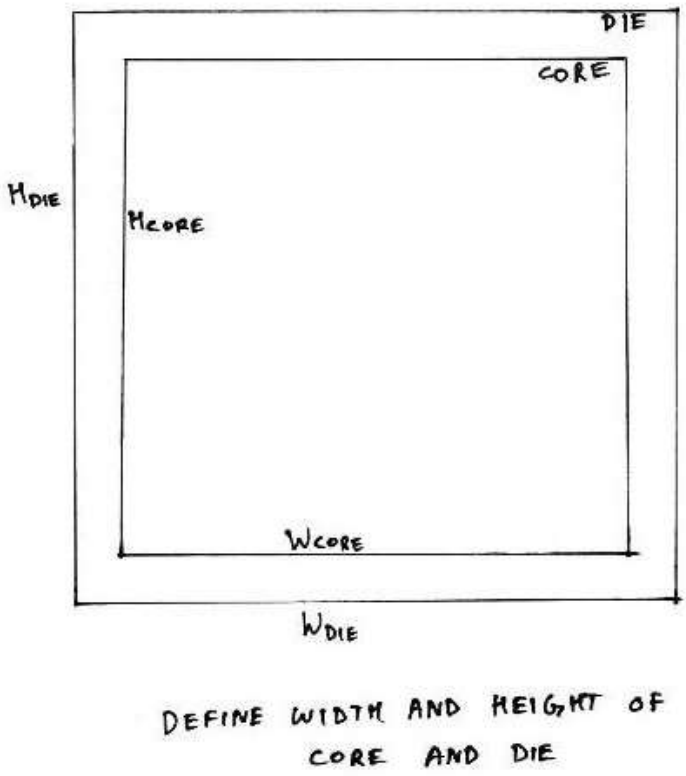
Introduction to Industrial Physical Design Flow.

Kunal Ghosh

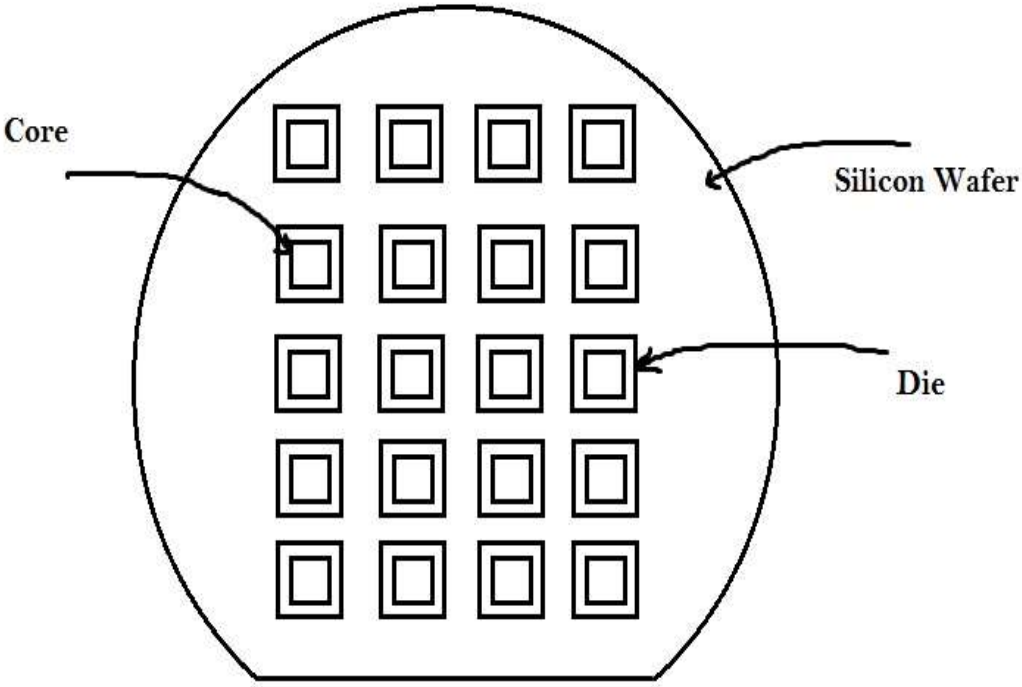
VLSI Physical Design Flow is an algorithm with several objectives. Some of them include minimum area, wire length and power optimization. It also involves preparing timing constraints and making sure, that netlist generated after physical design flow meets those constraints.

Following section will help you to understand the very basic and beginning steps for chip design. It is exactly the way it happens in leading VLSI chip design industries.

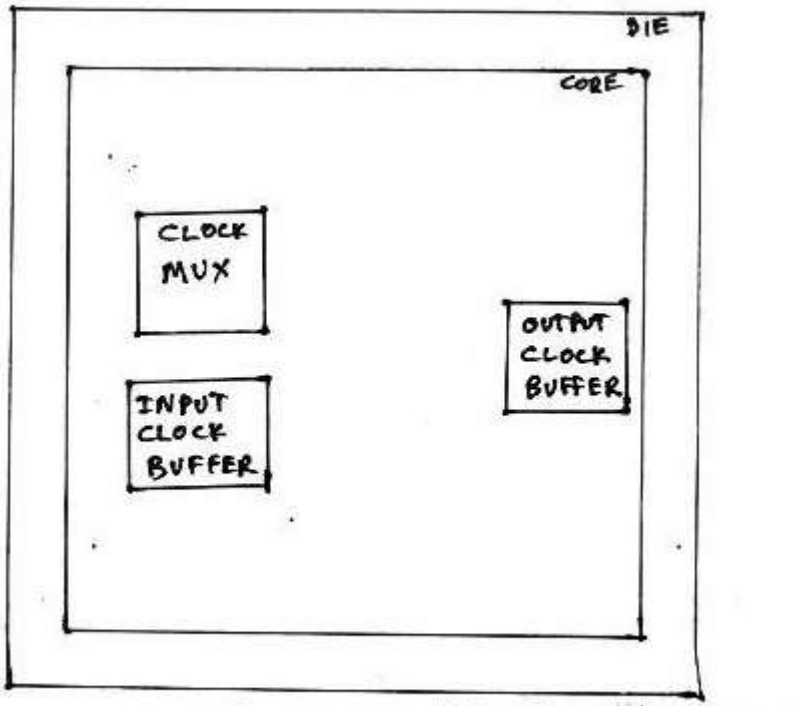


DEFINE WIDTH AND HEIGHT OF CORE AND DIE

The very first step in chip design is floor planning, in which the width and height of the chip, basically the area of the chip, is defined. A chip consists of two parts, ‘core’ and ‘die’.

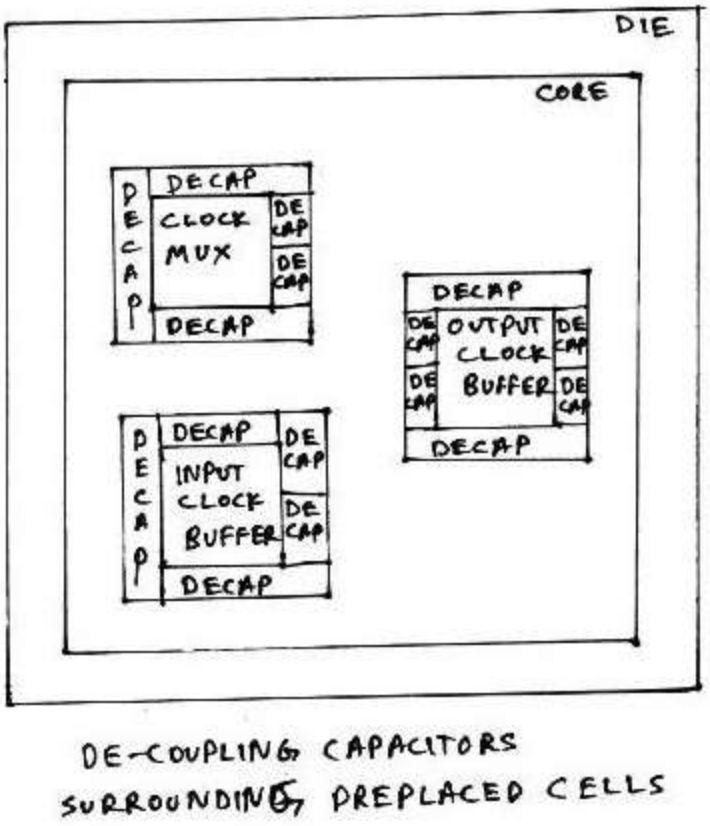


A ‘core’ is the section of the chip where the fundamental logic of the design is placed. A die, which consists of core, is small semiconductor material specimen on which the fundamental circuit is fabricated. IC’s are fabricated on a single 9 inches or 12-inch diameter silicon wafer, which contains hundreds of mirror images of the fundamental logic. This wafer is then cut into small pieces, each piece has similar functionality of the fundamental logic. This is called ‘die’



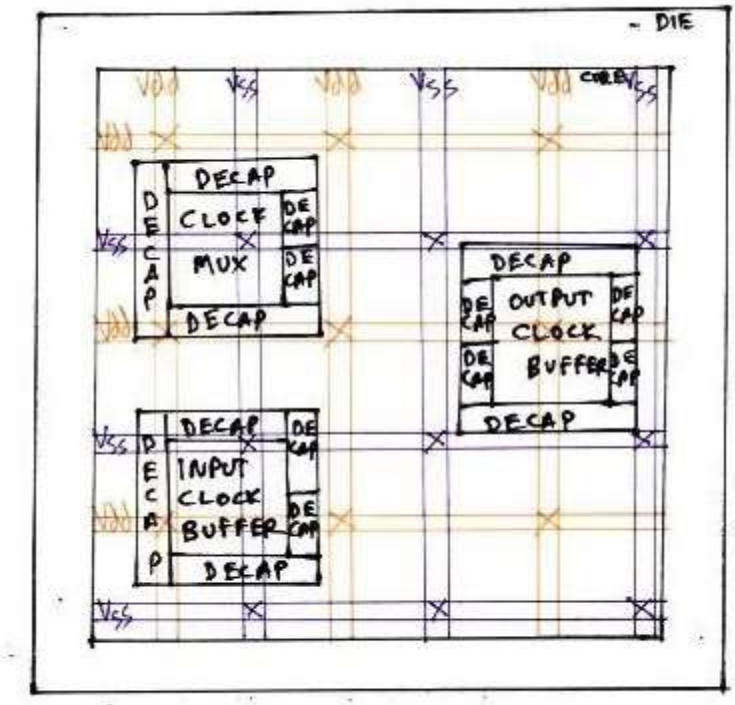
DEFINING LOCATIONS OF PREPLACED CELLS

During placement and routing, most of the placement tools, place/move logic cells based on floorplan specifications. Some of the important or critical cell's locations must be pre-defined before actual placement and routing stages. The critical cells are mostly the cells related to clocks, viz. clock buffers, clock mux, etc. and few other cells such as RAM's, ROM's etc. Since, these cells are placed in to core before placement and routing stage, they are called 'preplaced cells'. The above diagram describes the same.



DE-COUPLING CAPACITORS SURROUNDING PREPLACED CELLS

Once the critical cells are placed on the chip, it becomes necessary to surround the critical cells by decoupling capacitors. The placement of de-coupling capacitors surrounding the pre-placed cells improves the reliability and efficiency of the chip.

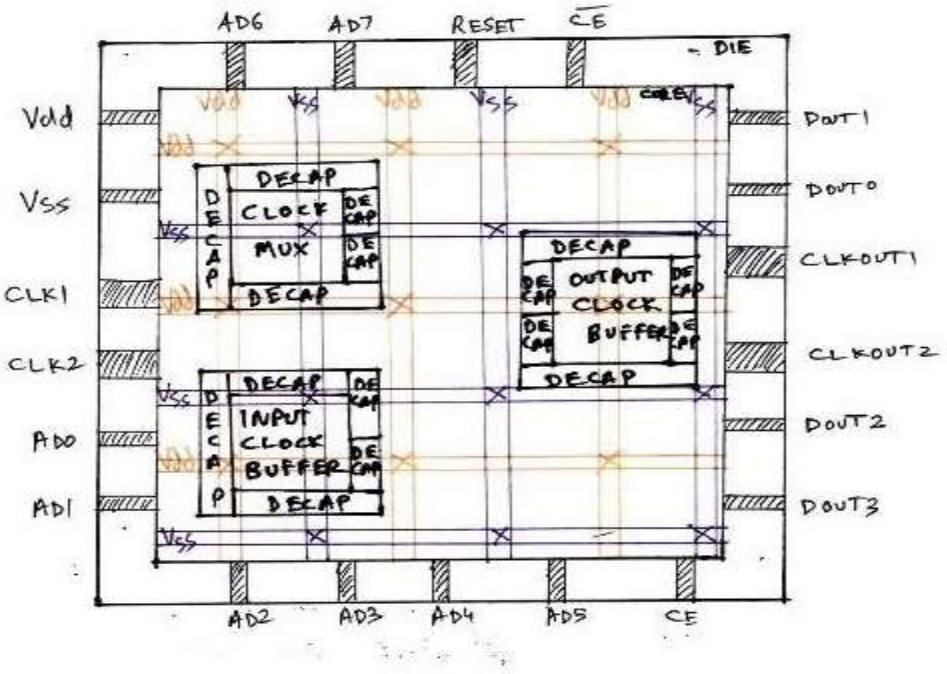


POWER PLANNING

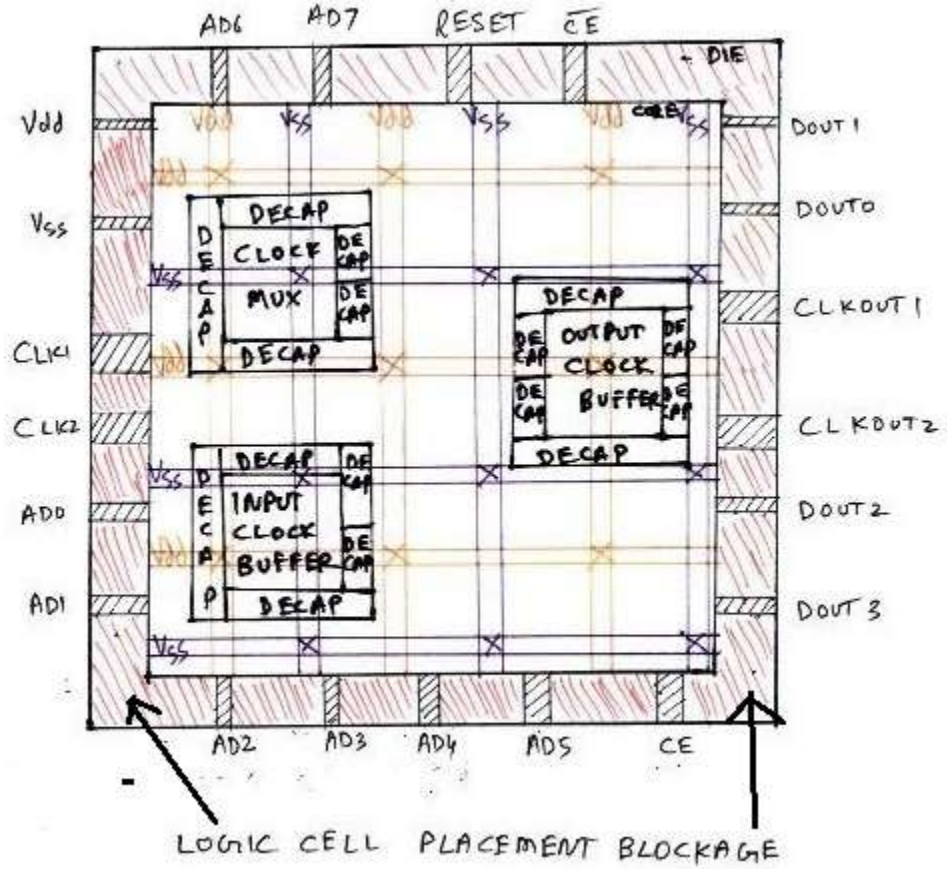
Usually, while drawing any circuit on paper, we have only one ‘Vdd’ at the top and one ‘Vss’ at the bottom. But on a chip, it becomes necessary to have a grid structure of power, with more than one ‘Vdd’ and ‘Vss’. The concept of power grid structure would be uploaded soon. It is the scaling trend that drives chip designers for power grid structure.

During floorplan, we define the width and height of both, core and die. The space between core and die is reserved for pin placement. For e.g. an 8085 has around 40 pins viz. reset, AD0, AD1, etc.

Also, the clock pins (for e.g. CLK1, CLK2, CLKOUT1, CLKOUT2 in above diagram) are wider compared to other pins on the chip. It is the clock on a chip that drives most of the logic inside the chip. Hence, it should have very low resistance, and thus wide area, as resistance is inversely proportional to area.



PIN PLACEMENT



LOGIC CELL PLACEMENT BLOCKAGE

To avoid the placement of cells by placement tools, in the area between core and die (which is reserved for pin placement), it needs to be blocked by logical cell placement blockages. It is very like blocking a road under renovation, so that no one drives on that road, and that road is reserved for some special purpose.

Once the floorplan is freeze, it is given as an input to the placement and routing (PNR) tools. These tools are built with intelligent algorithms which would consider the design requirements (usually called as 'constraints') such as clock frequency, timing margin, max capacitance etc., calculate the location of the logical cells (Flipflops, AND, OR, BUFFER, etc.) and place them in the floorplan. All the design requirements (or constraints) are stored in a single file called as 'design constraints' file, which is recognized by most of PNR tools.

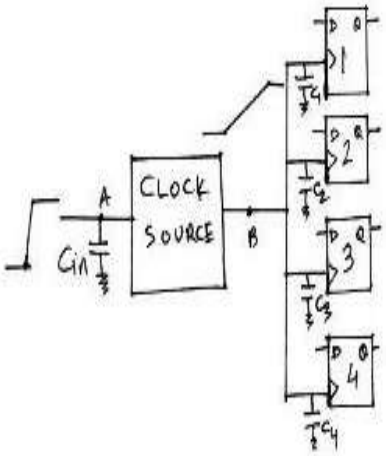
Let's talk about few examples on how the built-in algorithms of PNR tools behave after detecting design constraint. The inputs to PNR tools are the design netlist, floorplan, timing libraries and design constraints.

Timing libraries is a database that stores complete information about input capacitances, timing arcs, etc. of the logical cells. It also stores the list of all logical cells of different sizes.

Assume, the design has a constraint which specifies that **maximum capacitance on a net should not exceed 2F**. Now consider the schematic in the above diagram. Clock source has a node 'B' which is connected to the 'clk' pin of 4 flip-flops. Assume that the input capacitance at 'clk' pin of each flip-flop is 1F. So now, the PNR tool built-in algorithm calculates the total capacitance on node 'B' as addition of all input capacitance's at 'clk' pins of 4 flip-flops i.e. 4F. Then the tool compares this capacitance number with the **max capacitance constraint** in the constraints file which is 2F.

Since the capacitance at node 'B' is exceeding by 2F, the tool divides the load on node 'B' through 2 buffers as shown in right side of the above figure. It selects buffers from timing library in such a way that each of those buffers has an input capacitance of 1F, and builds a tree, called as 'clock tree'. The whole process of dividing the load on clock net is called 'Clock tree synthesis (CTS)'. Above example is one of the scenarios that is considered during CTS.

ASSUME $C_1 = C_2 = C_3 = C_4 = C_{BUF1} = C_{BUF2} = 1F$

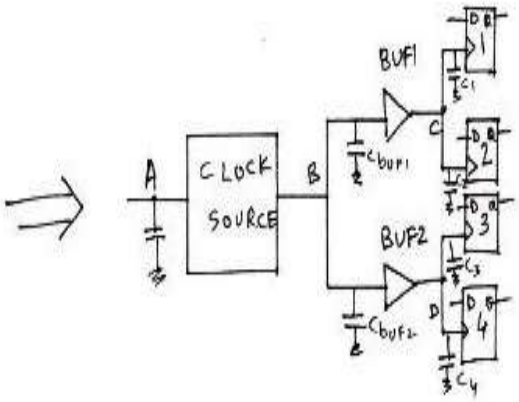


Total Capacitance at B ⇒

$C_{TOT_B} = C_1 + C_2 + C_3 + C_4$

$= 1 + 1 + 1 + 1$

$= 4F$



Total Capacitance at B

$C_{TOT_B} = C_{BUF1} + C_{BUF2}$

$= 1 + 1 = 2F$

Total Capacitance at C

$C_{TOT_C} = C_1 + C_2$

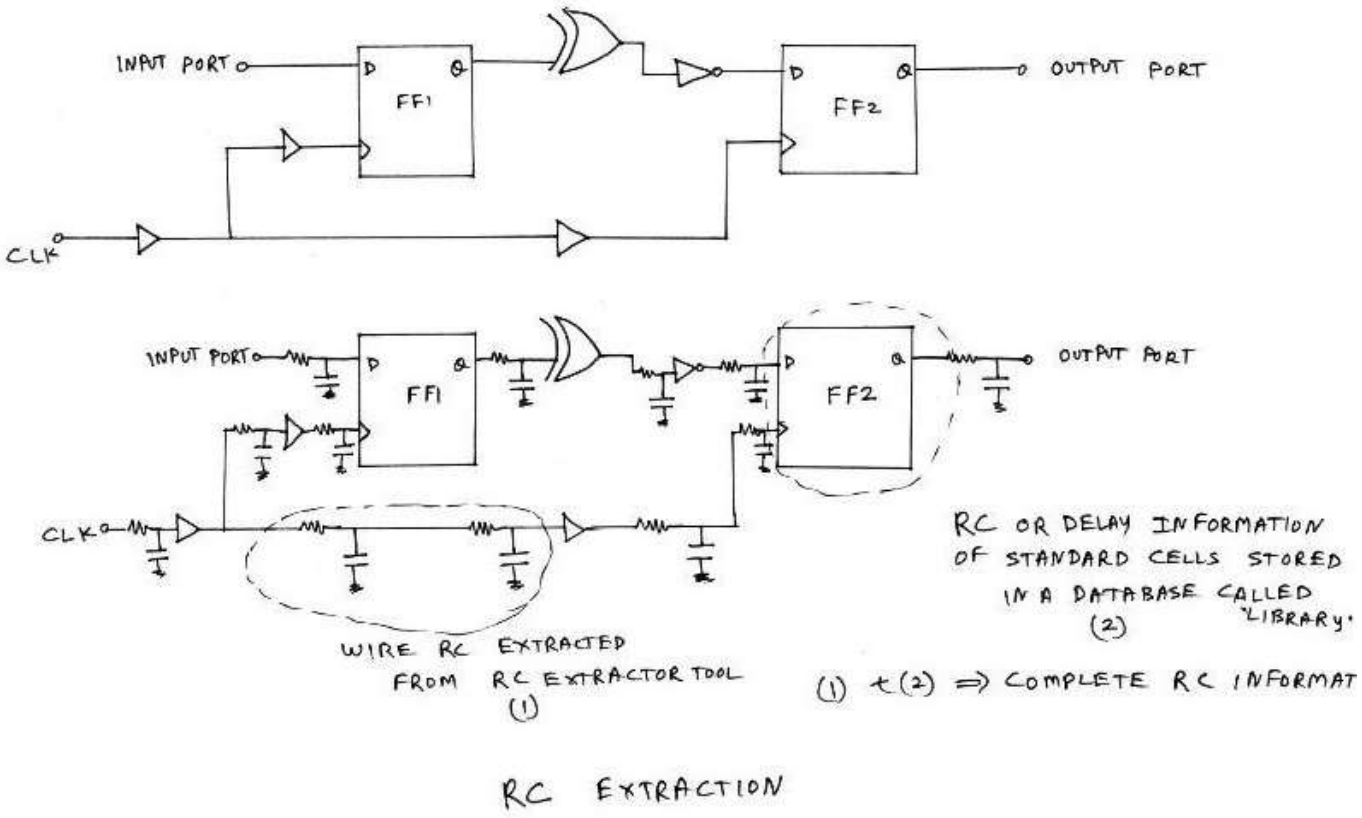
$= 1 + 1 = 2F$

Total Capacitance at D

$C_{TOT_D} = C_3 + C_4$

$= 1 + 1 = 2F$

CLOCK TREE SYNTHESIS



RC EXTRACTION

The PNR tools looks for any special physical requirements from user other than timing constraints.

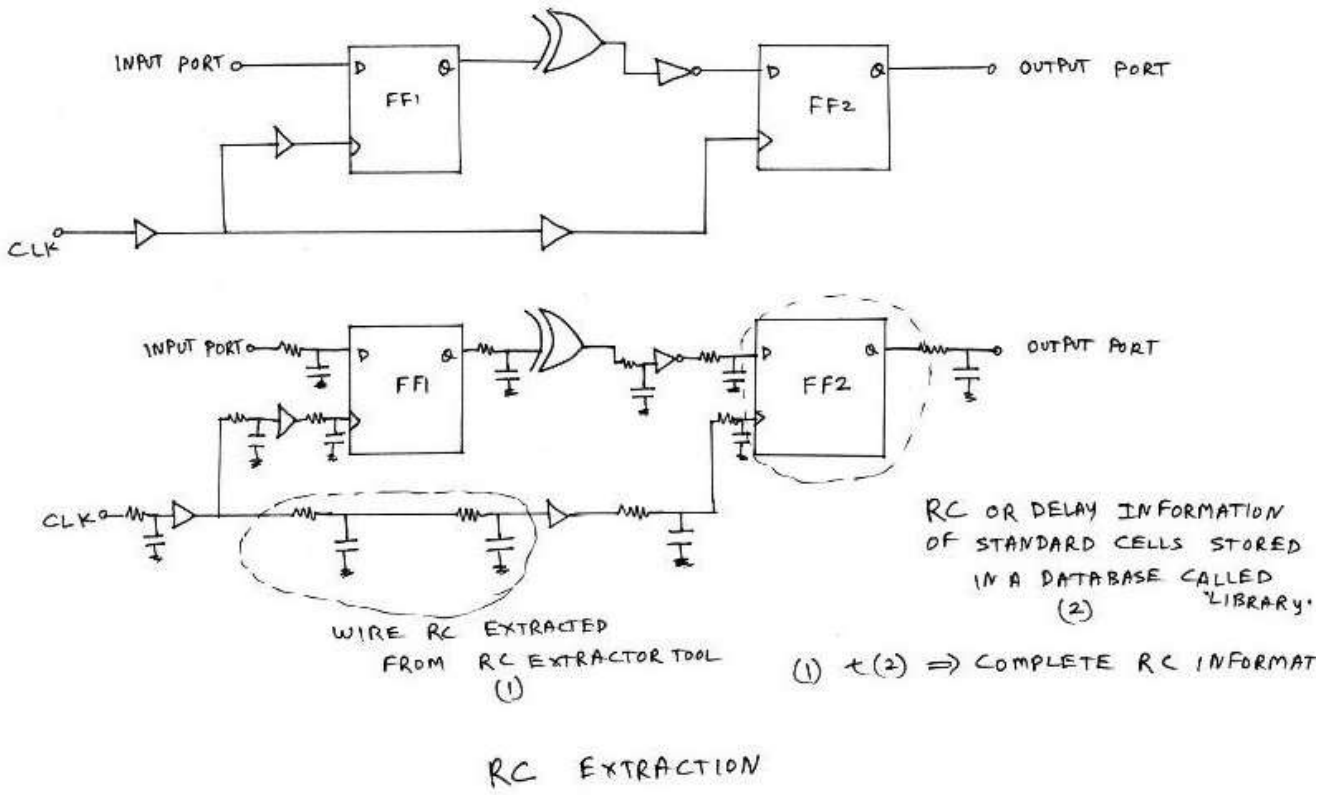
Let us create a scenario in which output net of clock buffer and output net of an AND gate is placed very close to each other, as shown in above figure. Also assume that output of AND gate is at logic '0', while the clock is switching at regular intervals. Consider clock switching from 'low' to 'high'. Since, the output net of AND gate is placed very close to the clock net, there is a high possibility that, during switching, the logic '1' might get coupled to the output of AND gate, which causes a bump with voltage 'V_{bump}'.

Since, a logic change or voltage level change on clock net, is causing a voltage level change on the nearby output net of AND gate, the clock net is called as 'AGGRESSOR' whereas the output net of AND gate is called as 'VICTIM'. If the bump voltage V_{bump} on VICTIM exceeds a certain margin or threshold, the output of AND gate switches to logic '1' which changes the functionality of the design. This phenomenon is called Crosstalk.

An efficient way to avoid the above scenario is to add a shield between VICTIM and AGGRESSOR which would break the coupling between them and hence logic level on the output net of AND gate would be retained. This requirement of adding shield around specific

nets could be fed as an external input to the PNR tool. Cost paid in the above scenario would be an increase in the chip area.

Finally, when design layout is complete, the PNR tool generates a new netlist which has information about any modifications done to the original netlist, for e.g. buffer addition, cell size changes, etc. It also creates a 'definition' file, which has the connectivity information between the logical cells, viz. wire length, width, locations, etc. This definition file is used to extract additional timing information due to wire inbuilt RC's (resistances and capacitance's) and store them into a separate file usually referred to as SPEF (Standard Parasitic Extraction Format) file. The design, which has the logical cells and the physical connectivity information between the cells, needs to be analyzed in terms of timing i.e. the design should meet timing constraints defined by user in the beginning of PNR. Plugging the SPEF information to logical design (which is the new netlist generated by PNR tool), the complete timing information of the design is fed as an input to any Static Timing Analysis (STA) tool.



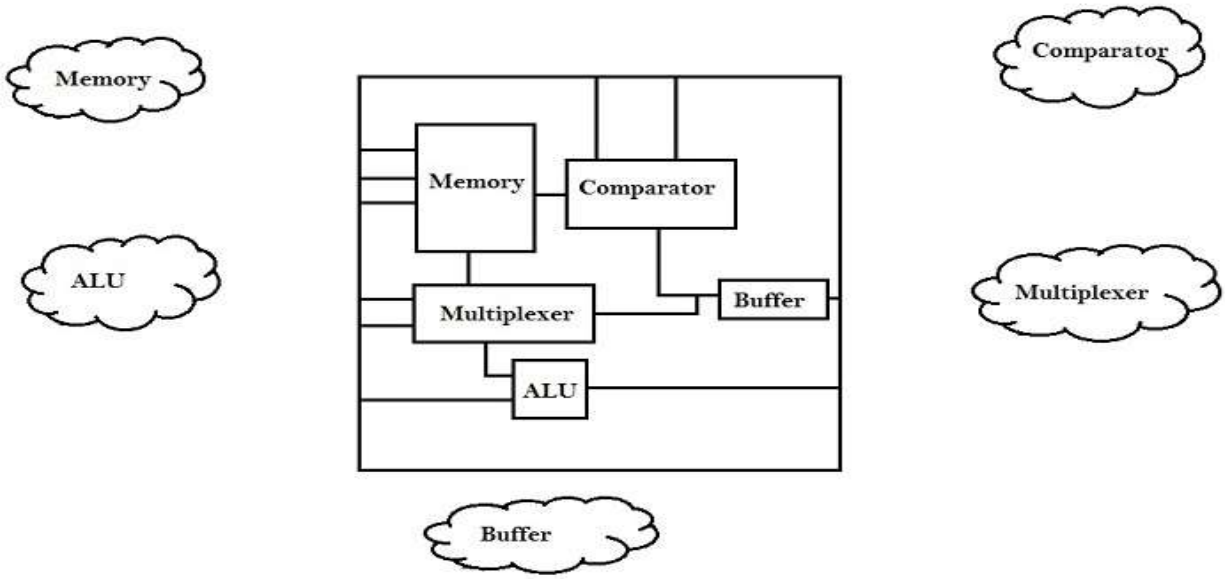
RC EXTRACTION

Static Timing Analysis (STA) tool helps to identify specific paths in the design which fail to meet timing requirements specified in the constraint file. These failing paths are flagged as 'VIOLATED'. There are 4 kinds of checks for which the design is tested viz. Setup check, Hold check, Max Capacitance check and Transition Check. Above figure displays the scenarios under which the timing analysis tool will flag or detect violations. Once these violations are detected, it becomes necessary to analyses these violations, and plug the fixes to these violations back to the PNR netlist. This process of fixing violations by modifying the routed PNR netlist is referred to as Engineering Change Order (ECO).

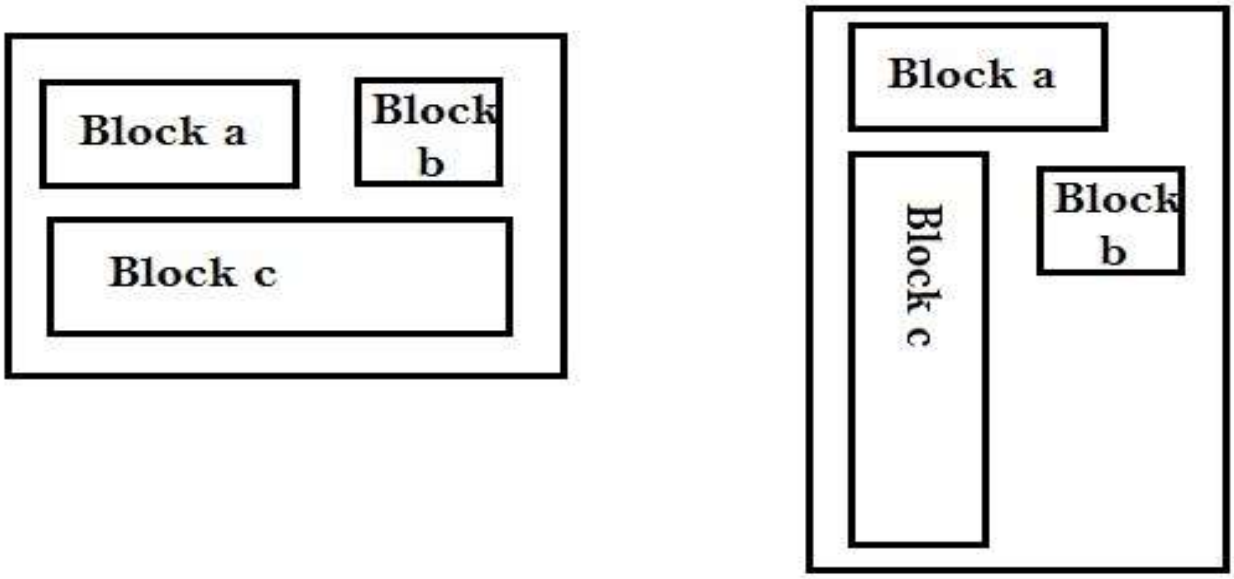
Floorplanning

Floorplanning is basically the arrangement of logical blocks (i.e. multiplexer, AND, OR gates, buffers) on silicon chip.

Consider a large design like microprocessor. The abstract level behavioral description of the processor is written using an RTL program. Large designs (e.g. microprocessor in our case) are usually synthesized into small modules. These modules are the basic building blocks of a microprocessor e.g. memory unit, adder/subtractor (ALU) unit, multiplexer unit, etc. Consider the following diagram



Modules are called as blocks, when it is assigned an appropriate area and aspect ratio (Height[h]/Width[w]). Blocks are usually classified as 'soft' and 'hard' blocks. In hard blocks, the area and aspect ratio of the blocks are fixed, whereas for soft blocks, the area is fixed but aspect ratio could vary. The placement of these blocks on a chip decides the size of the chip. This would be clear in the following diagram.



Two possible arrangements of blocks a, b and c

Consider, we have three blocks, a, b and c, as shown in above diagram. We have two possible arrangements of these blocks. The arrangement shown in left occupies minimum area, whereas the one on the right occupies larger area on chip, and hence facilitates the user to add more blocks (i.e. additional functionality) in to the chip. Now, of the above two arrangements, which one is selected, depends upon the end-user specifications. If, the specifications demand for minimum area, the left arrangement is selected, whereas, if the specification demands decent area as well as additional functionality, the second arrangement is selected.
Hence, floor planning makes sure of 3 things:

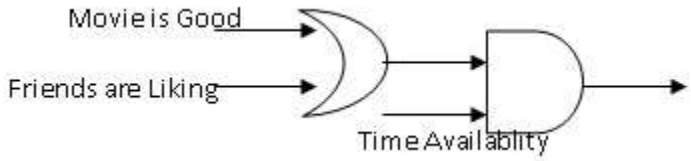
1) Every module has been assigned an appropriate area and aspect ratio.
2) Every pin of the module has connection with other modules or periphery of the chip.
3) Modules are arranged in a way such that it consumes lesser area on a chip.

Combinational Logic

Logic is the process of decision making based on some conditions. Combinational Logic refers to the decision making combining one or many conditions. The entire decision-making process when divided into fundamental decision-making blocks, they become the basic logic gates which are And, Or, Not, Nor, Nand, Xor, X-Nor. All these decisions are made based by associating Present/Absent or Yes / No or True/False to the individual conditions.

GATE	Decision
NOT	Inversion. (i.e. Condition Not present)
AND	All the conditions are present
OR	Any of the conditions are present
XNOR	Equality of Conditions
XOR	The conditions are Mutually Exclusive
NOR	Not even a single condition exists
NAND	Not all condition exists

From this any decision making, can be constructed from these basic decision-making elements. For e.g. If you want to go for a movie, the decision will be made like follows. (Movie is Good or Friends with you want to go) AND (You are free time to spend for movie).



Here again the Movie is Good, is a decision collective of many factors, like The Star Cast is Good OR Screen Play of the movie is Good OR Story line of the movie is Good. Like this each condition can be further evaluated. This is a classical scenario of a decision-making process. This is extensively used in the Digital circuit design as all the logical decision made in the Digital Domain are of the similar nature. E.g. If you want to create a logic to send some information to a receiver, it will be like The Information is available with you AND the Receiver is ready to accept the information. If this condition is met then you start sending the information.

Need for Sequential Logic

The above scenario of decision making, considers the present conditions existing only. This cannot be sufficient always to make a correct decision.

For e.g. If we consider the decision of a movie described above, you may want to go to a movie which you have not seen. So, the decision will be made based on the present existing factors as described above and from your PAST, that is you have not watched the movie. The fact that you have not watched the movie can be considered as a state.

Which means you decide collectively based on the present prevailing conditions and the state you are existing.

Hence there is a necessity to store these state information as it should be made available for the future decision making. Here comes Flip-Flops which are basic 1-bit storage elements for this purpose.

Finite State Machine

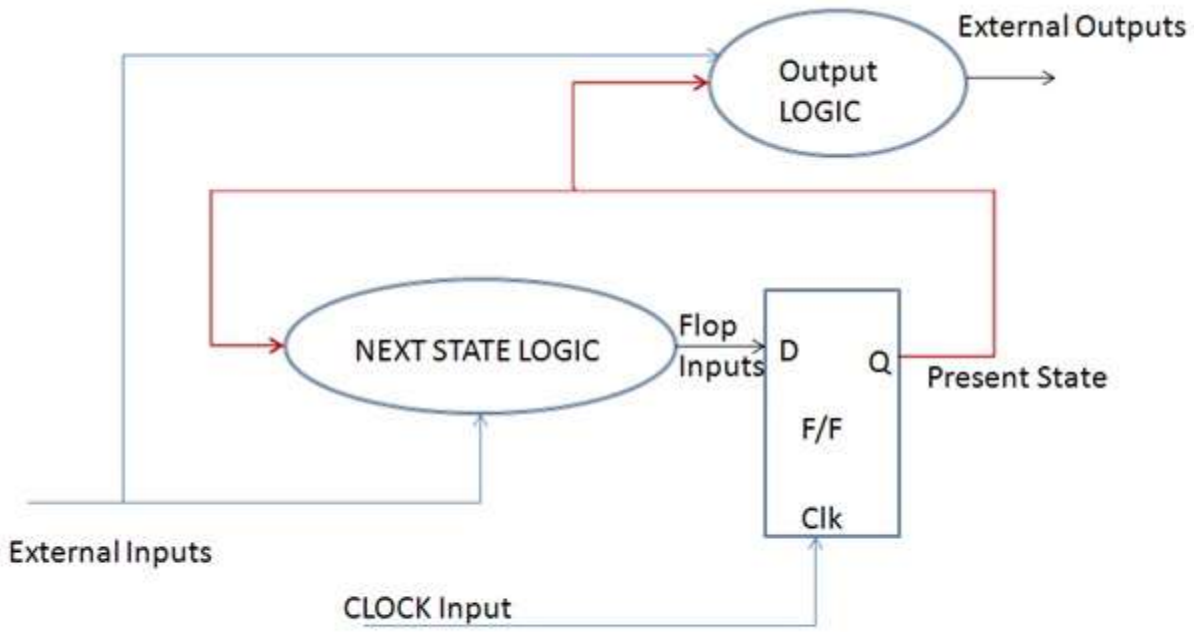
From the example above, the present state will be a factor for deciding the next state along with the present inputs. The same can also be told as the present state was derived from the past state with the present inputs. Both are same… it is just a play of words!!

So…the question is what is a state and how am I going to represent it??

The answer is simple. States are different stages in the decision-making process flow. In digital system, any information is represented or stored as 0s and 1s. So, the states are also represented in a binary code in the form of 0s and 1s.

These 0s and 1s are in form of a code where each bit of 0 or 1 is coming from a flip flop.

This entire information can be summarized in the below diagram.



General Block Diagram of a State Machine

The F/F acts as a storage element to store the state information.

There is a decision-making logic called the Next State Logic. It is to be noted that the inputs to the Next State Logic (NSL) are the present state (shown in brown) and the external input (shown in blue). Considering both, the decision for the next state is made. The decision made is latched by the flop on the clock edge.

So, in our example, the "Movie not seen in the past" comes from the flop and the other inputs like the "Movie Reviews are good, etc., etc." comes from the external inputs. Considering both the factors together the NSL decides whether to go to movie or not and this information is latched by the Flop on the clock edge. If the decision for watching the movie is taken the NSL generates '1'. which means the flop input is now '1' and the state is updated as "Movie Watched" …. So, this will be a driving factor in the Next decision-making process.

State Machine: A state machine is a digital system working on a predefined finite state. So, they are also referred as Finite State Machine or FSM in short.

Case Study: Let us understand the need for a state machine and what it is based on a case study.

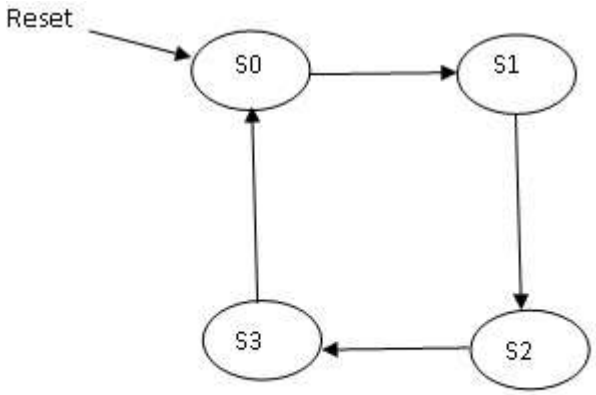
Let us consider the scenario of a 2-bit Up counter.

This machine is a counter which can count in the ascending (Up). The machine is a 2-bit machine. So, it will need 2 flops (1 Flop per bit).

[Remember a machine having n flops can have 2^n states (each flop can be 0 or 1 leading to 2^n permutations)].

On reset the machine is at "00". After reset, the machine starts counting on each clock edge. It will count as 00, 01, 10, 11 and after this again 00. So, the states are finite and hence the name FSM. After this the states keep repeating in the same sequence. If I generalize the states as S0, S1, S2, S3 instead of 00, 01 ,10 ,11 … then the machine will be repeating the states S0, S1, S2, S3, S0, S1,

This can be diagrammatically represented in a diagram called the State Diagram. In the state diagram, each state is represented by a bubble and the transition from one state to other is represented by an arrow. (The direction of the arrow is important as it shows from which state to which state the transition is happening. This is important because the Transition from S0 to S1 is totally different from the transition from S1 to S0 state).



Please Note: Upon Reset the state machine is initialized at S0. This is very important. Upon reset any state machine must be initialized to a **Known Valid State**.

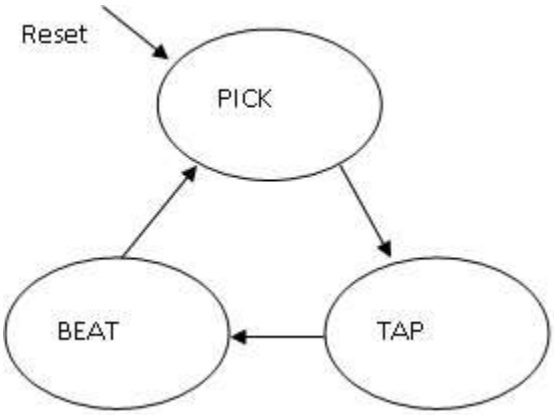
Can there be an Unknown State??

Ok Let us understand this clearly. There are 2 reasons why we must define the reset state of the machine.

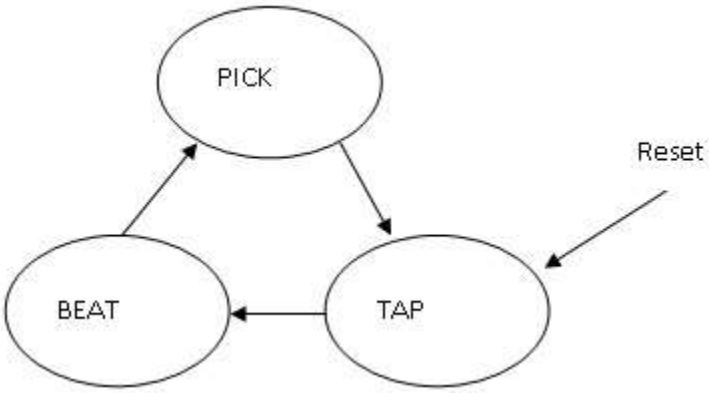
Reason1: Let us take the example of a machine which will beat the EGGs and give us. Upon reset, the machine will **pick up** an egg, then **tap it** (break it) and take the yoke out of it in a bowl and then it will **beat the yoke** in the bowl. This is the sequence the machine is going to operate. (states are indicated in bold).

Now if upon reset I leave the machine in the tapping state instead of the Picking up the egg, even without picking up an egg, it will try to tap it and break it. Two things can happen here, either the machine will wait for Egg in this state (it has not picked the egg, so it will wait for the egg) and it will hang here or it will miss the sequence and something that is not expected out of it will happen.

So, the reset state of a state machine is very important and must be defined clearly to a known valid state.



This is the correct sequence and the reset state of the machine is defined properly.



This is the in-correct sequence and the reset state of the machine is defined WRONGLY. The machine will fail to perform the expected behaviour.

Reason 2: Now let us take the example of a counter which needs to count 0 to 5, like 0,1,2,3,4,5,0,1,2, 3, Upon reset the machine should be set to state 0(known state). What is an unknown state then?? Ok …. From the implementation point of view …there are six states in the machine and we need 3 bits for implementation. But a machine which has n bits will have 2^n states which means here there are 8 states. But we are designing the machine to operate only in 6 states (0 to 5). Which means there are 2 unused states (6 and 7) or undefined

states for the machine. The machine will not know what to do in this state as we have not designed it for these states. So, it will hang or preform some totally unintended operation.

This can be very serious…. For e.g. Let us take a case of a digital system which is used for Missile Control.

There is a state machine which decides when to fire the missile. Suppose the machine goes to undefined state and fires the missile in the own base itself!! …It is comedy to read it here but in real life it will be a tragedy!!!!!

So, a good designer should always provide a means to initialize the machine in known state and if there are unused state and it is entered because of some error conditions…there should be a way to reach the initial state. Else it will be disastrous!

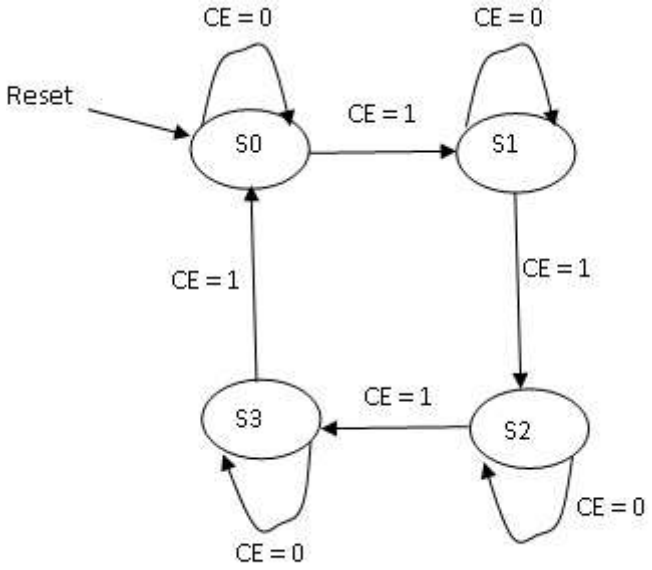
State Transitions: The change from any state to its corresponding next state is called a state transition. The transition from one state to another state can be a conditional transition or an unconditional transition.

What does this mean…?

Let us understand it with an example. Let us take the scenario of the same 2bit counter, the intent is to count only when the Count Enable is high, Else the counter should retain its present state.

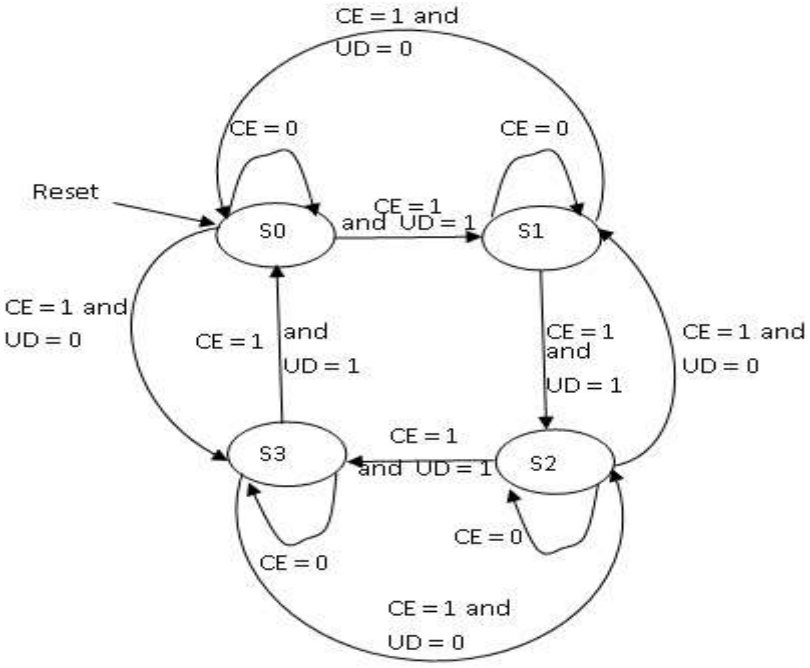
Now how to represent this scenario in the state Diagram…. Now from the given scenario it is very clear that the transition from the state is based on a condition [Count_Enable = 1], these types of transitions are called Conditional Transitions in a State Machine. The Transitions which we saw earlier are Unconditional Transitions. The Unconditional Transition can occur on every clock edge, but conditional transition can happen on the edge of the clock where the condition is met.

Let us see, how this diagram looks. (the **Count_Enable signal is shown as CE in the state diagram below**)



Now let us complicate the situation and see. In the above scenario, there is only one condition for every transition. But there can be more than a single condition to decide the transition from one state to another. The counter is a 2 bit Up-Down counter with Enable. When the counter is Enabled it should count (Ascending order) or Down (descending order) depending upon the UD signal. If UD = 1, it should count and if UD = 0 it should Count Down.

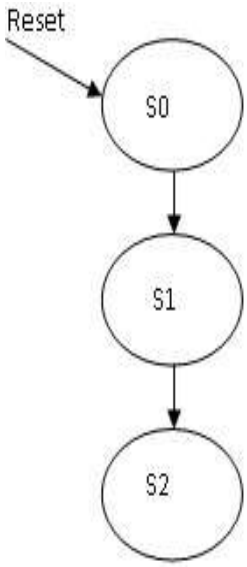
So, the state diagram must accommodate these conditions also….



From the state diagram, when the Count Enable (CE) is low, there is no transition in state irrespective of the value of Up-Down (UD). Hence the value of UD is NOT indicated in these cases.

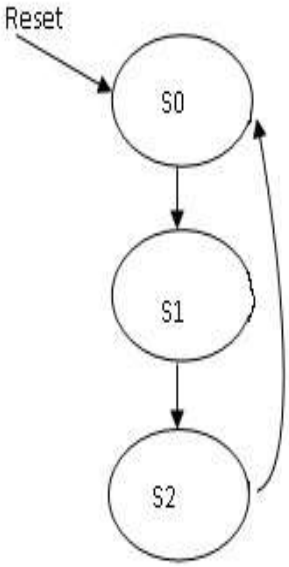
Whereas the value of UD is considered to Count Up / Count Down when CE = 1. Hence for the transitions when happening CE = '1' is indicated with the value of UD.

Please Note: The state diagram is a closed diagram…meaning the path from any bubble to any other bubble is a closed path. There should not be any state where the machine gets dead-locked, also there should not be any state for which the next state is not defined.



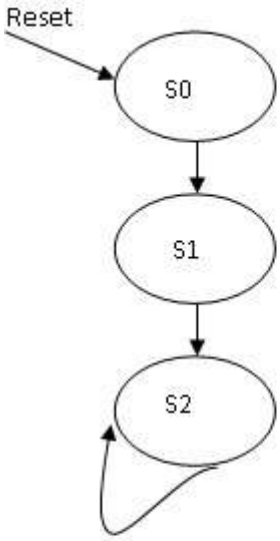
Incorrect Design:

Because the next state is not defined for state S2 . Machine is stuck at this state



Correct Design:

For every state the next state is defined. So the machine is not stuck at any particular state



Incorrect Design:

There is no way to come out of state S2. The machine is permanently locked there.

This is incorrect. The machine can be stuck in a state based on some condition but not perpetually.

Thestate machine also should not get inot deadlock condition. There are cases in which there are 2 state machines interacting with each other. Consider a scenario in which there are 2 Receiver-Transmitter pairs interacting with each other namely A and B. Both A and B are having state machine which is implemented to take care of the communication protocol. A is waiting for B to start the transmission and B is waiting for A to start. What will happen. These types of scenarios are called DEAD LOCKS and should be AVOIDED!!!

Example of a state machine with unused state: Let us consider the case of a state machine having 3 states (S0, S1, S2). Let S0 be the initial state. There are 3 states, which means we need a minimum of 2 bits per state, else it is not possible to have 3 states.

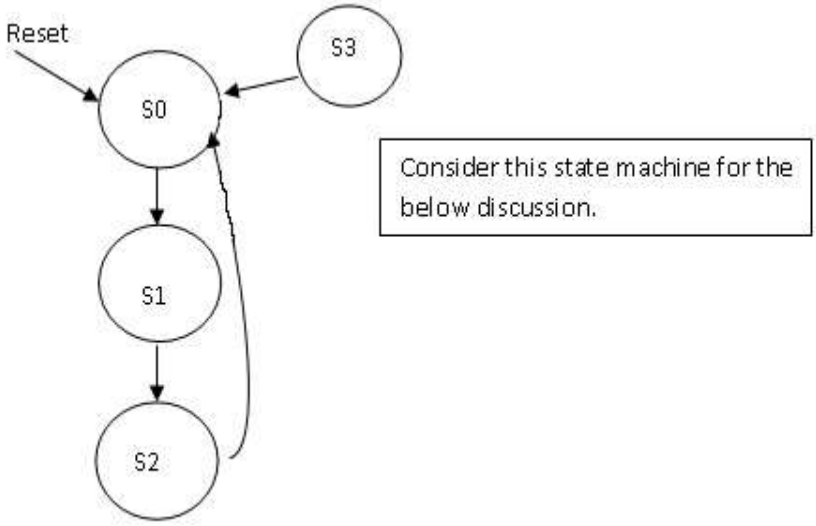
But if the states are represented in 2 bits, then there will be 4 states, which means that there is one unused state. Now the machine should not enter this state under no circumstances. In case if it enters because of some malfunctioning, it should not be stuck up there. How to do this???

This will be done by redirecting the state from any of the unknown state to a known initial state. (Note: We must point to the initial state, because we might not know if we go to some other state, it might not be meaningful…!!! Will elaborate this below).

Why we should go to initial state in case we reach unused state?

When the state machine lands in an unknown state, then it is a corruption of state. This cannot happen in the normal working condition of the state machine, because the machine is not designed to land in this state. But then how it can land in this state??? Ha! That's the point to understand. That's why I mentioned it as corruption of state earlier. That is because of one

or more state bit got flipped from 1 to 0 or vice versa due to some noise or disturbance in the surrounding. But there is no mean to say which bits have got flipped. This means we cannot say from which state we landed here. So, the best thing to do under such scenario will be to start from the initial state.



Confused??!! Ok …let me elaborate it. Consider the above state machine. Let S0 be "00", S1 be "01" and S2 be "10". Let S3 be "11" which is the unused state. Now there is a transition from state S1 to S2 that is the state bits are changing from 01 to 10, but because of some noise and error it became "11" which is state S3(here unused state) or it may also happen while changing from S2(10) to S0 (00) it got errored to S3(11). There is no way to say from which state we ended up in S3 as this is not defined for this machine. We can say that we reached S2 from S1, but we cannot say from where we reached S3. So, it is always better to reset it.

Now you may ask me, that whatever I am talking above can happen with used states also. That is while changing from S0 (used state) to S1 (another used state) there was error so it went State S2. Yes, it is quite possible and the state machine will fail to perform the required function. Now how to take care of this?

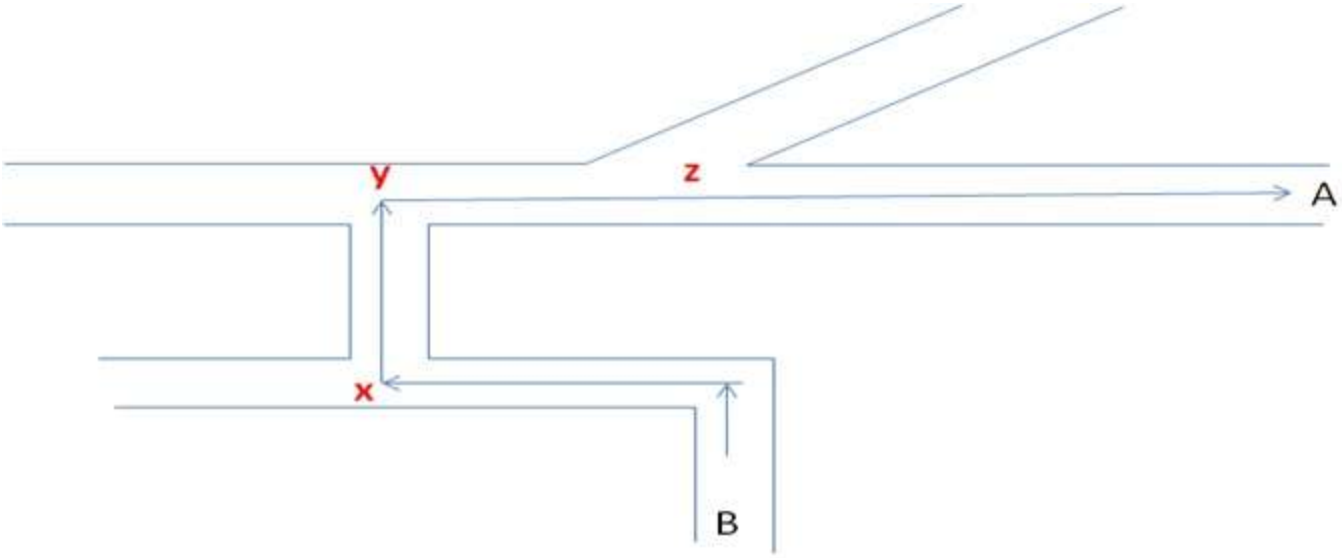
For understanding the above point, we need to understand the exact application of the state machine as to where it is used and how to implement a proper state machine.

Ok…let me not build too much suspense. From the discussion, so far one thing is clear. The state machine is a very good way of implementing decisions. That is using a state machine,

the decision is made using not only the present inputs but also based on the past state. So, the best application for the state machine is for decision making process. State machines can be used for much bigger complex decision-making process…basically they become the core of the control logic (Brain) of a digital system.

HOW??????

Let us understand this point with a real-life example



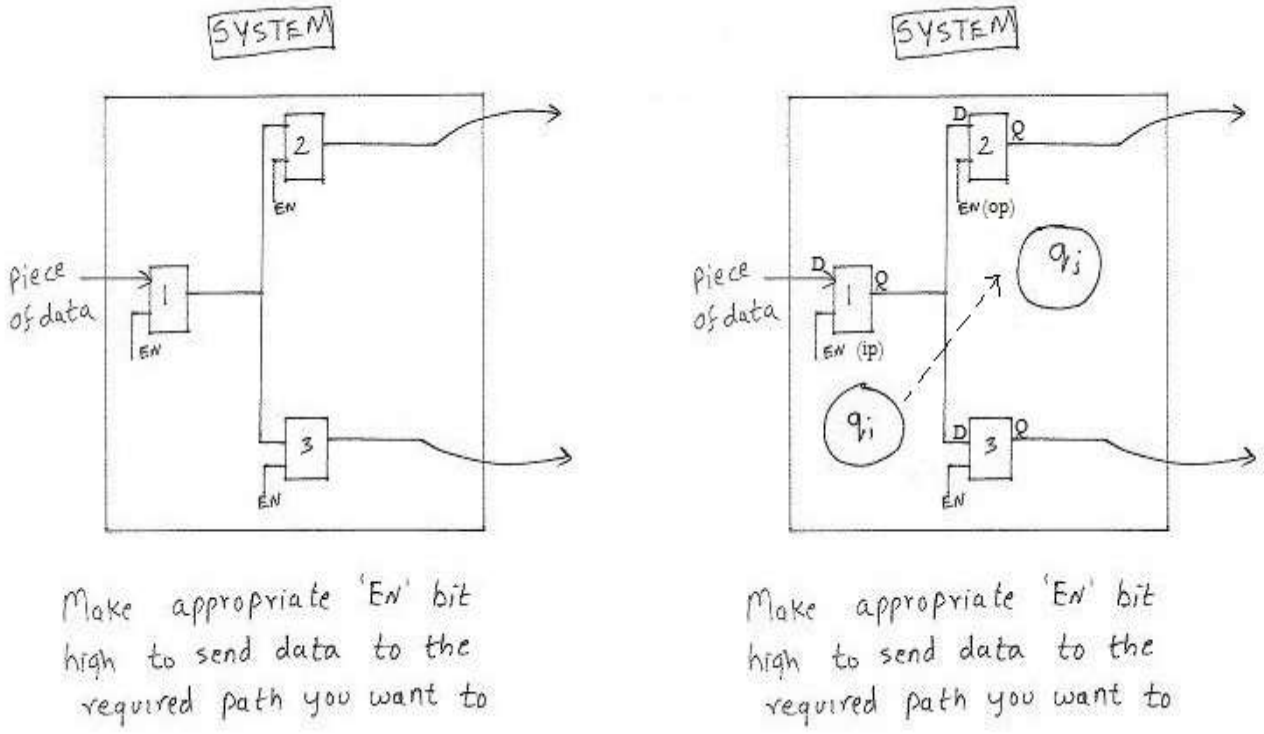
Consider the map shown above. Assume that you are standing at point B and you must travel to point A. It is very simple. You travel as shown take a right turn at point x and y and at junction z you don't turn and move straight. It is not as simple as it appears. You reach A from B if and only if You did not miss the turn x, made a proper turn in required direction at y and by not making a turn at z.

Now let us relate the same to a digital system. Let us now think A and B as some buffers in a system. The path is shown is a connection say a Bus. Now data from B must be sent to A. The information should take the same path as shown in the map. But the data is not having a brain like us to think as to where to turn and where not to turn. So, the data from B must be sent to A and only to A.

How will this happen?

There must be some logic, to steer the information from B to A. It should also take care to prevent the information not going to undesirable places.

The notion of Data Path and Control Path in a Digital System: With reference to the above example, the path in which the data traverses in a digital system is called the Data path. Now the data to travel in the desired path and to perform the required operation on data there needs to be some other signals generated. These are called the Control Signals. These are generated by the state machine. Hence the state machine is referred to as the brain of the digital system.



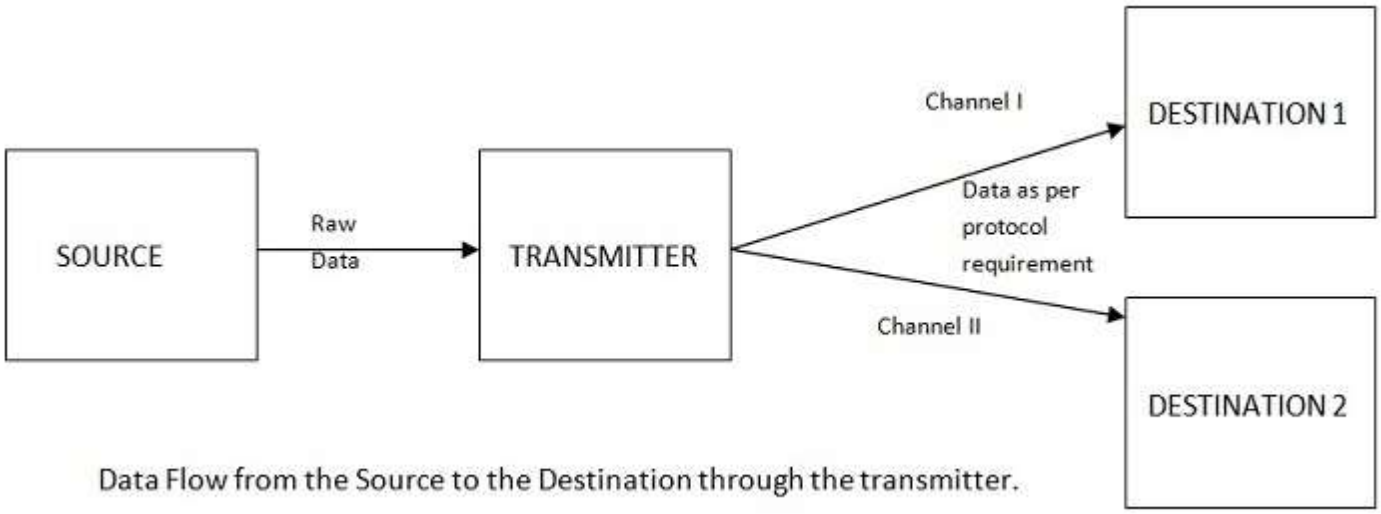
The same is explained in the figures above. The data from register 1 must be moved to register2. This is done by enabling the register1 and register 2 and more important by not enabling register3. Thereby we steer the data from register 1 to register 2. This can be associated with a state change as shown above from state Qi to Qj.

Let us do a case study.

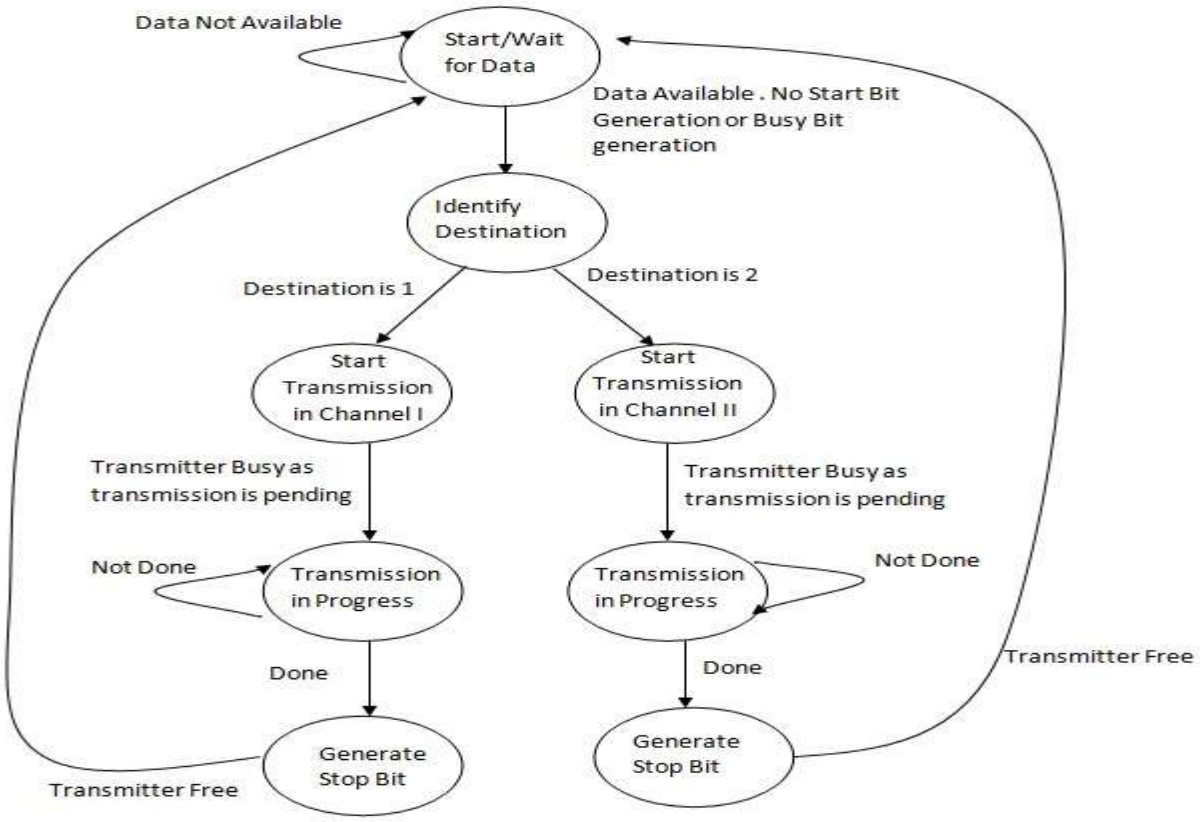
Let us discuss about a transmitter. The block waits for the data to transmit. The requirement is, it takes the data from the source and transmits it to destination1 or destination2 based as requested by the source. It should notify the sender to wait till it finishes the transmission before the sender can put a new data (else the data will be missed). The sender gives only the data. The transmitter takes care of generating the start and stop bits before the start of

transmission and after the transmission is over. Basically, the transmitter takes care of the protocol to be followed.

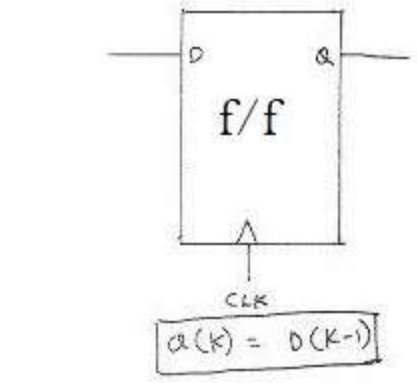
Let me describe this pictorially.

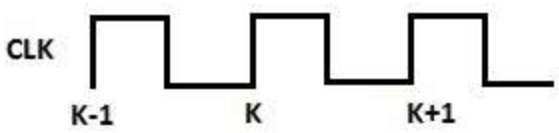


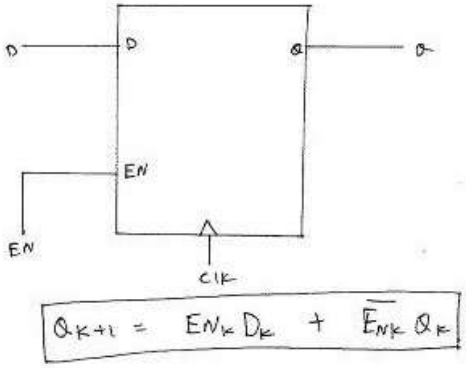
Data Flow from the Source to the Destination through the transmitter.



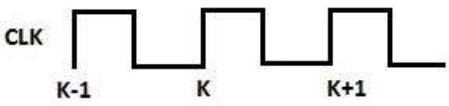
Here we realize the actual power of the state machine. The requirement was to move a data from Source to Destination1 or Destination II. But we see that there are lot of auxiliary signals required at different instance to complete the task. The state machine comes handy in these situations.





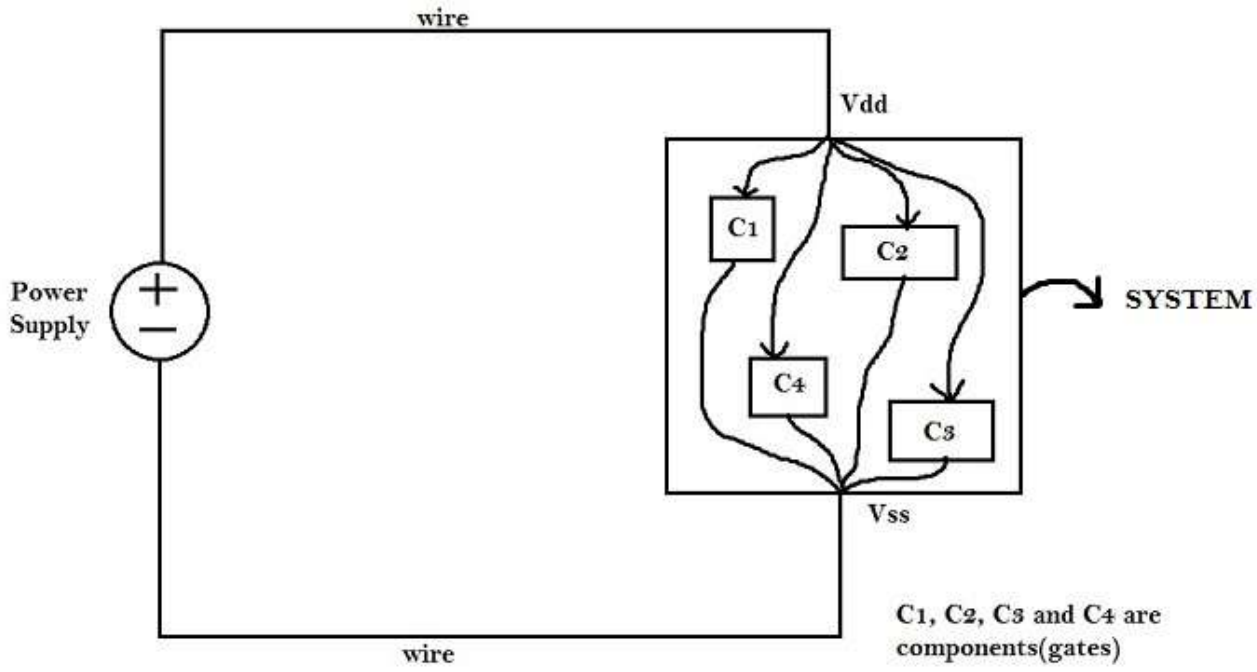


EN / Q	Q(K+1)
EN(K) = 0	Q(K)
EN(K) = 1	D(K)



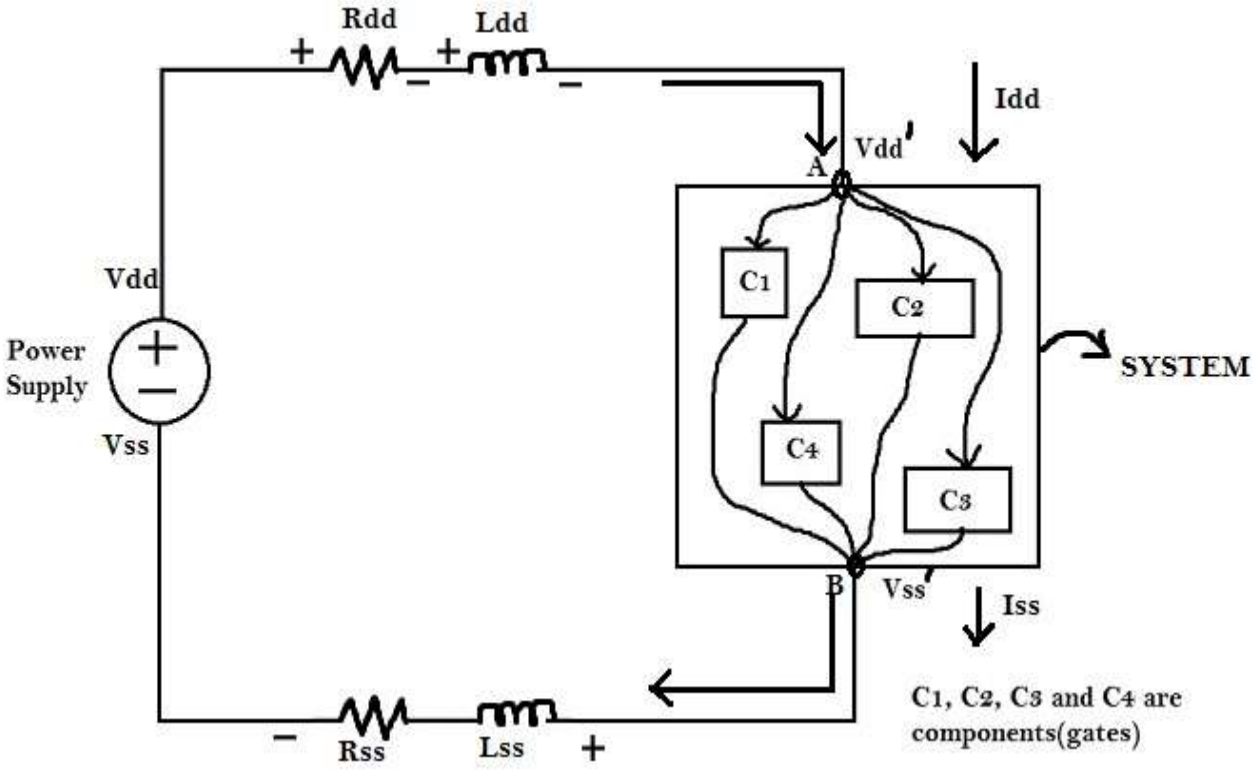
Concept of Decoupling Capacitors

A decoupling capacitor is a capacitor, which is used decouple the critical cells from main power supply, to protect the cells from the disturbance occurring in the power distribution lines and source. The purpose of using decoupling capacitors is to deliver current to the gates during switching. Herein, we would peep inside the reasons for the disturbance occurring in the power distribution lines. Observe the following figures.



If the wires were ideal, i.e. ‘zero’ resistance, ‘zero’ inductance and infinitely short, thus no issue of power distribution. To get the desired current during circuit operation, the source impedance should be minimal to do so.

Assume the ideal supply, and include wire resistance and inductance in the figure below.



For time being consider capacitance to be zero for the discussion. R_{dd}, R_{ss}, L_{dd} and L_{ss} are well defined quantities. During switching operation, the circuit demands switching current i.e. peak current (I_{peak}). Now, due to the presence of R_{dd} and L_{dd}, there will be a voltage drop across them and the voltage at Node 'A' would be Vdd' instead of Vdd.

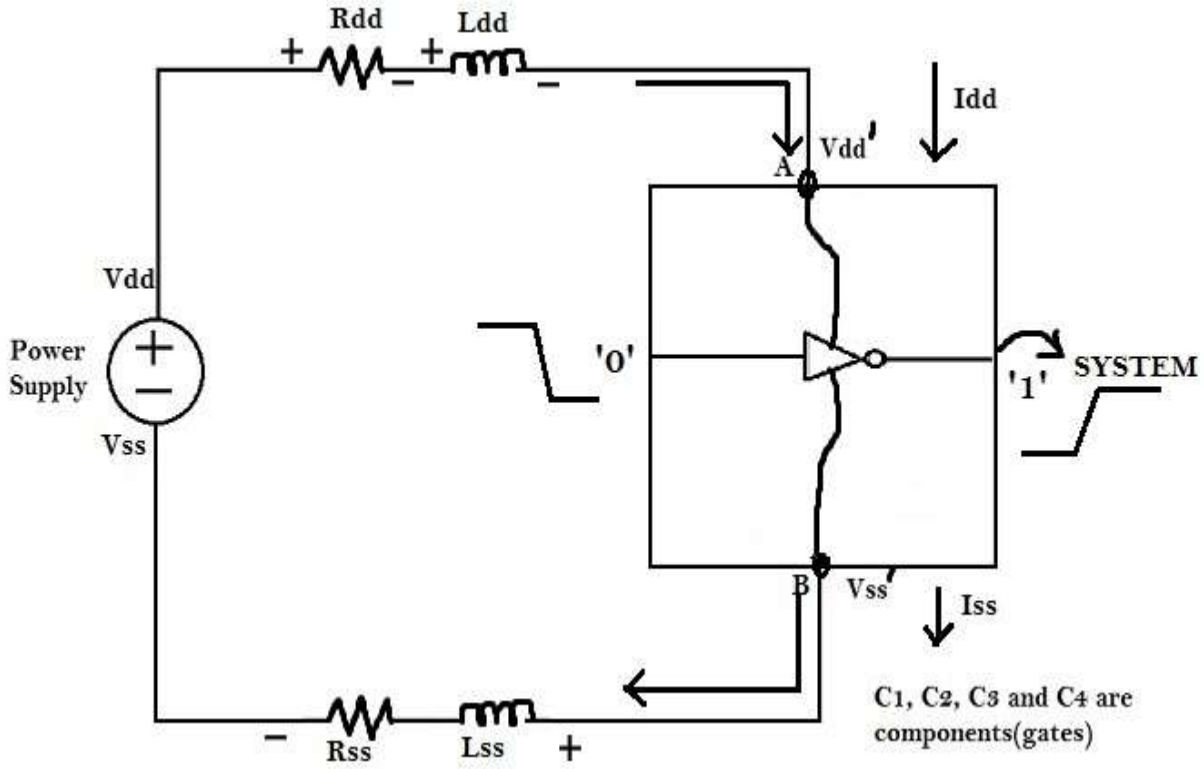
$Vdd - Vdd' = I_{peak} * R_{dd} + L_{dd} * (dI/dt)$

Similarly, same equation holds for ($V_{ss}' - V_{ss}$) but current will be in opposite direction. Hence, now node A will observe a disturbance i.e. lowering of power supply voltage, commonly referred to as **'voltage droop'**. Also. Vss', which is supposed to be at 'zero' potential, will be at some +ve potential, due to R_{ss} and L_{ss}. This is commonly referred to as **'ground bounce'**.

Now question is **"What all can go wrong due to this disturbance?"**

If the components were flip-flops, designed to operate at 1 V , and due to R_{dd} and L_{dd}, the voltage drops down to 0.6 V during switching, it's possible that flip-flop doesn't function correctly.

And also logic ‘1’ would no longer be a logic ‘1’, since the circuit might have exceeded the noise margin. To understand this, consider the following diagram:



When input of the inverter switches from logic ‘1’ to logic ‘0’, output of inverter should switch from logic ‘0’ to logic ‘1’. This essentially means that the output capacitance of inverter should charge till the supply voltage Vdd’. But if Vdd’ goes below the noise margin, due to R_{dd} and L_{dd}, the logic ‘1’ at the output of inverter won’t be detected as logic ‘1’ at the input of the circuit following the inverter.

Root cause of the problem is the disturbance occurring in the power supply line. As a system designer, you must assure three things

1) The noise injected is small enough that it doesn’t overcome the noise margin.
2) Local Power Supply should be healthy and create less disturbance in the performance of the Circuit.

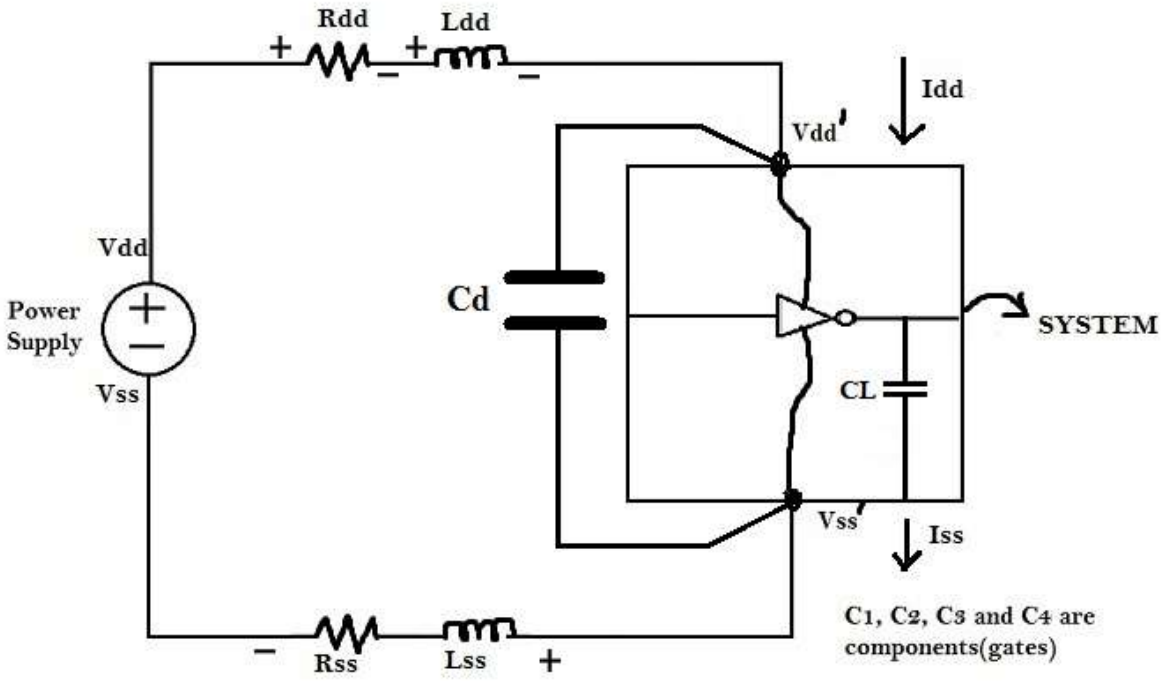
Solution
1) Keep R_{dd} and R_{ss} minimum by increasing width of wire.

2)Keep peak current I_{peak} and change in current dI/dt as small as possible.

$I_{peak} = C_L * V_{dd} / t_r$ $dI/dt = C_L * V_{dd} / t_r^2$

Hence, limit the rise time (t_r). If a circuit could run at 500 ps, it's unnecessary to run the circuit at 300 ps. The largest possible value of t_r should be selected. From above equation, I_{peak} and dI/dt also depends on load capacitance i.e. C_L. Hence, reducing load, will also reduce I_{peak} and dI/dt i.e. judicially decide the loading and the circuit components, as in turn it will not load the switching circuit.

3) Addition of **Decoupling Capacitor in parallel with the circuit:** The rise time and load capacitance could be controlled to reduce I_{peak}, but limited upto certain level. Hence, an optimum solution is to connect a high value capacitor across the critical cell to the control the lowering of supply voltage. As this capacitor decouples the critical cells from main power supply it is called as **decoupling capacitor**. Refer to figure below



Every time the critical cell (in above diagram, an inverter) switches, it draws current to charge the load capacitor C_L. This current consists of a both low and high frequency component. Most of the high frequency component, which is crucial part, is inserted by decoupling capacitor, whereas RL network replenish the charge (of C_D) i.e. when the cell switches, it charges upto C_L derived from C_D stored in the decoupling capacitor, the RL network will charge the decoupling capacitor back to C_D.

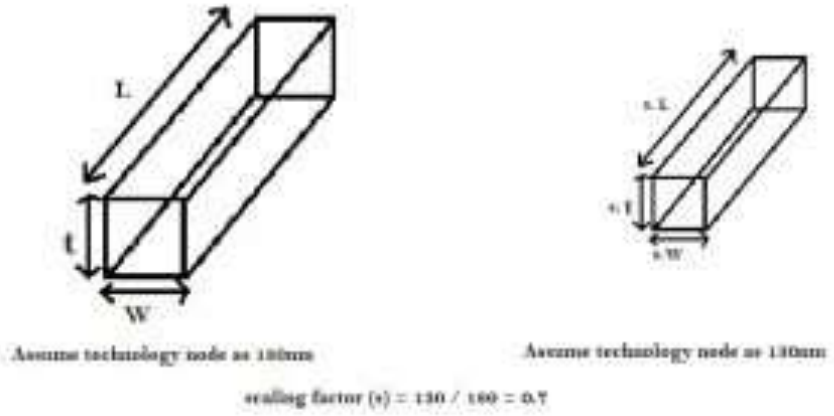
Interconnect Scaling trends

The dimensions of interconnects (wires) scales down, as VLSI technology scales down by a factor called as ‘scaling factor (s)’. The scaling factor is an integer by which the dimensions of interconnects shrinks down. Typically, the resistance, capacitance and inductance are the 3 main factors of wires that is affected by scaling. Following sections will explain how each factor gets affected by scaling, and concurrently, its impact on system.

Resistance:

Wires are usually classified as ‘short’ and ‘long’ wires. Lengths of short wires are scaled by a scaling factor ‘s’, whereas lengths of long wires are independent of scaling. ‘Short’ wires are typically used for local communication between gates that are placed with minimal distance, whereas ‘long’ wires are used for long range communication at the different corners of chip. Power supply rails can be an example of long wires.

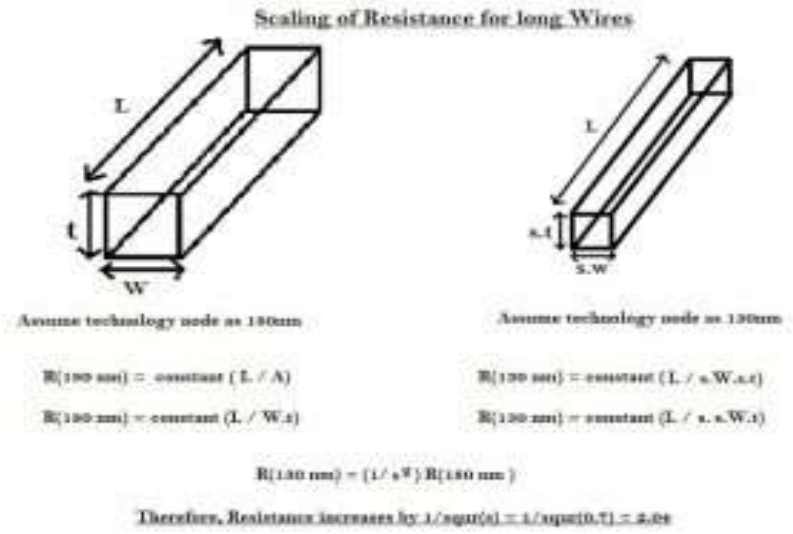
Consider the below diagram, which shows how resistance scales for ‘short’ and ‘long’ wires.



Assume, the gate length (node) is shrinking from 180nm to 130nm. The factor with which the gate length scales down is ~0.7 (i.e. 130nm / 180nm). Hence, s = 0.7 (approx.). Consider

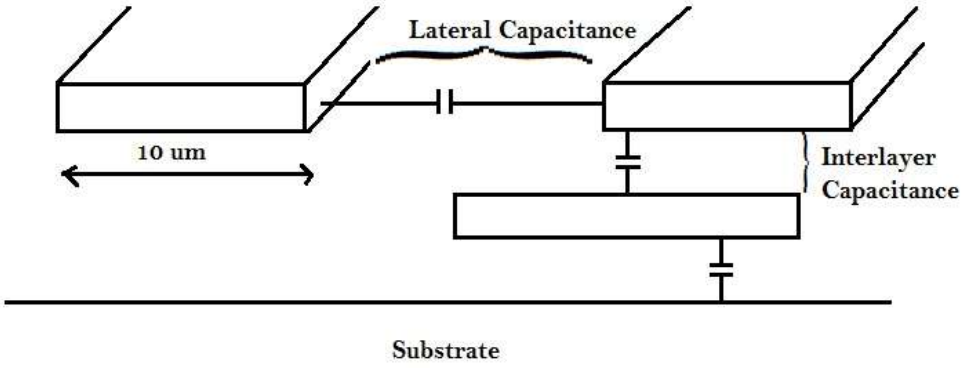
a short wire as shown above. For a short wire, all 3 factors (i.e. length, width and thickness) scales down by the same factor i.e. 0.7. Hence, as per above equations, due to scaling, the resistance of a short wire increases by 1.42.

On the other hand, for long wires, where the length of wire is independent of scaling, only thickness and width of the wire scales down. Hence, as per above equations, the resistance of long wire increases considerably by 2.04.



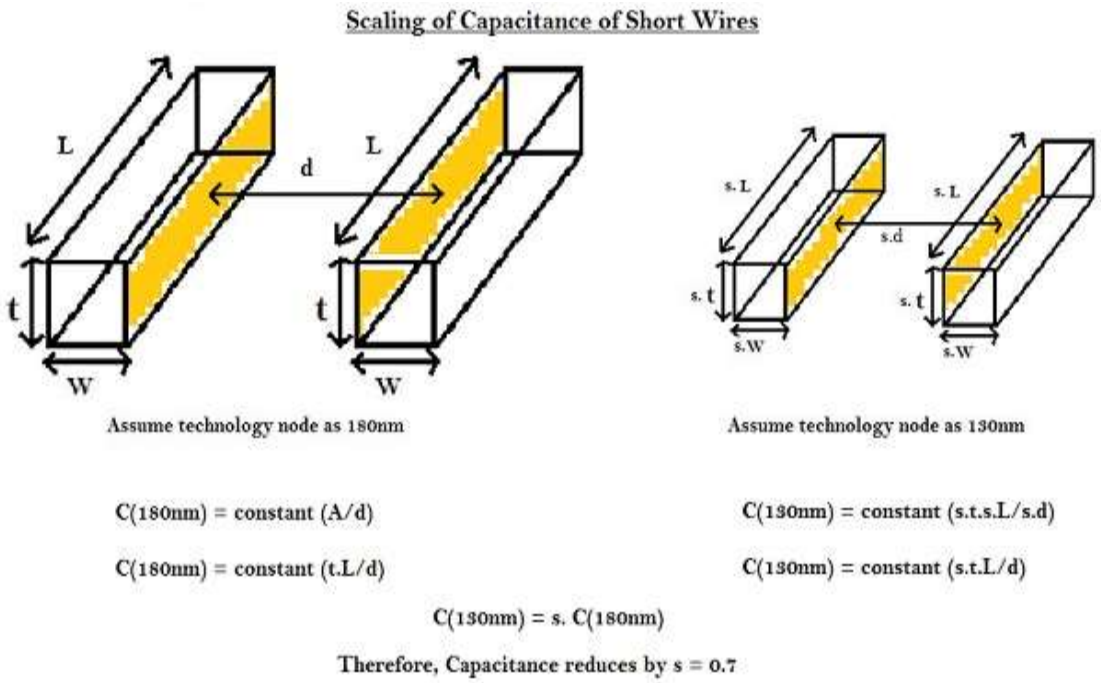
Capacitance:

In 5um technology, the width of the wire would be, say, 10 um and thickness, say, 1 um, as shown below:

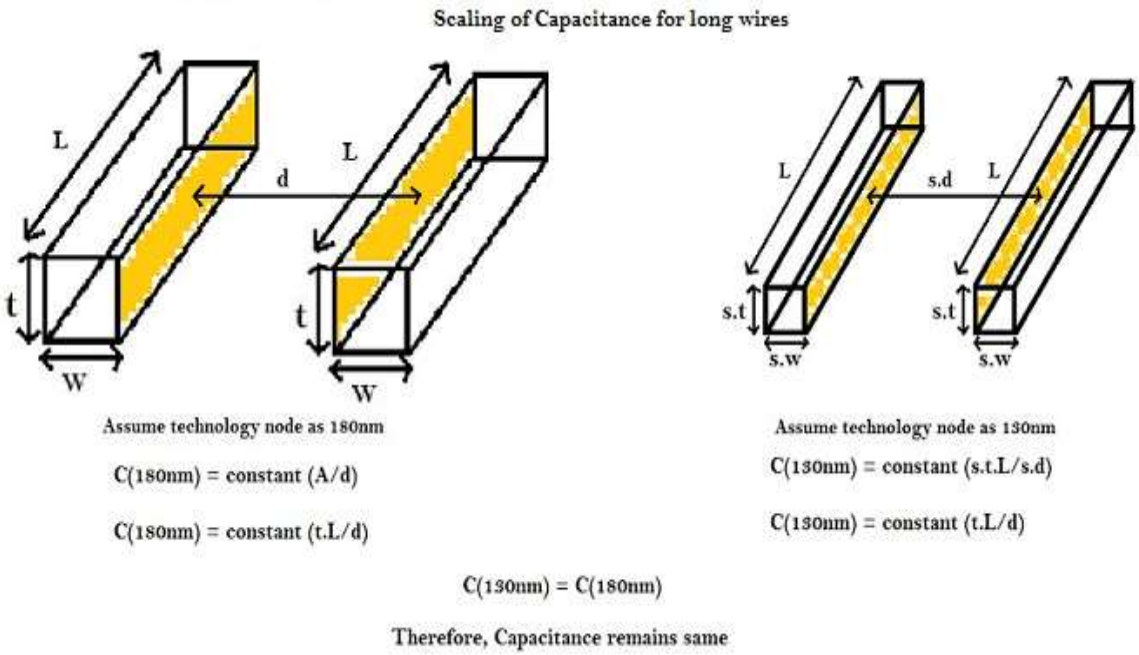


In the above 5um technology, the dominant capacitance, was the parallel plate capacitance between adjacent layers, i.e. inter layer capacitance. The lateral coupling capacitance's i.e. capacitance between two conductors on the same plane, exists, but ineffective.

Consider the below diagram, which shows how capacitance scales for 'short' wires.



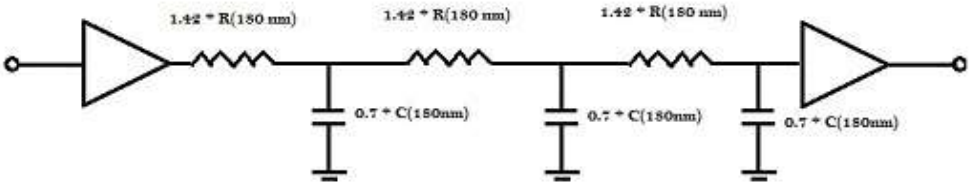
In case of short wires, all the parameters of wire scale down. Hence the capacitance goes down by the scaling factor i.e. 0.7, as per equations shown in above diagram.



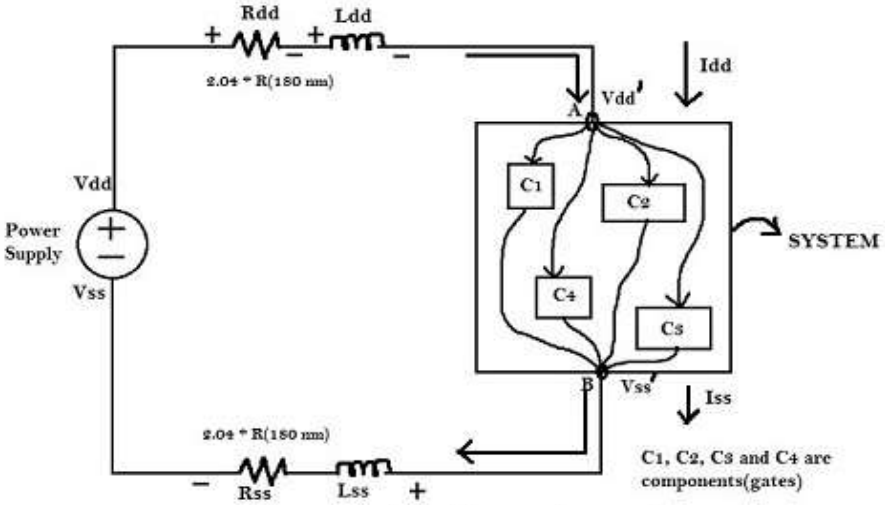
In case of long wires, all the parameters of wire scale down, except its length. Hence the capacitance remains same, as per equations shown in above diagram.

RC Delay:

Figure below shows the impact of scaling of short and long wires on RC delay.



Impact of Scaling on short wires



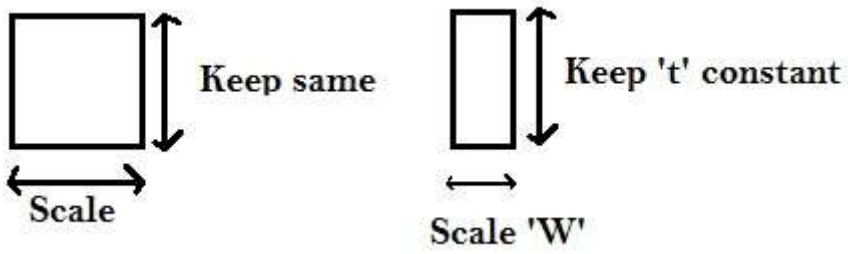
Impact of Scaling on long wires

For short wires, 'R' scales by (1/s), whereas 'C' scales by (s). Hence, **the RC delay for short wires is unaltered**, which means that the capability of short wire to transmit high frequency signals, is same.

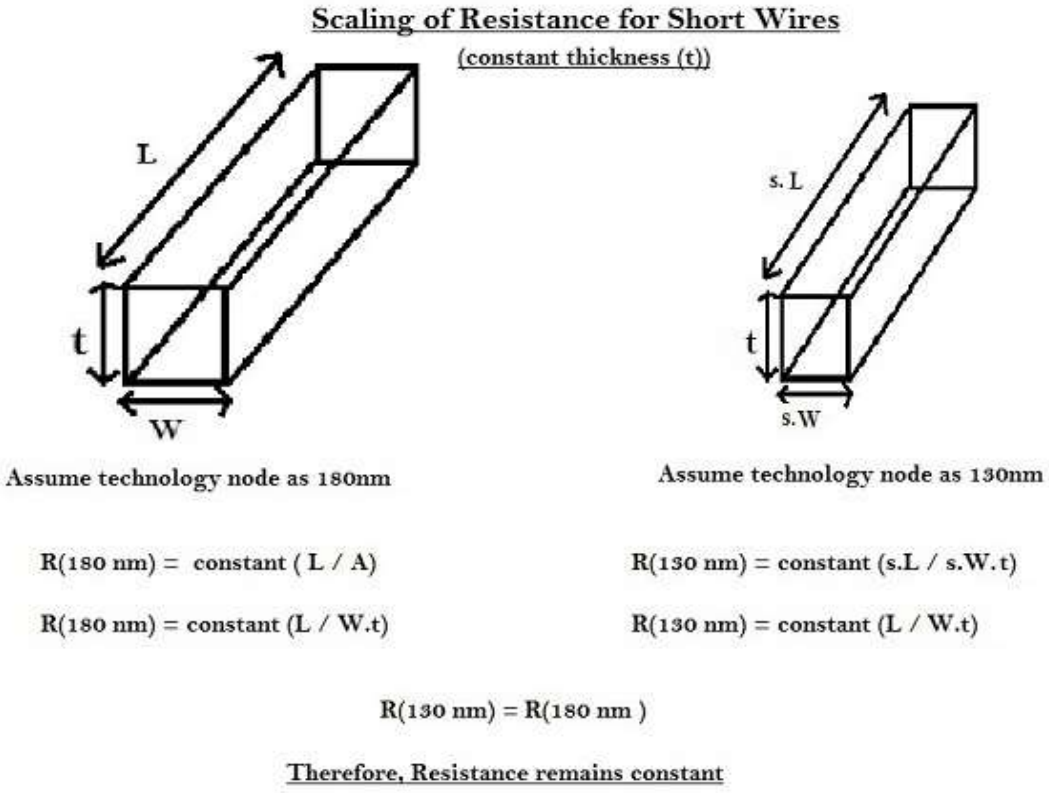
But, for long wires, 'R' scales up by ($1/s^2$), whereas the capacitance remains the same. Hence, **the RC delay for long wires increases by ($1/s^2$)**. If s = 0.7, then RC delay increases by 2.04. If long wires are used for power rails, it would affect the noise margin of the circuit. Also, if long wires are placed between two logic cells, the RC delay between the cells will increase significantly, which may lead to potential setup violation.

This trend of increased RC delay with scaling, combined scaling trends of 'R', which increases by ($1/s^2$), decreases the ability of long wires to transmit high frequencies.

An optimal solution is **not to** scale all dimensions of the wire as shown in diagram below.



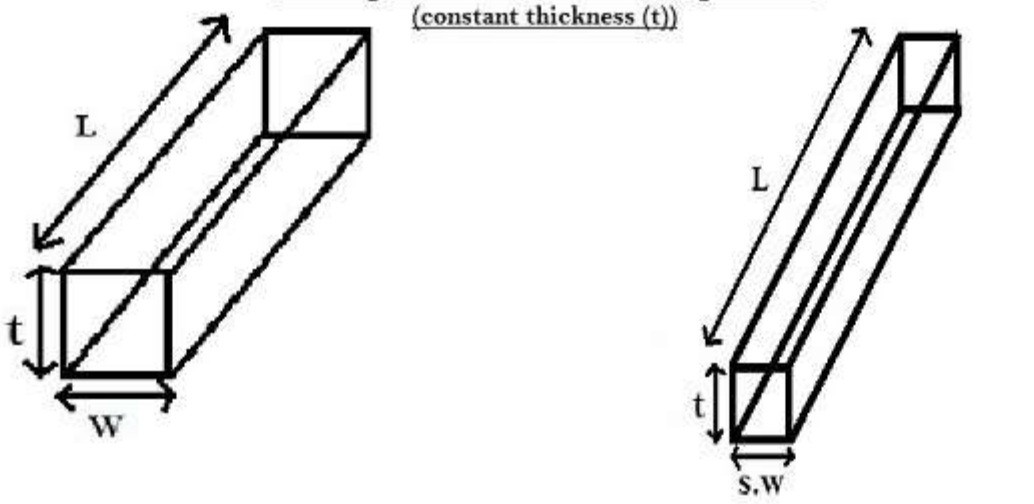
Considering the above trend of keeping ‘t’ constant and scaling only ‘W’, the resistance of short wires scales as shown in diagram below.



Thus, for short wires, with thickness kept constant, the resistance remains same. Similarly, for long wires, the scaling of resistance is as shown in below diagram.

Scaling of Resistance for long Wires

(constant thickness (t))



Assume technology node as 180nm

$R(180\ nm) = \text{constant}\ (L / A)$

$R(180\ nm) = \text{constant}\ (L / W.t)$

Assume technology node as 130nm

$R(130\ nm) = \text{constant}\ (L / s.W.t)$

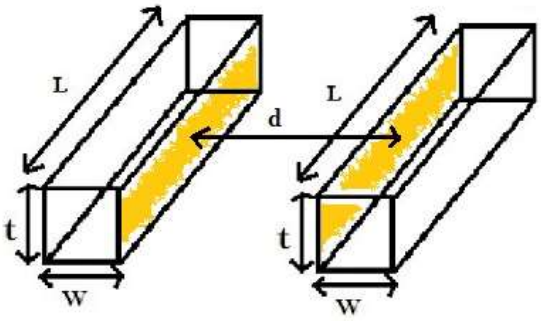
$R(130\ nm) = (1/s)\ R(180\ nm)$

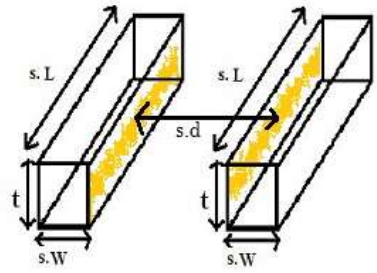
Therefore, Resistance increases by (1/s) i.e. (1/0.7) = 1.42

Thus, for long wires, with thickness constant, the resistance increases by (1/s) as per equations shown in above diagram.

Similarly, for capacitance, the effect of scaling wires with constant thickness is shown below

Scaling of Capacitance for short wires
(constant thickness)





Assume technology node as 180 nm

C(180nm) = constant (A/d)

C(180nm) = constant (t.L/d)

Assume technology node as 130 nm

C(130nm) = constant(t.s.L/s.d)

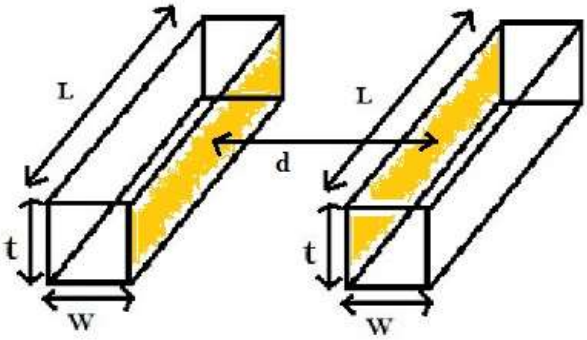
C(130nm) = constant(t.L/d)

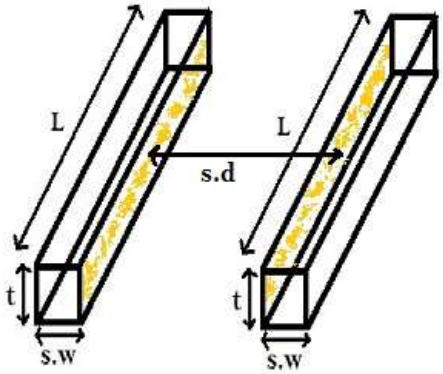
C(180nm) = C(130nm)

Therefore Capacitance remains same

Thus, for short wires, with thickness kept constant, capacitance remains same.

Scaling of Capacitance for long wires
(constant thickness)





Assume technology node as 180 nm

C(180nm) = constant (A/d)

C(180nm) = constant (t.L/d)

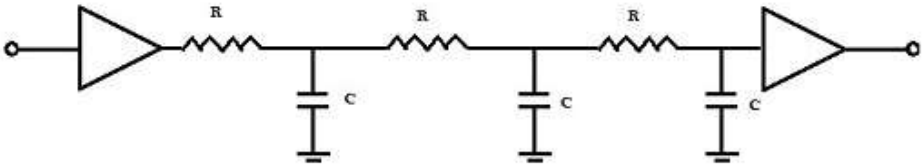
Assume technology node as 130 nm

C(130nm) = constant(t.L/s.d)

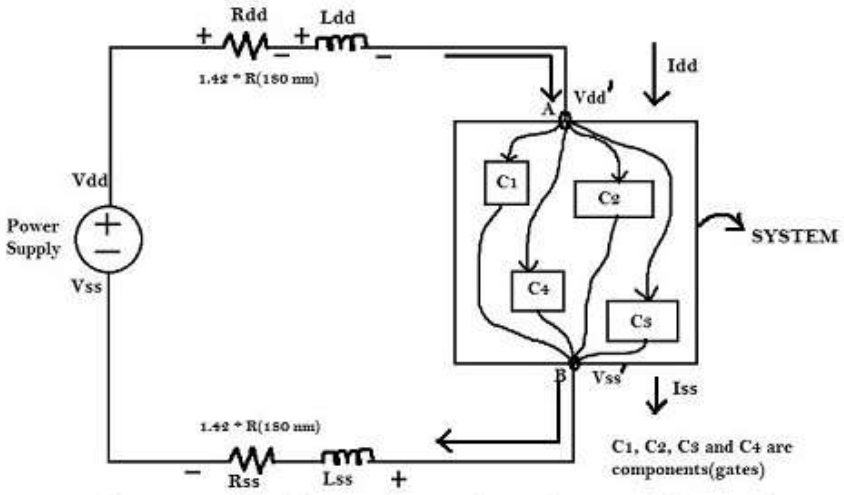
C(130nm) = (1/s) . C(180nm)

Therefore, Capacitance increases by (1/s) .i.e. (1/0.7) = 1.42

Thus, for long wires, with thickness kept constant, capacitance increases by 1.42, as per above equations



Impact of Scaling on short wires (constant thickness (t))



Impact of Scaling on long wires (constant thickness (t))

As shown in above figure, there is no impact on RC delay with thickness of wire kept constant.

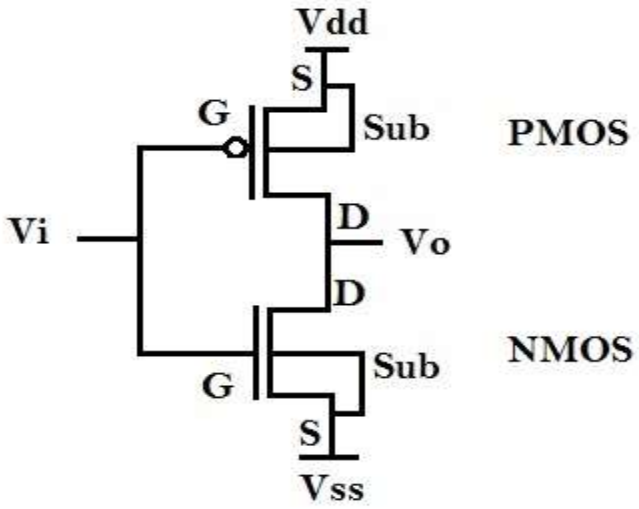
Scaling Summary

Full scaling	Short Wires	Long Wires
R	1/s	$1/s^2$
C	s	1
RC delay	1	$1/s^2$
Scaling with Thickness constant	**Short Wires**	**Long Wires**
R	1	1/s
C	1	1/s
RC delay	1	$1/s^2$

In certain cases, technically, the thickness can't be changed, and 'R' has trend of 'constant' for short wires, and (1/s) for long wires. To further reduce the resistance of wire, the only variable availed for scaling is 'width (W)' of wire. As the width of the wire increases, resistance decreases, but the routing density of wires in the specific layer increases. To reduce the routing density of a specific layer, due to wide wires, the number of layers, to make contacts between signals, must be increased. Earlier (in 1985), 2 metal layers were sufficient to make all signal connections on a silicon chip. With current scenario, 7-8 metal layers are used for same. Proportionately, 10% of metal layers are used only for power and ground supply.

Switching activity of CMOS

A CMOS, is basically an inverter logic (NOT gate), that consists of a PMOS at the top, and NMOS at the bottom (as shown in figure below), whose ‘gate’ and ‘drain’ terminal is tied together. The ‘gate’ terminals of both the MOS transistors is the input side of an inverter, whereas, the ‘drain’ terminals form the output side. The ‘source’ terminal of PMOS is connected to ‘Vdd’, whereas source of NMOS is connected to ‘Vss’, as shown below.

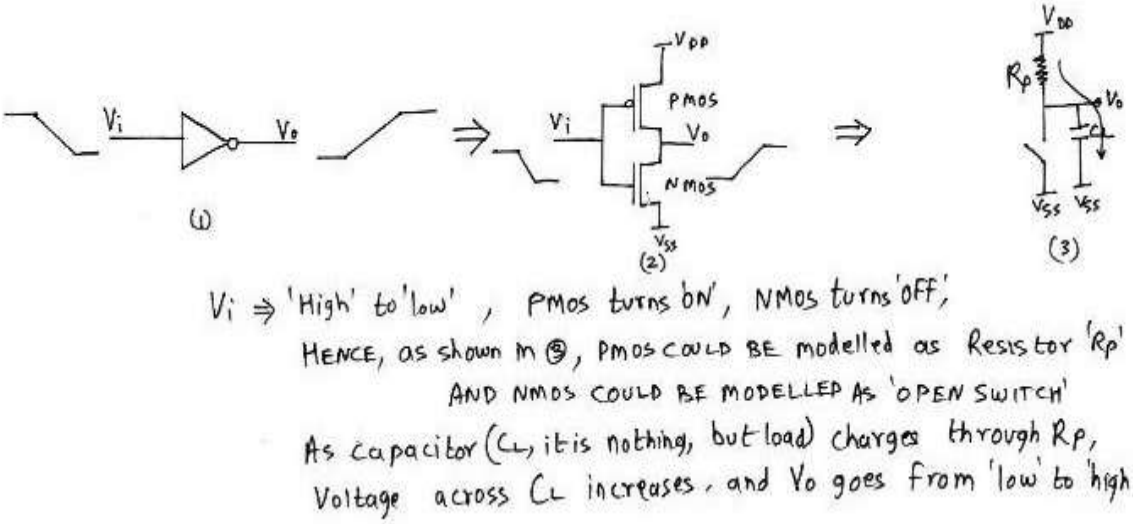


G = Gate Terminal
S = Source Terminal
D = Drain Terminal
Sub = Substrate Terminal

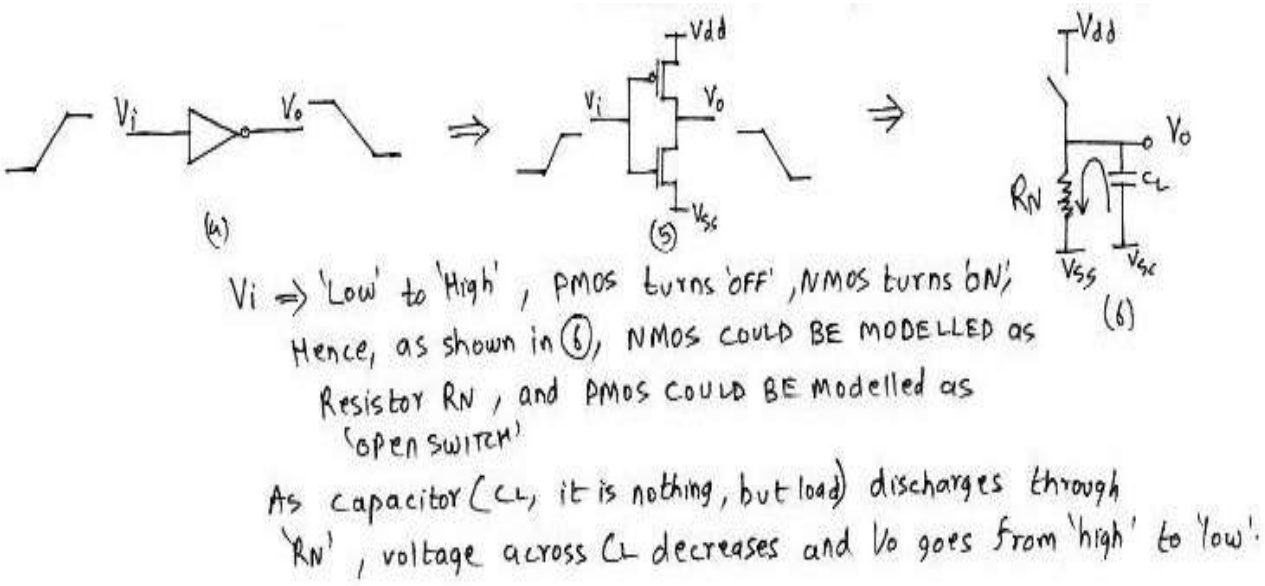
Schematic diagram of CMOS Inverter

There’s a fourth terminal for a MOS transistor commonly referred to as ‘Substrate’ terminal. It is connected to ‘Vdd’ for PMOS and to ‘Vss’ for NMOS. This is one of the terminals of MOS transistor, which could to be used to vary the threshold voltage of transistor by applying certain voltage to it.

Now, let us look at the transient response of an inverter. Consider the following figure. When 'Vi' switches from high to low, PMOS turns 'ON' whereas NMOS turns 'OFF'. During this operation of CMOS inverter, NMOS is modeled as an 'open switch', whereas PMOS is modeled as a resistance 'Rp' followed by a capacitor C_L. In a large circuit, every CMOS is superseded and/or preceded by logic gates, which is again, nothing but a bunch of NMOS and PMOS transistors. These transistors contribute to lot of capacitance, which contribute to C_L, load capacitance. Also, the wires connected to Vi and Vo of the inverter contribute to the load capacitance C_L. Hence, during the above mode of operation, C_L charges to Vdd, through Rp, and 'Vo' switches from logic '0' to logic '1'.



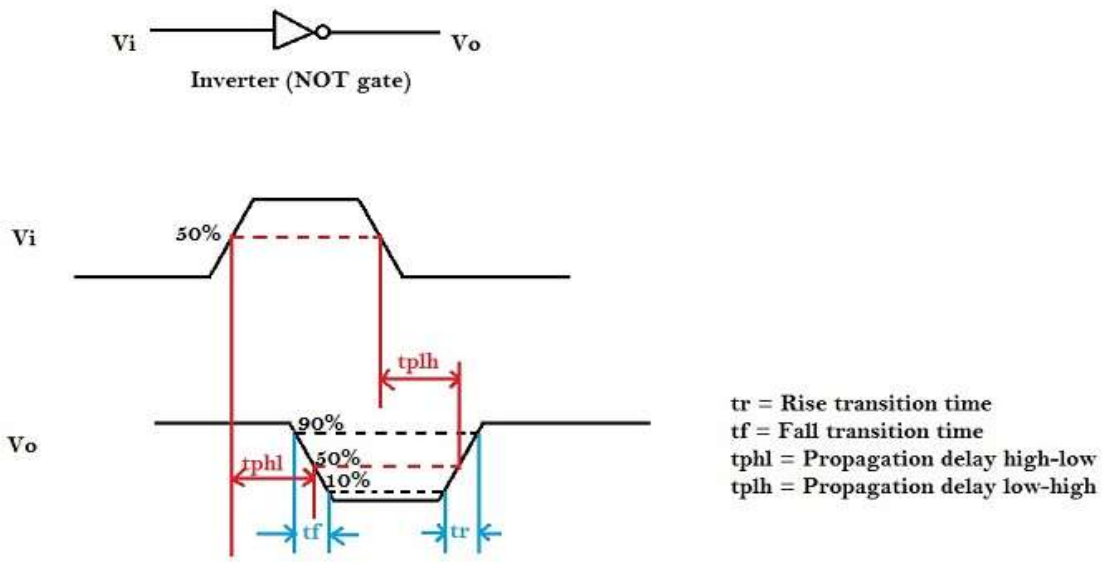
Now, when 'Vi' switches from low to high, PMOS turns 'OFF', whereas NMOS turns 'ON'. This behavior could be modeled as an 'open switch' for PMOS and resistance 'Rn' for NMOS followed by a capacitor C_L. Hence, during this mode of operation, C_L discharges to Vss, through Rn, and 'Vo' switches from logic '1' to logic '0', as shown in figure below.



We understand that when 'Vi' switches from low to high (or high to low), 'Vo' switches from high to low (or low to high). But, 'Vo' doesn't changes instantaneously, but after a finite amount of delay after application of 'Vi'. The time required for change in 'Vo' after application of 'Vi', is called as propagation delay of the inverter. This would be clearer in the next section.

Propagation Delay of CMOS inverter

The propagation delay of a logic gate e.g. inverter is the difference in time (calculated at 50% of input-output transition), when output switches, after application of input.



In the above figure, there are 4 timing parameters. Rise time (t_r) is the time, during transition, when output switches from 10% to 90% of the maximum value. Fall time (t_f) is the time, during transition, when output switches from 90% to 10% of the maximum value. Many designs could also prefer 30% to 70% for rise time and 70% to 30% for fall time. It could vary up to different designs.

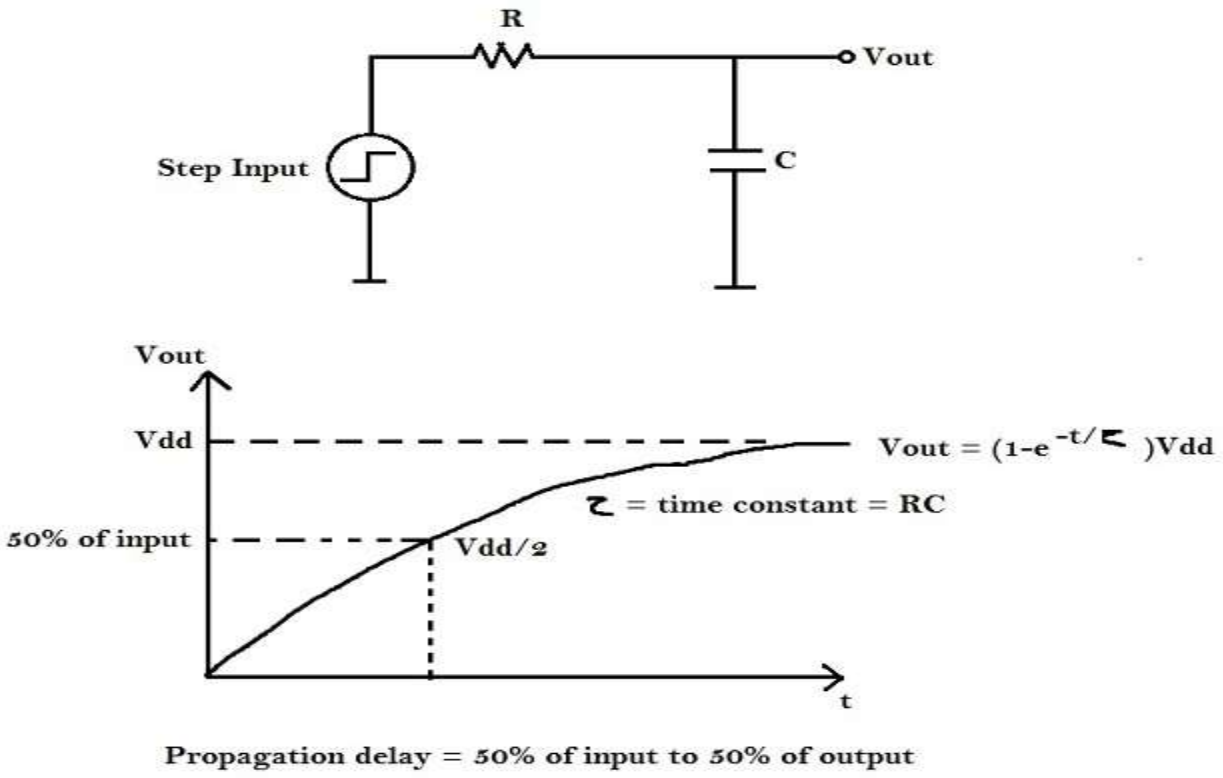
The propagation delay high to low (t_{pHL}) is the delay when output switches from high-to-low, after input switches from low-to-high. The delay is usually calculated at 50% point of input-output switching, as shown in above figure.

Now, to find the propagation delay, we need a model that matches the delay of inverter. As we have seen above, the switching behavior of CMOS inverter could be modeled as a

resistance R_{on} with a capacitor C_L, a simple first order analysis of RC network will help us to model the propagation delay.

First order RC network

Consider the following RC network to which we apply a step input.



Our aim is to find 't' at Vdd / 2.

Vout = $(1-e^{-t/\tau})$ Vdd, where $\tau = RC$ = time constant.

Substituting 'Vout' equal to Vdd/2, and 't' equal to 'tp' in above equation, we get the following:

Vdd/2 = $(1-e^{-tp/\tau})$ Vdd

Therefore, $t_p = \ln(2)\,\tau = 0.69\tau$

Hence, $t_p = 0.69RC$

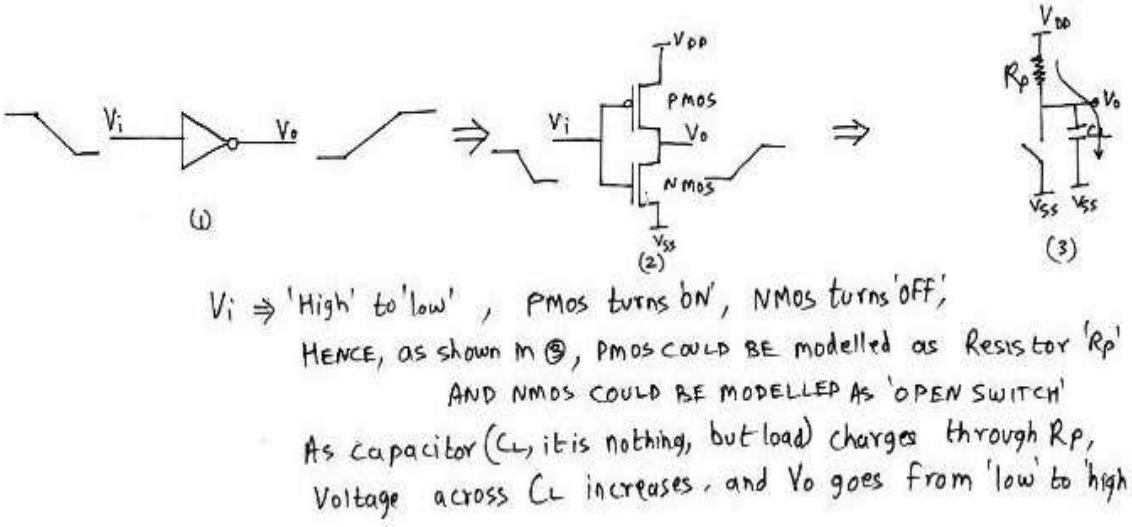
Hence, a CMOS inverter can be modeled as an RC network, where

R = Average ‘ON’ resistance of transistor

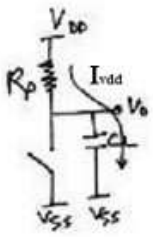
C = Output Capacitance

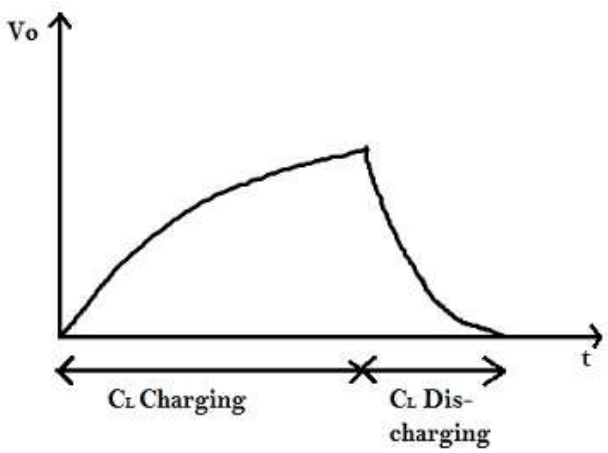
Power Consumption

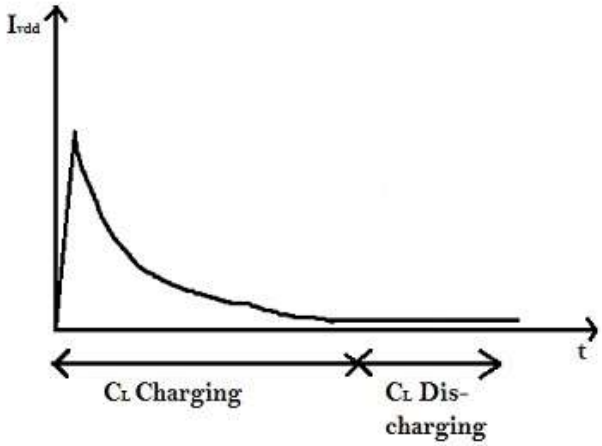
Consider the following scenario where output capacitor C_L charge from 0 to ‘Vdd’ as input switches from ‘Vdd’ to ‘0’:



The charging and discharging current and voltage waveforms could be shown as below:





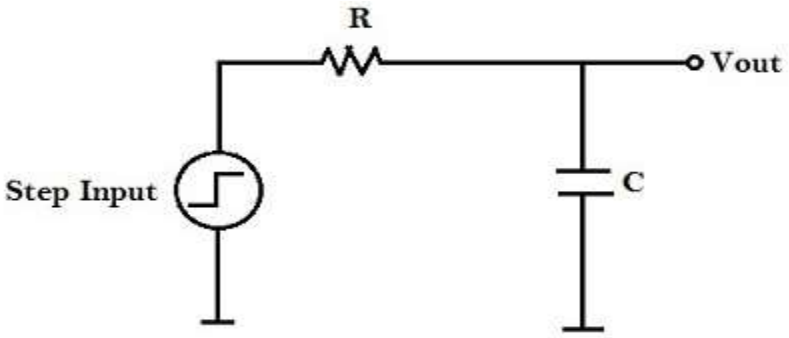


Hence, whenever output load C_L gets charged from '0' to 'Vdd', a finite amount of energy is drawn from the power supply. part of which is dissipated in Rp, whereas the remaining is stored in C_L. The reverse happens whenever C_L gets discharged from 'Vdd' to '0'i.e. the energy is dissipated in Rn.

The power consumed at any instance by a CMOS circuit is given by:

$p(t) = v(t) * i(t) = V_{supply} * i(t)$

Now, to calculate the amount of energy during a single switching, refer to the following RC equivalent network for CMOS switching from 'low' to 'high'



Energy = Power * Delay

Let $E_{0->1}$ denote energy consumed whenever output switches from '0' to 'Vdd'.

Therefore, $E_{0->1} = \int p(t).\, dt = Vdd \int idd\,(t).\, dt = Vdd \int C_L * Vdd.\, dVout = C_L * Vdd^2$

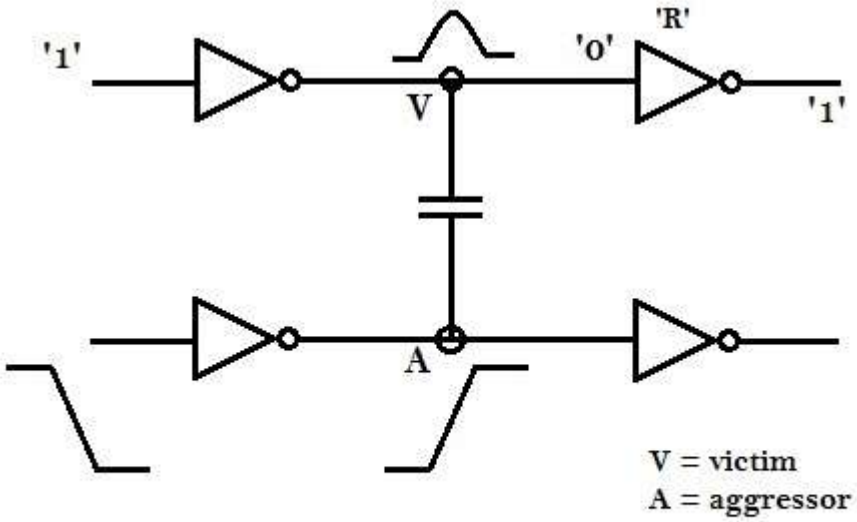
(use the current-voltage relationship to understand the above integration i.e. C = Q / V = I*t /V)
Hence, every time a circuit switches, it consumes $C_L * Vdd^2$ amount of energy. Thus, amount of energy consumed, if a circuit (e.g. clock) switches 'n' times, will be **$n * C_L * Vdd^2$,** which can gain attention.

Effect of Coupling Capacitance

In deep sub-micron technology (i.e. <130nm) and below, the lateral capacitance between nets/wires on silicon, becomes much more dominant than the interlayer capacitance. Hence, there is a capacitive coupling between the nets, that can lead to logic failures and degradation of timing in VLSI circuits. Crosstalk is a phenomenon, by which a logic transmitted in vlsi circuit or a net/wire creates undesired effect on the neighboring circuit or nets/wires, due to capacitive coupling.

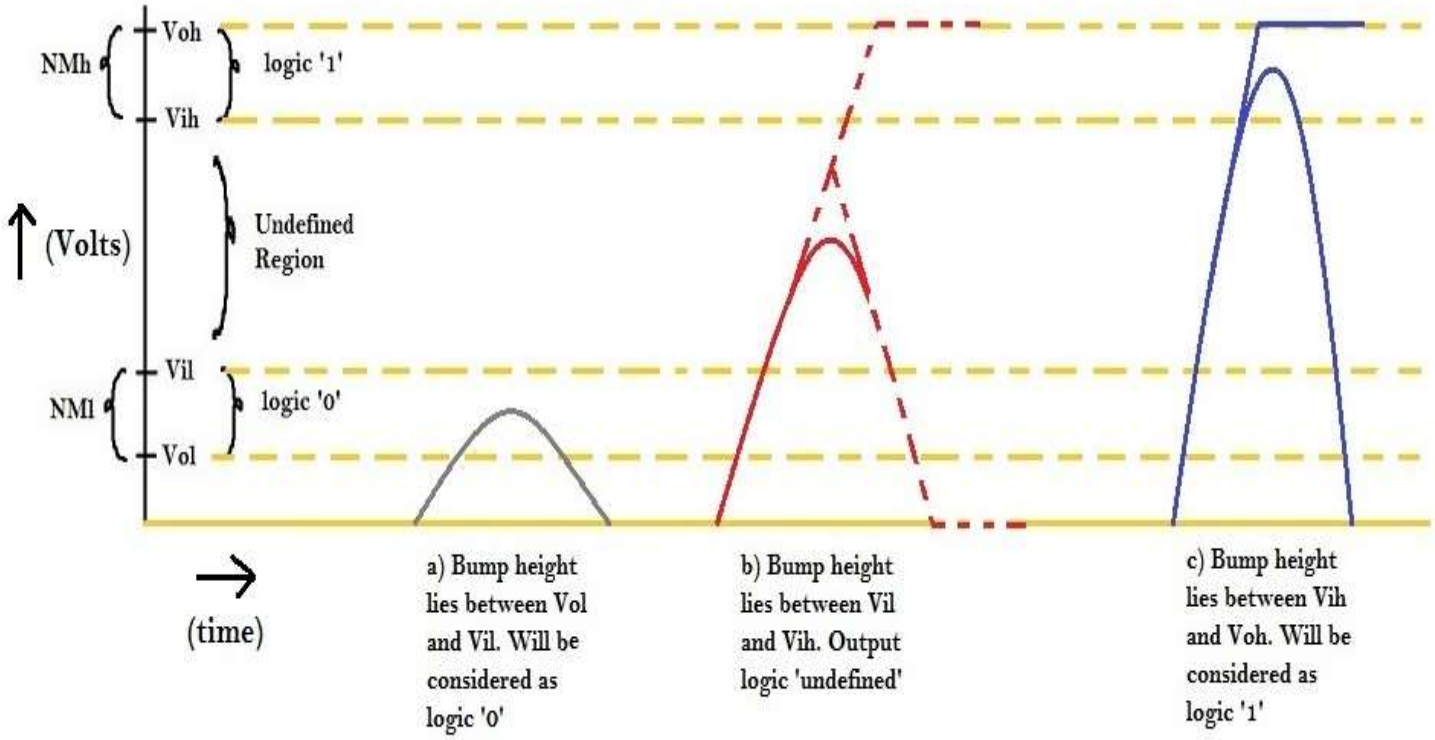
Refer to the diagram below to get a clear picture on the effect of coupling capacitance on functionality and timing of VLSI circuits.

Crosstalk Noise due to Coupling Capacitance



Crosstalk noise leading to logic failures

The disturbance at ‘A’ can potentially cause a disturbance at ‘V’, because of the mutual coupling capacitance, and if the disturbance at ‘V’ crosses noise threshold of the receiving gate ‘R’, then it may change the logic at the output of ‘R’ i.e., output of ‘R’, which is supposed to be at logic ‘1’, might switch to logic ‘0’, as it senses a logic ‘1’ at its input, due to the noise induced on its input by the disturbance at ‘A’. Refer to diagram below to understand noise-induced bump characteristics at different noise margin levels.



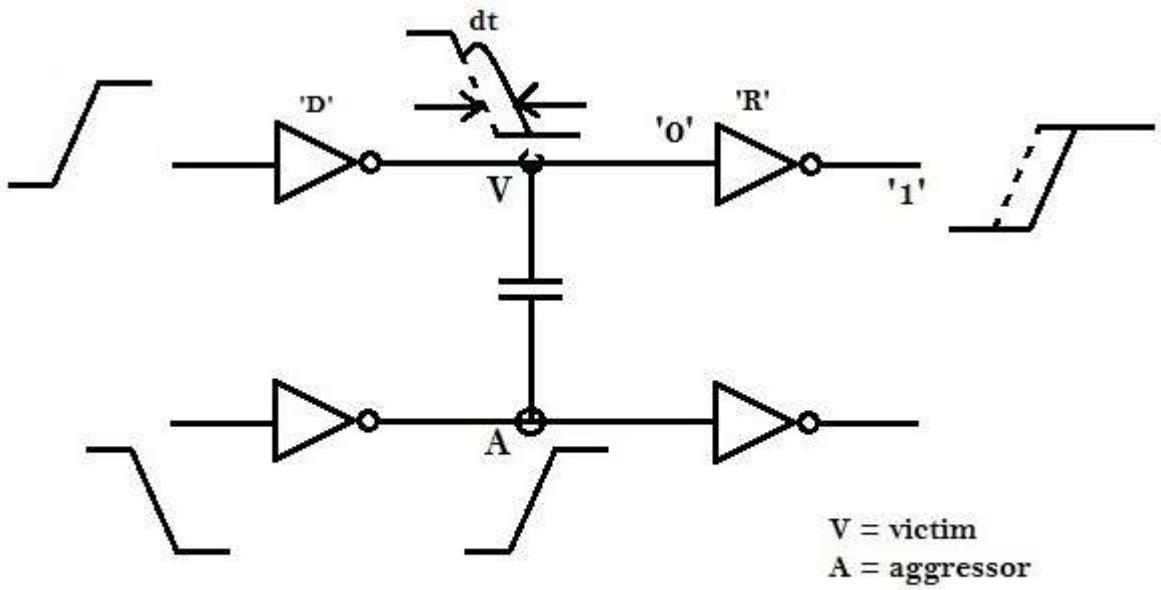
Noise induced bump characteristics at different noise margin levels

If the bump height at victim 'V' lies between NMl (Noise Margin low), then the logic at victim 'V' will remain at logic '0'.

If the bump height at victim 'V' lies between V_{il} and V_{ih}, then the logic at victim 'V' is undefined, i.e. it might switch to logic '1' or logic '0'.

If the bump height at victim 'V' lies between NMh (Noise Margin high), then the logic at victim 'V' will switch to logic '1', leading to logic failures.

Timing Degradation due to Coupling Capacitance



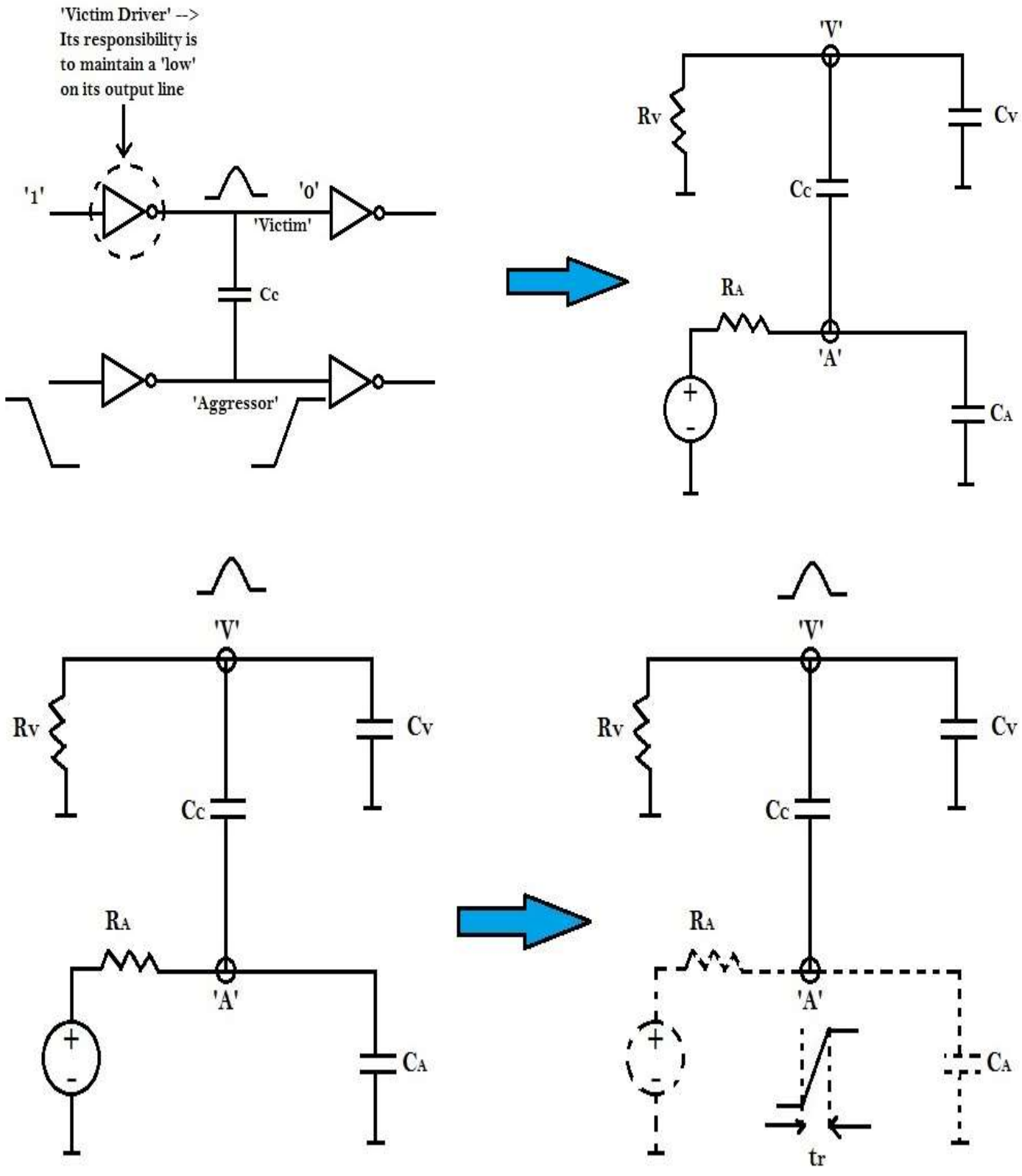
Timing Degradation leading to delay uncertainty

Consider input of driver ‘D’ switching from logic ‘0’ to logic ‘1’,thus the logic at node ‘V’ switches from ‘1’ to ‘0’. Now, if both ‘A’ and ‘V’ nodes have signal switching event at the same time interval, then, due to noise induced by signal transition at aggressor ‘A’, a change in the timing instant of the signal transition occurs at ‘V’, as shown in above figure. Due to this, the propagation delay of the driver ‘D’ increases by ‘dt’ amount of time, thus increasing the overall propagation delay of the circuit, which might lead to potential setup violation.

Crosstalk Modelling and Analysis

Refer diagram below to understand the basic model of crosstalk

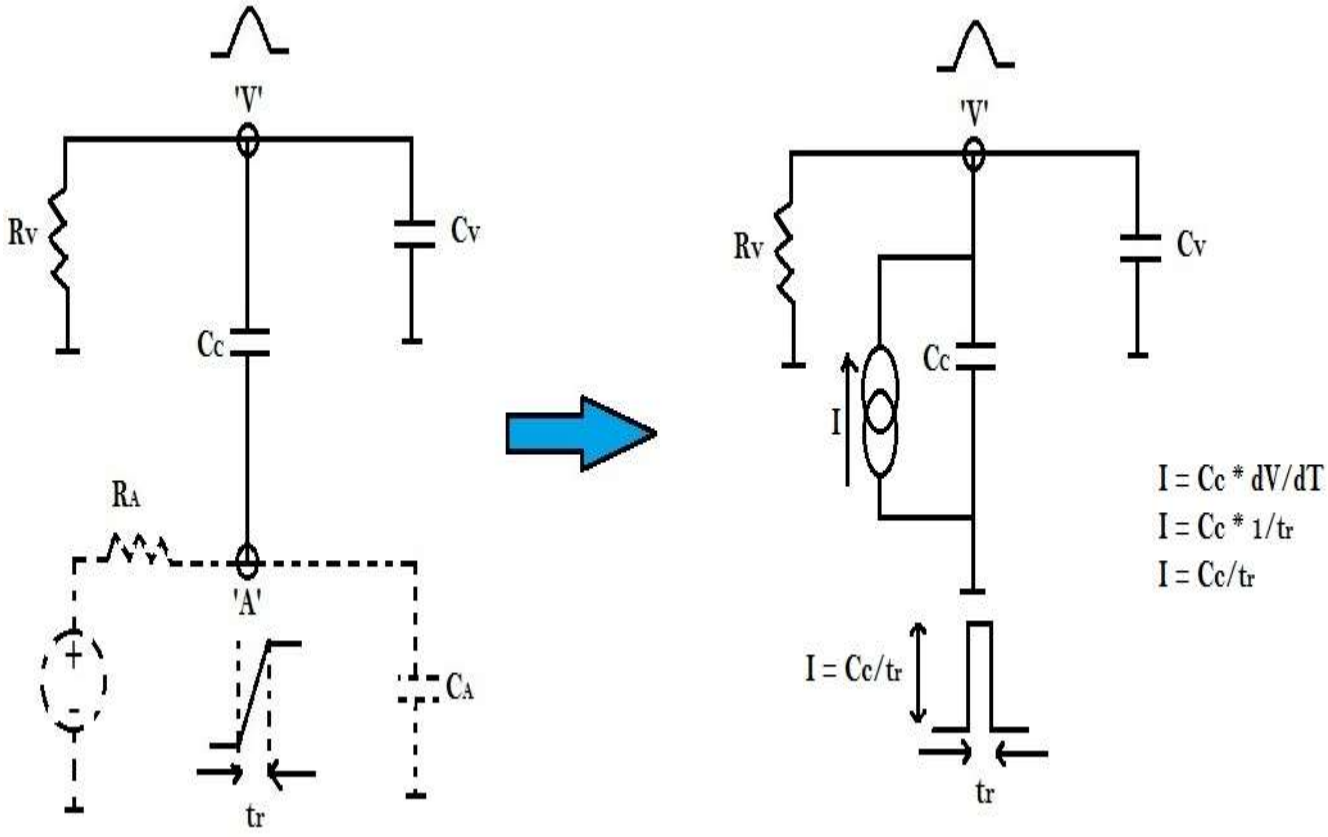
‘Victim’ and ‘aggressors’ drivers can be modeled by resistors ‘R_V’ and ‘R_A’ respectively. Whereas ‘victim’ and ‘aggressors’ loads can be modeled by capacitors ‘C_V’ and ‘C_A’ respectively. Let the coupling capacitance between them be C_C. The above model can be further simplified as shown in figure below.


'Victim Driver' -->
Its responsibility is
to maintain a 'low'
on its output line
'1'
'0'
'Victim'
Cc
'Aggressor'
'V'
Rv
Cv
Cc
RA
'A'
CA
'V'
Rv
Cv
Cc
RA
'A'
CA
'V'
Rv
Cv
Cc
RA
'A'
CA
tr

In the above figure, t_r is the rise time at the aggressor node ‘A’, which is related to the gate delay R_A as shown in below equation:

$t_r = R_A * (CC + C_A)$

Essentially, the above figure represents a voltage source connected at aggressor node ‘A’ with a series capacitance ‘C_C ‘. By Thevenin to Norton conversion, this voltage source can be replaced by a current source with parallel capacitance ‘C_C’ as shown below:

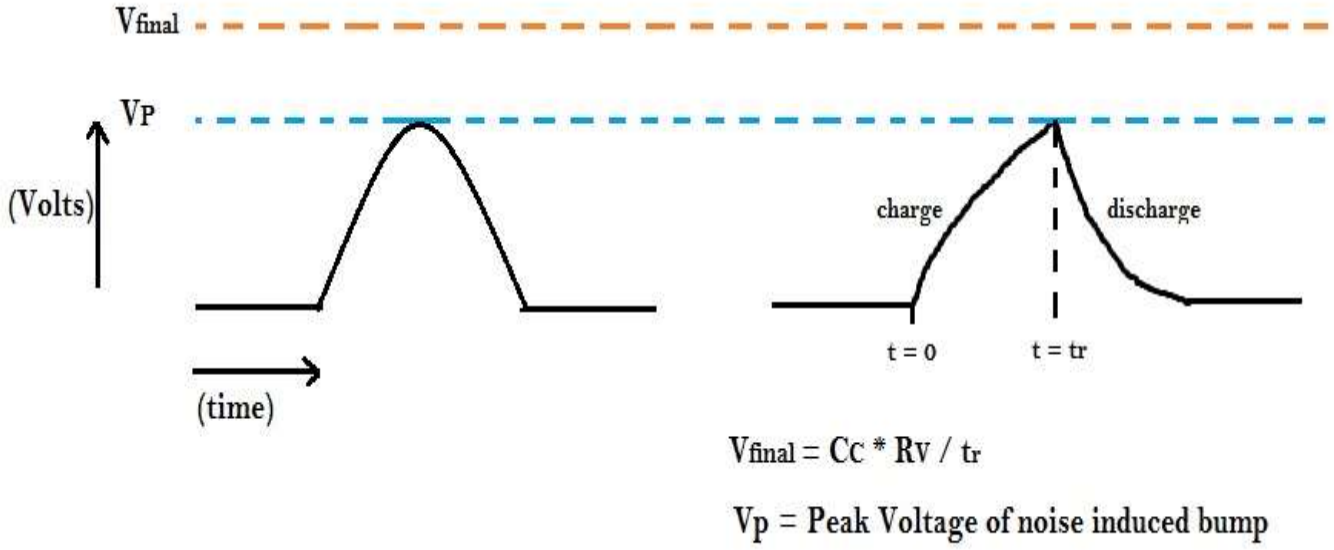


We need to find the voltage equation at victim ‘V’, considering the final value of voltage as V_{final} shown in equation below:

$V_{final} = I * R$

$V_{final} = (C_C / t_r) * R_V$

The noise induced bump is nothing but charging-discharging waveform across capacitor as shown below:



The charging voltage across capacitor can be deduced from the following equation:

$V_{charge} = V_{final} (1 – e^{-t/RC})$

At $t = t_r$

$V_{charge} = V_p$

$R = R_V$, $C = C_C + C_V$

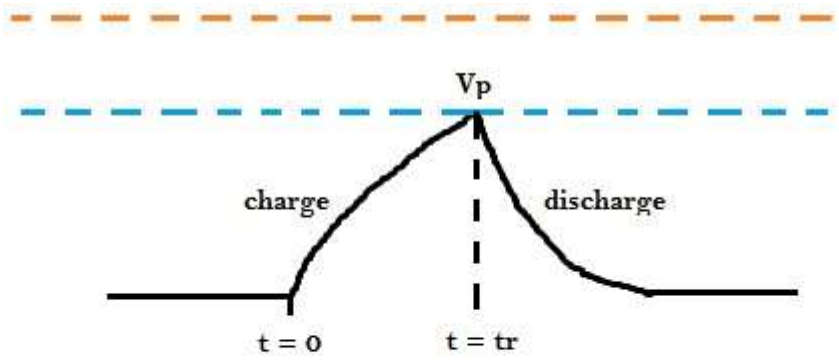
$R_V * (C_C + C_V)$ = Equivalent Time Constant

$V_p = (C_C / t_r) * R_V (1 – e^{-t_r / (R_V * (C_C + C_V))})$

The figure below shows how peak voltage is a function of coupling capacitance C_C, Victim drive strength R_V and rise time on aggressor line.

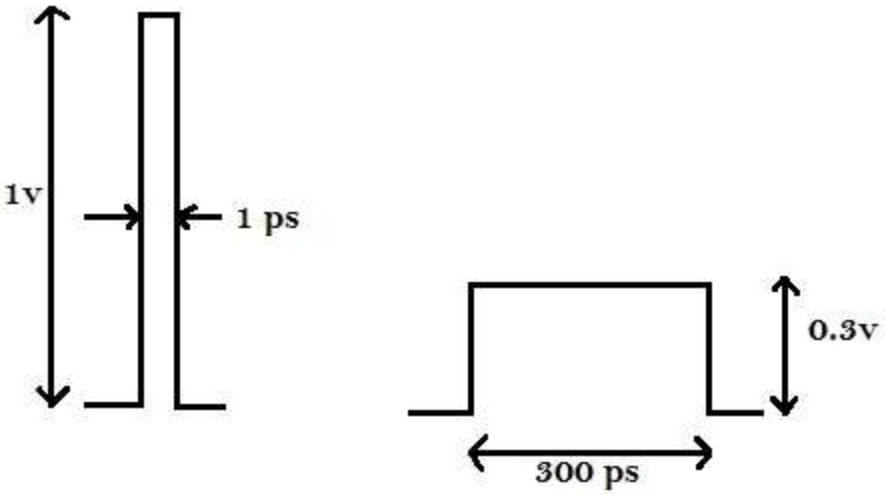
The higher V_p is, there are more chances that it would exceed noise margin. But, that is not the only thing.

Consider a case, where the pulse height V_p is high (1V), with small pulse width (e.g. 1ps) as opposed to another scenario, where the pulse height is low (e.g. 0.3V) and pulse width is large (e.g. 100ps). This can be illustrated in the diagram below.



$$V_{final} = C_C * R_V / t_r$$

$$V_p = C_C * R_V / t_r \left(1 - e^{\frac{-t_r}{R_V (C_C + C_V)}}\right)$$



If the receiving gate's RC delay is not in sync with the incoming pulse, it may not even recognize the incoming pulse (1V, 1ps). Therefore, even if the peak of the pulse is substantial, but pulse is narrower, it's possible that the receiving gate doesn't identify the existence of that pulse and it gets filtered out.

Therefore, we have 2 things to control

1) Peak Voltage (V_p)

$$V_p = \frac{C_C * R_V}{t_r} \left(1 - e^{\frac{-t_r}{R_V (C_C + C_V)}}\right)$$

$$\text{Let} \quad x = \frac{t_r}{R_V (C_C + C_V)}$$

Therefore,

$$V_p = \frac{C_C * R_V}{t_r} \left(1 - e^{-x}\right)$$

If x is very small i.e. $R_v (C_C + C_V)$ is large compared to t_r, then $e^{-x} \sim (1 - X)$

Therefore, V_p can be deduced as shown below:

$$V_p = \frac{C_C * R_V}{t_r} \left(1 - e^{-x}\right)$$

$e^{-x} = 1 - x$, for very very small x

$$V_p = \frac{C_C * R_V}{t_r} \left(1 - [1-x]\right)$$

$$= \frac{C_C * R_V}{t_r} [x]$$

$$= \frac{C_C * \cancel{R_V}}{\cancel{t_r}} \frac{\cancel{t_r}}{\cancel{R_V} * (C_C + C_V)}$$

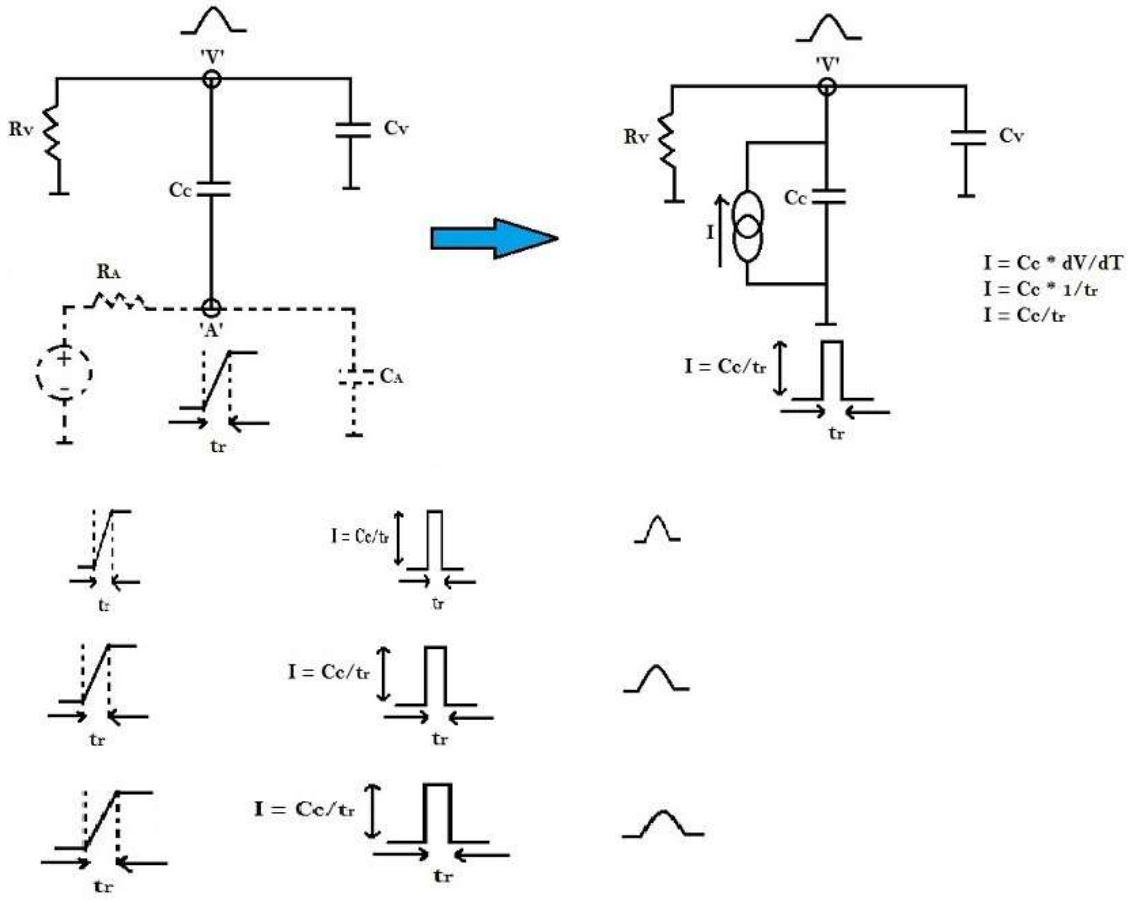
$$V_p = \frac{C_C}{(C_C + C_V)}$$ **Basic Coupling Formula**

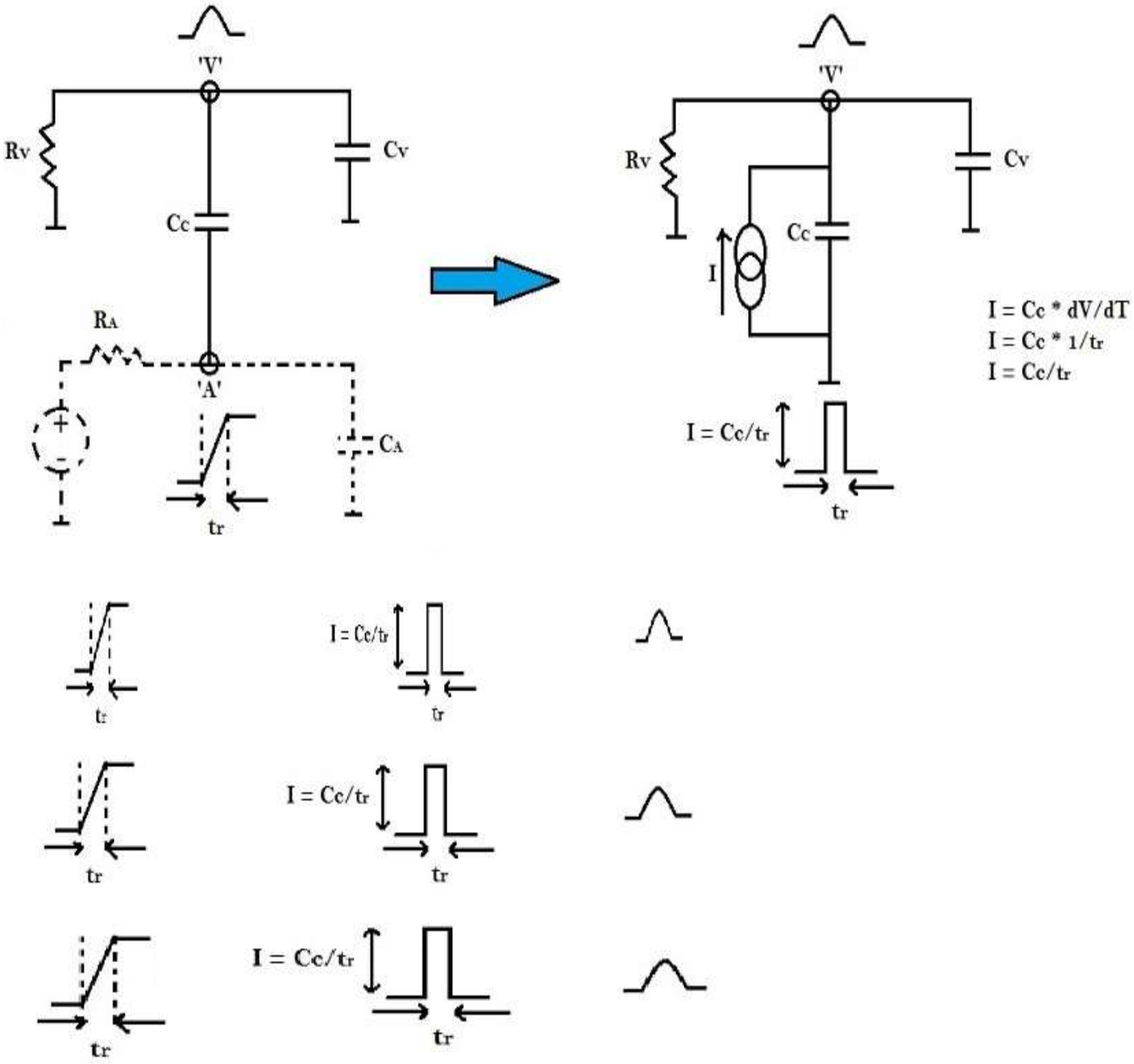
Hence, the first solution to reduce crosstalk noise, is to increase the Resistance of Victim driver (R_V).i.e. downsize the victim driver, so that, the high resistance of the victim driver restricts the supply of current and charging of victim net capacitance during the rise time (t_r) of aggressor signal, which would in turn reduce the bump height.

The second solution to reduce crosstalk noise, is to increase the Capacitance of Victim load (C_V).i.e. upsize the victim load, thus the resistance will reduce, which will in turn help the victim net to maintain a strong static voltage.

2) Pulse Width

Pulse width, depends upon the aggressor net transition. The steep the transition is, on aggressor, the shorter will be the pulse width. This can be illustrated as shown in below diagram.





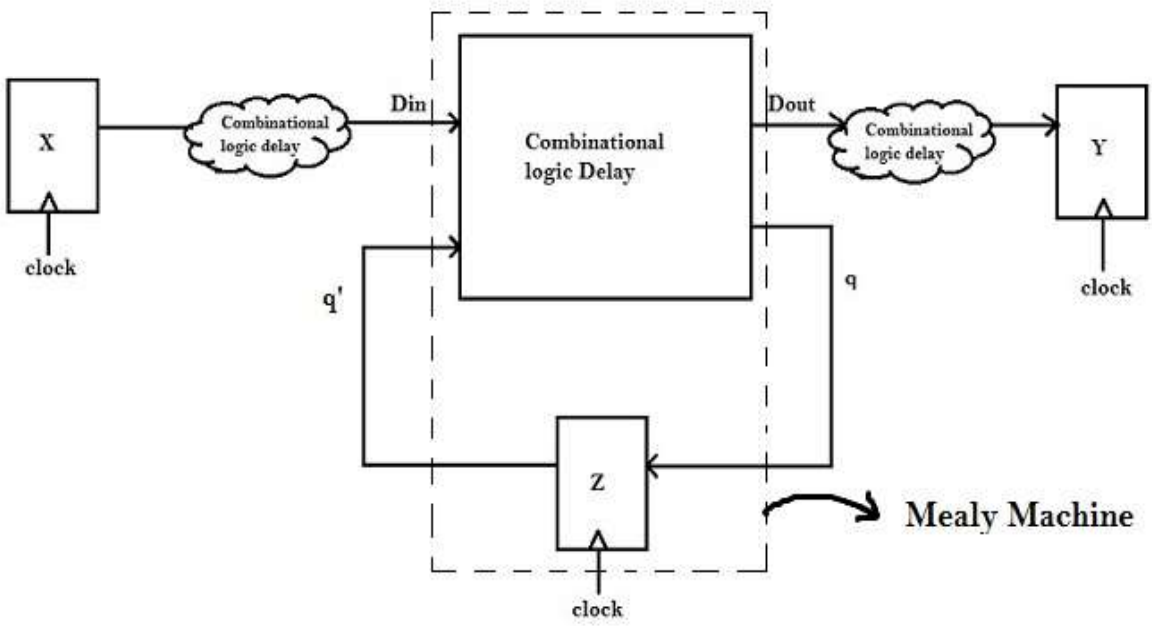
Hence, the third solution to reduce crosstalk noise, is to maintain sharp transitions on aggressor.

Another method to reduce crosstalk noise is to introduce shields in between victim and aggressor.

Static Timing Analysis (STA)

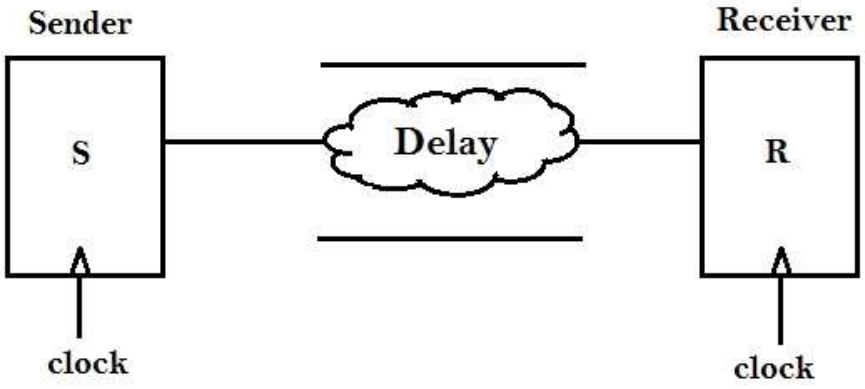
Static Timing Analysis (STA) is one of the techniques to verify design in terms of timing. This kind of analysis doesn't depend on any data or logic inputs, applied at the input pins. The input to an STA tool is the routed netlist, clock definitions (or clock frequency) and external environment definitions. The STA will validate whether the design could operate at the rated clock frequency, without any timing violations. Some of the basic timing violations are **setup violation** and **hold violation**

Consider the following Mealy Machine diagram to understand setup and hold timing checks



Above figure shows a basic description of a system in form of a Mealy Machine. Consider a flip-flop 'X' which generates data 'Din and it arrives as inputs to Mealy Machine after some delay q'(current state). Mealy Machine generates an output 'Dout', at **q** (next state). The receiver 'Y' receives 'Dout' after some delay.

The common thing between the flip-flops X, Y and Z is the **'clock'**. Same 'clock' is used to control all transfer of data between flip-flops. On the same clock edge, there would be some change at the output of X, there would be a change in the output of 'Z' and the change would propagate to **q** and 'Dout' after some delay. 'Dout', after some delay would propagate all the way to 'Y'. For correct operation, this change in 'Dout' should be captured at next clock edge by 'Y'. Hence, correctness of the system can be judged by tracing each Sender – Receiver pair. In the above example, Sender is 'X' whereas Receivers are 'Y' and 'Z'.



Above figure shows a Sender-Receiver pair. Assume, identical clock goes to both and there's some delay between Sender and Receiver. This delay will not be fixed as the effective load capacitance seen by each gate in the design is different. Other factors that affect the delay of a gate such as input transition, threshold voltage, drive strength, etc.

Let us study the following example of a NAND gate to understand how the delay varies within a gate.

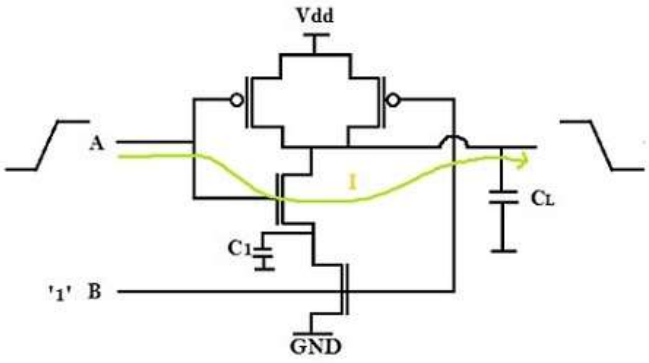


Fig. 1

C1 already discharged, since NMOS is 'ON' as 'B' is logic '1'.
Hence, Tpd = Req*CL

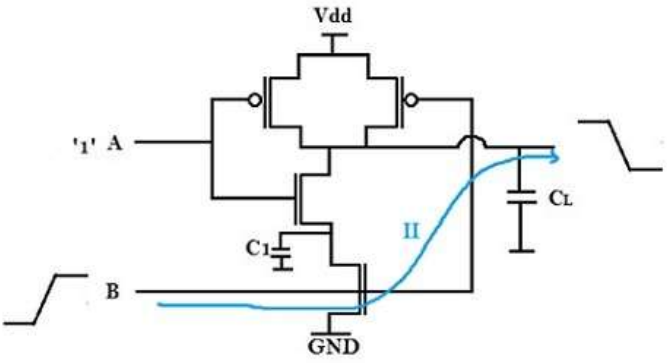


Fig. 2

NMOS is initially 'OFF', but turns 'ON' when 'B' switches from '0' to '1'. Total capacitance that needs to be discharged is C1 + CL.
Hence, Tpd = Req *(C1 + CL)

Above figure shows a condition under which same NAND gate has different delays

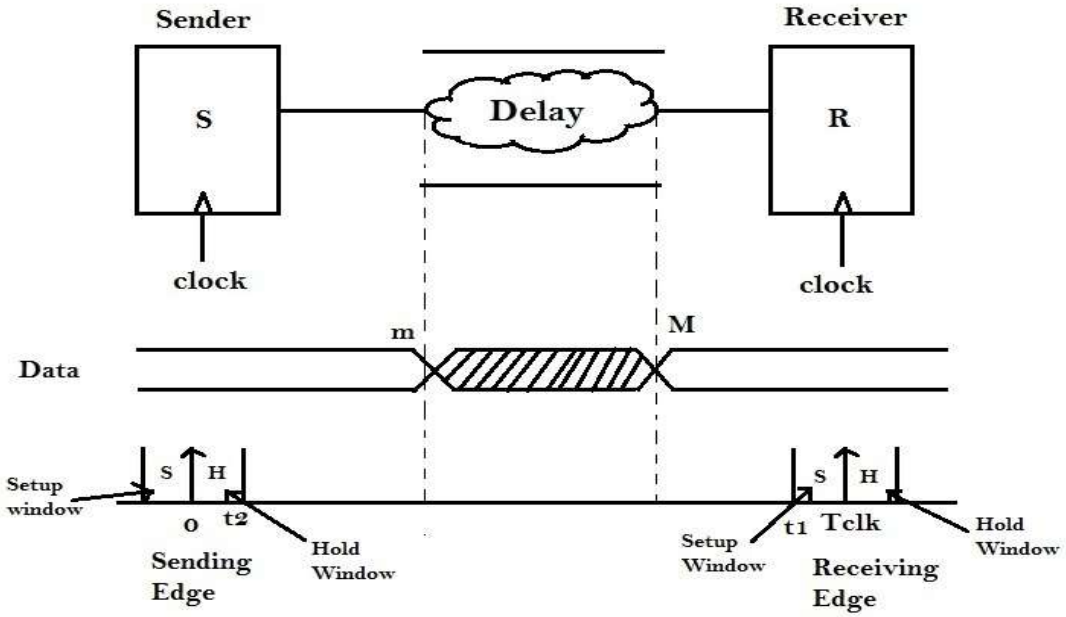
Assume that output was at logic ‘1’, which means either ‘A’ or ‘B’ was at logic ‘0’ and other at logic ‘1’.

Consider ‘B’ was at logic ‘1’, and ‘A’ at logic ‘0’. Output will switch from logic ‘1’ to logic ‘0’, when ‘A’ switches from logic ‘0’ to logic ‘1’ (Fig. 1). Let us consider this as **Delay I**. Consider ‘A’ was at logic ‘1’, and ‘B’ at logic ‘0’. Output will switch from logic ‘1’ to logic ‘0’, when ‘B’ switches from logic ‘0’ to logic ‘1’ (Fig. 2). Let us consider this as **Delay II**. Now the question is, “Is Delay I and Delay II same or different? If different, which delay path is faster (I or II)? Why?”

In Fig. 1, as ‘B’ is already at logic ‘1’, the NMOS, to which ‘B’ is connected is already ‘ON’ and hence the capacitor C_1 is already discharged. So, when ‘A’ switches from logic ‘0’ to logic ‘1’, only capacitor C_L must be discharged through the pull-down NMOS network. Whereas, in second case (Fig. 2), the NMOS to which ‘B’ is connected is ‘OFF’, since ‘B’ was at logic ‘0’. So, now, when ‘B’ switches from logic ‘0’ to logic ‘1’, capacitors C_1 and C_L, both must be discharged through the pull-down NMOS network, which takes more time compared to Delay I.

Hence **Delay I** and **Delay II** are different, where **Delay I** is less than **Delay II**.

The above example illustrates for 2-input NAND gate, it has two timing paths. More complex logic gates will have even more timing paths, e.g. 10 input MUX. Thus, the delay between Sender and Receiver is not constant, but will have range of values, as shown below.



Therefore, the delay ranges from m ≤ delay ≤ M.

Where m = minimum propagation delay of combinational logic

and M = Maximum propagation delay of combinational logic

There is a requirement from the receiver (basically, a flip-flop), that, if it must sample data at T_{clk}, the data should be available and stable before or at finite time (say 't1'), before clock edge T_{clk} arrives. Therefore, (T_{clk} – t1) is the setup time or setup margin or setup window (S) for the data to arrive to the receiver.

On the other hand, there's a requirement from Sender (again, a flip-flop), that if data must change after sending clk edge, that change should happen atleast after a finite time (say 't2'), after clock edge arrives. If this change in data happens before 't2', it might corrupt the data, which the flip-flop has already sampled. This finite time 't2' is called as Hold time or Hold margin or Hold window (H).

The finite time periods 't1' and 't2' are the internal delays of a flip-flop

The data is not expected to change between hold time 'H' to 'm' and 'M' to (T_{clk} – Setup time 'S'). Data changes somewhere between 'm' and 'M', and becomes stable after that.

Therefore, we have got certain defining equations for setup and hold time, and they are as follows :

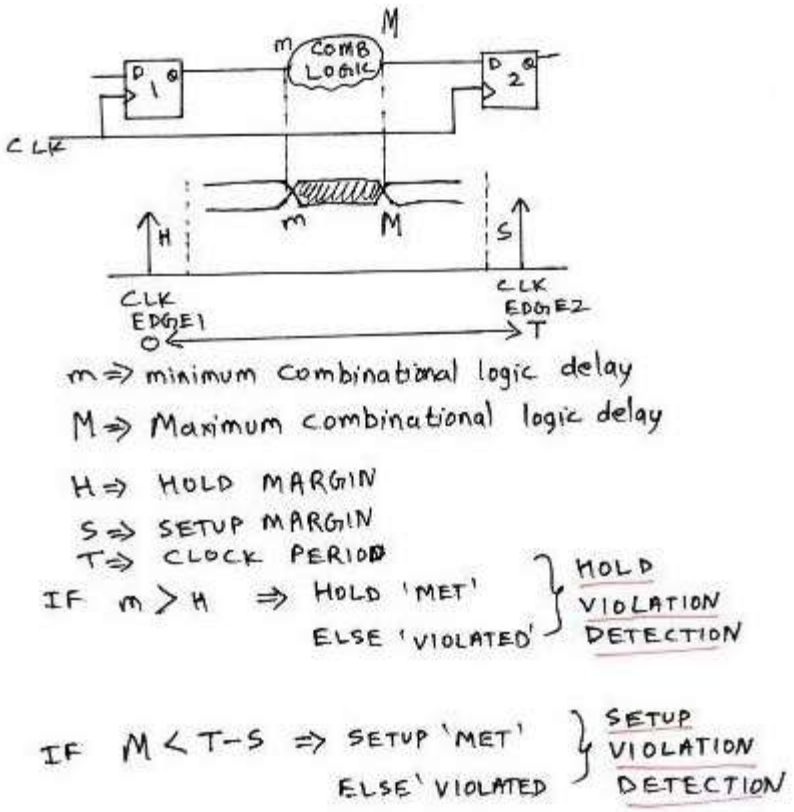
m > H i.e. Minimum propagation delay of the combinational logic should be greater than Hold Margin

If **m < H,** it results into timing violation, called as **Hold** violation. This means, that the combinational logic delay is very **less** and hence data change is very **fast**. To satisfy the **'hold'** requirement, the combinational logic delay should be **increased**.

M < T_{clk} – S i.e. Maximum propagation delay of the combinational logic should be less than Clock period (T_{clk}) minus the Setup Margin

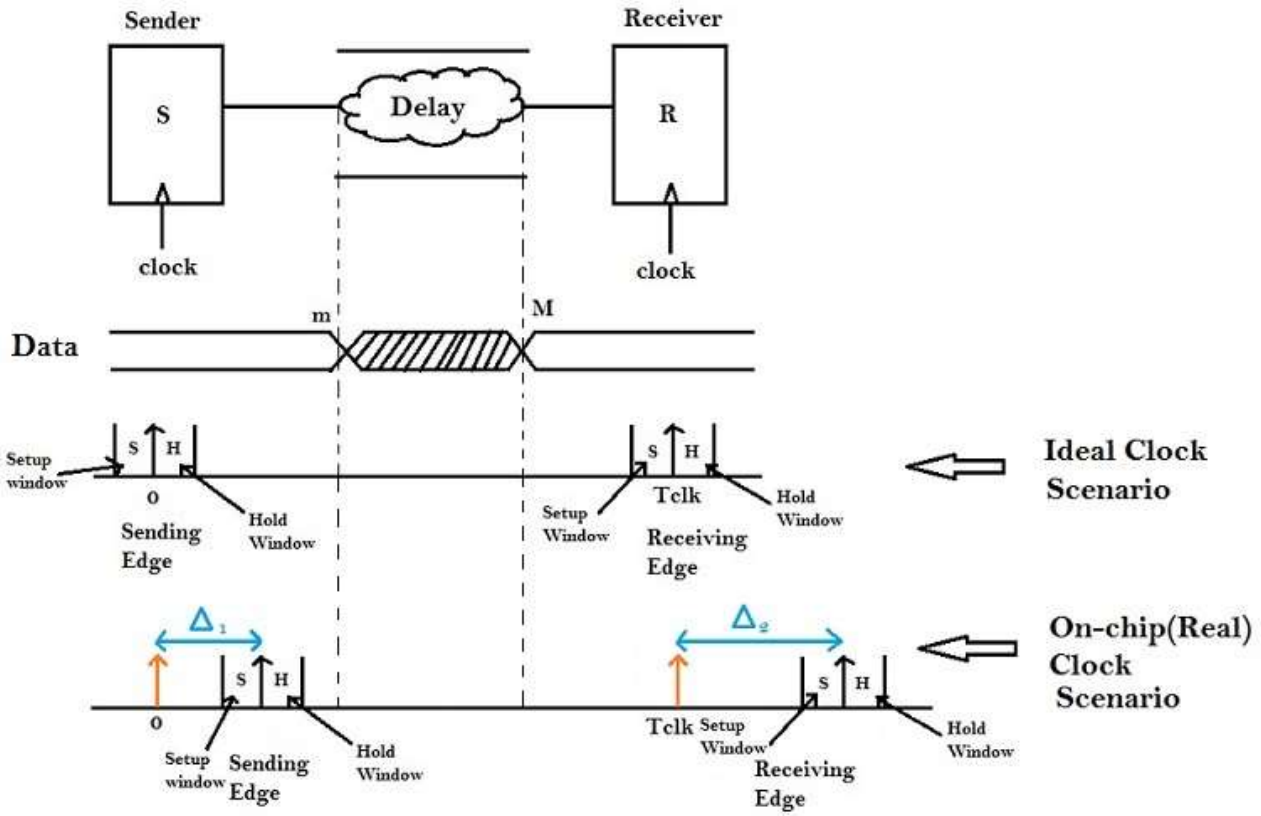
If **M > T_{clk} – S,** it results into timing violation, called as **Setup** violation. This means, that the combinational logic delay is very **large** and hence data change is very **slow**. To satisfy the **'setup'** requirement, the combinational logic delay should be **decreased**.

The process of fixing timing violation, and implement the fixes back to the PNR netlist, is referred to as Engineering Change order (ECO) .



Above figure summarizes the concept of setup and hold time for an ideal clock (i.e. both sender and receiver are clocked at the same time).

But, on a chip, this is usually not the case. Due to wire RC's, the clock might not reach the sender or receiver at the same time. It is possible, that clock for sender reaches at Δ_1 and clock for receiver reaches at Δ_2. This clock network delay could have captured in the following diagram.



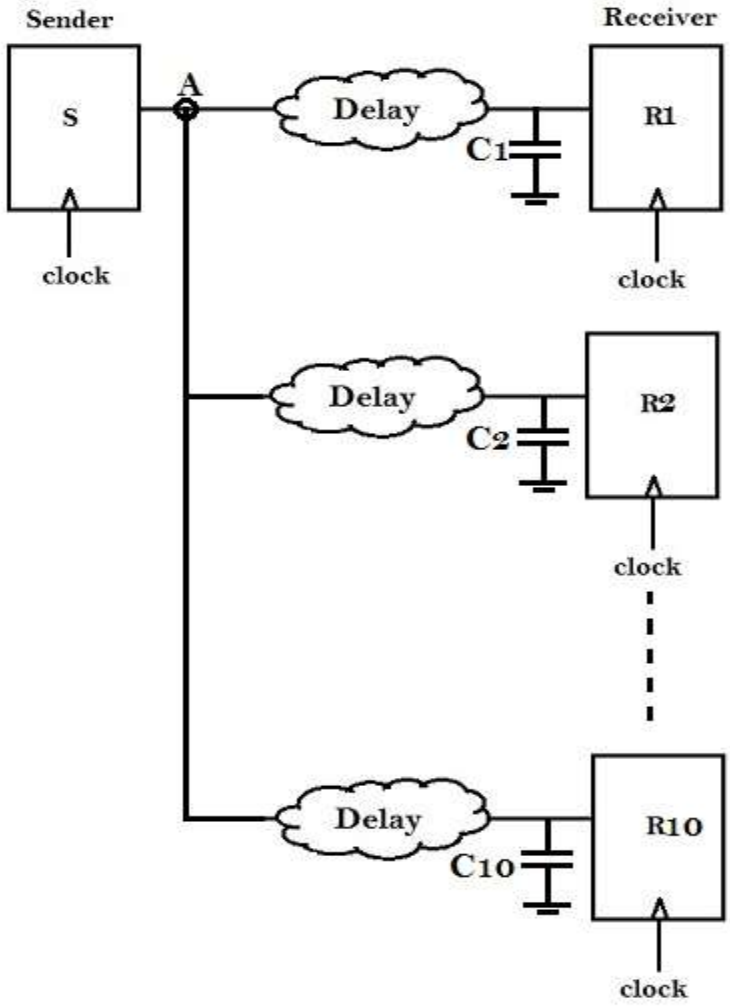
In the above figure, clock to sender reaches at Δ_1 and not at '0', whereas, clock to receiver reaches at ($T_{clk} + \Delta_2$) and not at T_{clk}. The modulus difference between Δ_1 and Δ_2 ($|\Delta_1 - \Delta_2|$) is referred to as **skew**.

Now, the defining equations for setup and hold time will change, and they are as follows:

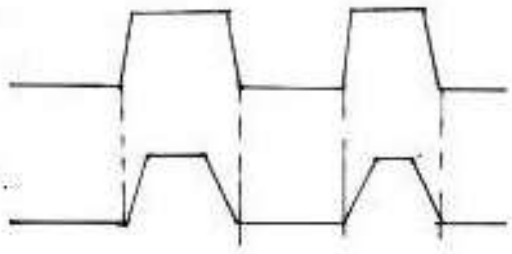
$m > H + \Delta_1$ i.e. Minimum propagation delay of the combinational logic should be greater than Hold Margin + the clock network delay for sender

$M < (T_{clk} - S) + \Delta_2$ i.e. Maximum propagation delay of the combinational logic should be less than Clock period (T_{clk} minus the Setup Margin) + clock network delay for receiver i.e. Δ_2

Now, consider the following scenario, where there is only one sender, and multiple (say '10') receivers.



In the above scenario, node ‘A’ is connected to ’10’ Receivers, i.e. the load on node ‘A’ has increased ’10x’ times compared to previous examples. So now, the time required for node ‘A’ to charge all the ’10’ input capacitances of the receiver, will increase, and the waveform at ‘A’, will look like as shown in figure below.



Waveform at node 'A' with one receiver

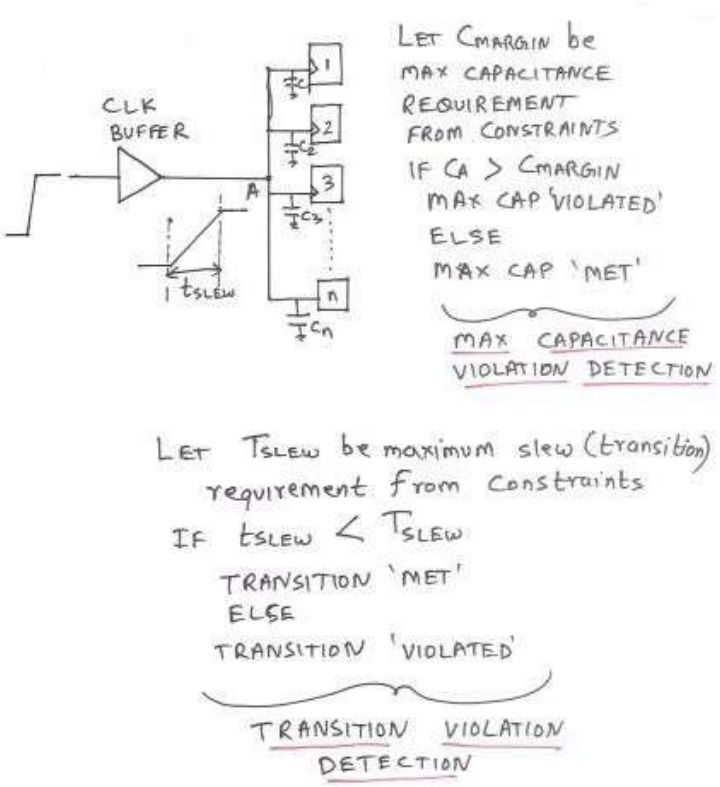
Waveform at node 'A' with 10 receivers

Every system has got a margin for ‘transition’ or ‘slew’ (say t_{slew}) and ‘max capacitance’ (say C_{margin}). Hence, if the transition (or slew) on node ‘A’ exceeds t_{slew}, it would result into timing violation, usually called as ‘transition’ violation.

Similarly, if the capacitance on node ‘A’ exceeds C_{margin}, it would result into violation, usually called as ‘max_capacitance’ violation.

The best way to fix max_capacitance and transition violation, is to either increase the drive strength of the Sender or buffer tree insertion

Following figure summarizes transition and max_capacitance requirement and violations.



Above figure shows an example of transition and max_capacitance violation on the output pin of clock buffer, which is connected to the ‘clk’ pin of ‘n’ receivers.