Static Timing Analysis

Part 2

Amr Adel Mohammady

in /amradelm

This Document Is Dedicated to Thousands of Palestinian Children Who

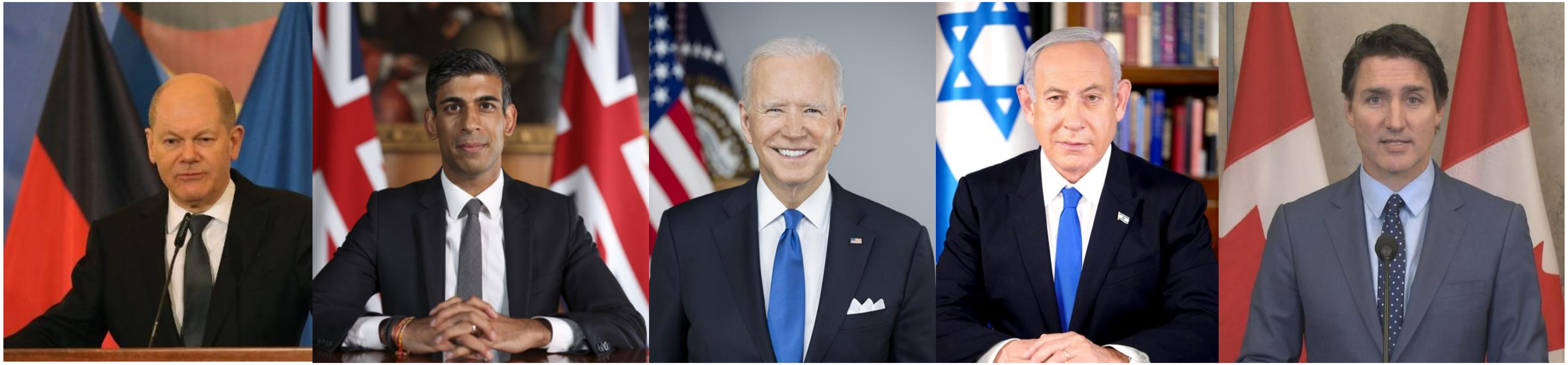
Were Killed

Lost Their Limps

Became Orphans

Are Starved

At The Hands of These War Criminals



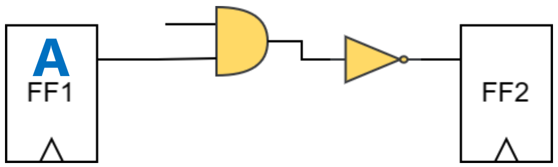
Introduction

- In part 1 we went through the basic principles that are needed to understand all VLSI timing checks. In this parts we will go through some of the checks in details
- **The timing checks covered in this part are:**
  - Setup timing
  - Hold timing

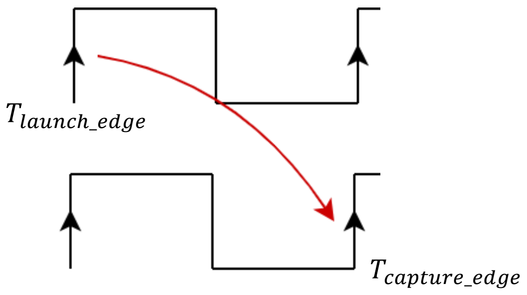
Setup Timing Analysis

Setup Time

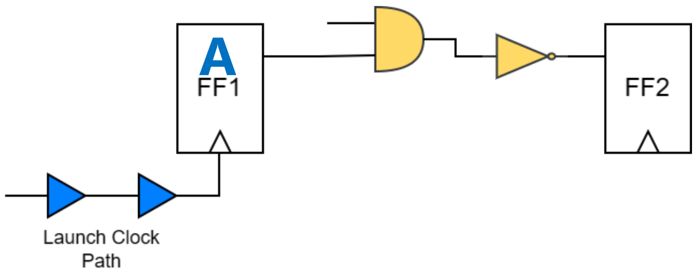
1 At time T=T_{launch_edge}, Data **A** is launched from FF1 to FF2. The data needs to make it to FF2 before the next clock edge arrives at FF2 at time $T_{capture\_edge}$. The next clock edge will arrive after a clock period



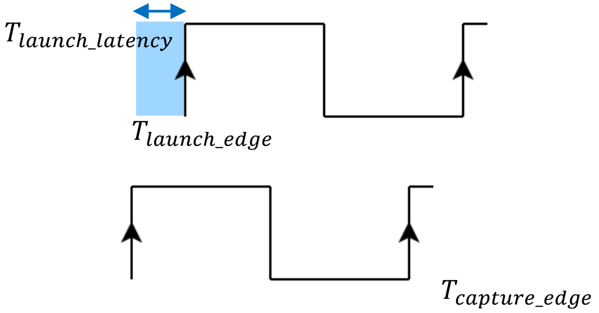
$$\begin{aligned} Launch &= T_{launch\_edge} \\ Capture &= T_{capture\_edge} \end{aligned}$$



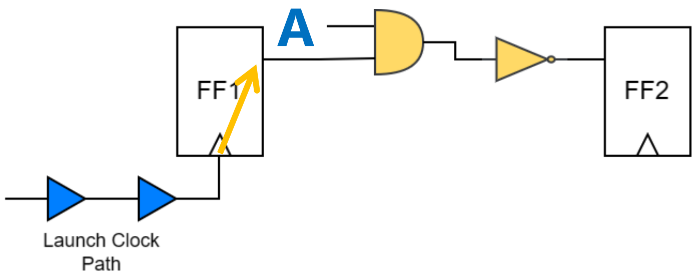
2 The clock takes some time to reach FF1 due to the buffers. The launch won't happen exactly at T=T_{launch_edge} but after the delay/latency of the clock buffers



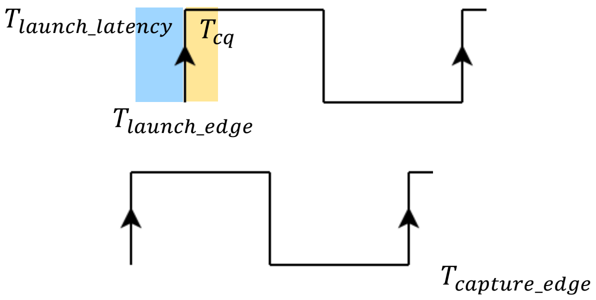
$$\begin{aligned} Launch &= T_{launch\_edge} + \boldsymbol{T_{launch\_latency}} \\ Capture &= T_{capture\_edge} \end{aligned}$$



3 As we saw in part 1, once the clock reaches the FF it takes some time to push the data out to the Q pin. We called this time T_{cq}. This is the 1st delay data **A** encounters to reach FF2

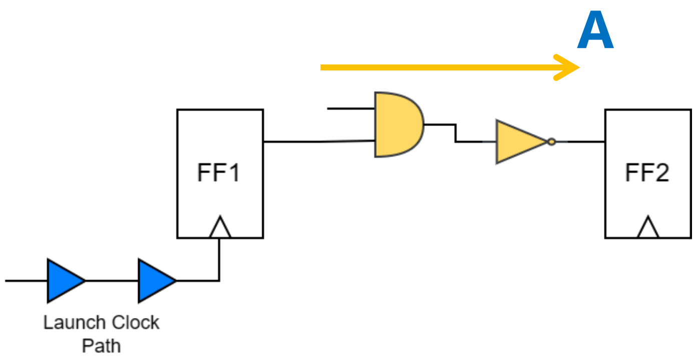


$$\begin{aligned} Launch &= T_{launch\_edge} + T_{launch\_latency} \\ Delay &= \boldsymbol{T_{cq}} \\ Capture &= T_{capture\_edge} \end{aligned}$$

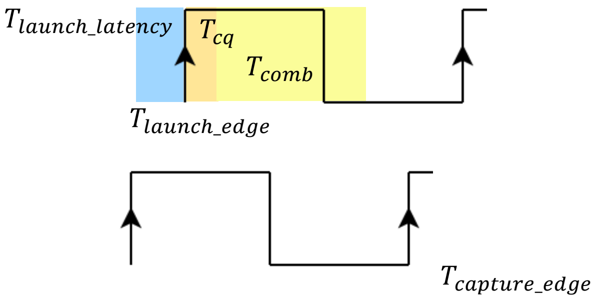


Setup Time

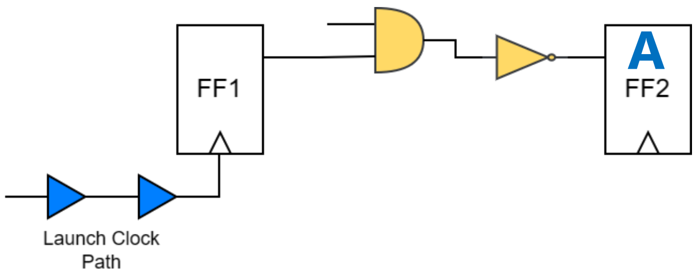
4 Data **A** will propagate through the combinational path to reach FF2. This is the 2nd delay it encounters



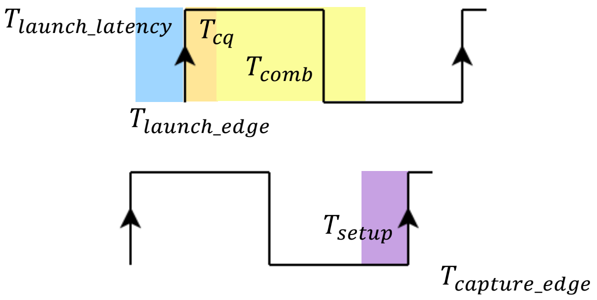
$$\begin{aligned} \mathit{Launch} &= T_{launch\_edge} + T_{launch\_latency} \\ \mathit{Delay} &= T_{cq} + \boldsymbol{T_{comb}} \\ \mathit{Capture} &= T_{capture\_edge} \end{aligned}$$



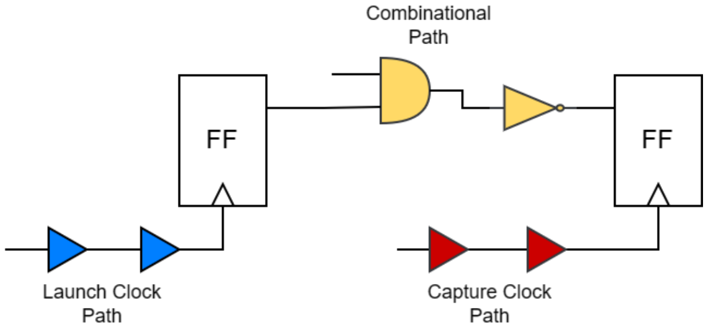
5 As we saw in part 1, the FF requires the data to arrive some time before the clock edge in order to avoid metastability. We called this time T_{setup}. Hence, we shouldn't capture data at $T_{capture\_edge}$ but at $T_{capture\_edge} - T_{setup}$



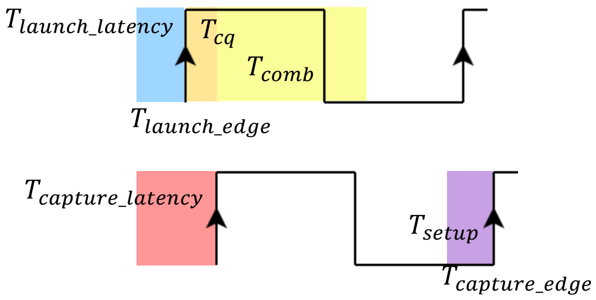
$$\begin{aligned} \mathit{Launch} &= T_{launch\_edge} + T_{launch\_latency} \\ \mathit{Delay} &= T_{cq} + T_{comb} \\ \mathit{Capture} &= T_{capture\_edge} - \boldsymbol{T_{setup}} \end{aligned}$$



6 The clock takes some time to reach FF2 due to the buffers. The capture won't happen exactly at $T_{capture\_edge} - T_{setup}$ but after the delay/latency of the clock buffers



$$\begin{aligned} \mathit{Launch} &= T_{launch\_edge} + T_{launch\_latency} \\ \mathit{Delay} &= T_{cq} + T_{comb} \\ \mathit{Capture} &= T_{capture\_edge} - T_{setup} + \boldsymbol{T_{capture\_latency}} \end{aligned}$$



Setup Time

7 To make sure a setup violation doesn't happen, we need to make sure data **A** arrives at FF2 before the required capture time

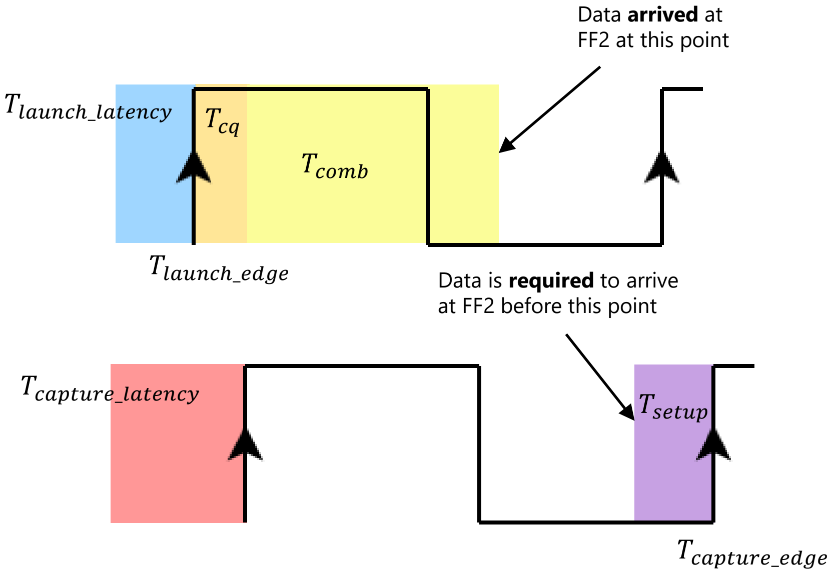
The difference between the required and arrival time is called the **slack**. If the slack is positive we pass setup and if negative we fail.

The launch FF is called the **startpoint** of the timing path and the capture FF is called the **endpoint**

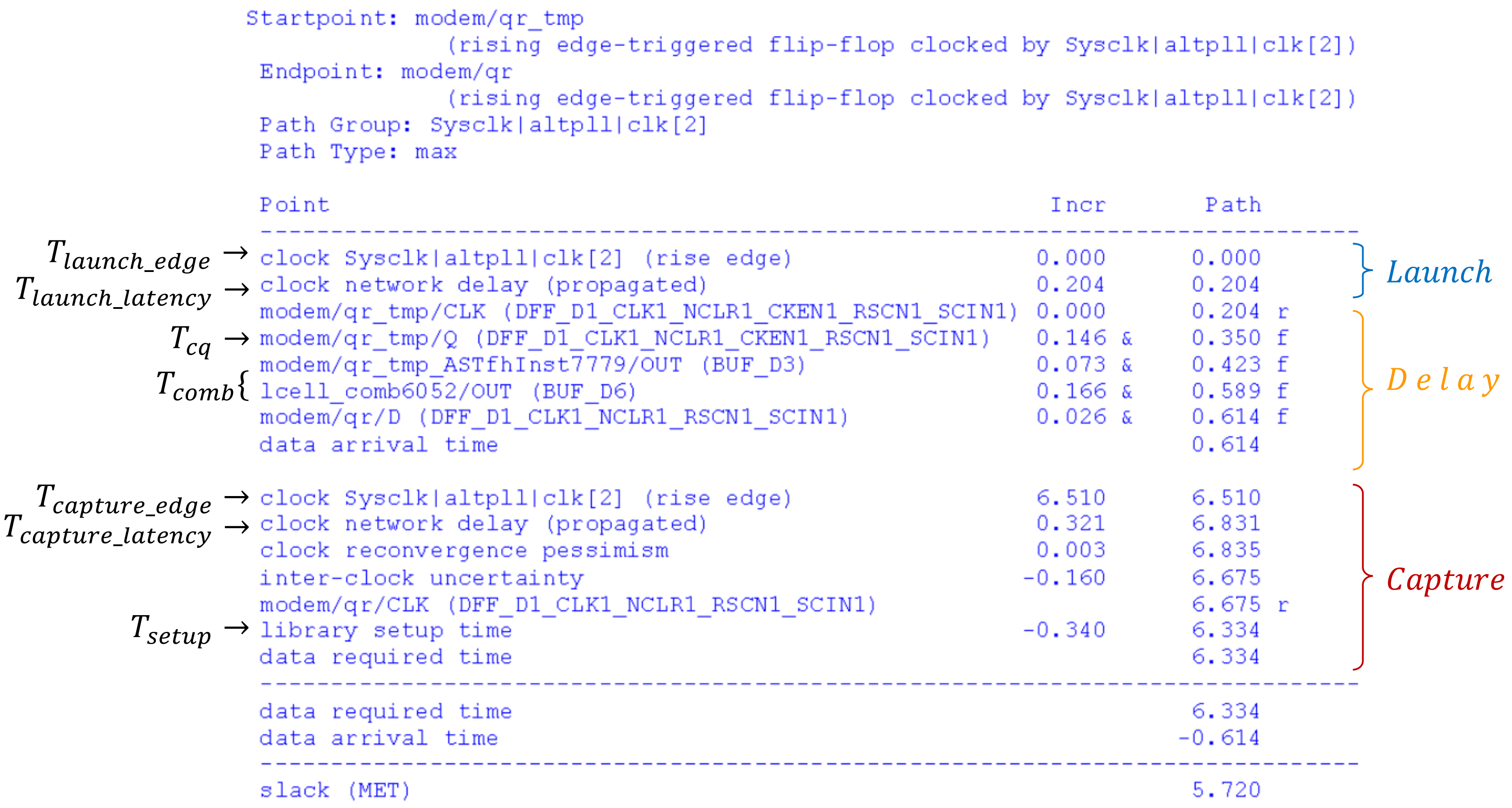
$$Launch + Delay \leq Capture$$

$$Arrival \leq Required$$

$$T_{launch\_edge} + T_{launch\_latency} + T_{comb} < T_{capture\_edge} - T_{setup} + T_{capture\_latency}$$



Example Timing Report



```
Startpoint: modem/qr_tmp
              (rising edge-triggered flip-flop clocked by Sysclk|altpll|clk[2])
 Endpoint: modem/qr
              (rising edge-triggered flip-flop clocked by Sysclk|altpll|clk[2])
 Path Group: Sysclk|altpll|clk[2]
 Path Type: max

 Point                                                    Incr       Path
 ------------------------------------------------------------------------------
 clock Sysclk|altpll|clk[2] (rise edge)                  0.000      0.000
 clock network delay (propagated)                        0.204      0.204
 modem/qr_tmp/CLK (DFF_D1_CLK1_NCLR1_CKEN1_RSCN1_SCIN1)  0.000      0.204 r
 modem/qr_tmp/Q (DFF_D1_CLK1_NCLR1_CKEN1_RSCN1_SCIN1)    0.146 &    0.350 f
 modem/qr_tmp_ASTfhInst7779/OUT (BUF_D3)                 0.073 &    0.423 f
 lcell_comb6052/OUT (BUF_D6)                             0.166 &    0.589 f
 modem/qr/D (DFF_D1_CLK1_NCLR1_RSCN1_SCIN1)              0.026 &    0.614 f
 data arrival time                                                  0.614

 clock Sysclk|altpll|clk[2] (rise edge)                  6.510      6.510
 clock network delay (propagated)                        0.321      6.831
 clock reconvergence pessimism                           0.003      6.835
 inter-clock uncertainty                                -0.160      6.675
 modem/qr/CLK (DFF_D1_CLK1_NCLR1_RSCN1_SCIN1)                       6.675 r
 library setup time                                     -0.340      6.334
 data required time                                                 6.334
 ------------------------------------------------------------------------------
 data required time                                                 6.334
 data arrival time                                                 -0.614
 ------------------------------------------------------------------------------
 slack (MET)                                                        5.720
```

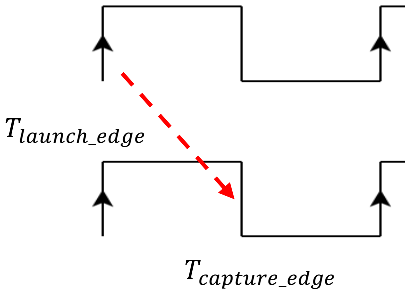
$$T_{launch\_edge} + T_{launch\_latency} + T_{comb} < T_{capture\_edge} - T_{setup} + T_{capture\_latency}$$

Reference : An 554: How to Read HardCopy PrimeTime Timing Reports By Intel

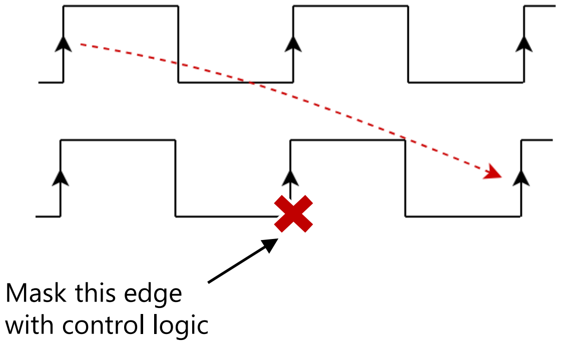
Setup Time

- The example we have shown is for a full cycle path where the $T_{capture\_edge}$ comes one clock cycle after T_{launch_edge}.
- This is not always the case. The capture edge could come half cycle later, multiple cycles later or from another clock.
  - ○ Half cycle paths occur when the launch and capture FFs use different clock edges
  - ○ Multi cycle paths occur when the first capture edge is masked by a control circuit and another edge is used.
  - ○ Multi clock paths occur when the launch and capture FFs use different clocks from each other. The diagram shows that there could be more than one launch/capture edges combination. The STA tools will consider the worst case (The **purple** one)[1]
- All what we learned still apply and nothing changes. We will just plug different values for the clock edges into the setup equation.
- We will now discuss how to fix a setup violation

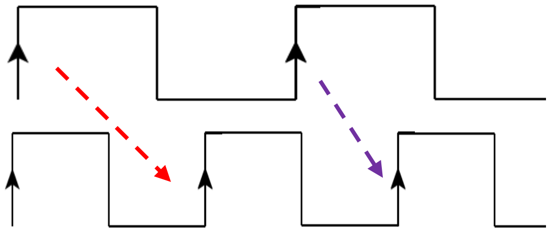
$$\boldsymbol{T_{launch\_edge} + T_{launch\_latency} + T_{comb} < T_{capture\_edge} - T_{setup} + T_{capture\_latency}}$$



Half Cycle Path



Multi Cycle Path

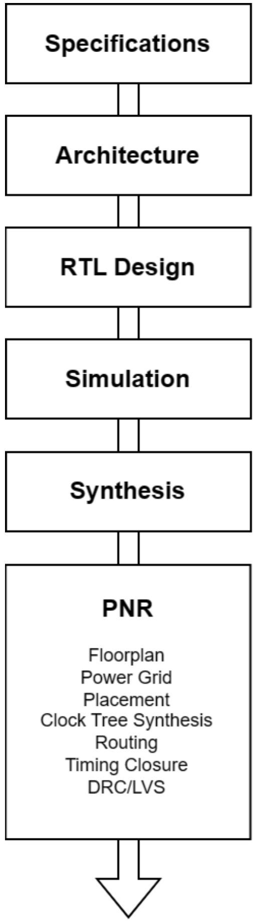


Multi Clock Path

[1]: The phase difference between the two clocks should be known in order to know exactly where the launch and capture edge are. If not, we can't run STA on such paths and we have to resort to clock domain crossing techniques.

Overview of The Digital VLSI Flow

- Before we discuss how to fix a setup violation we need to have a quick overview of the digital design flow[1].
- **Specifications**: The design process starts with the requirements to build the system (Functionality, Performance, Power consumption, Cost etc)
- **Architecture** : Based on the required specs, the architecture team will start building the system. They will answer questions and make decisions such as: What blocks are needed in the system to perform the functionality? How to implement these blocks as a digital circuit? Do we need memory or not? What is its size? What operating voltage do we use? What clock frequency do we need to meet the performance specs? What is the expected area of the chip and fabrication cost?
- **RTL Design** : The RTL team will start writing RTL code to implement the architecture and blocks of the system
- **Simulation** : The implemented design is tested through simulations to make sure it does the required function correctly
- **Synthesis** : The RTL code is translated into actual logic gates and digital blocks
- **PNR** : The place and route step involves several sub steps
  - Floorplan : Involves allocating space on the chip for various blocks and modules, including the placement of macros, and I/O ports
  - Power Grid : Creating the metal structure that delivers the power supply to the standard cells and blocks inside the chip
  - Placements : Placing the cells inside the chip
  - Clock Tree Synthesis : Creating the clock networks to deliver the clocks from the ports to the registers in the chip
  - Routing : Routing the metal interconnects (wires) between the cells
  - Timing Closure : Running STA on the chip to make sure it meets the timing requirements
  - DRC/LVS/EMIR : DRC ensures the final layout is compliant with the manufacturing rules. LVS ensures the final layout perform the same function of the schematic/logic description. EMIR ensures all cells get the required voltage without drop and the current flowing through the wire is within the required limits



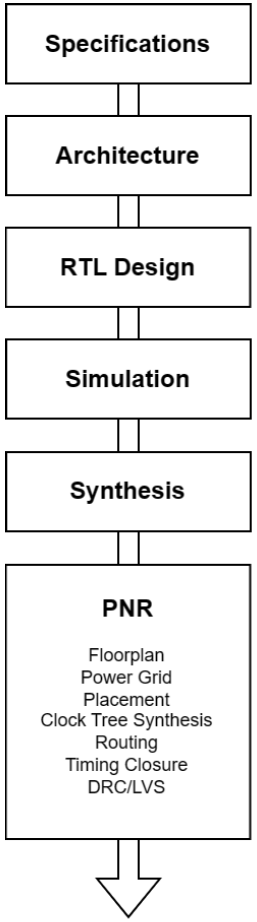
[1] : This is a very simplified view of the digital flow. There are more steps involved but we don't mention them because they won't affect STA

How to Fix a Setup Violation – Overview of The Digital VLSI Flow

- **PNR** :
  - The PNR engineer starts the flow trying to meet the requirements with the help of automation tools
  - The goal is to reach a good startpoint with good results before the manual work starts
  - Once the manual work starts, the startpoint is saved and frozen. The PNR flow is said to be in ECO mode (Engineering Change Order)
  - The manual work involves things like moving cells, changing their threshold voltage, manually routing wires, etc.

- **Fixing timing:**
  - Each of the above flow steps involves several optimizations to enhance the timing and fix the violations
  - The earlier steps solve larger timing violations that are difficult and sometimes impossible to fix in later stages
  - As we go through the flow, the ability to fix large violations decrease and we are more focused in fixing small but tricky violations that involves lots of manual work.

- We will now go through some of the ways to fix a setup violation. We will start with the solutions done in the early stages and go down till we see what can be done in later and final stages.



How to Fix a Setup Violation – Sol. 1

$$T_{launch\_edge} + T_{launch\_latency} + T_{comb} < T_{capture\_edge} - T_{setup} + T_{capture\_latency}$$

Reducing the Clock Frequency

- The easiest and simplest solution is to reduce the frequency (increase the period) of the clock to add time to the capture time
- Doing this degrade the performance (Data rate / CPU speed / Operations per second / etc)
- The decision to reduce the clock frequency is left to the architecture team and can't be modified individually by RTL or PNR engineers
- Sometimes this solution is not acceptable because the product standard requires specific data rate that needs to be met

How to Fix a Setup Violation – Sol. 2

$$T_{launch\_edge} + T_{launch\_latency} + T_{comb} < T_{capture\_edge} - T_{setup} + T_{capture\_latency}$$

Going To a Smaller Technology Node

- In part 1 we showed how the transistor length (tech node) affects the gate delay. A shorter length has smaller delay
- Going for a smaller tech node means higher fabrication cost and a longer design cycle because smaller tech nodes are more challenging to handle the on chip variations (OCV) and the physical design rule constraints (DRC) and preparing the design files (standard cell libraries, etc) for the new tech node will take time.
- Because of this, the target tech node is decided very early in the design process by doing experiments with the tech node to see if the target frequency will be feasible or not
- These experiments could be :
  - **Quick hand calculations** : By considering the average cell delay in the tech node and the average combinational path length. For example, $T_{avg} = 5ns$ and the average number of cells in a timing path = 20. So, on average, the combinational delay = $5 * 20 = 100ns$ meaning a maximum clock frequency of $\frac{1}{100e-9} = 10\ \text{MHz}$.

    This is, of course, a very rough estimation as it doesn't include the effects of wire delay, clock latencies, etc. But the more effort you put in these calculations the more accurate they get
  - **Doing a quick project** : By synthesizing a small block or a previous project to get an estimate of the maximum clock frequency you can achieve on this tech node

How to Fix a Setup Violation – Sol. 3

$$T_{launch\_edge} + T_{launch\_latency} + T_{comb} < T_{capture\_edge} - T_{setup} + T_{capture\_latency}$$

Increasing the Supply Voltage

- In part 1 we showed how the supply voltage affects the gate delay. A higher voltage has smaller delay. However, the power consumption increase quadratically.
- The higher voltage could be applied to certain parts of the chip that needs high performance while leaving other parts with the lower voltage to avoid higher power consumption. However, this adds several difficulties in the ASIC design process

$$t_{prop} = \frac{0.69\, VDD\, .\, C_L}{\frac{W}{2L}\mu C_{ox}(VDD - V_{th})^2}$$

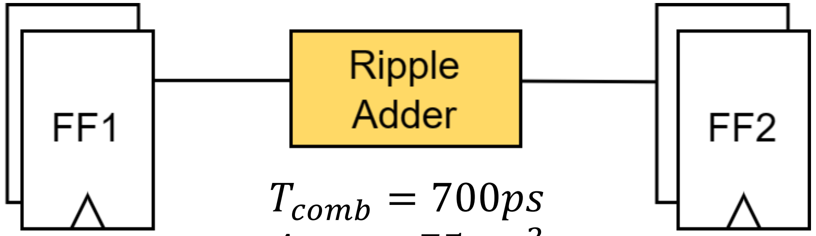
$$Power_{dynamic} = \alpha f C_L V_{DD}^2$$

How to Fix a Setup Violation – Sol. 4

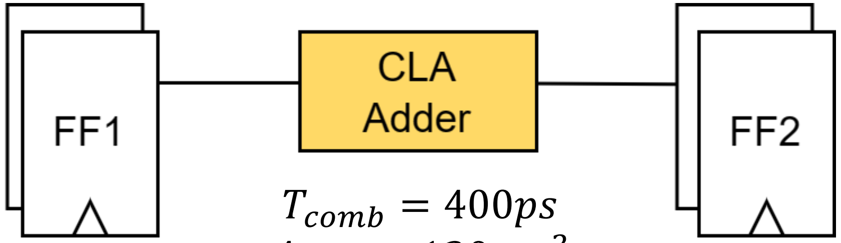
Changing the Architecture

$$T_{launch\_edge} + T_{launch\_latency} + T_{comb} < T_{capture\_edge} - T_{setup} + T_{capture\_latency}$$

- Digital blocks have a tradeoff between speed vs power and area. The designer might choose an implementation that consume more power or has larger area but higher speed.
- For example, there are different ways to implement binary adders. One implementation is the ripple adder which has small area and power consumption but has high T_{comb}, while a carry-look-ahead (CLA) adder has smaller T_{comb} but takes larger area.



$$T_{comb} = 700ps$$
$$Area = 75\mu m^2$$



$$T_{comb} = 400ps$$
$$Area = 130\mu m^2$$

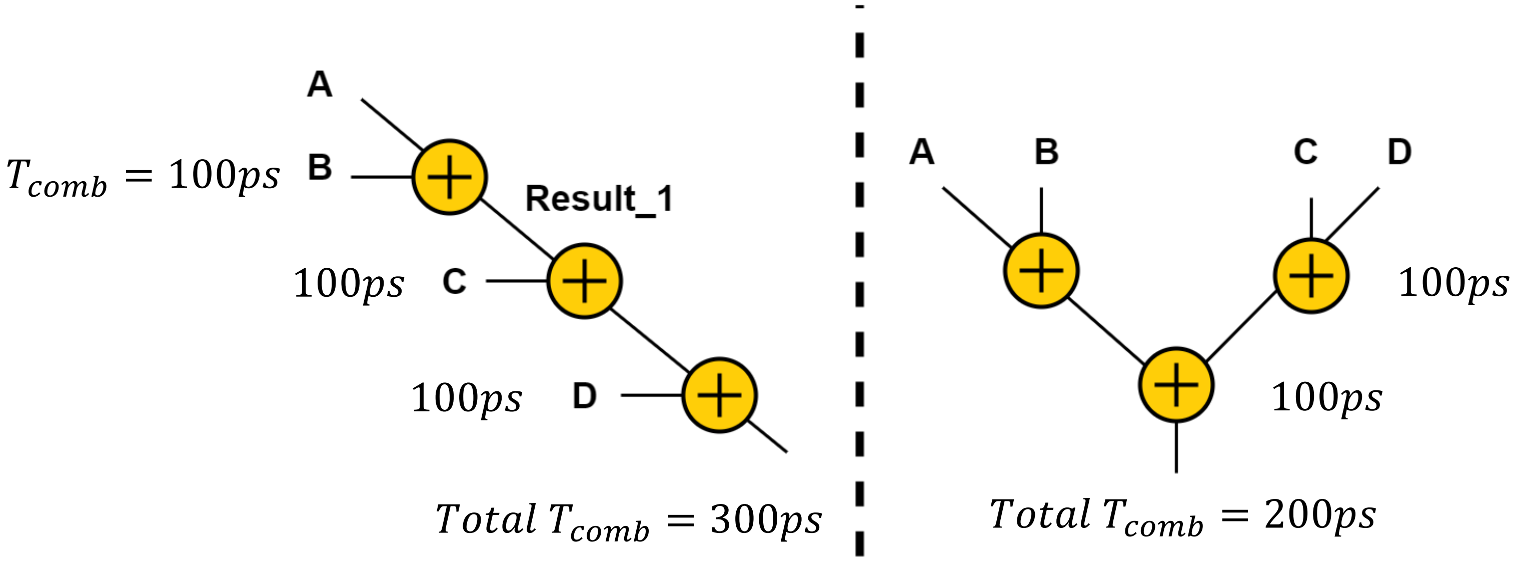
Reference : Kamanga, Isaack. Design Optimization of the 64-Bit Carry Look-Ahead Adder Based on FPGA and Verilog HDL

How to Fix a Setup Violation – Sol. 5

Optimizing the RTL Code

$$T_{launch\_edge} + T_{launch\_latency} + T_{comb} < T_{capture\_edge} - T_{setup} + T_{capture\_latency}$$

- The way the RTL is written affects the structure of the logic gates
- The example below shows 2 circuits that perform the same functionality however the on the right creates the adder in a chain fashion resulting in a delay of 3 adders in series while the one on the right is made in a parallel tree fashion and only has a delay of 2 series adders

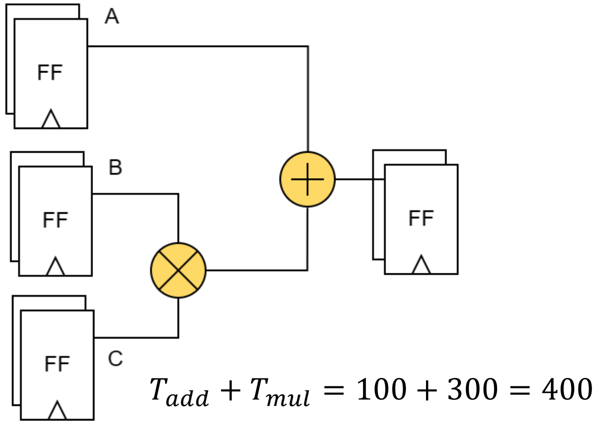


How to Fix a Setup Violation – Sol. 6

$$T_{launch\_edge} + T_{launch\_latency} + T_{comb} < T_{capture\_edge} - T_{setup} + T_{capture\_latency}$$

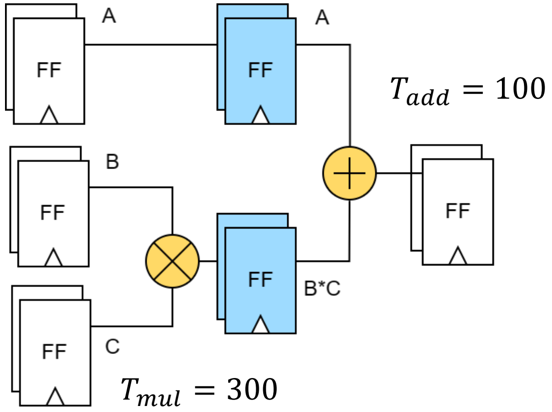
Pipelining

- The most common way to fix setup in RTL design is to add pipeline registers.
- The idea of pipelining is to split a large T_{comb} into multiple clock cycles.
- For example, to implement the equation $A + B * C$, one can do all the operations in one cycle or do the multiplication in one cycle then the addition in the next cycle as shown in the diagram
- **The disadvantages of pipelining is:**
  - More area due to the pipeline registers
  - More latency. Instead of finishing the operation in one cycle we finish it in multiple cycles.
  - Synchronization. Since the data is delayed by the pipeline registers, the downstream logic that will receive the data have to account for this delay. Notice also how we needed to add pipeline on A as well to synchronize A_1 with $B_1 * C_1$ otherwise we would have added A_2 from next sample to $B_1 * C_1$



$T_{add} + T_{mul} = 100 + 300 = 400$

Without Pipelining



$T_{add} = 100$

$T_{mul} = 300$

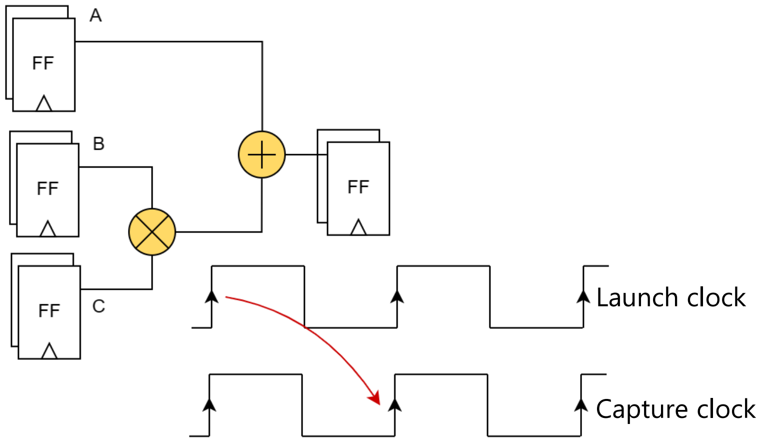
With Pipelining

How to Fix a Setup Violation – Sol. 7

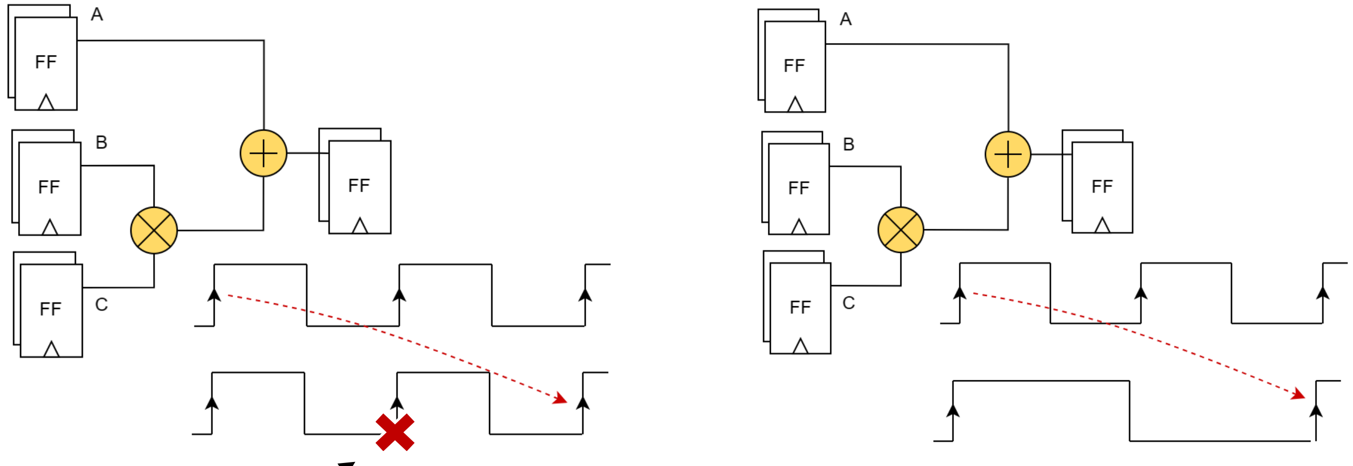
$$T_{launch\_edge} + T_{launch\_latency} + T_{comb} < T_{capture\_edge} - T_{setup} + T_{capture\_latency}$$

Multi Cycle Path (MCP)

- This method has some similarity to pipelining. Similarly, we will let the combinational path finish in multiple cycles.
- The difference is we won’t add pipeline registers. Instead, we will capture the data at another capture clock edge
- **This can be done in 2 ways[1]:**
    - Use a control circuit to mask the 1st capture edge and allow another one.
    - Use a divided clock for the capture FF as shown in the diagram below



Single Cycle



Mask this edge
with control logic

Multi Cycle Path

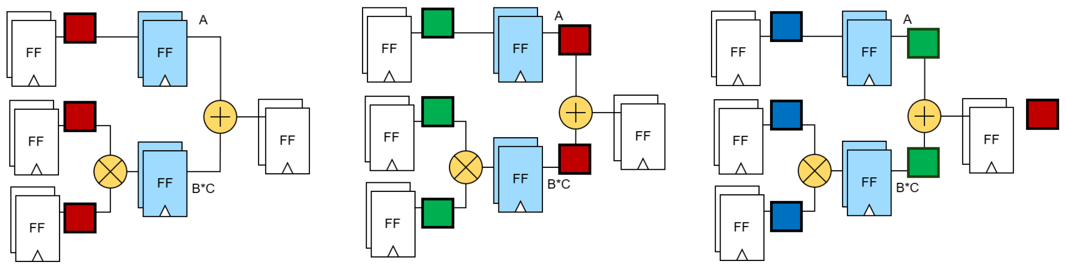
[1]: You need to inform the STA tool that you will mask the 1st edge since the tool has no knowledge about the functionality of the circuit. This is done using the “set_multicycle_path” command
https://docs.amd.com/r/2021.2-English/ug903-vivado-using-constraints/Multicycle-Paths

How to Fix a Setup Violation – Sol. 7

$$T_{launch\_edge} + T_{launch\_latency} + T_{comb} < T_{capture\_edge} - T_{setup} + T_{capture\_latency}$$

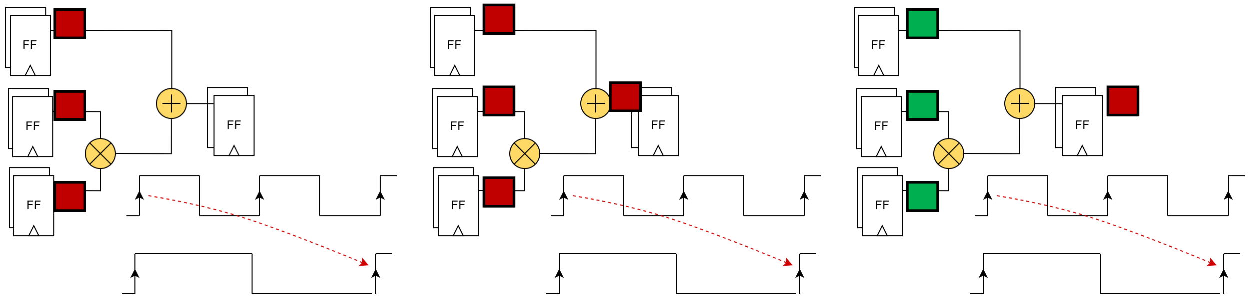
Multi Cycle Path vs Pipelining

- At first it might appear that multi cycle path and pipelining are the same. But a deep look shows the big difference
- **In the case of pipelining:**
  - In the 1st cycle A,B,C enters the 1st stage of the pipeline. In the 2nd cycle A,B,C enters the 2nd stage while a new sample enters 1st stage of the pipeline
  - We receive an output every clock cycle and the added latency due to the pipeline registers affects us at the beginning only
- **In the case of MCP:**
  - In the 1st cycle A,B,C enters the circuit. In the 2nd cycle, the circuit is still busy and we can't insert a new sample until it finishes.
  - We receive an output every 2 clock cycles
- This shows that pipelining fix setup and have high processing speed while MCP slows down the processing speed
- You can think of MCP as reducing the clock frequency but selectively in parts of the circuit and not on the entire circuit



1st cycle 2nd cycle 3rd cycle

Pipelining



1st cycle 2nd cycle 3rd cycle

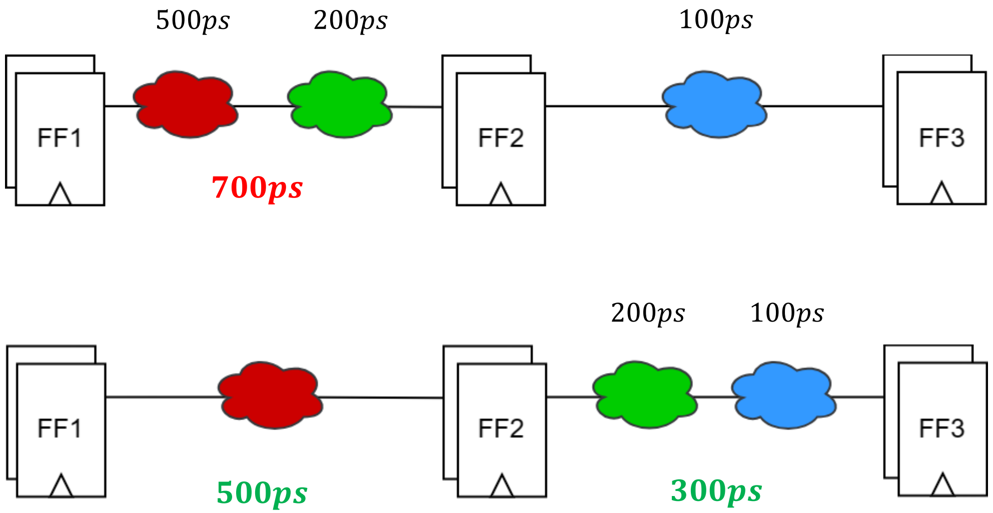
Multi Cycle Path

How to Fix a Setup Violation – Sol. 8

$$T_{launch\_edge} + T_{launch\_latency} + T_{comb} < T_{capture\_edge} - T_{setup} + T_{capture\_latency}$$

Retiming

- In this method if T_{comb} is large to fit in the clock cycle, we split the logic and move part of it to another cycle.
- **Consider the example below:**
  - The **red** and **green** logic combined make a $T_{comb} = \mathbf{700}ps$ which causes a setup violation.
  - We move the **green** logic to the next clock cycle to be combined with the **blue** logic.
  - This reduces T_{comb} between FF1 and FF2 to $\mathbf{500}ps$ instead of $\mathbf{700}ps$ which passes setup.
  - But increases T_{comb} between FF2 and FF3 to $\mathbf{300}ps$ instead of $\mathbf{100}ps$ but this is okay because it also passes setup. If the **blue** logic was big this method won't work



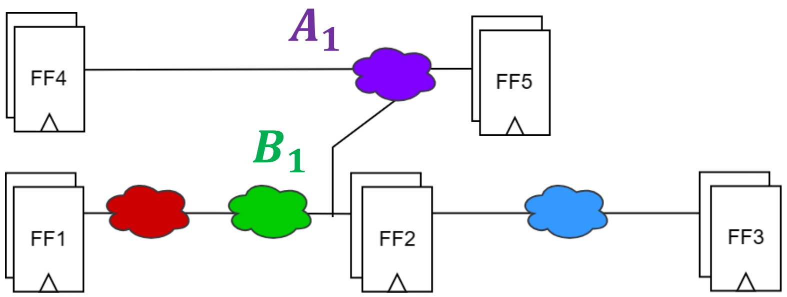
/amradelm

How to Fix a Setup Violation – Sol. 8

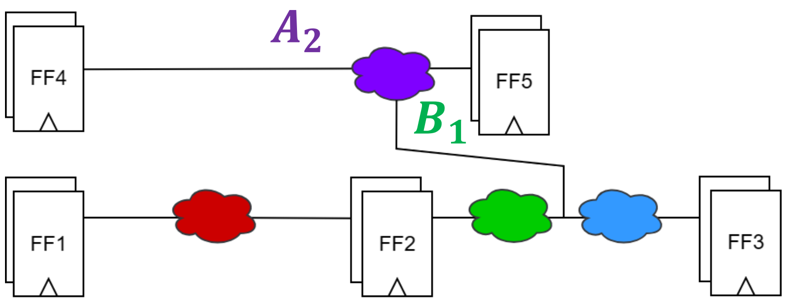
$$T_{launch\_edge} + T_{launch\_latency} + T_{comb} < T_{capture\_edge} - T_{setup} + T_{capture\_latency}$$

Retiming

- **Retiming can be done manually by the RTL designer or automatically by the synthesis tools**
  - In the example below, the **purple** logic takes as input A and B. If we move the **green** logic to the next cycle, we get B one cycle later than what was expected. When we wait for this one cycle, $\boldsymbol{A_1}$ will be gone and a new $\boldsymbol{A_2}$ will arrive which will get computed with sample $\boldsymbol{B_1}$. This will break the functionality of the circuit
  - Synthesis tools will avoid any retiming that breaks the functionality as this example did.
  - The RTL designer has full control over the code so he can fix this issue by, for example, adding a pipeline register before the purple logic to delay it one cycle and handle any new issues that will appear due to this added register
  - Hence, the RTL designer can do more aggressive retiming compared to the synthesis tools but with extra effort.



1st Cycle



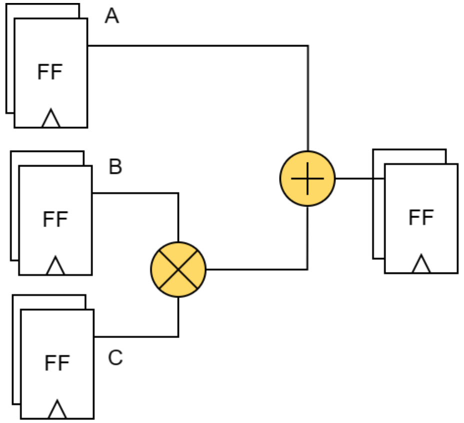
2nd Cycle

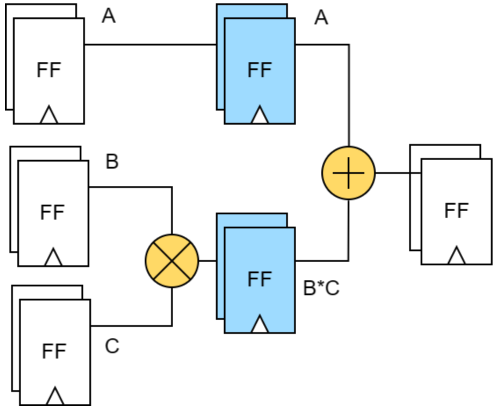
How to Fix a Setup Violation – Sol. 8

$$T_{launch\_edge} + T_{launch\_latency} + T_{comb} < T_{capture\_edge} - T_{setup} + T_{capture\_latency}$$

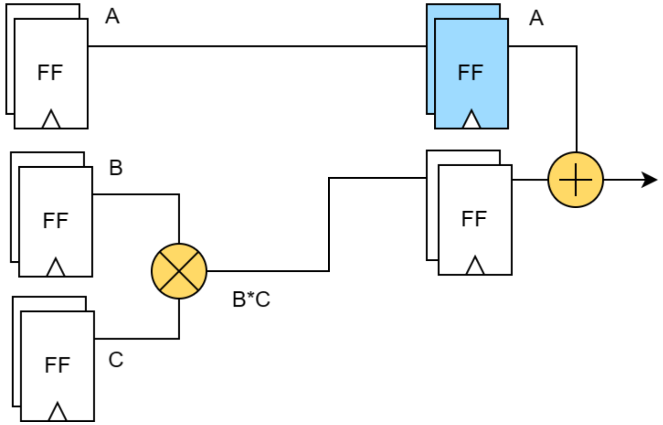
Retiming + Pipelining

- The previous example shows how retiming can be combined with pipelining.
- **Lets Consider the same example of $A + B * C$**
  - We can move the adder to the next clock cycle if there is margin there.
  - However, we get the same issue in the previous slide that A is not synchronized with B*C. So we add a pipeline register.
  - This way we fixed the setup violation and saved the area of the $B * C$ pipeline registers





Pipelining



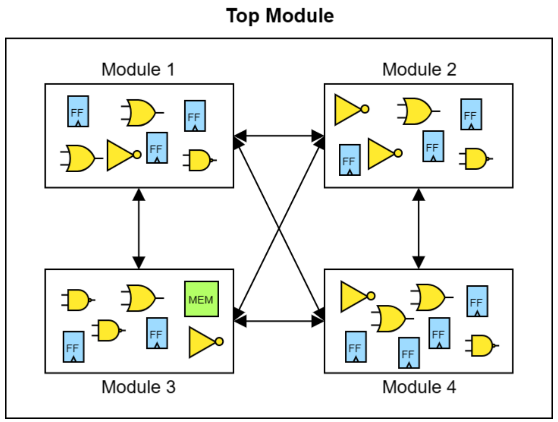
Pipelining + Retiming

How to Fix a Setup Violation – Sol. 9

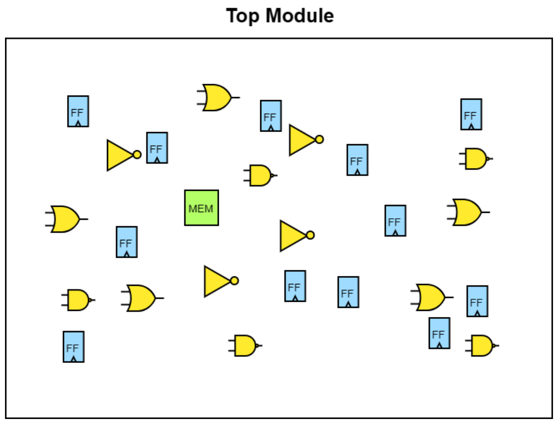
$$T_{launch\_edge} + T_{launch\_latency} + T_{comb} < T_{capture\_edge} - T_{setup} + T_{capture\_latency}$$

Optimizing Synthesis

- Synthesis tools have lots of features and switches that the engineer can use to enhance the timing and control the trade-offs between the PPA metrics.
- This topic is very large and needs a tutorial on its own, so we will demonstrate just a few of what can be done.
  - **Increase the timing effort**_: Most synthesis tools have switches that controls the effort the tool will put to fix a certain PPA metric or to do a certain optimization. Higher effort leads to better optimization but higher runtime while a lower effort leads to less optimization but better runtime.
  - **Decrease or disable area and power efforts** : Area and power optimizations usually degrade the timing of the circuit. Reducing the effort of these optimizations or disabling them all together may enhance the timing but worsen the area and power of your chip
  - **Enable Flattening** : The RTL code consists of several modules connected to each other. By default synthesis tools will synthesize each module separately and then connect them together in the top module, thus preserve the hierarchy and boundaries between the modules. Another approach is to remove the module boundaries and make all cells in one hierarchy. This is called flattening and generally produce better timing result[1]



No Flattening



With Flattening

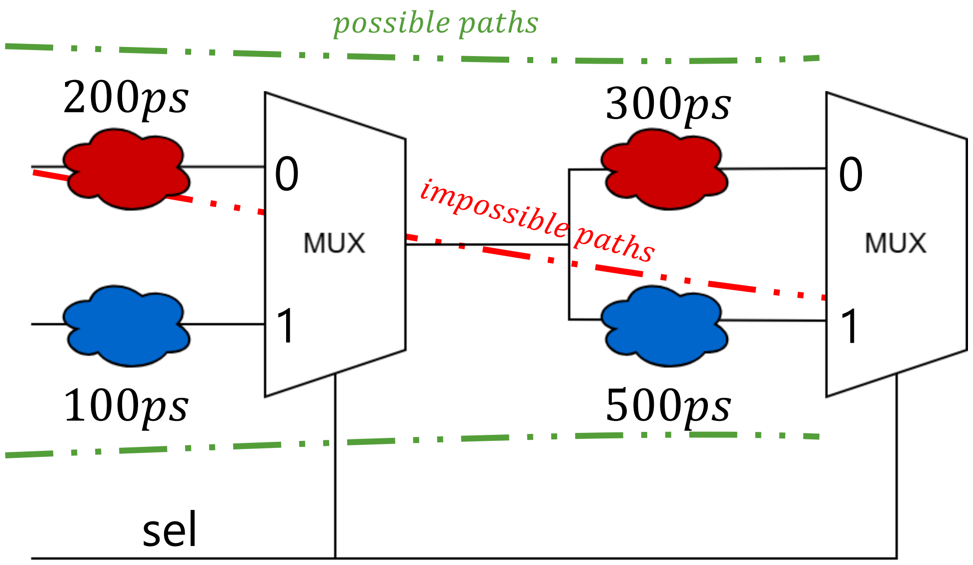
[1] : Flattening makes verification more difficult because the module boundaries are removed which makes tracing signals and referencing cells more difficult.

How to Fix a Setup Violation – Sol. 10

Applying False Paths in the Constraints

$$T_{launch\_edge} + T_{launch\_latency} + T_{comb} < T_{capture\_edge} - T_{setup} + T_{capture\_latency}$$

- False paths are timing paths that can't possibly occur due to the logic of the circuit
- **Consider the example below:**
  - Both muxes have the same select signal. This means we have 2 possible timing paths. The one going through both **red** logics $(200 + 300 = 500ps)$ and the one going through both **blue** logics $(100 + 500 = 600ps)$
  - The paths going through a **red** logic then a **blue** logic $(200 + 500 = 700ps)$ or **blue** logic then **red** logic $(100 + 300 = 400ps)$ is impossible to happen.
  - Unless we instruct the tool to ignore these false paths, they will be considered for timing analysis leading to the large T_{comb} of the red to blue path which will violate setup.
- If we don't apply correct constraints on these paths, not only do we get fake setup violations, but we hinder the synthesis and PnR tools ability to optimize the other real violating timing paths, because the tools apply extreme optimizations only on the critical and worst paths and it won't consider the less critical paths for these optimizations unless they solve the most critical ones.

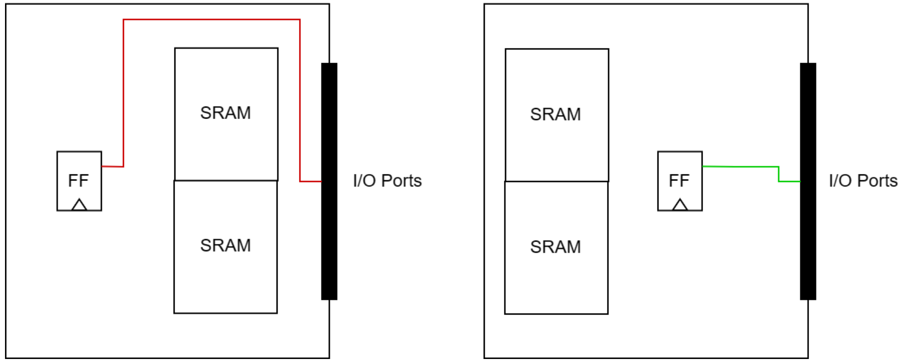


How to Fix a Setup Violation – Sol. 11

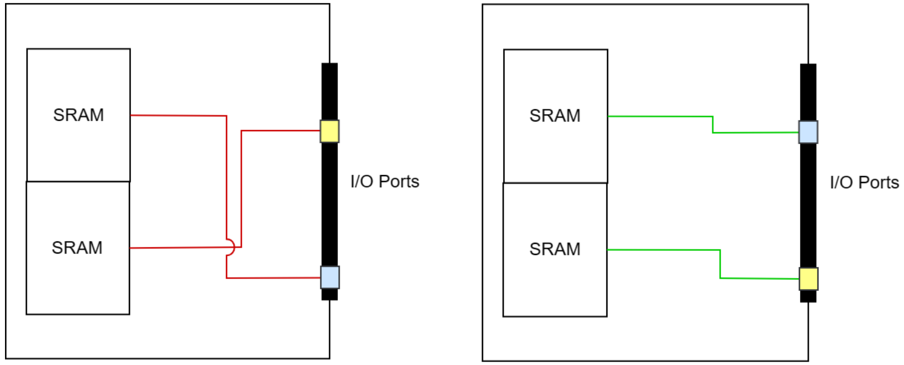
$$T_{launch\_edge} + T_{launch\_latency} + T_{comb} < T_{capture\_edge} - T_{setup} + T_{capture\_latency}$$

Optimizing the Floorplan

- Floorplaning is the 1st step in the PNR flow and involves things like creating the chip size and boundaries, manually placing the major blocks (analog, SRAM, etc) in the chip, and placing the chip ports
- **Here are some of the things that affects the setup in the circuit**
  - A small chip area might cause the cells to get closer to each other and closer to the ports which in turn will reduce the wire delays. However, if the size is too small several issues will appear such as big voltage drop, cell congestion, routing detours, crosstalk, etc[1].
  - The placement of the major blocks in the chip affects the timing. The example on the left shows how the placement of the SRAMs near the IO ports might block the standard cells from being placed near their relevant ports. Not only that but they will block the routing resulting in longer wire delays to go around them.
  - The placement of the ports also affect the timing. The example on the right shows how a bad placement of the ports can lead to long wire delays and buffering which will worsen T_{comb}



Block Placement



Port Placement

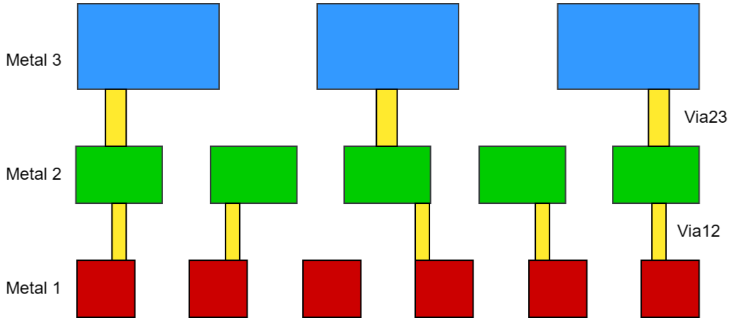
[1]: We won't discuss these issues because they are out of the scope of this document. You are advised to research these topics to get a better understanding of the slides

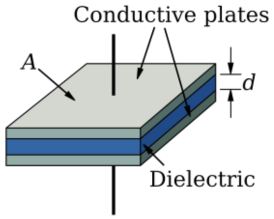
How to Fix a Setup Violation – Sol. 12

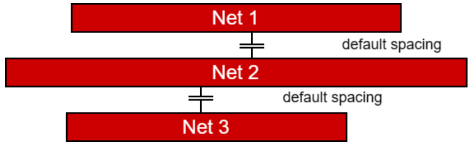
$$T_{launch\_edge} + T_{launch\_latency} + T_{comb} < T_{capture\_edge} - T_{setup} + T_{capture\_latency}$$

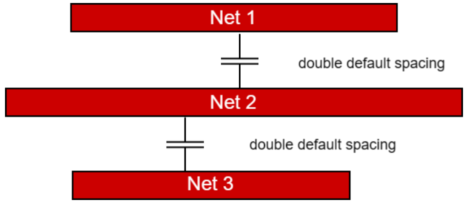
Optimizing the wire delay

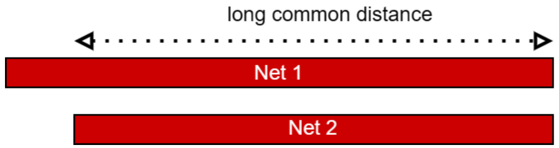
- In part 1 we showed how a signal propagating through an RC circuit will have a delay proportional to the resistance and the capacitance. Hence, to reduce this delay we need to reduce the resistance and capacitance of the wire.
- This will also decrease the load cap of the cell that drives the wire which will speed up the cell too.
- **Reducing the resistance $R = \frac{\rho L}{A}$:**
  1. Reducing the length L of the wire will reduce the delay. We showed some examples on how to reduce it using a better floorplan.
  2. Increasing the width will decrease the delay. Higher metal layers have higher default width and also bigger thickness hence larger area A. PNR tools will use these higher layers for long and critical nets to reduce their delay. The PNR engineer can manually move the wires to higher layers during ECO or apply non-default routing rules (NDR) on these nets to make the router route them in higher layers
- **Reducing the capacitance $C = \frac{\epsilon A}{d}$**
  1. Increasing the spacing d by moving the two wires aways from each other will reduce the capacitance between them. We can apply NDR on specific nets to tell the router that we want no nets to get routed very close to these nets
  2. Reducing the common distance. When two wires move along each other for a long distance the common area A will be big leading to bigger capacitance. We can move one of the two wires to another layer to reduce the delay

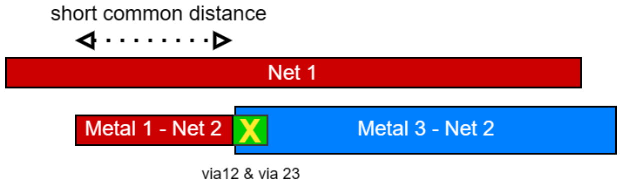










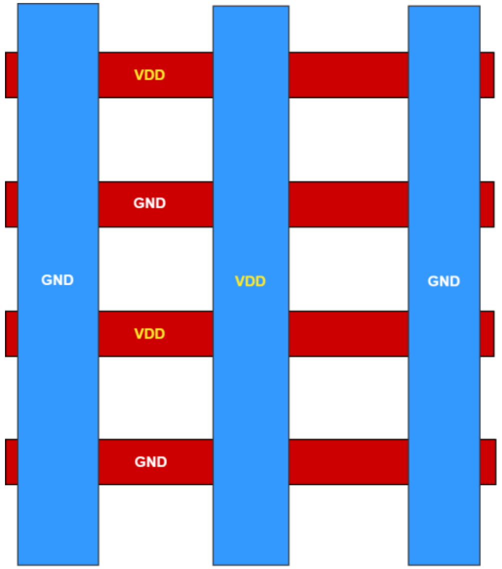


How to Fix a Setup Violation – Sol. 13

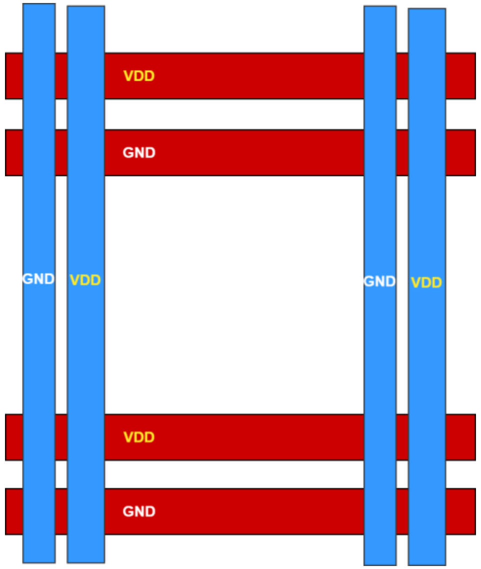
$$T_{launch\_edge} + T_{launch\_latency} + T_{comb} < T_{capture\_edge} - T_{setup} + T_{capture\_latency}$$

Relaxing the Power Grid

- The power grid is the metal connection that delivers the power from higher metal layers down to the standard cells
- We showed how the wire delay is affected by things like spacing and width, etc. A wide and compact power grid will leave few routing resource for the signal nets leaving no option for increasing spacing or width.
- However, relaxing the power grid will increase the resistance of the power network causing bigger voltage drop. So the PNR designer has to trade-off between enhancing timing and fixing voltage drop.



Compact PG



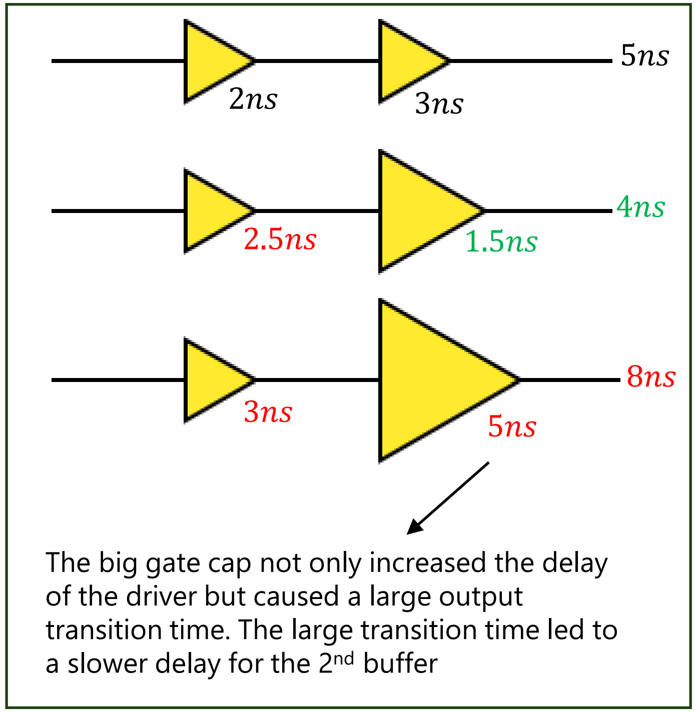
Relaxed PG

How to Fix a Setup Violation – Sol. 14

$$T_{launch\_edge} + T_{launch\_latency} + T_{comb} < T_{capture\_edge} - T_{setup} + T_{capture\_latency}$$

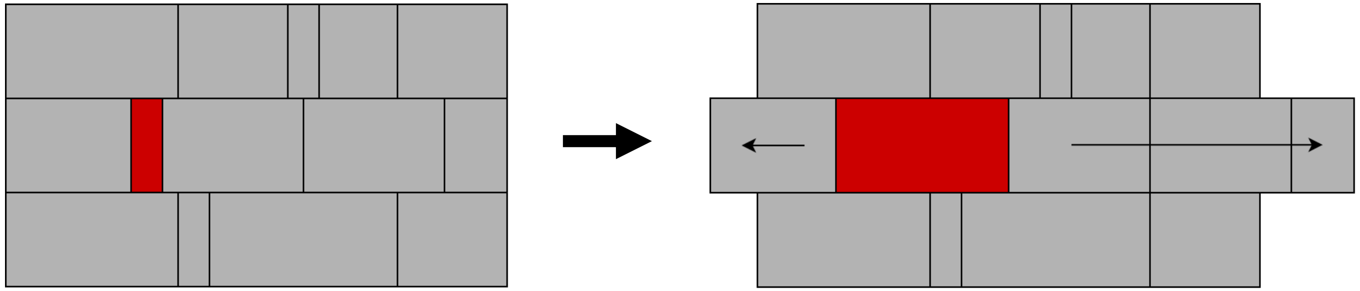
Upsizing

- We showed in part 1 how the MOSFET size affects the propagation delay of the cell. So to fix setup we can use larger cells that has less propagation delay
- **There are several considerations when doing this method:**
  - Bigger cells means more area and power consumption
  - Bigger cells has larger gate capacitance. This will slow down the cell that drives them because it now has larger load capacitance. The enhancement of upsizing the cell should overcome the slow down of the driving cell.
  - Since big cells consume more power they are likely to cause big voltage drop on the cells around them.
  - During ECO flow there might not be enough area to accommodate the bigger cell which require you to move the cells around it and then reroute the nets to their pins. The moving of the cells and the reroute could worsen the timing for these cells



The big gate cap not only increased the delay of the driver but caused a large output transition time. The large transition time led to a slower delay for the 2nd buffer

Effect of Upsizing on the Driver and Load



Before Upsizing

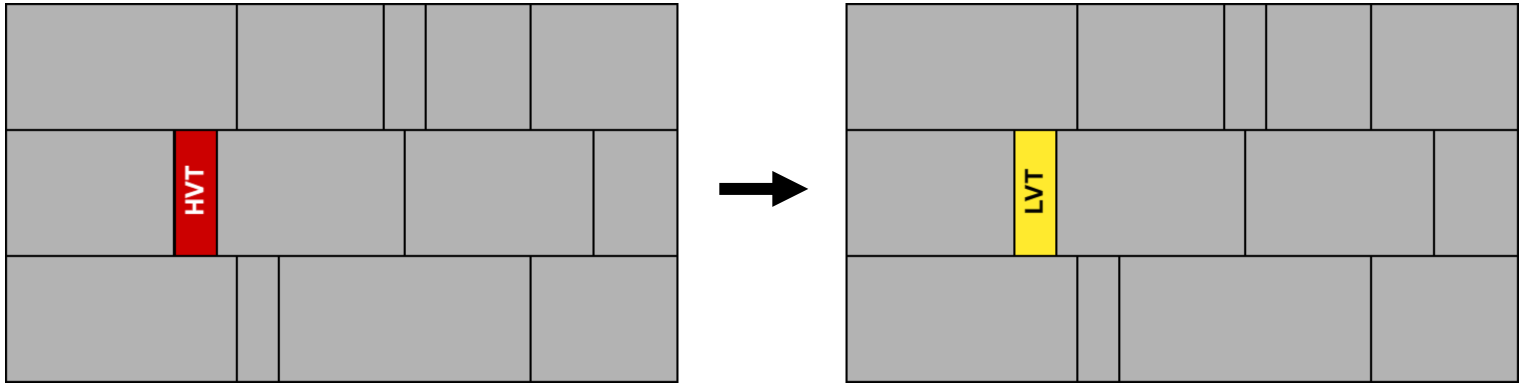
After Upsizing

How to Fix a Setup Violation – Sol. 15

$$T_{launch\_edge} + T_{launch\_latency} + T_{comb} < T_{capture\_edge} - T_{setup} + T_{capture\_latency}$$

MTCMOS

- Similarly the threshold voltage V_{th} of the MOSFET affects the propagation delay of the cell. So to fix setup we can use low V_{th} that has less propagation delay but this will increase the leakage power consumption.
- Synthesis and PNR tools allow you to apply a limit on the percentage of low V_{th} in your chip. Relaxing this limit will lead to a better overall timing[1].
- The gain from changing the flavor (threshold) is usually less than that of upsizing the cell. However, changing the cell flavor won't increase the cell area hence no moving of the cells or rerouting is required. This is why changing the flavor is the first go-to method for PNR engineers during ECO.



Before Changing Flavor

After Changing Flavor

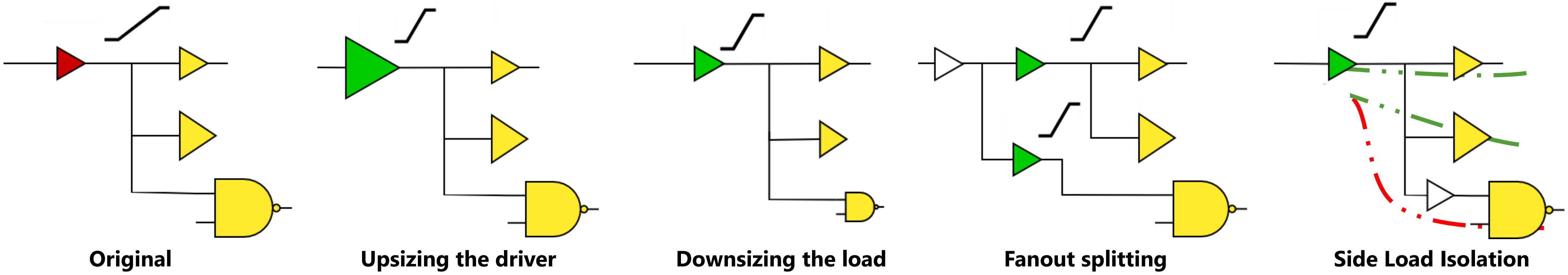
[1] : You need to be careful when relaxing the limit because the tool might resort to the easy solution of using low V_{th} cells and ignore other optimizations in the logic and wire delay leaving you with big leakage power consumption.

How to Fix a Setup Violation – Sol. 16

$$T_{launch\_edge} + T_{launch\_latency} + T_{comb} < T_{capture\_edge} - T_{setup} + T_{capture\_latency}$$

Increasing the Driving Strength

- When we discussed upsizing we showed that when a cell drives a large load capacitance its output transition time gets slower which in turn will slow down the load cells. Increasing the driver strength will enhance the transition time which in turn will enhance the load cells delay
- **There are several ways to enhance the driving strength**
  - Upsizing the driver cell : Bigger cells produce larger current and hence charge the load capacitance faster. This method combine the benefit of speeding up the driver by upsizing and the benefit of speeding up the load cells because they see a better input transition time.
  - Downsizing the load cells : this will decrease the load capacitance of the driver which will speed up the propagation and transition time which in turn will speed up the load cells. However, smaller cells has larger delay, so for this method to work the gain from enhancing the driving strength should overcome the increase in delay due to downsizing
  - Fanout splitting : Instead of one cell driving all the fanout we can duplicate the driver and split the fanout among them as shown in the diagram. But note that the driver of the driver is now seeing double the load cap which increases it's delay. So you have to balance things to make the overall gain overcome the increase in delay
  - Side load isolation : Add a small buffer that isolates a large load from the driver. In the example shown, the driver now sees the small cap of the buffer instead of the large cap of the large NAND. This will fix the **green** paths but will worsen the **red** path because the small buffer will add a delay that increases the overall delay of the **red** path. For this method to work, the **red** path should be passing setup check and have good a margin to accommodate the increase in delay

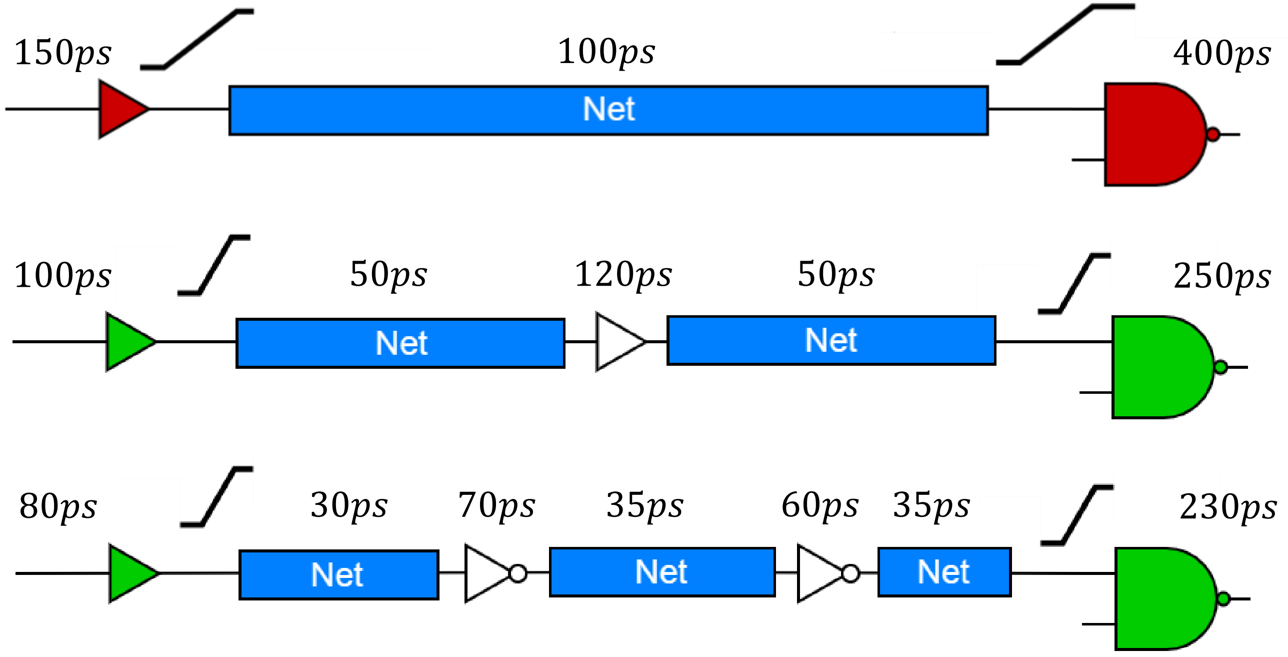


How to Fix a Setup Violation – Sol. 17

$$T_{launch\_edge} + T_{launch\_latency} + T_{comb} < T_{capture\_edge} - T_{setup} + T_{capture\_latency}$$

Breaking up Long Nets

- When a cell drives a very long wire with big capacitance it will have bad propagation and transition times. By breaking the long wire with buffers the overall enhancement could overcome the delay of the added buffers
- If the wire is very long we can split it with an inverter pair instead of a buffer. This is better because the delay of an inverter is less than that of a buffer of the same size[1]. This way we get more cuts in the wire (less load cap for each cell) with roughly the same delay of the added buffer



$Total\ T_{comb} = 650ps$

$Total\ T_{comb} = 570ps$

$Total\ T_{comb} = 540ps$

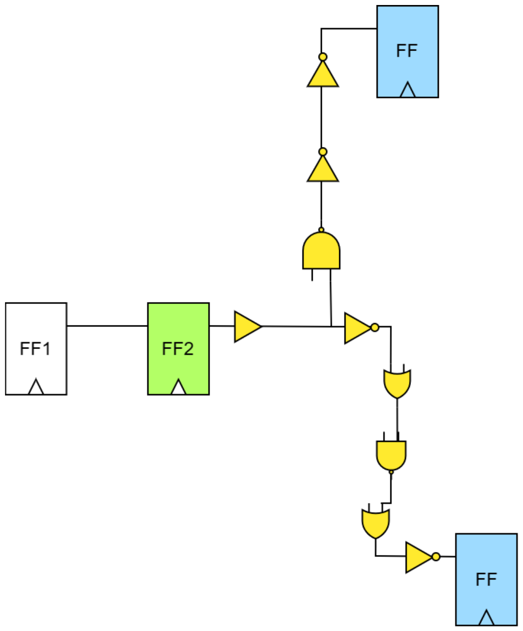
[1]: Buffers are basically 2 inverters connected in series

How to Fix a Setup Violation – Sol. 18

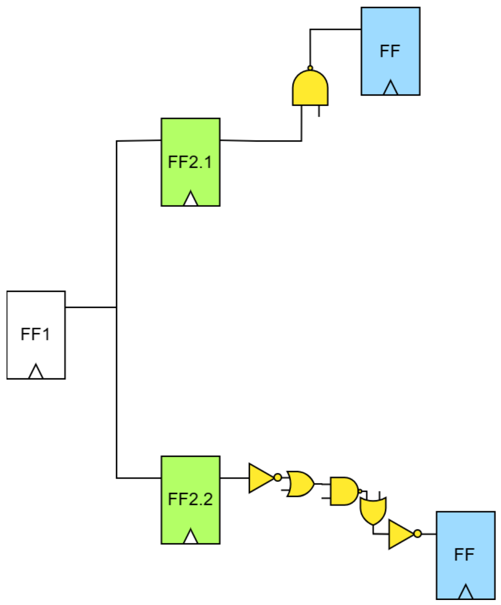
$$T_{launch\_edge} + T_{launch\_latency} + T_{comb} < T_{capture\_edge} - T_{setup} + T_{capture\_latency}$$

Register Duplication

- By duplicating registers, the timing paths can be shortened, reducing the wire and cell propagation delays.
- **Consider the example on the right :**
  - By duplicating the green registers we managed to move each copy near one of the blue register
  - This first, reduces the wire length between the green and blue registers and second, allows us to remove the buffers and inverter pairs on the nets and both reduce the total combinational delay
  - This shows that this method becomes more useful when the capture registers (the blue ones) are placed far away from each other in the chip.
  - However, FF1 now drives double the fanout so the delay of the timing path between FF1 and FF2 is increased. We need to make sure this increase doesn't cause the path to violate setup timing.
- Duplication can be done manually in the RTL or automatically by the synthesis and PnR tools.



Before Duplication



After Duplication

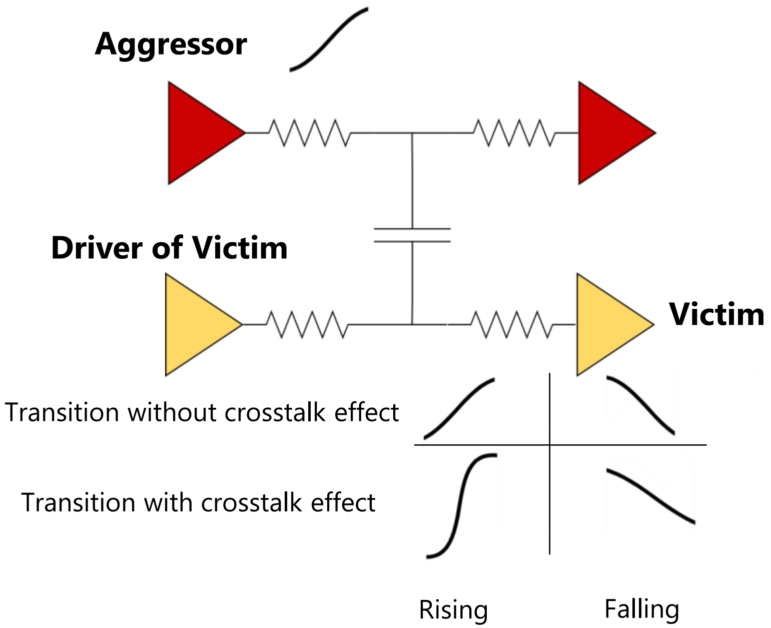
More details : https://community.intel.com/t5/FPGA-Wiki/Register-Duplication-for-Timing-Closure/ta-p/735917

How to Fix a Setup Violation – Sol. 19

$$T_{launch\_edge} + T_{launch\_latency} + T_{comb} < T_{capture\_edge} - T_{setup} + T_{capture\_latency}$$

Reducing Crosstalk

- When we discussed wire delays we showed that there is a capacitance between any two wires close to each other. This capacitance is called the coupling capacitance.
- When one of the two wires switches from 0->1 or 1->0, the other wire switches too with the same polarity. We call the first the aggressor and the second the victim.
- If the aggressor was switching and at the same time the victim was switching with the same polarity, the aggressor will speed up the input transition time of the victim. This will speed up the propagation delay of the victim.
- If the victim was switching with a different polarity than the aggressor, this will slow down the transition time and so slow down the propagation delay of the victim and therefore increase T_{comb}.
- **To decrease the effect of crosstalk and speed up the cell delay:**
  - Reduce the coupling capacitance by increasing the spacing between the wires. This combines the effect of wire delay optimizations and reducing crosstalk.
  - Shielding the wires of victim net with VSS wires will block the crosstalk.
  - Downsizing the aggressor cell will reduce its effect on the victim.
  - Upsizing the driver of victim will make it overcome the aggressor effect.

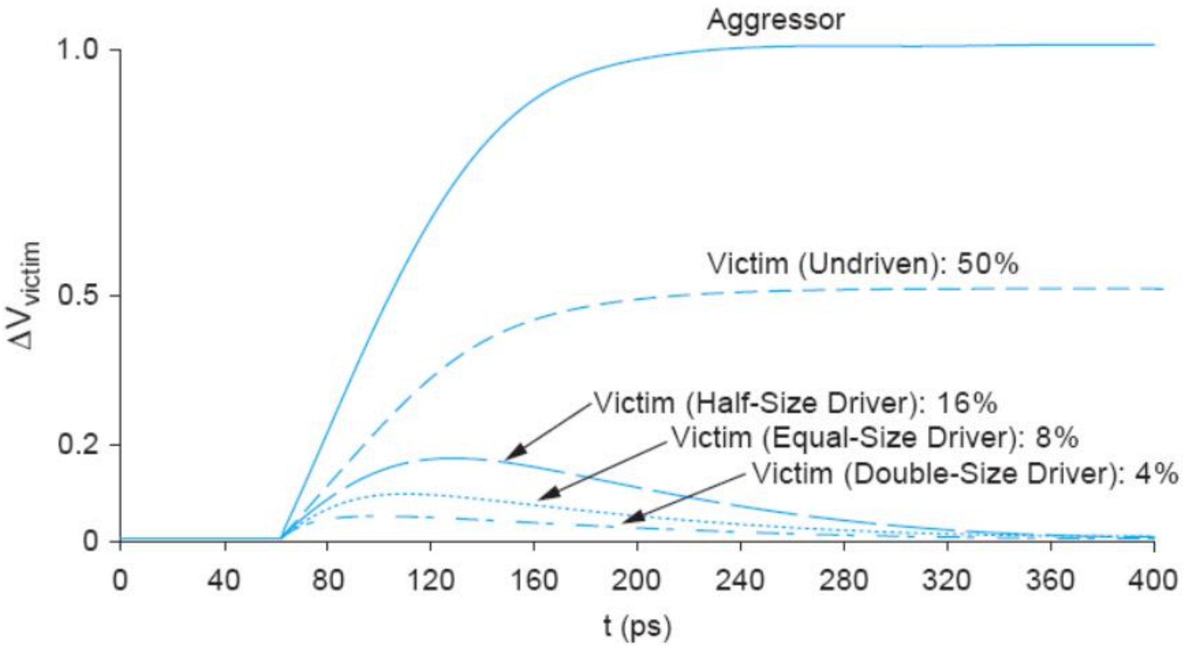

Aggressor
Driver of Victim
Victim
Transition without crosstalk effect
Transition with crosstalk effect
Rising
Falling

How to Fix a Setup Violation – Sol. 19

Reducing Crosstalk

$$T_{launch\_edge} + T_{launch\_latency} + T_{comb} < T_{capture\_edge} - T_{setup} + T_{capture\_latency}$$

- The image below shows an aggressor switching from 0->1 vs the victim transition
- We can see that the stronger the driver, the less the effect of the crosstalk.



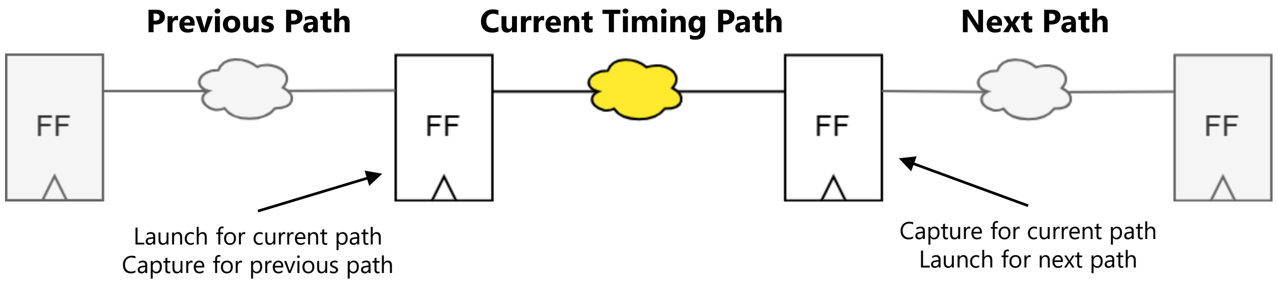
Reference : CMOS VLSI Design - https://pages.hmc.edu/harris/cmosvlsi/4e/index.html

How to Fix a Setup Violation – Sol. 20

$$T_{launch\_edge} + T_{launch\_latency} + T_{comb} < T_{capture\_edge} - T_{setup} + T_{capture\_latency}$$

Local Skew

- So far we have been discussing methods that reduce T_{comb}. Now we will consider the launch and capture latencies $T_{launch\_latency}$ & $T_{capture\_latency}$
- From the setup equation we can see that decreasing $T_{launch\_latency}$ or increasing $T_{capture\_latency}$ will enhance the setup. The difference $T_{capture\_latency} - T_{launch\_latency}$ is called the **skew** and to fix a setup violation we can increase the skew
- To decrease the launch latency we can use any of the methods we discussed such as upsizing, changing flavor, etc
- To increase the capture latency we can use the opposite of the methods we discussed such as downsizing, changing flavor to high V_{th}, etc or by adding buffers.
- **Changing the skew to fix a timing path will affect the previous and next paths:**
  - The launch FF of the current timing path is the capture for the previous one. So if you decreased the launch latency to fix the current path you will also decrease the capture latency for the previous one which might cause it to violate setup. And the same applies to the next path.
  - In other words, you are borrowing some of the positive slack from the prev and next paths.
  - That's why before changing the skew you have to check if the other prev and next paths are passing timing with a good margin or not

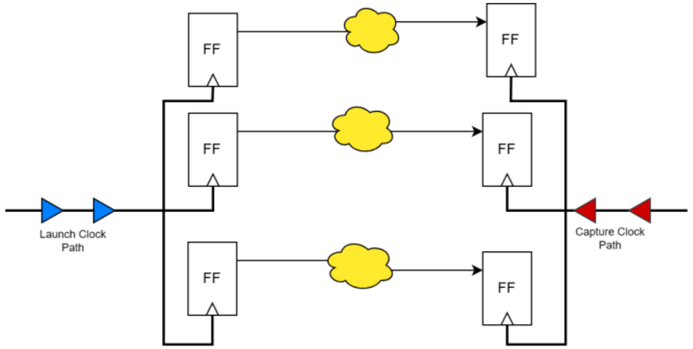


How to Fix a Setup Violation – Sol. 20

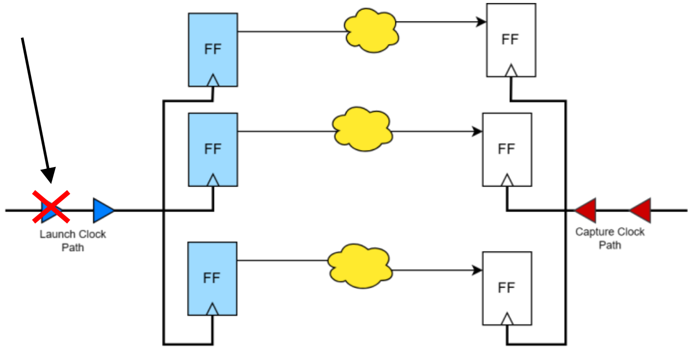
$$T_{launch\_edge} + T_{launch\_latency} + T_{comb} < T_{capture\_edge} - T_{setup} + T_{capture\_latency}$$

Local Skew

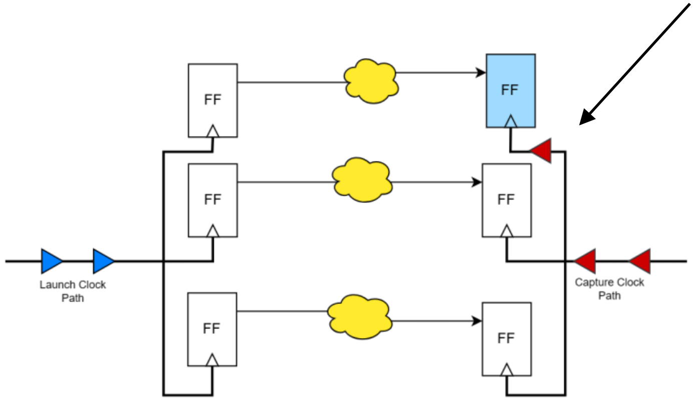
- In general, increasing the delay is a lot easier than decreasing it because we can simply add buffers. That's why ASIC engineers and PNR tools tend to focus on increasing the capture latency instead of decreasing the launch latency.
- **Another reason why increasing the capture latency is more favored :**
  - When the PnR tool build the clock tree network, usually multiple FFs are driven by the same clock buffer. If we try to modify the launch latency network to fix one timing path we will affect the other timing path that use the same clock buffer[1]
  - This is not the case for the capture clock network because we can add a buffer just in front of the clock pin of the FF while not affecting the rest of the FFs



Original



Decreasing Launch Latency
All blue FFs are affected



Increasing Capture Latency
Only the 1st blue FFs is affected

[1] : We don't want to affect the latencies of other timing paths because this may cause them to violate hold. More on this when we discuss hold.

Hold Timing Analysis

Hold Time

1 The waveform below shows the timing of 2 consecutive samples (**A** and **B**) going through the FFs

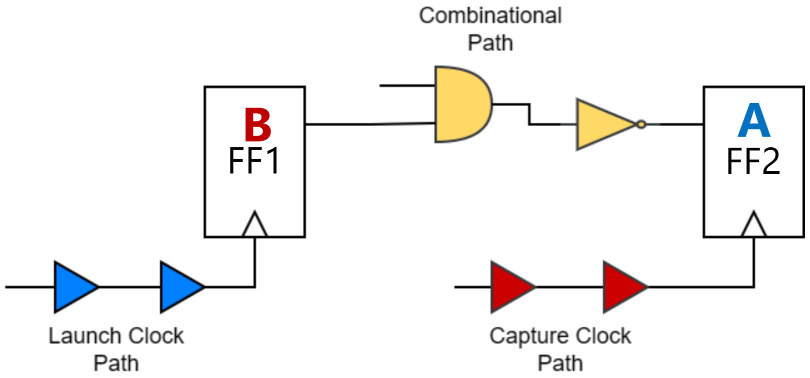
In order to avoid metastability, we want **A** to get captured and then remain stable at FF2 for an amount of time. we called this time the hold time T_{hold}

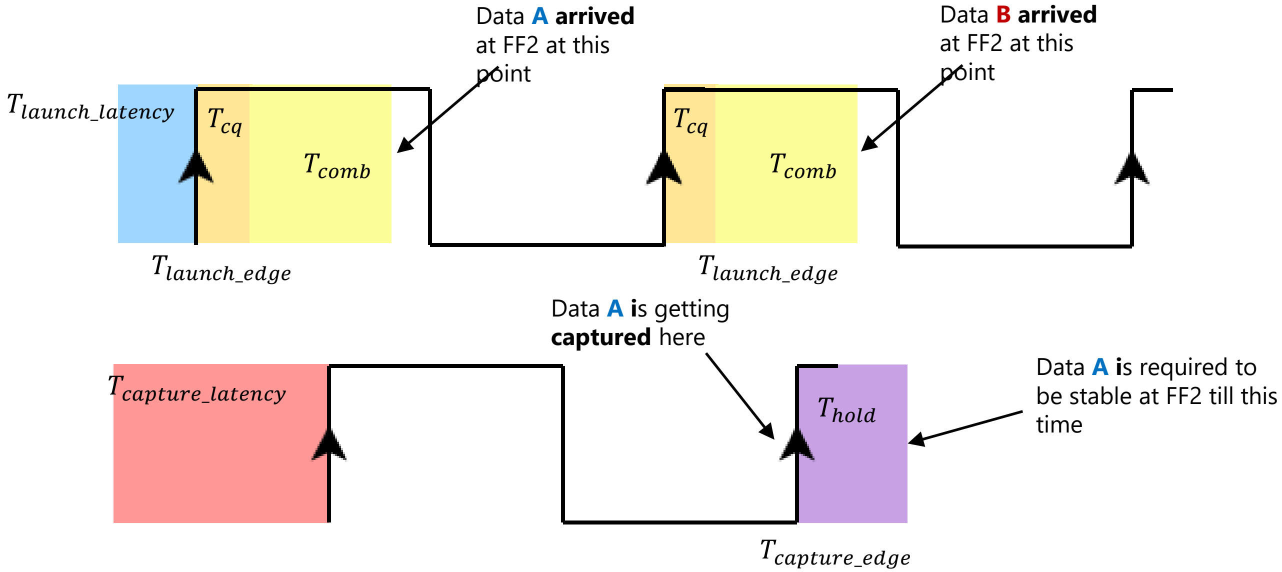
This means we want the arrival of **B** to come after the capturing and hold time of **A**

$$Launch + Delay \geq Capture$$

$$Arrival \geq Required$$

$$T_{launch\_edge} + T_{launch\_latency} + T_{comb} \geq T_{capture\_edge} + T_{capture\_latency} + T_{hold}$$





Data **A** **arrived** at FF2 at this point

Data **B** **arrived** at FF2 at this point

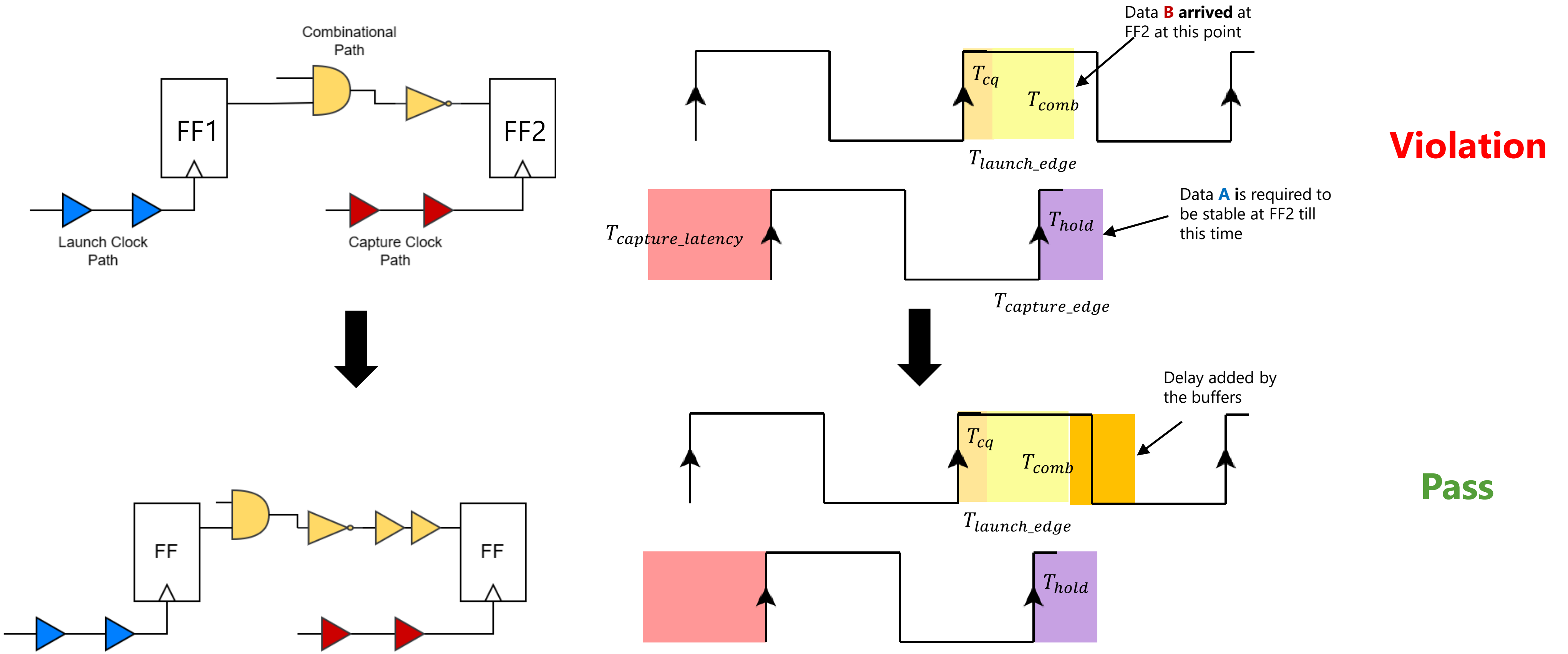
Data **A** is getting **captured** here

Data **A** is required to be stable at FF2 till this time

Hold Time

2 The example below violates this requirement because **B** arrived <u>before</u> **A** remained the necessary hold time

A quick solution is to insert buffers in the combinational path to increase T_{comb} and make **B** arrive after the required hold time



Data **B arrived** at FF2 at this point

Data **A** is required to be stable at FF2 till this time

Violation

Delay added by the buffers

Pass

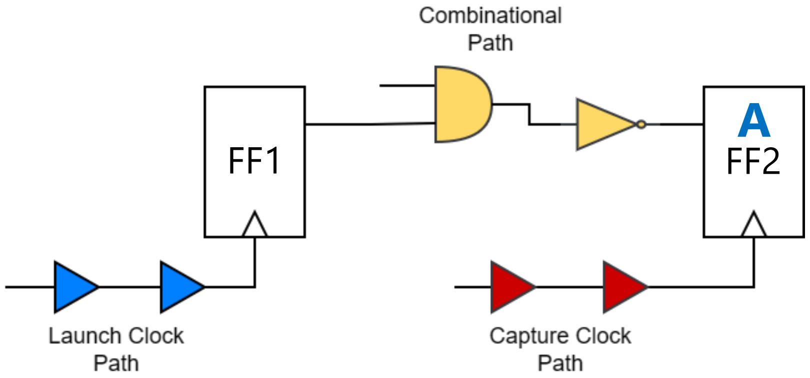
Hold Time

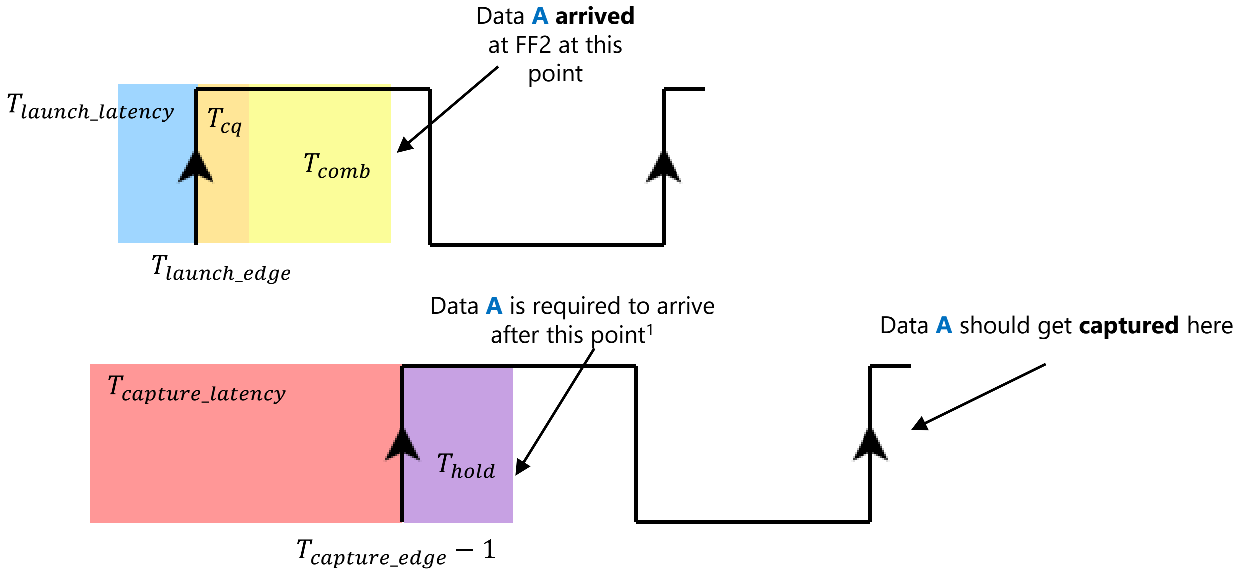
3 We also don't want **A** to be captured by an earlier edge as this will break the functionality.

$$Launch + Delay \geq Capture$$

$$Arrival \geq Required$$

$$T_{launch\_edge} + T_{launch\_latency} + T_{comb} \geq T_{capture\_edge} + T_{capture\_latency} + T_{hold}$$





Data **A arrived** at FF2 at this point

Data **A** is required to arrive after this point[1]

Data **A** should get **captured** here

[1]: Not only does A need to come after the earlier edge, it also needs to come after the hold time of that edge or it will cause metastability.

Example Timing Report

$T_{launch\_edge} \rightarrow$

$T_{launch\_latency} \rightarrow$

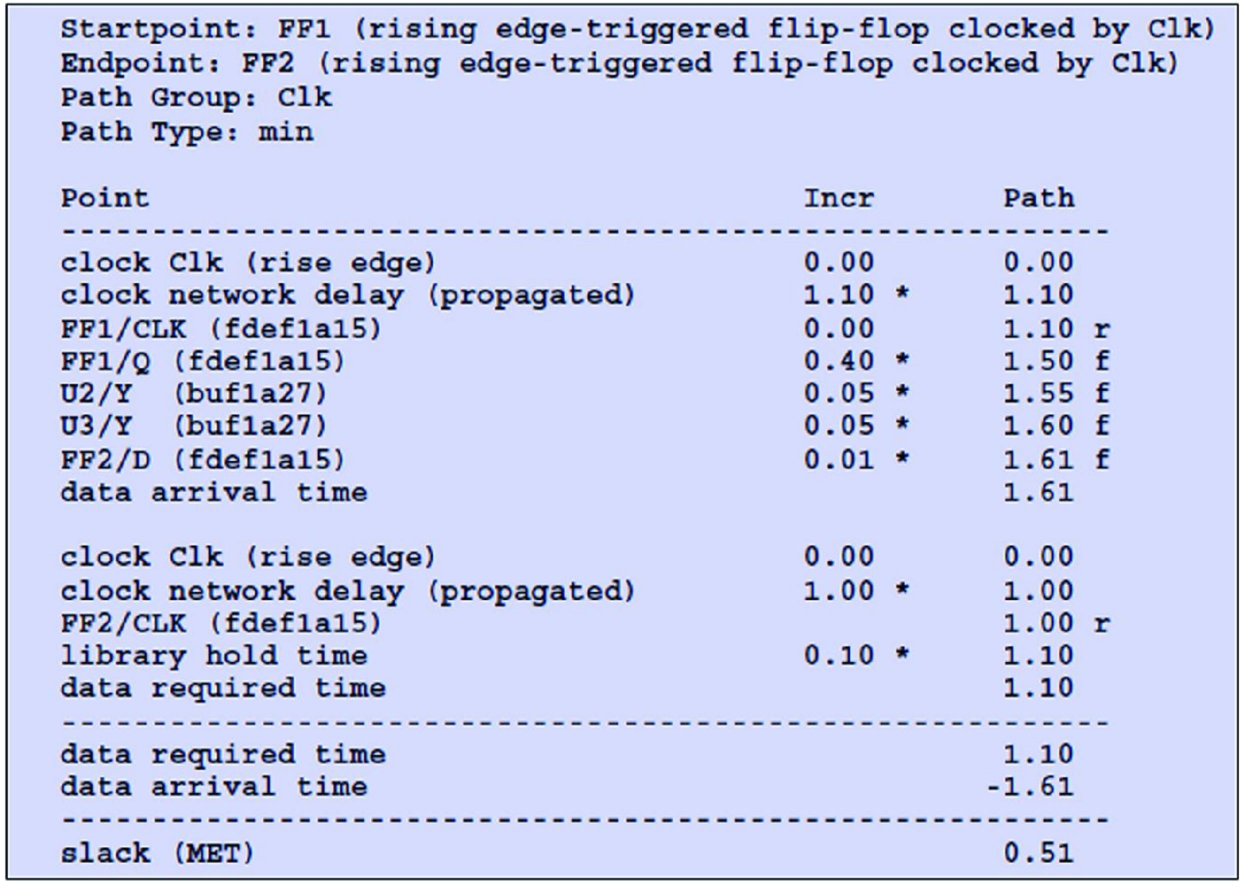
$T_{cq} \rightarrow$

$T_{comb}\{$

$T_{capture\_edge} \rightarrow$

$T_{capture\_latency} \rightarrow$

$T_{hold} \rightarrow$



```
Startpoint: FF1 (rising edge-triggered flip-flop clocked by Clk)
Endpoint: FF2 (rising edge-triggered flip-flop clocked by Clk)
Path Group: Clk
Path Type: min

Point                                       Incr       Path
-------------------------------------------------------------
clock Clk (rise edge)                       0.00       0.00
clock network delay (propagated)            1.10 *     1.10
FF1/CLK (fdef1a15)                          0.00       1.10 r
FF1/Q (fdef1a15)                            0.40 *     1.50 f
U2/Y  (buf1a27)                             0.05 *     1.55 f
U3/Y  (buf1a27)                             0.05 *     1.60 f
FF2/D (fdef1a15)                            0.01 *     1.61 f
data arrival time                                      1.61

clock Clk (rise edge)                       0.00       0.00
clock network delay (propagated)            1.00 *     1.00
FF2/CLK (fdef1a15)                                     1.00 r
library hold time                           0.10 *     1.10
data required time                                     1.10
-------------------------------------------------------------
data required time                                     1.10
data arrival time                                     -1.61
-------------------------------------------------------------
slack (MET)                                            0.51
```

} *Launch*

} *Delay*

} *Capture*

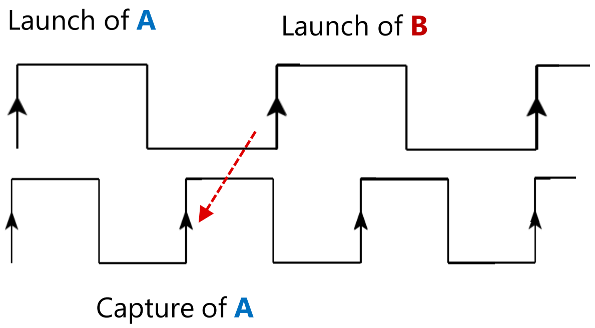
$$T_{launch\_edge} + T_{launch\_latency} + T_{comb} \geq T_{capture\_edge} + T_{capture\_latency} + T_{hold}$$

Reference : Advanced HDL Synthesis and SOC Prototyping: RTL Design Using Verilog | SpringerLink

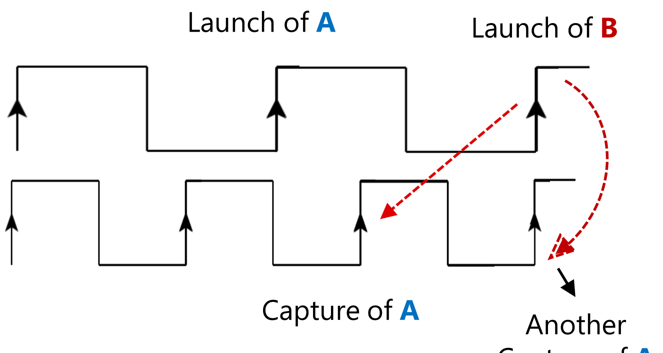
in /amradelm

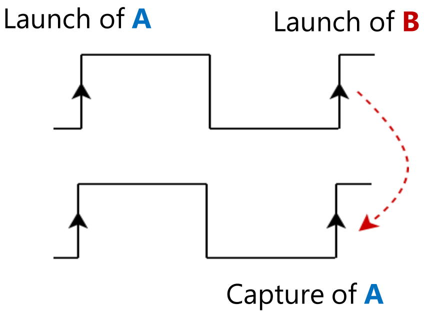
Hold Time

- Like setup, the hold timing path could be full cycle, half cycle, multiple cycles or multi clock.
- We consider the edge where **A** is captured and **B** (next data) is launched because **B** is what will overwrite **A.** The red arrows in the waveforms show the launch - capture edges.
- If there are more launch-capture combinations, like the case of multi clock path, the STA tool will consider the worst of them.
- Like setup. We will just plug different values for the clock edges into the hold equation and the concepts remain unchanged[1].

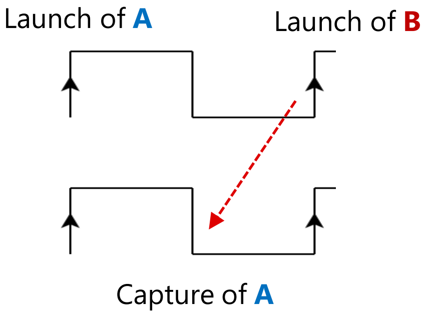


OR

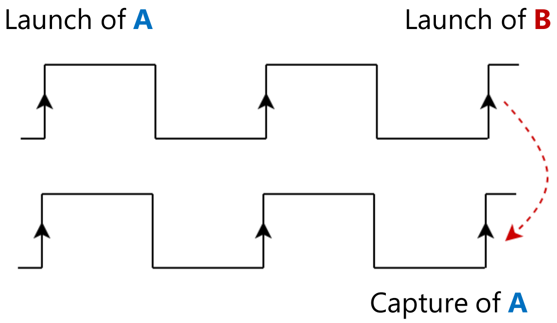




Full Cycle Path



Half Cycle Path



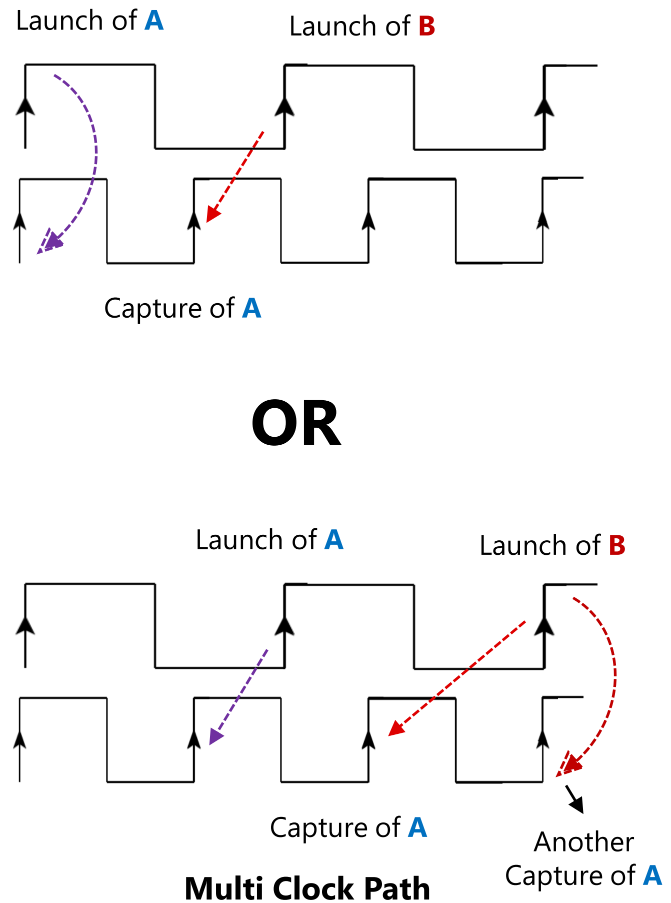
Multi Cycle Path

Multi Clock Path

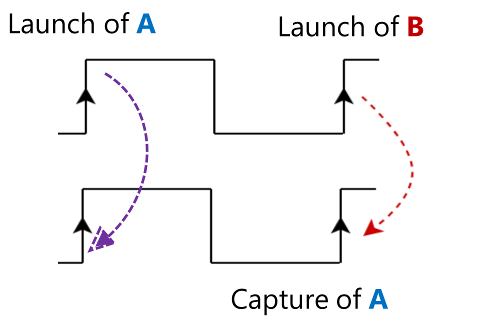
[1]: A common mistake is to say hold is not affected by the clock period. This is only true for full and multi cycle paths where the launch and capture edges occur at the same time. But since full cycle paths are the most common types of paths and also more susceptible to violation, engineers generalize and say hold is not affected by the clock period

Hold Time

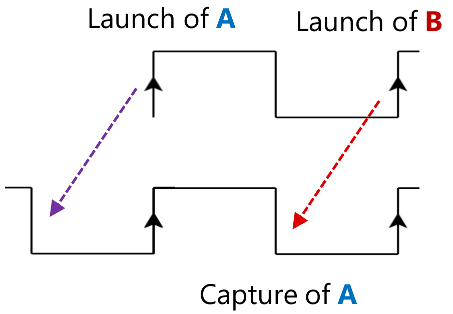
- We also don't want **A** to be captured by an earlier edge
- We should also check hold between the launch of **A** and the capture edge that comes before **A's** intended capture edge
- Now we know all the launch capture combinations and the tool will consider the worst of them[1]



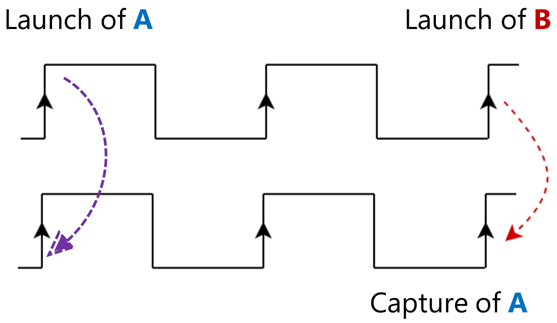
Multi Clock Path



Full Cycle Path



Half Cycle Path



Multi Cycle Path

[1]: Timing Analyzer Example: Clock Analysis Equations | Intel.

How to Fix a Hold Violation

- By comparing the setup equation with the hold equation, we find that fixing hold violations requires the opposite of the methods we discussed with setup.
- Instead of decreasing T_{comb} we will try to increase it by adding buffers, increasing wire delay, downsizing, etc. And instead of increasing the capture latency or decreasing the launch latency we will do the opposite.
- This shows that hold contradicts setup and fixing hold may worsen setup.
- We showed earlier that increasing delay is always easier than decreasing it. This means that fixing hold is generally easier than fixing setup.
- This is why setup has more priority over hold. Hold is only considered in PNR step and fixing hold violations starts when all setup violations are fixed[1].

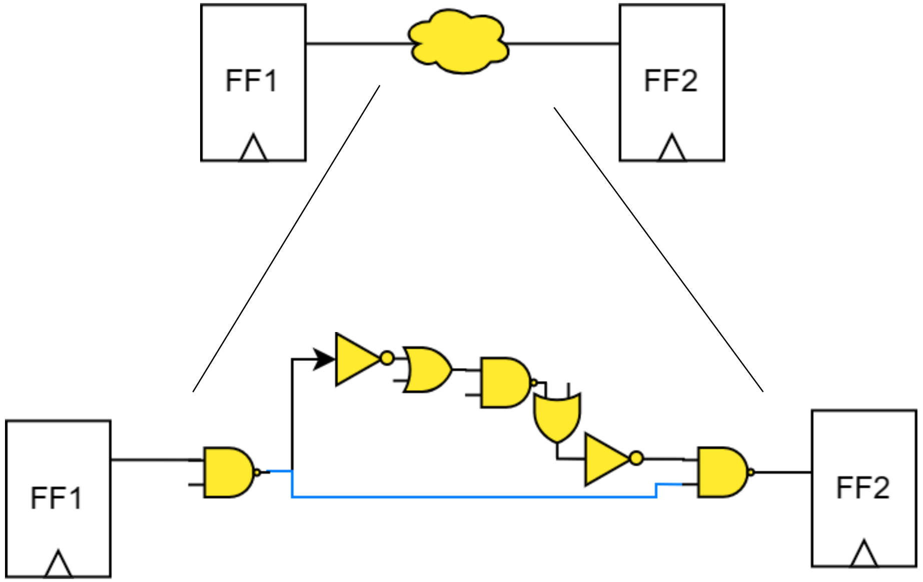
Setup : $T_{launch\_edge} + T_{launch\_latency} + T_{comb} < T_{capture\_edge} - T_{setup} + T_{capture\_latency}$

Hold : $T_{launch\_edge} + T_{launch\_latency} + T_{comb} \geq T_{capture\_edge} + T_{capture\_latency} + T_{hold}$

[1] : Hold is still monitored across the PNR stages and while we focus more on setup we make sure hold is solvable and under control

How to Fix a Hold Violation

- **Consider the example below:**
  - The STA engineer sees two violations, setup and hold, both having the same startpoint and endpoint. The engineer tries adding buffers in front of FF2 to fix hold but the setup is worsened, then tries to fix setup by changing flavor but hold is worsened. It seems we reached a dead end.
  - If we investigate the violations in depth, we can see there are two paths, the upper long one which violates setup and the lower short one (**blue**) that violates hold.
  - So, to fix the setup violations we can change the flavor of the cells in the upper path. And to fix hold we can add buffers along the lower **blue** path.
- This example shows that some hold violations can be tricky and need a deep look into the timing path.



Thank You!