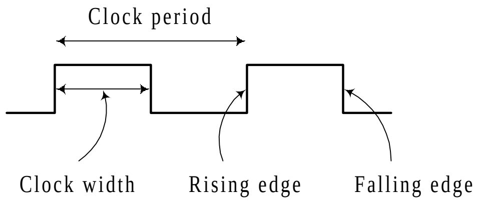
6. Sequential Logic – Flip-Flops

Combinatorial components: their output values are computed entirely from their present input values.
Sequential components: their output values are computed using both the present *and* past input values.
In other words, their outputs depend on the sequence of input values that have occurred over a period of time.
This dependence on the past input values requires the presence of memory elements.
The values stored in memory elements define the state of a sequential component.
Since memory is finite, therefore, the sequence size must always be finite, which means that the sequential logic can contain only a finite number of states.
So sequential circuits are sometimes called **finite-state machines**.
Sequential circuits can be a asynchronous or synchronous.
Asynchronous sequential circuits change their state and output values whenever a change in input values occurs.
Synchronous sequential circuits change their states and output values at fixed points of time, which are specified by the rising or falling edge of a free-running clock signal.
Clock period is the time between successive transitions in the same direction, i.e., between two rising or two falling edges.
Clock frequency = 1/clock period
Clock width is the time during which the value of the clock signal is equal to 1.
Duty cycle is the ratio of clock width and clock period.
Active high if the state changes occur at the clock's rising edge or during the clock width.



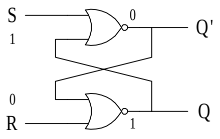
Active low if the state changes occur at the clock's falling edge.

Latches and **flip flops** are the basic storage elements that can store one bit of information.

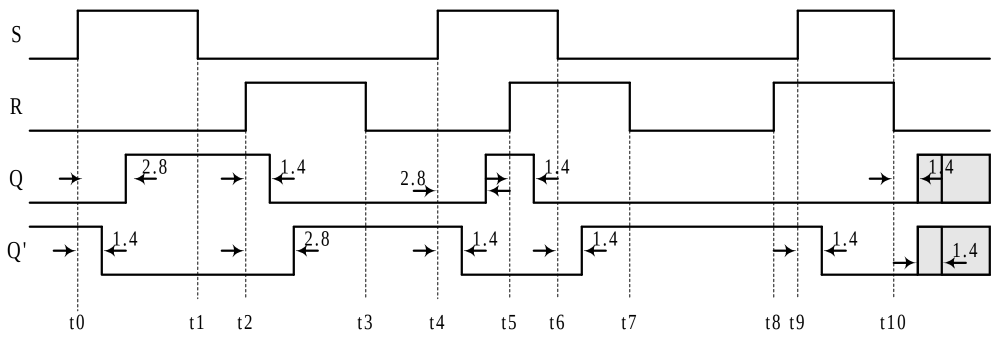
nor	0	1
0	1	0
1	0	0

6.1 SR Latch

The simplest memory element.
Consists of two cross-coupled NOR gates.
Inputs S (set) and R (reset) are normally 0.
Both active high.
Asserting S (setting S=1) will make output Q=1.
Asserting R (setting R=1) will make Q=0.

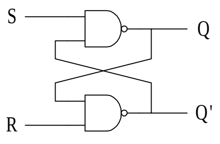


S	R	Q	Q_{next}	Q'_{next}
0	0	0	0	1
0	0	1	1	0
0	1	x	0	1
1	0	x	1	0
1	1	x	0	0



One problem inherent in the SR latch is the fact that if both S and R are disasserted at the same time, we cannot predict the latch output (as in t10).

The SR latch can also be implemented with NAND gates.
S and R are normally 1. They are active low.

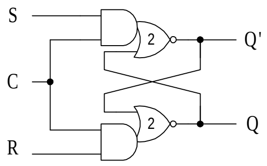


nand	0	1
0	1	1
1	1	0

S	R	Q	Q_{next}	Q'_{next}
1	1	1	1	0
1	1	0	0	1
1	0	x	0	1
0	1	x	1	0
0	0	x	1	1

6.2 SR Latch with Enable

Similar to the SR latch but with the extra control input C which enables or disables the operation of the S and R inputs.
When C=1, the gated SR latch operates as an SR latch.
When C=0, S and R are disabled and the circuit persists in the preceding state.

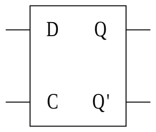


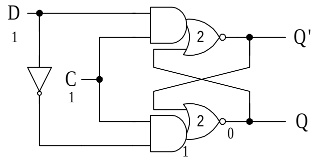
C	S	R	Q	Q_{next}
0	x	x	0	0
0	x	x	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	x	0
1	1	0	x	1
1	1	1	x	NA

6.3 Gated D Latch

D latch ensures that inputs S and R never equal to 1 at the same time.
Also SR latches are useful in control applications where we often think in terms of setting or resetting a flag to some condition.
However, we often need latches to store bits of information and a D latch may be used in such an application.
Gated D latch is constructed from a gated SR latch with an inverter added between the S and the R inputs and use a single D (data) input.

nor	0	1
0	1	0
1	0	0



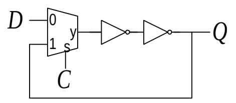


C	D	Q	Q_{next}
0	x	0	0
0	x	1	1
1	0	x	0
1	1	x	1

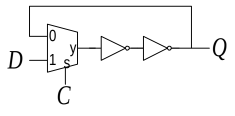
The C (control) input is active high in this design but can also be active low.
When the C input is asserted, the Q output follows the D input. In this situation, the latch is said to be "**open**" and the path from D input to Q output is "transparent"; the circuit is often called a **transparent latch** for this reason.
When the C input is negated, the latch "**closes**"; the Q output retains its last value and no longer changes in response to D.

Latches are often called **level-sensitive latches** because they are enabled and transparent whenever C is asserted.

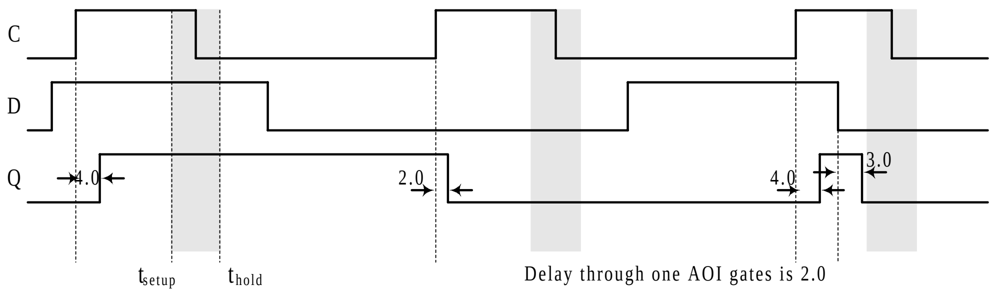
Method 2: Gated D latch can also be implemented using a multiplexer.



negative latch
D passes to Q when C=0



positive latch
D passes to Q when C=1



Problem with the D latch: there is a (shaded) window of time around the *falling edge of C* when the D input must not change. This window begins at time t_{setup} before the falling (latching) edge of C; t_{setup} is called the **setup time**. The window ends at time t_{hold} afterward; t_{hold} is called the **hold time**.

6.4 D Flip-Flop

A **positive-edge-triggered D flip-flop** combines a pair of D latches[1].

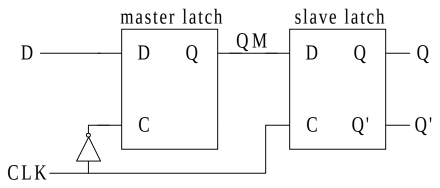
It samples its D input and changes its Q and Q' outputs only at the rising edge of a controlling CLK signal.

When CLK=0, the first latch, called the **master**, is enabled (open) and the content of D is transferred to QM.

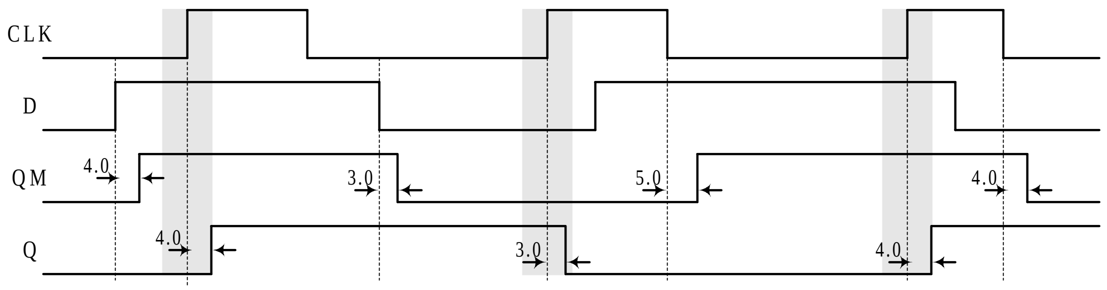
When CLK=1, the master latch is disabled (closed) and its output is transferred to the second latch, called the **slave**.

The slave latch is open all the while that CLK=1, but changes only at the beginning of this interval, because the master is closed and unchanging during the rest of the interval.

Advantage: since the master and slave latches are never enabled at the same time, the entire master-slave flip-flop is never transparent.



D	CLK	Q	Q'
0	↑	0	1
1	↑	1	0
x	0	last Q	last Q'
x	1	last Q	last Q'



Like a D latch, the edge-triggered D ff has a setup and hold time window during which the D inputs must not change. This window occurs around the triggering edge of CLK (rising clock edge for a positive-edge-triggered ff and falling clock edge for a negative-edge-triggered ff).

If the setup and hold times are not met, the ff output will usually go to a stable, though unpredictable, 0 or 1 state. In some cases, however, the output will oscillate or go to a metastable state halfway between 0 and 1.

All propagation delays are measured from the triggering edge of CLK, since that's the only event that causes an output change.

A **negative-edge-triggered D flip-flop** simply inverts the clock input, so that all the action takes place on the falling edge of the clock.

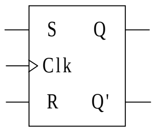
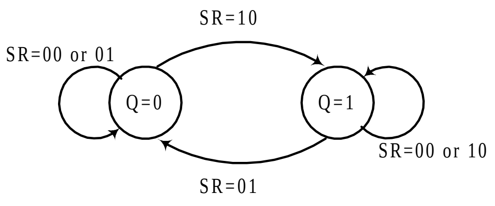
There are many different ways to construct flip-flops, but they all exhibit the following two characteristics:

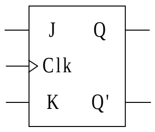
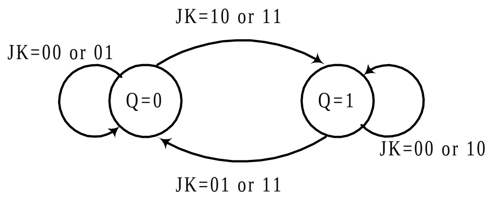
- a ff will change state only on the positive or negative edge of the clock signal.
- its data inputs must not change after time t_{setup} and before t_{hold}.

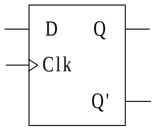
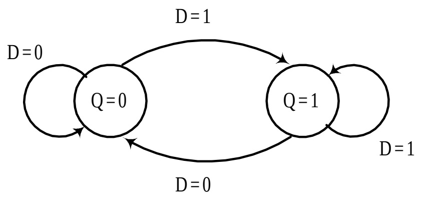
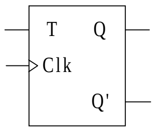
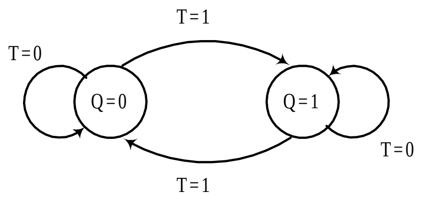
All ffs can be divided into four basic types: **SR**, **JK**, **D**, and **T**.

- The **SR** ff has two inputs, S (set) and R (reset), that set or reset the output Q when asserted.
- The **JK** ff has two inputs, J and K just like the S and R. However, when both J and K are asserted at the same time, the JK ff changes its state.
- The **D** ff has one input D (data) which sets the ff when D=1 and resets it when D=0.
- The **T** ff has one input T (toggle) which forces the flip-flop to change states when T=1.

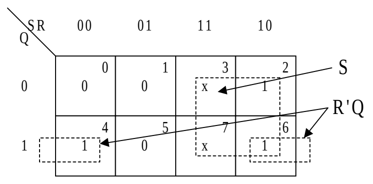
[1] (Gajski differentiates between master-slave ff and edge-triggered ff. Wakerly and many other books say the edge-triggered ff is the master-slave ff.)

Name	**FF Symbol**	**Characteristic (Truth) Table**	**State Diagram / Characteristic Equations**	**Excitation Table**
SR	 The triangle indicates that the ff is triggered by the rising edge.	*S R Q_{next}* 0 0 Q 0 1 0 1 0 1 1 1 NA	 $Q_{next} = S + R'Q$ $SR = 0$ [*]	*Q Q_{next} S R* 0 0 0 X 0 1 1 0 1 0 0 1 1 1 X 0

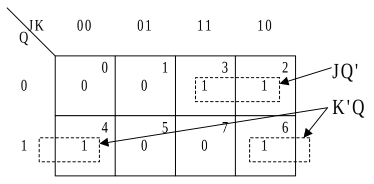
JK		*J K Q_{next}*
		0 0 Q
		0 1 0
		1 0 1
		1 1 Q’
			 JK=10 or 11
			JK=00 or 01
			Q=0
			Q=1
			JK=00 or 10
			JK=01 or 11
			$Q_{next} = J'K'Q + JK' + JKQ'$
			$= J'K'Q + JK'Q + JK'Q' + JKQ'$
			$= K'Q(J'+J) + JQ'(K'+K)$
			$= K'Q + JQ'$ [*]
				Q Q_{next} J K
				0 0 0 X
				0 1 1 X
				1 0 X 1
				1 1 X 0

D		*D Q_{next}* 0 0 1 1	 $Q_{next} = D$	*Q Q_{next} D* 0 0 0 0 1 1 1 0 0 1 1 1
T		*T Q_{next}* 0 Q 1 Q’	 $Q_{next} = TQ' + T'Q = T \oplus Q$	*Q Q_{next} T* 0 0 0 0 1 1 1 0 1 1 1 0

[*] SR



[*] JK

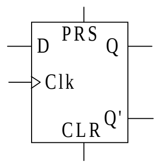


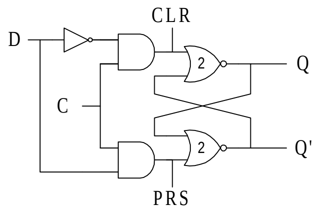
The **characteristic table** is a shorter version of the truth table, that gives for every set of input values and the state of the flip-flop before the rising edge, the corresponding state of the flip-flop after the rising edge of the clock. It is used during the analysis of sequential circuits.

The c**haracteristic equation** is just the functional expressions derived from the characteristic (truth) table. It formally describes the functional behavior of a latch or flip-flop. They specify the flip-flop's next state as a function of its current state and inputs.

The **excitation table** gives the value of the flip-flop inputs that are necessary to change the flip-flop's present state to the desired next state after the rising edge of the clock signal. It is obtained from the characteristic table by transposing input and output columns. It is used during the synthesis of sequential circuits.

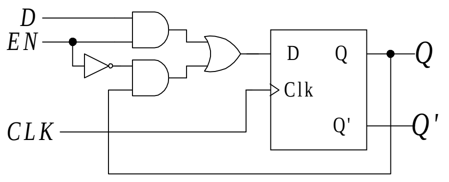
Some flip-flops have **asynchronous inputs** that may be used to force the flip-flop to a particular state independent of the CLK and D inputs. These inputs typically labeled **PRS** (preset) and **CLR** (clear), behave like the set and reset inputs on an SR latch.

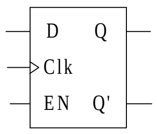




nor	0	1
0	1	0
1	0	0

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) through a multiplexer.





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