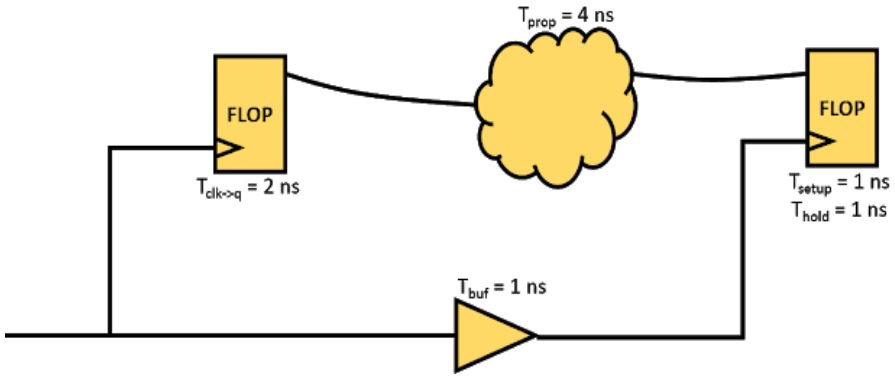
Assignment on Architectural Design of Digital Integrated Circuits

Number-10

(Each question carries 10 marks each)

1. Figure below shows a timing path from a positive edge-triggered flip-flop to a positive edge-triggered flip-flop. Considering clock frequency of 200 MHz, find the setup and hold slacks for this timing path.



Solution: As the clock frequency is given as a 200 MHz, time period = 1/frequency = 5 ns.

Let us first calculate the setup slack. The setup timing equation is given as:
$T_{ck->q} + T_{prop} + T_{setup} - Tskew < T_{period}$
And equation for setup slack is given as:
$SS = T_{period} - (T_{ck->q} + T_{prop} + T_{setup} - Tskew)$
Here,
$T_{ck->q}$ **= 2ns,** T_{prop} **(max value of delay of combinational logic) = 4 ns+** T_{setup} **= 1 ns,** T_{period} **= 5 ns, Tskew = 1 ns**
Putting these values into equation for setup slack, we get setup slack for this timing path.

SS = 5 - (2 + 4 + 1 - 1) ns
SS = -1 ns

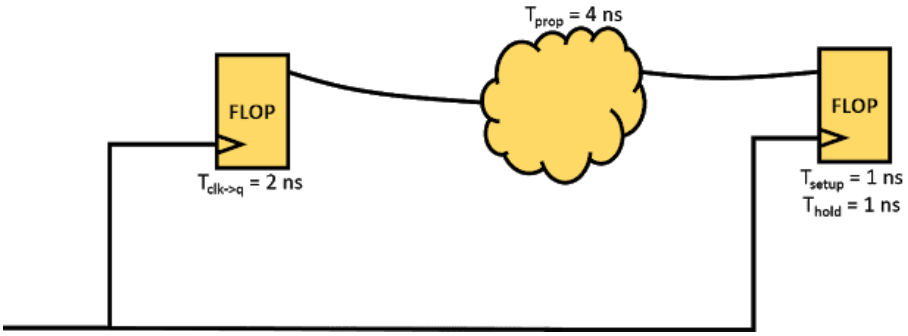
Now, hold slack can be found out from the hold timing equation. The hold timing equation is given as:
$T_{ck->q} + T_{prop} > T_{hold} + T_{skew}$

Here,
$T_{ck->q}$ = 2 ns, T_{prop} (min value of combinational propagation delay) = 4 ns, T_{hold} = 1ns, T_{skew} = 1 ns
And the equation for hold slack is given as:

HS = $T_{ck->q}$ + T_{prop} - (T_{hold} + T_{skew})
HS = 2 + 4 - (1 + 1) = 4 ns

So, for this timing path, setup slack value is -1 ns and hold slack value is 4 ns.

2. Figure below shows a timing path from a positive edge-triggered flip-flop to a positive edge-triggered flip-flop. Considering ideal clocks, and clock frequency of 100 MHz, find the setup and hold slacks for this timing path.



Solution:

Ideally, all the flip-flops in design should get clock at the same time. So, ideal clock means that launch as well as capture flip-flops get clock at zero time. In other words, we can assume that clock skew is zero between start and end points.

As the clock frequency is given as a 100 MHz, time period = 1/frequency = 10 ns.

Let us first calculate the setup slack. The setup timing equation is given as:
$T_{ck->q}$ + T_{prop} + T_{setup} - Tskew < T_{period}
And equation for setup slack is given as:
SS = **T_{period} - ($T_{ck->q}$ + T_{prop} + T_{setup} - Tskew)**
Here,
$T_{ck->q}$ = 2ns, T_{prop} (max value of delay of combinational logic) = 4 ns+ T_{setup} = 1 ns, T_{period} = 10 ns
Putting these values into equation for setup slack, we get setup slack for this timing path.

SS = 10 - (2 + 4 + 1 - 0) ns
SS = 3 ns

Now, hold slack can be found out from the hold timing equation. The hold timing equation is given as:

$T_{ck->q} + T_{prop} > T_{hold} + T_{skew}$

Here,

$T_{ck->q}$ = 2 ns, T_{prop} (min value of combinational propagation delay) = 4 ns, T_{hold} = 1 ns

And the equation for hold slack is given as:

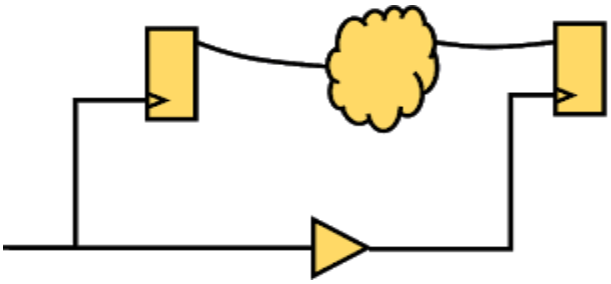
$HS = T_{ck->q} + T_{prop} - (T_{hold} + T_{skew})$
HS = 2 + 4 - (1 + 0) = 5 ns

So, for this timing path, setup slack value is 3 ns and hold slack value is 5 ns.

3. Which type of jitter matters for timing slack calculation? What will happen to clock jitter if I divide down the clock? What will happen to clock jitter for a multicycle path?

Solution:

Which type of jitter matters for timing slack calculation? If we look into the equation of setup slack for a positive edge-triggered flip-flop to another positive edge-triggered flip-flop, we see that setup slack depends upon "clock period". Now, if look closely, we will find that the clock period that we are talking about is actually distance between two clock edges. The larger the distance between the clock edges, greater will be the clock period. Hence, more positive will be setup slack.

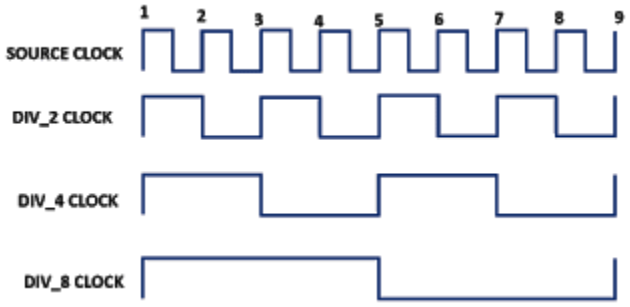


Now, period jitter represents the absolute deviation of clock period from its ideal clock period. So, the jitter we should be looking for is maximum value of "peak-to-peak period jitter". Peak-to-peak period jitter can either increase or decrease clock period. But, we need to take the effect of jitter to decrease clock period. This is because we have to take the worst case of clock period to have most pessimistic setup slack value. And the worst clock period will occur when peak-to-peak jitter is maximum.

So, we can say that for setup slack calculation,

Clock period (actual) = Clock period (ideal) - peak-to-peak jitter (maximum)

What will happen to clock jitter if I divide down the clock?



As we have discussed above, due to clock jitter, for setup calculation, we will assume that peak-to-peak period jitter has caused edge 2 to come closer to edge 1, thereby reducing actual clock period by that margin. Similarly, edge 3 can come closer to edge 2. So, ideally, if we look at DIV_2 clock, the possible jitter here should be 2 times the jitter of SOURCE_CLOCK. Similarly, a DIV_4 clock is expected to have 4 times the jitter and a DIV_8 clock is expected to have 8 times the jitter. And so on.

Now comes the tricky part. As per the definition of long term jitter, nth edge of clock cannot have a jitter more than long term jitter. So, if I say that a PLL has a long term jitter spec of 6 times that of maximum peak-to-peak period jitter, then a DIV_8 clock will have peak-to-peak jitter equal to 6 times the peak-to-peak period jitter of SOURCE_CLOCK. Even a DIV_16 clock will have same maximum jitter.

What will happen to clock jitter for a multicycle path?

Similar to the case of divided down version of clock, a multicycle path also involves other than consecutive edges. So, similar concepts will apply here. So, a multicycle path for setup of 2 will have a jitter of 2 times the peak-to-peak jitter of SOURCE_CLOCK, etc.

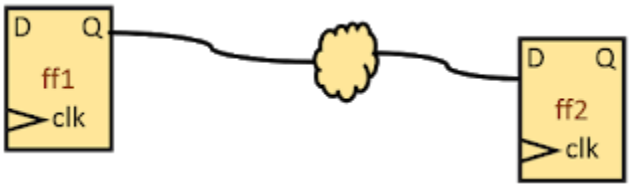
4. A) Can jitter in clock effect setup and hold violations? What is the difference between normal buffer and the clock buffer?

B) What is a glitch? How are glitches harmful?

A) Solution: First of all, we need to understand what is meant by jitter. In most simplistic language, jitter is the uncertainty of a clock source in production of clock edges. For example, if we say that there is a 100 MHz clock source. Ideally, it should produce a clock edge at 0 ns, 10 ns, 20 ns... So, if we say that there was a clock edge at time t = 30 ns, we should get the next clock edge at t = 40 ns. But this is hardly so; due to the uncertainty of getting a clock edge, we might get the next edge between 39.9 ns to 40.1 ns. So, we say that 0.1 ns is the jitter in the period of the

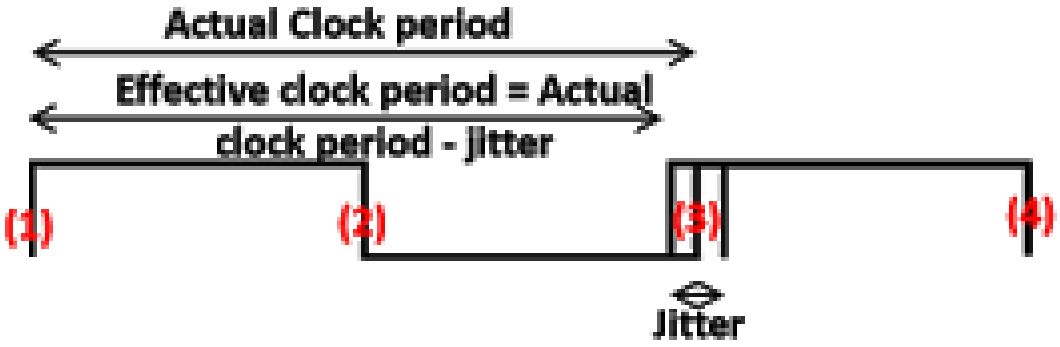
clock. In reality, the definition of jitter is more complex. But, for our scope, this understanding is sufficient.

Let us consider a simple timing path from a positive edge-triggered flip-flop to a positive edge-triggered flip-flop.



Now, let us come to our discussion. First, let us discuss the effect of clock jitter on setup slack.

Effect of clock jitter on setup slack for single cycle paths: From our knowledge of STA basics, setup check formed, in this case, will be from edge 1 -> edge 3. Now, if we know that edge 1 arrived at 20 ns, then edge 3 may arrive at any time (20 ns + CLOCK_PERIOD + jitter) and (20 ns + CLOCK_PERIOD - jitter). So, to cover worst case timing scenario, we need to time as per (20 ns + CLOCK_PERIOD - jitter). So, effectively, we will get (CLOCK_PERIOD - jitter) as effective clock period.



In other words, jitter in clock period makes the setup timing more tight. Or it decreases setup slack for single cycle timing paths.

Effect of clock jitter on hold slack for single cycle paths: Going on the same grounds as setup slack, hold check will be from edge 1 -> edge 1 only. And we know with certainty that edge 1 will leave the source at 20 ns only. So, hold slack should not get bothered by the amount of jitter present at the clock source for single cycle timing paths

Difference between normal buffer and the clock buffer: A buffer is an element which produces an output signal, which is of the same value as the input signal. We can also refer a buffer as a repeater which repeats the signal it is receiving, just as there are repeaters in telephone signal

transmission lines. You must have noticed that we have two kinds of buffers (or any logic gate) available in standard cell libraries as:

- Clock buffer: The clock buffers are designed specifically to have specific properties that are supposed to be good for clock distribution networks (clock trees). The specific properties that are required in an ideal clock tree buffer are given as below. However, it is not possible to attain these ideal properties for every buffer at every technology node. It may be only possible to get close to these properties.
    - Equal rise and fall times
    - Less delays
    - Less delay variations with PVT and OCV

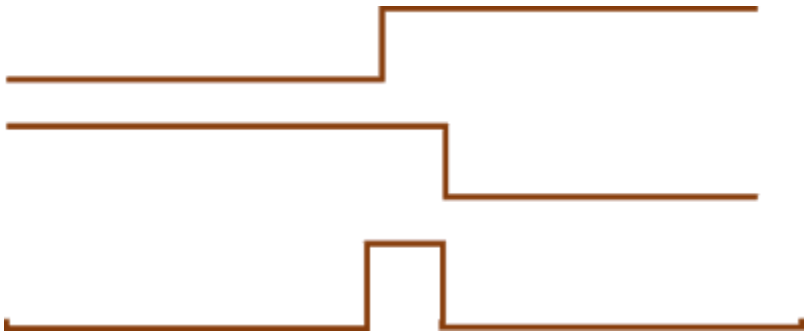
- Normal buffer/data buffer: For a data buffer, the above properties are usually less desired

Usually, we can say that following differences may exist between a clock buffer and a normal buffer:

- In SoCs, clock routing is done in higher metal layers as compared to signal routing. So, to provide easier access to clock pins from these layers, clock buffers may have pins in higher metal layers. That is, vias are provided in standard cell itself instead of necessitating on having in clock distribution network. For a data buffer, the pins are expected to be in lower layers only.
- Clock buffers are balanced. In other words, rise and fall times of clock buffers are nearly equal. The reason behind this is that if the clock buffers are not balanced, there will be duty cycle distortion in the clock tree, which can lead to pulse width violations. On the other hand, data buffers can compromise with either of rise/fall times. In other words, they dont need to have PMOS/NMOS size to be 2:1; and hence, can be of smaller size as compared to clock buffers.
- Due to above reason, clock buffers consume more power as compared to normal buffers.
- Generally, you will find clock buffers with higher drive strength as compared to normal buffers. So that a clock buffer can drive long nets and can have higher fanouts. This helps clock buffers, and hence, clock trees to have less overall delays

B) **What is a glitch**: As per definition, a glitch is any unwanted pulse at the output of a combinational gate. In other words, a glitch is a small spike that happens at the output of a gate. A glitch happens generally, if the delays to the combinational gate output are not balanced. For instance, consider an AND gate with one of its inputs getting inverted and delayed version of its other input. It then will produce a short pulse (or glitch) at the output whenever its input goes from zero to one.

As also said above, this is due to the fact that the delays to the AND gate through two paths are not balanced. Let us elaborate with the help of below waveform. When input goes from zero to one, the other input will go to zero after some time as there is a delay equal to that of an inverter. Due to this, there will be a glitch at the output of AND gate. It needs to be noted that lesser the delay difference between the two inputs at the input of AND gate, lower will be the duration of glitch.

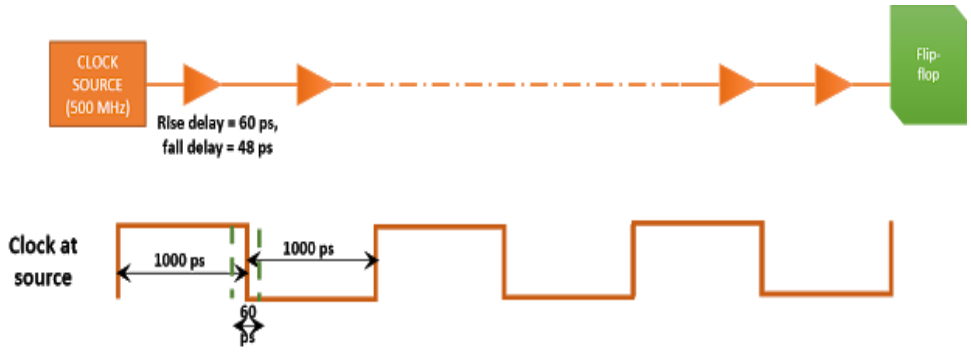


How are glitches harmful? Glitches may be harmful in two ways:

- **Timing/functional issue**: A glitch can be an issue if it propagates to the resultant logic or gets captured by a flip-flop. There can be two cases here:
    - **Synchronous timing paths**: These are timing paths wherein we are required to meet setup and hold timings. So, even if there is a glitch, it will be within the limits of minimum and maximum delays permissible from one flip-flop to another. So, there will be no timing issue provided that you have taken care of setup and hold timings.
    - **Asynchronous timing paths**: If the launch and capture clocks do not have any relationship, setup and hold cannot be ensured. So, if there is a glitch in the data path, it can get captured, hence, can cause issue. To prevent this, synchronizers are used and there are certain rules to be followed for asynchronous paths. These are to be followed to ensure that no wrong data gets captured due to clock glitches. It should be better to call this as functional issue as it can be taken care of only architecturally.
- **High power!!!** Every toggling causes power dissipation due to charging and discharging of gate capacitance. So, a glitch causes power dissipation. Even if there is no timing/functional issue associated with the glitch propagation, power dissipation can be an issue. Larger the combinational path leading to a node, larger the number of probable toggles possible; greater is the expected power dissipation

5. A) What is time borrowing?

B) Consider below figure, wherein minimum pulse width requirement of a flip-flop is 590 ps. It is getting clocked by a PLL of 500 MHz with a duty cycle variation of 60 ps. There are 30 buffers in clock path, each having a rise delay of 60 ps and fall delay of 48 ps. Will this setup be able to meet the duty cycle requirement of flip-flop? Find the slack available



C) How is there degradation in duty cycle of clock? How duty cycle degradation impacts timing?

A) Solution:

Latches exhibit the property of being transparent when clock is asserted to a required value. In sequential designs, using latches can enhance performance of the design. This is possible due to time borrowing property of latches. We can define time borrowing in latches as follows:
Time borrowing is the property of a latch by virtue of which a path ending at a latch can borrow time from the next path in pipeline such that the overall time of the two paths remains the same. The time borrowed by the latch from next stage in pipeline is, then, subtracted from the next path's time.

The time borrowing property of latches is due to the fact that latches are level sensitive; hence, they can capture data over a range of times than at a single time, the entire duration of time over which they are transparent. If they capture data when they are transparent, the same point of time can launch the data for the next stage (of course, there is combinational delay from data pin of latch to output pin of latch).

Let us consider an example wherein a negative latch is placed between two positive edge-triggered registers for simplicity and ease of understanding. The schematic diagram for the same is shown in figure 1 below:

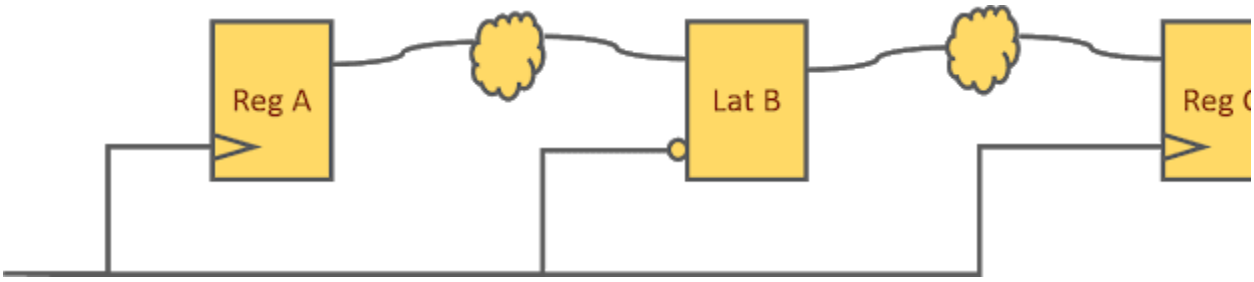


Figure 1: Negative level-sensitive latch between two positive edge-triggered registers

Figure 2 below shows the clock waveform for all the three elements involved. We have labeled the clock edges for convenience. As is shown, lat B is transparent during low phase of the clock. Reg A and Reg C (positive edge-triggered registers) can capture/launch data only at positive edge of clock; i.e., at Edge1, Edge3 or Edge5. Lat B, on other hand, can capture and launch data at any instant of time between Edge2 and Edge3 or Edge4 and Edge5.

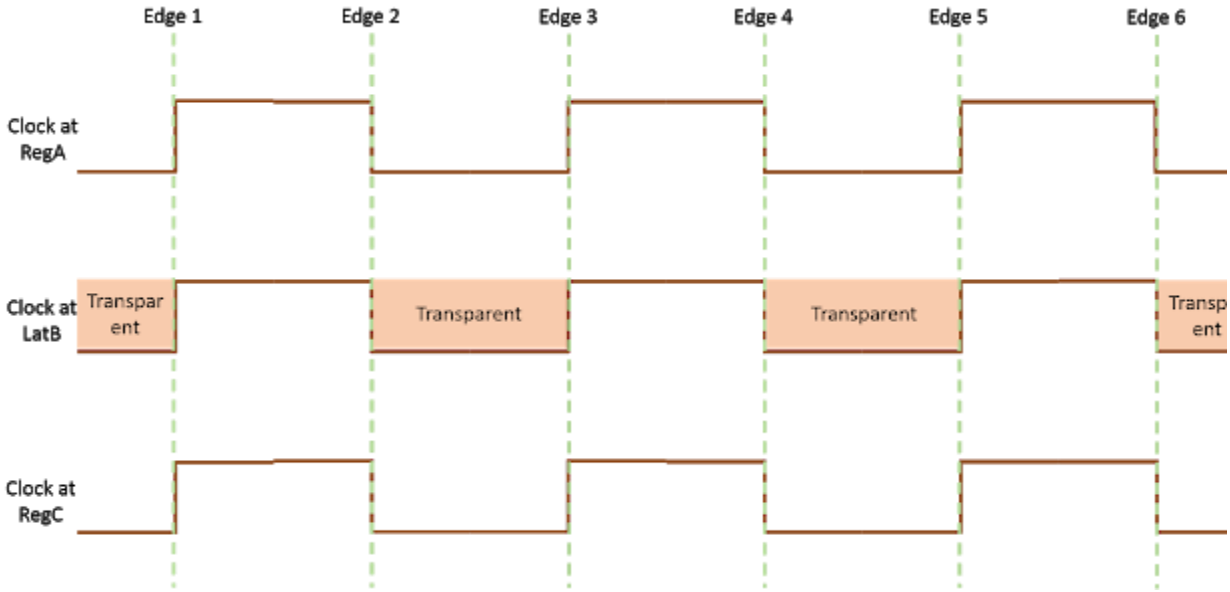


Figure 2: Clock waveforms

The time instant at which data is launched from Lat B depends upon the time at which data launched from Reg A has become stable at the input of lat B. If the data launched at Edge1 from Reg A gets stable before Edge2, it will get captured at Edge2 itself. However, if the data is not able to get stable, even then, it will get captured. This time, as soon as the data gets stable, it will get captured. The latest instant of time this can happen is the latch closing edge (Edge3 here). One point here to be noted is that at whatever point data launches from Lat B, it has to get captured at Reg C at edge3. The more time latch takes to capture the data, it gets subtracted from the next path. The worst case setup check at lat B is at edge2. However, latch can borrow time as needed. The maximum time borrowed, ideally, can be upto Edge3. Figure 3 below shows the setup and hold checks with and without time borrow for this case:

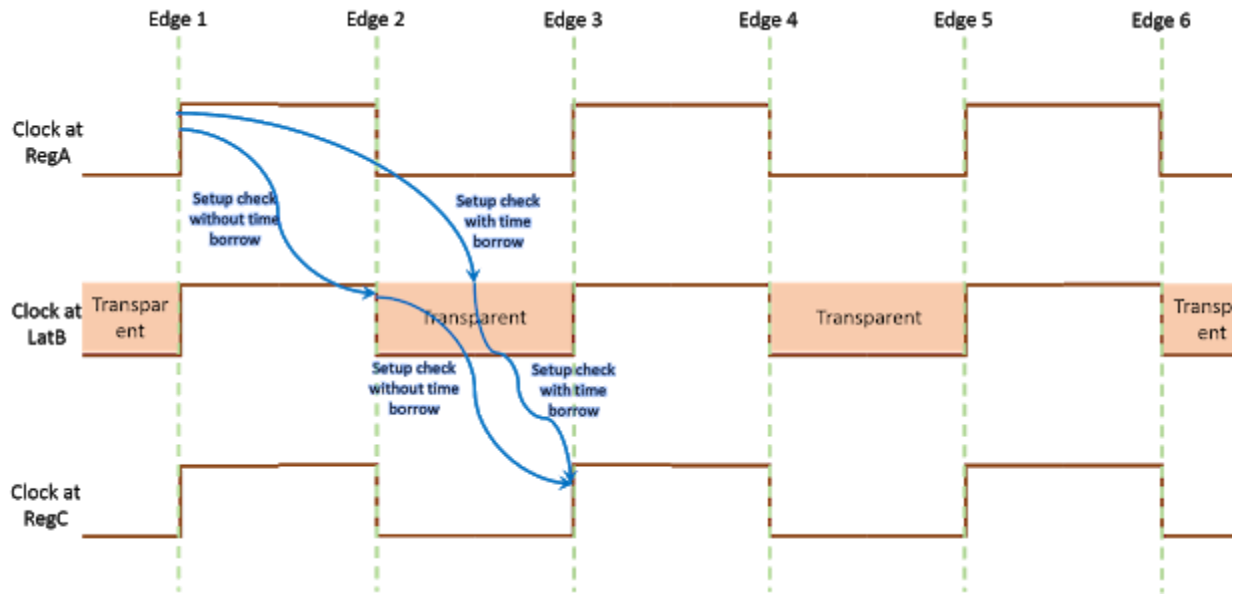


Figure 3: Setup check with and without time borrow

The above example consisted of a negative level-sensitive latch. Similarly, a positive level-sensitive latch will also borrow time from the next stage; just the polarities will be different

B) Solution: Here, we must remember that pulse can be either high pulse or low pulse. So, we need to check for both. Let us start with high pulse:

Pulse width check for high pulse: Here, we are left with calculating the latest possible arrival of rising edge and earliest possible arrival of falling edge at the flip-flop. It is given that

Ideal clock period = 2000 ps (500 MHz frequency)
Ideal half cycle = 1000 ps
Duty cycle variation of clock source = 60 ps
So, if we assume that positive edge of the clock has arrived at 0 time, negative edge can arrive at any time between 940 ps (1000 - 60) and 1060 ps (1000 + 60). Taking the pessimistic case, we have to assume negative edge arrives at 940 ps thereby making the high pulse as 940 ps at clock source.

Now, there are 30 buffers with rise delay of 60 ps and fall delay of 48 ps.

Rising edge will reach flip-flop at time (0 + 30 * 60) = 1800 ps.
Falling edge will reach flip-flop at time (940 + 30 * 48) = 2380 ps
Effective pulse width visible at flip-flop = 2380 - 1800 = 580 ps

Now, the pulse width requirement = 590 ps
Slack = Actual pulse width = Required minimum pulse width = -10 ps

So, we are violating the minimum high pulse width requirement by 10 ps.

Pulse width requirement for low pulse: Similar to the earlier case, we have to find the difference in arrival of latest negative edge and earliest positive edge.

Ideal clock period = 2000 ps (500 MHz)
Ideal half cycle = 1000 ps
Duty cycle variation of clock source = 60 ps
If we assume that negative edge arrived at 0 ps, positive edge can arrive at any time between 940 ps and 1060 ps. Taking the pessimistic case, low pulse width = 940 ps at clock source.

Now, there are 30 buffers with rise delay of 60 ps and fall delay of 48 ps.

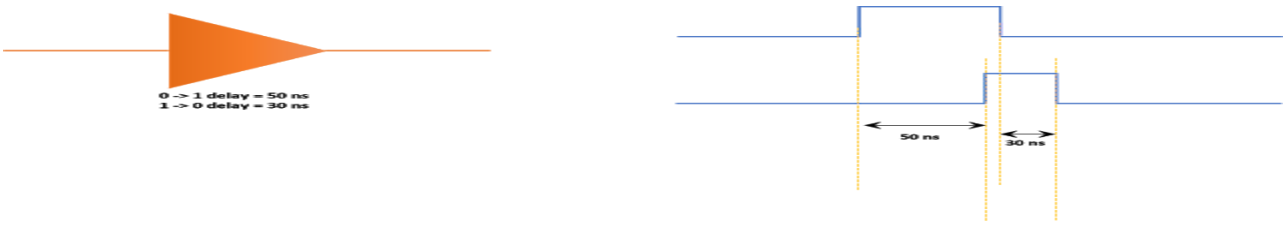
Falling edge will reach flip-flop at time (0 + 30 * 48) = 1440 ps
Rising edge will reach flip-flop at time (940 + 60 * 30) = 2740 ps
Effective pulse width visible at flip-flop = 2740 - 1440 = 1300 ps

Pulse width requirement = 590 ps
Slack = 1300 - 590 = 710 ps

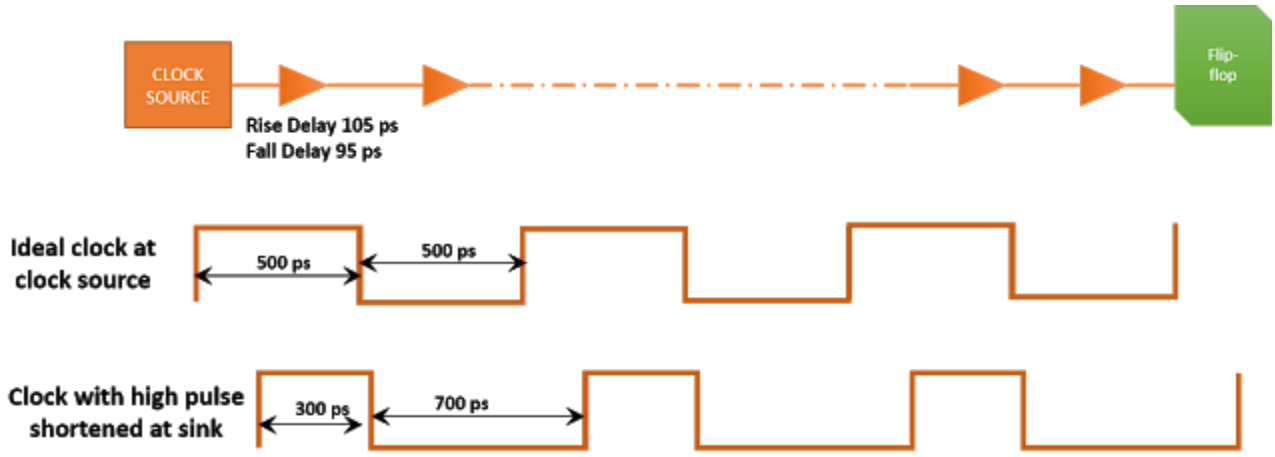
So, we are meeting the low pulse width requirement by 710 ps.

C) Solution:

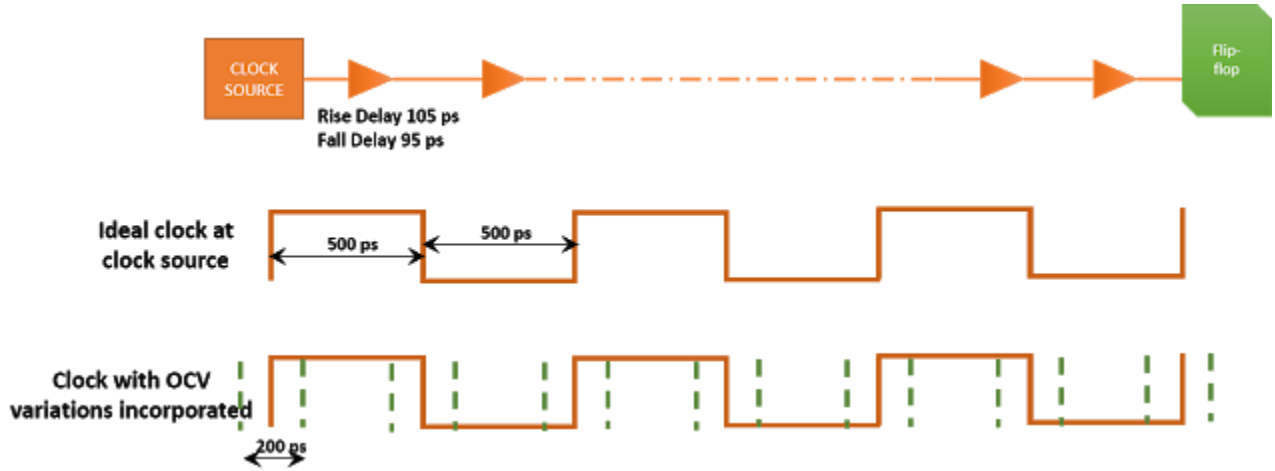
How is there degradation in duty cycle of clock: In addition to source duty cycle variation, there can be asymmetry in rise delay vs fall delay of clock elements. For instance, a buffer may have nominal rise (0 -> 1) delay of 50 ns whereas 48 ns for fall delay (1 -> 0). So, if a clock pulse passes through it, it will eat a portion of this clock pulse as shown in figure 1 below. For more clarity, we have exaggerated the scenario with a fall delay of 30 ns.



There are a lot of delay elements in clock distribution networks (also called clock tree network) inside the SoC. So, this problem is bound to happen there. Let us say, clock path has 20 buffers, each having a rise delay 10 ps greater than fall delay. So, the high pulse will get shortened when the clock reaches its sink. How the pulse gets shortened if there is asymmetry in rise vs fall delay of a delay element or logic gate is shown in below figure.



Even if we assume that the delay element has rise delay equal to fall delay, still, there is possibility of duty cycle degradation. Normally, a buffer (or inverter) has a nominal delay with some delay variations (for instance, OCVs) to be taken into account. For instance, it may have a rise delay of 100 ps with OCV variation to be taken of 5%. So, depending upon the scenario, we need to take its delay as either 95 ps or 105 ps. Similarly, even if we say that fall delay is exactly equal to rise delay, even then because of OCV variations, fall delay to be taken into account will be different than rise delay. Let us suppose, there are 20 such buffers in clock path with an ideal clock source. Then, we will have uncertainty in arrival of both rise and fall edges. And the effect will be visible in timing paths' slack.



How duty cycle degradation impacts timing: Degradation in duty cycle impacts timing wherever both rising and falling edges of clock are involved. For instance, it will impact half cycle timing paths as well as minimum pulse width check.