Some D flip-flops have asynchronous inputs that may be used to force the flip-flop to a particular state independent of the CLK and D inputs. These inputs, labeled PR (preset) and CLR (clear), behave like the set and reset inputs on an S-R latch. The logic symbol and NAND circuit for an edge-triggered D flip-flop with these inputs is shown in Figure 7-19. Although asynchronous inputs are used by some logic designers to perform tricky sequential functions, they are best reserved for initialization and testing purposes, to force a sequential circuit into a known starting state.

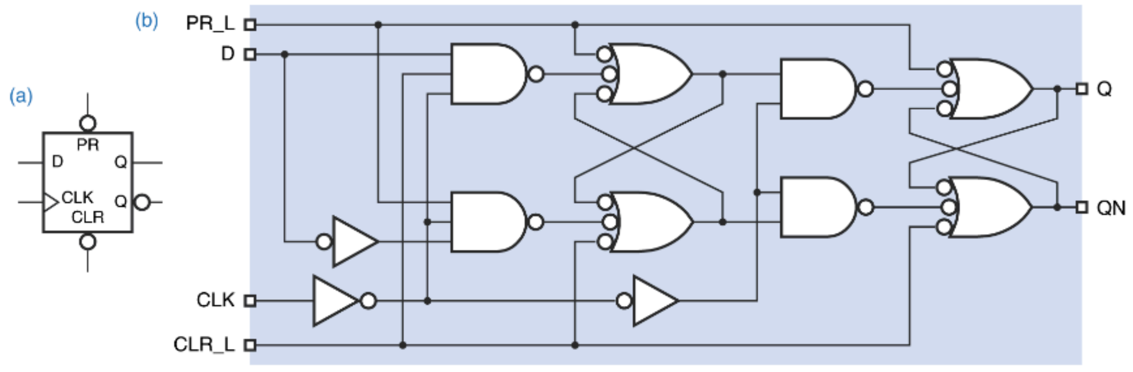


Figure 7-19
Positive-edge-triggered D flip-flop with preset and clear:
(a) logic symbol; (b) circuit design using NAND gates.

Commercial positive-edge triggered flip-flops do not use the master-slave design shown in Figure 7-19. Instead they use a smaller and faster design shown in Figure 7-210.

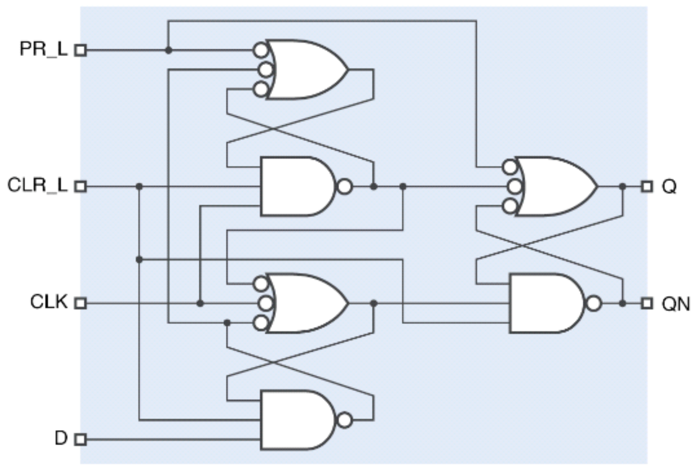
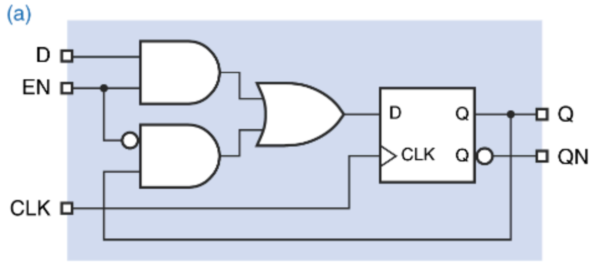


Figure 7-20
Commercial circuit for a positive-edge-triggered D flip-flop such as 74LS74.

7.2.6 Edge-Triggered D Flip-Flop with Enable

A commonly desired function in D flip-flops is the ability to hold the last value stored, rather than load a new value, at the clock edge. This is accomplished by adding an enable input, called EN or CE (clock enable). While the name “clock enable” is descriptive, the extra input's function is not obtained by controlling the clock in any way whatsoever. Rather, as shown in Figure 7-21 (a), a 2-input multiplexer controls the value applied to the internal flip-flop's D input. If EN is asserted, the external D input is selected; if EN is negated, the flip-flop's current output is used. The resulting function table is shown in (b). The flip-flop symbol is shown in (c); in some flip-flops, the enable input is active low, denoted by an inversion bubble on this input.

(a)



(b)

D	EN	CLK	Q	QN
0	1	↑	0	1
1	1	↑	1	0
x	0	↑	last Q	last QN
x	x	0	last Q	last QN
x	x	1	last Q	last QN

(c)

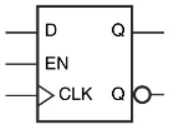
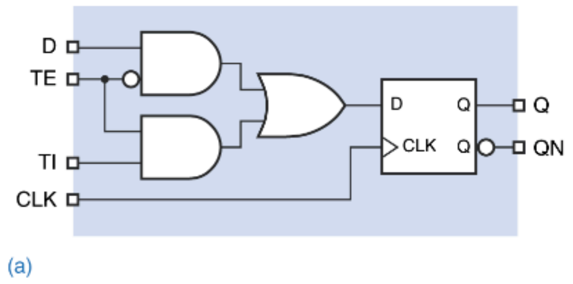


Figure 7-21
Positive-edge-triggered D flip-flop with enable: (a) circuit design; (b) function table; (c) logic symbol.

7.2.7 Scan Flip-Flop

An important flip-flop function for ASIC testing is so-called scan capability. The idea is to be able to drive the flip-flop's D input with an alternate source of data during device testing. When all of the flip-flops are put into testing mode, a test pattern can be "scanned in" to the ASIC using the flip-flops' alternate data inputs. After the test pattern is loaded, the flip-flops are put back into "normal" and all of the flip-flops are clocked normally. After one or more clock ticks, the flip-flops are put back into test mode, and the test results are “scanned out.”

Figure 7-22(a) shows the design of a typical scan flip-flop. It is nothing more than a D flip-flop with a 2-input multiplexer on the D input. When the TE (Test Enable) input is negated, the circuit behaves like an ordinary D flip-flop. When TE is asserted, it takes its data from TI (Test Input) instead of from D. This functional behavior is shown in (b), and a symbol for the device is given in (c).



(b)

TE	TI	D	CLK	Q	QN
0	x	0	↑	0	1
0	x	1	↑	1	0
1	0	x	↑	0	1
1	1	x	↑	1	0
x	x	x	0	last Q	last QN
x	x	x	1	last Q	last QN

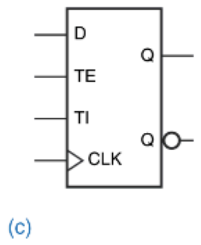


Figure 7-22
Positive-edge-triggered D flip-flop with scan: (a) circuit design; (b) function table; (c) logic symbol.

The extra inputs are used to connect all of an ASIC's flip-flops in a scan chain for testing purposes. Figure 7-23 is a simple example with four flip-flops in the scan chain. The TE inputs of all the flip-flops are connected to a global TE input, while each flip-flop's Q output is connected to another's TI input in serial (daisy-chain) fashion. The TI, TE, and TO (Test Output) connections are strictly for testing purposes; the additional logic connected to the D inputs and Q outputs needed to make the circuit do something useful is not shown.

To test the circuit, including the additional logic, the global TE input is asserted while n clock ticks occur and n test-vector bits are applied to the global TI input and are thereby scanned (shifted) into the n flip-flops; n equals 4 in Figure 7-23. Then TE is negated, and the circuit is allowed to run for one or more additional clock ticks. The new state of the circuit, represented by the new values in the n flip-flops, can be observed (scanned out) at TO by asserting TE while n more clock ticks occur. To make the testing process more efficient, another test vector can be scanned in while the previous result is being scanned out.

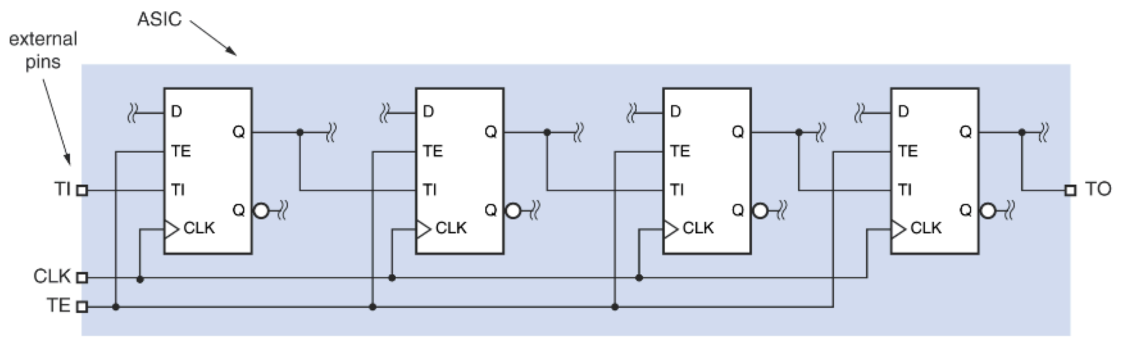


Figure 7-23
A scan chain with four flip-flops.

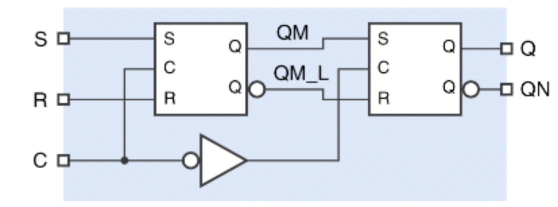
There are many different types of scan flip-flops, corresponding to different types of basic flip-flop functionality. For example, scan capability could be added to the D flip-flop with enable in Figure 7-21 by replacing its internal 2-input multiplexer with a 3-input one. At each clock tick the flip-flop would load D, TI, or its current state, depending on the values of EN and TE. Scan capability can also be added to other flip-flop types, such as J-K and T introduced later in this section.

7.2.8 Master/Slave S-R Flip-Flop

We indicated earlier that S-R latches are useful in "control" applications, where we may have independent conditions for setting and resetting a control bit. If the control bit is supposed to be changed only at certain times with respect to a clock signal, then we need an S-R flip-flop that, like a D flip-flop, changes its outputs only on a specific edge of the clock signal. This subsection and the next two describe flip-flops that are useful for such applications.

If we substitute S-R latches for the D latches in the negative-edge-triggered D flip-flop of Figure 7-18(a), we get a master/slave S-R flip-flop, shown in Figure 7-24.

(a)



(b)

S	R	C	Q	QN
x	x	0	last Q	last QN
0	0	⎍	last Q	last QN
0	1	⎍	0	1
1	0	⎍	1	0
1	1	⎍	undef.	undef.

(c)

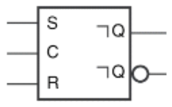


Figure 7-24
Master/slave S-R flip-flop: (a) circuit using S-R latches; (b) function table; (c) logic symbol.

Like a D flip-flop, the S-R flip-flop changes its outputs only at the falling edge of a control signal C. However, the new output value depends on input values not just at the falling edge, but during the entire interval in which C is 1 prior to the falling edge. As shown in Figure 7-25, a short pulse on S any time during this interval can set the master latch; likewise, a pulse on R can reset it. The value transferred to the flip-flop output on the falling edge of C depends on whether the master latch was last set or cleared while C was 1.

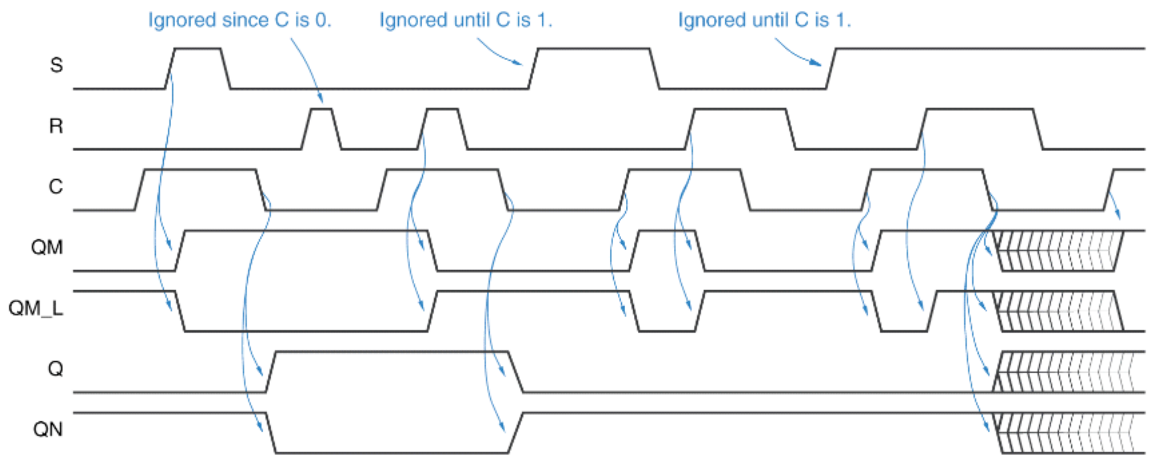


Figure 7-25
Internal and functional behavior of a master/slave S-R flip-flop.

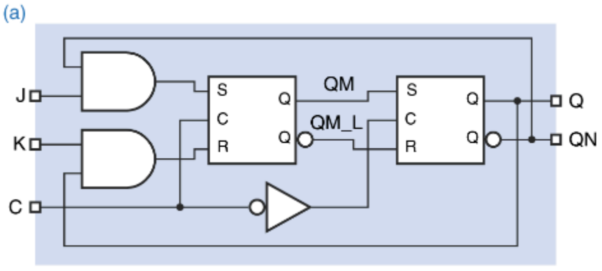
Shown in Figure 7-24(c), the logic symbol for the master/slave S-R flip-flop does not use a dynamic-input indicator, because the flip-flop is not truly edge triggered. It is more like a latch that follows its input during the entire interval that C is 1 but changes its output to reflect the final latched value only when C goes to 0. In the symbol, a postponed-output indicator indicates that the output signal does not change until enable input C is negated. Flip-flops with this kind of behavior are sometimes called pulse-triggered flip-flops.

The operation of the master/slave S-R flip-flop is unpredictable if both S and R are asserted at the falling edge of C. In this case, just before the fall in edge, both the Q and QN outputs of the master latch are 1. When C goes to 0, the master latch's outputs change unpredictably and may even become metastable. At the same time, the slave latch opens up and propagates this garbage to the flip-flop output.

7.2.9 Master/Slave J-K Flip-Flop

The problem of what to do when S and R are asserted simultaneously is solved in a master/slave J-K flip-flop. The J and K inputs are analogous to S and R. However, as shown in Figure 7-26, asserting J asserts the master's S input only if the flip-flop's QN output is currently 1 (i.e. Q is 0), and asserting K asserts the master's R input only if Q is currently 1. Thus, if J and K are asserted simultaneously, the flip-flop goes to the opposite of its current state.

(a)



(b)

J	K	C	Q	QN
x	x	0	last Q	last QN
0	0	⎍	last Q	last QN
0	1	⎍	0	1
1	0	⎍	1	0
1	1	⎍	last QN	last Q

(c)

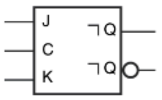


Figure 7-26
Master/slave J-K flip-flop: (a) circuit design using S-R latches; (b) function table; (c) logic symbol.

Figure 7-27 shows the functional behavior of a J-K master/slave flip-flop for a typical set of inputs. Note that the J and K inputs need not be asserted at the end of the triggering pulse for the flip-flop output to change at that time. In fact, because of the gating on the master latch's S and R inputs, it is possible for the flip-flop output to change to 1 even though K and not J is asserted at the end of the triggering pulse. This behavior, known as 1s catching, is illustrated in the second to last triggering pulse in the figure. An analogous behavior known as 0s catching is illustrated in the last triggering pulse. Because of this behavior, the J and K inputs of a J-K master/slave flip-flop must be held valid during the entire interval that C is 1.

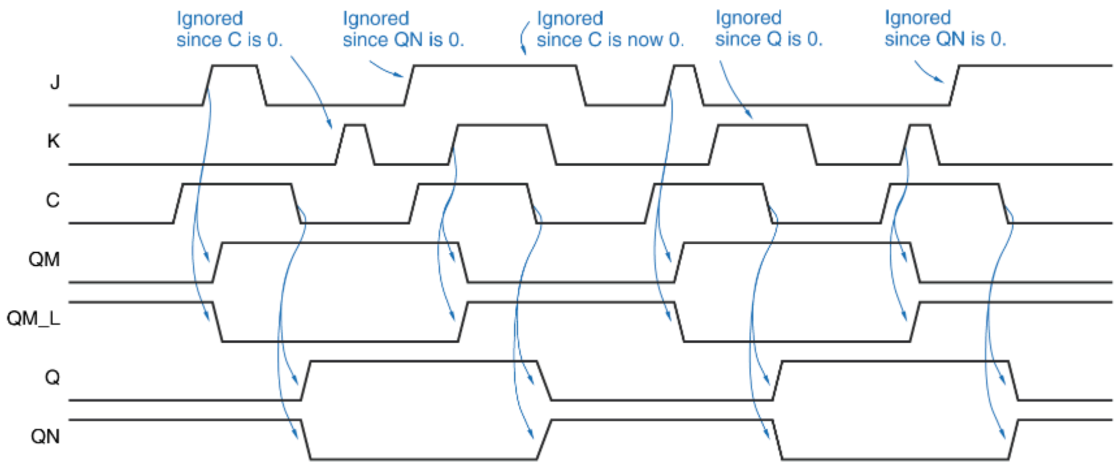
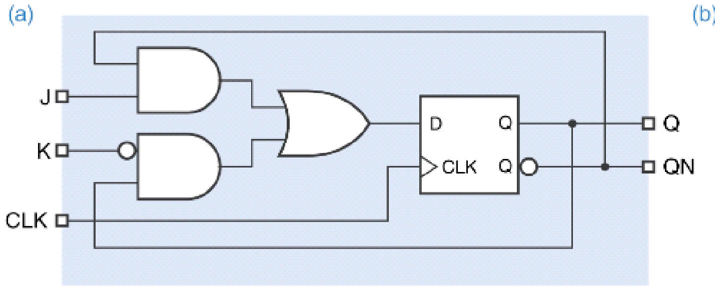


Figure 7-27
Internal and functional behavior of a master/slave J-K flip-flop.

7.2.10 Edge-Triggered J-K Flip-Flop

The problem of 1s and 0s catching is solved in an edge-triggered J-K flip-flop, whose functional equivalent is shown in Figure 7-28.

(a)



(b)

J	K	CLK	Q	QN
x	x	0	last Q	last QN
x	x	1	last Q	last QN
0	0	↑	last Q	last QN
0	1	↑	0	1
1	0	↑	1	0
1	1	↑	last QN	last Q

(c)

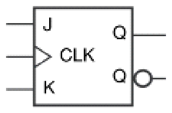


Figure 7-28
Edge-triggered J-K flip-flop: (a) equivalent function using an edge-triggered D flip-flop; (b) function table; (c) logic symbol.

Using an edge-triggered D flip-flop internally, the edge-triggered J-K flip-flop samples its inputs at the rising edge of the clock and produces its next output according to the "characteristic equation"

Q* = J • Q' + K' • Q (see Section 7.3.3)

Typical functional behavior of an edge-triggered J-K flip-flop is shown in Figure 7-29. Like the D input of an edge-triggered D flip-flop, the J and K inputs of a J-K flip-flop must meet published setup- and hold-time specifications with respect to the triggering clock edge for proper operation.

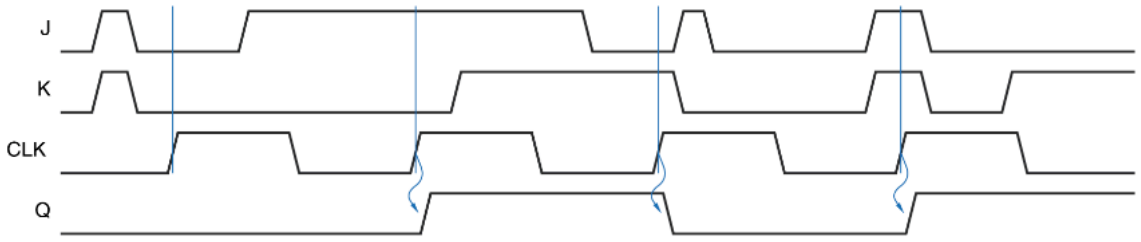


Figure 7-29
Functional behavior of a positive-edge-triggered J-K flip-flop.

Because they eliminate the problems of 1s and 0s catching and of simultaneously asserting both control inputs, edge-triggered J-K flip-flops have largely eliminated the older pulse-triggered types. The 74x109 is a TTL positive-edge triggered J-K' flip-flop with an active-low K input (see Figure 7-30).

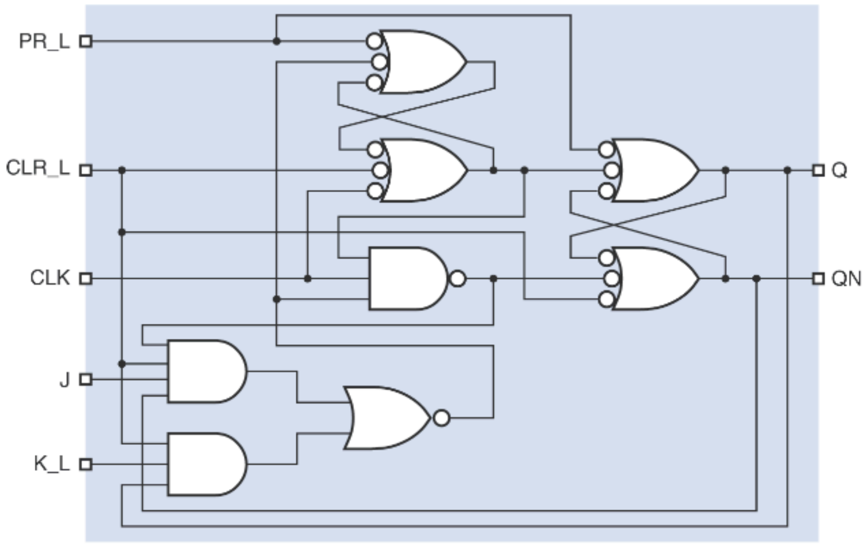


Figure 7-30
Internal logic diagram for the 74LS109 positive-edge-triggered J-$\overline{K}$ flip-flop.

J-K flip-flops aren't used much nowadays, but they're sometimes used in clocked synchronous state machines. As we'll explain in Section 7.4.5, the next-state logic for J-K flip-flops is sometimes simpler than for D flip-flops. However, most state machines are still designed using D flip-flops, because the design methodology is a bit simpler and because most sequential programmable logic devices contain D, not J-K, flip-flops. Therefore, we will focus most of our attention on D flip-flops.

7.2.11 T Flip-Flop

A T (Toggle) flip-flop changes state on every tick of the clock. Figure 7-31 shows the symbol and illustrates the behavior of a positive-edge-triggered T flip-flop. Notice that the signal on the flip-flop's Q output has precisely half the frequency of the T input.

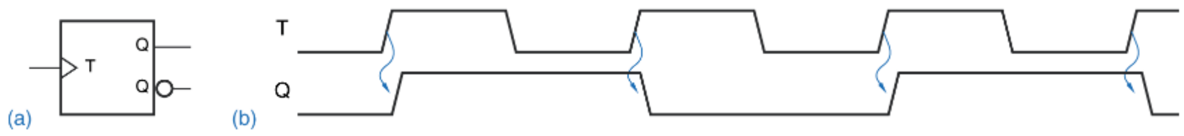


Figure 7-31
Positive-edge-triggered T flip-flop: (a) logic symbol; (b) functional behavior.

Figure 7-32 shows how to obtain a T flip-flop from a D or J-K flip-flop. T flip-flops are most often used in counters and frequency dividers, as we'll see in Section 8.4.

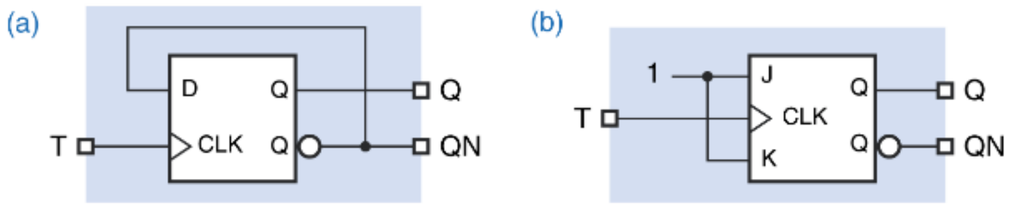


Figure 7-32
Possible circuit designs for a T flip-flop: (a) using a D flip-flop; (b) using a J-K flip-flop.

In many applications of T flip-flops, the flip-flop need not be toggled on every clock tick. Such applications can use a T flip-flop with enable. As shown Figure 7-33, the flip-flop changes state at the triggering edge of the clock only if the enable signal EN is asserted. Like the D, J, and K inputs on other edge-triggered flip-flops, the EN input must meet specified setup and hold times with respect to the triggering clock edge.

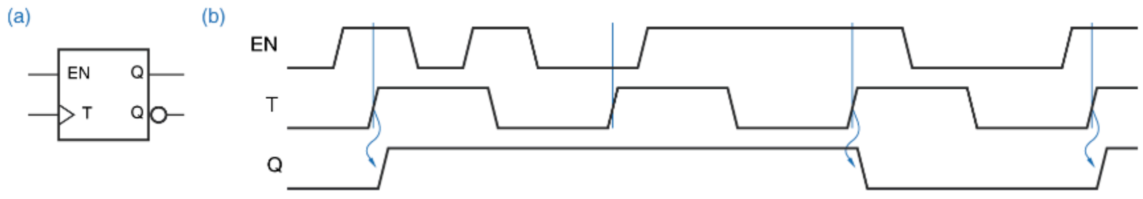


Figure 7-33
Positive-edge-triggered T flip-flop with enable: (a) logic symbol; (b) functional behavior.

The circuits of Figure 7-32 are easily modified to provide an EN input, as shown in Figure 7-34.

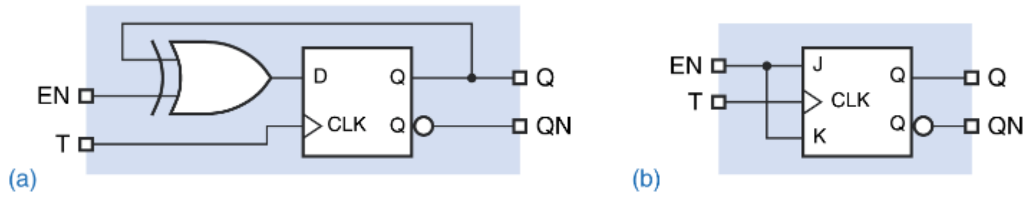


Figure 7-34
Possible circuits for a T flip-flop with enable: (a) using a D flip-flop; (b) using a J-K flip-flop.

7.3 Clocked Synchronous State-Machine Analysis

Although latches and flip-flops, the basic building blocks of sequential circuits, are themselves feedback sequential circuits that can be formally analyzed (in Section 7.9), we'll first study the operation of clocked synchronous state machines, since they are the easiest to understand. "State machine" is a generic name given to these sequential circuits; "clocked" refers to the fact that their storage elements (flip-flops) employ a clock input; and "synchronous" means that all of the flip-flops use the same clock signal. Such a state machine changes state only when a triggering edge occurs on the clock signal.

7.3.1 State-Machine Structure

Figure 7-35 shows the general structure of a clocked synchronous state machine. The state memory is a set of n flip-flops that store the current state of the machine, and it has 2^n distinct states. The flip-flops are all connected to a common clock signal that causes them to change state at each tick of the clock. What constitutes a tick depends on the flip-flop type (edge triggered, pulse trigger etc.). For the positive-edge-triggered D flip-flops considered in this section, a tick is the rising edge of the clock signal.

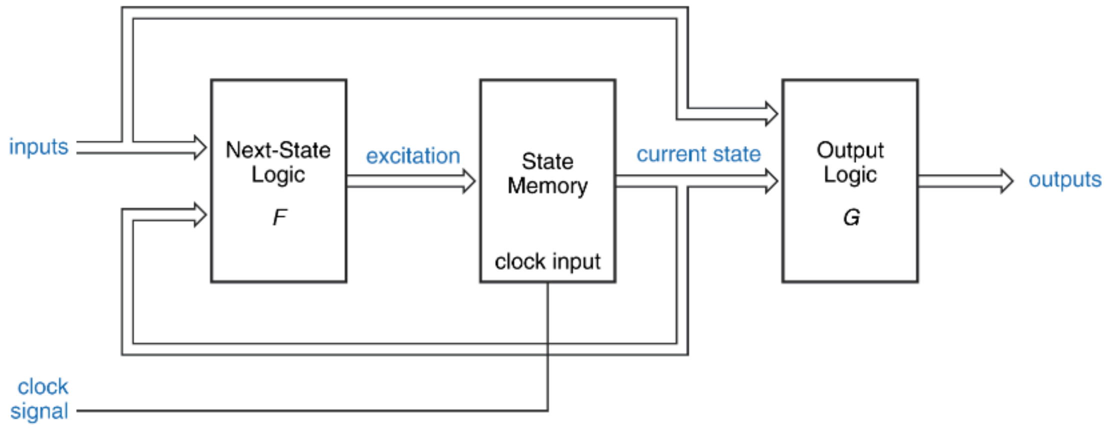


Figure 7-35
Clocked synchronous state-machine structure (Mealy machine).

The next state of the state machine in Figure 7-35 is determined by the next-state logic F as a function of the current state and input. The output logic G determines the output as a function of the current state and input. Both F and G are strictly combinational logic circuits. We can write

Next state = F(current state, input)

Output = G(current state, input)

State machines may use positive-edge-triggered D flip-flops for their state memory, in which case a tick occurs at each rising edge of the clock signal. It is also possible for the state memory to use negative-edge-triggered D flip-flops, D latches, or J-K flip-flops. However, inasmuch as most state machines are designed using PLDs, CPLDs, FPGAs, or ASICs with positive-edge-triggered D flip-flops, that's what we'll concentrate on.

7.3.2 Output Logic

A sequential circuit whose output depends on both state and input as shown in Figure 7-35 is called a Mealy machine. In some sequential circuits the output depends on the state alone:

Output = G(current state)

Such a circuit is called a Moore machine, and its general structure is shown in Figure 7-36.

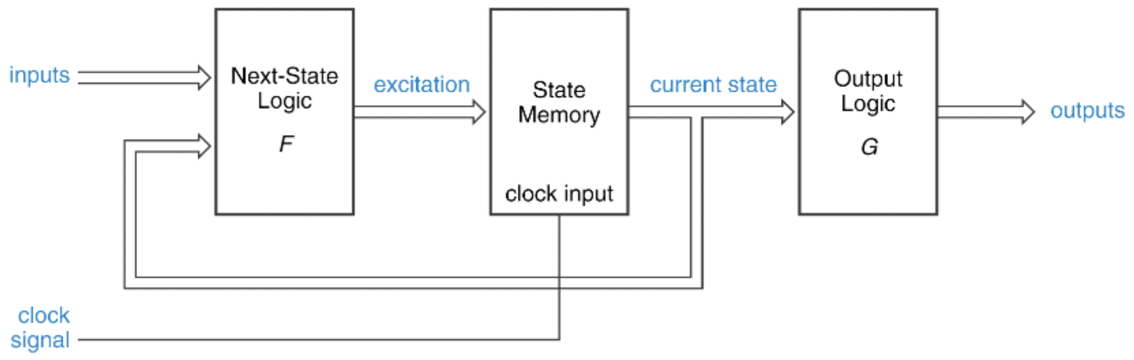


Figure 7-36
Clocked synchronous state-machine structure (Moore machine).

Obviously, the only difference between the two state-machine models is in how outputs are generated. In practice, most state machines must be categorized as Mealy machines, because they have one or more Mealy-type outputs that depend on input as well as state. However, many of these same machines also have one or more Moore-type outputs that depend only on state.

In the design of high-speed circuits, it is often necessary to ensure that state-machine outputs are available as early as possible and do not change during each clock period. One way to get this behavior is to encode the state so that the state variables themselves serve as outputs. We call this an output-coded state assignment; it yields a Moore machine in which the output logic of Figure 7-36 is nothing more than wires.

Another approach is to design the state machine so that the outputs during one clock period depend on the state and inputs during the previous clock period. We call these pipelined outputs, and they are obtained by attaching another stage of memory (flip-flops) to a machine's outputs, as shown for a Mealy machine in Figure 7-37.

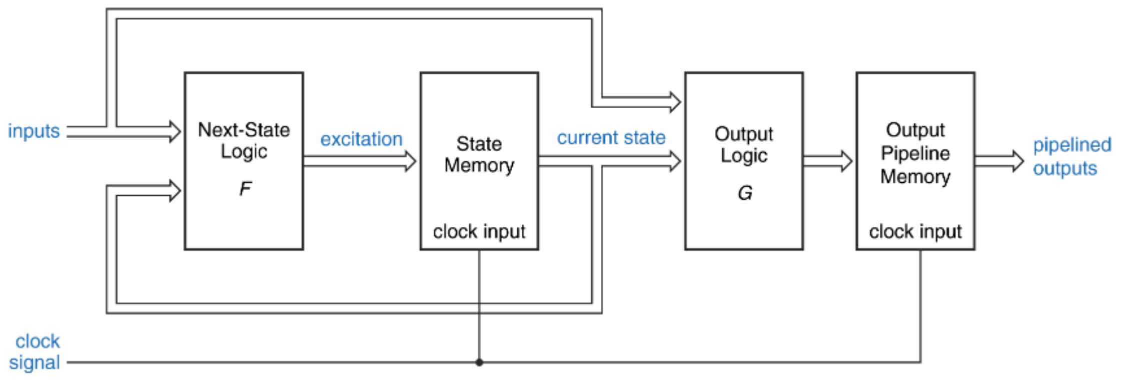


Figure 7-37
Mealy machine with pipelined outputs.

With appropriate circuit or drawing manipulations, you can map one state-machine model into another. For example, you could declare the flip-flops that produce pipelined outputs from a Mealy machine to be part of its state memory and thereby obtain a Moore machine with an output-coded state assignment

The exact classification of a state machine into one style or another is not critical. What's important is how you think about output structure and how it satisfies your overall design objectives, including timing and flexibility. For example, pipelined outputs are great for fast timing, but you can use them only in situations where you can figure out the desired next output value one clock period in advance. In any given application you may use different styles for different output signals.

7.3.3 Characteristic Equations

The functional behavior of a latch or flip-flop can be described formally by a characteristic equation that specifies the flip-flop's next state as a function of its current state and inputs. The characteristic equations of the flip-flops in Section 7.2 are listed Table 7-1. By convention, the * suffix in Q* means "the next value of Q." Notice that the characteristic equation does not describe detailed timing behavior of the device (latching vs. edge-triggered, etc.), only the functional response to the control inputs. This simplified description is useful in the analysis of state-machines.

Device Type	Characteristic Equation
S-R latch	$Q* = S + R' \cdot Q$
D latch	$Q* = D$
Edge-triggered D flip-flop	$Q* = D$
D flip-flop with enable	$Q* = EN \cdot D + EN' \cdot Q$
Master/slave S-R flip-flop	$Q* = S + R' \cdot Q$
Master/slave J-K flip-flop	$Q* = J \cdot Q' + K' \cdot Q$
Edge-triggered J-K flip-flop	$Q* = J \cdot Q' + K' \cdot Q$
T flip-flop	$Q* = Q'$
T flip-flop with enable	$Q* = EN \cdot Q' + EN' \cdot Q$

Table 7-1

Latch and flip-flop characteristic equations.

7.3.4 Analysis of State Machines with D Flip-Flops

Consider the formal definition of a state machine that we gave previously:

Next state = F(current state, input)

Output = G(current state, input)

Recalling our notion that "state" embodies all we need to know about the past history of the circuit, the first equation tells us that what we next need to know can be determined from what we currently know and the current input. The second equation tells us that the current output can be determined from the same information. The goal of sequential circuit analysis is to determine the next-state and output functions so that the behavior of a circuit can be predicted. The analysis of a clocked synchronous state machine has three basic steps:

1. **Determine the next-state and output functions F and G.**
2. **Use F and G to construct a state/output table that completely specifies the next state and output of the circuit for every possible combination of current state and input.**
3. **Optional - draw a state diagram that presents the information from the previous step in graphical form.**

Figure 7-38 shows a simple state machine with two positive-edge-edge triggered D flip-flops. To determine the next-state function F, we must first consider the behavior of the state memory. At the rising edge of the clock signal, each D flip-flop samples its D input and transfers this value to its Q output; since the characteristic equation of a D flip-flop is Q* = D. Therefore, to determine the next value of Q (i.e., Q*), we must first determine the current value of D.

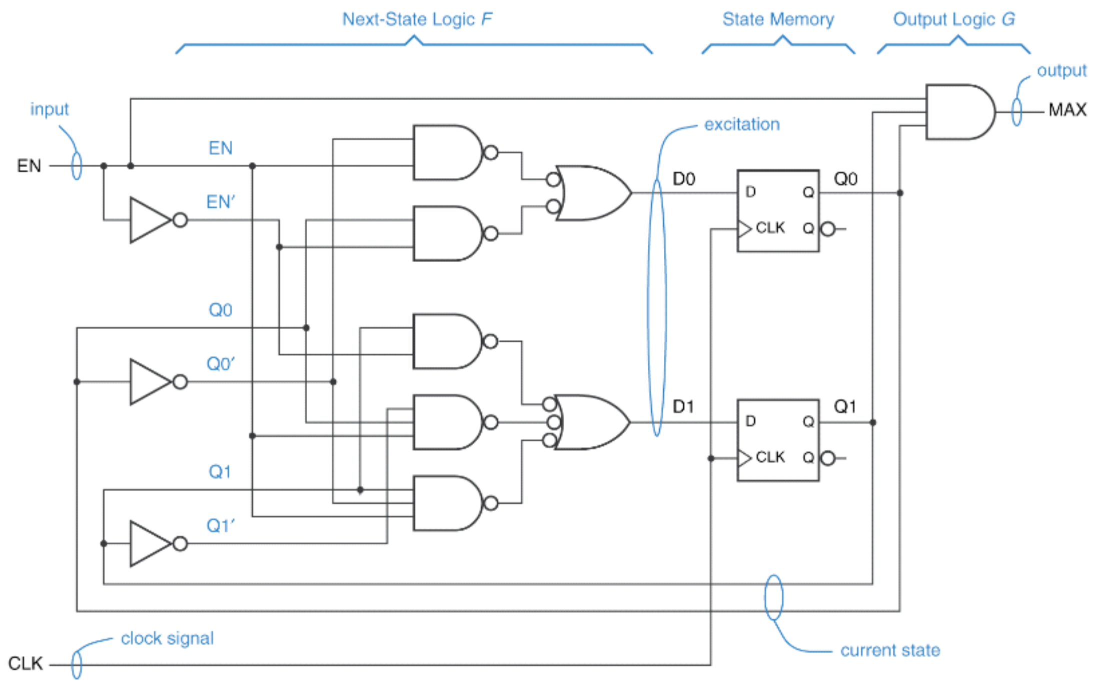


Figure 7-38
Clocked synchronous state machine using positive-edge-triggered D flip-flops.

In Figure 7-38 there are two D flip-flops, and we have named the signals on their outputs QO and Q1. These two outputs are the state variables; their value is the current state of the machine. We have named the signals on the corresponding D inputs DO and D1. These signals provide the excitation for the D flip-flops at each clock tick. Logic equations that express the excitation signals as functions of the current state and input are called excitation equations and can be derived directly from the circuit diagram:

DO = (QO • EN') + (QO' • EN)

D1 = (Q1 • EN') + (Q1' • QO • EN) + (Q1 • QO' • EN)

As noted previously, the next value of a state variable after a clock tick is denoted by appending a star to the state-variable name, for example, Q0* or Q1*. Using the characteristic equation of D flip-flops, Q* = D, we can describe the next-state function of the example machine with equations for the next value of the state variables:

$$\mathbf{QO^* = DO}$$
$$\mathbf{Q1^* = D1}$$

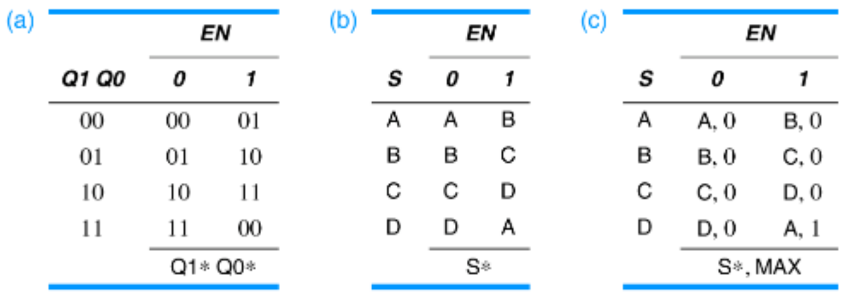
Substituting the excitation equations for DO and D1, we can write

$$\mathbf{QO^* = (QO \bullet EN') + (QO' \bullet EN)}$$
$$\mathbf{Q1^* = (Q1 \bullet EN') + (Q1' \bullet Q0 \bullet EN) + (Q1 \bullet QO' \bullet EN)}$$

These equations, which express the next value of the state variables as a function of current state and input, are called transition equations.

For each combination of current state and input value, the transition equations predict the next state. Each state is described by two bits, the current values of Q0 and Q1: (Q1 QO) = 00, 01, 10, or 11. The reason for "arbitrarily" picking the order (Q1 QO) instead of (QO Q1) will become apparent shortly. For each state, our example machine has just two possible input values, EN = 0 or EN = 1, so there are a total of 8 state/input combinations. In general, a machine with s state bits and i inputs has 2^{s+i} state/input combinations.

Table 7-2(a) shows a transition table that is created by evaluating the transition equations for every possible state/input combination.



(a)

	EN	
Q1 Q0	***0***	***1***
00	00	01
01	01	10
10	10	11
11	11	00
	Q1* Q0*	

(b)

	EN	
S	***0***	***1***
A	A	B
B	B	C
C	C	D
D	D	A
	S*	

(c)

	EN	
S	***0***	***1***
A	A, 0	B, 0
B	B, 0	C, 0
C	C, 0	D, 0
D	D, 0	A, 1
	S*, MAX	

Table 7-2

Transition, state, and state/output tables for the state machine in Figure 7-38.

Traditionally, a transition table lists the states along the left and the input combinations along the top of the table, as shown in the example.

The function of our example machine is apparent from its transition table - it is a 2-bit binary counter with an enable input EN. When EN = 0, the machine maintains its current count, but when EN = 1, the count advances by 1 at each clock tick, rolling over to 00 when it reaches a maximum value of 11.

If we wish, we may assign alphanumeric state names to each state. The simplest naming is 00 = A, 01 = B, 10 = C, and 11 = D. Substituting the state names for combinations of Q1 and QO (and Q1 * and QO*) in Table 7-2(a) produces the state table shown in (b) above. Here "S" denotes the current state and "S*" denotes the next state of the machine. A state table is usually easier to understand than a transition table, because in complex machines we can use state names that have meaning. However, a state table contains less information than a transition table because it does not indicate the binary values assumed by the state variables in each named state.

Once a state table is produced, we have only the output logic of the machine left to analyze. In the example machine there is only a single output signal, and it is a function of both current state and input (i.e. a Mealy machine). So we can write a single output equation:

$$\mathbf{MAX = Q1 \cdot QO \cdot EN}$$

The output behavior predicted by this equation can be combined with the next-state information to produce a state/output table as shown in Table 7-2(c) above.

State/output tables for Moore machines are slightly simpler. For example, in the circuit of Figure 7-38 suppose we removed the EN signal from the AND gate that produces the MAX output, producing a Moore-type output MAXS. Then MAXS is a function of the state only, and the state/output table can list MAXS in a single column, independent of the input values. This is shown in Table 7-3.

	EN		
S	0	1	MAXS
A	A	B	0
B	B	C	0
C	C	D	0
D	D	A	1
	S*		

Table 7-3
State/output table for a Moore machine.

A state diagram presents the information from the state/output table in a graphical format. It has one circle (or node) for each state and an arrow (or directed arc) for each transition. Figure 7-39 shows the state diagram for our example state machine. The letter inside each circle is a state name. Each arrow leaving a given state points to the next state for a given input combination; it also shows the output value produced in the given state for that input combination.

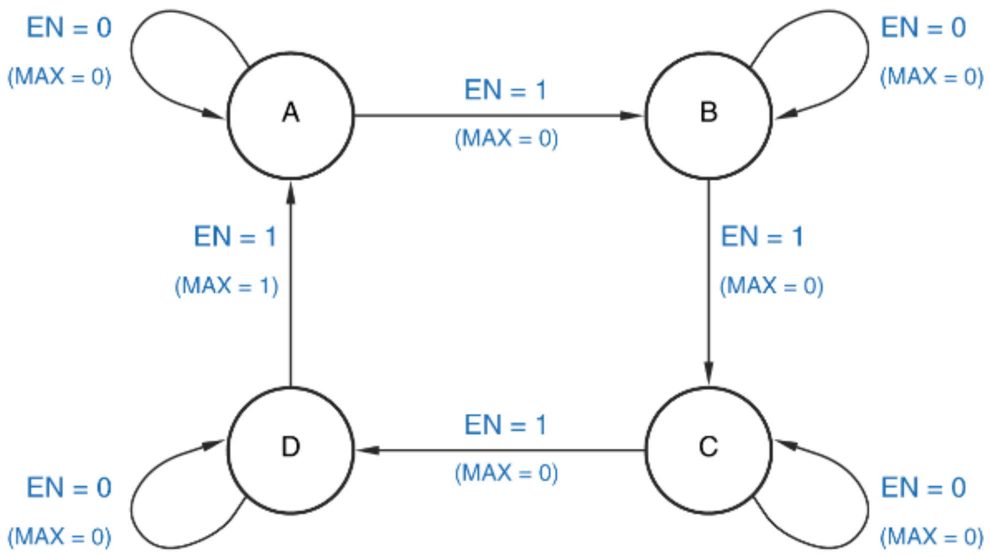


Figure 7-39
State diagram corresponding to the state machine of Table 7-2.

CLARIFICATION - the state-diagram notation for output values in Mealy machines is a little misleading. You should remember that the listed output value is produced continuously when the machine is in the indicated state and has the indicated input, not just during the transition to the next state.

The state diagram for a Moore machine can be somewhat simpler. In this case, the output values can be shown inside each state circle, since they are functions of state only. The state diagram for a Moore machine using this convention is shown in Figure 7-40.

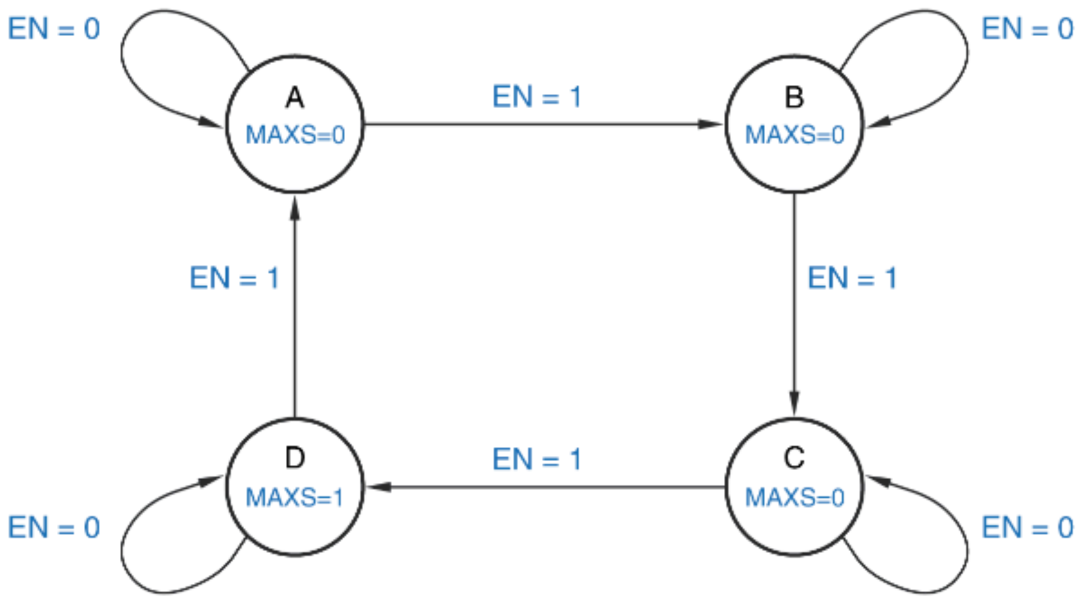


Figure 7-40
State diagram corresponding to the state machine of Table 7-3.

The original logic diagram of our example state machine, Figure 7-38, was laid out to match our conceptual model of a Mealy machine. However, nothing requires us to group the next-state logic, state memory, and output logic in this way. Figure 7-41 shows another logic diagram for the same state machine. To analyze this circuit, the designer can still extract the required information from the diagram as drawn. The only circuit difference in the new diagram is that we have used the flip-flops' QN outputs (which are normally the complement of Q) to save a couple of inverters.

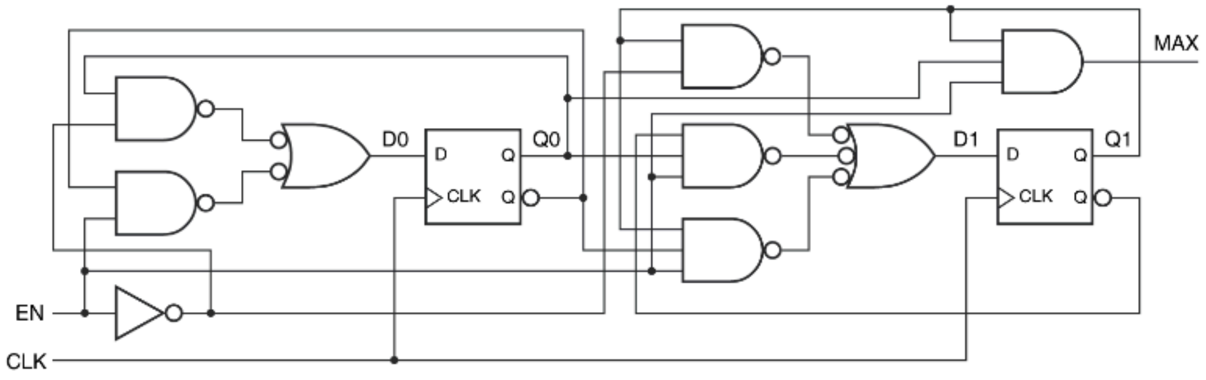


Figure 7-41
Redrawn logic diagram for a clocked synchronous state machine.

Using the transition, state, and output tables, we can construct a timing diagram that shows the behavior of a state machine for any desired starting state and input sequence. For example, Figure 7-42 shows the behavior of our example machine with a starting state of 00 (A) and a particular pattern on the EN input.

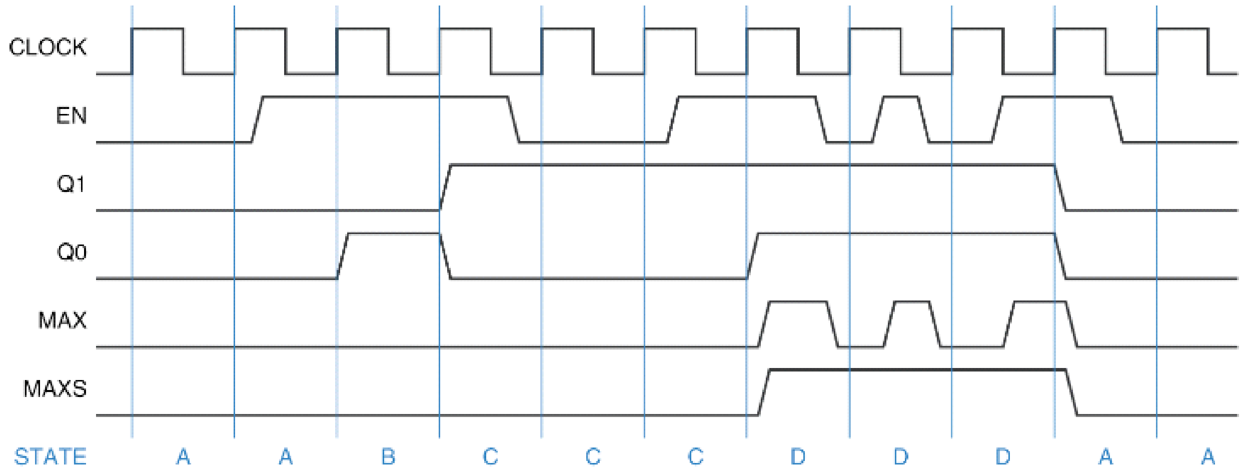


Figure 7-42
Timing diagram for example state machine

Notice that the value of the EN input affects the next state only at the rising edge of the CLOCK input; that is, the counter counts only if EN = 1 at the rising edge of CLOCK. On the other hand, since MAX is a Mealy-type output, its value is affected by EN at all times. If we also provide a Moore-type output MAXS as suggested in the text, its value depends only on state as shown in the figure.

The timing diagram is drawn in a way that shows changes in the MAX and MAXS outputs occurring slightly later than the state and input changes that cause them, reflecting the combinational-logic delay of the output circuits.

In summary, the detailed steps for analyzing a clocked synchronous state-machine are as follows:

1. **Determine the excitation equations for the flip-flop control inputs.**
2. **Substitute the excitation equations into the flip-flop characteristic equations to obtain transition equations.**
3. **Use the transition equations to construct a transition table.**
4. **Determine the output equations.**
5. **Add output values to the transition table for each state (Moore) or state/input combination (Mealy) to create a transition/output table.**
6. **Name the states and substitute state names for state-variable combinations in the transition/output table to obtain a state/output table.**
7. **Optional - draw a state diagram corresponding to the state/output table.**

We'll go through this complete sequence of steps to analyze another clocked synchronous state machine, shown in Figure 7-43.

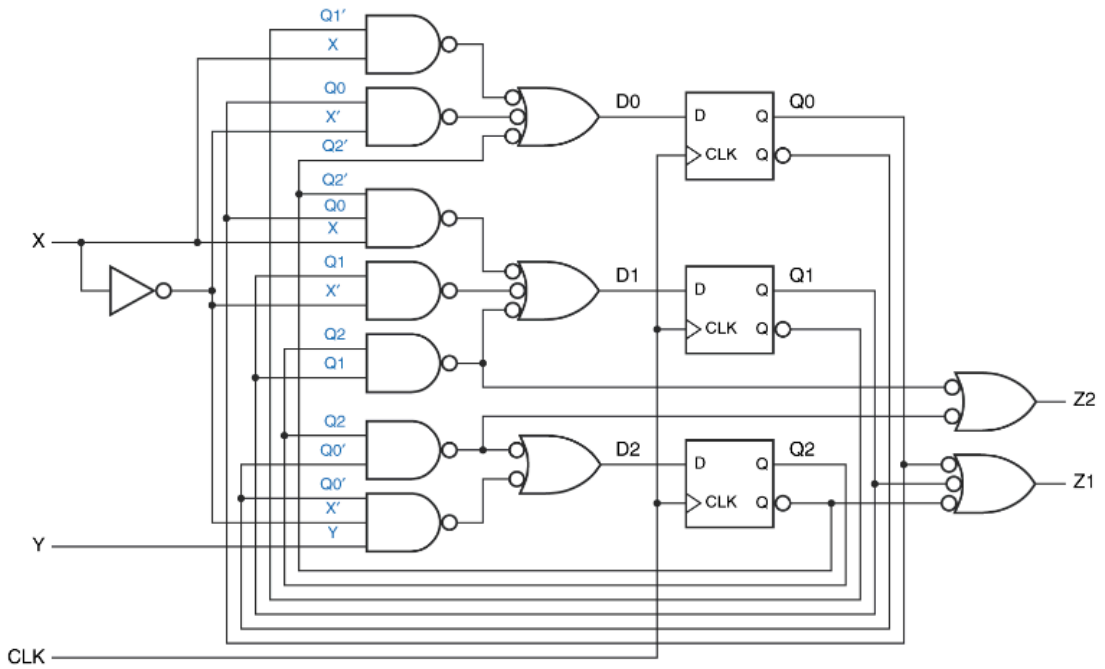


Figure 7-43
A clocked synchronous state machine with three flip-flops and eight states.

Reading the logic diagram, we find that the excitation equations are as follows:

DO = (Q1' • X) + (QO • X') + Q2
D1 = (Q2' • QO • X) + (Q1 • X') + (Q2 • Q1)
D2 = (Q2 • QO') + (QO' • X' • Y)

Substituting into the characteristic equation for D flip-flops, we obtain the transition equations:

QO* = (Q1' • X) + (QO • X') + Q2
Q1* = (Q2' • QO • X) + (Q1 • X') + (Q2 • Q1)
Q2* = (Q2 • QO') + (QO' • X' • Y)

A transition table based on these equations is shown in Table 7-4(a).

(a)

	XY				
Q2 Q1 Q0	00	01	10	11	Z1 Z2
000	000	100	001	001	10
001	001	001	011	011	10
010	010	110	000	000	10
011	011	011	010	010	00
100	101	101	101	101	11
101	001	001	001	001	10
110	111	111	111	111	11
111	011	011	011	011	11
	Q2* Q1* Q0*				

(b)

	XY				
S	00	01	10	11	Z1 Z2
A	A	E	B	B	10
B	B	B	D	D	10
C	C	G	A	A	10
D	D	D	C	C	00
E	F	F	F	F	11
F	B	B	B	B	10
G	H	H	H	H	11
H	D	D	D	D	11
	S*				

Table 7-4

Transition/output and state/output tables for the state machine in Figure 7-43.

Reading the logic diagram, we can write two output equations:

Z1 = (Q2 + Q1') + QO'

Z2 = (Q2 • Q1) + (Q2 • QO')

The resulting output values are shown in the last column of (a). Assigning state names A-H, we obtain the state/output table shown in (b).

A state diagram for the example machine is shown in Figure 7-44. Since our example is a Moore machine, the output values are written with each state. Each arc is labeled with a transition expression; a transition is taken for input combinations for which the transition expression is 1. Transitions labeled “1” are always taken.

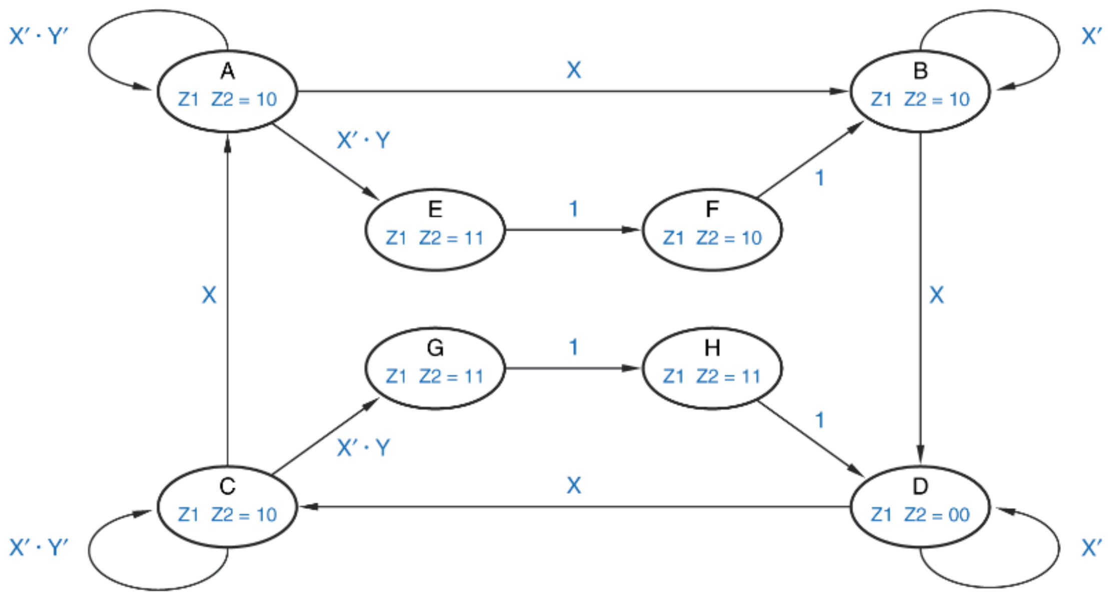


Figure 7-44
State diagram corresponding to Table 7-4.

The transition expressions on arcs leaving a particular state must be mutually exclusive and all-inclusive, as explained below:

1. **No two transition expressions can equal 1 for the same input combination, since a machine can't have two next states for one input combination.**
2. **For every possible input combination, some transition expression must equal 1, so that all next states are defined.**

Starting with the state table, a transition expression for a particular current state and next state can be written as a sum of minterms for the input combinations that cause that transition. If desired, the expression can then be minimized to give the information in a more compact form. Transition expressions are most useful in the design of state machines, where the expressions may be developed from the word description of the problem, as we'll show in Section 7.5.