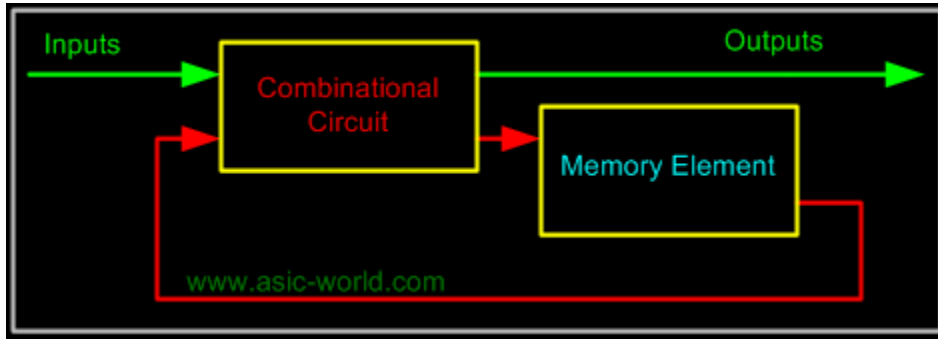


## UNIT- IV

### Synchronous Sequential Logic

#### ● Introduction

Digital electronics is classified into combinational logic and sequential logic. Combinational logic output depends on the inputs levels, whereas sequential logic output depends on stored levels and also the input levels.



The memory elements are devices capable of storing binary info. The binary info stored in the memory elements at any given time defines the state of the sequential circuit. The input and the present state of the memory element determines the output. Memory elements next state is also a function of external inputs and present state. A sequential circuit is specified by a time sequence of inputs, outputs, and internal states.

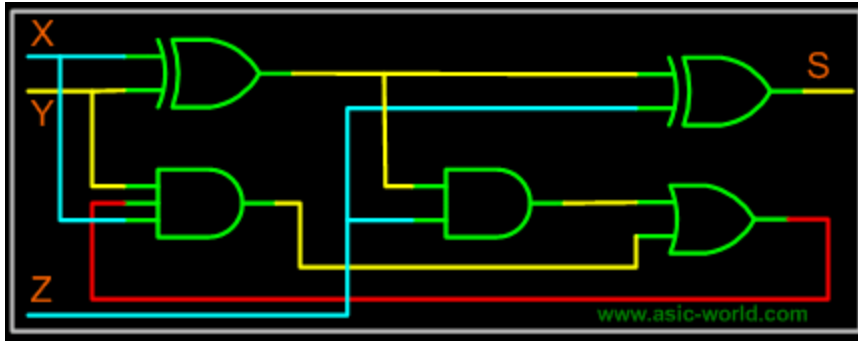
There are two types of sequential circuits. Their classification depends on the timing of their signals:

- Synchronous sequential circuits
- Asynchronous sequential circuits

#### ❖ Asynchronous sequential circuit

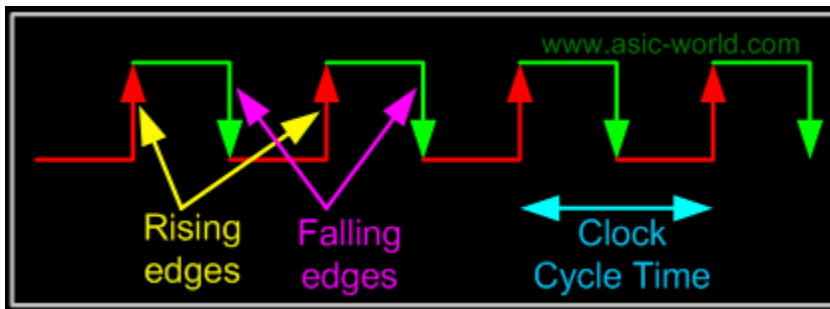
This is a system whose outputs depend upon the order in which its input variables change and can be affected at any instant of time.

Gate-type asynchronous systems are basically combinational circuits with feedback paths. Because of the feedback among logic gates, the system may, at times, become unstable. Consequently they are not often used.



## ❖ Synchronous sequential circuits

This type of system uses storage elements called flip-flops that are employed to change their binary value only at discrete instants of time. Synchronous sequential circuits use logic gates and flip-flop storage devices. Sequential circuits have a clock signal as one of their inputs. All state transitions in such circuits occur only when the clock value is either 0 or 1 or happen at the rising or falling edges of the clock depending on the type of memory elements used in the circuit. Synchronization is achieved by a timing device called a clock pulse generator. Clock pulses are distributed throughout the system in such a way that the flip-flops are affected only with the arrival of the synchronization pulse. Synchronous sequential circuits that use clock pulses in the inputs are called clocked-sequential circuits. They are stable and their timing can easily be broken down into independent discrete steps, each of which is considered separately.



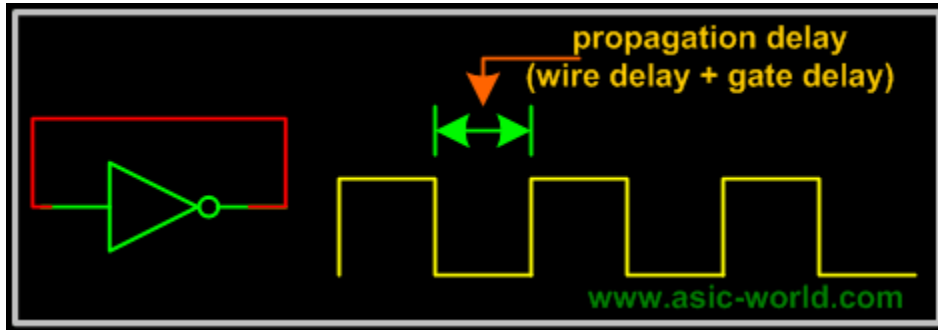
A clock signal is a periodic square wave that indefinitely switches from 0 to 1 and from 1 to 0 at fixed intervals. Clock cycle time or clock period: the time interval between two consecutive rising or falling edges of the clock.

Clock Frequency =  $1 / \text{clock cycle time}$  (measured in cycles per second or Hz)

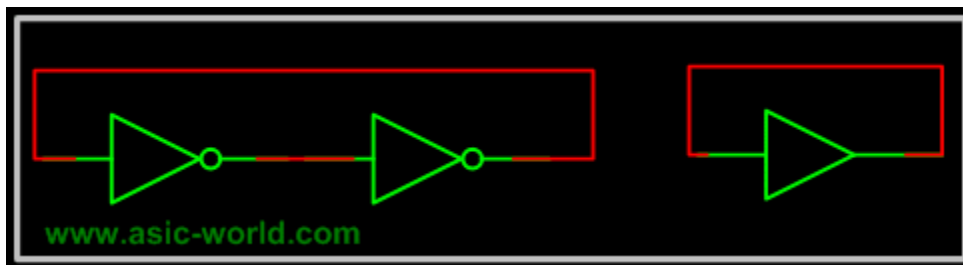
## ● Concept of Sequential Logic

A sequential circuit as seen in the last page, is combinational logic with some feedback to maintain its current value, like a memory cell. To understand the basics let's consider the basic feedback logic circuit below, which is a simple NOT gate whose

output is connected to its input. The effect is that output oscillates between HIGH and LOW (i.e. 1 and 0). Oscillation frequency depends on gate delay and wire delay. Assuming a wire delay of 0 and a gate delay of 10ns, then oscillation frequency would be (on time + off time = 20ns) 50Mhz.



The basic idea of having the feedback is to store the value or hold the value, but in the above circuit, output keeps toggling. We can overcome this problem with the circuit below, which is basically cascading two inverters, so that the feedback is in-phase, thus avoids toggling. The equivalent circuit is the same as having a buffer with its output connected to its input.



But there is a problem here too: each gate output value is stable, but what will it be? Or in other words buffer output can not be known. There is no way to tell. If we could know or set the value we would have a simple 1-bit storage/memory element.

## ● Latches and Flip-Flops

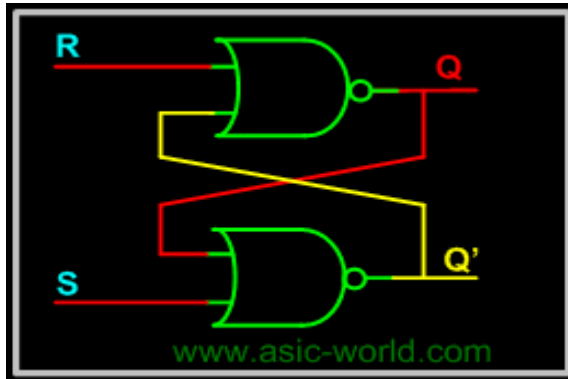
There are two types types of sequential circuits.

- Asynchronous Circuits.
- Synchronous Circuits.

As seen in last section, Latches and Flip-flops are one and the same with a slight variation: Latches have level sensitive control signal input and Flip-flops have edge sensitive control signal input. Flip-flops and latches which use this control signals are called synchronous circuits. So if they don't use clock inputs, then they are called asynchronous circuits.

## RS Latch

RS latch have two inputs, S and R. S is called set and R is called reset. The S input is used to produce HIGH on Q ( i.e. store binary 1 in flip-flop). The R input is used to produce LOW on Q (i.e. store binary 0 in flip-flop). Q' is Q complementary output, so it always holds the opposite value of Q. The output of the S-R latch depends on current as well as previous inputs or state, and its state (value stored) can change as soon as its inputs change. The circuit and the truth table of RS latch is shown below.



Truth Table:

S	R	Q	Q+
0	0	0	0
0	0	1	1
0	1	X	0
1	0	X	1
1	1	X	0

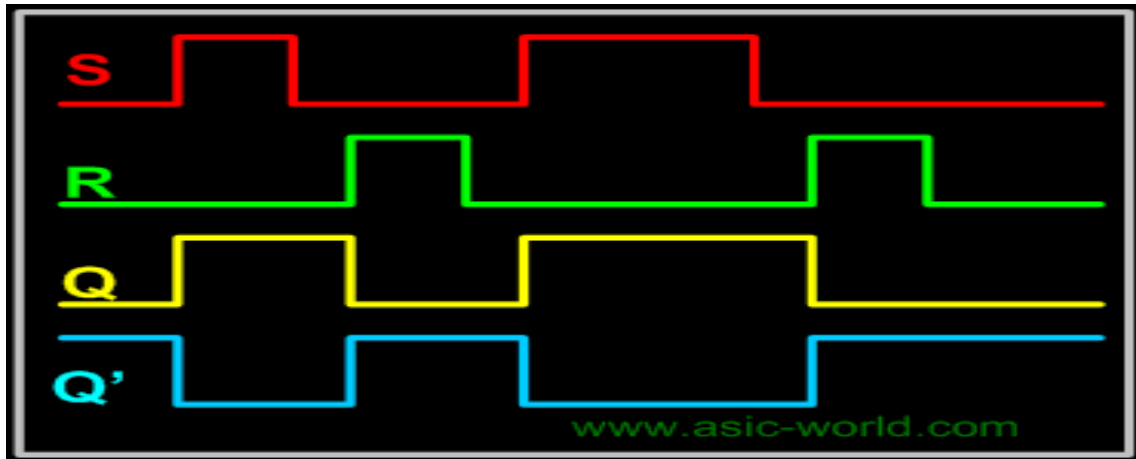
The operation has to be analyzed with the 4 inputs combinations together with the 2 possible previous states.

- **When S = 0 and R = 0:** If we assume Q = 1 and Q' = 0 as initial condition, then output Q after input is applied would be  $Q = (R + Q')' = 1$  and  $Q' = (S + Q)' = 0$ . Assuming Q = 0 and Q' = 1 as initial condition, then output Q after the input applied would be  $Q = (R + Q')' = 0$  and  $Q' = (S + Q)' = 1$ . So it is clear that when both S and R inputs are LOW, the output is retained as before the application of inputs. (i.e. there is no state change).
- **When S = 1 and R = 0:** If we assume Q = 1 and Q' = 0 as initial condition, then output Q after input is applied would be  $Q = (R + Q')' = 1$  and  $Q' = (S + Q)' = 0$ . Assuming Q = 0 and Q' = 1 as initial condition, then output Q after the input applied would be  $Q = (R + Q')' = 1$  and  $Q' = (S + Q)' = 0$ . So in simple words when S is HIGH and R is LOW, output Q is HIGH.
- **When S = 0 and R = 1:** If we assume Q = 1 and Q' = 0 as initial condition, then output Q after input is applied would be  $Q = (R + Q')' = 0$  and  $Q' = (S + Q)' = 1$ .

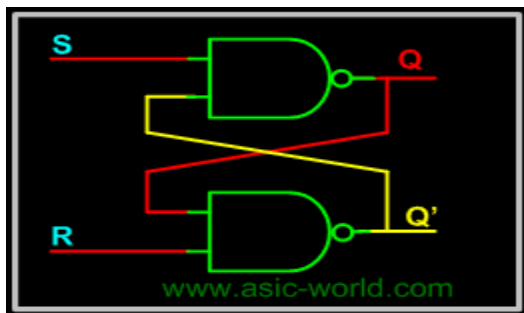
Assuming  $Q = 0$  and  $Q' = 1$  as initial condition, then output  $Q$  after the input applied would be  $Q = (R + Q')' = 0$  and  $Q' = (S + Q)' = 1$ . So in simple words when  $S$  is LOW and  $R$  is HIGH, output  $Q$  is LOW.

- **When  $S = 1$  and  $R = 1$  :** No matter what state  $Q$  and  $Q'$  are in, application of 1 at input of NOR gate always results in 0 at output of NOR gate, which results in both  $Q$  and  $Q'$  set to LOW (i.e.  $Q = Q'$ ). LOW in both the outputs basically is wrong, so this case is invalid.

The waveform below shows the operation of NOR gates based RS Latch.



It is possible to construct the RS latch using NAND gates (of course as seen in Logic gates section). The only difference is that NAND is NOR gate dual form (Did I say that in Logic gates section?). So in this case the  $R = 0$  and  $S = 0$  case becomes the invalid case. The circuit and Truth table of RS latch using NAND is shown below.

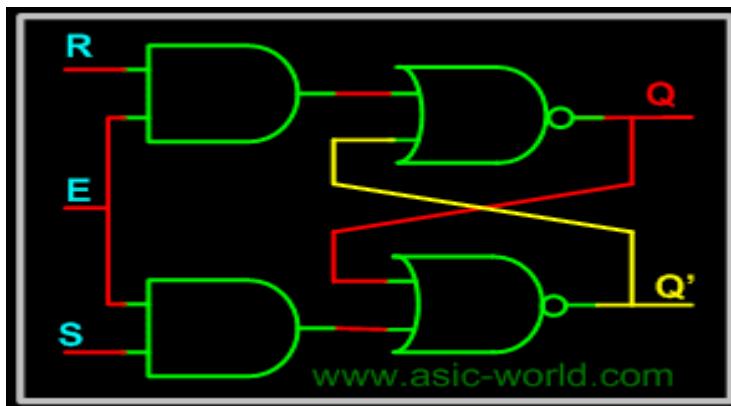


S	R	Q	Q+
1	1	0	0
1	1	1	1
0	1	X	0
1	0	X	1
0	0	X	1

If you look closely, there is no control signal (i.e. no clock and no enable), so this kind of latches or flip-flops are called asynchronous logic elements. Since all the sequential circuits are built around the RS latch, we will concentrate on synchronous circuits and not on asynchronous circuits.

### ❖ RS Latch with Clock

We have seen this circuit earlier with two possible input configurations: one with level sensitive input and one with edge sensitive input. The circuit below shows the level sensitive RS latch. Control signal "Enable" E is used to gate the input S and R to the RS Latch. When Enable E is HIGH, both the AND gates act as buffers and thus R and S appears at the RS latch input and it functions like a normal RS latch. When Enable E is LOW, it drives LOW to both inputs of RS latch. As we saw in previous page, when both inputs of a NOR latch are low, values are retained (i.e. the output does not change).

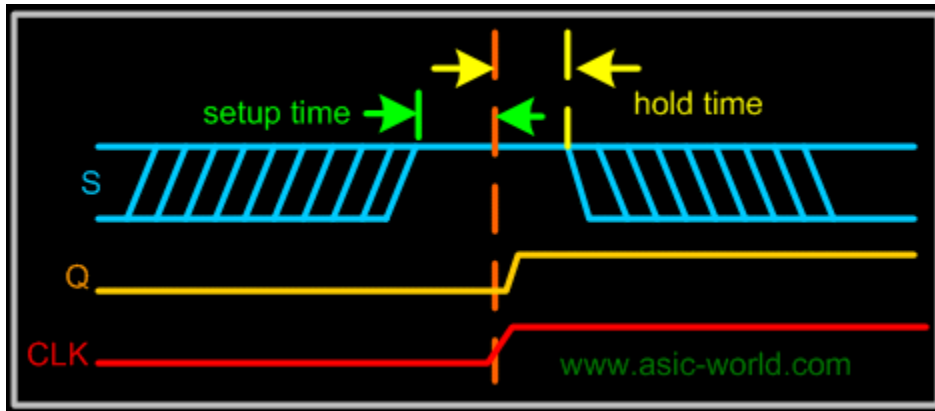


### ❖ Setup and Hold Time

For synchronous flip-flops, we have special requirements for the inputs with respect to clock signal input. They are

- **Setup Time:** Minimum time period during which data must be stable before the clock makes a valid transition. For example, for a posedge triggered flip-flop, with a setup time of 2 ns, Input Data (i.e. R and S in the case of RS flip-flop) should be stable for at least 2 ns before clock makes transition from 0 to 1.
- **Hold Time:** Minimum time period during which data must be stable after the clock has made a valid transition. For example, for a posedge triggered flip-flop, with a hold time of 1 ns, Input Data (i.e. R and S in the case of RS flip-flop) should be stable for at least 1 ns after clock has made transition from 0 to 1.
- If data makes transition within this setup window and before the hold window, then the flip-flop output is not predictable, and flip-flop enters what is known as **meta stable state**. In this state flip-flop output oscillates between 0 and 1. It takes some time for the flip-flop to settle down. The whole process is called **metastability**. You could refer to tidbits section to know more information on this topic.

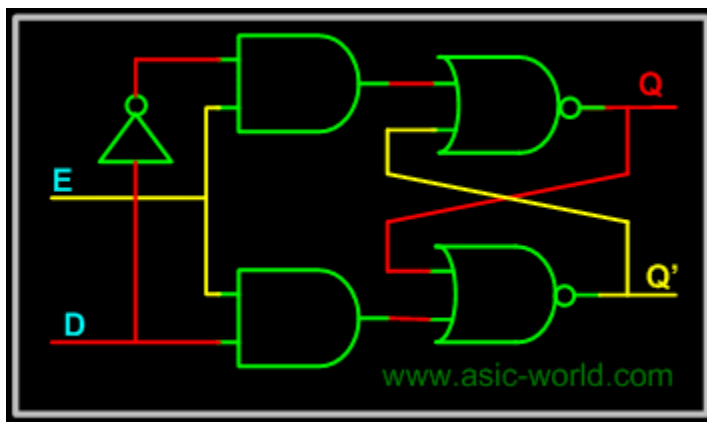
The waveform below shows input S (R is not shown), and CLK and output Q (Q' is not shown) for a SR posedge flip-flop.



### D Latch

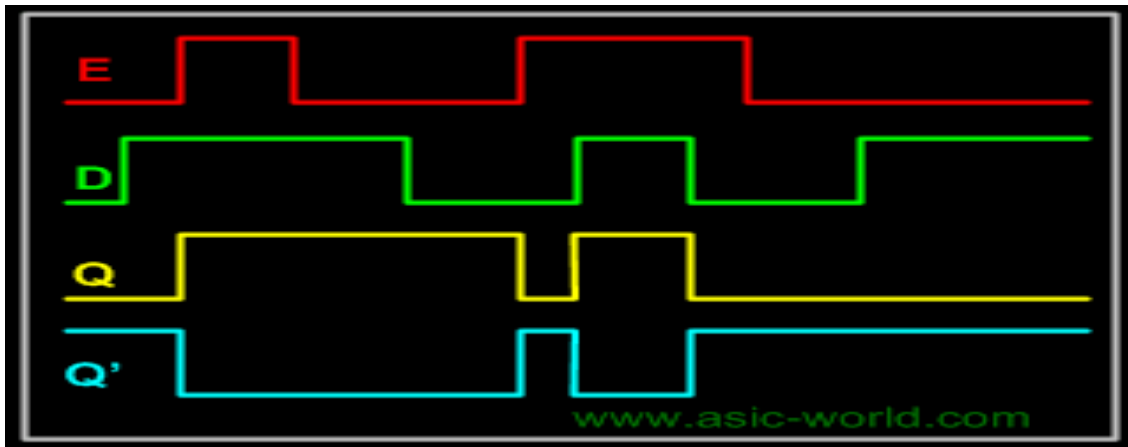
The RS latch seen earlier contains ambiguous state; to eliminate this condition we can ensure that S and R are never equal. This is done by connecting S and R together with an inverter. Thus we have D Latch: the same as the RS latch, with the only difference that there is only one input, instead of two (R and S). This input is called D or Data input. D latch is called D transparent latch for the reasons explained earlier. Delay flip-flop or delay latch is another name used. Below is the truth table and circuit of D latch.

In real world designs (ASIC/FPGA Designs) only D latches/Flip-Flops are used.



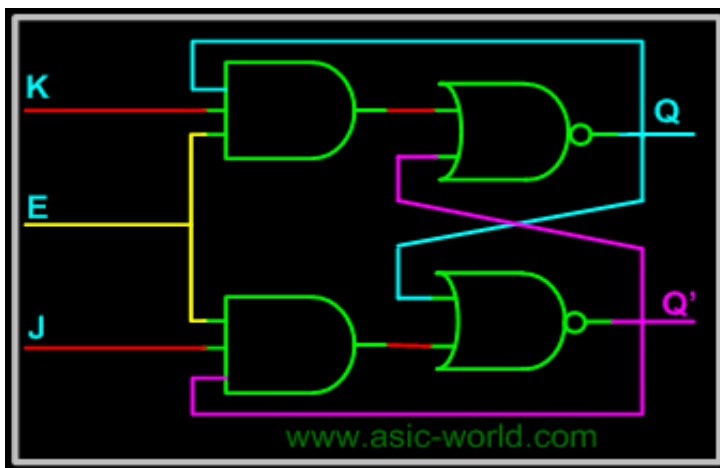
D	Q	Q+
1	X	1
0	X	0

Below is the D latch waveform, which is similar to the RS latch one, but with R removed.



### JK Latch

The ambiguous state output in the RS latch was eliminated in the D latch by joining the inputs with an inverter. But the D latch has a single input. JK latch is similar to RS latch in that it has 2 inputs J and K as shown figure below. The ambiguous state has been eliminated here: when both inputs are high, output toggles. The only difference we see here is output feedback to inputs, which is not there in the RS latch.

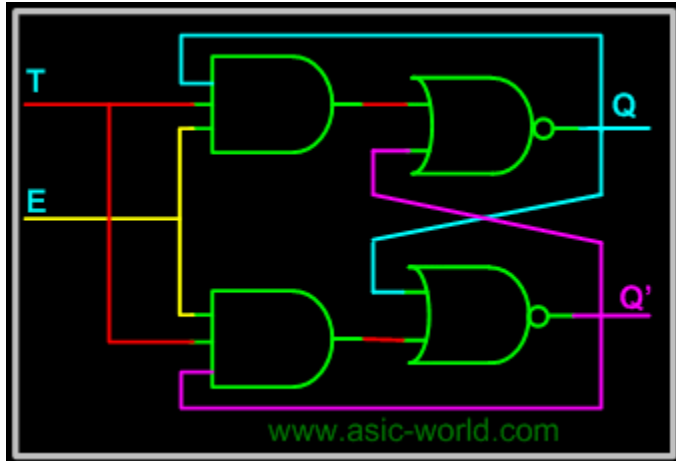


J	K	Q
1	1	0
1	1	1
1	0	1
0	1	0



## ❖ T Latch

When the two inputs of JK latch are shorted, a T Latch is formed. It is called T latch as, when input is held HIGH, output toggles.

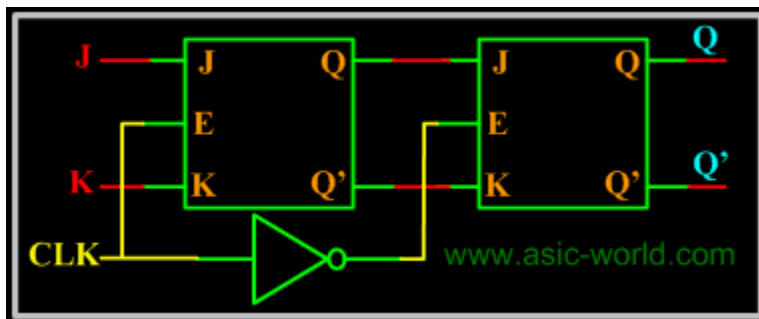


T	Q	Q+
1	0	1
1	1	0
0	1	1
0	0	0

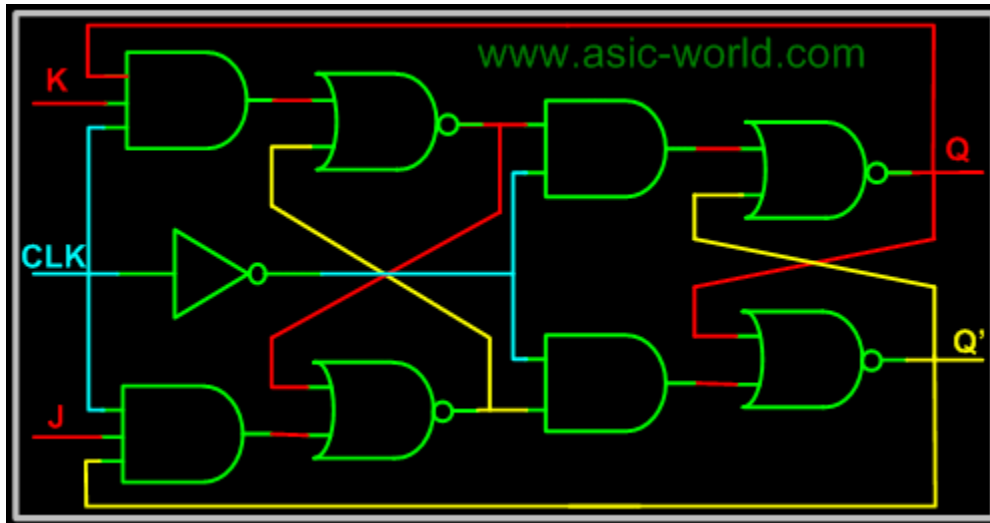
## ❖ JK Master Slave Flip-Flop

All sequential circuits that we have seen in the last few pages have a problem (All level sensitive sequential circuits have this problem). Before the enable input changes state from HIGH to LOW (assuming HIGH is ON and LOW is OFF state), if inputs changes, then another state transition occurs for the same enable pulse. This sort of multiple transition problem is called racing.

If we make the sequential element sensitive to edges, instead of levels, we can overcome this problem, as input is evaluated only during enable/clock edges.



In the figure above there are two latches, the first latch on the left is called master latch and the one on the right is called slave latch. Master latch is positively clocked and slave latch is negatively clocked.



### Sequential Circuits Design

We saw in the combinational circuits section how to design a combinational circuit from the given problem. We convert the problem into a truth table, then draw K-map for the truth table, and then finally draw the gate level circuit for the problem. Similarly we have a flow for the sequential circuit design. The steps are given below.

- Draw state diagram.
- Draw the state table (excitation table) for each output.
- Draw the K-map for each output.
- Draw the circuit.

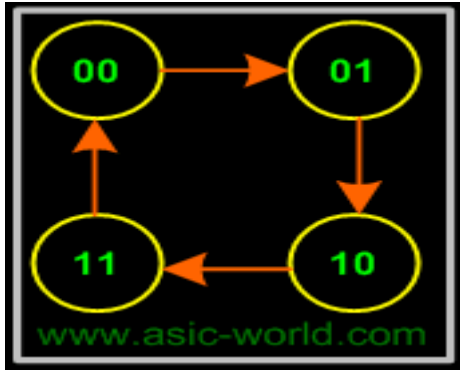
Looks like sequential circuit design flow is very much the same as for combinational circuit.

### State Diagram

The state diagram is constructed using all the states of the sequential circuit in question. It builds up the relationship between various states and also shows how inputs affect the states.

To ease the following of the tutorial, let's consider designing the 2 bit up counter (Binary counter is one which counts a binary sequence) using the T flip-flop.

Below is the state diagram of the 2-bit binary counter.



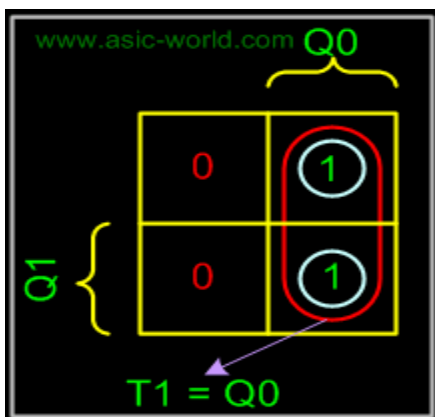
### ❖ State Table

The state table is the same as the excitation table of a flip-flop, i.e. what inputs need to be applied to get the required output. In other words this table gives the inputs required to produce the specific outputs.

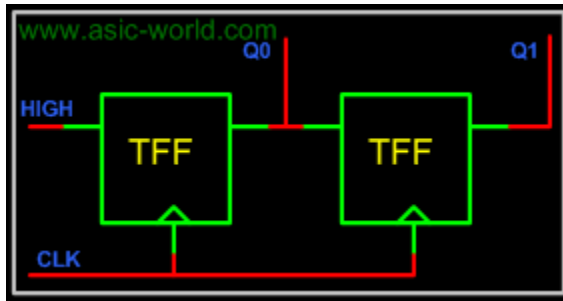
Q1	Q0	Q1+	Q0+	T1	T0
0	0	0	1	0	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	0	0	1	1

### ❖ K-map

The K-map is the same as the combinational circuits K-map. Only difference: we draw K-map for the inputs i.e. T1 and T0 in the above table. From the table we deduct that we don't need to draw K-map for T0, as it is high for all the state combinations. But for T1 we need to draw the K-map as shown below, using SOP.

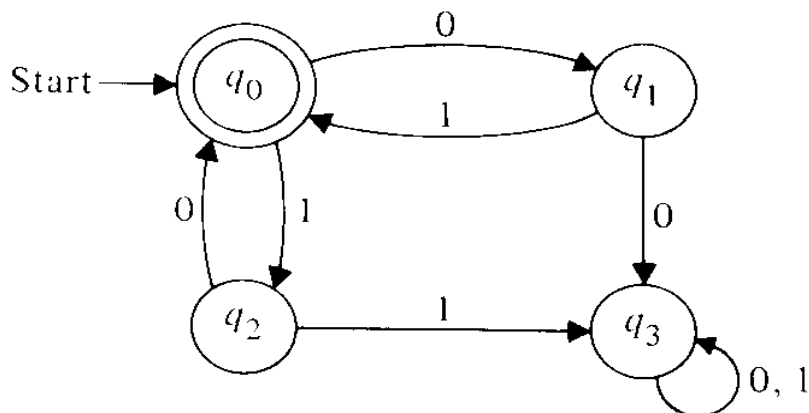


There is nothing special in drawing the circuit, it is the same as any circuit drawing from K-map output. Below is the circuit of 2-bit up counter using the T flip-flop.



### Finite State Machine

A model of computation consisting of a set of states, a start state, an input alphabet, and a transition function that maps input symbols and current states to a next state. Computation begins in the start state with an input string. It changes to new states depending on the transition function. There are many variants, for instance, machines having actions (outputs) associated with transitions (Mealy machine) or states (Moore machine), multiple start states, transitions conditioned on no input symbol (a null) or more than one transition for a given symbol and state (nondeterministic finite state machine), one or more states designated as accepting states (recognizer), etc.



**Definition 1** A finite state machine is a 5-tuple,  $(S, A, R, \_, s_0)$  where  $S$  is a finite set of states,  $A$  is a finite alphabet,  $R$  is a finite alphabet of responses and  $\_$  is a transition function such that for any state,  $s \in S$  and symbol  $a \in A$ ,  $\_(s, a) = (s_0, r_0)$  indicates the next state,  $s_0$  and the output symbol,  $r_0 \in R$ .  $s_0$  is the initial state.

**Definition 2** A recogniser is a special kind of finite state machine in which the output alphabet contains two special symbols: accept and reject. The machine responds to any finite sequence of input symbols, terminated with a special end of input symbol ( $\_$ ), with either accept or reject.

## Limitations of Finite State Machines:

- Number of states in the composed FSM grows dramatically (state explosion problem)
- Composing FSMs of  $n$  subsystems, with  $k_1, k_2, k_3, \dots, k_n$ , states respectively, results in a system whose FSM has  $k_1 \times k_2 \times \dots \times k_n$  states - This growth is exponential with the number of subsystems, not linear (i.e.,  $k_1 + k_2 + \dots + k_n$ ).
- Since at any time, a global state of the system must be defined and a single transition must occur, FSM model is not suitable for describing asynchronous concurrent activities in the system.

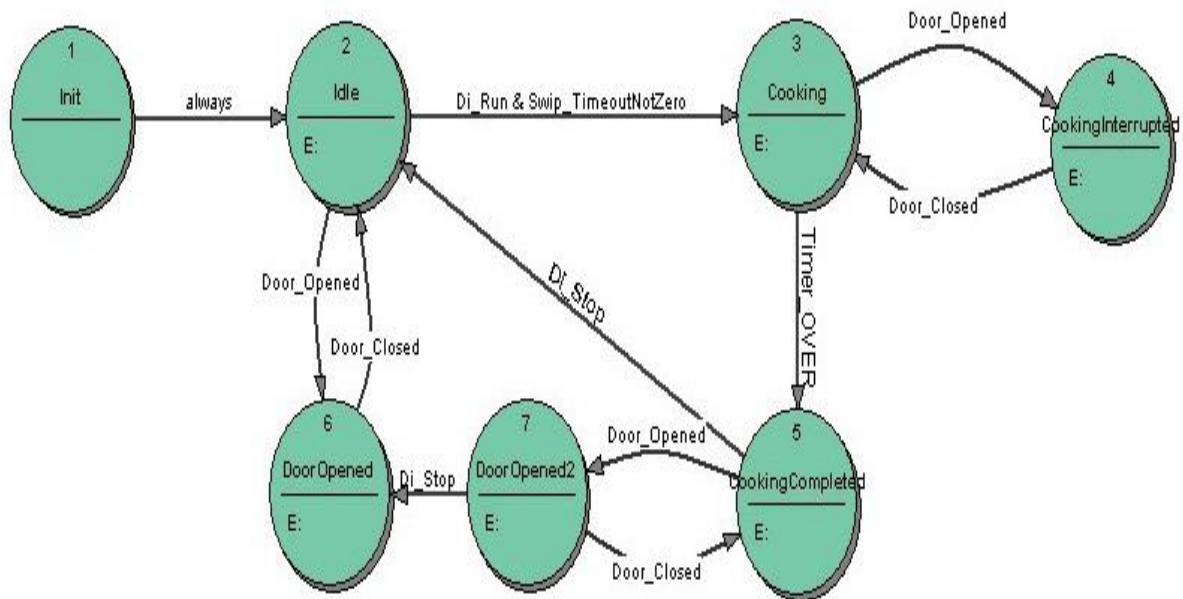
## Mealy And Moore Models

Mealy and Moore models are the basic models of state machines. A state machine which uses only Entry Actions, so that its output depends on the state, is called a Moore model. A state machine which uses only Input Actions, so that the output depends on the state and also on inputs, is called a Mealy model.

The models selected will influence a design but there are no general indications as to which model is better. Choice of a model depends on the application, execution means (for instance, hardware systems are usually best realised as Moore models) and personal preferences of a designer or programmer. In practise, mixed models are often used with several action types. On an example we will show the consequences of using a specific model. As the example we have taken a Microwave Oven control presented already in another paper on our web site. This time we will make a very thorough analysis of the behaviour exposing the specifics of the model used. The oven has a momentary-action push button Run to start (apply the power) and a Timer that determines the cooking length. Cooking can be interrupted at any time by opening the oven door. After closing the door the cooking is continued. Cooking is terminated when the Timer elapses. When the door is open a lamp inside the oven is switched on, when the door is closed the lamp is off. During cooking the lamp is also switched on. The cooking period (timeout value) is set by a potentiometer which supplies a voltage to the control system: the voltage is represented by a numeric value 0.4095 which is scaled by the Ni object to 1799. This arrangement allows the maximum cooking time to equal 1799 seconds, i.e. 30 minutes. The solution should also take into account the possibility that the push button Run could get blocked continuously in the active position (which is easy to demonstrate if testing the system in SWLab, which has only the two-positions buttons): in such a case cooking must not start again until it is deactivated when the cooking is terminated (otherwise our meal which we wanted to heat for instance for 5 minutes could be burned until we discover that the button has got stuck in the active position). In other words, each cooking requires intentional activation of the Run button. The control system has the following inputs: Run momentary-action push button - when activated starts cooking, Timer - while this runs keep on cooking, Door sensor - can be true (door closed) or false (door open). And the following outputs: Power - can be true (power on) or false (power off), Lamp - can be true (lamp on) or false (lamp off).

Moore model :

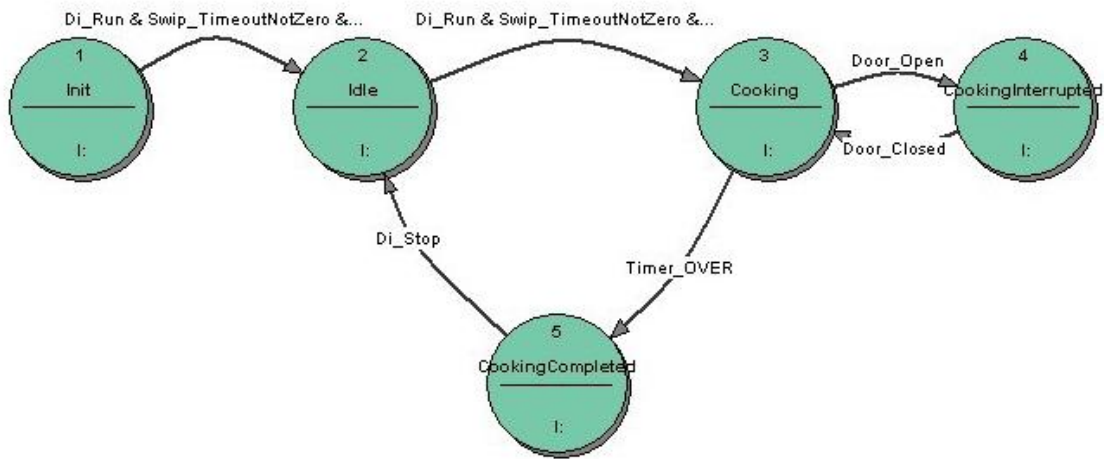
Using Moore model we get a state machine whose state transition diagram is shown in Figure 1. This solution requires 7 states. Figure 2, Figure 3 and Figure 4 show state transition tables for three of those states: Init, Cooking and CookingInterrupted. The state machine uses only Entry actions. Other states can be studied in the provided file MWaveOven\_Moore.fsm. While specifying that state machine the states dominate. We think in the following manner: if the input condition changes the state machine changes its state (if a specific transition condition is valid). Entering the new state, the state machine does some actions and waits for the reaction of the controlled system. In a Moore model the entry actions define effectively the state. For instance, we would think about the state Cooking: it is a state where the Timer runs and the state machines waits for the Timer OVER signal.



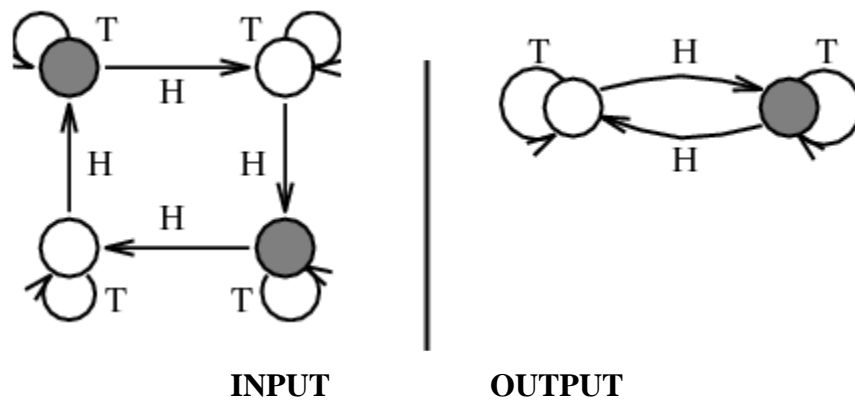
### ***Mealy model***

The Mealy model is shown in Figure 5. It requires only 5 states. The states: Idle, Cooking and Cooking Interrupted for that model (see Figure 6, Figure 7 and Figure 8) illustrate its features. Other states can be studied in the provided file MWaveOven\_Mealy.fsm. All activities are done as Input actions, which means that actions essential for a state must be performed in all states which have a transition to that state. The Timer must be now started in both states: Idle and Cooking Interrupted.

This may be considered as a disadvantage: the functioning becomes a bit confusing.



### Minimization of FSM:



### Binary multiplier:

A **binary multiplier** is a electronic circuit used in digital electronics, such as a computer, to multiply two binary numbers. It is built using binary adders.

#### *Theory*

Using long multiplication, a product of two N-bit numbers can be expressed as the sum of N N-bit partial products, which are then added to produce a 2N-bit product.

$$\begin{array}{r}
 \begin{array}{cccc}
 & a_3 & a_2 & a_1 & a_0 \\
 \times & b_3 & b_2 & b_1 & b_0 \\
 \hline
 & a_3b_0 & a_2b_0 & a_1b_0 & a_0b_0 \\
 & a_3b_1 & a_2b_1 & a_1b_1 & a_0b_1 \\
 & a_3b_2 & a_2b_2 & a_1b_2 & a_0b_2 \\
 & a_3b_3 & a_2b_3 & a_1b_3 & a_0b_3 \\
 \hline
 p_7 & p_6 & p_5 & p_4 & p_3 & p_2 & p_1 & p_0
 \end{array}
 \end{array}$$

The partial products can be trivially computed from the fact that  $a_i \times b_j = a_i \text{ AND } b_j$ . The complexity of the multiplier is in adding the partial products.

### Implementation

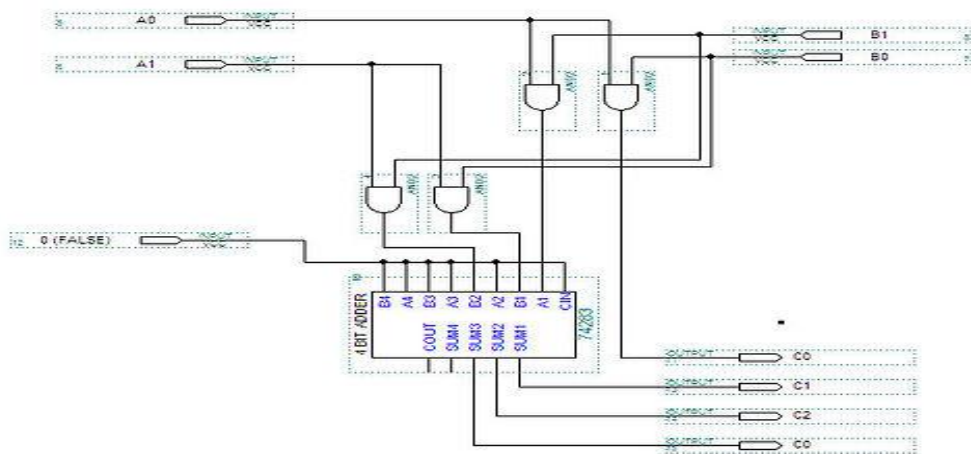
There are several ways to implement a binary multiplier.

### Multiple adders

Partial products are added in pairs using binary adders until the entire product is computed similar to multiplying large numbers by hand. This requires  $N-1$  adders.

Typically those adders are arranged as an adder compressor tree.

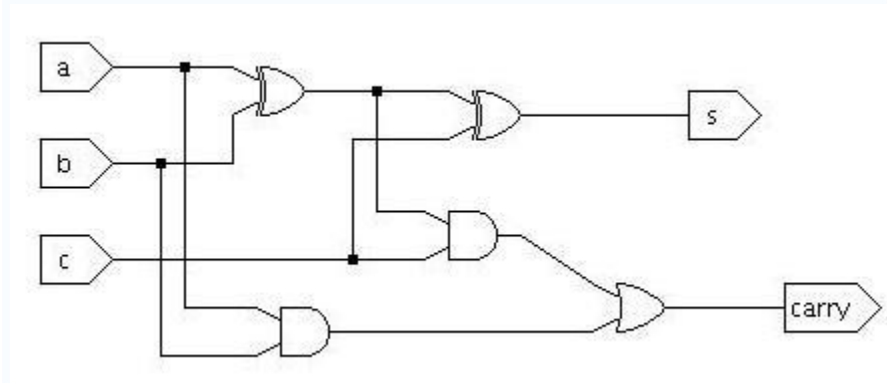
### Example:



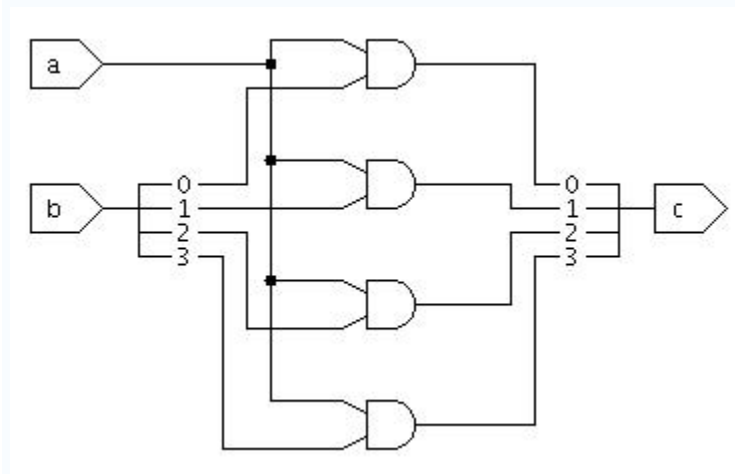


## 2 Bit by 2 Bit Binary Multiplier Using a 4 Bit + 4 Bit Adder

(The following are to clarify each level of abstraction)



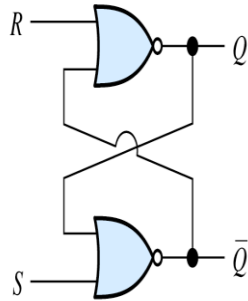
## A simple adder 1-bit adder



## 4-bit Adder Using 4 1-bit adders

Summary of flip flops:

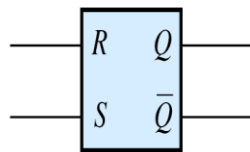
### SR Flip Flop



**Figure 7.35** An SR flip-flop can be implemented by cross coupling two NOR gates.

$R$	$S$	$Q_n$
0	0	$Q_{n-1}$
0	1	1
1	0	0
1	1	Not allowed

(a) Truth table

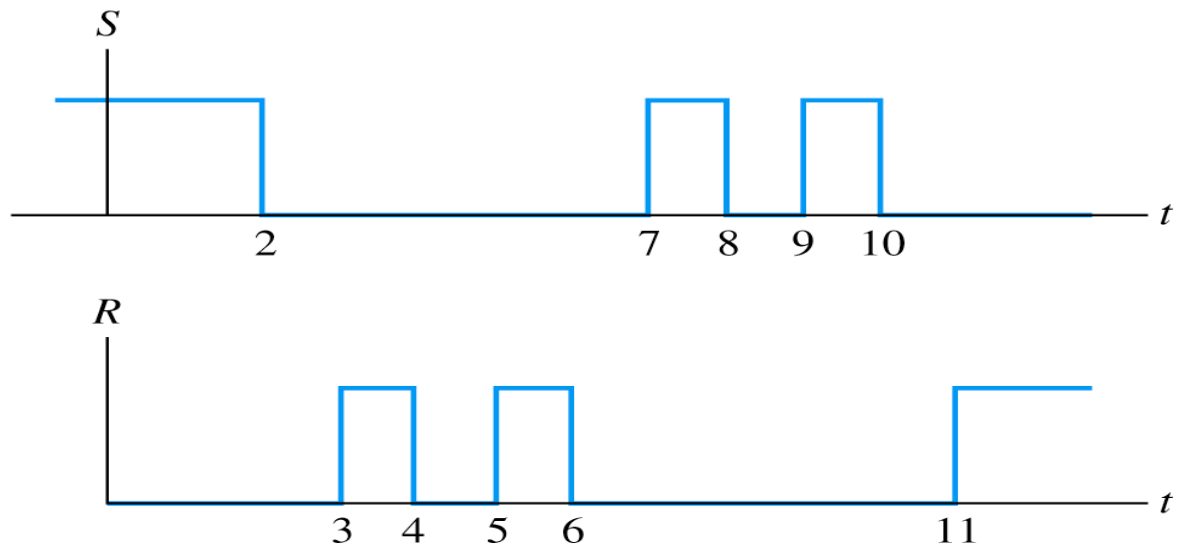


(b) Circuit symbol

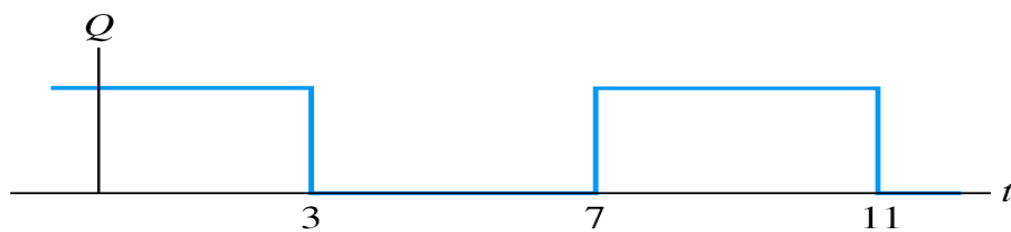
**Figure 7.36** The truth table and symbol for the SR flip-flop.

### Exercise

For a given S and R inputs to SR flip-flop, sketch the output signal Q



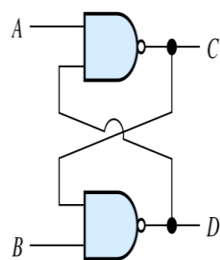
**Figure 7.38** See Exercise 7.18.



**Figure 7.39** Answer for Exercise 7.18.

### SR Flip Flop

SR (set-reset) flip-flop based on two nand gates

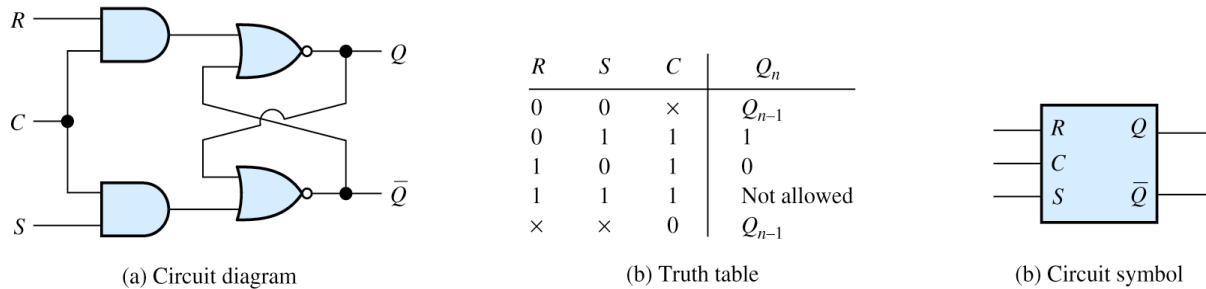


**Figure 7.40** A flip-flop implemented with NAND gates. See Exercise 7.19.

### Clocked SR Flip Flop Circuit

Clock controlled flip-flop changes its state

only when the clock  $C$  is high



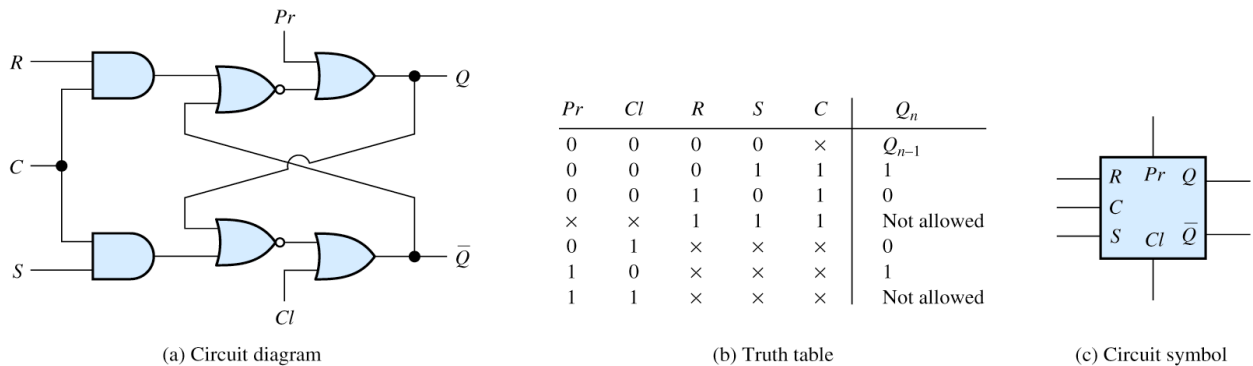
**Figure 7.41** A clocked  $SR$  flip-flop.

### Clocked SR Flip Flop Circuit with Reset

Some flip-flops have asynchronous preset  $Pr$  and clear  $Cl$  signals.

Output changes once these signals change, however the input signals must wait for a change in clock

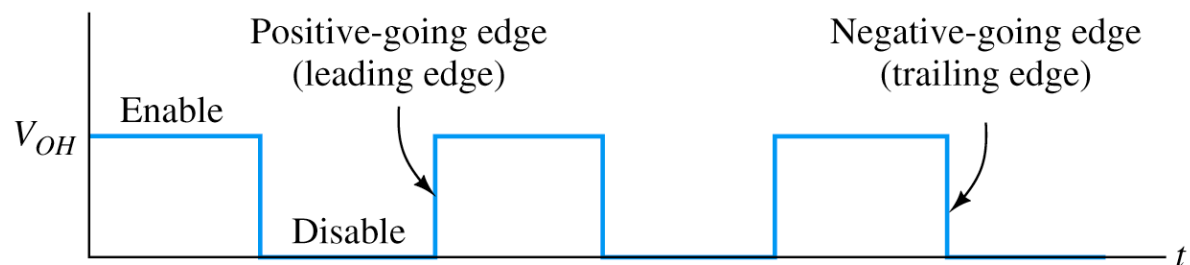
to change the output.



**Figure 7.42** A clocked  $SR$  flip-flop with asynchronous preset and clear inputs.

### Edge Triggered Flip Flop

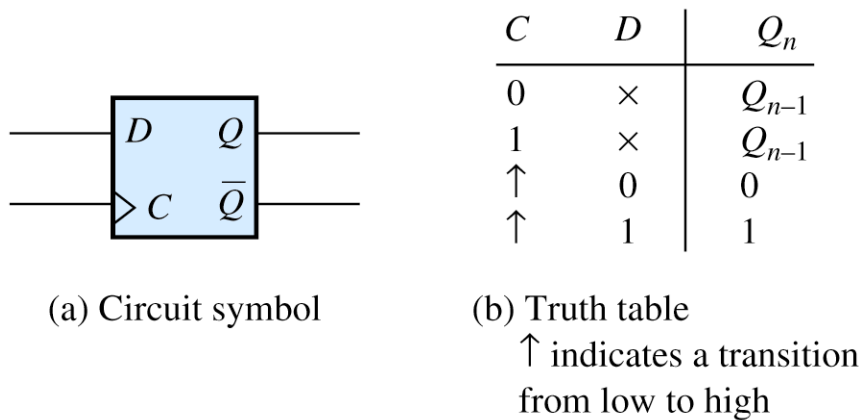
Edge triggered flip-flop changes only when the clock  $C$  changes



**Figure 7.43** Clock signal.

**Positive Edge Triggered Flip Flop**

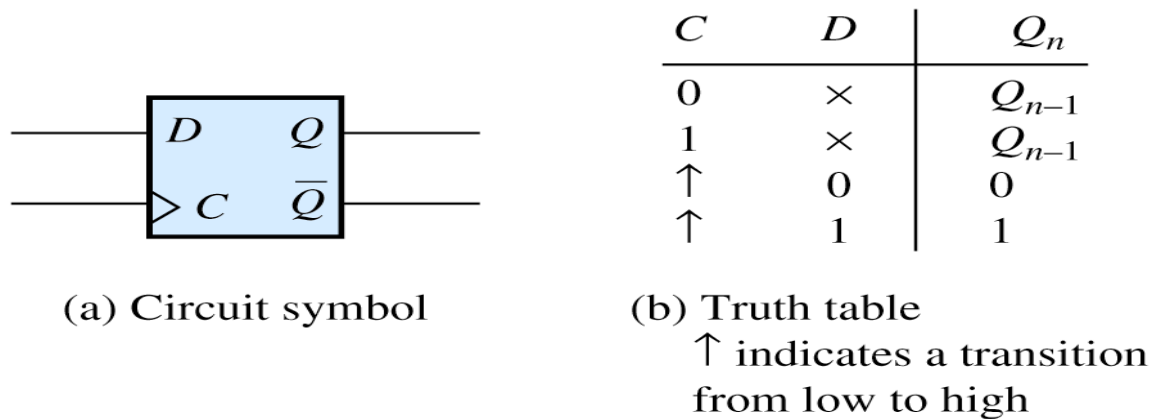
Positive-edge triggered flip-flop changes only on the rising edge of the clock C



**Figure 7.44** A positive-edge-triggered  $D$  flip-flop.

**Exercise**

The input  $D$  to a positive-edge triggered flip-flop is shown Find the output signal  $Q$



**Figure 7.44** A positive-edge-triggered  $D$  flip-flop.

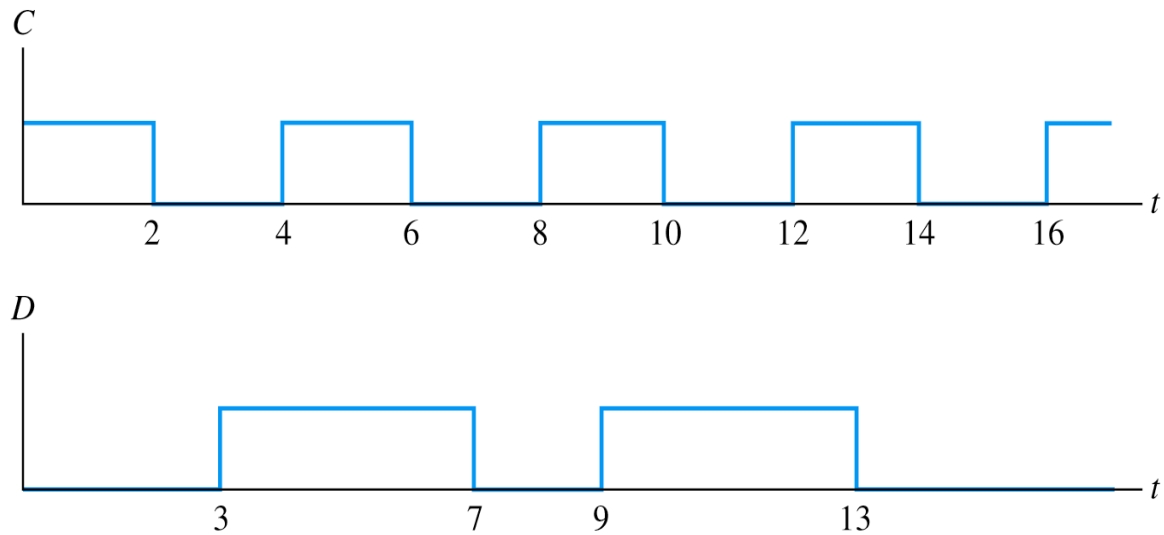


Figure 7.45 See Exercise 7.20.

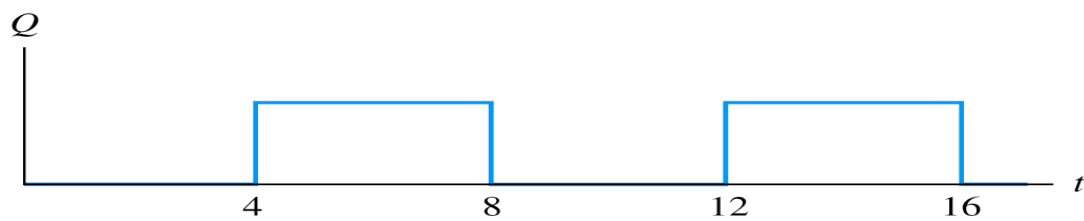
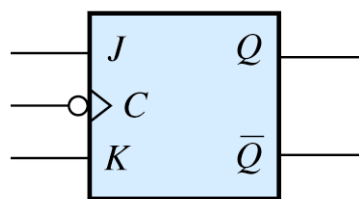


Figure 7.46 Answer for Exercise 7.20.

### Negative Edge Triggered JK Flip Flop

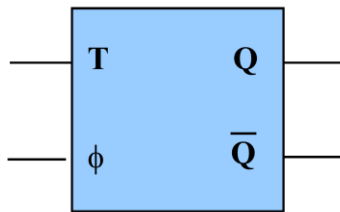
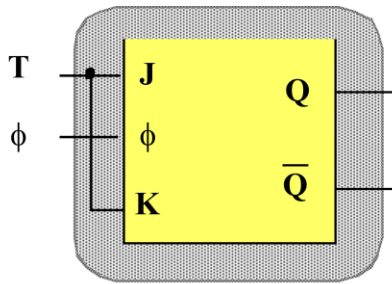
	$C$	$J$	$K$	$Q_n$	Comment
	0	$\times$	$\times$	$Q_{n-1}$	Memory
	1	$\times$	$\times$	$Q_{n-1}$	Memory
	$\downarrow$	0	0	$Q_{n-1}$	Memory
	$\downarrow$	0	1	0	Reset
	$\downarrow$	1	0	1	Set
	$\downarrow$	1	1	$\overline{Q}_{n-1}$	Toggle

(a) Circuit symbol

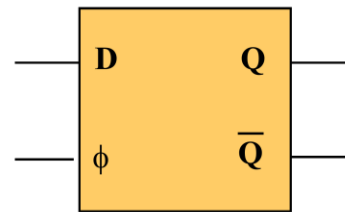
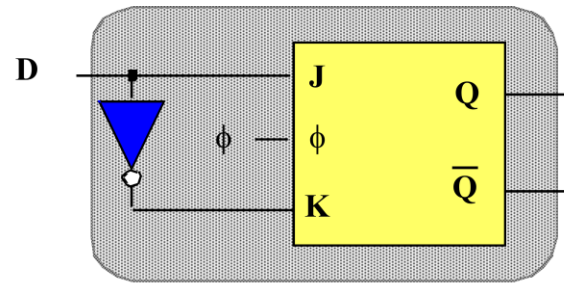
(b) Truth table  
 $\downarrow$  indicates a transition  
 from low to high

Figure 7.47 Negative-edge-triggered  $JK$  flip-flop.

## Other Flip Flops

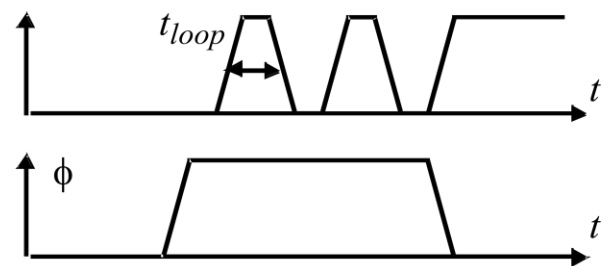
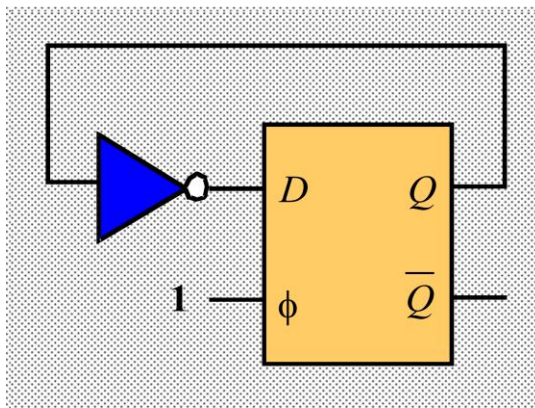


## Toggle Flip-Flop



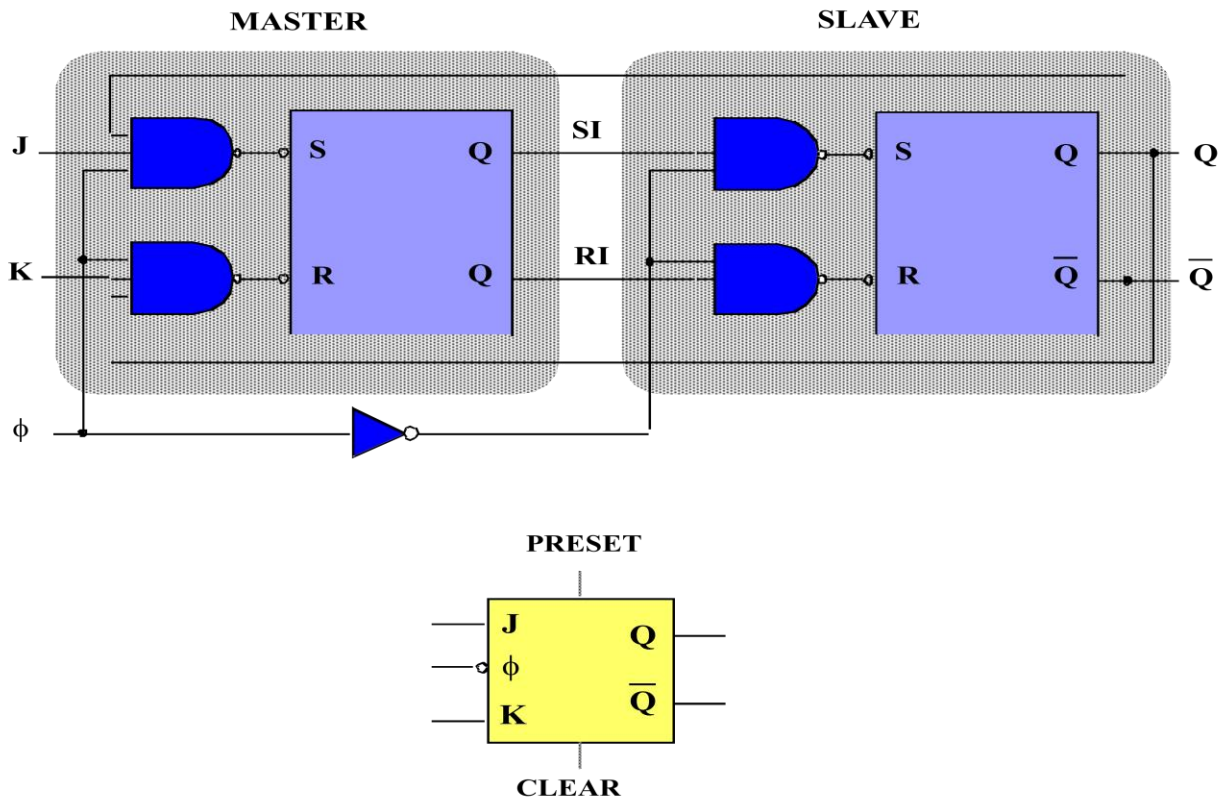
### Delay Flip-Flop (D-latch)

## Race Problem



**Signal can race around during  $\phi = 1$**

## Master-Slave Flip Flop Implementation



Master transmits the signal to the output during the high clock phase and slave is waiting for the clock to change this prevents race conditions.