DEPARTMENT OF COMPUTER SYSTEM ENGINEERING

Digital Integrated Circuits - ENCS333

Dr. Khader Mohammad

Lecture #12

Timing

Agenda

- Intro (What is timing all about)
- Static timing versus dynamic timing
- Timing primer
- Flop based timing
- How clock uncertainty impacts timing analysis
- Latch based timing (transparency, time borrowing, time borrowing FFs etc)
- What is a design window?
- Gated clock & domino timing
- Phase paths, MCPs & cycle adjusted margins
- Understanding EVRs & BVRs
- Full chip rollups and slack allocation
- Advanced Topics (loop paths, zero-cycle paths)

What do we mean by “Timing” ?

- Timing Analysis is a method of analyzing and solving timing problems to ensure that the design meets the target frequency.

- A circuit meets target frequency when all of the signals:
  - Arrive just before they are needed.
  - Remain until no longer needed.
  - Be guaranteed stable in between.

- Critical paths are paths in the circuit along which the signal doesn’t reach its destination on time; this causes the functionality of that part of the circuit to be impaired.

Static Timing

- Static timing analysis computes the worst case delays for paths in a given circuit.
- It is a non-simulation approach to timing analysis used to analyze the propagation of delays.
- Pathmill and Tango-XT are examples of static timing tools.

Static Timing

- Searches path by path. Performs static timing analysis (STA) of a circuit by analyzing the circuit in a non-simulated way (no input vectors.)
  - Traverses all possible paths between each source-sink pair.
  - Calculates the path delay and the arrival time at each signal along the path and internal sampling requirements.
  - Sorts the paths by the amount of violation at their sink node and prints them in a report.
- Report can be used to:
  - Determine how to fine tune the timing of the design to meet the target frequency
  - Analyze timing violations and performance bottlenecks in circuits.

Static vs. Dynamic Timing

- Static timing Analysis:
  - Non simulation approach used to analyze propagation of delays.
  - Computes worst case path delays by accumulating pre-characterized device delays along a given path.
  - Path determination is based on the possibility of an event on a device input causing an event on a device output.
- Dynamic Timing Analysis:
  - Circuit simulation approach that obtains a very accurate timing analysis of a path.
  - Uses input waveforms and path sensitization, generates output waveforms.

Comparison of Static vs. Dynamic Timing

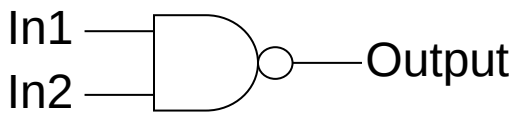
Static (path analysis)	Dynamic (simulation)
• No input vectors; full coverage	• Uses input vectors; coverage depends on vector selection
• Points to exact origin of timing problems (critical paths)	• Requires a debug process to find the origin of timing problems
• Ensures full coverage (Much faster than simulation)	• Requires a lot of CPU time for good coverage
• Relies on approximated models which are less accurate than simulation	• Highest degree of accuracy possible through software simulation
• Inherent problem: false paths	• Inherent problem: test vector generation
• Simulates only one input switching at a time	• Simulates multiple inputs switching simultaneously
• Gives worst case delays only	• Gives precise delay information

When do we use Static or Dynamic Timing?

- Static Timing is used more often due to the size of the designs.

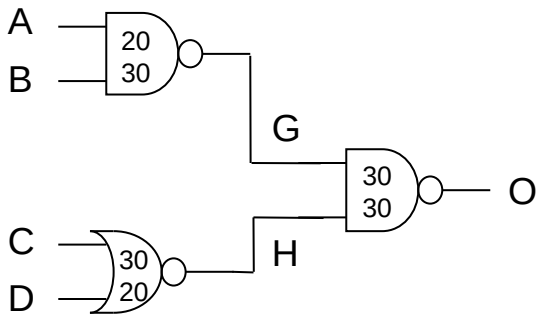
- Dynamic timing is typically used where analog behavior is expected and not modeled correctly by static timing.

Static vs. Dynamic Timing: Example 1



- Given a simple two-input NAND gate, static analysis will yield four paths analyzed:
  - In1 rising -> output falling
  - In1 falling -> output rising
  - In2 rising -> output falling
  - In2 falling -> output rising
- Given the same NAND gate, dynamic analysis can yield six main transitions. The above four, as well as
  - In1 rising & In2 rising -> output falling
  - In1 falling & In2 falling -> output rising
- Theoretically, Dynamic Timing has an infinite number of different paths, as the user determines the exact timing of the rise/fall of the inputs in relation to each other.

Static vs Dynamic Timing: Example2



Path #1 A -> O

A Rising -> G Falling -> O Rising

20ps + 30ps = 50ps

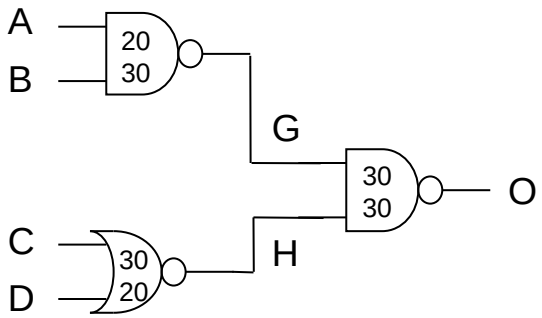
Path #2 A -> O

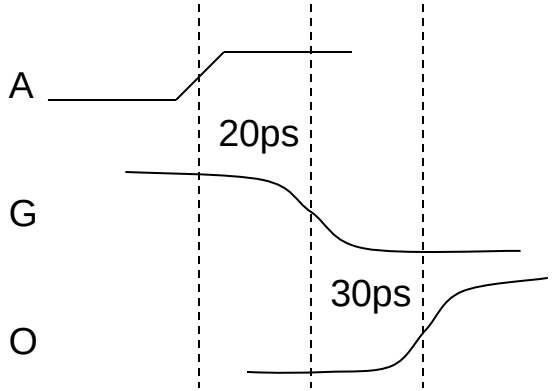
A Falling -> G Rising -> O Falling

30ps + 30ps = 60ps

- Static timing analysis is based upon the possibility of a path existing.
- All paths are traversed and delays are calculated.
- These delays are used to check signal arrival times in order to determine setup and hold margin.

Static vs Dynamic Timing: Example2 (cont.)



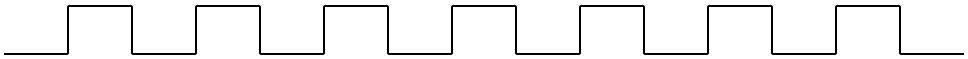


- Using Dynamic Timing on the same example, inputs B, C, and D need to be sensitized to exercise the path: A rising -> G falling -> O rising
  - B must be set high
  - C and D must be set low

Timing Primer

- Terms we need to learn very very well:
  - Reference Clock
  - Lead edge signals
  - Trail edge signals
  - Valid Windows
  - Required Windows
  - Margin
  - Margin Violations

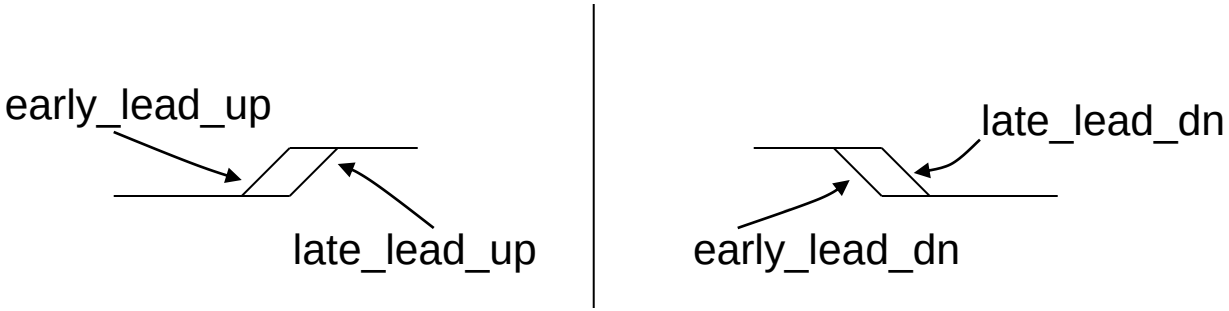
What is a Reference Clock?



- An ideal clock that does not have jitter or skew
- Generally does not exist anywhere in the design
- All timing is analyzed and specified against a reference clock
- There may be more than one per design

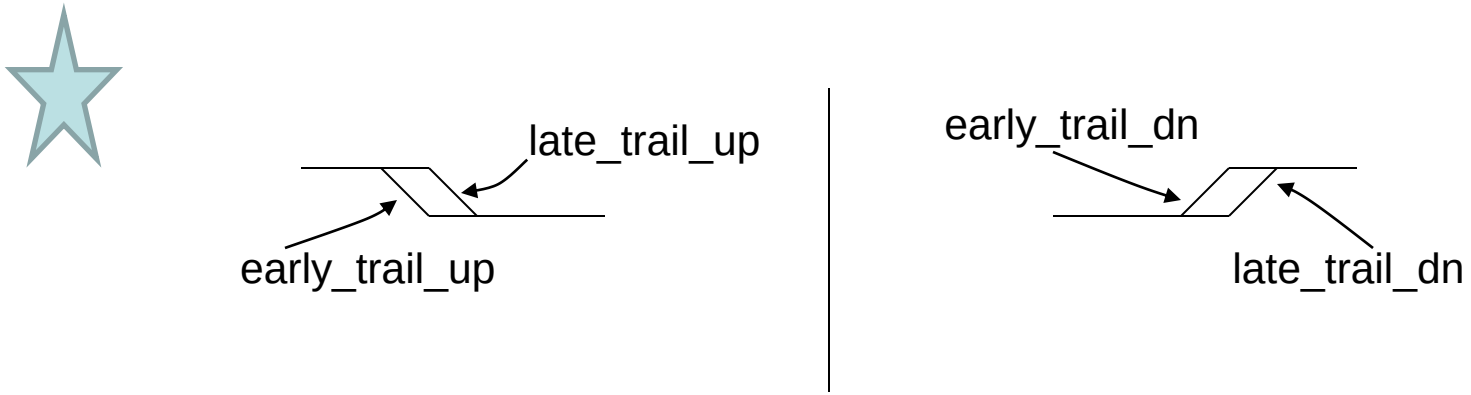
What are Lead Edge Signals ?

- The lead edge is the time that the signal become stable/active.
- If a signal rises and becomes active, this is **lead_up**.
- If the signal falls and becomes active, it is a **lead_dn** edge.
- A signal may have multiple lead up edges at different times. For this case we can describe them using the terms: **early_lead** and **late_lead**



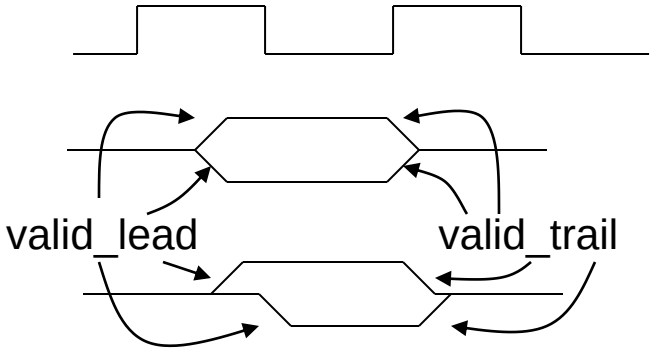
What are Trail Edge Signals ?

- The trail edge is the time that the signal become inactive (or goes away).
- The **trail_up** edge (falling edge) refers to the trail edge of the “up” (logical 1) window.
- The **trail_dn** edge (rising edge) refers to the trail of the “dn” (logical 0) window.



What are Valid Windows?

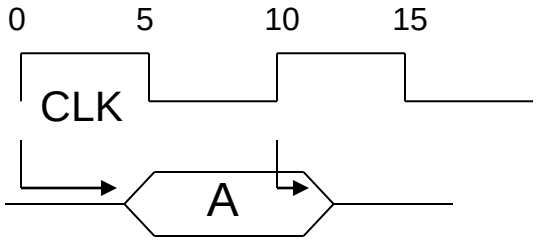
- Valid windows are referenced to a clock event and indicate the time when a signal is stable.
- The valid_lead_up edge and the valid_lead_dn edge need not have the same timing. These edges may come from entirely different paths of logic.

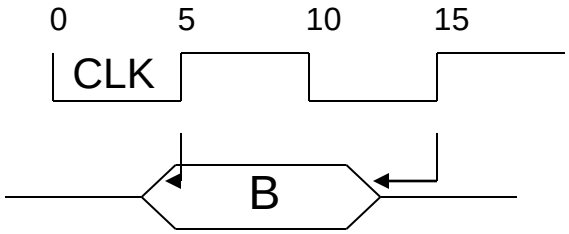


- valid_lead will correspond to late_lead
- valid_trail will correspond to early_trail

Valid Window Examples

- Valid times are conventionally specified as
  AR or AF (after rise or after fall)



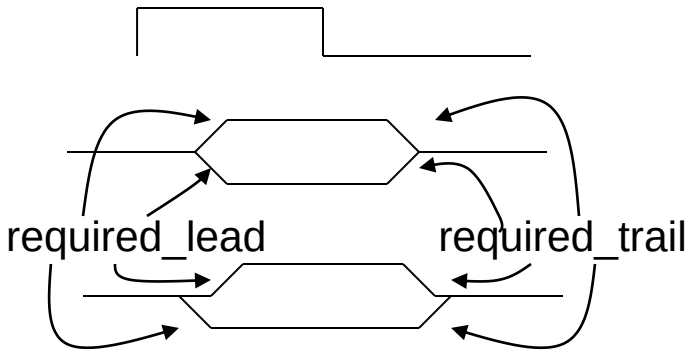


A: valid_lead = 4 ar CLK
valid_trail = 2 ar CLK

B: valid_lead = -1 ar CLK
valid_trail = -3 ar CLK

What are Required Windows?

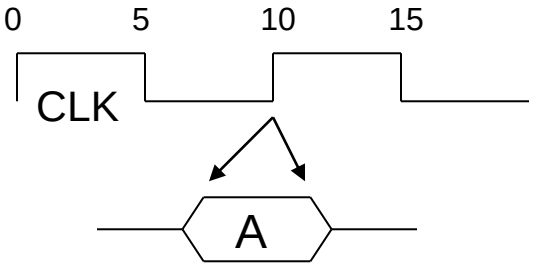
- Required windows indicate the time when a signal must be valid or active to meet timing. This signal must be valid for the whole time of the defined window.
- Required windows come from circuit timing requirements for setup times and hold times.

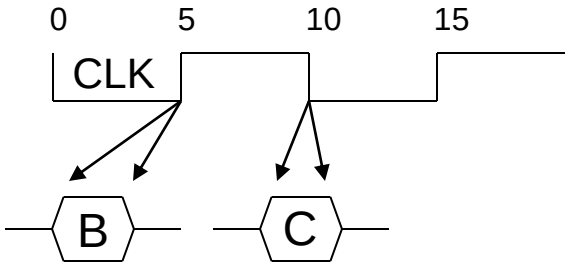


- required_lead will correspond to early_lead
- required_trail will correspond to late_trail

Required Window Examples

- Required times are conventionally specified as
  BR or BF (before rise or before fall)





A: required_lead = 4 br CLK
required_trail = -3 br CLK

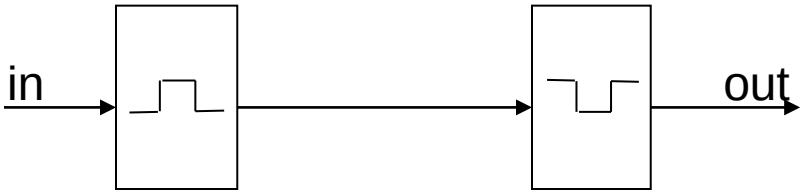
B: required_lead = 5 br CLK
required_trail = 2 br CLK

C: required_lead = 2 bf CLK
required_trail = -1 bf CLK

A Note about Coloring Conventions

- The color (af/ar) of the VALID lead is always referenced to the “opening” edge of the closest sequential element to the output along a path.

- The color (bf/br) of the REQUIRED lead edge is always referenced to the “closing” edge of the closest sequential element to the input along a path.

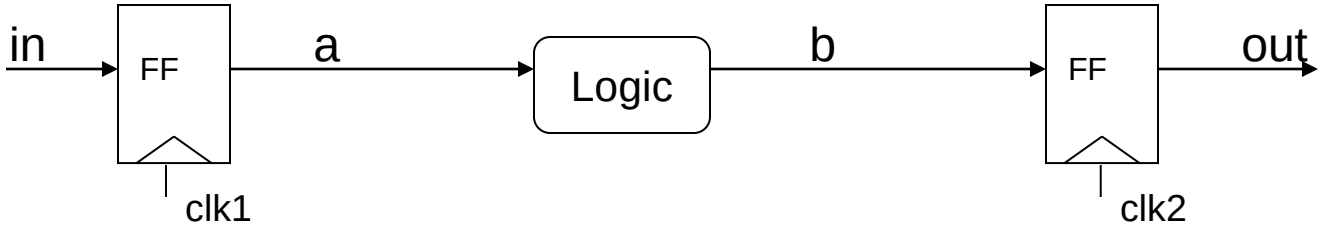


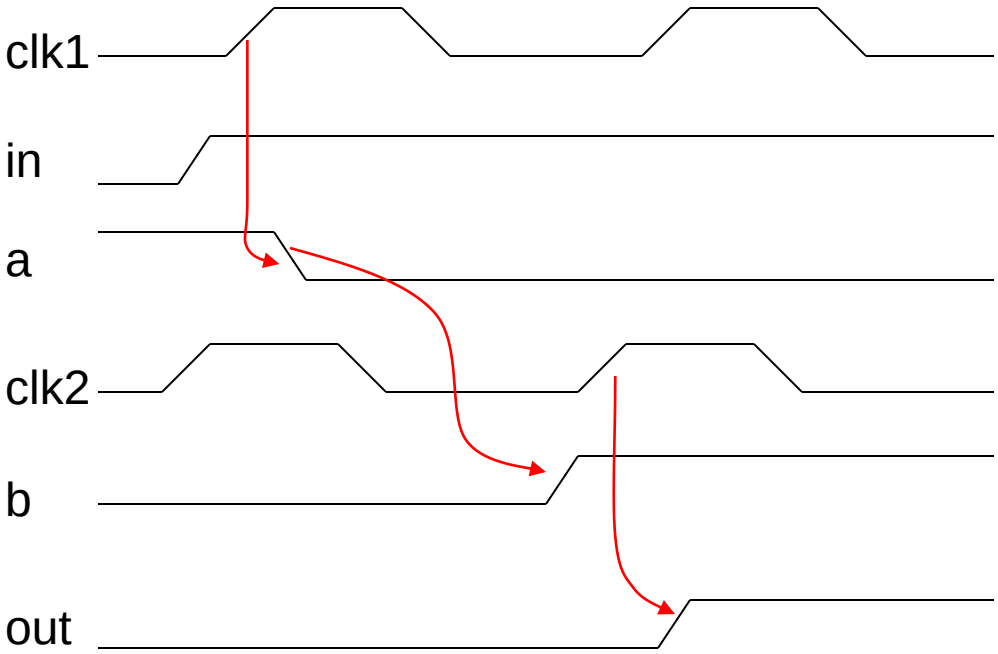
For the example:

VALID = af

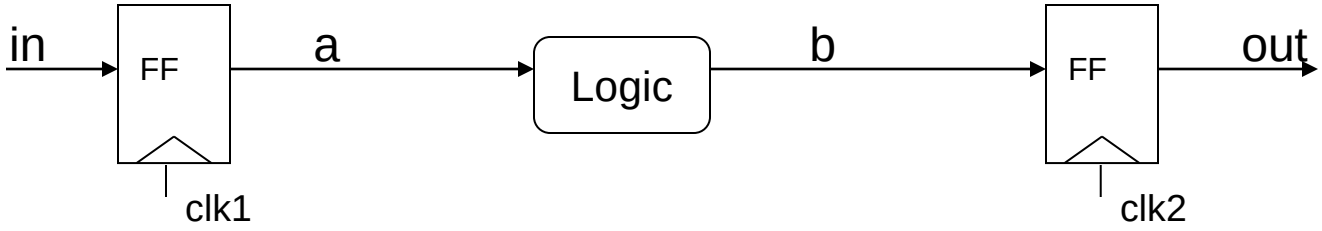
REQUIRED = bf

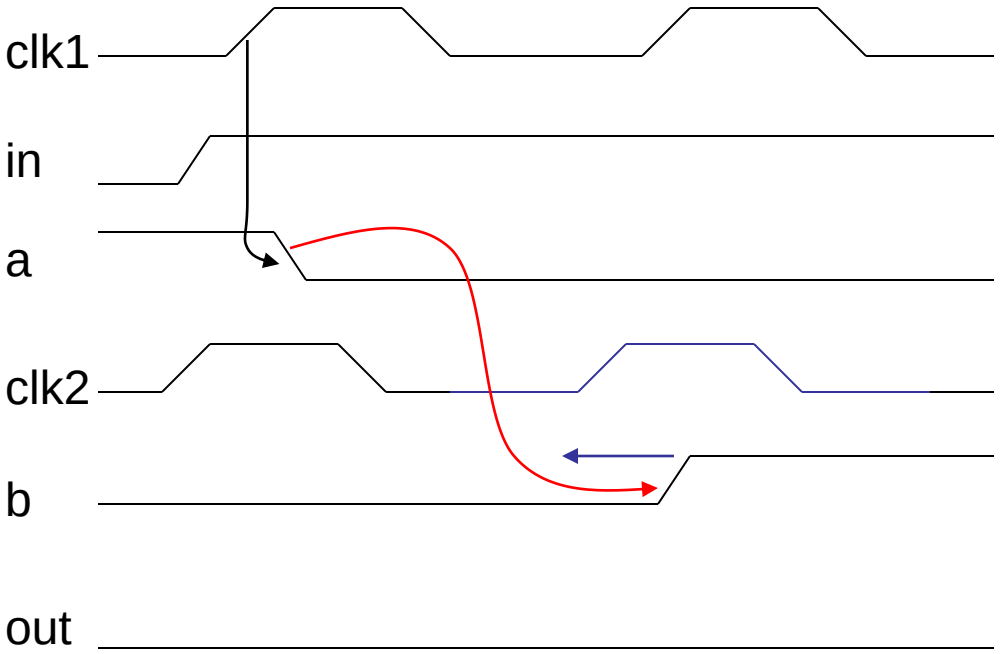
Max/Min Delay Margins Example 3





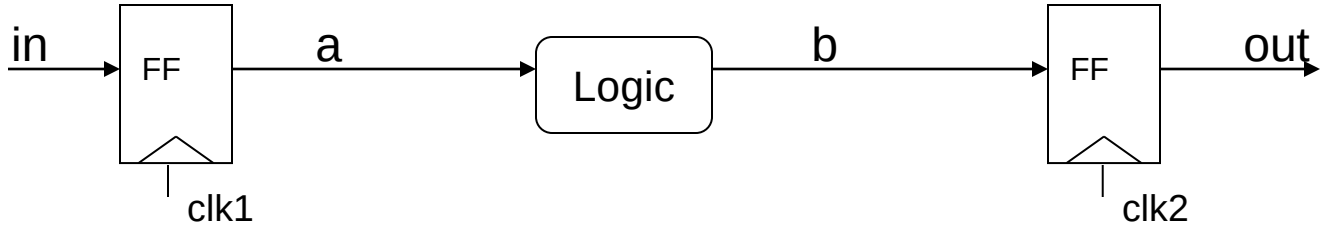
Max/Min Delay Margins Example 3 (cont.)

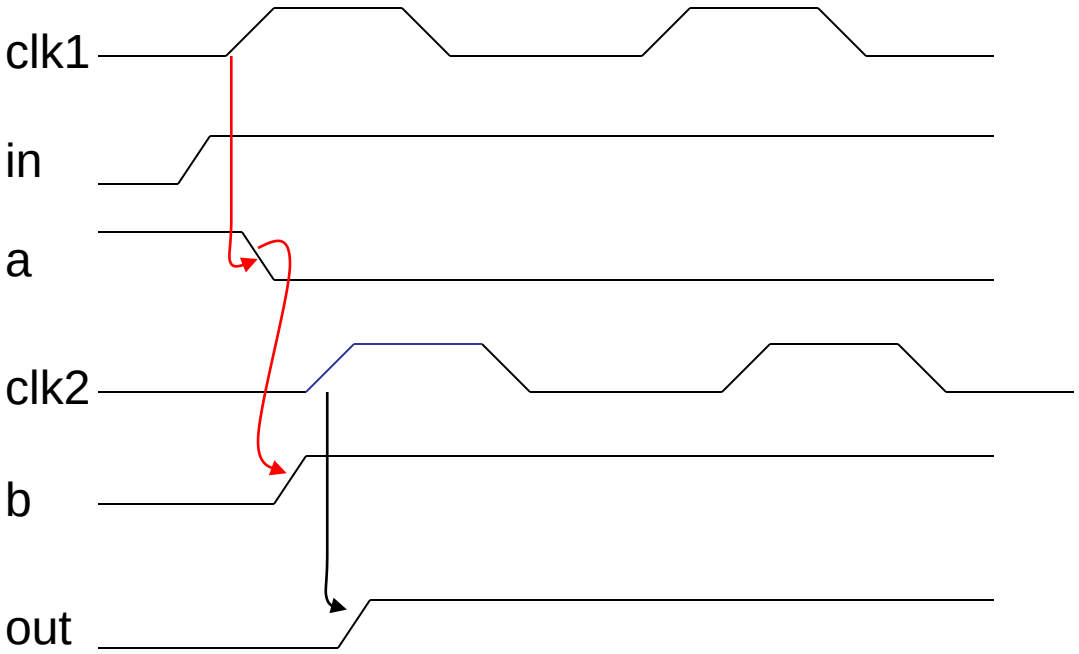




Max Delay Failure!!! On the second ff

Max/Min Delay Margins Example 3 (cont.)





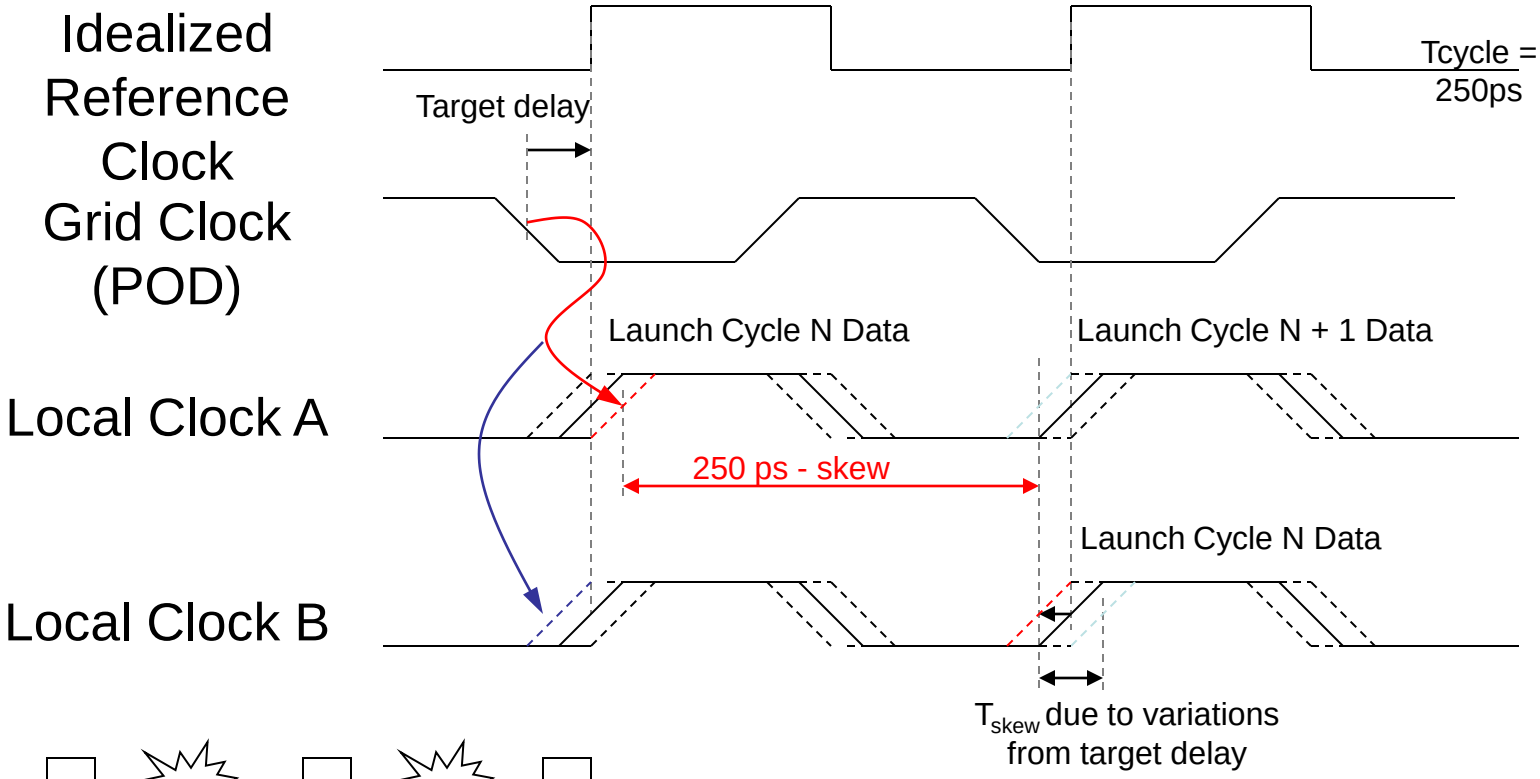
Min Delay Failure!!!

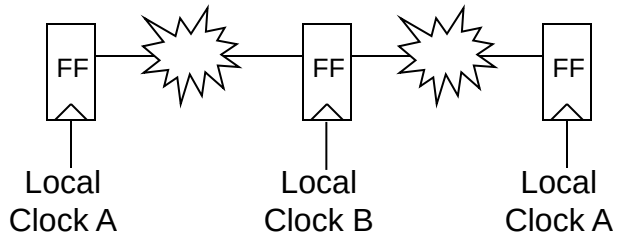
Sources of Clock Uncertainty

- Skew – Unintentional mismatch in the arrival of two local clock edges produced by the same global clock edge. Main causes are:
  - RC mismatch in the clock network
  - Cross-coupling in the clock network
  - (P)rocess, (V)oltage, (T)emperature mismatch in the clock distribution network
- Jitter* – Variation in the cycle time of the global clock. Main causes are:
  - PLL Jitter
  - Coupling in the global clock distribution network

* My definition of jitter might vary from what other people call jitter. For our conversation we'll use my definition

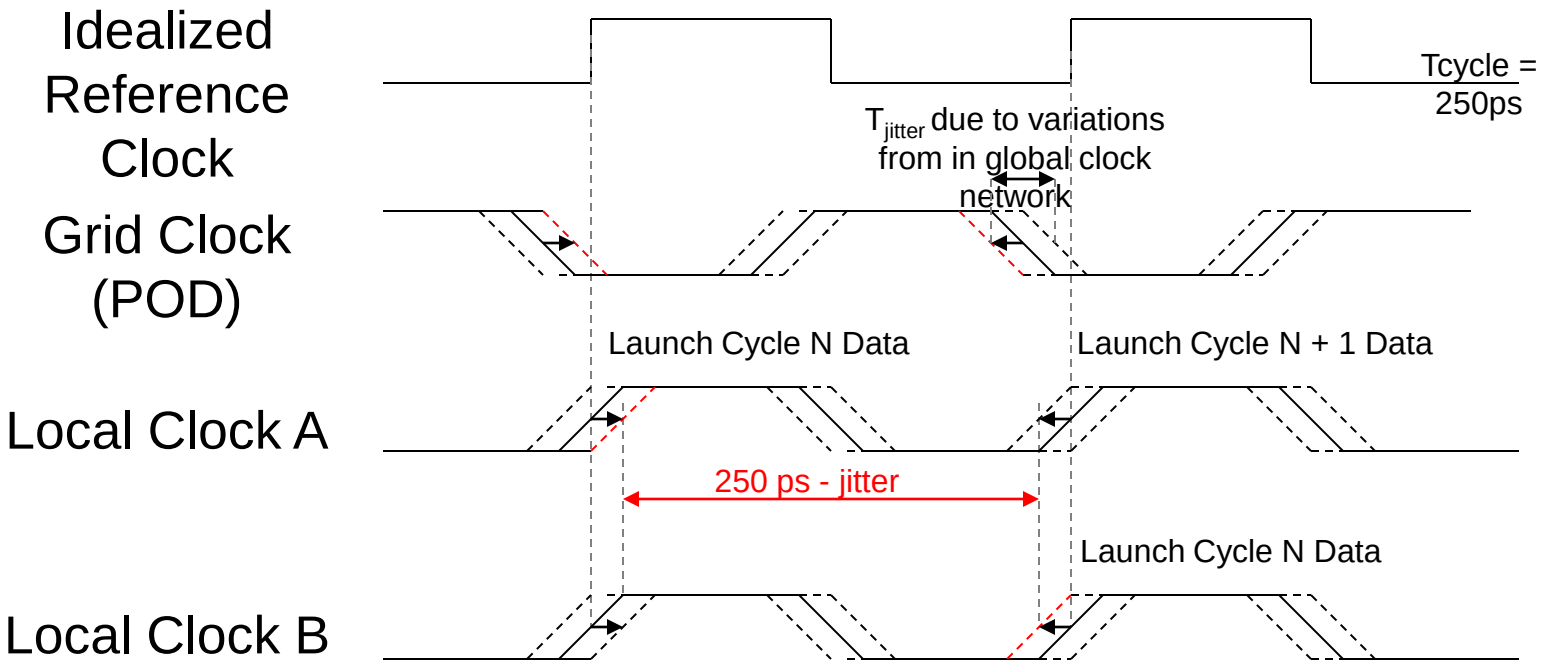
Clock Skew Illustrated

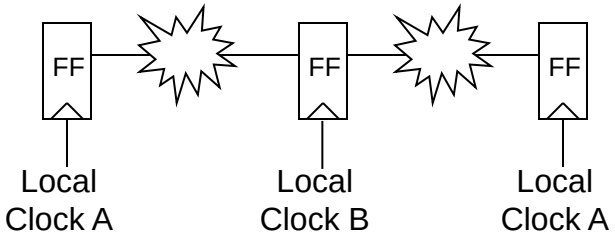




- Variations in distribution cause unexpected delay differences from the POD (point of divergence) to the local clock @ the sequential
- Skew impacts both setup and hold checks

Clock Jitter Illustrated

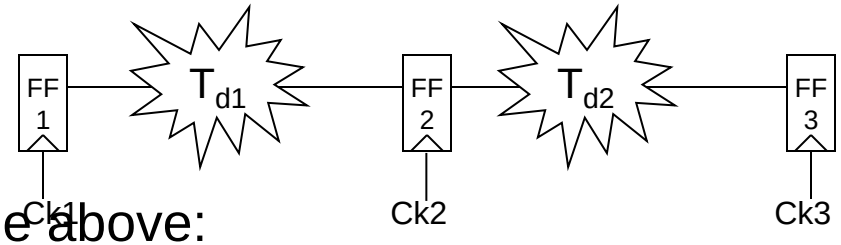




- Variations in PLL & global distribution network cause jitter
- Jitter only impacts setup checks

How Skew & Jitter Fit in the Setup Margin Calculation

- Without Skew/Jitter
  - $Margin_{setup} = Required_{abs} – Valid_{abs}$
- With Skew/Jitter
  - $Margin_{setup} = Required_{abs} - Valid_{abs} – MaxSkew$



- For the example above:

$Margin_{setup} = Tcycle – Clk->Q_{FF1} – T_{d1} – Setup_{FF2} – (MaxSkew_{ck1-ck2} + Jitter)$

$Margin_{setup} = Tcycle – Clk->Q_{FF2} – T_{d2} – Setup_{FF3} – (MaxSkew_{ck2-ck3} + Jitter)$

Each cycle pays setup, skew & jitter

Setup Margin Examples

$Margin_{setup\ FF1->FF2} = T_{cycle} - Clk->Q_{???} - T_{d??} - Setup_{???} - (MaxSkew_{???} + Jitter)$

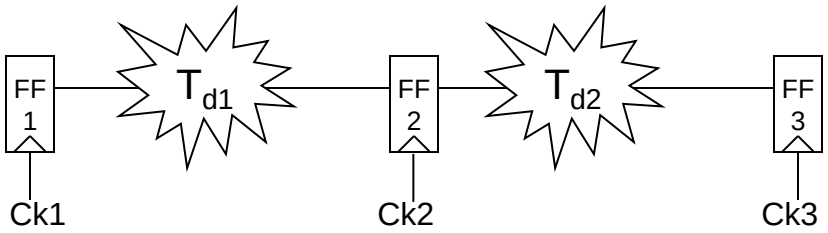
$= 250-(23+185+37+15+16)$

$= 250-276=-26ps$

$Margin_{setup\ FF2->FF3} = T_{cycle} - Clk->Q_{???} - T_{d???} - Setup_{???} - (MaxSkew_{???} + Jitter)$

$= 250 -(25+155+41+10+16)$

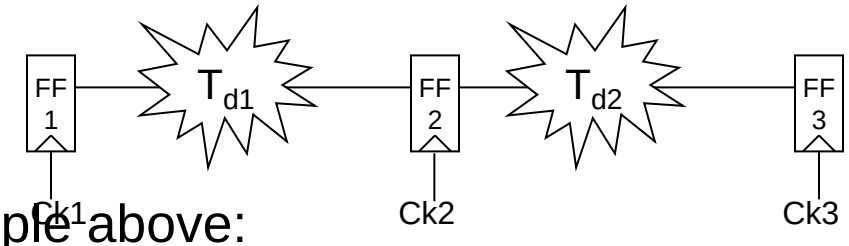
$=250-247=3ps$



T_{d1}	0.185		T_{d2}	0.155
$Clk->Q_{FF1}$	0.023		$Setup_{FF1}$	0.038
$Clk->Q_{FF2}$	0.025		$Setup_{FF2}$	0.037
$Clk->Q_{FF3}$	0.024		$Setup_{FF3}$	0.041
$MaxSkew_{Ck1-Ck2}$	0.015		$MaxSkew_{Ck2-Ck3}$	0.010
Tcycle	0.250		Jitter	0.016

How Skew Fits in the Hold Margin Calculation

- Without Skew
  - $Margin_{hold} = Valid_{abs} – Required_{abs}$
- With Skew
  - $Margin_{hold} = Valid_{abs} - Required_{abs} – MinSkew$



- For the example above:

$Margin_{hold} = Clk->Q_{FF1} + T_{d1} – Hold_{FF2} – MinSkew_{ck1-ck2}$

$Margin_{hold} = Clk->Q_{FF2} + T_{d2} – Hold_{FF3} - MinSkew_{ck2-ck3}$

Hold Margin Examples

$Margin_{hold\ FF1->FF2} = Clk->Q_{???} + T_{d??} - Hold_{???} - MinSkew_{???}$

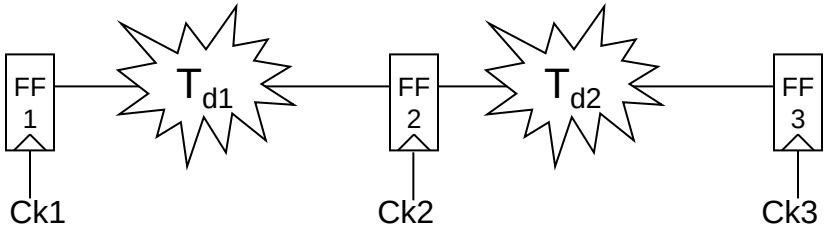
$= 18+55-25-45$

$= 3$

$Margin_{hold\ FF2->FF3} = Clk->Q_{???} + T_{d??} - Hold_{???} - MinSkew_{???}$

$=20+40-28-40$

$=-8$



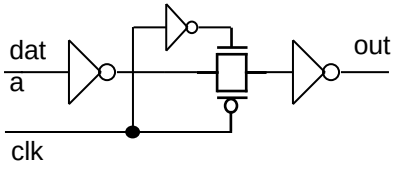
T_{d1}	0.055		T_{d2}	0.040
$Clk->Q_{FF1}$	0.018		$Hold_{FF1}$	0.026
$Clk->Q_{FF2}$	0.020		$Hold_{FF2}$	0.025
$Clk->Q_{FF3}$	0.022		$Hold_{FF3}$	0.028
$MinSkew_{Ck1-Ck2}$	0.045		$MinSkew_{Ck2-Ck3}$	0.040

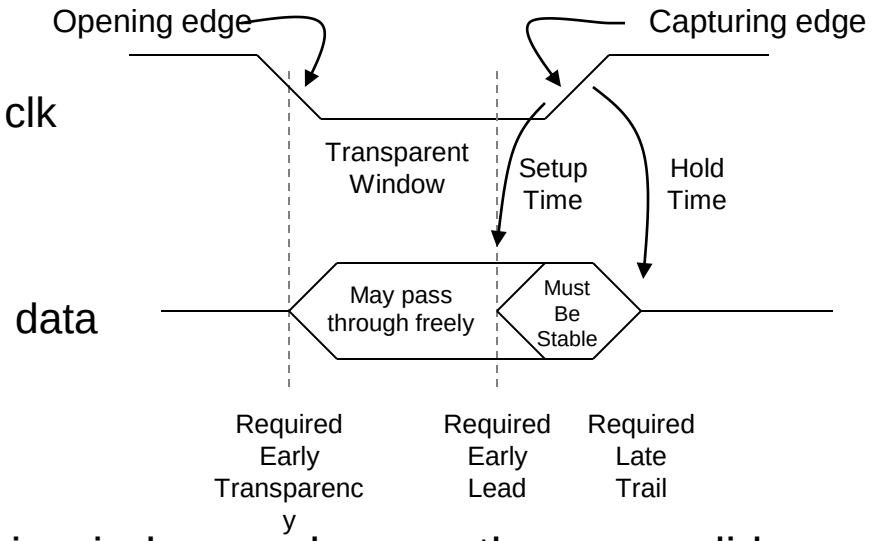
Clock Uncertainty Key Notes

- Always assume worst case skew
  - Launching clocks are slow in setup analysis, capturing clocks are fast
  - Launching clocks are fast in hold analysis, capturing clocks are slow
- Skew can be minimized by the designer, jitter cannot
  - Since skew is related to local clock variation, you can control this by keeping the POD as close as possible
  - Jitter is mainly caused by the PLL & the global grid, you can’t impact this as a fub designer
- Max skew and Min skew are different
  - Min skew is bigger because it must encompass more of the natural variation because a hold failure is fatal
  - Max skew is smaller because a setup failure only impacts binsplit

What is a Transparency Window?

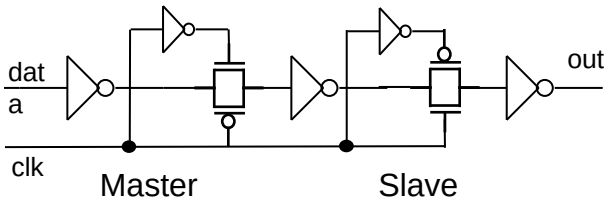
- Transparency Window
  - The time when a data input flows unblocked through a sequential

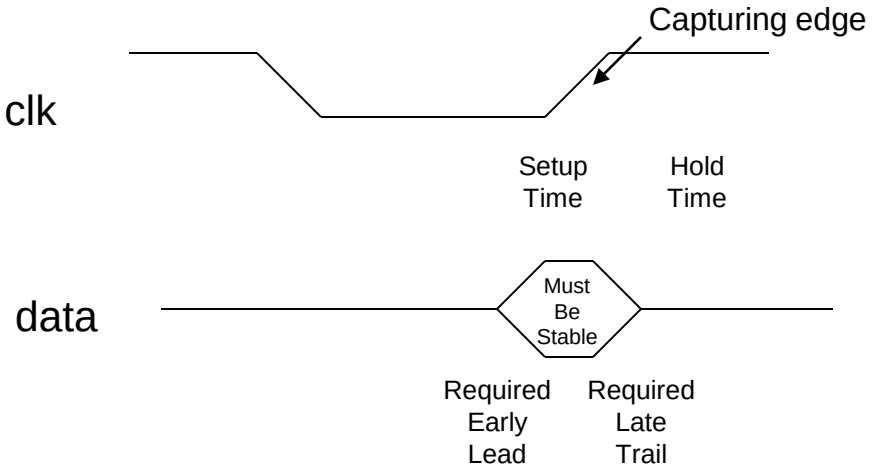




  - Inputs may switch anytime in this window, so long as they are valid before the capturing edge of the clock, and remain valid until the hold time (+ skew)

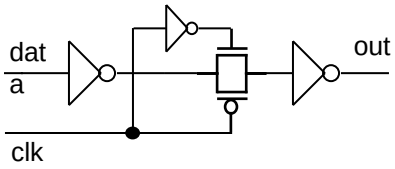
Flop vs. Latch Timing Requirements

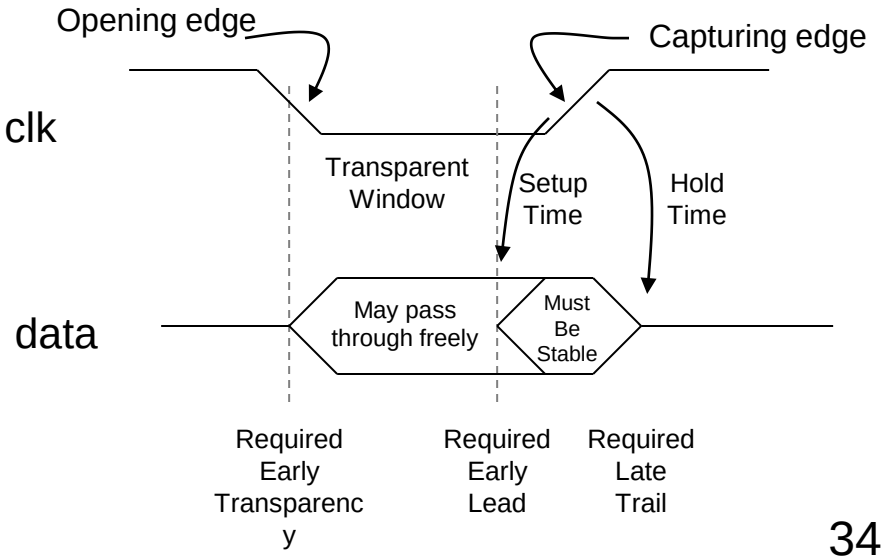




FLOP Timings

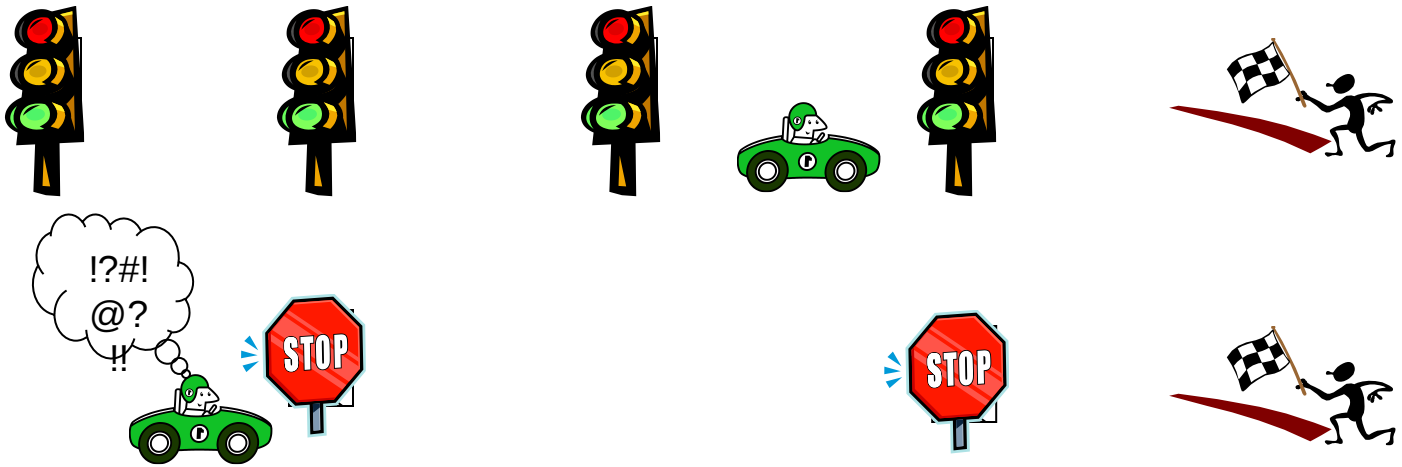
Latch Timings





A Real World Example of Transparency

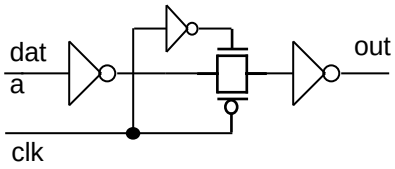
- Level sensitive latches allow transparency
  - When the latch opens, data may arrive and pass freely from input to output *without waiting for clock*
  - Data only needs to be stable when the latch closes
- Think of a latch like a street light and a flop light a stop sign. Catch the green and it's a free ride

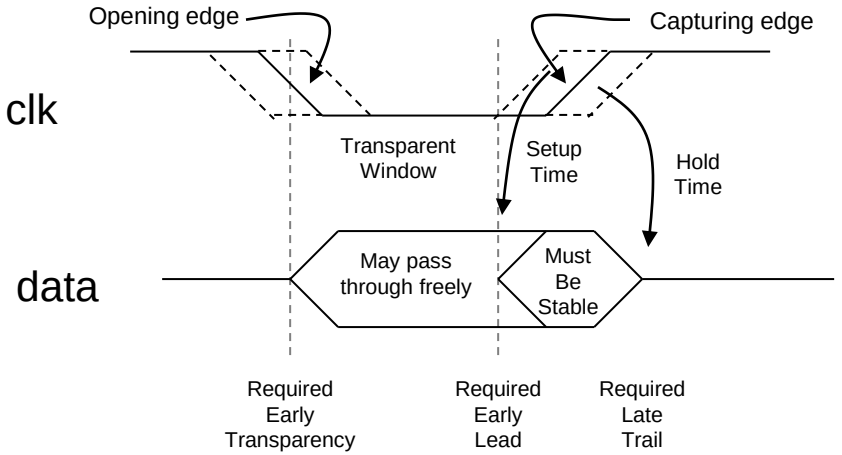


Why Transparency Helps

- Transparency reduces the impact of clock uncertainty
  - If the opening clock edge is early, the valid time at the output of the latch for a transparent path is ***unchanged***
  - If the opening clock edge is late, *but not as late as the transparent data*, the valid time at the output is ***unchanged***
- As long as the data is switching somewhere in the middle of the transparency window, the clock has ***no*** impact on the timing through this latch

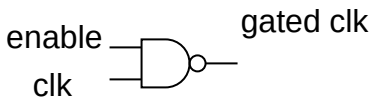
Note that the width of the transparency window directly impacts your ability to meet this requirement

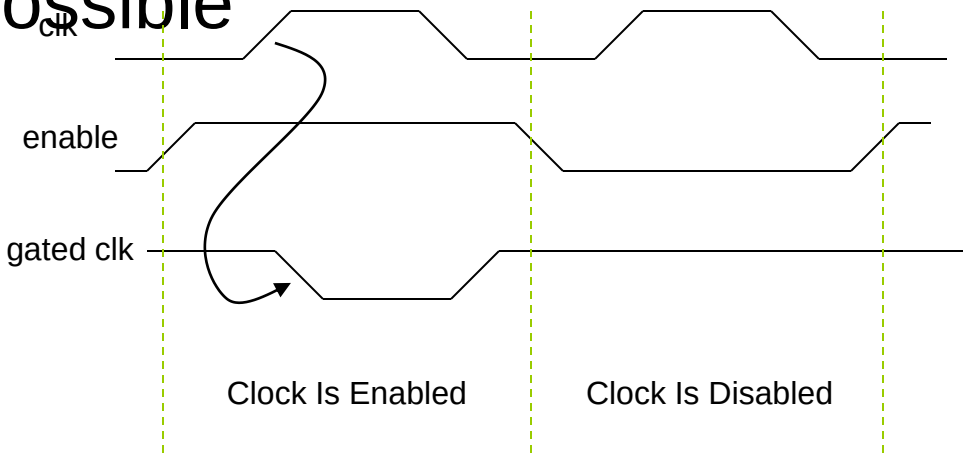




What Are Gated Clocks?

- When a data signal is used to enable/disable a clock, we call the resulting clock a gated clock
- In general we prefer that the resulting clock timing is dependent only upon the driving clock, not the data
  - This is not always possible

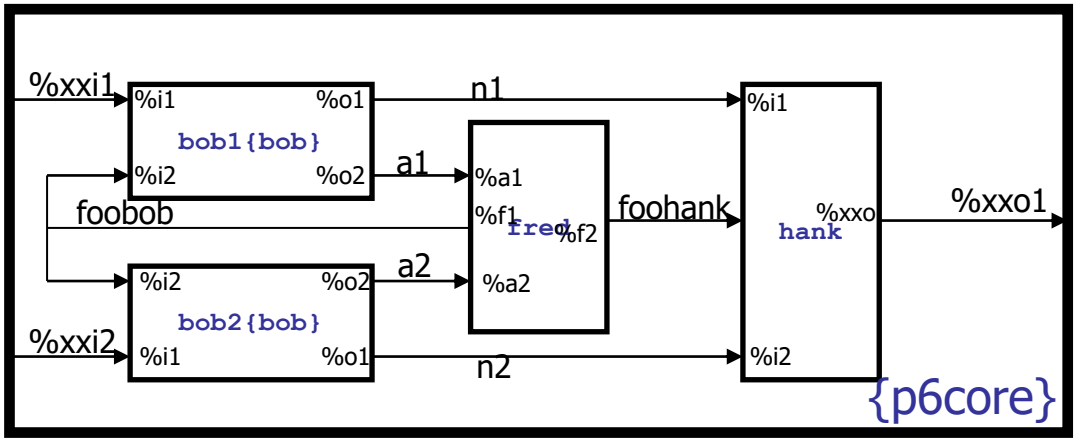




Fullchip Timing Concepts

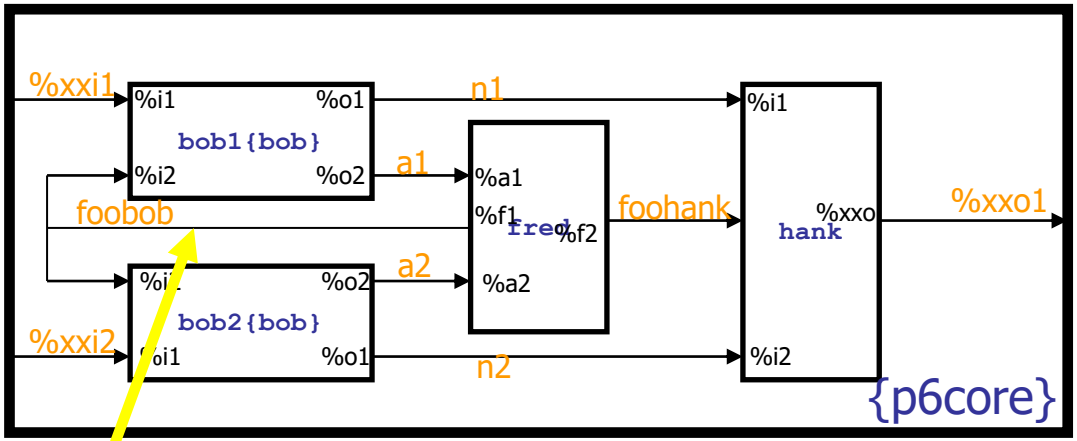
- Sea of BVRs
- Fullchip RC-Extract
- .rcdly
- Timing Window Propagation

Sea of BVRs



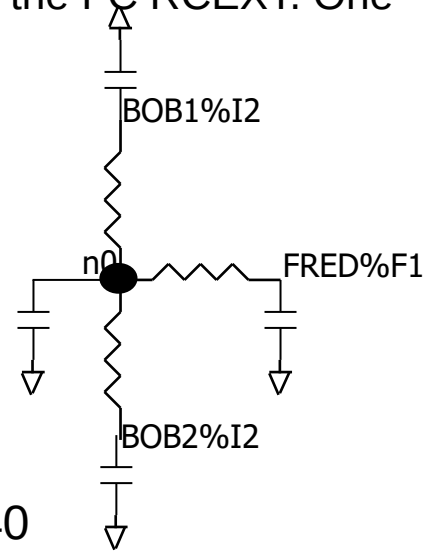
- The block, p6core, contains 4 instances of 3 BVRs (note that BVR “bob” is instantiated twice).
- Note that the node names do not necessarily agree with the pins the nodes connect to.
- Timing/Electrical information for p6core (3 pins) will be coming from **`p6core.evr`**.

Fullchip RC Extract

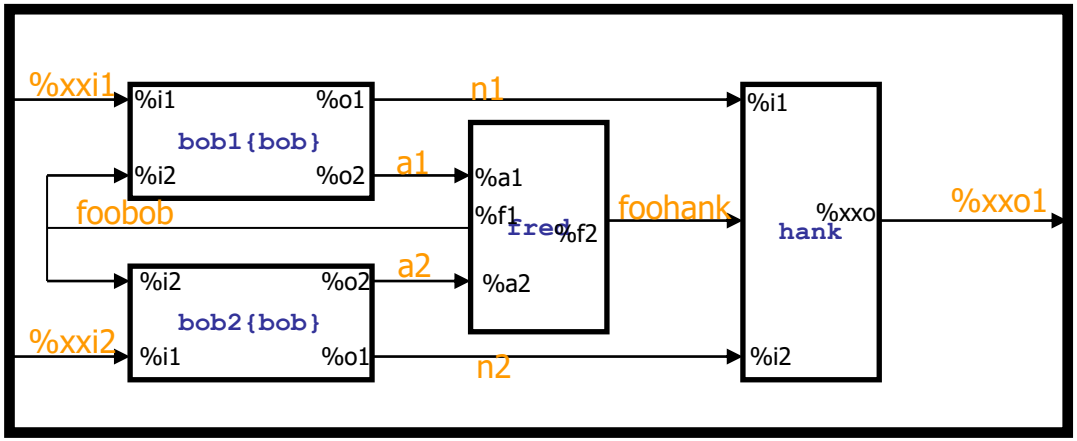


- The nodes in p6core have RCDLY and need to be timed, but before we can do that we need to extract the Rs and Cs. This occurs during the FC RCEXT. One output format for FC RCEXT is a NTCL file:

```
.MACRO FOOBOB BOB1%I2 BOB2%I2 FRED%F1
R1 FRED%F1 N0 5
R2 N0 BOB1%I2 5
R3 N0 BOB2%I2 5
C1 N0 30fF
C2 FRED%F1 10fF
C3 BOB1%I2 5 fF
C4 BOB2%I2 15 fF
.EOM
```



RCDLY generation



- Now that we have the parasitic extraction, we need to compute the delay between the BVRs.
- The specifics on how to generate the RCDLY file will not be discussed here, but here's what a .rcdly file looks like:

```
# drivingpin receivingpin MaxRCDlyUp MaxRCDlyDn MinRCDlyUp
   MinRCDlyDn MaxRcvrSlopeUp MaxRcvrSlopeDn MinRcvrSlopeUp
   MinRcvrSlopeDn
FRED%F1 BOB1%I2 0.100 0.100 0.080 0.080 0.050 0.050 0.040 0.040
FRED%F1 BOB1%I2 0.120 0.120 0.085 0.085 0.060 0.060 0.045 0.045
BOB1%O2 FRED%A1 0.030 0.032 0.022 0.023 0.040 0.043 0.037 0.039
```

SRAM PROJECT:
https://www.youtube.com/watch?v=68Dn1x6cZ4g
https://www.youtube.com/watch?v=SHJPFNI5Mzo
https://www.youtube.com/watch?v=KrqyvpU9Cu0

Multiplier -**A Comparative Study On Low Power Multiplier Using Microwind Tool**

- https://www.youtube.com/watch?v=4-I_PGPog9o
- https://www.youtube.com/watch?v=MCFG7XD16Ek
- http://www.ijsret.org/pdf/EATHD-15026.pdf
- https://www.youtube.com/watch?v=rqwkrUcNyH4
- https://www.youtube.com/watch?v=4-I_PGPog9o
- https://www.youtube.com/watch?v=WxSR2Yhnqk4&t=30s
- https://www.acsu.buffalo.edu/~phaniram/bootstrap-prestructure22_files/images/paper_1.pdf