# Digital VLSI Design

# Lecture 5: Timing Analysis

Semester A, 2018-19

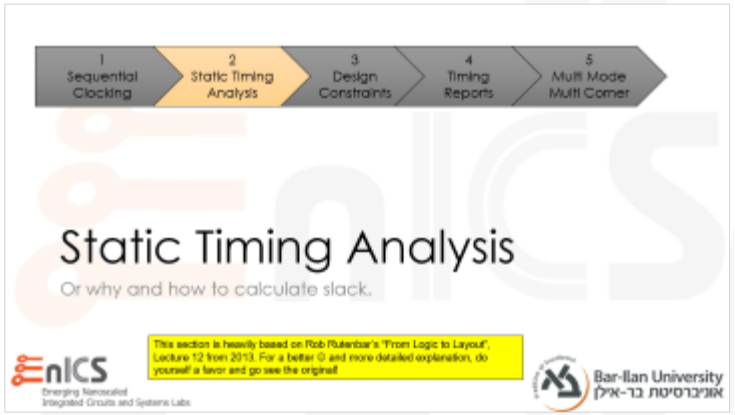
Lecturer: Dr. Adam Teman

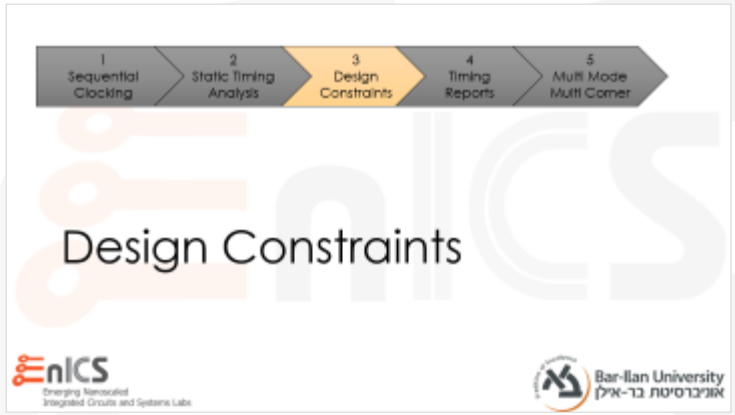
December 7, 2018

Emerging Nanoscaled Integrated Circuits and Systems Labs

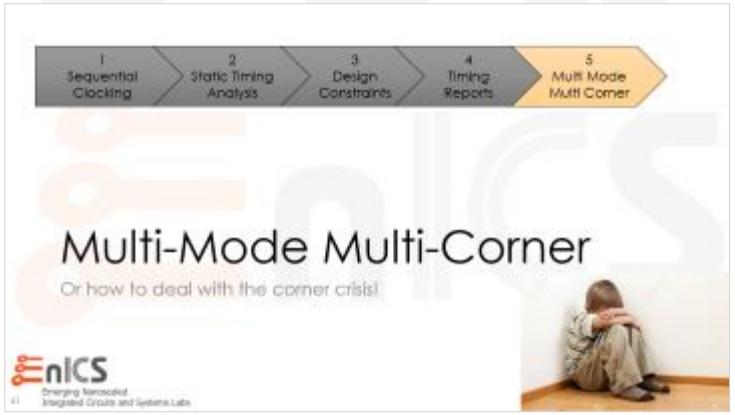
Bar-Ilan University
אוניברסיטת בר-אילן

# Lecture Outline

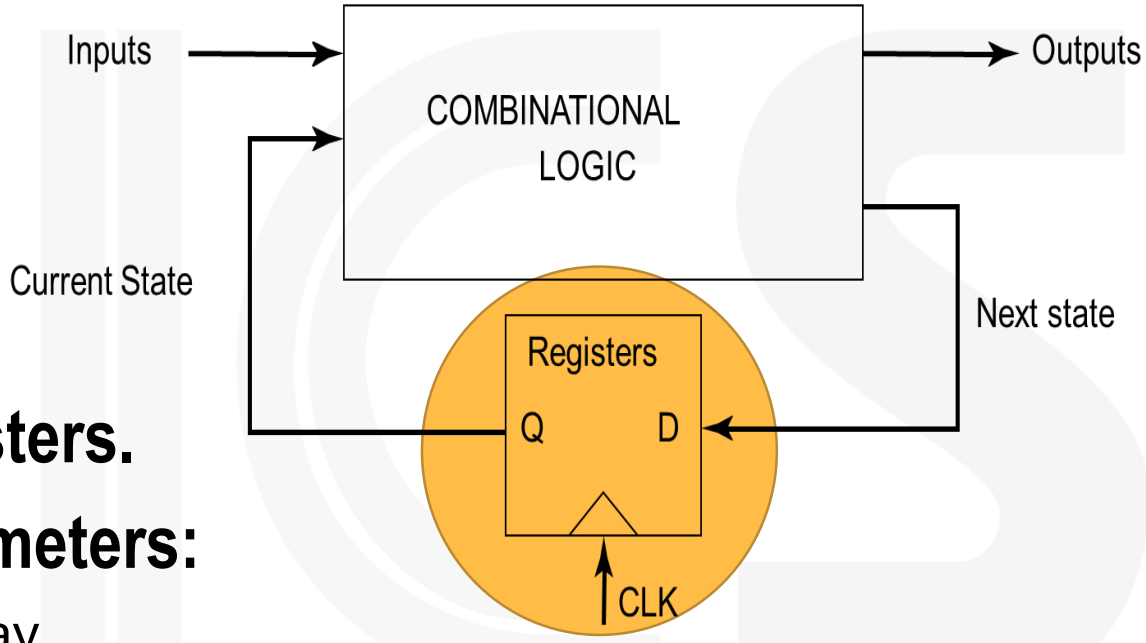


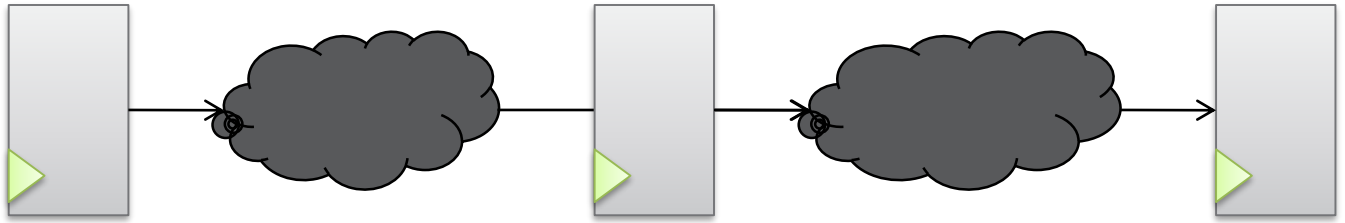


# Sequential Clocking

# Synchronous Design - Reminder

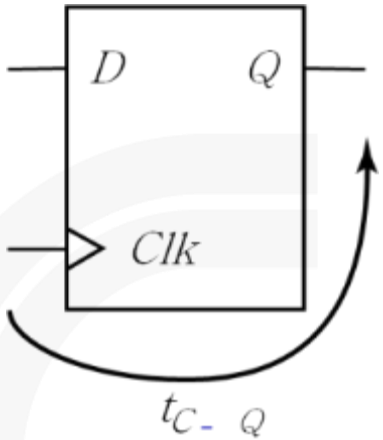
- **The majority of digital designs are *Synchronous* and constructed with *Sequential Elements*.**
  - Synchronous design eliminates races (like a traffic light).
  - Pipelining increases throughput.
- **We will assume that all *sequentials* are *Edge-Triggered*, using D-Flip Flops as registers.**
- **D-Flip Flops have three critical timing parameters:**
  - $t_{cq}$ – clock to output: essentially a propagation delay
  - $t_{setup}$ – setup time: the time the data needs to arrive before the clock
  - $t_{hold}$ – hold time: the time the data has to be stable after the clock

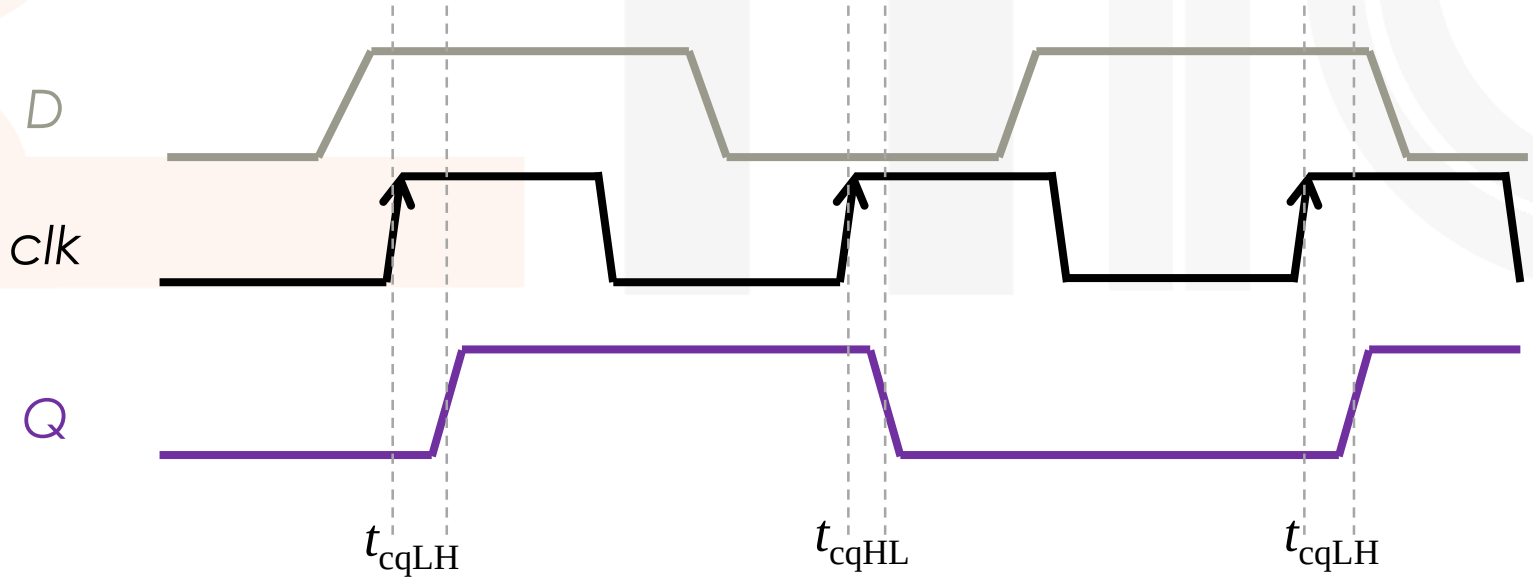


# Timing Parameters - $t_{cq}$

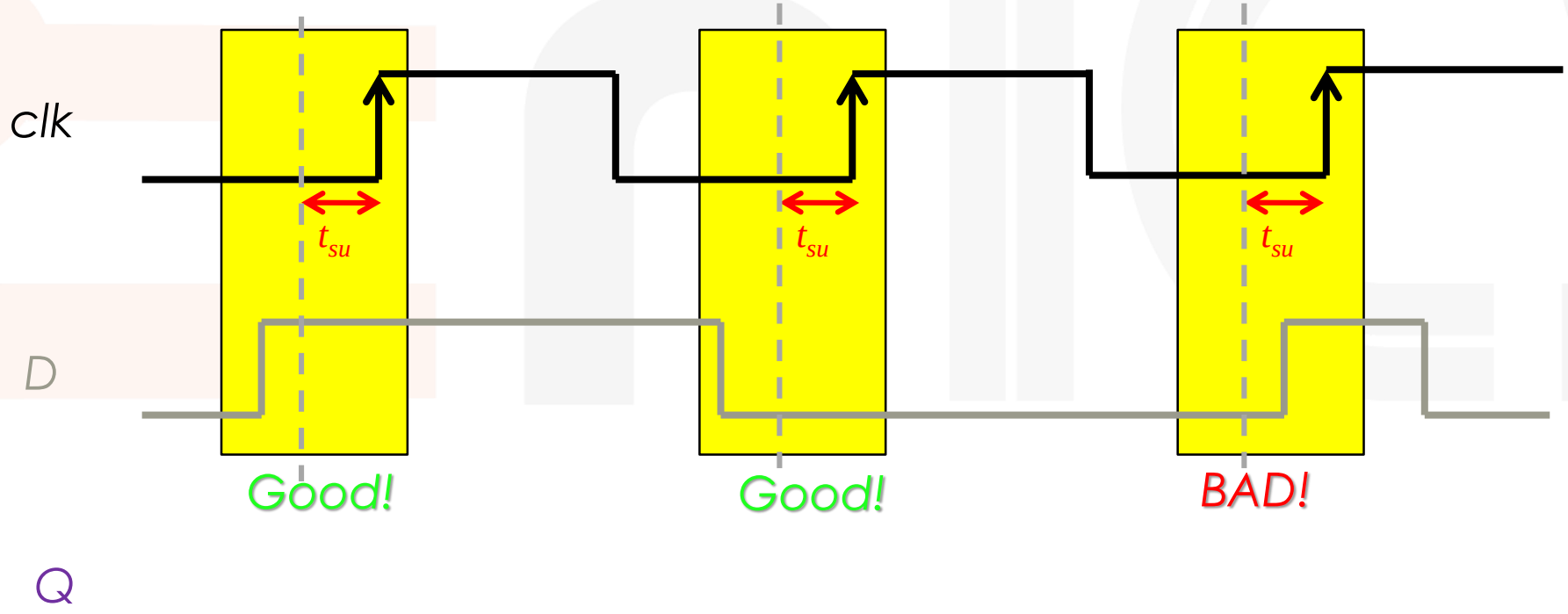
- $t_{cq}$ is the time from the clock edge until the data appears at the output.
- The $t_{cq}$ for rising and falling outputs is different.



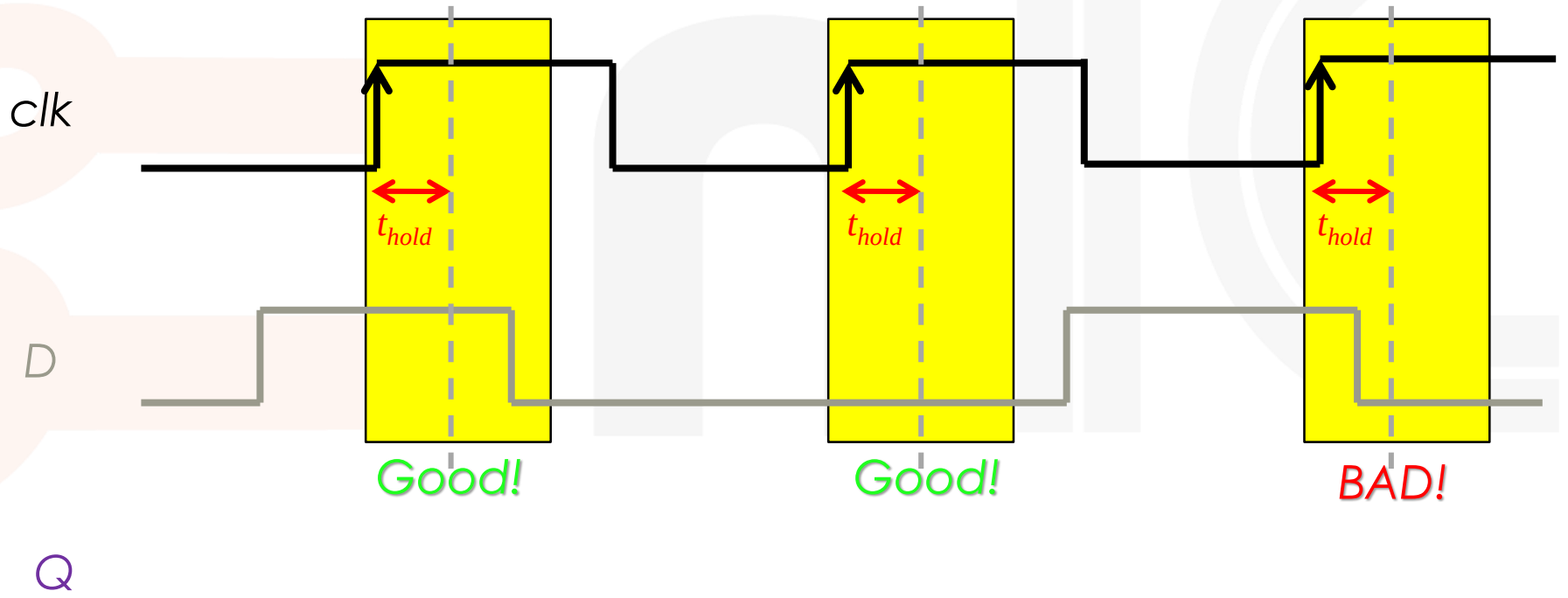
# Timing Parameters - $t_{setup}$

- $t_{setup}$ - Setup time is the time the data has to arrive ***before the clock*** to ensure correct sampling.



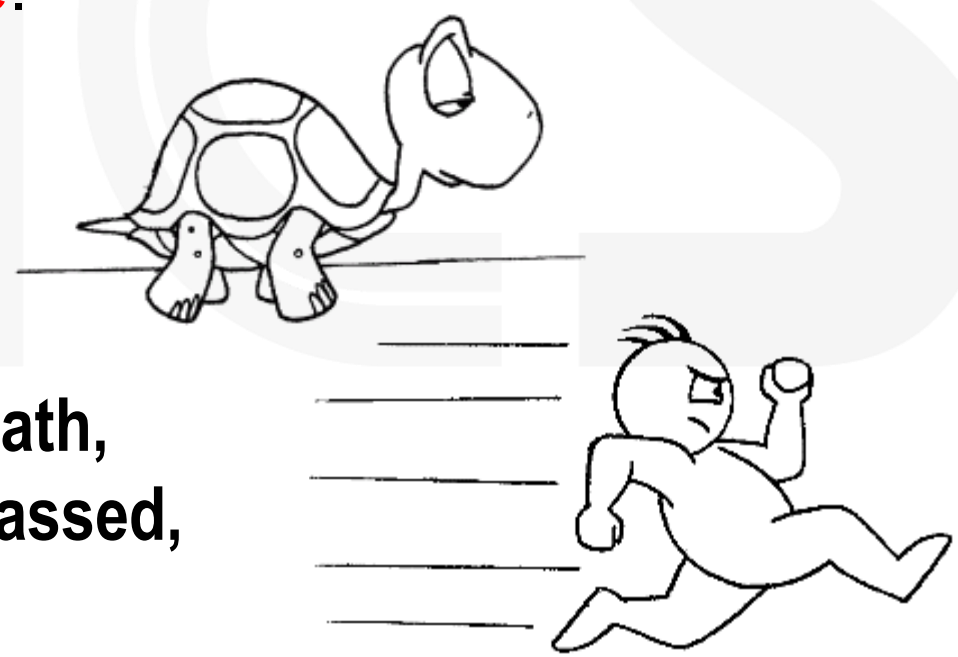
# Timing Parameters - $t_{hold}$

- $t_{hold}$ - Hold time is the time the data has to be stable *after the clock* to ensure correct sampling.

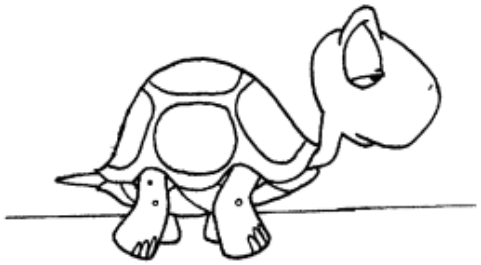


# Timing Constraints

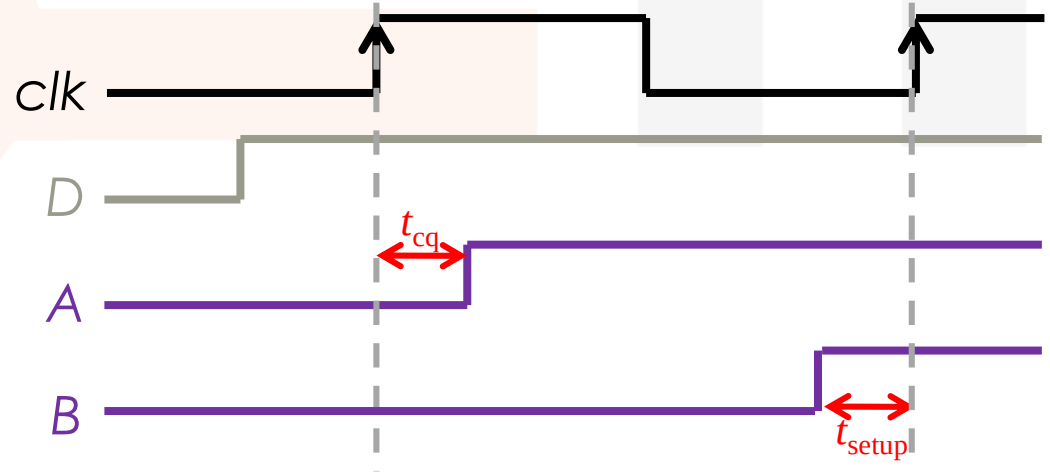
- **There are two main problems that can arise in synchronous logic:**
  - Max Delay: The data doesn't have enough time to pass from one register to the next ***before the next clock edge***.
  - Min Delay: The data path is so short that it passes through several registers ***during the same clock cycle***.
- **Max delay violations are a result of a slow data path, including the registers' $t_{setup}$, therefore it is often called the "Setup" path.**
- **Min delay violations are a result of a short data path, causing the data to change before the $t_{hold}$ has passed, therefore it is often called the "Hold" path.**

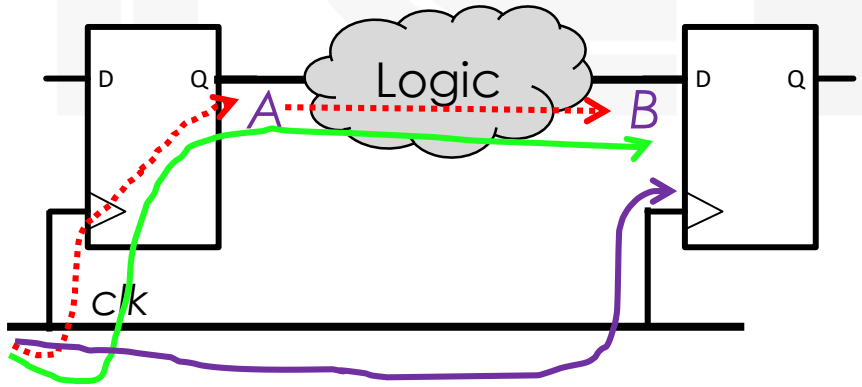


# Setup (Max) Constraint

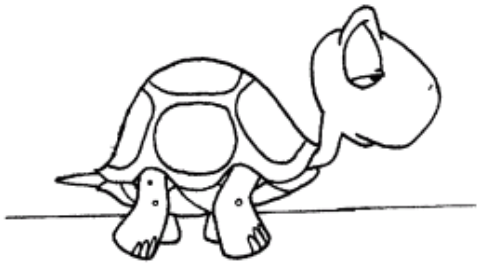


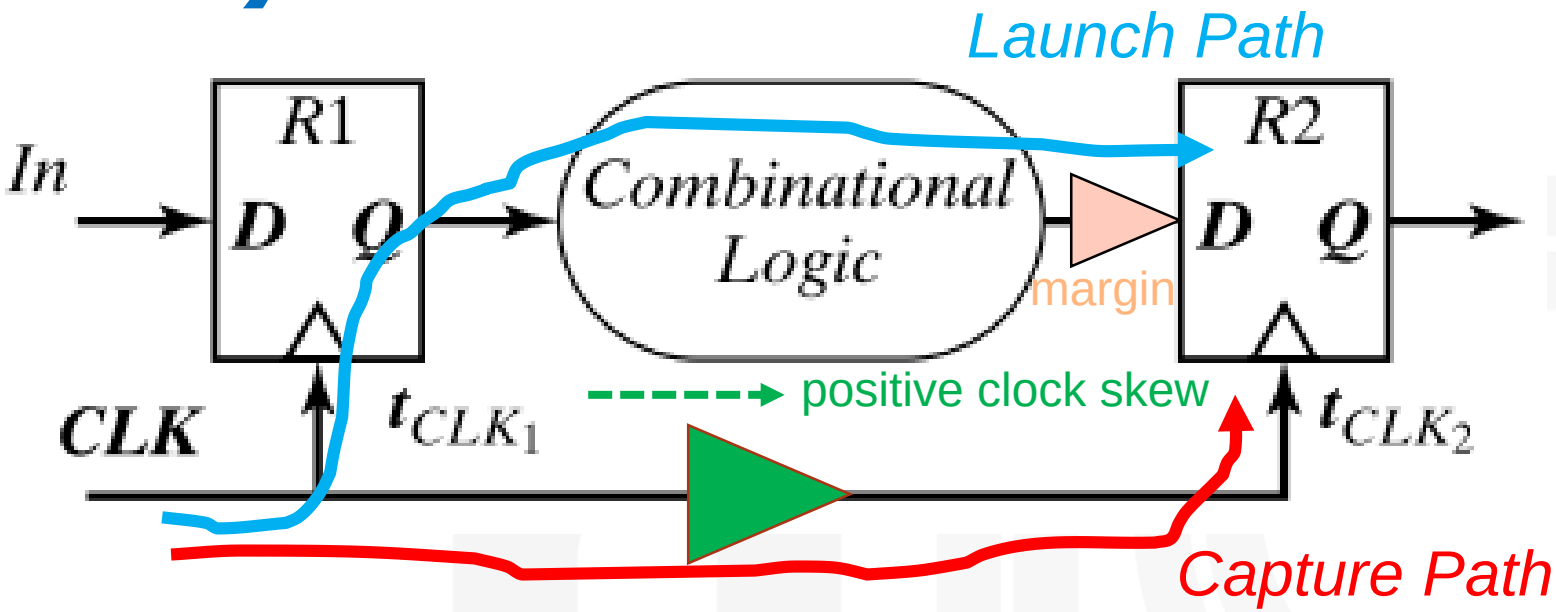
- **Let's see what makes up our clock cycle:**
  - After the clock rises, it takes $t_{cq}$ for the data to propagate to point A.
  - Then the data goes through the delay of the logic to get to point B.
  - The data has to arrive at point B, $t_{setup}$ before the next clock.
- **In general, our timing path is a race:**
  - Between the Data Arrival, starting with the *launching* clock edge.
  - And the Data Capture, one clock period later.



# Setup (Max) Constraint





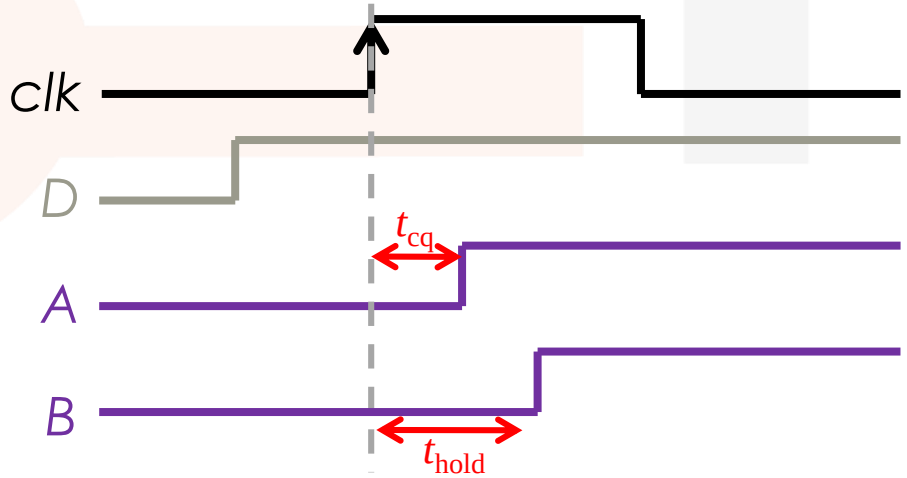

$$T > t_{cq} + t_{logic} + t_{setup}$$

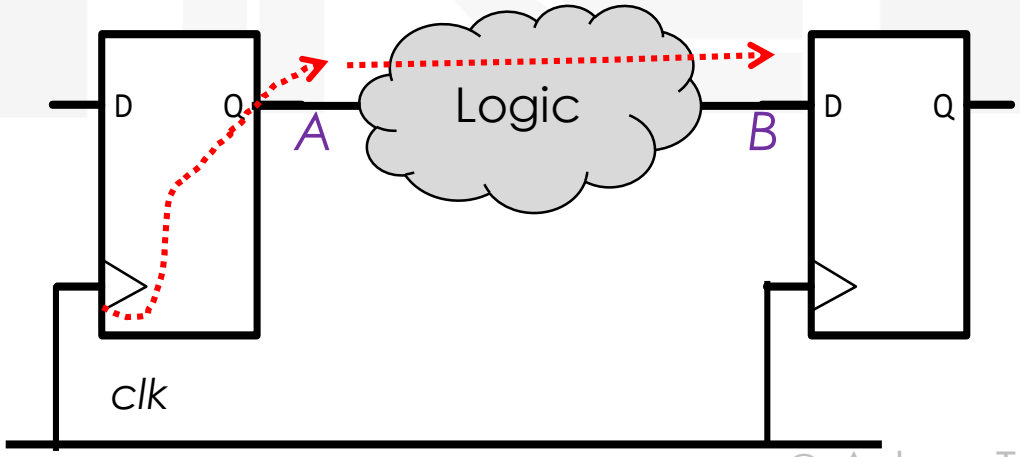
Adding in clock skew and other guardbands:

$$T + \delta_{skew} > t_{cq} + t_{logic} + t_{setup} + \delta_{margin}$$

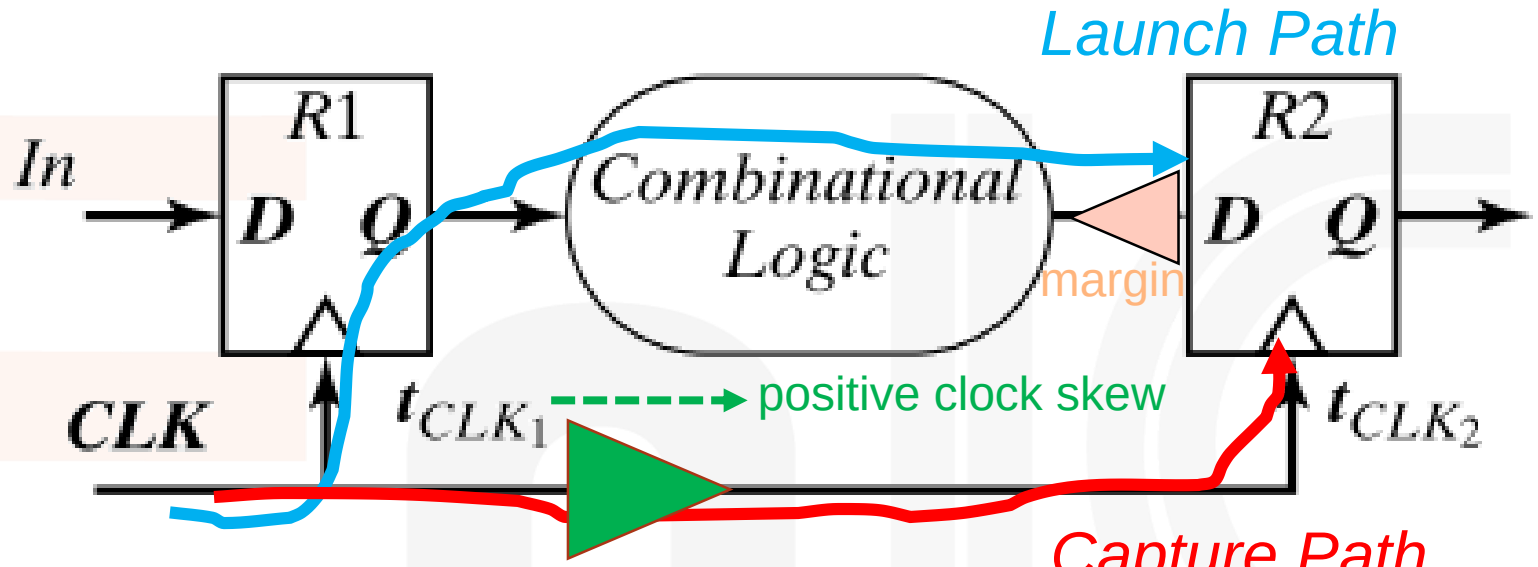
# Hold (Min) Constraint

- **Hold problems occur due to the logic changing before $t_{hold}$ has passed.**
- **This is not a function of cycle time – it is relative to a single clock edge!**
- **Let's see how this can happen:**
  - The clock rises and the data at A changes after $t_{cq}$.
  - The data at B changes $t_{pd}$(logic) later.
  - Since the data at B had to stay stable for $t_{hold}$ after the clock (for the second register), the change at B has to be at least $t_{hold}$ **after** the clock edge.



# Hold (Min) Constraint

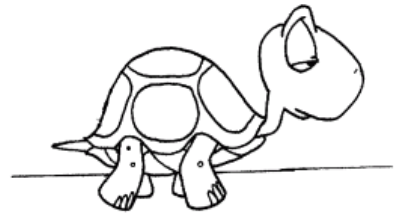



$$t_{cq} + t_{logic} > t_{hold}$$

Adding in clock skew and other guardbands:

$$t_{cq} + t_{logic} - \delta_{margin} > t_{hold} + \delta_{skew}$$

# Summary



- **For *Setup* constraints, the data has to propagate fast enough to be captured by the next clock edge:**
  - This sets our maximum frequency.
  - If we have setup failures, we can always just **slow down the clock**.

$$t_{launch} < T + t_{capture}$$

$$T + \delta_{skew} > t_{cq} + t_{logic} + t_{setup} + \delta_{margin}$$

- **For *Hold* constraints, the data path delay has to be long enough so it isn't accidentally captured by the same clock edge:**
  - This is **independent of clock period**.
  - If there is a hold failure, you can throw your chip away!

$$t_{launch} > t_{capture}$$

$$t_{cq} + t_{logic} - \delta_{margin} > t_{hold} + \delta_{skew}$$

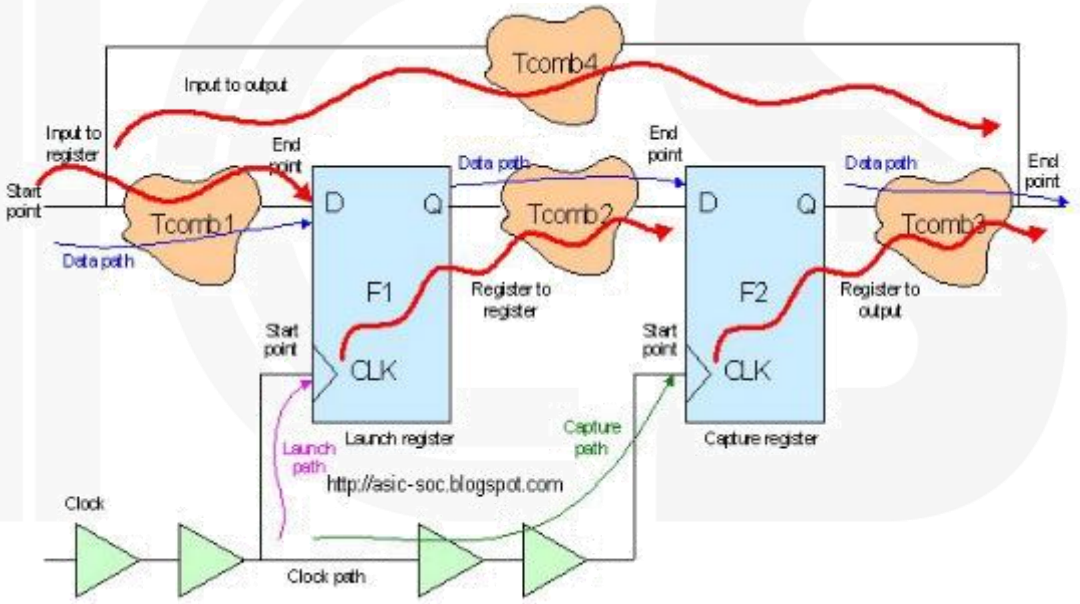
# Static Timing Analysis

Or why and how to calculate slack.

This section is heavily based on Rob Rutenbar's "From Logic to Layout", Lecture 12 from 2013. For a better ☺ and more detailed explanation, do yourself a favor and go see the original!

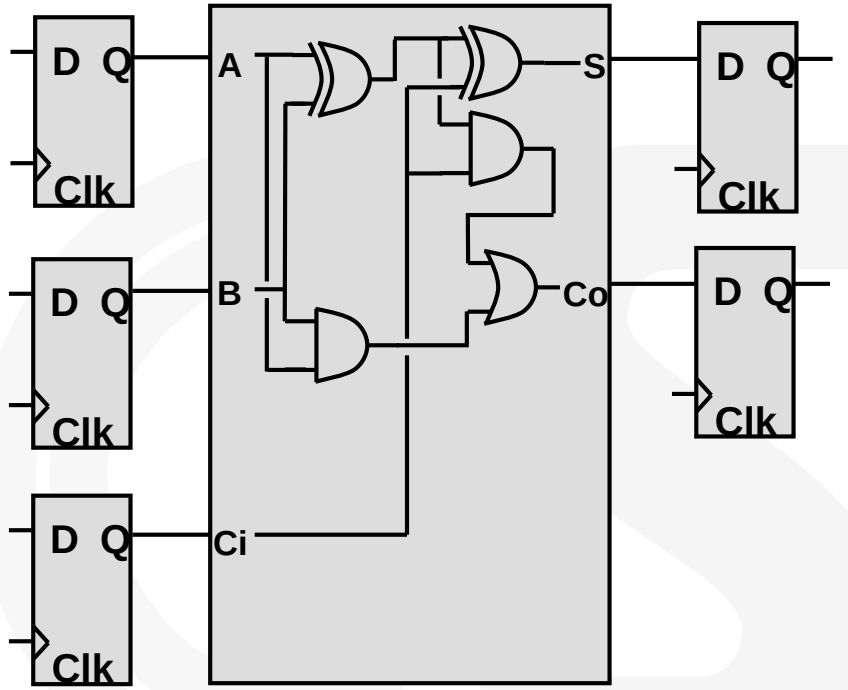
# Static Timing Analysis (STA)

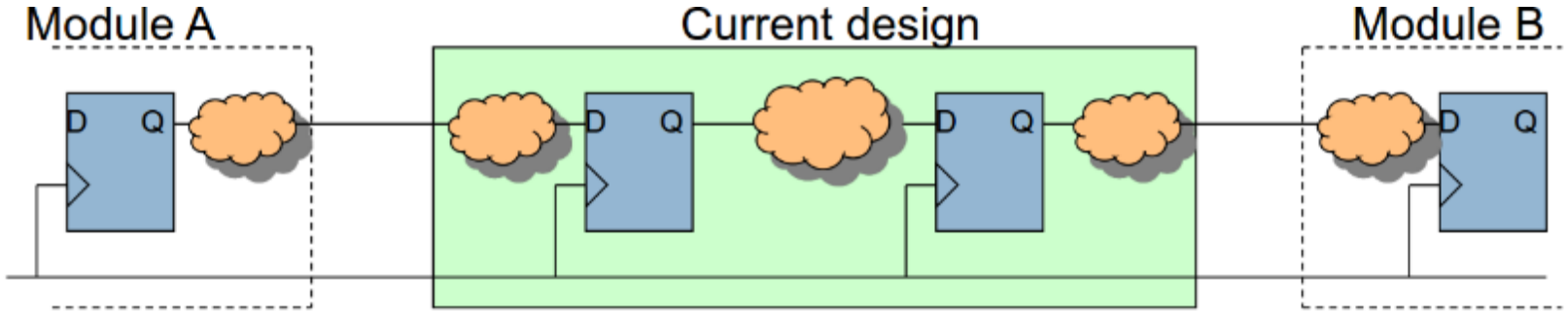
- **STA checks the worst case propagation of *all* possible vectors for min/max delays.**
- **Advantages:**
  - Much faster than timing-driven, gate-level simulation
  - Exhaustive, i.e., **every (constrained) timing path is checked**.
  - Vector generation NOT required
- **Disadvantages:**
  - Proper circuit functionality is NOT checked
  - Must define timing requirements/exceptions (garbage in → garbage out!)
- **Limitations:**
  - Only useful for *synchronous* design
  - Cannot analyze combinatorial feedback loops
    - e.g., a flip-flop created out of basic logic gates
  - Cannot analyze asynchronous timing issues
    - Such as clock domain crossing
  - Will not check for glitching effects on asynchronous pins
    - Combinatorial logic driving asynch (set/reset) pins of sequential elements will not be checked for glitching



# Timing Paths

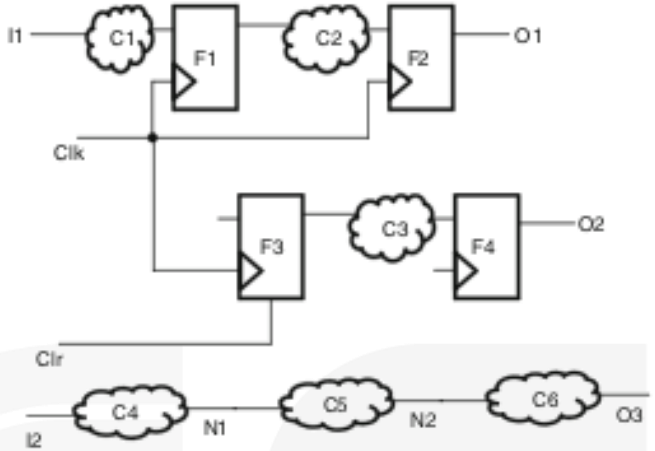
- **A path is a route from a Startpoint to an Endpoint.**
- **Startpoint (SP)**
  - Clock pins of the flip flops
  - Input ports , a.k.a Primary Inputs (PI)
- **Endpoints (EP)**
  - Input pins of the flip flops (except the clock pins)
  - Output ports, a.k.a Primary Outputs (PO)
  - Memories / Hard macros
- **There can be:**
  - Many paths going to any one endpoint
  - Many paths for each start-point and end-point combination

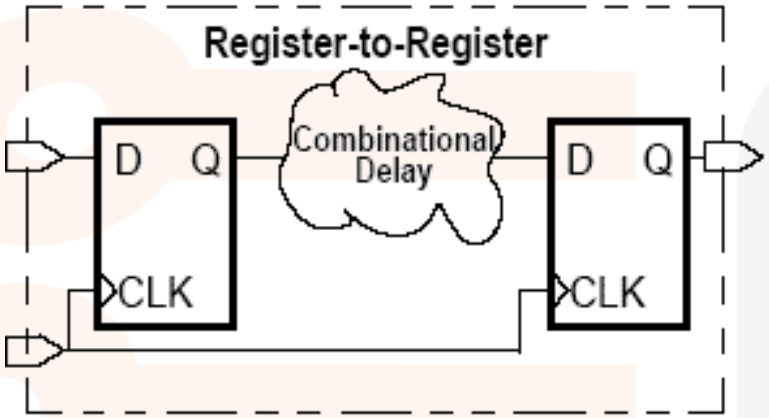


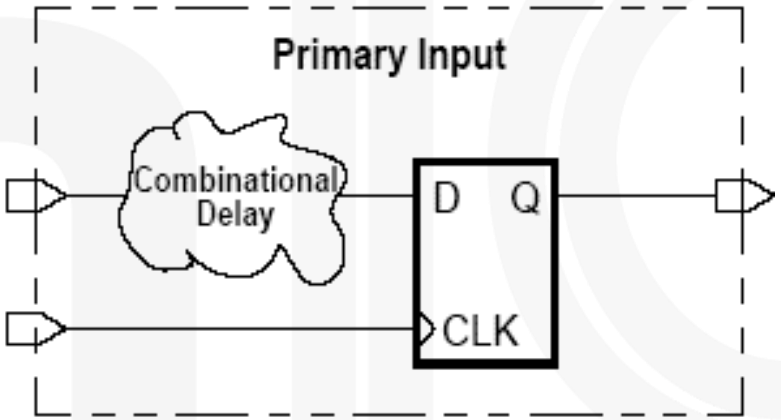



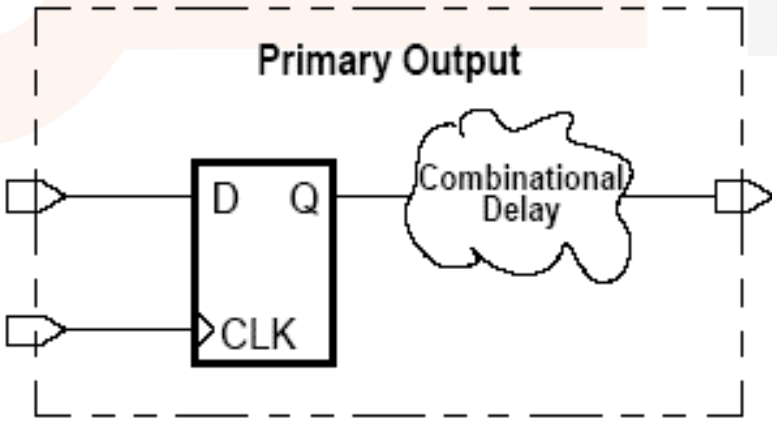
# Static Timing Analysis

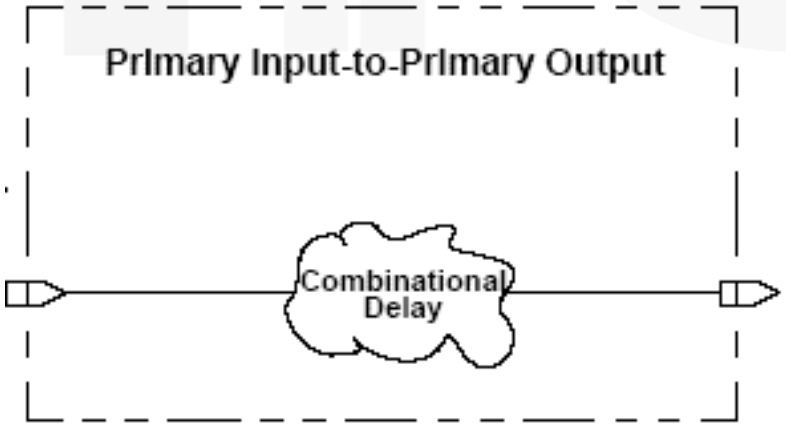
- **Four categories of timing paths**
  - Register to Register (reg2reg)
  - Register to Output (reg2out)
  - Input to Register (in2reg)
  - Input to Output (in2out)



# Goals of Static Timing Analysis

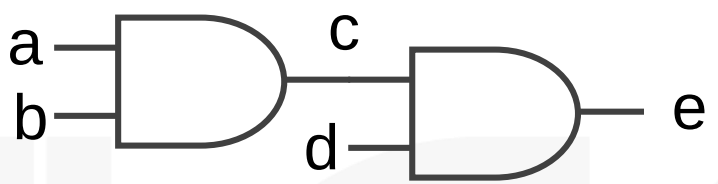
- **Verify max delay and min delay constraints are met for all paths in a design.**
  - Start with a Gate-Level Netlist.
  - Timing Models are provided for every gate in the library.
  - Static Timing Analysis needs to report if any path violates the max/min delay constraints.
- **But is this enough?**
  - No!
  - We want to know *all* the paths that violate the timing constraints.
  - In fact, we want to know the timing of *all* paths reported *in order* of length.
  - And we want to know *where* the problems are so we can go about fixing them.
- **Let's see the basic idea of how this can be done.**

# Some basic assumptions

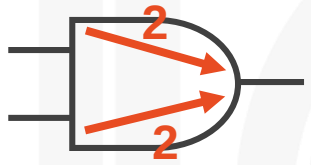
- **Our design is *synchronous***
  - In addition, we will only be showing how to deal with combinational elements and max delay constraints.
- **We will assume a *pin-to-pin* delay model**
  - In other words, each gate has a single, constant delay from input to output.
  - In the real world, gate delay is affected by many factors, such as gate type, loading, waveform shape, transition direction, particular pin, and random variation.
  - As we saw earlier, a real design gets all this data from the .lib files.
- **We will take a topological approach**
  - In other words, we disregard the logical functionality of the gates and therefore, consider all paths, though some of them cannot logically happen.
  - More on this later…

# Simple path representation

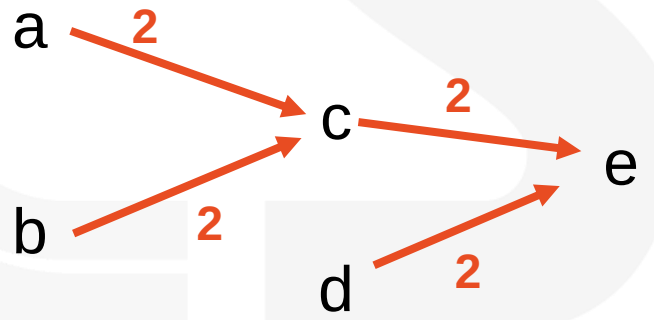
- **Let's say we have the following circuit:**



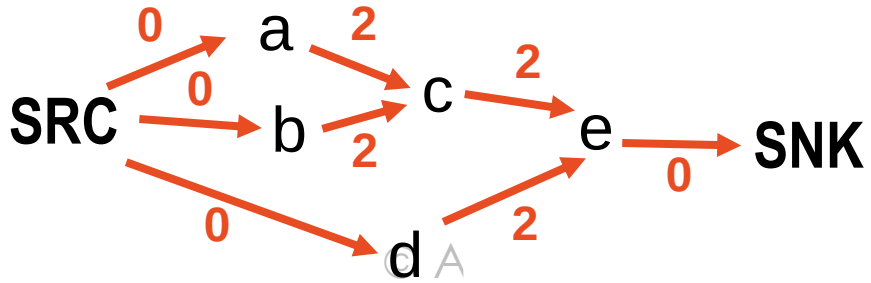

  - And the timing model of our AND gate is:




- **We will build a graph:**
  - Vertices: Wires, 1 per gate output and 1 for each SP and EP.
  - Edges: Gates, input pin to output pin, 1 edge per input with a delay for each edge.

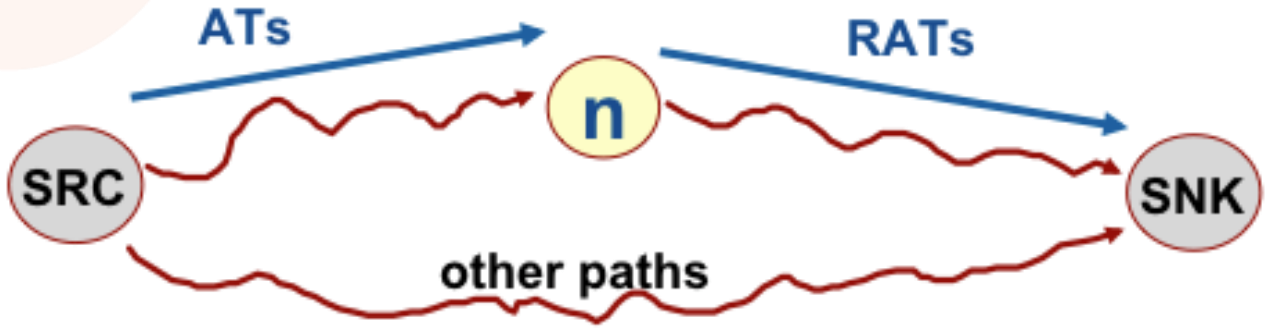



- **Finally, add Source/Sink Nodes:**
  - 0-weight edge to each SP and from each EP.
  - That way all paths start and end at a single node.



# Node oriented timing analysis

- **If we would *enumerate* every path, we would quickly get *exponential explosion* in the number of paths.**
- **Instead, we will use *node-oriented* timing analysis**
  - For *each node*, find the *worst delay* to the node along *any path*.
- **For this, we need to define two important values:**
  - **Arrival Time at a node (AT)**: the longest path from the source to the node.
  - **Required Arrival Time at node (RAT)**: the latest time the signal *is allowed* to leave the node to make it to the sink in time.

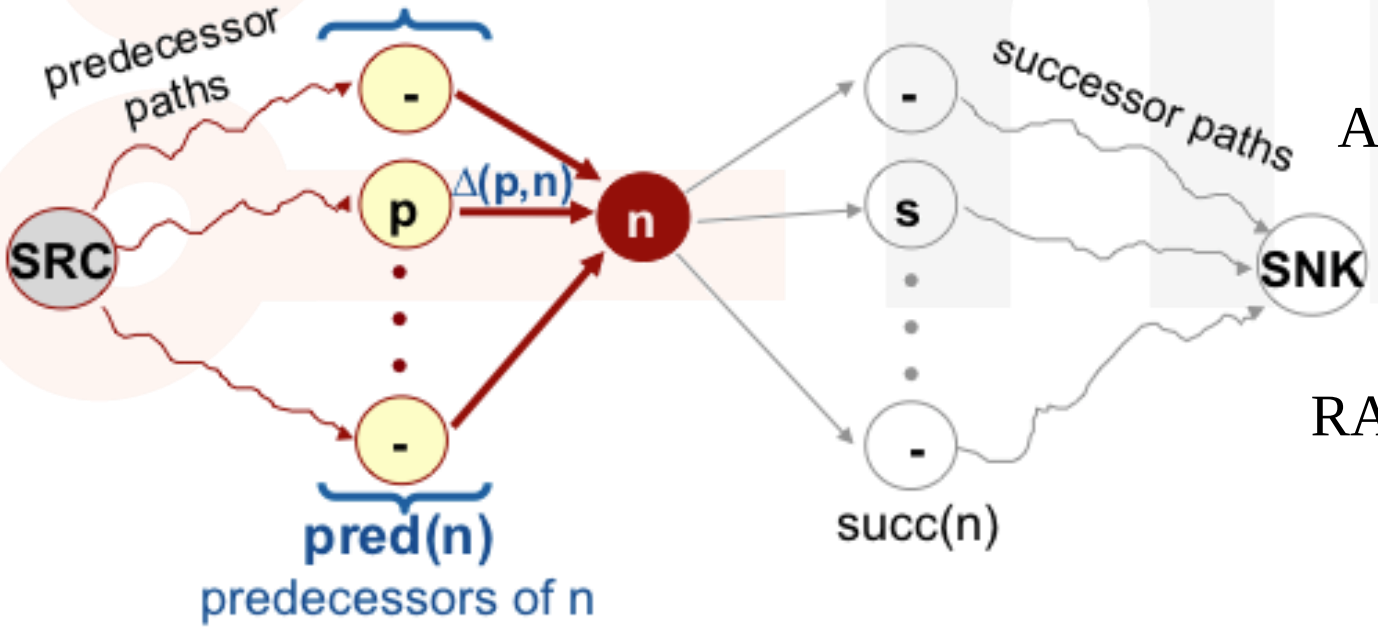



**Slack at node *n* is defined as:**

$$\text{Slack}(n) = \text{RAT}(n) - \text{AT}(n)$$

# How do we compute ATs and RATs?

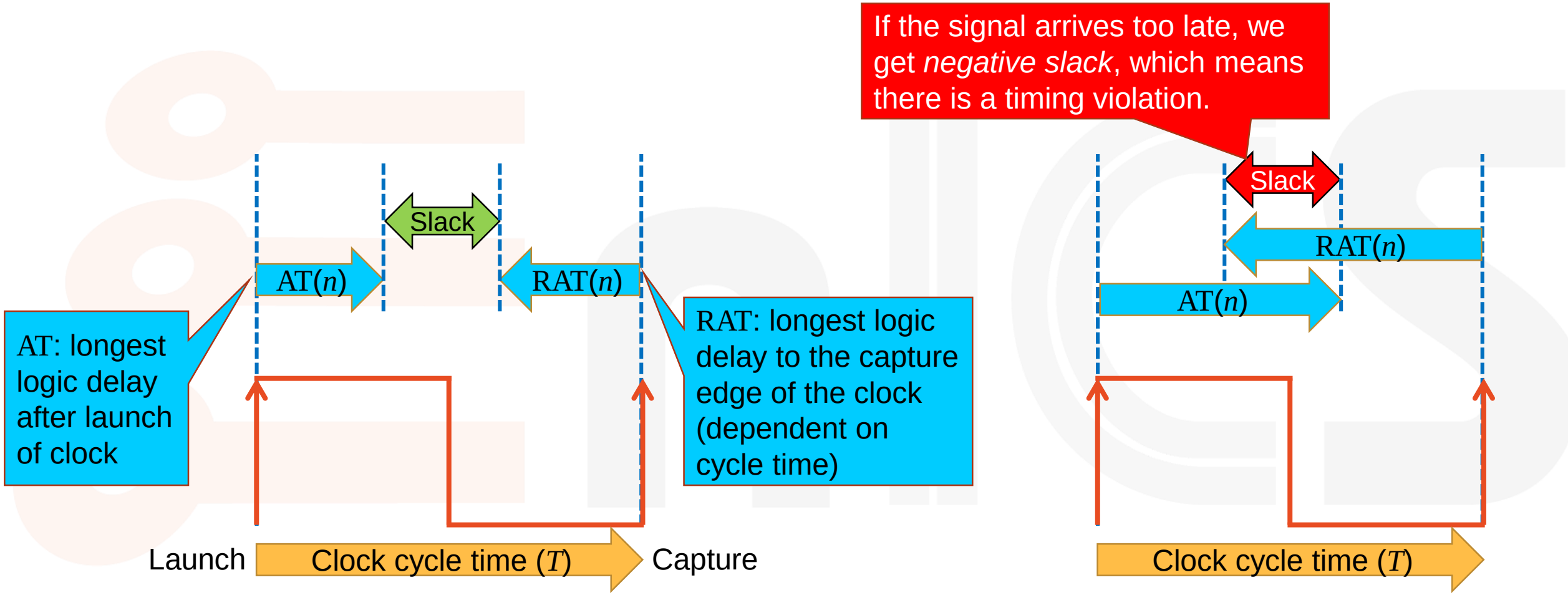
- **Recursively!**
  - The **Arrival Time** at a node is just the <u>maximum</u> of the ATs at the *predecessor* nodes <u>plus</u> the *delay from that node*.
  - The **Required Arrival Time** to a node is just the <u>minimum</u> of the RATs at the *successor* nodes <u>minus</u> the *delay to that node*.




$$\mathrm{AT}(n) = \begin{cases} 0 & n = \mathrm{SRC} \\ \max\limits_{p \in \mathrm{pred}(n)} \left[ \mathrm{AT}(p) + \Delta(p,n) \right] & n \neq \mathrm{SRC} \end{cases}$$

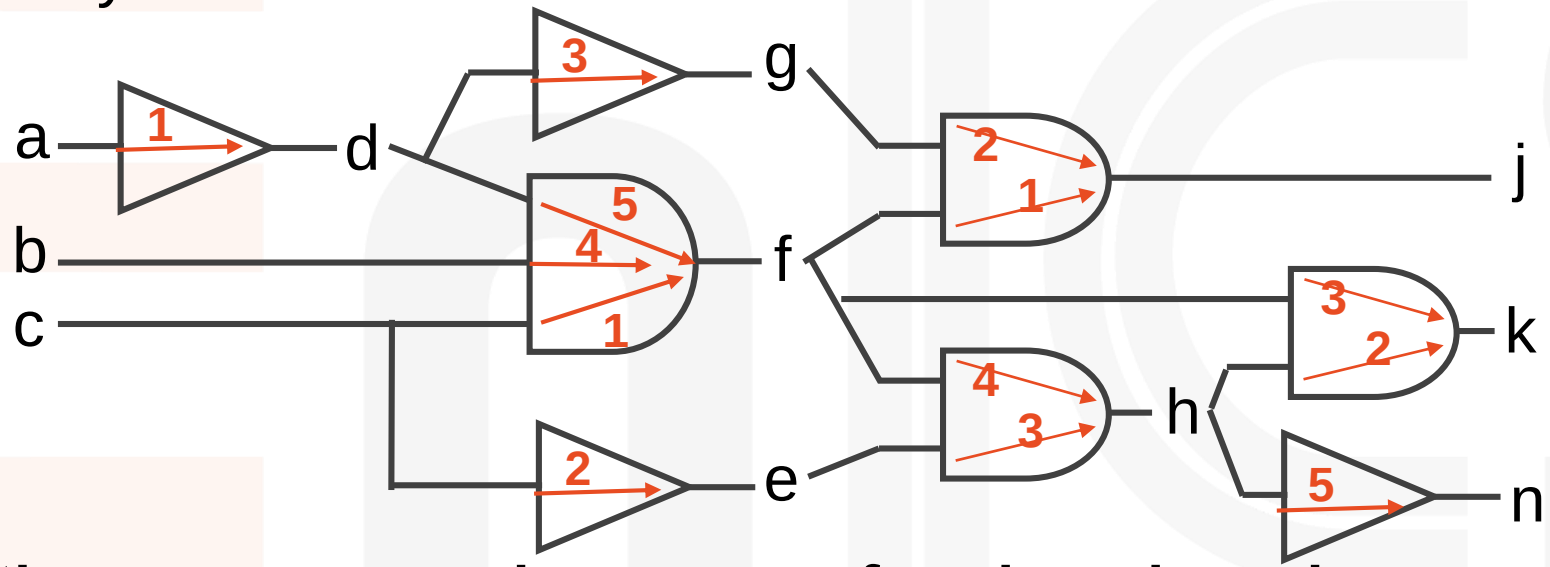
$$\mathrm{RAT}(n) = \begin{cases} T & n = \mathrm{SNK} \\ \min\limits_{s \in \mathrm{succ}(n)} \left[ \mathrm{RAT}(s) - \Delta(n,s) \right] & n \neq \mathrm{SNK} \end{cases}$$

# So let's try to understand AT, RAT, and Slack



# Now let's see an example

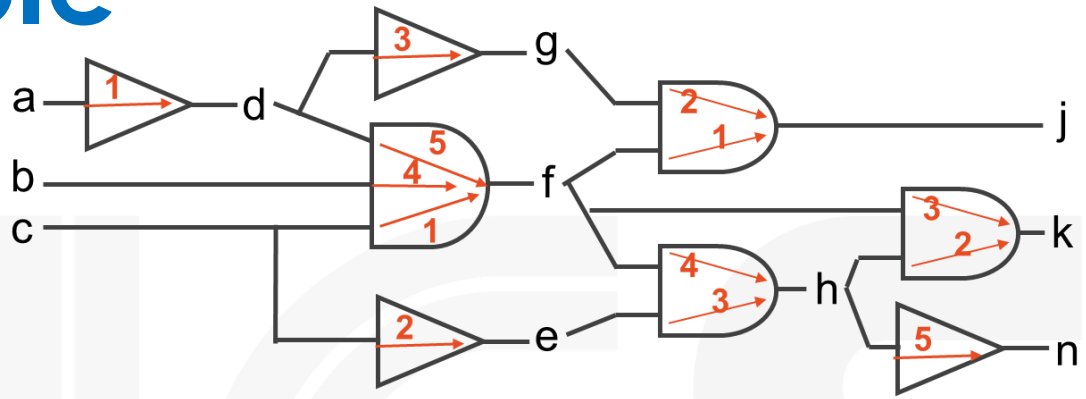
- **Just look at this path and try to find the worst path.**
  - Does it meet a cycle time of T=12 ?

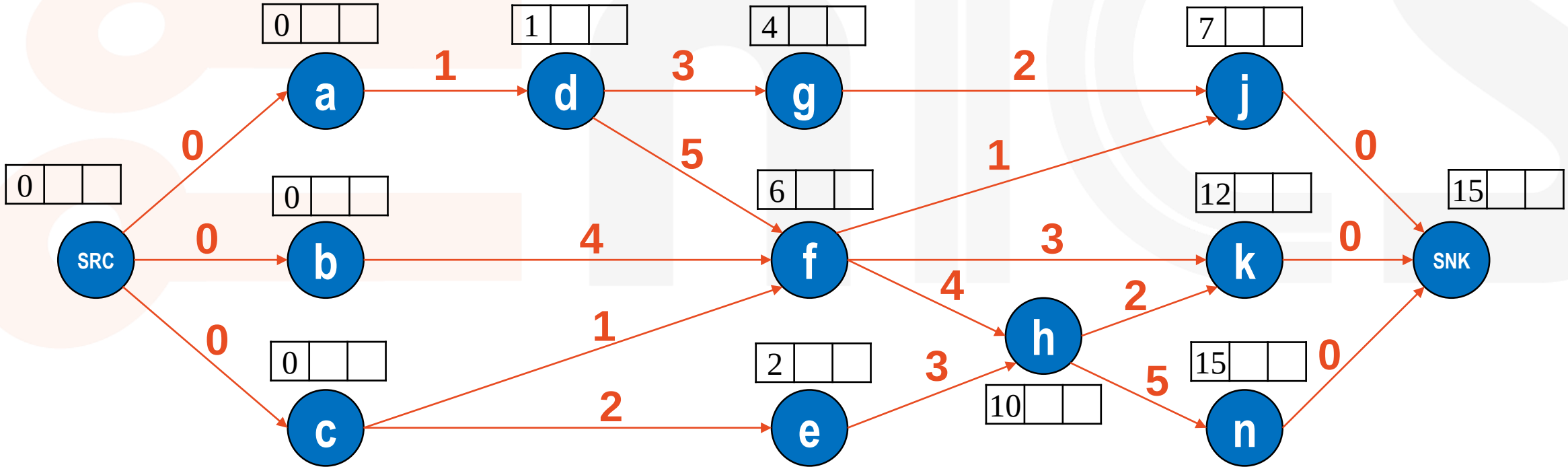



- **Now let's fill in the RAT, AT, and SLACK of each node and:**
  - Quickly find out if we meet timing
  - Figure out what the worst path is

# Now let's see an example

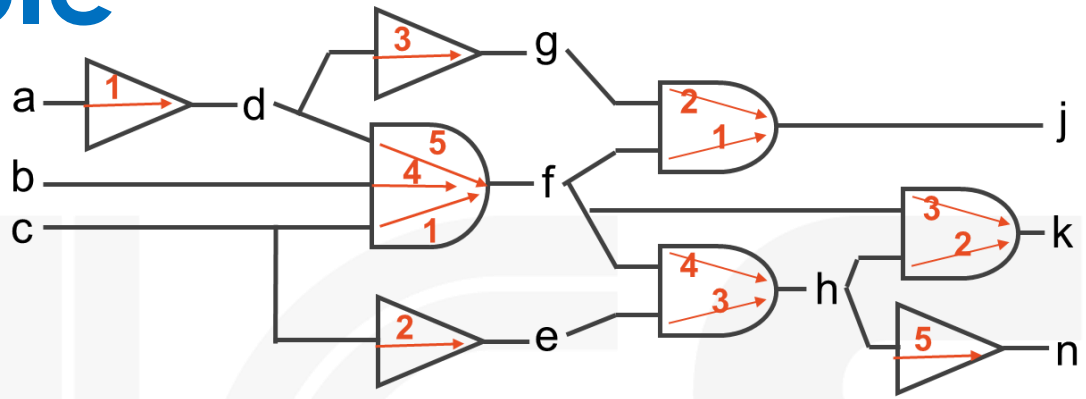
- We'll start by representing it as a directed acyclic graph (DAG)
- Next, we'll compute ATs from SRC to SNK

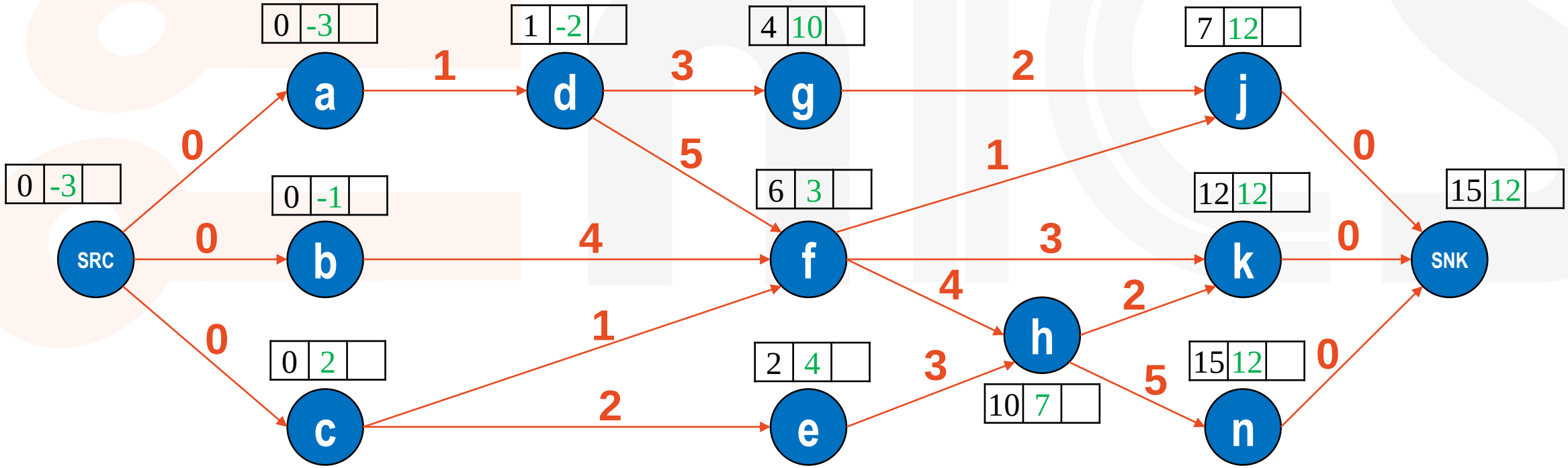


# Now let's see an example

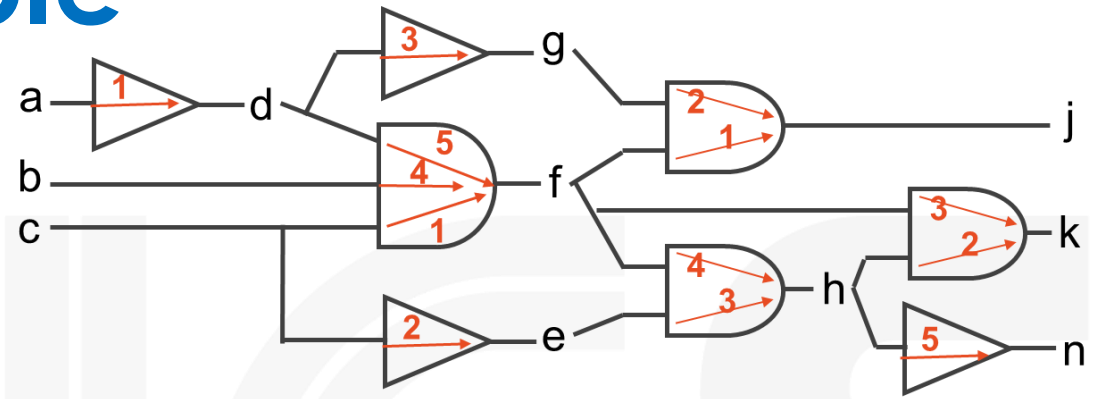
- And now **RAT** from SNK to SRC

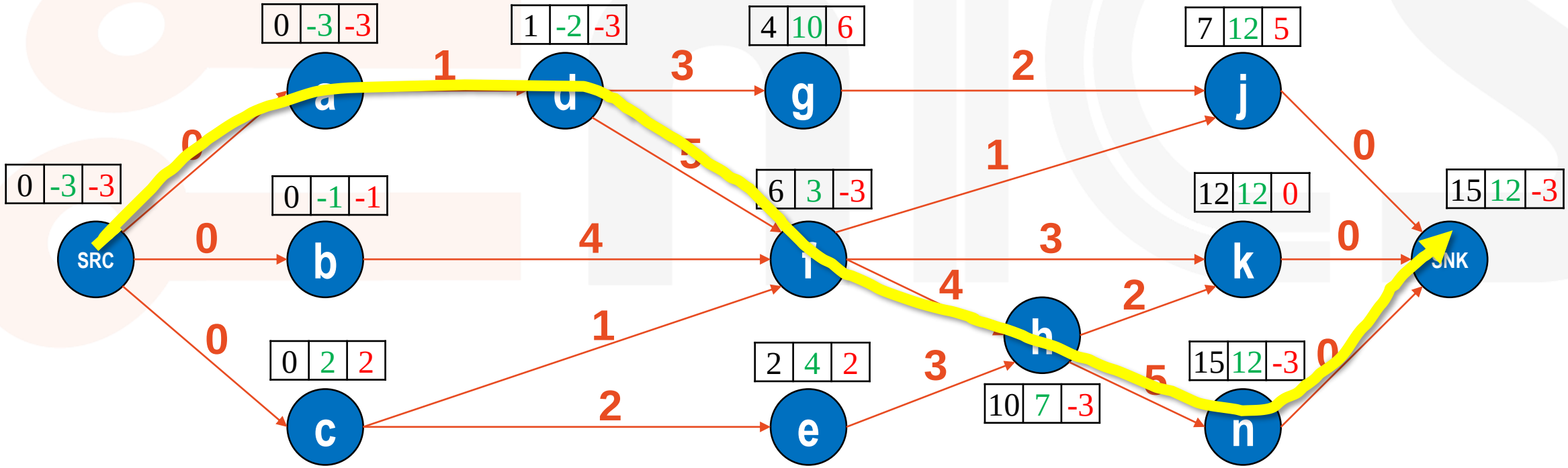


# Now let's see an example

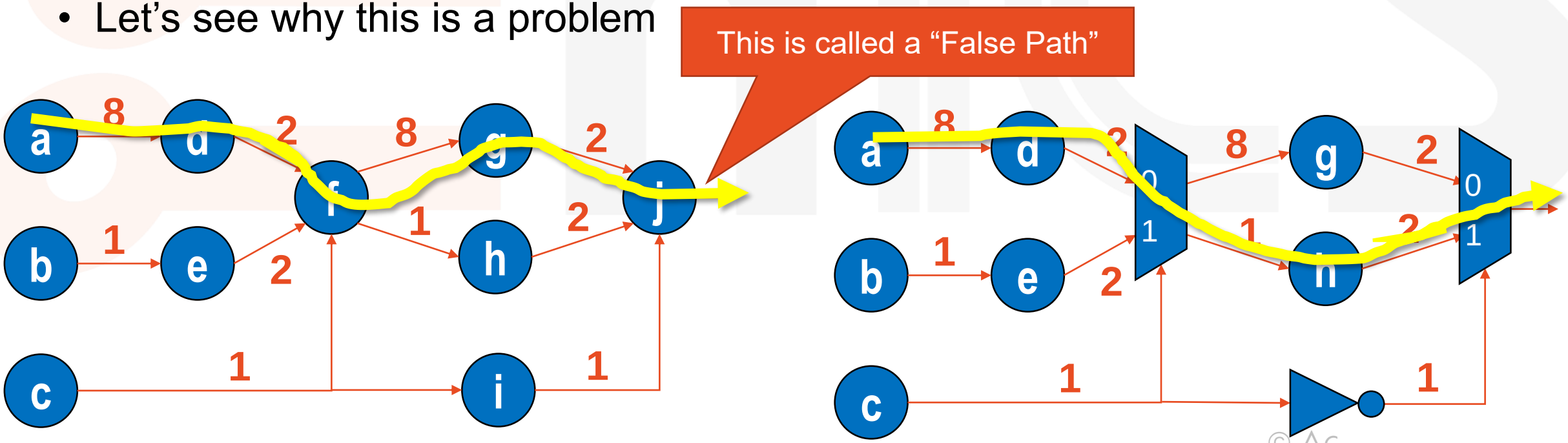
- And finally, we can calculate the slack.
- And guess what – we found the critical path!



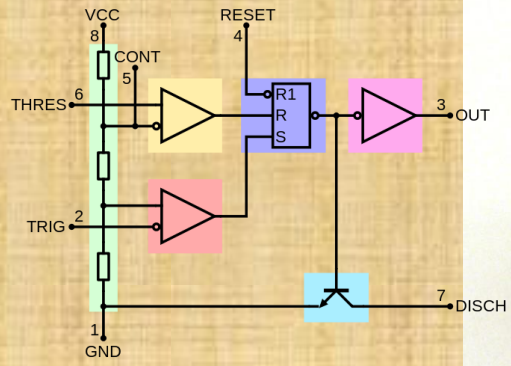
# False Paths

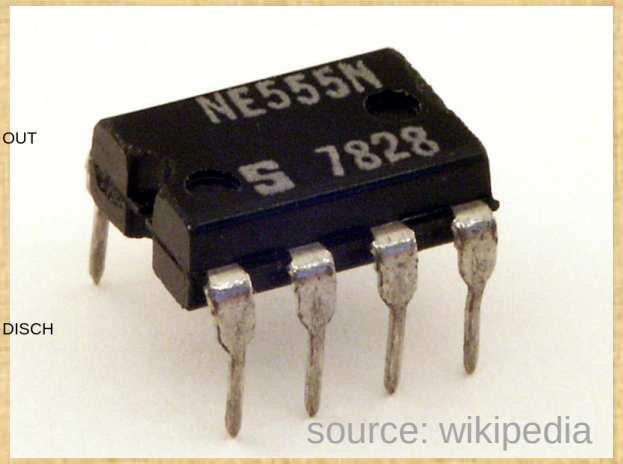
- **We saw how to find the RAT, AT and Slack at every node.**
  - All of this can be done very efficiently and be adapted for min timing, sequential elements, latch-based timing, etc.
  - Even better, we can quickly report the order of the critical paths.
- **However, this was all done topologically (i.e., without looking at logic).**
  - Let's see why this is a problem



# The Chip Hall of Fame

- **Speaking about Timing, we shouldn't forget the**



source: wikipedia

# Signetics NE555 Timer

- A simple timing chip that is still popular today.
- Release date: 1971
- 23 transistors, 16 resistors, 2 diodes
- BiPolar Process  8-pin DIP
- Can function as a timer, pulse generator or an oscillator.
- Designed by Hans Camenzind, who also introduced the Phase Locked Loop to integrated circuits (ISSCC 1969)
- Approximately 1B units were manufactured per year in 2003.

Photo: Evil Mad Scientist Laboratories

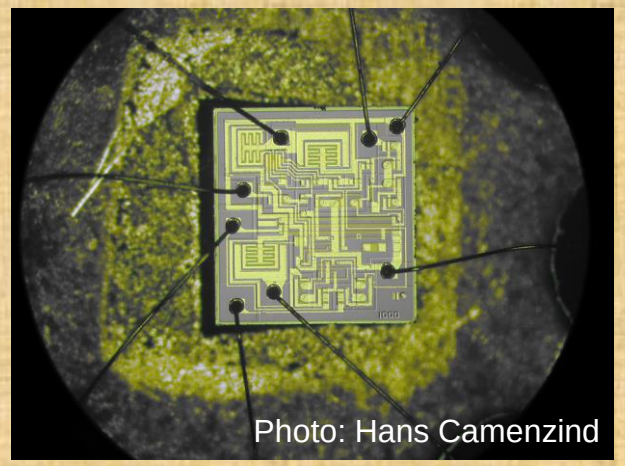
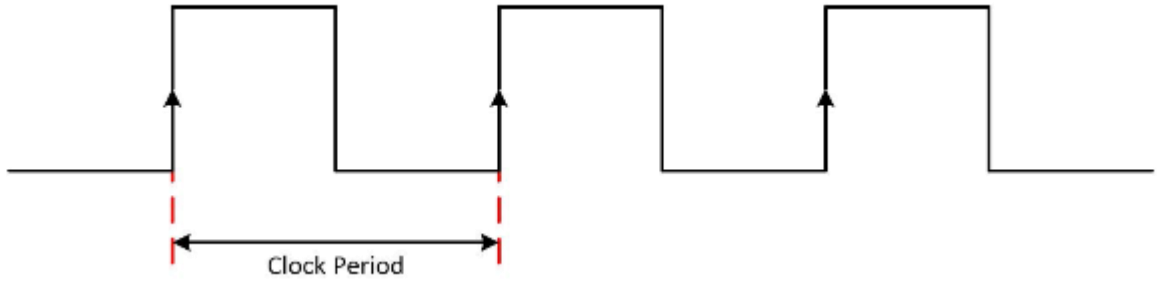


Photo: Hans Camenzind

2017 Inductee to the IEEE Chip Hall of Fame

# Design Constraints

# Timing Constraints




- **"Stupid Question":**
  - How does the STA tool know what the required clock period is?
- **Obvious Answer…**
  - We have to tell it!
  - We have to define *constraints* for the design.
  - This is usually done using the Synopsys Design Constraints (SDC) syntax, which is a superset of TCL.
- **Three main categories of timing constraints:**
  - Clock definitions
  - Modeling the world external to the chip
  - Timing exceptions

# Collections

- **So you think you know `TCL`, right?**
  - Well EDA tools sometimes use a different data structure called a "`collection`"
- **A collection is similar to a `TCL` list, but:**
  - The value of a collection is not a string, but rather a pointer, and we need to use special functions to access its values.
  - For example, if you were to run `foreach` on a collection, it would just have one element (the pointer to the collection). Instead, use `foreach_in_collection`.
  - I won't go into the specifics here (see SynopsysCommandsReference), but these are some of the collection accessing functions:

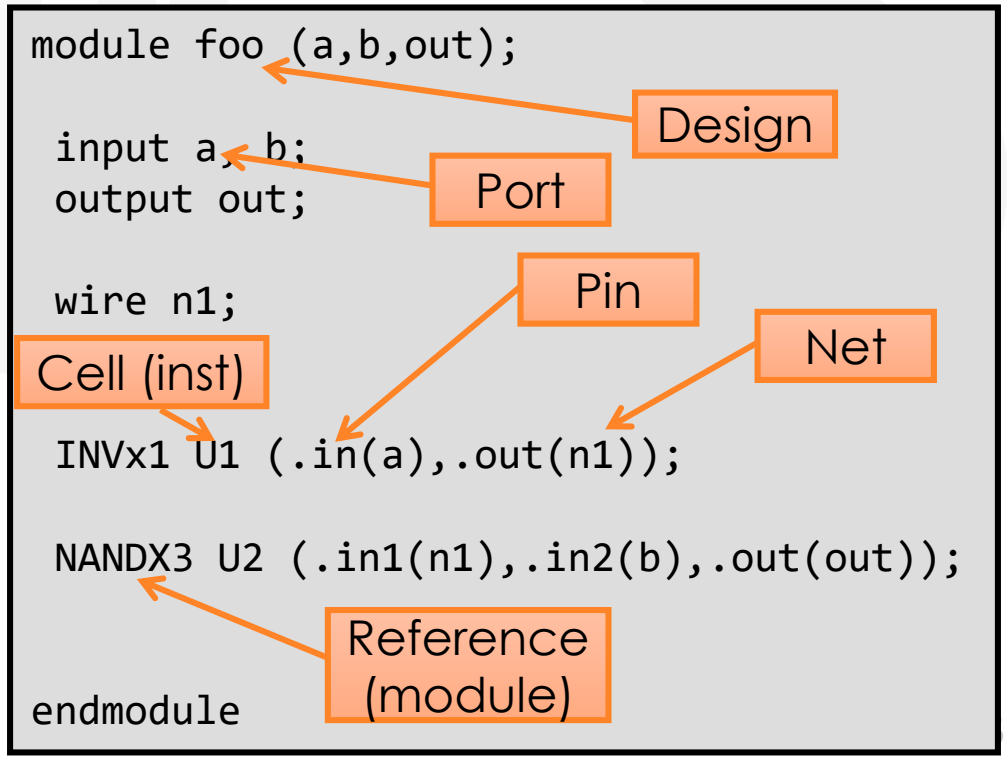
```
foreach_in_collection
index_collection
sizeof_collection
sort_collection
```

```
filter_collection
add_to_collection
compare_collections
```

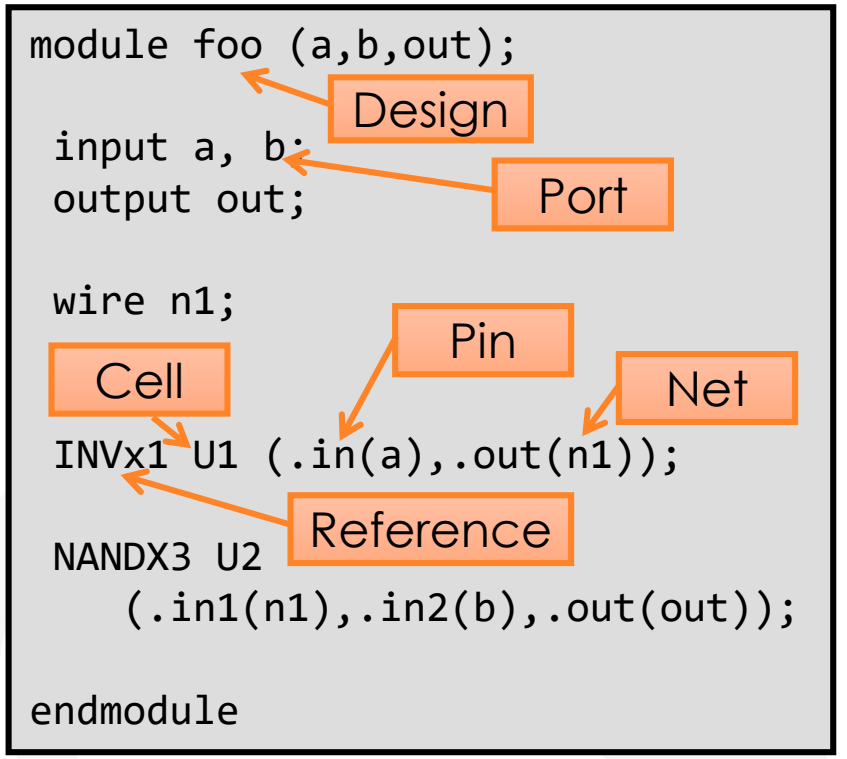
```
copy_collection
get_object_name
remove_from_collection
```

# Design Objects

- **Design:** A circuit description that performs one or more logical functions (i.e Verilog module).
- **Cell:** An instantiation of a design within another design (i.e Verilog instance).
  - Called an ***inst*** in Stylus Common UI.
- **Reference:** The original design that a cell "points to" (i.e Verilog sub-module)
  - Called a *module* in Stylus Common UI.
- **Port:** The *input*, *output* or *inout* port of a **Design**.
- **Pin:** The *input*, *output* or *inout* pin of a **Cell** in the **Design**.
- **Net:** The wire that connects **Ports** to **Pins** and/or **Pins** to each other.
- **Clock:** **Port** of a **Design** or **Pin** of a **Cell** explicitly defined as a clock source.
  - Called a *clock_tree* in Stylus Common UI.



# SDC helper functions

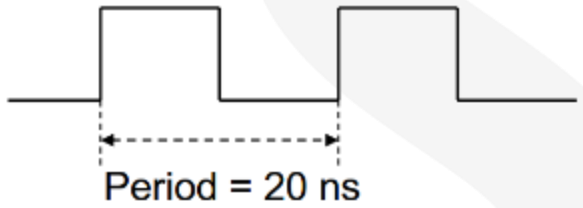



- **Before starting with constraints, let's look at some very useful built in commands:**
  - Note that all of these return `collections` and not `TCL lists`!
  - These will only work after design elaboration!
- **"`get`" commands:**
  - `[get_ports string]` – returns all ports that match string.
  - `[get_pins string]` – returns all cell/macro pins that match string.
  - `[get_nets string]` – returns all nets that match string.
    - Note that adding the `-hier` option will search hierarchically through the design.
- **"`all`" commands:**
  - `[all_inputs]` – returns all the primary inputs (ports) of the block.
  - `[all_outputs]` – returns all the primary outputs (ports) of the block.
  - `[all_registers]` – returns all the registers in the block.

# Clock Definitions

- **To start, we must define a clock:**
  - Where does the clock come from? (i.e., input port, output of PLL, etc.)
  - What is the clock period? (=operating frequency)
  - What is the duty-cycle of the clock?

```
create_clock –period 20 –name my_clock [get_ports clk]
```




- **Can there be more than one clock in a design?**
  - Yes, but be careful about clock domain crossings! (…more later)
  - If a clock is produced by a clock divider, define a "generated clock":

```
create_generated_clock –name gen_clock \
       -source [get_ports clk] –divide_by 2 [get_pins FF1/Q]
```

# Clock Definitions (2)

- But during synthesis, we assume the clock is ideal, so:

```
set_ideal_network [get_ports clk]
```

- However, for realistic timing, it should have some transition:

```
set_clock_transition 0.2 [get_clocks my_clock]
```

- And we may want to add some jitter, so:

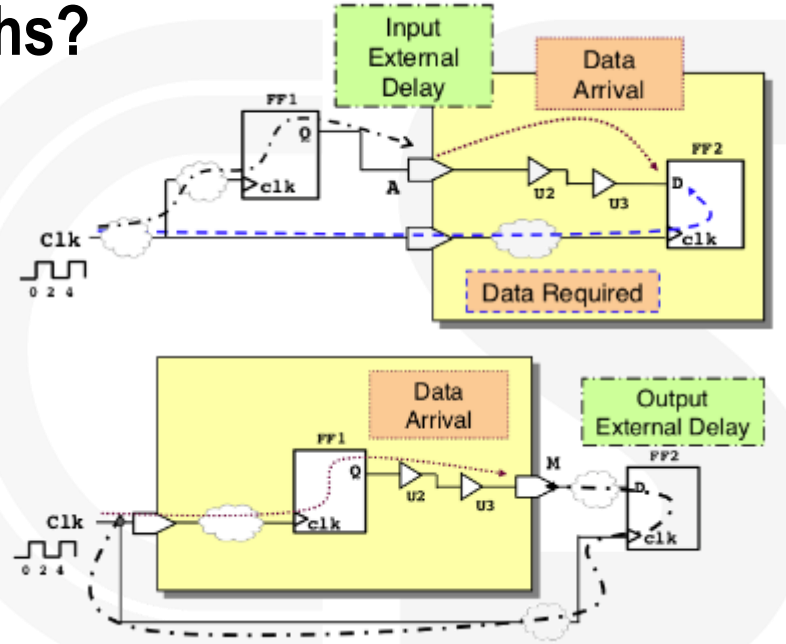
```
set_clock_uncertainty 0.2 [get_clocks my_clock]
```

- Finally, after building a clock tree, we do not want the clock to be ideal anymore, so:

```
set_propagated_clock [get_clocks my_clock]
```

# I/O Constraints

- **Now that the clock is defined, `reg2reg` paths are sufficiently constrained. However, what about `in2reg`, `reg2out`, and `in2out` paths?**
  - First, what clock toggles an I/O port?
  - And what about the time needed outside the chip?




- **Define I/O constraints:**
  - Input and output delays model the length of the path outside the block:

```
set_input_delay 0.8 -clock clk \
       [remove_from_collection [all_inputs] [get_ports clk]]
set_output_delay 2.5 -clock clk [all_outputs]
```

- **Note that a better methodology is to define a "virtual clock", but let's not confuse you too much at this point…**

# I/O Constraint (2)

- **An alternative approach is to define max delays to/from I/Os:**

```
set_max_delay 5 \
        -from [remove_from_collection [all_inputs] [get_ports clk]]
set_max_delay 5 -to [all_outputs]
```

- **Additionally, we must model the transitions on the inputs:**

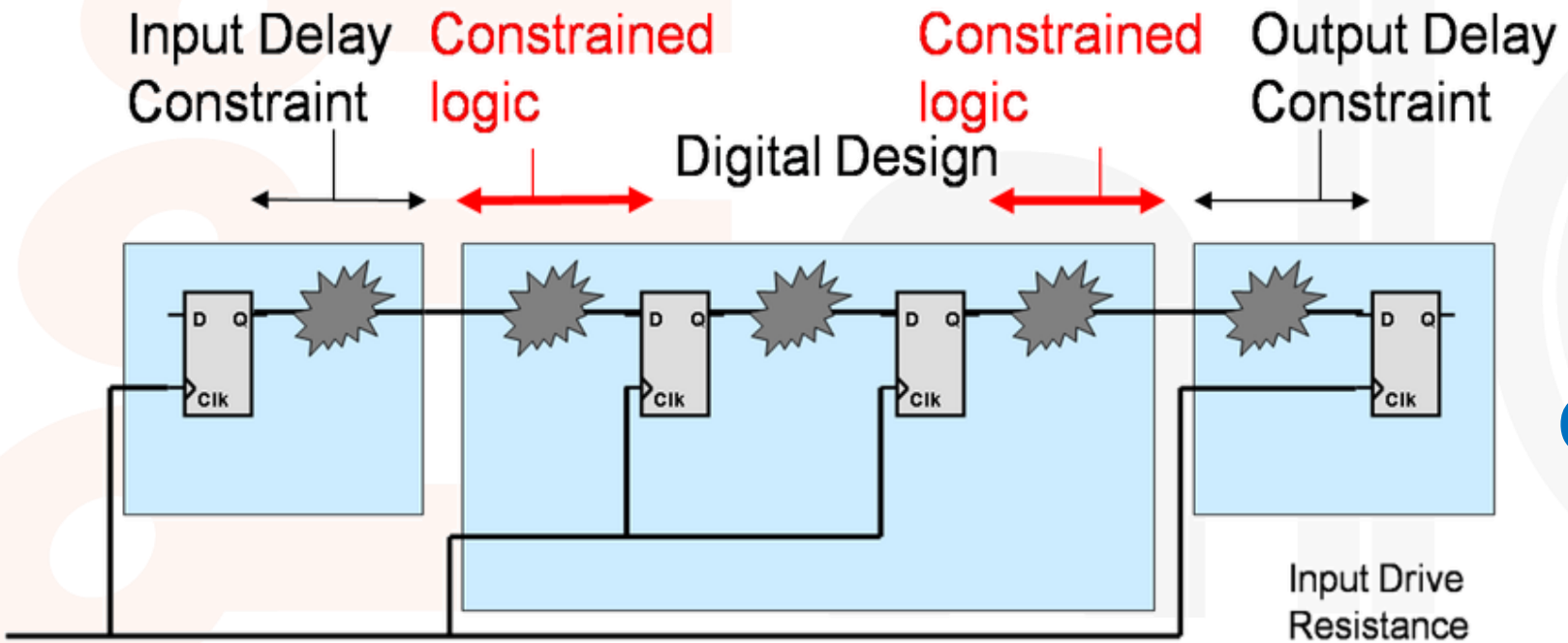
```
set_driving_cell -cell [get_lib_cells MYLIB/INV4] -pin Z \
        [remove_from_collection [all_inputs] [get_ports clk]]
```

- **And capacitance of the outputs:**

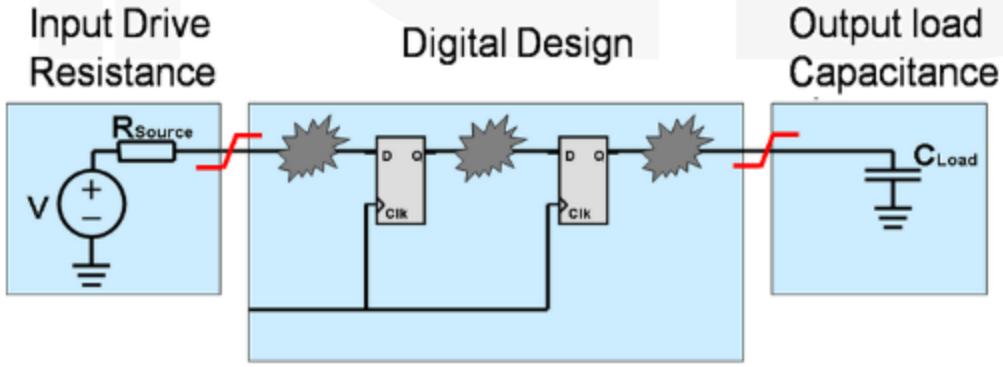
```
set_load $CIN_OF_INV [all_outputs]
```

# I/O Constraint (3)

- **Graphically, we can summarize the I/O constraints, as follows:**



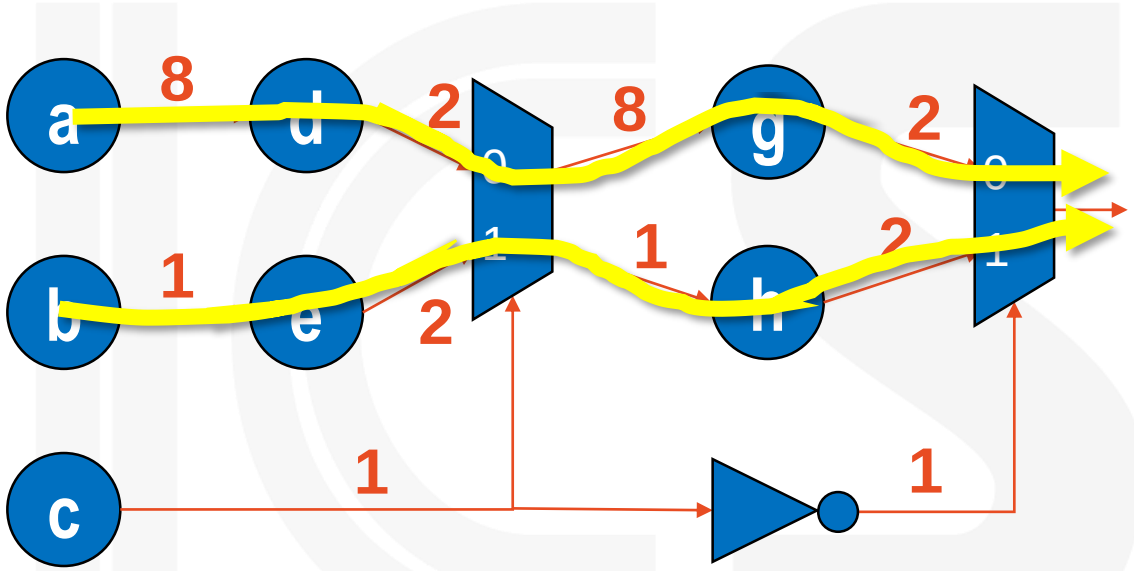

Input and Output Delays




Input drive and output cap modeling

# Timing Exceptions

- There are several cases when we need to define exceptions that should be treated differently by STA.
- For example, looking into the topology of the network we saw earlier:



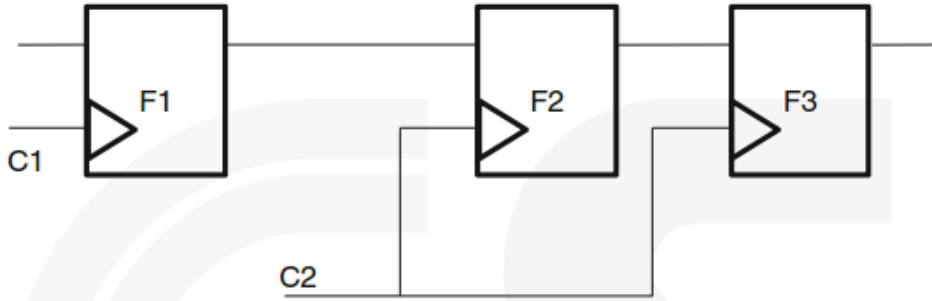

- In this case, we would define a *false path*:

```
set_false_path -through [get_pins mux1/I0] -through [get_pins mux2/I0]
set_false_path -through [get_pins mux1/I1] -through [get_pins mux2/I1]
```

# Timing Exceptions (2)

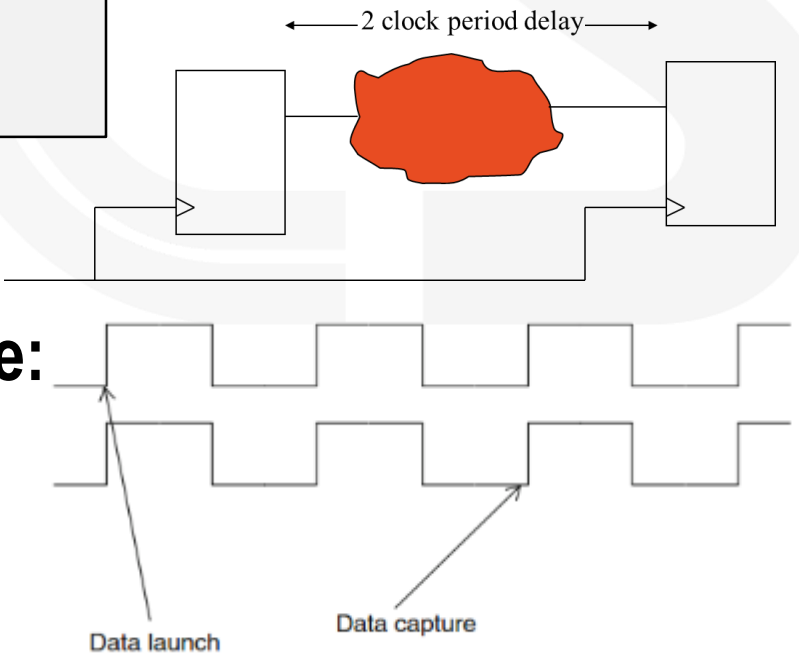
- Another common case of a false path is a clock domain crossing through a synchronizer:

```
set_false_path –from F1/CP –to F2/D
```




- Alternatively, this can be defined with:

```
set_clock_groups –logically_exclusive \
       –group [get_clocks C1] –group [get_clocks C2]
```




- If an equal-phase (divided) slow clock is sending data to a faster clock, a multi-cycle path may be appropriate:

```
set_multicycle_path –setup –from F1/CP –to F2/D   2
set_multicycle_path –hold  –from F1/CP –to F2/D   1
```

# Case Analysis

- **A common case for designs is that some value should be assumed constant**
  - For example, setting a register for a certain operating mode.
- **In such cases, many timing paths are false**
  - For example, if the constant sets a multiplexer selector.
  - Or a ‘0’ is driven to one of the inputs of an AND gate.
- **To propagate these constants through the design and disable irrelevant timing arcs, a `set_case_analysis` constraint is used:**

```
set_case_analysis 0 [get_ports TEST_MODE]
```

# Design Rule Violations (DRV)

- **You can set specific design rules that should be met, for example:**
  - Maximum transition through a net.

```
set_max_transition $MAX_TRAN_IN_NS
```

  - Maximum Capacitive load of a net.

```
set_max_capacitance $MAX_CAP_IN_PF
```

  - Maximum fanout of a gate.

```
set_max_fanout $MAX_FANOUT
```

# Yield-driven and Advanced STA

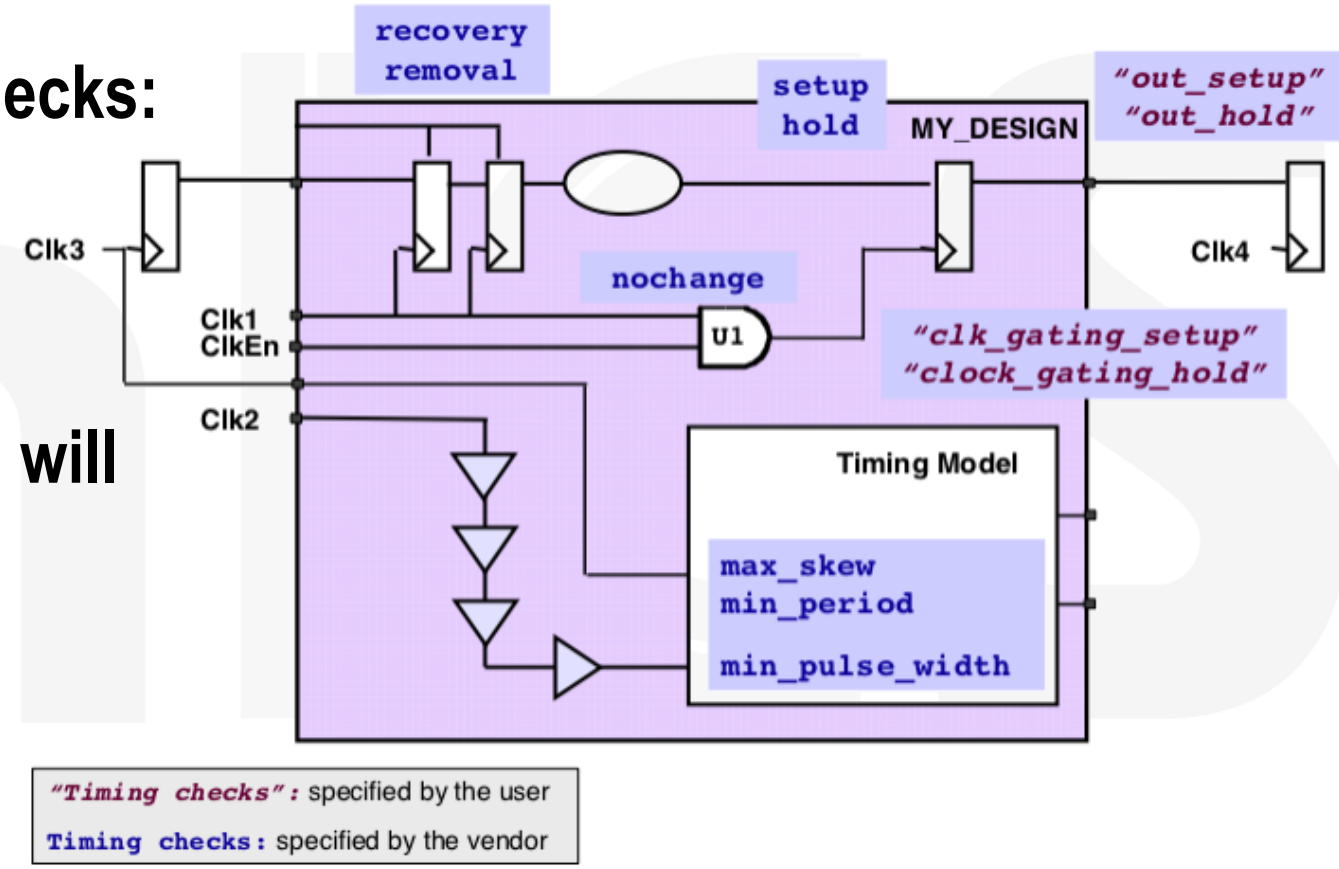
- **There are many more concepts, approaches, and terminologies used in timing analysis for high-yield signoff:**
  - On-chip Variation (OCV)
  - Advanced On-Chip Variation (AOCV)
  - Signal Integrity (SI)
  - and more and more…*
- **We will end with the basics now and get back to this towards the end of the course.**

** Between the time I wrote this slide and presented it to you, each EDA vendor has presented another method for timing closure that you just must know about and have to use ☺.*

# Timing Reports

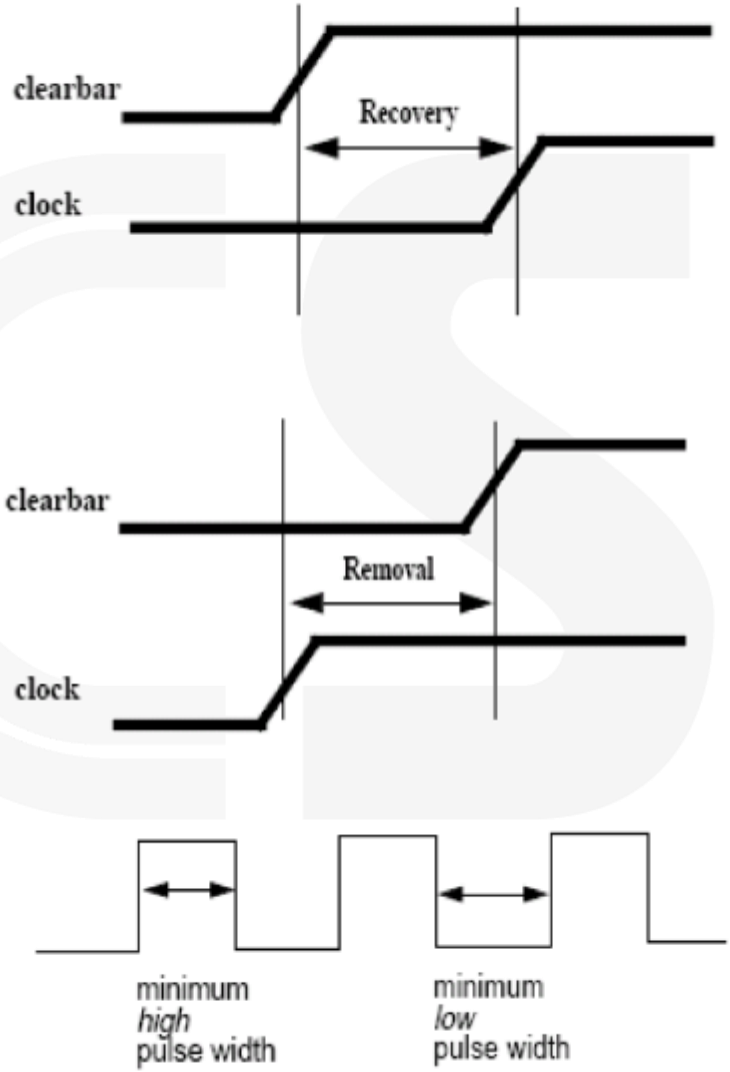
# Check Types

- **Throughout this lecture, we have discussed the two primary timing checks:**
  - Setup (max) Delay
  - Hold (min) Delay
- **However, in practice, there are other categories of timing checks that you will encounter:**
  - Recovery
  - Removal
  - Clock Gating
  - Min Pulse Width
  - Data-to-Data



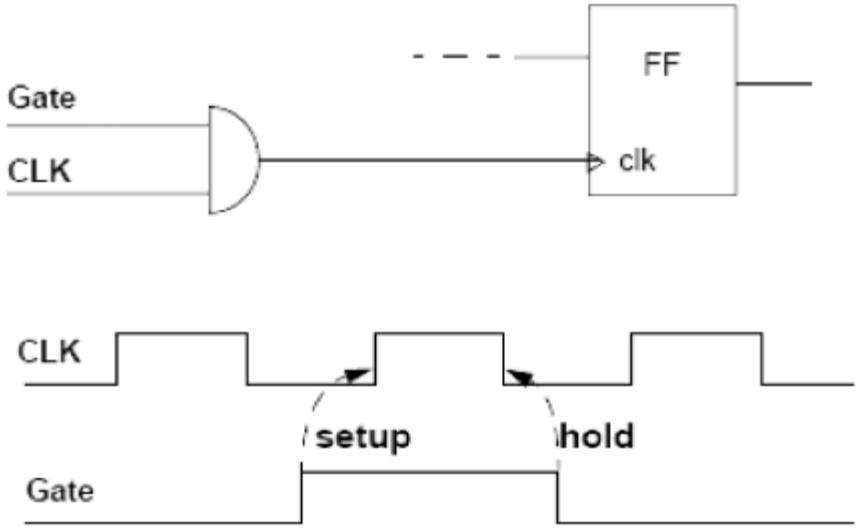
# Recovery, Removal and MPW

- **Recovery Check**
  - The minimum time that an asynchronous control input pin must be stable *after* being *deasserted* and before the *next* clock transition (active-edge)
- **Removal Check**
  - The minimum time that an asynchronous control input pin must be stable *before* being *deasserted* and *after* the previous clock transition (active edge)
- **Minimum Clock Pulse Width (MPW)**
  - The amount of time *after* the rising/falling edge of a clock that the clock signal must remain stable.

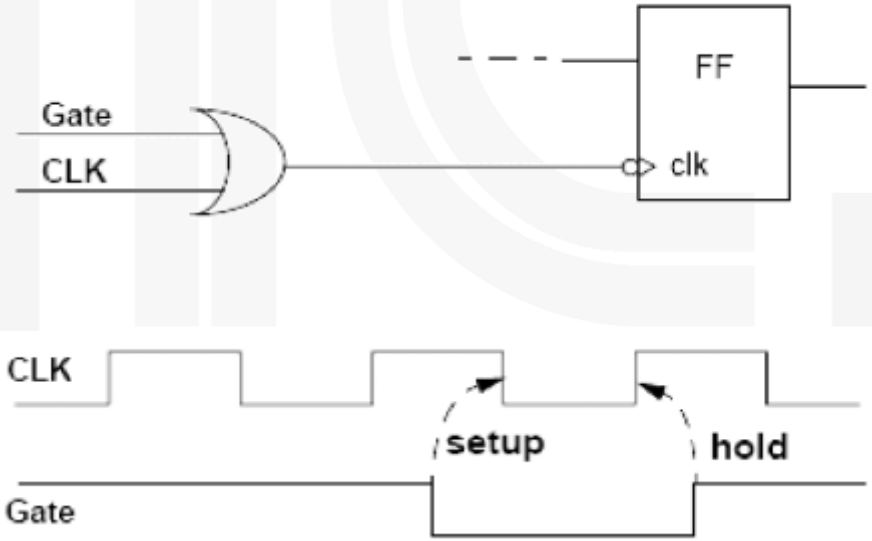


# Clock Gating Check

- **Clock gating occurrences are any signals on the clock path that block (gate) the clock from propagating.**
- **The enable path of the clock gate must arrive enough time before the clock itself to ensure glitch-free functionality (and similarly hold after the edge).**




Ex. 1: Gating signal should only change when the clock is in the low state




Ex. 2: Gating signal should only change when the clock is in high low state

# Checking your design

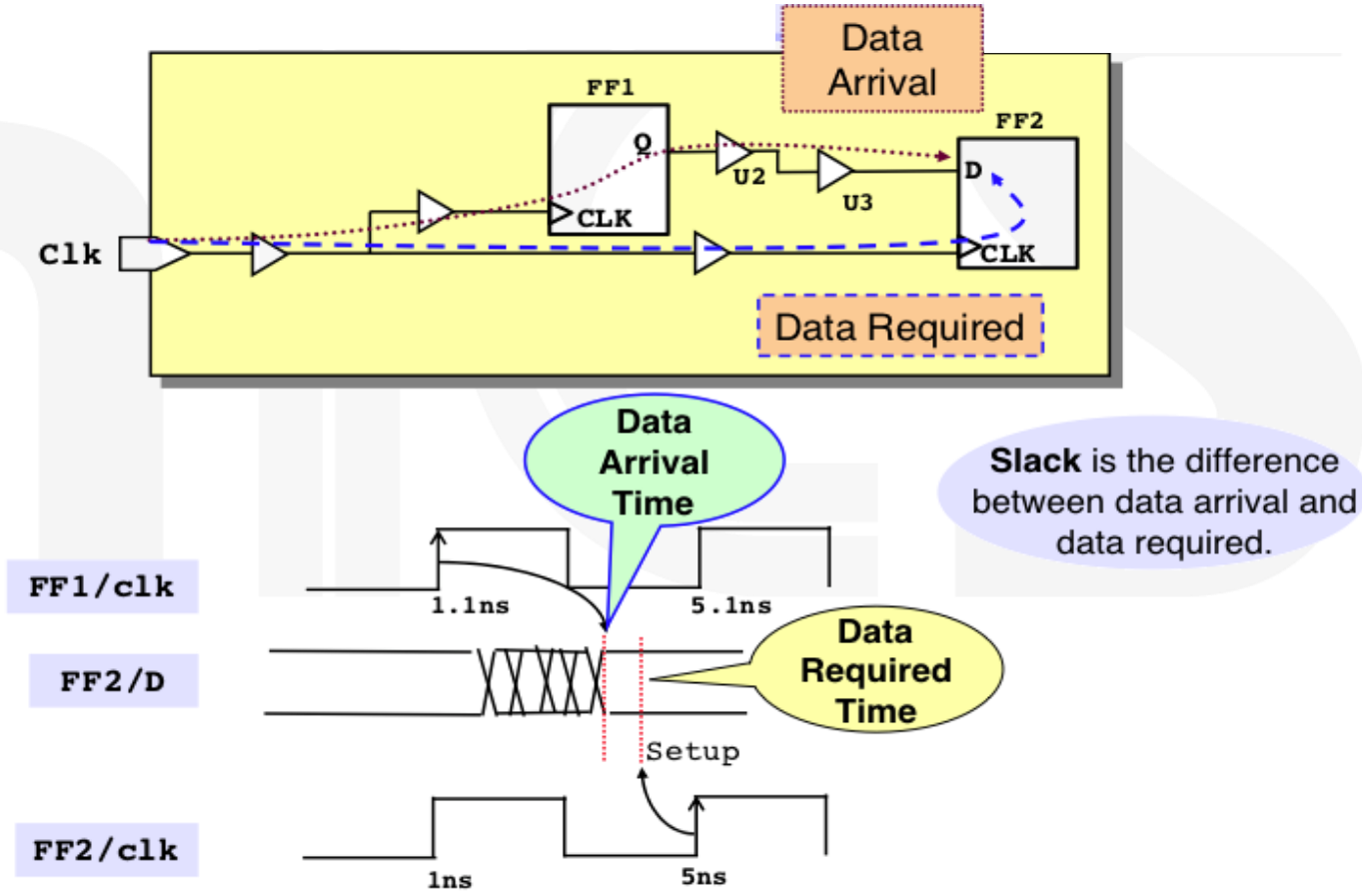
- `report_analysis_coverage` **checks that you have fully constrained your design.**

```
#  Command:          report_analysis_coverage
################################################################
     +-------------------------------------------------------------------+
     |                  TIMING CHECK COVERAGE SUMMARY                    |
     |-------------------------------------------------------------------|
     |      Check Type      | No. of |     Met      | Violated |  Untested |
     |                      | Checks |              |          |           |
     |----------------------+--------+--------------+----------+-----------|
     |  Clock Gating Setup  |  4544  | 4544 (100%)  |  0 (0%)  |  0 (0%)   |
     | ExternalDelay (Late) |  505   |  439 (86%)   |  0 (0%)  |  66 (13%) |
     |      PulseWidth      | 96067  | 87786 (91%)  |  0 (0%)  | 8281 (8%) |
     |       Recovery       |  8281  | 8281 (100%)  |  0 (0%)  |  0 (0%)   |
     |        Setup         | 41125  | 41113 (99%)  |  0 (0%)  |  12 (0%)  |
     |      TimeBorrow      |  5536  | 5536 (100%)  |  0 (0%)  |  0 (0%)   |
     +-------------------------------------------------------------------+
```

- `check_timing` **Performs a variety of consistency and completeness checks on the timing constraints specified for a design.**

# Report Timing

- **Perhaps the most important command in any synthesis or place and route tool is `report_timing`.**
  - For convenience, we will look at the Stylus Common UI syntax and reports.
  - For other tools, the concepts are similar.



# Report Timing

```
report_timing
```

Header

Launch path

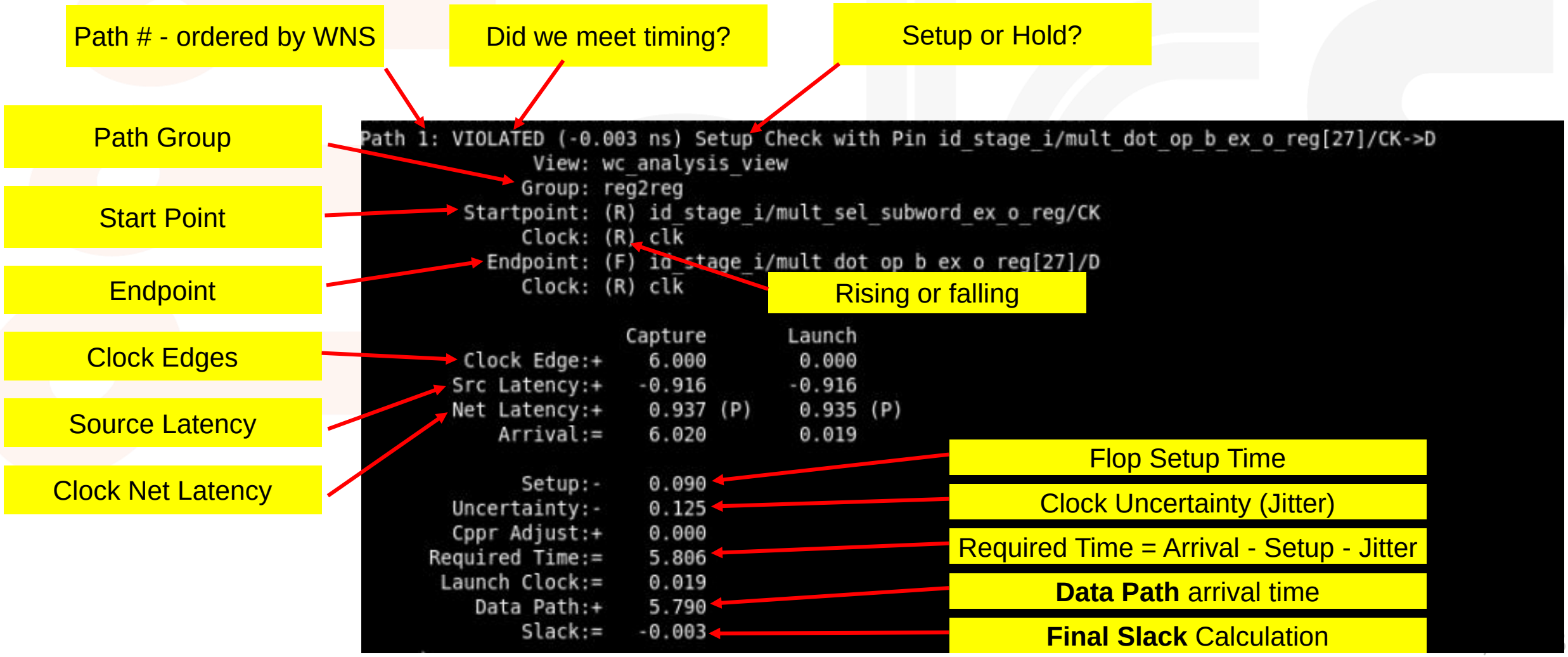
```
Path 1: VIOLATED (-0.003 ns) Setup Check with Pin id_stage_i/mult_dot_op_b_ex_o_reg[27]/CK->D
              View: wc_analysis_view
             Group: reg2reg
       Startpoint: (R) id_stage_i/mult_sel_subword_ex_o_reg/CK
             Clock: (R) clk
         Endpoint: (F) id_stage_i/mult_dot_op_b_ex_o_reg[27]/D
             Clock: (R) clk

                      Capture       Launch
        Clock Edge:+    6.000        0.000
       Src Latency:+   -0.916       -0.916
       Net Latency:+    0.937 (P)    0.935 (P)
           Arrival:=    6.020        0.019

             Setup:-    0.090
        Uncertainty:-   0.125
       Cppr Adjust:+    0.000
     Required Time:=    5.806
      Launch Clock:=    0.019
         Data Path:+    5.790
             Slack:=   -0.003
```

| Timing Point | Flags | Arc | Edge | Fanout | Trans (ns) | Delay (ns) | Arrival (ns) |
|---|---|---|---|---|---|---|---|
| id_stage_i/mult_sel_subword_ex_o_reg/CK | | CK | R | 13 | 0.100 | | 0.019 |
| id_stage_i/mult_sel_subword_ex_o_reg/Q | | CK->Q | F | 1 | 0.100 | 0.396 | 0.415 |
| ex_stage_i_mult_i/FE_RC_1472_0/Y | | B->Y | R | 2 | 0.044 | 0.119 | 0.534 |
| ex_stage_i_mult_i/FE_RC_418_0/Y | | A1->Y | F | 1 | 0.143 | 0.135 | 0.669 |
| ex_stage_i_mult_i/FE_OFC298_FE_RN_210_0/Y | | A->Y | F | 21 | 0.101 | 0.171 | 0.840 |

⋮

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| ex_stage_i_mult_i/sra_107_120_g4848/Y | | A0->Y | F | 1 | 0.139 | 0.106 | 4.950 |
| ex_stage_i_mult_i/FE_RC_888_0/Y | | A->Y | R | 1 | 0.098 | 0.135 | 5.085 |
| ex_stage_i_mult_i/FE_RC_887_0/Y | | A->Y | F | 1 | 0.161 | 0.115 | 5.199 |
| g2031/Y | | A0->Y | R | 1 | 0.105 | 0.123 | 5.323 |
| g1998/Y | | A->Y | F | 6 | 0.104 | 0.111 | 5.433 |
| id_stage_i/FE_OCPC3458_regfile_alu_wdata_fw_27/Y | | A->Y | F | 1 | 0.115 | 0.117 | 5.551 |
| id_stage_i/g34311/Y | | A->Y | R | 1 | 0.034 | 0.082 | 5.633 |
| id_stage_i/FE_RC_546_0/Y | | C->Y | F | 3 | 0.119 | 0.176 | 5.809 |
| id_stage_i/mult_dot_op_b_ex_o_reg[27]/D | | D | F | 3 | 0.187 | 0.002 | 5.809 |

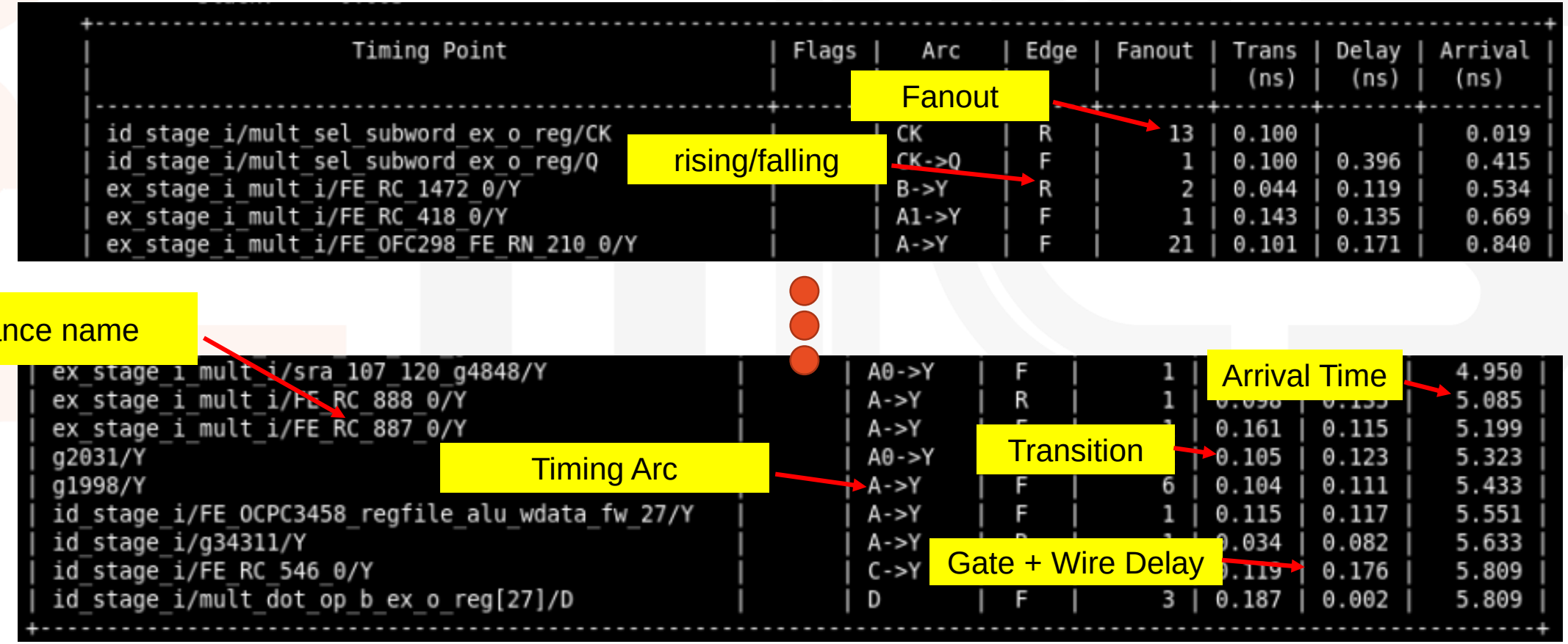
# Report Timing - Header




```
Path 1: VIOLATED (-0.003 ns) Setup Check with Pin id_stage_i/mult_dot_op_b_ex_o_reg[27]/CK->D
           View: wc_analysis_view
          Group: reg2reg
     Startpoint: (R) id_stage_i/mult_sel_subword_ex_o_reg/CK
          Clock: (R) clk
       Endpoint: (F) id_stage_i/mult_dot_op_b_ex_o_reg[27]/D
          Clock: (R) clk

                    Capture       Launch
     Clock Edge:+     6.000        0.000
     Src Latency:+   -0.916       -0.916
     Net Latency:+    0.937 (P)    0.935 (P)
         Arrival:=    6.020        0.019

           Setup:-    0.090
     Uncertainty:-    0.125
     Cppr Adjust:+    0.000
   Required Time:=    5.806
    Launch Clock:=    0.019
       Data Path:+    5.790
           Slack:=   -0.003
```

# Report Timing – Launch Path

- **Standard timing report only shows the data delay of the launch path and very basic information.**



# Report Timing – Full Clock

- **To get more data about the clock propagation, use the full_clock option:**

```
report_timing -path_type full_clock
```

- **Pay attention – timing calculation has changed!**

```
                   Capture      Launch
     Clock Edge:+    6.000       0.000
      Drv Adjust:+   0.079       0.079
     Src Latency:+  -0.916      -0.916
     Net Latency:+   0.857 (P)   0.856 (P)
         Arrival:=   6.020       0.019

           Setup:-   0.090
     Uncertainty:-   0.125
      Cppr Adjust:+  0.000
    Required Time:=  5.806
     Launch Clock:=  0.019
        Data Path:+  5.790
            Slack:= -0.003
Timing Path:
+-------------------------------------------------------------------------------------------------------------+
|                  Timing Point                  | Flags |   Arc    | Edge | Fanout | Trans | Delay | Arrival |
|                                                |       |          |      |        | (ns)  | (ns)  |  (ns)   |
+-------------------------------------------------------------------------------------------------------------+
| clk_i                                          |       | clk_i    | R    |      3 | 0.163 | 0.018 |  -0.819 |
| g1033/Y                                        |       | B->Y     | R    |      3 | 0.166 | 0.173 |  -0.646 |
| CTS_ccl_a_BUF_clk_G1_L2_3/Y                    |       | A->Y     | R    |      4 | 0.071 | 0.160 |  -0.486 |
| id_stage_i/CTS_ccl_a_BUF_clk_G1_L3_10/Y        |       | A->Y     | R    |      5 | 0.096 | 0.163 |  -0.324 |
| id_stage_i/RC_CG_HIER_INST9/RC_CGIC_INST/ECK   |       | CK->ECK  | R    |      4 | 0.083 | 0.167 |  -0.157 |
| id_stage_i/CTS_ccl_a_BUF_clk_G2_L5_102/Y       |       | A->Y     | R    |     13 | 0.093 | 0.175 |   0.019 |
| id_stage_i/mult_sel_subword_ex_o_reg/Q         |       | CK->Q    | F    |      1 | 0.100 | 0.396 |   0.415 |
| ex_stage_i_mult_i/FE_RC_1472_0/Y               |       | B->Y     | R    |        |       |       |   0.534 |
| ex_stage_i_mult_i/FE_RC_418_0/Y                |       | A1->Y    | F    |        |       |       |   0.669 |
| ex_stage_i_mult_i/FE_OFC298_FE_RN_210_0/Y      |       | A->Y     | F    |     21 | 0.101 | 0.171 |   0.840 |
```

Source insertion delay is calculated to average out I/O clocking

Clock Port

Launch Clock

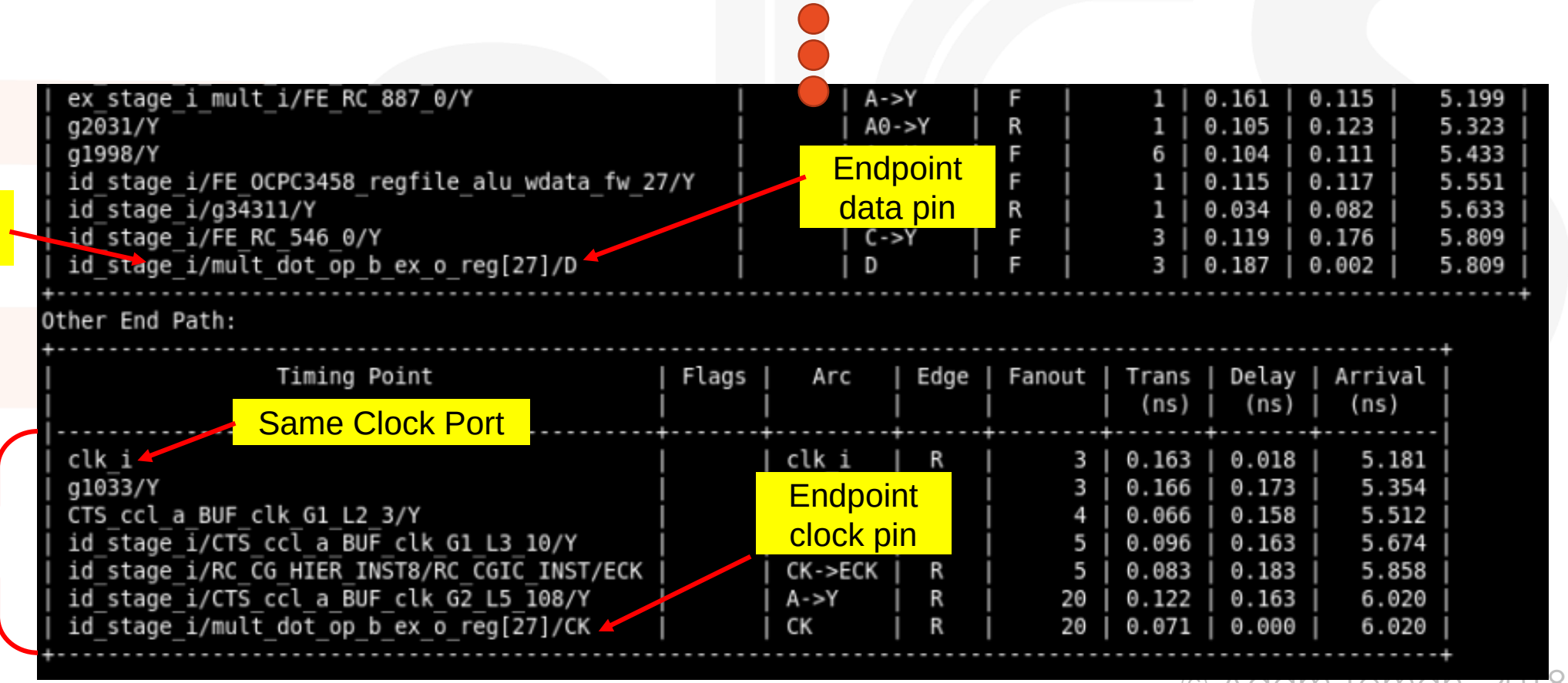
Data Start Point

Timing report continues…

# Report Timing – Full Clock (2)

- **We also get to see the Capture Clock.**

Continued from last slide



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| ex_stage_i_mult_i/FE_RC_887_0/Y | | A->Y | F | 1 | 0.161 | 0.115 | 5.199 |
| g2031/Y | | A0->Y | R | 1 | 0.105 | 0.123 | 5.323 |
| g1998/Y | | | F | 6 | 0.104 | 0.111 | 5.433 |
| id_stage_i/FE_OCPC3458_regfile_alu_wdata_fw_27/Y | | | F | 1 | 0.115 | 0.117 | 5.551 |
| id_stage_i/g34311/Y | | | R | 1 | 0.034 | 0.082 | 5.633 |
| id_stage_i/FE_RC_546_0/Y | | C->Y | F | 3 | 0.119 | 0.176 | 5.809 |
| id_stage_i/mult_dot_op_b_ex_o_reg[27]/D | | D | F | 3 | 0.187 | 0.002 | 5.809 |

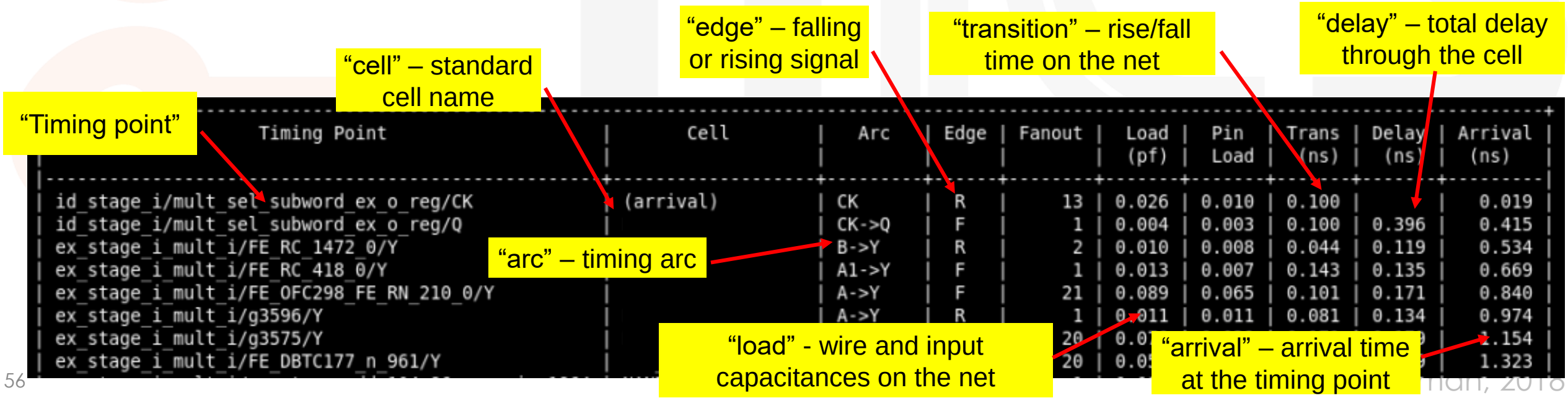
Other End Path:

| Timing Point | Flags | Arc | Edge | Fanout | Trans (ns) | Delay (ns) | Arrival (ns) |
|---|---|---|---|---|---|---|---|
| clk_i | | clk_i | R | 3 | 0.163 | 0.018 | 5.181 |
| g1033/Y | | | | 3 | 0.166 | 0.173 | 5.354 |
| CTS_ccl_a_BUF_clk_G1_L2_3/Y | | | | 4 | 0.066 | 0.158 | 5.512 |
| id_stage_i/CTS_ccl_a_BUF_clk_G1_L3_10/Y | | | | 5 | 0.096 | 0.163 | 5.674 |
| id_stage_i/RC_CG_HIER_INST8/RC_CGIC_INST/ECK | | CK->ECK | R | 5 | 0.083 | 0.183 | 5.858 |
| id_stage_i/CTS_ccl_a_BUF_clk_G2_L5_108/Y | | A->Y | R | 20 | 0.122 | 0.163 | 6.020 |
| id_stage_i/mult_dot_op_b_ex_o_reg[27]/CK | | CK | R | 20 | 0.071 | 0.000 | 6.020 |

# Report Timing – `fields` option

- To debug timing, we would like more information, for example, the net name, the wire capacitance, the pin capacitance, etc.
- Use the `-fields` option to get the info you *really* need.
- For example:

```
report_timing -fields "timing_point cell arc edge fanout load pin_load transition delay arrival"
```



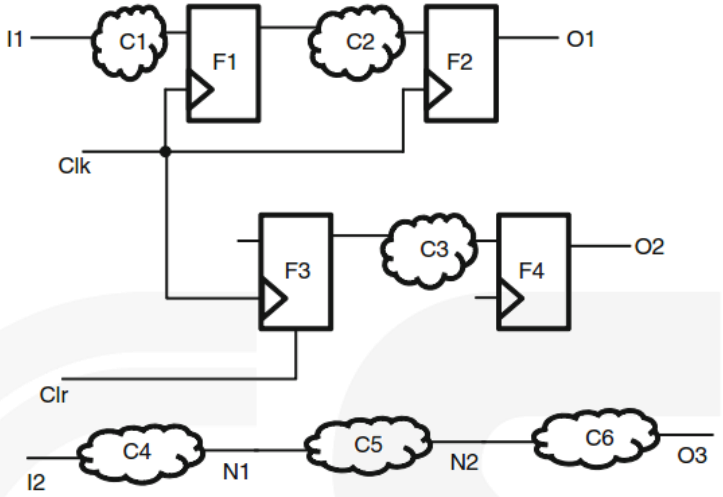

| Timing Point | Cell | Arc | Edge | Fanout | Load (pf) | Pin Load | Trans (ns) | Delay (ns) | Arrival (ns) |
|---|---|---|---|---|---|---|---|---|---|
| id_stage_i/mult_sel_subword_ex_o_reg/CK | (arrival) | CK | R | 13 | 0.026 | 0.010 | 0.100 | | 0.019 |
| id_stage_i/mult_sel_subword_ex_o_reg/Q | | CK->Q | F | 1 | 0.004 | 0.003 | 0.100 | 0.396 | 0.415 |
| ex_stage_i_mult_i/FE_RC_1472_0/Y | | B->Y | R | 2 | 0.010 | 0.008 | 0.044 | 0.119 | 0.534 |
| ex_stage_i_mult_i/FE_RC_418_0/Y | | A1->Y | F | 1 | 0.013 | 0.007 | 0.143 | 0.135 | 0.669 |
| ex_stage_i_mult_i/FE_OFC298_FE_RN_210_0/Y | | A->Y | F | 21 | 0.089 | 0.065 | 0.101 | 0.171 | 0.840 |
| ex_stage_i_mult_i/g3596/Y | | A->Y | R | 1 | 0.011 | 0.011 | 0.081 | 0.134 | 0.974 |
| ex_stage_i_mult_i/g3575/Y | | | | 20 | 0.0 | | | | 1.154 |
| ex_stage_i_mult_i/FE_DBTC177_n_961/Y | | | | 20 | 0.0 | | | | 1.323 |

# Report Timing – Selecting Paths

- **By default, `report_timing` shows you the most critical path**
  - i.e., the path with the worst negative slack (WNS)
- **But sometimes, we want to analyze a specific path or set of paths.**
  - For example, I only want to see the paths that come from a primary input…
- **Use the `-from`, `-to`, `-through` flags and their variants:**
  - `-from`: To select a start point (= input port or register/IP clock pin)
  - `-to`: To select an endpoint (= output port or register/IP data pin)
  - `-through`: to select any other pin
- **You can specify direction (i.e., `-through_rise`), clock (i.e., `-clock_from`)**

```
report_timing -from ff1/CK -through_fall mux1/I0 -to [all_outputs]
```

# Report Timing - Path Groups




- **Path groups are categories of paths that are both optimized and reported separately.**
  - Default path groups, as we saw before, are `reg2reg`, `in2reg`, `reg2out`, `in2out`.
  - In addition, paths ending at clock gates (`reg2cgate`) are treated separately.
  - To automatically create these groups, use the `create_basic_path_groups` command in Innovus.

```
create_basic_path_goups -expanded
```

- **To create specific path groups for your design, use the `group_path` command:**

```
group_path –from ff1/CLK –to ff2/D –name my_path
```

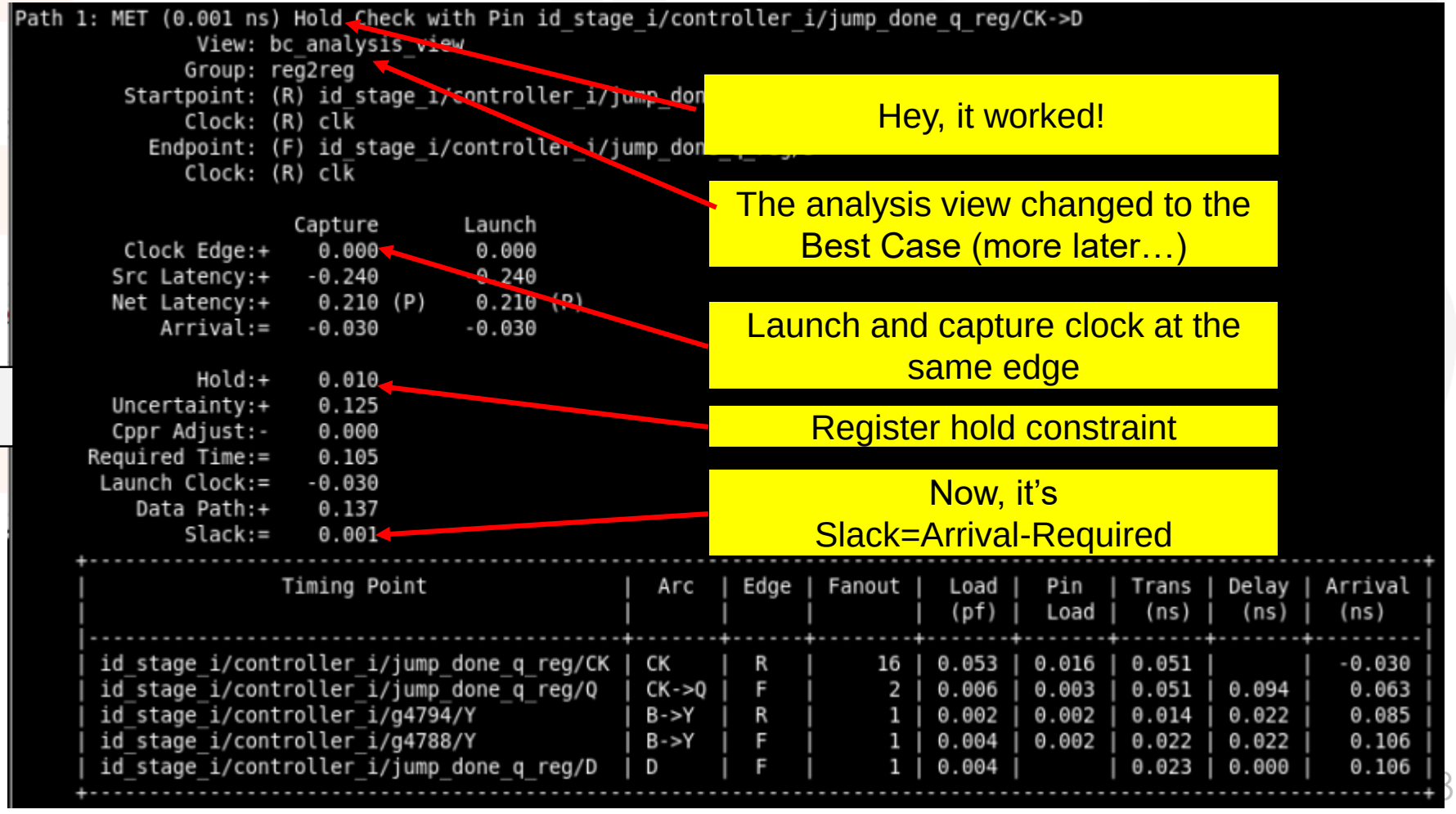
- **To report timing for a certain path group:**

```
report_timing –path_group my_path
```

# Report Timing - Hold

- **By default, the `report_timing` command reports setup (max delay) timing.**
  - After clock tree synthesis, you will want to make sure your design meets hold (min_delay), as well.
- **To report hold timing, just add the `-early` option**

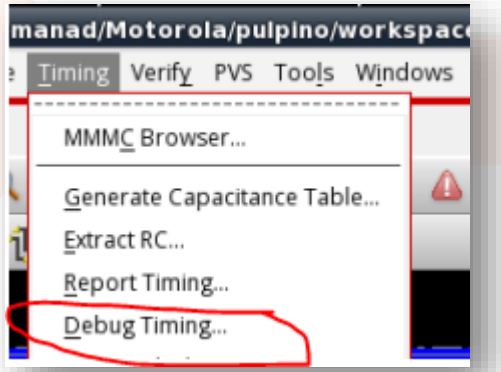
```
report_timing -early
```

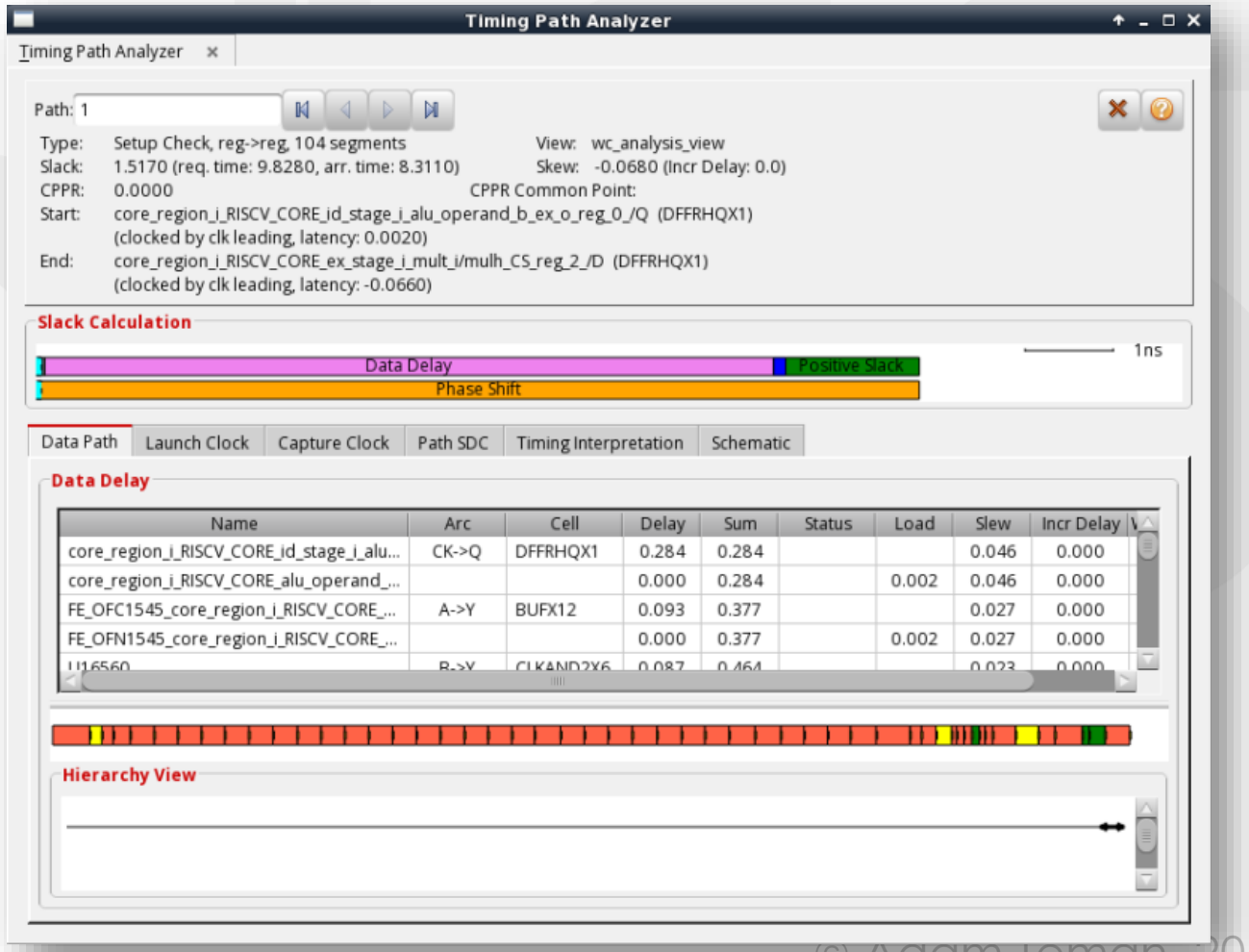



| Timing Point | Arc | Edge | Fanout | Load (pf) | Pin Load | Trans (ns) | Delay (ns) | Arrival (ns) |
|---|---|---|---|---|---|---|---|---|
| id_stage_i/controller_i/jump_done_q_reg/CK | CK | R | 16 | 0.053 | 0.016 | 0.051 | | -0.030 |
| id_stage_i/controller_i/jump_done_q_reg/Q | CK->Q | F | 2 | 0.006 | 0.003 | 0.051 | 0.094 | 0.063 |
| id_stage_i/controller_i/g4794/Y | B->Y | R | 1 | 0.002 | 0.002 | 0.014 | 0.022 | 0.085 |
| id_stage_i/controller_i/g4788/Y | B->Y | F | 1 | 0.004 | 0.002 | 0.022 | 0.022 | 0.106 |
| id_stage_i/controller_i/jump_done_q_reg/D | D | F | 1 | 0.004 | | 0.023 | 0.000 | 0.106 |

# Report Timing Debugger

- **A very good GUI option is to use the Innovus “Debug Timing” tool.**
  - This tool lets you explore the timing report interactively, even showing path schematics, SDC, and highlighting the path in the layout.



# Multi-Mode Multi-Corner

Or how to deal with the corner crisis!

# More than one operating mode

- **During synthesis, we (usually) target timing for a worst-case scenario.**
  - But, what is "worst-case"?
  - Intuitively, that would be a **slow corner**, (i.e., SS, VDD-10%, 125C)
  - No need for hold checking, since clock is ideal (No skew = No hold)
- **But, what if there is an additional operating mode?**
  - For example, a test (scan) mode.
  - Do we have to close timing at the same (high) clock speed?

| Mode | TEST_MODE | FREQ |
|---|---|---|
| Functional | 0 | 1 GHz |
| Test | 1 | 10 MHz |

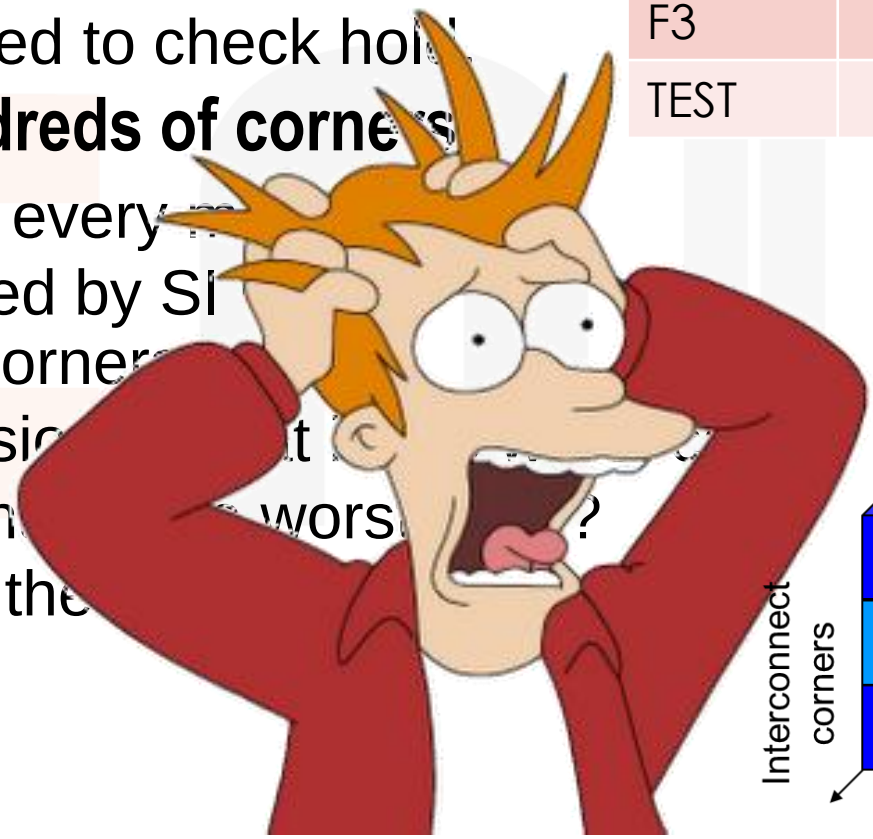
- **No problem, we'll just deal with both modes separately**
  - Prepare an additional SDC and rerun STA/optimization.

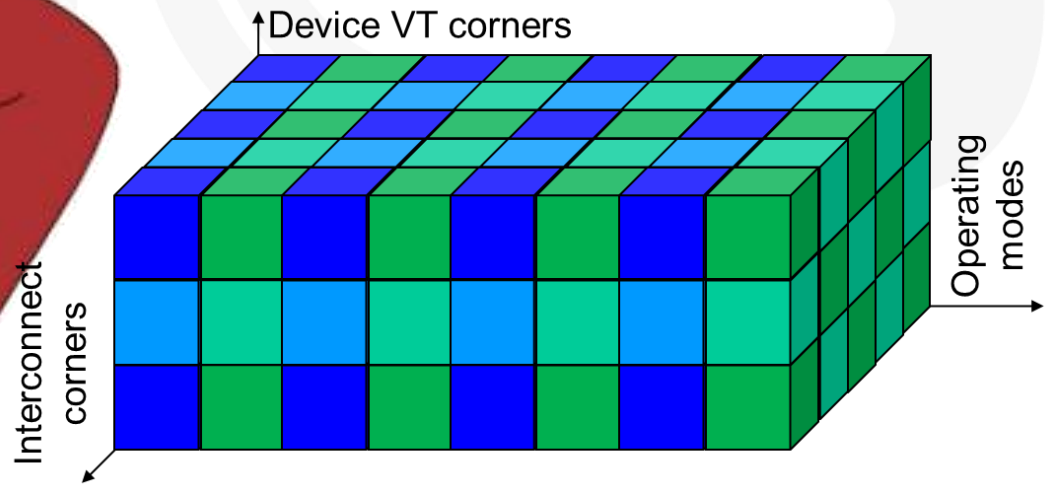
```
set_case_analysis 1 [get_ports TEST_MODE]
create_clock -period [expr $TCLK/100] -name TEST_CLK [get_ports TEST_CLK]
```

# Many, many, corners…

- **But real SoCs are much more complex:**
  - Many operating modes.
  - Many voltage domains.
  - With real clock, need to check hol
- **We easily get to hundreds of corne**
  - Setup and hold for every
  - Hold can be affected by Sl
    check hold for all corner
  - Temperature inversio
  - RC Extraction – wh wors ?
  - Leakage – what is the
- **Aaaaarrrrrgggghhhh!**

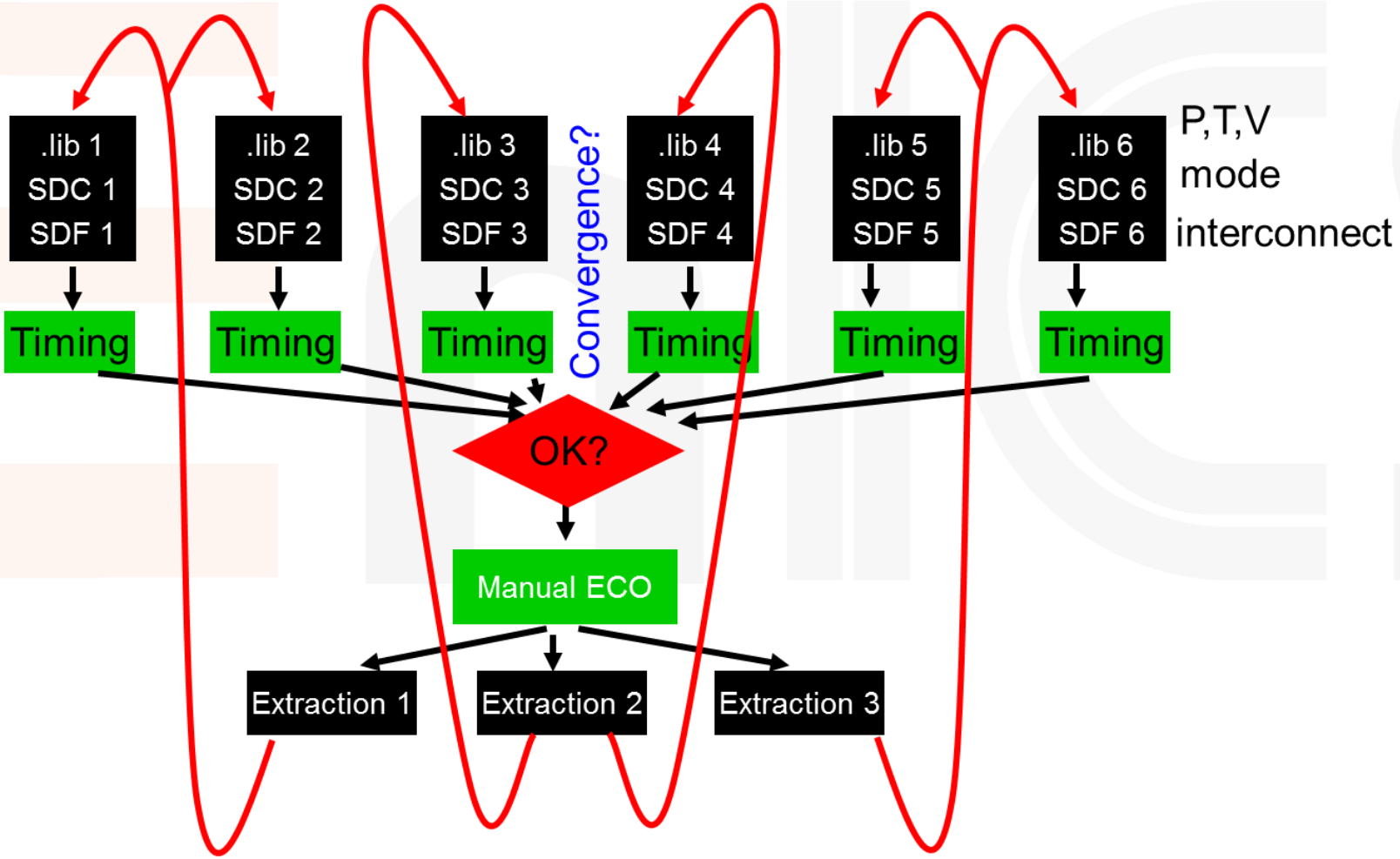
| Mode | VDD1 | FREQ1 | VDD2 | FREQ2 |
|---|---|---|---|---|
| F1 | 1.2 V | 2 GHz | 0.8 V | 500 MHz |
| F2 | 0.8 V | 400 MHz | 0.8 V | 400 MHz |
| F3 | Off | Off | 0.5 V | 50 MHz |
| TEST | 1.2 V | 10 MHz | 1.2 V | 10 MHz |





# The corner crisis

- **Traditional approach not feasible**



# Multi-Mode, Multi-Corner

- **MMMC to the rescue!**
  - It's implemented in a slightly (!) confusing way, but it really simplifies things.
- **The basic concept is that we create *analysis views* that can then be selected for setup and hold (max and min) constraints.**

Analysis View = "Turbo"

VDD=1.2V
Freq=2 GHz
Corner=SS
Temp=125 C
OpMode="Turbo"

Analysis View = "Low Power"

VDD=0.5V
Freq=50 MHz
Corner=SS
Temp=125 C
OpMode="LP"

Analysis View = "Turbo - Hold"

VDD=1.3V
Freq=2 GHz
Corner=FF
Temp= -40 C
OpMode="Turbo"

Setup checks: "Turbo" and "Low Power" Modes
Hold checks: "Turbo – Hold" Mode

# Multi-Mode, Multi-Corner

- **Defining Analysis Views is done in hierarchical fashion.**
  - An *analysis view* is constructed from a *delay corner* and a *constraint mode*.

```
create_analysis_view -name turbo \
       -constraint_mode turbo_mode -delay_corner slow_corner_vdd12
create_analysis_view -name low_power \
       -constraint_mode low_power_mode -delay_corner slow_corner_vdd05
create_analysis_view -name turbo_hold \
       -constraint_mode turbo_mode -delay_corner fast_corner_vdd13

set_analysis_view -setup {turbo low_power} -hold {turbo_hold}
```

  - A *delay corner* tells the tool how the delays are supposed to be calculated. Therefore it contains timing libraries and extraction rules.
  - A *constraint mode* is basically the relevant SDC commands/conditions for the particular operating mode.

# Multi-Mode, Multi-Corner

- **So now, let's define the lower levels of the MMMC hierarchy.**
  - A *constraint mode* is simply a list of relevant SDC files. When you move between analysis views, the STA tool will automatically apply the relevant constraints to the design.

```
create_constraint_mode -name turbo_mode -sdc_files {turbo.sdc}
create_constraint_mode -name low_power_mode -sdc_files {low_power.sdc}
```

  - A *delay corner* is a bit more complex. It comprises a *timing condition*, an *RC corner* and a few other things that we won't discuss right now.

```
create_delay_corner -name slow_corner_vdd12 \
        -rc_corner {RCmax} -timing_condition {ss_1p2V_125C}
create_delay_corner -name slow_corner_vdd05 \
        -rc_corner {RCmax} -timing_condition {ss_0p5V_125C}
create_delay_corner -name fast_corner_vdd13 \
        -rc_corner {RCmin} -timing_condition {ff_1p3V_m40C}
```

# Multi-Mode, Multi-Corner

- **Confused yet? Well, we still have more to go.**
  - A *timing condition* is a collection of library sets to be used for a certain power domain. For this course, we will just automatically connect a *timing condition* to a *library set*.

```
create_timing_condition -name tc_ss_1p2V_125C \
       -library_sets ss_1p2V_125C
```

  - A *library set* is a collection of the `.lib` characterizations that should be used for timing the relevant gates.
    This includes the standard cells and other macros, such as RAMs and I/Os.
    There also may be special “SI” characterizations for noise.

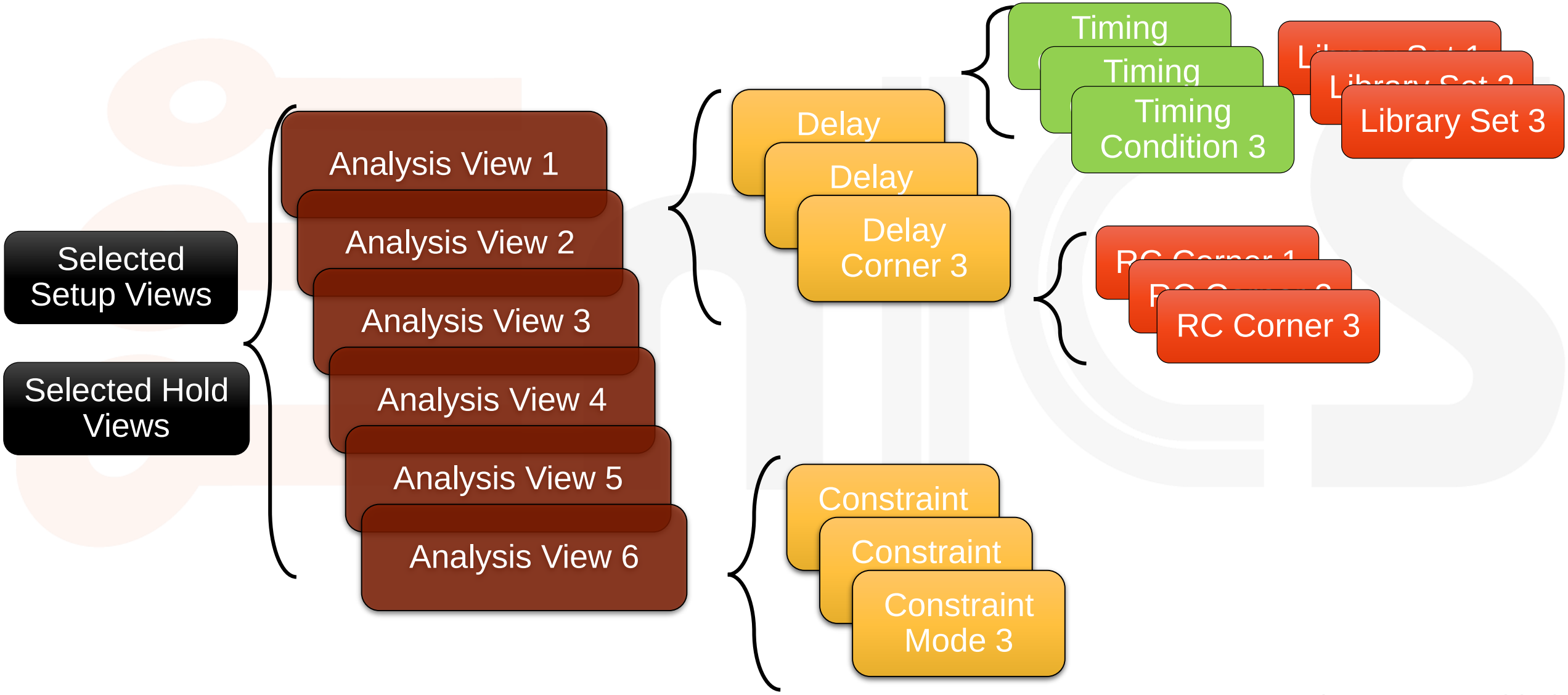
```
create_library_set -name ss_1p2V_125C \
       -timing [list ${sc_libs}/ss_1p2V_125.lib ${mem_libs}/ss_1p2V_125.lib \
        ${io_libs}/ss_1p8V_125.lib] –si ${sc_libs}/ss_1p2V_125.si
```

# Multi-Mode, Multi-Corner

- And finally, an *RC corner* is a collection of the rules for RC extraction. There may be a “capacitance table” for quick extraction and a `QRC techfile` for accurate extraction.
  The temperature is also defined in the RC corner, but it is taken into account in the `.lib` file, as well.

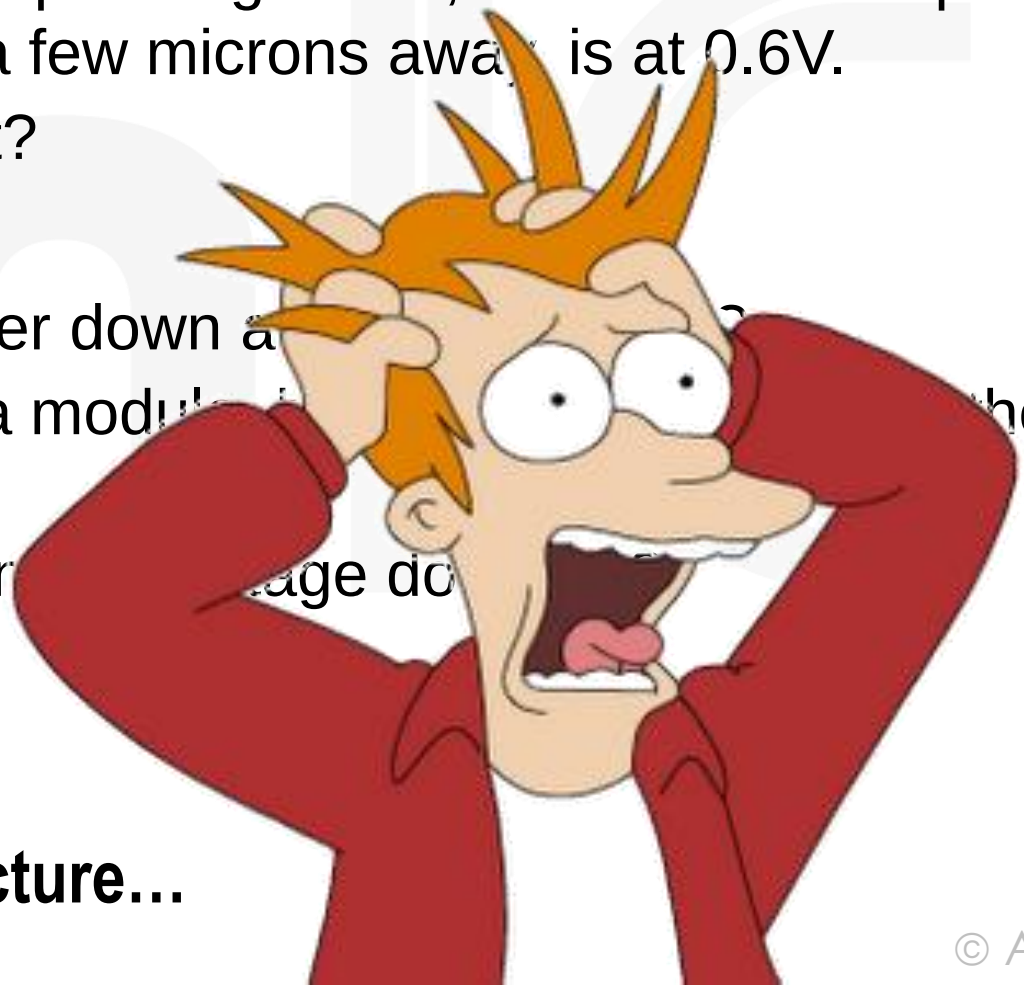
```
create_rc_corner -name RCmax -cap_table ${tech}/RCmax.CapTbl} -T {125} \
        -qx_tech_file ${tech}/RCmax.qrctech
```

# Multi-Mode, Multi-Corner - Summary



# …So you think that was complicated?

- **What if I have multiple voltage domains?**
  - Now, for example, in a certain operating mode, one inverter is operated at 1.2V, while another one, only a few microns away, is at 0.6V.
  - How do I define *that* library set?
- **Even worse…**
  - What happens if I want to power down a
  - What if I want to power down a modu he value stored in the flip flops)?
  - How do I transfer data betwee age do
- **Arrrrrgggghhhh!**



- **We'll briefly discuss this next lecture…**

# References


- Gil Rahav, BGU
- Gangadharan, Churiwala “Constraining Designs for Synthesis and Timing Analysis: A Practical Guide to Synopsys Design Constraints (SDC)”, Springer, 2013
- Synopsys SourceLink (+Synthesis Quick Reference)
- Cadence Support (+Genus and Innovus Text Command References)
- Rob Rutenbar “From Logic to Layout”, Coursera