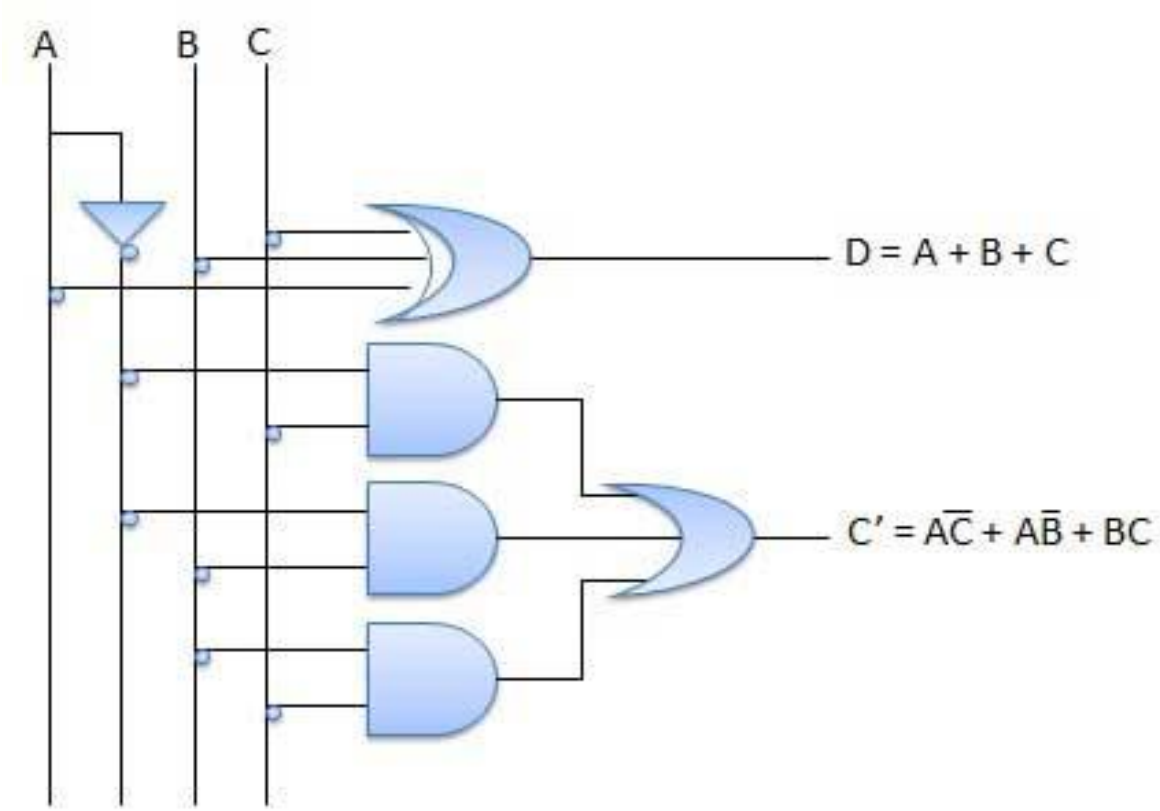
HOW TO TIME LOGIC

When we speak about timing in logic circuits, several factors come into picture. A circuit is timed in mainly two manners pre-synthesis and post-synthesis. Pre-synthesis is the time when you have finished implementing a logic in your code, and you are ready for simulation followed by synthesis. Post-synthesis is the time when you are done with synthesis and you have the synthesizer reporting the slack. So, you run the compiler and look if your code is synthesizable. If it is, you move on to synthesis, if not, the compiler throws a bunch of errors which makes you go back to your code and fix them. Some of those errors might be syntaxial, some might be logical.

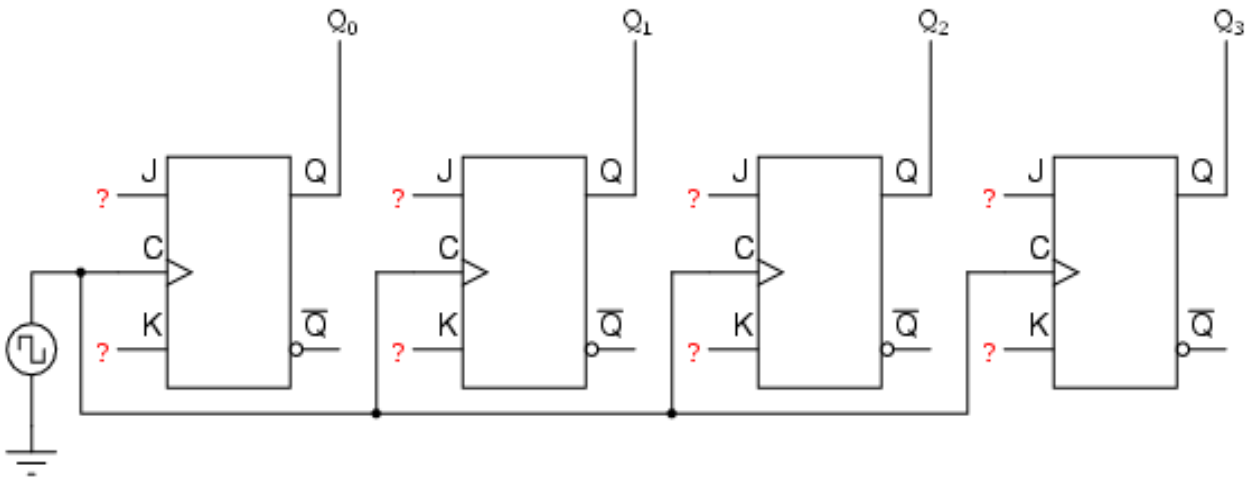
In simpler terms, timing a logic circuit essentially means making sure that the implemented logic gives the desired output at the desired time. There are two types of logic circuits that one has, one is combinational and the other is sequential. To build logic using these circuits, one needs to make clear what logic is. For Software Engineers, the logic that they write is the Business Logic (how a particular business is supposed to function). For ASIC Design Engineers, logic is how they want a circuit to behave, which is very similar to Business Logic. So, combinational circuits are the circuits which have absolutely no memory, they can't hold a value. They are wholly dependent on the present input values that are presented to them. We get output from them whenever input values change. Logic gates such as AND, OR, NAND, NOR are accompanied by some Multiplexers, Demultiplexers and Adders if required, to make a combinational circuit. Delay associated with these circuits is entirely dependent on how long they take to process the inputs.

An example of combination circuit:



Unlike Combinational Circuits, Sequential Circuits have memory and can store data. Internally, they are made up of logic gates. Their output is dependent on inputs from previous stage as well as the present inputs. This is where the state of a sequential circuit comes into picture. They are of three types, one that gets triggered on a positive edge of a clock, one that gets triggered on the negative edge of the clock and one that is triggered when a stable clock level is reached. Flip Flops are used for constructing a sequential circuit. The circuits that get triggered on an edge of the clock are called Edge Triggered circuits and which are triggered on clock level are called level triggered circuits (also known as Latches). Delay associated with sequential circuits depends on which library and technology is in use.

Example of a Sequential Circuit:



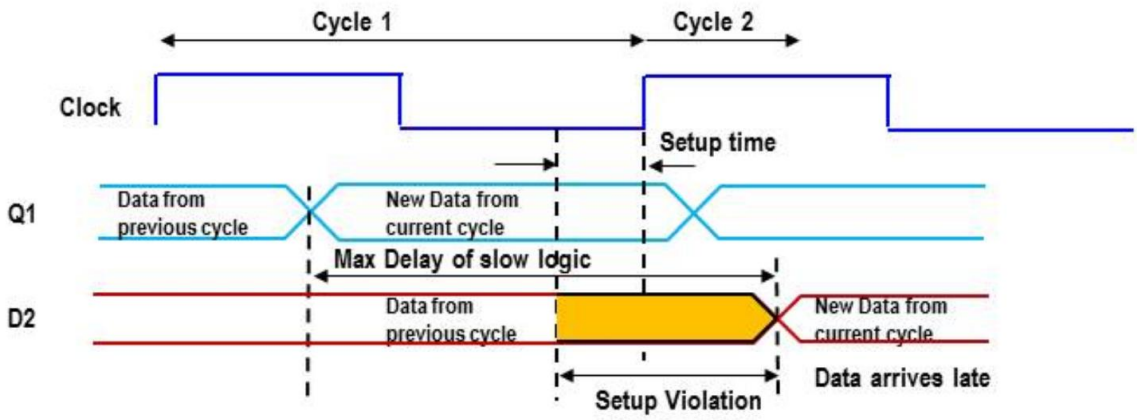
We time the circuits by introducing a Clock pulse into the circuit. The clock pulse basically acts like a driving force for the circuits and makes them output a data.

Before we dive into knowing how to time logic, there are few parameters that need consideration:

1. Input Setup Time.
2. Input Hold Time.
3. Clock to Q time (C → Q).
4. Logic Delay.
5. Clock Skew.
6. Clock Jitter.
7. Metastable condition.

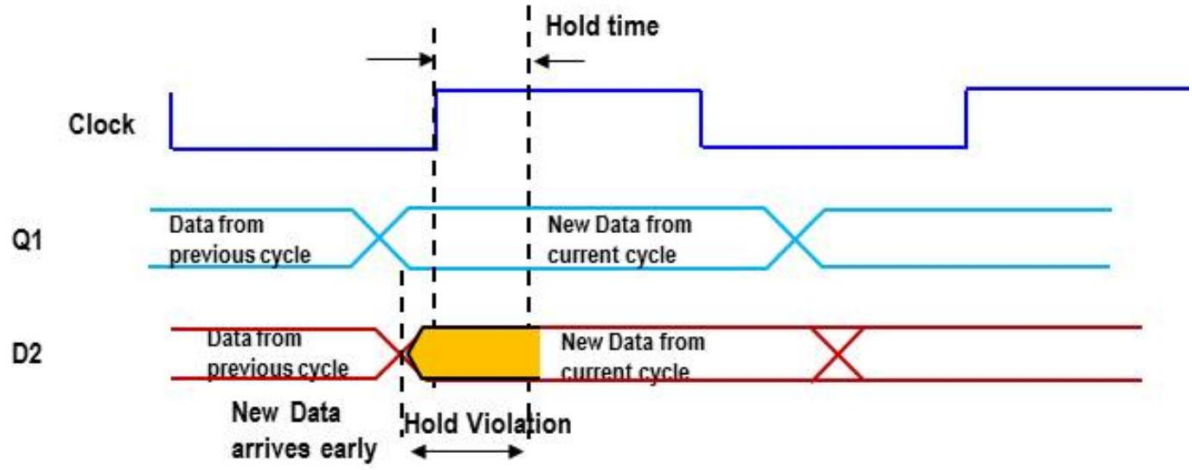
INPUT SETUP TIME:

Input Setup time is defined as "the time duration for which the input data (D_{in}) should be stable before the active clock edge appears". Setup Time is crucial because the logic in a latch or a flip flop takes longer than the clock to appear at the inputs and any sequential circuit would want its input to be stable before it starts working and get a stable output. If IS wasn't there, then it is likely that the circuit in attention can get an unstable output.



INPUT HOLD TIME:

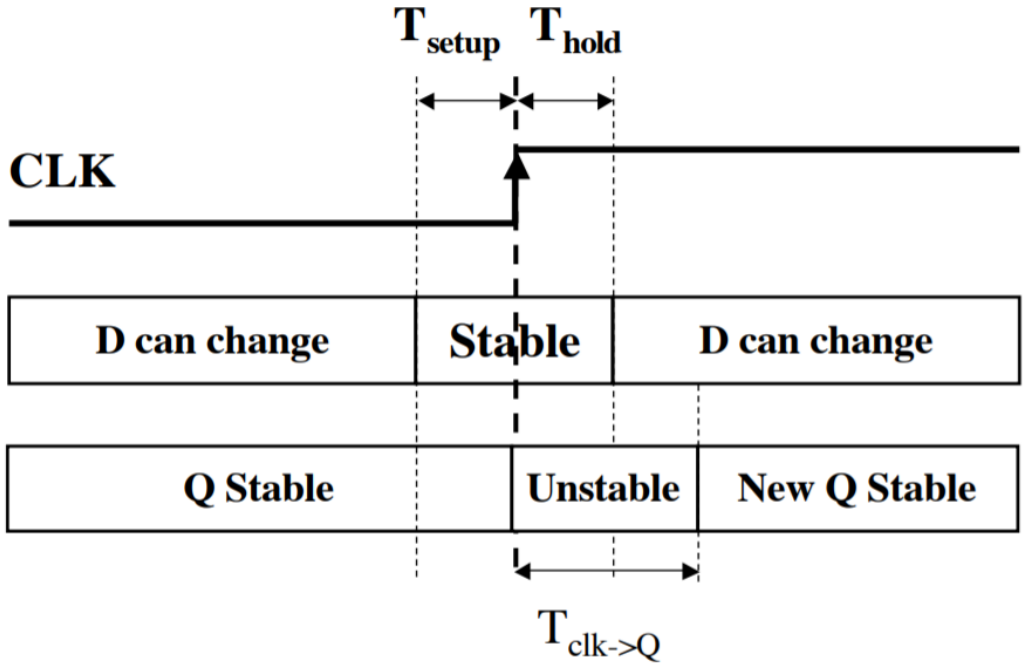
Input hold time is defined as the time for which the input should be kept stable after the active edge has occurred. This time is required so as to store the user desired value and not a garbage value. This stable data that is stored in a memory element, is then required by the subsequent stages for calculation of the final output. IH is determined by the technology library.



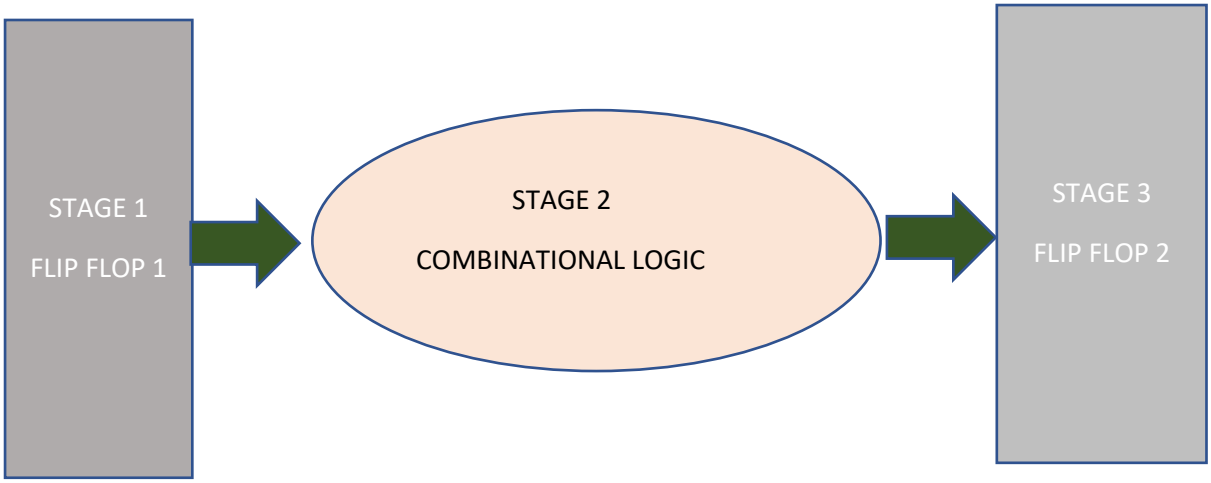
If IS and IH are violated, the state of the circuit can go into metastable state.

CLOCK TO Q TIME (C → Q):

Clock to Q time is the time duration that is required to wait after the clock to get a stable output. C → Q time and Input Hold time have a same initial point.



LOGIC DELAY (LD):



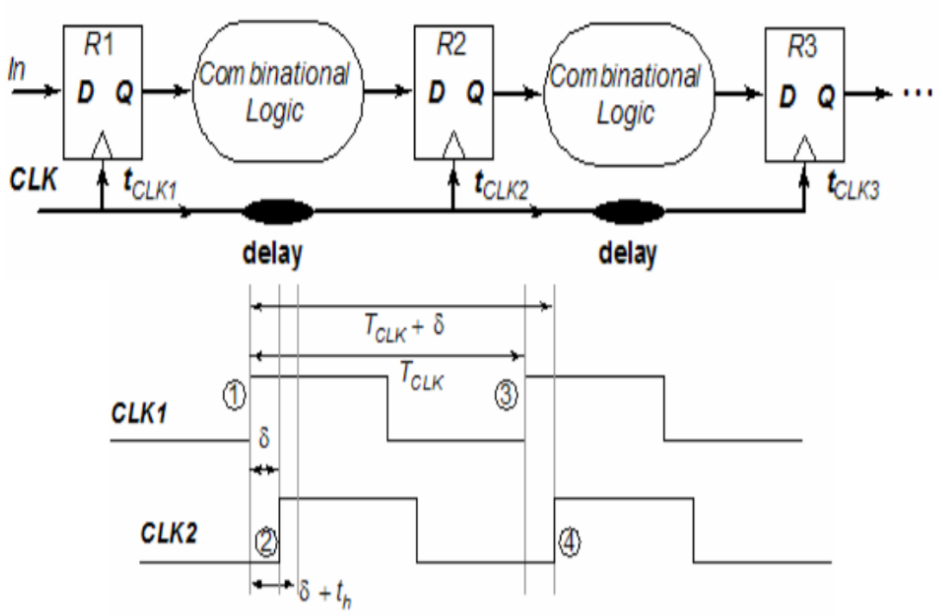
Logic delay is the time that is taken by the combinational circuit which computes the appropriate output according to the desired logic to produce output and feed it to the next stage. More often than not, this stage adds up to the maximum of the circuit delay. In the above diagram, the stage 2 represents the logic delay.

CLOCK SKEW:

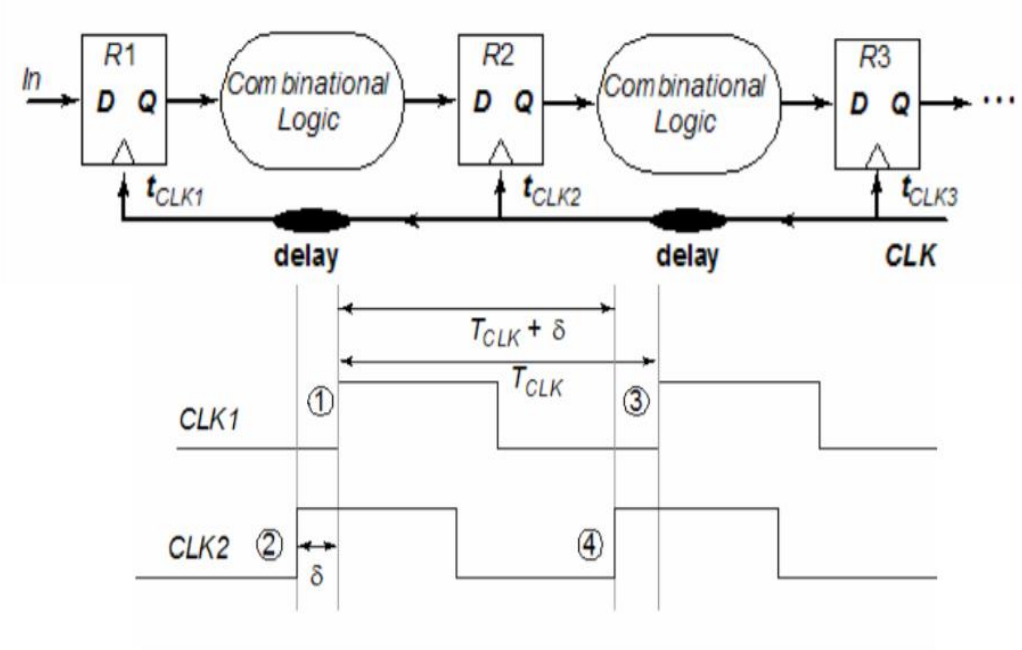
Clock Skew is defined as the duration of the clock in a synchronous sequential circuit at which the clock pulse arrives at a Flip Flop stage. The skew changes with the arrival of the clock pulse. The clock pulse may appear early, which will result in a positive clock skew, and the clock pulse may arrive later than expected which will result in a negative clock skew. Clock skew may often result in a 'Race' problem. If clock skew is analyzed and reduced to a

certain level, it can improve performance. If increased too much, that will result into race conditions.

POSITIVE SKEW:

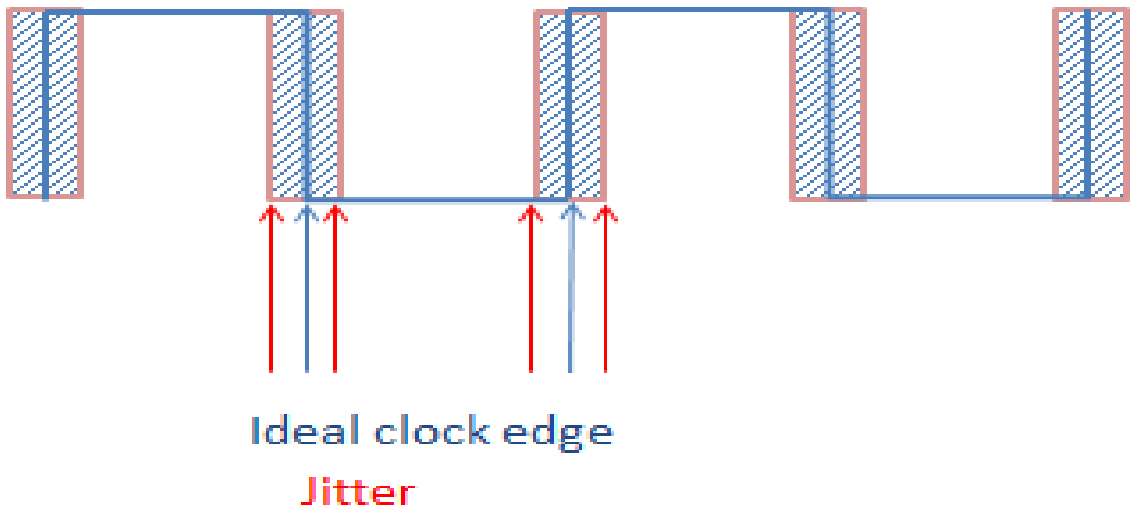


NEGATIVE SKEW:



CLOCK JITTER:

Clock Skew can be often confused with clock jitter. Clock jitter can be explained as the range of the skew containing the positive skew and the negative skew. This is essentially calculated so as to get an idea of the worst case scenario for the skew.



METASTABLE CONDITION:

Some of the flip flops that are used in sequential circuits sometime enter into a metastable state due to some forbidden inputs. This state results into a garbage value. But this metastable state is only for a less period of time after which the flip flop returns to a stable state. This in result causes glitch in the output. An Example would be SR Flip Flop. Below is the truth table and diagram for SR flip flop which shows the forbidden state which causes a Glitch.

Truth Table for SR Flip Flop:

State	**S**	**R**	**Output (Q)**
Store	0	0	Q (no change)
Set	1	0	1
Reset	0	1	0
Forbidden	1	1	-

Whenever the input is S = 1 and R = 1, the SR flip flop goes into a forbidden state which is called metastable state.

The equation for metastability is given by Vm which is Vin = Vout.

Now that we understand the parameters related to timing, the basic equations for IS and IH can be written:

Input Setup Time Equation :

$$C.T >= 2 * skew + C \rightarrow Q + IS + LD$$

Where,
C.T. is Cycle Time, Skew = Clock Skew, LD is the logic delay (the time the logic takes to output a data), IS is Input Setup Time, C → Q is the Clock to Q time (or output hold time).

Input Hold Time Equation :

$$C->Q + LD > 2*skew + IH$$

Where all the parameters have same meaning as above and IH is the Input Hold Time.

After consideration all the timing delays in the input setup time equation and input hold time equation, we conclude that LD is the only parameter that is controllable and is in under our control as to not violate the IS and IH time equations. This makes it really important that we optimize and reduce the logic delay, in order to achieve the required clock period and attain maximum frequency.

SLACK :

Slack is defined as the difference between the expected time for data arrival versus the actual data arrival time for any given path. It is used to determine if the data is processed correctly at the desired frequency or speed. When a design is put to synthesis, user puts in some constraints. According to these constraints, an expected time for data arrival is determined by the synthesizer for every path in the design. If the arrival time violates the expected time, that is, arrives late or early, slack occurs. Slack is given by following formula :

Slack = Data required time - Data Arrival Time

Slack can be both positive and negative depending on the data arrival time.

- If data arrival time is less than the data expected arrival time, then the data is said to have arrived early than expected time which gives a positive slack. That means the designer still has some scope to alter the design.
- If the data arrival time is higher than the data expected arrival time, then the data is said to have arrived late compared with the expected arrival time. This gives a negative slack. That means the design is not working as expected and may need some design changes.
- If the data arrival time and the data expected arrival time is same, this means that we have a zero (0) slack. Hence, our design is said to be working as required at an expected frequency.

CRITICAL PATH :

A critical path is defined as the path in the design that takes the longest to process data. It has a big propagation delay. The synthesizer analyses the critical path during synthesis and gives the slack associated with that path. It is the responsibility of the designer to optimize the critical paths as much as possible.

FALSE PATH :

A false path is defined as a path that exists just in the design but is not the logically correct path. Therefore, the data does not flow through this path. The purpose of Static Timing Analysis is to calculate the timing for true path, the false path is ignored and does not impact on design performance.

There are essentially three ways to minimize a particular logic and make it time efficient.

1. Boolean Algebra.
2. K-Maps (Karnaugh Maps).
3. Quine McCluskey algorithm.

1. **Boolean Algebra:**
   Boolean algebra is a method of logic minimization which follows certain rules of logic design which in turn optimizes a given logic problem. There can be a bunch of logic that is unneeded, without which the logic can compute a desired result. Logic minimization through Boolean Algebra can be used to get rid of such logic. Below image states the basic rules for Boolean Algebra:

Basic Rules of Boolean Algebra

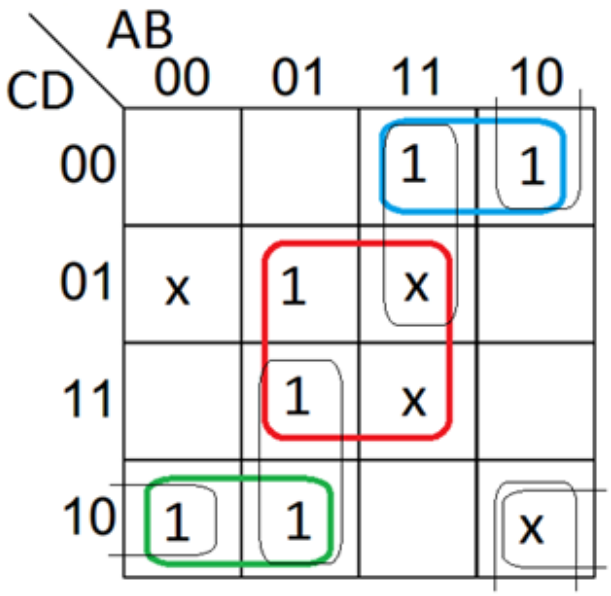
1. $A + 0 = A$	7. $A \cdot A = A$
2. $A + 1 = 1$	8. $A \cdot \overline{A} = 0$
3. $A \cdot 0 = 0$	9. $\overline{\overline{A}} = A$
4. $A \cdot 1 = A$	10. $A + AB = A$
5. $A + A = A$	11. $A + \overline{A}B = A + B$
6. $A + \overline{A} = 1$	12. $(A + B)(A + C) = A + BC$

DeMorgan's Theorem

$$\overline{(AB)} = (\overline{A} + \overline{B}) \qquad \overline{(A + B)} = (\overline{A}\ \overline{B})$$

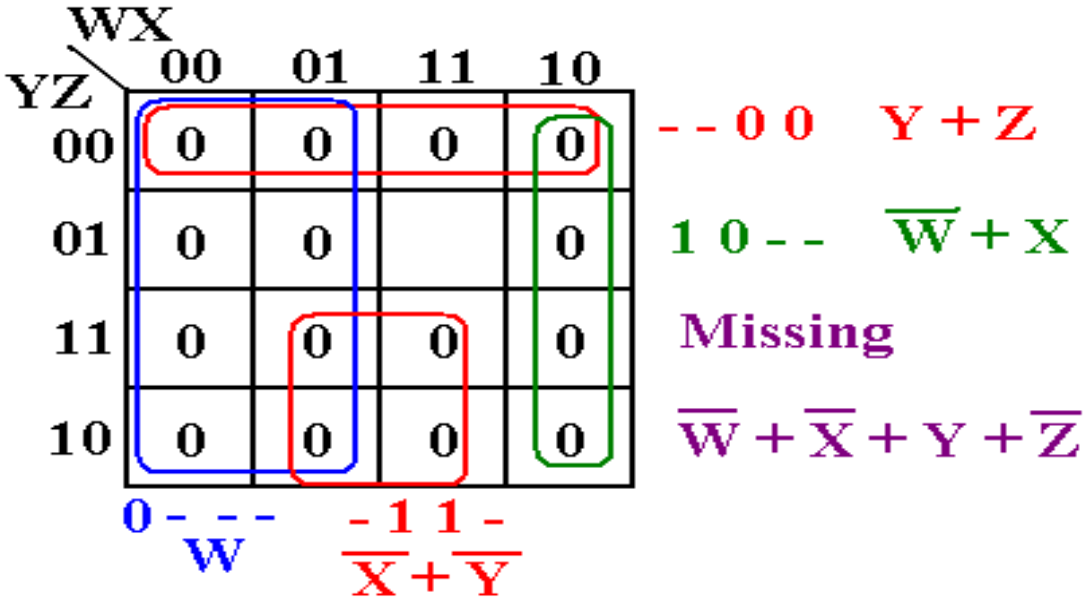
2. **K-MAPS:**
   This is a method of logic minimization that uses SOP (Sum of Products) or POS (Product of Sums) for reduction of logic. In SOP, binary 1's are grouped together and in POS, binary 0's are grouped together. While grouping, we make sure that the terms don't have more than a single bit change in their inputs. Grouping terms with two bit changes can cause the circuit enter into a metastable state. Then we obtain a Boolean expression from the SOP or POS grouping which helps us to design a logic diagram from it. The disadvantage of Karnaugh Maps is that it can be used only till 6 variables. Examples for each type of K-maps can be seen below:

   Sum of Products:



BD + AC'D' + A'CD'

Product of Sums:

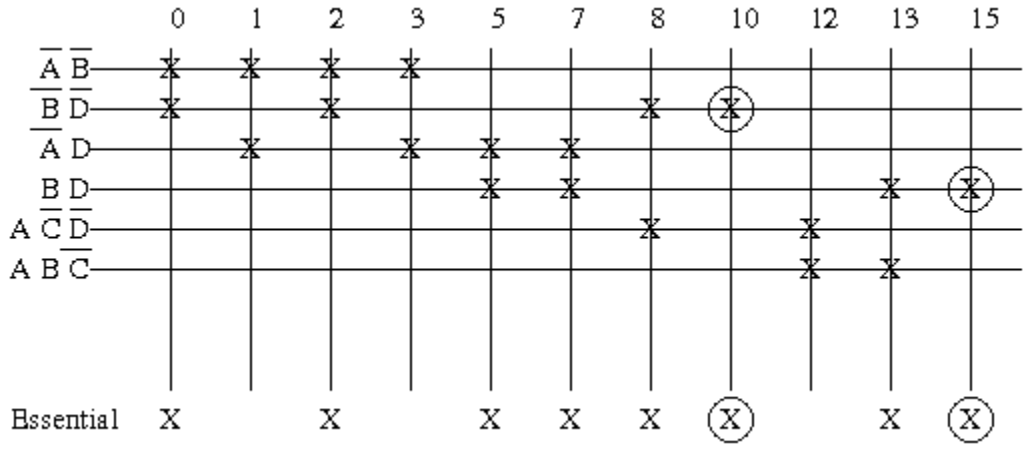


3. **Quine McCluskey:**
   This method of logic minimization uses a tabular approach for reduction. Quine McCluskey method has an advantage over Karnaugh Maps, it can be used for any number of variables. We group the terms here according to their weights, that is, the number of binary 1's a term consist of. Grouping is done just like K – Maps, by observance of 1 bit change.

Table 1

Group	Column I			Column II			Column III		
2	24	0011000	√	24, 28	0011_00	√	24, 28, 88, 92	_011_00	*
3	28	0011100	√	24, 88	_011000	√	~~24, 88, 28, 92~~	~~_011_00~~	
	70	1000110	√	28, 92	_011100	√	70, 86, 102, 118	1_ _0110	*
	88	1011000	√	70, 86	10_0110	√	~~70, 102, 86, 118~~	~~1_ _0110~~	
4	39	0100111	√	70, 102	1_00110	√			
	86	1010110	√	88, 92	1011_00	√			
	92	1011100	√	39, 47	010_111	*			
	102	1100110	√	86, 118	1_10110	√			
	105	1101001	*	102, 118	11_0110	√			
5	47	0101111	√						
	118	1110110	√						

In the above picture, the terms in subsequent weight groups can be grouped together. For example, 24 can be checked for grouping with 28,70 and 88 but can't be grouped with 39,86,92,102,105 as there are two binary 1's added in the terms (two bit change).



Suppose we have reduced our logic using the above methods. We still need to know what logic delay to should we have to avoid problems like Long Path and Short Path (or Race). Long path is a path which travels through a chunk of logic but takes the longest to reach the output as compared to other paths. Short path or race problem is where a data travels

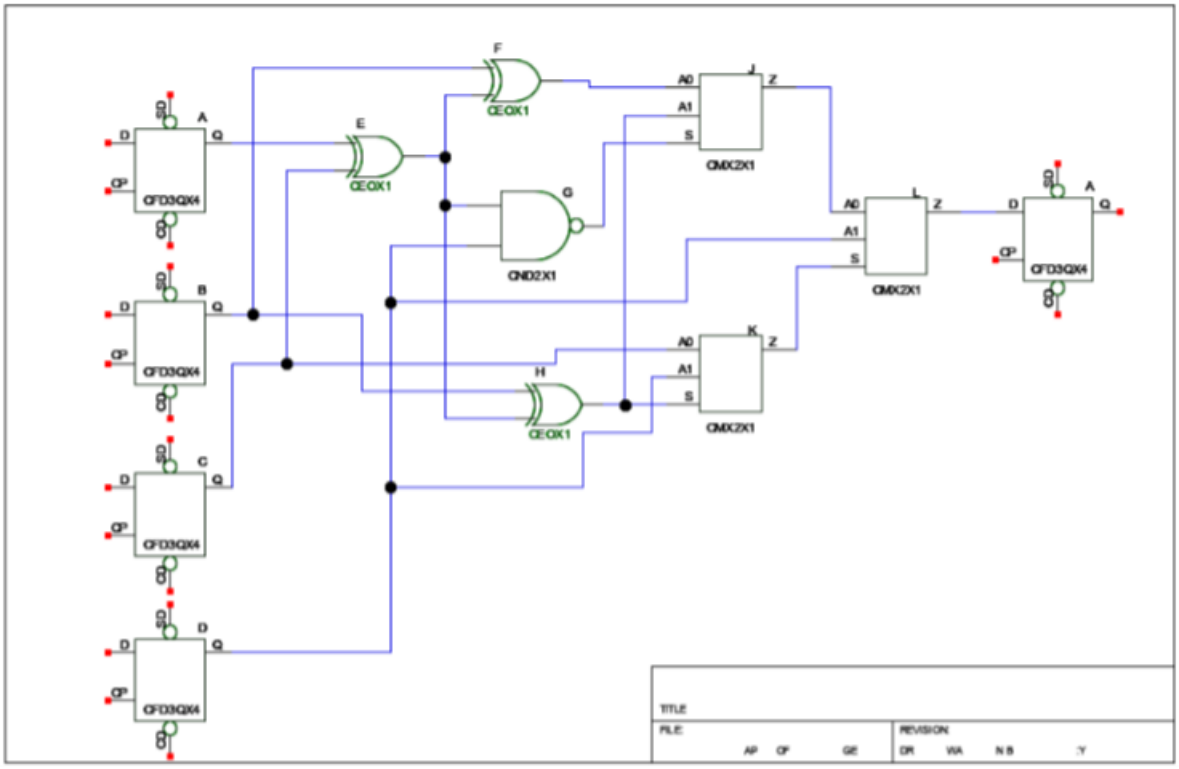
through two (2) Flip Flops in one cycle. This happens because of incorrect timing estimation for logic.

One way to know the range of the logic and know the worst case scenario for long path and short path through the design is by Static Timing Analysis.

Long path problem (Static Timing Analysis):

Suppose we have the following circuit:

Assuming 9.5ff/load:



The datasheet for the gates used in the circuit is as follows:

Gate Name	Const Rising	Factor Rising	Const falling	Factor falling	Load A	Load B	Load C	Load D	Load S
CEOX1	0.057500	0.01511	0.04850	0.00670	8.44	8.28			
CFD3QX4	0.08262	0.00157	0.11836	0.00081				4.88	
CMX2X1	0.05348	0.00667	0.05512	0.00418	4.31	4.36			6.44
CND2X1	0.03525	0.00664	0.03688	0.00684	3.91	4.24			
CNR2X1	0.04525	0.00634	0.03231	0.00327	3.67	4.04			

We start with calculation of capacitance each gate is driving. Following formula for ideal gate delay:

Capacitance = (N x net capacitance) + the capacitance of the inputs it is driving.

For example, Gate A is driving the input A or the upper input of gate E. Therefore, the capacitance driven by gate A is given by :

$C_L = (1 \times 9.5) + 8.44$

$= 17.94.$

Likewise, we calculate the Load Capacitance driven by each gate in Step 1.

In step 2, we calculate the propagation delay associated with each gate for rising (positive) edge and for falling (negative) edge of the clock. For this calculation, following formula is used :

T_p = Constant + (Factor x C_L).

So, to calculate propagation delay for gate A, we substitute the Constant, Factor and Load Capacitance value from the datasheet in the formula above.

For rising edge,

$T_p = 0.02862 + (0.00157 \times 17.94)$

$= 0.0567858$

For falling edge,

$T_p = 0.11836 + (0.00081 \times 17.94)$

$= 0.132891$

In step 3, we obtain the edge relations for every gate. For example, for gate E which is a NOR gate,

For Rising = Max ($A_{falling}$, $B_{falling}$),

For Falling = Max (A_{rising} , B_{rising})

To calculate net delay for Gate A is calculated by,

For Rising = Max of falling delays of the inputs of previous gate + rising propagation delay of gate A.

For Falling = Max of rising delays of the inputs of the previous gate + falling propagation delay of gate A.

After the calculation of Load Capacitance, Gate Delay and Net Delay we get the delays through every path. We can obtain the shortest path and the longest path by observing these Net Delays through the gates.

In Step 4, we note the Net Delays through the gates and start going backwards from output to input. This can also be called Back Propagating through the logic. This is done to find the longest and the shortest path.