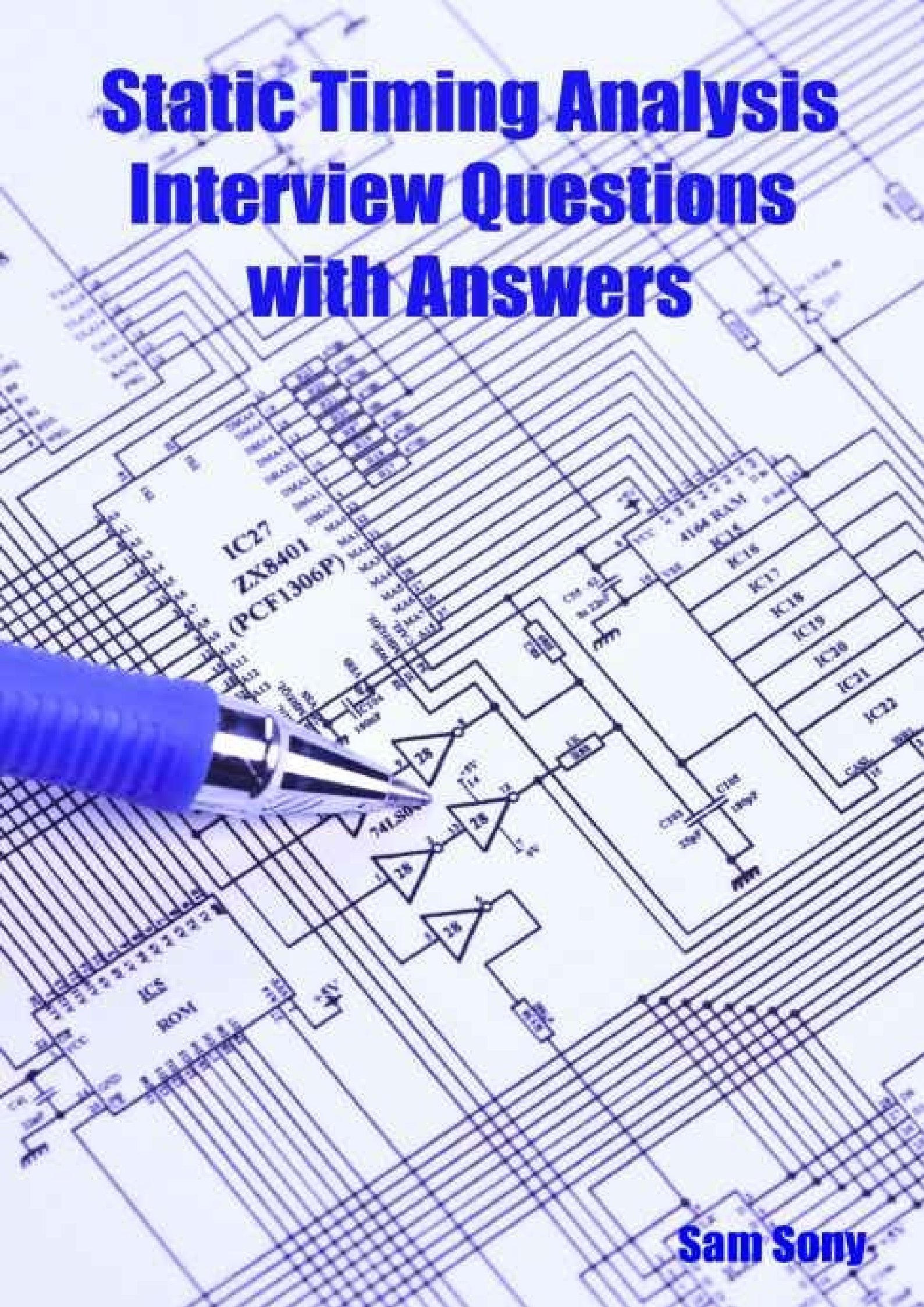
Static Timing Analysis Interview Questions with Answers



Sam Sony

Contents

What is STA ?

Question W1): What is STA (Static Timing Analysis) ?
Answer W1):
Static Timing Analysis is a technique of analysing timing paths in a digital logic by adding up delays along a timing path (both gate and interconnect) and comparing it with constraints (clock period) to check whether the path meets the constraint.

In contrast to the dynamic spice simulation of whole design, static timing analysis performs a worst case analysis using very simple models of device and wire delays. A lookup table model or a simple constant current or voltage source based model of device is used. Elmore delay or equivalent model is used to quickly figure out wire delays.

Static Timing Analysis is popular because it is simple to use and only needs commonly available inputs like technology library, netlist, constraints, and parasitics(R and C).

Static Timing Analysis is comprehensive and provides a very high level of timing coverage. It also honours timing exception to exclude the paths that are either not true path are not exercised in an actual design. A good static timing tool correlates well with actual silicon.

Question W2): What are all the items that are checked by static timing analysis ?
Answer W2):
Static Timing Analysis is used to check mainly the setup and hold time checks. But it also checks for the assumptions made during timing analysis to be holding true. Mainly it checks for cells to be within the library characterization range for input slope, output load capacitance. It also checks for integrity of clock signal and clock waveform to guarantee the assumptions made regarding the clock waveforms. A partial list of things it checks is here :

Setup Timing
Hold timing
Removal and Recovery Timing on resets
Clock gating checks
Min max transition times
Min/max fanout
Max capacitance
Max/min timing between two points on a segment of timing path.
Latch Time Borrowing
Clock pulse width requirements

Setup and Hold Time Violations.

Question S1): Describe a timing path.
Answer S1):
For standard cell based designs, following figure illustrates basic timing path. Timing path typically starts at one of the sequential (storage element) which could be either a flip-flop or a latch.

The timing path starts at the clock pin of the flip-flop/latch. Active clock edge on this element triggers the data at the output of such element to change. This is the first stage delay which is also called clock -> data out(Q) delay.

Then data goes through stages of combinational delay and interconnect wires. Each of such stage has its own timing delay that accumulates along the path. Eventually the data arrives at the sampling storage element, which is again a flip-flop or a latch.

That's where data has to meet setup and hold checks against the clock of the receiving flip-flop/latch. Also notice for the timing paths in the same clock domain, generating flip-flop clock and sampling flip-flop clocks are derived from a single source, which is called the point of divergence.

In reality, actual start point for a synchronous clock based circuits is the first instance where clocks branch off to generating path and sampling path as shown here in the picture, which is also called point of divergence.

To simplify analysis we agree that clock will arrive at very much a fixed time at the clock pin of all sequentials in the design. This simplified the analysis of the timing path. from one sequential to another sequential.

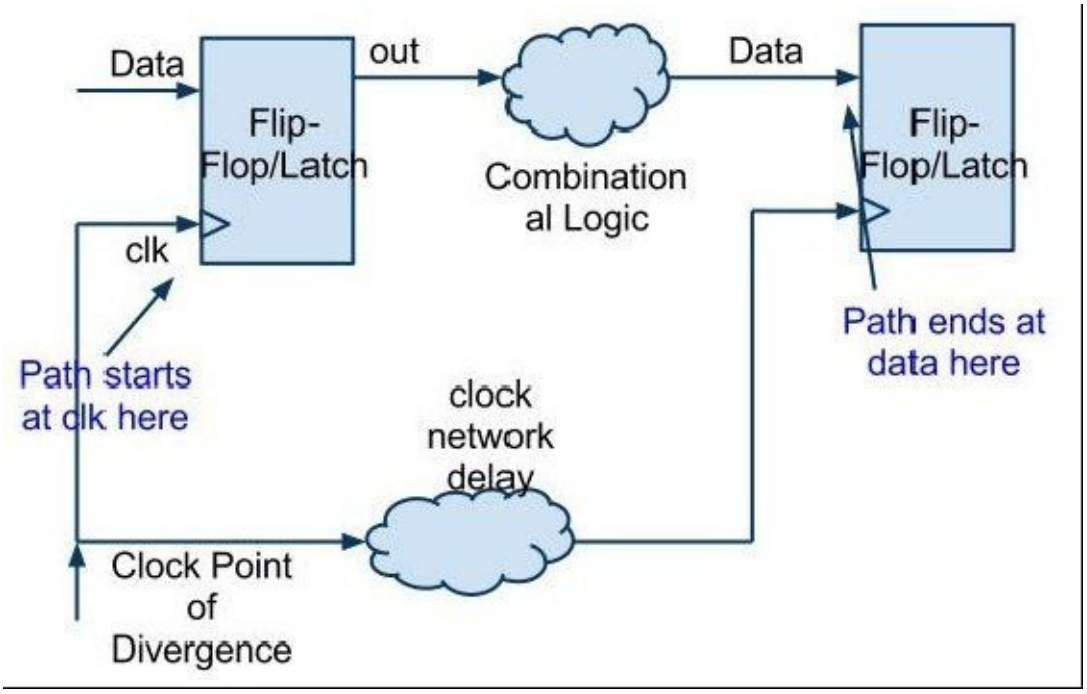


Figure S1. Timing path from one Flipflop to another Flipflop.

Question S2): What are different types of timing paths ?

Answer S2):

A digital logic can be broken down into a number of timing paths. A timing path can be any of the following:

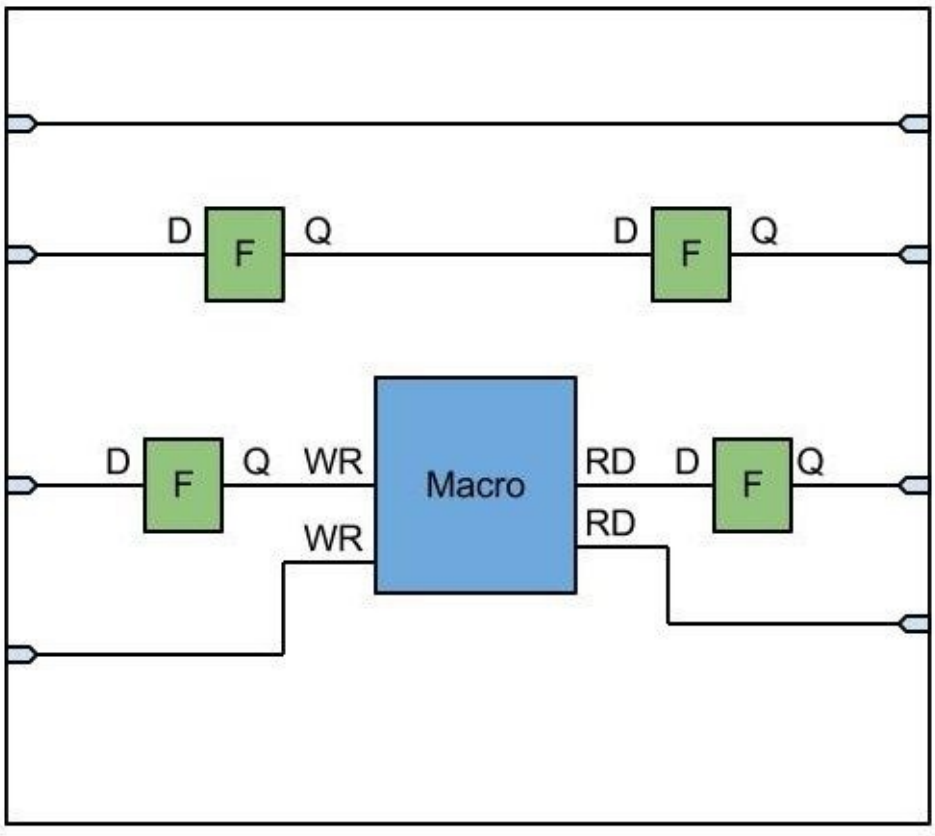


Figure S2. Various types of timing paths.

i. A path between the clock pin of register/latch to the d-pin of another register/latch.
ii. A path between primary input to the d-pin of a register or latch.
iii. A path between clock-pin of a register to a primary output.
iv. A timing path from primary input to macro input pin.
v. A timing path from macro output pin to primary output pin.
vi. A timing path from a macro output pin to another macro input pin(not shown in the figure)
vii. A path passing through input pin and output pin of a block through combinational logic inside the block.

Question S3): What is a launch edge ?
Answer S3):
In synchronous design, certain activity or certain amount of computation is done within a clock cycle. Memory elements like flip-flop and latches are used in synchronous designs to hold the input values stable during the clock cycle while the computations are being performed.

Beginning of the clock cycle initiate the activity and by the end of the clock cycle activity has to be completed and results have to be ready. Memory elements in a design transfer data from input to output on either rising or the falling edge of the clock. This edge is called the active edge of the clock.
During the clock cycle, data propagates from output of one memory element, through the combinational logic to the input of second memory element. The data has to meet a certain arrival time requirement at the input of the second memory element.

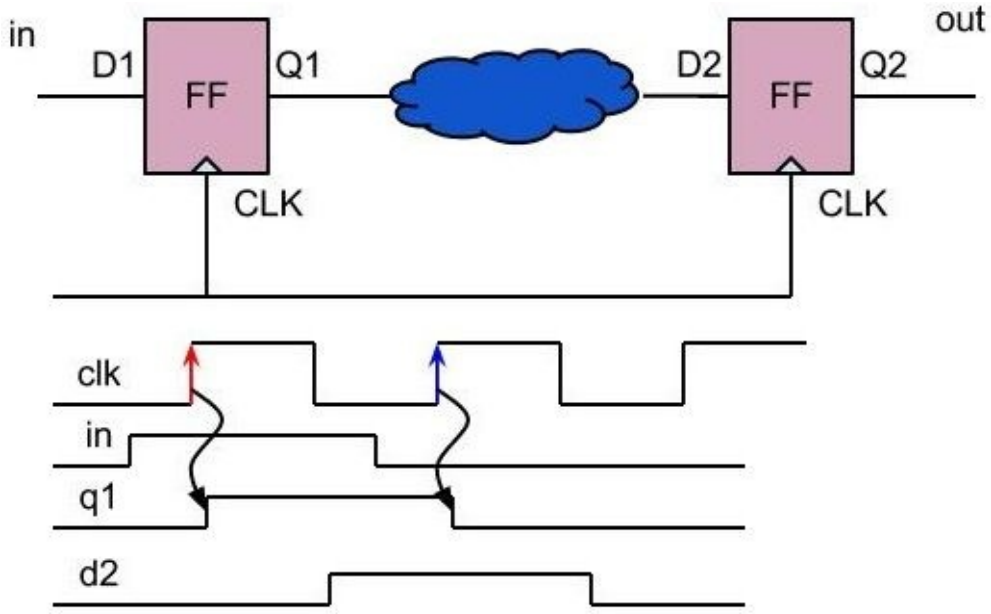


Figure S3. Launch edge and capture edge.

As shown in the above figure, the active edge of the clock(shown in red) at the first memory element makes new data available at the output of the memory element and starts data to propagate through the logic. Input ‘in’ has risen to one before the first active(rising) edge of the clock, but this value of ‘in’ is transferred to Q1 pin only when clock rises. This active edge of the clock is called the launch edge, because it launches the data at the output of first memory element, which eventually has to be captured by next memory element along the data propagation path.

Question S4): What is capture edge ?
Answer S4):
As we discussed in previous question, the way synchronous circuits work, certain amount of computation has to be done within a clock cycle. At the launch edge of the clock, memory elements transfer fresh set of data at the output pin of the launching memory elements. This new data, ripples through the combinational logic that carries out the stipulated computation.

By the end of the clock cycle, new computed data has to be available at the next set of memory elements. Because next active clock edge, which signifies the end of one clock cycle, captures the computed results at the D2 pin of the memory element and transfers the results to the Q2 pin for the subsequent clock cycle. This next active edge of the clock, show in blue at figure 1, is called the capture edge, as it really is capturing the results at the end of the clock cycle.

There are some caveats to be aware of. The data D2 has to arrive certain time before the capture edge of clock, in order to be captured properly. This is called setup time requirement, which we will discuss later.

Although it is said that computation has to be done within one clock cycle, it is not always the case. In general it is true that computation has to be done within one clock cycle, but many times, computation can take more than one cycle. When this happens we call it a multi cycle path.

Question S5): What is setup time ?
Answer S5):
For any sequential element e.g. latch or flip-flop, input data needs to be stable when clock-capture edge is active. Actually data needs to be stable for a certain time before clock-capture edge activates, because if data is changing near the clock-capture edge, sequential element (latch or flip-flop) can get into a metastable state and it could take unpredictable amount of time to resolve the metastability and could settle at at state which is different from the input value, thus can capture unintended value at the output.

The time requirement for input data to be stable before the clock capture edge activates is called the setup time of that sequential element.

Question S6) What is hold time ?

Answer S6)
As we saw in previous question about setup time, for any sequential element e.g. latch or flip-flop, data needs to be held stable when clock-capture edge is active. Actually data needs to be held stable for a certain time after clock-capture edge deactivates, because if data is changing near the clock-capture edge, sequential element can get into a metastable state and can capture wrong value at the output.

This time requirement that data needs to be held stable for after the clock capture-edge deactivates is called hold time requirement for that sequential.

Question S7): What does the setup time of a flop depend upon ?
Answer S7):
Setup time of a flip-flop depends upon the Input data slope, Clock slope and Output load.

Question S8): What does the hold time of a flip-flop depend upon ?
Answer S8):
Hold time of a flip-flop depends upon the Input data slope, Clock slope and Output load.

Question S9) Explain signal timing propagation from one flip-flop to another flip-flop through combinational delay.
Answer S9)
Following is a simple structure where output of a flop goes through some stages of combinational logic, represented by pink bubble and is eventually samples by receiving flop. Receiving flop, which samples the FF2_in data, poses timing requirements on the input data signal.

The logic between FF1_out to FF2_in should be such that signal transitions could propagate through this logic fast enough to be captured by the receiving flop. For a flop to correctly capture input data, the input data to flop has to arrive and become stable for some period of time before the capture clock edge at the flop.

This requirement is called the setup time of the flop. Usually you'll run into setup time issues when there is too much logic in between two flop or the combinational delay is too small. Hence this is sometimes called max delay or slow delay timing issue and the constraints is called max delay constraint.

In figure there is max delay constraint on FF2_in input at receiving flop. Now you can realize that max delay or slow delay constraint is frequency dependent. If you are failing setup to a flop and if you slow down the clock frequency, your clock cycle time increases, hence you've larger time for your slow signal transitions to propagate through and you'll now meet setup requirements.

Typically your digital circuit is run at certain frequency which sets your max delay

constraints. Amount of time the signal falls short to meet the setup time is called setup or max, slack or margin.

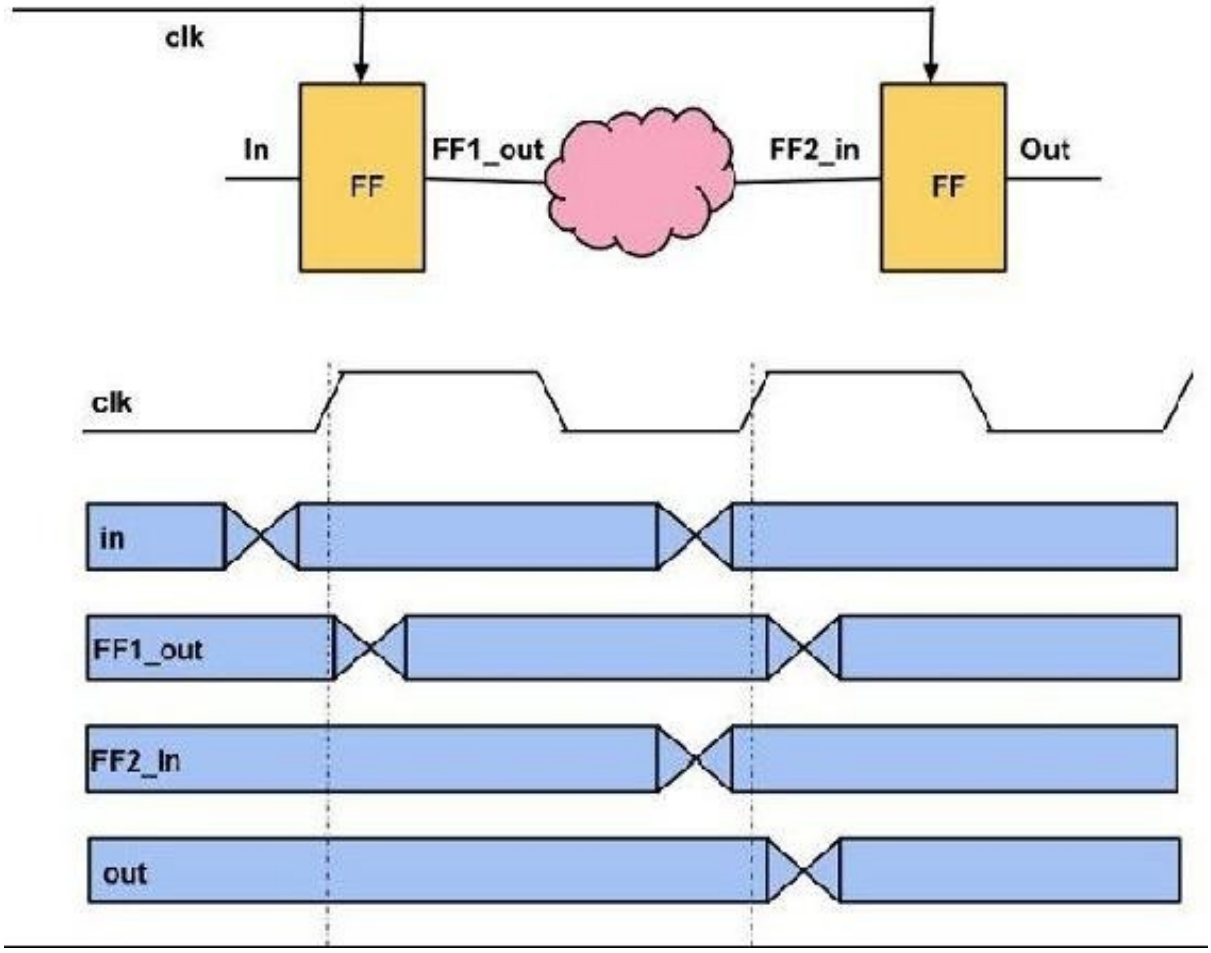


Figure S9. Signal timing propagation from flip-flop to flip-flop

Question S10) Explain setup failure to a flip-flop.

Answer S10)

Following figure describes visually a setup failure. As you can see that first flop releases the data at the active edge of clock, which happens to be the rising edge of the clock. FF1_out falls sometime after the clk1 rises.

The delay from the clock rising to the data changing at output pin is commonly referred to as clock to out delay. There is finite delay from FF1_out to FF2_in through some combinational logic for the signal to travel.

After this delay signal arrives at second flop and FF2_in falls. Because of large delay from FF1_out to FF2_in, FF2_in falls after the setup requirement of second flop, indicated by the orange/red vertical dotted line. This means input signal to second flop FF2_in, is not held stable for setup time requirement of the flop and hence this flop goes metastable and doesn't correctly capture this data at it's output.

As you can see one would've expected 'Out' node to go low, but it doesn't because of setup time or max delay failure at the input of the second flop. Setup time requirement dictates that input signal be steady during the setup window (which is a certain time before the clock capture edge).

As mentioned earlier if we reduce frequency, our cycle time increases and eventually FF2_in will be able to make it in time and there will not be a setup failure. Also notice that a clock skew is observed at the second flop. The clock to second flop clk2 is not aligned with clk1 anymore and it arrives earlier, which exacerbates the setup failure.

This is a real world situation where clock to all receivers will not arrival at same time and designer will have to account for the clock skew. We'll talk separately about clock skew in details

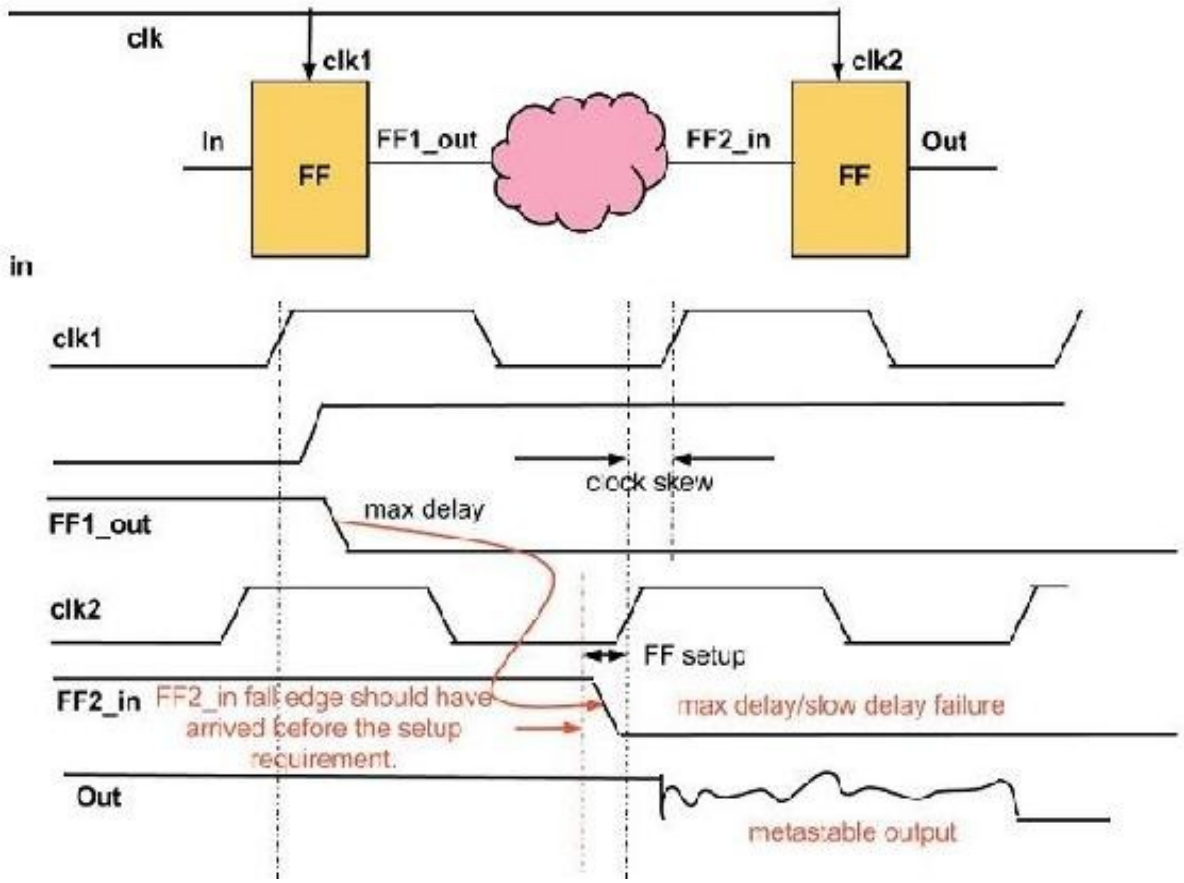


Figure S10. Setup/Max delay failure to a flip-flop.

Question S11) Explain hold failure to a flip-flop.
Answer S11)
Like setup, there is a 'Hold' requirement for each sequential element (flop or a latch). That requirement dictates that after the assertion of the active/capturing edge of the sequential element input data needs to be stable for a certain time/window.

If input data changes within this hold requirement time/window, output of the

sequential element could go metastable or output could capture unintentional input data. Therefore it is very crucial that input data be held till hold requirement time is met for the sequential in question.

In our figure below, data at input pin 'In' of the first flop is meeting setup and is correctly captured by first flop. Output of first flop 'FF1_out' happens to be inverted version of input 'In'.

As you can see once the active edge of the clock for the first flop happens, which is rising edge here, after a certain clock to out delay output FF1_out falls. Now for sake of our understanding assume that combinational delay from FF1_out to FF2_in is very very small and signal goes blazing fast from FF1_out to FF2_in as shown in the figure below.

In real life this could happen because of several reasons, it could happen by design (imagine no device between first and second flop and just small wire, even better think of both flops abutting each-other), it could be because of device variation and you could end up with very very fast device/devices along the signal path, there could be capacitance coupling happening with adjacent wires, favoring the transitions along the FF1_out to FF2_in, node adjacent to FF2_in might be transitioning high to low(fall) with a sharp slew rate or slope which couples favorably with FF2_in going down and speeds up FF2_in fall delay.

In short in reality there are several reasons for device delay to speed up along the signal propagation path. Now what ends up happening because of fast data is that FF2_in transitions within the hold time requirement window of flop clocked by clk2 and essentially violates the hold requirement for clk2 flop.

This causes the the falling transition of FF2_in to be captured in first clk2 cycle where as design intention was to capture falling transition of FF2_in in second cycle of clk2.

In a normal synchronous design where you have series of flip-flops clocked by a grid clock(clock shown in figure below) intention is that in first clock cycle for clk1 & clk2, FF1_out transitions and there would be enough delay from FF1_out to FF2_in such that one would ideally have met hold requirement for the first clock cycle of clk2 at second flop and FF2_in would meet setup before the second clock cycle of clk2 and when second clock cycle starts, at the active edge of clk2 original transition of FF1_out is propagated to Out.

Now if you notice there is skew between clk1 and clk2, the skew is making clk2 edge come later than the clk1 edge (ideally we expect clk1 & clk2 to be aligned perfectly, that's ideally !!). In our example this is exacerbating the hold issue, if both clocks were perfectly aligned, FF2_in fall could have happened later and would have met hold requirement for the clk2 flop and we wouldn't have captured wrong data !!

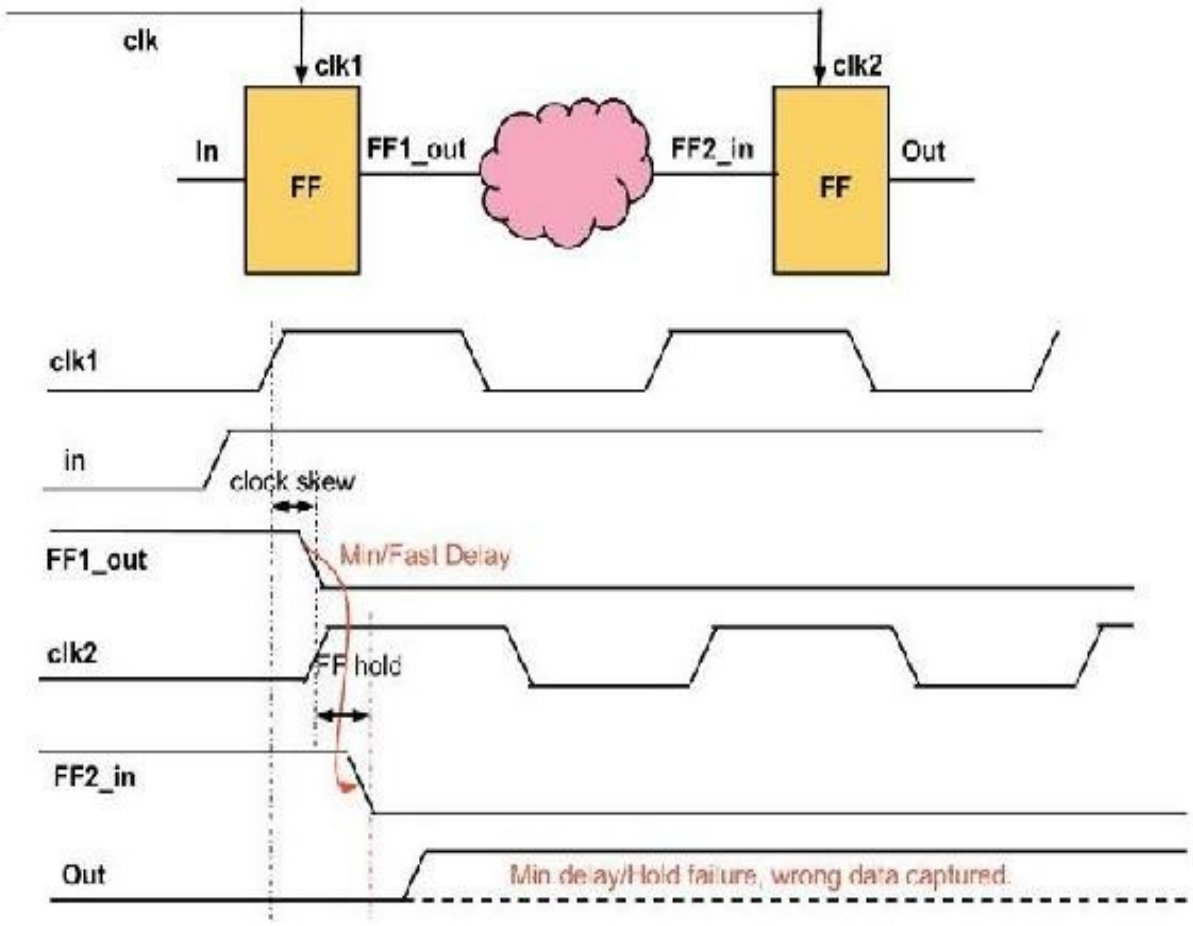


Figure S11. Hold/Min delay requirement for a flop.

Question S12): If hold violation exists in design, is it OK to sign off design? If not, why?

Answer S12):

No you can not sign off the design if you have hold violations. Because hold violations are functional failures. Setup violations are frequency dependent. You can reduce frequency and prevent setup failures. Hold violations stemming from the same clock edge race, are frequency independent and are functional failures because you can end up capturing unintended data, thus putting your state machine in an unknown state.

Question S13) What are setup and hold checks for clock gating and why are they needed ?

Answer S13):

The purpose of clock gating is to block the clock pulses and prevent clock toggling. An enable signal either masks or unmasks the clock pulses with the help of an AND gate. As it is clock signal which is in consideration here, care has to be taken such that we do not change the shape of the clock pulse that we are passing through and we don’t introduce any glitches in the clock pulse that we are passing through.

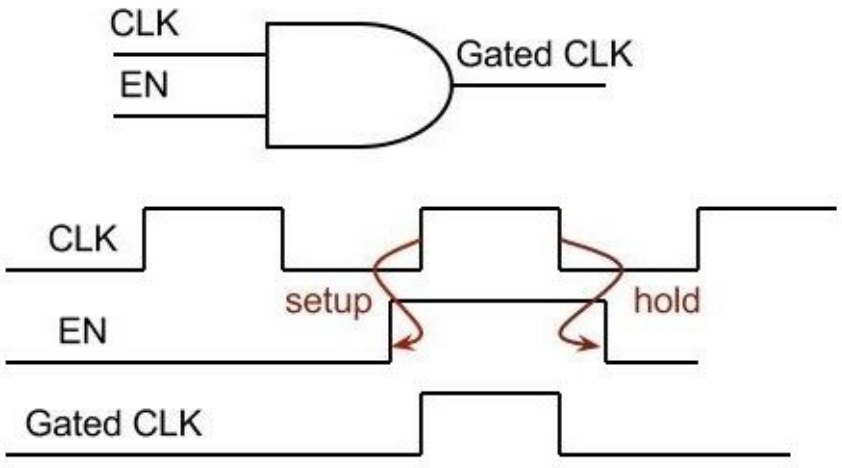


Figure S13. Clock gating setup and hold check

As you can see in the figure the enable signal has to setup in advance of the rising edge of the clock in such a way that it doesn't chop the rising edge of the clock. This is called the clock gating setup or clock gating default max check.
Similarly the tuning off or going away edge of the enable(EN) signal has to happen well past the turning off or going away edge of the clock, again to make sure it doesn't get chopped off. This is called the clock gating hold or clock gating default min check.

Question S14): What determines the max frequency a digital design will work on. Why hold time is not included in the calculation for the above ?
Answer S14):
Worst max margin will decide the max frequency a design will work on. As setup failure is frequency dependent. Hold failure is not frequency dependent hence it is not factored into the frequency calculation.

Question S15). One chip which came back after being manufactured fails setup test and another one fails a hold test. Which one may still be used how and why ?
Answer S15):
Setup failure is frequency dependent. If certain path fails setup requirement, you can reduce frequency and eventually setup will pass. This is because when you reduce frequency you provide more time for the flop/latch input data to meet setup. Hence we call setup failure a frequency dependent failure. While hold failure is not frequency dependent. Hold failure is functional failure.

Following figure shows frequency dependence of setup failure.

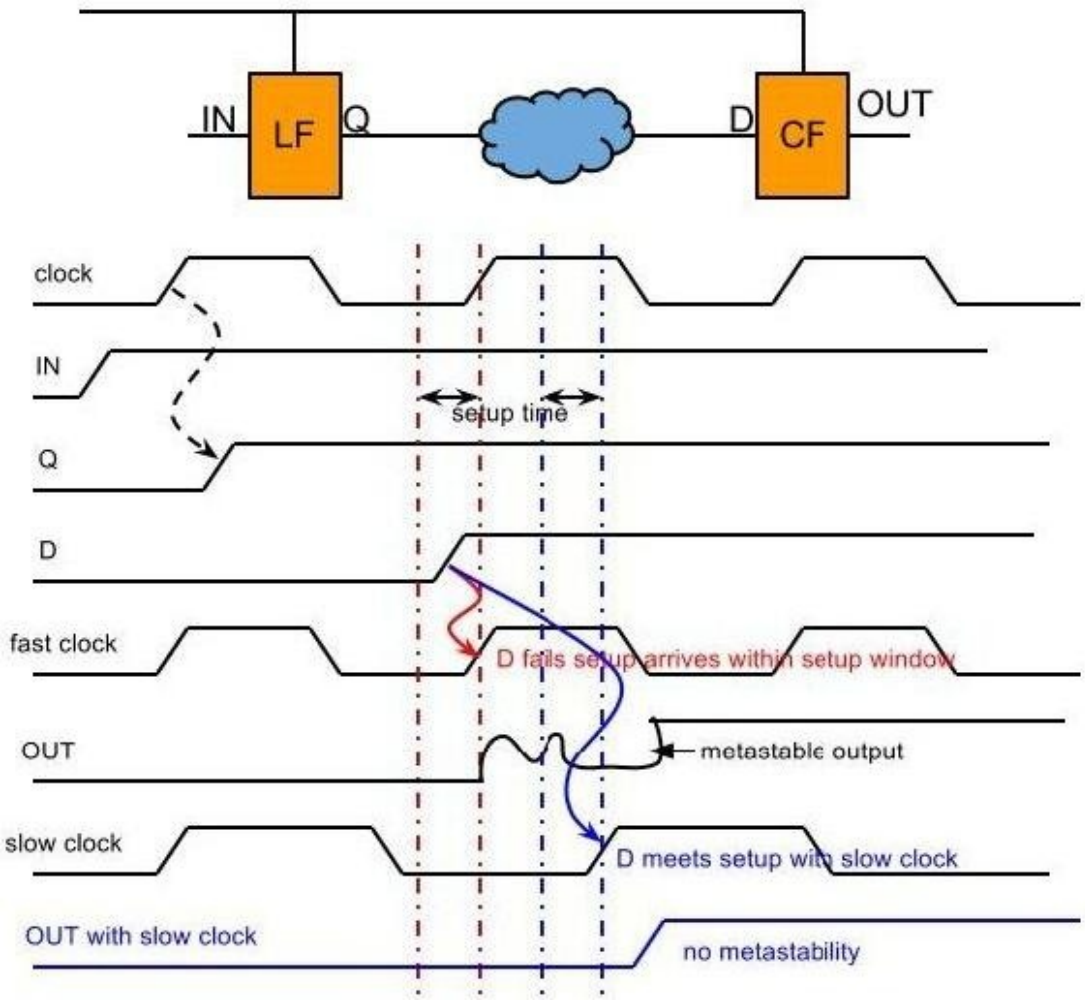


Figure S15a. Frequency dependence of setup failure.

You can see in the above figure that with faster clock, D input to the capture edge fails setup. The red vertical line shows the setup window of the capture flop. The D input should have arrived before the setup window shown by the red dotted vertical lines. As D fails setup the output node OUT goes metastable and takes some time before it settles down. This metastability could cause problems downstream in the circuit.

Now if the clock is slowed down, you can see that D will meet the setup for the capture flop. Although it is not shown in the figure but for simplicity reasons, but the launch clock is also slow now, although the launch clock can be assumed to the same as fast clock. You can see that setup window doesn't change with clock as it is the property of the capture flop and doesn't depend upon clock. That is why we can meet setup with slow clock.

Following figure illustrates why slowing down frequency doesn't resolve hold failures.

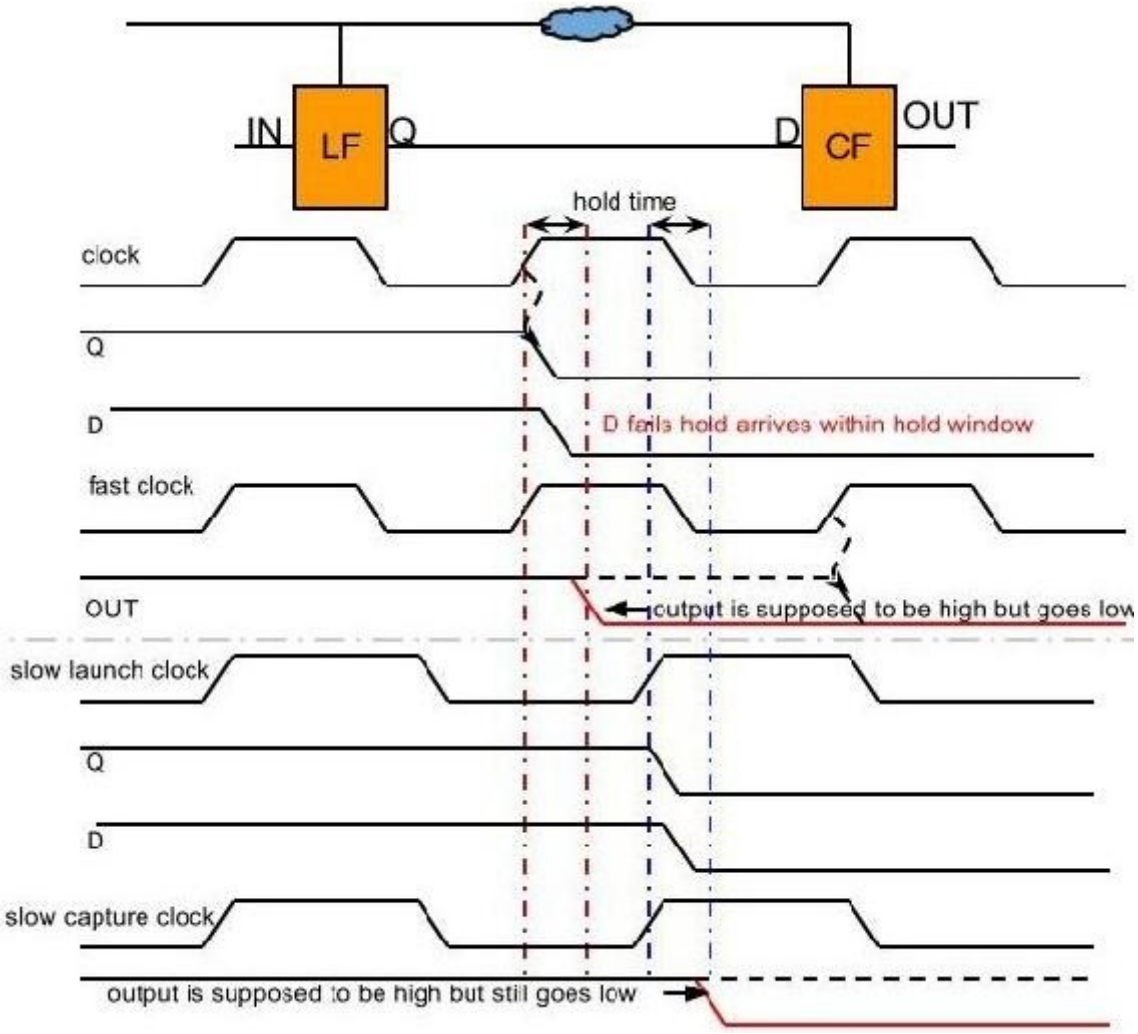


Figure S15b. Frequency independent hold failure.

As you can see in the figure, the hold failure is a data race. Because ‘IN’ goes low, the Q output of launch flop (LF) goes low and this is supposed to be captured by capture flop (CF) and output of capture is supposed to go low after a clock cycle. But because there is no (very small) delay from Q to D, D goes low within the hold window of the capture flop. In other words D goes low and violates the hold time for the capture flop. In such cases either capture flop output can go metastable or the new value of D could be captured right away at the output ‘OUT’ of the capture flop. In the figure above it is shown that ‘OUT’ also goes low right away. The design intention was for ‘OUT’ to go low after a clock cycle but because of fast data, data at input ‘D’ snuck into the current clock cycle and appeared at the ‘OUT’, causing ‘OUT’ to have wrong value for this clock cycle. This means unknown state for the downstream logic, because of the wrong ‘OUT’ value.

As you can see in the bottom portion of the waveforms, even if the slower clock is used, the problem persists. Because this is really a data race issue because of fast data delay from Q to D, which is still there even if we change the clock frequency as it is independent of the clock frequency. Hence we can see that hold failures could be frequency independent.

Question S16): What is Max Timing Equation ?
Answer S16):

Best way to understand max timing equation is to look at the waveforms. Please go through following figure carefully.

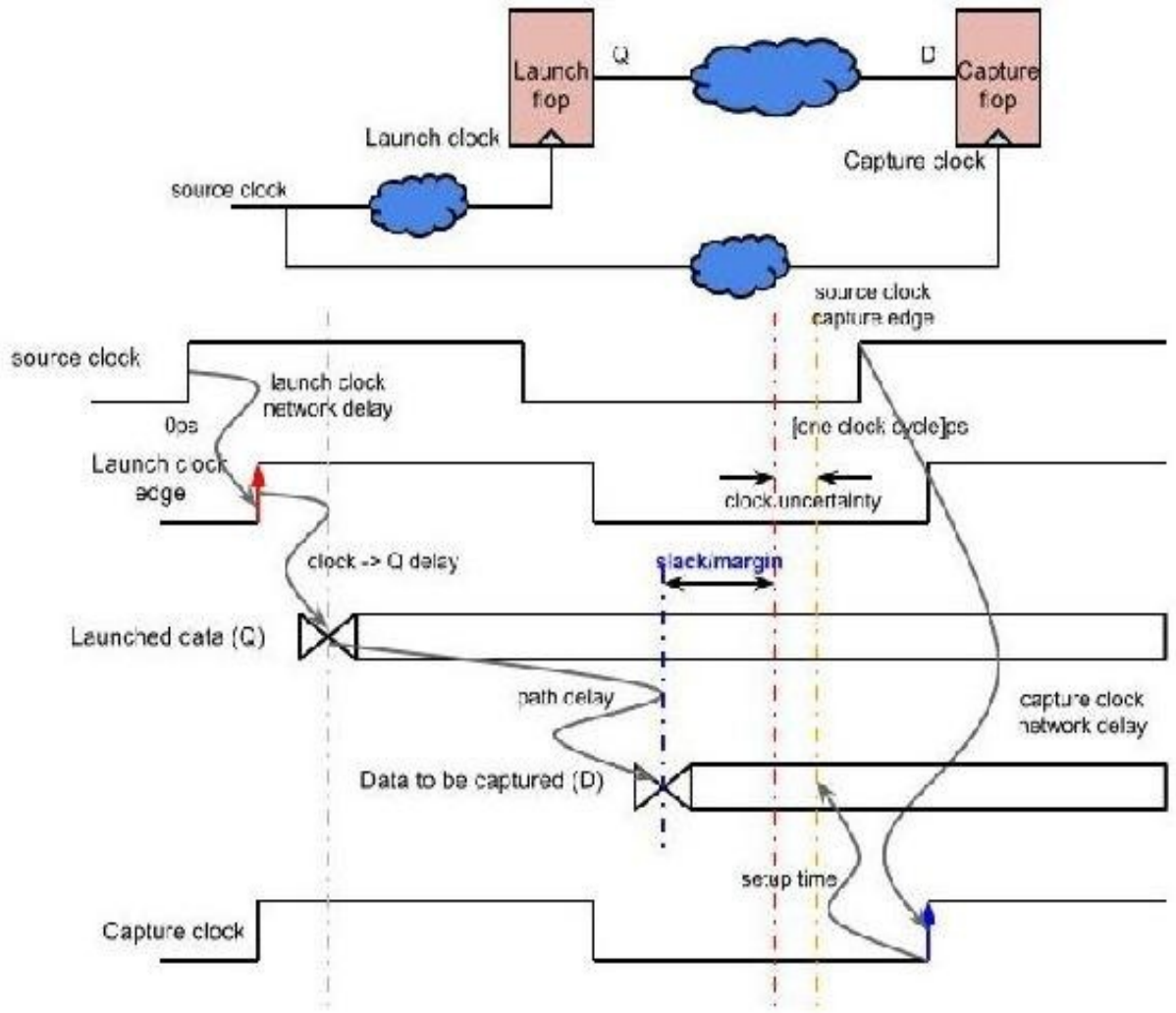


Figure S16 Max timing equation.

Above mentioned figure visually describes what constitutes a max or setup timing path along with all the components that are involved in coming up with the max timing slack.

Source clock in the above figure is the original source of the clock which could be PLL output or wherever the starting point of the source clock is defined. For clocks which are not the direct output of PLLs, or coming from primary chip input, they are referred to as derived clock, virtual clock or generated clock. This is our master reference clock and most of the time this is the start point or the 0ps point.

We start with the source clock at 0 ps in time. From source clock there is clock network delay from the source clock to the launch flop that we add up. One the launch clock active edge arrives at the launch flop, it releases data after clock to Q delay, we add this up. From Q pin of the flop data travels through cells and wires to arrives at the D input pin of the capture flop. This is called the path delay as this is the path from launch flop to capture flop, we add this up. The sum so far represents the data arrival at the capture flop input pin. This event has to happen before the setup requirement, or in other words, this sum has to be less than or equal to the setup or capture requirements, shown in figure with vertical dashed red line. Remember that in STA we worst case the analysis, hence we will take the slowest delay upto the capture flop input.

Lets look at the capture requirement. We know that capture happens one cycle later with respect to the launch clock, hence we start with source clock capture edge which is one cycle later with respect to launch edge at time equivalent to one clock cycle. Similar to launch there is clock network delay from source clock to the capture flop, we add this up as this in reality is pushing out the capture clock. To worst case, we use the fastest capture clock delay, because faster the capture clock less time we will have to meet setup. Now once the capture clock arrives at the capture flop, the input data at the flop has to meet the setup requirement. This is a requirement where by the input data to the capture flop has to arrive that much earlier, hence we subtract setup time from our capture requirement calculation. On top of this we need to account for clock uncertainty as because of variation, IR drop and other reasons, actual clock arrival times could vary and we need to build additional margin for this uncertainty. This is a penalty or requirement and as such forces data to arrive even earlier, which means we subtract this value from the capture requirement.

Source launch clock edge(0 ps) + Launch clock network slowest delay + Clock to Q slowest delay + Slowest Path delay (cell + interconnect) =< Source capture clock edge(One clock cycle) + Capture clock network fastest delay - Setup time - Max clock uncertainty.

Also.

Max margin/slack = [Source capture clock edge(One clock cycle) + Capture clock network fastest delay - Setup time - Max clock uncertainty] - [Source launch clock edge(0 ps) + Launch clock network slowest delay + Clock to Q slowest delay + Slowest path delay (cell + interconnect)]

Question S17): What is min timing equation ?
Answer S17):

Lets go through the following waveforms to better understand the min timing equation.

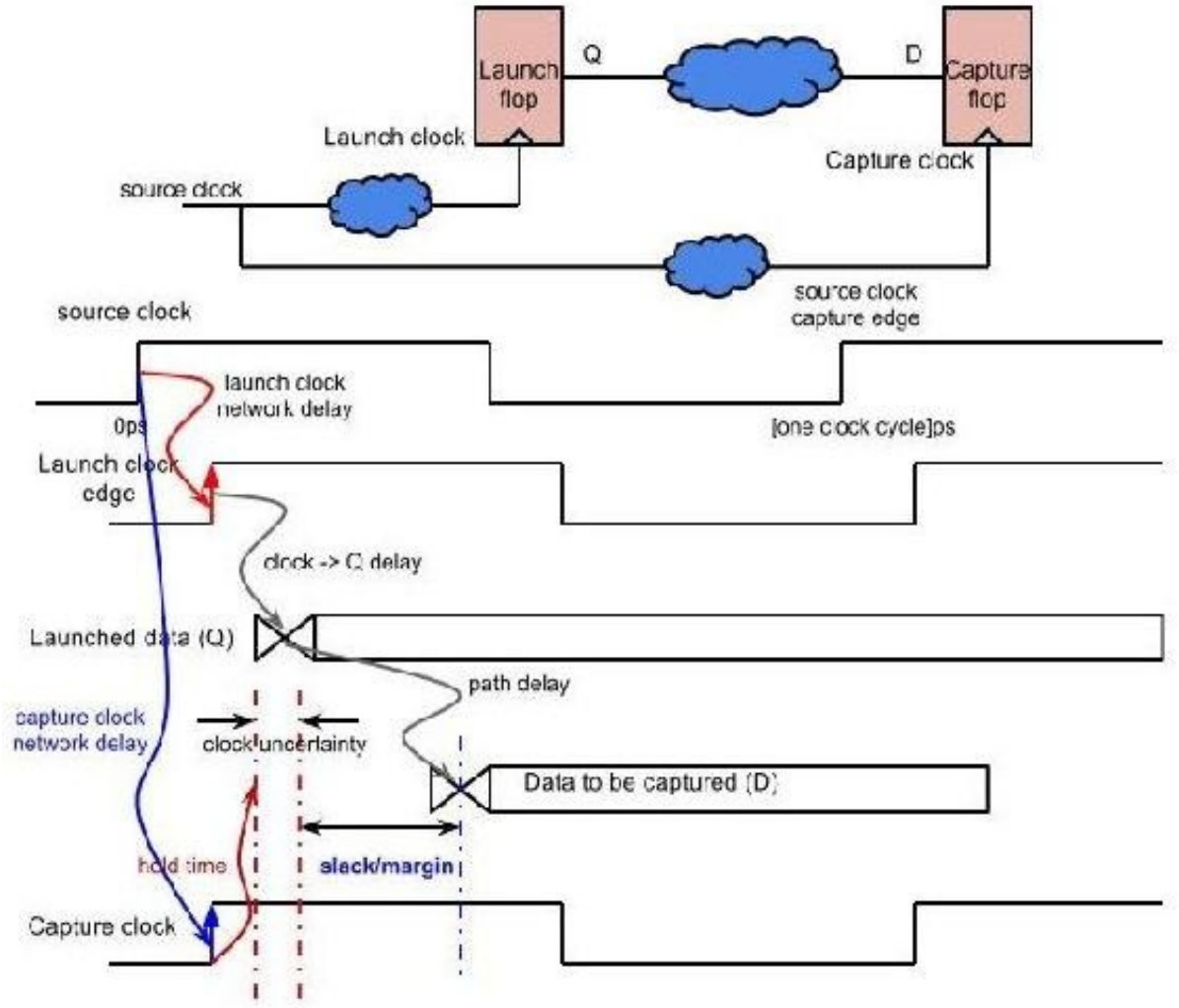


Figure S17. Min timing equation.

As we know that min timing check or hold time check is essentially ensuring that the data launched on the launch edge at the launch flop is not inadvertently captured by the capture flop at the launch edge, because launched data is supposed to be captured one cycle later and not in the current clock itself.

Just like max timing source clock is the master reference and source clock start(rising) edge is the start point. From there just clock travels to the launch flop through launch clock network, so we add up launch clock network delay. Once launch clock edge arrives at the launch flop, it releases the data at the output of launch flop after clock to Q delay, we add up this delay. Next we add up path delay. Now the data has arrived at capture flop. This data has to have arrived after the hold or min time requirement. Next we calculate hold time requirements.

For hold requirement on capture side we start with the same clock edge that we started on at the launch side. One of the alternative way to look at this is to look at setup capture clock edge for the same launch and capture flop timing path and pick clock edge which is one clock cycle earlier. Actually that is how many of the timing tools

like PrimeTime figure out which edge to check hold time requirement against. The tool first find out the setup requirement capture clock edge, which is one clock cycle after the launch edge, then it traces back one clock cycle, which is the same clock edge as the launch edge.

From this clock edge we add the hold time requirement, as input data arriving at the capture flop input pin, has to hold past the hold time requirement for that flop. We add clock uncertainty as clock edge at the capture flop could arrive that much later.

The launched data has to have arrived later than this hold time requirement at capture flop. Again to make the analysis worst case, we use the fastest delay upto capture flop input and we use slowest delay for the capture clock network.

Source clock launch clock edge(0ps) + Launch clock network fastest delay + Clock to Q fastest delay + Fastest path delay (cell + interconnect delays) >= Source clock launch edge(Source clock capture edge corresponding to the setup path - 1 clock period, same as 0ps) + Capture clock network slowest delay + Capture flop library hold time + Hold time clock uncertainty

And

Min margin = [Source clock launch clock edge(0ps) + Launch clock network fastest delay + Clock to Q fastest delay + Fastest path delay (cell + interconnect delays)] - [Source clock launch edge(Source clock capture edge corresponding to the setup path - 1 clock period, same as 0ps) + Capture clock network slowest delay + Capture flop library hold time + Hold time clock uncertainty]

Question S18): Is the clock period enough for the given circuit ?
Answer S18):

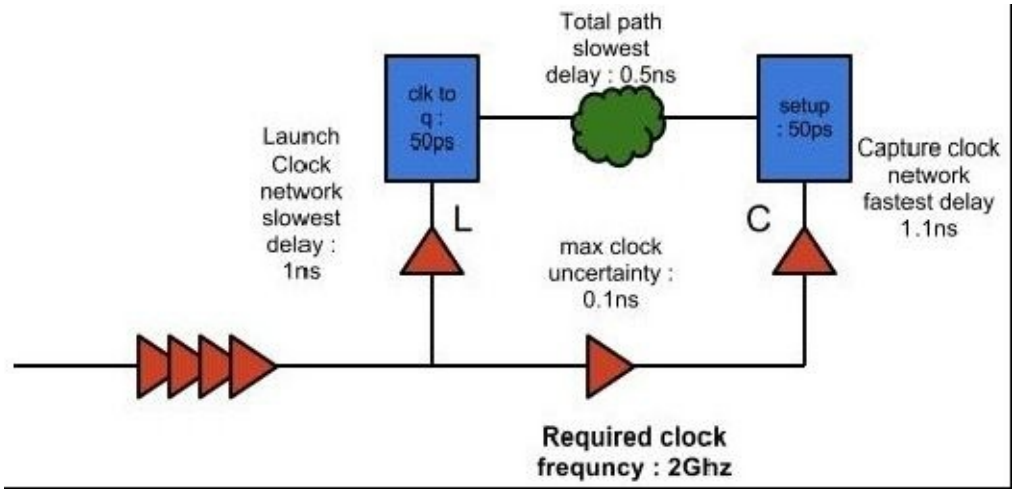


Figure S18: Clock frequency question.

We know the max timing equation.

Max margin = [Clock cycle + capture clock network fastest delay - setup time - max clock uncertainty] - [0ps + launch clock network slowest delay + clk to q slowest delay + slowest path delay]

Max margin = [0.5ns(2Ghz) + 1.1ns - 0.05ns - 0.1ns] - [1ns + 0.05ns + 0.5ns]

Max margin = [1.45ns] - [1.55ns] = -0.1ns

Max margin is negative means clock period is not enough and capture flop setup time check is violated.

Question S19): What is reset recovery time ?

Answer S19):

For a flip flop with asynchronous reset pin, only the asserting edge(active edge) of reset is asynchronous. Which means if reset pin is active low(reset bar), only reset signal going down(falling) can happen asynchronously without the knowledge of the clock. But once the reset has gone active, it has to de-assert at some point in time and has to get the flip flop out of the reset state. This reset de-assertion can not happen independently of the clocks. The way such flip flops are designed, the reset de assertion has to happen certain time before the active edge of the clock for the flip flop. This is very similar to setup check for data, and this requirement of reset de-assertion before the active edge of the clock is called the recovery time.

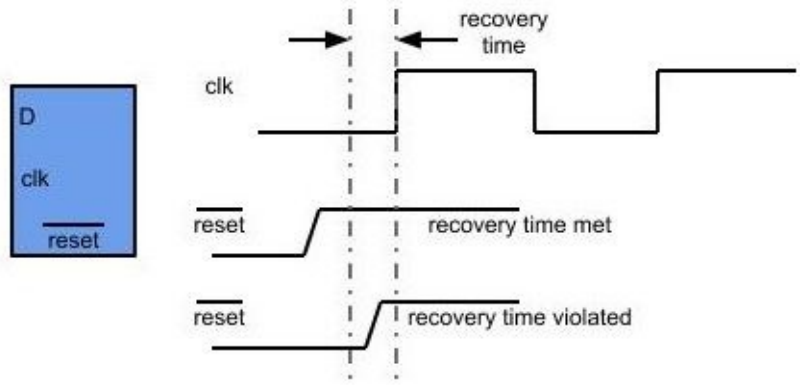


Figure S19. Reset recovery time

Question S20) : What is rest removal time.

Answer S20):

Removal time is the counterpart of recovery time. It is exactly hold time equivalent of recovery time. Just like in recovery time, reset deassertion has to happen certain time before the active edge of the clock, removal time requirement is where the reset deassertion has to hold past the active edge of the clock. Reset deassertion can not happen right around the clock edge, it has to happen certain time after the active edge of the clock.

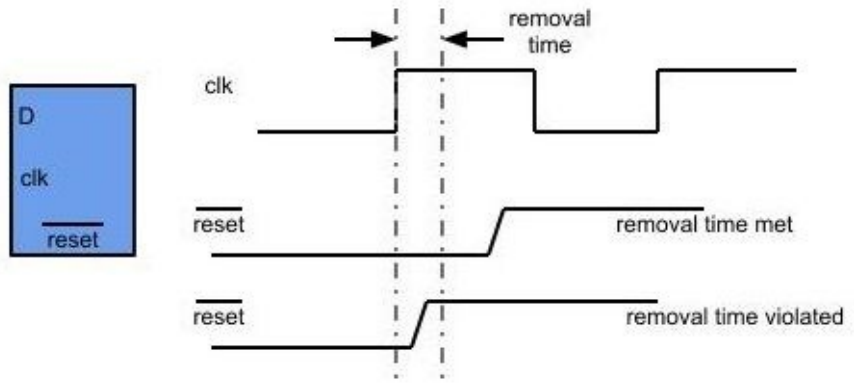


Figure S20. Reset removal time

Question S21): Given a setup check from an launch element to capture element, how does timing analysis tool decide to perform the hold check ?
Answer S21):

This question might seem vague at first, but the key is to understand following behavior of the timing analysis too. This is mainly applicable to PrimeTime tool, other STA tools may not follow the same method.

One key thing to remember is that, hold check is performed always with reference to setup check. WHich means timing tools first finds out which clock edges to perform setup check, and then it infers hold checks based on the setup check. For following analysis we assume both launch and capture flops are rising edge triggered. Normally for a setup check, the capture clock edge is chosen to be the active clock edge which comes one clock cycle after the launch edge.

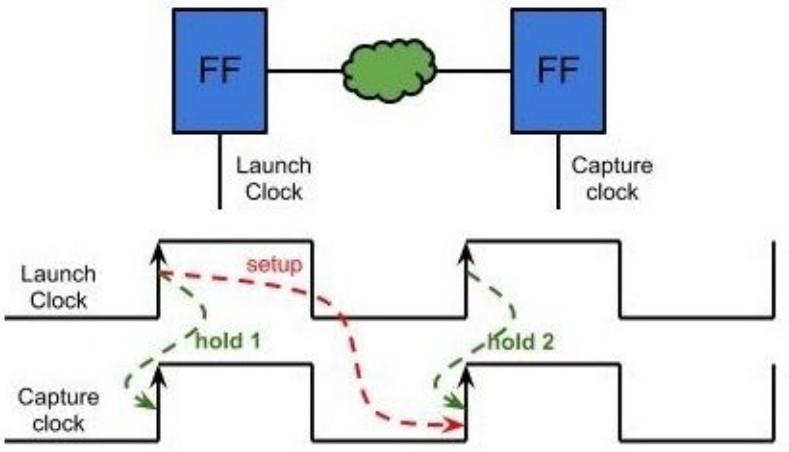


Figure S21. Hold & Setup clock edges.

We know that once an active edge of clock is picked as launch clock edge, the setup check is done to the capture edge is one clock cycle later.

Once setup check edges have been identified, the timing tools looks at two scenarios to find the clock edges to be picked up for the actual hold check. It first checks

whether the data launched by the launch clock edge corresponding to the setup check, is held enough to not inadvertently get captured by the same edge at the capture flop. This is shown in figure by green dotted arrow number 1. Then it looks for second scenario. This time it starts at the capture clock edge corresponding to the setup check and it ensures that the data released at the output of launch flop by this clock edge is not inadvertently captured by the capture flop at the same clock edge. This is shown in figure with green dotted arrow number 2. Having looked at both scenarios, timing tool picks the more stringent hold check and it performs that hold time check. In the above figure case, we can see that both scenarios, number 1 and number 2 are identical, so timing tool would just pick either.

One clarification about hold checks. The hold check is supposed to be more stringent when capture edge is very close to the launch edge, because that is when it is more likely that data launched by launch clock could be inadvertently get captured by nearby capture edge, which is in reality meant for the subsequent capture edge. As you see, more the launch edge happens later in time compared to capture edge, less the risk of hold time violation. Basically more the launch edge launches past the capture edge, more readily we know the data launched by launch edge will be held past the capture edge.

Question S22): What type of setup and hold checks will be performed when launch and capture clock are not of the same frequency ?

Answer S22):
Let's consider three case scenarios here.

Scenario 1) Launch clock is a multiple of capture clock and is twice as fast as capture clock.
Scenario 2) Capture clock is a multiple of launch clock and is twice as fast as launch clock.
Scenario 3) Launch and capture clocks are not multiple of each other.

One has to hammer this deep into their mind. Static timing analysis is a worst case analysis. Whenever a certain check is performed, tool will find the worst possible case to do the analysis or perform the check.

Take setup check. STA tool will always perform worst case setup check. Which means, once an active clock edge launches data at the launch flop, tool will find the earliest possible next active edge when the data can be captured at the capture flop. In other words, it will take launch clock and capture clock and find out the smallest distance between active launch edge and active capture edge, which is greater than zero(it won't pick the same edge, as it is obvious that it is not the correct edge) and it will use that for setup check.

And we know from previous question that it derives hold check with reference to setup check. Again in hold check, it looks at two scenarios and picks the worst one.

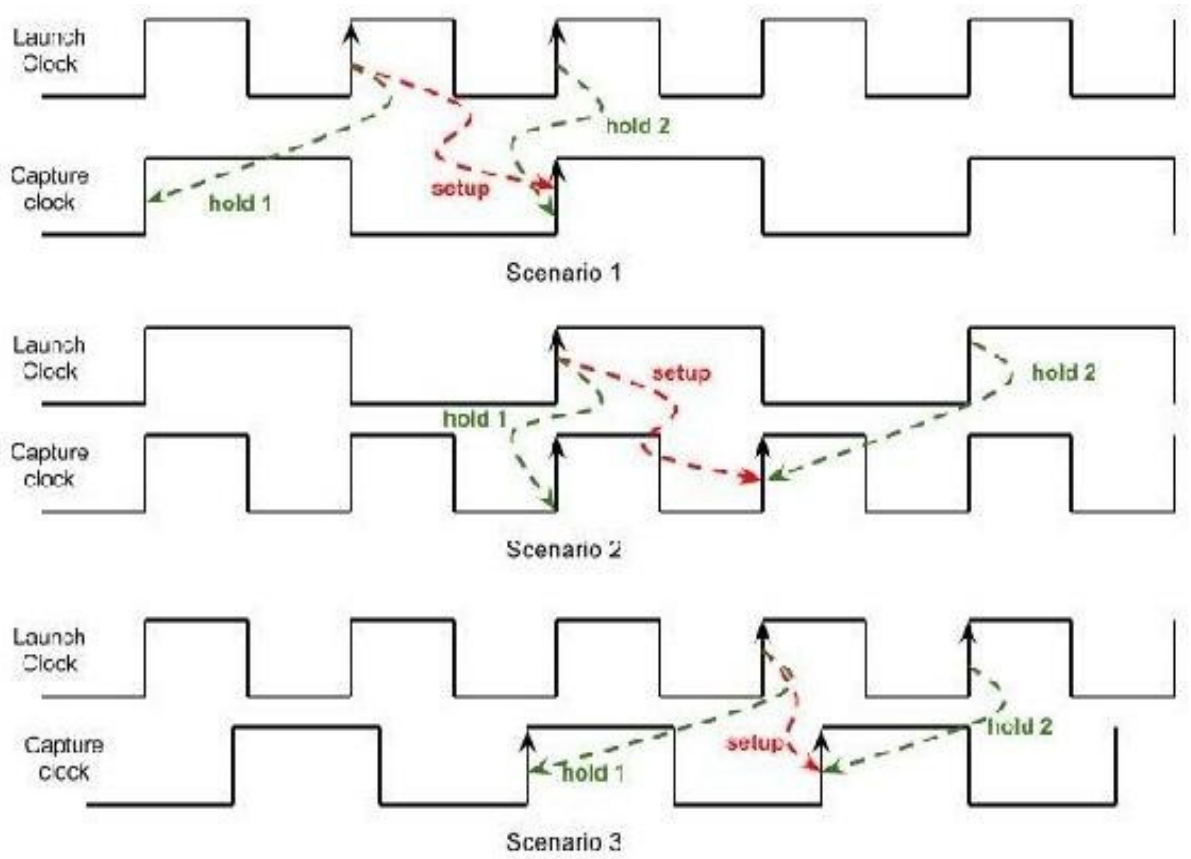


Figure S22 Setup and Hold clock edges.

We will assume rising edge launch and capture flops here. Lets look at scenario 1), here the launch clock is faster than the capture clock. As stated earlier, tool will pick the shortest distance between two active edges in launch and capture clock for the setup timing check. The clock edges and actual setup check is shown with dotted red line. Setup check is relatively straightforward. Once having found setup check, it will look at the two typical possibilities for finding hold check. Two hold check possibilities are shown with green dotted line. First one is from the launch of the setup check to the one clock cycle earlier than the setup capture edge of the capture clock. This is hold check number 1. Second possibility is the check from active edge of the launch clock which is one cycle later than the setup check launch edge to the setup capture edge of the capture clock again shown in green dotted line as hold check number 2. As you can see in the figure hold check 2 is more stringent, hence tool will pick scenario 2 for hold check.

In scenario 2) where launch clock is slower than the capture clock. Analysis is similar to earlier scenario and we can see that hold check 1 is more stringent, hence tool picks hold check 1.

In scenario 3) hold 2 seems to be more stringent and will be picked as the hold check.

Unless specific overrides or exceptions to instructions are given to timing tool, it

doesn’t care about the nature and frequency of the launch and capture clocks, it will stick to the worst-case behavior to perform the timing checks.

Many times, the tool might perform wrong checks, as it might violation design intent while performing the worst case check. We will look at such case in subsequent questions.

Question S23): Are clock domain crossing issues detected by STA tool ?
Answer S23):
No clock Domain crossing issues are not detected by Static Timing Analysis tool. As mentioned earlier, tool simply tries to find out the worst case setup and hold checks between launch and capture edge. Designer has to design for clock domain crossings.

Question S24): How does lockup latch help with avoiding hold violations.
Answer S24):

If you understand hold time check very well, or if you have been analyzing the waveforms for hold time check, you will realize that hold time issues start happening as soon as launch and capture clock edge align with each other or are very close to each other.

We know that more spread apart launch and capture edge are in such a way that launch edge is later than the capture edge, less of a hold time concern there is.

We know that when launch and capture clock are from the same source and have same waveform, the greatest distance between an edge in launch clock and an edge in capture clock can not be greater than clock phase. Because if try to do that you will approach one of the edge closer on the other side.

If the falling edge of clock is the launch edge and rising edge of clock is capture edge, we know that launch and capture edge would be a phase apart and as long as launch edge happens after capture edge, we would have a phase worth of margin for hold check. This is true for the case where falling clock edge is capture edge and rising edge is launch edge. The key is that they are a clock phase apart and launch happens later than capture.

This is what exactly a lock up latch achieves. It changes the launch edge from rising to falling edge and capture edge remains rising. So we get launch and capture edges to be farthest apart(clock phase) giving us best possible hold time protection. Also launch happens later than capture, which is what we want. Lets take a look at the figure below to better understand this.

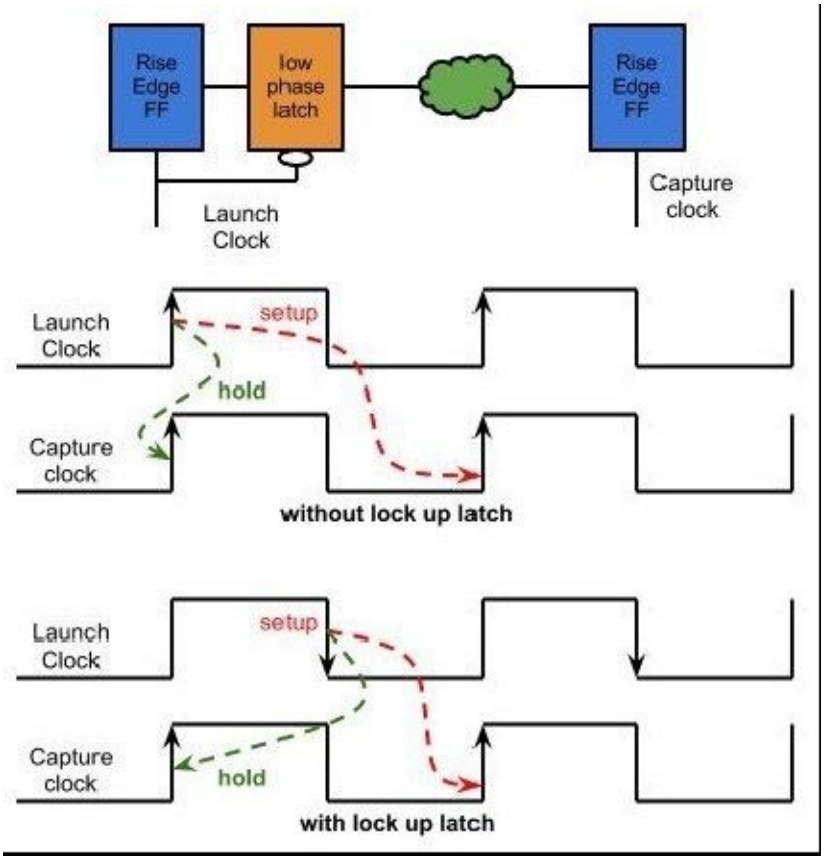


Figure S24a. Lock up latch.

Here we are assuming launch and capture flops to be rising edge triggered. As shown in figure before lock up latch, it's a simple setup and hold check. The issue is this type of hold check (also called race as launch and capture edges are the same, it is like a data race), it could be very difficult and expensive to fix this type of hold violations, if launch and captured clock common points are far apart, there could be quite a large clock uncertainty. This is very typical for scan or test clocks where last flop in one scan chain is in a specific clock domain and first cell of next scan chain is in a different clock domain. There could be large hold violations for such paths.

Low phase latch, launches data at the falling edge of the clock and remains transparent during low phase. Essentially by introducing the lockup latch, we moved launch edge from rising to falling, and now our launch and capture edges are a clock phase apart, we have a clock phase worth of margin(slack) to meet the hold time requirement.

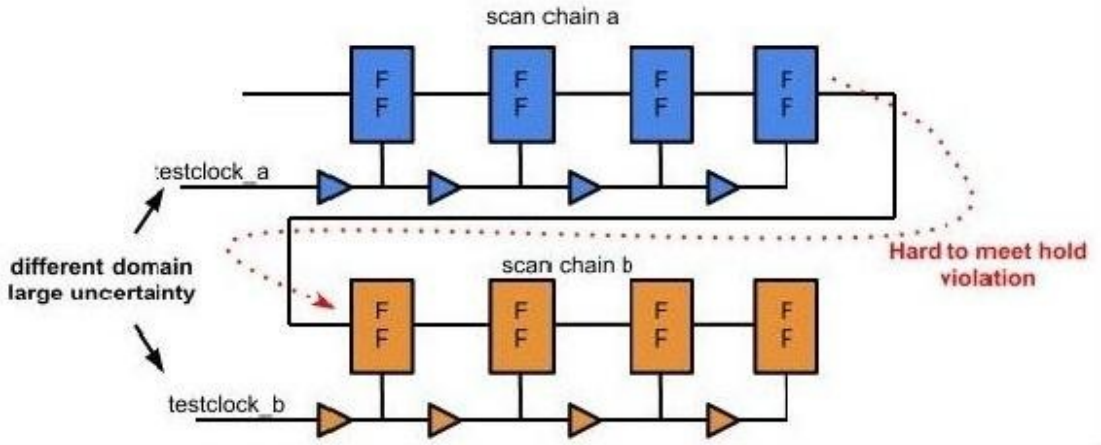


Figure S24b Test clock hold violation.

As shown in figure there can be a large uncertainty between testclock_a and testclock_b. If you recall from the hold margin equation, larger the clock uncertainty larger the negative slack that will have to be fixed.

In such situations the lockup latch is introduced between the two chains to address the hold violation.

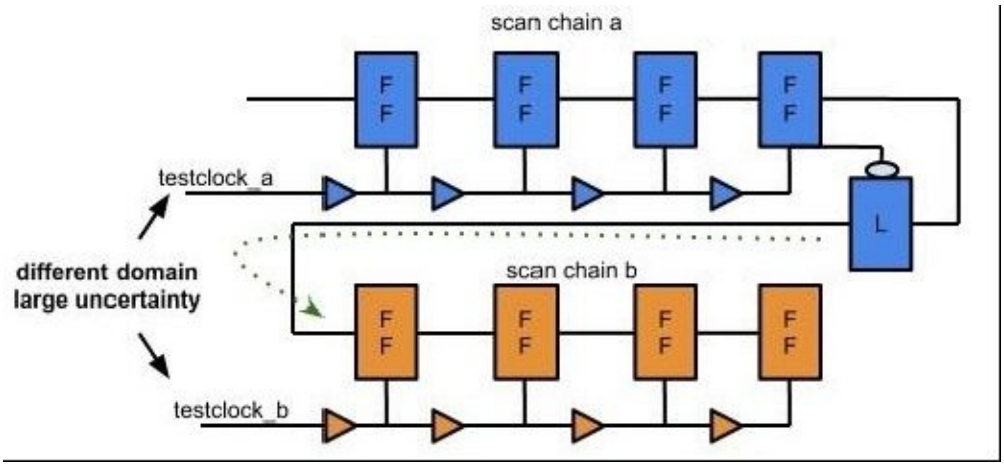


Figure S24c. Inter scan chain lockup latch

One has to realize that lockup latch doesn't come completely free. Because it changes the launch edge from rising to falling, we are modifying our setup or max timing path from the original launch flop to capture flop from a full clock cycle to half clock cycle(clock phase). Normally you would think of adding lockup latch only if you had hold issues to begin with which means, there was not a setup problem to begin with. Because if you had hold problems that means there wasn't much path delay from the launch flop to capture flop.

Question S25): Does location of lockup latch matter ? What if in previous example you moved lockup latch from near launch flop to capture flop ?
Answer S25):
The location of lockup latch very much matters. When you introduce lockup latch in

between two flops, you are essentially breaking timing path into two segments. One path from the original launch flop to the lockup latch and other timing path from the lockup latch to the original capture flop.

There is a reason why we didn't bother about the timing path from the launch flop to the lockup latch. Original launch flop launches data at rising edge of the clock and low phase lockup latch captures data at the rising edge of the clock as well. This could be a hold time issues, but it really is not because we clocked the lockup latch with the same clock that was clocking the launch flop. In Fact it is essential we do this and place low phase lockup latch right next to the launch flop. Doing so will ensure that there is no hold time issue from the launch flop to the lockup latch, as essentially it is the same clock net that is driving both, hence there can not be a data race from the launch flop to the lockup latch.

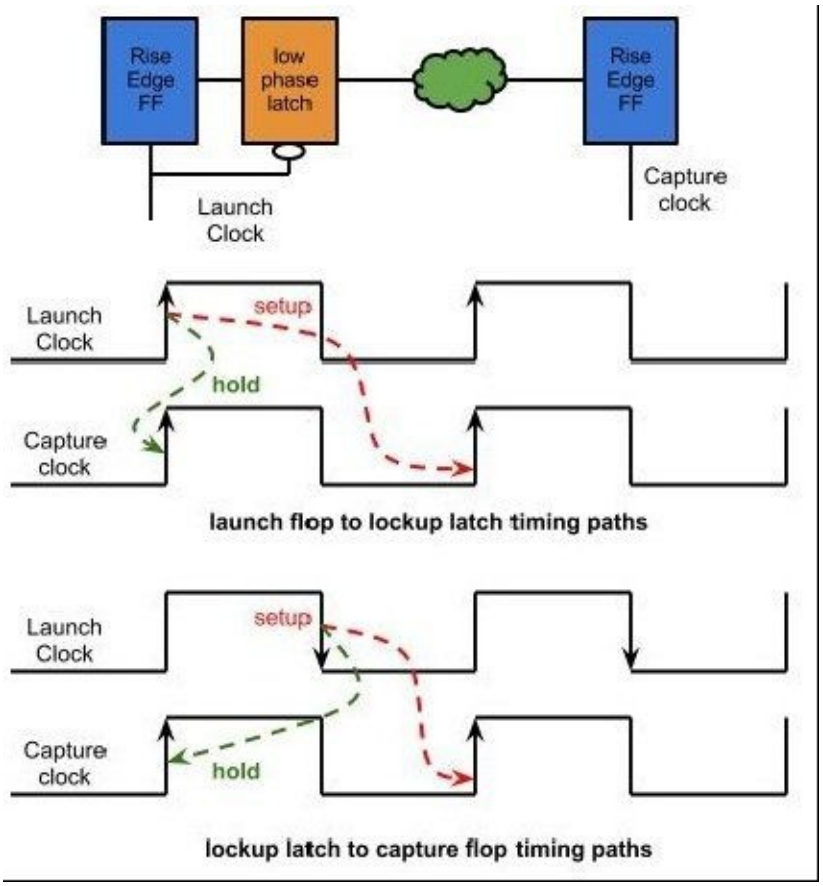


Figure S25a. Timing path split after lockup latch

As shown in this figure, there is a hold check that is supposed to happen from the launch flop to the lockup latch, but it really is not an issue because of the same clock edge first launching data and then capturing the data. Many timing tools understand this configuration and might not report this hold check, and even if timing tool reports this hold check, it should pass.

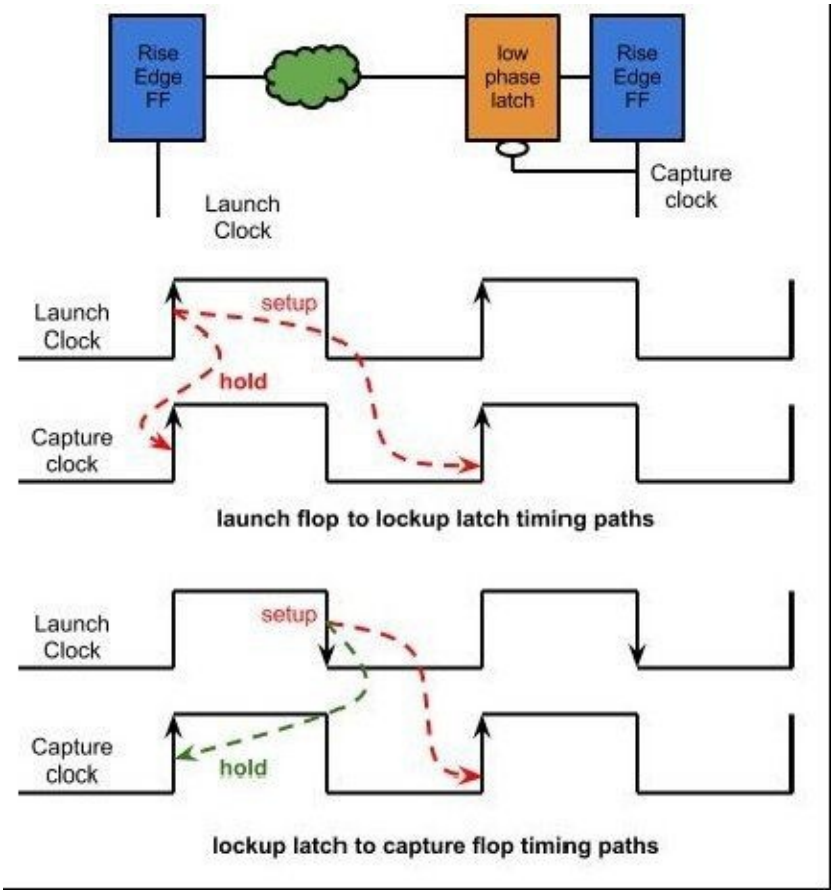


Figure S25b. Wrong lockup latch location.

You can see that once the lockup latch is moved close to capture flop, the hold violation from the launch flop to the lockup latch becomes the real issue as both clocks are now different and could come from different domain as we saw in test clock example and lockup latch is really not serving any purpose to fix the hold violation. Hence it is vital to place the lockup latch at correct location with correct clock.

Question S26): What are your options to fix a timing path ?
Answer S26):

There are several different possibilities for fixing a timing path.
- Obvious logic optimization.
Do you have redundant chain of buffers or inverters ? Can you drive with fewer buffer or inverters ? If there is a NAND followed by latch, do you have a library NAND-latch available t replace ?

- Better placement.
Can you move logic around to fix the path ? Is subgroup of logic along the path, just too far away from launch and capture flop ? In that case can you just move that logic closer to the launch and capture flop ? Can you move launch flop close to the capture flop along with the logic in between ? Or vice versa ?

- More pipelining.

Can you introduce new flop on the failing path ? Does architectural performance allow for that ? Basically by introducing extra flop, you are introducing one extra clock cycle along the timing path and hurting overall throughput of the logic.

- Move logic to previous pipe stage ?
Can you move some logic from current failing path to before the launch flop ? You have to make sure you don't break functionality and your formal equivalence with RTL has to still pass. For example if there is a NAND gate right after launch flop, one can investigate if the other input of NAND gate which is not coming from launch flop in question, does that other input have a previous clock cycle version available ? If that is available the NAND gate can be moved before flop.

- Replicate drivers.
If a specific stage is too slow and the reason for the stage is too much load with multiple receivers, replicate driver and split number of receiving gates among the replicated drivers. If a single buffer was driving 8 receivers, replicate buffer and have each of them drive 4 receivers.

- Parallelism in RTL
Search for opportunities in RTL where by you can change serial operations into parallel. Serial operations take more time as there are more stages within a clock cycle. If we can split a large serial operation into multiple smaller length parallel operations we can easily meet timing on each of the individual operation.

- Use of Macro.
Is there synthesized logic, which is actually a memory ? If that is the case RAMs are much faster than the synthesized flops. Map the logic to SRAM or register file.

- Synthesized if...elseif...elseif series.
If random logic has been synthesized for such if...elseif series, the logic can be mapped to passgate 4:1 mux, given that such library cell is available. Most of the time such mux is going to be faster than the static gates.

- One Hot instead of Binary coded State Registers
If possible convert binary coded state registers to one hot registers. In one hot configuration only one of them is going to switch at a given time and overall operation will be faster,

- Physical design techniques.
Can you promote metal wires to higher metal ? Or can the wire be widened ? In both cases the driver will see more capacitance, so driver strength will need to be increased. Can the spacing between wire be increased ? This will reduce capacitance and speed up wire delays.

- Power trade off techniques.
Switch to low threshold voltage library cells. These cells have lower threshold voltage,

higher gate leakage and faster speed. You will increase speed at the expense of leakage power, you will have to be within an overall budget for the chip for usage of such devices.
You can use time borrowing capture flip flops. Such flops have clock delayed by certain stages. What this does is the pushed out the capture clock which helps meet setup time as capture edge happens later. But more clock buffers along the clock path means more clock toggling and more active power, more devices means more leakage and more variation.

Question S26): By default design compiler(DC) tries to optimise the path with worst violation. Is there anything can be done to make it work on more paths than just worse ?
Answer S26):
You can use group_path command to achieve this. One can specify 'critical_range' on that group to have DC focus on a slack range.

Timing Exceptions(Overrides).

Question TE1)What are multi cycle paths ?
Answer TE1):
By default timing paths are single cycle long. Here is what it really means. In digital circuits, memory elements like flip flops or latches, launch new data at the beginning of the clock cycle. During the clock cycle, the actual computation is performed through the combinational logic and at the end of the clock cycle data is ready and is captured by the next memory element at the rising edge of the next clock cycle, which is the same as ending of the current clock cycle. Following figure illustrates this.

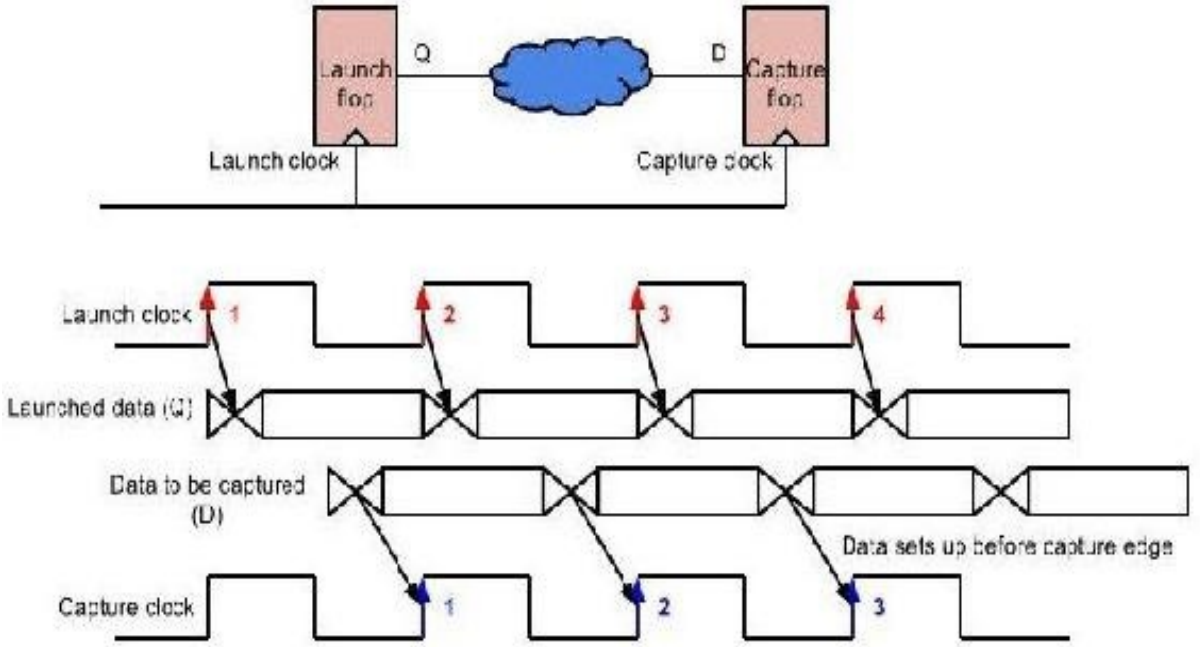


Figure TE1a. Single cycle (default) timing path

As shown in the figure, the launching flop keeps generating new set of data at the output pin Q of the launch flop with every rising edge of the clock cycle. Similarly capture flop keeps sampling input data every rising edge of the clock cycle. As you can see in the figure the data launched on rising edge ‘1’ (in red) is supposed to be captured by capture edge ‘1’ (in blue). Similarly capture edge ‘2’ corresponds to launch edge ‘2’ and so on.
This is called a single cycle timing path. There is one clock cycle from the launch of the data to the capture of the data. By default, timing tools assume this to be the circuit behavior. Timing tools will perform a setup check with respect to a capture clock edge, which is one clock cycle after the launch clock edge.

But this may not be the case every time. Many times what happens is that the combinational delay from the launch flop to the capture flop is more than one clock cycle. In such cases, one can not keep launching data at the beginning of the every

clock cycle and hope to capture correct data at the end of every clock cycle. In such cases data launched at the beginning of a clock cycle will just not reach the capture edge at the end of clock cycle.

When this is the case, the circuit designer has to account for this fact and design of the circuit. If the combinational delay from launch flop to the capture flop is more than one clock cycle, but less than two clock cycles, the circuit designer has to design the circuit in such a way that data is not launched from the launch flop at every clock cycle, but is launched at every other clock cycle. And the data launched at the beginning of a clock cycle is captured not after one clock cycle, but two clock cycles. Following figure depicts this.

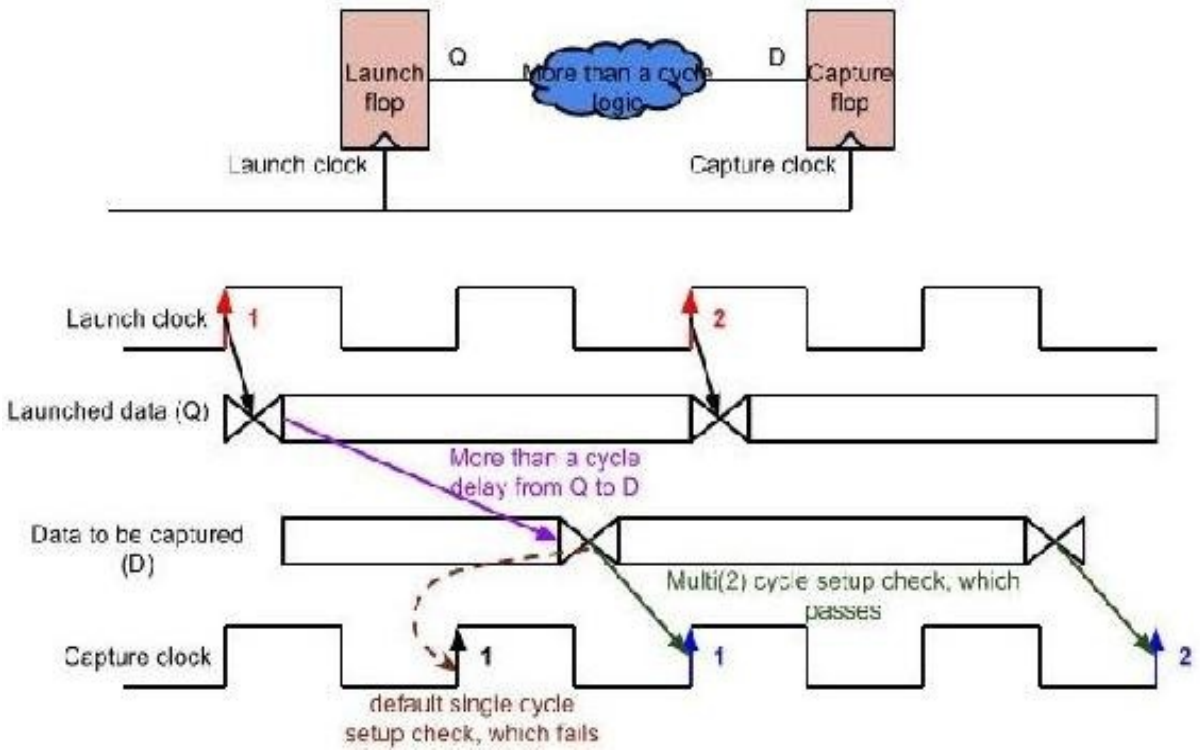


Figure TE1b. Multi(2) cycle timing path.

As shown in the figure, let's say that we have a circuit where we know that combinational delay from launch flop to the capture flop is more than one clock cycle but quite a bit less than two clock cycles, such that it can meet setup time requirements comfortably in two clock cycles, but it doesn’t meet setup in one clock cycle.

You can see in the figure the data launched at the launch clock ‘1’, approximately arrives at the capture flop (Data to be captured (D)) after about one and half clock cycles. As it was mentioned earlier, by default timing tools think that all timing paths are one clock cycle long. In other words, if the data was launched at launch clock ‘1’, the timing tool will think that it needs to be captured at the capture edge which is one clock cycle after the launch edge, which is the capture edge shown in the figure with black rising arrow.

Timing tool by default will check setup with respect the capture edge shown in figure with black rising arrow and will report that input data to the capture flop (Data to be captured (D)), fails the setup to the capture flop as it arrives later than the capture edge. This check is shown in figure with dotted line. In reality we know that this is false setup check. The capture flop input setup check should be against the capture edge shown by the blue color. As stated earlier, our design here is such that we expect data to be take two clock cycles to travel from launch flop to capture flop and we have designed our circuit such that launch flop doesn't launch new data every clock cycle, but it launches every other clock cycle as shown by the red color launch edges.

In such scenario, we need to provide the timing tool with an exception or an override and we need to tell the timing tool that, it needs to postpone its default setup check by one clock cycle. In other words, we need to ask timing tool to give the one additional clock cycle time for the setup check.

Usually this is achieved by something like following.

set_multi_cycle 2 -from <start_point> -to <end_point>

Where '2' is the clock cycle count. It instructs timing tool to use 2 clock cycles and not just default '1' for the cases where we want timing tool to use 2 clock cycles.

Question TE2). What are false paths
Answer TE2):

Static timing analysis is exhaustive by nature. Timing tool will exhaustively look at all possible timing paths and will perform timing checks. Because of this, it will also perform timing checks on timing paths which can not really happen. Best way to understand this is by examples.

Consider the circuit described in the image below.

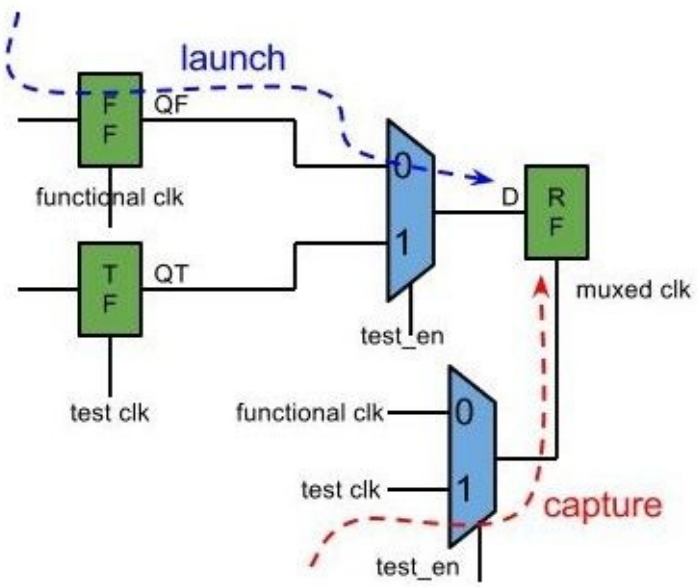


Figure TE2a. False timing path.

This type of circuit configuration is very common in digital circuits. Functional clock is active only in functional mode and test clock is active only test mode. This means when a timing path starts with functional clock launching data at functional flop(FF) output QF, it should be captured by receiving flop(RF) and capture clock should only be functional clock.

Because of the exhaustive nature of timing tools, it will also time a path where functional clock launches data at QF output of function flop(FF) and is captured at D input of the receiving flop(RF) through test_clk. Given that functional clock and test clock are not active at the same time, this timing path is false and can never happen. When functional clock launches data at QF output of functional flop(FF) and it captured at D input of receiving flop(RF), it can only be sampled through functional clock and not test clock, as only functional clock will be active at that time.

To drive this point further, take a look at the following circuit.

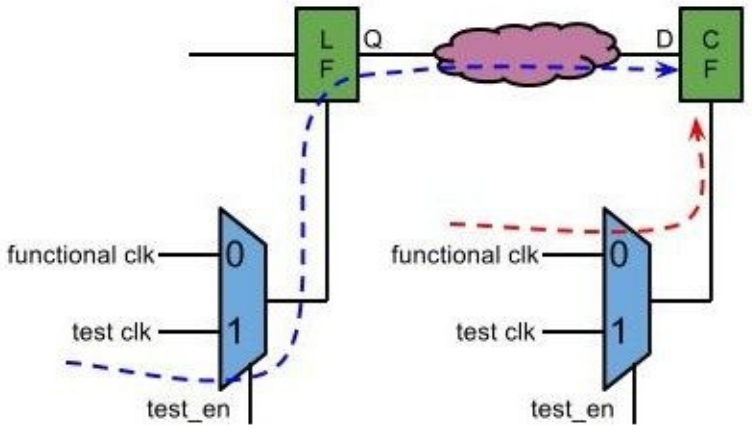


Figure TE2b. One more false path.

In the above figure a mux is used to select between functional clock and test clock. In functional mode only functional clock is active and test clock in inactive. In test mode only test clock active and functional clock is turned off.

For the above circuit there are only two valid timing paths.

First timing path is where functional clock launches the data at Q output of launch flop (LF) and this data is captured again by functional clock at capture flop(CF) input D.

Second timing path is similar but with test clock, i.e. where test clock launches the data at Q output of launch flop(LF) and this data is captured by test clock at capture flop(CF) input D.

But because of the exhaustive nature of the static timing analysis, timing tool by default come up with four timing paths.

1) Functional clock launch => Functional clock capture.
2) Functional clock launch => Test clock capture.
3) Test clock launch => Test clock capture.
4) Test clock launch => Functional clock capture.

As you can see only paths 1) and 3) are valid and paths 2) and 4) are false. An explicit exception or override needs to be provided to the timing tool to address this false paths.

Question TE3): What happens if a multicycle exception is provided only for setup or max time in PrimeTime ?
Answer TE3):
Most of the time it is not enough to just provide multicycle exception for max only in PrimeTime. Because as discussed earlier, hold timing check is with respect to setup check and when you provide multicycle exception for setup only it changes the default behavior of the setup check and because hold check is dependent upon setup check, default hold check behavior is also changed. So we also have to provide multicycle exception for hold time, but it is more of a correcting behavior than anything else. Following diagram should clarify this.

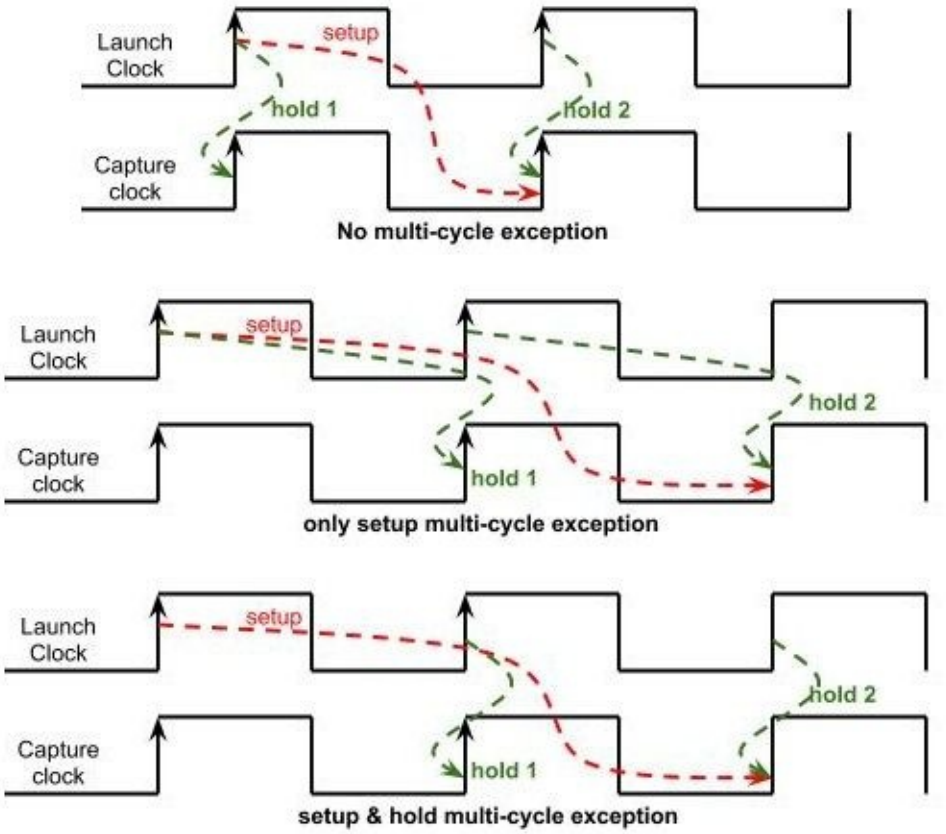


Figure TE3. Multicycle setup only exception problem.

As you can see in figure, first waveform is without any exceptions, which we have analyzed before. Second waveform is with setup only multicycle exception. When a multicycle exception with -setup option is provided in PrimeTime like tools, what happens is that the capture edge for the setup check is pushed out by the amount of extra cycles specified in the exception. In the above figure, one extra cycle is specified so capture edge is pushed out by one cycle.

Actual multicycle exception would have looked like following :
set_multicycle_exception 1 -setup
Here '1' is the number of extra cycles we want to allow, in our case it is one additional cycle. Remember this number is in additional to default '1' cycle. Setting multicycle exception by '1' additional cycle gives total two clock cycles for setup check.

Now we know that hold checks are performed with respect to the setup check. Based on the new exception based setup check, new hold check options are derived as you can see by green dotted lines. You can see that both hold checks are violating by about a cycle. Is this violation real ? Not typically. In our design we still have back to back rising edge triggered flops and most likely the cell and interconnect delay between them is large enough to warrant a multicycle exception. But we still want the hold check to reflect the normal timing check where we want to ensure that launched data does not rush through to the same cycle capture edge, as launched data is meant to be captured at the end of clock cycle. This means, even with the setup multicycle exception we want hold check to look like just regular hold check without any

exception, just like first set of waveforms.

Obviously what tool inferred to be the hold check with multicycle setup exception is wrong. If you look at the wrong hold checks, you can see that we really need the launch edges for the hold check to be pushed out by one cycle. Which is what a multicycle exception with -hold option does. Hence we conclude that whenever we use multicycle exception with -setup option, we need to add multicycle exception with -hold option with equal number of extra cycles.

Last set of waveforms confirm this, we can see that once both -setup and -hold exceptions are in place we get correct setup and hold timing checks.

Signal Integrity.

Question SI1): What is signal integrity ?
Answer SI1):
In integrated circuits, one of the phenomenon that designers have to be aware of is, cross-coupling effect or cross-talk on wires. Because of this effect, your signal can lose its integrity and you might end up capturing bad signal data.

On integrated circuits, wires are routed next to each other with insulating material in between them. This forms the capacitors and switching behavior of one wire affects other wires. If signal is rising, or going from low value to high value on one wire, it couple in the rising direction with neighboring wires and pushes neighboring wire signal values to bump high, similarly one wires can couple high to low with other wires. Because of this coupling effect, signal can lose its integrity and may lose its value.

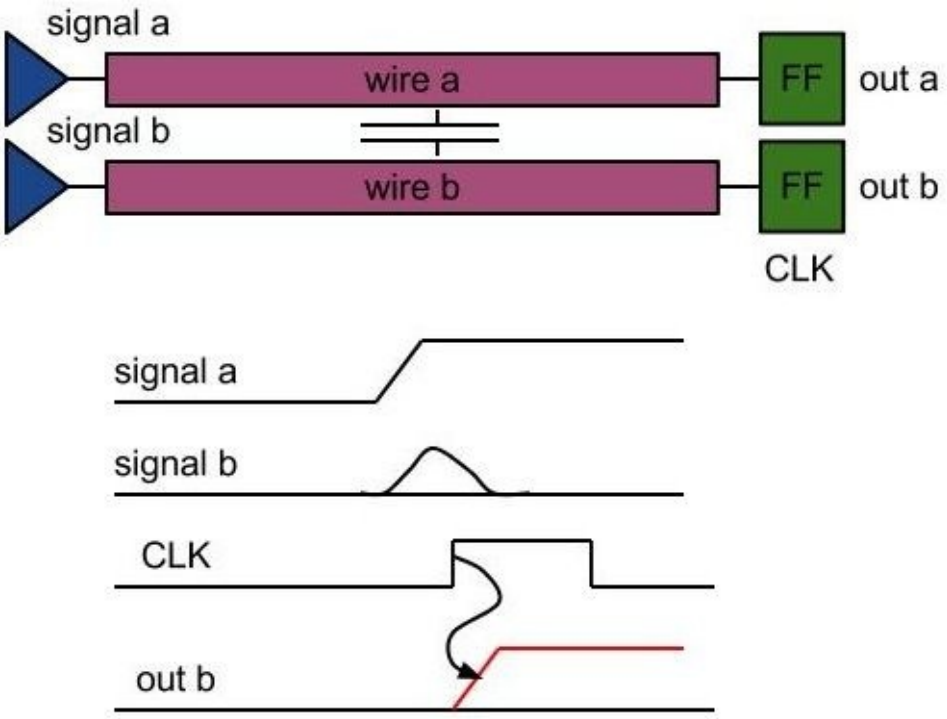


Figure SI1. Wire to wire coupling.

As shown in the figure above wire ‘a’ and wire ‘b’ are routed next to each other and they carry signal ‘a’ and ‘b’ respectively. Lets say signal b is supposed to remain low throughout the time period that we are observing. As shown in figure, if signal ‘a’ switches and goes low to high, it is going to couple with wire ‘b’. Because of this coupling effect, signal ‘b’, which is normally at low level, is going to start rising.

Although signal ‘b’ is not going continue rising for a long time. There are two reasons for that. Signal ‘a’ coupling with signal ‘b’ is a transient event, signal ‘a’ is only going to couple with signal ‘b’ while it is rising, once signal ‘a’ is done rising, the effect is

gone and there will not be further effect to push signal ‘b’ to rise any more. Second reason is that when signal ‘b’ is low, it mean the at the driver of signal ‘b’, the nmos device is on and as soon as new charge on wire ‘b’ appears because of the coupling from neighbor, nmos device will discharge wire ‘b’ and pull it down to low level.

As you can see that if, the capture edge of the clock for the signal ‘b’, happens to arrive very close to the coupling glitch on signal ‘b’, the capture flip-flop will capture a wrong value for signal ‘b’. Capture flop should have really captured, low value for signal ‘b’, but as you can see it captures high value(shown in red) for signal ‘b’ and signal ‘b’ has lost its integrity now.

The height and duration of the coupling glitch observed on signal ‘b’ will depend upon several factors, including following ones.
- Capacitance between two wires.
- Strength of the driver for signal ‘a’ and the slope or slew rate for signal ‘a’.
- Strength of the driver for signal ‘b’.

Mind well, that this is only one of the event that causes the signal integrity to be lost. There are several other factors that also can cause the signal integrity to be lost, especially power droop and propagated glitch.

Question SI2): How to fix crosstalk glitch ?
Answer SI2):
Layout tools will take crosstalk analysis into account when you start routing. You can tell your physical layout tool to leave adjacent routes of high frequency nets, like clock trees, to be empty. Once you have a routed database, you can get some sort of cross talk reports and then fix it in the layout tool. Two of the common methods I can think off would be to add stronger buffers to the victim nets (of course that can affect your timing) or you could route the nets in a jogged formation so that the adjacent area between the victim and the aggressor is comparatively less.

Question SI3): How is signal integrity(SI) in related to Timing? How do you improve timing from SI effects?
Answer SI3)
With shrinking technologies, cross coupled capacitance between metal layers increases. As a result, when a signal is actively switching, it can cause "cross talk glitch" or a "cross talk delay" on a neighboring signal(victim net).

Assume you have a net that is being driven by a strong driver and it is next to another net (victim) which is being driven by a relatively weaker driver. When the victim is going from high to low and at the same time, if aggressor goes from low to high, the aggressor will cause the victim switch from high to low a little later. Think of it as if the aggressor prevented it from going from high to low right away by pulling it in the same direction as its own. Because of this delay on the victim net going from high to low increases, and further down the logic cone, this can cause failures. This is called “crosstalk delta delay”.

In another case, if the victim is low and if strong aggressor switches from low to high. This will pull the victim net state from low to high for a short time, enough to cause a glitch on the victim net, which is referred as “crosstalk glitch”. This glitch could be captured by a flop downstream and can cause state machine corruption.

In both cases of crosstalk delay and glitch the amount of delay or the height of the glitch depends upon the amount of cross coupling capacitance, strength of the aggressor and the strength of the victim. Closer the aggressor and victim are more cross coupling capacitance will be. Stronger the aggressor, more coupling effect will be. Strong the victim driver, less of the coupling effect it will observe as it will be able to restore the original logic level on the victim net sooner and will be able to recover from glitch faster.

One can increase spacing to reduce cross coupling. One can reduce the aggressor driver strength or one can increase victim drive strength to minimize the cross coupling timing slow down or the cross talk glitch.

Variation.

Question V1): What is on chip variation ?
Answer V1):
For timing sign-off, digital designers typically simulate circuits at extreme process corners. That analysis typically assumes that gate and interconnect performance at any given corner match across the chip or die. It is assumed that all cells or all interconnect across whole chip to be at the given, worst-case or best-case corner. Unfortunately, that assumption is no longer valid. Because in reality that is not the case. Given the complexities and intricate nature of deep-submicron processes, the variation between devices and interconnect characteristics on the same die, can no longer be ignored.

A device with specific size is expected to run at certain specific speed at a certain process corner on the chip. We want all devices with same size across the chip to run at the same speed. In reality that is not the case. Because in reality, different part of chip gets manufactured with some variation. Actual devices shapes come out to be different in different parts of the chip.

Manufacturers' libraries typically characterize these layers at the absolute worst-, typical-, and best-case corners. The final die most likely is not going to have have both worst-case-speed and best-case-speed transistors on it, but it will have significant variation in both effective channel length and width of transistors.

Different devices intended to be same width(size) ends up having slightly different width compared to original intention, purely because of some of the limitations in manufacturing of such devices.

Following are some of the variation across chip/die that can directly affect timing.
- Threshold voltage variation (Vth)
- Channel length variation (Le)
- Transistor width variation.
- Interconnect variation.
- IR drop variation.
- Temperature variation.

Some of the major sources of this variation are.
- The CMP (chemical-mechanical-planarization) process.
- The proximity effects in the photolithography.
- The proximity effects in etch processes.

The CMP (chemical-mechanical-planarization) process
There is a difference in hardness between the interconnect material and the dielectric. after the designer has etched trenches into the dielectric below an interconnect layer and copper on the wafer, the CMP process removes the unwanted copper, leaving only wire lines and vias. The copper line is softer than the dielectric material, resulting in

‘dishing’ and erosion, which cause uneven removal of the copper and dielectric. Dishing is a function of line width and density, and erosion is a function of line space and density

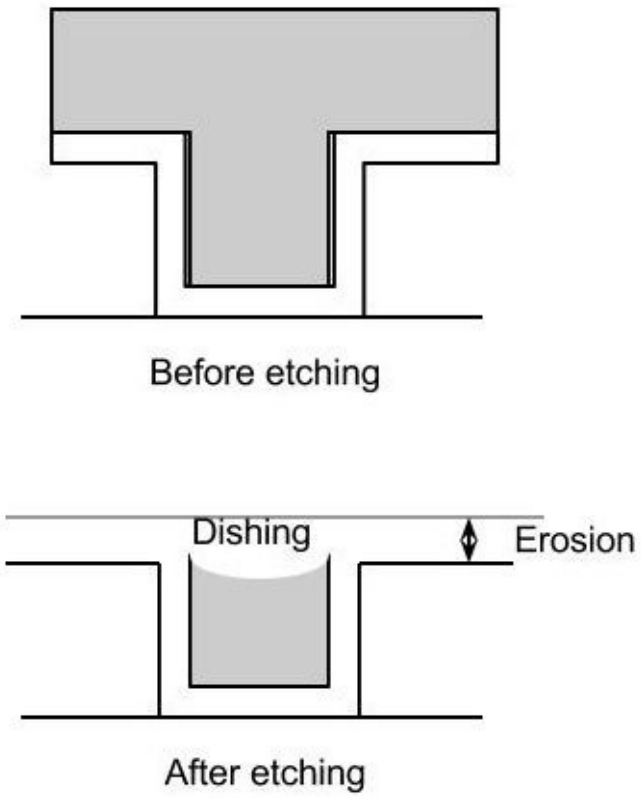


Figure V1a. Dishing and Erosion.

CMP also causes a more random variation in interconnect thickness, resulting in a gradient across the wafer. This gradient is visible in die-to-die variations and even across-die variations for large die.

The proximity effects in the photolithography

Electron beam lithography provides needed high resolution patterning for sub-micron technologies. However, the effect of electron scattering in resist and substrate leads to an undesired influence in the regions adjacent to those exposed by the electron beam. This effect is called the proximity effect. Result is unintended effect on neighboring circuits. The effects are pattern and density dependent and vary across die, resulting in variation in device and interconnect characteristics across the chip.

The proximity effects in etch processes.

The etch rate of silicon, during reactive ion etching (RIE), depends on the total exposed area. This is called the loading effect. However, local variations in the pattern density will, in a similar way, cause local variations in the etch rate. This effect is caused by a local depletion of reactive species and is called the microloading effect.

Basically etch rate is dependent upon pattern density and as pattern density varies across chip/die, the etch rate varies too and gives rise to variation. Isolated wires are

over etched. Over-etched trenches are wider and results in wider wires for isolated wires.

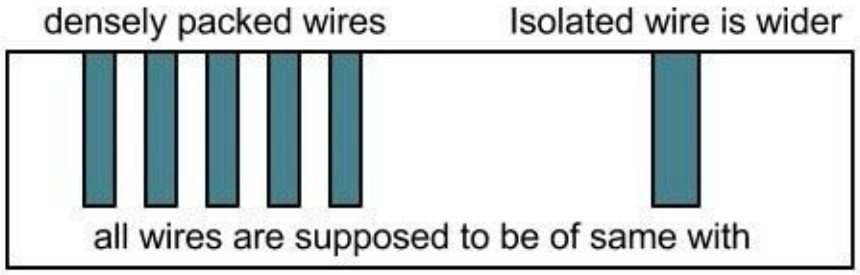


Figure V1b. Isolated wires turn out wider.

Components of variation :
One can describe on-chip variation having a random and a systematic component. The random component of critical parameter variation occurs from lot to lot, wafer to wafer, and die to die. Examples are variations in gate-oxide thickness, implant doses, and metal or dielectric thickness. The systematic component comprises variations that you can predict from their location on the wafer or the nature of surrounding patterns. These variations relate to proximity effects, density effects, and the relative distance of devices. Examples are variations in gate channel length or width and interconnect height and width

Delay variation and timing impact :
Variations cause changes in both the width of a transistor's active area and the device's effective channel length

One of the issue that complicates timing analysis is RC-interconnect variation. Variations in critical dimensions of minimum-width lines can occur during mask generation and in the process stages for active and polysilicon layers. Variations in the width or height of interconnecting metal lines can cause large RC variation along longer nets. Variation in interconnect height and width, results in variation in both resistance and capacitance of wires.

Interconnect delay in deep-submicron processes can often dominate total delay as geometries shrink. Because of this, one should model interconnect variation as accurately as possible.

Metal line resistive variation will cause IR drop, which directly affects the standard cell delays. Different parts of chip ends up having different voltages, even if the original intent was to have same voltage across the chip. This is because different parts of chip will experience different IR drop because different resistive variations along different paths along with potentially different temperature.

Temperature variations can also cause differences in electrical behavior and, hence affect timing. Fortunately, it is not common to find extreme temperature corners on the same die during operation. But nonuniform on-chip power distribution, interconnect

heating, and thermal characteristics of the die and package materials can influence actual operating temperatures.

Some of the ways to minimize variation
One can minimize variation by choosing cells that are wider than minimum. By restricting the clock-tree cells to high-drive cells, you ensure smallest variation and thus smallest mismatch across the various subtrees.

Clocks.

Question C1): What are the main clock distribution styles used ?
Answer C1):
In digital designs there are two main clock distribution styles that are used.
1) Clock mesh or Clock grid distribution system.
2) CTS(Clock tree synthesis), or Clock tree distribution system.

Question C2): What is Clock mesh or clock grid distribution system ?
Answer C2):
The objective of the clock mesh or clock grid distribution system is to provide a fixed amount of delay from the source of the clock (PLL) to the all end receivers of the clock, which are flops, latches, macros and clock-gaters. In other words, it aims to minimize clock skew across all the final receivers of the clock. The clock distribution scheme, tries to achieve same overall delay, by balancing number of stages from the source to all distribution and by having same stage delay for all stages in such distribution.

This distribution also aims to limit the number of stages to smallest possible number. This number would vary depending mainly on the size of the die, type of process technology etc.

Such a distribution can usually be thought of as a two step distribution. The reason is that most of the time a chip is divided into blocks. The PLL lies within one of the blocks and there are two levels of distribution. One from the PLL to all blocks boundaries within chip, another distribution from the block boundary to inside of the block.

Following figure shows the first level of distribution.

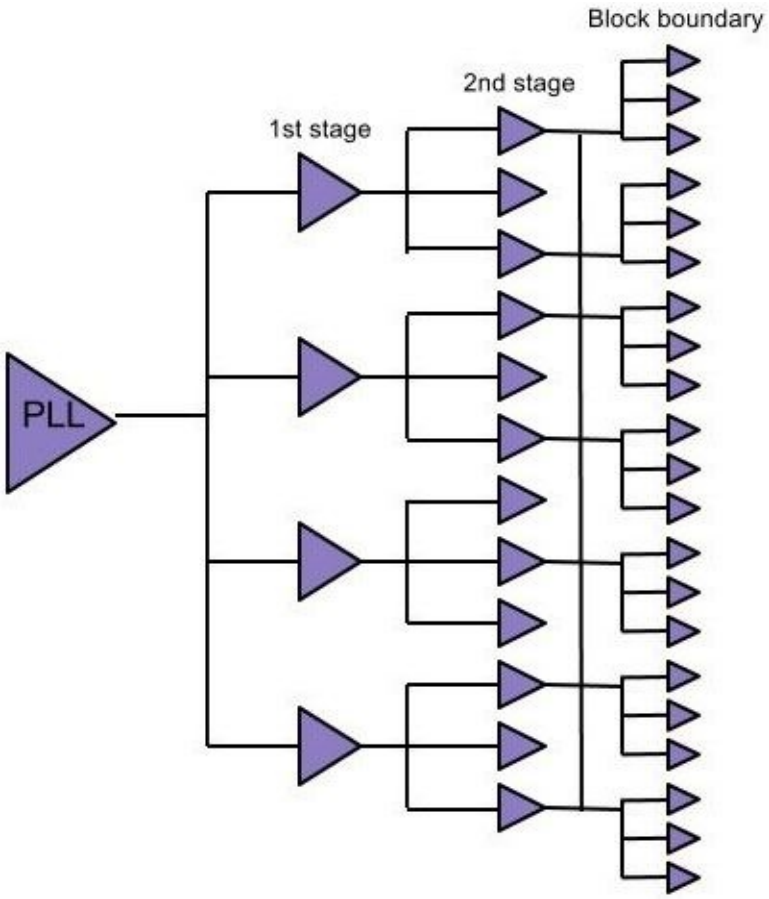


Figure C2a. Clock distribution from PLL to blocks.

The second level of distribution happens inside block. Here is where, the clock distribution through fixed number of stages is achieved through a grid or a mesh of clock buffers. If clock is to be distributed within a region, a symmetric mesh or grid of final clock buffers is formed within that region. Idea is that no matter where the flop or final receiver is residing within that region, there should be local clock buffer in the vicinity of that receiver. This final clock buffer is driven by another buffer which is again part of a symmetric mesh or grid, which would be less dense as it would need to drive just the symmetrically placed grid of last stage drivers.

Below figure illustrates the idea of the clock mesh. Lets assume that figure represents

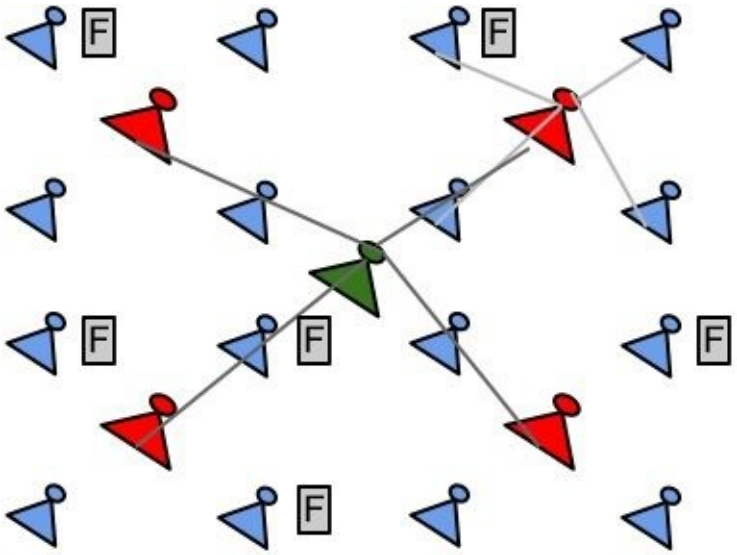


Figure C2b. Clock mesh distribution.

the area or the block where final clock receivers(flops) are scattered throughout the region, shown here with ‘F’. As you can see the placement of ‘F’ could very likely be random.

In this figure green buffer represents the starting point for the distribution within this region. Usually there are few stages of clock buffers driving from PLL to the green buffer stage, which is the first stage Green inverter drives all red inverters, which are symmetrically placed with respect to green drivers to balance delays. Red drivers in turn drive blue drivers, which again are symmetrically placed with respect to red drivers for balancing purpose. Finally blue drivers drive the nearby flops. The purpose of this scheme is to balance clock delay from green driver to all flop receivers.

As you can see that delay to all flop receivers will be very close, but it can not be identical, because of several factors, like device variation etc. Usually clock routes are shielded to minimize the coupling effects and variation stemming from coupling effects.

Question C3): What is clock tree distribution system ?
Answer C3):
Clock tree distribution system varies from clock mesh distribution at block level, as to how clock is distributed to all final clock receivers. The key difference is that, a clock buffer tree is build from the source(main) clock driver at the block input to all receivers. Such clock tree aims to have an optimal number of branch stages to the clock receiver.

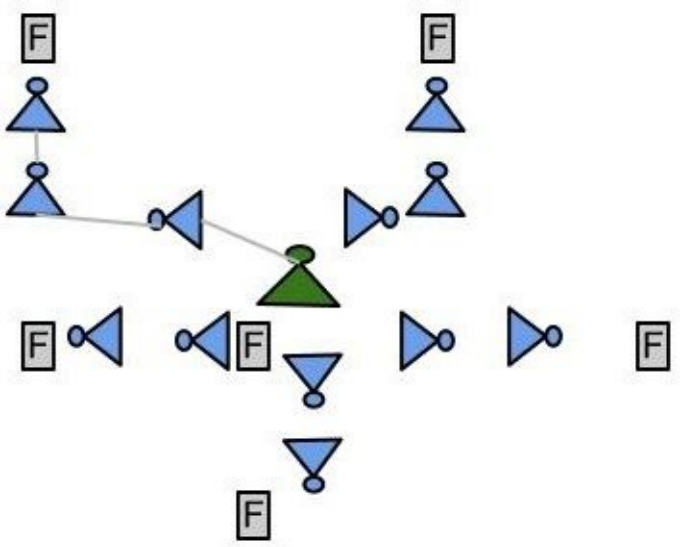


Figure C3. Clock tree distribution.

As you can see, there could be different number of clock inverter/buffer stages to different receivers. The idea is to not have a same number of stages to all receivers, but have an optimal number of stages along each branch. The reason for choosing this option is that, though clock mesh gives much better skew, it is at the expense of more power dissipation, because it needs more number of devices to achieve balancing and many times, the output of certain stages are shorted to minimize skew.

Usually clock tree distribution is used for relatively slower clocks in your design. It should be noted that clock tree distribution does not completely ignore clock skew, but it does not try hard to balance clock delays to the receivers.

Question C4): Explain CTS (Clock Tree Synthesis) flow.
Answer C4):
Many times, people get confused with clock tree distribution and clock tree synthesis. They are two different things. Clock tree synthesis is the design step to form the clock tree distribution. The goal of the CTS flow is to minimize the clock skew and the clock insertion delay. This is the flow where actual clock distribution tree is synthesized. Before CTS timing tools use ideal clock arrival times. After CTS real clock distribution tree is available so real clock arrival times are used.

Question C5) What is clock skew ?
Answer C5):
In synchronous circuit design, we want clock to arrive at the same time for all sequential receivers for the clock, as that would make it easier to do static timing analysis. We attempt to achieve this by designed a balanced clock distribution scheme. In reality we can not achieve exact same clock arrival time at all receivers, but clock arrives at different times at different clock receivers in the design. This phenomenon of clock arriving at different times at different places is called ‘clock skew’.

Mostly clock skew is unintentional, but it could be intentional as well.

One of the main reason for the clock skew is design limitations. We want clock to arrive at the same at all receivers, but they do not, because of several reasons like, device delay variation because of threshold voltage and channel length variation, on chip device variation, differing interconnect/wire delays, interconnect delay variation, temperature variation, capacitive coupling, varying receiver load, bad clock distribution tree design.

In industry clock skew is commonly referred as clock uncertainty.

Sometimes the clock skew is introduced intentionally in the design. Clock skew can help fix setup violations by delaying the capturing edge of the clock at the expense of more power dissipation. More details are available in the questions regarding how to fix setup violations.

Skew could help or hurt in your design. If in reality clock arrives later than expected at a sampling element, and if there is minimal data delay from previous sampling element, new data can race through from the previous sampling element and can get inadvertently captured at the sampling element where clock arrives late.

Or if there is enough of data delay from previous sampling element to the current element, the late arriving clock compared to data can help meet setup requirement at the sampling element. Following figure describes the false data capture because of clock being late and data being fast.

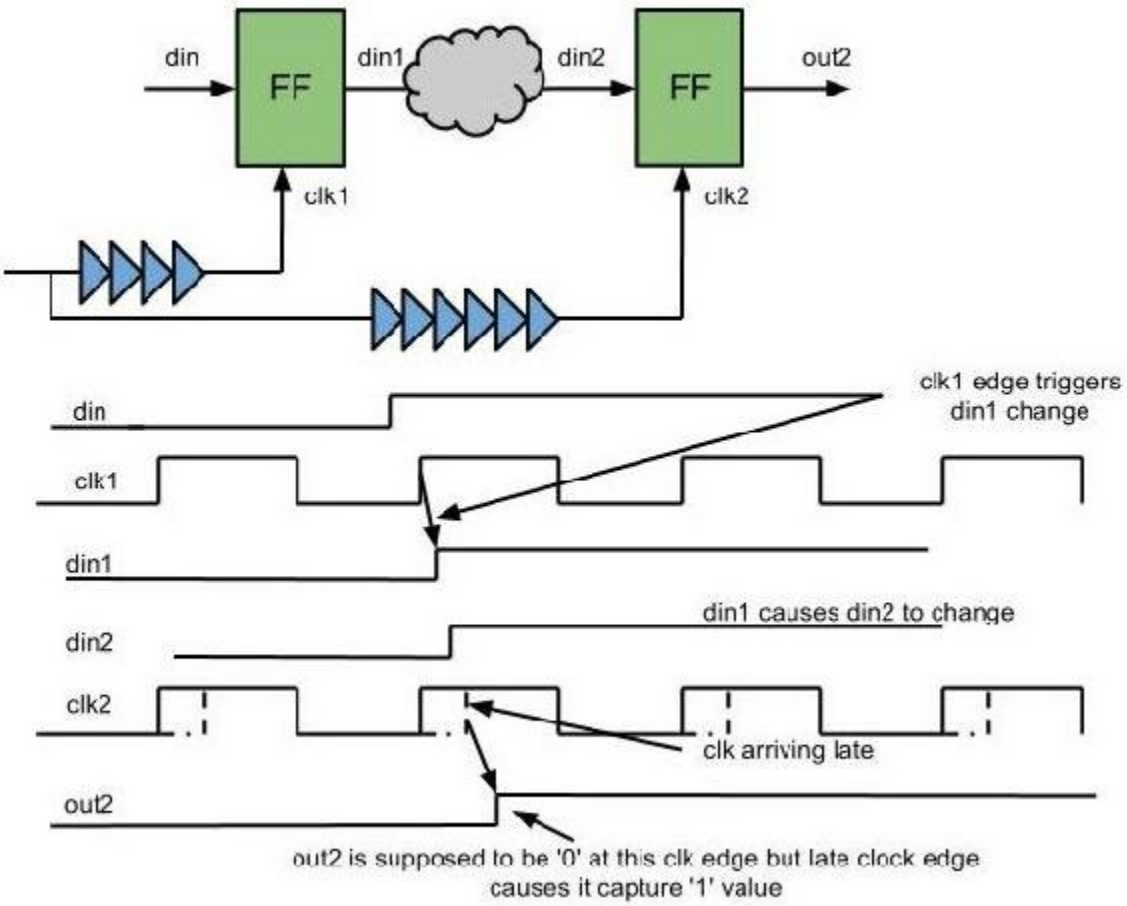


Figure C5. False data capture because of late clock (clock skew)

As you can see the input to the series of back to back flop is ‘din’. Input ‘din’ is initially low and right before the second rising clock edge arrives, it goes high. The first flop needs to capture the high value of ‘din’, which it does correctly as ‘din’ meets the setup time.

Soon after clk1 rises(second edge), ‘din1’ goes high as it follows ‘din’. There is certain delay from din1 to din2, which means ‘din2’ goes after after that delay, as it follows ‘din1’.

This high going edge of ‘din2’ is supposed to be captured by second flop, the third rising edge of clk2. That is how normal digital synchronous design with back to back flop is supposed to work. But what happens is second flop captures the rising value of ‘din2’ at the second rising edge of the clk2, which is wrong data to be captured by second flop.
Now the state at the output of second flop is wrong and subsequent calculations and the
state machine enters into an unknown state. This is the type of damage clock skew can do.

As mentioned previously we will talk about intentional clock skew to help with setup

failures in a separate question.

Question C6) What is clock-gating ?
Answer C6)
Clock gating is a power saving technique. In synchronous circuits a logic gate (AND) is added to the clock net, where other input of the AND gate can be used to turn off clock to certain receiving sequentials which are not active, thus saving power because of toggling clock.

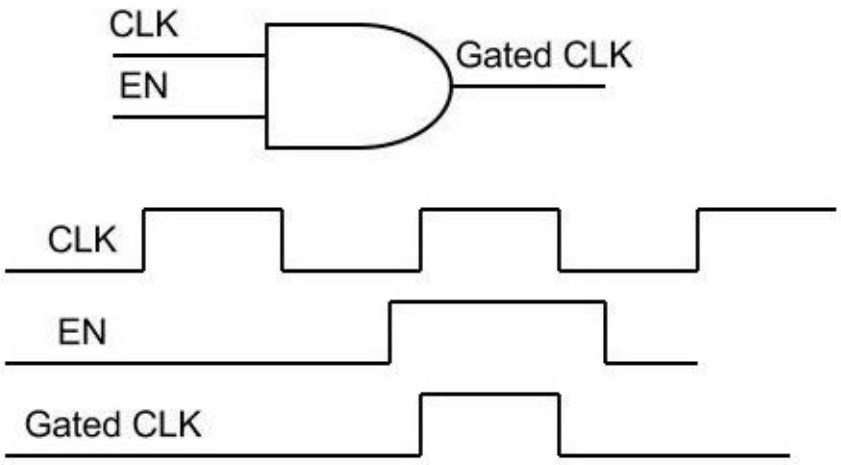


Figure C6 Gated Clock.

Question C7). Why is clock gating done ?
Answer C7)
As you can see in the figure by clock gating we can mask certain clock pulses, or in other words we can control the clock toggling activity. Clock is usually a very high fanout signal which is distributed throughout the chip. Because it usually drives large number of elements and normally continuously toggles, it account for major portion of dynamic power dissipation of the chip. Ability to turn off clock toggling when not needed is the most effective dynamic power saving mechanism.

From timing perspective, one has to ensure that clock gating doesn't introduce glitches or changes the shape of the clock pulse. There are setup and hold checks to ensure this.

Question C8): What does CRPR stand for and what does it mean ?
Answer C8):
CRPR stands for Clock Reconvergence Pessimism Removal.
Static timing analysis is a worst case analysis. For setup analysis, it uses the slowest possible launch clock network delay, the slowest possible clock to Q for the launch flop, the slowest possible path delay from the launch flop to Q pin to the capture flop D pin. Also it uses fastest possible clock network delay for capture clock. This way it tries to worst case whole analysis.

If the launch and capture clock networks share a common path, then above mentioned

worse casing is pessimistic as one a common path, you can not have slowest delay and fastest delay happen simultaneously.

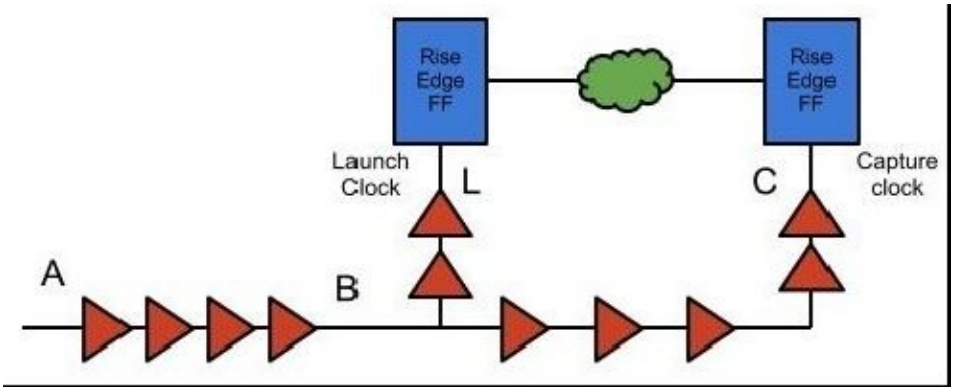


Figure C8. CRPR

For max analysis STA tool will do following.

Use slowest clock network delay from A through B to L
Use fastest clock network delay from A through B to C.

We know that for common segment A through B, using slowest for launch and fastest for capture is pessimistic as it will have only one type of delay.

STA tool will give credit which is equivalent to the difference between slowest delay from A through B and fastest delay from A to B. This credit will be applied during the timing analysis. This is true for min timing analysis as well.

Metastability.

Question M1): What is metastability and what are its effects ?
Answer M1):
Whenever there is setup or hold time violations in a flip-flop, it enters a state where its output is unpredictable. This state of unpredictable output is known as metastable state.

To understand the cause metastability, one has to understand how a latch or a flop works. As shown in following figure a latch(or a flop) has an inverter feedback loop which acts as a memory element.

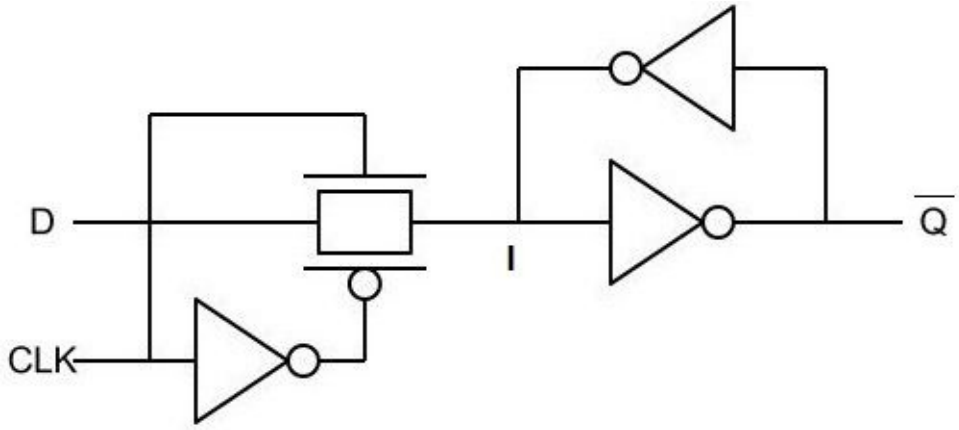


Figure M1a. D latch

If you recall inverter voltage transfer curve, it look something like this.

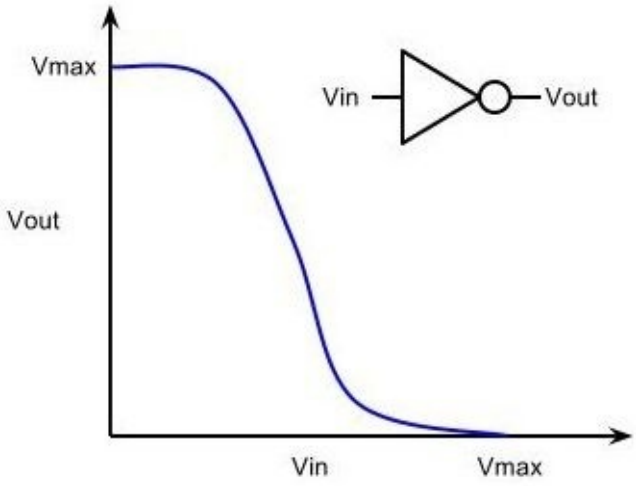


Figure M1b. Inverter voltage transfer curve.

For the inverter loop in the latch above, the voltage transfer curve gets superimposed and it can be seen in the figure below.

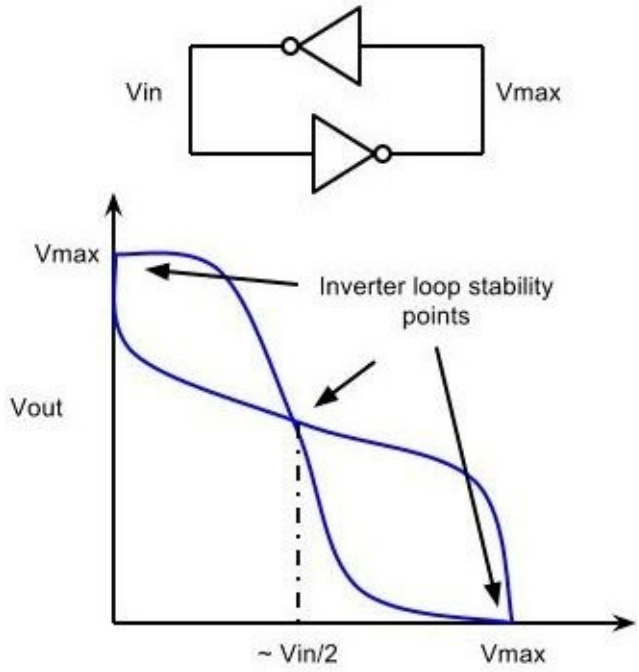


Figure M1c. Inverter loop voltage transfer curve.

Wherever two curves are intersecting, those are the stable points for the inverter loop. You can see it has stability points along the X and Y axis. Those are the cases, input of the inverter loop is either at ‘0’ or Vmax voltage.

You can notice that there is one more point, where transfer curves intersect. This is when the input to the inverter loop is at Vmax/2 voltage. This is the metastable point along the curve. The loop can get stuck here for a very long period of time. But this point is not the most stable point for the inverters loop. The most stable points along the curve are the points along axis, when input is either ‘0’ or Vmax. Depending on the sizes of the inveters in the loop, eventually loop will converge to the most stable points.

When a value is to be written into the latch, the input value is driven through the input pin ‘D’ of the latch, while pass gate is open. This causes the latch node ‘I’ to be written ‘0’ or ‘1’ as input is strongly driven through pin ‘D’. Now if the pass gate is turned off earlier, while latch node ‘I’ has not completely reached to ‘0’ level or Vmax level, the node ‘I’ can get stuck near Vmax/2 value, which is a metastable point.

To avoid from this happening, the input has to be stable a certain time before the the pass gate closes. Pass gate closes when the clock arrives. As we’ll learn in other question, this is called the setup time for the latch.

Metastable state is also called quasi stable state. At the end of metastable state, the flip-flop settles down to either '1' or '0'. The whole process is known as metastability.

Question M2) How to avoid metastability ?
Answer M2)

If we ensure that input data meets setup and hold requirements, we can guarantee that we avoid metastability. Sometimes it's not possible to guarantee to meet setup/hold requirements, especially generating signal is coming from a different clock domain compared to sampling clock.

In such cases, what we do is place back to back flip-flops and allocate extra timing cycles of clocks to sample the data. Such a series of back to back flops is called a metastability hardened flop.

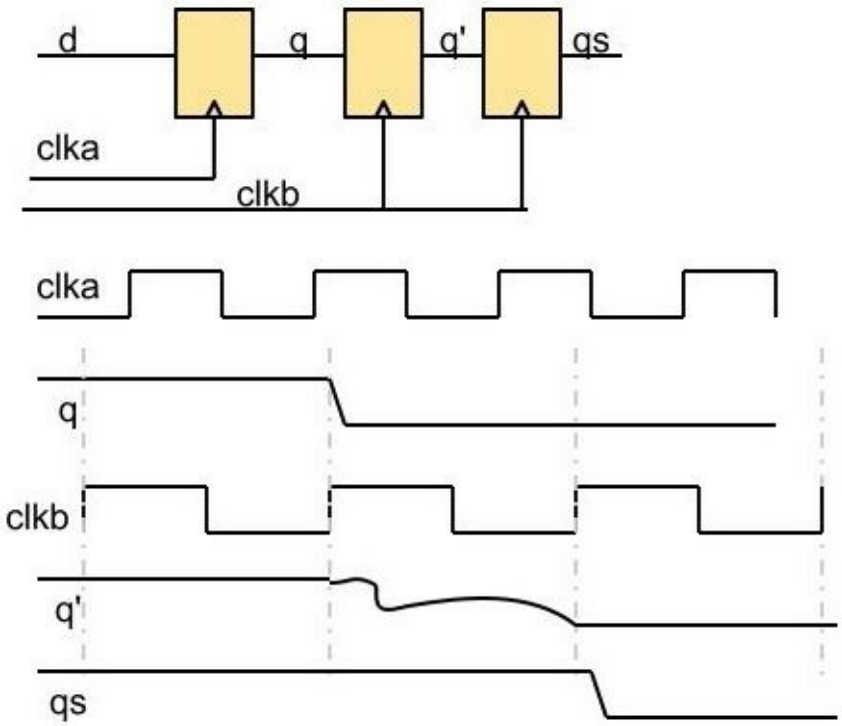


Figure M2. Metastability hardened flops

As you can see in figure, input q to the first flop clocked by clkb changes right when clock is rising, by violating the setup time of this flop. This causes the flip-flop to go metastable, during first sampling clock cycle and we give first flop a full sampling clock cycle to recover from metastability. Within first cycle first flop recovers to correct value, we capture correct value at output second flip-flop at beginning of second clock cycle and qs is the correctly synchronized value available at the beginning of second clock cycle of clkb. If first flop recovers to wrong stage we've to wait for one more cycle i.e. beginning of 3rd cycle of sampling clock to capture the correct value.

Sometimes it's possible that first flop takes longer than one sampling clock cycle to recover to stable value, in which case 3 flip-flops in series can be used. More flops in series reduces the failure in capturing the correct value at output at expense of more number of cycles.

Question M3): How do you synchronize between 2 clock domains?

Answer M3):
There are two ways to do this.
1) Asynchronous FIFO,
2) Synchronizer.

Question M4): Explain the working of a FIFO.
Answer M4):
FIFO is used for high throughput asynchronous data transfer. When you're sending data from one domain to another domain and if high performance is required, you can not just get away with simple synchronizer(Metaflop).

As you can't afford to loose clock cycles(In synchronizer you merely wait for additional clock cycles until you guarantee metastability free operation), you come up with storage element and reasonably complex handshaking scheme for control signals to facilitate the transfer.

An Asynchronous FIFO has two interfaces, one for writing the data into the FIFO and the other for reading the data out of FIFO. It has two clocks, one for writing and the other for reading.

Block A writes the data in the FIFO and Block B reads out the data from it. To facilitate error free operations, we have FIFO full and FIFO empty signals. These signals are generated with respect to the corresponding clock.

Keep in mind that, because control signals are generated in their corresponding domains and such domains are asynchronous to each other, these control signals have to be synchronized through the synchronizer !

FIFO full signal is used by block A (when FIFO is full, we don't want block A to write data into FIFO, this data will be lost), so it will be driven by the write clock. Similarly, FIFO empty will be driven by the read clock. Here read clock means block B clock and write clock means block A clock

Asynchronous FIFO is used at places when the performance matters more, when one does not want to waste clock cycles in handshake and more resources are available.

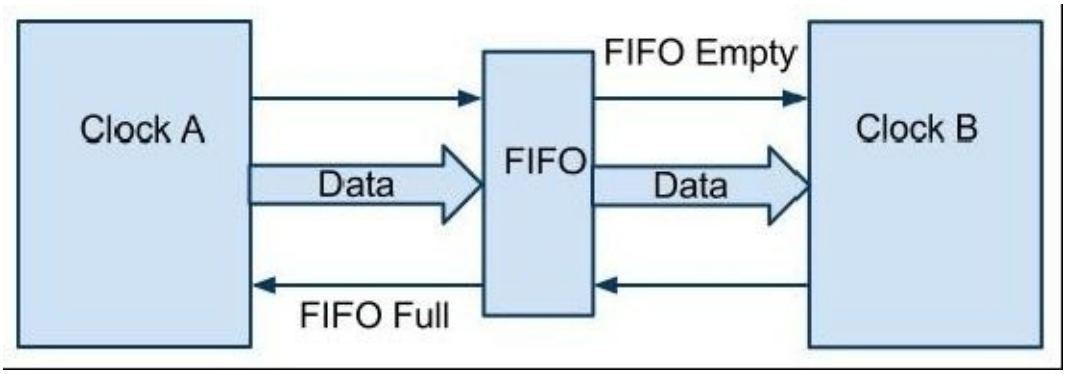


Figure M4. Asynchronous FIFO.

Question D27): How is FIFO depth/size determined ?

Answer D27):

Size of the FIFO depends on both read and write clock domain frequencies, their respective clock skew and write and read data rates. Data rate can vary depending on the two clock domain operation and requirement and frequency. FIFO has to be able to handle the case when data rate of writing operation is maximum and for read operation is minimum.

Miscellaneous.

Question MI1): Design a flip flop using a mux.
Answer MI1):
There is a two step answer to this problem. A flip-flop is designed using two latches in a master slave configuration. Meaning a flip-flop is designed using connecting two latches back to back, first latch being the master and second latch being the slave.

Also we can make a latch using a mux. Hence, we first make a latch using a mux and then connect two such latches back to back in a master slave configuration to come up with the flip-flop.

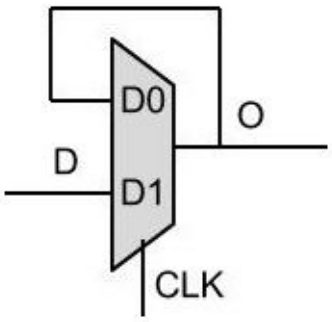


Figure MI1a. Latch using 2:1 MUX.

As shown in the figure, if we tie output of a 2:1 MUX back to D0 pin of mux, whenever select is at ‘0’, we hold the output state as it is fed back to itself through the D0 input of the MUX. We tie input ‘D’ to D1 pin of the MUX and tie select line to CLK(Clock). Whenever select line, which is clock, is high we pass value on input pin D to the output O.
Now let’s explore how a flip-flop is made using two latches.

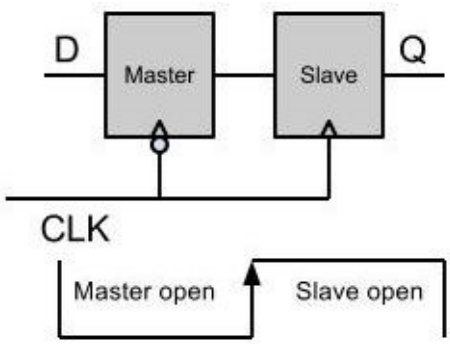


Figure MI1b. Master-Slave flip-flop

As shown in the figure, when you connect a low phase latch followed by a high phase

latch without any delay in between them, you get a flip-flop. First latch is called master and second latch is called slave, because second latch always follows the data that first latch captures. In a sense first latch acts as a master device and second latch always follows master.

Master latch is active low phase, hence it is open during the low phase of the clock and input data at 'D' pin of this latch has to setup to the rising edge of the clock, as that is the edge when master latch closes, or captures the data.
It is important to realize that the data at input pin 'D' of the master latch that arrives at during the transparent window of the master latch, can not push transparently through slave, as while master latch is open during the low phase of clock, slave latch is closed during this time.
When clock rises, master latch capture and transfers data from input 'D' pin to the output the master latch, at the same time, because clock has risen, the slave latch opens and becomes transparent and allows the data at slave latch input to appear on th 'Q' pin of the master-slave.
As you can see the only time a valid output appears from input 'D' to the output 'Q' pin is when clock rises. This way the positive edge triggered master slave flip-flop works.

Having learn about how to come up with a latch using 2:1 MUX and how to make a flip-flop using latches, we can now come up with a flip-flop using 2:1 MUX like following.

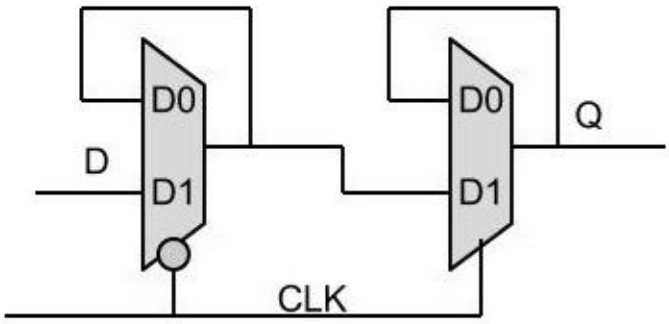


Figure MI1c. Flip-flop using 2:1 MUX

Question MI2): How will a flip flop respond if the clock and D input of a D flipflop are shorted and clock connected to this shorted input?
Answer MI2):
One can expect this flip flop to be in metastable state most of the time. Because with clock and data input tied together, every time clock rises the data will also rise and will definitely violate the setup time and hold time for that flip flop. With the continuous violation of setup and hold time we can expect flip flop to be in metastable state for at least very large amount of time.

Question MI3): How do you fix timing path from latch to latch ?
Answer MI3):
Latch to latch setup time violation is fixed just like flop to flop path setup time violation, where you either speed up the data from latch to latch by either optimizing logic count, or speeding up the gate delays and/or speeding up wire delays.

You can speed up gates by upsizing them and you can speed up wires by either promoting them to higher layers, or widening their width or increasing spacing or shielding them.

You can also fix the timing issues by delaying the sampling clock or speeding up the generating clock. Latch to latch hold violations have inherent protection of a phase or half a clock cycle.

Question MI4) What is the difference between a latch and a flip-flop.
Answer MI4):
Latch is level sensitive device, while flip-flop is edge sensitive. Actually a D flip-flop is made from two back to back latches, in master-slave configuration.

A low level master latch is followed by a high level slave latch to form a rising edge sensitive D flip-flop. Latch is made using fewer devices hence lower power compared to flip-flop, but flip-flip is immune to glitches while latch will pass through glitches.

Question MI5) What happens to delay if you increase load capacitance?
Answer MI5):
Usually device slows down if load capacitance is increased. Device delay depends on three parameters, 1) the strength of device, which usually is the width of the device 2) input slope or slew rate and 3) output load capacitance. Increasing strength or the width increases self load as well.

Question MI6) STA tool reports a hold violation on following circuit. What would you do ?
Answer MI6)

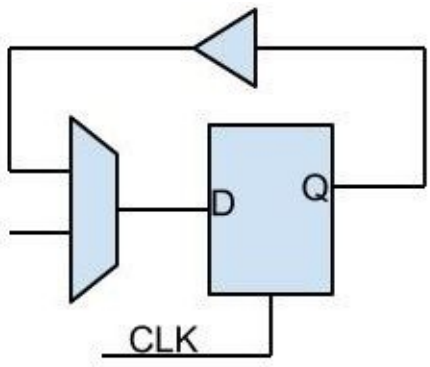


Figure MI6. Hold/Min delay requirement for a flop.

If you go through previous question about hold violation failure, you'll realize that

hold violation normally arises when generating clock and sampling clock are physically separate clocks and because usually there is large clock skew between the clock to the generating flop and the clock to sampling flop.

In this example we're referring to the hold violation reported by tool where the timing path starts at the CLK pin goes through the Q pin through buffer, MUX and comes back to D input pin of the same flop and ends there.

It is obvious that generating CLK edge and sampling CLK is essentially the same edge. This path would never have a real hold violation as we're referring to the same CLK edge. Many times STA tools have limitations and it doesn't realize this situation.

Because the data is released by the very active edge of CLK, against which the hold check is performed, we'll never have a hold violation as long as combined delay for CLK -> Q, buffer and MUX is more than the intrinsic hold requirement of the flop. Remember from the previous question about hold time, that sequentials(flop or latch) have intrinsic hold time requirement which would be more than zero ps in most of the cases.

The key to understand here is that we're referring to the same CLK edge hence no CLK skew and no hold violation.

Question MI7): How much is the max fan out of a typical CMOS gate. Or alternatively,
discuss the limiting factors.
Answer MI7):
Fanout for CMOS gates, is the ratio of the load capacitance (the capacitance that it is driving) to the input gate capacitance. As capacitance is proportional to gate size, the fanout turns out to be the ratio of the size of the driven gate to the size of the driver gate.

Fanout of a CMOS gate depends upon the load capacitance and how fast the driving gate can charge and discharge the load capacitance. Digital circuits are mainly about speed and power tradeoff. Simply put, CMOS gate load should be within the range where driving gate can charge or discharge the load within reasonable time with reasonable power dissipation.

Typical fanout value can be found out using the CMOS gate delay models. Some of the CMOS gate models are very complicated in nature. Luckily there are simplistic delay models, which are fairly accurate. For sake of comprehending this issue, we will go through an overly simplified delay model.

We know that I-V curves of CMOS transistor are not linear and hence, we can't really assume transistor to be a resistor when transistor is ON, but as mentioned earlier we can assume transistor to be resistor in a simplified model, for our understanding. Following figure shows a NMOS and a PMOS device. Let's assume that NMOS

device is of unit gate width ‘W’ and for such a unit gate width device the resistance is ‘R’. If we were to assume that mobility of electrons is double that of holes, which gives us an approximate P/N ratio of 2/1 to achieve same delay(with very recent process technologies the P/N ratio to get same rise and fall delay is getting close to 1/1). In other words to achieve the same resistance ‘R’ in a PMOS device, we need PMOS device to have double the width compared to NMOS device. That is why to get resistance ‘R’ through PMOS device device it needs to be ‘2W’ wide.

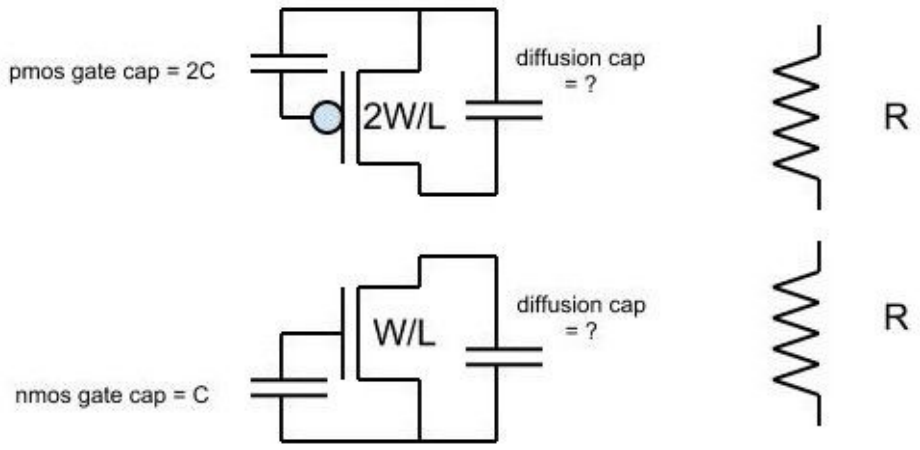


Figure MI7a. R and C model of CMOS inverter

Our model inverter has NMOS with width ‘W’ and PMOS has width ‘2W’, with equal rise and fall delays. We know that gate capacitance is directly proportional to gate width. Lets also assume that for width ‘W’, the gate capacitance is ‘C’. This means our NMOS gate capacitance is ‘C’ and our PMOS gate capacitance is ‘2C’. Again for sake of simplicity lets assume the diffusion capacitance of transistors to be zero.

Lets assume that an inverter with ‘W’ gate width drives another inverter with gate width that is ‘a’ times the width of the driver transistor. This multiplier ‘a’ is our fanout. For the receiver inverter(load inverter), NMOS gate capacitance would be a*C as gate capacitance is proportional to the width of the gate.

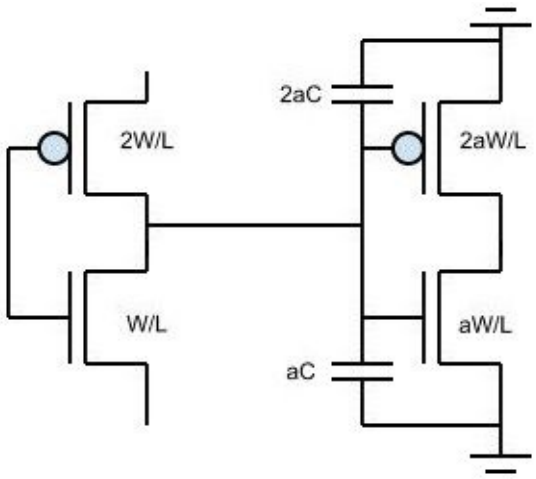


Figure MI7b. Unit size inverter driving ‘a’ size inverter

Now let's represent this back to back inverter in terms of their R and C only models.

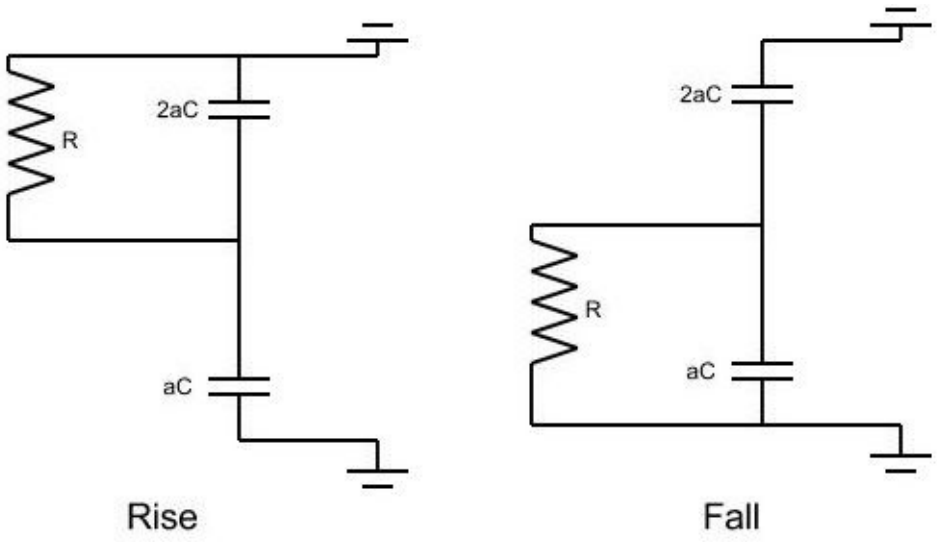


Figure MI7c. Inverter R & C model

For this RC circuit, we can calculate the delay at the driver output node using Elmore delay approximation. If you can recall in Elmore delay model one can find the total delay through multiple nodes in a circuit like this : Start with the first node of interest and keep going downstream along the path where you want to find the delay. Along the path stop at each node and find the total resistance from that node to VDD/VSS and multiply that resistance with total Capacitance on that node. Sum up such R and C product for all nodes.

In our circuit, there is only one node of interest. That is the driver inverter output, or the end of resistance R. In this case total resistance from the node to VDD/VSS is 'R' and total capacitance on the node is 'aC+2aC=3aC'. Hence the delay can be approximated to be 'R*3aC= 3aRC'

Now to find out the typical value of fanout 'a', we can build a circuit with chain of back to back inverters like following circuit.

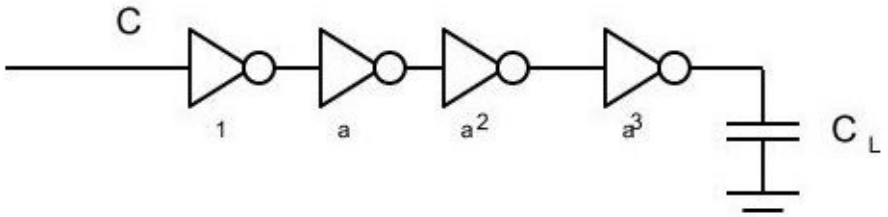


Figure MI7d. Chain of inverters.

Objective is to drive load C_L with optimum delay through the chain of inverters. Lets assume the input capacitance of first inverter is 'C' as shown in figure with unit width. Fanout being 'a' next inverter width would 'a' and so forth.

The number of inverters along the path can be represented as a function of C_L and C like following.

Total number of inverters along chain D = $Log_a(C_L/C) = ln(C_L/C)/ln(a)$

Total delay along the chain D = Total inverters along the chain * Delay of each inverter.

Earlier we learned that for a back to back inverters where driver inverter input gate capacitance is ‘C’ and the fanout ration of ‘a’, the delay through driver inverter is 3aRC

Total delay along the chain D = $ln(C_L/C)/ln(a) * 3aRC$

If we want to find the minimum value of total delay function for a specific value of fanout ‘a’, we need to take the derivative of ‘total delay’ with respect to ‘a’ and make it zero. That gives us the minima of the ‘total delay’ with respect to ‘a’.

$D = 3*RC*ln(C_L/C)*a/ln(a)$

$dD/da = 3*RC* ln(C_L/C) [(ln(a) -1)/ln^2(a)] = 0$

For thsi to be true

$(ln(a) -1) = 0$

Which means : $ln(a) = 1$, the root of which is $a = e$.

This is how we derive the fanout of ‘e’ to be an optimal fanout for a chain of inverters. If one were to plot the value of total delay ‘D’ against ‘a’ for such an inverter chain it looks like following.

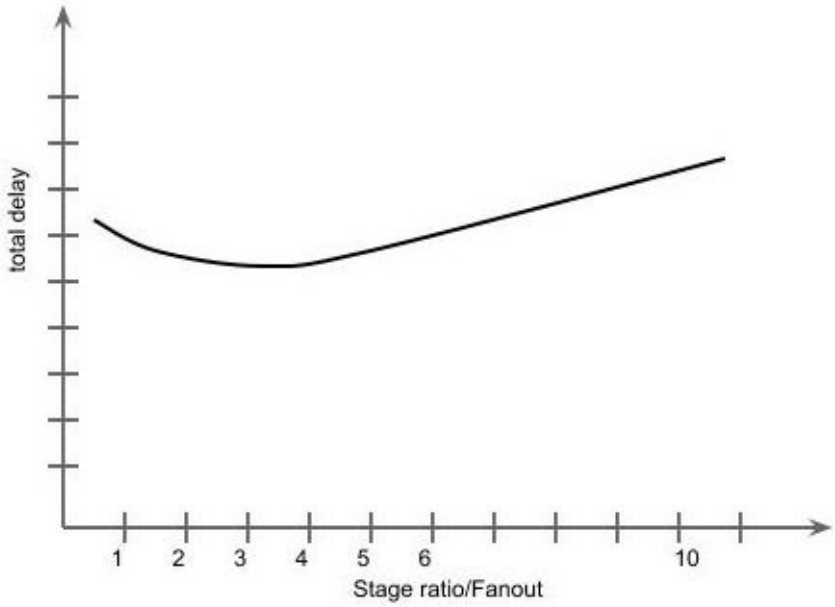


Figure MI7e. Total delay v/s Fanout graph

As you can see in the graph, you get lowest delay through a chain of inverters around ratio of ‘e’. Of course we made simplifying assumptions including the zero diffusion capacitance. In reality graph still follows similar contour even when you improve inverter delay model to be very accurate. What actually happens is that from fanout of 2 to fanout of 6 the delay is within less than 5% range. That is the reason, in practice a fanout of 2 to 6 is used with ideal being close to ‘e’.

One more thing to remember here is that, we assumed a chain of inverter. In practice many times you would find a gate driving a long wire. The theory still applies, one just have to find out the effective wire capacitance that the driving gate sees and use that to come up with the fanout ratio.

Question MI8): How is RC delayed modelled by tools ? What are the RC delay models ?

Answer MI8):

RC delay is modeled as Pi model of varying degree of accuracy.

Most popular model for RC network is Elmore delay model. If you assume that your RC is network is composed of Pi segment of R resistance and Capacitance, following represents the RC structure.

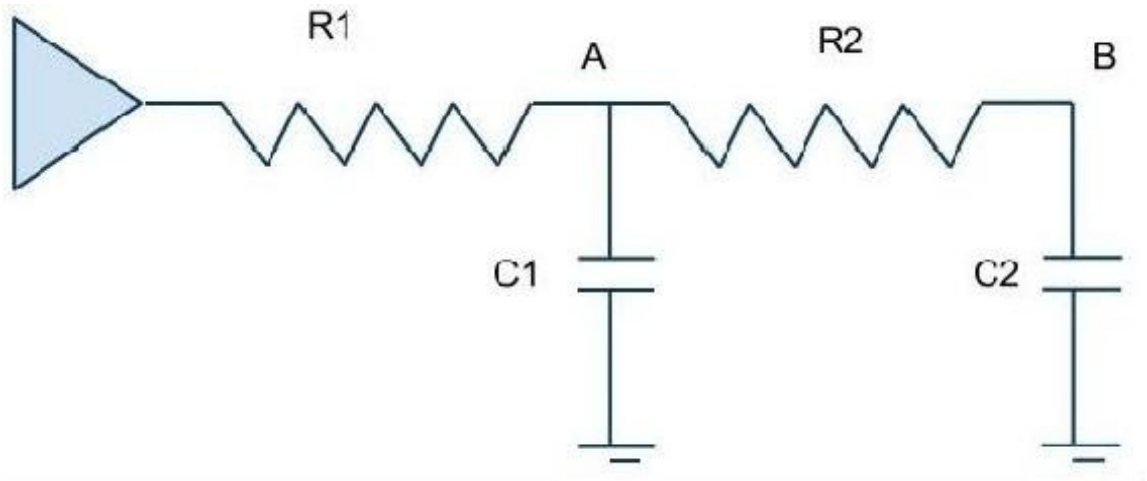


Figure MI8. Elmore delay model.

Question MI9): What's the equation for Elmore RC delay ?
Answer MI9):
For the above mentioned picture, Elmore delay is following
Total Delay at node B = R1C1 + (R1+R2)C2
If this modular structure is extended than delay at the last node can be represented as following.

Total Delay at node N = R1C1 + (R1+R2)C2 + (R1+R2+R3)C3 + ….
(R1+R2+....+RN)CN

Question MI10): what is Statistical STA ?
Answer :
When we refer to statistical STA, we are differentiating between statistical and deterministic STA. Tradition STA is deterministic STA. Because in traditional STA, gate delays and interconnect delays are deterministic and at the end of the analysis we have a deterministic answer of whether the circuit will run at specific frequency or not.

One issue where traditional STA is not very good at is, modelling in die or on chip variation accurately. With fast shrinking geometries, on chip variation is becoming more and more dominant. In traditional STA, worst case analysis along with clock uncertainty, clock and data derating and explicit margins are used to model for in die or on chip variation. Worst case analysis tends to be very pessimistic as it is not practical to assume all devices on the die to be at the worst case at the same time. For clock tree or data branch as the stage count increases the variation effect tends to get mitigated over the stage counts.

This is where the statistical STA comes into picture. Statistical STA tries to solve this modelling problem, by providing a statistical approach to timing analysis which addresses the pessimism in modelling variation. In statistical STA gate and interconnect delays and primary input arrival times are not modelled as deterministic values, but are modeled as random variables and resulting timing criticality of the

circuit is expressed in terms of probability density functions.

Table of Contents