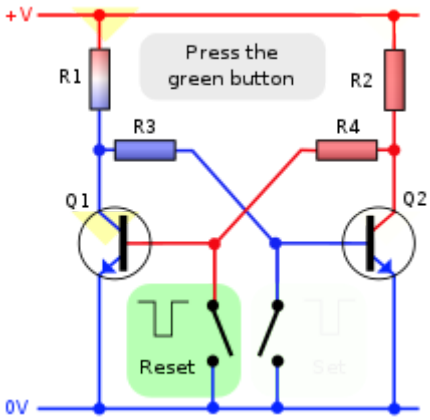
Flip-flop (electronics)

In electronics, a **flip-flop** or **latch** is a circuit that has two stable states and can be used to store state information – a bistable multivibrator. The circuit can be made to change state by signals applied to one or more control inputs and will have one or two outputs. It is the basic storage element in sequential logic. Flip-flops and latches are fundamental building blocks of digital electronics systems used in computers, communications, and many other types of systems.



An animated interactive SR latch (*R1, R2* = 1 kΩ; *R3, R4* = 10 kΩ).

Flip-flops and latches are used as data storage elements. A flip-flop is a device which stores a single *bit* (binary digit) of data; one of its two states represents a "one" and the other represents a "zero". Such data storage can be used for storage of *state*, and such a circuit is described as sequential logic in electronics. When used in a finite-state machine, the output and next state depend not only on its current input, but also on its current state (and hence, previous inputs). It can also be used for counting of pulses, and for synchronizing variably-timed input signals to some reference timing signal.

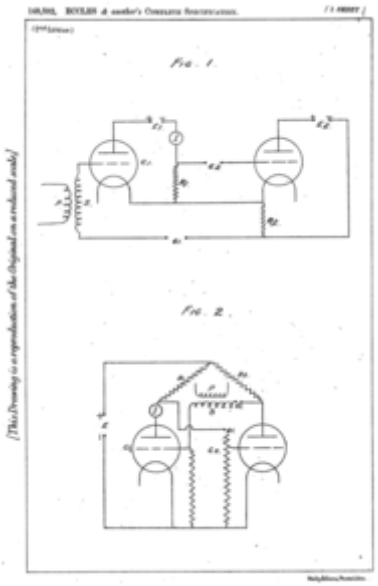
Flip-flops can be either level-triggered (asynchronous, transparent or opaque) or edge-triggered (synchronous, or clocked). The term flip-flop has historically referred generically to both level-triggered and edge-triggered circuits that store a single bit of data using gates. Recently, some authors reserve the term *flip-flop* exclusively for discussing clocked circuits; the simple ones are commonly called *transparent latches*.[1][2] Using this terminology, a level-sensitive flip-flop is called a transparent latch, whereas an edge-triggered flip-flop is simply called a flip-flop. Using either terminology, the term "flip-flop" refers to a device that stores a single bit of data, but the term "latch" may also refer to a device that stores any number of bits of data using a single trigger. The terms "edge-triggered", and "level-triggered" may be used to avoid ambiguity.[3]

When a level-triggered latch is enabled it becomes transparent, but an edge-triggered flip-flop's output only changes on a single type (positive going or negative going) of clock edge.

Contents

History

The first electronic flip-flop was invented in 1918 by the British physicists William Eccles and F. W. Jordan.[4][5] It was initially called the *Eccles–Jordan trigger circuit* and consisted of two active elements (vacuum tubes).[6] The design was used in the 1943 British Colossus codebreaking computer[7] and such circuits and their transistorized versions were common in computers even after the introduction of integrated circuits, though flip-flops made from logic gates are also common now.[8][9] Early flip-flops were known variously as trigger circuits or multivibrators.

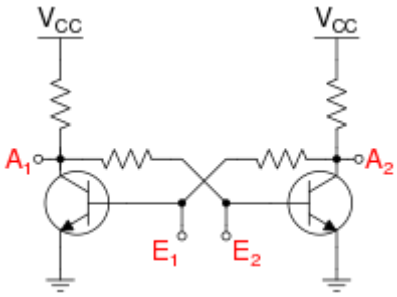


Flip-flop schematics from the Eccles and Jordan patent filed 1918, one drawn as a cascade of amplifiers with a positive feedback path, and the other as a symmetric cross-coupled pair

According to P. L. Lindley, an engineer at the US Jet Propulsion Laboratory, the flip-flop types detailed below (SR, D, T, JK) were first discussed in a 1954 UCLA course on computer design by Montgomery Phister, and then appeared in his book *Logical Design of Digital Computers*.[10][11] Lindley was at the time working at Hughes Aircraft under Eldred Nelson, who had coined the term JK for a flip-flop which changed states when both inputs were on (a logical "one"). The other names were coined by Phister. They differ slightly from some of the definitions given below. Lindley explains that he heard the story of the JK flip-flop from Eldred Nelson, who is responsible for coining the term while working at Hughes Aircraft. Flip-flops in use at Hughes at the time were all of the type that came to be known as J-K. In designing a logical system, Nelson assigned letters to flip-flop inputs as follows: #1: A & B, #2: C & D, #3: E & F, #4: G & H, #5: J & K. Nelson used the notations "*j*-input" and "*k*-input" in a patent application filed in 1953.[12]

Implementation

Flip-flops can be either simple (transparent or asynchronous) or clocked (synchronous). In the context of hardware description languages, the simple ones are commonly described as *latches*,[1] while the clocked ones are described as *flip-flops*.[2]



A traditional (simple) flip-flop circuit based on bipolar junction transistors

Simple flip-flops can be built around a single pair of cross-coupled inverting elements: vacuum tubes, bipolar transistors, field effect transistors, inverters, and inverting logic gates have all been used in practical circuits.

Clocked devices are specially designed for synchronous systems; such devices ignore their inputs except at the transition of a dedicated clock signal (known as clocking, pulsing, or strobing). Clocking causes the flip-flop either to change or to retain its output signal based upon the values of the input signals at the transition. Some flip-flops change output on the rising edge of the clock, others on the falling edge.

Since the elementary amplifying stages are inverting, two stages can be connected in succession (as a cascade) to form the needed non-inverting amplifier. In this configuration, each amplifier may be considered as an active inverting feedback network for the other inverting amplifier. Thus the two stages are connected in a non-inverting loop although the circuit diagram is usually drawn as a symmetric cross-coupled pair (both the drawings are initially introduced in the Eccles–Jordan patent).

Flip-flop types

Flip-flops can be divided into common types: the **SR** ("set-reset"), **D** ("data" or "delay"[13]), **T** ("toggle"), and **JK**. The behavior of a particular type can be described by what is termed the characteristic equation, which derives the "next" (i.e., after the next clock pulse) output, Q_{next} in terms of the input signal(s) and/or the current output, Q.

Simple set-reset latches

When using static gates as building blocks, the most fundamental latch is the simple *SR latch*, where S and R stand for *set* and *reset*. It can be constructed from a pair of cross-coupled NOR or NAND logic gates. The stored bit is present on the output marked Q.

SR NOR latch

While the R and S inputs are both low, feedback maintains the Q and $\overline{\text{Q}}$ outputs in a constant state, with $\overline{\text{Q}}$ the complement of Q. If S (*Set*) is pulsed high while R (*Reset*) is held low, then the Q output is forced high, and stays high when S returns to low; similarly, if R is pulsed high while S is held low, then the Q output is forced low, and stays low when R returns to low.

SR latch operation[3]

Characteristic table				Excitation table			
S	R	Q_{next}	Action	Q	Q_{next}	S	R
0	0	Q	Hold state	0	0	0	X
0	1	0	Reset	0	1	1	0
1	0	1	Set	1	0	0	1
1	1	X	Not allowed	1	1	X	0

Note: X means don't care*, that is, either 0 or 1 is a valid value.*

The R = S = 1 combination is called a **restricted combination** or a **forbidden state** because, as both NOR gates then output zeros, it breaks the logical equation Q = **not** Q. The combination is also inappropriate in circuits where *both* inputs may go low *simultaneously* (i.e. a transition from *restricted* to *keep*). The output would lock at either 1 or 0 depending on the propagation time relations between the gates (a race condition).

To overcome the restricted combination, one can add gates to the inputs that would convert `(S, R) = (1, 1)` to one of the non-restricted combinations. That can be:

- Q = 1 (1, 0) – referred to as an *S (dominated)-latch*
- Q = 0 (0, 1) – referred to as an *R (dominated)-latch*

This is done in nearly every programmable logic controller.

- Keep state (0, 0) – referred to as an *E-latch*

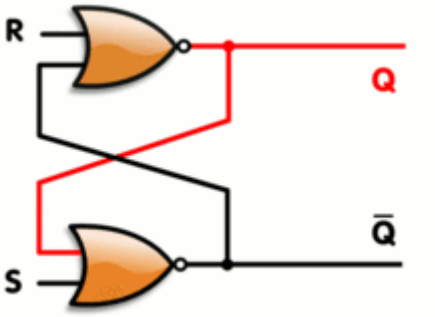
Alternatively, the restricted combination can be made to *toggle* the output. The result is the JK latch.

The characteristic equation for the SR latch is :

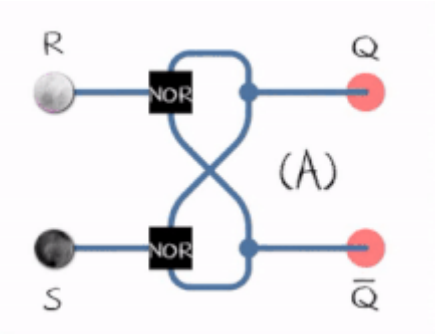
$$Q_{\text{next}} = \bar{R}Q + \bar{R}S \text{ or } Q_{\text{next}} = \bar{R}(Q + S).$$[14]

Another expression is :

$$Q_{\text{next}} = S + \bar{R}Q \text{ with } SR = 0$$[15]



An animation of a SR latch, constructed from a pair of cross-coupled NOR gates. Red and black mean logical '1' and '0', respectively.



An animated SR latch. Black and white mean logical '1' and '0', respectively.

A. S = 1, R = 0: Set

B. S = 0, R = 0: Hold

C. S = 0, R = 1: Reset

D. S = 1, R = 1: Not allowed

Transitioning from the restricted combination (D) to (A) leads to an unstable state.

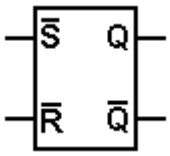
$\overline{\text{SR}}$ NAND latch

The circuit shown below is a basic NAND latch. The inputs are generally designated S and R for Set and Reset respectively. Because the NAND inputs must normally be logic 1 to avoid affecting the latching action, the inputs are considered to be inverted in this circuit (or active low).

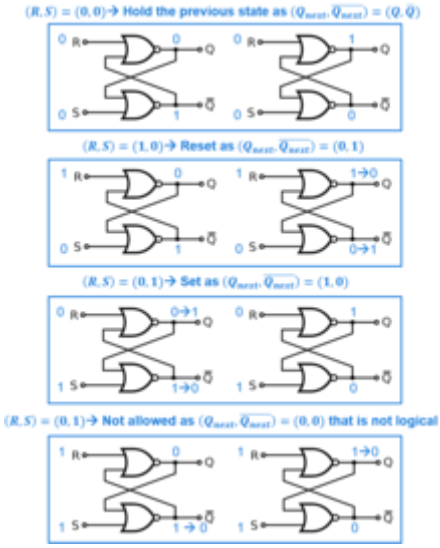
The circuit uses feedback to "remember" and retain its logical state even after the controlling input signals have changed. When the S and R inputs are both high, feedback maintains the Q outputs to the previous state.

$\overline{SR}$ latch operation

$\overline{S}$	$\overline{R}$	Action
0	0	Q = 1, $\overline{Q}$ = 1; not allowed
0	1	Q = 1
1	0	Q = 0
1	1	No change; random initial



Symbol for an $\overline{SR}$ NAND latch



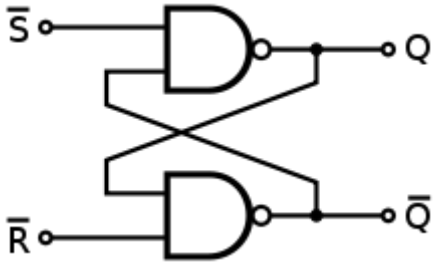
How an SR NOR latch works.

SR AND-OR latch

From a teaching point of view, SR latches drawn as a pair of cross-coupled components (transistors, gates, tubes, etc.) are often hard to understand for beginners. A didactically easier to understand way is to draw the latch as a single feedback loop instead of the cross-coupling. The following is an SR latch built with an AND gate with one inverted input and an OR gate. Note that the inverter is not needed for the latch functionality, but rather to make both inputs High-active.

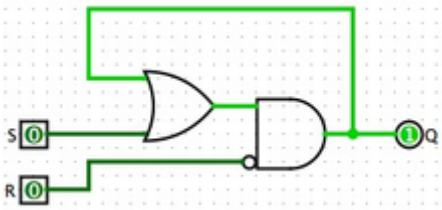
SR AND-OR latch operation

S	R	Action
0	0	No change; random initial
1	0	Q = 1
X	1	Q = 0



An $\overline{SR}$ latch constructed from cross-coupled NAND gates.

Note that the SR AND-OR latch has the benefit that S = 1, R = 1 is well defined. In above version of the SR AND-OR latch it gives priority to the R signal over the S signal. If priority of S over R is needed, this can be achieved by connecting output Q to the output of the OR gate instead of the output of the AND gate.



An SR AND-OR latch. Light green means logical '1' and dark green means logical '0'. The latch is currently in hold mode (no change).

The SR AND-OR latch is easier to understand, because both gates can be explained in isolation. When neither S or R is set, then both the OR gate and the AND gate are in "hold mode", i.e., their output is the input from the feedback loop. When input S = 1, then the output of the OR gate becomes 1, regardless of the other input from the feedback loop ("set mode"). When input R = 1 then the output of the AND gate becomes 0, regardless of the other input from the feedback loop ("reset mode"). And since the output Q is directly connected to the output of the AND gate, R has priority over S. Latches drawn as cross-coupled gates may look less intuitive, as the behaviour of one gate appears to be intertwined with the other gate.

Note that the SR AND-OR latch can be transformed into the SR NOR latch using logic transformations: inverting the output of the OR gate and also the 2nd input of the AND gate and connecting the inverted Q output between these two added inverters; with the AND gate with both inputs inverted being equivalent to a NOR gate according to De Morgan's laws.

JK latch

The JK latch is much less frequently used than the JK flip-flop. The JK latch follows the following state table:

JK latch truth table

J	K	Q_{next}	Comment
0	0	Q	No change
0	1	0	Reset
1	0	1	Set
1	1	$\overline{Q}$	Toggle

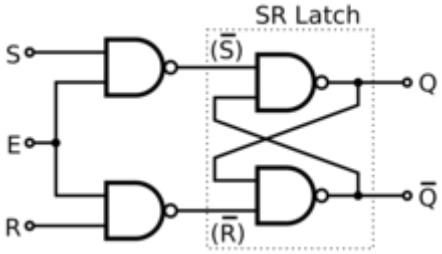
Hence, the JK latch is an SR latch that is made to *toggle* its output (oscillate between 0 and 1) when passed the input combination of 11.[16] Unlike the JK flip-flop, the 11 input combination for the JK latch is not very useful because there is no clock that directs toggling.[17]

Gated latches and conditional transparency

Latches are designed to be *transparent.* That is, input signal changes cause immediate changes in output. Additional logic can be added to a simple transparent latch to make it *non-transparent* or *opaque* when another input (an "enable" input) is not asserted. When several *transparent* latches follow each other, using the same enable signal, signals can propagate through all of them at once. However, by following a *transparent-high* latch with a *transparent-low* (or *opaque-high*) latch, a master–slave flip-flop is implemented.

Gated SR latch

A *synchronous SR latch* (sometimes *clocked SR flip-flop*) can be made by adding a second level of NAND gates to the **inverted** SR latch (or a second level of AND gates to the **direct** SR latch). The extra NAND gates further invert the inputs so $\overline{SR}$ latch becomes a gated SR latch (and a SR latch would transform into a gated $\overline{SR}$ latch with inverted enable).

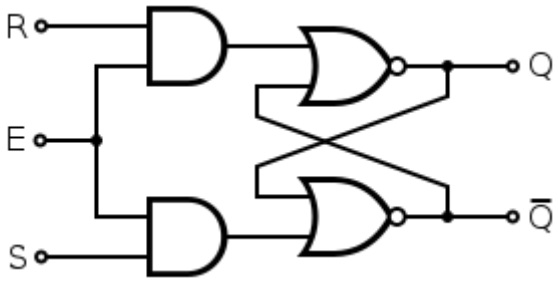


NAND Gated SR Latch (Clocked SR flip-flop). Note the inverted inputs.

With E high (*enable* true), the signals can pass through the input gates to the encapsulated latch; all signal combinations except for (0, 0) = *hold* then immediately reproduce on the (Q, $\overline{Q}$) output, i.e. the latch is *transparent.*

With E low (*enable* false) the latch is *closed (opaque)* and remains in the state it was left the last time E was high.

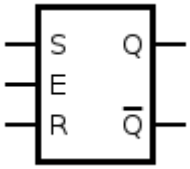
The *enable* input is sometimes a clock signal, but more often a read or write strobe. When the *enable* input is a clock signal, the latch is said to be **level-sensitive** (to the level of the clock signal), as opposed to **edge-sensitive** like flip-flops below.



A gated SR latch circuit diagram constructed from AND gates (on left) and NOR gates (on right).

Gated SR latch operation

E/C	Action
0	No action (keep state)
1	The same as non-clocked SR latch



Symbol for a gated SR latch

Gated D latch

This latch exploits the fact that, in the two active input combinations (01 and 10) of a gated SR latch, R is the complement of S. The input NAND stage converts the two D input states (0 and 1) to these two input combinations for the next $\overline{SR}$ latch by inverting the data input signal. The low state of the *enable* signal produces the inactive "11" combination. Thus a gated D-latch may be considered as a *one-input synchronous SR latch*. This configuration prevents application of the restricted input combination. It is also known as *transparent latch*, *data latch*, or simply *gated latch*. It has a *data* input and an *enable* signal (sometimes named *clock*, or *control*). The word *transparent* comes from the fact that, when the enable input is on, the signal propagates directly through the circuit, from the input D to the output Q. Gated D-latches are also **level-sensitive** with respect to the level of the clock or enable signal.

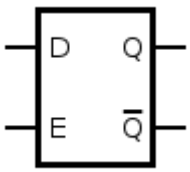
Transparent latches are typically used as I/O ports or in asynchronous systems, or in synchronous two-phase systems (synchronous systems that use a two-phase clock), where two latches operating on different clock phases prevent data transparency as in a master–slave flip-flop.

Latches are available as integrated circuits, usually with multiple latches per chip. For example, 74HC75 is a quadruple transparent latch in the 7400 series.

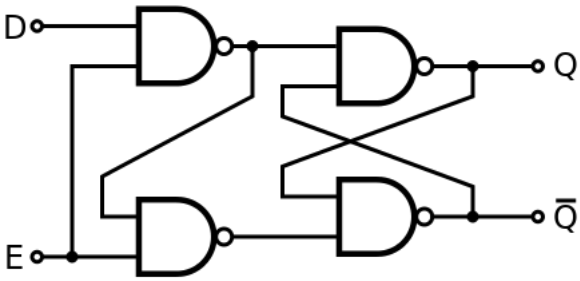
The truth table below shows that when the *e*nable/*c*lock input is 0, the D input has no effect on the output. When E/C is high, the output equals D.

Gated D latch truth table

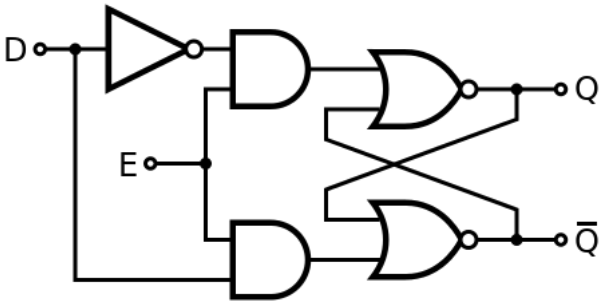
E/C	D		Q	$\overline{Q}$	Comment
0	X		Q_{prev}	$\overline{Q}_{prev}$	No change
1	0		0	1	Reset
1	1		1	0	Set



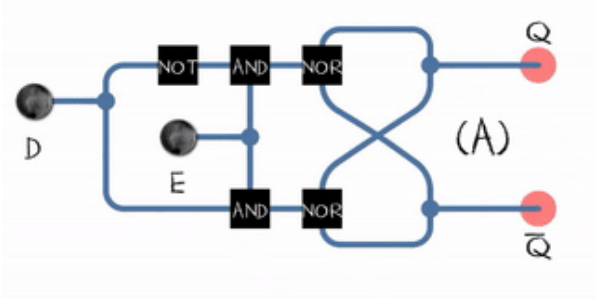
Symbol for a gated D latch



A gated D latch based on an $\overline{SR}$ NAND latch



A gated D latch based on an SR NOR latch



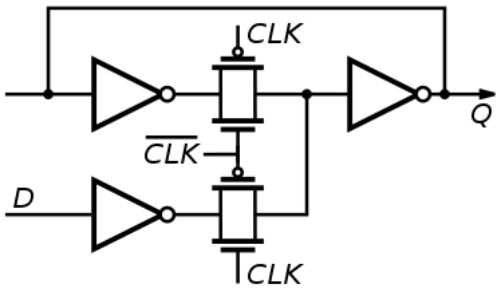
An animated gated D latch. Black and white mean logical '1' and '0', respectively.

A. D = 1, E = 1: set

B. D = 1, E = 0: hold

C. D = 0, E = 0: hold

D. D = 0, E = 1: reset



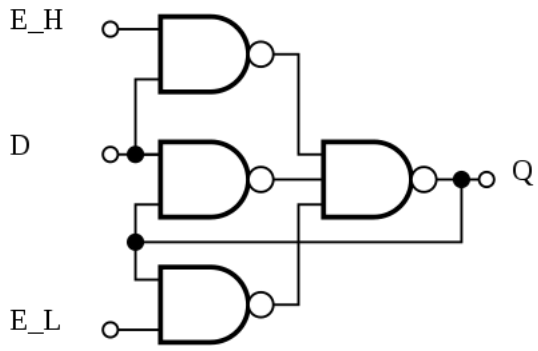
A gated D latch in pass transistor logic, similar to the ones in the CD4042 or the CD74HC75 integrated circuits.

Earle latch

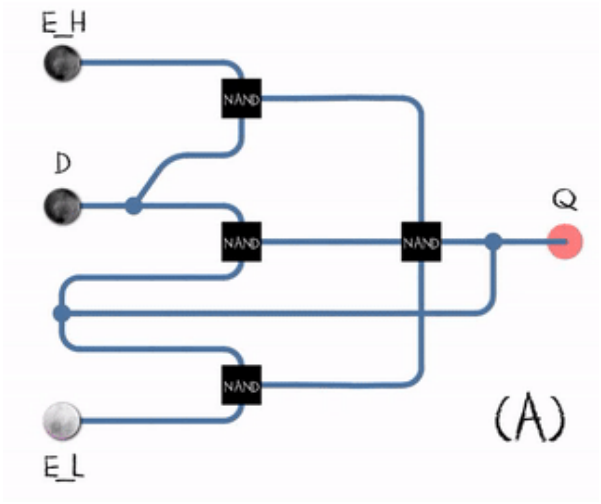
The classic gated latch designs have some undesirable characteristics.[18] They require double-rail logic or an inverter. The input-to-output propagation may take up to three gate delays. The input-to-output propagation is not constant – some outputs take two gate delays while others take three.

Designers looked for alternatives.[19] A successful alternative is the Earle latch. It requires only a single data input, and its output takes a constant two gate delays. In addition, the two gate levels of the Earle latch can, in some cases, be merged with the last two gate levels of the circuits driving the latch because many common computational circuits have an OR layer followed by an AND layer as their last two levels. Merging the latch function can implement the latch with no additional gate delays.[18] The merge is commonly exploited in the design of pipelined computers, and, in fact, was originally developed by John G. Earle to be used in the IBM System/360 Model 91 for that purpose.[20]

The Earle latch is hazard free.[21] If the middle NAND gate is omitted, then one gets the **polarity hold latch**, which is commonly used because it demands less logic.[21][22] However, it is susceptible to logic hazard. Intentionally skewing the clock signal can avoid the hazard.[22]



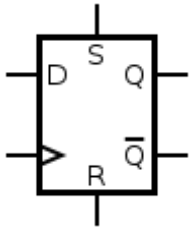
Earle latch uses complementary enable inputs: enable active low (E_L) and enable active high (E_H)



An animated Earle latch. Black and white mean logical '1' and '0', respectively.
A. D = 1, E_H = 1: set
B. D = 0, E_H = 1: reset
C. D = 1, E_H = 0: hold

D flip-flop

The D flip-flop is widely used. It is also known as a "data" or "delay" flip-flop.



D flip-flop symbol

The D flip-flop captures the value of the D-input at a definite portion of the clock cycle (such as the rising edge of the clock). That captured value becomes the Q output. At other times, the output Q does not change.[23][24] The D flip-flop can be viewed as a memory cell, a zero-order hold, or a delay line.[25]

Truth table:

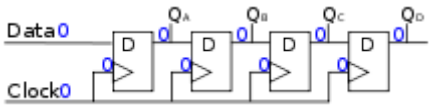
Clock	D	Q_{next}
Rising edge	0	0
Rising edge	1	1
Non-rising	X	Q

(*X* denotes a *don't care* condition, meaning the signal is irrelevant)

Most D-type flip-flops in ICs have the capability to be forced to the set or reset state (which ignores the D and clock inputs), much like an SR flip-flop. Usually, the illegal S = R = 1 condition is resolved in D-type flip-flops. Setting S = R = 0 makes the flip-flop behave as described above. Here is the truth table for the other possible S and R configurations:

Inputs				Outputs	
S	**R**	**D**	**>**	**Q**	**$\overline{Q}$**
0	1	X	X	0	1
1	0	X	X	1	0
1	1	X	X	1	1

These flip-flops are very useful, as they form the basis for shift registers, which are an essential part of many electronic devices. The advantage of the D flip-flop over the D-type "transparent latch" is that the signal on the D input pin is captured the moment the flip-flop is clocked, and subsequent changes on the D input will be ignored until the next clock event. An exception is that some flip-flops have a "reset" signal input, which will reset Q (to zero), and may be either asynchronous or synchronous with the clock.



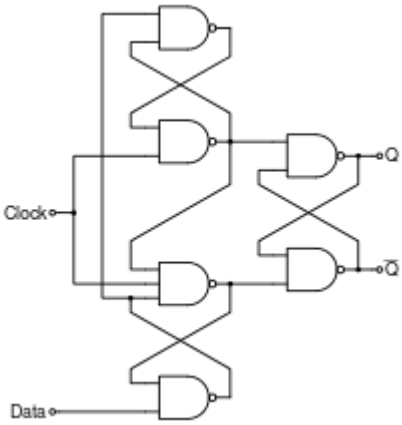
4-bit serial-in, parallel-out (SIPO) shift register

The above circuit shifts the contents of the register to the right, one bit position on each active transition of the clock. The input X is shifted into the leftmost bit position.

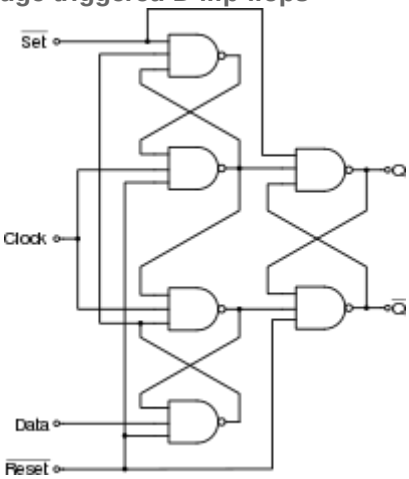
Classical positive-edge-triggered D flip-flop

This circuit[26] consists of two stages implemented by SR NAND latches. The input stage (the two latches on the left) processes the clock and data signals to ensure correct input signals for the output stage (the single latch on the right). If the clock is low, both the output signals of the input stage are high regardless of the data input; the output latch is unaffected and it stores the previous state. When the clock signal changes from low to high, only one of the output voltages (depending on the data signal) goes low and sets/resets the output latch: if D = 0, the lower output becomes low; if D = 1, the upper output becomes low. If the clock signal continues staying high, the outputs keep their states regardless of the data input and force the output latch to stay in the corresponding state as the input logical zero (of the output stage) remains active while the clock is high. Hence the role of the output latch is to store the data only while the clock is low.

A few different types of edge triggered D flip-flops



A positive-edge-triggered D flip-flop



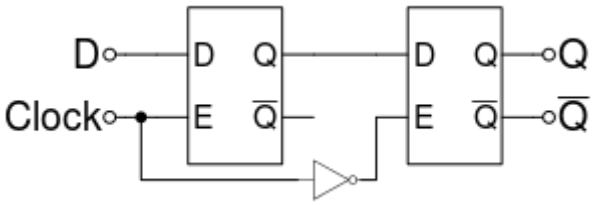
A positive-edge-triggered D flip-flop with set and reset

The circuit is closely related to the gated D latch as both the circuits convert the two D input states (0 and 1) to two input combinations (01 and 10) for the output $\overline{SR}$ latch by inverting the data input signal (both the circuits split the single D signal in two complementary $\overline{S}$ and $\overline{R}$ signals). The difference is that in the gated D latch simple NAND logical gates are used while in the positive-edge-triggered D flip-flop $\overline{SR}$ NAND latches are used for this purpose. The role of these latches is to "lock" the active output producing low voltage (a logical zero); thus the positive-edge-triggered D flip-flop can also be thought of as a gated D latch with latched input gates.

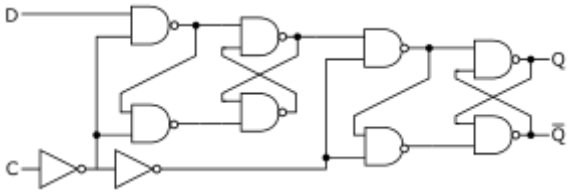
Master–slave edge-triggered D flip-flop

A master–slave D flip-flop is created by connecting two gated D latches in series, and inverting the *enable* input to one of them. It is called master–slave because the master latch controls the slave latch's output value Q and forces the slave latch to hold its value whenever the slave latch is enabled, as the slave latch always copies its new value from the master latch and changes its value only in response to a change in the value of the master latch and clock signal.

For a positive-edge triggered master–slave D flip-flop, when the clock signal is low (logical 0) the "enable" seen by the first or "master" D latch (the inverted clock signal) is high (logical 1). This allows the "master" latch to store the input value when the clock signal transitions from low to high. As the clock signal goes high (0 to 1) the inverted "enable" of the first latch goes low (1 to 0) and the value seen at the input to the master latch is "locked". Nearly simultaneously, the twice inverted "enable" of the second or "slave" D latch transitions from low to high (0 to 1) with the clock signal. This allows the signal captured at the rising edge of the clock by the now "locked" master latch to pass through the "slave" latch. When the clock signal returns to low (1 to 0), the output of the "slave" latch is "locked", and the value seen at the last rising edge of the clock is held while the "master" latch begins to accept new values in preparation for the next rising clock edge.



A master–slave D flip-flop. It responds on the falling edge of the *enable* input (usually a clock)



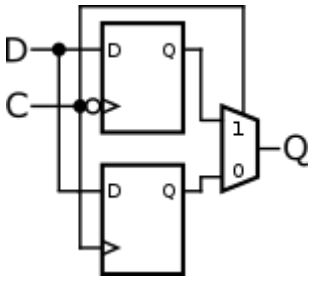
An implementation of a master–slave D flip-flop that is triggered on the rising edge of the clock

Removing the leftmost inverter in the circuit creates a D-type flip-flop that strobes on the *falling edge* of a clock signal. This has a truth table like this:

D	Q	>	Q_{next}
0	X	Falling	0
1	X	Falling	1

Dual-edge-triggered D flip-flop

Flip-Flops that read in a new value on the rising and the falling edge of the clock are called dual-edge-triggered flip-flops. Such a flip-flop may be built using two single-edge-triggered D-type flip-flops and a multiplexer as shown in the image.



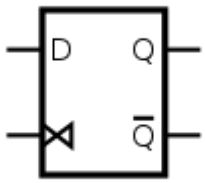
An implementation of a dual-edge-triggered D flip-flop

Edge-triggered dynamic D storage element

An efficient functional alternative to a D flip-flop can be made with dynamic circuits (where information is stored in a capacitance) as long as it is clocked often enough; while not a true flip-flop, it is still called a flip-flop for its functional role. While the master–slave D element is triggered on the edge of a clock, its components are each triggered by clock levels. The "edge-triggered D flip-flop", as it is called even though it is not a true flip-flop, does not have the master–slave properties.

Edge-triggered D flip-flops are often implemented in integrated high-speed operations using dynamic logic. This means that the digital output is stored on parasitic device capacitance while the device is not transitioning. This design of dynamic flip flops also enables simple resetting since the reset operation can be performed by simply discharging one or more internal nodes. A common dynamic flip-flop variety is the true single-phase clock (TSPC) type which performs the flip-flop operation with little power and at high

speeds. However, dynamic flip-flops will typically not work at static or low clock speeds: given enough time, leakage paths may discharge the parasitic capacitance enough to cause the flip-flop to enter invalid states.



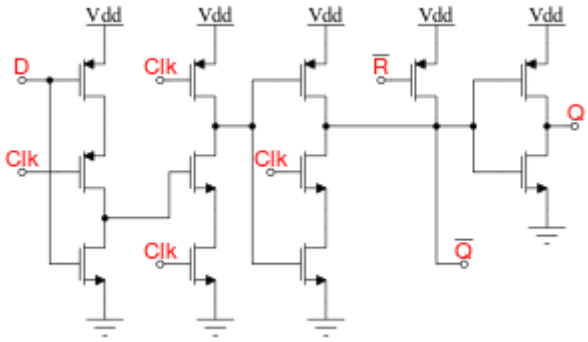
Circuit symbol of a dual-edge-triggered D flip-flop

T flip-flop

If the T input is high, the T flip-flop changes state ("toggles")[27] whenever the clock input is strobed. If the T input is low, the flip-flop holds the previous value. This behavior is described by the characteristic equation:

$$Q_{\text{next}} = T \oplus Q = T\overline{Q} + \overline{T}Q \text{ (expanding the XOR operator)}$$

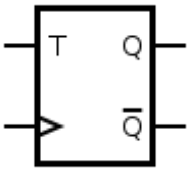
and can be described in a truth table:



A CMOS IC implementation of a dynamic edge-triggered flip-flop with reset

T flip-flop operation[28]

Characteristic table				Excitation table			
T	Q	Q_{next}	Comment	Q	Q_{next}	T	Comment
0	0	0	Hold state (no clock)	0	0	0	No change
0	1	1	Hold state (no clock)	1	1	0	No change
1	0	1	Toggle	0	1	1	Complement
1	1	0	Toggle	1	0	1	Complement



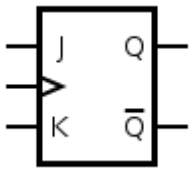
A circuit symbol for a T-type flip-flop

When T is held high, the toggle flip-flop divides the clock frequency by two; that is, if clock frequency is 4 MHz, the output frequency obtained from the flip-flop will be 2 MHz. This "divide by" feature has application in various types of digital counters. A T flip-flop can also be built using a JK flip-flop (J & K pins are connected together and act as T) or a D flip-flop (T input XOR Q_{previous} drives the D input).

JK flip-flop

The JK flip-flop augments the behavior of the SR flip-flop (J: Set, K: Reset) by interpreting the J = K = 1 condition as a "flip" or toggle command. Specifically, the combination J = 1, K = 0 is a command to set the flip-flop; the combination J = 0, K = 1 is a command to reset the flip-flop; and the combination J = K = 1 is a command to toggle the flip-flop, i.e., change its output to the logical complement of its current value. Setting J = K = 0 maintains the current state. To synthesize a D flip-flop, simply set K equal to the

complement of J (input J will act as input D). Similarly, to synthesize a T flip-flop, set K equal to J. The JK flip-flop is therefore a universal flip-flop, because it can be configured to work as an SR flip-flop, a D flip-flop, or a T flip-flop.



A circuit symbol for a positive-edge-triggered JK flip-flop

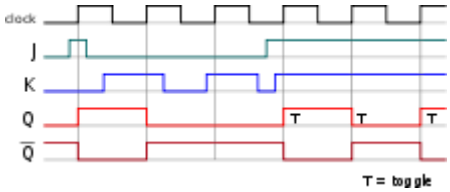
The characteristic equation of the JK flip-flop is:

$$Q_{\text{next}} = J\overline{Q} + \overline{K}Q$$

and the corresponding truth table is:

JK flip-flop operation[28]

Characteristic table				Excitation table				
J	K	Comment	Q_{next}	Q	Q_{next}	Comment	J	K
0	0	Hold state	Q	0	0	No change	0	X
0	1	Reset	0	0	1	Set	1	X
1	0	Set	1	1	0	Reset	X	1
1	1	Toggle	$\overline{Q}$	1	1	No change	X	0

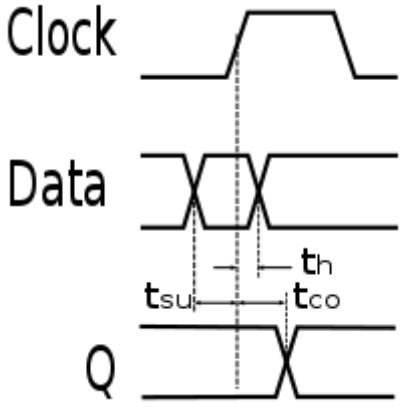


JK flip-flop timing diagram

Timing considerations

Timing parameters

The input must be held steady in a period around the rising edge of the clock known as the aperture. Imagine taking a picture of a frog on a lily-pad.[29] Suppose the frog then jumps into the water. If you take a picture of the frog as it jumps into the water, you will get a blurry picture of the frog jumping into the water—it's not clear which state the frog was in. But if you take a picture while the frog sits steadily on the pad (or is steadily in the water), you will get a clear picture. In the same way, the input to a flip-flop must be held steady during the **aperture** of the flip-flop.



Flip-flop setup, hold and clock-to-output timing parameters

Setup time is the minimum amount of time the data input should be held steady **before** the clock event, so that the data is reliably sampled by the clock.

Hold time is the minimum amount of time the data input should be held steady **after** the clock event, so that the data is reliably sampled by the clock.

Aperture is the sum of setup and hold time. The data input should be held steady throughout this time period.[29]

Recovery time is the minimum amount of time the asynchronous set or reset input should be inactive **before** the clock event, so that the data is reliably sampled by the clock. The recovery time for the asynchronous set or reset input is thereby similar to the setup time for the data input.

Removal time is the minimum amount of time the asynchronous set or reset input should be inactive **after** the clock event, so that the data is reliably sampled by the clock. The removal time for the asynchronous set or reset input is thereby similar to the hold time for the data input.

Short impulses applied to asynchronous inputs (set, reset) should not be applied completely within the recovery-removal period, or else it becomes entirely indeterminable whether the flip-flop will transition to the appropriate state. In another case, where an asynchronous signal simply makes one transition that happens to fall between the recovery/removal time, eventually the flip-flop will transition to the appropriate state, but a very short glitch may or may not appear on the output, dependent on the synchronous input signal. This second situation may or may not have significance to a circuit design.

Set and Reset (and other) signals may be either synchronous or asynchronous and therefore may be characterized with either Setup/Hold or Recovery/Removal times, and synchronicity is very dependent on the design of the flip-flop.

Differentiation between Setup/Hold and Recovery/Removal times is often necessary when verifying the timing of larger circuits because asynchronous signals may be found to be less critical than synchronous signals. The differentiation offers circuit designers the ability to define the verification conditions for these types of signals independently.

Metastability

Flip-flops are subject to a problem called metastability, which can happen when two inputs, such as data and clock or clock and reset, are changing at about the same time. When the order is not clear, within appropriate timing constraints, the result is that the output may behave unpredictably, taking many times longer than normal to settle to one state or the other, or even oscillating several times before settling. Theoretically, the time to settle down is not bounded. In a computer system, this metastability can cause corruption of data or a program crash if the state is not stable before another circuit uses its value; in particular, if two different logical paths use the output of a flip-flop, one path can interpret it as a 0 and the other as a 1 when it has not resolved to stable state, putting the machine into an inconsistent state.[30]

The metastability in flip-flops can be avoided by ensuring that the data and control inputs are held valid and constant for specified periods before and after the clock pulse, called the **setup time** (t_{su}) and the **hold time** (t_h) respectively. These times are specified in the data sheet for the device, and are typically between a few nanoseconds and a few hundred picoseconds for modern devices. Depending upon the flip-flop's internal organization, it is possible to build a device with a zero (or even negative) setup or hold time requirement but not both simultaneously.

Unfortunately, it is not always possible to meet the setup and hold criteria, because the flip-flop may be connected to a real-time signal that could change at any time, outside the control of the designer. In this case, the best the designer can do is to reduce the probability of error to a certain level, depending on the required reliability of the circuit. One technique for suppressing metastability is to connect two or more flip-flops in a chain, so that the output of each one feeds the data input of the next, and all devices share a common clock. With this method, the probability of a metastable event can be reduced to a negligible value, but never to zero. The probability of metastability gets closer and closer to zero as the number of flip-flops connected in series is increased. The number of flip-flops being cascaded is referred to as the "ranking"; "dual-ranked" flip flops (two flip-flops in series) is a common situation.

So-called metastable-hardened flip-flops are available, which work by reducing the setup and hold times as much as possible, but even these cannot eliminate the problem entirely. This is because metastability is more than simply a matter of circuit design. When the transitions in the clock and the data are close together in time, the flip-flop is forced to decide which event happened first. However fast the device is made, there is always the possibility that the input events will be so close together that it cannot detect which one happened first. It is therefore logically impossible to build a perfectly metastable-proof flip-flop. Flip-flops

are sometimes characterized for a maximum settling time (the maximum time they will remain metastable under specified conditions). In this case, dual-ranked flip-flops that are clocked slower than the maximum allowed metastability time will provide proper conditioning for asynchronous (e.g., external) signals.

Propagation delay

Another important timing value for a flip-flop is the clock-to-output delay (common symbol in data sheets: t_{CO}) or propagation delay (t_P), which is the time a flip-flop takes to change its output after the clock edge. The time for a high-to-low transition (t_{PHL}) is sometimes different from the time for a low-to-high transition (t_{PLH}).

When cascading flip-flops which share the same clock (as in a shift register), it is important to ensure that the t_{CO} of a preceding flip-flop is longer than the hold time (t_h) of the following flip-flop, so data present at the input of the succeeding flip-flop is properly "shifted in" following the active edge of the clock. This relationship between t_{CO} and t_h is normally guaranteed if the flip-flops are physically identical. Furthermore, for correct operation, it is easy to verify that the clock period has to be greater than the sum $t_{su} + t_h$.

Generalizations

Flip-flops can be generalized in at least two ways: by making them 1-of-N instead of 1-of-2, and by adapting them to logic with more than two states. In the special cases of 1-of-3 encoding, or multi-valued ternary logic, such an element may be referred to as a *flip-flap-flop*.[31]

In a conventional flip-flop, exactly one of the two complementary outputs is high. This can be generalized to a memory element with N outputs, exactly one of which is high (alternatively, where exactly one of N is low). The output is therefore always a one-hot (respectively *one-cold*) representation. The construction is similar to a conventional cross-coupled flip-flop; each output, when high, inhibits all the other outputs.[32] Alternatively, more or less conventional flip-flops can be used, one per output, with additional circuitry to make sure only one at a time can be true.[33]

Another generalization of the conventional flip-flop is a memory element for multi-valued logic. In this case the memory element retains exactly one of the logic states until the control inputs induce a change.[34] In addition, a multiple-valued clock can also be used, leading to new possible clock transitions.[35]

See also

- Latching relay
- Positive feedback
- Pulse transition detector
- Static random-access memory
- Sample and hold, analog latch

References

1. Pedroni, Volnei A. (2008). *Digital electronics and design with VHDL* (https://books.google.com/books?id=-ZAccwyQeXMC). Morgan Kaufmann. p. 329. ISBN 978-0-12-374270-4.

2. Latches and Flip Flops (http://rfic.eecs.berkeley.edu/ee100/pdf/lect24.pdf) (EE 42/100 Lecture 24 from Berkeley) *"...Sometimes the terms flip-flop and latch are used interchangeably..."*
3. Roth, Charles H. Jr. "Latches and Flip-Flops." Fundamentals of Logic Design. Boston: PWS, 1995. Print.
4. GB 148582 (https://worldwide.espacenet.com/textdoc?DB=EPODOC&IDX=GB148582), Eccles, William Henry & Jordan, Frank Wilfred, "Improvements in ionic relays", published 1920-08-05
5. See:
    - W. H. Eccles and F. W. Jordan (19 September 1919) "A trigger relay utilizing three-electrode thermionic vacuum tubes," (https://archive.org/details/electricaljourna83lond#page/298/mode/2up) *The Electrician*, **83** : 298.
    - Reprinted in: W. H. Eccles and F. W. Jordan (December 1919) "A trigger relay utilizing three-electrode thermionic vacuum tubes," (https://babel.hathitrust.org/cgi/pt?id=mdp.39015021318277;view=1up;seq=165) *The Radio Review*, **1** (3) : 143–146.
    - Summary in: W. H. Eccles and F. W. Jordan (1919) "A trigger relay utilising three electrode thermionic vacuum tubes," (https://archive.org/stream/reportofbritisha20adva#page/270/mode/2up) *Report of the Eighty-seventh Meeting of the British Association for the Advancement of Science: Bournemouth: 1919, September 9–13*, pp. 271–272.
6. Pugh, Emerson W.; Johnson, Lyle R.; Palmer, John H. (1991). *IBM's 360 and early 370 systems* (https://archive.org/details/ibms360early370s0000pugh). MIT Press. p. 10 (https://archive.org/details/ibms360early370s0000pugh/page/10). ISBN 978-0-262-16123-7.
7. Flowers, Thomas H. (1983), "The Design of Colossus" (http://www.ivorcatt.com/47c.htm), *Annals of the History of Computing*, **5** (3): 249, doi:10.1109/MAHC.1983.10079 (https://doi.org/10.1109%2FMAHC.1983.10079), S2CID 39816473 (https://api.semanticscholar.org/CorpusID:39816473)
8. Gates, Earl D. (2000-12-01). *Introduction to electronics* (https://books.google.com/books?id=IwC5GIA0cREC) (4th ed.). Delmar Thomson (Cengage) Learning. p. 299. ISBN 978-0-7668-1698-5.
9. Fogiel, Max; Gu, You-Liang (1998). *The Electronics problem solver, Volume 1* (https://books.google.com/books?id=6oXuRAAACAAJ) (revised ed.). Research & Education Assoc. p. 1223. ISBN 978-0-87891-543-9.
10. P. L. Lindley, Aug. 1968, EDN (magazine), (letter dated June 13, 1968).
11. Phister, Montgomery (1958). *Logical Design of Digital Computers* (https://books.google.com/books?id=Ri1IAAAAIAAJ). Wiley. p. 128. ISBN 9780608102658.
12. US 2850566 (https://worldwide.espacenet.com/textdoc?DB=EPODOC&IDX=US2850566), Nelson, Eldred C., "High-speed printing system", published 1958-09-02, assigned to Hughes Aircraft Co.
13. Shiva, Sajjan G. (2000). *Computer design and architecture* (https://books.google.com/books?id=kKQFttdG7hcC) (3rd ed.). CRC Press. p. 81. ISBN 978-0-8247-0368-4.
14. Langholz, Gideon; Kandel, Abraham; Mott, Joe L. (1998). *Foundations of Digital Logic Design* (https://books.google.com/books?id=4sX9fTGRo7QC). Singapore: World Scientific Publishing Co. Ptc. Ltd. p. 344. ISBN 978-981-02-3110-1.
15. "Summary of the Types of Flip-flop Behaviour" (https://www.ee.usyd.edu.au/tutorials/digital_tutorial/part3/fl-types.htm). Retrieved on 16 April 2018.
16. Hinrichsen, Diederich; Pritchard, Anthony J. (2006). *Mathematical Systems Theory I: Modelling, State Space Analysis, Stability and Robustness* (https://books.google.com/books?id=u4JYt7OdptYC&pg=PA63). Springer. pp. 63–64. ISBN 9783540264101.

17. Farhat, Hassan A. (2004). *Digital design and computer organization* (https://books.google.com/books?id=jwZZcgAACAAJ). Vol. 1. CRC Press. p. 274. ISBN 978-0-8493-1191-8.
18. Kogge, Peter M. (1981). *The Architecture of Pipelined Computers*. McGraw-Hill. pp. 25–27. ISBN 0-07-035237-2.
19. Cotten, L. W. (1965). "Circuit Implementation of High-Speed Pipeline Systems". *AFIPS Proc. Fall Joint Computer Conference*: 489–504. doi:10.1145/1463891.1463945 (https://doi.org/10.1145%2F1463891.1463945). S2CID 15955626 (https://api.semanticscholar.org/CorpusID:15955626).
20. Earle, John G. (March 1965). "Latched Carry-Save Adder". *IBM Technical Disclosure Bulletin*. **7** (10): 909–910.
21. Omondi, Amos R. (1999-04-30). *The Microarchitecture of Pipelined and Superscalar Computers* (https://books.google.com/books?id=Pf2ZbKM2-5MC). Springer. pp. 40–42. ISBN 978-0-7923-8463-2.
22. Kunkel, Steven R.; Smith, James E. (May 1986). "Optimal Pipelining in Supercomputers". *ACM SIGARCH Computer Architecture News*. ACM. **14** (2): 404–411 [406]. CiteSeerX 10.1.1.99.2773 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.99.2773). doi:10.1145/17356.17403 (https://doi.org/10.1145%2F17356.17403). ISSN 0163-5964 (https://www.worldcat.org/issn/0163-5964). S2CID 2733845 (https://api.semanticscholar.org/CorpusID:2733845).
23. The D Flip-Flop (http://www.play-hookey.com/digital/sequential/d_nand_flip-flop.html)
24. "Edge-Triggered Flip-flops" (https://web.archive.org/web/20130908211358/http://www.ee.usyd.edu.au/tutorials/digital_tutorial/part2/flip-flop02.html). Archived from the original (http://www.ee.usyd.edu.au/tutorials/digital_tutorial/part2/flip-flop02.html) on 2013-09-08. Retrieved 2011-12-15.
25. A Survey of Digital Computer Memory Systems (http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4051207&tag=1)
26. SN7474 TI datasheet (http://focus.ti.com/lit/ds/symlink/sn7474.pdf)
27. "Understanding the T Flip-Flop" (https://www.oemsecrets.com/articles/understanding-the-t-flip-flop). *oemsecrets.com*. Retrieved 29 April 2021.
28. Mano, M. Morris; Kime, Charles R. (2004). *Logic and Computer Design Fundamentals, 3rd Edition*. Upper Saddle River, NJ, USA: Pearson Education International. p. 283. ISBN 0-13-191165-1.
29. Harris, S; Harris, D (2016). *Digital Design and Computer Architecture - ARM Edition*. Morgan Kaufmann, Waltham, MA. ISBN 978-0-12-800056-4.
30. Chaney, Thomas J.; Molnar, Charles E. (April 1973). "Anomalous Behavior of Synchronizer and Arbiter Circuits". *IEEE Transactions on Computers*. **C-22** (4): 421–422. doi:10.1109/T-C.1973.223730 (https://doi.org/10.1109%2FT-C.1973.223730). ISSN 0018-9340 (https://www.worldcat.org/issn/0018-9340). S2CID 12594672 (https://api.semanticscholar.org/CorpusID:12594672).
31. Often attributed to Don Knuth (1969) (see Midhat J. Gazalé (2000). *Number: from Ahmes to Cantor* (https://books.google.com/books?id=WO1gQYIrG24C). Princeton University Press. p. 57. ISBN 978-0-691-00515-7.), the term *flip-flap-flop* actually appeared much earlier in the computing literature, for example, Bowdon, Edward K. (1960). *The design and application of a "flip-flap-flop" using tunnel diodes (Master's thesis)* (https://books.google.com/books?id=0pA7AAAAMAAJ&q=flip-flap-flop+core). University of North Dakota., and in Alexander, W. (Feb 1964). "The ternary computer" (https://scholar.google.com/scholar?gcx=c&q=alexander+flip-flap-flop&um=1&ie=UTF-8&hl=en&sa=N&tab=ws). *Electronics and Power*. IET. **10** (2): 36–39. doi:10.1049/ep.1964.0037 (https://doi.org/10.1049%2Fep.1964.0037).

32. "Ternary "flip-flap-flop" " (https://web.archive.org/web/20090105193858/http://www.goldenmuseum.com/1411FlipFlap_engl.html). Archived from the original (http://www.goldenmuseum.com/1411FlipFlap_engl.html) on 2009-01-05. Retrieved 2009-10-17.
33. US 6975152 (https://worldwide.espacenet.com/textdoc?DB=EPODOC&IDX=US6975152)
34. Irving, Thurman A.; Shiva, Sajjan G.; Nagle, H. Troy (March 1976). "Flip-Flops for Multiple-Valued Logic". *IEEE Transactions on Computers*. **C-25** (3): 237–246. doi:10.1109/TC.1976.5009250 (https://doi.org/10.1109%2FTC.1976.5009250). S2CID 34323423 (https://api.semanticscholar.org/CorpusID:34323423).
35. Wu, Haomin; Zhuang Nan (1991). "Research into ternary edge-triggered JKL flip-flop". *Journal of Electronics (China)*. **8** (Volume 8, Number 3 / July, 1991): 268–275. doi:10.1007/BF02778378 (https://doi.org/10.1007%2FBF02778378). S2CID 61275953 (https://api.semanticscholar.org/CorpusID:61275953).

External links

- FlipFlop Hierarchy (http://teahlab.com/Multivibrators_FlipFlop/) Archived (https://web.archive.org/web/20150408221630/http://teahlab.com/Multivibrators_FlipFlop/) 2015-04-08 at the Wayback Machine, shows interactive flipflop circuits.
- The J-K Flip-Flop (http://www.allaboutcircuits.com/vol_4/chpt_10/6.html)

Retrieved from "https://en.wikipedia.org/w/index.php?title=Flip-flop_(electronics)&oldid=1077253513"

This page was last edited on 15 March 2022, at 10:16 (UTC).

Text is available under the Creative Commons Attribution-ShareAlike License 3.0; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.