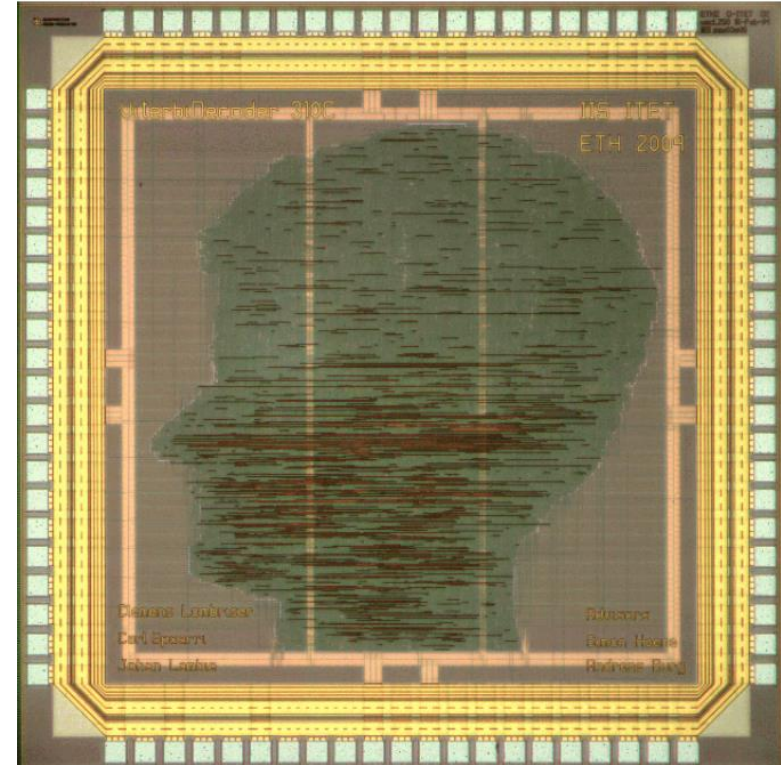


Digital Implementation

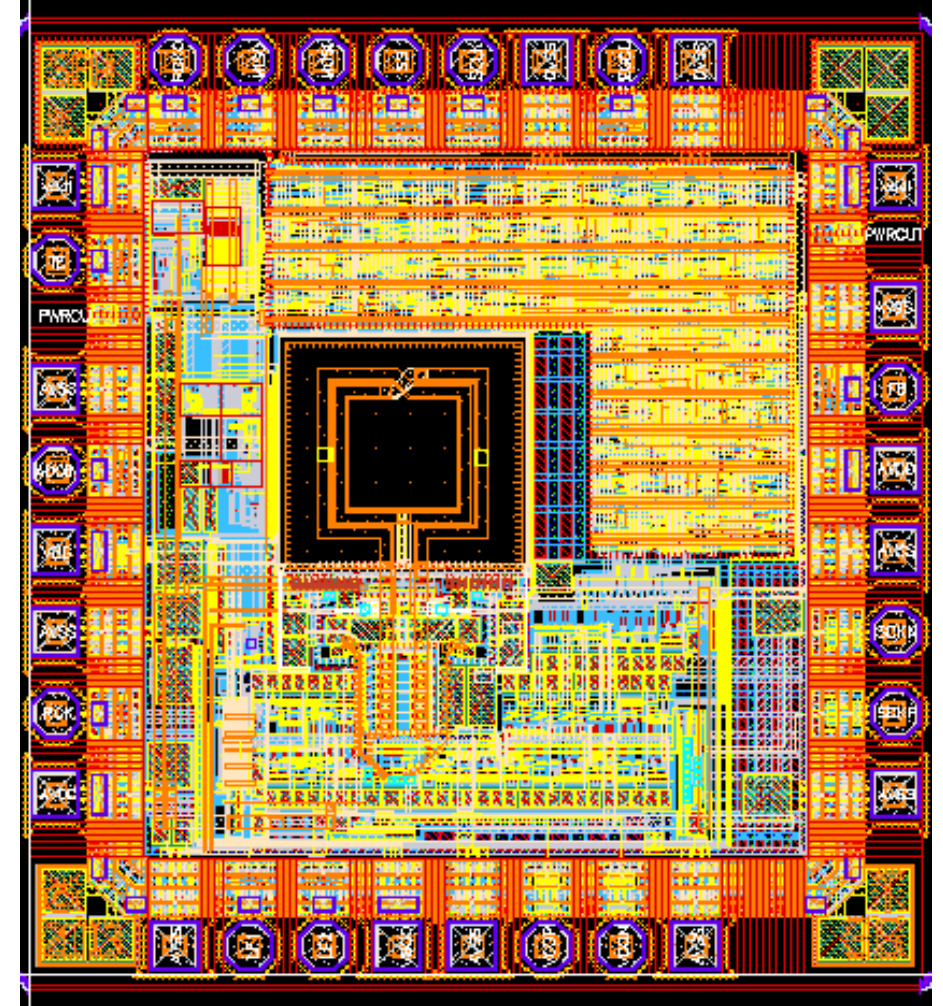
Clock Tree Synthesis






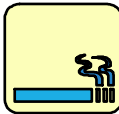
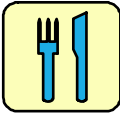
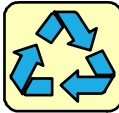




After completing this unit, you should be able to:

- ❑ List the status of the design prior to CTS
- ❑ Set up the design for clock tree synthesis
- ❑ Identify implicit clock tree start/end points and when explicit modifications are needed
- ❑ Control the constraints and targets used by CTS
- ❑ Describe the three different skew optimization methods
- ❑ Execute the recommended clock tree synthesis and optimization flow
- ❑ Analyze timing and clock specifications post-CTS

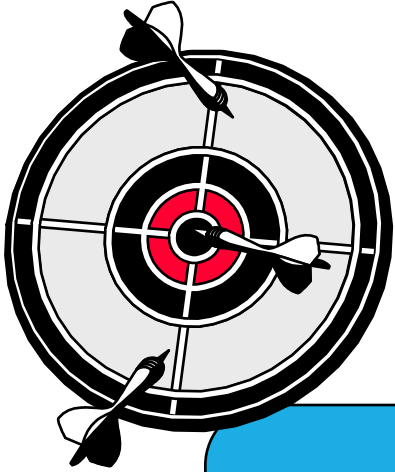


Building Hours		Phones	
Emergency		Messages	
Restrooms		Smoking	
Meals		Recycling	



Please turn off cell phones and pagers

Workshop Goal

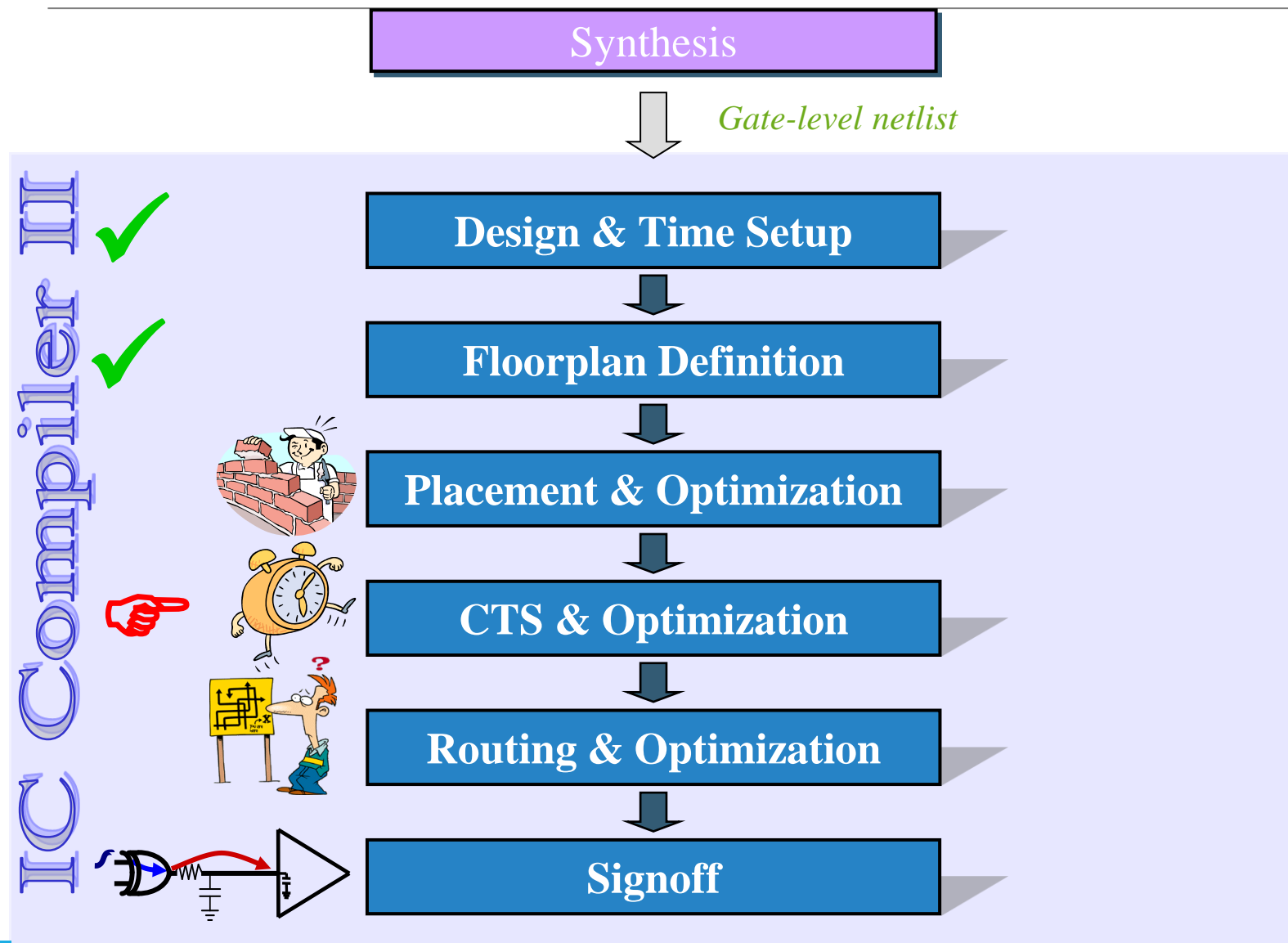


Use IC Compiler II to perform placement, DFT, CTS, routing and optimization, achieving timing closure for designs with moderate to high design challenges.

ASIC, back-end or layout designers with experience in standard cell-based automatic Place&Route.



High-Level IC Compiler Flow



Design Status, Start of CTS Phase

- ❑ Placement - completed
- ❑ Power and ground nets – prerouted
- ❑ Estimated congestion - acceptable
- ❑ Estimated timing - acceptable (~0ns slack)
- ❑ Estimated max cap/transition – no violations
- ❑ High fanout nets:
 - Reset, Scan Enable synthesized with buffers
 - Clocks are still not buffered

```
check_design -checks pre_clock_tree_stage
```



Why are there no buffers on clock nets?

Understand Your Clock Structure First

- ❑ **Questions you should be asking:**
 - **What are the different clock trees and their roots?**
 - **What are the stop (sink) and exclude pins?**
 - **Are there any preexisting clock cells or trees?**
 - **Are the generated clocks defined correctly?**
 - **Are there converging or overlapping clock trees?**
 - **What are the requirements between clocks?**

Understand Your Clock Tree Goals

❑ Skew Goal

- What are the skew requirements for your design?
- Are there different skew targets for small and large clocks?

❑ Insertion Delay Goal

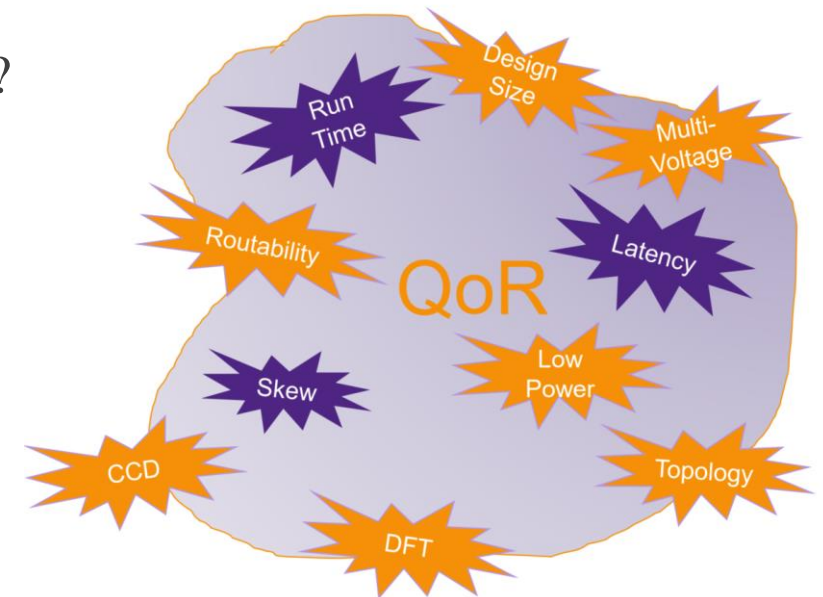
- What are the insertion delay specs for your block?
- What is a reasonable target based on the size and floorplan of your block/chip?

❑ Nondefault rules to prevent SI problems

❑ DRC Requirements

- Are signal net DRCs different from clock net DRCs?

❑ Find out the order of significance or importance of all the clocks in the design

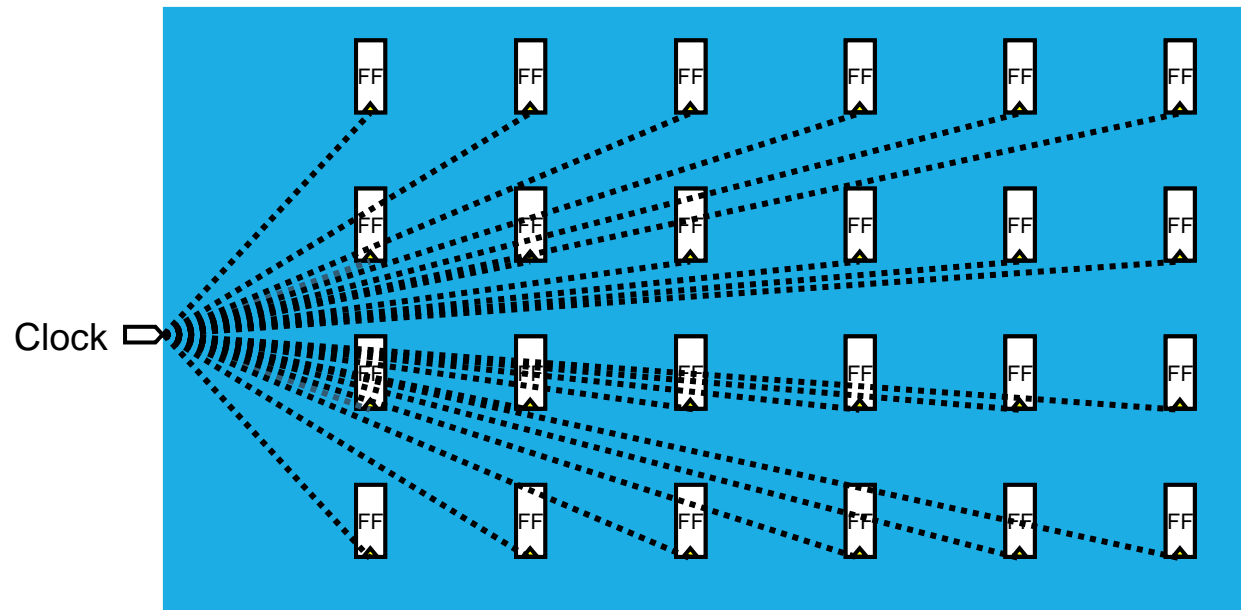


Understand Your Clock Tree Specs

- ❑ **Check SDC constraints and verify them against the specification**
 - Pay close attention to any generated clocks and muxed clocks and understand how
 - `set_case_analysis` switches between modes of operation
 - `create__clock` (clock source)
 - `create_generated_clock` (derived clock)
 - `set_clock_latency` (insertion delay)
 - `set_clock_uncertainty` (skew + jitter + margin)
 - `set_clock_transition` (transition for sync)
 - `set_case_analysis` (case analysis)
- ❑ **Find out what clocks to set as false paths which prevent unrelated logics from optimization**

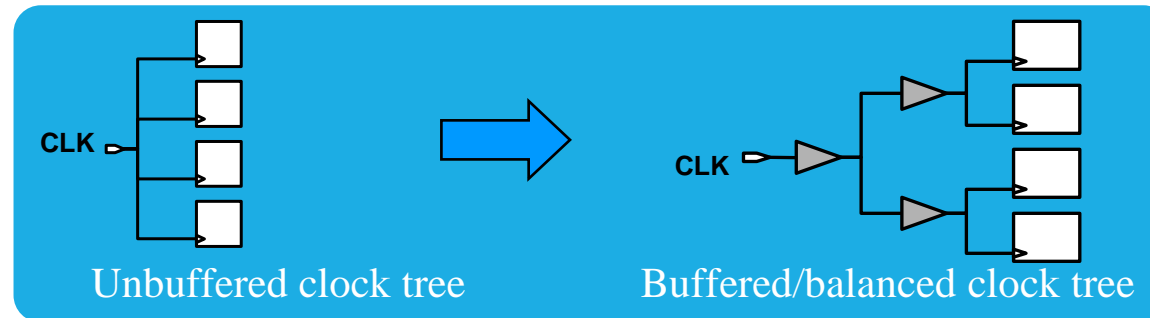
Clock Delay Problems

- ❑ All clock pins are driven by a single clock source
- ❑ All clock pins are from a source of clock pulses in various geometrical distances

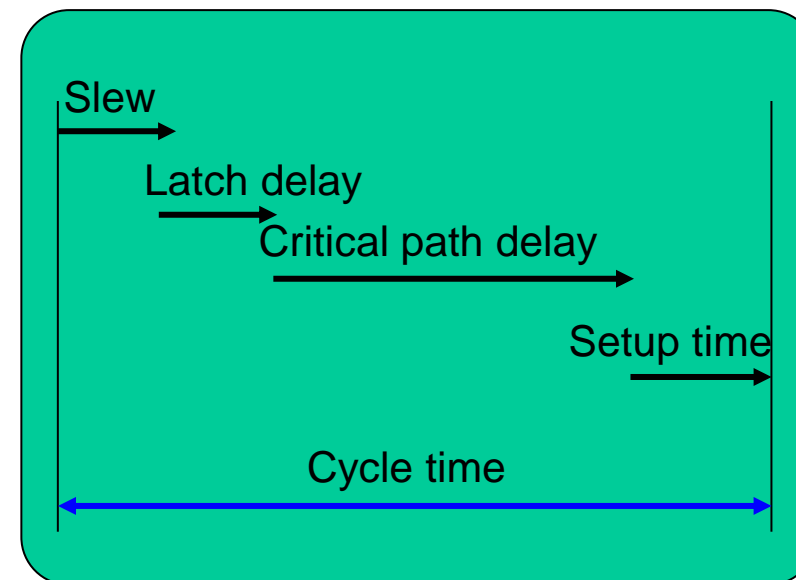
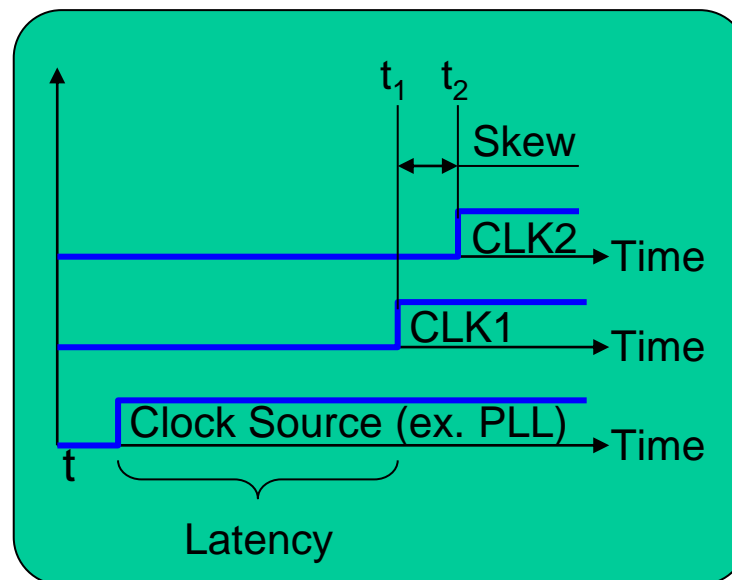


CTS Problem

- **CTS is the process of distributing clock signals to clock pins based on physical/layout information**
- After placement of cells the tree of synchronization is synthesized
- Balanced clock tree is synchronized with the addition of buffers
- After routing CT optimization is made



- **Goal**
 - Basic connectivity
- **Metrics**
 - Skew
 - Power
 - Area
 - Slew rates



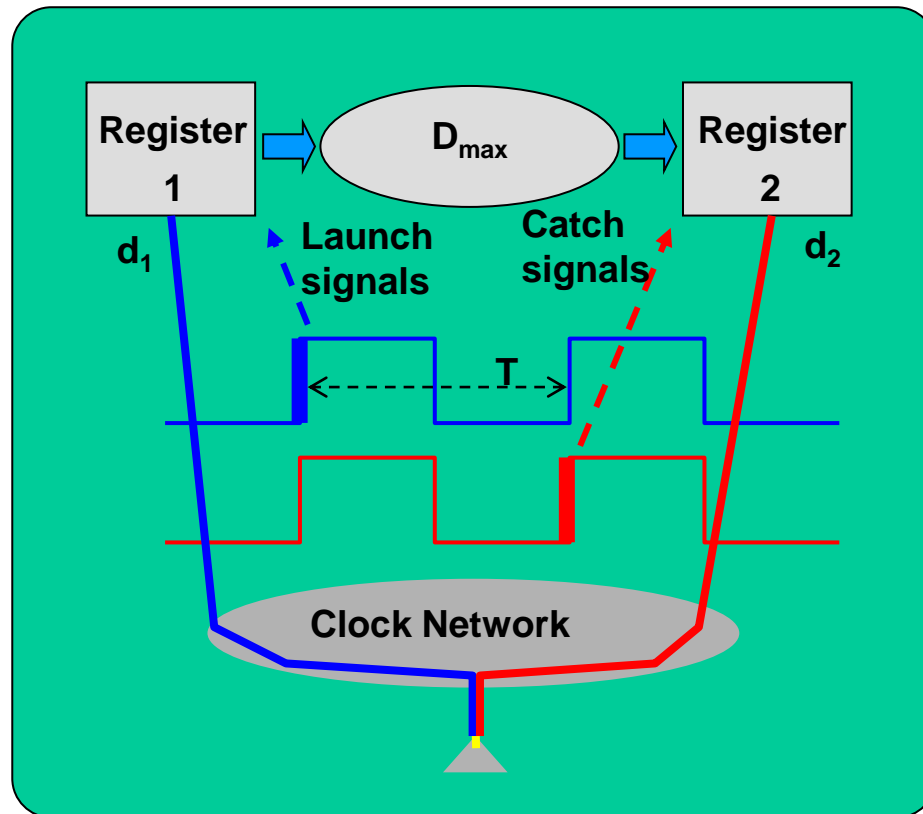
General Concepts: Clock Skew: Definition, Causes and Effects

- Clock Skew is the time difference between the arrival of the same edge of a clock signal at the Clock pin of the capture flop and launch flop

$$\text{Skew} = d_1 - d_2$$

$$\text{Zero skew: } d_1 = d_2$$

$$\text{Useful skew, } d_1 - d_2 = \delta_{12}$$



- ❑ **Global**

- Recommended - fastest

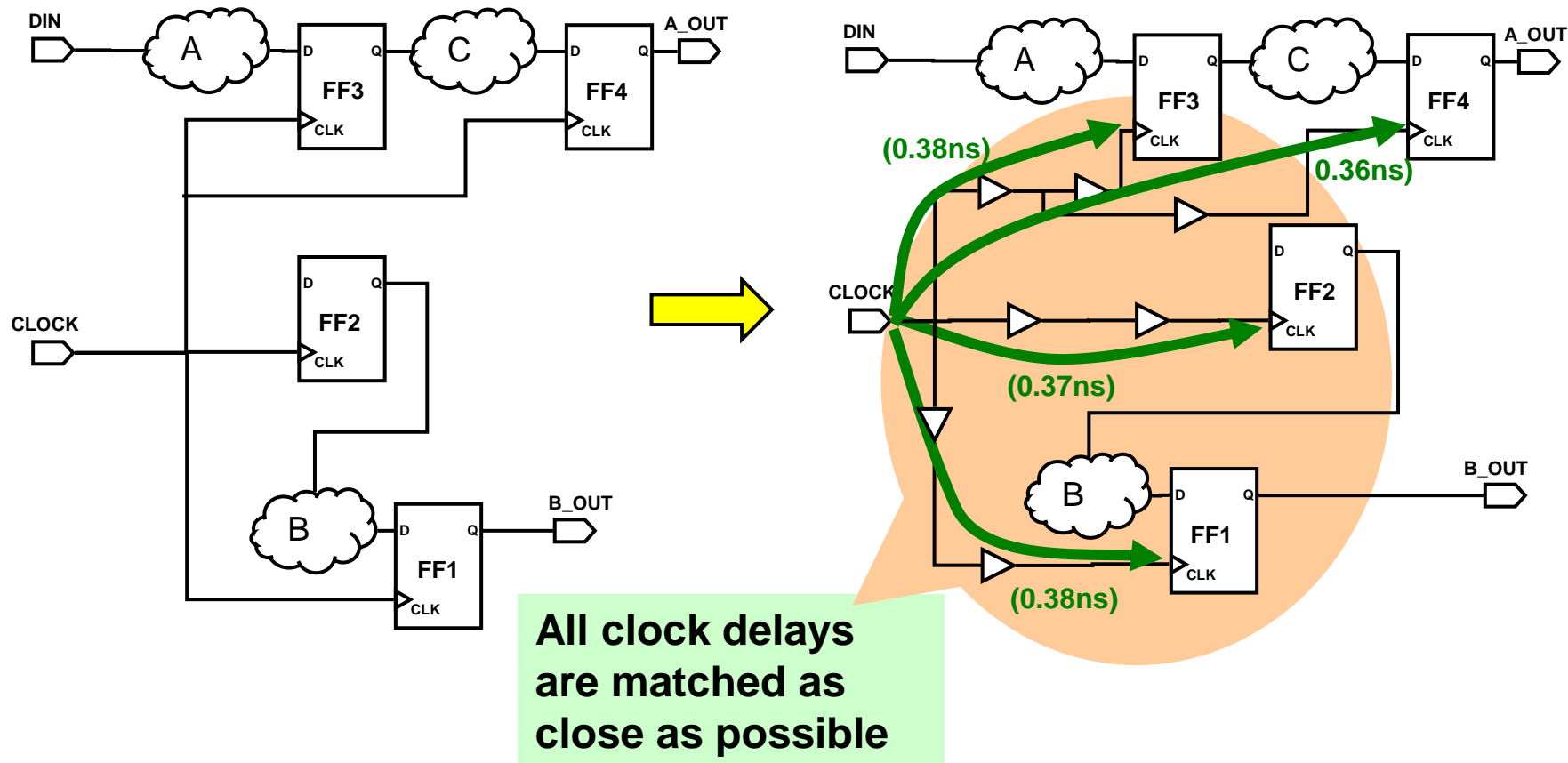
- ❑ **Local**

- Longer runtime

- ❑ **Useful**

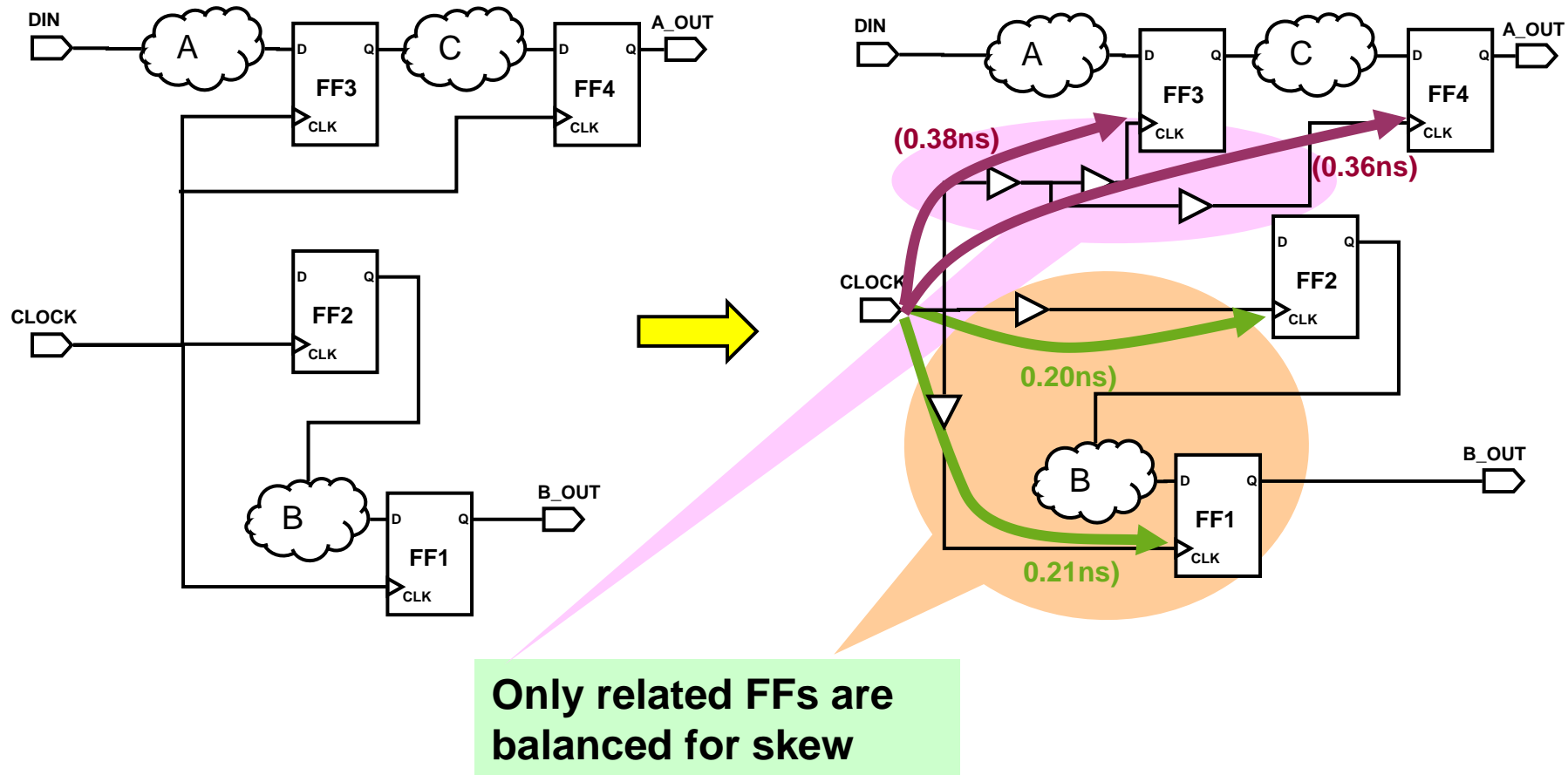
- Used to fix small timing violations

Clock Skew Types: Global



**Global skew is recommended – fastest runtime
(may add unnecessary buffers)**

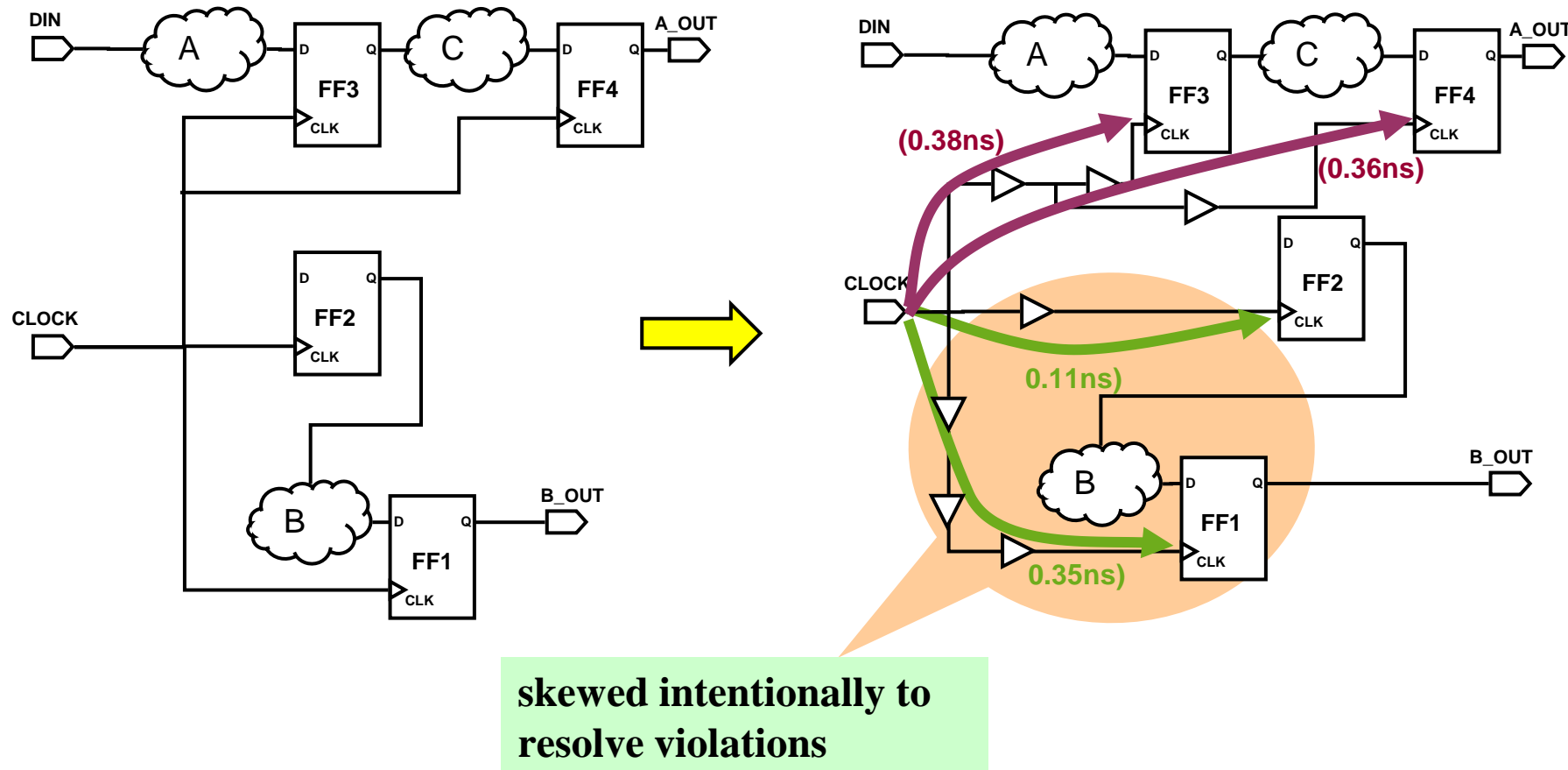
Clock Skew Types: Local



Possibly fewer buffers – much longer runtime

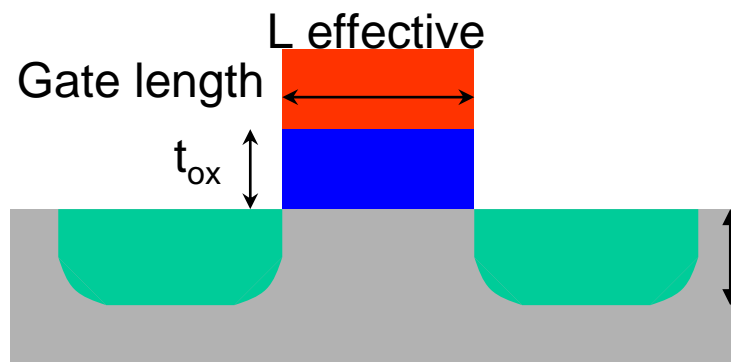
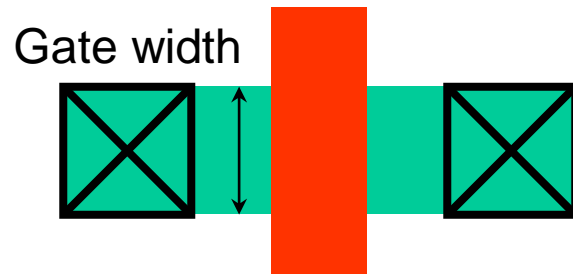
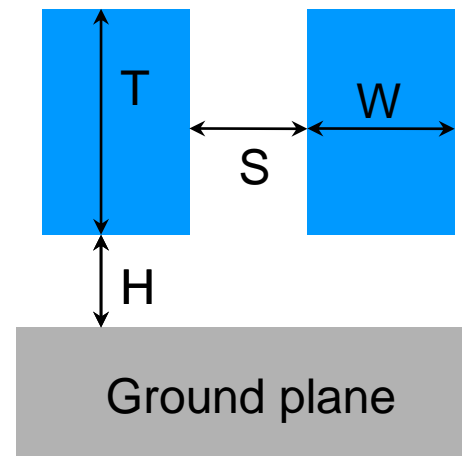
Clock Skew Types: Useful

- Useful skew- If the clock is skewed intentionally to resolve violations, it is called useful skew.

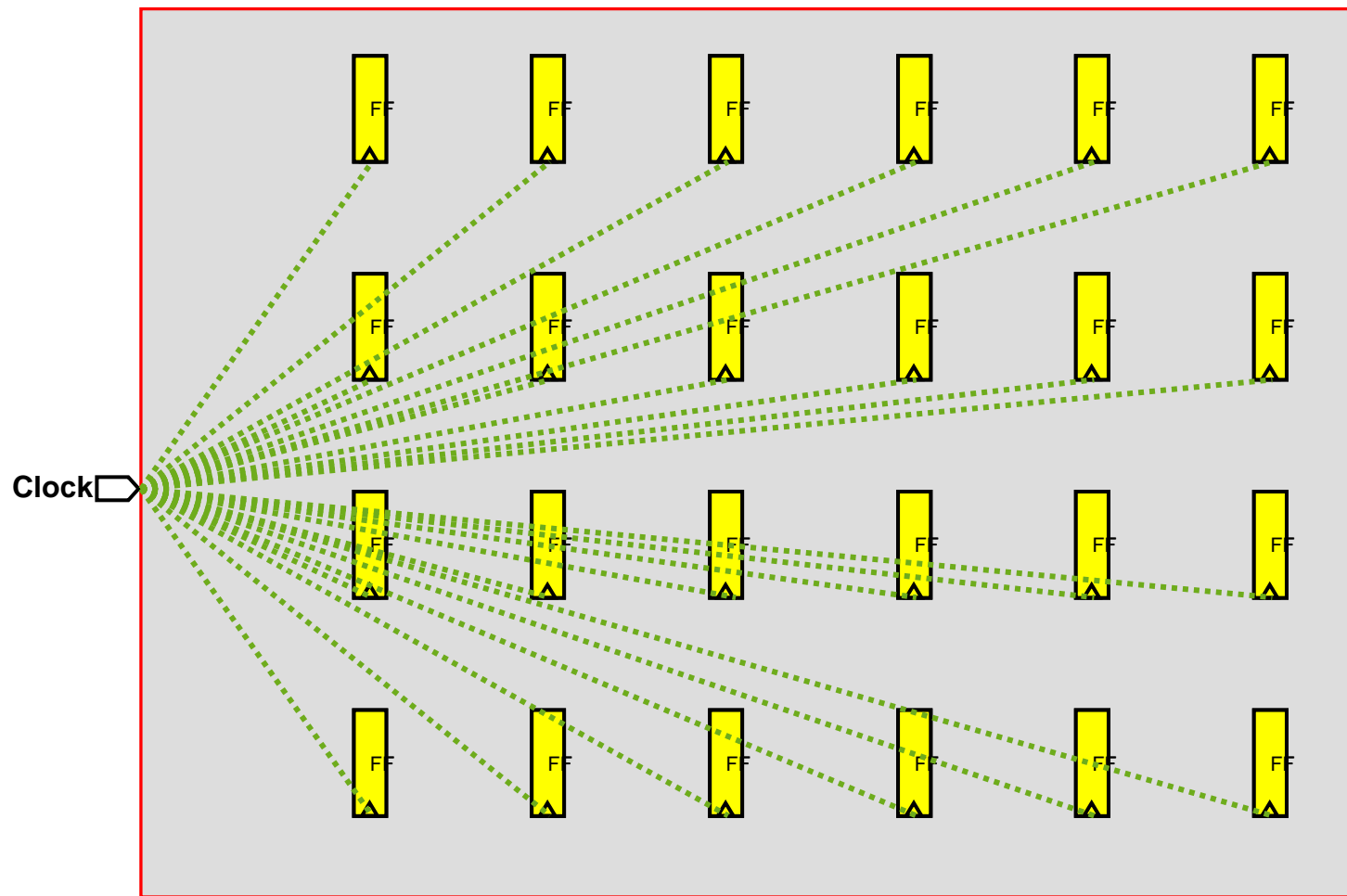


Clock Skew Variations

- Process variations
- Power supply noise
- Temperature variations
- ...

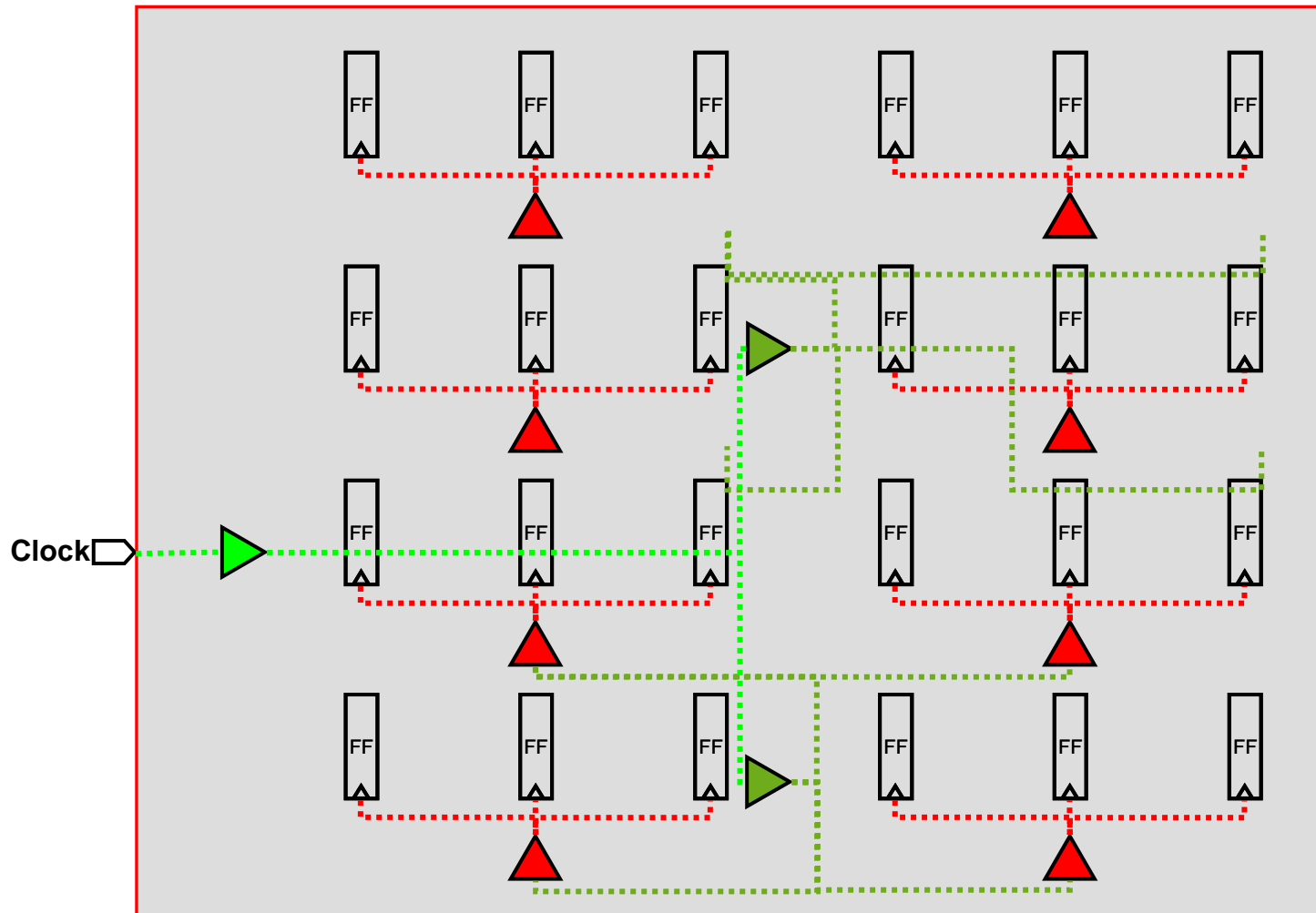


Starting Point



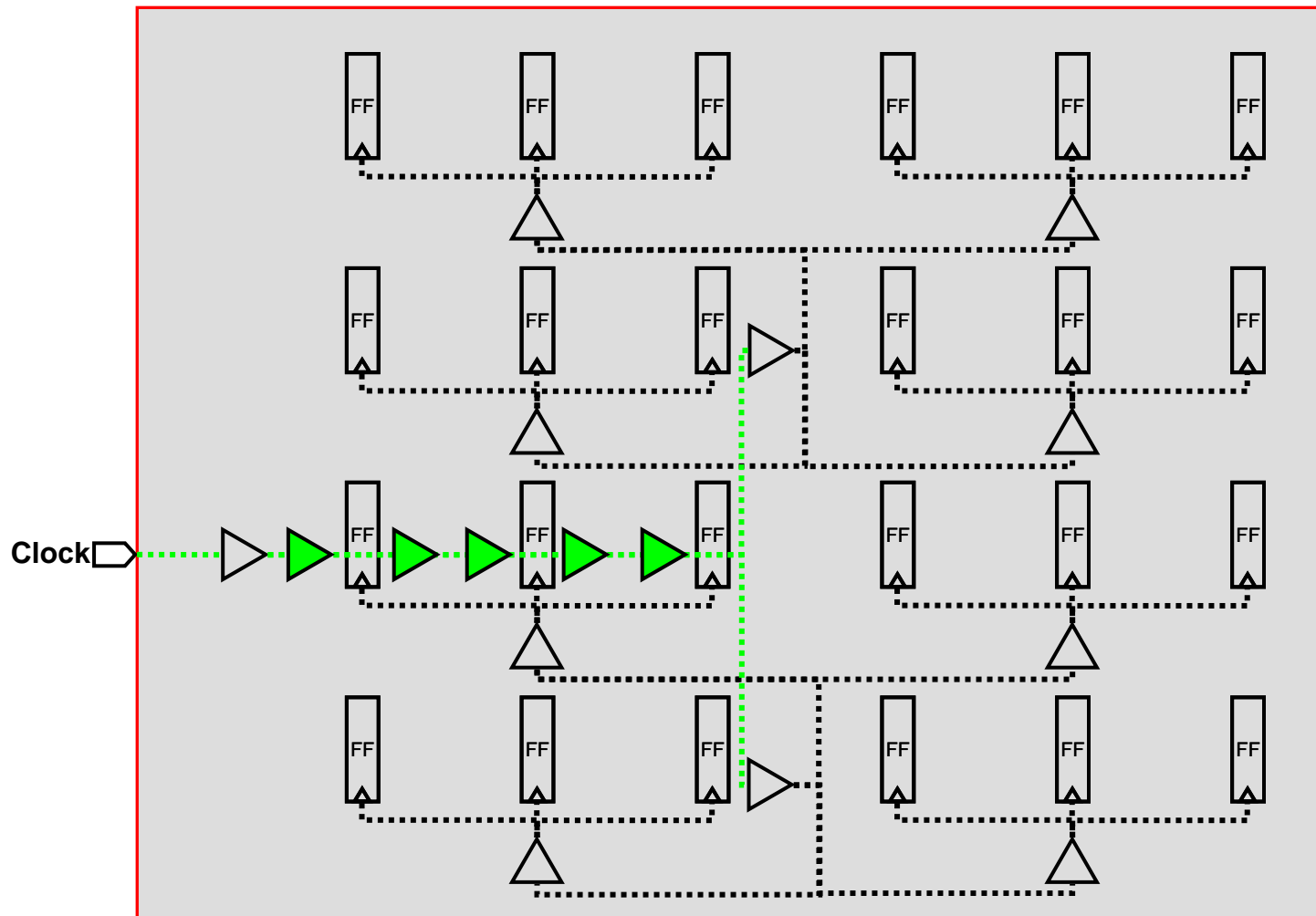
All clock pins are driven by a single clock source.

Clock Tree Synthesis (CTS) (1/2)



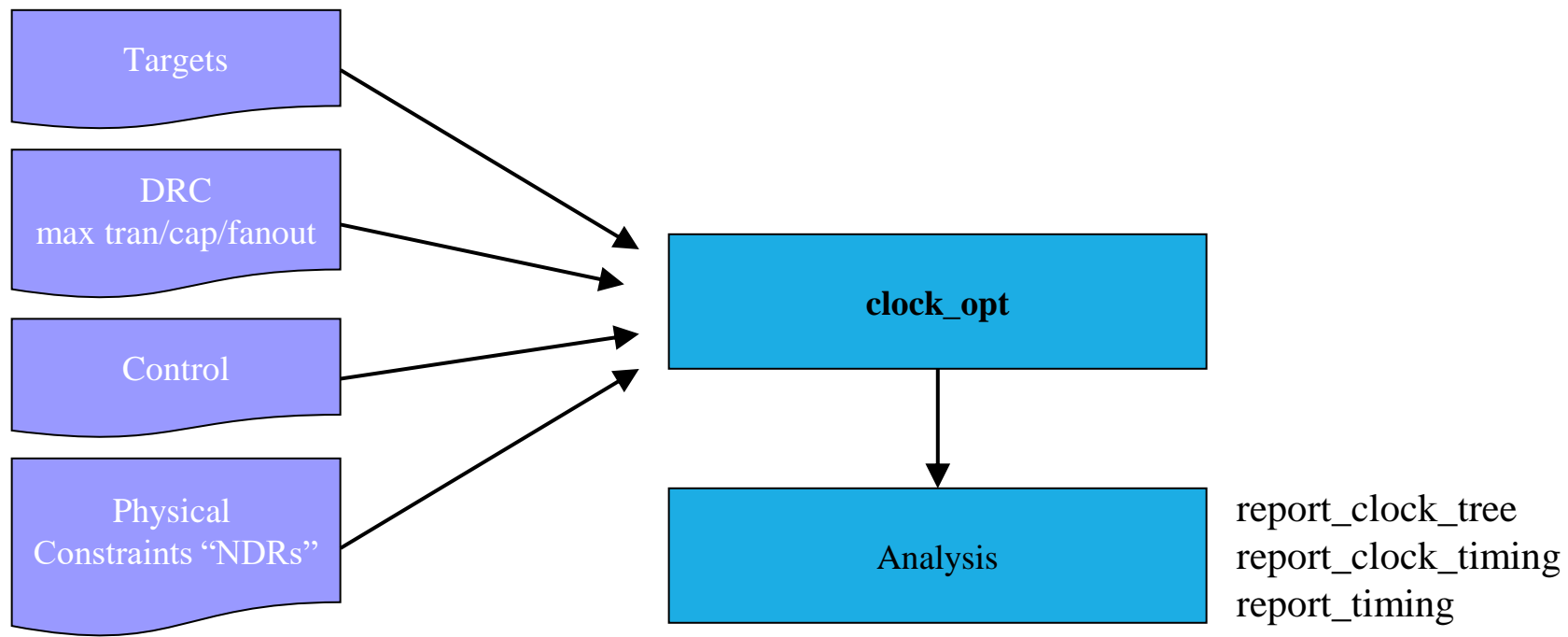
A buffer tree is built to balance the loads and minimize the skew.

Clock Tree Synthesis (CTS) (2/2)



A “delay line” is added to meet the minimum insertion delay.

Clock Tree Synthesis



Clock Trees Are Built in Two Phases

□ Phase 1: Clock Tree Synthesis (CTS)

- Builds an initial load-balanced, DRC-clean clock tree

□ Phase 1: Clock Tree Optimization (CTO)

- Performs cell sizing, relocation, buffer insertion
- Tries to achieve clock tree targets

Phase 1: CTS

❑ Meet the clock tree Design Rule Constraints (DRC):

- **Maximum transition delay**
(0.5ns, by default)
- **Maximum load capacitance**
(0.6pf, by default)
- **Maximum fanout**
(2000, by default)
- **Maximum buffer levels**
(50, by default)



Constraints are upper-bound goals. If constraints are not met, violations will be reported.

Phase 1: CTO

❑ Meet the clock tree targets:

- **Maximum skew**
(0ns, by default)
- **Minimum insertion delay**
(0ns, by default)



Targets are "nice to have" goals. If targets are not met, no violations will be reported.

Clock Tree Synthesis Goal and Flows

❑ Placement should be completed

- Acceptable congestion, setup timing, and logical DRCs
- High fanout signal nets (reset, scan enable, etc.) are buffered

❑ The goal of the clock tree synthesis design phase is to:

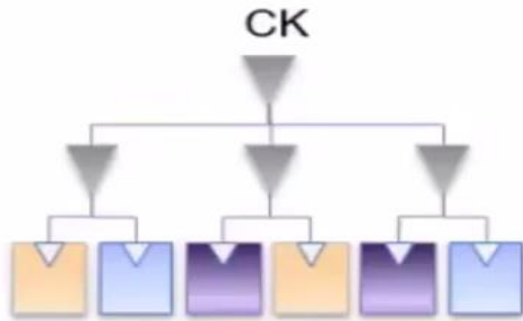
- Build the clock tree buffer structure
- Route the clock nets
- Optimize datapath logic for setup and hold timing, and DRCs

❑ Two CTS flows are supported:

- **Classic CTS flow:** CTS first, followed by data path optimization
- **Concurrent Clock & Data flow (CCD):** CTS and data path optimization performed concurrently
 - Recommended for timing-critical designs

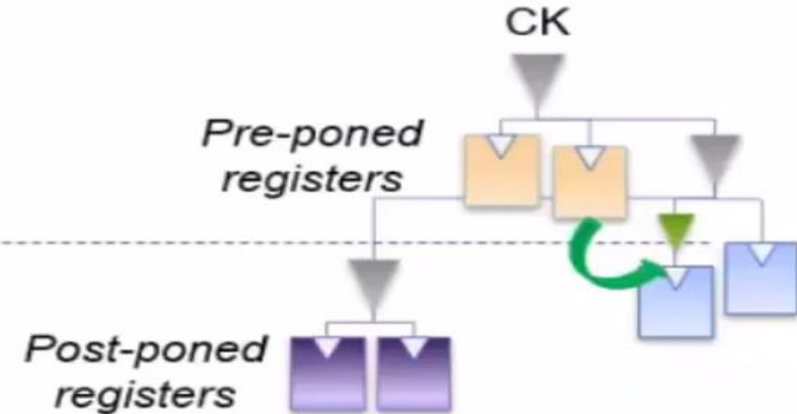
Comparing Classic CTS versus CCD

Classic CTS



- Clock tree is built while ignoring the data paths
- Goal: Minimize skew
 - Allows full clock-cycle reg-to-reg timing
- Data path is optimized post-CTS
 - Clock tree remains untouched

Concurrent Clock & Data



- Clock tree is built with full knowledge of data path timing
- Goal: Meet setup/hold timing
 - Reg-to-reg timing may be larger or smaller than a clock cycle
- Data path is optimized along with incremental clock tree modifications (green buffer)

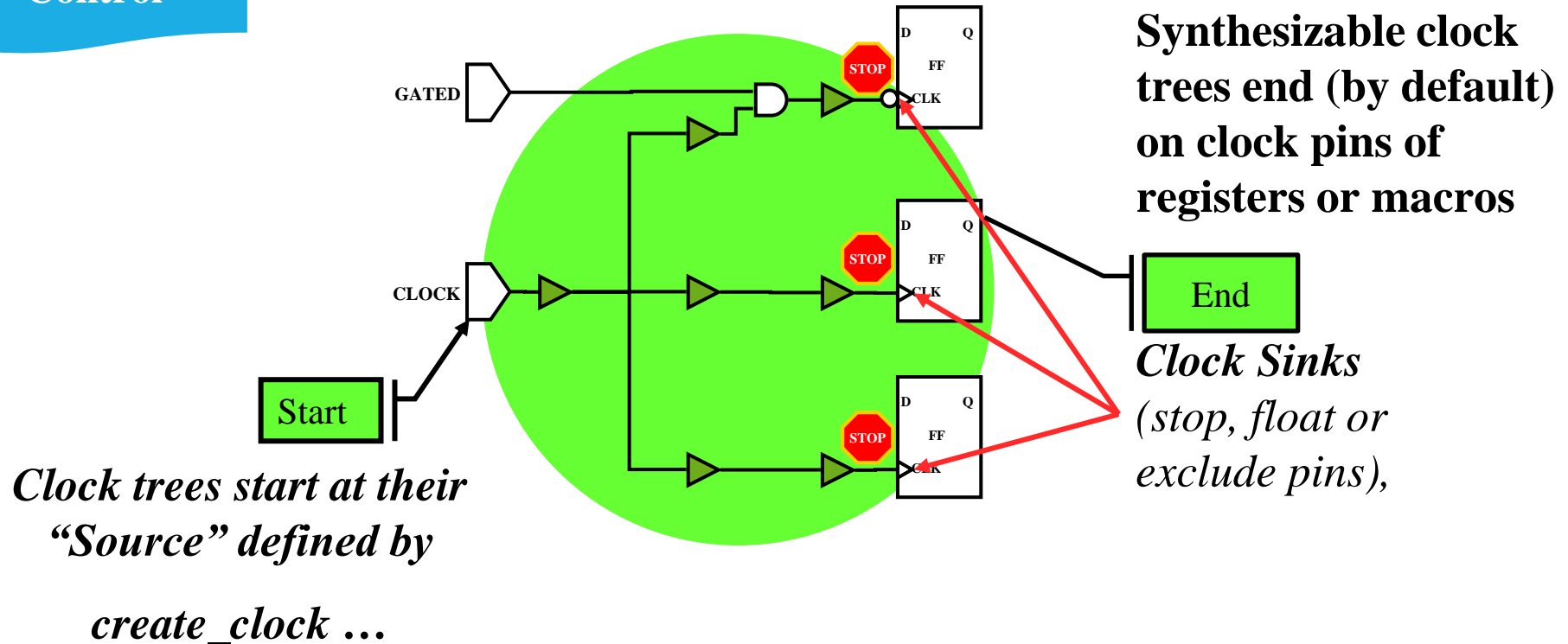


- ☐ When are the pros and cons of setting a tight constraint for `target_skew`?

- ☐ When are the pros and cons of setting a relaxed constraint for `max_transition`?

Where does the Clock Tree Begin and End?

Control



To see the source(s) of a clock, you can either run **report_clocks**, or use the sources attribute:

```
icc2_shell> get_attribute [get_clocks PCI_CLK] sources {pclk}
```

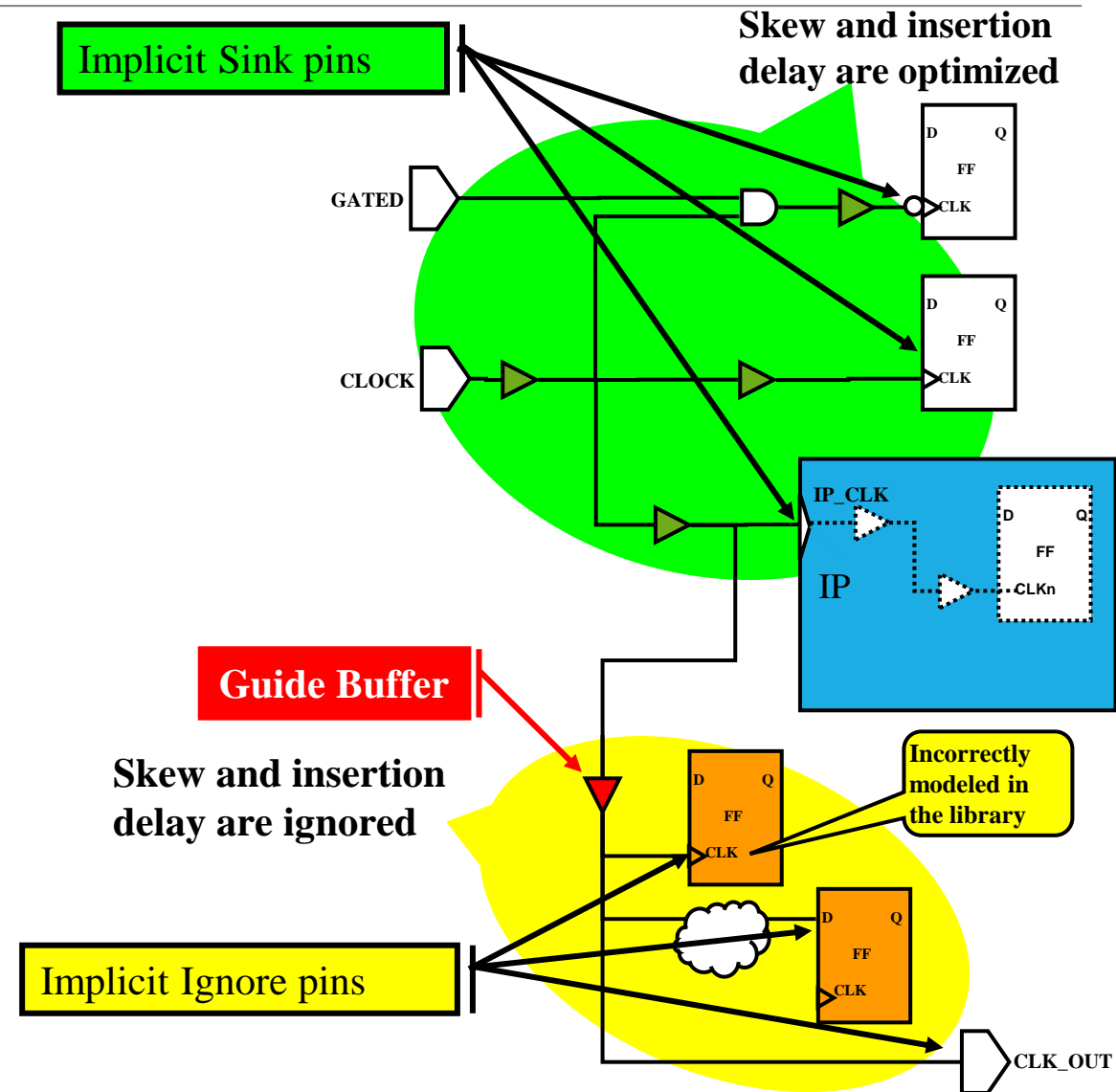
Balance Points: Sink and Ignore Pins

❑ Sink Pins:

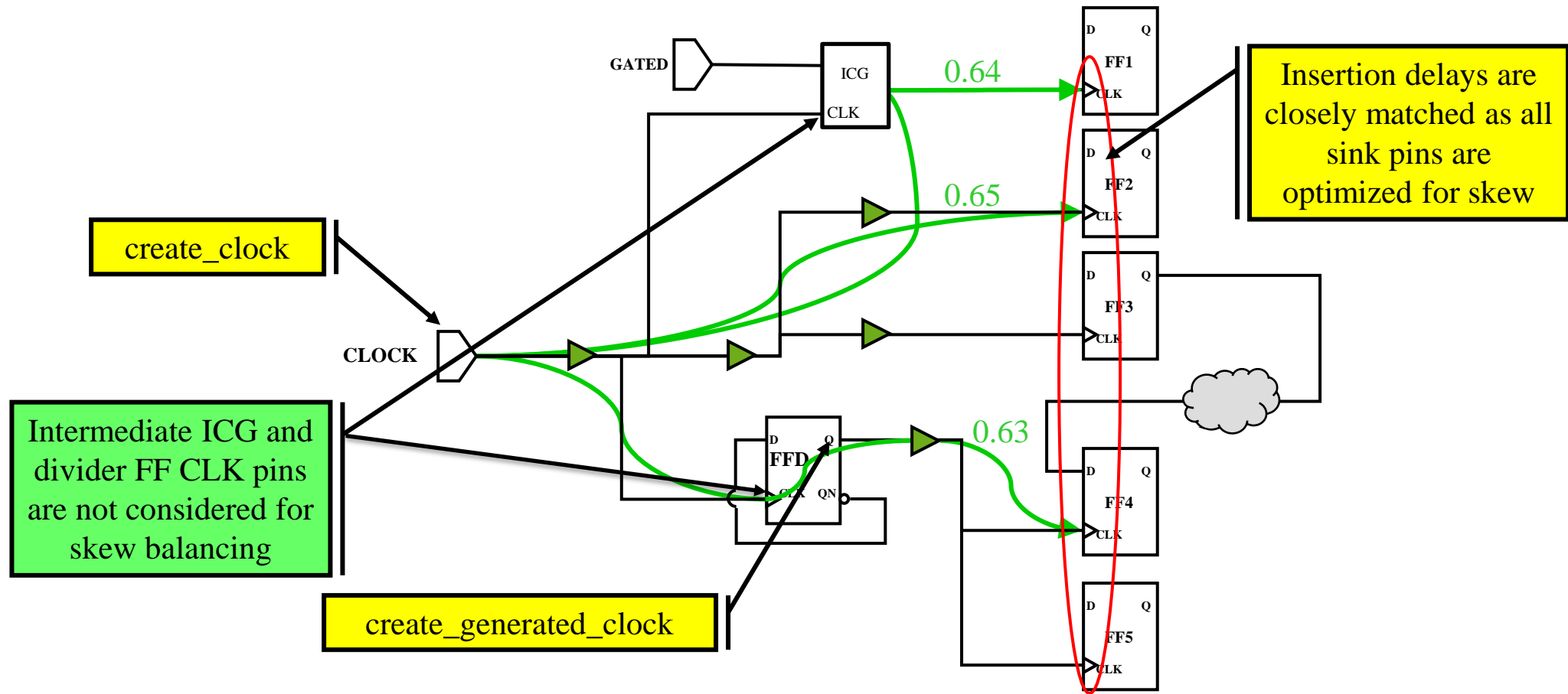
- CTS optimizes for DRC and clock tree targets (skew, insertion delay)
- Optionally, can consider internal insertion delay

❑ Ignore Pins:

- CTS adds a small buffer “guide buffer” to isolate all pins
 - Only DRCs fixed after the guide buffer
- CTS ignores skew and insertion delay targets



Generated and Gated Clocks

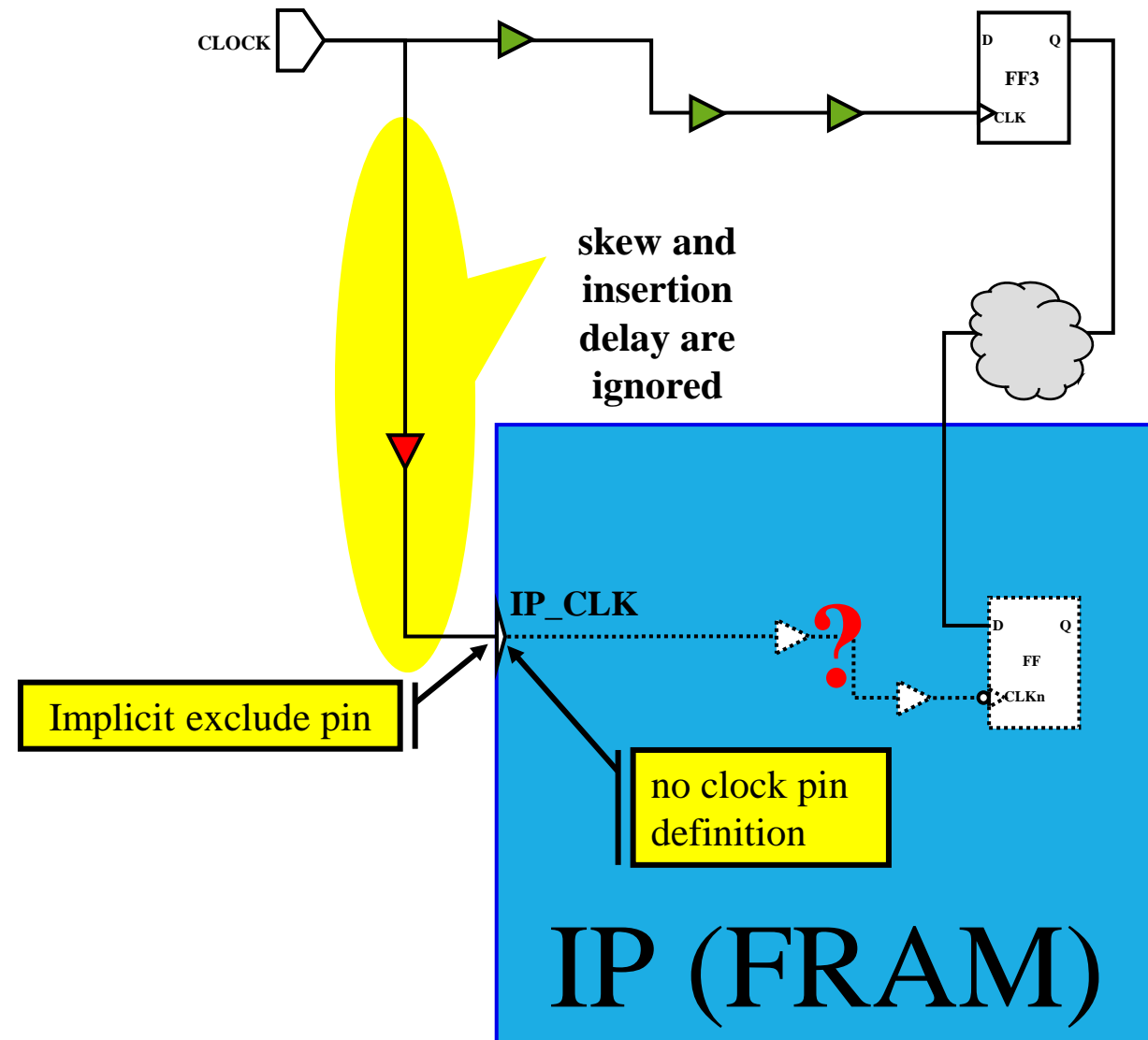


Master and generated clocks are part of the same clock domain. Skew is balanced globally (across all clock pins) within each clock domain

User-defined or Explicit Sink Pins

Scenario: If the clock pin inside a macro cell is correctly defined, CTS will treat that pin as an implicit Sink pin. In this example the clock pin is not defined. What is the problem here?

The macro's clock pin is marked as an implicit exclude "Ignore" pin – no skew optimization!

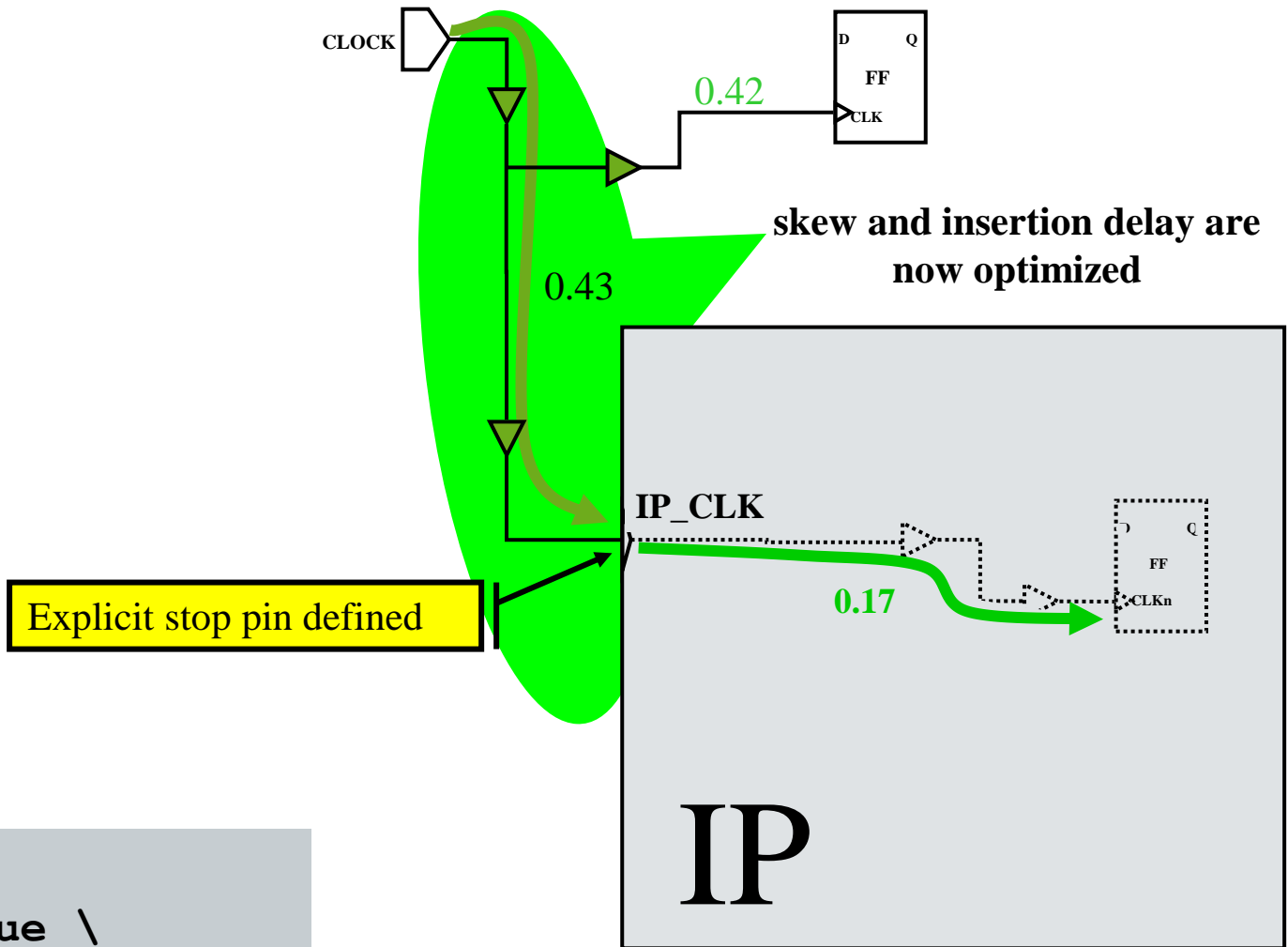


Defining an Explicit Sink Pin

Defining an explicit Sink pin allows CTS to optimize for skew and insertion delay targets.

CTS has no knowledge of the IP-internal clock delay – it can only “see” up to the stop pin!

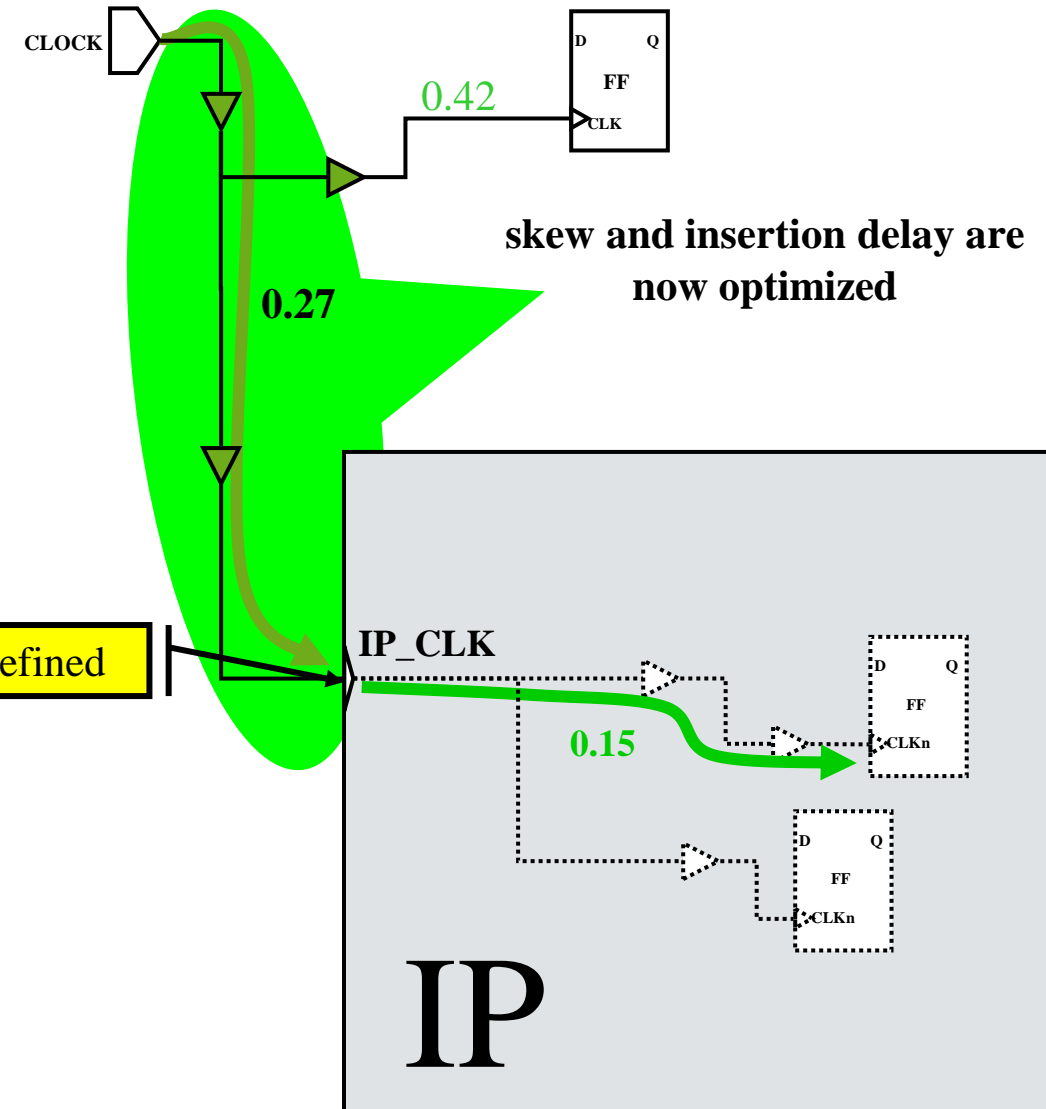
```
set_clock_balance_points \
    -consider_for_balancing true \
    -balance_points [get_pins IP/IP_CLK]
```



Defining an Explicit Sink Pin With Delay

Control

Defining an explicit sink pin with delay allows CTS to adjust the insertion delays based on specification.



Explicit delay balanced defined

```
set_clock_balance_points \
    -consider_for_balancing true \
    -corner ss125c \
    -delay 0.15 -late \
    -balance_points [get_pins IP/IP_CLK]
```

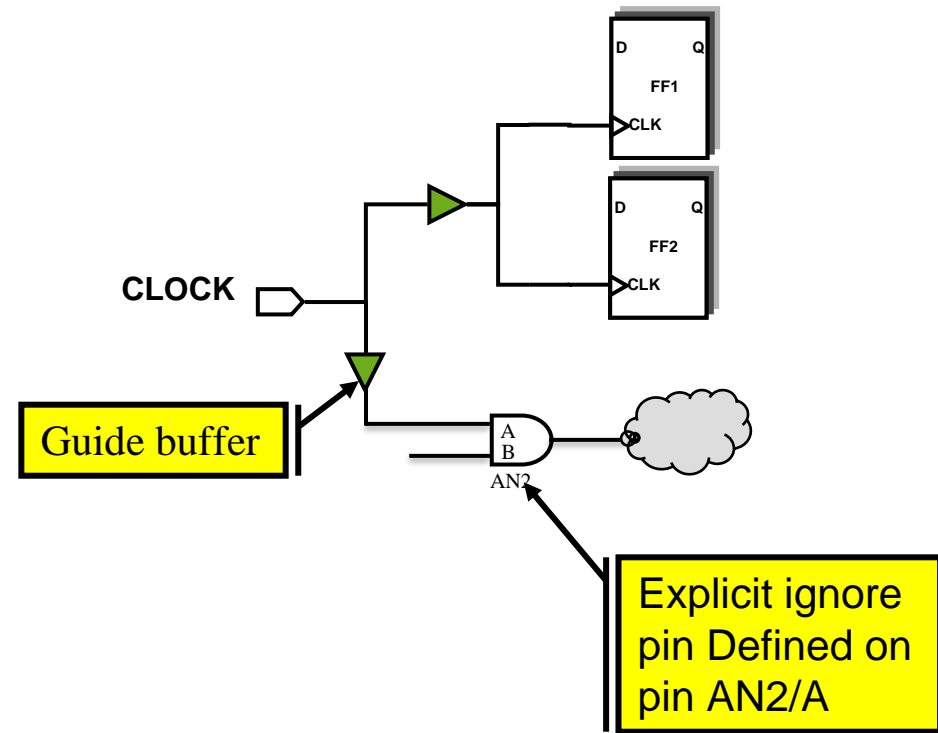


Defining an Explicit Ignore (or Exclude) Pin

Explicit ignore pins cause CTS to ignore this pin and down-stream sinks for skew balancing

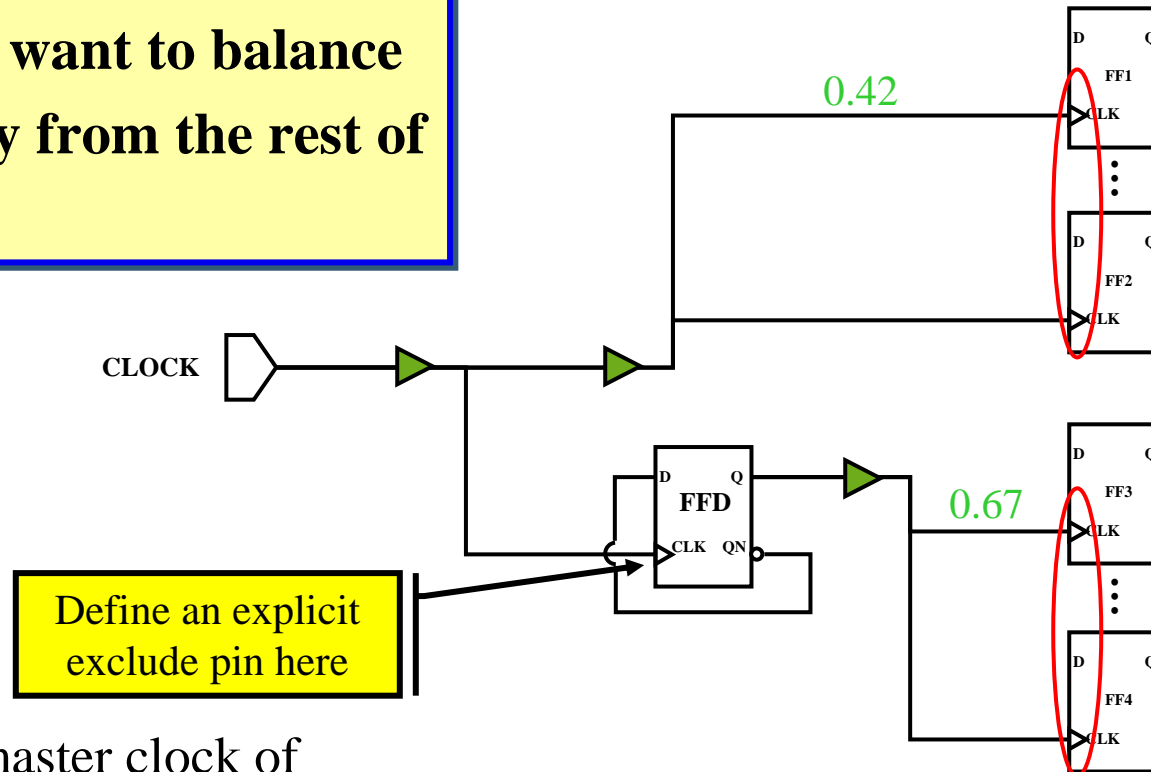
```
set_clock_balance_points \
    -clock Clock \
    -consider_for_balancing false \
    -balance_points [get_pins AN2/A]
```

→ CTS inserts a guide buffer; CTS engine fixes DRCs only along the clock path past ignore pins



Defining a Clock Skew Group

Defining a clock skew group if you want to balance a group of clock pins independently from the rest of clock tree.

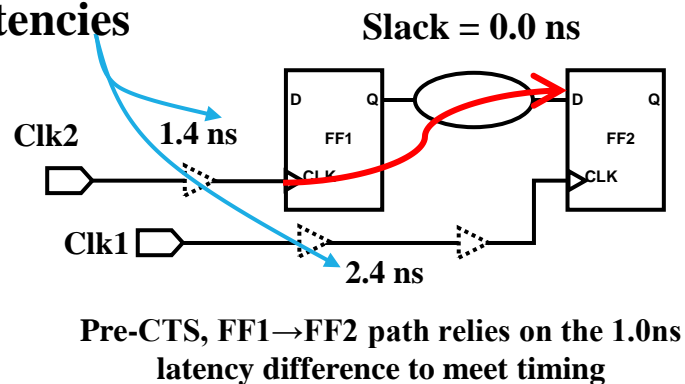


The target skew will be the same as the master clock of this skew group. You cannot set the skew independently.

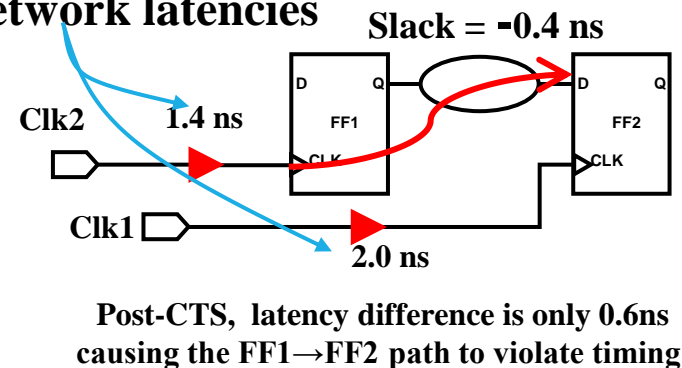
```
Create_clock_skew_group -mode TEST -objects {FF1/CLK FF2/CLK}
```

- ❑ **The default target skew and target latency is 0 ns**
 - SDC uncertainty and network latency constraints are ignored
- ❑ **Relax clock skew targets for non-timing critical clocks**
 - Reduces overall buffer count, power and run time
- ❑ **Specify network latency targets to help post-CTS timing (see below)**
 - An alternative method using automatically-derived balance points will also be shown

Ideal network latencies



Propagated network latencies



CCD: Skew and Latency Considerations

- ❑ For the classic CTS flow, it is important to specify accurate
- ❑ skew and latency targets For the CCD flow:
 - Target skew is not relevant, because the goal is to intentionally introduce skew to meet timing
 - Target latency can be applied if needed (for chip-level inter-block timing considerations, for example), but keep in mind that the final agencies have larger skews compared to classic CTS

User-Defined Clock Tree Targets

```
remove_clock_tree_options -all -target_skew -target_latency  
set_clock_tree_options -target_skew 0.2 -corners [all_corners]
```

→ max skew target of 0.2 applied to all corners,
all clocks, all modes

```
current_corner C1
```

```
set_clock_tree_options -target_skew 0.1
```

→ max skew target of 0.1 applied in C1, all clocks,
all modes

```
set_clock_tree_options -clocks CLK1 -target_latency 1.4
```

```
set_clock_tree_options -clocks CLK2 -target_latency 2.4
```

→ latency target of 1.4 applied in C1, to CLK1 in
current mode

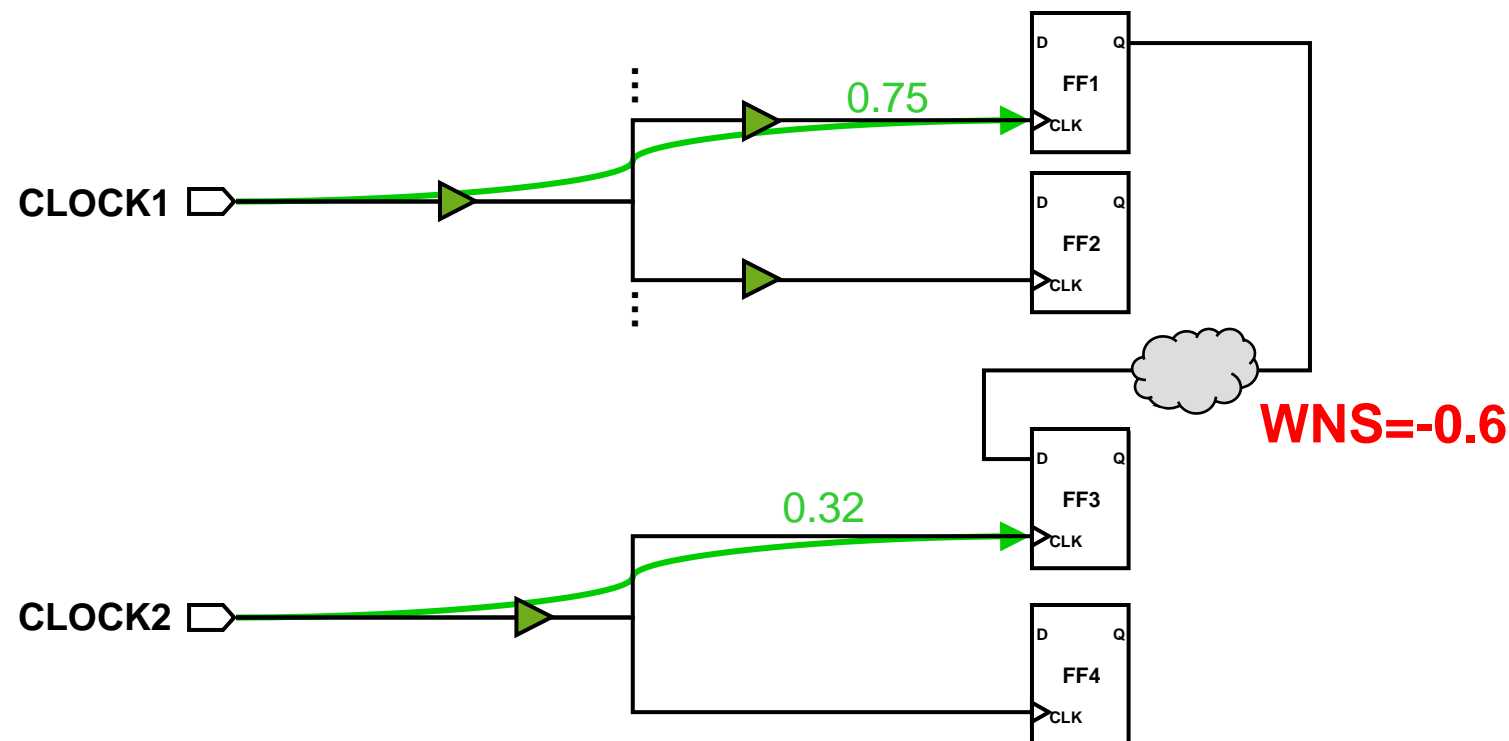
→ latency target of 2.4 applied in C1, to CLK2 in
current mode

```
report_clock_tree_options
```

Targets apply to:

- All clocks if no clock is specified
- Current corner if no corner is specified
- Current mode if clock is specified, otherwise all modes

Balancing Multiple Synchronous Clocks



By default CTS does not perform inter-clock skew balancing
→ May result in timing violations

Create Balancing Groups

❑ There are two ways to balance the clock groups:

- Manual balancing constraints:

```
foreach mode {m1 m2} {  
    current_mode $mode  
    create_clock_balance_group -name grp1 \  
        -objects [get_clocks "CLK1 CLK2"]  
}
```

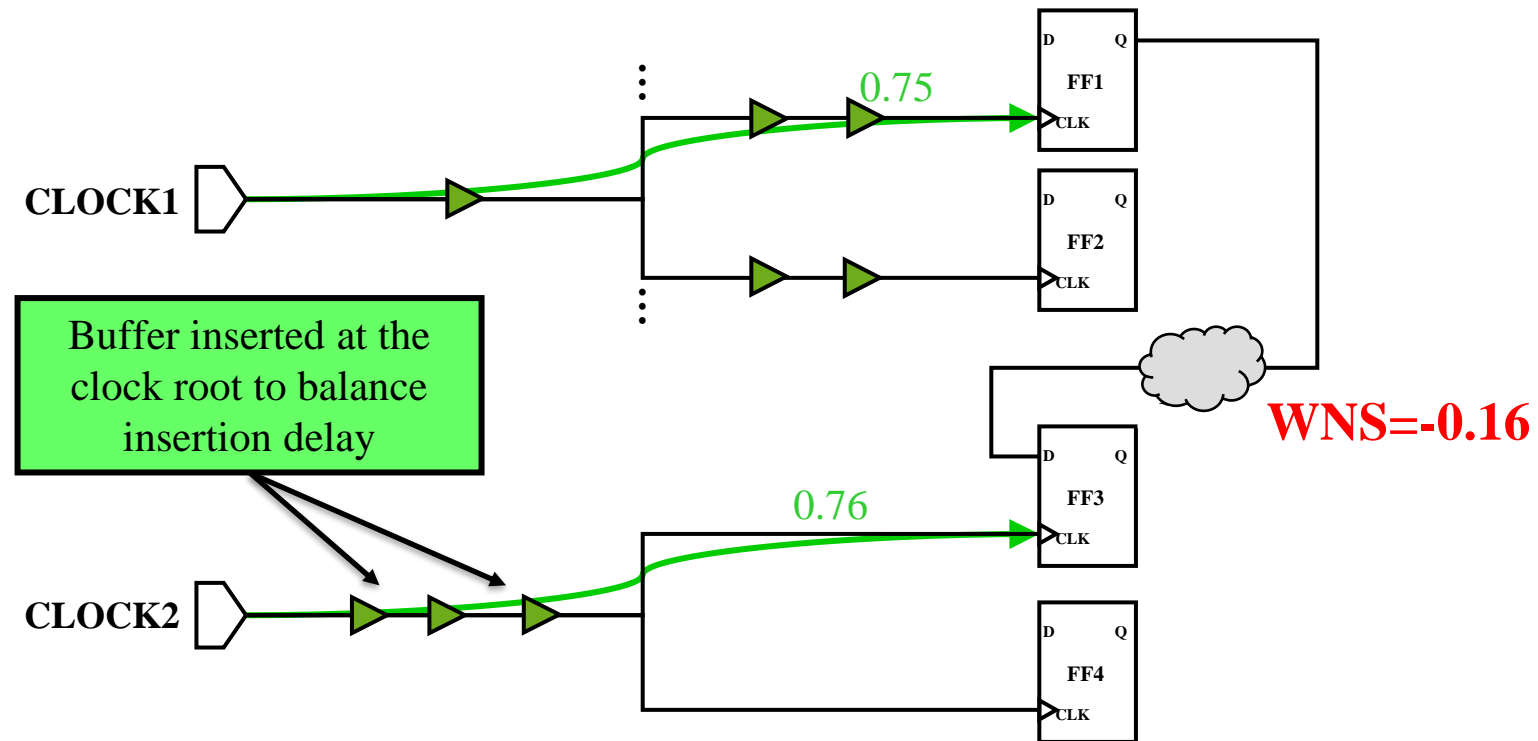
- Auto-derived balancing constraints:

```
derive_clock_balance_constraints -slack_less_than -0.3  
report_clock_balance_groups
```

- Constraints will be derived for all modes
- Only clocks with cross-clock paths meeting the -slack_less_than specification will be selected

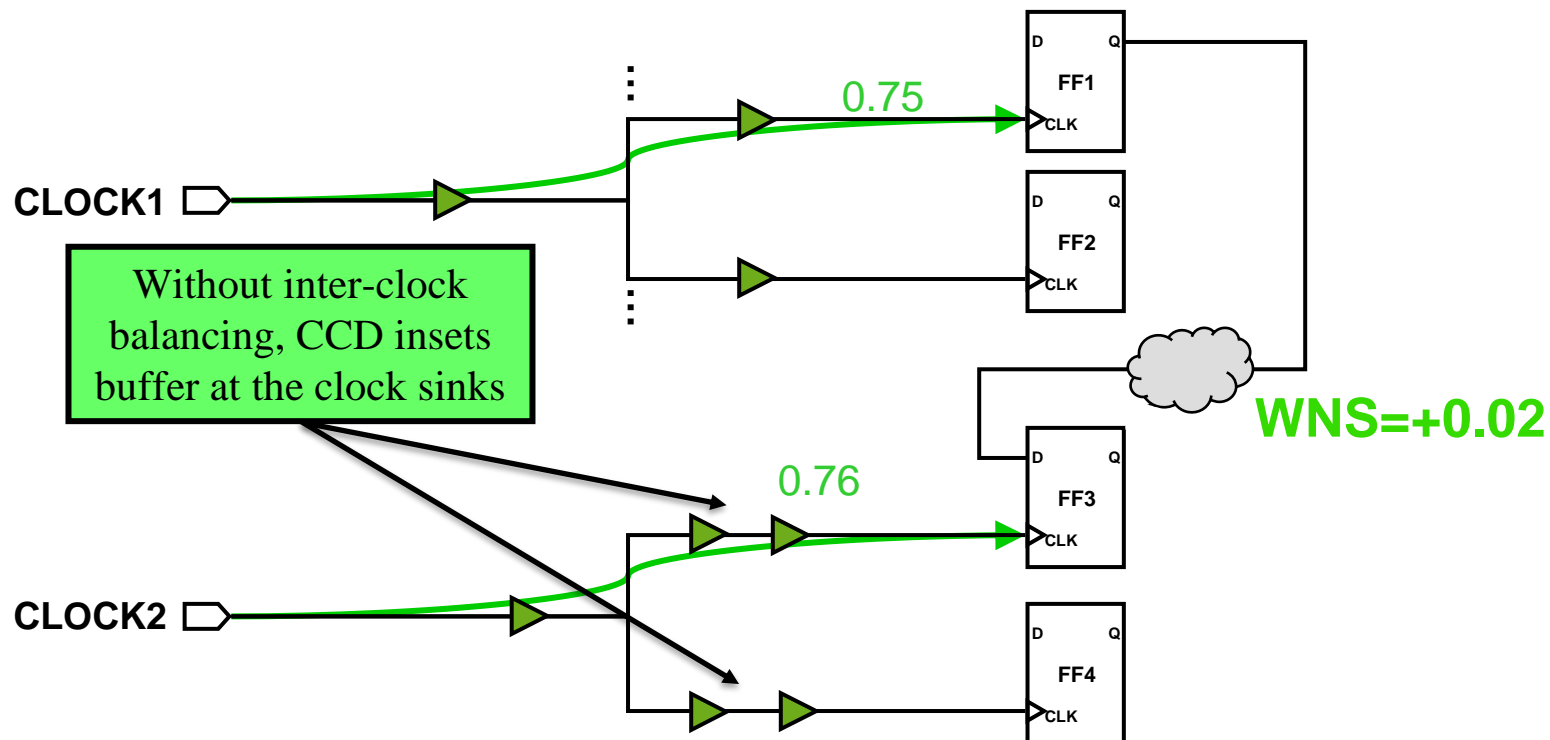
Balancing Occurs During clock opt

Balancing occurs during the `build_clock` stage



Inter-clock Balancing is Important for CCD

- If inter-clock balancing is not enabled for the CCD flow, CCD optimization will try to meet timing by skewing individual sinks (instead of the root)
 - Much longer runtime
 - Inserts many more buffers



Control CTS Cell Selection

- ❑ In the following example, only \$cts_libcells are used to build the clock trees. Modify as needed:

```
set cts_libcells [get_lib_cells \  
    "*/INVX*_LVT */BUFX8_LVT */BUFX16_LVT \  
    */MUX*_LVT */AO* */CG* */FF*"]  
  
set_lib_cell_purpose -exclude cts [get_lib_cells]  
set_lib_cell_purpose -include cts $cts_libcells  
set_dont_touch $cts_libcells false
```

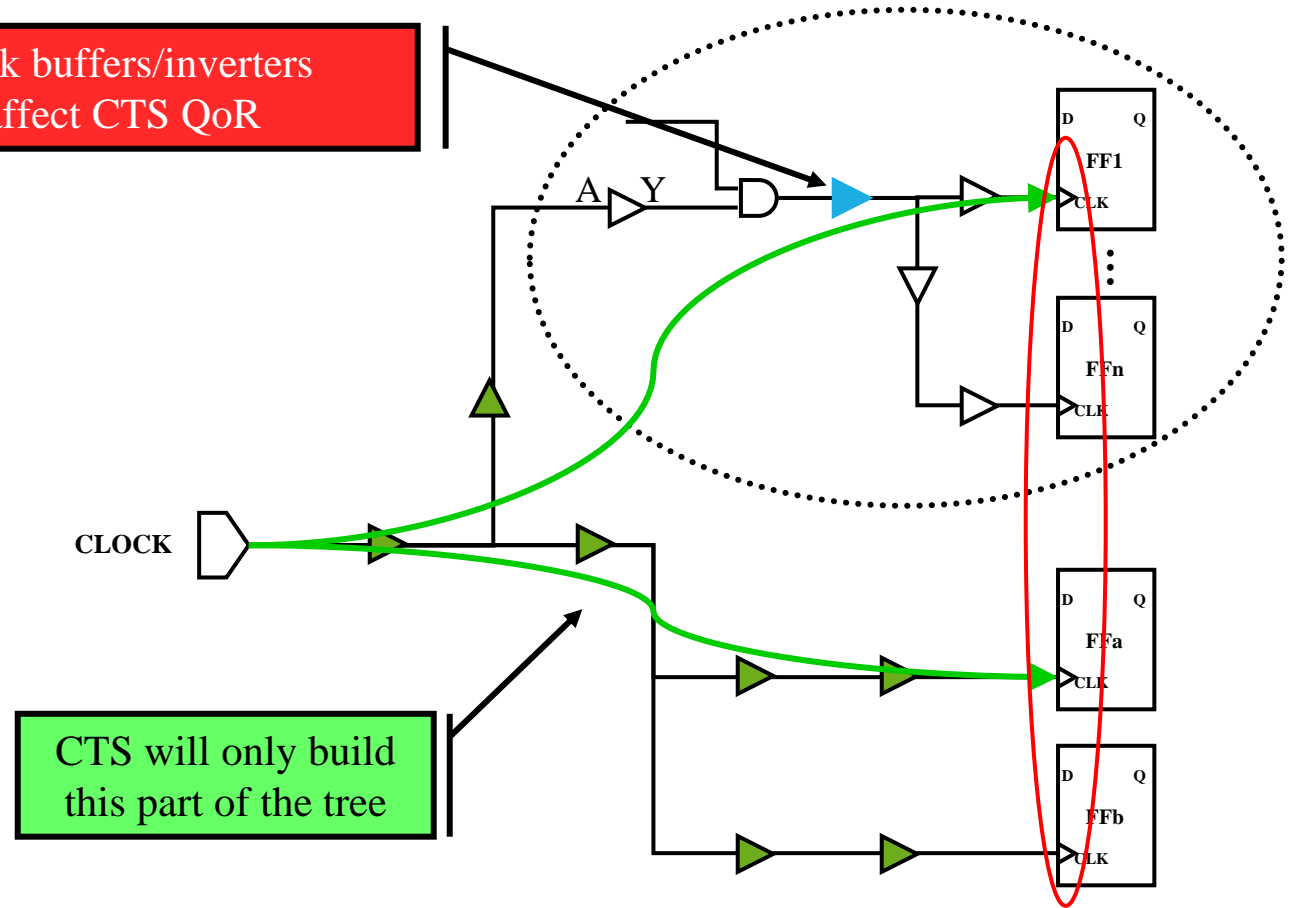
- In addition to buffers/inverters, include logically equivalent cells of any gates (MUX, AND, ICG, etc.) and FlipFlops*1 in the clock tree Allows these to be resized, if needed
- Always-on buffers should be added, to allow always-on CTS for MV designs
 - If a clock branch needs to feed through a shut-down voltage area, always-on CTS inserts AO buffers- otherwise, must go around it

Pre-existing clock Buffers Are Removed

Control

Pre-existing clock buffers/inverters may negatively affect CTS QoR

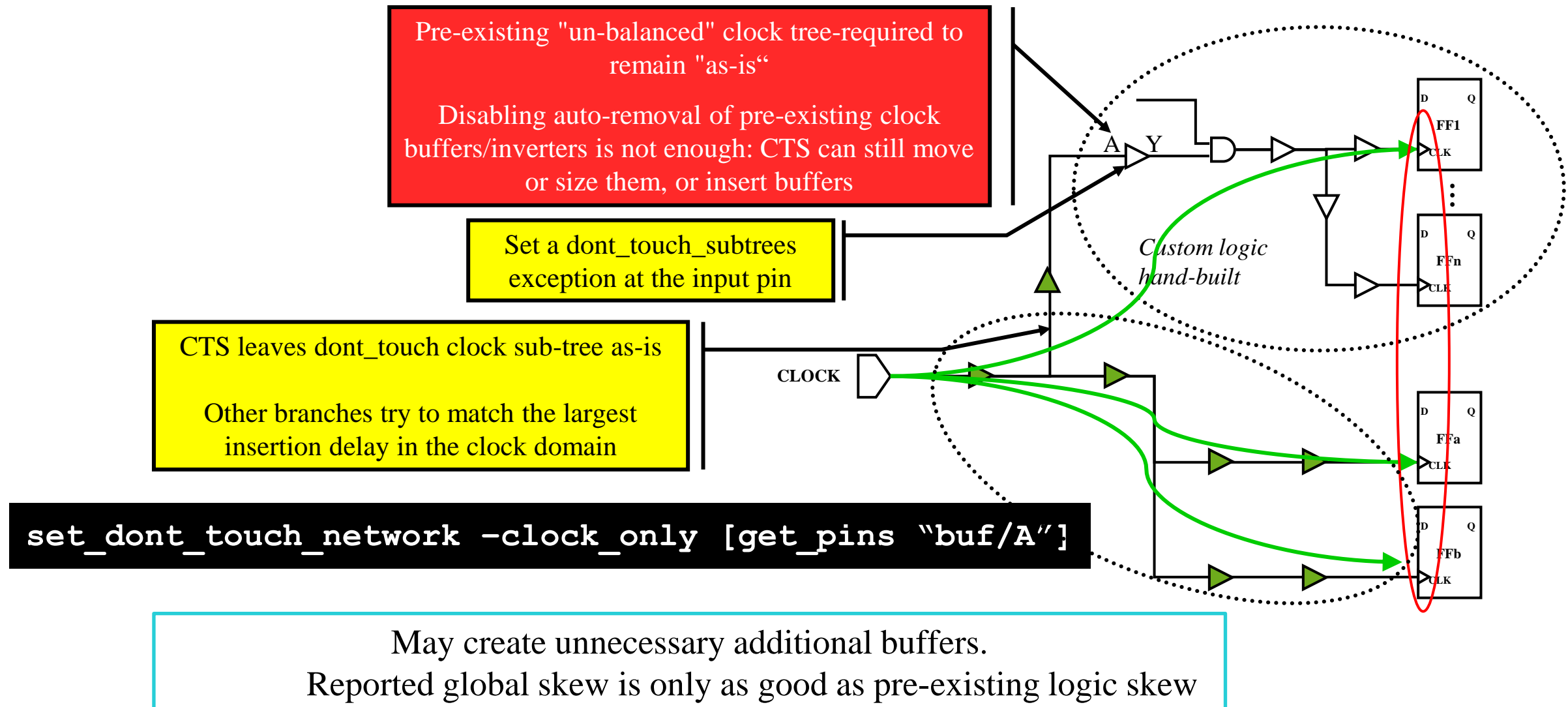
By default, CTS removes pre-existing clock buffers/inverters and builds a balanced clock tree with fewer buffers



`cts.compile.remove_existing_clock_tress`

Default: **true**

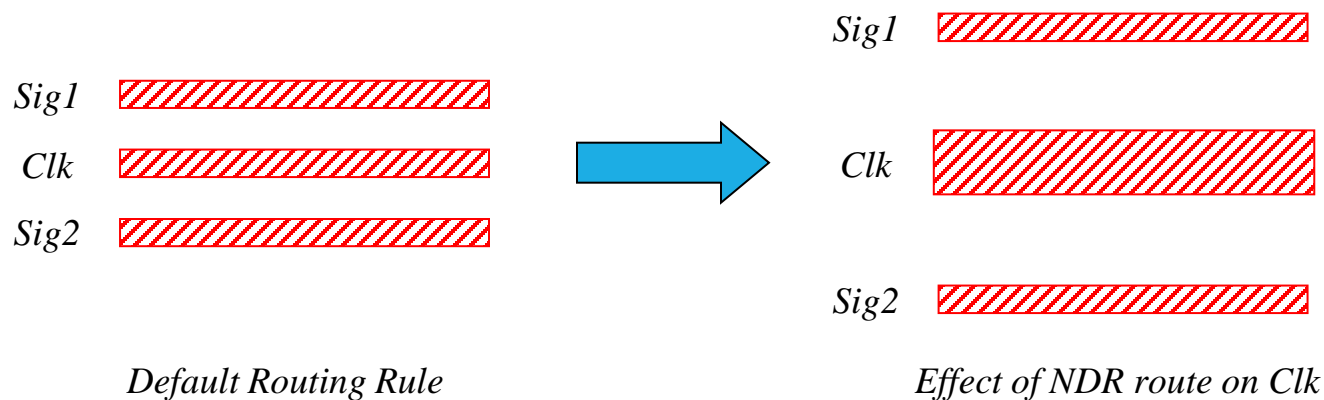
Preserving Pre-Existing Clock Trees



Non-Default Clock Routing

Physical Constraints “NDR”

- ❑ ICC II can route the clocks using non-default routing rules (NDR), e.g. double-spacing, double-width, shielding
 - NDR rules are treated as physical DRCs- reported as violations if not met
- ❑ Non-default rules are often used to “harden” the clock, e.g. to make the clock routes less sensitive to Cross Talk or EM effects



Defining Non-Default Routing and Via Rules

- Define the NDR rules:

```
create_routing_rule 2xS_2xW_CLK_RULE \
-widths {M1 0.11 M2 0.11 M3 0.14 M4 0.14 M5 0.14} \
-spacings {M1 0.4 M2 0.4 M3 0.48 M4 0.48 M5 1.1} \
-cuts { ... \
    {VIA3 {Vrect 1} } \
    {VIA5 {Vrect 1} }}
```

Layer Name

cutNameTB1 name

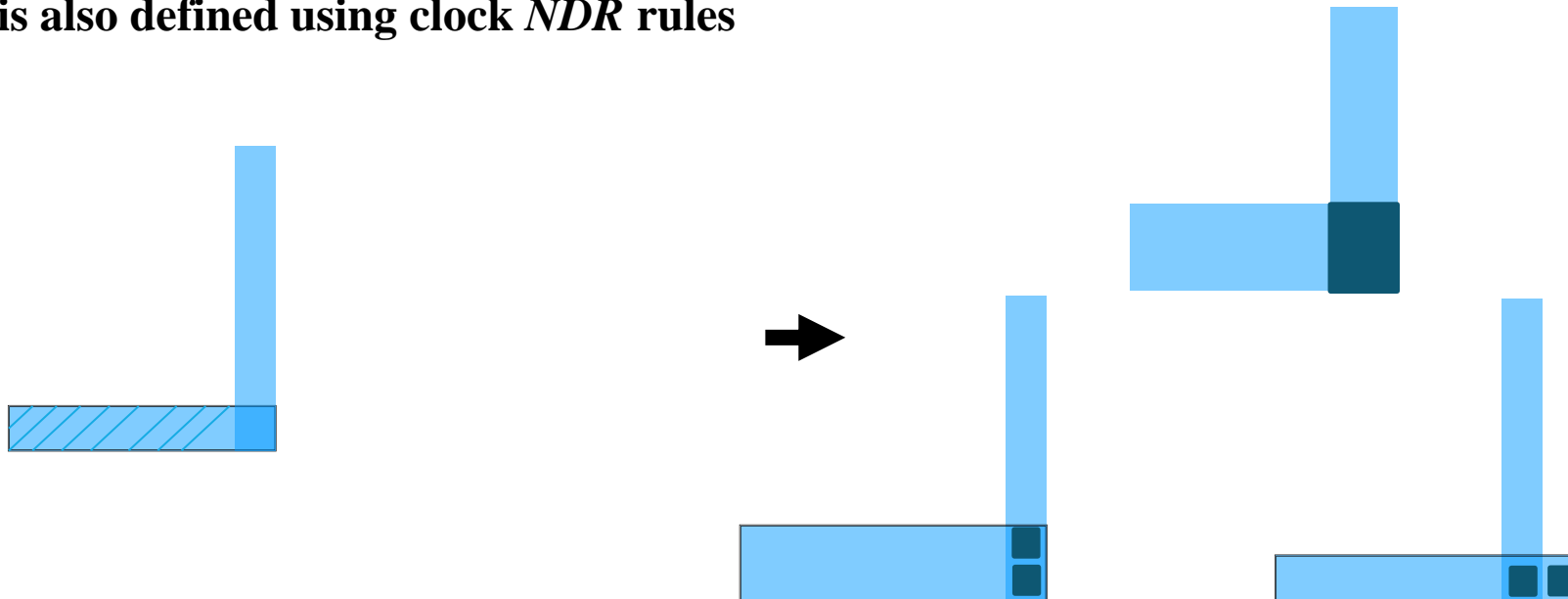
Minimum # of cuts

- With the -cuts option you use “symbolic” via names defined in the technology file as cutNameTbl
- This simplifies the NDR definition because one symbolic via can match many specific vias and arrays

Example Technology File

```
Layer "VIAS" {
    fatTblThreshold = (0, 0.13, 0.26)
    fatTblFatContactNuinber = ( "2,3,4", "5,6,20", "5,6,20")
    fatTblFatContactMinCuts = ( "1,1,1", "1,1,2", "2,2,4")
    cutNameTbl = ( Vsq , Vrect )
    cutWidthTbl = ( 0.05, 0.05 )
    cutHeightTbl = ( 0.05, 01.013 )
}
contactCode "VIA34,LH" {
    contactCodeNumber = 5
    cutwidth = 0.13
    cutHeight = 0.05
    ...
}
contactCode "VIA34,LV" {
    contactCodeNumber = 6
    cutwidth = 0.05
    cutHeight = 0.13
    ...
}
ContactCode "VIA34_P" {
    contactCodeNumber = 20
    cutwidth = 0.05
    cutHeight = 0.05
    ...
}
```

- ❑ **Clock nets are hardened further by often requiring 100% via optimization to enhance reliability**
 - Minimum size single cut vias are replaced with:
 - Multiple-cut via arrays and/or
 - and/or Larger “square” or “bar” cuts
- ❑ **This is also defined using clock *NDR* rules**



Defining Via NDRs with -vias

- ❑ You may also use the -vias option
- ❑ This will require you to specify the exact Contactcode names defined in the technology file
- ❑ This example is the equivalent of using

-cuts { {via3 {vrect 1} } shown two pages earlier

```
create_routing_rule 2xS_2xW_CLK_RULE \  
-widths {M1 0.11 M2 0.11 M3 0.14 M4 0.14 M5 0.14} \  
-spacings {M1 0.4 M2 0.4 M3 0.48 M4 0.48 M5 1.1} \  
-vias { \  
{VIA34_LH 1x1 R} {VIA34_LH 1x1 NR} {VIA34_LV 1x1 R} \  
{VIA34_LV 1x1 NR} {VIA34_LH 1x2 R} {VIA34_LH 1x2 NR} \  
{VIA34_LH 2x1 R} {VIA34_LH 2x1 NR} {VIA34_LV 1x2 R} \  
{VIA34_LV 1x2 NR} {VIA34_LV 2x1 R} {VIA34_LV 2x1 NR}} \  
##  
VIA45 and VIA56 definitions go here}
```

Applying Non-Default Routing Rules

❑ Configure clock tree routing:

```
create_routing_rule 2xS_2xW_CLK_RULE ...  
  
set_clock_routing_rules -rule 2xS_2xW_CLK_RULE \  
    -min_routing_layer M4 \  
    -max_routing_layer M5
```

❑ Note that the NDR was specified also for M1-M3

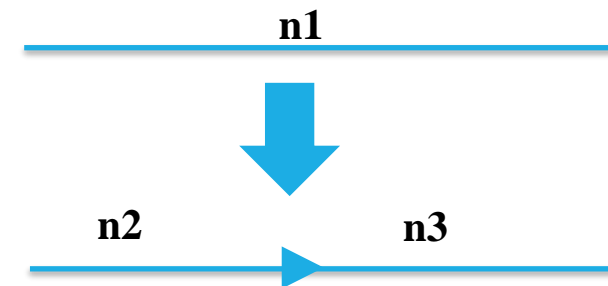
- Although the clocks will be primarily routed on M4/M5 as shown above, they need to route on lower layers to connect to the standard cell pins, therefore should be covered by an NDR as well!



❑ **ICC II CTS supports various types of NDR specifications when building the clock tree**

1. Net-specific NDR from `set_routing_rule <net_list>`:
NDR will not get propagated to newly created nets
2. Net-specific NDR from `set_clock_routing_rules -nets`:
NDR will get propagated to newly created nets
3. Clock-specific NDR from `set_clock_routing_rules -clocks`:
Supports separate NDRs for root, sink and internal nets which are clock-specific
4. Global NDR from `set_clock_routing_rules`:
Supports separate NDRs for root, sink, and internal nets

❑ **The order of priority is $1 > 2 > 3 > 4$**

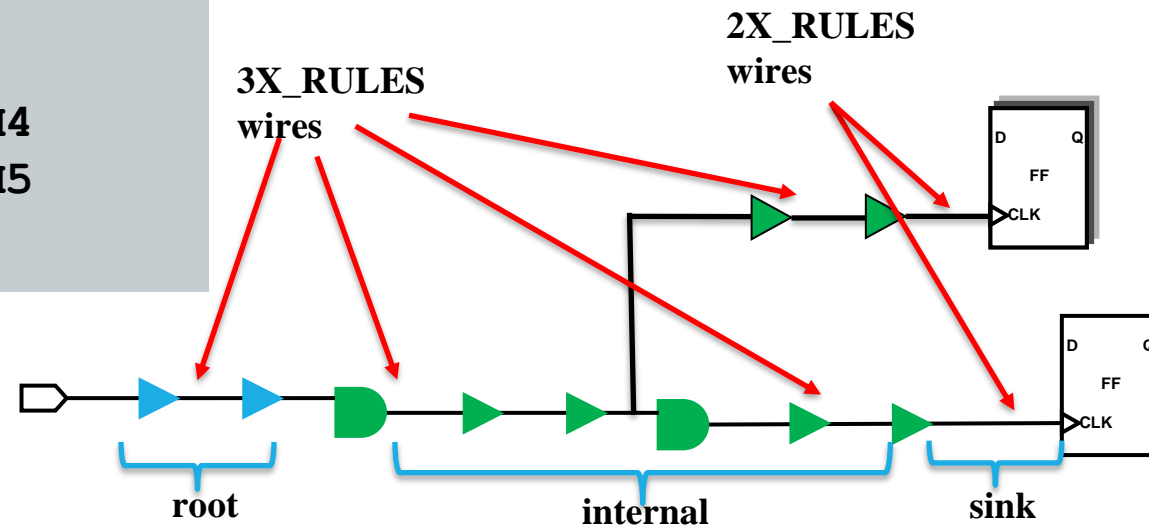


Different NDRs for Root, Internal, Sink Nets

```
set_clock_routing_rules
  -rules 3X_RULES
  -min_routing_layer M4
  -max_routing_layer M5
```

```
set_clock_routing_rules
  -net_type sink
  -rules 2X_RULES
  -min_routing_layer M4
  -max_routing_layer M5
```

-net_type:
root | internal | sink



The highlighted options allow you to specify less aggressive NDR rules for clock tree “leaf” nets, while allowing more aggressive NDRs for the other nets.

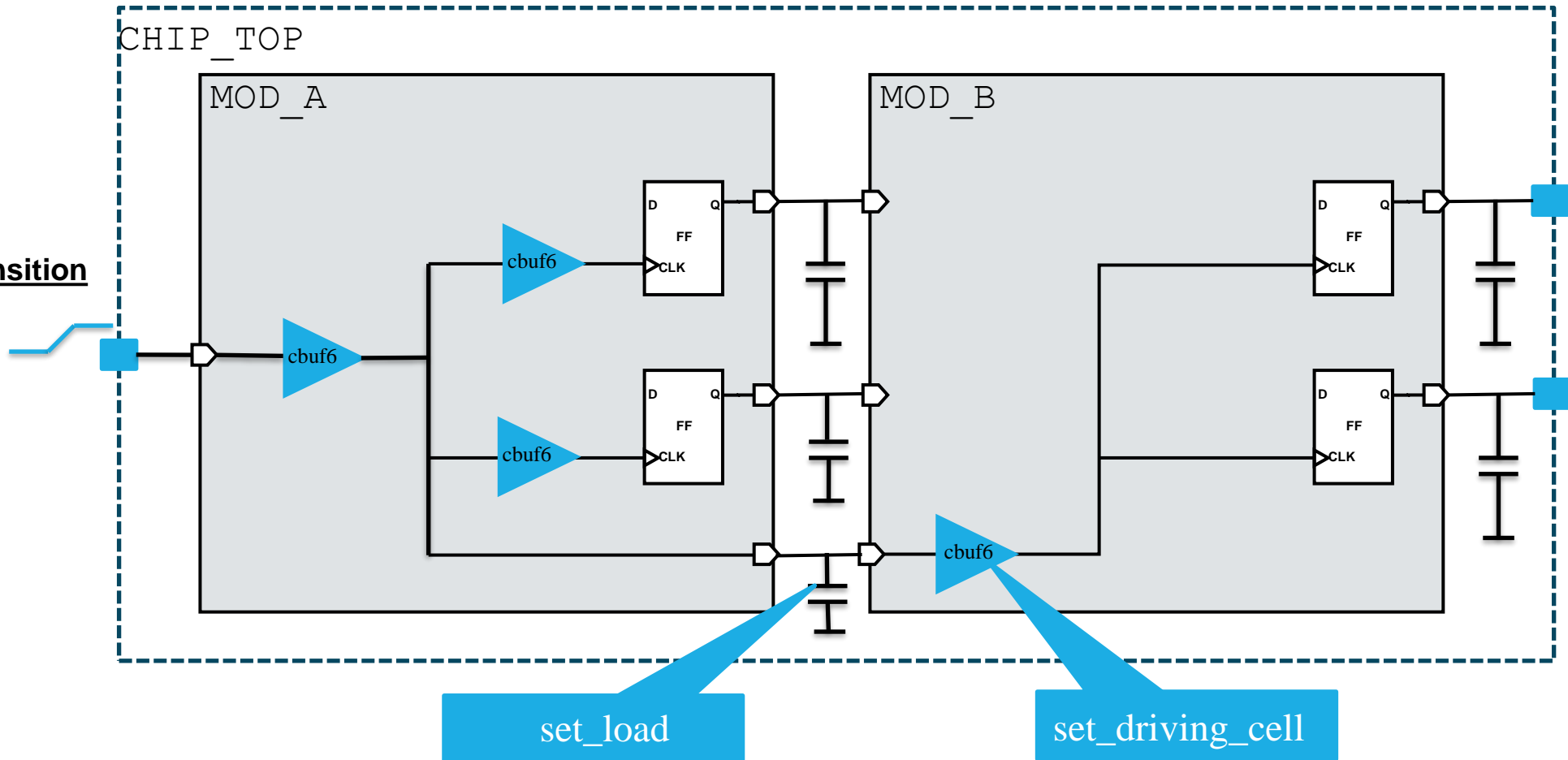
Helps to prevent routing DRC violations as well as SI issues

Are all Clock Drivers and Loads Specified?

Constraints

Ensure that all clock input ports have a slew constraint needed for accurate clock delay calculation during and after CTS

Specify input transition



Defining CTS-Specific DRC Values

- ❑ Max transition and max capacitance design rules can be specified in two ways: Library and SDC
 - ICC II will always use the smallest value
- ❑ You can define your own CTS-specific DRC values:

```
set_max_transition 0.5 -clock_path [all_clocks]  
set_max_capacitance 0.6 -clock_path [all_clocks]
```

- ❑ Design rule constraints can be selectively applied per clock and per scenario (applies to current scenario, by default):

```
set_raax_transition 0.2 -clock_path \  
    -scenarios "SI S4" \  
    [get_clocks SYS_CLK]
```

Remove Skew from Uncertainty

❑ SDC constraints usually include

`set_clock_uncertainty -setup | -hold <number>` applied to each clock

- Models estimated effects of clock skew, jitter, and additional timing margin on setup/hold timing, pre-CTS

❑ After clock tree propagation, to avoid pessimistic timing analysis, remove or reduce uncertainty by the estimated skew:

```
set_clock_uncertainty -scenarios si -setup | -hold <SMALLER_#> CLK1
set_clock_uncertainty -scenarios {s2 s3} -setup | -hold <SMALLER_#> CLK2
# OR
remove_clock_uncertainty [all_clocks] -scenarios [all_scenarios ]
```

Reporting Settings Summary

```
# Report clock tree max_tran/cap/references/... in all clocks+modes:
report_clock_settings
    [-clock CLOCKS]
    [-type type]
        type can be: configurations , routing_rules ,
        references, spacing_rules , all

# Report clock tree target skew/latency constraints:
report_clock_tree_options

# Report explicit balance points (sink/exclude pins), and groups:
report_clock_balance_points
report_clock_balance_groups

# Report non-default routing rules in a more compact way
report_clock_routing_rules
```

- ❑ ICC II builds clock trees for all clocks in all active setup and/or hold scenarios
- ❑ CTS will also perform clock net logical PRC fixing on scenarios enabled for max_transition and/or max_capacitance
- ❑ If you do not want a scenario to be used during CTS:

```
set_scenario_status -active false {s1}
```

Hold Fixing

- ❑ Hold fixing occurs in all scenarios that are enabled for hold:

```
set_scenario_status { s2 } -hold true
```

- ❑ Control what cells are used to fix hold violations:

```
set_lib_cell_purpose -exclude hold [get_lib_cells]  
set_lib_cell_purpose -include hold \  
    [get_lib_cells "*/DELLN*_HVT */NBUFF*_HVT \  
                  */DELLN*_RVT */NBUFF*_RVT"]
```

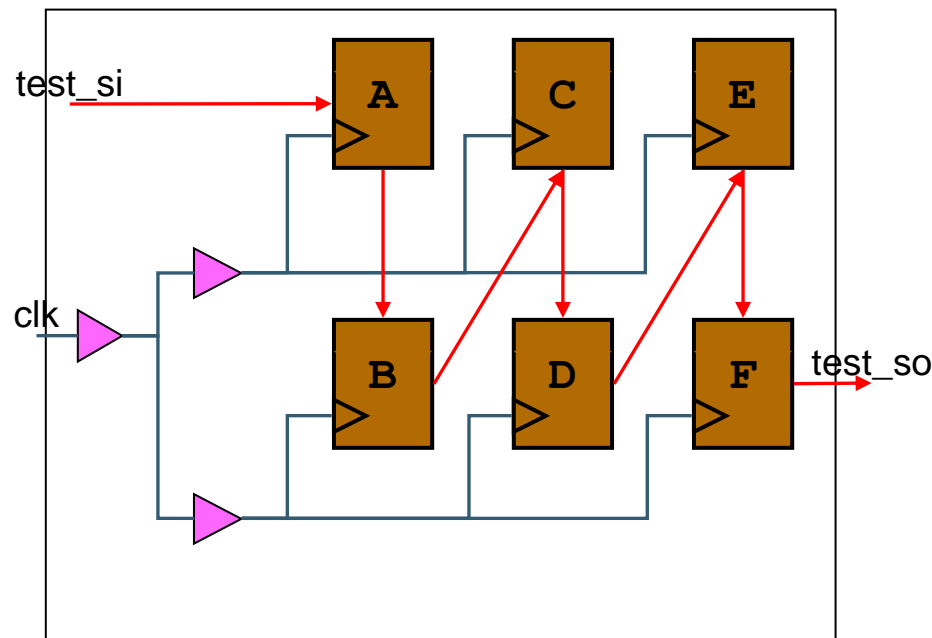
- ❑ Control the effort for hold fixing:

```
set_app_options -list {  
    clock_opt.hold.effort none|low|medium |high}
```

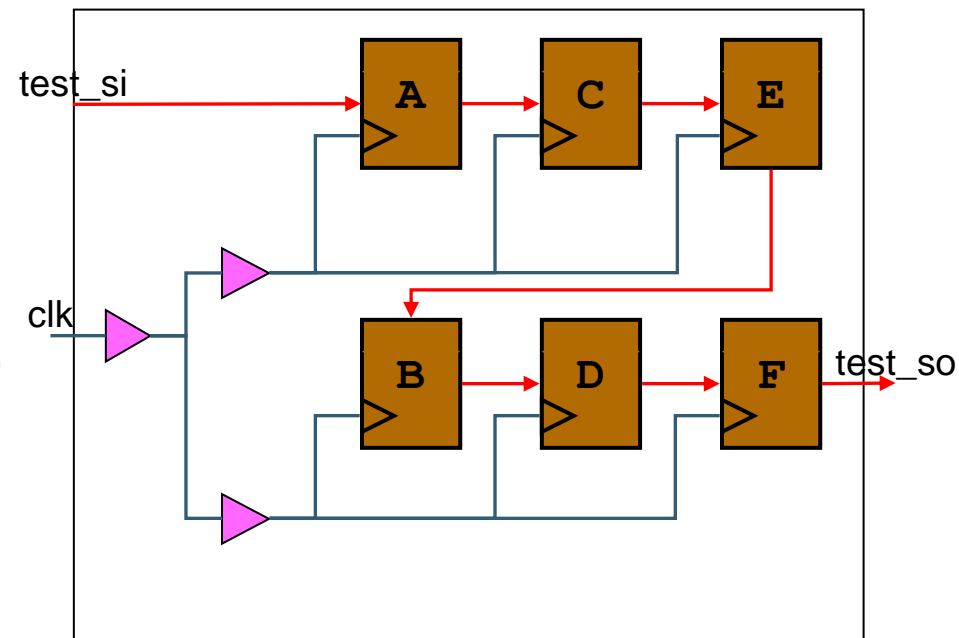
Minimize Hold Time Violations in Scan Paths

- Enable scan chains reordering to minimize branch crossings
- Can reduce hold time violations in the scan chain

```
set_app_options -list opt.dft.clock_aware_scan_reorder true
```

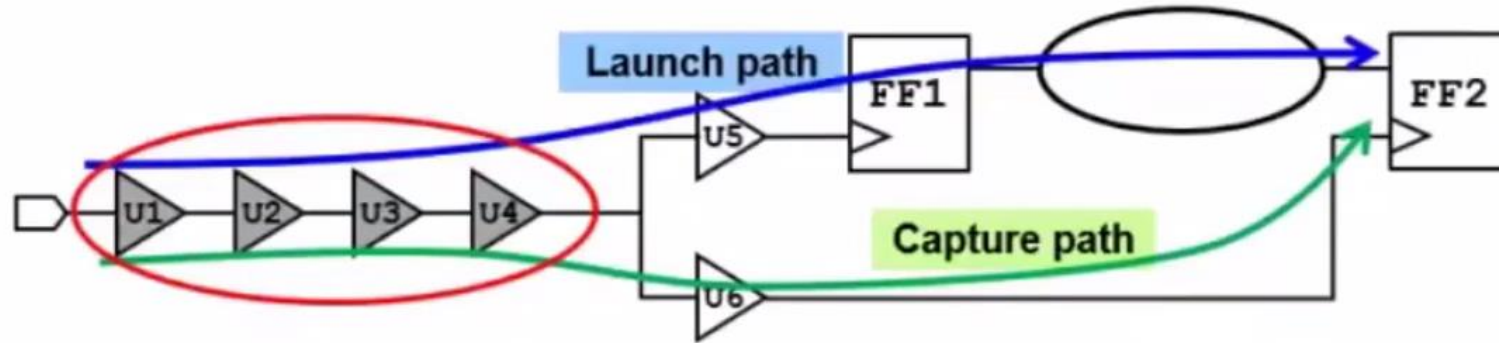


Without clock tree based reordering



With clock tree based reordering

Clock Reconvergence Pessimism



- ❑ To reduce OCV effects, clock trees try to share as many buffers as possible
- ❑ Remove the pessimism from the timing calculation caused by shared clock paths (late/early for launch/capture calculation)

```
set_app_options -name time.remove_clock_reconvergence_pessimism
                 -value true
```

Clock Reconvergence Pessimism Removal

- The pessimism is removed for setup timing by adding the CRP in the capture path, and by subtracting for hold

Example setup timing report: Added CRP in capture path

clock SYS_CLK (rise edge)	5.00	5.00
clock network delay (propagated)	1.76	6.76
clock reconvergence pessimism	0.08	6.84
clock uncertainty	-0.10	6.74
I_BLENDER_0/s4_op1_reg[30]/CLK (SDFFX1_LVT)	0.00	6.74 r
library setup time	-0.26	6.58
data required time		6.58

For a *hold* report this would be negative

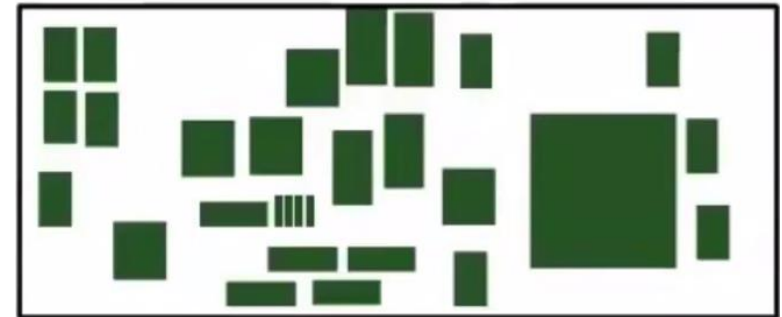
Congestion-Aware Initial CTS

❑ Initial clock tree synthesis (**build_clock**) is not congestion aware!

- Based on virtual routing, by default

❑ On large complex-floorplan designs, this may lead to:

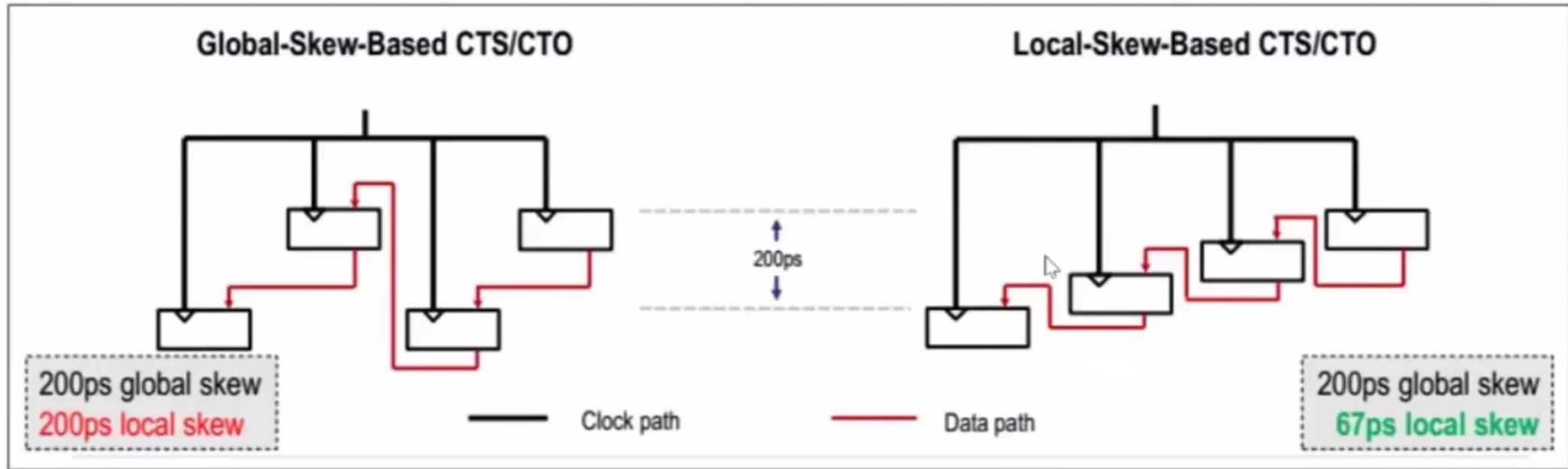
- Pre- vs. post-route clock skew, latency, and logical DRC degradation
- Congestion hotspots
- Route DRCs



❑ Enable global routing for congestion estimation and congestion-aware clock tree construction during the **build_clock** stage:

```
set_app_options -list {cts.compile.enable_global_route true}
```

Local Skew Optimization



- ❑ Performs timing-aware clustering (as opposed to location-based)
- ❑ Optimizes local skew directly for setup and hold timing critical pairs
- ❑ Can relax skew target where possible (if slack is positive) to save clock buffer area

❑ Timing aware clustering:

- Clusters timing-related clock tree nodes together to improve path sharing without degrading latency.

❑ Automatic derivation of target skew:

- Derives target skew for clocks based on timing QoR seen with early estimation of timing QoR. As a safeguard, derived value is limited to within 3% to 10% of clock period. CTS works to meet derived skew instead of trying to achieve “0” skew and thus reducing clock buffer/cell area.
- Local skew optimization ensures that timing-related registers have minimal skew.

❑ Local skew optimization during CTO:

- After (traditional) global skew optimization stage in CTO, it now looks at timing QoR, and optimizes local skew for timing critical pairs to improve timing QoR.
- Makes sure that skew/latency/DRCs across scenarios are not degraded.

Enable Local Skew CTS and CTO

- ❑ When using CCD, local skew CTS and CTO are enabled by default
 - Because timing is taken into account for the initial clock tree, this reduces the work that CCD has to do to further optimize the clock tree to meet timing
- ❑ To use local skew for classic CTS, you need to enable it specifically:

```
set_app_options -list {  
    cts.compile.enable_local_skew true  
    cts.optimize.enable_local_skew true }
```

- ❑ By default, local skew optimization will calculate relaxed skew targets. You might see messages like:

```
Computing global skew target..  
Setting target skew for clock: SYS_2x_CLK as 0.240000
```

- ❑ CTS is doing this in order to save on clock area. If, after CTS, hold timing is degrading more than what you are comfortable with, you might consider turning this feature off using the application option

```
cts.common.enable auto_skew_target_for_local_skew
```

Restricting the Amount of CCD Skewing

- ❑ By default, CCD skews as much as needed to meet the timing
- ❑ You can limit the amount of skewing if needed, for example:

```
set_app_options -name ccd.max_prepone -value 0.2  
set_app_options -name ccd.max_postpone -value 0.4
```

- **ccd.max_prepone**: max latency reduction (advance) allowed to a sink
- **ccd.max_postpone**: max latency increase (delay)
- Setting a limit can impact the timing QoR as this restricts CCD

Skipping Path Groups during CCD

- ❑ You Can configure CCD to Skip the clock pins that belong to particular Path groups during clock latency adjustment
- ❑ Data path optimization will still be performed on these path groups

```
set_app_options -name ccd.skip_path_groups  
                -value { pathgroup1 {scenarioA pathgroup2} }
```

- If you specify a path group name without a scenario name, all path groups with the name specified from all scenarios are skipped
- If you specify a path group name with a scenario name, only the path group in the specified scenario is skipped



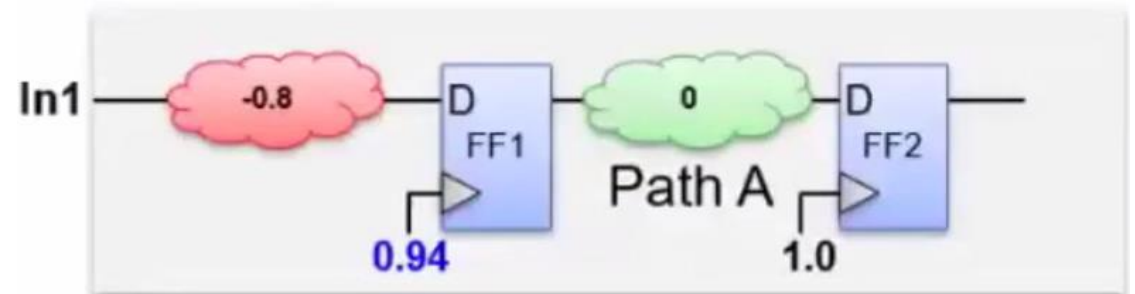
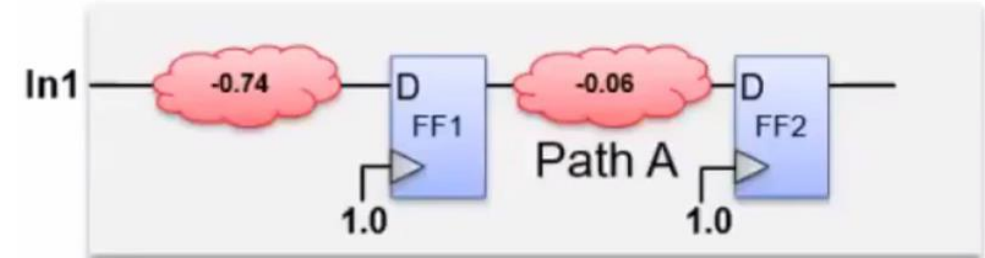
Ignore I/O Timing for Boundary Register Skewing

□ It might be desirable to prevent CCD Skewing based on I/O timing

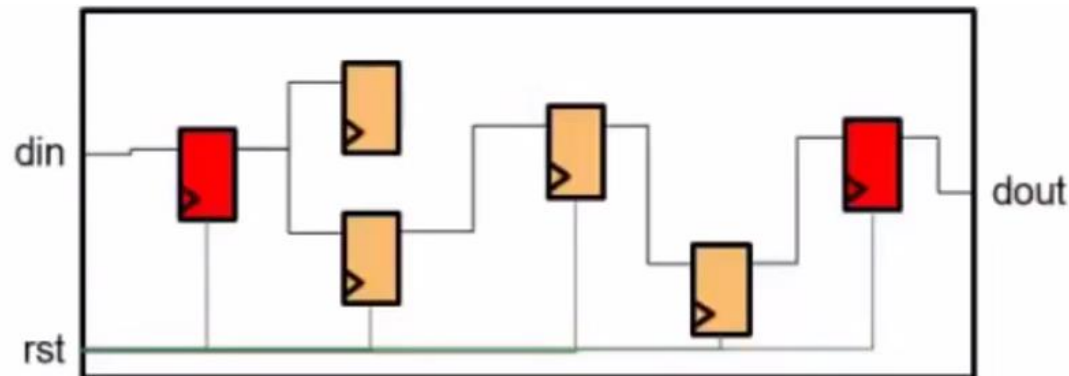
- I/O timing inaccurate or too pessimistic
- To ignore timing on some or all I/Os for CCD:

```
group_path -name MY_IN -from [get_ports In*]  
set_app_options -name ccd.skip_path_groups -value {MY_IN}  
clock_opt
```

- This allows CCD to skew the FF1 boundary register to help Path A's timing



CCD and Boundary Registers (1 of 2)



- ❑ By default, CCD skews all registers (including boundary FFs) for timing
- ❑ To prevent latency adjustment on boundary registers, use:

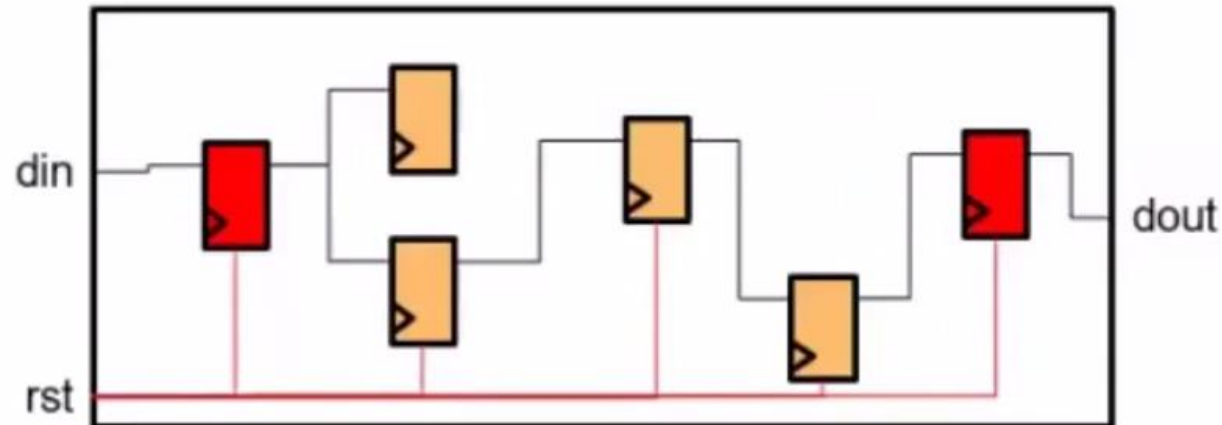
`ccd.optimize_boundary_timing` → set to **false**

Boundary CK pin: en_launch/CK because of port din

Boundary CK pin: ff_1_1/CK because of port din

Optimize_boundary_timing is set to false, 2 registers will not be optimized, 20 registers will be optimized.

CCD and Boundary Registers (2 of 2)



- ❑ Primary Ports are used to differentiate between boundary and internal FFs
- ❑ Ignore ports for boundary identification (reset, scan_en, ...) using:

`ccd.ignore_scan_reset_for_boundary_identification` → set to **false**

- to ignore specific ports for boundary identification, use:

```
set_app_options \
    -name ccd.ignore_ports_for_boundary_identification
    -value {my_en en_A}
```

Hold Criticality during CCD

- ❑ Although CCD setup optimizations are hold-aware, hold violations can be introduced to improve setup timing
 - Usually, data path optimizations are able to address these hold violations
- ❑ You may want to prioritize hold during CCD if
 - Fixing hold in your design is difficult / not possible
 - Your design is area-critical, and the hold-buffer area increase is to be avoided
- ❑ To specify hold criticality during CCD, use:

`ccd.hold_control_effort`

Default: low

- Set to **medium** or **high** to reduce hold degradation
- Use only if hold timing is critical, since setup optimizations may degrade
- Only affects `final_opto`, as well as CCD optimization during `route_opt`

Classic CTS and CCD Execution

- ❑ Clock tree synthesis, clock tree routing, and data path optimization are all executed by:

`Clock_opt`

- CCD is enabled using the application option `clock_opt.flow.enable_ccd`

- ❑ The four stages of `clock_opt` are:



- ❑ You can control which stages are executed with `-from/- to`

- For example, to perform only clock tree synthesis (CTS+CTO):

`Clock_opt -to build_clock`

Classic CTS Flow Stages

Command	What does it do?
<code>clock_opt \</code> <code>-to build_clock</code>	Builds GR clock trees*, and performs inter-clock balancing, for minimum skew
<code>clock_opt \</code> <code>-from route_clock \</code> <code>-to route_clock</code>	Routes the clock nets Updates I/O latencies
<code>clock_opt -from</code> <code>final_opto</code>	Performs data path optimization to meet timing and DRCs

After the `build_clock` phase, the clock trees are global-routed. The global routing occurs during the “optimization” (CTO) part

Classic CTS Flow: clock_opt vs. Atomic

❑ To analyze intermediate results, you can either:

- Run clock opt stages individually using -from/-to options
- Use atomic commands

clock_opt stage	Atomic commands
clock_opt \ -to build_clock	synthesize_clock_trees balance_clock_groups
clock_opt \ -from route_clock \ -to route_clock	route_group -all_clock_nets \ -reuse_existing_global_route true compute_clock_latency
clock_opt -from final_opto	



Post-CTS: Post-route Clock Tree Optimization

❑ Problems:

- Possibility of clock tree QoR degradation (skew, latency, logical DRC) due to miscorrelation between global route(**build_clock**) and detail route(**route_clock**)
- Clock tree QoR can further degrade due to coupling capacitance and crosstalk effects

❑ Solution: Post-route clock tree optimization

- Works on detail-routed clock trees to address these degradations
- Optimizations performed include on-route buffer insertion, buffer sizing, and gate sizing

```
synthesize_clock_trees -postroute
```

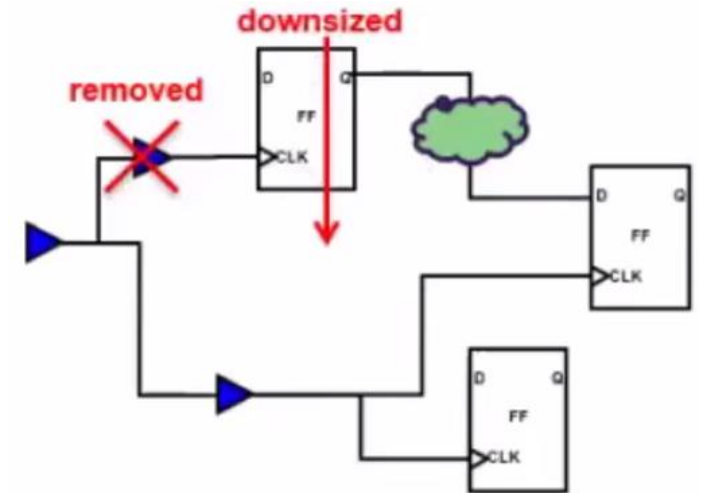
❑ Post-route CTO can also be performed after signal routing

❑ If CCD is enabled, the command will only fix logical DRCs

CCD Power or Area Recovery

□ The CCD engine can recover power or area from the clock network

- Power/area recovered without any timing QoR impact
- Optimization includes repeater removal, clock cell and register sizing.
- Applies to either the classic CTS or CCD flow
- Runs during the **final_opto** stage of **clock_opt**

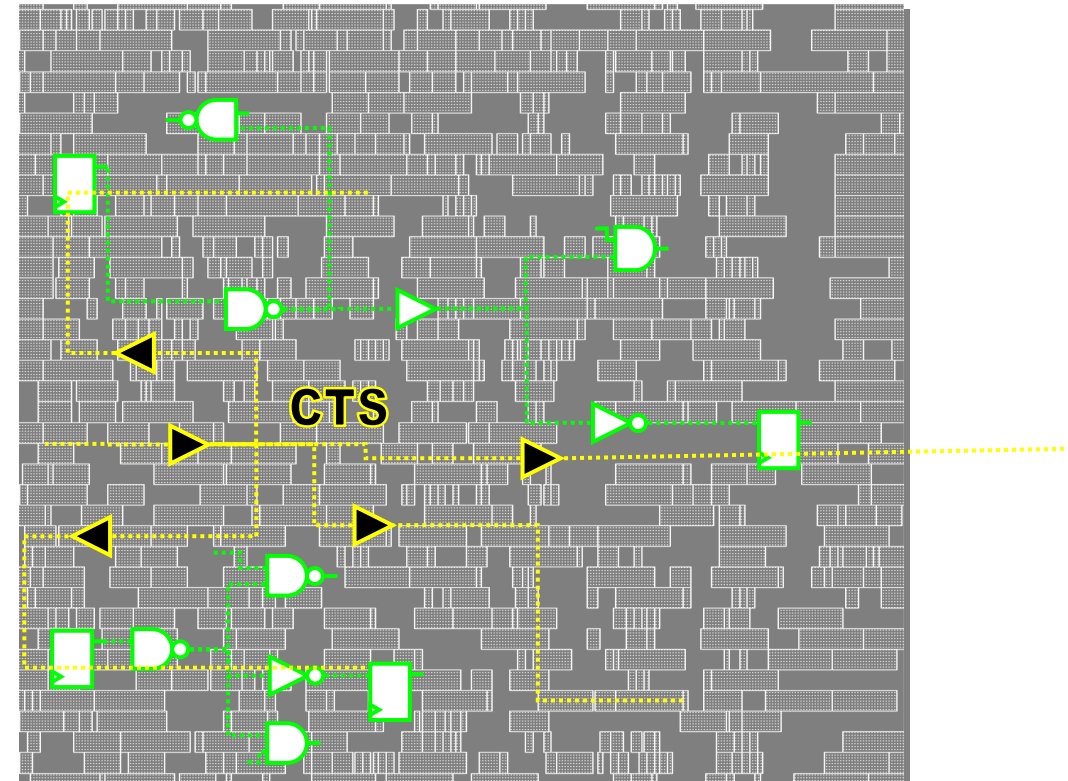


□ Choose between power or area recovery:

- Power: Should use accurate SAIF files for accurate dynamic power calculation
- Area: Use if accurate SAIF files are not available or if area is an optimization goal

Effects of Clock Tree Synthesis

- ❑ Clock buffers added
- ❑ Congestion may increase
- ❑ Non clock cells may have been moved to less ideal locations
- ❑ Can introduce new timing and max tran/cap violations



How do you handle new violations?

Analyzing CTS Results

```
report_clock_qor \  
[-type area|balance_groups|drc_violators| latency|  
local_skew | power | robustness| structure|summary] \  
[-histogram_type latency|transition|level|...] \  
[-modes ...] [-corners ...] ...
```

Details covered in the lab!

- ❑ Reports max global skew (by default), late/early insertion delay, clock DRC violations, number of clock tree references (buffers), structure of the clock tree, robustness, histogram...

Analyzing CTS Results: Clock Timing Report

```
report_clock_timing  
    -type summary|transition |  
    -modes {m1 m2}  
    -corners {c1 c2} ...
```

- ❑ Reports actual, relevant skew, latency, inter-clock latency, etc. for paths that are related
- ❑ Includes effects of early/late timing derates
 - report_clock_qor does not consider derates
- ❑ Example:

```
report_clock_timing -type skew
```

□ CTS browser

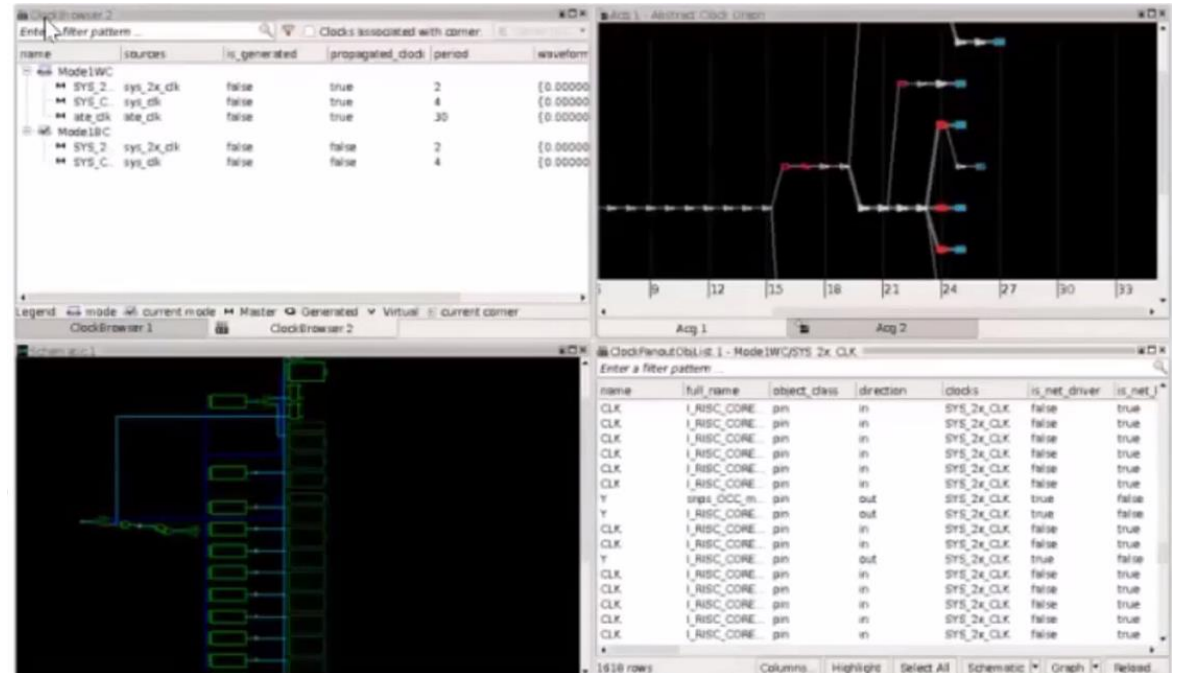
- Identify clock tree object properties and attributes
- Traverse clock tree levels

□ CTS schematic

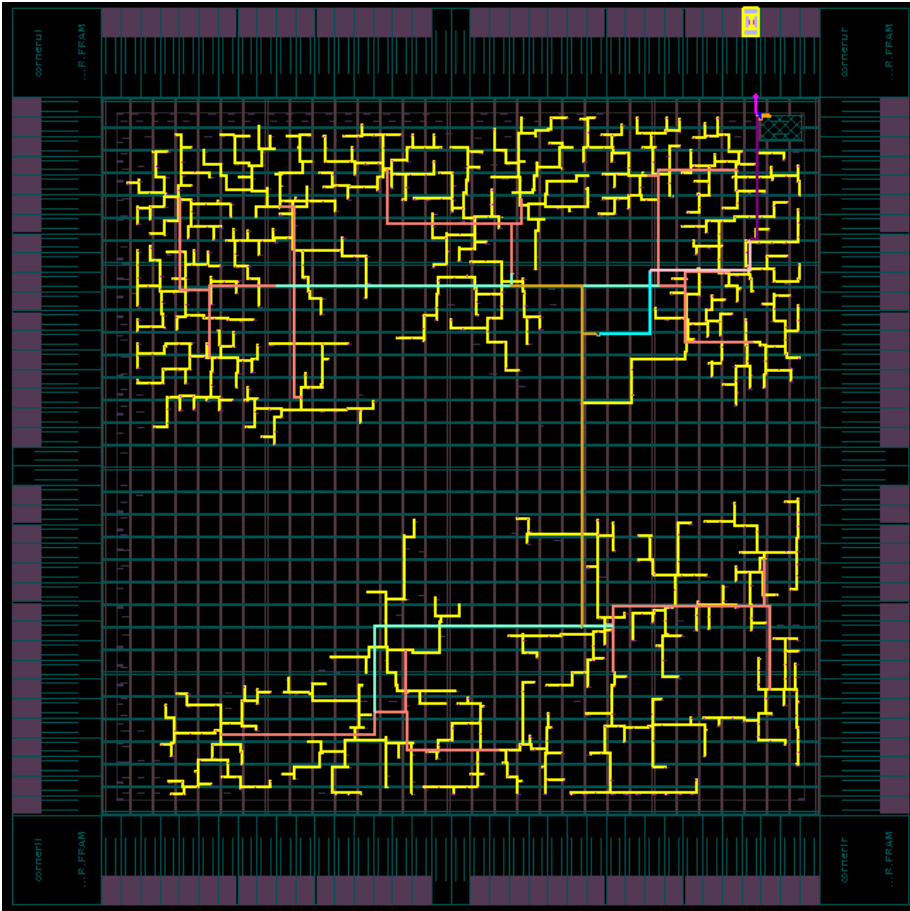
- Trace clock paths
- Cross-probe with layout view

□ Clock tree-levelized graph

□ Clock tree latency graph per corner



Clock Tree Final View



- ❑ **Ron Rutenbar “From Logic to Layout”**
- ❑ **Synopsys University Courseware**
- ❑ **Synopsys Documentation**
- ❑ **IDESA**
- ❑ **Cadence Documentation**

Thank You 😊