



CANARA ENGINEERING COLLEGE

Benjanapadavu, Bantwal Taluk - 574219

Department of Computer Science & Engineering



VISION

To be recognized as a center of knowledge dissemination in Computer Science and Engineering by imparting value-added education to transform budding minds into competent computer professionals.

MISSION

- M1.** Provide a learning environment enriched with ethics that helps in enhancing problem solving skills of students and, cater to the needs of the society and industry.
- M2.** Expose the students to cutting-edge technologies and state-of-the-art tools in the many areas of Computer Science & Engineering.
- M3.** Create opportunities for all round development of students through co-curricular and extra-curricular activities.
- M4.** Promote research, innovation and development activities among staff and students.

PROGRAMME EDUCATIONAL OBJECTIVES

- A. Graduates will work productively as computer science engineers exhibiting ethical qualities and leadership roles in multidisciplinary teams.
- B. Graduates will adapt to the changing technologies, tools and societal requirements.
- C. Graduates will design and deploy software that meets the needs of individuals and the industries
- D. Graduates will take up higher education and/or be associated with the field so that they can keep themselves abreast of Research & Development

PROGRAMME OUTCOMES

Engineering graduates in Computer Science and Engineering will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specific needs with appropriate consideration for the public health and safety, and the cultural, societal and environmental considerations.

4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods, including design of experiments, analysis and interpretation of data and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Select/Create and apply appropriate techniques, resources and modern engineering and IT tools, including prediction and modeling to complex engineering activities, taking comprehensive cognizance of their limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and Sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts and demonstrate the knowledge of and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the relevant scientific and/or engineering practices.
9. **Individual and team work:** Function effectively as an individual and as a member or leader in diverse teams and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with the society-at-large, such as being able to comprehend and write effective reports and design documentation, make effective presentations and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work as a member and leader in a team to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for and above have the preparation and ability to engage in independent and life-long learning in the broadcast context of technological changes.

PROGRAMME SPECIFIC OUTCOMES

1. **Computer System Components:** Apply the principles of computer system architecture and software to design, develop and deploy computer subsystem.
2. **Data Driven and Internet Applications:** Apply the knowledge of data storage, analytics and network architecture in designing Internet based applications.

Analog & Digital Electronics [As per Choice Based Credit System (CBCS) scheme] (Effective from the academic year 2018 - 2019) SEMESTER – III			
Subject Code	18CS33	IA Marks	40
Number of Lecture Hours/Week	3:0:0	Exam Marks	60
Total Number of Lecture Hours	50	Exam Hours	03
CREDITS – 03			
Module – 1			Teaching Hours
Photodiodes, Light Emitting Diodes and Optocouplers, BJT Biasing :Fixed bias ,Collector to base Bias , voltage divider bias, Operational Amplifier Application Circuits: Multivibrators using IC-555, Peak Detector, Schmitt trigger, Active Filters, Non-Linear Amplifier, Relaxation Oscillator, Current-to-Voltage and Voltage-to-Current Converter , Regulated Power Supply Parameters, adjustable voltage regulator ,D to A and A to D converter. Text Book 1 :Part A:Chapter 2(Section 2.9,2.10,2.11), Chapter 4(Section 4.2,4.3,4.4),Chapter 7 (section (7.2,7.3.1,7.4,7.6 to 7.11), Chapter 8 (section (8.1,8.5), Chapter 9 RBT: L1, L2			8 Hours
Module – 2			
Karnaugh maps: minimum forms of switching functions, two and three variable Karnaugh maps, four variable karnaugh maps, determination of minimum expressions using essential prime implicants, Quine-McClusky Method: determination of prime implicants, The prime implicant chart, petricks method, simplification of incompletely specified functions, simplification using map-entered variables Text book 1:Part B: Chapter 5 (Sections 5.1 to 5.4) Chapter 6(Sections 6.1 to 6.5) RBT: L1, L2			8 Hours
Module – 3			
Combinational circuit design and simulation using gates: Review of Combinational circuit design, design of circuits with limited Gate Fan-in ,Gate delays and Timing diagrams, Hazards in combinational Logic, simulation and testing of logic circuits Multiplexers, Decoders and Programmable Logic Devices: Multiplexers, three state buffers, decoders and encoders, Programmable Logic devices, Programmable Logic Arrays, Programmable Array Logic. Text book 1:Part B: Chapter 8,Chapter 9 (Sections 9.1 to 9.6) RBT: L1, L2			8 Hours
Module – 4			
Introduction to VHDL: VHDL description of combinational circuits, VHDL Models for 08 multiplexers, VHDL Modules. Latches and Flip-Flops: Set Reset Latch, Gated Latches, Edge-Triggered D Flip Flop 3,SR Flip Flop, J K Flip Flop, T Flip Flop, Flip Flop with additional inputs, Asynchronous Sequential Circuits Text book 1:Part B: Chapter 10(Sections 10.1 to 10.3),Chapter 11 (Sections 11.1 to 11.9) RBT: L1, L2			8 Hours
Module – 5			
Registers and Counters: Registers and Register Transfers, Parallel Adder with accumulator, shift registers, design of Binary counters, counters for other sequences, counter design using SR and J K Flip Flops, sequential parity checker, state tables and graphs Text book 1:Part B: Chapter 12(Sections 12.1 to 12.5),Chapter 13(Sections 13.1,13.3 RBT: L1, L2			8 Hours
Course outcomes: The students should be able to:			

<ul style="list-style-type: none"> • Design and analyze application of analog circuits using photo devices, timer IC, power supply and regulator IC and op-amp. • Explain the basic principles of A/D and D/A conversion circuits and develop the same. • Simplify digital circuits using Karnaugh Map, and Quine-McClusky Methods • Explain Gates and flip flops and make us in designing different data processing circuits, registers and counters and compare the types. • Develop simple HDL programs
<ul style="list-style-type: none"> • Question paper pattern: • The question paper will have ten questions. • Each full Question consisting of 20 marks • There will be 2 full questions (with a maximum of four sub questions) from each module. • Each full question will have sub questions covering all the topics under a module. • The students will have to answer 5 full questions, selecting one full question from each module
TextBooks:
1. Charles H Roth and Larry L Kinney, Analog and Digital Electronics, Cengage Learning, 2019
Reference Books:
1. Anil K Maini, Varsha Agarwal, Electronic Devices and Circuits, Wiley, 2012. 2. Donald P Leach, Albert Paul Malvino & Goutam Saha, Digital Principles and Applications, 8th Edition, Tata McGraw Hill, 2015. 3. M. Morris Mani, Digital Design, 4th Edition, Pearson Prentice Hall, 2008. 4. David A. Bell, Electronic Devices and Circuits, 5th Edition, Oxford University Press, 2008

COURSE OBJECTIVES:

1	Explain the use of photoelectronics devices, 555 timer IC, Regulator ICs and uA741 opamp IC
2	Make use of simplifying techniques in the design of combinational circuits.
3	Illustrate combinational and sequential digital circuits
4	Demonstrate the use of flip flops and apply the same for registers
5	Design and test counters, Analog-to-Digital and Digital-to-Analog conversion

COURSE OUTCOMES (COs):

SL. NO	DESCRIPTION
	After Completing thus course, the students will be able to:
CO:1	Design and analyse application of analog circuits using photo devices, timer IC, power supply and regulator IC and op-amp.
CO:2	Simplify digital circuits using Karnaugh Map, and Quine-McCluskey Methods
CO:3	Designing different data processing circuits
CO:4	Develop simple HDL programs and Explain Gates and flip flops
CO:5	Explain and develop Registers & Counters

Module-4

Table of contents

Chapter		Topic	Page No.
1	1.1	Introduction to VHDL	1
	1.2	VHDL description of combinational circuits	1
	1.2.1	Dataflow description	2
	1.2.2	Behavioural description	3
	1.2.3	Structural description	3
	1.2.4	Inertial delay model	5
	1.2.5	Ideal (Transport) delay	6
	1.2.6	VHDL models for multiplexers:	6
	1.3	VHDL modules	8
2	2.1	Introduction to latches	12
	2.2.1	Set Reset latch	12
	2.2.1.1.	Applications of S-R Latch	15
	2.3	Gated d latch:	16
	2.4	Edge-triggered d flip-flop	17
	2.5	S-R flip-flop	20
	2.6	J-K flip-flop	22
	2.7	T flip-flop	23
	2.8	Flip-flops with additional inputs	25
	2.9	Characteristic Equations of Flip-Flop	27
	2.10	Flip-Flops as Finite State Machine	28
		Exercise problems	29
		Web resources	31
		Video resources	31
		Question bank	32
		University questions	35

MODULE – 4

VHDL, LATCHES AND FLIP-FLOPS

CHAPTER 1

1.1 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.

1.2 VHDL DESCRIPTION OF COMBINATIONAL CIRCUITS

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* like 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.

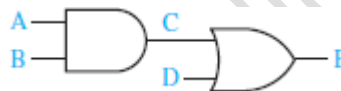


Fig 1.1 Gate Circuit

1.2.1 Dataflow Description: The two assignment statements (given below) give a dataflow description of the above circuit, where it is assumed that each gate has a 5-ns propagation delay. When these statements are simulated, the first statement will be evaluated any time A or B changes, and the second statement will be evaluated any time C or D changes. Suppose that initially A = 1, and B = C = D = E = 0; and if B changes to 1 at time 0, C will change to 1 at time = 5 ns. Then, E will change to 1 at time = 10 ns.

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

VHDL signal assignment statements (as given above) are *concurrent statements*. The VHDL simulator monitors the right side of each 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 after an appropriate delay. This is exactly the way the hardware works. Any time a gate input changes, the gate output is recomputed by the hardware, and the output changes after the gate delay. Unlike a sequential program, the order of the above concurrent statements is unimportant.

1.2.2 Behavioral Description: A behavioral description of the above circuit shown is

$$E \leq D \text{ or } (A \text{ and } B);$$

Parentheses are used to specify the order of operator execution.

1.2.3 Structural Description: The above circuit shown can also be described using structural VHDL code. To do so requires that a two-input AND-gate component and a 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 a separate instantiation statement to specify how it is connected to other components and to the port inputs and outputs. An instantiation statement is a concurrent statement that executes anytime one of the input signals in its port map changes. The circuit shown is described by instantiating the AND gate and the OR gate as follows:

Gate1: AND2 port map (A, B, D); 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 D. Similarly, the second statement passes changes in C or D to the Gate 2 inputs, and then the component computes a new value of E. This is exactly how the real hardware works. (The order in which the instantiation statements appear is irrelevant).

Instantiating a 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 a new output value whenever its input changes.

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.

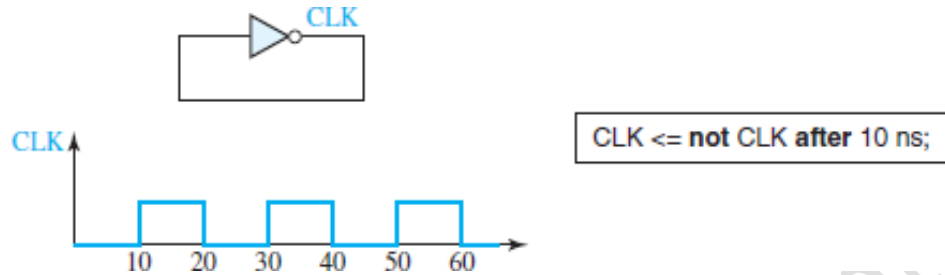


Fig 1.2

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 $5 + \Delta$.

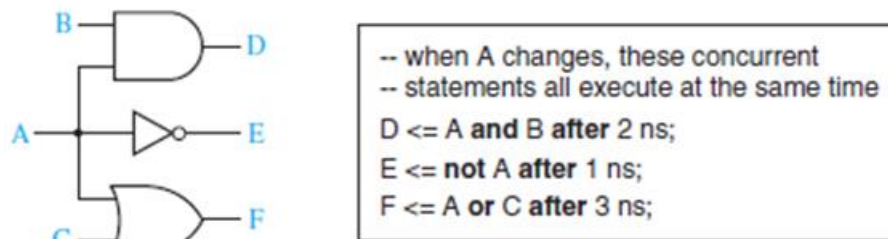


Fig 1.3

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 $B \leq "0110"$, 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.

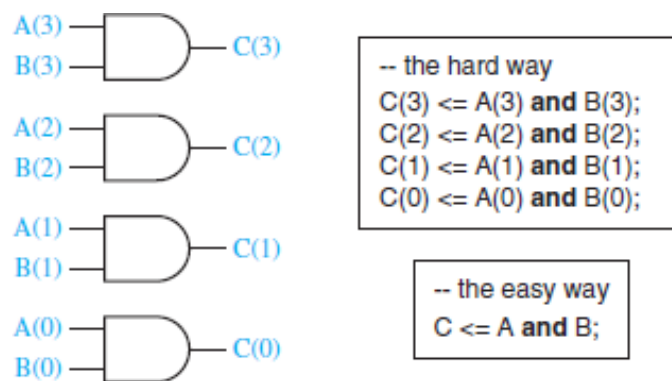


Fig 1.4 Array of 4 AND Gates

1.2.4 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 $C \leq A \text{ and } B \text{ after } 10 \text{ ns};$

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.

1.2.5 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 $C \leq \text{transport } A \text{ and } B \text{ after } 10 \text{ ns};$ 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.

1.2.6 VHDL MODELS FOR MULTIPLEXERS:

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

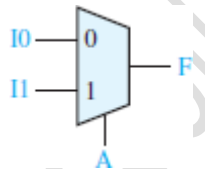


Fig 1.5. 2:1 MUX

The MUX output is $F = A' I_0 + A I_1$. The corresponding VHDL statement is

$F \leq (\text{not } A \text{ and } I_0) \text{ or } (A \text{ and } I_1);$

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

$F \leq I_0 \text{ when } A = „0” \text{ else } I_1;$

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 a conditional signal assignment statement is

**signal_name <= expression1 when condition1
else expression2 when condition2
[else expressionN];**

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.

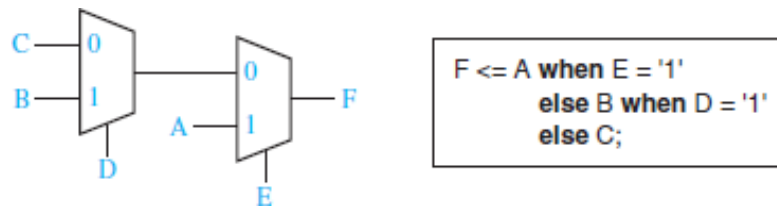


Fig. 1.6. 4:1 MUX

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 $F = A'B'I_0 + A'BI_1 + AB'I_2 + ABI_3$.

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):

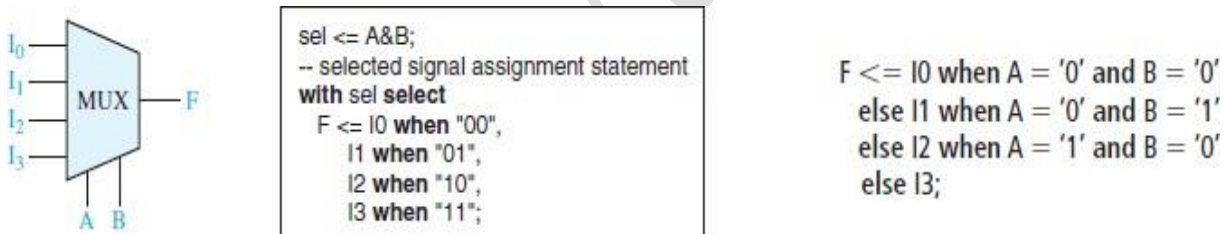


Fig. 1.6. 4:1 MUX

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.

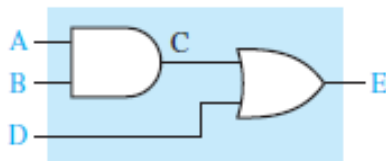
The general form of a selected signal assignment statement is

```
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.

1.3 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;
```

Fig. 1.7

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.

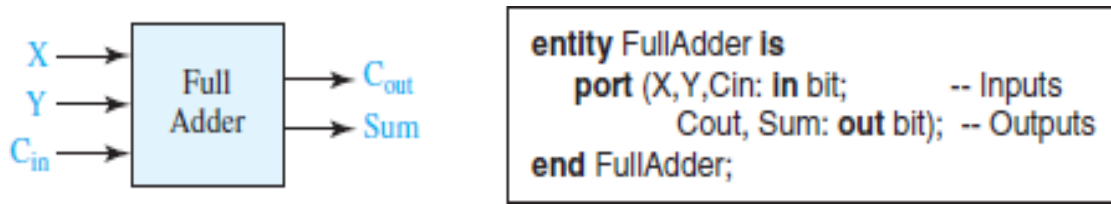


Fig. 1.8

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

```
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];
```

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] };

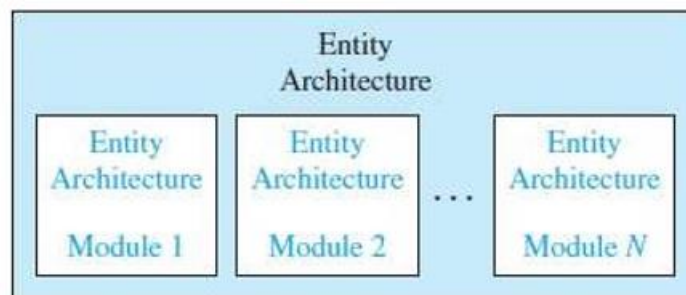


Fig. 1.9. VHDL Program Structure

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 (see the following Figure).

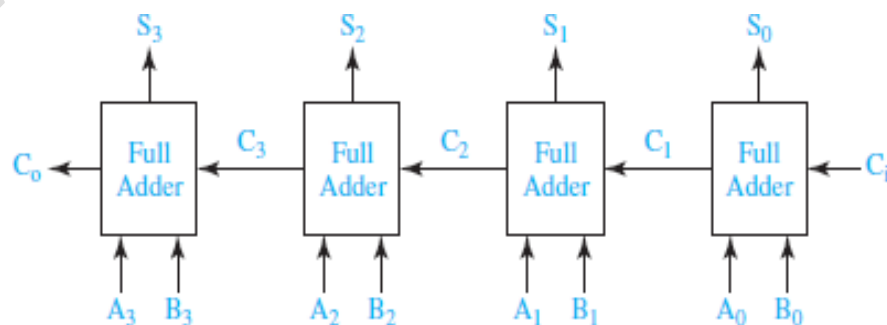


Fig.1.10. 4-Bit Full Adder

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 down to 0.

Next, specify the Full Adder 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 Full Adder component. Each copy of Full Adder 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.

```
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;
```


LATCHES & FLIP-FLOPS

CHAPTER 2

2.1 INTRODUCTION TO LATCHES

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 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

2.2.1 SET RESET LATCH

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

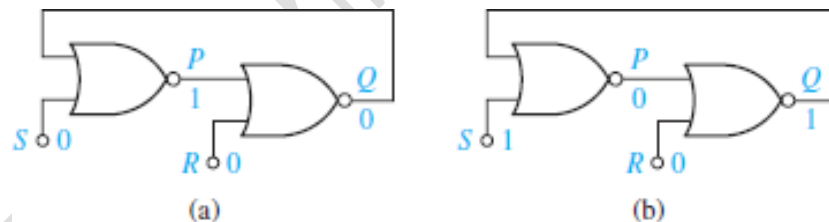


Fig 1.11 Latch using NOR gate

- (a) $S = 0$ & $R = 0$: 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.
- (b) $S = 1$ & $R = 0$: 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.

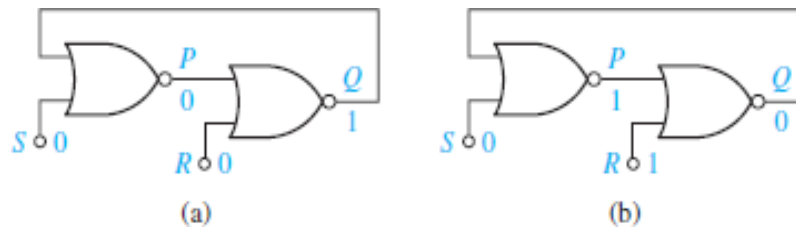


Fig 1.12. S-R Latch

- (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 $S = 1$ sets the output to $Q = 1$, and an input $R = 1$ resets 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 operation of the two gates, the circuit is often drawn in cross-coupled form, as shown in the following Figure (a).

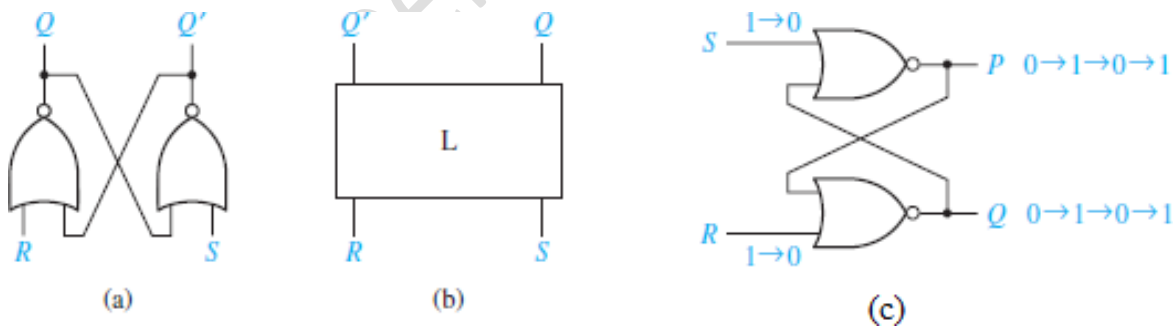


Fig 1.13. Different representations of S-R Latch

If $S = R = 1$, the latch will not operate properly, as shown in above Figure (c). Note that, when S and R are both 1, 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 t_1 , Q changes to 1 a short time (ϵ - response time or delay time of latch) later. At time t_2 , when S changes back to 0, Q does not change. At time t_3 , 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.

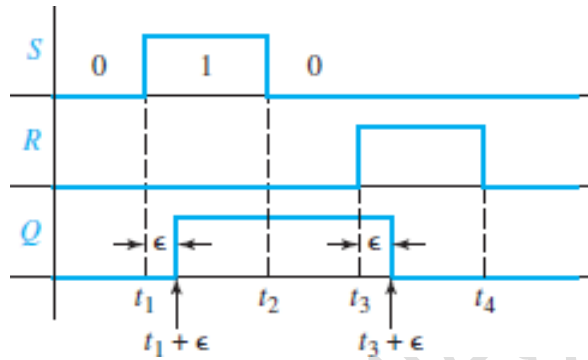


Fig. 1.14. Timing Diagram of S-R Latch

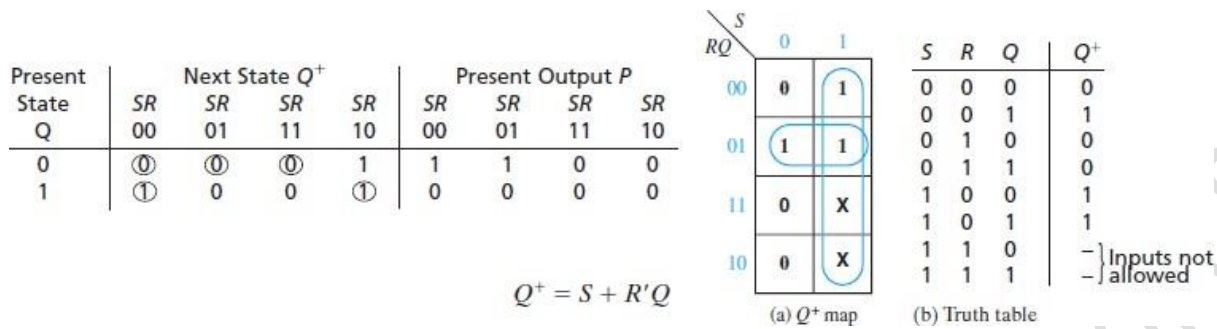
When discussing latches and flip-flops, we use the term present state to denote the state of the Q output of the latch or flip-flop at the time any input signal changes, and the term next state to denote the state of the Q output after the latch or flip-flop has reacted to the input change and stabilized.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

$$Q(t + \epsilon) = R(t)'[S(t) + Q(t)] = R(t)'S(t) + R(t)'Q(t) \quad \text{or} \quad Q^+ = R'S + R'Q$$

The equation for output P is;

$$P(t) = S(t)'Q(t)' \quad \text{or} \quad P = S'Q'$$

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.



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.

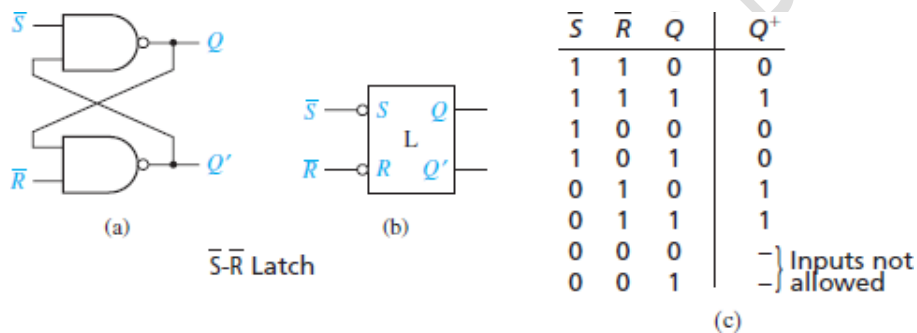


Fig.1.15 S-R Latch

2.2.1.1. 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.

2.3 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

$$Q^+ = G'Q + GD.$$

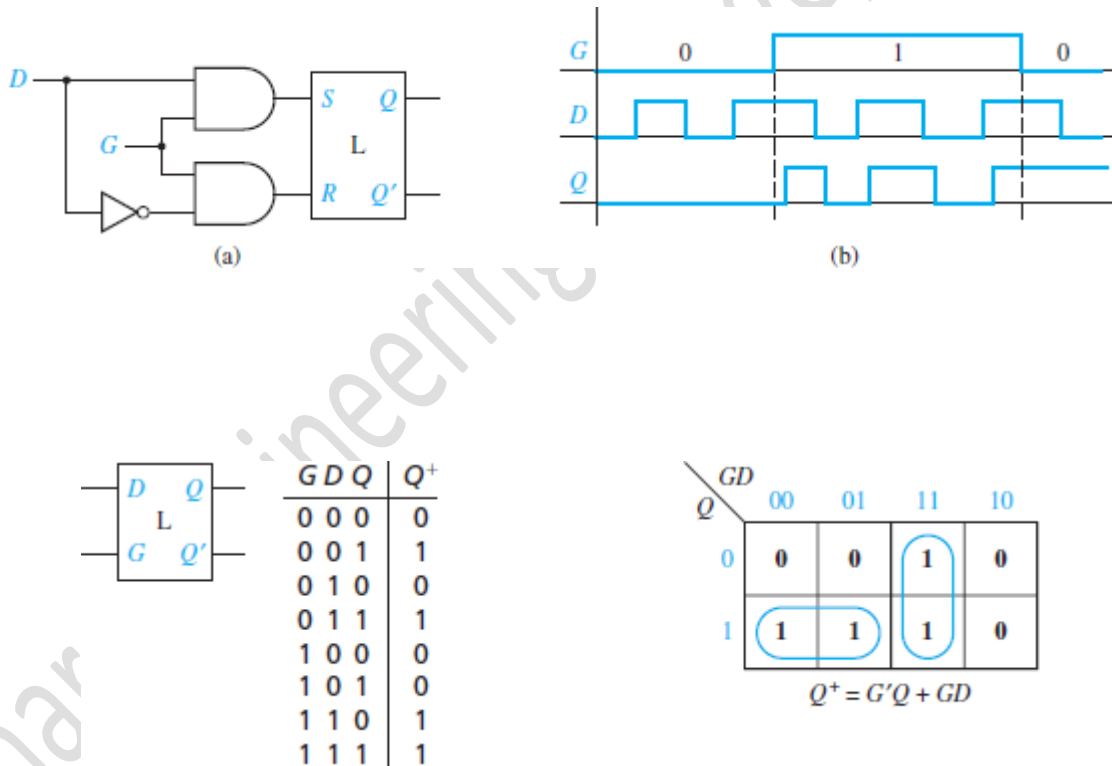


Fig.1.16. Gated D-Latch

2.4 EDGE-TRIGGERED D FLIP-FLOP

A D flip-flop has two inputs, D (data) and Ck (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.

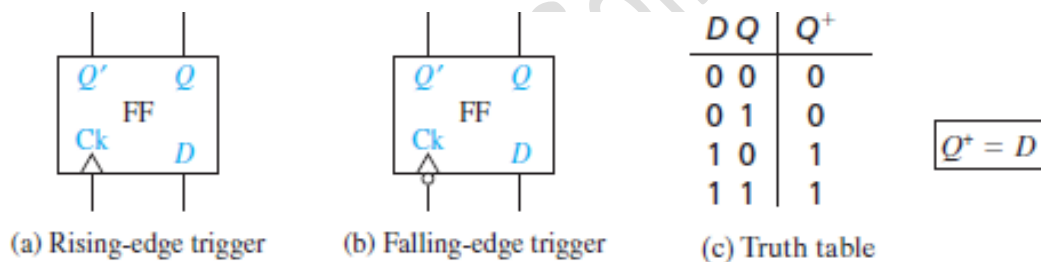


Fig. 1.17. D-Flipflops

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.

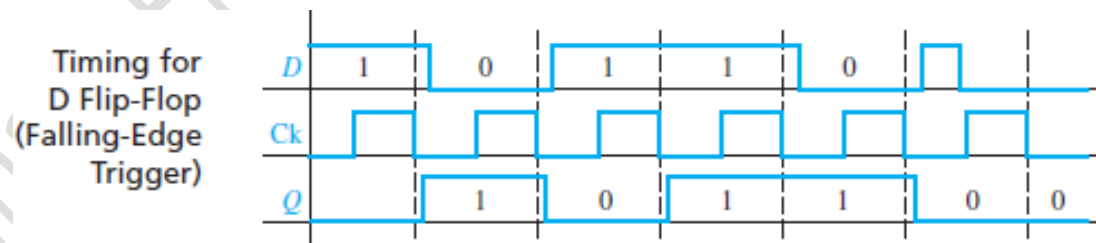


Fig.1.18

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).

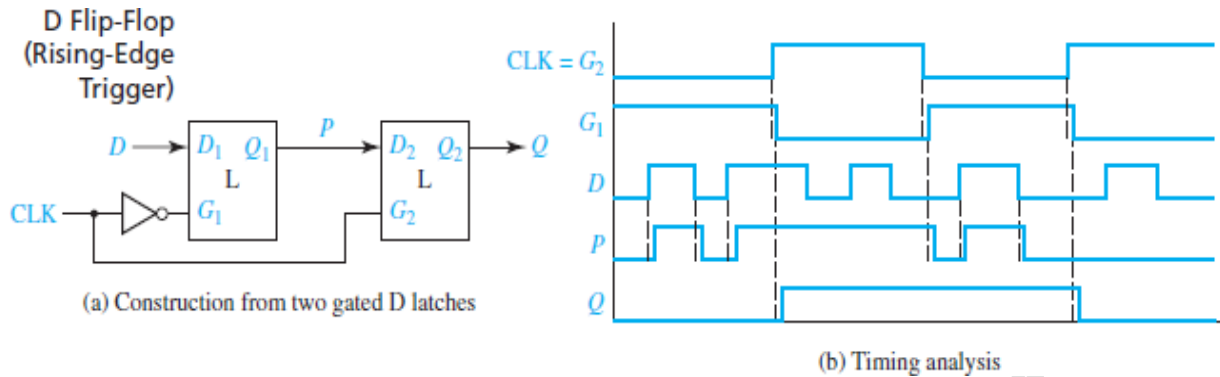


Fig.1.19

When $CLK = 0$, $G_1 = 1$, and the first latch is transparent so that the P output follows the D input. Because $G_2 = 0$, the second latch holds the current value of Q . When CLK changes to 1, G_1 changes to 0, and the current value of D is stored in the first latch. Because $G_2 = 1$, 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.

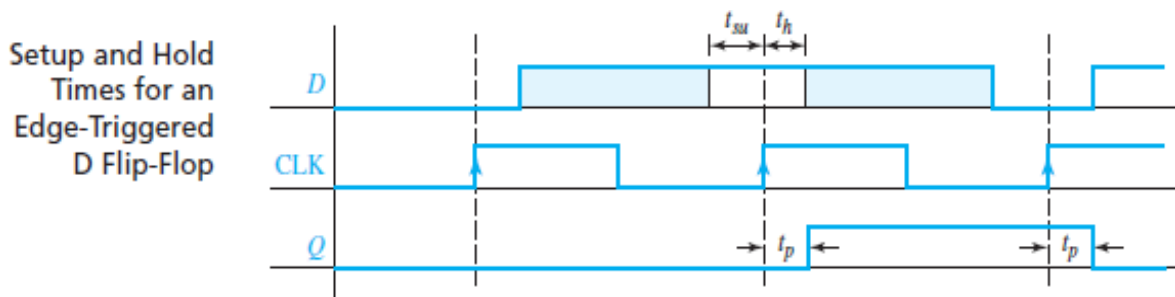


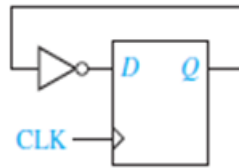
Fig. 1.20

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

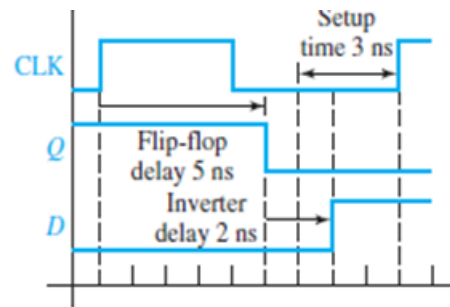
Using these timing parameters, we can determine the minimum clock period for a circuit which will not violate the timing constraints. Consider the circuit of following Figure (a). Suppose the inverter has a propagation delay of 2 ns, and suppose the flip-flop has a propagation delay of 5 ns and a setup time of 3 ns. (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, 5 ns 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 flip-flop 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.

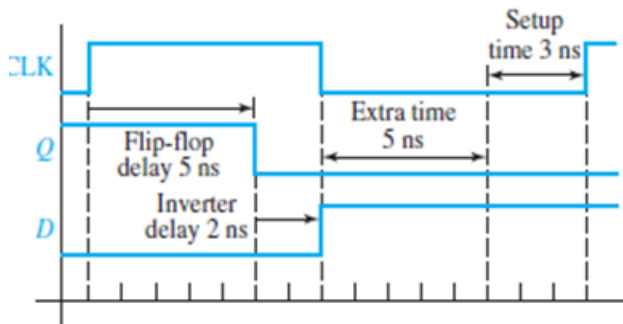
Determination of Minimum Clock Period



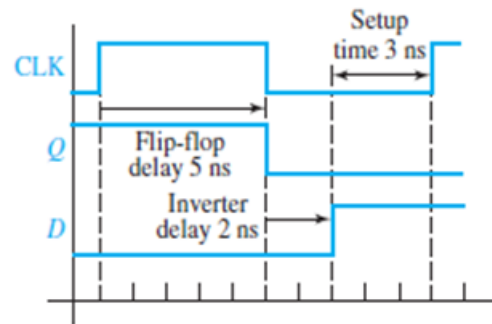
(a) Simple flip-flop circuit



(b) Setup time not satisfied



(c) Setup time satisfied

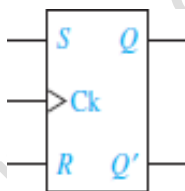


(d) Minimum clock period

Fig.1.21. Timing Diagram of D-Flipflop

2.5 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.



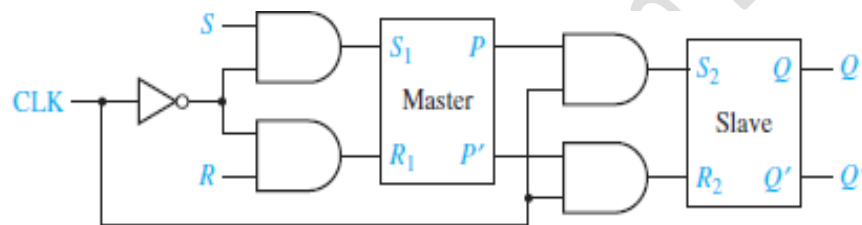
Operation summary:

$S = R = 0$	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

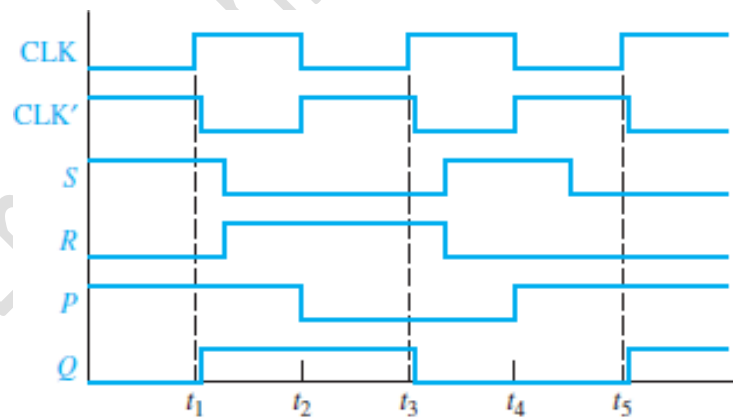
Fig.1.22. S-R Flipflop

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 t_1 . Then $R = 1$ and Q changes to 0 at t_3 .



(a) Implementation with two latches



(b) Timing analysis

Fig.1.23. S-R Flipflop Timings

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 t_4 , $S = 1$ and $R = 0$, so P changes to 1. Then S changes to 0 at t_5 , but P does not change, so at t_5 , Q changes to 1 after the rising edge of CLK. However, at t_5 , $S = R = 0$, 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.

2.6 J-K FLIP-FLOP

The J-K flip-flop (shown in the following Figure) 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.

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).

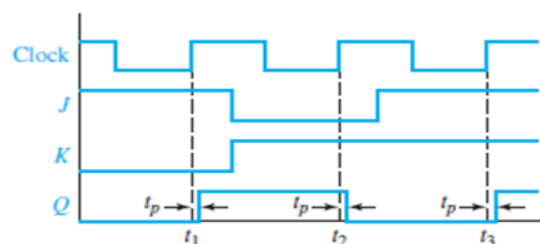
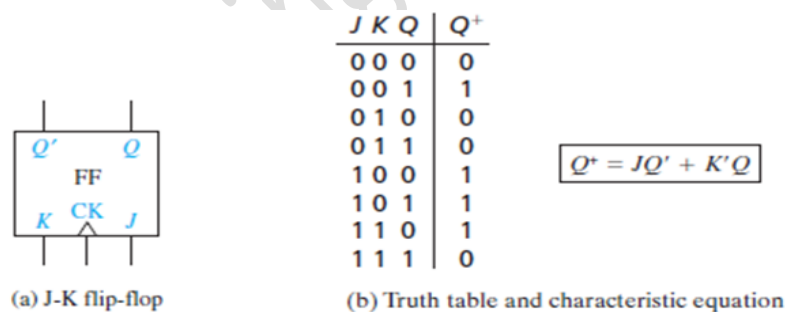


Fig.1.24. J-K Flipflop

Figure (c) shows the timing for a J-K flip-flop. This flip-flop changes state a short time (t_p) 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 $Q = 0$, $J = 1$, and $K = 0$ before the first rising clock edge, Q changes to 1 at t_1 .

Because $Q = 1$, $J = 0$, and $K = 1$ before the second rising clock edge, Q changes to 0 at t_2 . Because $Q = 0$, $J = 1$, and $K = 1$ before the third rising clock edge, Q changes to 1 at t_3 .

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.

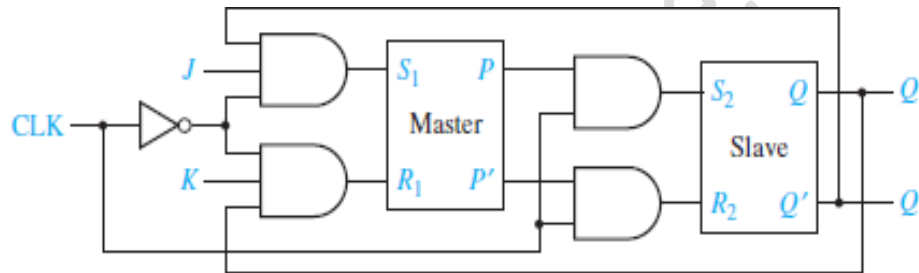


Fig.1.25 Master Slave Flipflop

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 $S = JQ'Clk'$ and $R = K'QClk'$, 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.

2.7 T FLIP-FLOP

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.

The T flip-flop (shown in the following 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 iff the present state (Q) is 1 and $T = 0$ or the present state is 0 and $T = 1$.

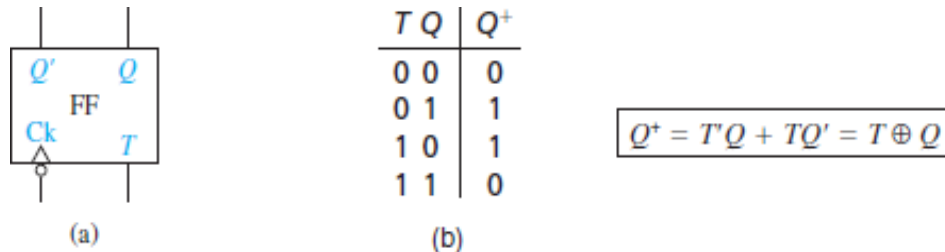


Fig.1.26 T Flipflop

The following Figure shows a timing diagram for the T flip-flop. At times t_2 and t_4 the T input is 1 and the flip-flop state (Q) changes a short time (t_p) after the falling edge of the clock pulse. At times t_1 and t_3 the T input is 0, and the clock edge does not cause a change of state.

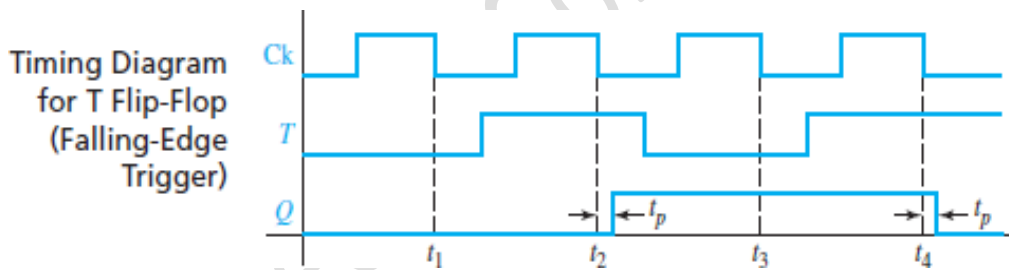


Fig.1.27

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 $Q \oplus T$, so $Q^+ = Q \oplus TQ' + T'Q$, which is the characteristic equation for the T flip-flop.

2.8 FLIP-FLOPS WITH ADDITIONAL INPUTS

Flip-flops often have additional inputs which can be used to set the flip-flops to an initial state independent of the clock. The following Figure shows a D flip-flop with *clear* and *preset* inputs. The small circles (inversion symbols) on these inputs indicate that a logic 0 (rather than a 1) is required to clear or set the flip-flop. This type of input is often referred to as *active-low* because a low voltage or logic 0 will activate the clear or preset function. We will use the notation *ClrN* or *PreN* to indicate active-low clear and preset inputs. Thus, a logic 0 applied to *ClrN* will reset the flip-flop to $Q = 0$, and a 0 applied to *PreN* will set the flip-flop to $Q = 1$.

These inputs override the clock and D inputs. That is, a 0 applied to the *ClrN* will reset the flip-flop regardless of the values of D and the clock. *ClrN* and *PreN* are often referred to as *asynchronous* clear and preset inputs because their operation does not depend on the clock.

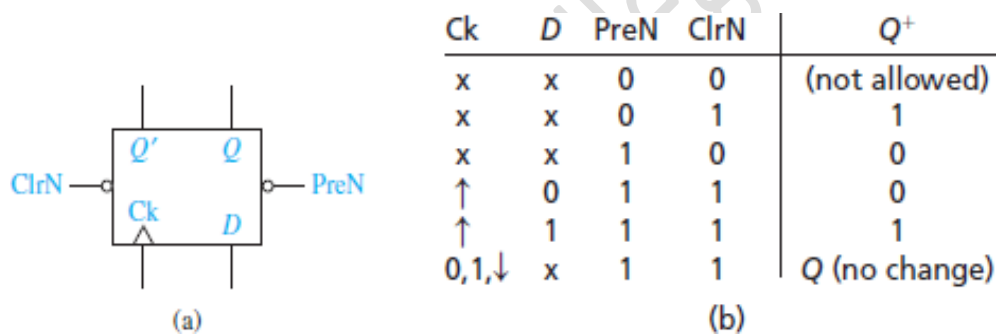


Fig. 1.28. D flipflop with Clr and Pre

The above table (Figure (b)) summarizes the flip-flop operation. In the table, *c* indicates a rising clock edge, and *X* is a don't-care. The last row of the table indicates that if Clk is held at 0, held at 1, or has a falling edge, Q does not change.

The following Figure illustrates the operation of the *clear* and *preset* inputs. At t_1 , *ClrN* = 0 holds the Q output at 0, so the rising edge of the clock is ignored. At t_2 and t_3 , normal state changes occur because *ClrN* and *PreN* are both 1. Then, Q is set to 1 by *PreN* = 0, but Q is cleared at t_4 by the rising edge of the clock because D = 0 at that time.

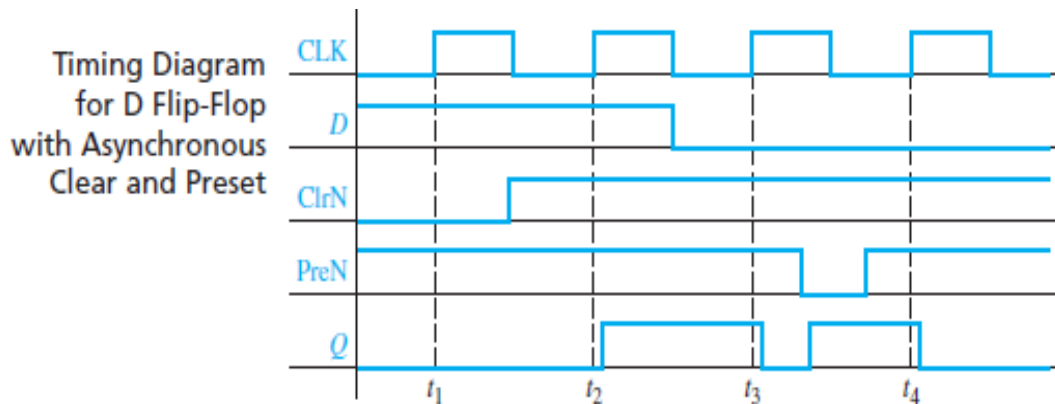


Fig. 1.29

In synchronous digital systems, the flip-flops are usually driven by a common clock so that all state changes occur at the same time in response to the same clock edge. When designing such systems, we frequently encounter situations where we want some flip-flops to hold existing data even though the data input to the flip-flops may be changing. One way to do this is to gate the clock, as shown in the following Figure (a).

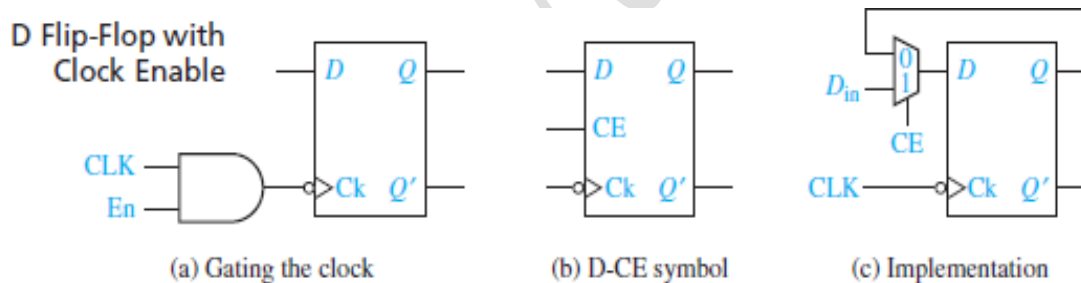


Fig 1.30

When $En = 0$, the clock input to the flip-flop is 0, and Q does not change. This method has two potential problems. First, gate delays may cause the clock to arrive at some flip-flops at different times than at other flip-flops, resulting in a loss of synchronization. Second, if En changes at the wrong time, the flip-flop may trigger due to the change in En instead of due to the change in the clock, again resulting in loss of synchronization. Rather than gating the clock, a better way is to use a flip-flop with a *clock enable* (CE). Such flip-flops are commonly used in CPLDs and FPGAs.

Figure (b) shows a D flip-flop with a clock enable, which we will call a *D-CE* flip-flop. When $CE = 0$, the clock is disabled and no state change occurs, so $Q^+ = Q$. When $CE = 1$, the flip-flop acts like a normal D flip-flop, so $Q^+ = D$. Therefore, the characteristic equation is $Q^+ = Q \cdot CE' + D \cdot CE$. The

D - CE flip-flop is easily implemented using a D flip-flop and a multiplexer (Figure (c)). For this circuit, the MUX output is $Q^+ = D + Q \cdot CE + Din \cdot CE$. Since, there is no gate in the clock line; this cannot cause a synchronization problem.

2.9 Characteristic Equations of Flip-Flop

The *characteristic equations* of flip-flops are useful in analyzing circuits made of them. Here, next output, Q_{n+1} , is expressed as a function of present output Q_n and the input to the flip-flops. Karnaugh map can be used to get the optimized expression.

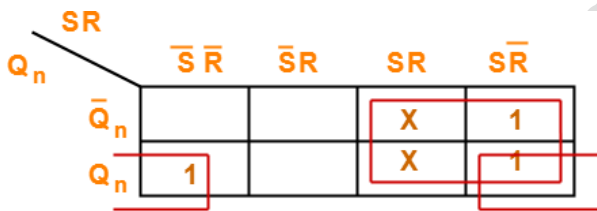
S	R	Q^+
0	0	Q
0	1	0
1	0	1
1	1	?

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

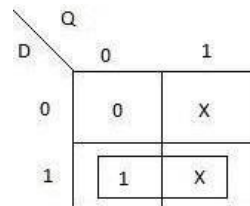
J	K	Q^+
0	0	Q
0	1	0
1	0	1
1	1	0

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

SR Flip-Flop:

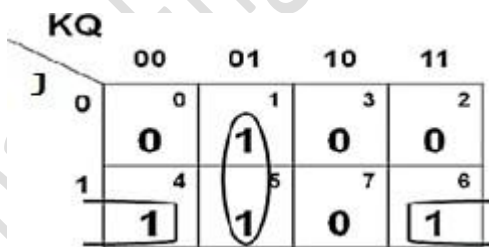


D Flip-Flop:



JK Flip-Flop:

K Map



T Flip-Flop:

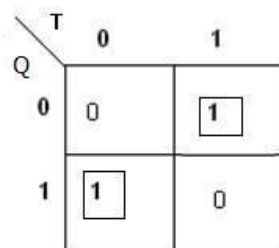


Fig.1.31. Characteristic Equations of SR, D, JK & T Flip-Flops

The characteristic equations for the latches and flip-flops discussed so far are:

$$Q^+ = S + R'Q \quad (SR = 0) \quad (\text{S-R latch or flip-flop})$$

$$Q^+ = GD + G'Q \quad (\text{gated D latch})$$

$$Q^+ = D \quad (\text{D flip-flop})$$

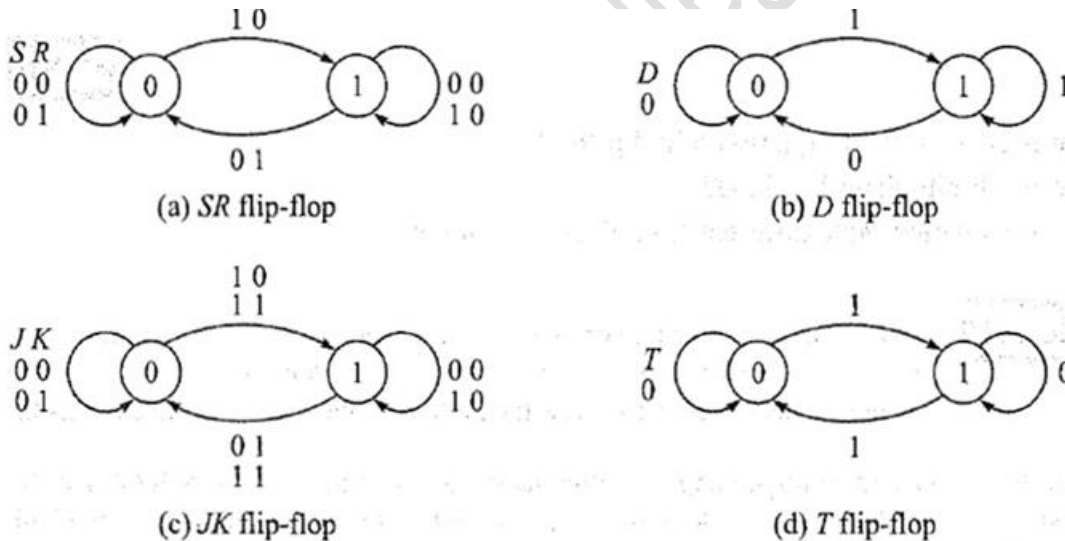
$$Q^+ = D \cdot CE + Q \cdot CE' \quad (\text{D-CE flip-flop})$$

$$Q^+ = JQ' + K'Q \quad (\text{J-K flip-flop})$$

$$Q^+ = T \oplus Q = TQ' + T'Q \quad (\text{T flip-flop})$$

2.10 Flip-Flops as Finite State Machine

In a sequential logic circuit, the value of all memory elements at a given time defines the *state* of that circuit at that time. *Finite State Machine (FSM)* concept offers a better alternative to truth table in understanding progress of sequential logic with time.



State transition diagram of (a) *SR flip-flop*, (b) *D flip-flop*, (c) *JK flip-flop*, (d) *T flip-flop*

Fig.1.32 State Transition diagram

2.11 Flip-Flop Excitation Table:

In synthesis or design problem, excitation tables are very useful. *Excitation table* of a flip-flop is looking at its truth table in a reverse way; here, flip-flop output is presented as a dependent function of transition $Q \rightarrow Q_{n+1}$ and comes later in the table.

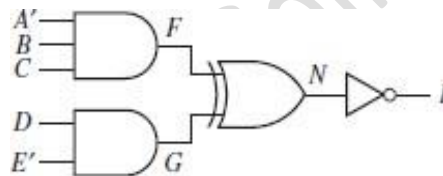
$Q \rightarrow Q_{n+1}$		S	R	J	K	D	T
0	0	0	x	0	x	0	0
0	1	1	0	1	x	1	1
1	0	0	1	x	1	0	1
1	1	x	0	x	0	1	0

Fig.1.33 Excitation Table of Flip-Flops

Exercise Problems

1] Write VHDL statements that represent the following circuit:

- Write a statement for each gate.
- Write one statement for the whole circuit.



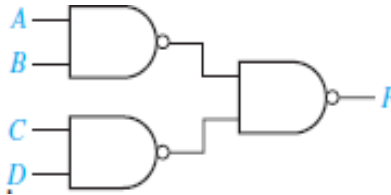
2] Draw the circuit represented by the following VHDL statements: $F \leq E$ **and** I ;

$I \leq G$ **or** H ;

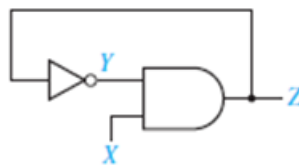
$G \leq A$ **and** B ;

$H \leq \text{not } C$ **and** D ; 3] Write

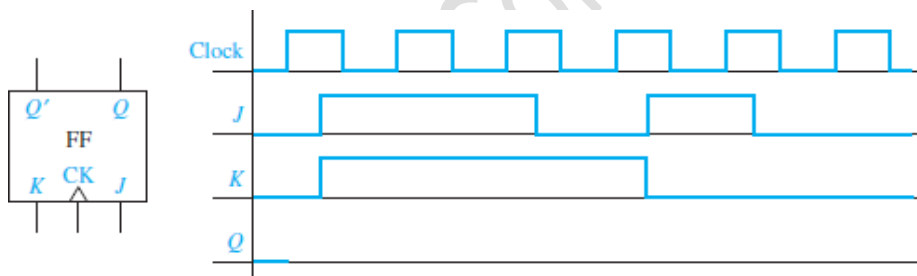
- a complete VHDL module for a two-input NAND gate with 4-ns delay.
- Write a complete VHDL module for the following circuit that uses the NAND gate module of Part (a) as a component.



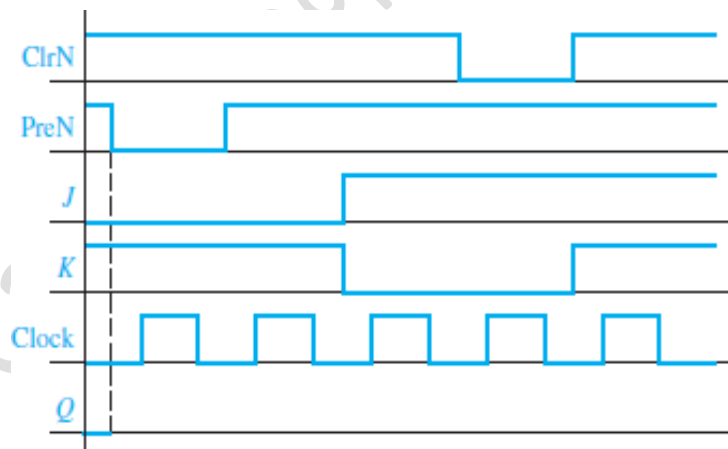
3] Assume that the inverter in the given circuit has a propagation delay of 5 ns and the AND gate has a propagation delay of 10 ns. Draw a timing diagram for the circuit showing X, Y, and Z. Assume that X is initially 0, Y is initially 1, after 10 ns X becomes 1 for 80 ns, and then X is 0 again.



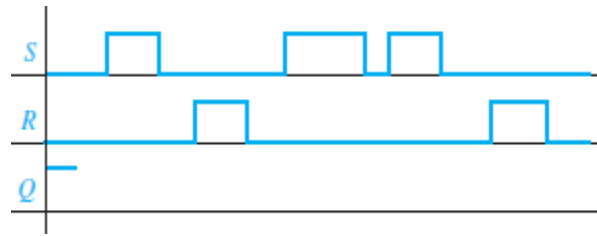
4] Complete the following timing diagram for the flip-flop:



5] Complete the following timing diagram for a J-K flip-flop with a falling-edge trigger and asynchronous ClnN and PreN inputs.



6] Complete the following timing diagram for an S-R latch. Assume Q begins at 1.



7] Convert by adding external gates: (a) a D flip-flop to a J-K flip-flop; (b) a T flip-flop to a D flip-flop; (c) a T flip-flop to a D flip-flop with clock enable.

Web Resources

1. https://www.tutorialspoint.com/vlsi_design/vlsi_design_vhdl_introduction.htm
2. <https://surf-vhdl.com/vhdl-syntax-web-course-surf-vhdl/vhdl-behavioral-modeling-style/>
3. <https://www.xilinx.com/support/documentation/university/Vivado-Teaching/HDL-Design/2015x/VHDL/docs-pdf/lab1.pdf>
4. <https://dept-info.labri.fr/~strandh/Teaching/AMP/Common/Strandh-Tutorial/flip-flops.html#:~:text=Both%20latches%20and%20flip%2Dflops,a%20flip%2Dflop%20always%20does.>
5. <https://www.geeksforgeeks.org/difference-between-flip-flop-and-latch/>

Video Resources

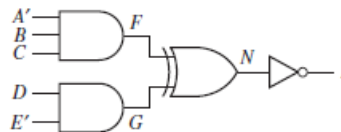
1. <https://www.youtube.com/watch?v=MzLslDsjdN4>
2. https://www.youtube.com/watch?v=G6V_bZaHezk
3. <https://www.youtube.com/watch?v=Wn9Z-IFxxl8>
4. <https://www.youtube.com/watch?v=NBRLYDXITj0>
5. <https://www.youtube.com/watch?v=AEC7SSIn-so>
6. <https://www.youtube.com/watch?v=nCXMIXhmP5w>
7. <https://www.youtube.com/watch?v=p31ERMgleA4>
8. https://www.youtube.com/watch?v=446dN_nTipg

9. <https://www.youtube.com/watch?v=1PA1JX77XI8>

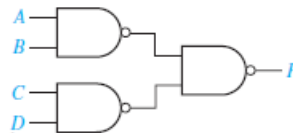
Question Bank

1. Mention the difference between combinational & sequential circuits with block diagram.
2. Mention the difference between asynchronous & synchronous circuits with example.
3. Differences between Latch & Flipflop. Give example.
4. Define clocked sequential circuit.
5. Difference between Characteristic & Excitation table.
6. Explain the operation of different types of flipflop.
7. What is Race around condition. Explain.
8. Explain the operation of JK flip-flop. With logic diagram, characteristic table.
9. Discuss how unstable condition $S=R=1$ is avoided in storage latch of the following:
a) D latch b) JK flipflop c) T flipflop
10. Explain clocked RS flipflop with logic diagram.
11. Show that clocked D flip-flop can be reduced by one gate.
12. Explain how D & T flipflop works with logic diagram.
13. Discuss state table, state diagram, and state equation with example.
- 14) Draw the logic diagram of clocked D-flip-flop. Write its truth table, characteristic equation, state diagram and excitation table. What is the drawback of SR flip-flop?
- 15) Using behavioural model, write Verilog HDL code for a D-flip-flop.
- 16) With the help of a block diagram Explain the working of a JK Master-Slave flip-flop.
- 17) Show how a SR flip-flop can be converted to a JK flip-flop.
- 18) Explain the different types of flip flops along with their truth table.
- 19) Explain basic SR flip-flop by using NOR gate. What is the drawback of SR flip flop? How JK flip flop is obtained from SR flip flop?
- 20) Find out characteristic equations of JK flip-flop and D flip-flop.
- 21) Explain the working of a JK flip-flop. Write its truth table, state diagram and excitation table.

- 22) Draw a switch debouncer using a SR latch and show the waveforms of switch bounce and debounce.
- 23) Describe i) Master-slave JK-flip flop ii) Edge triggered flip flop.
- 24) Explain the advantages of an edge triggered flip flop over a pulse triggered flip flop.
- 25) Give state transition diagram of SR, D, JK and T flip flops.
- 26) What do you mean by characteristic equation of a flip-flop? Derive characteristic equation for SR flip-flop
- 27) Write VHDL code for
- 2:1 MUX using VHDL statement and conditional assignment statement
 - 4-to-1 MUX using VHDL statement and conditional assignment statement.
- 28) Write VHDL statements that represent the following circuit: [CO3: P7-Hint]
- Write a statement for each basic gate.
 - Write one statement for the whole circuit.



- 29) Write
- a complete VHDL module for a two-input NAND gate with 4-ns delay.
 - Write a complete VHDL module for the following circuit that uses the NAND gate module of Part (a) as a component.



- 30) Draw the circuit represented by the following VHDL statements:

T1 <= **not A and not B and I0;**

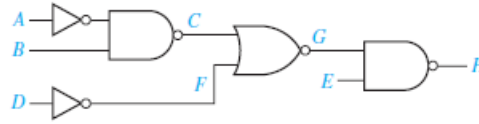
T2 <= **not A and B and I1;**

T3 <= **A and not B and I2;**

T4 <= A and B and I3;

F <= T1 or T2 or T3 or T4;

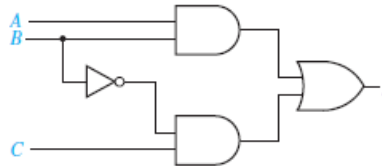
31) Write a single concurrent VHDL statement to represent the following circuit



32) Write the VHDL code for

- A full adder
- A full subtractor

33) In the following circuit, all gates, including the inverter, have an inertial delay of 10 ns.



- Write VHDL code that gives a dataflow description of the circuit. All delays should be inertial delays.
- Using the Direct VHDL simulator simulate the circuit. (Use a View Interval of 100 ns.) Initially set A = 1, B = 1 and C = 1, then run the simulator for 40 ns. Change B to 0, and run the simulator for 40 ns. Record the waveform [Even you can draw waveform for given conditions]
- Change the VHDL code of Part (a) so that the inverter has a delay of 5 ns.
- Repeat Part (b)
- Change the VHDL code of Part (c) so that the output OR gate has a transport delay rather than an inertial delay
- Repeat Part (b)
- Explain any differences between the waveforms for Parts (b), (d), and (f).

University Questions

1. Explain the structure of VHDL program. Write VHDL code for 4-bit parallel adder using full adder as component.
2. Explain the working of SR latch using NOR gates. Show how SR latch can be used for switch debouncing.
3. Differentiate between Latch and Flip Flop. Show how SR flipflop can be converted to D flip flop.
4. Derive the characteristics equations for D, T, SR and JK flipflops.
5. Draw the logic diagram of master slave JK flipflop using NAND gates and explain the working with suitable timing diagram.
6. With example explain the syntax of conditional signal assignment statement in VHDL.