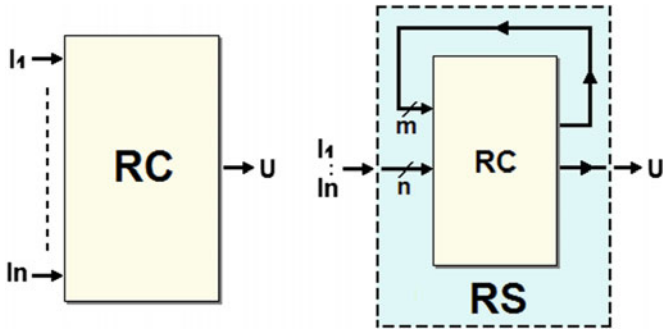
Chapter 5
Introduction to Sequential Networks

Abstract The transition from combinational to sequential networks is explained step by step, starting from a simple gate with feedback and arriving to the structure and behavior of the principal types of flip-flops. They are classified according to their temporal response (direct command, level enabled, master–slave, and edge triggered) and the logical operation (SR, D, JK). The timing parameters of physically implemented devices are considered. The chapter introduces the concept and techniques for synchronization that will be further examined in the following ones.

It is very rare for a digital device to be based only on combinational networks. In a real situation, it is important to have devices that can memorize data, generate sequences, and respond to conditions that change over time.

5.1 From Combinational Networks to Sequential Networks

We have seen combinational networks (see RC in the figure below left) where, at any given moment, the output U is the function of only the I inputs. Combinational networks are identified precisely by the fact that every input combination *always* produces the same output $U = f(I1, I2, \ldots In)$.



G. Donzellini et al., *Introduction to Digital Systems Design*,
https://doi.org/10.1007/978-3-319-92804-3_5

A *sequential network* (RS, below right) does not follow this rule. The same input combination can generate different outputs when applied at different moments.

We can obtain a sequential network by starting with a combinational network and bringing one or more of its outputs into the inputs, as seen in the figure above. This type of connection is called *feedback*.

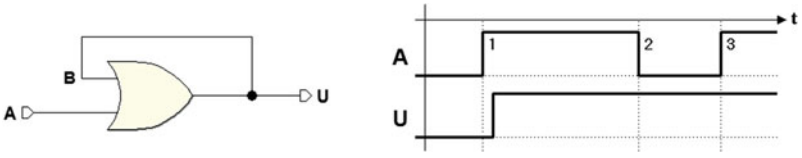
Generally speaking, connecting m outputs of a combinational network to as many inputs makes it so that the outputs' behavior as a function of the inputs depends *in part on the outputs themselves*. If we consider recursively that each set of current outputs was produced by the inputs plus the preceding outputs, we can say that the outputs depend not only on the *current* inputs but on their *history*.

In other words, the functioning of the network depends on the *sequence of inputs* that produce a *sequence of outputs*. A logical network that behaves this way is unsurprisingly called a *sequential network*.

Next we will see that in sequential networks, the outputs, the inputs, and the special conditions of the network at that moment (called *"state"*) can all be expressed analytically. As we will see, the concept of *state* will allow us to concisely express the *history* of the network.

5.1.1 Introductory Example

Let's look at this simple example of a sequential network that uses the OR function, which as we know is a combinational network. Let's construct a *feedback* by bringing the output U to one of the two inputs (B), as in the following figure.



Let's try to understand how this network acts, operating on the only available input A, since B is already driven by U, so it is unavailable. Let's suppose that A and U (and therefore B) are initially equal to 0. This is described in the timing diagram to the right.

If we force input A to 1 at a certain moment, the output will go to 1. If this were a combinational network this change would have exhausted the number of possible cases (two), taking the number of inputs (one) into account. If we reduce input A to zero, this should also force output U to zero, but actually, this does not happen. Because of the feedback connection between U and B, output U remains *forced* at 1 for any input value A so it is impossible to get it back to zero.

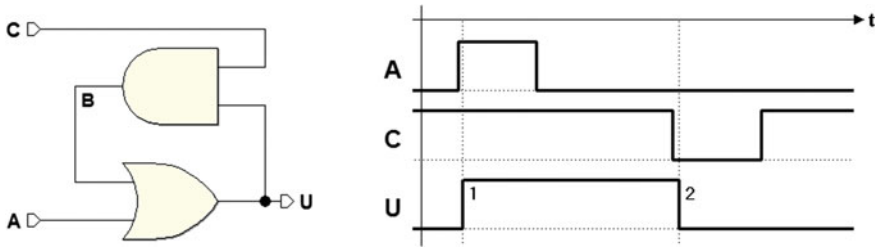
We can say that the network has *memorized* the value 1. Note that the diagram shows in an approximate way the propagation time of the OR gate.

Evidently, this is no longer a combinational network but a *sequential* one, where the output value depends on the *history* of the inputs. We can no longer describe how it works through a truth table like we did with combinational networks.

5.1.2 Memorizing an Information Bit: Flip-Flops

If we find a way to force the output to 0, the simple sequential network we have just seen can be used to store an *information bit*, i.e., to memorize both the value 0 and 1.

A potential change is shown in the figure below, where an AND gate was inserted in the feedback loop and the input C was added.



The AND gate and input C were added to establish or remove the connection between U and B. If $C = 1$ the network will behave as before since every variation of U is transferred through the AND gate on B ($B = U \cdot 1 = U$). If $C = 0$, then B is at 0, regardless of the value of U ($B = U \cdot 0 = 0$).

In the timing diagram in the figure, we assume that we start with input A and output U at 0 and with input C at 1. Then, the activation of the input A forces the output to 1. The new value $U = 1$, brought from the AND gate on input B of the OR, makes it so that further variations of A can no longer change the output. We have memorized a bit at 1.

To force the output U to 0, we will have to *open* the feedback loop by applying the value 0 to input C, as shown in the timing diagram. Forcing the output to zero memorizes a bit at 0. Further variations of C do not change the situation. As before, in the figure the propagation times are represented in a approximated way.

Note that each input can be considered in two ways:

- the purpose that each command has in the network;
- the logical modality that produces an effect.

In this case, input A produces the effect of memorizing 1 in the output, which happens when it is brought to 1. This is called an *active-high* input, that is, it does its job when it is *brought to* 1 and is *inactive* when it is at 0.

Input C has the job of memorizing a 0, which happens when it is *brought to* 0. We say that this input at 0 is *active-low* and when it is *inactive* it is at 1.

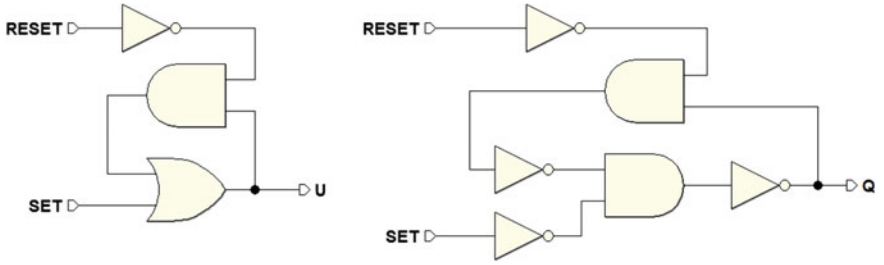
With this sequential network, we are able to *memorize a bit* whose value is maintained until a new value is memorized. This is one of the ways to create an *elementary memory cell*, also called *bistable element*, *one-bit register* or, more commonly, *"flip-flop."* Flip-flops are the basic logical elements generally used to build sequential digital systems.

Now let's try to slightly modify the network, making the commands symmetrical and easier to operate. We want to gradually build the network analyzed in the paragraph below, the classic elementary memory cell called *Set-Reset flip-flop*.

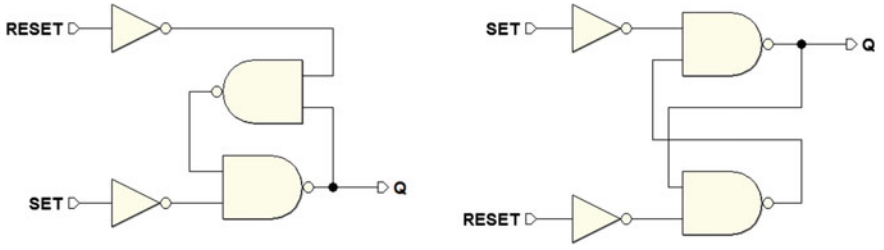
To achieve this, we make the following changes. Let's

- add a NOT before input C, to make *active-high* the command memorizing the zero.
- call the new input we obtain $RESET$ (because it brings the output to 0).
- change the name of input A to SET (because it brings the output to 1).
- change the name of output U to Q (to follow an established naming tradition).

The network with these changes appears in the figure below left. To continue transforming it, we apply De Morgan's theorem and substitute the OR with an AND (in the figure at the right).



The NOTs directly following the ANDs suggest that we should use the NANDs to get the network below left. The transformation is now complete; it is convenient to re-draw the network to highlight its symmetry without altering the connections (in the figure below right).



The network in the right is called *Set-Reset flip-flop*.

5.1.3 Flip-Flop Classification: Logical Type and Command Type

In the last section, we saw how to derive the sequential structure called the *Set-Reset flip-flop*. This is one of many flip-flops used to memorize a bit of information. Flip-flops are fundamental blocks used to build more complex sequential networks. We classify the flip-flops according to their *"logical type"* and to their *"command type."*

The Logical Type

The type of flip-flop whose function is to activate the output (*Set* it) or deactivate it (*Reset* it) is the logical *Set-Reset*. Further on, we will examine other logical types, such as *D* (*Delay*), which memorizes the output value of the unique input *D* and the variant *JK* of the *Set-Reset*. This variant substitutes inputs *J* and *K* with *S* and *R*, respectively, and allows the output's state to reverse. We will also see logical types *T* and *E* that derive from *JK* and *D*, respectively.

As we will see, the logical type is described by its *function table*, which indicates the output values in function of the *logical inputs*, that is, the inputs that characterize the logical type of flip-flop and that give it its name.

The Command Type

The command type describes input behavior, which comes in three forms: *direct*, *level-enabled*, or *edge-triggered*.

In the case of direct command, logical input action is not subordinated to any other enable or synchronization input but it directly controls the behavior of the flip-flop which, in this case, is called *"asynchronous."*

In the other two cases, there is an added input that enables/disables logical inputs. Enabling can happen simply as a function of the logical level of this added input (*level-enabled* command), or when it presents a logical level transition (*edge-triggered* command).[1] In this case, the flip-flop is called *"synchronous."*

Let's turn our attention to direct command types. Before we begin to deal with other types of commands, we will need to examine some important concepts like initialization and synchronization of sequential networks.

5.2 Direct Command Flip-Flops

Now, let's look at the three types of direct commands: *SR*, *D*, and *JK*. The *SR* type has already been introduced but some of its possible variants will be dealt with here.

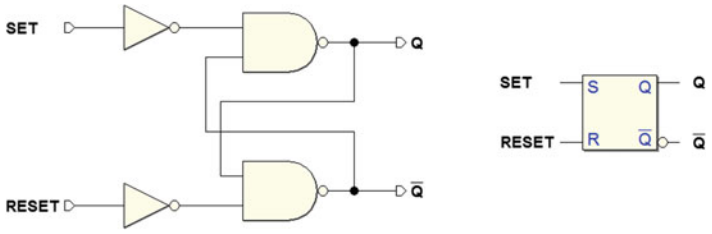
5.2.1 SR Flip-Flop

SR Flip-Flop (Active-High Commands, NAND Version)

The schematic in the figure below represents a *Set-Reset* (SR) flip-flop built with NAND (and NOT) gates and *active-high* inputs. This is like the version examined above but with the added output $\overline{Q}$, which takes the opposite value of Q under normal operating conditions, as we will soon see. This is the structure that implements the

[1] The transition can be "*positive*" (from 0 to 1, or "*rising edge*") or "*negative*" (from 1 to 0, the "*falling edge*").

Set-Reset logical type with *direct command*. Drawn on the right side of the figure is the logical symbol that represents this flip-flop in schematics:



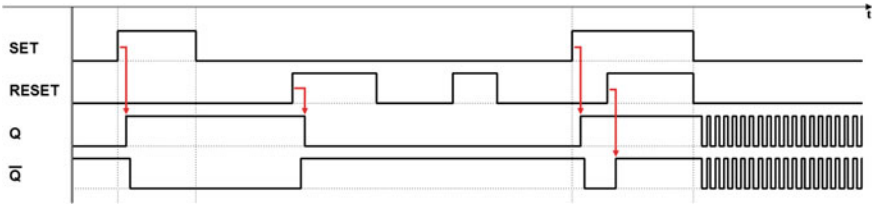
As we have seen before, this is a sequential network that can *memorize a bit*. The activation of the input SET memorizes a 1 in the cell, while the input $RESET$ imposes a 0. Given their opposite actions, it would naturally make little sense to activate SET and $RESET$ at the same time. Below, its *function table* shows what we have just described in English. The table gives both the outputs Q and $\overline{Q}$.

Set-Reset Flip-flop (Active-high Commands)				
SET	$RESET$	Q	$\overline{Q}$	
1	1	Q_p	$\overline{Q_p}$	Previous state
0	1	1	0	SET command
1	0	0	1	$RESET$ command
0	0	1	1	Invalid

The table shows that if the inputs are kept *idle* (at 0), the flip-flop stays in its previous state (Q_p, $\overline{Q_p}$), that is it maintains the previously memorized bit in the output. The next two rows describe what the SET and $RESET$ commands do. To be thorough, the last row in the table shows the *invalid* case, where both command inputs are activated.

In this network, the *invalid* combination simultaneously activates the two outputs: Q and $\overline{Q}$. The invalid configuration deserves to be treated separately later on because this limit condition has technically interesting aspects that depend in part on the specific configuration of the circuit.

The following timing diagram, obtained with the *Deeds* simulator, shows how the flip-flop behaves under various driving conditions. The duration of the signals and the visual scale of the diagram were chosen in order to show the *delay times* between the activation of SET, $RESET$, and the outputs Q and $\overline{Q}$, as evaluated by the simulator:



Let's analyze the behavior of the network point by point.

- In the simulated interval, we assume $Q = 0$ and $\overline{Q} = 1$ at the beginning.
- The activation of SET forces output Q to 1 and, accordingly, $\overline{Q}$ to 0.
- After SET is deactivated, $Q = 1$ remains memorized in the flip-flop.
- Activating $RESET$ forces output Q to zero.
- After $RESET$ is deactivated, $Q = 0$ remains memorized.
- Further activations and deactivations of the $RESET$ input produce no changes because output Q is already at zero.
- Output Q changes to 1 only on the next activation of SET.

In the final part of the diagram, we see the effects of applying the *invalid* input configuration. When SET and $RESET$ are both active, outputs Q and $\overline{Q}$ are both forced to 1. By examining the logical network of the flip-flop, it is easy to verify that the feedback has no effect on the NAND inputs under this condition since there is a 0 on the other input.

The network has lost both the feedback and the data memory! We can get out of this anomalous situation with no problem if we first deactivate one of the inputs and then the other. We will then get to an input configuration that would force the memorization of a known value that, at that moment, is coherent with the value of SET and $RESET$.

The timing diagram shows the *limit condition* where the inputs SET and $RESET$ are simultaneously deactivated. Normally, if SET and $RESET$ are both inactive, we get the memorization of a value. In this case, however, the information has just been lost. The simulator shows that the outputs oscillate. That is, Q and $\overline{Q}$ periodically switch between the two levels, at the same time and with the same logical value.

The simulation produces this result because the model of the components is simplified and idealized; the logical gates have the same delay time (transport). When the inputs switch at the same time, they cause the outputs to switch at the same time, too. The feedback brings them to the inputs, which then causes a further change in the level of both the outputs together and on the same instant, and so on.

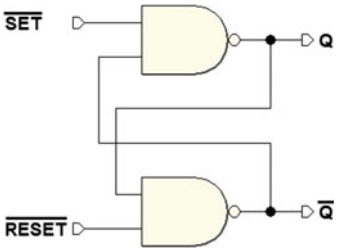
This behavior would be very unlikely in a real network. In reality, the logical components are *nonlinear amplifiers*. An in-depth study of their behavior in the limit case examined here could become very complex and involve concepts like *metastability*, which will be dealt with further on.

For our purposes, it suffices to point out that the real gates' propagation times are similar but not identical. One of the gates will be faster than the other, and the feedback quickly ends up stabilizing the outputs by forcing the flip-flop into the memorization condition (although to an a priori *unknown value*).

SR Flip-Flop (Active-Low Commands, NAND Version)

If the two inverters are eliminated, we get the same flip-flop with active-low commands. Because of its simplicity, the SR flip-flop with this type of command forms the *base structure* on witch other logical types are built on.

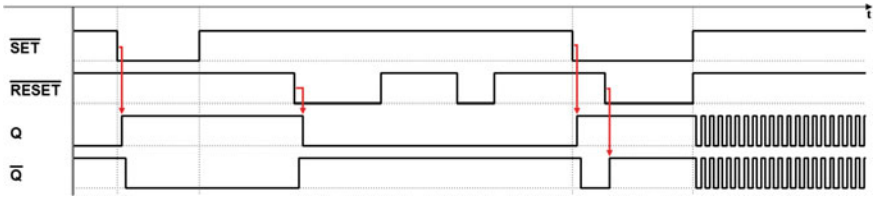
Hereafter, we will refer to this network as *"base cell."*



Obviously, this type is not substantially different from the type with active-high commands. Only the logical level of the command changes. Below, we have its *function table*. As we can see, this is just like the previous one except that the inputs are active-low.

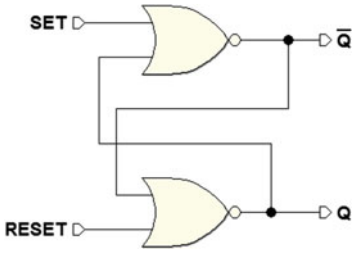
Flip-flop *Set-Reset* (active-low commands)				
$\overline{SET}$	$\overline{RESET}$	Q	$\overline{Q}$	
1	1	Q_p	$\overline{Q_p}$	Previous state
0	1	1	0	SET command
1	0	0	1	$RESET$ command
0	0	1	1	Invalid

The timing diagram below shows a simulation sequence that is identical to the previous one but with complemented input signals.



SR Flip-Flop (Active-High Commands, NOR Version)

To be thorough, let's now outline a version of the *Set-Reset* flip-flop with NOR gates. To obtain that, we go back to the network composed only of an AND and an OR, seen previously, but this time let's transform it into a network with only NOR gates. Ignoring the intermediate steps where the NOT gates are eliminated, we obtain the following network:



This cell has active-high inputs. Note that, unlike the NAND version, here, output Q is generated by the gate that receives the $RESET$ signal and output $\overline{Q}$ by the gate that receives SET.

Despite its simplicity, this NOR version is not commonly used since it is more convenient to use NAND gates in many technologies. Here is its function table.

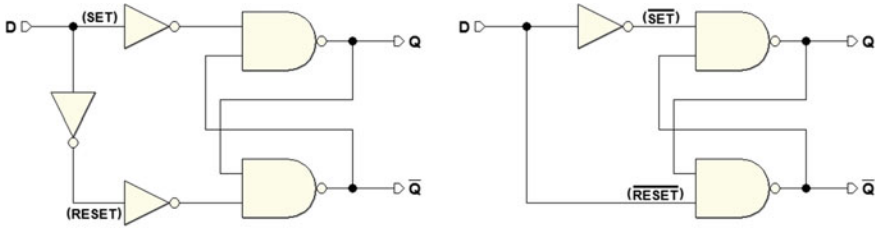
Set-Reset Flip-flop (NOR gate version)				
SET	$RESET$	Q	$\overline{Q}$	
0	0	Q_p	$\overline{Q_p}$	Previous state
1	0	1	0	SET command
0	1	0	1	$RESET$ command
1	1	0	0	Invalid

This is similar to the table for the flip-flop with NAND gates except for its behavior in the invalid condition. Here, when both inputs are activated, the network responds with both outputs at 0. The analysis made before on simultaneously deactivating the inputs is still valid.

5.2.2 D Flip-Flop

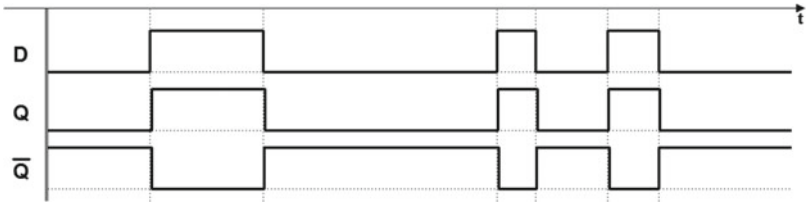
Here, we introduce the concept of the logical type D flip-flop. We want one single data input D, rather than two separate SET and $RESET$ controls. The idea is to simply memorize the bit by submitting it at the input and to avoid the problems related to the *invalid* configuration.

In the figure below, on the left, we get rid of one input by adding a NOT to the SR flip-flop structure with active-high inputs. The data in input D is applied to the SET, while we attach $\overline{D}$ to the $RESET$. Once the two cascading NOTs are eliminated, we get the network below right:



The NOT also assures that the inputs $\overline{SET}$ and $\overline{RESET}$ of the base cell will never have the same logical level. This prevents the critical condition previously discussed.

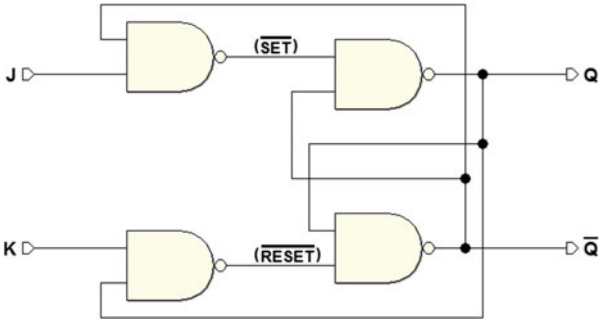
However, as seen in the timing diagram, output Q always reproduces input D. There is no command configuration that memorizes the data so this circuit is useless in the direct command version.



The D flip-flop is an important functional element in the *enabled command* versions, as we will see further on.

5.2.3 JK Flip-Flop

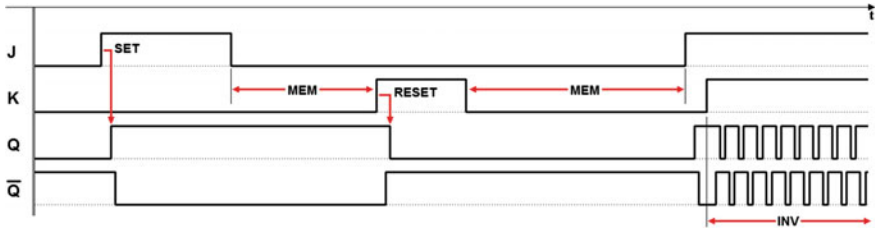
An efficient approach to eliminating the invalid configuration is the JK flip-flop with direct commands. This circuit derives from the *Set-Reset* flip-flop with active-low inputs plus two NAND gates, as shown in the next figure. The new inputs are assigned the name J and K (hence the name JK, in honor of *Jack Kilby* for his contribution to the birth of integrated electronics).



Here, we see that through one of the two new NANDs the input J drives the $\overline{SET}$ conditioned by $\overline{Q}$. Likewise, the input K drives the $\overline{RESET}$ conditioned by Q. This way, $\overline{SET}$ and $\overline{RESET}$ can never be activated at the same time. Here, the function table describes how inputs J and K act on the outputs:

JK Flip-flop				
J	K	Q	$\overline{Q}$	
0	0	Q_p	$\overline{Q_p}$	Previous state
1	0	1	0	SET command
0	1	0	1	$RESET$ command
1	1	$\overline{Q_p}$	Q_p	Toggle

In the first row of the table, J and K are both at 0, so $\overline{SET}$ and $\overline{RESET}$ are at 1, and the outputs keep the previous value. In the timing simulation below, we find this input configuration in the “MEM” time intervals:



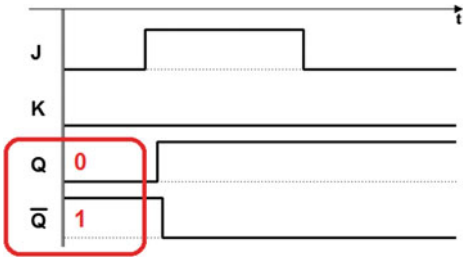
In the second row of the table $J = 1$ and $K = 0$, so the input $\overline{SET}$ is activated and output Q is forced to 1 (operation identified as "SET" in the timing simulation). In the third row, we have the opposite case and output Q is at zero ("RESET" in the same figure).

The JK flip-flop is different from the SR type because the invalid condition in SR is used to accomplish a useful function, the *inversion* of the output values (as shown in the last line of the table), also known as "*toggle*." When both inputs J and K are at 1, $\overline{SET}$ is activated if $Q = 0$, or $\overline{RESET}$ is activated if $Q = 1$. This reverses the outputs (time interval indicated as "INV" in the timing diagram).

Further on, we will see that the *toggle* function is only completely usable in *edge-triggered* JK flip-flops. Under the conditions in the timing diagram above, that is with J and K kept at 1 for a long enough time, outputs Q and $\overline{Q}$ reverse their values *continually*. At every change of outputs Q and $\overline{Q}$, the feedback reverses the values in $\overline{SET}$ and $\overline{RESET}$, in turn creating another inversion, thus giving rise to a cyclical behavior.

5.3 Initialization of a Sequential Network

Before we continue, a small digression: in the timing diagrams that we have studied, the sequences always purposefully begin with $Q = 0$ and $\overline{Q} = 1$, as in the following example:



A specific moment in the normal functioning of the network has been chosen as the time to *begin to observe* the network. In the previous section, we willfully hid an important aspect of all sequential networks. To simplify things, we avoided mentioning the fact that we must launch the network with a *known configuration*. This problem involves all sequential networks and will be dealt with further when we study aspects of their design.

At the launch of a sequential network, we say in technical terms, that it must be *"initialized"* so that it can work coherently in normal operation. There will be many flip-flops in a real network: when the system is activated, in absence of specific measures, every one of these will tend to take on a *random value*.

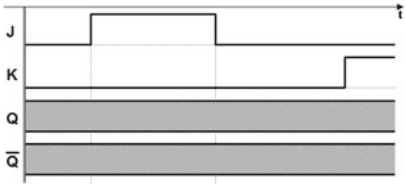
We can see that this is unacceptable for a complex network so we need to turn to circuit techniques that allow us to supply every flip-flop with a known value *before* launching the normal functioning of the network.

In simpler networks, initialization could be irrelevant. For example, for a circuit that makes an LED light flash on a panel, it doesn't make much difference if the LED lights immediately when turned on or whether it lights half a cycle later. In almost all real cases, however, it is unacceptable for the network to start at *random* values.

Think of a digital network that controls something *intrinsically dangerous*, like opening the bulkheads in the dyke of a hydroelectric basin. We could not afford for the flip-flops to be positioned randomly at the start of a system that opens the bulkheads. Rather, we need to take every precaution so that they stay *rigorously closed*, opening only after an explicit command.

Let's return our attention to the JK flip-flop with direct commands. With a real component, when powering on the circuit, output Q of the elementary memory cell will be forced to an *a priori unknown* value. From that moment, the network will start to work (even though it starts from a random value, 0 or 1) and will follow the path imposed by the evolution of the inputs.

Still, if we try to *simulate* the network we see that the simulator cannot *resolve* the network's behavior and it gives us back the result in the figure below, where the bands represent an *unknown* value.



The simulator's behavior is *formally correct*: since it does not know the initial value of outputs Q and $\overline{Q}$, the simulator cannot calculate the following values. As we have seen, they depend on the inputs and also on the previous value of the outputs, which is unknown.

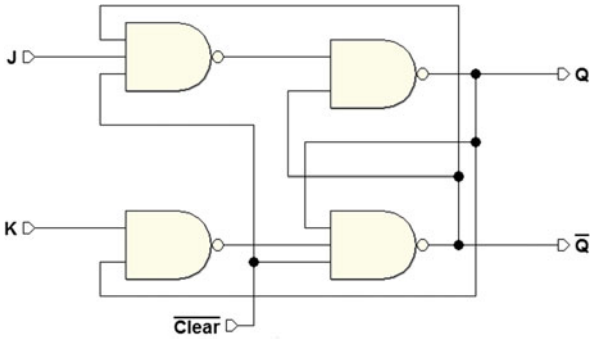
The simulator is a *tool for development and verification*, and for it to be efficient, the developer must be able to see errors and oversights. Here, the simulator tells us that the network *is not formally able* to begin working from a *known configuration*.

If the simulator resolved these situations by "hypothesizing" starting values, this would not do us any great favor because it could mask possible design errors.

5.3.1 Flip-Flop Initialization Inputs

So how was it possible to do a JK flip-flop network simulation? It was actually another network that was simulated, as you see in the next figure. Here an *auxiliary*

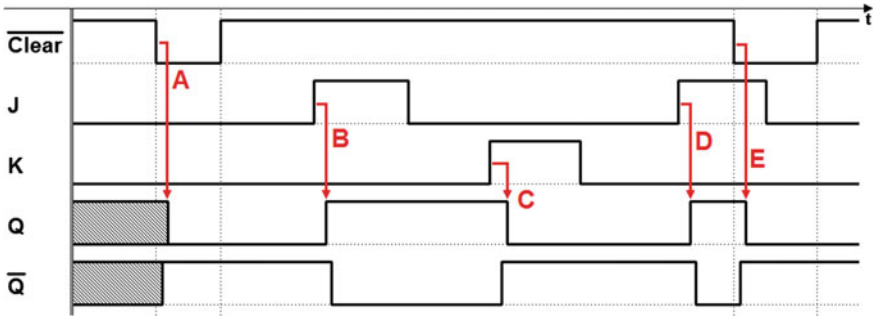
initialization input called $\overline{Clear}$ acts on two of the network's NANDs (the *negation bar* highlights the fact that it is *active-low*):



As we learn from the logical schematic, when $\overline{Clear} = 1$, the network behaves as if there were no $\overline{Clear}$ input. When it is brought to 0, however, the configuration that brings output Q to zero is *forced* on the elementary cell.

It is important to note that input $\overline{Clear}$ is *prioritized* over J and K. In fact, for all the time that it is active, $\overline{Clear}$ *keeps the output at zero* and prevents J and K from influencing the elementary cell.

In the figure below, we see the network simulation with the activation of input $\overline{Clear}$ highlighted.



As we can see, outputs Q and $\overline{Q}$ are initially indeterminate but the activation of input $\overline{Clear}$ forces (A) a definition (the delays are due to propagation times). Then deactivating $\overline{Clear}$ allows the flip-flop to work freely (B)(C)(D), under the control of inputs J and K. Finally, on the right, we see $\overline{Clear}$ activated once more (E), which forces the flip-flop to return to zero.

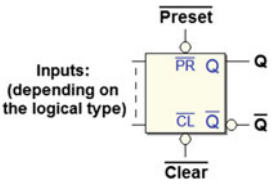
To be thorough and versatile, flip-flops generally have two initialization inputs: $\overline{Clear}$ and $\overline{Preset}$. The action of $\overline{Preset}$ is perfectly symmetrical to $\overline{Clear}$: it forces the output to 1.

Notice that the function of inputs $\overline{Clear}$ and $\overline{Preset}$ is functionally equivalent to that of the $\overline{RESET}$ and $\overline{SET}$ of an RS flip-flop with active-low commands. Nevertheless, the purpose of initialization inputs remains and they should not be used differently.

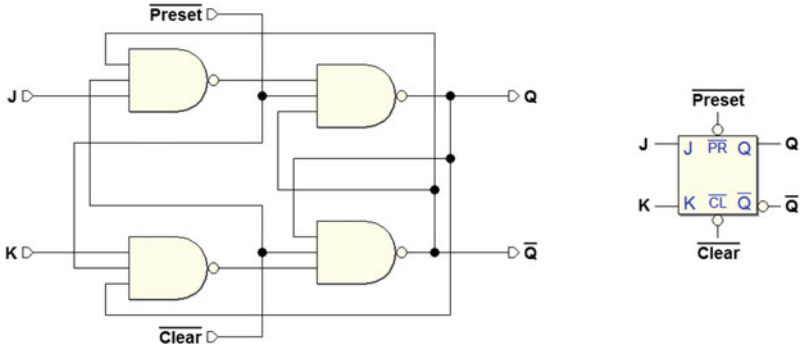
For this reason, $\overline{Clear}$ and $\overline{Preset}$ should be considered *mutually exclusive*. In order to initialize the flip-flop, we use only one of them based on the project needs, while the other will be connected to a constant high logic level, so as to remain inactive.

Regardless of logical type, inputs $\overline{Clear}$ and $\overline{Preset}$ exist in all flip-flops, whether they be physical or CAD system component.

In the figure below, we see the terminations common among all the logical types: inputs $\overline{Clear}$ and $\overline{Preset}$ and outputs Q and $\overline{Q}$. The inputs are shown here in a generic form seeing that they vary depending on the logical type.



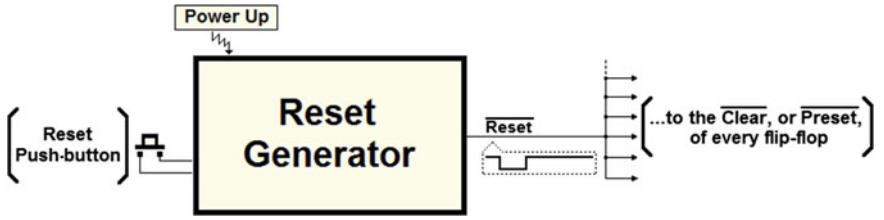
For completeness, the entire logical schematic of the JK flip-flop with direct commands, inputs J and K, outputs Q and $\overline{Q}$, and inputs $\overline{Clear}$ and $\overline{Preset}$ appears below left. The one on the right is the corresponding schematic symbol.



5.3.2 *Generating an Initialization Signal*

After discussing the need to initialize sequential networks, we must now examine how the signal that is fed to inputs $\overline{Clear}$ (or $\overline{Preset}$) of the individual flip-flops is produced when the system is powered up.

The next figure shows what is called a *Reset Generator*: a mixed analog and digital circuit, that we will study only from the functional point of view.



When the system is powered up, this circuit *automatically* generates a pulse of a determined duration on the $\overline{Reset}$ line, that is linked to all the flip-flops in the network (either to $\overline{Clear}$ or to $\overline{Preset}$ according to need).

There is often a push-button that the user may press to manually re-initialize the system (as with PCs for example).

Activating the $\overline{Reset}$ keeps all the system's flip-flops blocked during the initial *power up transient*. The power supply takes a certain amount of time (a few tens mS) to bring the circuit voltages from zero to nominal values. The generator keeps the $\overline{Reset}$ active for the time necessary and then deactivates it, allowing the system to start working.

5.4 Level-Enabled Flip-Flops

All the flip-flops discussed previously share the commonality that the action of the logical inputs is not subordinated to any other input. When designing a digital system, however, it is often necessary that the outputs of the flip-flops change at the *same instant*, namely that they be *synchronized*.

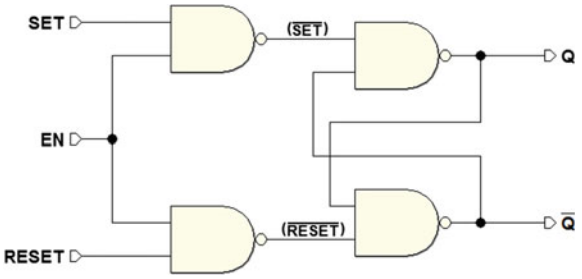
In this section, we will deal with *level-enabled* flip-flops, where a specific input conditions the action of the inputs. This type of flip-flop is also called *Latch*.

Take note: to make the explanation clear, the logical networks of the flip-flops presented here are simplified and do not include initializing circuits (inputs $\overline{Clear}$ and $\overline{Preset}$ will not appear in the descriptions). Remember that real flip-flops of course have these inputs, as described before.

5.4.1 SR-Latch Flip-Flop

In the following schematic, the base structure of the flip-flop is SR type with active-low inputs and has been integrated with two NAND gates.

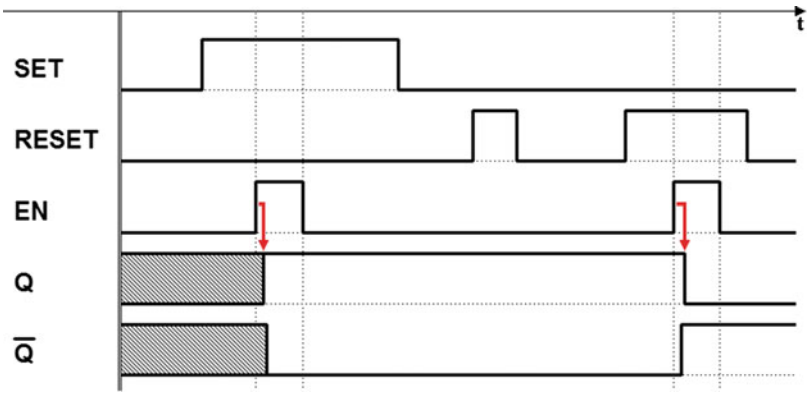
The two new gates make the SET and $RESET$ act upon the base cell only if the *Enable* input $EN = 1$. If so, the network's overall behavior is identical to that of the SR flip-flop with direct command and active-high inputs.



If $EN = 0$, the two NANDs that condition the input generate a 1, regardless of the value of inputs SET and $RESET$, so the base cell maintains the value of the outputs. The function table shown below summarizes this information.

Set-Reset Flip-flop (Level-enabled)					
EN	SET	$RESET$	Q	$\overline{Q}$	
0	–	–	Q_p	$\overline{Q_p}$	Previous state
1	0	0	Q_p	$\overline{Q_p}$	Previous state
1	1	0	1	0	SET command
1	0	1	0	1	$RESET$ command
1	1	1	1	1	Invalid

In the timing simulation below, we observe the behavior of the SR flip-flop as the inputs vary (for simplicity's sake, the invalid configuration was omitted).

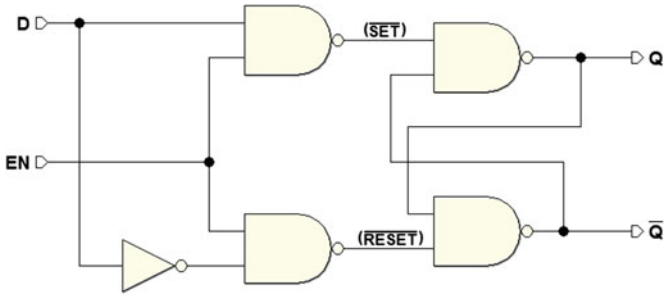


In the first part of the image, the simulator cannot predict an initial value so it indicates the outputs as *undefined*. The first activation of input EN allows input SET to force the output Q to 1. Afterward, the flip-flop keeps the value of the outputs since input EN is inactive. Then EN is active again, while input $RESET$ is active, forcing output Q to zero.

As we can see, input EN is used to restrict the changes of the outputs of the flip-flop within the time intervals it is active. By shortening these intervals, we get the first form of *synchronization*, as we will see further on.

5.4.2 D-Latch Flip-Flop

In the schematic below, a NOT was added to the structure of the SR latch flip-flop. Input SET of the SR flip-flop was renamed D. Negated, it drives the input $RESET$.

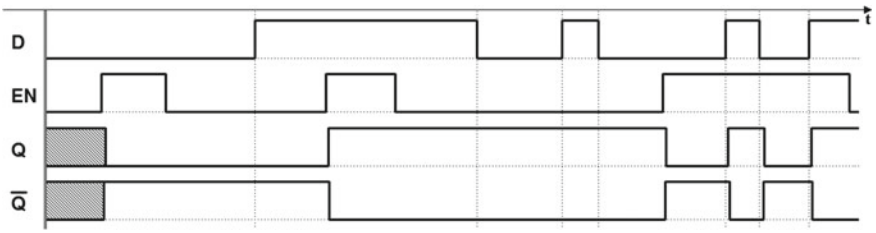


This way, we obtained a D flip-flop but this time the enable input EN is present (this flip-flop is usually called *D-Latch*). The invalid configuration is automatically eliminated by the NOT, as we saw with the direct command D flip-flop. Thanks to the enable input, however, in this case we can *memorize a bit*. The *function table* for this flip-flop is shown below:

D-Latch Flip-flop				
EN	D	Q	$\overline{Q}$	
0	–	Q_p	$\overline{Q_p}$	Previous state
1	1	1	0	SET command
1	0	0	1	$RESET$ command

When EN is at 1 (active), output Q copies the value of input D, whereas when $EN = 0$, there is no transmission between input D and the base cell of memory, which keeps its value.

Due to its simplicity and economy, the D-Latch flip-flop is commonly used to make many types of *registers* and *semiconductor memory devices* that have applications in sequential networks and in computers in general. To store the information, we need first to submit the bit to be memorized at input D, and then activate and release input EN. The timing diagram below shows a typical sequence of use.



As in the other cases, we do not know the value of the outputs at the start of the simulation. First, input D of the data is set to 0 and, during this interval, input EN

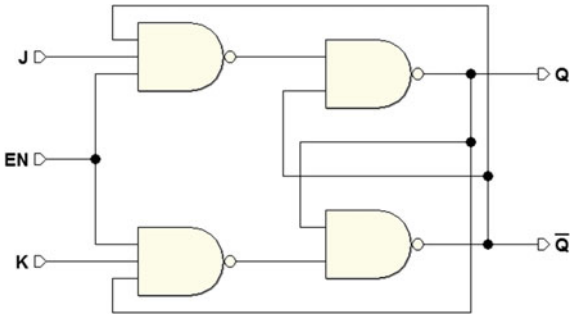
is activated for a certain time. The data at input D is *transferred* to output Q. When EN is deactivated, the flip-flop *captures the value* at the output at that moment (it is said to *memorize the last transited value*).

The same activation sequence is repeated immediately after, but with input data $D = 1$: the flip-flop memorizes 1 on the output. The diagram shows that in the interval when $EN = 0$, input D does not produce changes in the outputs even though it changes many times.

Lastly, the diagram shows that EN is maintained at 1 for a certain period of time. We see that the output repeats the value of the input D (when it is enabled, the flip-flop is said to be *transparent*).

5.4.3 JK-Latch Flip-Flop

The schematic below shows a variant of the direct command JK flip-flop with an added enable input EN:



As with the direct command JK flip-flop, the input J is conditioned by output $\overline{Q}$, and input K by output Q, making it impossible to simultaneously activate the inputs of the base cell.

Input EN conditions both the inputs J and K. When EN is active the network behaves exactly like a direct command JK. Otherwise it keeps the previously memorized value (the condition $EN = 0$ is equivalent to having both J and K at 0). The function table summarizes the relationship among the inputs J, K, and EN, and the outputs Q and $\overline{Q}$.

JK Flip-flop (Level-enabled, or JK-Latch)					
EN	J	K	Q	$\overline{Q}$	
0	–	–	Q_p	$\overline{Q_p}$	Previous state
1	0	0	Q_p	$\overline{Q_p}$	Previous state
1	1	0	1	0	SET command
1	0	1	0	1	$RESET$ command
1	1	1	$\overline{Q_p}$	Q_p	Toggle

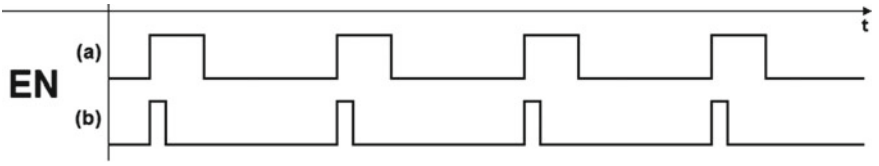
This type of *level-enabled* JK flip-flop is not commonly used because the *toggle* configuration ($J = 1$ and $K = 1$) can only be used if the duration of EN is shorter than the network propagation time. As mentioned, the *toggle* function is used in *edge-triggered* JK flip-flops.

5.5 Synchronization of Sequential Networks

Level-enabled structures only partially satisfy the requirements of modern digital systems where it is required that the outputs of flip-flops change *periodically* and *simultaneously*. This type of system is called *synchronous*. To design a synchronous system, we must use more elaborate sequential components than those we have seen so far. Above all it is important to completely understand what *synchronization* of sequential networks really means and what issues are involved. Before introducing the flip-flops used in synchronous systems in the following sections, we will first introduce the concept of *synchronicity* using familiar elements.

5.5.1 The Synchronization Signal

Consider a network that uses level-enabled flip-flops. To satisfy the requirement of *periodicity*, the enabling command EN must take on a cyclical form as seen in part (a) of the figure below.



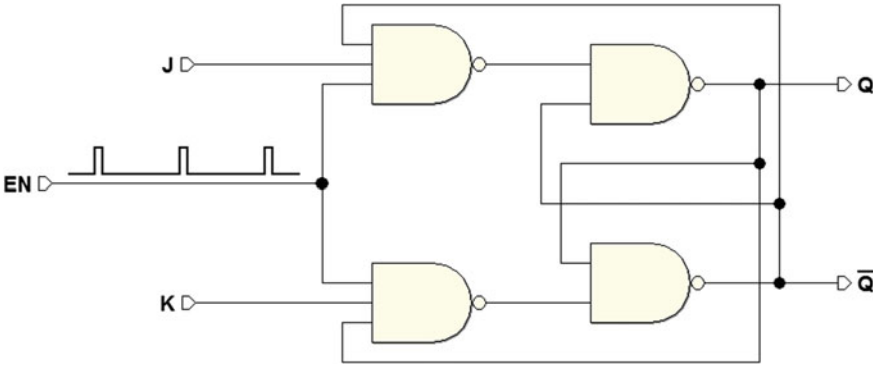
Flip-flops will change their outputs only in response to the *periodic activation* of the EN enable command. This becomes the *synchronization signal* meaning the *time reference* for the time evolution of the network. Level-enabled flip-flops, however, only partly guarantee simultaneous changes.

Their outputs can only change when EN is active and since the activation interval is finite, transitions can still occur at different times given that outputs can change the whole time EN is active.

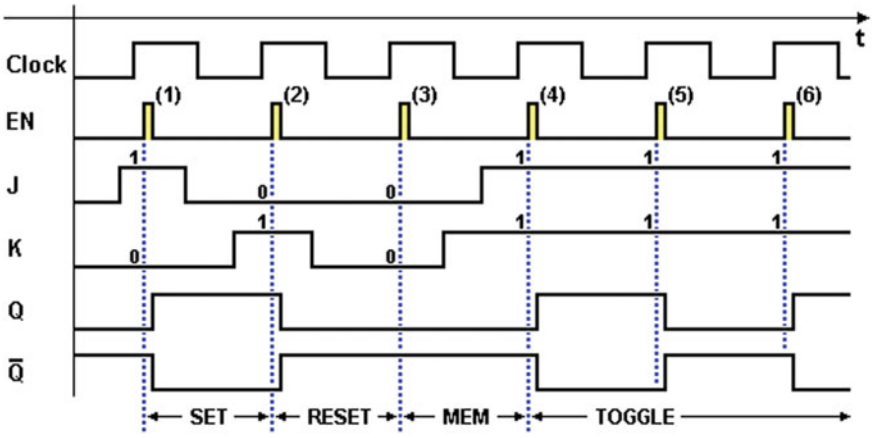
If the duration of EN is restricted (part (b) in the figure above), it reduces the interval in which the outputs can change. This brings us closer to the outputs' *simultaneity* requirement.

5.5.2 Pulse Command in Level-Enabled Flip-Flops

Let's consider a level-enabled JK flip-flop identical to the one examined in previous sections. Let's drive its EN enable input with a periodic pulse sequence.



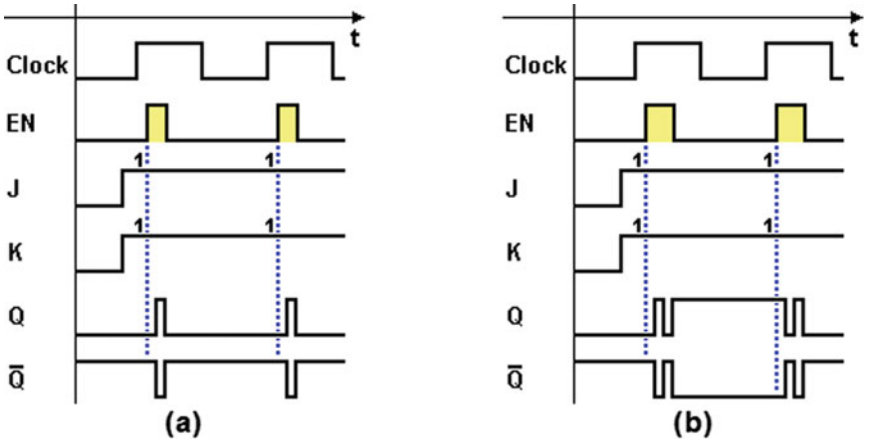
The flip-flop will change outputs Q and $\overline{Q}$ when the pulses on EN occur, as seen in the timing diagram below, which describes typical usage. Note that the flip-flop responds to inputs and only changes its outputs (after the network's propagation times) when EN is activated (see the vertical dotted lines).



In the previous figure, the pulse (1) on EN where $J = 1$ and $K = 0$, forces output Q to 1. Pulse (2) where $J = 0$ and $K = 1$ forces Q to zero. Pulse (3) where $J = 0$ and $K = 0$ keeps the previous output value. From pulse (4) on, where $J = 1$ and $K = 1$, the output value is inverted at every cycle (*toggle*).

Note that when EN is active and $J = 1$ and $K = 1$, the short duration of the enable command does not allow for the continuous switching of the outputs typical of the implementation of the same flip-flop with direct command. The duration of the pulse must be carefully assessed at the project level in relation to network timing.

The two figures below describe two abnormal situations, where the excessive duration of the EN pulse allows for two output inversions in case (a) and three in case (b).



The function table below summarizes the logical behavior of the level-enabled JK flip-flop that is driven impulsively.

JK Flip-flop (Level-enabled), *Impulsively Driven*					
J	K	EN	Q	$\overline{Q}$	
0	0	⊓	Q_p	$\overline{Q_p}$	Previous state
1	0	⊓	1	0	SET command
0	1	⊓	0	1	$RESET$ command
1	1	⊓	$\overline{Q_p}$	Q_p	Toggle

The symbols in the EN column represent the *pulse command*. Keep in mind while reading the table that a given combination of inputs J and K corresponds to the indicated outputs only after input EN is *pulse activated*.

5.5.3 The "Clock" and the "Edge-Triggered Command"

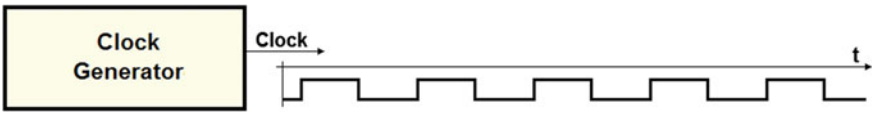
This example has shown us that level-enabled flip-flops make it possible to create synchronous systems. As we have seen, however, defining the length of the pulse poses a downside. In the example above, the inputs of the flip-flop are supposed to remain stable during EN activation, but it is not always possible to fulfill this condition, necessary to guarantee *simultaneity* of output changes.

To resolve these problems, we need to shorten the length of the activation pulse as much as possible in level-enabled flip-flops. Technically, however, we cannot go lower than a certain minimum value. Historically, level-enabled flip-flops were

mainly used in the first logical circuits using *discrete components*, because of their simplicity.

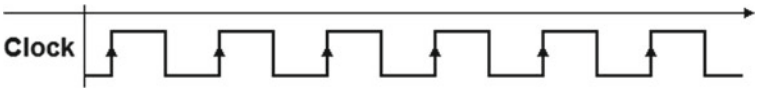
With integrated circuits, which began to substitute discrete components as of the 1970s, the situation has evolved. Since microelectronics greatly lowered the cost of a single logical function, more dependable and economical, although more circuitally complex structures have been developed. Current digital systems no longer employ *level-enabled* flip-flops but rather *edge-triggered*, i.e., from a *level transition* of the synchronization signal.

The synchronization signal in digital systems is traditionally called *"clock."* A clock is a two-level periodic signal generated by a dedicated circuit, the *Clock Generator*:



The timing evolution of a periodic signal is called a *"waveform."* In the example below, the signal is a symmetrical *square wave* with a *duty cycle* (percentage of the time the signal is high in its period) of 50%. The designer chooses the clock's *oscillation frequency* based on the system specifications.

In the figure below, the clock signal's *transitions* from 0 to 1 are highlighted by arrows pointing to the top:

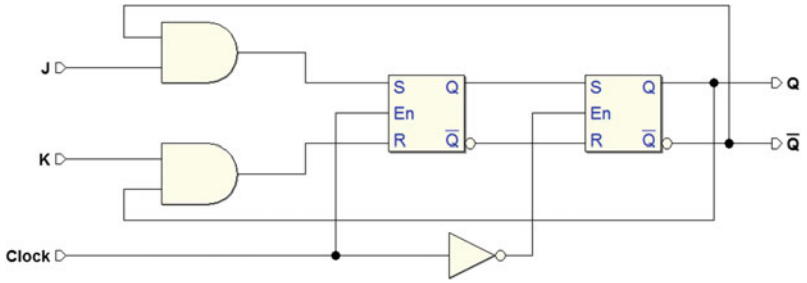


This type of transition is called the *positive (or rising) edge*; its opposite, the *negative (or falling) edge*. As we will soon see, this type of device will render the timing evolution of digital systems rigorously *synchronized* by the edge of the clock (there are positive-edge-triggered components and negative-edge-triggered components).

5.5.4 Master–Slave Structure

The first general-use structure where the change in outputs is synchronized by an edge was the *master–slave* structure. It is no longer commonly used but it is useful to examine it before going on to more modern structures.

The master–slave configuration uses two level-enabled RS flip-flops. The *Clock* signal controls the triggering of the two flip-flops and thus the data transfer from the input to the output. Here we refer to a master–slave structure that creates a JK flip-flop (the logic type it was developed for).



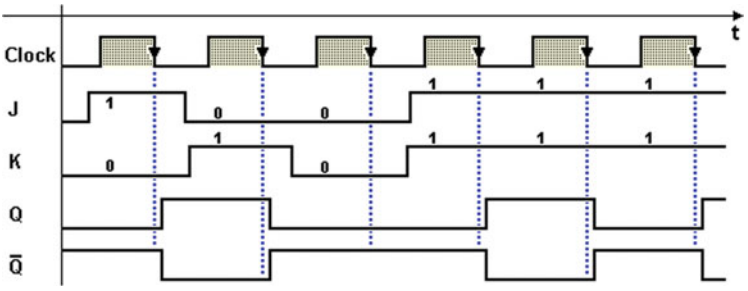
The RS flip-flop that the input data is applied to (*master*) is driven directly by the $Clock$, while the other (*slave*) is controlled by $\overline{Clock}$.

When $Clock = 1$, the data on J and K is applied to the master flip-flop but this has no effect on the slave since its inputs are disabled ($\overline{Clock} = 0$). When the $Clock$ goes back to 0, the master input is disabled and the data is transferred to the slave.

The function table that describes the JK master–slave is similar to that of the level-enabled JK flip-flop that is driven impulsively, but note that the output changes occur on the falling edge of the $Clock$.

JK Flip-Flop (*Master-slave*)					
J	K	$Clock$	Q	$\overline{Q}$	
0	0	⊓↓	Q_p	$\overline{Q_p}$	Previous state
1	0	⊓↓	1	0	SET command
0	1	⊓↓	0	1	$RESET$ command
1	1	⊓↓	$\overline{Q_p}$	Q_p	Toggle

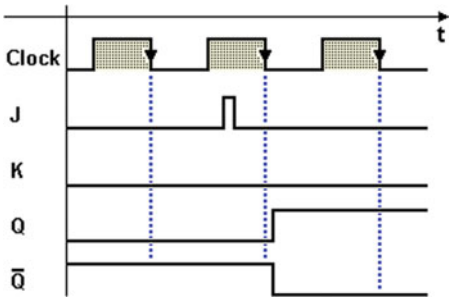
The timing diagram in the next figure shows the behavior of the master–slave JK flip-flop. The *semiperiods* of the $Clock$ when it is at 1 are shown by dotted lines to demonstrate that a master–slave flip-flop acquires inputs the whole time the $Clock$ is at 1 while it changes its outputs according to its *falling edges*.



In the figure above, the triggering sequence is the same as the one for the JK flip-flop that is driven impulsively. All the possible combinations for J and K are represented including the *toggle* mode.

The master–slave structure eliminates the limitation on the duration of triggering pulses typical of the level-enabled structures. Note, however, that inputs J and K must be *stable* when the $Clock$ is at 1 to make the JK master–slave flip-flop work

correctly. One drawback of this structure is that it is sensitive to the changes in J and K during this interval.



In the figure above, for example, a brief pulse on J (if $Q = 0$) is memorized in the master flip-flop causing a change in output on the falling edge.

Thus, we can say this structure is edge-triggered, provided that the inputs for the time when the $Clock$ is at 1 are guaranteed to be stable. This problem was resolved by a variant called *data lock-out (DLO)*, which takes on the values of J and K on the *$Clock$'s rising edge* and changes its outputs on the *falling edge*, while remaining impervious to the changes when the $Clock$ is at 1.

Master–slave and DLO triggered flip-flops are now obsolete. They have been replaced by flip-flops that work *on only one edge*. We examine them in the next section.

5.6 Edge-Triggered Flip-Flops

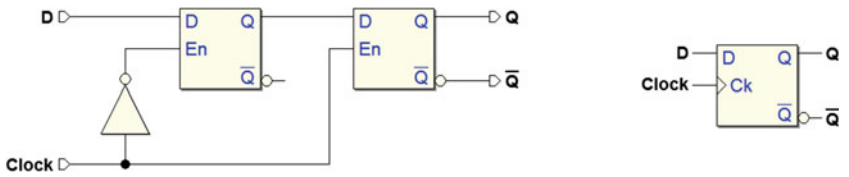
Edge-triggered flip-flops are widely used in the current implementations of digital system. In this structure, both the acquisition of data and the change of outputs occur on a transition (*edge*) of the $Clock$.

If the active edge is the rising one the structure is called *Positive-Edge-Triggered* (PET); if the falling edge is active, it is called *Negative-Edge-Triggered* (NET).

The flip-flop remains impervious to inputs throughout the rest of the time. Thus, the inputs can change at any time except for a brief interval around the active edge of the $Clock$. This aspect is examined in the next section.

5.6.1 D-PET Flip-Flop

In the figure below left, we observe a logical network that creates the D-PET type *positive-edge-triggered* flip-flop. The structure is reminiscent of the master–slave flip-flop but it uses two level-enabled D flip-flops.



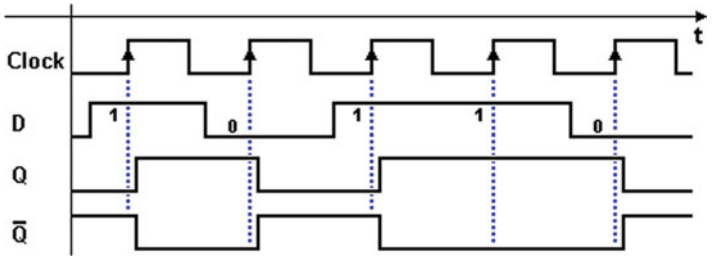
For the whole time that the *Clock* is at 0, the first flip-flop, having an active *En*, follows the variations of input *D*. In the same time interval, the second flip-flop with an inactive *En* keeps the previous value. When the *Clock* changes to 1 the first flip-flop memorizes the last value in D, while the data in its output is transferred to the second flip-flop. The figure at the right shows the logical symbol of the D-PET flip-flop. The small triangle at the input of the *Clock* indicates the sensitivity to the edge.

In the function table below, the symbol in the *Clock* column indicates the *rising edge* to which the changes in the outputs are associated.

D-PET Flip-flop				
D	$Clock$	Q	$\overline{Q}$	
0	↓	0	1	Memorizes 0
1	↓	1	0	Memorizes 1
–	0	Q	$\overline{Q_p}$	Previous state
–	1	Q	$\overline{Q_p}$	Previous state

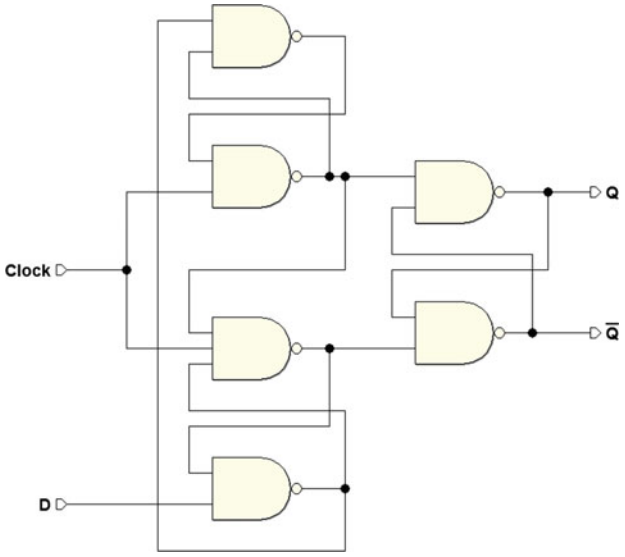
The last two rows on the table are superfluous but they highlight the fact that the flip-flop keeps the previous outputs in the absence of an edge. Further on, these two rows will not appear in the other tables.

The timing diagram in the next figure shows that the output Q of the flip-flop copies the value of input D on the *Clock*'*s* rising edges and keeps it stable for the length of one clock period.



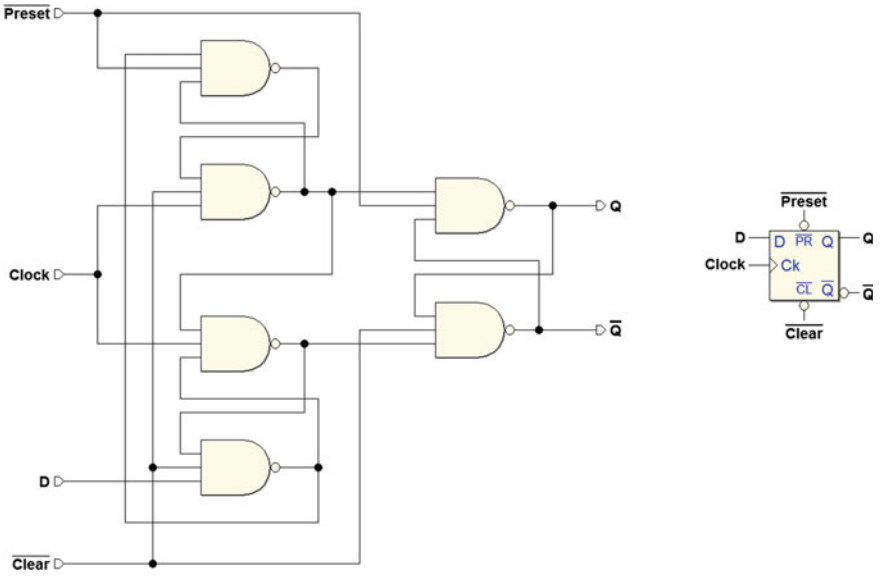
Note that Q changes only in response to the *Clock* rising fronts. A signal that keeps a *fixed timing relation* with the *Clock* is defined as *"synchronous."*

The figure below shows the circuit of the D-PET flip-flop, which is commonly used in many commercial products. It is fast and economical, and it is built using three *base cells.*



Its function table and timing diagram coincide with those of the D-PET structure we have just seen before.

The figure below shows the schematic of a D-PET flip-flop like the previous one but with $\overline{Clear}$ and $\overline{Preset}$ networks. On the right, we see the corresponding circuit symbol.



The top rows on the function table of the D-PET flip-flop with $\overline{Clear}$ and $\overline{Preset}$ (seen below) describe how the initialization inputs function.

D-PET Flip-Flop (with $\overline{Clear}$ and $\overline{Preset}$)						
$\overline{Clear}$	$\overline{Preset}$	D	$Clock$	Q	$\overline{Q}$	
0	1	–	–	0	1	Action of Clear
1	0	–	–	1	0	Action of Preset
0	0	–	–	1	1	(invalid)
1	1	0	↓	0	1	Memorizes 0
1	1	1	↓	1	0	Memorizes 1

Note that D and $Clock$ have no effect on the component when either $\overline{Clear}$ or $\overline{Preset}$ is active.

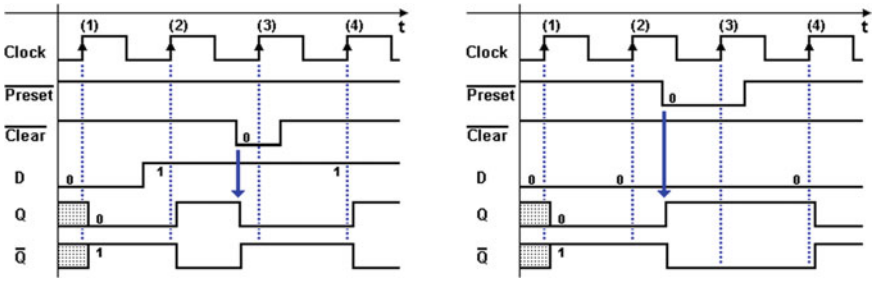
The behavior of the D-PET flip-flop with inactive $\overline{Clear}$ and $\overline{Preset}$ (the bottom two rows of the table) corresponds to the previous table of the flip-flop with no initialization inputs.

For structures based on the NAND cell, simultaneously activating $\overline{Clear}$ and $\overline{Preset}$ forces outputs Q and $\overline{Q}$ to 1 creating a parallel situation to that of the base cell of the SR flip-flop, with the same problems. However, this combination of values is easily avoidable because we would normally choose to initialize the flip-flop by using $\overline{Clear}$ or $\overline{Preset}$ and connecting the other to the constant 1.

Now let's analyze the time behavior of the D-PET flip-flop, including its response to the activation of inputs $\overline{Clear}$ and $\overline{Preset}$.

To make this analysis, we must focus not only on the level of input D signal when the active edge of the $Clock$ occurs but also on the level of the asynchronous $\overline{Clear}$ and $\overline{Preset}$.

As we have seen, $\overline{Clear}$ and $\overline{Preset}$ are *prioritized* and act *independently* of the $Clock$. The figure below shows the example of $\overline{Clear}$ activation and the other, the activation of $\overline{Preset}$.



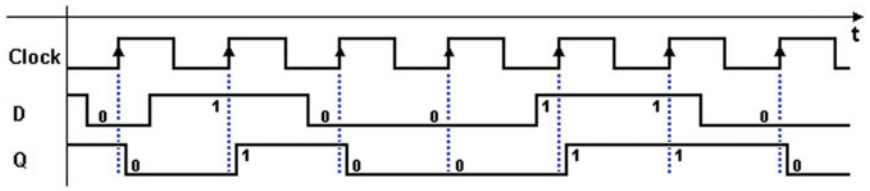
In the example on the left, we initially do not know the value of Q but with the rising edge (1) of the $Clock$ the value 0 on D is memorized by the flip-flop and appears in output Q. At the rising edge (2), the flip-flop takes on the new value of D, forcing Q to 1.

Activating $\overline{Clear}$ involves asynchronously forcing the output to zero and the insensitivity to edge (3) of the $Clock$. Note that deactivating $\overline{Clear}$ produces no changes in the output.

In the example on the right, the $\overline{Preset}$ input is activated, which *forces* the output to 1. Note that the flip-flop is insensitive to the edges of the *Clock* during the activation interval of $\overline{Preset}$ as well.

5.6.1.1 Example (D-PET as "synchronizer")

Finally, let's observe the behavior of output Q in the figure below in relation to input D.



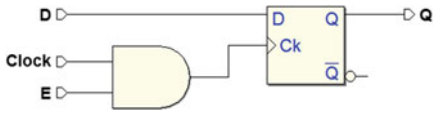
As we can see, input D changes irregularly, without respect to the edges of the *Clock*. At each occurrence of the active edge of the *Clock*, memorization of the signal at input D creates a *"synchronized"* copy in the output, i.e., with a *fixed timing relation* to the *Clock*.

A typical application of the D-PET flip-flop is the *synchronization of signals*. Further on, we will see how important signal synchronization is and the issues involved.

5.6.2 E-PET Flip-Flop

Digital networks, especially those based on large-scale integration devices, employ another type of flip-flop called "E-PET," an extension of the D type. The E-PET flip-flop has the capacity to store an input value only upon request. This differs from the D type, which stores a new value at each active edge of the clock.

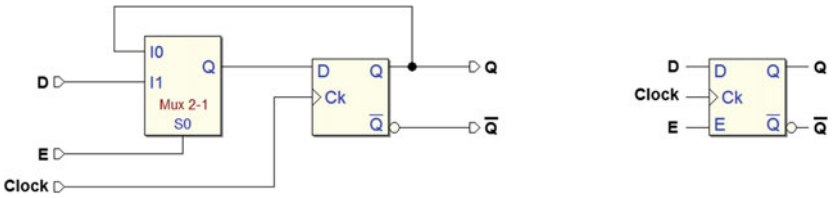
We would be wrong to try to achieve this by using a D flip-flop and condition its *Clock* to an E enable signal, as in the following network:



The logical gate allows us to block the *Clock* when input E is at 0. This technique is not used, however, because the gate's propagation time defeats the requirement that the outputs should be simultaneous to other flip-flops in the system.[2]

[2] In large-scale integration devices that use a high number of flip-flops, the designer takes great care with the physical connections of the clock to avoid time misalignments among the various elements of the network. Logical gates along the clock path are also invalid.

Thus, we make use of a different structure where the input of the D-PET flip-flop is driven by a multiplexer, as in the figure below left. At right is the circuit symbol of the E-PET flip-flop.

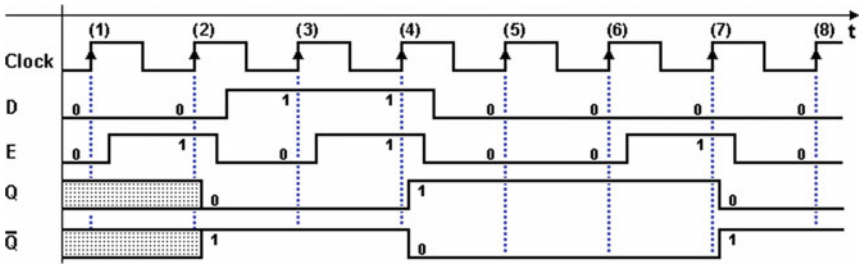


The multiplexer's input E ("Enable") allows us to copy the value of output Q to the input of the D flip-flop if $E = 0$, or the value of the external input D if $E = 1$. When the rising edge of $Clock$ arrives, in the first case the memorized data stays the same. In the second case, output Q takes on the value of external input D. Note that the $Clock$ is not affected.

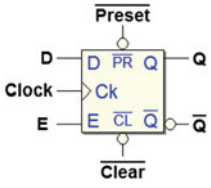
See below the function table of the E-PET flip-flop with no initialization inputs.

E-PET Flip-flop					
E	D	$Clock$	Q	$\overline{Q}$	
0	–	↓	Q_p	$\overline{Q_p}$	Previous value
1	0	↓	0	1	Memorizes 0
1	1	↓	1	0	Memorizes 1

In the timing diagram below, we see the effect of input E, in response to the active edges of the $Clock$. The rising edges of the $Clock$ where the flip-flop memorizes the new value are: (2), (4) and (7).



Finally, let's consider the complete version of the E-PET flip-flop with initialization inputs. Its logical symbol and function table are found below.

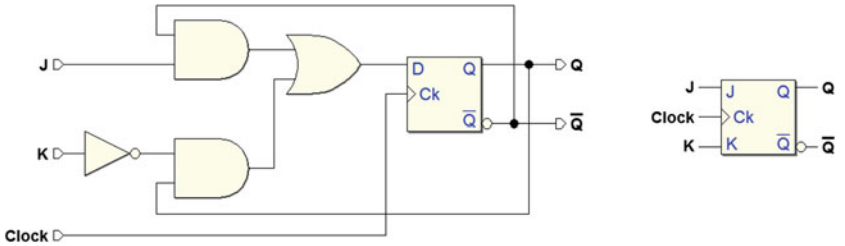


E-PET Flip-flop					
E	D	$Clock$	Q	$\overline{Q}$	
0	–	↓	Q_p	$\overline{Q_p}$	Previous value
1	0	↓	0	1	Memorizes 0
1	1	↓	1	0	Memorizes 1

5.6.3 JK-PET Flip-Flop

In the figure below left, we see a logical network that creates a *positive-edge-triggered* JK-PET flip-flop. We derive this structure from that of the D-PET where input D is controlled by an AND-OR combinational network, which calculates D's value based on inputs J and K and outputs Q and $\overline{Q}$.

The image on the right shows the corresponding logical symbol. The triangle at the input of $Clock$ indicates it is sensitive to the edge.



The function table that describes the JK-PET flip-flop is similar to that of other JK structures we have seen before, but in this case, the output changes occur on the rising edge of the $Clock$.

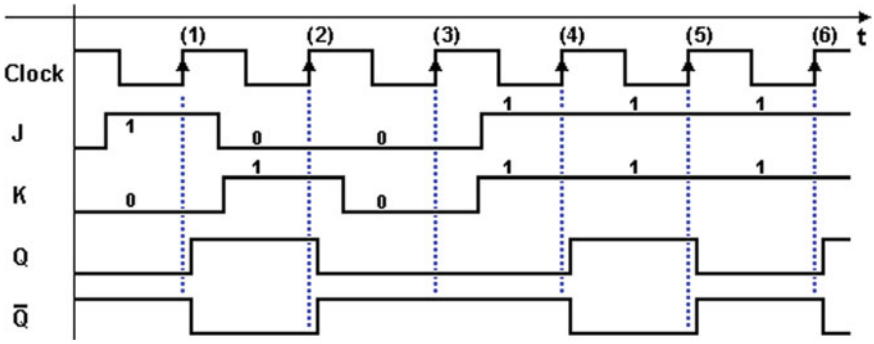
JK-PET Flip-flop					
J	K	$Clock$	Q	$\overline{Q}$	
0	0	↓	Q_p	$\overline{Q_p}$	Previous state
1	0	↓	1	0	SET command
0	1	↓	0	1	$RESET$ command
1	1	↓	$\overline{Q_p}$	Q_p	Toggle

The AND-OR combinational network that generates the value of D is derived by analyzing the function table. This analysis provides us with the table below, which allows us to go directly to the synthesis of the network.

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

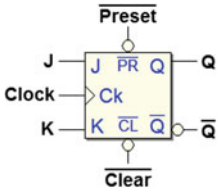
This gives us: $D = (J\,\overline{Q} + \overline{K}\,Q)$

The timing diagram in the figure describes the typical behavior of a JK-PET flip-flop. At the rising edge of the $Clock$, the flip-flop responds to the inputs and then updates the outputs after the network's propagation time. The input activation sequence is the same as that for previous versions of flip-flops.



On edge (1) Q is activated in response to the acquisition of J. On (2), it is deactivated (because $K = 1$), while on (3) its value is maintained ($J = K = 0$). Finally, on the following edges (4, 5, 6) the outputs switch their value (given that $J = K = 1$).

Like the other logical types, the JK-PET flip-flop often includes inputs $\overline{Clear}$ and $\overline{Preset}$ in its commercial or library implementations. Find below its logical symbol and function table.



JK-PET Flip-Flop (with $\overline{Clear}$ and $\overline{Preset}$)							
$\overline{Clear}$	$\overline{Preset}$	J	K	$clock$	Q	$\overline{Q}$	
0	1	–	–	–	0	1	Action of Clear
1	0	–	–	–	1	0	Action of Preset
0	0	–	–	–	1	1	(invalid)
1	1	0	0	↓	Q_p	$\overline{Q_p}$	Previous value
1	1	1	0	↓	1	0	SET command
1	1	0	1	↓	0	1	$RESET$ command
1	1	1	1	↓	$\overline{Q_p}$	Q_p	Outputs reversed

5.6.4 T-PET Flip-Flop

If the two inputs of a JK flip-flop are connected, we get a type T (*Toggle*) flip-flop. The figure below shows the network based on the JK-PET and the corresponding logical symbol.

The function table of the T-PET is derived from the JK-PET table eliminating the rows where J and K are different.

The T-PET Flip-Flop				
T	$Clock$	Q	$\overline{Q}$	
0	↓	Q_p	$\overline{Q_p}$	Previous state
1	↓	$\overline{Q_p}$	Q_p	Toggle

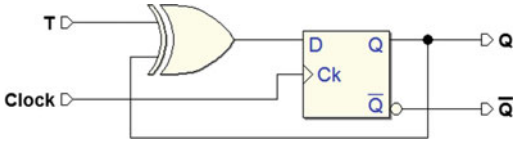
The T flip-flop can also be obtained from the D type through the same procedure as above, deriving the JK from the D. By doing this, we get:

$$D = (J\,\overline{Q} + \overline{K}\,Q)$$

given that $T = J = K$, the expression is reduced to:

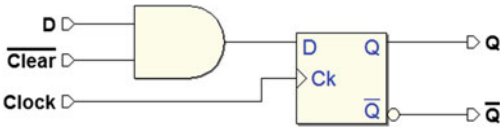
$$D = T\,\overline{Q} + \overline{T}\,Q = T \oplus Q$$

From this, we get the network below. In the following, we will come back to this a few times.



5.6.5 Synchronous Initialization of Flip-Flops

Networks at a certain level of complexity use a *synchronous initialization* structure where the $Clock$ synchronizes the actions of $\overline{Clear}$ and $\overline{Preset}$. The figure below provides an example:



It shows a possible structure of the synchronous $\overline{Clear}$ for a D-PET-type flip-flop. When input $\overline{Clear}$ is activated (at 0), it prevails over input D and forces the output to 0 on the active edge of the $Clock$.

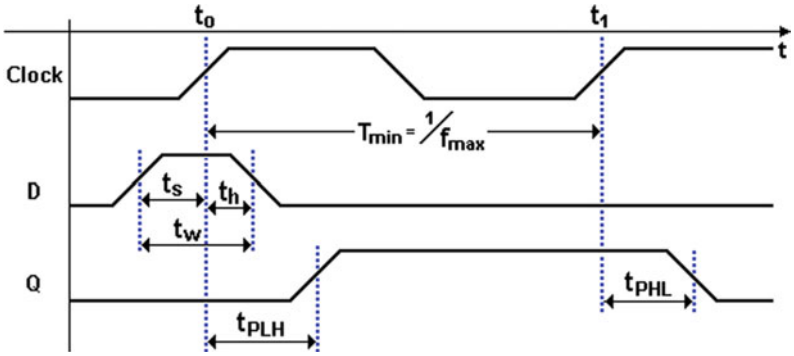
5.7 Timing Parameters of Flip-Flops

Flip-flops, like the combinational components they are made of, are subject to *propagation delays*. As we saw with combinational networks, propagation times are measured by taking 50% of the signal edges as a reference. Propagation times are published on the producer data sheets on statistical bases, as their *minimal*, *typical*, or *maximum* values are declared.

Aside from *propagation times*, there are other timing parameters that are statistically quantified such as *safety margins*, which must be observed to guarantee the flip-flop works as expected. The table below gives a succinct definition of the main timing parameters.

$\mathbf{t_{PLH}}$	*Propagation time* measured from the activation of the $Clock$ to the output's transition from low to high (L-H)
$\mathbf{t_{PHL}}$	*Propagation time* measured from the activation of the $Clock$ to the output's transition from high to low (L-H)
$\mathbf{t_s}$	*Setup Time*: the time interval the value of a synchronous input must remain stable *before* the active edge of the $Clock$
$\mathbf{t_h}$	*Hold time*: the interval when the value of a synchronous input must remain stable *after* the active edge of the $Clock$
$\mathbf{t_w}$	*Minimum width* of an input signal
$\mathbf{T_{min}}$	Minimum $Clock$ period
$\mathbf{F_{max}}$	Maximum $Clock$ frequency

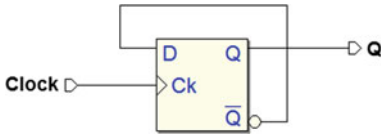
The timing diagram in the figure below shows the timing parameters of a D-PET flip-flop in the acquisition sequence of a 1 followed by a 0. The **tw** in the figure refers to input D and in this example, it is given by the sum of the **ts** and **th** times.



If the *safety margins* are not observed, unpredictable behavior can result. This is called *metastability*, and it will be dealt with in the next section.

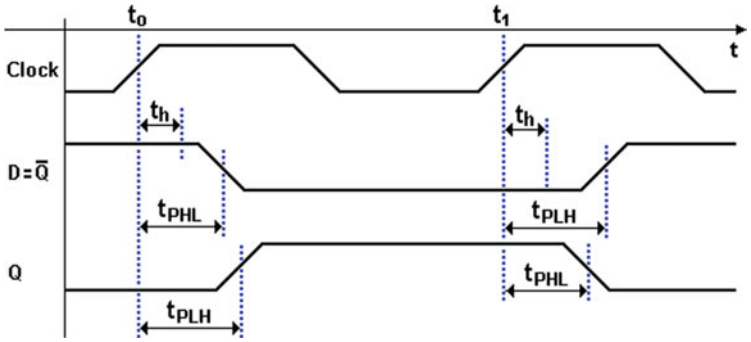
5.7.1 Relationship Between Propagation and Hold Times

Let's use the example from the figure below to study the relationship between propagation times and hold times.



The circuit is composed of a D-PET flip-flop whose input D is connected to its negated output $\overline{Q}$. At each active edge of the $Clock$, it will toggle its value, as it would in a T-PET-type flip-flop with an input of $T = 1$.

If the value of input D is to be acquired correctly, it must remain stable for at least a time **th** after the rising edge of $Clock$.

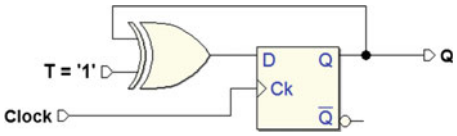


The situation is exemplified in the timing diagram above where we see that **th** must be shorter than both the two propagation times ($\mathbf{t}_{\mathbf{PHL}}$ and $\mathbf{t}_{\mathbf{PLH}}$). This condition is at the base of all sequential networks and all the flip-flops are designed to satisfy it.

5.7.2 *Maximum Clock Frequency of a Network with Flip-Flops*

A flip-flop's timing parameters enter into play when evaluating the *maximum clock frequency* of the network that uses it. We have previously seen the implementation of the T-PET flip-flop based on the D-PET. In the network in the figure below, we continually force T to 1, imposing a configuration in which the output *changes value* at every rising edge of the $Clock$.

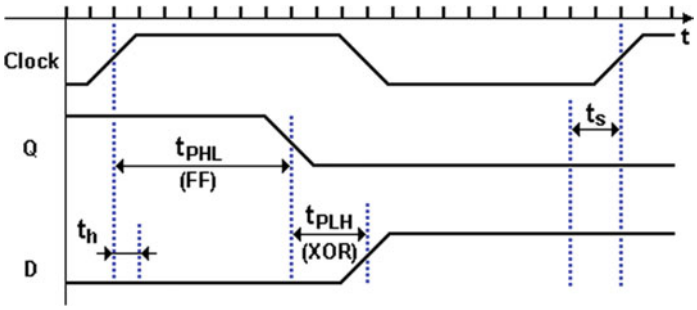
The purpose of this example is to take into account the propagation delay of a combinational network along the feedback path.



Input D is driven by output Q through the XOR gate which works like a NOT gate since it has an input at 1. For the network to function correctly, the signal at D must be stable at least from time **ts**, at the moment when the $Clock$ sees the rising edge until time **th** after the transition.

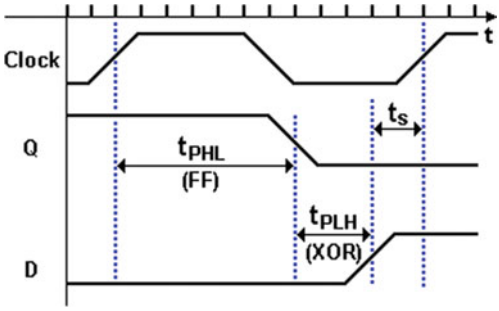
Notice that the time scale in the following three timing diagrams corresponds to **1 nS** (between two notches on the time axis). The values: $\mathbf{ts = 2\ nS}$, $\mathbf{th = 1\ nS}$, $\mathbf{t_{PHL}(FF) = 7\ nS}$, $\mathbf{t_{PLH}(XOR) = 3\ nS}$ have been taken.

What the three situations have in common is that signal D is stable after time $\mathbf{t_{PHL}(FF) + t_{PLH}(XOR)}$ regardless of the $Clock$ period. In the first diagram below, the sequence is characterized by $Clock$ with period $\mathbf{T = 20\ nS}$ (corresponding to a frequency of **50 MHz**):



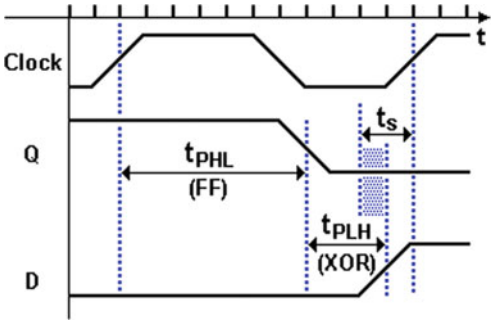
The next rising edge samples data **D**, which was stable for a time longer than **ts**, so the network functions correctly. Also, the condition of **th** is maintained since **th** is shorter than the propagation time of the flip-flop (in the next two figures, the representation of **th** is omitted).

In the figure below, we see the limit case where D is stable for exactly a time **ts** before the edge with a $Clock$ period of **12 nS**. The network still functions correctly.



The period **12 nS** corresponds to a frequency of about **83 MHz**, the *maximum clock frequency* of the network.

In the figure below, however, the *Clock* period is **11 nS** (corresponding to about **91 MHz**). The following *Clock* edge occurs when *D* is already stable but the **ts** is not observed. In this case, there is no guarantee the network will function; we are over the network's *maximum working frequency*.



5.8 Flip-Flops: Graphic Symbols and Tables

At the beginning of this chapter, we described flip-flops by their *logical type* and their *command type*. This section summarizes the characteristics and graphic symbols of the most commonly used flip-flops.

5.8.1 Logical Types

The following *function tables* recap the behavior of the *logical types* of flip-flop that were introduced in this chapter (SR, JK, D, E, and T). They do not describe the command structure but only the *logical inputs*, assuming they are *active-high*.

SR			
SET	$RESET$	Q	
0	0	Q_p	Previous state
1	0	1	SET Command
0	1	0	$RESET$ Command
1	1	1	Invalid

JK			
J	K	Q	
0	0	Q_p	Previous state
1	0	1	SET Command
0	1	0	$RESET$ Command
1	1	$\overline{Q_p}$	Toggle

D		
D	Q	
0	0	Memorizes 0
1	1	Memorizes 1

E			
E	D	Q	
0	–	Q_p	Previous value
1	0	0	Memorizes 0
1	1	1	Memorizes 1

T		
T	Q	
0	Q_p	Previous state
1	$\overline{Q_p}$	Toggle

5.8.2 Command Types

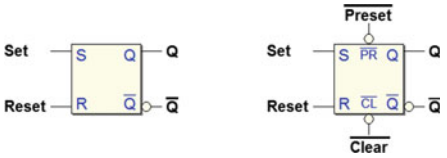
This summary includes only the most commonly used flip-flops. Let's recap the command configurations that flip-flops use to acquire the inputs and update the outputs.

- **Direct command**:
  A command is *direct* when the action of the *logical inputs* does not depend on any other input. They directly control the flip-flop's behavior, which is *asynchronous* in this case.
- **Level-enabled commands**:
  A command is *level-enabled* when there is an added input that enables/disables the inputs according to its logical level. Another term to identify this type is *Latch commands*.

- **Edge-triggered command**:
  We have an *edge-triggered* command when there is an added synchronization input, usually called $Clock$, which enables the inputs to function in response to their rising or falling edges. These flip-flops are *synchronous.*

Direct Commands

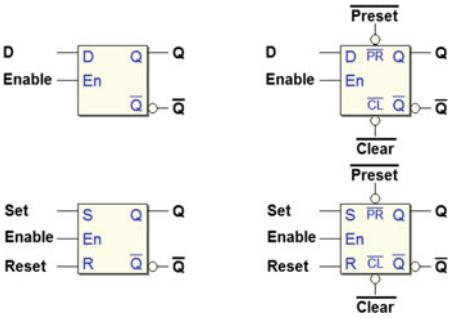
The direct command structure is currently used only with the *Set-Reset* logical type. The figure below left shows the symbol without initialization inputs. The one on the right shows a symbol that includes $\overline{Clear}$ and $\overline{Preset}$.



$\overline{Clear}$ and $\overline{Preset}$ might seem superfluous since they do the same thing as the logical inputs, but it is often useful in a project to separate the initialization network from the rest.

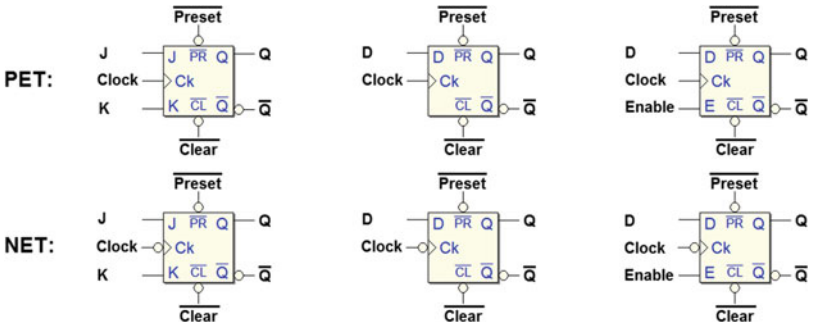
Level-Enabled Commands

Level-enabled command structures are mainly used in D-Latch type-flip-flops and often in large structures that employ hundreds of thousands of them, as the *read/write memories* (not covered in this book). The level-enabled Set-Reset is used as internal block of more complex flip-flops. The considerations about initialization inputs are valid here as well.



Edge-Triggered Commands

PET or NET *edge-triggered command* structures are the most commonly used especially in the D and E logical-type flip-flops. The use of the E type is spreading in complex networks while use of the JK type is in decline. The figure below shows both PET and NET flip-flop symbols. They generally have the initialization inputs $\overline{Clear}$ and $\overline{Preset}$.



5.8.3 *Excitation Tables*

Excitation tables give us another way to describe the behavior of a flip-flop. Function tables, which we have seen before, indicate the assumed value of an output in relation to the inputs. Excitation tables provide values to assign to the logical inputs of a flip-flop in relation to the desired *output transition.*

Below left is the function table of the *Set-Reset* flip-flop. On the right is the corresponding excitation table where each of the four possible output transitions is shown next to the input configuration needed to obtain it.

Set-Reset Function Table			
Set	*Reset*	Q	
0	0	Q_p	Previous state
1	0	1	SET command
0	1	0	RESET command
1	1	1	Invalid

Set-Reset Excitation table		
$Q_p \rightarrow Q$	*Set*	*Reset*
$0 \rightarrow 0$	0	–
$0 \rightarrow 1$	1	0
$1 \rightarrow 0$	0	1
$1 \rightarrow 1$	–	0

The excitation table is easily derived by the function table. For example, the output transition of Q from 0 to 0 (written as: $0 \rightarrow 0$) can be obtained by keeping the previous state of the flip-flop ($Set = 0$, $Reset = 0$, row 1 of the function table), or with a Reset command ($Set = 0$, $Reset = 1$, row 3).

This is translated in the express terms by the top row of the excitation table. *Set* must be at 0, while the value of *Reset* is *don't-care*. The same holds for the transition $1 \rightarrow 1$, while the other two transitions have no *don't-cares*.

The function and excitation tables of the JK flip-flop are shown below.

JK Function table			
J	K	Q	
0	0	Q_p	Previous state
1	0	1	SET command
0	1	0	RESET command
1	1	$\overline{Q_p}$	Toggle

JK Excitation table		
$Q_p \rightarrow Q$	J	K
$0 \rightarrow 0$	0	–
$0 \rightarrow 1$	1	–
$1 \rightarrow 0$	–	1
$1 \rightarrow 1$	–	0

This excitation table shows a *don't-care* for each of the four transitions. For example, the transition $0 \rightarrow 1$ is obtained through a Set command ($J = 1$, $K = 0$), or a *toggle* command ($J = 1$, $K = 1$). In short, about the transition $0 \rightarrow 1$ the value of K is *don't-care* while J must be at 1.

To be thorough, we have put the tables for the D flip-flop below even though the excitation table is immediately derived from the function table. The table for the E flip-flop, when it is enabled by input E, is identical.

D Function table		
D	Q	
0	0	Memorizes 0
1	1	Memorizes 1

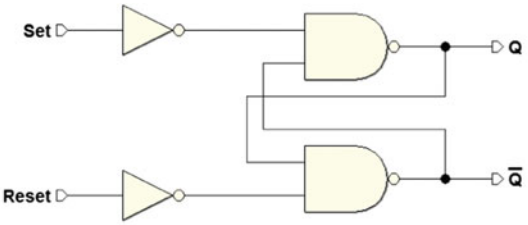
D Excitation table	
$Q_p \rightarrow Q$	D
$0 \rightarrow 0$	0
$0 \rightarrow 1$	1
$1 \rightarrow 0$	0
$1 \rightarrow 1$	1

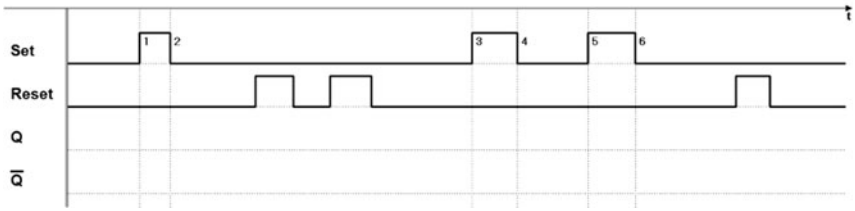
5.9 Exercises

For each network below, complete the timing diagram (all the diagrams to be completed are also available in *PDF* in the *digital contents* of the book, on the *Deeds* Web site).

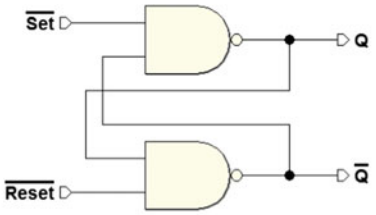
We suggest to draw first the diagrams without using the simulator and then check your answers with it (the network files are on the Web site, too).

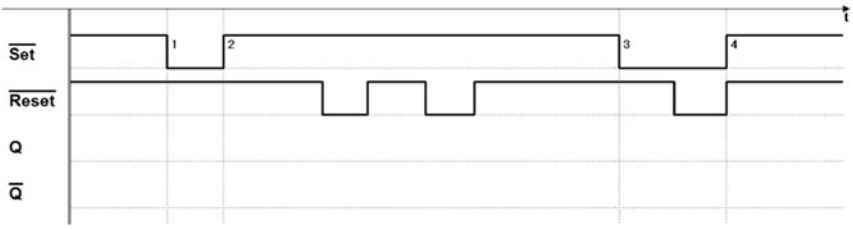
1. Set-Reset flip-flop (*direct command*)



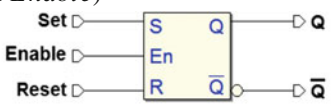


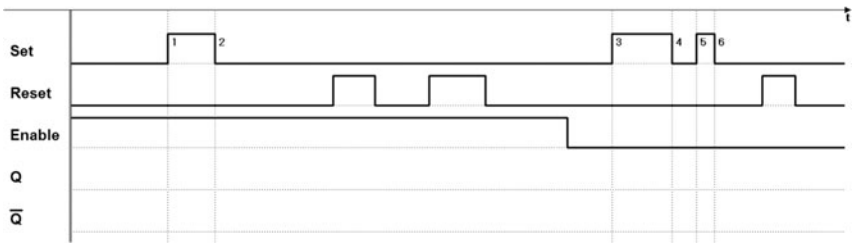
2. Set-Reset flip-flop (*direct command, NAND base cell*)



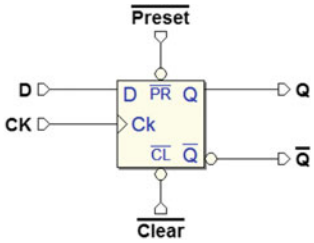


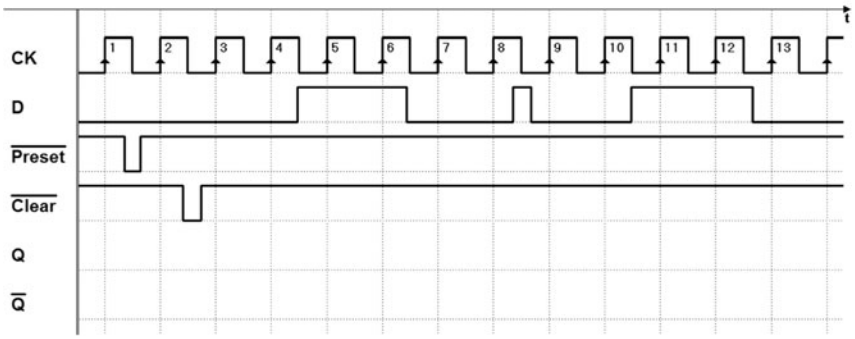
3. Set-Reset flip-flop (with *Enable*)



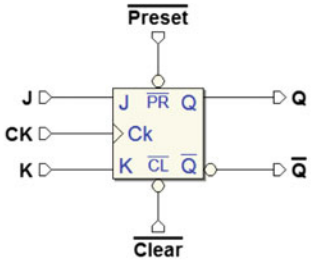


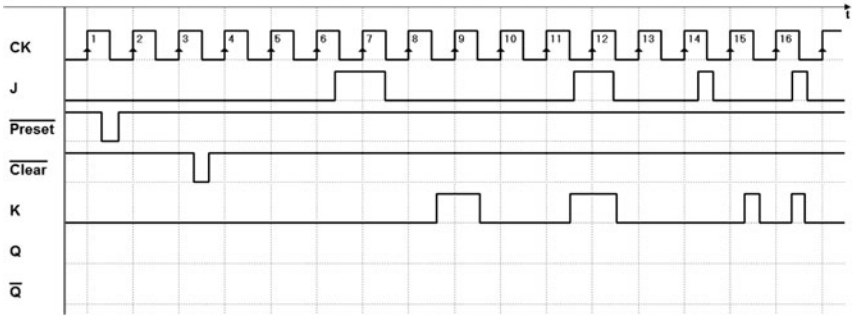
4. D-PET flip-flop (with $\overline{Preset}$ and $\overline{Clear}$)



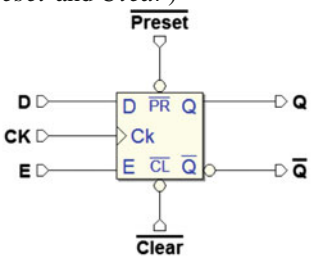


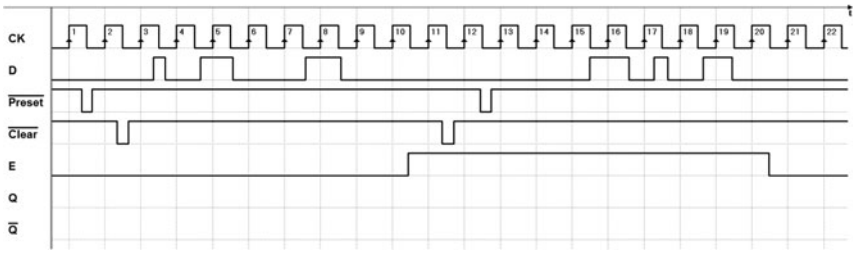
5. JK-PET flip-flop (with $\overline{Preset}$ and $\overline{Clear}$)





6. E-PET flip-flop (with $\overline{Preset}$ and $\overline{Clear}$)

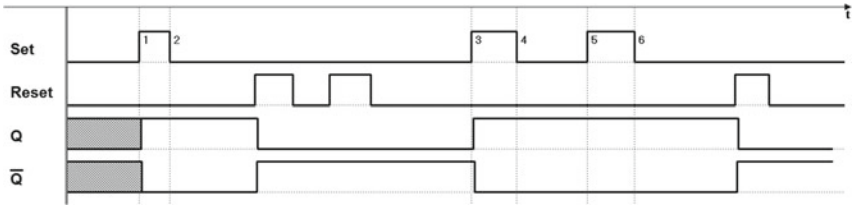




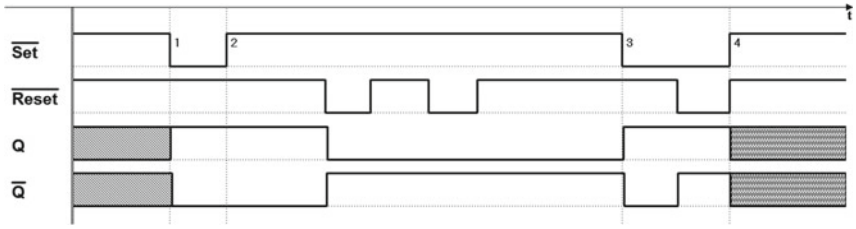
5.10 Solutions

The timing diagrams below were obtained using the *Deeds* timing simulation. The files of all the networks are available on the *digital contents* of the book so that the answers can be checked through the simulator.

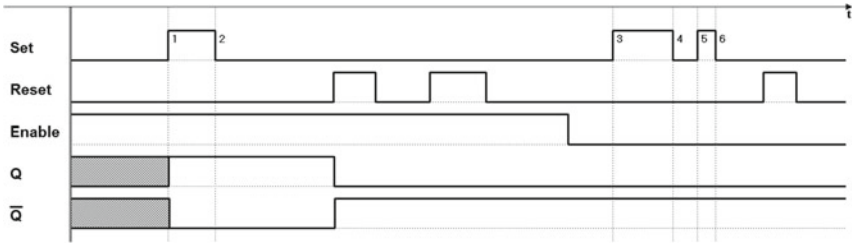
1. Set-Reset flip-flop (*direct command*)



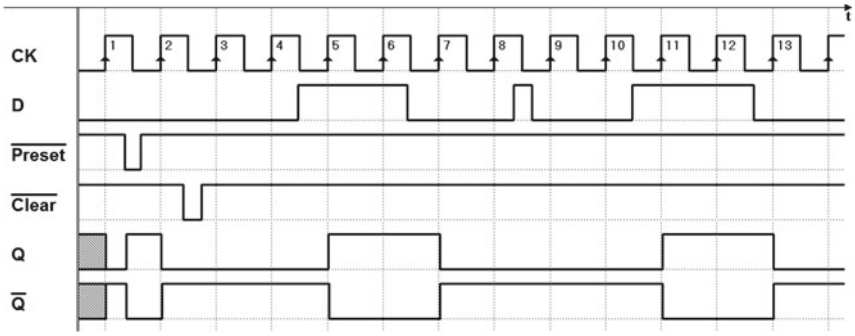
2. Set-Reset flip-flop (*direct command, NAND base cell*)



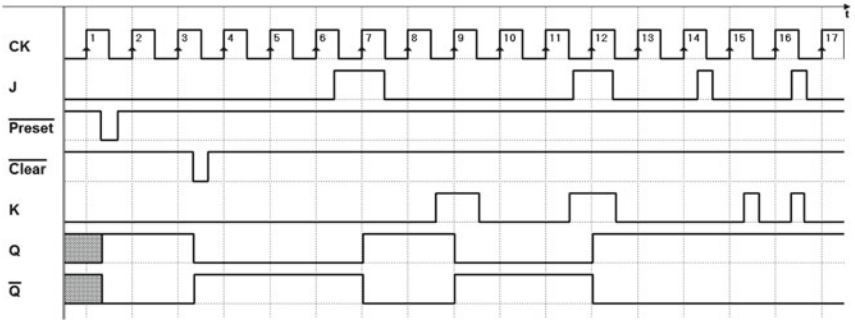
3. Set-Reset flip-flop (with *Enable*)



4. D-PET flip-flop (with $\overline{Preset}$ and $\overline{Clear}$)



5. JK-PET flip-flop (with $\overline{Preset}$ and $\overline{Clear}$)



6. E-PET flip-flop (with $\overline{Preset}$ and $\overline{Clear}$)