Fig. 5.10, except for the additional reset input connections to three NAND gates.
- When the reset input is 0, it forces output Q to stay at 1, which, in turn, clears output Q to 0, thus resetting the flip-flop.
- The graphic symbol for the D flip-flop with a direct reset has an additional input marked with R. The bubble along the input indicates that the reset is active at the logic-0 level.
- Flip-flops with a direct set use the symbol S for the asynchronous set input.
- The function table specifies the operation of the circuit. When R = 0, the output is reset to 0. This state is independent of the values of D or Clk.



(a) Circuit diagram



(b) Graphic symbol

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

(b) Function table

FIGURE 5.14
***D* flip-flop with asynchronous reset**