Module 4
Chapter 1: Introduction to VHDL.

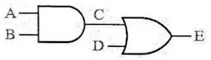
The acronym VHDL stands for VHSIC-HDL (Very High Speed Integrated Circuit-Hardware Description Language). *VHDL* is a hardware description language that is used to describe the behavior and structure of digital systems. *VHDL* is a general-purpose hardware description language which can be used to describe and simulate the operation of a wide variety of digital systems, ranging in complexity from a few gates to an interconnection of many complex integrated circuits.
VHDL was originally developed to allow a uniform method for specifying digital systems. The VHDL language became an IEEE standard in 1987, and it is widely used in industry. IEEE published a revised VHDL standard in 1993.
VHDL can describe a digital system at several different levels—behavioral, data flow, and structural. For example,

- A binary adder could be described at the *behavioral* level in terms of its function of adding two binary numbers, without giving any implementation details.
- The same adder could be described at the *data flow* level by giving the logic equations for the adder.
- Finally, the adder could be described at the *structural* level by specifying the interconnections of the gates which make up the adder.

4.1 VHDL DESCRIPTION OF COMBINATIONAL CIRCUITS:

Let's begin by describing a simple gate circuit using VHDL. A VHDL signal is used to describe a signal in a physical system. The VHDL Language also includes a variables similar to variables in programming languages, but to obtain a synthesizable code for hardware, signals should be used to represent hardware signals. VHDL variables are not used in the text. The gate circuit for the circuit shown in the figure below has five signals: A, B, C, D, and E. Symbol "<=" is the signal assignment operator which indicates that the value computed on the right hand side is assigned to the signal on the left hand side.

A behavioral description of the circuit shown in the figure is E<= D or (A and B); Parentheses are used to specify the order of the operator execution.



```
C <= A and B after 5 ns;
E <= C or D after 5 ns;
```

FIGURE: Gate Circuit

The two assignment statement in figure above give a data flow description of the circuit where it is assumed that each gate has a 5ns propagation delay. When the statements in figure above are simulated, that initially A=1, and B = C =D =E = 0. If B changes to 1 at time 0, C will change to 1 at time=5ns. Then, E change to 1 at time=10ns.

The circuit shown in the figure can also be described using the structural VHDL code. To do so requires that a two Input AND-Gate component and the two input OR-gate component be declared and defined. Components may be declared and defined either in a library or within the architecture part of the VHDL code. Instantiation statements are used to specify how components are connected. Each copy of a component requires separate instantiation statement to specify how it is connected to other components and to the port inputs and outputs.

Instantiation statement is a concurrent statement that executes anytime, one of the input signals in its port map changes.

The circuit of figure above is described by instantiating the AND gate and OR gate as follows:

Gate1: AND2 port map (A, B, C);
Gate2: OR2 port map (C, D, E);

The port map for Gate1 connects A and B to the AND-gate inputs, and it connects D to the AND gate output. Since an instantiation statement is concurrent, whenever A or B changes, these changes go to the Gate1 inputs, and then the component computes a new value of C. Similarly the second statement passes the changes in C or D to the Gate2 inputs, and then the component compute the new value of E. This is exactly how the real hardware works. Instantiating new component is different than calling a function in a computer program. A function returns a new value whenever it is called, but an instantiated component computes new output value whenever its input changes.

VHDL signal assignment statements such as the ones shown in the figure, are examples of concurrent statements. VHDL simulator monitors the right side of the concurrent statement, and any time a signal changes the expression on the right side is immediately re-evaluated. The new value is assigned to the signal on the left side of the appropriate delay. This is exactly the way the hardware works. Anytime a gate input changes, the gate output is recomputed by the hardware, and the output changes after the gate delay.
When we initially describe a circuit, we may not be concerned about propagation delays. If we write
C <= A and B;
E <= C or D;

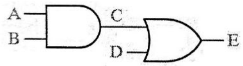
This implies that the propagation delay are 0ns. In this case The Simulator will assume and infinitesimal delay referred to as Δ (delta). Assume that initially A=1 and B=C=D=E=0. If B is changed to 1 at time = 1ns, then C will change at a time 1+Δ and E will change the time 1 + 2Δ.
Unlike a sequential program, the order of concurrent statement is unimportant. If we write
E <= C or D;
C <= A and B;
The simulation results would be exactly same as before.
In VHDL, a signal assignment statement has the form:
signal_name <= expression [after delay];

The expression is evaluated when the statement is executed, and the signal on the left side is scheduled to change after delay. The square brackets indicate that after delay is optional. If after delay is omitted, then the signal is scheduled to be updated after a *delta delay, Δ* (infinitesimal delay).

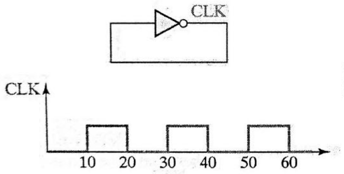
A VHDL *signal* is used to describe a signal in a physical system. The VHDL language also includes *variables* similar to variables in programming languages.

In general, VHDL is *not case sensitive*, that is, capital and lower case letters are treated the same by the compiler and the simulator. Signal names and other VHDL identifiers may contain letters, numbers, and the underscore character (_). An identifier must start with a letter, and it cannot end with an underscore. Thus, C123 and ab_23 are legal identifiers, but 1ABC and ABC_ are not. Every VHDL statement must be terminated with a semicolon. Spaces, tabs, and carriage returns are treated in the same way. This means that a VHDL statement can be continued over several lines, or several statements can be placed on one line. In a line of VHDL code, anything following a double dash (--) is treated as a comment. Words such as *and*, *or*, and *after* are reserved words (or keywords) which have a special meaning to the VHDL compiler.

The gate circuit of the following Figure has five signals: A, B, C, D, and E. The symbol "<=" is the signal assignment operator which indicates that the value computed on the right-hand side is assigned to the signal on the left side.



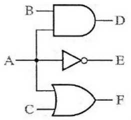
The following Figure shows an inverter with the output connected back to the input. If the output is "0", then this "0" feeds back to the input and the inverter output changes to "1" after the inverter delay, assumed to be 10 ns. Then, the "1" feeds back to the input, and the output changes to "0" after the inverter delay. The signal CLK will continue to oscillate between „0" and „1", as shown in the waveform. The corresponding concurrent VHDL statement will produce the same result. If CLK is initialized to "0", the statement executes and CLK changes to "1" after 10 ns. Because CLK has changed, the statement executes again, and CLK will change back to "0" after another 10 ns. This process will continue indefinitely.



```
CLK <= not CLK after 10 ns;
```

FIGURE : Inverter with Feedback

The following Figure shows three gates that have the signal A as a common input and the corresponding VHDL code. The three concurrent statements execute simultaneously whenever A changes, just as the three gates start processing the signal change at the same time. However, if the gates have different delays, the gate outputs can change at different times. If the gates have delays of 2 ns, 1 ns, and 3 ns, respectively, and A changes at time 5 ns, then the gate outputs D, E, and F can change at times 7 ns, 6 ns, and 8 ns, respectively. However, if no delays were specified, then D, E, and F would all be updated at time 5 + Δ.

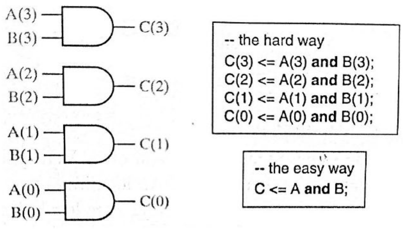


```
-- when A changes, these concurrent
-- statements all execute at the same time
D <= A and B after 2 ns;
E <= not A after 1 ns;
F <= A or C after 3 ns;
```

FIGURE: Three Gates with a Common Input and Different Delays

In these examples, every signal is of type bit, which means it can have a value of "0" or "1". (Bit values in VHDL are enclosed in single quotes to distinguish them from integer values). In digital design, we often need to perform the same operation on a group of signals. A one-dimensional array of bit signals is referred to as a bit-vector. If a 4-bit vector named B has an index range 0 through 3, then the four elements of the bit-vector are designated B(0), B(1), B(2),

and B(3). The statement B <= “0110”, assigns “0” to B(0), “1” to B(1), “1” to B(2), and “0” to B(3).
The following Figure shows an array of four AND gates. The inputs are represented by bit-vectors A and B, and the outputs by bit-vector C. Although we can write four VHDL statements to represent the four gates, it is much more efficient to write a single VHDL statement that performs the AND operation on the bit-vectors A and B. When applied to bit-vectors, the AND operator performs the AND operation on corresponding pairs of elements.



```
-- the hard way
C(3) <= A(3) and B(3);
C(2) <= A(2) and B(2);
C(1) <= A(1) and B(1);
C(0) <= A(0) and B(0);
```

```
-- the easy way
C <= A and B;
```

FIGURE: Array of AND Gates

Inertial delay model: Signal assignment statements containing “after delay” create what is called an inertial delay model. Consider a device with an inertial delay of D time units. If an input change to the device will cause its output to change, then the output changes D time units later. However, this is not what happens if the device receives two input changes within a period of D time units and both input changes should cause the output to change. In this case the device output does not change in response to either input change.

Example: consider the signal assignment **C <= A and B after 10 ns;**
Assume A and B are initially 1, and A changes to 0 at 15 ns, to 1 at 30 ns, and to 0 at 35 ns. Then C changes to 1 at 10 ns and to 0 at 25 ns, but C does not change in response to the A changes at 30 ns and 35 ns; because these two changes occurred less than 10 ns apart.
A device with an inertial delay of D time units filters out output changes that would occur in less than or equal to D time units.

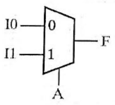
Ideal (Transport) delay: VHDL can also model devices with an ideal (transport) delay. Output changes caused by input changes to a device exhibiting an ideal (transport) delay of D time units are delayed by D time units, and the output changes occur even if they occur within D time units. The VHDL signal assignment statement that models ideal (transport) delay is

Signal_name <= transport expression after delay

Example: consider the signal assignment **C <= transport A and B after 10 ns;**
Assume A and B are initially 1 and A changes to 0 at 15 ns, to 1 at 30 ns, and to 0 at 35 ns. Then C changes to 1 at 10 ns, to 0 at 25 ns, to 1 at 40 ns, and to 0 at 45 ns. Note that the last two changes are separated by just 5 ns.

4.2 VHDL MODELS FOR MULTIPLEXERS:

The following Figure shows a 2-to-1 multiplexer (MUX) with two data inputs and one control input.



```
-- conditional signal assignment statement
F <= I0 when A = '0' else I1;
```

FIGURE: 2-to-1 Multiplexer

The MUX output is $F = A'I_0 + AI_1$. The corresponding VHDL statement is

F <= (not A and I0) or (A and I1);

Alternatively, we can represent the MUX by a conditional signal assignment statement,

F <= I0 when A = "0" else I1;

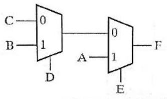
This statement executes whenever A, I0, or I1 changes. The MUX output is I0 when A = "0", and else it is I1. In the conditional statement, I0, I1, and F can either be bits or bit-vectors. The general form of conditional signal assignment statement is

Signal_name <= expression1 when condition1

else expression2 when condition2

[else expression N];

The following Figure shows how two cascaded MUXes can be represented by a conditional signal assignment statement. The output MUX selects A when E = "1"; or else it selects the output of the first MUX, which is B when D = "1", or else it is C.



```
F <= A when E = '1'
        else B when D = '1'
        else C;
```

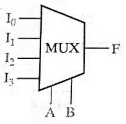
FIGURE: Cascaded 2-to-1 MUXes

The following Figure shows a 4-to-1 MUX with four data inputs and two control inputs, A and B. The control inputs select which one of the data inputs is transmitted to the output. The logic equation for the 4-to-1 MUX is $F = A'B'I0 + A'BI1 + AB'I2 + ABI3$.

One way to model the MUX is with the VHDL statement

*F <= (not A **and not** B **and** I0) **or** (not A **and** B **and** I1) **or***
*(A **and not** B **and** I2) **or** (A **and** B **and** I3);*

Another way to model the 4-to-1 MUX is to use a conditional assignment statement (given in Figure below):



```
sel <= A&B;
-- selected signal assignment statement
with sel select
  F <= I0 when "00",
       I1 when "01",
       I2 when "10",
       I3 when "11";
```

FIGURE: 4-to-1 Multiplexer

The expression A&B means A concatenated with B, that is, the two bits A and B are merged together to form a 2-bit vector. This bit vector is tested, and the appropriate MUX input is selected. For example, if A = "1" and B = "0", A&B = "10" and I2 is selected. Instead of concatenating A and B, we could use a more complex condition also (as given in above Figure).

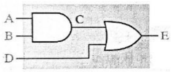
A third way to model the MUX is to use a selected signal assignment statement; we first set Sel equal to A&B. The value of Sel then selects the MUX input that is assigned to F.
The general form of a selected signal assignment statement is

```
with expression_s select
        signal_s <= expression1 [after delay-time ] when choice1,
                    expression2 [after delay-time ] when choice2,
                    …..
                   [expression_n [after delay-time ] when others];
```

First, expression_s is evaluated. If it equals choice1, signal_s is set equal to expression1; if it equals choice2, signal_s is set equal to expression2; etc. If all possible choices for the value of expression_s are given, the last line should be omitted; otherwise, the last line is required. When it is present, if expression_s is not equal to any of the enumerated choices, signal_s is set equal to expression_n. The signal_s is updated after the specified delay-time, or after if the "after delay-time" is omitted.

4.3 VHDL MODULES:

To write a complete VHDL module, we must declare all of the input and output signals using an entity declaration, and then specify the internal operation of the module using an architecture declaration. As an example, consider the following Figure.

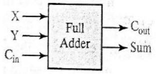


```
entity two_gates is
    port (A,B,D: in bit; E: out bit);
end two_gates;
architecture gates of two_gates is
    signal C: bit;
begin
    C <= A and B; -- concurrent
    E <= C or D; -- statements
end gates;
```

FIGURE: VHDL Module with Two Gates

When we describe a system in VHDL, we must specify an entity and architecture at the top level. The entity declaration gives the name "two_gates" to the module. The port declaration specifies the inputs and outputs to the module. A, B, and D are input signals of type bit, and E is an output signal of type bit. The architecture is named "gates". The signal C is declared within the architecture because it is an internal signal. The two concurrent statements that describe the gates are placed between the keywords begin and end.

Example: *To write the entity and architecture for a full adder module.*
The entity specifies the inputs and outputs of the adder module, as shown in the following Figure. The port declaration specifies that X, Y and Cin are input signals of type bit, and that Cout and Sum are output signals of type bit.



```
entity FullAdder is
    port (X,Y,Cin: in bit;          -- Inputs
              Cout, Sum: out bit);  -- Outputs
end FullAdder;
```

FIGURE: Entity Declaration for a Full Adder Module

The operation of the full adder is specified by an architecture declaration:

To write a complete VHDL module, we must declare all of the input and output signals using an **entity** declaration, and then specify the internal operation of the module using an **architecture** declaration. The two concurrent statements that describe the gates are placed between the keywords **begin** and **end**.
When we describe a system in VHDL, we must specify an entity and an architecture at the top level, and also specify an entity and architecture for each of the component modules that are part of the system. Each entity declaration includes a list of interface signals that can be used to connect to other modules or to the outside world. We will use entity declarations of the form:
entity entity-name is
[port(interface-signal-declaration);]
end [entity] [entity-name];

The items enclosed in square brackets are optional. The interface-signal-declaration normally has the following form:

list-of-interface-signals: mode type [: _ initial-value]
{; list-of-interface-signals: mode type [: _ initial-value]};

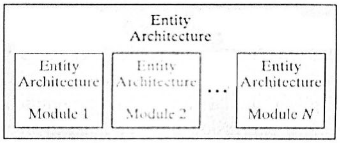


FIGURE: VHDL Program Structure

The curly brackets indicate zero or more repetitions of the enclosed clause. Input signals are of mode **in**, output signals are of mode **out**, and bi-directional signals are of mode **inout.** Associated with each entity is one or more architecture declarations of the form:

architecture architecture-name **of** entity-name **is**
[declarations]
begin
architecture body
end [**architecture**] [architecture-name];

In the declarations section, we can declare signals and components that are used within the architecture. The architecture body contains statements that describe the operation of the module.

```
architecture Equations of FullAdder is
begin           -- concurrent assignment statements
    Sum <= X xor Y xor Cin after 10 ns;
    Cout <= (X and Y) or (X and Cin) or (Y and Cin) after 10 ns;
end Equations;
```

In this example, the architecture name (Equations) is arbitrary, but the entity name (FullAdder) must match the name used in the associated entity declaration. The VHDL assignment statements for Sum and Cout represent the logic equations for the full adder. Several other architectural descriptions such as a truth table or an interconnection of gates could have been used instead. In the Cout equation, parentheses are required around (X and Y) because VHDL does not specify an order of precedence for the logic operators.
Four-Bit Full Adder:

The Full-Adder module defined above can be used as a component in a system which consists of four full adders connected to form a 4-bit binary adder as shown in the figure below.

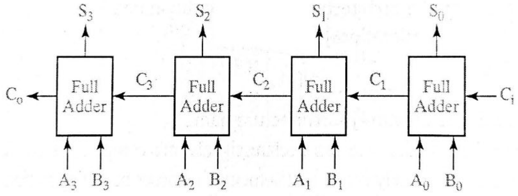


FIGURE: 4-Bit Binary Adder

First declare the 4-bit adder as an entity (see the following Figure). Since, the inputs and the sum output are four bits wide, declare them as bit_vectors which are dimensioned 3 downto 0. Next, specify the FullAdder as a component within the architecture of Adder4 (see the following Figure). The component specification is very similar to the entity declaration for the full adder, and the input and output port signals correspond to those declared for the full adder. Following the component statement, declare a 3-bit internal carry signal C.

In the body of the architecture, create several instances of the FullAdder component. Each copy of FullAdder has a name (such as FA0) and a port map. The signal names following the port map correspond one-to-one with the signals in the component port. Thus, A(0), B(0), and Ci correspond to the inputs X, Y, and Cin, respectively. C(1) and S(0) correspond to the Cout and Sum outputs. Note that the order of the signals in the port map must be the same as the order of the signals in the port of the component declaration.

In preparation for simulation, we can place the entity and architecture for the Full- adder and for the Adder4 together in one file and compile. Alternatively, we could compare the full address separately and place the resulting code in a library which is linked in when we compile Adder4.

Figure below shows the structural description of 4 bit adder.

```
entity Adder4 is
        port (A, B: in bit_vector(3 downto 0); Ci: in bit; – Inputs
                S: out bit_vector(3 downto 0); Co: out bit); – Outputs
end Adder4;
architecture Structure of Adder4 is
component FullAdder
        port (X, Y, Cin: in bit; – Inputs
                Cout, Sum: out bit); – Outputs
end component;
signal C: bit_vector(3 downto 1);
begin – instantiate four copies of the FullAdder
        FA0: FullAdder port map (A(0), B(0), Ci, C(1), S(0));
        FA1: FullAdder port map (A(1), B(1), C(1), C(2), S(1));
        FA2: FullAdder port map (A(2), B(2), C(2), C(3), S(2));
        FA3: FullAdder port map (A(3), B(3), C(3), Co, S(3));
end Structure;
```

FIGURE: Structural Description of 4-Bit Adder

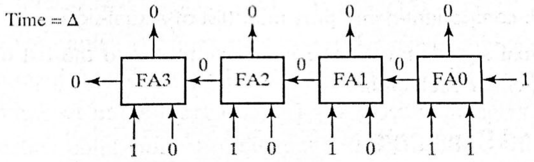
We will use the following simulator commands to test Adder4.

```
add list A B Co C Ci S          – put these signals on the output list
force A                         – set the A inputs to 1111
force B 0001                    – set the B inputs to 0001
force Ci 1                – set Ci to 1
run 50 ns                       – run the simulation for 50 ns
force Ci 0
force A 0101
force B 1110
run 50 ns
```

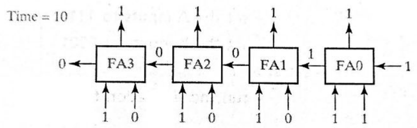
We have chosen to run the simulation for 15 nanoseconds because this is more than enough time for the carry to propagate through all of the four Full-adders. The simulation results for the above command list are:

```
ns    delta    a      b      co    c      ci    s
0     +0       0000   0000   0     000    0     0000
0     +1       1111   0001   0     000    1     0000
10    +0       1111   0001   0     001    1     1111
20    +0       1111   0001   0     011    1     1101
30    +0       1111   0001   0     111    1     1001
40    +0       1111   0001   1     111    1     0001
50    +0       0101   1110   1     111    0     0001
60    +0       0101   1110   1     110    0     0101
70    +0       0101   1110   1     100    0     0111
80    +0       0101   1110   1     100    0     0011
```

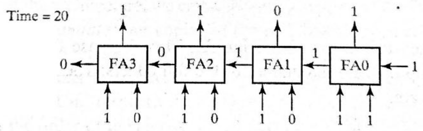
The listing shows how the carry propagate one position every 10 nanoseconds. The full adder inputs change at time = Δ:



The sum and carry are computed by each FA and appear at the FA outputs 10 ns later:



Because the inputs to FA1 have changed, the outputs change 10 ns later:



The final simulation results are:

1111+0001+1=0001 with a carry of 1 (at time=40ns) and

0101+1110+0=0011 with a carry of 1 (at time =80ns)

The simulation stops at 80 nanoseconds because no for the changes occur after that time. Components used within the architecture are declared at the beginning of architecture using a component declaration of the form

component component_name
port (list-of-interface-Signals-and-their-types);
end component;

The port Claus used in the component declaration as the same format support Claus used in the entity declaration. The connection to each component used in the circuit are specified by using a component instance creation statement of the form

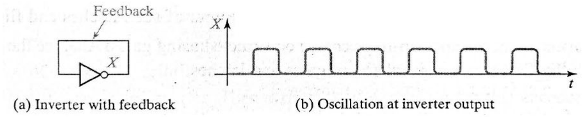
label: component-name **port map** (list-of-actual-signal);

The list of actual signals must correspond one-to-one to the list of interface signals specified in the component declaration.

Module 4
Chapter 2: Latches and Flip-Flops.

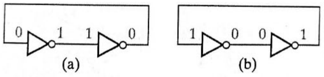
Sequential switching circuits have the property that the output depends not only on the present input but also on the past sequence of inputs. In effect, these circuits must be able to "remember" something about the past history of the inputs in order to produce the present output. Latches and flip-flops are commonly used memory devices in sequential circuits. Basically, latches and flip-flops are memory devices which can assume one of two stable output states and which have one or more inputs that can cause the output state to change.

Each of the flip flops are the clock Input, and the flip-flop can only change its state in response to a clock pulse. A memory element that has no clock input is often called a **latch**.
In order to construct a switching circuit that has a memory such as latch or flip flop we must introduce feedback into the circuit. In simple cases we can analyze circuits with feedback by tracing signal through the circuit. For example consider the circuit shown in the figure below.



(a) Inverter with feedback (b) Oscillation at inverter output

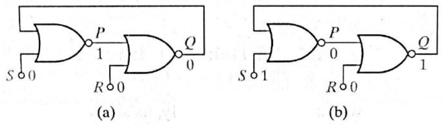
If at some instant of time inverter input is 0, this 0 will propagate through the inverter and cause the output to become 1 after the inverter delay. This 1 is fed back into the input, so after the propagation delay the inverter output will become 0. When this 0 feeds back into the input the output again switch to 1 and so forth. The inverter output will continue to oscillate back and forth between 0 and 1 as shown in the figure above and it will never reaches stable condition. An oscillator can be created by using any odd number of inverters. The oscillator waveform has a high and low time that is the sum of propagation times of the inverters. For example with n inverters and with all having the same delay, the oscillator waveform high time is (n + 1)/2 times the high-to-low inverter propagation delay plus (n - 1)/2 times low-to-high inverter propagation delay.



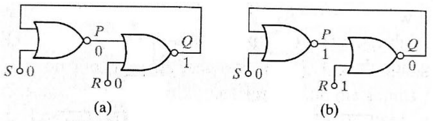
Consider a feedback loop which has two inverters in it as shown in the figure above. in this case the circuit as to stable condition, often referred to as a stable States. If the input to the first inverter is 0, its output will be 1. Then 0 will feed back into the first inverted, but because this input is already 0 no changes will occur. As shown in figure (b) a second stable state of the circuit occurs when the input to the first inverter is 1 and the input to the second inverter is 0. A simple loop of two inverters lacks any external means of initializing the state to one of the stable States.
Sequential circuits must contain feedback, but not all circuits with feedback are sequential. There are few circuits containing feedback that are combinational.

4.4 SET RESET LATCH:

A simple latch can be constructed by introducing feedback into a NOR-gate circuit, as given in the below figure (a). As indicated, if the inputs are S = R = 0, the circuit can assume a stable state with Q = 0 and P = 1.



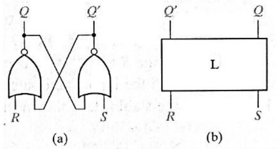
1. S = 0 & R = 0: A stable condition of the circuit because P = 1 feeds into the second gate forcing the output to be Q = 0, and Q = 0 feeds into the first gate allowing its output to be 1.
2. S = 1 & R = 0: An unstable condition or state of the circuit because both the inputs and output of the second gate are 0; therefore Q will change to 1, leading to the stable state.



a) S = 0 & R = 0: The circuit will not change state because Q = 1 feeds back into the first gate, causing P to remain 0. Note that the inputs are again S = 0 & R = 0, but the outputs are different than those with which we started. Thus, the circuit has two different stable states for a given set of inputs.

b) S = 0 & R = 1: Q will become 0 and P will then change back to 1. An input S = 1 *set*s the output to Q = 1, and an input R = 1 *reset*s the output to Q = 0.The circuit is commonly referred to as a *set-reset (S-R) latch* (restriction that R and S cannot be 1 simultaneously).

This circuit is said to have memory because its output depends not only on the present inputs, but also on the past sequence of inputs. If we restrict the inputs so that R = S = 1 is not allowed, the stable states of the outputs P and Q are always complements, that is, P = Q'. To emphasize the symmetry between the operations of the two gates, the circuit is often drawn in cross-coupled form, as shown in the following Figure (a).



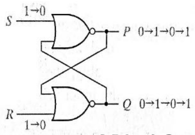


FIGURE: S-R Latch **FIGURE: Improper S-R Latch Operation**

If S = R = 1, the latch will not operate properly, as shown in above Figure (c). Note that, when S and R are both l, P and Q are both 0. Therefore, P is not equal to *Q'*, and this violates a basic rule of latch operation.

The following Figure shows a timing diagram for the S-R latch. Note that when S changes to 1 at time tl, Q changes to 1 a short time (ε - response time or delay time of latch) later. At time t2, when S changes back to 0, Q does not change. At time t3, R changes to 1, and Q changes back to 0 a short time (ε) later. The duration of the S (or R) input pulse must normally be at least as great as ε in order for a change in the state of Q to occur.

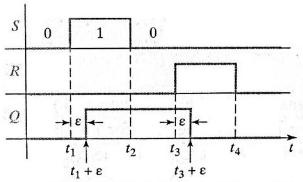


FIGURE: Timing Diagram for S-R Latch

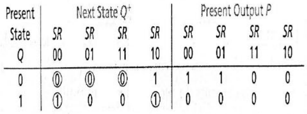
When discussing latches and flip-flops, we use the term present state to denote the state of the Q output of the latch or flip-flop at the time any input signal changes, and the term next state to denote the state of the Q output after the latch or flip-flop has reacted to the input change and stabilized. If we let Q(t) represent the present state and Q(t + ε) represent the next state, an equation for Q(t + ε) can be obtained from the circuit by conceptually breaking the feedback loop at Q and considering Q(t) as an input and Q(t + ε) as the output. Then for the S-R latch;

$$Q\left(t+\varepsilon\right) = R\left(t\right)'\left[S\left(t\right) + Q\left(t\right)\right] = R\left(t\right)'S\left(t\right) + R\left(t\right)'Q\left(t\right) \qquad or\ Q^{+} = R'S + R'Q$$

The equation for output P is;

$$P\left(t\right) = S\left(t\right)'Q\left(t\right)' \qquad or \qquad P = S'Q'$$

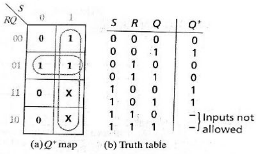
These equations are mapped in the next-state and output tables as given in the following Table. The stable states of the latch are circled. Note that for all stable states, P = Q except when S = R = 1. Making S = R = 1, a don't-care combination allows simplifying the next-state equation.



Present State	Next State Q^+				Present Output P			
Q	SR 00	SR 01	SR 11	SR 10	SR 00	SR 01	SR 11	SR 10
0	0	0	0	1	1	1	0	0
1	1	0	0	1	0	0	0	0

TABLE: S-R Latch Next State and Output

$$Q^{+} = S + R'Q$$

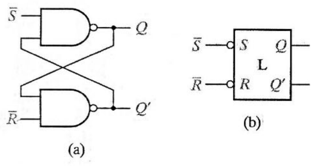


S	R	Q	Q^+
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	–
1	1	1	–

} Inputs not allowed

FIGURE: Derivation of Q+ for an S-R Latch

An equation that expresses the next state of a latch in terms of its present state and inputs will be referred to as a **next-state equation**, or **characteristic equation**.
An alternative form of the S-R latch uses NAND gates, as shown in the following Figure.



$\bar{S}$	$\bar{R}$	Q	Q^+
1	1	0	0
1	1	1	1
1	0	0	0
1	0	1	0
0	1	0	1
0	1	1	1
0	0	0	– } Inputs not allowed
0	0	1	–

(c)

FIGURE: S-R Latch

Applications of S-R Latch: S-R latch is often used as a component in more complex latches and flip-flops and in asynchronous systems. Another useful application of the S-R latch is for ***debouncing switches***.

When a mechanical switch is opened or closed, the switch contacts tend to vibrate or bounce open and closed several times before settling down to their final position. This produces a noisy transition, and this noise can interfere with the proper operation of a logic circuit. The input to the switch in the following Figure is connected to a logic 1 (+ V).

The pull-down resistors connected to contacts *a* and *b* assure that when the switch is between *a* and *b* the latch inputs S and R will always be at a logic *0*, and the latch output will not change state. The timing diagram shows what happens when the switch is flipped from *a* to *b*. As the switch leaves *a*, bounces occur at the R input; when the switch reaches *b*, bounces occur at the S input. After the switch reaches *b*, the first time S becomes 1, after a short delay the latch switches to the Q = 1 state and remains there. Thus Q is free of all bounces even though the switch contacts bounce.

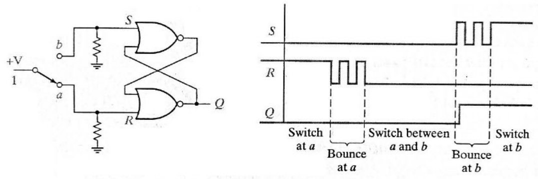


FIGURE : Switch Debouncing with an S-R Latch

4.5 Gated Latches:

Gate latches has an additional input called the gate or enable input. When the gate input is inactive which may be the high or low value, the state of the latch cannot be changed. When the gate input is active the latch is controlled by the other inputs. A NAND-gate version of gated S-R latch shown in the figure below.

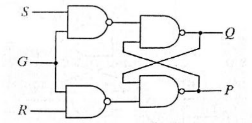


FIGURE : NAND-Gate Gated S-R Latch

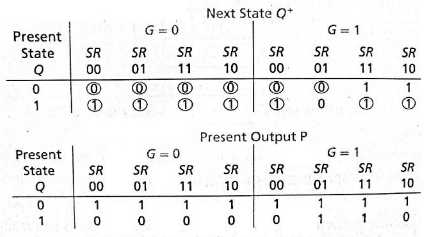
The next-state equation or characteristic equation is

$$Q^{+} = SG + Q\left(R' + G'\right)$$

And the equation for the P output is

$$P = Q' + RG$$

- The next-state and output tables how tables are shown in the figure below.When G=0, the circuit is always in a stable state; when G=1, S=1 sets the latch and R=1 resets the latch.
- Note that $P = Q'$ whenever the latch is in a stable state except for the input combination G=S=R=1; consequently, as for the basic latch, the S=R=1 input combination is disallowed.



Next State Q^{+}

Present State Q	G = 0 SR 00	SR 01	SR 11	SR 10	G = 1 SR 00	SR 01	SR 11	SR 10
0	⓪	⓪	⓪	⓪	⓪	⓪	1	1
1	①	①	①	①	①	0	①	①

Present Output P

Present State Q	G = 0 SR 00	SR 01	SR 11	SR 10	G = 1 SR 00	SR 01	SR 11	SR 10
0	1	1	1	1	1	1	1	1
1	0	0	0	0	0	1	1	0

TABLE: Next-State and Output of Gated S-R Latch

- Another reason for disallowing the S=R=1 input combination is illustrated by considering a change in G from 1 to 0 with S=R=1.
- When G changes, both inputs to the basic S-R latch change from 0 to 1, as shown in figure below.

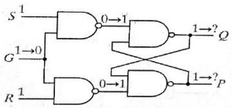


FIGURE: Race Condition in the Gated S-R Latch

- This causes both gates in basic S-R latch to attempt to change from 1 to 0; a race condition exists and the propagation delays of the gates determine whether the latch stabilizes with Q=0 or Q=1. This is illustrated in figure below for the simple NOR-gate latch shown below.

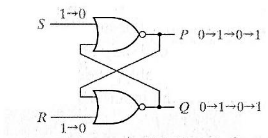
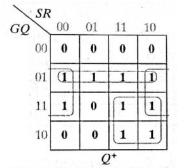


FIGURE: Improper S-R Latch Operation

Let's examine the problem from a different viewpoint.

- If the equation for Q^+ is plotted on a Karnaugh map, it is evident that Q^+ has a static 1-hazard for the input combinations G=1, S=1, R=1, Q=1 and G=0, S=1, R=1, Q=1.
- Consequently, when G changes from 1 to 0 between these two input combinations, it is possible for Q to change from 1 to 0 and, because of the feedback, to cause Q to remain at Q=0.
- This is simply a different interpretation of the race condition described above.



GQ \ SR	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	0	1	1
10	0	0	1	1

Q^+

FIGURE : Karnaugh Map for Q+

There is another restriction regarding gated S-R latches.

- The S and R inputs must not be changing or contain glitches while G=1.For example, assume Q=0 when G changes from 0 to 1 and S=0.
- If S and R remain at 0 until G returns to 0, then Q remains at 0. However, if S contains a 1 glitch, maybe due to a static 1-hazard in its circuit, then Q may be forced to a 1 and will remain there after G becomes 0.
- A similar problem occurs if S does not change from 1 to 0 until after G changes to 1.
- This is referred to as the 1's catching problem. A NOR-gate version of the gated S-R latch has a 0's catching problem.

4.6 GATED D LATCH:

A gated D latch (given in Figure below) has two inputs—a data input (D) and a gate input (G). The D latch can be constructed from an S-R latch and gates. When G = 0, S = R = 0, so Q does not change.
When G = 1 and D = 1, S = 1 and R = 0, so Q is set to 1. When G = 1 and D = 0, S = 0 and R = 1, so Q is reset to 0.In other words, when G = 1, the Q output follows the D input, and when G = 0, the Q output holds the last value of D (no state change). This type of latch is also referred to as a transparent latch because when G = 1, the Q output is the same as the D input. From the truth table, the characteristic equation for the latch is $Q+ = G'\,Q + GD$.

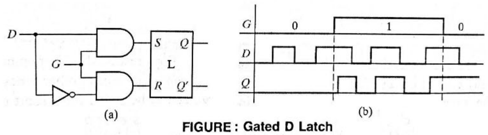
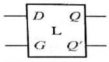
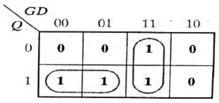


FIGURE : Gated D Latch



G D Q	Q^+
0 0 0	0
0 0 1	1
0 1 0	0
0 1 1	1
1 0 0	0
1 0 1	0
1 1 0	1
1 1 1	1



$Q^+ = G'Q + GD$

FIGURE: Symbol and Truth Table for Gated Latch

- Most digital system use a clock signal to synchronize the change in outputs of the system's flip-flop to an edge of the clock signal, either the positive (0 to 1) or the negative (1 to 0) edge of the clock.
- It is tempting to think that gated latches could be used as flip-flop where the clock signal is connected to the gate inputs of the latches. However, this is not a practical approach. The following example illustrates the difficulty.
- In the circuit of figure shown below, it seems that when the Clk is 1, the next value of Q should be Q′ when the input x=1, and should be Q when x=0.

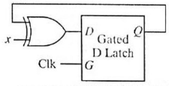


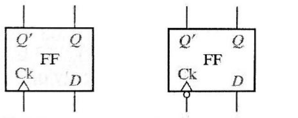
FIGURE: Unreliable Gated D Latch Circuit

- However, when Clk=1 and x=1, D= Q′ causes Q to change and, if Clk remains 1, the change in Q will feed back and cause Q to change again.
- If Clk remains at 1, Q will oscillate. Consequently, the circuit will only operate as intended if Clk remains at 1 for short time; it has to 1 just long enough to allow Q to change but short enough to prevent the change from feeding back and causing a second change.
- With a single latch, it may be possible to control the clock high time so the latch operates as intended, but in a system with several latches, the variation in gate delays would make it impossible to provide the correct clock width to all latches.
- To avoid this timing problem, more complicated flip-flop restrict the flip-flop outputs to only change on an edge of the clock, and the outputs cannot change at other times even if the inputs change.
- If the inputs to the flip-flop only need to be stable for a short period of time around the clock edge, then we refer to the flip-flop as ***edge-triggered***.
- The term ***master-slave*** flip-flops refers to a particular implementation that uses two gated latches in such a way that the flip-flop outputs only change on clock edge.
- The ***master-slave*** flip-flops are not necessarily edge-triggered flip-flops because they may require the flip-flop inputs to be stable at times during the clock period other than just around the clock edge.

4.7 EDGE-TRIGGERED D FLIP-FLOP:

A D flip-flop has two inputs, D (data) and Clk (clock). The small arrowhead on the flip-flop symbol identifies the clock input. Unlike the D latch, the flip-flop output changes only in response to the clock, not to a change in D.

- If the output can change in response to a 0 to 1 transition on the clock input, we say that the flip flop is triggered on the rising edge (or positive edge) of the clock.
- If the output can change in response to a 1 to 0 transition on the clock input, we say that the flip flop is triggered on the falling edge (or negative edge) of the clock.
- An inversion bubble on the clock input indicates a falling-edge trigger (Figure (b)), and no bubble indicates a rising-edge trigger (Figure (a)).
- The term *active edge* refers to the clock edge (rising or falling) that triggers the flip-flop state change.



(a) Rising-edge trigger (b) Falling-edge trigger

D Q	Q^+
0 0	0
0 1	0
1 0	1
1 1	1

$\boxed{Q^+ = D}$

(c) Truth table

FIGURE: D Flip-Flops

Since, the Q output of the flip-flop is the same as the D input, except that the output changes are delayed until after the active edge of the clock pulse, as illustrated in the following.

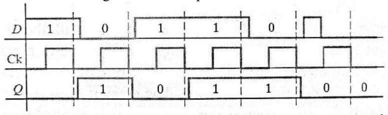
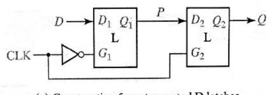
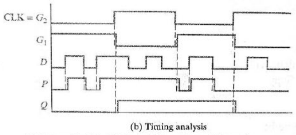


FIGURE : Timing for D Flip-Flop (Falling-Edge Trigger)

A rising-edge-triggered D flip-flop can be constructed from two gated D latches and an inverter, as shown in Figure the following Figure (a). The timing diagram is shown in Figure (b).



(a) Construction from two gated D latches



(b) Timing analysis

FIGURE: D Flip-Flop (Rising- Edge Trigger)

When CLK = 0, G1 = 1, and the first latch is transparent so that the P output follows the D input. Because G2 = 0, the second latch holds the current value of Q. When CLK changes to 1, G1 changes to 0, and the current value of D is stored in the first latch. Because G2 = 1, the value of P flows through the second latch to the Q output. When CLK changes back to 0, the second latch takes on the value of P and holds it and, then, the first latch starts following the D input again. If the first latch starts following the D input before the second latch takes on the value of P, the flip-flop will not function properly. Therefore, the circuit designers must pay careful attention to timing issues when designing edge-triggered flip-flops. With this circuit, output state changes occur only following the rising edge of the clock. The value of D at the time of the rising edge of the clock determines the value of Q, and any extra changes in D that occur between rising clock edges have no effect on Q.

A flip-flop changes state only on the active edge of the clock, the propagation delay of a flip-flop is the time between the active edge of the clock and the resulting change in the output. However, there are also timing issues associated with the D input.

To function properly, the D input to an edge-triggered flip-flop must be held at a constant value for a period of time before and after the active edge of the clock. If D changes at the same time as the active edge, the behavior is unpredictable.

The amount of time that the D input must be stable before the active edge is called the ***setup time (t_{su})***, and the amount of time that the D input must hold the same value after the active edge is the ***hold time (t_h)***. The times at which D is allowed to change during the clock cycle are shaded in the timing diagram of the following Figure.

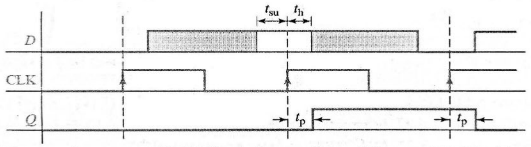


FIGURE: Setup and Hold Times for an Edge-Triggered D Flip-Flop

The propagation delay (tp) from the time the clock changes until the Q output changes is also indicated in the above Figure.

Using these timing parameters, we can determine the minimum clock period for a circuit which will not violate the timing constraints. Consider the circuit of following Figure (a). Suppose the inverter has a propagation delay of 2 ns, and suppose the flip-flop has a propagation delay of 5 ns and a setup time of 3ns. (The hold time does not affect this calculation). Suppose, as in following Figure (b), that the clock period is 9 ns, i.e., 9 ns is the time between successive active edges (rising edges for this figure). Then, 5ns after a clock edge, the flip-flop output will change, and 2 ns after that, the output of the inverter will change. Therefore, the input to the flip-flop will change 7 ns after the rising edge, which is 2 ns before the next rising edge. But the setup time of the flip-flop requires that the input be stable 3 ns before the rising edge; therefore, the flip-flop may not take on the correct value.

Suppose instead that the clock period were 15 ns, as in following Figure (c). Again, the input to the flipflop will change 7 ns after the rising edge. However, because the clock is slower, this is 8 ns before the next rising edge. Therefore, the flip-flop will work properly. Note in Figure (c) that there is 5 ns of extra time between the time the D input is correct and the time when it must be correct for the setup time to be satisfied. Therefore, we can use a shorter clock period, and have less extra time, or no extra time. Figure (d) shows that 10 ns is the minimum clock period which will work for this circuit.

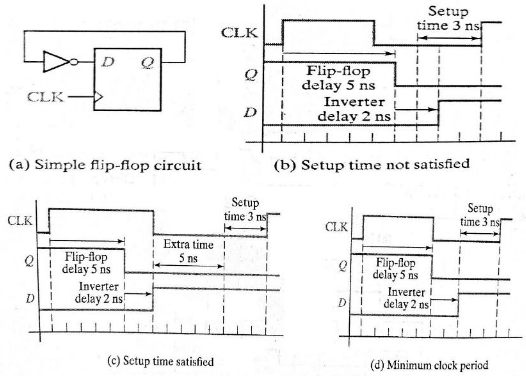
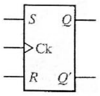


FIGURE: Determination of Minimum Clock Period

4.8 S-R FLIP-FLOP:

An S-R flip-flop (following Figure) is similar to an S-R latch in that S = 1 sets the Q output to 1, and R = 1 resets the Q output to 0. The essential difference is that the flip-flop has a clock input, and the Q output can change only after an active clock edge.



Operation summary:

$S = R = 0$	No state change
$S = 1, R = 0$	Set Q to 1 (after active Ck edge)
$S = 0, R = 1$	Reset Q to 0 (after active Ck edge)
$S = R = 1$	Not allowed

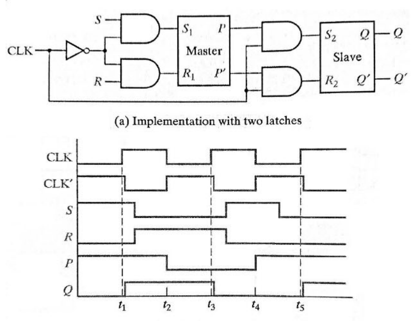
FIGURE: S-R Flip-Flop

The truth table and characteristic equation for the flip-flop are the same as for the latch, but the Interpretation of Q+ is different. For the latch, Q+ is the value of Q after the propagation delay through the latch, while for the flip-flop, Q+ is the value that Q assumes after the active clock edge.

The following Figure (a) shows an S-R flip-flop constructed from two S-R latches and gates. This flip-flop changes state after the rising edge of the clock. The circuit is often referred to as a master-slave flip-flop.

When CLK = 0, the S and R inputs set the outputs of the master latch to the appropriate value while the slave latch holds the previous value of Q. When the clock changes from 0 to 1, the value of P is held in the master latch and this value is transferred to the slave latch. The master latch holds the value of P while CLK = 1, and, hence, Q does not change. When the clock changes from 1 to 0, the Q value is latched in the slave, and the master can process new inputs.

Figure (b) shows the timing diagram. Initially, S = 1 and Q changes to 1 at t1. Then R = 1 and Q changes to 0 at t3.



(a) Implementation with two latches

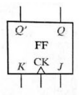
(b) Timing analysis

FIGURE: S-R Flip-Flop Implementation and Timing

For a rising-edge-triggered flip-flop, the value of the inputs is sensed at the rising edge of the clock, and the inputs can change while the clock is low. For the master-slave flip-flop, if the inputs change while the clock is low, the flip-flop output may be incorrect. For example, (in above Figure (b)), at t4, S = 1 and R = 0, so P changes to 1. Then S changes to 0 at t5, but P does not change, so at t5, Q changes to 1 after the rising edge of CLK. However, at t5, S = R = 0, so the state of Q should not change. We can solve this problem if we only allow the S and R inputs to change while the clock is high.

4.9 J-K FLIP-FLOP:

The J-K flip-flop shown in figure below is an extended version of the S-R flip-flop. The J-K flip-flop has three inputs—J, K, and the clock (CLK). The J input corresponds to S, and K corresponds to R. That is, if J = 1 and K = 0, the flip-flop output is set to Q = 1 after the active clock edge; and if K = 1 and J = 0, the flip-flop output is reset to Q = 0 after the active edge.

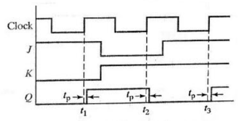


(a) J-K flip-flop

J K Q	Q^+
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

$$\boxed{Q^+ = JQ' + K'Q}$$

(b) Truth table and characteristic equation



(c) J-K flip-flop timing

FIGURE: J-K Flip-Flop (Q Changes on the Rising Edge)

Unlike the S-R flip-flop, a 1 input may be applied simultaneously to J and K, in which case the flip-flop changes state after the active clock edge. When J = K = 1, the active edge will cause Q to change from 0 to 1, or from 1 to 0. The next-state table and characteristic equation for the J-K flip-flop are given in Figure (b).

Figure (c) shows the timing for a J-K flip-flop. This flip-flop changes state a short time (tp) after the rising edge of the clock pulse, provided that J and K have appropriate values. If J = 1 and K = 0 when Clock = 0, Q will be set to 1 following the rising edge. If K = 1 and J = 0 when Clock = 0, Q will be set to 0 after the rising edge.

Similarly, if J = K = 1, Q will change state after the rising edge. Referring to Figure 11-20(c), because Q = 0, J = l, and K = 0 before the first rising clock edge, Q changes to 1 at t1. Because Q = 1, J = 0, and K = 1 before the second rising clock edge, Q changes to 0 at t2. Because Q = 0, J = 1, and K = 1 before the third rising clock edge, Q changes to 1 at t3. One way to realize the J-K flip-flop is with two S-R latches connected in a master-slave arrangement, as shown in the following Figure.

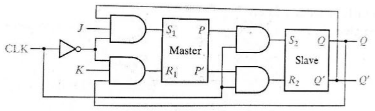
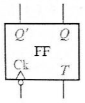


FIGURE : Master-Slave J-K Flip-Flop (Q Changes on Rising Edge)

This is the same circuit as for the S-R master-slave flip-flop; except S and R have been replaced with J and K, and the Q and Q outputs are feeding back into the input gates. Because $S = J\,Q'CLK'$ and $R = K'Q\,CLK'$, only one of S and R inputs to the first latch can be 1 at any given time. If Q = 0 and J = 1, then S = 1 and R = 0, regardless of the value of K. If Q = 1 and K = 1, then S = 0 and R = 1, regardless of the value of J.

<u>4.10 T FLIP-FLOP:</u>

The T flip-flop, also called the toggle flip-flop, is frequently used in building counters. Most CPLDs and FPGAs can be programmed to implement T flip-flops.



(a)

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

$$\boxed{Q^+ = T'Q + TQ' = T \oplus Q}$$

(b)

FIGURE: T Flip-Flop

The T flip-flop shown in above figure (a) has a T input and a clock input. When T = 1 the flip-flop changes state after the active edge of the clock. When T = 0, no state change occurs. The next-state table and characteristic equation for the T flip-flop are given in Figure (b). The characteristic equation states that the next state of the flip-flop (Q+) will be 1, if the present state (Q) is 1 and T = 0 or the present state is 0 and T = 1.

The following Figure shows a timing diagram for the T flip-flop. At times t2 and t4 the T input is 1 and the flip-flop state (Q) changes a short time (tp) after the falling edge of the clock pulse. At times tl and t3 the T input is 0, and the clock edge does not cause a change of state.

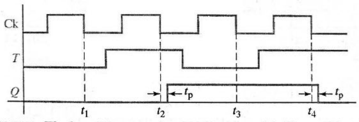
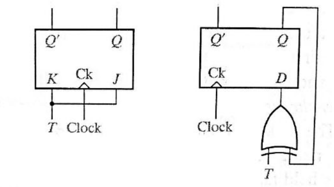


FIGURE: Timing Diagram for T Flip-Flop (Falling- Edge Trigger)

One way to implement a T flip-flop is to connect the J and K inputs of a J-K flip-flop together, as shown in the following Figure (a). Substituting T for J and K in the J-K characteristic equation gives;

$$Q^+ = JQ' + K'\,Q = TQ' + T'Q$$

which is the characteristic equation for the T flip-flop. Another way to realize a T flip-flop is with a D flip-flop and an exclusive-OR gate [Figure (b)].
The D input is Q ⊕ T, so Q^+ = Q ⊕ TQ′ + T′Q, which is the characteristic equation for the T flip-flop.



(a) Conversion of J-K to T (b) Conversion of D to T

FIGURE: Implementation of T Flip-Flops

Characteristic Equations of Flip-Flop:

The characteristics equations of flip-flops are useful in analysing circuits made of them. Here, next output, Qn+1, is expressed as a function of present output Qn and the input to the flip-flops. Karnaugh map can be used to get the optimized expression.

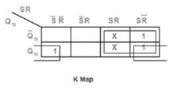
S	R	Q^+
0	0	Q
0	1	0
1	0	1
1	1	?

D	Q	Q^+
0	0	0
0	1	0
1	0	1
1	1	1

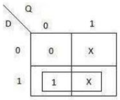
J	K	Q^+
0	0	Q
0	1	0
1	0	1
1	1	$\bar{Q}$

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

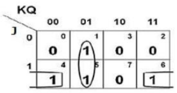
SR Flip-Flop:



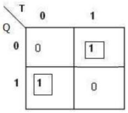
D Flip-Flop:



JK Flip-Flop:



T Flip-Flop:



$Q^+ = S + R'Q \ (SR = 0)$	(S-R latch or flip-flop)
$Q^+ = GD + G'Q$	(gated D latch)
$Q^+ = D$	(D flip-flop)
$Q^+ = D{\cdot}CE + Q{\cdot}CE'$	(D-CE flip-flop)
$Q^+ = JQ' + K'Q$	(J-K flip-flop)
$Q^+ = T \oplus Q = TQ' + T'Q$	(T flip-flop)