Contents

1. **Inputs and outputs of Physical Design**
2. **Checks and care about before starting the Design**
3. **Floorplanning**
4. **Power planning**
5. **Placement**
6. **Clocktree synthesis - CTS**
7. **Clocktree Optimization - CTO**
8. **Routing**
9. **Design Rule check - DRC**
10. **LVS**
11. **Parasitic Extraction**
12. **Power Analysis**
13. **Crosstalk Analysis**
14. **Electron Migration**
15. **Timing Analysis**
16. **Chip Finishing (DFM or DFY)**
17. **Low-power Techniques**
18. **Designs Executed (Projects)**
19. **Multi-Voltage Design**
20. **Physical Design objective type questions and answers**
21. **ASIC Design check list**
22. **Companies interview questions**

1. Inputs and outputs of physical design implementation

The inputs required for any physical design tool is summarized in Table (1) and the outputs generated from the same are listed in Table (2).

Data Input Requirements for Physical Design Tool

Input data Required for Physical Design	
File Format	**File Content**
Technology file (.tf in synopsys format and .techlef in cadence format)	It describes the units, drawing patterns, layers design rules, vias, and parasitics resistance and capacitance of the manufacturing process
Physical Libraries (In general Lef of GDS file for all design elements like macro, std Cell, IO pads etc., and in synopsys format .CEL, .FRAM views for the above)	Contains complete layout information and Abstract model for placement and routing like pin accessibility, blockages etc.,
Timing, Logical and Power Libraries (.lib or LM view -.db for all design elements)	Contains Timing and Power info
TDF file (.tdf or .io)	Contains pad or pin arrangements like order and location of the same. For full chip the instantiation of VDD and VSS pads Power Cut diode etc., (Whichever is not available in verilog netlist)
Constraints (.sdc)	Contain all design related constraints like Area, power, timing
Physical Design Exchange Format –PDEF (optional)	Contains, row, cell placement locations etc.,
Design Exchange Format –DEF (optional)	Contains, row, cell placement locations etc.,

Output data from Physical Design Tool	
File Format	**File Content**
Standard delay format (.sdf)	Timing Details (Except load info)
Parasitic format (.spef, .dspf)	Resistance and Capacitance info of cells and nets
Post routed Netlist (.v) Can be of flattened or hierarchical	Contains connectivity info of all cells
Physical Layout (.gds)	Physical Layout info
Design Excahnge format (.def)	Contains, row, cell, net placement locations etc.,

Libraries in Physical Design

Technology libraries are integral part of the ASIC backend EDA tools. Important two libraries are briefly explained below.

Technology File Libraries

Technology file defines basic characteristic of cell library pertaining to a particular technology node. They are units used in the design, graphical characteristics like colors, stipple patterns, line styles, physical parameters of metal layers, coupling capacitances, capacitance models, dielectric values, device characteristics, design rules. These specifications are divided into technology file sections.

Units for power, voltage, current etc are defined in technology section.

The color section defines primary and display colors that a tool uses to display designs in the library. Stipple pattern are defined in stipple sections. Different layer definitions like its current density, width etc are defined in layer section. Fringe capacitances generated by crossing of interconnects are defined in fringe cap section.

Similarly several other specifications like metal density, design rules that apply to design in library, place and route (P&R) rules, slot rule, resistance model are defined in their respective sections.

Standard Cell Libraries, I/O Cell Libraries, Special Cell Libraries

A standard cell library is a collection of pre designed layout of basic logic gates like inverters, buffers, ANDs, ORs, NANDs etc.

All the cells in the library have same standard height and have varied width. These standard cell libraries are known as ***reference libraries in Astro***.

These reference libraries are technology specific and are generally provided by ASIC vendor like TSMC, Artisan, IBM etc. Standard cell height for 130 TSMC process is 3.65 µM.

In addition to standard cell libraries, reference libraries contain I/O and Power/Ground pad cell libraries. It also contain IP libraries for reusable IP like RAMs, ROMs and other pre-designed, standard, complex blocks.

The TSMC universal I/O libraries include several power/ground cells that supply different voltages to the core, pre-drivers and post drivers. Internal pull-up or pull-down is provided to some cells in I/O libraries.

2. Checks and Care About Before Starting the Design

The goal of the HandOff environment is to provide early checking capabilities to ensure that once the design is routed and has converged in timing on the optimization environments, the SignOff of the design will produce no surprises. Among surprises that are safeguarded by the HandOff environment are:

- Naming issues among the different SignOff environments tools: extraction, DRC, LVS.
- LVS issues with respect to power/ground proper definition.- Bad clock implementation planning.
 In case latency are not budgeted properly, it is common to ends-up re implementing the design from scratch with updated latency/clock definitions.If lucky, it's only a matter of redoing CTS and all the routing phase.
In the worst condition, the design is hacked with ECOs on the clock trees to fix timing using large skews with potential silicon failure.
- Bad optimization through constraints checking. Same consequence as above if not done properly.
- Bad interpretation of High Fanout nets requirements.

Note:
Failure to budget high fan-out nets(HFN) and clocks can cause full placement re-spin because of incorrect pre-cts optimization.

In this flow the following information are obtained by the users and validated:

•Inter clock relation ship of the clocks must be aligned together
•Basic checking of constraints in the Zero WLM
• defines all clock roots
• defines all high fanout nets
• budgets all clock latencies and insertion delays of high fanout nets(HFN). The min insertion delay must be equal to the max insertion delay (known problem).
• if a scan chain description is available, user validate that the budgeted latencies does not cause too large skews on the scan chains
• user validate that all "create_clocks" have an equivalent clock root in , user validate that all high fanout nets are defined
• user export pre-cts clock latencies.
•In Unix, user updates the uncertainty definitions of pre/post cts files.
•It is recommended

Note:

Failure to budget high-fanout nets and clocks can cause full placement re-spin because of incorrect pre-cts optimization.

Multi Voltage design
Definition: A multi voltage design is a hierarchical design with some blocks being supplied with a different voltage than the top level. The flow today only support a single voltage per place and route partitions, but by combining the partitions together, to have indeed multiple configuration.

Check:

As buffers are connected to the periphery of the circuit, it is a good design practice to put them at the top level.
A net with more than one driver is reported.
A driver is an input top port or an output pin.

A top level port of a block cannot be left unconnected

A top level port of a top design pad cell cannot be left unconnected.

A pass output of a cell having a input pad must be connected only with cells of celltype interface. Every unbuffered package (with a ***pass*** output) pad input must be buffered (of celltype ***interface***).

ClockTree Check:

Vth Consistency:
The clock tree must use cells from the allowed Vth list. Today, that list of Vth correspond to all Vths available and there is no limitation set.

The clock tree must be using the same Vth or oxide thickness. The check report only the first instance with a different Vth. This is to ensure different branches of the clock are seeing the same variation with respect to average power supply, temperature, process.

Max Drive:
It is not allowed to have cells which have a too large drive. There is 2 reasons for this. First reason is to limit **electromigration risks**. Second reason is to limit the configuration with **large miller capacitances**.

It is not allowed to have cells that are **too weak**. This is to limit the amount of variations.

It is not allowed to have **delay cells** in the clock tree. Delay cells have very particular layout and there is a risk of large pulses being filtered by the delay cell.

It is not allowed to have a **too long string of cells with a fanout of 1**. If this is the case, it is possible that the CTS engine compensated wire delays with cells delays. In such case, you can activate cross corner signoff to ensure signoff is still valid and ignore the error.

difference in cell depth between two branches of the clocktree
It is not allowed to have a very **long clock tree** branch and a **very short one in the same clock tree**, or between different clock roots. This error is reported for branch belonging to the same clock root. If this is the case, the skew computed during delay calculation or timing analysis is not

guaranteed. If this error is reported between clock roots, you can waive the issue by ensuring that there is no path, or that the full OCV margin is applied to the timing check, or that **deskewing** latches are present.

The clock tree must use cells from the allowed Transitor L list. Today, that list of Transistor L correspond to all Transistor L available and there is no limitation set

sdcCheck:

To check the alignment of the timing constraints and the netlist. The following checks are performed:

Consistency checks between constraints and netlist.
The **get_nets** statement should not be used in constraints, since net names may be modified by the implementation flow.
Clocks and generated clocks are created on **physical pins or ports** in the netlist, not on hierarchical levels.
load/driving_cells on ports
input/output delays on ports

Clock Latencies:

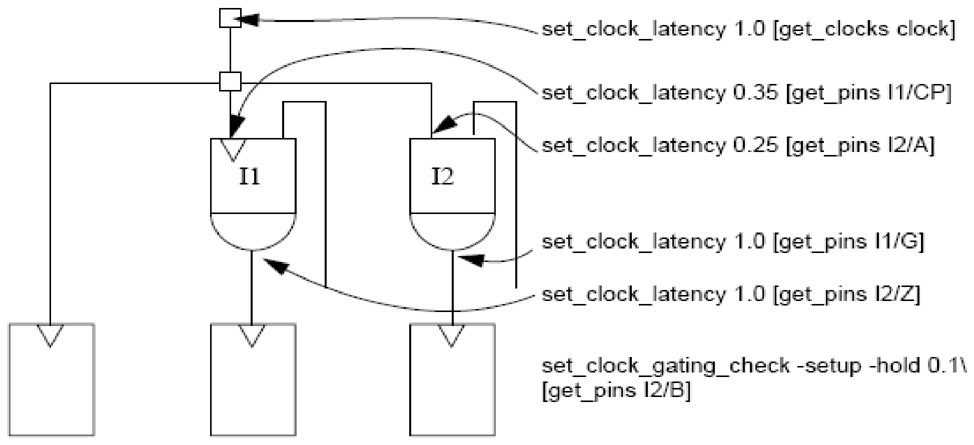
In “post cts mode”, the file will contain updated “set_clock_uncertainty” commands and “set_clock_latency” for the **virtual clocks** eventually used to define the **input delays and output d**elays of the design.

Define **clock uncertainty** to be equal to the **jitter** of the PLL in the **post cts** file. It is usually not necessary to take additional margins in setup with respect to Signoff. For **pre-cts** implementation, add **additional margin** depending on the design to ensure reasonable convergence.

For the modes where **CRPR** is enabled in signoff, define **inter clock uncertainties for HOLD** that will ensure CRPR based signoff will not violates. These holds will be fixed pre-cts.

Define intra clock uncertainties of 0.0 for hold in pre-cts. It is not needed to try to fix holds intra clock holds in pre-cts as most of them will be false. Define clock uncertainty for hold in post cts to be equal to your target uncertainty in Signoff (if no CRPR is used), or to a reasonable number if CRPR is enabled in Signoff. One way to define a reasonable number is to find the amount of violators to fix in the post-cts physopt prior to running the fix.

set_clock_latency must be defined for all clocks and **all gating element**. The enable signals of clock gating components are severely under constrained during any pre-cts optimization if no latency are defined. The figure 2 shows an example of setting clock latency on clock gating components. This assumes that clock gating components are connected directly to CP pins of flip-flop and have a small fanout.This ensure that clock tree synthesis will not put any buffer after the gating component. The creation of this information is not automatic and should be done by the user


set_clock_latency 1.0 [get_clocks clock]
set_clock_latency 0.35 [get_pins I1/CP]
set_clock_latency 0.25 [get_pins I2/A]
I1
I2
set_clock_latency 1.0 [get_pins I1/G]
set_clock_latency 1.0 [get_pins I2/Z]
set_clock_gating_check -setup -hold 0.1\
[get_pins I2/B]

I1: clock gating component with integrated latch

virtual_clock do not necessarily require uncertainty depending on how the input/output delay are defined.

When a block is packaged during Signoff, only the block exceptions are exported because it contains the only information useful to time the block on the top level.

create_clock #create_generated_clock should be in exceptions set_input_delay set_output_delay set_max_delay set_min_delay set_clock_gating_check

For PrimeTime sign-off, constraints should be described in single operating condition mode. This means that -min, -max, -setup, -hold options are not present in PrimeTime constraints. For implementation tools (Astro, Physical compiler) constraints need to be in min/max mode. This means that -min, -max, -setup, -hold options are required on constraints, when applicable.

Validate that no unexpected issues will happen when the design is placed in propagated mode. In this task, the clocks are ideal and correspond to what the Synthesis user has specified. **When we will perform the Signoff timing analysis or the post-cts optimization, propagated clock will be used. It can happen on complex designs that 2 flip-flops that do have a constrained path are not aligned when implementing the clock tree. Because of the miss alignments a large amount of violations may appear after clock tree-synthesis while the pre-cts optimization was clean**. This typically happen with chains of generated clocks that are difficult to trace. The task report to the user the clocks that do have paths together and the clock generation tree. Again if false paths are missing between clocks, the user can update the constraints set based on the information available here. All this information are usually invisible during the synthesis where all the registers are usually assumed to be synchronised together which is not necessarily reasonable from a clock tree implementation stand-point.

set_clock_latency on the virtual clocks to match the clock tree insertion delay in case of a block level implementation. It is possible to leave unchanged values if the estimated latencies is reasonably close to the implemented clock tree insertion delay. In case of a top level implementation, input/output delay might not change at all with respect to the chip internal insertion delay. This is highly dependant on the IO logic involved.

set_clock_uncertainty that must not include margin for clock tree implementation any more (but typically still include PLL jitters for setup, and functional hold margin with respect to substrate noise).

set_clock_latency and set_clock_transition on real clocks should be removed. The flow scripting automatically removes the spurious set_clock_transition, set_clock_latency and put the clocks in propagated mode automatically in Physopt flow.

CTS options:

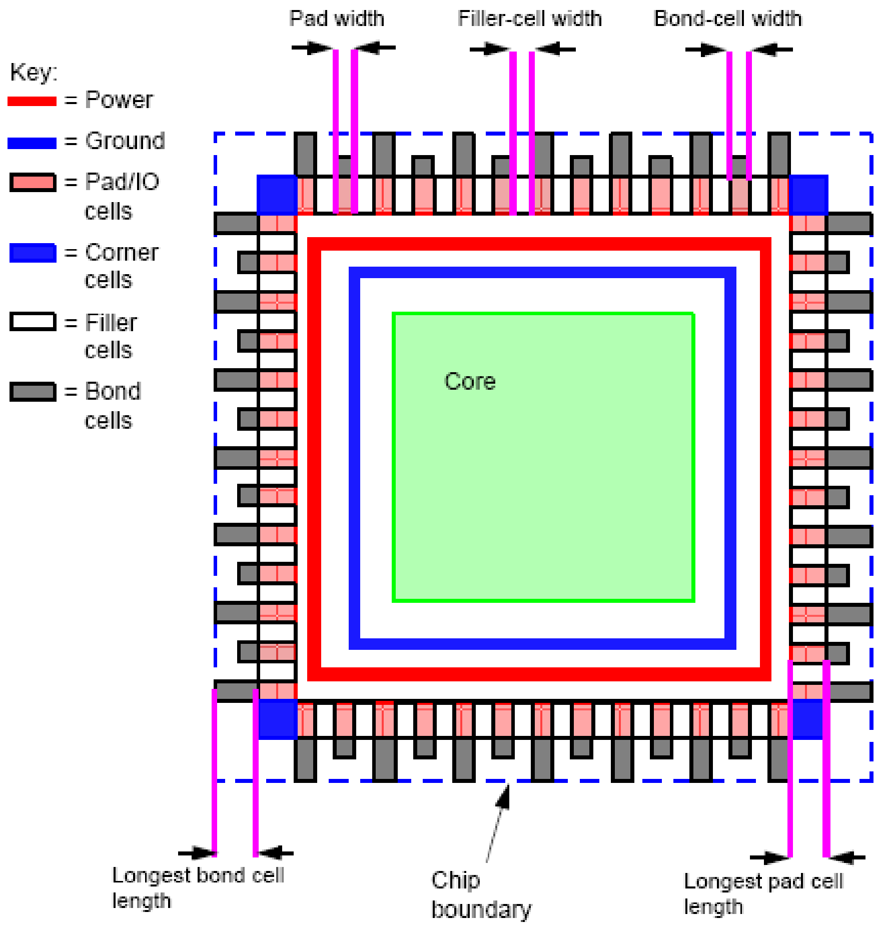
•**Max transition**: maximum slope allowed at leaf pins.
•**Max skew:** maximum skew targeted among leaf pins
•Names of **buffers** allowed for insertion.
•Names of **inverters** allowed for insertion.
•**Routing rule**: specific routing rule used for routing clock tree nets.
•**Routing mode**: select the CTS mode.

3. Floorplanning

1. **What is Pad limited design and core limited design. Is there any difference in approaches to handle these?**
   Pad limited design:
   The Area of pad limits the size of die. No of IO pads may be lager. If die area is a constraint, we can go for staggered IO Pads.

   Core limited design:
   The Area of core limits the size of die. No of IO pads may be lesser. In these designs In line IOs can be used.

2. **How will we decide chip core area?**



$$Core\ Size = \frac{Sandard\ Cell\ Area}{S\tan dared\ Cell\ Utilization} + (Macro\ Area + Hallo)$$

$$DieSize = CoreSize + IOtoCoreClearance + Areaof\ Pad(IncludingIO\ PitchArea) + Areaof\ BondlongestPad$$

I/O-core clearances is the space from the core boundary to the inner side of I/O pads(Design Boundary)

How to arrive at the value of utilization factor and aspect ratio during initial floorplan?

Utilization Percentages:

The Assumption is that the Standard Cells occupies 70 % of Base Layers and the remaining 30 % is utilized for Routing. If the area of macros is more then utilization can be increased accordingly
Chip utilization, flat

$$Chip\ Utilization = \frac{Area\ of\ [Sandard\ Cell + Macro + (Pad, PadFiller, CornerPad)]}{Area\ of\ Chip}$$

Blockages, macros, and pads are combined in the denominator of the effective Utilization.

The effective utilization definition is that all standard cells are placed outside of the blockage areas. This includes buffers, which (for the purposes of computing utilization) are assumed to be placed outside of non-buffer blockage areas.

Best Aspect Ratio:

Consider a **five-layer design** in which layers 1, 3, and 5 are horizontal and layers 2 and 4 are vertical. Usually, *layer 1* is occupied by the standard cell geometries and is *unusable* for routing. Metal *layer 2* often connects to metal layer 1 pins through vias. These *vias* tend to *obstruct* about *20 percent* of the potential *vertical routing* on metal layer 2. If routing pitch is the same on all layers, the ratio between horizontal and vertical layers is approximately ***2 : 1.8***. This means that the available **vertical** routing resource is **less** than the horizontal routing resource, which dictates a **chip aspect ratio** that is **wider than** it is **high**.

Using the ratio of horizontal-to-vertical routing resources, the best aspect ratio is 1.11; therefore, the chip aspect ratio is **rectangular** rather than square and is **wider** than it is high:

$$AspectRatio = \frac{W}{H} = \frac{Horizontal\ Routing\ \mathrm{Re}sources}{Vertical\ Routing\ \mathrm{Re}sources}$$

Next, consider a **four-layer design**. metal *layer 1* is not *usable* for routing, and metal *layer 2* is 20 percent *obstructed* by vias connecting layer 1 andlayer 2. Layer 3 is horizontal and fully available, and layer 4 is vertical and fully available. For this case, there is 80 percent more vertical routing resource than there is horizontal resource. Therefore, the ratio of horizontal to vertical routing resource is **0.56**, and the **vertical** dimension of this chip is **larger** than its horizontal dimension.

aspect ratio = W/H = 1/1.8 = .56

The assumptions is that metal layer 1 is unavailable for routing and that metal layer 2 is 20 percent obstructed by vias.

4. What is an HALO? How is it different from the blockage?

Block halos can be specified for hard macros, black boxes, or committed partitions. When you add a halo to a block, it becomes part of the blocks properties. If you move the block, the halo moves with it.
Blockages can be specified for nay part of the design. If we move the a block, the blockage will not.

4. What is the best place to put an HARD macro if it is a power hungry device and dissipates lot of heat?

By placing Power hungry macros near the boundary of core, the required amount of current can be supplied, hence avoid dropped voltage supplyied to std cells and avoid Electron migration.

4. Is there any thumb rule to be followed for deciding the assignment of different layers?

The top most layers have to be may be power

4. How will you do floor planning for multi supply design?

- Create Voltage Regions
-

4. What is the minimum clearance (placement and routing) around macro?

-That will vary between macros, we need to check the Macro data sheet and decide.

4. How is floorplanning done in hierarchical flow?

- Partitioning
-

4. How to decide on the shape of the floorplan for soft macro. Explanation with case study is helpful.

4. How to Decide pin/pad location?

To meet

- Top level requirements (If it is block) Timing
- Timing and congestion
- Board design Requirement or standard
- Area and Power

4. How much utilization is used in the design?

There is no hard and fast rule, even though if the following values maintained then the design can be closed without much congesstion

Floor Plan - 70 %	Placement - 75 %
CTS - 80 %	Routing - 85 %

During GDSii Generation – 100 %

4. What is the difference between standard cells and IO cells? Is there any difference in their operating voltages? If so why is it.

- Std Cells are logical cells. But the IO cells interact between Core and Outside world.
- IO cells contains some protection circuits like short circuit, over voltage.
- There will be difference between Core operating Voltage and IO operating voltage. That depends on technology library used. For 130 nm generic library the Core voltage is 1.2 v and IO voltage is 2.5/3.3.

4. What is the significance of simultaneous switching output (SSO) file?

SSO: The abbreviation of "Simultaneously Switching Outputs", which means that a certain number of I/O buffers switching at the same time with the same direction (H ! L, HZ ! L or L ! H, LZ ! H). This simultaneous switching will cause noise on the power/ground lines because of the large di/dt value and the parasitic inductance of the bonding wire on the I/O power/ground cells.

SSN: The noise produced by simultaneously switching output buffers. It will change the voltage levels of power/ground nodes and is so-called "Ground Bounce Effect". This effect is tested at the device output by keeping one stable output at low "0" or high "1", while all other outputs of the device switch simultaneously. The noise occurred at the stable output node is called "Quiet Output Switching" (QOS). If the input low voltage is defined as Vil, the QOS of "Vil" is taken to be the maximum noise that the system can endure.

DI: The maximum copies of specific I/O cell switching from high to low simultaneously without making the voltage on the quiet output "0" higher than "Vil" when single ground cell is applied. We take the QOS of "Vil" as criterion in defining DI because "1" has more noise margin than "0". For example, in LVTTL specification, the margin of "Vih" (2.0V) to VD33 (3.3V) is 1.3V in typical corner, which is higher than the margin of "Vil" (0.8V) to ground (0V). DF: "Drive Factor" is the amount of how the specific output buffer contributes to the SSN on the power/ground rail. The DF value of an output buffer is proportional to dI/dt, the derivative of the current on the output buffer. We can obtain DF as:
DF = 1 / DI

4. Explain Floor planning, from scratch to end?

Floorplanning is the process of:
• Positioning blocks on the die or within another block, thereby defining routing areas between them.
• Creating and developing a physical model of the design in the form of an initial optimized layout
Because floorplanning significantly affects circuit timing and performance, especially for complex hierarchical designs, the quality of your floorplan directly affects the quality of your final design

- Calculation of Core, Die size and Aspect Ratio.
- 70% of the core utilization is provided
- Aspect ratio is kept at 1
- Initializing the Core
- Rows are flipped, double backed and made channel less
-
- If we have multi height cells in the reference library separate placement rows have to be provided for two different unit tiles.
- Creating I/O Rings
- Creating the Pad Ring for the Chip
- Creating I/O Pin Rings for Blocks
- Preplacing Macros and Standard Cells using Data Flow diagram and by fly-line analysis.
- Prerouting Buses
- The core area is divided into two separate unit tile section providing larger area for Hvt unit tile as shown in the Figure 3.
-
- Creating Placement Blockages

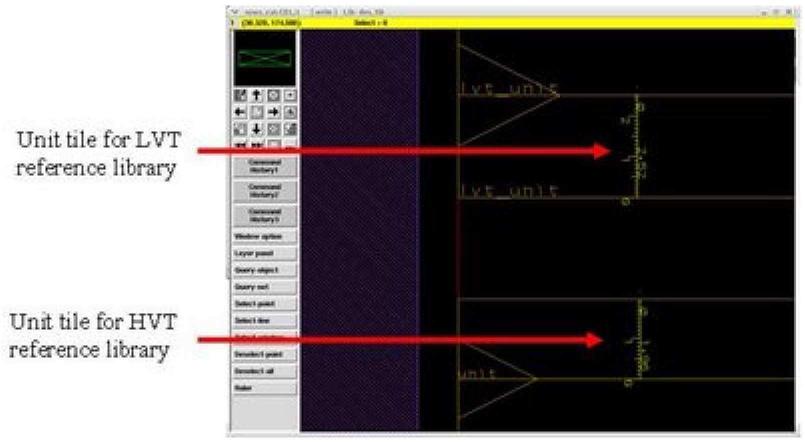

Unit tile for LVT reference library
Unit tile for HVT reference library
lvt_unit
lvt_unit
unit

Figure -3. Different unit tile placement

First as per the default floor planning flow rows are constructed with unit tile. Later rows are deleted from the part of the core area and new rows are inserted with the tile "lvt_unit". Improper allotment of area can give rise to congestion. Some iteration of trial and error experiments were conducted to find best suitable area for two different unit tiles. The "unit" tile covers 44.36% of core area while "lvt_unit" 65.53% of the core area. PR summary report of the design after the floor planning stage is provided below.

PR Summary:

Number of Module Cells: 70449

Number of Pins: 368936

Number of IO Pins: 298

Number of Nets: 70858

Average Pins Per Net (Signal): 3.20281

Chip Utilization:

Total Standard Cell Area: 559367.77

Core Size: width 949.76, height 947.80; area 900182.53

Chip Size: width 999.76, height 998.64; area 998400.33

Cell/Core Ratio: 62.1394%

Cell/Chip Ratio: 56.0264%

Number of Cell Rows: 392

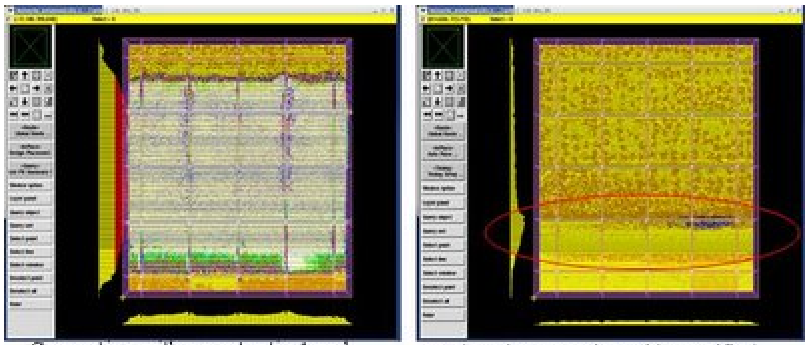
Placement Issues with Different Tile Rows

Legal placement of the standard cells is automatically taken care by Astro tool as two separate placement area is defined for multi heighten cells. Corresponding tile utilization summary is provided below.

PR Summary:

```
[Tile Utilization]

================================================================

unit 257792 114353 44.36%

lvt_unit 1071872 702425 65.53%

================================================================
```

But this method of placement generates unacceptable congestion around the junction area of two separate unit tile sections. The congestion map is shown in Figure 4.



Congestion with aspect ratio 1 and core utilization of 70%

Reduced Congestion with specified core height and width set to 950 um

Figure 4. Congestion

There are two congestion maps. One is related to the floor planning with aspect ratio 1 and core utilization of 70%. This shows horizontal congestion over the limited value of one all over the core area meaning that design can't be routed at all. Hence core area has to be increased by specifying height and width. The other congestion map is generated with the floor plan wherein core area is set to 950 µm. Here we can observe although congestion has reduced over the core area it is still a concern over the area wherein two different unit tiles merge as marked by the circle. But design can be routable and can be carried to next stages of place and route flow provided timing is met in subsequent implementation steps.

Tighter timing constraints and more interrelated connections of standard cells around the junction area of different unit tiles have lead to more congestion. It is observed that increasing the area isn't a solution to congestion. In addition to congestion, situation verses with the timing optimization effort by the tool. Timing target is not able to meet. Optimization process inserts several buffers around the junction area and some of them are placed illegally due to the lack of placement area.

Corresponding timing summary is provided below:

```
Timing/Optimization Information:

[TIMING]

Setup Hold Num Num

Type Slack Num Total Target Slack Num Trans MaxCap Time

==========================================================

A.PRE -3.491 3293 -3353.9 0.100 10000.000 0 8461 426 00:02:26

A.IPO -0.487 928 -271.5 0.100 10000.000 0 1301 29 00:01:02

A.IPO -0.454 1383 -312.8 0.100 10000.000 0 1765 36 00:01:57

A.PPO -1.405 1607 -590.9 0.100 10000.000 0 2325 32 00:00:58

A.SETUP -1.405 1517 -466.4 0.100 -0.168 6550 2221 31 00:04:10

==========================================================
```

Since the timing is not possible to meet design has to be abandoned from subsequent steps. Hence in a multi vt design flow cell library with multi heights are not preferred.

Floor Planning

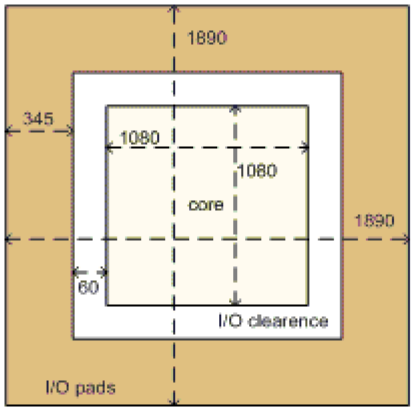
Floor plan determines the size of the design cell (or die), creates the boundary and core area, and creates wire tracks for placement of standard cells. It is also a process of positioning blocks or macros on the die.

Floor planning control parameters like aspect ratio, core utilization are defined as follows:

Aspect Ratio= Horizontal Routing Resources / Vertical Routing Resources

Core Utilization= Standard Cell Area / (Row Area + Channel Area)

Total 4 metal layers are available for routing in used version of Astro. M0 and M3 are horizontal and M2 and M4 are vertical layers. Hence aspect ratio for SAMM is 1. **Total number of cells =1645**; **total number of nets=1837** and **number of ports (excluding 16 power pads) = 60**. The figure depicting floor plan-die size (µm) of SAMM is shown beside.



Top Design Format (TDF) files provide Astro with special instructions for planning, placing, and routing the design. TDF files generally include pin and port information. Astro particularly uses the I/O definitions from the TDF file in the starting phase of the design flow. [1]. Corner cells are simply dummy cells which have ground and power layers. The TDF file used for SAMM is given below. The SAMM IC has total 80 I/O pads out of which 4 are dummy pads. Each side of the chip has 20 pads including 2 sets of power pads. Number of power pads required for SAMM is calculated in power planning section. Design is **pad limited** (pad area is more than cell area) and **inline bonding** (same I/O pad height) is used.

How do you place macros in a full chip design?

- First check flylines i.e. check net connections from macro to macro and macro to standard cells.
- If there is more connection from macro to macro place those macros nearer to each other preferably nearer to core boundaries.
- If input pin is connected to macro better to place nearer to that pin or pad.
- If macro has more connection to standard cells spread the macros inside core.
- Avoid crisscross placement of macros.
- Use soft or hard blockages to guide placement engine.

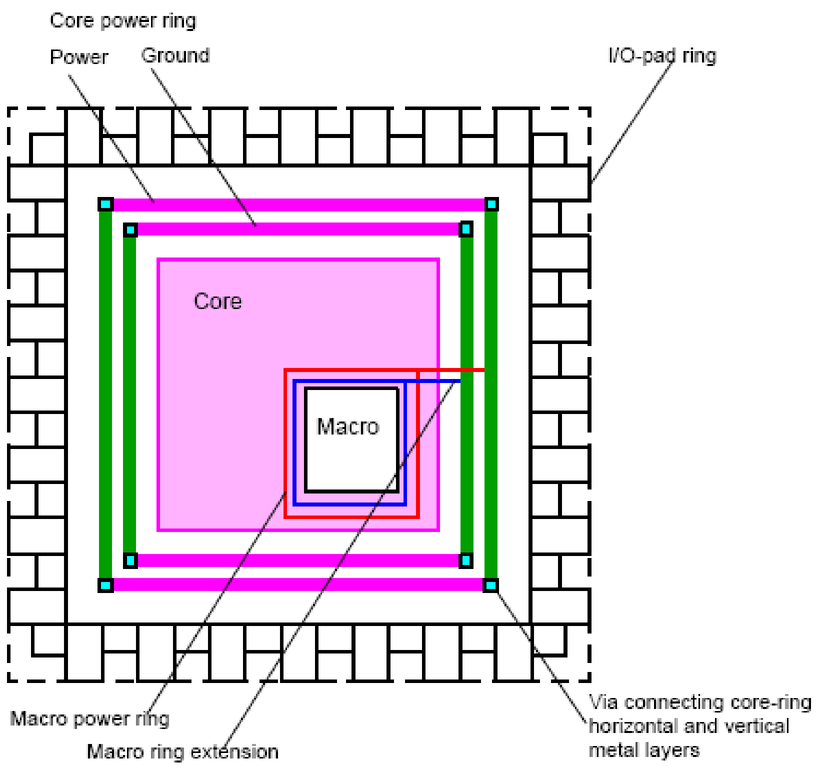
Differentiate between a Hierarchical Design and flat design?

- Hierarchial design has blocks, subblocks in an hierarchy; Flattened design has no subblocks and it has only leaf cells.
- Hierarchical design takes more run time; Flattened design takes less run time.

4. **What are constraints you consider for floor planning of Standard Cells?**
4. **What is SITE? How to specify IO constraints?**
4. **How do we calculate the die size from cell count of our design?**
4. **How different is the floor-plan approach if we go for the flip-chip approach?**
4.

4. PowerPlanning

Typical Power Structures—Power Rings



1. How to do power planning?

Total Core Power	= 1.5w
Worst case core Voltage	= 1.2V

Current drawn by core = (Total Core Power / Worst Case Voltage)

Width Of the Core (Wcore)	= 0.005 m
Height Of the Core (Hcore)	= 0.007 m
Sheet Resiatance of Metal6	= 0.043 ohm
Sheet Resiatance of Metal7	= 0.043 ohm

Metal Layers Used For Core Ring and Strap Metal-7 and Metal-6.

Current Density of Metal-7	= 0.002 A/um
Current Density of Metal-6	= 0.002 A/um

I/O VDD Pad being used	= PVDD2DGZ
I/O VSS Pad being used	= PVSS2DGZ

Core VDD Pad being used	= PVSS1DGZ
Current handling ability of PVDD1DGZ	= 0.045 A
Current handling ability of PVSS1DGZ	= 0.04A

Thumb rule Ratio to be used for IO Power/Gnd calculation = 40 / 40

Thump Rule is: 1Power/Gnd pad for every 4 OR 6 IO pads.We have followed as per Old Nile Design IO Power/Gnd pad.ie We would have taken 1 Power/Gnd pad for every 8 IO pads

Number of sides = 4
Core Power Pad for each side of the chip = (Total Core Power / (Number of sides * Worst case Core Voltage * Max.I of Powerpad) = 6.94

Total Core Power Pad = (Core Power pad for each side of the chip * Number of sides) = 28
Total Core Power Pad = 35

Core Ground Pad for each side of the chip = (Total Core Power / (Number of sides * Worst case Core Voltage * Max.I of Ground pad) = 7.81
Total Core Ground Pad = (Core Ground pad for each side of the chip * Number of sides) = 31
Total Core Ground Pad = 42

Core Ring Width Calculation
Core ring width for Metal6 = (Current Drawn by core / (2 * Jmetal6 * Core Power pad for each side of the chip)) = 45 um
Note: Current into Core ring Splits into two branches. So we put multiply by 2 in the denominator part

Core ring width for Metal7 = (Current Drawn by core / (2 * Jmetal7 * Core Power pad for each side of the chip))
Core ring width = D41 = 45 um
Mesh Width Calculation
Itop = Ibottom = (Icore*(Wcore / (Wcore+Hcore))) / 2 = 0.2604167 A
Iright = Ileft = (Icore * (Hcore / (Wcore + Hcore))) / 2 = 0.3645833 A
Wmesh-vertical = (Wmesh-top = Wmesh-bottom) = (Itop / Jmetal6) = 130.20833 um
Wmesh-horizontal = (Wmesh-left = Wmesh-right) = (Ileft / Jmetal7) = 188.90328 um

Mesh Count = 60
Each Mesh Width for vertical Direction = (Wmesh-vertical / Mesh Count) = 2.2 um
Note :e the Mesh Count is 60. Total mesh width is bifurcated into 60 and we got the result is 4.5um. So Consider Each mesh width is 4.5um
Each Mesh Width for Horizontal Direction = (Wmesh-horizontal / Mesh Count) = 3.1 um

EM RULE CALCULATION
Wmesh-vertical (Max.allowable) = (Icore / Jmetal6) = 647.66839 um
Wmesh-horizontal (Max.allowable) = (Icore / Jmetal5) = 647.66839 um

IR Drop Calculation
Lmesh < ((0.1* VDD) / (Rsh *Jmetal4)) = 1395.3 um
Note : Mesh Length is
Derived from R= (Rsh*L /A)

Rmesh = ((Rsh * (Lmesh / 2)) / Wmesh) = 0.230400 ohm
Resistance of the Mesh is

Note : from R= (Rsh*L /A)

Vdrop = Imesh * Rmesh < 5%VDD= FALSE

Note : TSMC Recommedation, if the IR drop Problem increased beyond 5% vdd inside the chip then we should increase the Mesh width or increase the power pad in the corresponding location.
If the Result is False We ought to increase the Mesh Width.
Wmesh-vertical(Max.allowable) > ((Itop *Rsh *Hcore) / (0.1 * VDD)) = 0.000653 m

Note : If Mesh Length is More than 2946um, increase the Mesh Width in Vertical Direction if IR Problem Dominates.If the IR drop Problem Dominates more than 5%VDD drop inside the chip, then we can increase the Mesh width upto 647um.
Wmesh-horizontal(Max.allowable) > ((Ileft * Rsh * Wcore) / (0.1 * VDD)) = 0.0006532 m

Note : If Mesh Length is More than 2946um, increase the Mesh Width in Horizontal Direction if IR Problem Dominates.If the IR drop Problem Dominates more than 5%VDD drop inside the chip,then we can increase the Mesh width upto 647um

Mesh Width =D64
Mesh Count = 60
Each Mesh Width = (Mesh Width / Mesh Count) = 0.00001089 m

Note : We take the Mesh Count is 60.Total mesh width is bifurcated into 60 and we got the result is 11um.So Consider Each mesh width is 11um if IR problem Dominates

Power Planning

There are two types of power planning and management. They are **core cell power management** and **I/O cell power management**. In **core cell power management** VDD and VSS power rings are formed around the core and macro. In addition to this straps and trunks are created for macros as per the power requirement. In I**/O cell power management**, power rings are formed for I/O cells and trunks are constructed between core power ring and power pads. Top to bottom approach is used for the power analysis of flatten design while bottom up approach is suitable for macros.

The power information can be obtained from the front end design. The synthesis tool reports static power information. Dynamic power can be calculated using **Value Change Dump (VCD)** or **Switching Activity Interchange Format (SAIF)** file in conjunction with RTL description and test bench. Exhaustive test coverage is required for efficient calculation of peak power. This methodology is depicted in Figure (1).

For the hierarchical design budgeting has to be carried out in front end. Power is calculated from each block of the design. Astro works on flattened netlist. Hence here top to bottom approach can be used. JupiterXT can work on hierarchical designs. Hence bottom up approach for power analysis can be used with JupiterXT. IR drops are not found in floor planning stage. In placement stage rails are get connected with power rings, straps, trunks. Now IR drops comes into picture and improper design of power can lead to large IR drops and core may not get sufficient power.

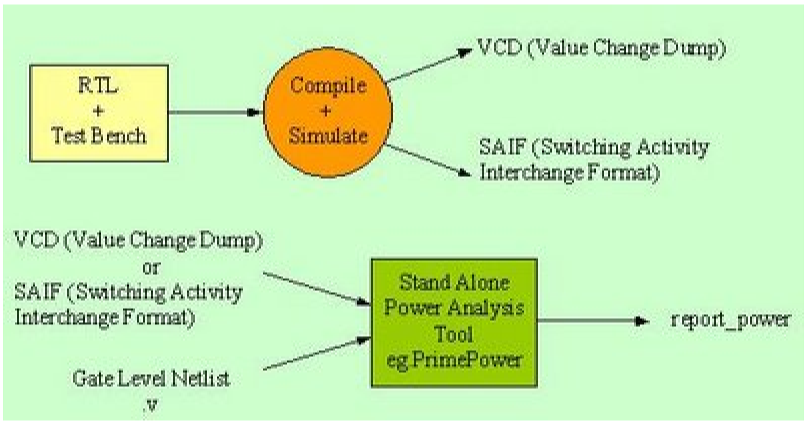

RTL
+
Test Bench
Compile
+
Simulate
VCD (Value Change Dump)
SAIF (Switching Activity Interchange Format)
VCD (Value Change Dump)
or
SAIF (Switching Activity Interchange Format)
Gate Level Netlist
.v
Stand Alone Power Analysis Tool eg PrimePower
report_power

Figure (1) Power Planning methodology

Below are the calculations for flattened design of the SAMM. Only static power reported by the Synthesis tool (Design Compiler) is used instead of dynamic power.

- **The number of the core power pad required for each side of the chip**

= total core power / [number of side*core voltage*maximum allowable current for a I/O pad]

= 236.2068mW/ [4 * 1.08 V * 24mA] (Considering design SAMM)

= 2.278 =~ 2

Therefore for each side of the chip 2 power pads (2 VDD and 2 VSS) are added.

- **Total dynamic core current (mA)**

= total dynamic core power / core voltage

⇨ 236.2068mW / 1.08V = 218.71 mA
⇨

- **Core PG ring width**

= (Total dynamic core current)/ (No. of sides * maximum current density of the metal layer used (Jmax) for PG ring)
=218.71 mA/(4*49.5 mA/µm) =~1.1 µm
~2 µm

- **Pad to core trunk width (µm)**

= total dynamic core current / number of sides * J_{max} where Jmax is the maximum current density of metal layer used

= 218.71 mA / [4 * 49.5 mA/µm]

= 1.104596 μm

Hence pad to trunk width is kept as 2μm.

Using below mentioned equations we can calculate vertical and horizontal strap width and required number of straps for each macro.

- Block current:

$I_{block} = P_{block} / V_{ddcore}$

- Current supply from each side of the block:

$I_{top} = I_{bottom} = \{ I_{block} * [W_{block} / (W_{block} + H_{block})] \}/2$

$I_{left} = I_{right} = \{ I_{block} * [H_{block} / (W_{block} + H_{block})] \}/2$

- Power strap width based on EM:

$W_{strap\_vertical} = I_{top} / J_{metal}$

$W_{strap\_horizontal} = I_{left} / J_{metal}$

- Power strap width based on IR:

$W_{strap\_vertical} >= [I_{top} * R_{oe} * H_{block}] / 0.1 * VDD$

$W_{strap\_horizontal} >= [I_{left} * R_{oe} * W_{block}] / 0.1 * VDD$

- Refresh width:

$W_{refresh\_vertical} = 3 * \text{routing pitch} + \text{minimum width of metal (M4)}$

$W_{refresh\_horizontal} = 3 * \text{routing pitch} + \text{minimum width of metal (M3)}$

- Refresh number

$N_{refresh\_vertical} = \max (W_{strap\_vertical}) / W_{refresh\_vertical}$

$N_{refresh\_horizontal} = \max (W_{strap\_horizontal}) / W_{refresh\_horizontal}$

- Refresh spacing

$S_{refresh\_vertical} = Wblock / N_{refresh\_vertical}$

$S_{refresh\_horizontal} = Hblock / N_{refresh\_horizontal}$

2. Is there any checklist to be received from the front end related to switching activity of any nets to be taken care of at the floorplanning stage?

Yes. The Switching activities of Macros will be available in checklist, it contains the power consumption of each macro at different frequencies are also available.

3. What is power trunk ?

Power trunk is the piece of metal connects the IO pad and Core ring.

4. How to handle hotspot in an chip?

Increasing the number of power straps or increasing the width of power strap will help us to reduce hot spot created by voltage drop and to maintain the voltage drop less than 10 %.

5. What is power gating?

Power gating is one of power reduction technique. This helps by shutting down the particular area of chip from utilizing power.

6. Whether macro power ring is mandatory or optional?

For hierarchical design the macro power ring is mandatory. For flat design the macro power ring is optional.

7. While putting the mesh what are the problems are faced in the Design?

8. The VDD and VSS for Macro-1 is tapped from another macro power strap instead of core power strap.

If you have both IR drop and congestion how will you fix it?

a. -Spread macros
b. -Spread standard cells
c. -Increase strap width
d. -Increase number of straps
e. -Use proper blockage

Is increasing power line width and providing more number of straps are the only solution to IR drop?

f. -Spread macros
g. -Spread standard cells
h. -Use proper blockage

5. what is tie-high and tie-low cells and where it is used

Tie-high and Tie-Low cells are used to connect the gate of the transistor to either power or ground. In deep sub micron processes, if the gate is connected to power/ground the transistor might be turned on/off due to power or ground bounce. The suggestion from foundry is to use tie cells for this purpose. These cells are part of standard-cell library. The cells which require Vdd, comes and connect to Tie high...(so tie high is a power supply cell)...while the cells which wants Vss connects itself to Tie-low.

Inserting Tap Cells

Tap cells are a special nonlogic cell with well and substrate ties. These cells are typically used when most or all of the standard cells in the library contain no substrate or well taps. Generally, the design rules specify the maximum distance allowed between every transistor in a standard cell and a well or the substrate ties.

You can insert tap cells in your design before or after placement:

- You can insert tap cell arrays before placement to ensure that the placement complies with the maximum diffusion-to-tap limit.

- You can insert them after placement to fix maximum diffusion-to-tap violations.

Adding Tap Cell Arrays

Before global placement (during the floorplanning stage), you can add tap cells to the design that form a two-dimensional array structure to ensure that all standard cells placed subsequently will comply with the maximum diffusion-to-tap distance limit.

You need to specify the tap distance and offset, based on your specific design rule distance limit. The command has no knowledge of the design rule distance limit. After you run the command, it is recommended that you do a visual check to ensure that all standard cell placeable areas are properly protected by tap cells.

 Every other row – Adds tap cells in every other row (in the odd rows only). This pattern reduces the number of added tap cells by approximately half, compared to the normal pattern.
 The distance value should be approximately twice that of the distance value specified in the design rule.
 Fill boundary row/Fill macro blockage row – Fills the section of a row that is adjacent to the chip boundary or the macro/blockage boundary to avoid tap rule violation (the default). When deselected, the section of the row adjacent to the chip boundary or the macro/blockage boundary might need to rely on taps outside the boundary to satisfy the tap distance rule.

 Stagger every other row – Adds tap cells in every row. Tap cells on even rows are offset by half the specified offset distance relative to the odd rows, producing a checkerboard-like pattern. Make sure you enter the offset distance to be used.
 The distance value should be approximately four times that of the distance value specified in the design rule.
 Boundary row double density/Macro blockage row double density – Doubles the tap density on the section of a row that is adjacent to the chip boundary or the macro/blockage boundary to avoid tap rule violation (the default). When deselected, the section of the row adjacent to the chip boundary or the macro/blockage boundary needs to rely on taps outside the boundary to satisfy the tap distance rule.

 Normal – Adds tap cells to every row, using the specified distance limit.
 The distance value should be approximately twice that of the distance value specified in the design rule.

 Control the tap cell placement using the following options:

- Ignore soft blockages - Ignores soft blockages during tap cell insertion. The default is false.

- Ignore existing cells - Ignores any standard cells already placed. The default is false. When this option is selected, tap cell placement may overlap existing standard cells.

- At distance tap insertion only - When selected, tap cells are inserted at distance d or at d/2 only. The distance specified with the -distance option is d, and the default is false. With this option, tap cells are placed uniformly but might cause DRC violations.

Tap distance-based
This method is typically used when the internal layout of a standard cell is not available. The command uses a simple distance model where the specified distance from a standard cell to a tap cannot be violated.
Type the tap distance limit, or keep the default.

DRC spacing-based: This method is used when the internal layout of a standard cell is available. The command reads the well, diffusion, and contact layers of the standard cell layout and, by using the intersection of the given layers, identifies the p- and n-transistor diffusion area and the substrate and well contact locations. Also, a tap inside a standard cell or tap cell can be used by the transistor diffusion area of another standard cell. This method makes the most efficient use of the taps and results in fewer taps being inserted.
Specify the maximum distance design rule from a p- or n-diffusion to a substrate or well tap.

Select the name of the following layers, as needed: n-well layer, n-diffusion layer, p-well layer, n-diffusion layer, and contact layer.

- Freeze standard cell – Does not move standard cells. In this method, a higher number of tap cells might need to be inserted, and the resulting placement might not be free of DRC violations.

- Allow moving standard cells – Moves standard cells to avoid overlapping with tap cells. In this method, the timing can sometimes be affected.

9. Which one is best? interleaving or non-interleaving for power planning?
10. Why is power planning done and how? which metal should we usefor power and ground ring & strips and why?

11. What is the use of bloat blockage?
12. How halo rule used in power planning ?
13. How is the power planning for a hierarchical design done?

5. Placement

1. What are the placement optimization methods are used in SOCE and Astro Tool Design?

- PreplaceOpt	- Inplace Opt
- Post Place Opt	- Incremetal Opt
- Timing Driven	- Congestion Driven

2. What is Scan chain reordering? How will it impact Physical Design?

Grouping together cells that belong to a same region of the chip to allow scan connections only between cells of a same region is called scan Clustering. Clustering also allows the degree of congestion and timing violations to be eliminated.

Types of scan cell ordering

- Cluster based scan cell order -power - driven scan cell order.
- Power optimized routing constrained scan cell order.

Power driven scan cell order

• Determining the chaining of the scan cells so as to minimize the toggling rate in the scan chain during shifting operations.

• Identifying the input and output of the scan cells of the scan chain to limit the propagation of transitions during the scan operations.

If scan chain wire length is reduced, it will increase the wireability or reduces the chip die area while at the same time increasing signal speed by reducing capacitive loading effects that share register pins with the scan chains.

After scan synthesis, connecting all the scancells together may cause routing congestion during PAR. This cause area overhead a and timing closure issues.

Scan chain optimization- task of finding a new order for connecting the scan elements such that the wire length of the scan chain is minimized

Placement

Complete placement flow is illustrated in Figure (1).

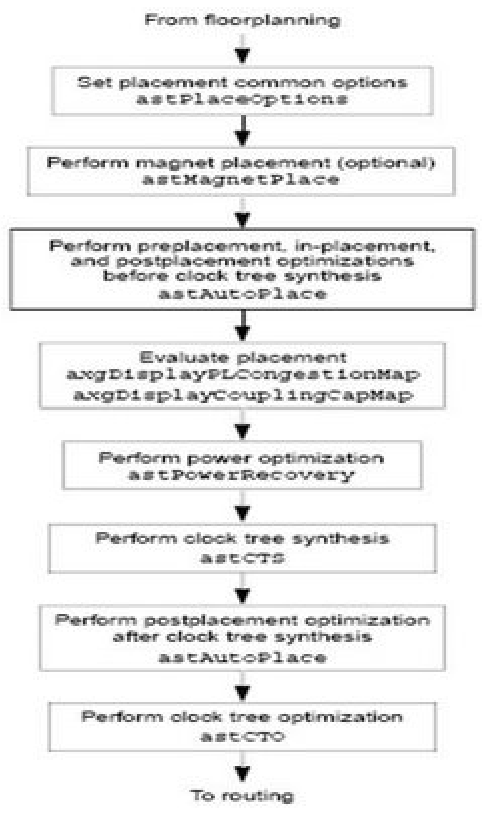

From floorplanning
Set placement common options
astPlaceOptions
Perform magnet placement (optional)
astMagnetPlace
Perform preplacement, in-placement, and postplacement optimizations before clock tree synthesis
astAutoPlace
Evaluate placement
axgDisplayPLCongestionMap
axgDisplayCouplingCapMap
Perform power optimization
astPowerRecovery
Perform clock tree synthesis
astCTS
Perform postplacement optimization after clock tree synthesis
astAutoPlace
Perform clock tree optimization
astCTO
To routing

Figure (1) Placement flow [1]

Before the start of placement optimization all **Wire Load Models (WLM)** are removed. Placement uses RC values from **Virtual Route (VR)** to calculate timing. **VR is the shortest Manhattan distance between two pins.** VR RCs are more accurate than WLM RCs.

Placement is performed in four optimization phases:

1. **Pre-placement optimization**
2. **In placement optimization**
3. **Post Placement Optimization (PPO) before clock tree synthesis (CTS)**
4. **PPO after CTS.**

Pre-placement Optimization optimizes the netlist before placement, HFNs are collapsed. It can also downsize the cells.

In-placement optimization re-optimizes the logic based on VR. This can perform cell sizing, cell moving, cell bypassing, net splitting, gate duplication, buffer insertion, area recovery. Optimization performs iteration of setup fixing, incremental timing and congestion driven placement.

Post placement optimization before CTS performs netlist optimization with ideal clocks. It can fix setup, hold, max trans/cap violations. It can do placement optimization based on global routing. It re does HFN synthesis.

Post placement optimization after CTS optimizes timing with propagated clock. It tries to preserve clock skew.

In scan chains if some flip flops are +ve edge triggered and remaining flip flops are -ve edge triggered how it behaves?

For designs with both positive and negative clocked flops, the scan insertion tool will always route the scan chain so that the negative clocked flops come before the positive edge flops in the chain. This avoids the need of lockup latch.

For the same clock domain the negedge flops will always capture the data just captured into the posedge flops on the posedge of the clock.

For the multiple clock domains, it all depends upon how the clock trees are balanced. If the clock domains are completely asynchronous, ATPG has to mask the receiving flops.

What you mean by scan chain reordering?

Based on timing and congestion the tool optimally places standard cells. While doing so, if scan chains are detached, it can break the chain ordering (which is done by a scan insertion tool like DFT compiler from Synopsys) and can reorder to optimize it.... it maintains the number of flops in a chain.

Answer2:

During placement, the optimization may make the scan chain difficult to route due to congestion. Hence the tool will re-order the chain to reduce congestion.

This sometimes increases hold time problems in the chain. To overcome these buffers may have to be inserted into the scan path. It may not be able to maintain the scan chain length exactly. It cannot swap cell from different clock domains.

What is JTAG?

Answer1:

JTAG is acronym for "Joint Test Action Group".This is also called as IEEE 1149.1 standard for Standard Test Access Port and Boundary-Scan Architecture. This is used as one of the DFT techniques.

Answer2:

JTAG (Joint Test Action Group) boundary scan is a method of testing ICs and their interconnections. This used a shift register built into the chip so that inputs could be shifted in and the resulting outputs could be shifted out. JTAG requires four I/O pins called clock, input data, output data, and state machine mode control.

The uses of JTAG expanded to debugging software for embedded microcontrollers. This elimjinates the need for in-circuit emulators which is more costly. Also JTAG is used in downloading configuration bitstreams to FPGAs.

JTAG cells are also known as boundary scan cells, are small circuits placed just inside the I/O cells. The purpose is to enable data to/from the I/O through the boundary scan chain. The interface to these scan chains are called the TAP (Test Access Port), and the operation of the chains and the TAP are controlled by a JTAG controller inside the chip that implements JTAG.

1. **What is cluster based design? Describe about cluster based region?**
2. **What are the problems are faced when placing long net of FF to FF path and Short net of FF to FF path?**
3. **Is timing driven placement advantageous over the functionality based placement? Explain briefly.**
4. **Explain In Place Optimization and Timing Delay?**
5. **How to do Congestion optimization and balance slew?**

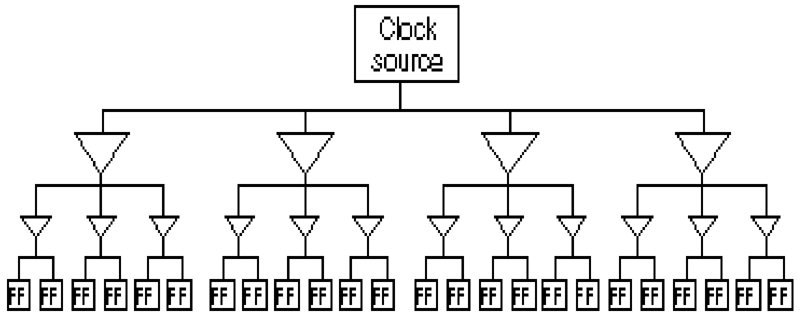
6. Clocktree Synthesis - CTS

1. What is CTS?

Clock tree synthesis is a process of balancing clock skew and minimizing insertion delay in order to meet timing, power requirements and other constraints.

Clock tree synthesis provides the following features to achieve timing closure:

- Global skew clock tree synthesis
- Local skew clock tree synthesis
- Real clock useful skew clock tree synthesis
- Ideal clock useful skew clock tree synthesis
- Interclock delay balance
- Splitting a clock net to replicate the clock gating cells
- Clock tree optimization
- High-fanout net synthesis
- Concurrent multiple corners (worst-case and best-case) clock tree synthesis
- Concurrent multiple clocks with domain overlap clock tree synthesis



2. What are the SDC constraints associated with Clock tree ?

If there are no create_clock statements in the SDC file loaded, CTS will not run. Make sure you have at least one create_clock in your SDC file. If you define create_clock on a pin that is not present physically and is only present in the hierarchical netlist, CTS will not be able to run.

It is good practice to have set_clock_transition, set_clock_latency, and set_clock_uncertainty also defined.

Clock tree synthesis has the following clock tree constraints:

- Maximum transition delay
- Maximum load capacitance
- Maximum fanout
- Maximum buffer level

3. How are the number of Buffer (logic) levels determined during CTS?

In block mode, the number of buffer levels is calculated based on the target load capacitance (0.3 is the default) and the distance between the clock source and clock pins.

The clock tree analyzer (CTA) will calculate the optimal value for the load capacitance during CTS. It will use the value it calculates, or the values you specify for transition, fanout, and load capacitance constraints.

The tool will take the more limiting of these and then convert it to a load capacitance that will drive CTS.

4. Which is better compared to buffer and inverter? If so, Why?

Inverters, Since the Transition time is less for Inverters. It reduced **current flow** between VDD and VSS rail and hence Power reduction. Better to use both with all drive strength to get good skew and insertion delay.

One other advantage of using inverters in a clock tree is the reduction of **duty cycle distortion**. The delay models for a cell library are usually characterized at three different operation conditions or corners: worst, typical, and best. But, there are other effects that are not modeled at these corners. You can have clock jitter introduced by the PLL, variances in the doping of PFETs or NFETs, and other known physical effects of a manufacturing process.

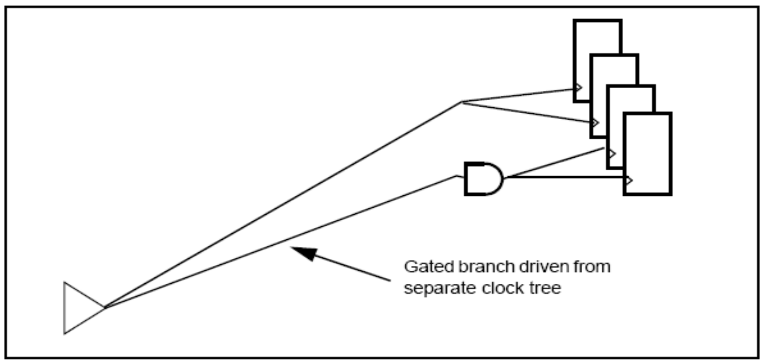
5. While Doing CTS which buffer and inverters are used in the Design?

Clock tree synthesis uses buffers or inverters in clock tree construction. The tool identifies the buffers and inverters if their Boolean function is defined in library preparation. By default, clock tree synthesis synthesizes clock trees with all the buffers and inverters available in your libraries. It is not necessary to specify all of them explicitly in the Buffers/Inverters

6. How will you built Clock tree for Gated Clocks?

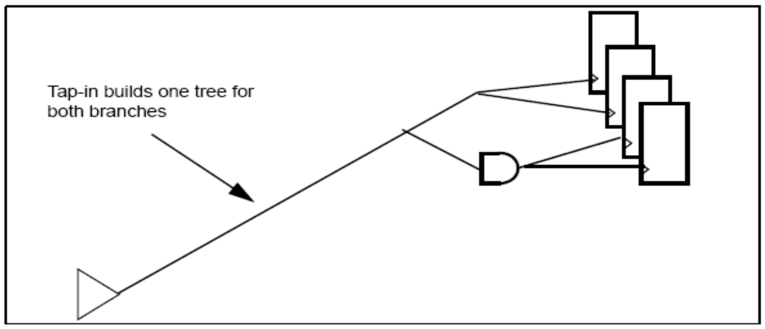
Historically, separate trees are built for any net that drives clock gating elements as well as clock leaves. The two trees diverge at the root of the net. This typically results in excessive insertion delays and makes the clock tree more susceptible to failure due to on-chip variation (OCV).

Gated clock tree with no tap-in



By default, the clock tree synthesizer attempts to tap the gated branches into a lower point in the clock tree, sharing more of the clock tree topology with the non-gated branches. It attempts to insert negative offset branch points earlier in the main tree.

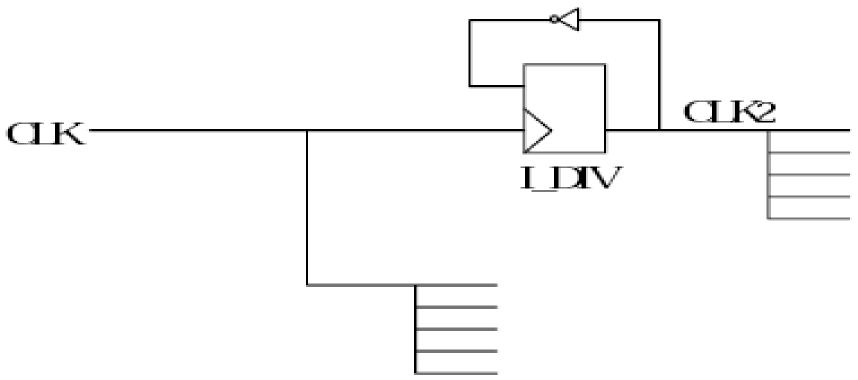
Gated clock tree with tap-in



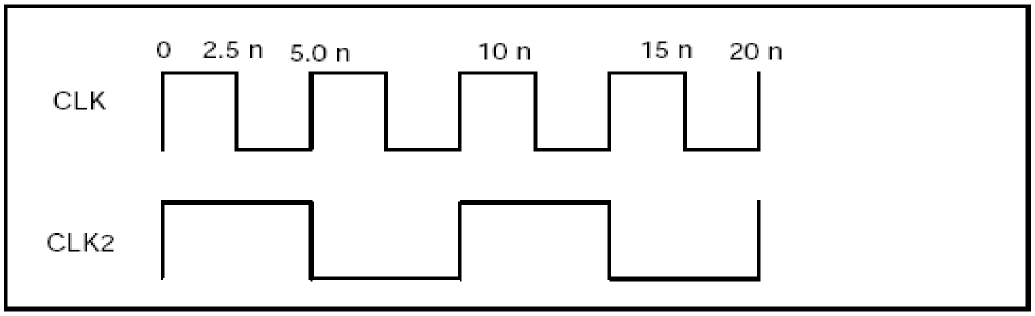
In many cases, this results in fewer buffers being inserted as well as lower clock insertion delays. Sharing the clock tree topology between gated and non-gated branches typically also reduces the local OCV impact on timing. You should disable the clock tap-in feature if too many buffers are inserted or if the clock tree insertion delay is too large.

7. How will you built Clock tree for Generated clocks(Divided by 2 clock)?

Historically, balancing a generated clock with its source was a manual process. In addition, there was no good way to specify groups of pins to be balanced separately from the main clock tree, or to balance one clock tree to another.



A common application of skew grouping is found in circuits that contain clock dividers. Any generated clocks in the design is balanced with the source clock that is generating them. Consider a simple divided clock example. Here are the waveforms that are created by these two clocks



In the normal case, you want to make sure that the rising edge of CLK2 occurs at the same time as the rising edge of CLK at all endpoints. Because these are two clocks, if nothing special is done, they are each balanced separately and have no relationship to one another.

The both edges of the generated clock CLK2:R and CLK2:F into the skew phase of the rising edge of the source clock CLK:R. This is done because the rising edge of the source clock triggers both edges of the generated clock, due to the division taking place.

A skew anchor or Sink point on the clock pin of the divider circuit has to be defined. This is done to alert the clock router that this pin is not to be treated like a standard clock endpoint during the balancing of the clocks.

The presence of a skew anchor in the default skew group causes tool to treat that anchor like a clock gating pin rather than an endpoint. It detects that there is another downstream clock tree from this point, and this clock pin is tapped into the main clock tree early to attempt to compensate for the anticipated additional delay in the CLK2 tree.

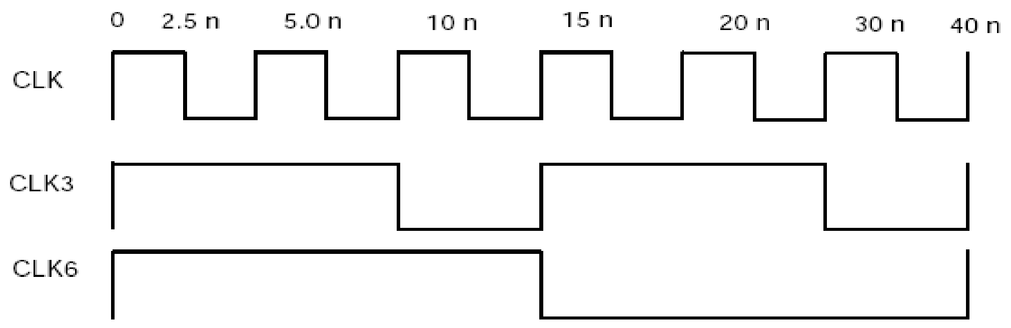
The final constraint takes the falling phase of the root clock CLK:F and places it into the skew phase CLK:R. With the addition of the generated clock phases into the CLK:R skew phase, the CLK:F skew phase contains only the CLK:F clock phase. In many common configurations, not doing this can result in very different arrival time requirements on the rising and falling edges of clock pins in the CLK:F and CLK:R clock phases.

This happens when the CLK:R skew phase arrival time is generated by a late arrival in the CLK2:R or CLK2:F clock phase. This late arrival is propagated to all clock phases in the CLK:R skew phase (remember, CLK:R, CLK2:R, and CLK2:F are all in the
CLK:R skew phase.) When the arrival time for the CLK:F skew phase is calculated, it only has to examine the CLK:F clock phase, which might not be very “deep.” So, the CLK:F skew phase required time is created without knowledge about the CLK:R skew phase arrival time. The discrepancy is removed by moving the CLK:F clock phase into the CLK:R skew phase, and all arrival times are generated in a consistent manner.

Complex Generated Clocks

This example contains several additional structures that can be found in many complicated clock divider circuits. In this example, you add multistage generated clocks as well as a more complicated state machine divider.

Consider the following waveforms:



Generation of these clocks is accomplished by a two-register state machine (that generates the divide-by-three clock CLK3) and a simple divide-by-two clock (that generates CLK6 by dividing CLK3 by 2). Assume that the registers generating CLK3 are called I_DIV3_0 and I_DIV3_1. The register divider generating CLK6 is called I_DIV6.

Notice one of the key differences between this example and “Simple Generated Clocks”.
One of the generated clocks (CLK3) is created by a state machine of more than one register.

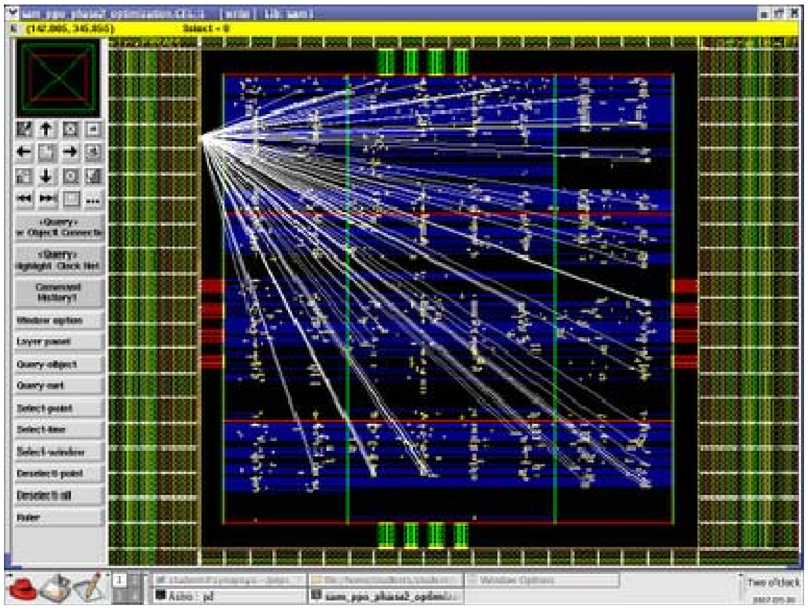
An important concern is that the two clock pins that comprise this state machine should be skew balanced to each other. The register that drives the actual generated clock (I_DIV3_0) must have its clock pin placed in the CLK tree early to account for its downstream delay. Because the other register in the state machine (I_DIV3_1) needs to be balanced to I_DIV3_0, it must also be in the tree early. To achieve this, you must create a skew group.

Astro CTS understands generated clocks so you need not convert create_generated_clock into create_clock.

You can specify the source clock and Astro CTS will traverse through the generated clock definition point and balance the generated clocks's sinks when balancing the source clock

Clock Tree Synthesis (CTS)

The goal of CTS is to minimize skew and insertion delay. Clock is not propagated before CTS as shown in Figure (1).



After CTS hold slack should improve. Clock tree begins at .sdc defined clock source and ends at stop pins of flop. There are two types of stop pins known as ignore pins and sync pins. ‘Don’t touch’ circuits and pins in front end (logic synthesis) are treated as ‘ignore’ circuits or pins at back end (physical synthesis). ‘Ignore’ pins are ignored for timing analysis. If clock is divided then separate skew analysis is necessary.

Global skew achieves zero skew between two synchronous pins without considering logic relationship.

Local skew achieves zero skew between two data dependant synchronous pins while considering logic relationship.

If clock is skewed intentionally to improve setup slack then it is known as **useful skew.**

Rigidity is the term coined in Astro to indicate the relaxation of constraints. Higher the rigidity tighter is the constraints.

In **Clock Tree Optimization (CTO)** clock can be shielded so that noise is not coupled to other signals. But shielding increases area by 12 to 15%. Since the clock signal is global in nature the same metal layer used for power routing is used for clock also. CTO is achieved by buffer sizing, gate sizing, buffer relocation, level adjustment and HFN synthesis. We try to improve setup slack in pre-placement, in placement and post placement optimization **before CTS stages** while **neglecting hold slack**. In post placement optimization after CTS hold slack is improved. As a result of CTS lot of buffers are added. Generally for 100k gates around 650 buffers are added. Global skew report is shown below.

```
******************************************************************
*
* Clock Tree Skew Reports
*
* Tool : Astro
* Version : V-2004.06 for IA.32 -- Jul 12, 2004
* Design : sam_cts
* Date : Sat May 19 16:09:20 2007
*
******************************************************************

======== Clock Global Skew Report ============================

Clock: clock
Pin: clock
Net: clock

Operating Condition = worst
The clock global skew = 2.884
The longest path delay = 4.206
The shortest path delay = 1.322

The longest path delay end pin: \mac21\/mult1\/mult_out_reg[2]/CP
The shortest path delay end pin: \mac22\/adder1\/add_out_reg[3]/CP

The Longest Path:
====================================================================
Pin Cap Fanout Trans Incr Arri Master/Net
-----------------------------------------------------------------
clock 0.275 1 0.000 0.000 r clock
U1118/CCLK 0.000 0.000 0.000 r pc3c01
U1118/CP 3.536 467 1.503 1.124 1.124 r n174
\mac21\/mult1\/mult_out_reg[2]/CP
4.585 3.082 4.206 r sdnrq1
[clock delay] 4.206
====================================================================
```

The Shortest Path:

```
============================================================================
Pin Cap Fanout Trans Incr Arri Master/Net
----------------------------------------------------------------------
clock 0.275 1 0.000 0.000 r clock
U1118/CCLK 0.000 0.000 0.000 r pc3c01
U1118/CP 3.536 467 1.503 1.124 1.124 r n174
\mac22\/adder1\/add_out_reg[3]/CP
1.701 0.198 1.322 r sdnrq1
[clock delay] 1.322
============================================================================
```

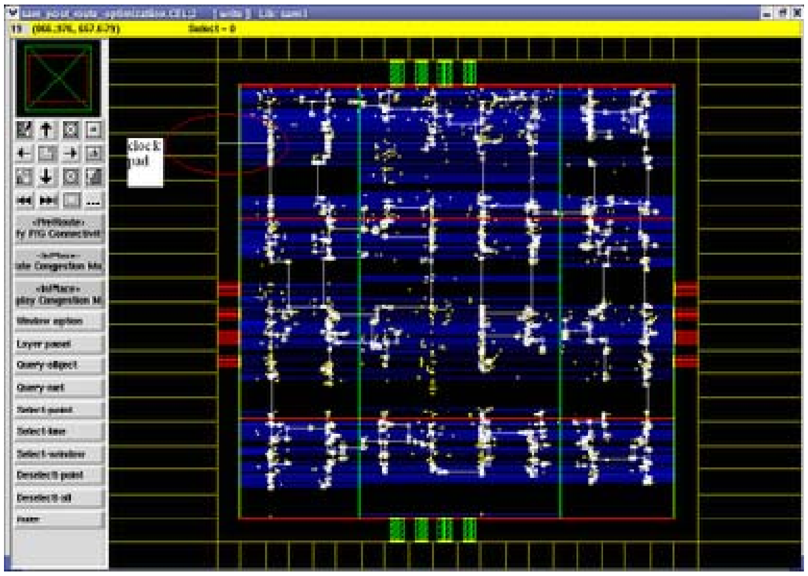


Figure (2) Clock after CTS and CTO

8. Explain Clock tree Options for building better Clock Tree?

Five special clock options are available to address this situation. They greatly expand your ability to control the clock building.

- **Clock phase:**
  - ⇒ A clock phase is a timer event that is associated with a particular edge of the source clock.
  - ⇒ Each clock domain created with two clock phases:
    - (i)The rising edge
    - (ii)The falling edge.
  - ⇒ The clock phases are named after the timing clock with a :R or :F to denote rising or falling clock phase.
  - ⇒ These phases propagate through the circuit to the endpoints, so that events at the clock pins can be traced to events driven by the clocks defined.
  - ⇒ Because Tool is capable of propagating multiple clocks through a circuit, any clock pin can have two or more clock phases associated with it.

⇒ For example, if CLKA and CLKB are connected to the i0 and i1 inputs of a 2:1 MUX, all clock pins in the fan-out of this MUX have four clock phases associated with them—CLKA:R, CLKA:F, CLKB:R, and CLKB:F. (This assumes that you allow the propagation of multiple clock phases).

- **Skew phase:**
  ⇒ A skew phase is a collection of clock phases.
  ⇒ Each clock phase is placed into the skew phase of the same name.
  ⇒ When a clock is defined, skew phases are also automatically created. They are created with the same names as the clock phases that are created.

- **Skew group**
  ⇒ Clock tree skew balancing is done on a **per-skew group** basis.
  ⇒ A skew group is a **subdivision** of a **clock phase**.
  ⇒ Normally, all pins in a clock phase are in group 0 and are balanced as a group.
  ⇒ If you have created a set of pins labeled as group 1,
  For example,
  ⇒ Then the skew phase containing these pins will be broken into two skew groups: one containing the user-specified group, and one containing the "normal" clock pins.
  ⇒ This feature is useful if we want to segregate certain sets of clock pins and not balance them with the default group. We can now define multiple groups of pins and balance them independently.

- **Skew anchor or Sink Point**
  ⇒ A skew anchor is a clock endpoint pin that controls downstream clock tree.
  ⇒ For example, a register that is a divide-by-2 clock generator has a clock input pin that is a skew anchor, because the arrival time of the clock at that clock pin affects the arrival times of all the clocks in the generated domain that begins at the register Q pin.

- **Skew offset**
  ⇒ The skew offset a floating point number to describe certain phase relationships that exist, when placing multiple clocks with different periods or different edges of the same clock different phases into the same skew phase.
  ⇒ Use the skew offset to adjust the arrival time of a specific clock phase when you want to compare it to another clock phase in the same group.

9. How does a skew group relate to the clock phase and skew phase?

⇒ A skew group is a set of clock pins that have been declared as a group. By default, all clock pins are placed in group 0. So each skew phase contains one group.
⇒ If the user has created a group of pins labeled by the number 1, for example, then the skew phase that contains these pins will be broken into two skew groups:
⇒ **(i)** The **"normal"** clock pins
⇒ **(ii)** The **user-specified** group.
⇒ This is useful for segregating groups of clock pins that have special circumstances and that you do not want to be balanced with the default group.
⇒ Skew optimization is performed on a skew-group basis that takes place after the basic clock is inserted

⇒

Why to reduce Clock Skew?

⇒ Reducing clock skew is not just a performance issue, it is also a manufacturing issue.
⇒ Scan based testing, which is currently the most popular way to structurally test chips for manufacturing defects, requires minimum skew to allow the error free shifting of scan vectors to detect stuck-at and delay faults in a circuit.
⇒ Hold failures at the best-case PVT Corner is common of these circuits since there are typically no logic gates between the output of one flop and the scan input on the next flop on the scan chain.
⇒ Managing and reducing clock skew in this case often resolves these hold failures.

Cluster-based approach eases clock tree synthesis

The physical realization of the clock tree network directly impacts the control of clock skew and jitter.

Crosstalk from the clock grid affects circuit speed. And the speed and accuracy of the clock net directly contribute to the minimum cycle time of the chip.

The speed of a design is often its costliest component. But clock skew is an important factor in deciding a clock Period. The skew should be reduced — or, alternatively — utilized to the maximum extent possible within the defined cycle time.

In deep sub-micron (DSM) processes it is a challenge, as it is difficult to control the delay in the clock path due to process variation.

The role of skew in a design, the types of clock trees used in current technology, the effective use of customized cluster based clock-tree synthesis (CTS), the merits of cluster based CTS for skew controlling and, finally, realistic delay consideration for static timing analysis (STA).

Deep submicron effects

A shrink in the feature size, a reduction in operating voltage, dual threshold libraries, multi-voltage or multi-clock domains, diminished gate oxide thickness and manufacturing inaccuracies are examples of technology changes that lead to big challenges in achieving the desired performance.

The interconnect delay now contributes 70~80% of the clock Period. Even so, the faster transistors provided in each successive process generation have led designers to expect that clock period will shrink with each new process.

the supply voltage at each new process node has been reduced to save the power and protect the increasingly thin gate oxides. With each reduction, problems such as ground bounce, cross talk, glitch propagation and so forth were exacerbated because of low noise margin and increased leakage power. These problems, in turn, can force designers to slow down the slew rates and clock rise/fall times in their circuits.

Role of skew in a design

Any factor that contributes delay to a clock net — a mismatch in the RC delay or buffering the clock network, for example — can contribute to skew. Variations in manufacturing also can contribute unwanted skew in the clock network. The unwanted skew caused by process variation becomes the bottleneck for improving the clock frequency in high-frequency designs.

The clock delay of a register is the time needed for a clock-edge to propagate from the clock source to the register. Because of the net delays and/or buffer delays (together here called "**insertion delay**"), the clock edges reach the registers at different times. That is, each register receives the clock with a different insertion delay.

By definition, the **clock skew** is the difference between the longest insertion delay and the shortest insertion delay of a clock network. If d_l and d_c are the clock arrival time of launch and capture registers, then $\mathbf{d_l > d_c}$ leads to **negative skew** and $\mathbf{d_l < d_c}$ leads to **positive skew**. To make sure the clock operates within the required cycle time,

$(d_c – d_l)$ must be no more than (min logic delay + register delay – hold) for hold

And

$(d_l – d_c)$ must be no more than (cycle time – (max logic delay + register delay + setup)) for setup

To meet these constraints make the skew as close as possible to zero. But to achieve zero skew, the design needs to have a huge amount of buffer insertion and wire sizing, to make the insertion delay equal at all registers. Clearly, this may lead to unwanted area and power consumption.

Alternatively, a design may accept a small amount of skew in order to avoid blowing up area and power consumption. Controlling such non-zero skew is the critical part of clock tree synthesis. This article explains cluster-based clock tree synthesis, which delivers an optimal result on skew control.

Types of clock trees

There are many clock tree structures used widely in the design industry, each of which has its own merits and demerits.

1. H-tree (figure 1)
2. Balance tree (figure 2)
3. Spine tree (figure 3)
4. Distributed driven buffer tree (figure 4).

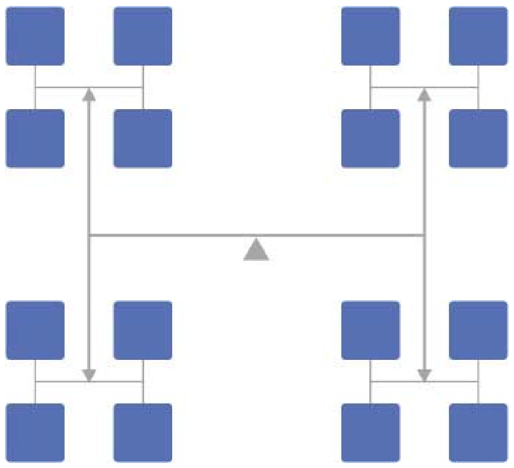


Figure 1 — H-tree, Because of the balanced construction, it is easy to reduce clock skew in the H-tree clock structure. **A disadvantage** to this approach is that the fixed clock plan makes it difficult to fix **register placement**. It is rigid in fine-tuning the clock tree.

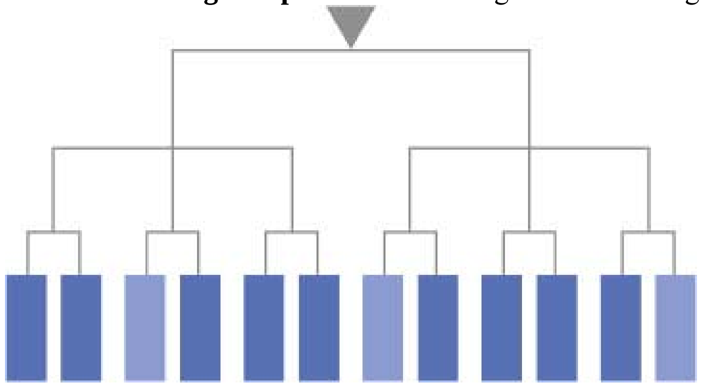


Figure 2 — **Balanced tree** makes it easy to adjust capacitance of the net to achieve the skew requirements. But the **dummy cells** used to balance the load increase **area** and **power**.

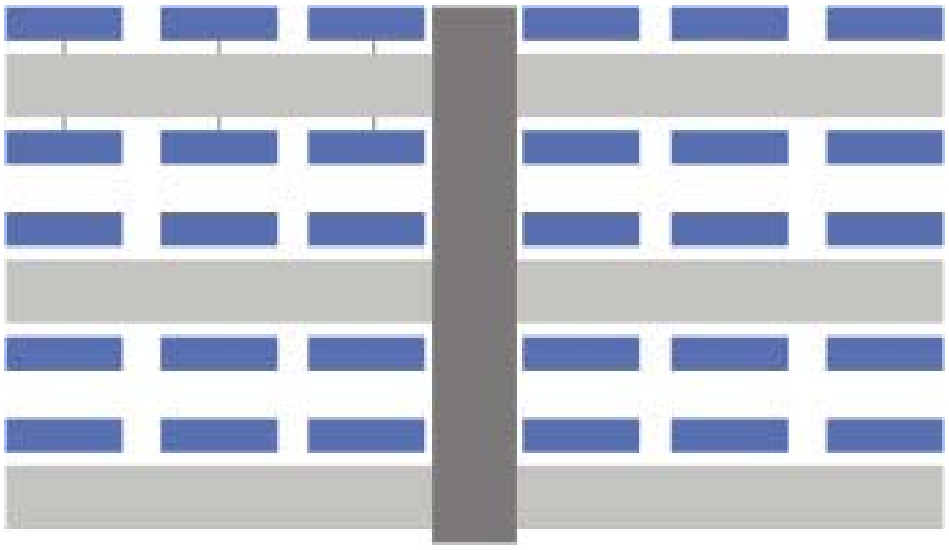


Figure 3 — The **spine tree (Fish bone)** arrangement makes it easy to reduce the skew. But it is heavily influenced by process parameters, and may have problems with **phase delay**.

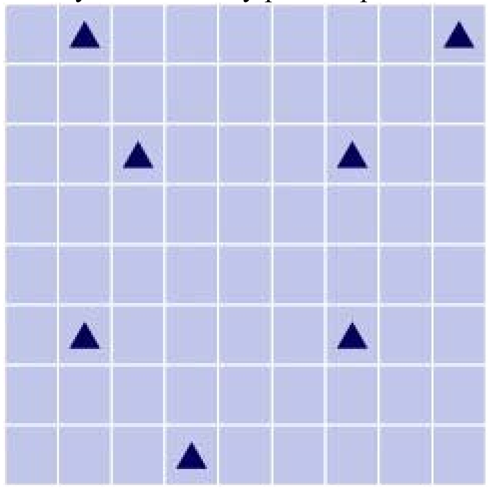


Figure 4 — **Distributed driven buffer tree**. Distributed buffers make it easy to reduce skew and power. Clock routing may not be an issue. However, since buffering is customized, it may be a less area efficient method.

Cluster based CTS

A better plan in principle would be to individually balance and distribute the clock from the source to each of the logic elements. If we were simply to lay out the logic and then route a clock signal to each logic element, using constant wire width and no buffers, then obviously the register furthest from the source will receive the clock with the greatest delay.

Delay can then be equalized by adding buffers/repeaters or lengthening the wires in the shorter nets so that the clock reaches all the registers at the same time. But this is not always true in practice, because in real life routing the clock directly from the source may not be possible in high speed and multi clock domains. A practical approach for such problems can be customized cluster-based clock tree synthesis.

In a customized cluster-based clock tree plan, the logic elements operated in a single clock domain or the logic with the same input timing registers are combined together to form a group. A reference register for each group is selected to establish a reference arrival time.

Initially these clusters are placed to meet the latency requirements based on the clock insertion delay from the source. These clusters can be individually optimized to minimize skew, so that the skew will be within the allowable range for the desired cycle time.

The cluster level routing can use any of the routing topologies mentioned above, based on the priorities of the design. For example, a cluster sensitive to skew can use H-tree or balanced-tree designs.

Obviously, the trade off between power, area and amount of buffers added into the network are to be preplanned before selecting a method for each group. It is not advisable to limit the performance based on the clock tree topology.

Advantages of cluster based CTS

In a deep submicron process, millions of gates and very high frequencies — sometimes multiple GigaHertz — are becoming normal. In order to achieve such high frequency requirements, the clock tree network needs to be very well planned and elaborated. The designers should be able to plan for the skew requirements to achieve the minimum cycle period, instead of trying to force the skew to zero in the presence of perhaps poorly-characterized process variations.

One of the biggest advantages of doing cluster-based CTS is that the delay due to voltage drop in the interconnect can be modeled or incorporated into static timing analysis. A voltage drop of 10% of Vdd may lead to more than 15% delay variation in nanometer technologies.

Typically the voltage drop will be more severe at the center of the chip, so the standard cells characterized for either worst case or best can not give accurate delay values — they cannot account for voltage variations that are happening on the fly. Though simultaneous, mixed min-max or on-chip variation (OCV) kinds of analysis are derived for such situation, there is unfortunately no static timing engine that considers the delay of standard cells due to the variation in the voltage. A cluster near the center of a chip, for example, may not receive 100% of Vdd, so the delay also will vary at the center since the voltage of those standard cells are less than Vdd.

When we use a cluster-based customized clock tree synthesis method, we always treat the cluster based on its priority within the chip. If the voltage drop at the center of the die is a potential problem, then the clusters at the center would always have timing priorities with setup and hold margins sufficient to prevent the voltage drop from causing timing problem. There always needs to be a trace-back analysis with actual voltage drop numbers to define the setup and hold margin. In such cases, the clusters can be operated based on the weighting of timing, area, power, and other factors.

Conclusion

For many designers, timing closure is the major issue in high-speed designs, which leads to any number of iterations and schedule-consuming tasks. At 350nm or 250nm technology, the EDA companies understood timing closure issues and they updated their tools to a certain extent to meet the designers' needs.

But in the current scenario, timing closure is the major part of the design cycle. Because of time-to-market pressure, new design methodologies at the system integration level like system on chip and network on chip are using on-chip buses and multiple clock domains. This kind of design requirement pushes designers to use new design tricks like timing driven placement, useful-skew concepts, or customized cluster-based CTS.

Since there are plenty of IP cores available in the market, simply designing an SoC may not consume much time. But after the blocks have been selected, the physical implementation into hardware consumes more time and effort. We may have to develop a customized design flow for each and every stage of the design cycle to address the current challenges in chip implementation.

Clock tree synthesis is a case in point. As we have discussed throughout the paper, the designers should not be limited to any fixed topology of clock routing algorithms. Rather, they should be encouraged to mix up any of the described clock routing schemes into a single chip. This customized cluster-based clock tree synthesis utilizes the best topology to meet requirements like skew, area, and power at every stage, and it improves the top-level system performance.

Handling Clock Gating Cells with Non-Integrated Clock Gating Cells

Clock gating cells are used in many designs to reduce power consumption. Tools liked power compiler will insert gating cells automatically. One type of clock gating is done using a Latch and an AND gate. There may be hundreds on instances of this gating “Cell” spread throughout the design.

The master clock is defined at the top cell port and the output of each clock gating cell drives several flip flops. The clock connects to the G port (clock of FF) of the latch and the B input of the AND gate. Clk2 is the output of this clock gating cell(output of AND gate), which will be connected to the FF or other clock gating cells depending on the implemented clock gating structure. Net-1 is coming from another gate (D Input of FF) which is usually has the test inputs.

The requirement for this type of design is for the clock to arrive at the A and B ports of the AND gate at the same time. To achieve this, the clock tree – when built –should not synchronize the latch with other FFs, If CTS synchronizes the latch with the FF, the clock gating check at the A and B inputs of the AND gate will fail because the clock will arrive early compared to its arrival at the clock ports of the latch and the FFs. Astro CTS will synchronize the latch with other FFs when there is no integrated clock gating cell in the library. For integrated clock gating cells, CTS is supported automatically.

Soultion:

1) Get the clock go through the latch – not taking the G pins as a sync port.
2) Define the latch as a clock cell in the cLF fiel –this will let the clock to go through it.

What are all the Checks to be done before doing CTS?

• Hierarchical pins should not be defined as a clock source

• Generated clock should have a valid master clock source
A generated clock does not have a valid master clock source in
the following situations:

- The master clock specified in create_generated_clock does not exist.
- The master clock specified in create_generated_clock does not drive the source pin of the generated clock.
- The source pin of the generated clock is driven by multiple clocks, and some of the master clocks are not specified with create_generated_clock.

• Clock (master or generated) with no sinks
• Looping clock
• Cascaded clock with an un-synthesized clock tree in its fan-out
• Multiple-clocks-per-register propagation not enabled, but the
design contains overlapping clocks
• Clock tree exceptions should not be ignored.

• Stop pin or float pin defined on an output pin is an issue.

What are all the Best Practices of Clock Tree Synthesis (CTS)?

Avoiding Clock Tree Synthesis (CTS) Pitfalls.

Here, some common problems are discussed that many design engineers today face when they try to run CTS.

The suggestions and steps you can take to avoid these problems are discussed.

It is always a good idea to invest the time at the beginning to understand the clock structure and the clock gating in your design.

You should also know where your gating elements are placed and the topology of your floor plan including the location of sync points.

That will ensure that you are building clock trees that will give you good timing results. This knowledge will also help you make clock tree decisions that will help CTS build better quality clock trees.

The following list provides suggestions for dealing with problems you may encounter.

A. Big insertion delays.

Some of the things you can check if you have big insertion delays are:

1. *Are there delay cells in the design?*

Check to see if there are delay cells in the netlist that are present in the current design. These could be causing delay that cannot be optimized and CTS is building clock trees which match all other paths to this worst insertion delay.

2. *Are there cells marked "don't touch" in the design?*

There could be cells in the design that are marked "dont touch" which prevents CTS from deleting them and building optimal clock trees.

3. *Can the floor-plan be modified to be more clocks friendly?*

Sometimes it helps to consider CTS (and timing) as a constraint for floor-planning. Long skinny channels leading to more long skinny placement channels will give both timing optimization and CTS problems.

Consider using soft blockages or refloorplan.

4. *Can you define new create_clocks that will assist CTS (divide and rule)?*

Many times running CTS on the main clock pin is not the optimal way to build clock trees. It may help to divide the clock tree based on the floorplan and the syncpins and build sub clocks, then define the sync pins and build the upper main clock.

5. *Are the syncPins defined correctly for macros?*

It is a good idea to check the syncPins file to see if the sync pins make sense. Also check that the numbers are accurate and that the time units are correct.

6. *If there are ignore pins in the design are they defined as ignore pins?*

If there are ignore pins in the design, make sure you define these as ignore pins before running CTS.

7. *Have you used varRouteRules and propogated by astMarckClockTree?*
   - Defining varRouteRules helps to reduce the insertion delay.
   - Define shielding, and double or more width rules for clock nets, and propagate them using astMarkClockTree.
8. *Are the CTU buffers marked as "dont use"?*

Some technologies use clock tree buffers. Make sure you are using these only for your clock tree. Also make sure they are not marked "dont use".

9. *Be creative and use different CTS intParams to get better results.*

There are several CTS options in the form that you can try to change to get better or more desireable CTS results.

10. *Use the Block option in CTS in the first attempt.*

This usually gives better insertion and skew results. If your design is less than 5% std cell utilization try the Top option.

11. *CTO is designed to work on skew and will not reduce insertion delay once it is built.*

    Try providing a higher skew goal during CTS.

12. *Use inverters only to build the clock tree if possible.*
13. *Define variable route rules with greater than default widths and clearance and also shield the clock nets. Then propagate these rules using astMarkClockTree. This will help insertion delay.*

B. Unreasonable skew.

Some of the things you can check if you have unreasonable skew are:

1. *All the above except A(11).*

   All the issues discussed above also apply to skew debugging.

2. *Do you have derived clocks that do not need skew matching?*

If you have clocks that get divided and some branches do not need skew balancing with the rest, then build clock trees for them separately and do not allow skew calculation between them.

   You can define sync pins or ignore pins at cross-over oints.

3. *Look closely at your worst path(s) for possible culprits.*

It is quite likely that some of your worst paths have an issue which is preventing CTS from optimizing them and is causing all other paths to get delay added to match the insertion delay or better skew.

4. *Use intParams areaBased and ECOWinSize to help the Overlap Removal (OV) engine.*

C. **CTS doesn't run properly.**

Some of the things you can check if CTS doesn't run properly are:

1. *All of the above.*

For general quality of results issues, all the above points should be checked.

2. *Are the SDC constraints loaded and is create_clock defined?*

If there are no create_clock statements in the SDC file loaded, CTS will not run. Make sure you have at least one create_clock in your SDC file.

It is good practice to have set_clock_transition, set_clock_latency, and set_clock_uncertainty also defined.

For the SDC latency values to be honored, the intParam axSetIntParam "acts" "clock uncertainty goal" 1 should be set.

CTS uses constraints in the CTS form as first priority, then it uses the constraints in the intParams, and then it uses SDC constraints.

Having these in the SDC file will also enable the timer to account for your skew and insertion delay in optimization steps.

3. *If you have multiple create_clock statements in a path, did you set the "define ataPropagateClockThruCreateClock" correctly?*

This is a new define parameter in the 2003.06 release and it needs to be set correctly.

4. Build the clock tree on lower clocks, then define the sync pins and run CTS on next level up (divide and conquer).

This is a good practice when building clock trees. Always remember to define sync pins if you need them.

5. *astSetDontTouch ?clock_buffers.list? #f done?*

If your CTS buffers have a "dont use" property in your library, you need to set that to false.

6. *Are the clock nets marked "dont touch" or is set_case_analysis defined?*

Occasionally you may end up with a "dont touch" property on your clock net as a results of your analysis. Make sure you reset this using the astmarkClockTree command.

Also if your SDC constraints have a set_case_analysis defined that disables the clock net, CTS will not build clock trees.

7. *Is create_clock defined on a non-physical hierarchical pin?*

If you define create_clock on a pin that is not present physically and is only present in the heirarchical netlist, CTS will not be able to run.

8. *Try different CTS options and use the one that gives the best results.*

As always, it is a good idea to experiment and try out different CTS options and intParams to get the best result.

Clock Timing Constraints for Synthesis, Clock Tree, and Propagated Clock Timing

Question:

What are the timing constraints in Design Compiler that can cause problems in the backend, clock tree synthesis, and propagated clocks?

Answer:

Certain clock constraints are acceptable in Design Compiler but can cause problems in the backend during clock tree synthesis and propagated mode timing analysis.

Because all clock insertion delays are zero during synthesis, many incorrect definitions don't cause problems during synthesis but cause serious clock tree synthesis and propagated clock analysis problems.

Ideally, Primetime signoff constraints must be used throughout the flow: synthesis, clock tree synthesis, and during signoff timing analysis with Primetime. If this is not possible, at least the clock constraints must be consistent throughout the flow.

These critical clock constraints are discussed below.

(1) Root pin of Create Clock and Create Generated Clock

(2) Should it be Create Clock or Create Generated Clock?

(3) Clock waveform

(4) Clock Latency: source and propagated

(5) Master (Source) of Generated Clock

(6) Create Generated Clock: div_by 1 or -comb

(1) Root pin of Create Clock or Create Generated Clock?
===
Ideal Case
==========
Make sure that the root pin of the create clock constraint is one of the following:
(a) input port
(b) output pin of a hard macro block such as PLL or analog block

Make sure that the root pin of the create generated clock constraint is
the output pin of a register.

Non-Ideal Case
==============
If the root pin of a clock is anything other than the above, make sure that
you are aware of the ramifications in the propagated clock mode and during
Clock Tree Synthesis.

If the create_clock constraint is defined on an internal pin, be aware of
the following:

(a) Every create_clock constraint becomes the startpoint for insertion delay
calculation in propagated mode. Try using the create_generated_clock command.

(b)Every create_clock constraint triggers a new clock tree during clock
tree synthesis. Every create_clock constraint automatically becomes the sync
pin for any upstream clock traversing to that point. Try using the
create_generated_clock command.

(c) Every create_clock constraint overrides any other clock present at that
point unless the -add option is used. Try using the
create_generated_clock command.

(d) Every create_generated_clock constraint overrides any other clock present
at that point unless the -add option is used. Try defining multiple
create_generated_clock. Note that the create_generated_clock command enables
insertion delay calculation from the master (source) clock, unlike the
create_clock command.

(e) A create_clock constraint defined on a hierarchical pin is not supported
during clock tree synthesis. Clock tree synthesis requires input port or an
instance pin as the startpoint of a clock.

It is highly recommended that you do NOT synthesize the clock logic.
Instantiation is best. Besides unpredictable logic and clock glitches, it
may not be possible to define the start, sync, exclude and ignore pins
during clock tree synthesis.

For example:

Four clocks go to two 2-input MUXes. It is best to instantiate these MUXes.

If synthesized, the resulting logic may not have MUXes at all. Logic gates
could be shared along with the input pins. That is, A and B pins of the MUXes
cannot be identified.

It is no longer possible to propagate a particular clock through one MUX and
not the other, because the input pins are not identifiable.

(2) Should it be Create Clock or Create Generated Clock?
===

Normally create clock should be defined for all the clocks from input ports and output pins of hard macros such as PLL and analog blocks. For any internal pins, the create_generated_clock constraint is best.

When create_generated_clock is defined, latency (insertion delay) of this clock is calculated from its master (source) clock. When create_clock is defined, latency (insertion delay) of this clock is calculated from the point where it is defined.

For example, assume that a create_generated_clock constraint is defined at the output pin of a MUX with a source clock defined. This implies that the insertion delay of the generated clock needs to be calculated from the source clock through the MUX and to the sync pins of the generated clock.

If a create_clock constraint is defined instead at the output of the MUX, then the insertion delay is calculated from the output pin of the MUX and NOT from the source clock.

Double check that all the create clock definitions that are defined on the internal pins of the design. They might have to be changed to the create_generated_clock constraint.

(3) Clock Wave form
===================

Incorrect
=========

For simplicity, assume that a PLL generates two clocks of the same frequency.

```
create_clock -name CLKA -pin PLL/CLKA -waveform {0 5} -period 10
create_clock -name CLKB -pin PLL/CLKB -waveform {2 7} -period 10
```

There is an insertion delay of 2ns coded into the waveform of CLKB.

It is difficult for paths from CLKA to CLKB to meet timing, therefore, this insertion delay is included with CLKB to provide an additional 2ns time. This insertion delay is expected to be added by the clock tree synthesis tool.

This intention of the designer is not evident from the waveform shifting and the create_clock definition. This constraint adds an additional 2ns even afterclock tree synthesis add the 2ns delay, causing incorrect timing analysis in propagated mode.

Correct
=======

```
create_clock -name CLKA -pin PLL/CLKA -waveform {0 5} -period 10
create_clock -name CLKB -pin PLL/CLKB -waveform {0 5} -period 10

set_clock_latency -clock CLKB 2ns
```

The above set of commands defines a network insertion delay of 2ns. In ideal mode, this constraint is obeyed. In propagated mode, this constraint is ignored and actual insertion delay is calculated.

The intention of the designer is clear in this case.

(4) Clock Latency, source and propagated
===

Use the set_clock_latency command to define the estimated network insertion delay OR inherent source delay as follows:

```
set_clock latency -clock CLKA 2ns
set_clock_latency -clock CLKA 1ns -source
```

The first constraint specifies that the predicted insertion delay for clock CLKA is 2ns and is used to shift the CLKA waveform in "ideal" mode. This constraint is ignored in propagated mode. It is not necessary to define or estimate network insertion delay for a clock, unless the design needs it to meet timing as discussed above.

The second constraint specifies that there is an inherent insertion delay of 1ns for the clock CLKA. This is present BOTH in "ideal" AND "propagated" mode. Assume this source latency to be coming from outside of the chip or in the above PLL case, arriving from the PLL itself.

The clock_latency constraint can be applied to generated clocks as well.

(5) Master (Source) of Generated Clock
=======================================

For every create_generated_clock, the timing tool tries to find a path to the source (master) clock. If it cannot find the path, CTS issues an error and the timing tool might give unpredictable results.

Nested Generated Clocks
=======================

The master (source) of a generated clock can be another generated clock or a create_clock constraint. In the case of several nested create_generated_clocks, it is better to define each create_generated_clock constraint with respect to the previous create_generated_clock. Although doing so seems cumbersome, it is a recommended approach.

The reasons are as follows. By default, for every generated clock defined, timing tool tries to find a path to the source (master) clock by tracing backwards. Recall that most generated clocks are defined at the output of div_by registers. So, in trying to find a path to the source clock, the timing tool allows itself to jump one sequential cell (such as a register).

Defining nested generated clock source as the previous generated clock helps the tool find the right sequence of registers to calculate insertion delay.

Consider the following example.

```
create_clock -name CLKA -port CLKA -waveform {0 5} -period 10

create_generated_clock -name CLKB -source CLKA -div_by 2 core/div_by_2_reg/Q
create_generated_clock -name CLKC -source CLKB -div_by 2 core/div_by_4_reg/Q
create_generated_clock -name CLKD -source CLKC -div_by 2 core/div_by_8_reg/Q
```

The above sequence of commands is better than defining all the create_generated_clocks with the source as CLKA. The reason is that, in propagated mode, the above nested definition helps the timing tool calculate the insertion delay correctly along the following path:

port CLKA ---> div_by_2 ----> div_by_4 ----> div_by_8 registers.

Instead, if all the create_generated clocks are defined with respect to the main CLKA source, then the timing tool may find some other path other than the above to calculate the insertion delay, resulting in incorrect timing numbers.

How many generated clocks are needed?

Note that the create_generated_clock constraint at a pin removes any other clock definition at that point unless the -add option is used.

Consider an example in which two clocks, CLKA and CLKB, traverse to a MUX. The output of the MUX goes to a div_by_2_reg.

```
create_clock -name CLKA -pin PLL/CLKA -waveform {0 5} -period 10
   ----> goes to A pin of MUX
create_clock -name CLKB -pin PLL/CLKB -waveform {0 5} -period 10
   ----> goes to B pin of MUX
```

Assume that the MUX goes to the CK pin of core/div_by_2 register, that is,

MUX ---> core/div_by_2_reg.

```
create_generated_clock -name CLKC -div_by 2 -pin core/div_by_2_reg/Q -source CLKA
```

This create_generated_clock definition will prevent CLKB from traversing through the MUX.

That is, during timing analysis, CLKB is never seen after the div_by_2_reg.

During clock tree synthesis, only CLKA is balanced through the MUX and div_by_2_reg and CLKB stops at the same register and treats that as a SYNC pin. If CLKB is also desired to trace through the div_by_2_reg, then add the additional generated clock as follows to the above constraints:

```
create_generated_clock -name CLKD -div_by 2 -pin core/div_by_2_reg/Q -source CLKB -add
```

(6) Create Generated Clock: div_by 1 or -comb
==

By default, for every generated clock defined, the timing tool attempts to find a path to the source (master) clock by tracing backwards. Recall that most generated clocks are defined at the output of div_by registers. So, in trying to find a path to the source clock, the timing tool allows itself to jump one sequential cell (such as a register).

Consider the case when the generated clock is a -div_by 1 clock. This could allow the tool to find a path to the source by jumping through a register or through a combinational path. This may not be appropriate, if the correct path from the source is combinational. In such cases, define the generated_clock as -comb so that the tool is forced to find the combinational path.

The following command is useful to get the full details of how the clock latency is being calculated and the path it finds from the source clock to the generated clock.

```
report_timing -type full_clock_expanded
```

10. Handling of the PLL to pad connection while building CTS.

11. Explain Clock tree Options for building better Clock Tree?

Five special clock options are available to address this situation. They greatly expand your ability to control the clock building.

- **Clock phase:**
  - ⇒ A clock phase is a timer event that is associated with a particular edge of the source clock.
  - ⇒ Each clock domain created with two clock phases: one for the rising edge and one for the falling edge.
  - ⇒ The clock phases are named after the timing clock with a :R or :F to denote rising or falling clock phase.
  - ⇒ These phases propagate through the circuit to the endpoints, so that events at the clock pins can be traced to events driven by the clocks defined.
  - ⇒ Because Tool is capable of propagating multiple clocks through a circuit, any clock pin can have two or more clock phases associated with it.
  - ⇒ For example, if CLKA and CLKB are connected to the i0 and i1 inputs of a 2:1 MUX, all clock pins in the fanout of this MUX have four clock phases associated with them—CLKA:R, CLKA:F, CLKB:R, and CLKB:F. (This assumes that you allow the propagation of multiple clock phases).

- **Skew phase:**
  - ⇒ A skew phase is a collection of clock phases.
  - ⇒ Each clock phase is placed into the skew phase of the same name.

- ⇒ When a clock is defined, skew phases are also automatically created. They are created with the same names as the clock phases that are created.
- ⇒ Each clock phase is placed into the skew phase of the same name.

- **Skew group**
  - ⇒ Clock tree skew balancing is done on a per-skew group basis.
  - ⇒ A skew group is a subdivision of a clock phase.
  - ⇒ Normally, all pins in a clock phase are in group 0 and are balanced as a group.
  - ⇒ If you have created a set of pins labeled as group 1,
    For example,
  - ⇒ Then the skew phase containing these pins will be broken into two skew groups: one containing the user-specified group, and one containing the "normal" clock pins.
  - ⇒ This feature is useful if we want to segregate certain sets of clock pins and not balance them with the default group. We can now define multiple groups of pins and balance them independently.

- **Skew anchor or Sink Point**
  - ⇒ A skew anchor is a clock endpoint pin that controls downstream clock tree.
  - ⇒ For example, a register that is a divide-by-2 clock generator has a clock input pin that is a skew anchor, because the arrival time of the clock at that clock pin affects the arrival times of all the clocks in the generated domain that begins at the register Q pin.

- **Skew offset**
  - ⇒ The skew offset a floating point number to describe certain phase relationships that exist, when placing multiple clocks with different periods or different edges of the same clock different phases into the same skew phase.
  - ⇒ Use the skew offset to adjust the arrival time of a specific clock phase when you want to compare it to another clock phase in the same group.

12. How does a skew group relate to the clock phase and skew phase?

- ⇒ A skew group is a set of clock pins that have been declared as a group. By default, all clock pins are placed in group 0. So each skew phase contains one group.
- ⇒ If the user has created a group of pins labeled by the number 1, for example, then the skew phase that contains these pins will be broken into two skew groups:
- ⇒ **(i)** one containing all of the **"normal"** clock pins
- ⇒ **(ii)** A separate group containing the **user-specified** group.
- ⇒ This is useful for segregating groups of clock pins that have special circumstances and that you do not want to be balanced with the default group.

Skew optimization is performed on a skew-group basis that takes place after the basic clock is inserted

How will you synthesize clock tree?

- o -Single clock-normal synthesis and optimization
- o -Multiple clocks-Synthesis each clock seperately

- -Multiple clocks with domain crossing-Synthesis each clock seperately and balance the skew

How many clocks were there in this project?

- -It is specific to your project
- -More the clocks more challenging !

How did you handle all those clocks?

- -Multiple clocks-->synthesize seperately-->balance the skew-->optimize the clock tree

Are they come from seperate external resources or PLL?

- -If it is from seperate clock sources (i.e.asynchronous; from different pads or pins) then balancing skew between these clock sources becomes challenging.
- -If it is from PLL (i.e.synchronous) then skew balancing is comparatively easy.

Why buffers are used in clock tree?

- To balance skew (i.e. flop to flop delay)

Which is more complicated when u have a 48 MHz and 500 MHz clock design?

- 500 MHz; because it is more constrained (i.e.lesser clock period) than 48 MHz design.

Clock Gating

Clock tree consume more than 50 % of dynamic power. The components of this power are:

1) Power consumed by combinatorial logic whose values are changing on each clock edge
2) Power consumed by flip-flops and
3) The power consumed by the clock buffer tree in the design.

It is good design idea to turn off the clock when it is not needed. Automatic clock gating is supported by modern EDA tools. They identify the circuits where clock gating can be inserted.

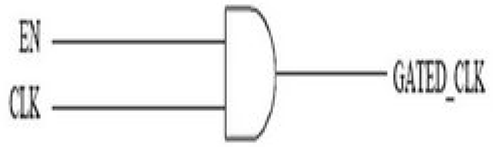
RTL clock gating works by identifying groups of flip-flops which share a common enable control signal. Traditional methodologies use this enable term to control the select on a multiplexer connected to the D port of the flip-flop or to control the clock enable pin on a flip-flop with clock enable capabilities. RTL clock gating uses this enable signal to control a clock gating circuit which is connected to the clock ports of all of the flip-flops with the common enable term. Therefore, if a bank of flip-flops which share a common enable term have RTL clock gating implemented, the flip-flops will consume zero dynamic power as long as this enable signal is false.

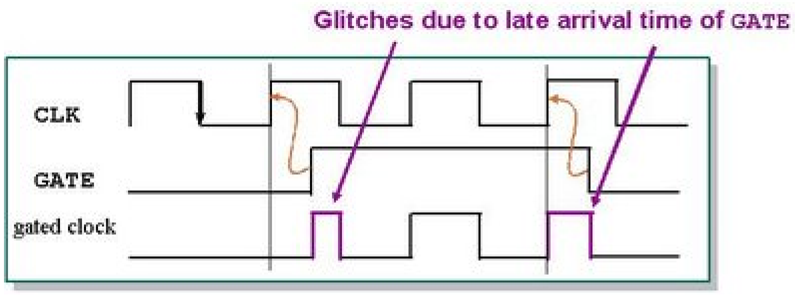
There are two types of clock gating styles available. They are:

1) Latch-based clock gating
2) Latch-free clock gating.

Latch free clock gating

The latch-free clock gating style uses a simple AND or OR gate (depending on the edge on which flip-flops are triggered). Here if enable signal goes inactive in between the clock pulse or if it multiple times then gated clock output either can terminate prematurely or generate multiple clock pulses. This restriction makes the latch-free clock gating style inappropriate for our single-clock flip-flop based design.

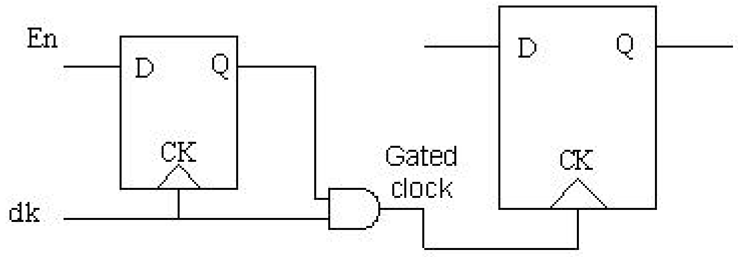




Latch free clock gating

Latch based clock gating

The latch-based clock gating style adds a level-sensitive latch to the design to hold the enable signal from the active edge of the clock until the inactive edge of the clock. Since the latch captures the state of the enable signal and holds it until the complete clock pulse has been generated, the enable signal need only be stable around the rising edge of the clock, just as in the traditional ungated design style.



Latch based clock gating

Specific clock gating cells are required in library to be utilized by the synthesis tools. Availability of clock gating cells and automatic insertion by the EDA tools makes it simpler method of low power technique. Advantage of this method is that clock gating does not require modifications to RTL description.

What is difference between normal buffer and clock buffer?

Clock net is one of the High Fanout Net(HFN)s. The clock buffers are designed with some special property like high drive strength and less delay. Clock buffers have equal rise and fall time. This prevents duty cycle of clock signal from changing when it passes through a chain of clock buffers.

Normal buffers are designed with W/L ratio such that sum of rise time and fall time is minimum. They too are designed for higher drive strength.

What is difference between HFN synthesis and CTS?

HFNs are synthesized in front end also.... but at that moment no placement information of standard cells are available... hence backend tool collapses synthesized HFNs. It resenthesizes HFNs based on placement information and appropriately inserts buffer. Target of this synthesis is to meet delay requirements i.e. setup and hold.

For **clock** no synthesis is carried out in front end (why.....????..because no placement information of flip-flops ! So synthesis won't meet true skew targets !!) ... in backend clock tree synthesis tries to meet "skew" targets...It inserts clock buffers (which have equal rise and fall time, unlike normal buffers !)... There is no skew information for any HFNs.

Is it possible to have a zero skew in the design?

Theoretically it is possible....!

Practically it is impossible....!!

Practically we cant reduce any delay to zero.... delay will exist... hence we try to make skew "equal" (or same) rather than "zero"......now with this optimization all flops get the clock edge with same delay relative to each other.... so virtually we can say they are having "zero skew " or skew is "balanced".

13. How will you handle bi-directional pins during CTS?
14. Handling of high fan out nets like clock ,scan enable and reset pins
15. How to Build the clock tree for Hard macros ?
16. If you want to synchronize a nonclock pin—such as a combination logic gate's pin, a macro's pin, or a sequential gate's set/reset pin—you must define it as a synchronous pin explicitly.
17. Explain all CT topology? Which topology will you prefer for ur design?

- H –Tree
- Single Fish born
- Double fish born

18. What is the difference between CLOCK throughput and Offset?
19. What is Amoeba placement ? what its use?

7. Clocktree Optimization

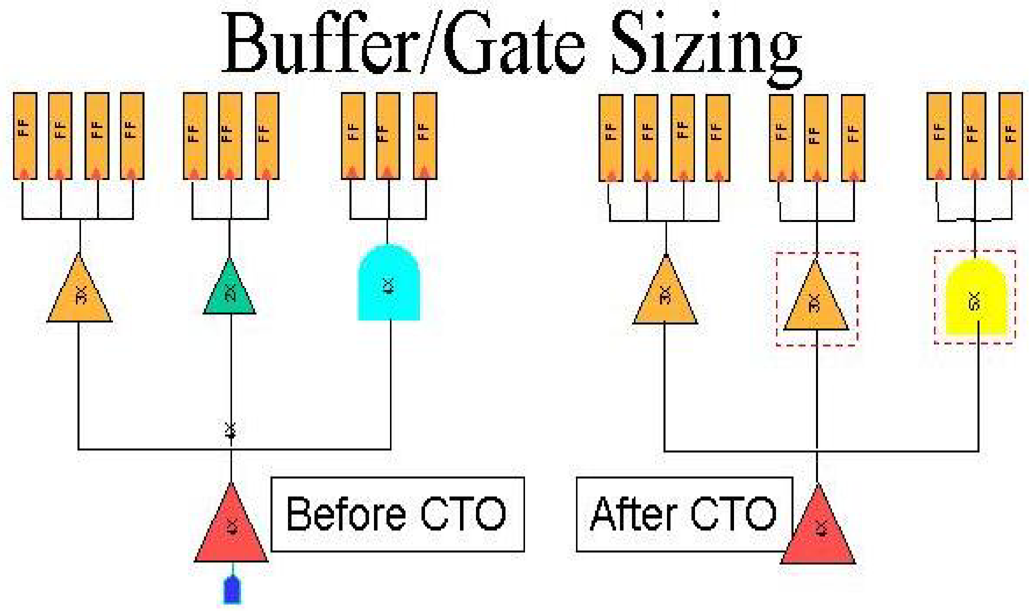
1. What kind of optimizations done in CTO?

The different options in CTO to reduce skew are described in the following list

Buffer and Gate Sizing

- Sizes up or down buffers and gates to improve both
- skew and insertion delay.
- You can impose a limit on the type of buffers and gates
- to be used.
- No new clock tree hierarchy will be introduced during
- this operation. See figure 1.

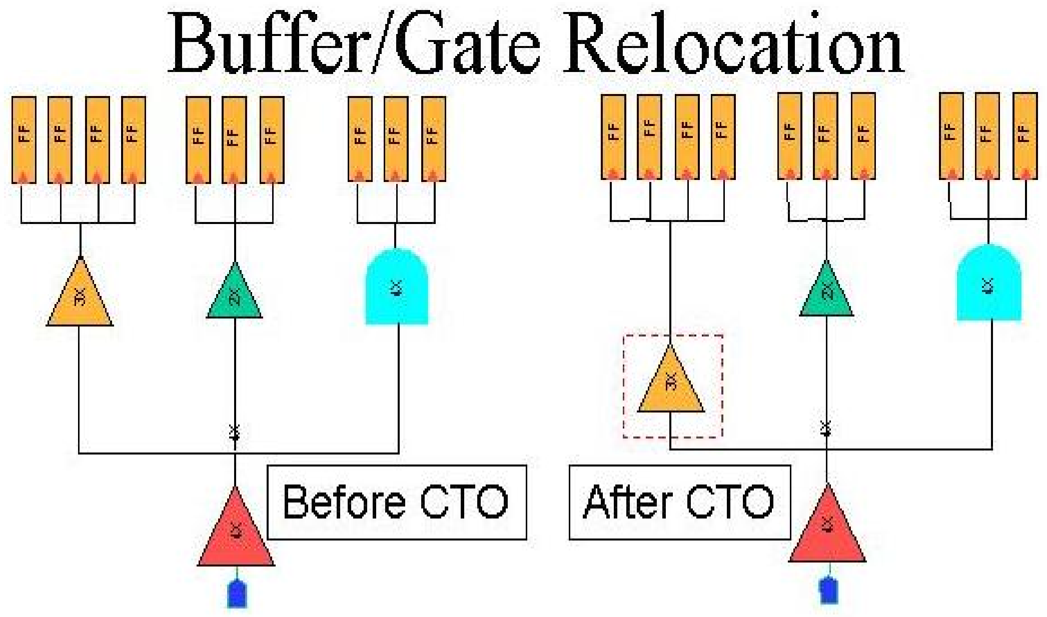
Figure 1



2. Buffer and Gate Relocation

- Physical location of the buffer or gate is moved
- to reduce skew and insertion delay.
- No new clock tree hierarchy will be introduced during
- this operation.
- See figure 2.

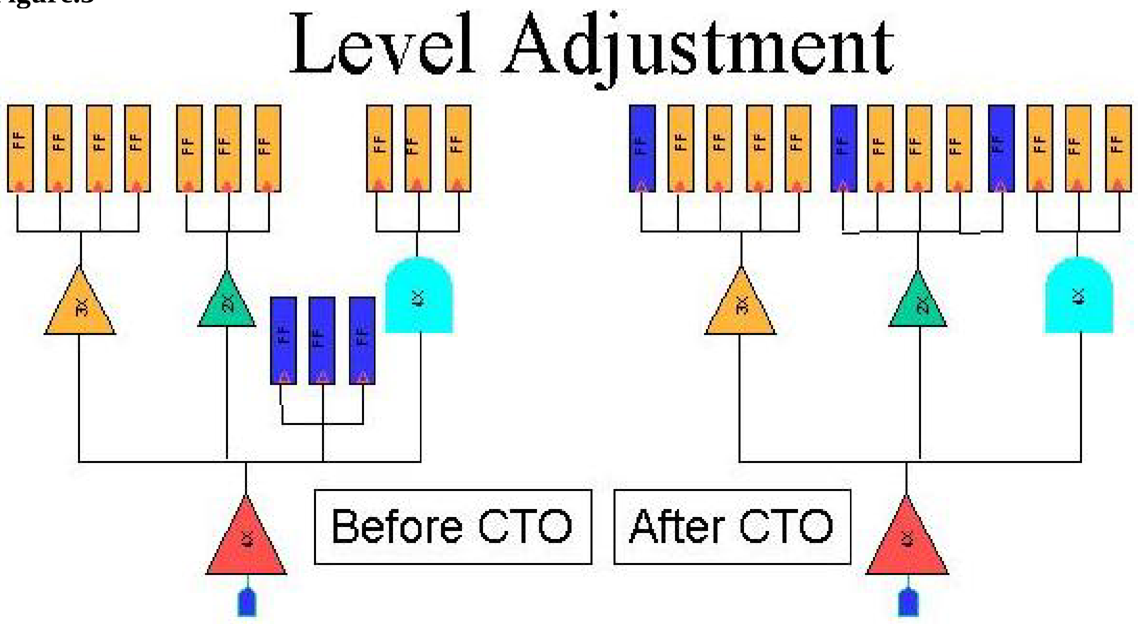
Figure 2



3. Level Adjustment

- o Adjust the level of the clock pins to a higher or
- o lower part of the clock tree hierarchy.
- o No new clock tree hierarchy will be introduced during
- o this operation.
- o See figure 3.

Figure.3

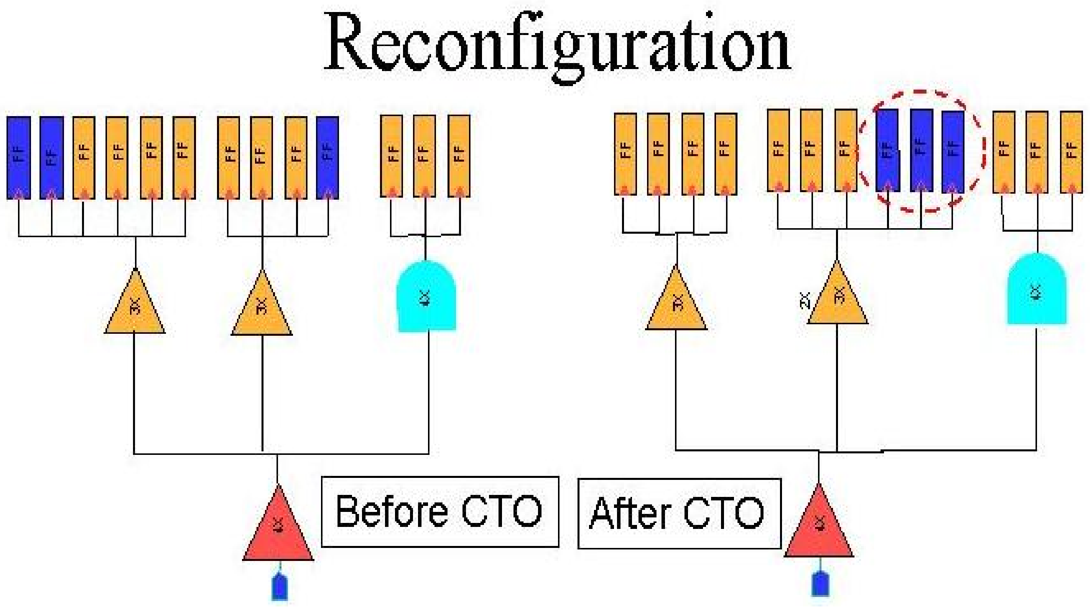


4. Reconfiguration

- o Clustering of sequential logic.

- o Buffer placement is performed after clustering.
- o Longer runtimes.
- o No new clock tree hierarchy will be introduced during
- o this operation.
- o See figure 4.

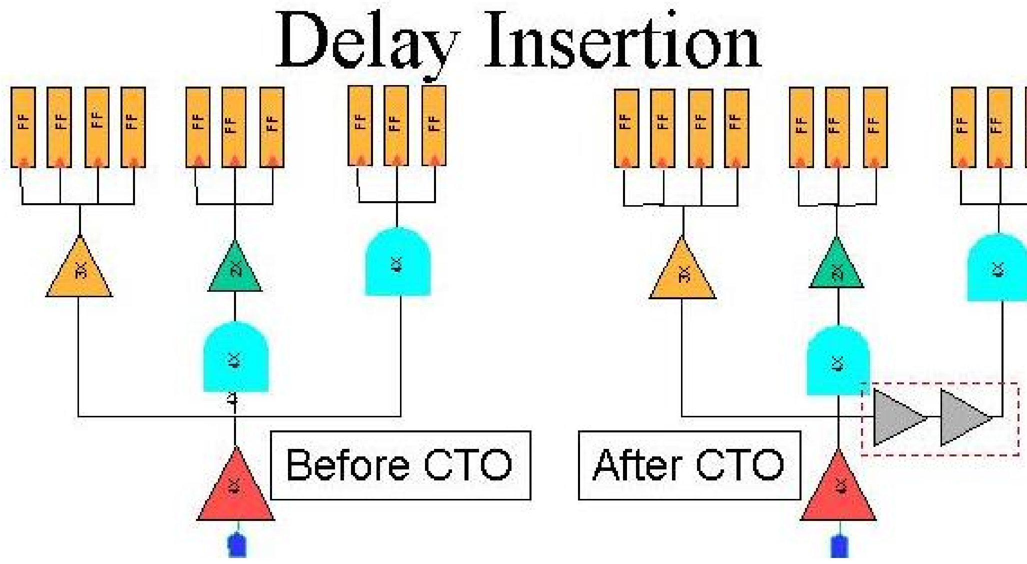
Figure 4



5. Delay Insertion

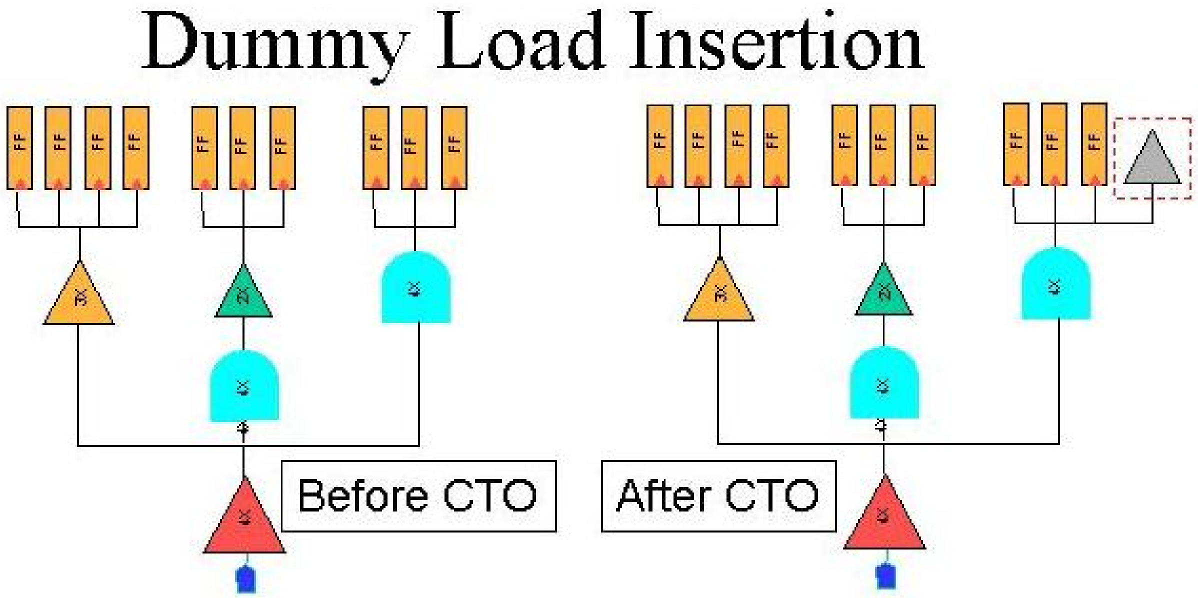
- o Delay is inserted for shortest paths.
- o Delay cells can be user defined or can be extracted
- o from by the tool.
- o By adding new buffers to the clock path the clock
- o tree hierarchy will change.
- o See figure 5.

Figure 5



6. Dummy Load Insertion
o Uses load balancing to fine tune the clock
o skew by increasing the shortest path delay.
o Dummy load cells can be user defined or can be
o extracted by the tool.
7. No new clock tree hierarchy will be introduced during this operation.
o See figure 6.

Figure 6



Routing

Routing flow is shown in the Figure (1).

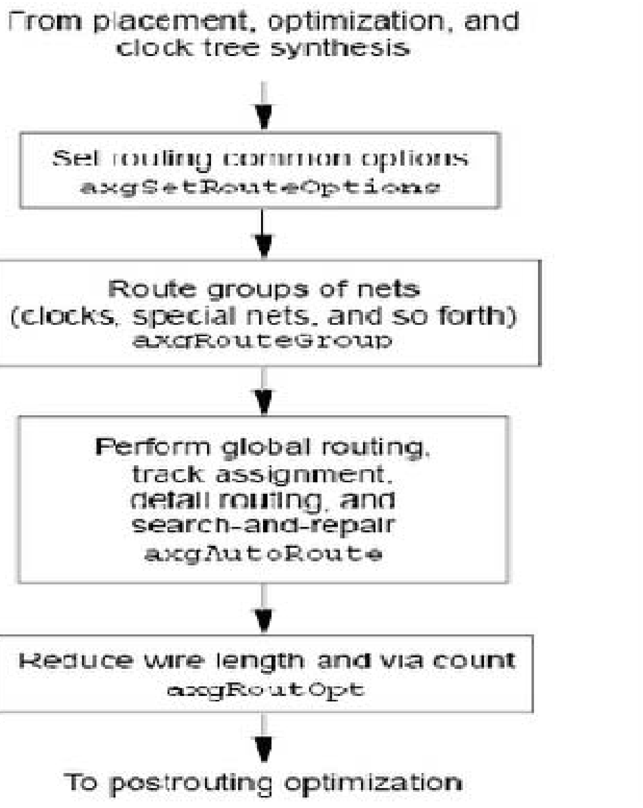

From placement, optimization, and clock tree synthesis
Set routing common options
axgSetRouteOptions
Route groups of nets (clocks, special nets, and so forth)
axgRouteGroup
Perform global routing, track assignment, detail routing, and search-and-repair
axgAutoRoute
Reduce wire length and via count
axgRoutOpt
To postrouting optimization

Figure (1) Routing flow [1]

Routing is the process of creating **physical connections based on logical connectivity**. Signal pins are connected by routing metal interconnects. Routed metal paths must **meet timing, clock skew, max trans/cap requirements** and also physical DRC requirements.

In grid based routing system each metal layer has its own tracks and preferred routing direction which are defined in a unified cell in the standard cell library.

There are four steps of routing operations:

1. **Global routing**

2. Track assignment

2. **Detail routing**
3. **Search and repair**

Global Route assigns nets to specific metal layers and global routing cells. Global route tries to avoid congested global cells while minimizing detours. Global route also avoids pre-routed P/G, placement blockages and routing blockages.

Track Assignment (TA) assigns each net to a specific track and actual metal traces are laid down by it. It tries to make long, straight traces to avoid the number of vias. DRC is not followed in TA stage. TA operates on the entire design at once.

Detail Routing tries to fix all DRC violations after track assignment using a fixed size small area known as "SBox". Detail route traverses the whole design box by box until entire routing pass is complete.

Search and Repair fixes remaining DRC violations through multiple iterative loops using progressively larger SBox sizes.

What is congestion?

- If the number of routing tracks available for routing is less than the required tracks then it is known as congestion.

8. Static Timing Analysis – STA

Timing Analysis in Physical Design

Timing analysis at back end requires knowledge of all clock related constraints provided at front end. When .sdc file given to physical design tool (like Astro) its first object is to remove all **Wire Load Models (WLM)** which are used for front end timing analysis. In backend there is no term called as wire load model. Actual delays are calculated based on the RC value of metal layers. All RC values like sidewall, junction and fringe capacitances are stored as **Table Look Up (TLU)** format in technology file.

In backend design hold violation has higher priority compared to setup violation because hold violation is related to data path of the design. Setup violation can be eliminated by slowing down the clock.

Placement and routing goal is always to meet timing constraints provided by the .sdc file. If **latency and uncertainty** are not set for clock at front end then at backend doing Clock Tree Synthesis (CTS) is not possible.

Cell delay and net delay are stored as look up table.

Cell delay consists of **transition, timing arcs and capacitances** while net delay is constituted by RCs only. Cell delays are available in libraries.

Net delays are specified in technology files. (In front end it is in WLM). Cell delays are fixed. Net delays are not fixed and they depend on interconnect length and width. Net delay parameters Rnet and Cnet are available as Table Look Up (TLU) provided by the vendor.

There is one more set of file TLU+ which account for Ultra Deep Sub Micron (UDSM) effects. UDSM effects are not included in TLU file. A mapping file maps TLU to TLU+. UDSM effects like **Optical Proximity Correction (OPC), Resumption Enhanced Technology (RET) and Litho Compliance Check (LCC)** are not taken care by Astro. For the placement stage virtual RC (based on Manhattan distance) **Layout Parasitic Extraction (LPE)** mode is used. For CTS real R and virtual C is used and for routing Real RC is used.

Clock definition given to SAMM in front end design flow is generated as .sdc file from Design Compiler is given below. It includes clock frequency, rise and fall time, setup and hold, skew and insertion delay.

```
###################################################
# Created by Design Compiler write_sdc on Fri May 11 18:35:45 2007
###################################################
```

```
create_clock -period 4.85 -waveform {0 2.425} [get_ports {clock}]
set_clock_transition -rise 0.04 [get_clocks {clock}]
set_clock_transition -fall 0.04 [get_clocks {clock}]
set_clock_uncertainty 0.485 -setup [get_clocks {clock}]
set_clock_uncertainty 0.27 -hold [get_clocks {clock}]
set_clock_latency 0.45 [get_clocks {clock}]
set_clock_latency -source 0.45 [get_clocks {clock}]
```

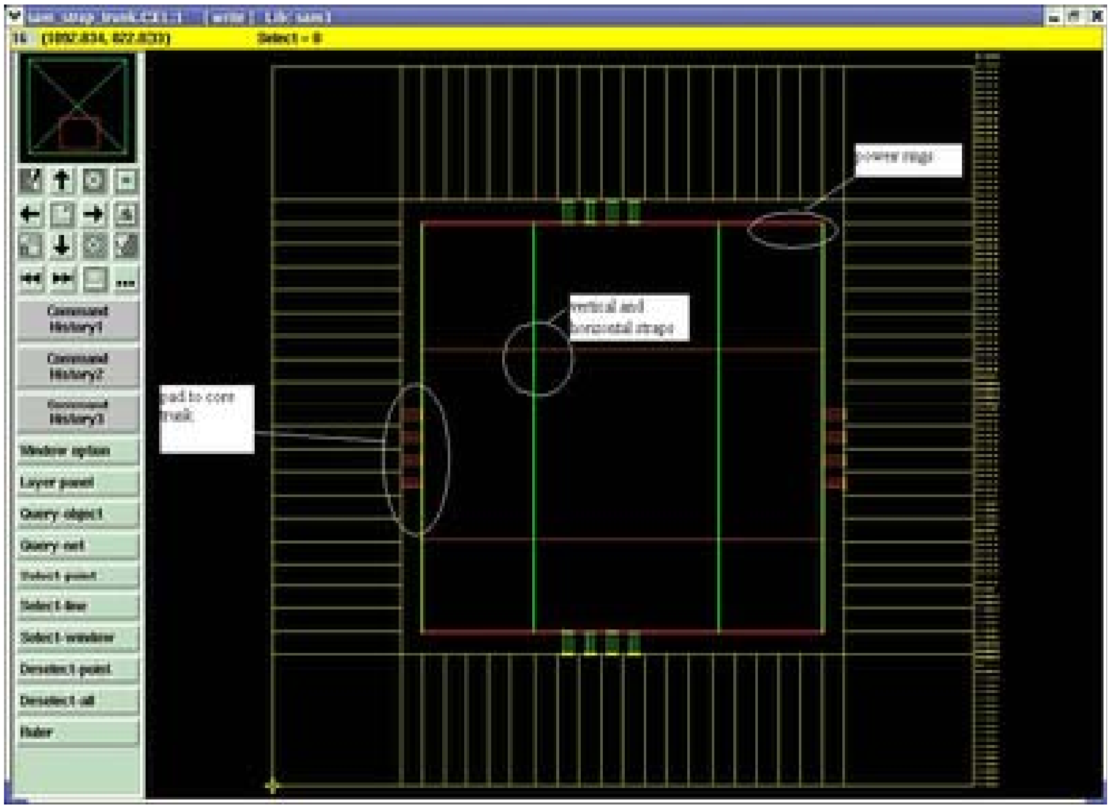


Figure (2) Showing core power ring, Straps and Trunks

Why am I getting violation inside Block (which has closed in block level) during top level Timing Analysis?

The input transition on the clock input affects the access time and setup-time of the flops clocked by it. Choosing a fixed value of input clock transition in sub-chip level STA is optimistic and can cause block internal violations in full-chip flat STA if the actual transition turns out to be different Also because of OCV effects in the upstream logic, even the actual transition value can have a min/max range. Even though CRPR removes any delay variation in the common portion of the clock path, the transition variation at the split-point is however not compensated. This small variation in clock-transition can result in increased clock skew, causing sub-chip level violations when analyzed at the top-level. Such violations are not fixable at the top-level and needs fixing at the sub-chip only.

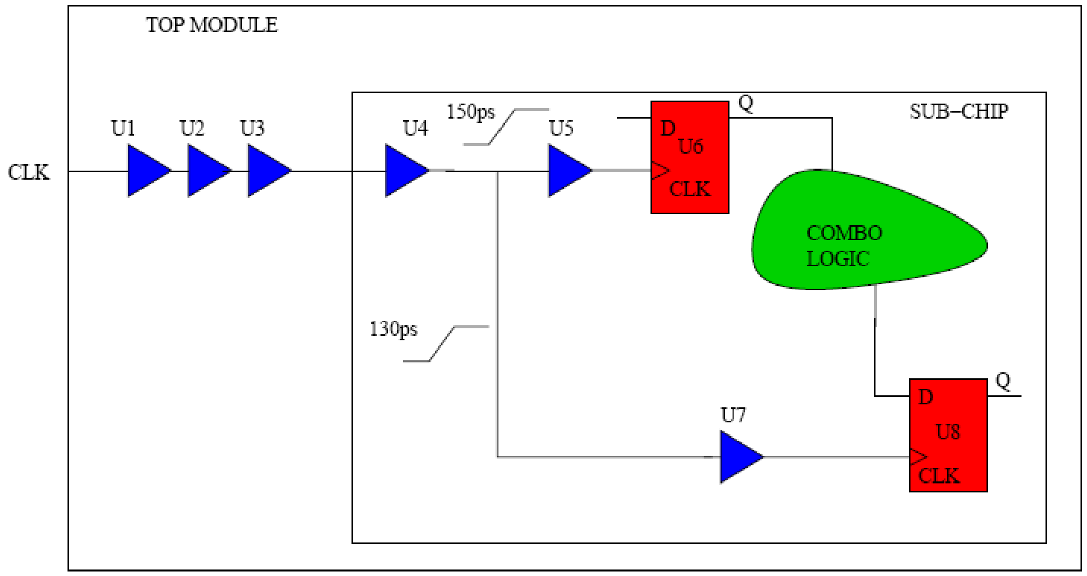


Figure 2: Example of clock input transition variation

In Figure 2, the point U4/Y is the clock split-point. Also the maximum transition of the clock at that point is 150 ps while the transition for the min delay type is 130 ps. CRPR compensates for the difference in delays accumulated till that point. However, the subsequent delay variation because of small transition time difference is not compensated for and the resultant clock-skew can cause additional timing violations inside the sub-chip. Note that this behavior is pessimistic in PT but in the absence of any automated technique in the tool to remove this, it is important to specify a mi-max transition window in sub-chip level STA.

Typically the maximum transition value is set to the transition target that is going to be used in synthesizing the clock network of the chip while the minimum transition value is typically set to the transition seen at the output of a strong clock-buffer driving a single load cell.

Internal Sub-chip violations due to rise/fall clock

Half-cycle paths inside a sub-chip can start violating in full-chip STA analysis if proper budgeting is not done to take care of below:-

1) Duty-cycle distortion of sub-chip input clock

2) Rise/Fall transition variation of sub-chip input clock

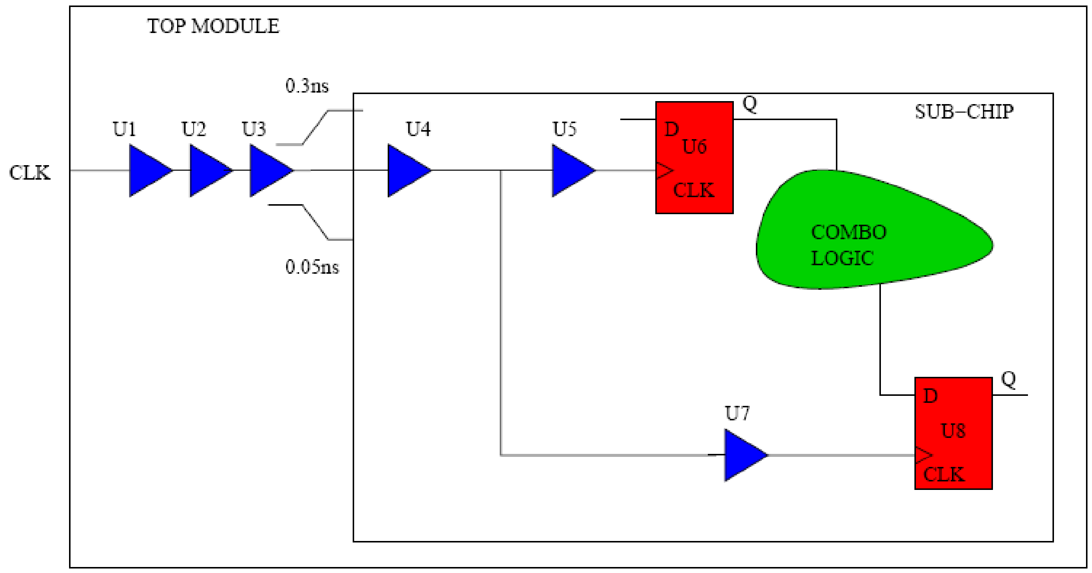


Figure 3: Example of sub-chip input clock non-idealities

In Figure 3, the top-level input clock (CLK) reaching the sub-chip has a duty cycle of 55%. If this actual duty-cycle at the sub-chip input clock pin is different and pessimistic than what was assumed during sub-chip level analysis, this could cause half-cycle path violations internal to the sub-chip. Similarly, differences in rise and fall transition of the sub-chip input clock can further deteriorate the duty-cycle inside the sub-chip. Note this also affects the timing windows of nets that are clocked by the input clock and can cause new crosstalk violations if not properly modeled.
Solution: Duty cycle distortion effect can be taken care of in PT by modifying the input clock waveform at the sub-chip level. However this method has two disadvantages:

(1) During sub-chip closure it is not known whether the high-time of the clock is more than the low-time or vice-versa.

(2) Optimistic for PT-SI analysis where timing window changes with changing duty cycle.

One solution is to expand the latency window of the sub-chip input clock fall edge, which has been discussed in section 4.2, by the maximum allowed duty cycle distortion (for example 10% of the clock period) in both directions. Figure 4 described one such example.

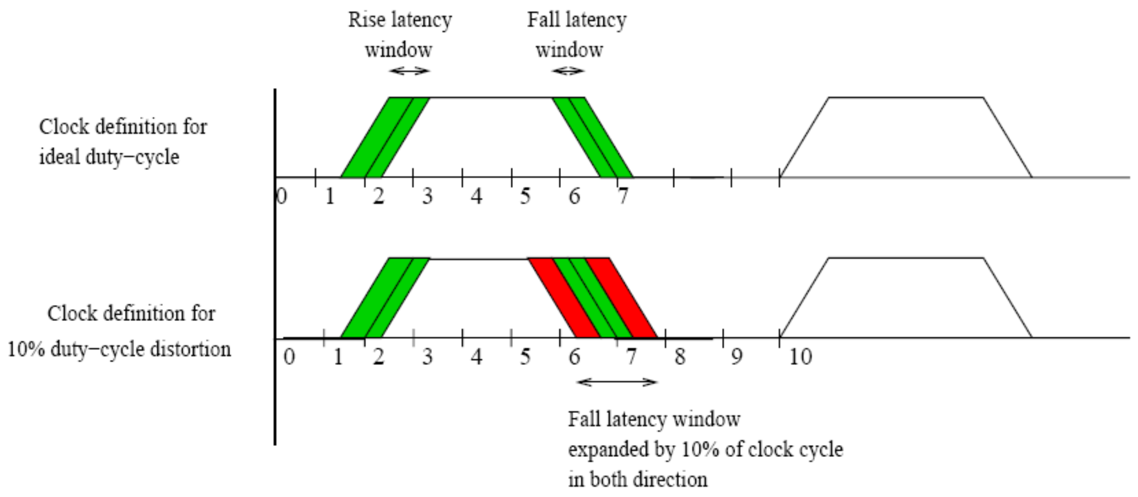

Rise latency window
Fall latency window
Clock definition for ideal duty−cycle
0 1 2 3 4 5 6 7
Clock definition for 10% duty−cycle distortion
0 1 2 3 4 5 6 7 8 9 10
Fall latency window expanded by 10% of clock cycle in both direction

Figure 4: Clock definition to incorporate duty-cycle distortion

Modeling arrival time of sub-chips inputs
For PT-SI analysis, the aggressors are pruned on the basis of overlap between the timing windows of the victim and aggressor signals. The min-max input delay assigned to the sub-chip input is taken as the timing window for PT-SI analysis. In top-level analysis, if the arrival window at a block input pin is different than what is being assumed during block-level closure, it can cause SI related violations inside the block. Figure 4 shows the example of an input port, which was constrained to a min-max input delay of 3 ns and 7 ns respectively. However at the top-level, the same input ended up having a timing window of 5 ns (higher than the budgeted timing window of 4 ns).

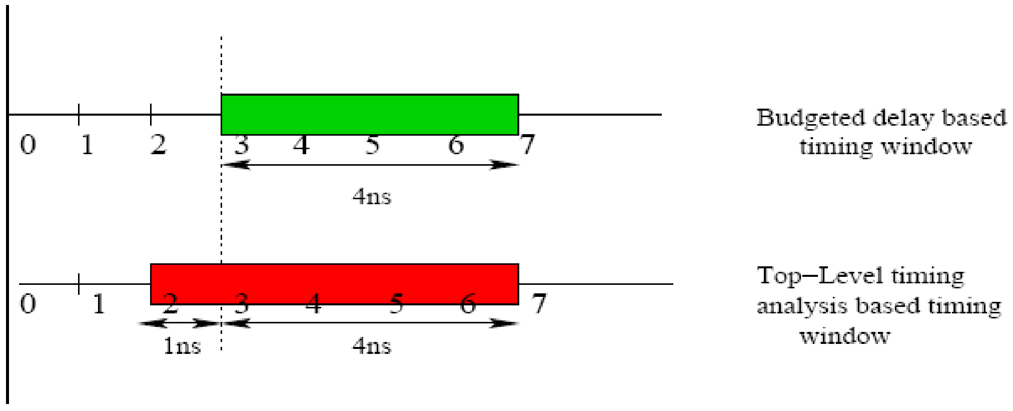

0 1 2 3 4 5 6 7
4ns
Budgeted delay based timing window
0 1 2 3 4 5 6 7
1ns
4ns
Top−Level timing analysis based timing window

Figure 5: Input timing window change

The electro migration verification requires average, peak and/or RMS current density information. If the current in the metal flows in one direction, failures are mainly caused by material transportations, and the **average current** should be used for **EM check**. On the other hand, if the **current** flows **bi-directionally**, thermal effect is the main cause for failures, and the **RMS current** should be used.

Operating Condition Analysis Modes

Semiconductor device parameters can vary with conditions such as fabrication process, operating temperature, and power supply voltage. In PT, the `set_operating_conditions` command specifies the operating conditions for analysis, so that PT can use the appropriate set of parameter values in the technology library.

PT offers three analysis modes with respect to operating conditions, called the single, best-case/worst-case, and on-chip variation modes:

- In the single operating condition mode, PT uses a single set of delay parameters for the whole circuit, based on one set of process, temperature, and voltage conditions.
- In the best-case/worst-case mode, PT simultaneously checks the circuit for the two extreme operating conditions, minimum and maximum. For setup checks, it uses maximum delays for all paths. For hold checks, it uses minimum delays for all paths. This mode lets you check both extremes in a single analysis run, thereby reducing overall runtime for a full analysis.
- In the on-chip variation mode, PT performs a conservative analysis that allows both minimum and maximum delays to apply to different paths at the same time. For a setup check, it uses maximum delays for the launch clock path and data path, and minimum delays for the capture clock path. For a hold check, it uses minimum delays for the launch clock path and data path, and maximum delays for the capture clock path.

Table 11-1 and Table 11-2 show the clock arrival times, delays, operating conditions, and delay derating used for setup checks and for hold checks under each of the operating condition analysis modes.

Table 11-1 Timing Parameters Used for Setup Checks

Analysis mode	**Launch clock path**	**Data path**	**Capture clock path**
Single operating condition	Late clock, maximum delay in clock path, single operating cond. (no derating)	Maximum delay, single operating cond. (no derating)	Early clock, minimum delay in clock path, single operating cond. (no derating)
Best-case/worst-case mode	Late clock, maximum delay in clock path, late derating, worst-case operating cond.	Maximum delay, late derating, worst-case operating cond.	Early clock, minimum delay in clock path, early derating, worst-case operating cond.
On-chip variation mode	Late clock, maximum delay in clock path, late derating, worst-case	Maximum delay, late derating, worst-case operating cond.	Early clock, minimum delay in clock path, early derating, best-case

	operating cond.		operating cond.

Table 11-2 Timing Parameters Used for Hold Checks

Analysis mode	**Launch clock path**	**Data path**	**Capture clock path**
Single operating condition	Early clock, minimum delay in clock path, single operating cond. (no derating)	Minimum delay, single operating cond. (no derating)	Late clock, maximum delay in clock path, single operating cond. (no derating)
Best-case/worst-case mode	Early clock, minimum delay in clock path, early derating, best-case operating cond.	Minimum delay, early derating, best-case operating cond.	Late clock, maximum delay in clock path, late derating, best-case operating cond.
On-chip variation mode	Early clock, minimum delay in clock path, early derating, best-case operating cond.	Minimum delay, early derating, best-case operating cond.	Late clock, maximum delay in clock path, late derating, worst-case operating cond.

Min-Max Delay Calculations

The `set_operating_conditions` command defines the operating conditions for timing analysis and specifies the analysis type: single, best-case/worst-case, or on-chip variation. The operating conditions must be defined in a specified library or pair of libraries.

By default, PT performs analysis under one set of operating conditions at a time (single operating condition mode). Using this mode, you need to perform multiple analysis runs to handle multiple operating conditions. Typically, you need to analyze at least two operating conditions to ensure that the design has no timing violations: best case (minimum path report) for hold checks and worst case (maximum path report) for setup checks.

PT can simultaneously perform timing analysis for the best-case and worst-case operating conditions. Compared to running single operating condition analysis, you can save time by using minimum and maximum (min-max) analysis in a single timing analysis run to report timing paths at the two extremes of operating conditions. You can choose either best-case/worst-case analysis or on-chip variation analysis.

In the best-case/worst-case mode, each setup check uses delays computed at the maximum operating condition and each hold check uses delays computed for at the minimum operating condition.

In the on-chip variation mode, PT performs a conservative analysis that allows both minimum and maximum delays to apply to different paths at the same time. For setup checks, it uses maximum delays for the launch clock path and data path, and minimum delays for the capture clock path. For hold checks, it uses minimum delays for the launch clock path and data path, and maximum delays for the capture clock path.

In the on-chip variation mode, when a path segment is shared between the clock paths that launch and capture data, the path segment might be treated as having two different delays at the same time. Not accounting for the shared path segment can result in a pessimistic analysis. For a more accurate analysis, this pessimism can be corrected. For more information, see "Clock Reconvergence Pessimism Removal".

A min-max analysis considers the minimum and maximum values specified for the following design parameters:

- Input and output external delays
- Delays annotated from Standard Delay Format (SDF)
- Port wire load models
- Port fanout number
- Net capacitance
- Net resistance
- Net wire load model
- Clock latency
- Clock transition time
- Input port driving cell

For example, to calculate a maximum delay, PT uses the longest path, worst-case operating conditions, latest-arriving clock edge, maximum cell delays, longest transition times, and so on.

You can do min-max analysis using a single library with minimum and maximum operating conditions specified, or two libraries, one for the best-case conditions sand one for the worst-case conditions. For more information, see "Using Two Libraries for Analysis".

You can enable minimum and maximum analysis in one of these two ways:

- Set the minimum and maximum operating conditions with the `set_operating_conditions` command:
  ```
  pt_shell> set_operating_conditions -min BCCOM -max WCCOM
  ```
- Read in an SDF file, then use the option of the SDF reader to read both the minimum and maximum delay from the SDF file:
  ```
  pt_shell> read_sdf -analysis_type bc_wc my_design.sdf
  ```

Min-Max Cell and Net Delay Values

Each timing arc in the design can have a minimum and a maximum delay to account for variations in operating conditions. You can specify these values in either of the following ways:

- Annotate delays from one or two SDF files
- Have PT calculate the delay

To annotate delays from one or two SDF files, do one of the following:

- Use min and max from the SDF triplet.
- Use two SDF files.
- Use either of the preceding choices, with additional multipliers for maximum and minimum value.

To have PT calculate the delay, do one of the following:

- Use a single operating condition with timing derating factors to model variation.
- Use two operating conditions (best-case and worst-case) to model the possible on-chip variation.
- Use either of the preceding choices with timing derating factors for the minimum and maximum value.
- Use separate derating factors for cells versus nets.
- Use separate derating factors for different timing checks (setup, hold, and so forth).

Table 11-3 summarizes the usage of minimum and maximum delays from SDF triplet data.

Table 11-3 Min-Max Delays From SDF Triplet Data

Analysis mode	**Delay based on operating conditions**	**One SDF file**	**Two SDF files**
Single operating condition	Setup - max data at operating condition - min capture clock at operating cond. Hold - min data at operating condition - max capture clock at operating cond.	Setup - (a:b:***c***) - (a:b:***c***)Hold - (a:b:***c***) - (a:b:***c***)	
Best case–worst case	Setup - max data at worst case - min capture clock at worst caseHold - min data at best case - max capture clock at best case	Setup - (a:b:***c***) - (a:b:***c***)Hold - (***a***:b:c) - (***a***:b:c).	Setup SDF1 - (a:b:***c***) SDF2 - (a:b:***c***)Hold SDF1 - (***a***:b:c) SDF2 - (***a***:b:c)
On-chip variation	Setup - max data at worst case - min capture clock at best caseHold - min data best case - max capture clock at worst case	Setup - (a:b:***c***) - (***a***:b:c)Hold - (***a***:b:c) - (a:b:***c***)	Setup SDF1 - (a:b:***c***) SDF2 - (***a***:b:c)Hold SDF1 - (***a***:b:c) SDF2 - (a:b:***c***)
The triplet displayed in ***bold italic*** is the triplet that PT uses.			

Setup and Hold Checks

This section provides examples of how setup and hold timing checks are done for a single operating condition, for simultaneous best-case/worst-case operating conditions, and for on-chip variation.

Path Delay Tracing for Setup and Hold Checks

Figure 11-1 shows how setup and hold checks are done in PT.

Figure 11-1 Design Example

- The setup timing check from pin DL2/ck to DL2/d considers
    - Maximum delay for clock_path1
    - Maximum delay for data path (data_path_max)
    - Minimum delay for clock_path2
- The hold timing check from pin DL2/ck to DL2/d considers
    - Minimum delay for clock_path1
    - Minimum delay for data path (data_path_min)
    - Maximum delay for clock_path2

The data_path_min and data_path_max values can be different due to multiple topological paths in the combinational logic that connects DL1/q to DL2/d.

Setup Timing Check for Worst-Case Conditions

Figure 11-2 shows how cell delays are computed for worst-case conditions. To simplify the example, the net delays are ignored.

Figure 11-2 Setup Check Using Worst-Case Conditions

PT checks for a setup violation as follows

In the equation,

clockpath1 = 0.8 + 0.6 = 1.4 datapathmax = 3.8 clockpath2 = 0.8 + 0.65 = 1.45 setup = 0.2

The clock period must be at least 1.4 + 3.8 – 1.45 + 0.2 = 3.95.

Hold Timing Check for Best-Case Conditions

Figure 11-3 shows how cell delays are computed for best-case conditions. Note that the cell delays are different from the delays in Figure 11-2.

Figure 11-3 Hold Check Using Best-Case Conditions

PT checks for a hold violation as follows:

In the equation,

clockpath1 = 0.5 + 0.3 = 0.8 datapathmin = 1.6 clockpath2 = 0.5 + 0.35 = 0.85 hold = 0.1

No hold violation exists because 0.8 + 1.6 – 0.85 – 0.1 = 1.45, which is greater than 0.

Simultaneous Best-Case/Worst-Case Conditions

PT can operate in a mode that computes delays simultaneously for best-case and worst-case conditions for hold and setup checks, enabling you to check both conditions in one timing analysis run. In terms of path delay tracing, PT uses worst-case conditions for setup checks and best-case conditions for hold checks. The timing reports for setup and hold checks are the same as for Figure 11-2 and Figure 11-3.

In simultaneous mode, PT does not compare data arrival at worst-case conditions to clock arrival at best-case conditions. In this mode, the timing reports show delays computed in the same operating condition (worst case for setup or best case for hold).

Path Tracing in the Presence of Delay Variation

In Figure 11-4, each delay of a cell or a net has an uncertainty because of on-chip variation. For example, you can specify that on-chip variation can be between 80 percent and 100 percent of the nominal delays for worst-case conditions. Figure 11-4 shows the resulting delays.

Figure 11-4   On-Chip Variation for Worst-Case Conditions

In this mode, for a given path, the maximum delay is computed at 100 percent of worst case, and the minimum delay is computed at 80 percent of worst case. PT checks for a setup violation as follows:

In the expression,

clockpath1 = 0.80 + 0.60 = 1.40 (at 100% of worst case) datapath_max = 3.80 (at 100% of worst case) clockpath2 = 0.64 + 0.52 = 1.16 (at 80% of worst case) setup = 0.2 The clock period must be at least 1.40 + 3.80 - 1.16 + 0.2 = 4.24

On-chip variation affects the clock latencies; therefore, you only need it when you are using propagated clock latency. If you specify ideal clock latency, you can have PT consider on-chip variation by increasing the clock uncertainty values with the `set_clock_uncertainty` command.

Specifying the Analysis Mode

The following examples demonstrate how to specify the operating condition analysis modes and how PT calculates the delays in each mode.

Single Operating Condition Analysis

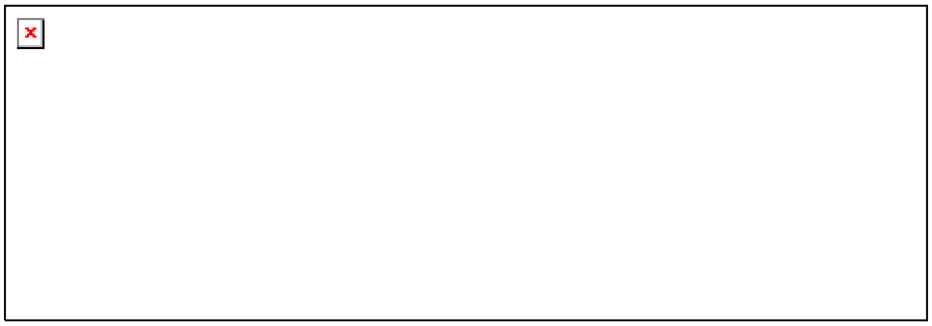
The commands for reporting one operating condition (best case or worst case) and the reported paths are shown in the following sections.

The commands for setting and reporting only the best-case operating conditions are

```
pt_shell> set_operating_conditions BEST
pt_shell> report_timing -delay_type min
```

Figure 11-5 shows the path reported.

Figure 11-5   Timing Path for One Operating Condition—Best Case

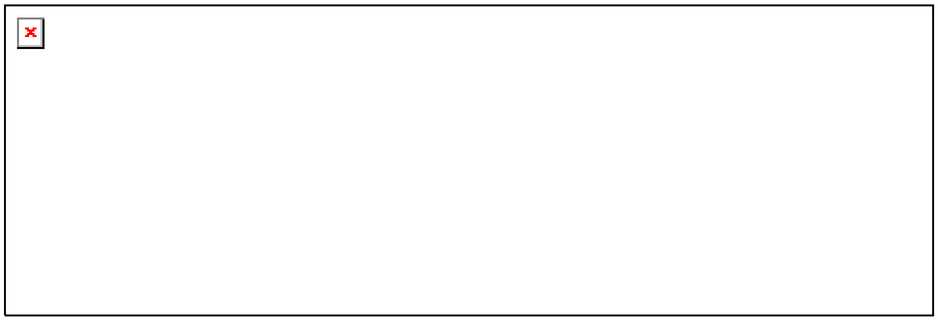


The commands for setting and reporting only the worst-case operating conditions are

```
pt_shell> set_operating_conditions WORST
pt_shell> report_timing -delay_type max
```

Figure 11-6 shows the path reported.

Figure 11-6   Timing Path for One Operating Condition—Worst Case



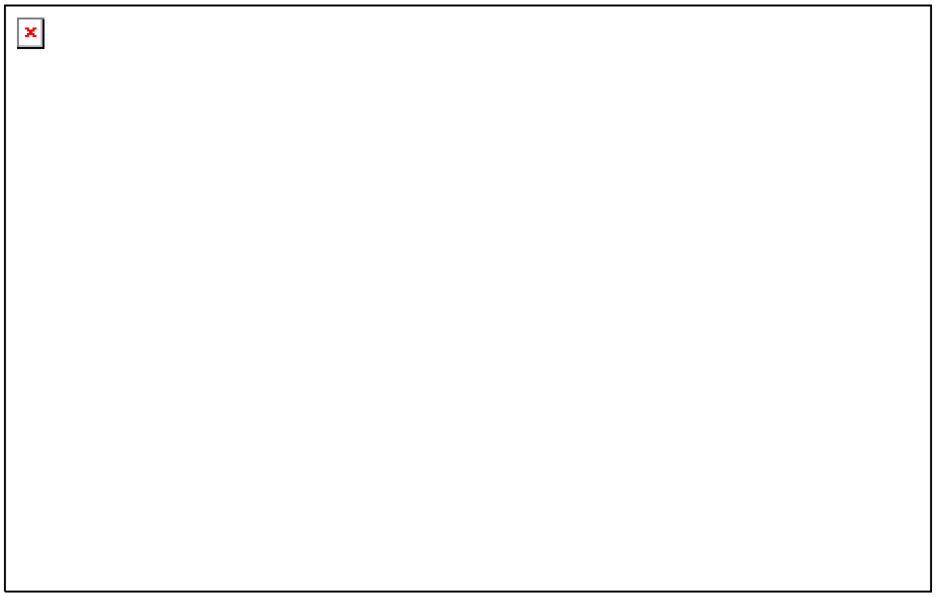
Best-Case/Worst-Case Analysis

The command sequence for setting and reporting simultaneous best-case and worst-case operating conditions is:

```
pt_shell> set_operating_conditions -min BEST -max WORST
pt_shell> report_timing -delay_type min
pt_shell> report_timing -delay_type max
```

Figure 11-7 shows the paths reported for the best case and the worst case when you set simultaneous best-case and worst-case operating conditions.

Figure 11-7 Timing Paths for Best-Case/Worst-Case Operating Conditions



On-Chip Variation Analysis

The command sequence for running on-chip variation analysis is

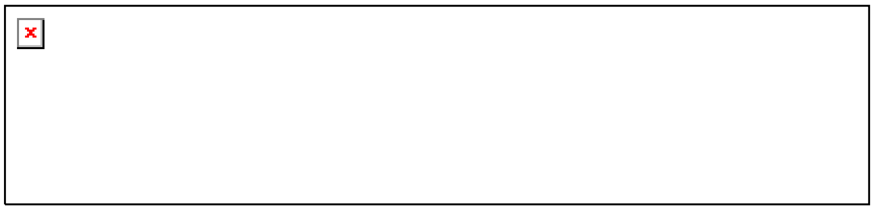
```
pt_shell> set_operating_conditions -analysis_type
on_chip_variation -min MIN -max MAX
pt_shell> report_timing -delay_type min
pt_shell> report_timing -delay_type max
```

Figure 11-8 shows the paths reported when you run on-chip variation analysis.

Figure 11-8 Timing Path Reported During On-Chip Variation Analysis

As shown in Figure 11-9, the early arrival time at the AND gate is 0.5 ns and the late arrival time is 1.5 ns, assuming the gate delay is zero.

Figure 11-9   Early and Late Arrival Time



Here are some additional on-chip variation examples.

Example 1

This command sequence performs timing analysis for on-chip variation 20 percent below the worst-case commercial (WCCOM) operating condition.

```
pt_shell> set_operating_conditions -analysis_type
on_chip_variation WCCOM
pt_shell> set_timing_derate -early 0.8
pt_shell> report_timing
```

Example 2

This command sequence performs timing analysis for on-chip variation between two predefined operating conditions: WCCOM_scaled and WCCOM.

```
pt_shell> set_operating_conditions -analysis_type
on_chip_variation -min WCCOM_scaled -max WCCOM
pt_shell> report_timing
```

Example 3

This command sequence performs timing analysis with on-chip variation. For cell delays, the on-chip variation is between 5 percent above and 10 percent below the SDF back-annotated values. For net delays, the on-chip variation is between 2 percent above and 4 percent below the SDF back-annotated values. For cell timing checks, the on-chip variation is 10 percent above the SDF values for setup checks and 20 percent below the SDF values for hold checks.

```
pt_shell> set_timing_derate -cell_delay -early 0.90
pt_shell> set_timing_derate -cell_delay -late 1.05
pt_shell> set_timing_derate -net -early 0.96
pt_shell> set_timing_derate -net -late 1.02
pt_shell> set_timing_derate -cell_check -early 0.80
pt_shell> set_timing_derate -cell_check -late 1.10
```

Using Two Libraries for Analysis

The `set_min_library` command directs PT to use two technology libraries simultaneously for minimum-delay and maximum-delay analysis. For example, you can choose two libraries that have the following characteristics:

- Best-case and worst-case operating conditions
- Optimistic and pessimistic wire load models
- Minimum and maximum timing delays

To perform an analysis of this type, use the `set_min_library` command to create a minimum/maximum association between two libraries. You specify one library to be used for maximum delay analysis and another to be used for minimum delay analysis. Only the maximum library should be present in the link path.

When you use the `set_min_library` command, PT first checks the library cell in the maximum library, and then looks in the minimum library to see if a match exists. If a library cell with the same name, the same pins, and the same timing arcs exists in the minimum library, PT uses that timing information for minimum analysis. Otherwise, it uses the information in the maximum library. For information about the `set_min_library` command, see the man page.

Setting Derating Factors

You can have PT adjust minimum delays and maximum delays by specified factors to model the effects of operating conditions. This adjustment of calculated delays is called derating. Derating affects the delay and slack values reported by the `report_timing` command and other reporting commands.

The `set_timing_derate` command specifies the adjustment factors and the scope of the design to be affected by derating. For example,

```
pt_shell> set_timing_derate -early -cell_delay 0.9
pt_shell> set_timing_derate -late -cell_delay 1.2
```

The first of these two commands causes all early (shortest-path) delays to be decreased by 10%, such as the capture clock path in a setup check. The second command causes all late (longest-path) delays to be increased by 20%, such as the launch clock path and the data path in a setup check. These changes result in a more conservative analysis than leaving delays at their original calculated values.

The `-early` or `-late` option specifies whether the shortest-path or longest-path delays are affected by the derating factor. Exactly one of these two options must be used in the command. To set both early and late derating, use two `set_timing_derate` commands. Early path delays include the capture clock path for a setup check, and the launch clock path and data path for a hold check. Late path delays include the launch clock path and data path for a setup check, and the capture clock path for a hold check.

The fixed-point value in the command specifies the derating factor applied to the delays. Use a value of less than 1.0 to reduce the delay of any given path or cell check. Similarly, use a derating factor greater than 1.0 to increase the delay of any given path or cell check. Typically, derating factors less than 1.0 are used for early (shortest-path) delays and greater than 1.0 for late (longest-path) delays. This approach results in a more conservative analysis.

The last derating value to be set overrides any previously set values. To reset the derating factor globally to 1.0, use the `reset_timing_derate command`.

Delays are adjusted according to the following formula:

delay_new = old_delay + ((derate_factor – 1.0) * abs(old_delay))

When the delay is a positive value (the usual case), this equation is reduced to:

delay_new = old_delay * derate_factor

For negative delay, the delay adjustment equation is reduced to:

delay_new = old_delay * (2.0 – derate_factor)

Delays can be negative in some unusual situations. For example, if the input transition is slow and the output transition is fast for a cell, the time at which the input signal reaches the 50% trip point can be later than the time at which the output signal reaches the 50% trip point. The resulting delay from input to output is negative. If you are using PT SI, a similar situation can occur for a net due to a change in delay caused by crosstalk.

The `-rise` or `-fall` option specifies whether rise delays or fall delays are affected by derating. If neither option is used, both types of delays are affected. The `-clock` or `-data` option specifies whether clock paths or data paths are affected by derating. If neither option is used, both types of paths are affected. A clock path is a path from a clock source to the clock input pin of a launch or capture register. A data path is a path starting from an input port or from the data output of a launch register, and ending at an output port or at the data input of a capture register.

The `-net_delay`, `-cell_delay`, and `-cell_check` options let you specify the functional scope of the design to be affected by derating. The `-net_delay` option derates net delays (the delay from a net driver to a net load). The `-cell_delay` option derates cell delays (the delay from a cell input to a cell output). The `-cell_check` option derates cell timing checks (cell hold and removal times for early derating, or cell setup and recovery times for late derating). You can use any combination of these three options. If you do not use any of these options, the derating factor applies to net delays and cell delays, but not cell timing checks.

If you want only some cells or nets to be affected by derating, you can list them in the command. The list can include library cells, hierarchical cells, leaf-level cells, and nets. In the absence of a list, the whole design is affected.

To set a derate factor on all nets within a hierarchy, use the `-net_delay` option. For example,

```
pt_shell> set_timing_derate -net_delay -early 0.8 [get_cells
hier_cell]
```

If you do not specify the `-net_delay` option, only the cells have the derate factor set on them. This feature allows you to specify different derate factors for delta net delays and non-delta net delays. To set a derate factor for non-delta net delays only use the `-static` option:

```
set_timing_derate -net_delay -static -early/late -data/clock $net
$value1
```

To set a derate factor for delta net delays only use the `-dynamic` option:

```
set_timing_derate  -net_delay  -dynamic  -early/late  -data/clock
$net $value2
```

If both `-static` and `-dynamic` options are omitted, the derate factor is applied to both delta and non-delta net delays.

Different sets of derate factors can be specified for a deterministic PT analysis and a variational PT analysis. To set a derate factor for deterministic delays, use the `-scalar` option. To set a derate factor for delays that have been computed using variational information, use the `-variation` option. For more information about using derate factors in a variation analysis, see the *PT User Guide: Advanced Timing Analysis*.If you set a derating factor on a net that crosses a hierarchical boundary, the derating affects the whole net, not just the part of the net listed in the command. Also, if you set a derating factor on a hierarchical cell, the derating factor extends to cells and nets within and below the hierarchy. PT issues a warning message when it extends derating across a hierarchical boundary.

In case of conflict between different `set_timing_derate` values specified for different levels of the design, the more detailed command (with the narrowest scope) has precedence. For example, the following commands have decreasing order of precedence:

```
pt_shell> set_timing_derate -late -cell_delay 1.4 [get_cells
inv*]
        # derates a collection of cells in the design
pt_shell> set_timing_derate -late -cell_delay 1.3 \
[get_lib_cells class/ND*]
        # derates cells in a collection of cells in a library
class
pt_shell> set_timing_derate -late -cell_delay 1.2
        # derates all cells
pt_shell> set_timing_derate -late -1.1
        # derates all nets and cells
```

In a hierarchical design, derating is applied to cells in the following order of precedence (from highest to lowest priority):

1. Leaf-level cell
2. Hierarchical cell (containing leaf-level cells)
3. Library cell
4. Design

PT stores and checks global timing derating factors against block scope. These stored values are checked to make sure that the top-level ranges are completely within the block-level range. For cases where a single value is used at the block level (that is, when the early derate is the same as the late derate), the top-level values must be the same as well. If derate values are not specified at either the block or top levels, the derate value used is 1.0 and the scope check is performed against this value.

If you use the `set_timing_derate` command while the analysis is set to single operating condition mode, PT automatically changes to the on-chip variation mode, like using this command:

```
pt_shell> set_operating_conditions -analysis_type
on_chip_variation
```

From this mode, you can change to the best-case/worst-case mode, if desired:

```
pt_shell> set_operating_conditions -analysis_type bc_wc
```

If the analysis mode is already set to best-case/worst-case, it stays in that mode when you use `set_timing_derate`.

If you use the `-derate` option with the `report_timing` command, the global, net-specific, or instance-specific derate factor is applied to each cell or net is reported in a column next to the incremental delay for each item.

To obtain a report on the current timing derating factors that have been set, use the `report_timing_derate` command. For example,

```
pt_shell> report_timing_derate
        # reports all derating settings
pt_shell> report_timing_derate [get_cells U*]
        # reports derating settings on all cells named U*
```

Use the `-include_inherited` option to include reporting of derating values inherited from other objects, such as a lower-level cell that inherits its derating values from a higher-level cell. Otherwise, the command reports only the derating values set directly on the specified object.

Clock Reconvergence Pessimism Removal

Clock reconvergence pessimism is an accuracy limitation that occurs when two different clock paths partially share a common physical path segment and the shared segment is assumed to have a minimum delay for one path and a maximum delay for the other path. This condition can occur any time that launch and capture clock paths use different delays, most commonly with on-chip variation analysis (see Table 11-4). Automated correction of this inaccuracy is called clock reconvergence pessimism removal CRPR.

PT performs clock reconvergence pessimism removal, when enabled, at the same time as regular timing analysis. You need to enable the clock reconvergence pessimism removal before you do timing analysis. The following examples demonstrate how the pessimism removal works.

On-Chip Variation Example

Consider the following command sequence for running on-chip variation analysis and the corresponding design in Figure 11-10:

```
pt_shell> set_operating_conditions -analysis_type
on_chip_variation -min MIN -max MAX
pt_shell> set_timing_derate -net -early 0.80
pt_shell> report_timing -delay_type min
pt_shell> report_timing -delay_type max
```

Figure 11-10 Clock Reconvergence Pessimism Example

Figure 11-10 shows how the analysis is done. Each delay (considered equal for rising and falling transitions to simplify this example) has a minimum value and a maximum value computed for the minimum and maximum operating conditions.

The setup check at LD2/CP considers the clock path to the source latch (CLK to LD1/CP) at 100 percent worst case, and the clock path to the destination latch (CLK to LD2/CP) at 80 percent worst case.

Although this is a valid approach, the test is pessimistic because clock path1 (CLK to LD1/CP) and clock path2 (CLK to LD2/CP) share the clock tree until the output of U1. The shared segment is called the *common portion*, consisting of just cell U1 in this example. The last cell output in the shared clock segment is called the *common point*, which is the output of U1 in this case.

The setup check considers that cell U1 simultaneously has two different delays, 0.64 and 0.80, resulting in a pessimistic analysis in the amount of 0.16. This amount, obtained by subtracting the earliest arrival time from the latest arrival time at the common point, is called the *clock reconvergence pessimism*.

This inaccuracy also occurs in an analogous way for the hold test at the LD2 latch.

Reconvergent Logic Example

Figure 11-11 shows a situation where clock reconvergence can occur, even in the absence of on-chip variation analysis. In this example, there is reconvergent logic in the clock network. The two clock paths that feed into the multiplexer cannot be active at the same time, but an analysis could consider both the shorter and longer paths for one setup or hold check.

Figure 11-11 Reconvergent Logic in a Clock Network

Minimum Pulse Width Checking Example

The `report_constraints` command checks for minimum pulse width violations in clock networks (as specified by the `set_min_pulse_width` command) and at cell inputs (as specified in the technology library). Clock reconvergence pessimism removal, when enabled, can increase the accuracy of minimum pulse width checking.

For example, consider the circuit shown in Figure 11-12. The external clock source has early and late source latency set on it. In addition, the two buffers in the path have minimum and maximum rise and fall delay values defined.

The `report_constraints` command checks the pulse width of the clock signal in the clock network and upon reaching the flip-flop. For level-high pulse width checking, PT considers maximum delay for the rising edge and minimum delay for the falling edge of the clock (and conversely for level-low pulse width checking).

For the example shown in Figure 11-12, in the absence of clock reconvergence pessimism removal, the worst-case pulse width is very small and violates the pulse width constraint of the flip-flop. However, this analysis is pessimistic because it assumes simultaneous worst-case delays for rising and falling edges. In a real circuit, rising-edge and falling-edge delays are at least somewhat correlated. For example, for the delay from the external clock source to the CLK input port, if the rising-edge delay is at the minimum, –1.3, the falling-edge delay is probably equal or close to –1.3, and not at the maximum of +1.4.

To account for correlation between rising-edge and falling-edge delays, enable clock reconvergence pessimism removal. In that case, PT adds a certain amount of slack back into the minimum pulse width calculation. The amount added is equal to the range of minimum rise delay or the range maximum fall delay for the path, whatever is smaller:

where *crp* = clock reconvergence pessimism, *Mr* = cumulative maximum rise delay, *mr* = cumulative minimum rise delay, *Mf* = cumulative maximum fall delay, and *mf* = cumulative minimum fall delay. For an example of this calculation applied to pulse width checking, see Figure 11-12.

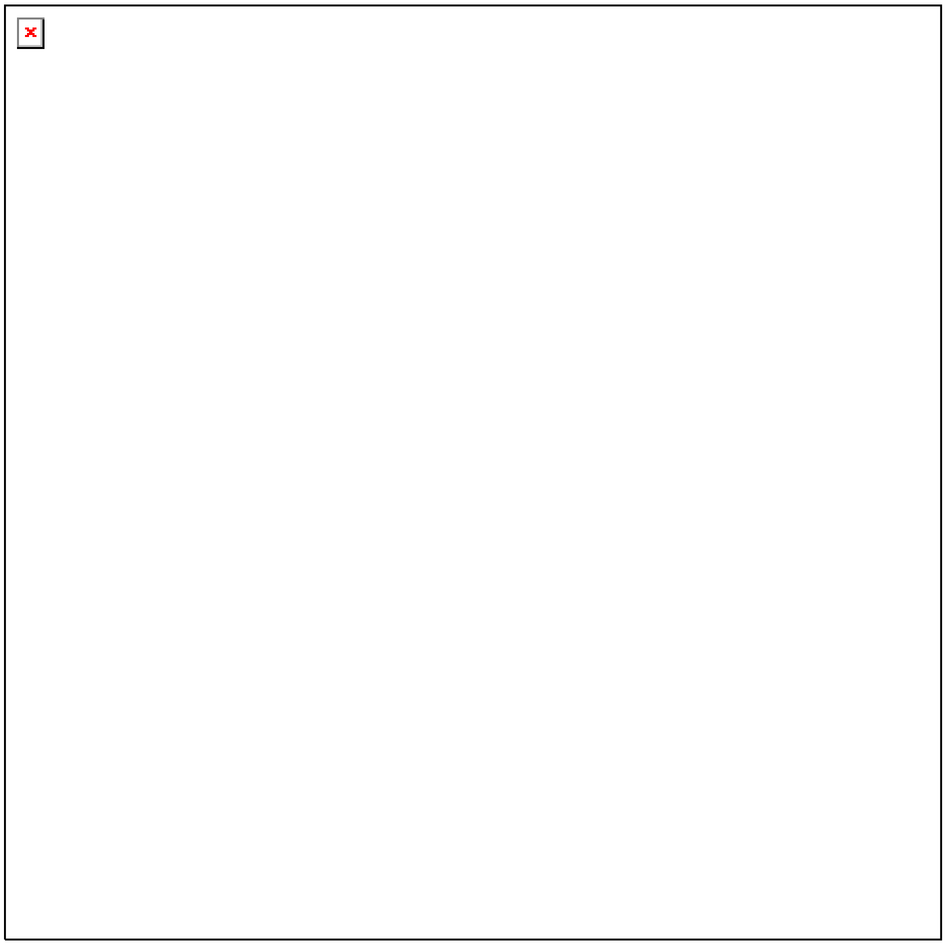


Figure 11-12 Minimum Pulse Width Analysis

Using CRPR Commands

To enable clock reconvergence pessimism removal, set the `timing_remove_clock_reconvergence_pessimism` variable to true before you begin timing analysis. By default, the setting is false and the algorithm is disabled. Using the algorithm results in a more accurate (less pessimistic) analysis, but causes an increase in runtime and memory usage. Any change in this variable setting causes a complete timing update, like the `update_timing -full` command.

When clock reconvergence pessimism removal is enabled, PT uses the correction algorithm and reports the results in all the major types of reports: `report_timing`, `report_constraint`, `report_analysis_coverage`, and `report_bottleneck_analysis`.

For the on-chip variation example shown in Figure 11-10, with clock reconvergence pessimism removal enabled, you might produce a report like this:

```
pt_shell> report_timing -delay_type max

****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : my_design
****************************************

Startpoint: LD1 (rising edge-triggered flip-flop clocked by
CLK)
  Endpoint: LD2 (rising edge-triggered flip-flop clocked by
CLK)
  Path Group: CLK
  Path Type: max

  Point                                    Incr       Path
  ---------------------------------------------------------
  clock CLK (rise edge)                    0.00       0.00
  clock network delay (propagated)         1.40       1.40
  LD1/CP (FD2)                             0.00       1.40 r
  LD1/Q (FD2)                              0.60       2.00 f
  U1/z (AN2)                               3.20       5.20 f
  data arrival time                                   5.20

  clock CLK (rise edge)                    6.00       6.00
  clock network delay (propagated)         1.16       7.16
clock reconvergence pessimism             0.16 7.32
  clock uncertainty                        0.00       7.32
  LD2/CP (FD2)                                        7.32 r
  library setup time                      -0.20       7.12
  data required time                                  7.12
  ---------------------------------------------------------
  data required time                                  7.12
  data arrival time                                  -5.20
  ---------------------------------------------------------
  slack (MET)                                         1.92
```

Pessimism can also be introduced on paths that fan out from a clock source to the data pin of a sequential device, such as the path shown in Figure 11-13 (technically, this would not be considered clock reconvergence pessimism because the launching path would be considered to be a data path, not a clock path).

Figure 11-13 Timing Path Fanout from Clock Source to Data Pin

To remove pessimism for these paths, set the timing_crpr_remove_clock_to_data_crp variable to `true` before you begin timing analysis.

Note: Calculating CRP for such paths involves analyzing the sequential device associated with each clock to data path separately in the timing analysis. This may cause a severe performance penalty during a timing update.

To specify whether to perform clock reconvergence pessimism removal on clock paths that have opposite-sense transitions in a shared path segment, set the timing_clock_reconvergence_pessimism variable. You can set this variable to `normal` (the default setting) or `same_transition`. For the default setting, PT still performs clock reconvergence pessimism removal with opposite-sense transitions in the shared path segment. In that case, if the two transition types produce different correction values, the smaller of the two values is used. For `same_transition`, PT performs clock reconvergence pessimism removal only if transitions in the shared path segment have the same sense.

Table 11-4 summarizes the application of either rising or falling clock reconvergence pessimism based on the transition types at the common point in the clock path and `timing_clock_reconvergence_pessimism` variable setting.

Table 11-4 Application of Rise/Fall CRP Value Based on Variable Setting

Transition types at common point in clock paths	**timing_clock_reconvergence_ pessimism = normal**	**timing_clock_reconvergence_ pessimism = same_transition**
Both rising	crp_rise pessimism removed	crp_rise pessimism removed
Both falling	crp_fall pessimism removed	crp_fall pessimism removed
One rising, one falling	smaller of crp_rise or crp_fall removed	No pessimism removed
Transition types at common point in clock paths	timing_clock_reconvergence_ pessimism = normal	timing_clock_reconvergence_ pessimism = same_transition

For the minimum pulse width checking example shown in Figure 11-12, you could set up the analysis with a script similar to this:

```
set_operating_conditions -analysis_type on_chip_variation
create_clock -period 8 CLK
set_propagated_clock [all_clocks]
set_clock_latency -source -early -1.3  CLK
set_clock_latency -source -late  1.4  CLK
set_annotated_delay -cell -from CT1/A -to CT1/Z 1
set_annotated_delay -cell -from CT2/A -to CT2/Z -min -rise 0.8
set_annotated_delay -cell -from CT2/A -to CT2/Z -max -rise 1.8
set_annotated_delay -cell -from CT2/A -to CT2/Z  -min -fall 0.85
set_annotated_delay -cell -from CT2/A -to CT2/Z -max -fall 2.03
```

With clock reconvergence pessimism removal enabled, a `report_constraint` report on the path would look like this:

```
Point                                   Incr       Path
-----------------------------------------------------------
clock CLK (rise edge)                   0.00       0.00
clock network delay (propagated)        4.20       4.20 r
FF1/CP                                  0.00       4.20 r
open edge arrival time                             4.20

clock CLK (fall edge)                   4.00       4.00
clock network delay (propagated)        0.55       4.55 f
FF1/CP                                  0.00       4.55 f
clock reconvergence pessimism 3.70 8.25
close edge arrival time 8.25
-----------------------------------------------------------
required pulse width (high)                        3.50
actual pulse width 4.05
-----------------------------------------------------------
slack 0.55
```

You can set a threshold that allows PT to ignore small amounts of clock reconvergence pessimism. For computational efficiency, PT merges multiple points for CRPR calculations when the CRP differences between adjacent points are smaller than the specified threshold. This merging of nodes can reduce CPU and memory usage, without a significant loss of accuracy when an appropriate threshold is set.

The `timing_crpr_threshold_ps` variable specifies the threshold. The units are picoseconds, irrespective of the time units of the technology library. By default, the variable is set to 20, which allows adjacent nodes to be merged when the difference in clock reconvergence pessimism is 20 ps or less.

For a good balance between performance and accuracy, try setting this variable to one-half the stage delay of a typical gate in the clock network. (The stage delay is gate delay plus net delay.) You might want to use a larger value during the design phase for faster analysis, and then a smaller value for sign-off accuracy.

CRPR and Crosstalk Analysis

When you perform crosstalk analysis using PT SI, a change in delay due to crosstalk along the common segment of a clock path can be pessimistic, but only for a zero-cycle check. A zero-cycle check occurs when the exact same clock edge drives both the launch and capture events for the path. For other types of paths, a change in delay due to crosstalk is not pessimistic because the change cannot be assumed to be identical for the launch and capture clock edges.

Accordingly, the CRPR algorithm removes crosstalk-induced delays in a common portion of the launch and capture clock paths only if the check is a zero-cycle check. In a zero-cycle check, aggressor switching affects both the launch and capture signals in the same way at the same time.

Here are some cases where the CRPR might apply to crosstalk-induced delays:

- Standard hold check
- Hold check on a register with the Q-bar output connected to the D input, as in a divide-by-2 clock circuit
- Hold check with crosstalk feedback due to parasitic capacitance between the Q-bar output and D input of a register
- Hold check on a multicycle path set to zero, such as circuit that uses a single clock edge for launch and capture, with designed-in skew between launch and capture
- Certain setup checks where transparent latches are involved

CRPR with Dynamic Clock Arrivals

A similar scenario to CRPR with crosstalk can occur if dynamic annotations have been set in the clock network. Dynamic annotations include dynamic clock latency and dynamic rail voltage, set by `set_clock_latency` and `set_voltage` commands respectively. These dynamic annotations can lead to dynamic clock arrivals. CRPR handles these dynamic clock arrivals in the same way as it handles delays due to crosstalk. The CRPR value is calculated using dynamic clock arrivals for zero cycle paths only. For all other paths, only the static component of the clock arrival is used thus producing more accurate results.

An example of such dynamic annotations is as follows:

```
pt_shell> set_clock_latency -early -source 2.5 -dynamic -0.5
[get_clocks CLK]
pt_shell> set_clock_latency -source -late 5.5 -dynamic 0.5
[get_clocks CLK]
```

The first of these two commands specifies a total early source latency of 2.5, consisting of static source latency of 3.0 and dynamic source latency of –0.5. The second command specifies a total late source latency of 5.5, consisting of static source latency of 5.0 and dynamic source latency of 0.5. In this case, the static CRP is equal to 2 (5 minus 3). The dynamic CRP is equal to 3 (5.5 minus 2.5).

Transparent Latch Edge Considerations

For a path that ends at a transparent latch, PT calculates two clock reconvergence pessimism values: one for rising edges and one for falling edges at the common node. The opening and closing clock edges are effectively shifted, each by an amount equal to its corresponding pessimism value, as indicated in Figure 11-14.

Figure 11-14 Clock Reconvergence Pessimism Removal for Latches

In practice, the opening-edge pessimism value affects the slack of nonborrowing paths and also reduces the amount of time borrowed for borrowing paths. Meanwhile, the closing-edge pessimism value increases the maximum amount of time borrowing allowed at a latch and reduces the amount of the violation for a path that fails to meet its setup constraint.

To get a report about the calculation of clock reconvergence pessimism values for level-sensitive latches, use the `report_crpr` command.

Reporting CRPR Calculations

The `report_crpr` command reports the calculation of clock reconvergence pessimism (CRP) between two register clock pins or ports. It reports the time values calculated for both static and dynamic conditions, and the choice from among those values actually used for pessimism removal. In the command, you specify the pins of the launch and capture registers, the clock, and type of check (setup or hold). For example,

```
pt_shell> report_crpr -from [get_pins ffa/CP] -to [get_pins
ffd/CP] -clock CLK -setup
```

The command generates a report showing the location of the common node, the launch and capture edge types (rising or falling), the four calculated arrival times at the common point (early/late, rise/fall), the calculated CRP values (rise and fall), and the values actually used for opening-edge and closing-edge pessimism removal.

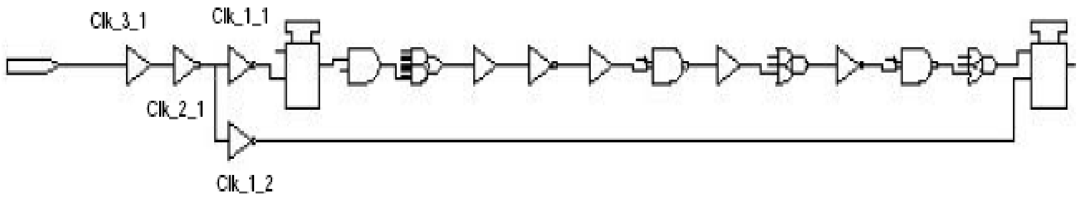
The amount of CRP reported by `report_crpr` can be slightly different from the amount reported by `report_timing`. This is because `report_timing`, for computational efficiency, “merges” multiple points for CRPR calculations when the CRP differences between adjacent points are too small to be significant. The `timing_crpr_threshold_ps` variable sets the time threshold for merging, which is 20 picoseconds by default. When `report_crpr` and `report_timing` give different CRP values, the value reported by `report_crpr` is more accurate because it does not do any merging of adjacent points. For more information about the `report_crpr` command, see the man page.

1. What is on-chip variation (OCV)? How it will affect the Design?
Constructing a cell on an ASIC is a process that involves many variables. Examples of variables that can occur on a single chip include small variations in the mask, imperfections in optical proximity correction, and etch variations. Additionally, many of these variations can occur over a very small area.

The many variables involved in the manufacturing process mean the gate delay of a cell is really a Gaussian distribution with a mean and standard deviation determined by the cell design and these many variables. if a design is shown to work at both best case and worst case, it is assumed to work at any point in between.

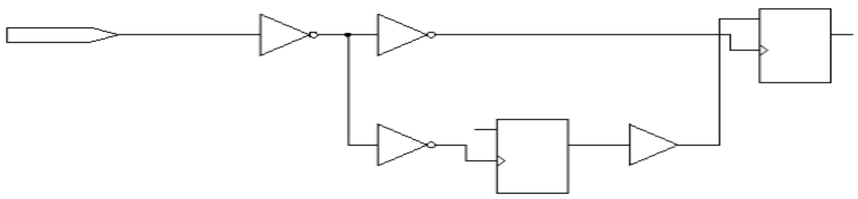
Setup Problems
The circuit in figure 3 shows one of the critical paths on the chip. While the differences in loading of the clock drivers and parasitics of the clock nets will be included in calculating the clock arrival times at the flops, the on-chip variation effects are not being considered when doing a worst-case only analysis.

If the clock path clock driver ends up being slightly faster than the data path clock driver, there is a potential of manufacturing chips which have setup violations that your static timing analysis didn't tell you about.



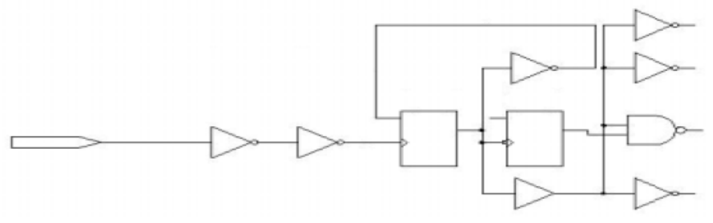
[Figure 2. Critical path used in setup examples]

Hold Problems
In the circuit in figure 3, we have a very short logic path between two flops. Again, a worst-case or best-case only analysis will consider loading and parasitics of the clock tree, but the on-chip variation effects will not be modeled.


[Figure 3. Short path used in hold examples]

If the data path clock driver ends up being slightly faster than the clock path clock driver, there is a potential of manufacturing chips which have hold violations that your static timing analysis didn't tell you about.

Clock Gating
The circuit in figure 4 is a clock gating circuit. It starts with a clock divider register. The clock tree fanning out from there starts with a single clock buffer in the first level. The second level of the clock tree has four clock drivers, one of which is a NAND gate performing a clock gating function.


[Figure 4. Circuit used in clock gating examples]

14. How to solve the OCV problems?
Modeling On-Chip Variation in PT
To do on-chip variation analysis, every timing arc in the design must have both a minimum and a maximum delay which account for the on-chip variation. These are not the same delays that you would use for a simple min-max analysis. Those delays represent the minimum and maximum delays that will be seen across all chips from that process.

For on-chip variation analysis, we want the minimum and maximum delays that could be seen across a single chip. For on-chip variation analysis we will make two analysis runs. The "slow-chip" analysis will set the maximum delays at the worst case delays for the process and the minimum delays slightly faster than that. For the "fast-chip" analysis, we will set the minimum delays to the best case delays for the process, and we will set the maximum delays slightly slower than the best case delays.

Annotate from one SDF file
read_sdf –analysis_type on_chip_variation foo.sdf

Annotate from two SDF files
read_sdf –analysis_type on_chip_variation –min_file foo_min.sdf \
–max_file foo_max.sdf

Two Operating conditions

Sometimes you may have operating conditions specifically scaled for use with on-chip variation analysis. For a slow chip analysis, the worst case operating condition is specified with –max and the scaled worst case operating condition is specified with –min. For a fast chip analysis, the best case operating condition is specified with –min and the scaled best case operating condition is specified with –max.

```
set_operating_conditions –analysis_type on_chip_variation –min $WCOCV –max $WC
```

Single Operating condition
On-chip variation analysis can still be run even if you have only one operating condition. In this case the minimum and maximum times are simply scaled versions of the delays determined from the single operating condition that is loaded. The scaling factors are set by the set_timing_derate command described next.

```
set_operating_conditions –analysis_type on_chip_variation $OP_CON
set_timing_derate –min 0.8 –max 1.0
```

Set_timing_derate
The set_timing_derate command can be used with any of the above methods to provide further scaling of the timing paths. Scaling factors can be set independently for data paths, clock paths, cell delays, net delays, and cell timing checks. If neither –clock nor -data is specified, the derating factors apply to both clock and data paths. If –cell_delay, -net_delay, and –cell_check are all omitted from the command, the derating factors apply to both cell and net delays, but not to cell timing checks such as setup and hold times.

For example the following commands could be used to do a fast chip analysis with the best case operating condition loaded, **10%** variation in **cell delays**, **5%** variation in **net delays**, and a **5%** variation in **cell timing** checks.

```
set_operating_conditions –analysis_type on_chip_variation $BC_OP_CON
set_timing_derate –min 1.0 –max 1.10 –cell_delay
set_timing_derate –min 1.0 –max 1.05 –net_delay
set_timing_derate –min 1.0 –max 1.05 –cell_check
```

we are telling Primetime to assume that the min paths can be twenty percent faster than the max paths. More typically on-chip variation is modeled at six or eight or maybe ten percent. Your ASIC or library vendor should tell you what is appropriate for the library that you are using.

Setup Problems
When doing a setup test, the timing tool wants to verify that the latest possible data arrival can still be captured by the earliest possible clock arrival at the endpoint register.

The latest possible data arrival is determined by taking the maximum delays along the clock path to the startpoint register and the maximum delays along the slowest data path from the startpoint register to the endpoint register.

The earliest possible clock arrival at the endpoint register is determined by taking the minimum delays along the clock path to the endpoint register.

Hold Problems

When doing a hold test, the timing tool wants to verify that the earliest possible data arrival does not arrive at the endpoint register before the latest possible clock arrival.

So we use min delays for the clock path to the startpoint register, min delays through the shortest data path, and max delays for the clock path to the endpoint register.

Clock Gating Problems
With an AND gate, we want to verify that the gating signal can only change while the clock is low. A hold check is done to verify that the gating signal changes after the falling edge of the clock, and a setup check is done to verify that the gating signal changes before the next rising edge of the clock. We're not looking at the setup check here because it has lots of positive slack and is therefore not very interesting. The hold check is going to use min delays along the gating path and max delays along the clock path.

How OCV Analysis improves silicon performance?
While many ASIC vendors do not require youto run OCV analysis, this does not mean that they have figured out how to build chips with zerovariation. If the vendor does not have a manufacturing test that can catch the failures, or if theyare not willing to accept the resulting yield loss, they must still account for on-chip variationsduring static timing analysis. It may be done through padding the setup and hold margins of theflops, requiring extra set_clock_uncertainty even after the clocks are placed and routed.

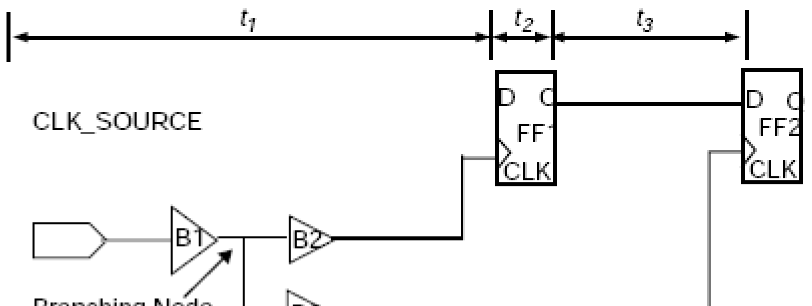
15. What is Clock Reconvergence Pessimism Removal CRPR? Explain ?
Clock Reconvergence Pessimism Removal (CRPR) is the process of identifying and removing the pessimism introduced in the slack reports for clock paths when the clock paths have a segment in common.

In the on-chip variation methodology (using the setAnalysisMode -BcWc and the setTimingDerate commands or setAnalysisMode -ocv), during setup checks if both the launch clock late path and the capture clock early path share a portion of the clock network, then for the common clock network, a pessimism equal to the difference in maximum and minimum delay values is introduced in the slack values.

When you use the setAnalysisMode -BcWc and the setTimingDerate commands, the software calculates maximum delay value by multiplying the delay by the scale value that you set using setTimingDerate –late. Similarly, the software calculates the minimum delay value by multiplying the delay by the scale value that you set using setTimingDerate -early.

As shown in Figure 20-3, the setup check at FF2 compares the maximum delay data at the D pin against minimum delay clock at the CLK pin. The maximum delay data at FF2/D consists of a sum of maximum signal delay from FF1/Q to FF2/D, the maximum delay from CLK_SOURCE to FF1/CLK, and the delay from FF1/CLK to FF1/Q. Similarly, the minimum delay clock arrival time at FF2/CLK is the minimum delay from CLK_SOURCE to FF2/CLK.

Figure 20-3 Example Signal Path



The setup check equation for the example in Figure 20-3 with pessimism is as follows:

$$t_{1max} + t_{2max} + t_{3max} \le t_{4min} + t_{pa} - t_{su}$$

where,
t1 = Delay value for launch clock late path
t2 = Delay between FF1/CLK and FF1/Q
t3 = Delay between FF1/Q and FF2/D
t2 + t3 = Delay value for late data path
t4 = Delay value for capture clock early path
tpa = Period adjustment
tsu = Setup time

The setup check equation incorrectly implies that the common clock network, B1, can simultaneously use maximum delay for the launch clock late path (clock source to FF1/CLK) and minimum delay for the capture clock early path (clock source to FF2/CLK). You use the CRPR to remove this pessimism.

Setup check equation using CRPR is as follows:

$$t_{1max} + t_{2max} + t_{3max} \le t_{4min} + t_{pa} - t_{su} + t_{crpr}$$

Where,
tcrpr = Difference in the maximum and minimum delay from the clock source to the branching node

Similarly, **hold check** equation using CRPR is as follows:
Where,

$$t_{1min} + t_{2min} + t_{3min} + t_{crpr} \le t_{4max} + t_{H}$$

t1 = Delay value for launch clock early path
t2 = Delay between FF1/CLK and FF1/Q
t3 = Delay between FF1/Q and FF2/D
t2 + t3 = Delay value for early data path
t4 = Delay value for capture clock late path
tcrpr = Difference in the maximum and minimum delay from the clock source to the branching node
tH = Hold time

For example, you use the following setTimingDerate command for setup check delays in Figure 20-3 on page 648 in scaled on-chip variation analysis methodology:

setTimingDerate -max –late 1 –early 0.9 –clock

With the setTimingDerate command, if the delay through B1 is 1ns, the software removes the pessimism of 0.1ns. Without CRPR the analysis tool incorrectly assumes that B1 can have a delay of 1 and 0.9. The common path pessimism time (tcrpr) is calculated as follows:

B1 * Late Derate - B1 * Early Derate = tcrpr

Therefore,
tcrpr = 1.0ns * 1.0 - 1.0ns * 0.9 = 0.1ns

16. What kinds of timing violations are in a typical timing analysis report?

- Setup time violations
- Hold time violations
- Minimum delay
- Maximum delay
- Slack
- External delay

17. List the possible techniques to fix a timing violation

18.

19.

- Buffering
- Wire-sizing
- Transistor sizing
- Re-routing
- Placement updates
- Re-synthesis (logic transformations)
- Cloning
- Taking advantage of useful skew
- Making sure we don't have false violations (false path, etc.)

20. Can a latch based design be analyzed using STA?

Setup and Hold Checking for Latches

Latch-based designs typically use two-phase, non overlapping clocks to control successive registers in a data path. In these cases, Timing Engine can use time borrowing to lessen the constraints on successive paths. For example, consider the two-phase, latch-based path shown in Figure 1-8. All three latches are level-sensitive, with the gate active when the G input is high. L1 and L3 are controlled by PH1, and L2 is controlled by PH2. A rising edge launches data from the latch output, and a falling edge captures data at the latch input. For this example, consider the latch setup and delay times to be zero.

Figure 1-8 Latch-Based Paths

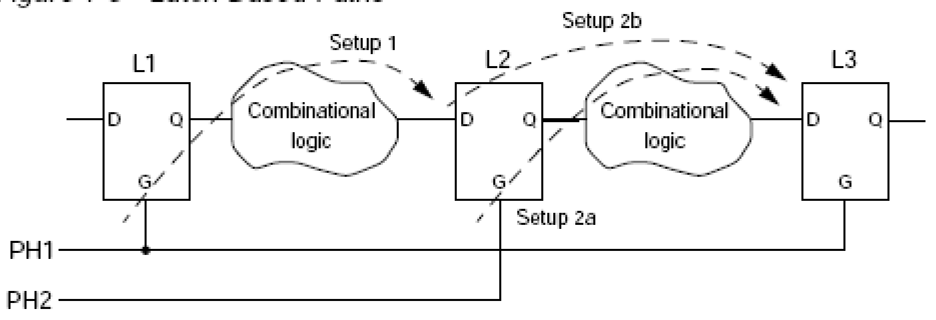
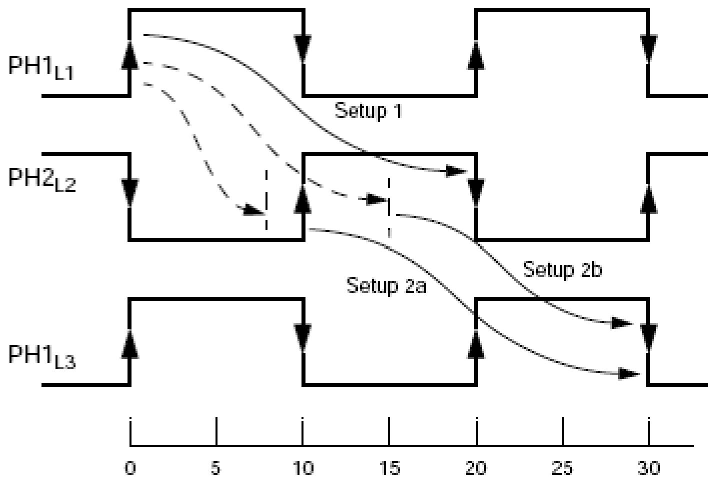


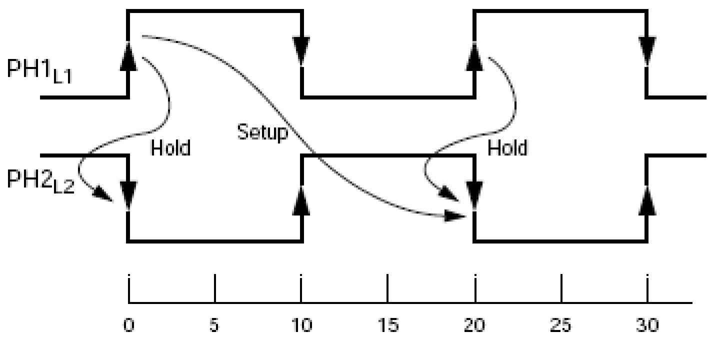
Figure 1-9 shows how Timing Engine performs setup checks between these latches. For the path from L1 to L2, the rising edge of PH1 launches the data. The data must arrive at L2 before the closing edge of PH2 at time=20. This timing requirement is labeled Setup 1. Depending on the amount of delay between L1 and L2, the data might arrive either before or after the opening edge of PH2 (at time=10), as indicated by the dashed-line arrows in the timing diagram. Arrival after time=20 would be a timing violation.

Figure 1-9 Time Borrowing in Latch-Based Paths



If the data arrives at L2 before the opening edge of PH2 at time=10, the data for the next path from L2 to L3 gets launched by the opening edge of PH2 at time=10, just as a synchronous flip-flop would operate. This timing requirement is labeled Setup 2a. If the data arrives after the opening edge of PH2, the first path (from L1 to L2) borrows time from the second path (from L2 to L3). In that case, the launch of data for the second path occurs not at the opening edge, but at the data arrival time at L2, at some time between the opening and closing edges of PH2. This timing requirement is labeled Setup 2b. When borrowing occurs, the path originates at the D pin rather than the G pin of L2.

For the first path (from L1 to L2), Timing Engine reports the setup slack as zero if borrowing occurs. The slack is positive if the data arrives before the opening edge at time=10, or negative (a violation) if the data arrives after the closing edge at time=20. To perform hold checking, Timing Engine considers the launch and capture edges relative to the setup check. It verifies that data launched at the startpoint does not reach the endpoint too quickly, thereby ensuring that data launched in the previous cycle is latched and not overwritten by the new data. This is depicted in Figure 1-10.
Figure 1-10 Hold Checks in Latch-Based Paths



1 **How delay affected by PVT (Process-Voltage-Temperature)?**

Process-Voltage-Temperature (PVT) Variations and Static Timing Analysis

How delays vary with different PVT conditions? Show the graph.

- P increase->dealy increase
- P decrease->delay decrease
- V increase->delay decrease
- V decrease->delay increase
- T increase->delay increase
- T decrease->delay decrease

What is cell delay and net delay?

- **Gate delay**
- Transistors within a gate take a finite time to switch. This means that a change on the input of a gate takes a finite time to cause a change on the output.[Magma]
- Gate delay =function of(i/p transition time, Cnet+Cpin).
- Cell delay is also same as Gate delay.

Cell delay

- For any gate it is measured between 50% of input transition to the corresponding 50% of output transition.
- Intrinsic delay
- Intrinsic delay is the delay internal to the gate. Input pin of the cell to output pin of the cell.
- It is defined as the delay between an input and output pair of a cell, when a near zero slew is applied to the input pin and the output does not see any load condition.It is predominantly caused by the internal capacitance associated with its transistor.
- This delay is largely independent of the size of the transistors forming the gate because increasing size of transistors increase internal capacitors.

- **Net Delay (or wire delay)**
- The difference between the time a signal is first applied to the net and the time it reaches other devices connected to that net.
- It is due to the finite resistance and capacitance of the net.It is also known as wire delay.
- Wire delay =fn(Rnet , Cnet+Cpin)

What are delay models and what is the difference between them?

- Linear Delay Model (LDM)
- Non Linear Delay Model (NLDM)
- Composite current source modeling (CCS)

What is wire load model?

- Wire load model is NLDM which has estimated R and C of the net.

The major design challenges of ASIC design consist of microscopic issues and macroscopic issues [1]. The microscopic issues are ultra-high speeds, power dissipation, supply rail drop, growing importance of interconnect, noise, crosstalk, reliability, manufacturability and the clock distribution. The macroscopic issues are time to market, design complexity, high levels of abstractions, reuse, IP portability, systems on a chip and tool interoperability.

To meet the design challenge of clock distribution, the timing analysis is performed. Timing analysis is to estimate when the output of a given circuit gets stable. Timing Analysis (TA) is a design automation program which provides an alternative to the hardware debugging of timing problems. The program establishes whether all paths within the design meet stated timing criteria, that is, that data signals arrive at storage elements early enough valid gating but not so early as to cause premature gating. The output of Timing Analysis includes 'Slack" at each block to provide a measure of the severity of any timing problem [13].

Static vs. Dynamic Timing Analysis
Timing analysis can be **static** or **dynamic**.

Static Timing Analysis (STA) works with timing models where as the Dynamic Timing Analysis (DTA) works with spice models. STA has more pessimism and thus gives maximum delay of the design. DTA overcomes this difficulty because it performs full timing simulation. The problem associated with DTA is the computational complexity involved in finding the input pattern(s) that produces maximum delay at the output and hence it is slow. The static timing analyzer will report the following delays: Register to Register delays, Setup times of all external synchronous inputs, Clock to Output delays, Pin to Pin combinational delays. The clock to output delay is usually just reported as simply another pin-to-pin combinational delay. Timing analysis reports are often pessimistic since they use worst case conditions.

The wide spread use of STA can be attributed to several factors [2]:

The basic STA algorithm is linear in runtime with circuit size, allowing analysis of designs in excess of 10 million instances.

The basic STA analysis is conservative in the sense that it will over-estimate the delay of long paths in the circuit and under-estimate the delay of short paths in the circuit. This makes the analysis ”safe”, guaranteeing that the design will function at least as fast as predicted and will not suffer from hold-time violations.

The STA algorithms have become fairly mature, addressing critical timing issues such as interconnect analysis, accurate delay modeling, false or multi-cycle paths, etc.
Delay characterization for cell libraries is clearly defined, forms an effective interface between the foundry and the design team, and is readily available. In addition to this, the Static Timing Analysis (STA) does not require input vectors and has a runtime that is linear with the size of the circuit [9].

PVT vs. Delay

Sources of variation can be:

- Process variation (P)
- Supply voltage (V)
- Operating Temperature (T)

Process Variation

This variation accounts for deviations in the semiconductor fabrication process. Usually process variation is treated as a percentage variation in the performance calculation. Variations in the process parameters can be impurity concentration densities, oxide thicknesses and diffusion depths. These are caused bye non uniform conditions during depositions and/or during diffusions of the impurities. This introduces variations in the sheet resistance and transistor parameters such as threshold voltage. Variations are in the dimensions of the devices, mainly resulting from the limited resolution of the photolithographic process. This causes (W/L) variations in MOS transistors.

Process variations are due to variations in the manufacture conditions such as temperature, pressure and dopant concentrations. The ICs are produced in lots of 50 to 200 wafers with approximately 100 dice per wafer. The electrical properties in different lots can be very different. There are also slighter differences in each lot, even in a single manufactured chip. There are variations in the process parameter throughout a whole chip. As a consequence, the transistors have different transistor lengths throughout the chip. This makes the propagation delay to be different everywhere in a chip, because a smaller transistor is faster and therefore the propagation delay is smaller.

Supply Voltage Variation

The design’s supply voltage can vary from the established ideal value during day-to-day operation. Often a complex calculation (using a shift in threshold voltages) is employed, but a simple linear scaling factor is also used for logic-level performance calculations.

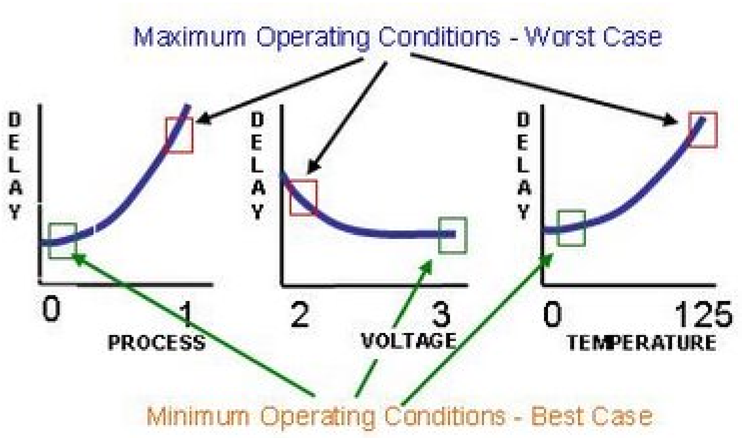
The saturation current of a cell depends on the power supply. The delay of a cell is dependent on the saturation current. In this way, the power supply inflects the propagation delay of a cell. Throughout a chip, the power supply is not constant and hence the propagation delay varies in a chip. The voltage drop is due to nonzero resistance in the supply wires. A higher voltage makes a cell faster and hence the propagation delay is reduced. The decrease is exponential for a wide voltage range. The self-inductance of a supply line contributes also to a voltage drop. For example, when a transistor is switching to high, it takes a current to charge up the output load. This time varying current (for a short period of time) causes an opposite self-induced electromotive force. The amplitude of the voltage drop is given by .V=L*dI/dt, where L is the self inductance and I is the current through the line.

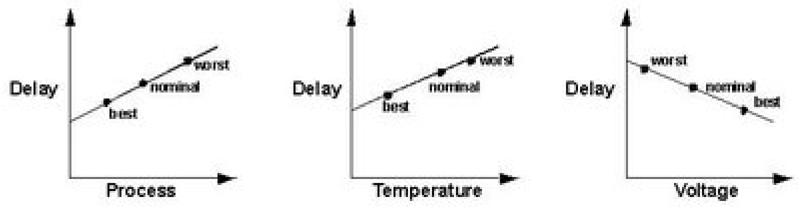
Operating Temperature Variation

Temperature variation is unavoidable in the everyday operation of a design. Effects on performance caused by temperature fluctuations are most often handled as linear scaling effects, but some submicron silicon processes require nonlinear calculations.

When a chip is operating, the temperature can vary throughout the chip. This is due to the power dissipation in the MOS-transistors. The power consumption is mainly due to switching, short-circuit and leakage power consumption. The average switching power dissipation (approximately given by Paverage = Cload*Vpower supply 2*fclock) is due to the required energy to charge up the parasitic and load capacitances. The short-circuit power dissipation is due to the finite rise and fall times. The nMOS and pMOS transistors may conduct for a short time during switching, forming a direct current from the power supply to the ground. The leakage power consumption is due to the nonzero reverse leakage and sub-threshold currents. The biggest contribution to the power consumption is the switching. The dissipated power will increase the surrounding temperature. The electron and hole mobility depend on the temperature. The mobility (in Si) decreases with increased temperature for temperatures above –50 °C. The temperature, when the mobility starts to decrease, depends on the doping concentration. A starting temperature at –50 °C is true for doping concentrations below 1019 atoms/cm3. For higher doping concentrations, the starting temperature is higher. When the electrons and holes move slower, then the propagation delay increases. Hence, the propagation delay increases with increased temperature. There is also a temperature effect, which has not been considered. The threshold voltage of a transistor depends on the temperature. A higher temperature will decrease the threshold voltage. A lower threshold voltage means a higher current and therefore a better delay performance. This effect depends extremely on power supply, threshold voltage, load and input slope of a cell. There is a competition between the two effects and generally the mobility effect wins.

The following figure shows the PVT operating conditions.





The best and worst design corners are defined as follows:

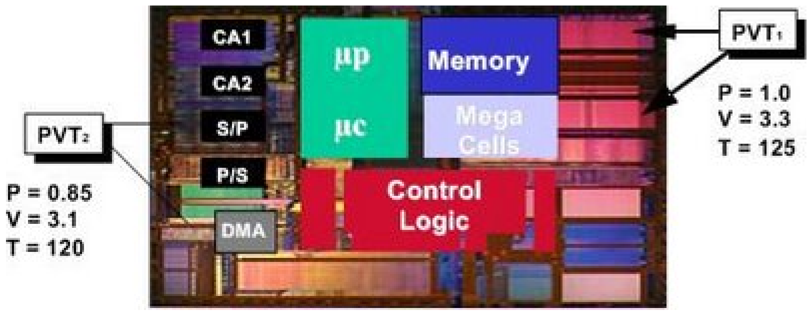
- **Best case:** fast process, highest voltage and lowest temperature
- **Worst case:** slow process, lowest voltage and highest temperature

On Chip Variation

On-chip variation is minor differences on different parts of the chip within one operating condition. On-Chip variation (OCV) delays vary across a single die due to:

- Variations in the manufacturing process (P)
- Variations in the voltage (due to IR drop)
- Variations in the temperature (due to local hot spots etc)

This need is to be modeled by scaling the coefficients. Delays have uncertainty due to the variation of Process (P), Voltage (V), and Temperature (T) across large dies. On-Chip variation allows you to account for the delay variations due to PVT changes across the die, providing more accurate delay estimates.



Timing Analysis With On-Chip Variation

- For cell delays, the on-chip variation is between 5 percent above and 10 percent below the SDF back-annotated values.
- For net delays, the on-chip variation is between 2 percent above and 4 percent below the SDF back-annotated values.
- For cell timing checks, the on-chip variation is 10 percent above the SDF values for setup checks and 20 percent below the SDF values for hold checks.

In Prime Time, OCV derations are implemented using the following commands:

- pt_shell> read_sdf -analysis_type on_chip_variation my_design.sdf
- pt_shell> set_timing_derate -cell_delay -min 0.90 -max 1.05
- pt_shell> set_timing_derate -net -min 0.96 -max 1.02
- pt_shell> set_timing_derate -cell_check -min 0.80 -max 1.10

In the traditional deterministic STA (DSTA), process variation is modeled by running the analysis multiple times, each at a different process condition. For each process condition, a so-called corner file is created that specifies the delay of the gates at that process condition. By analyzing a sufficient number of process conditions, the delay of the circuit under process variation can be bounded.

The uncertainty in the timing estimate of a design can be classified into three main categories.

- **Modeling and analysis errors:** Inaccuracy in device models, in the extraction and reduction of interconnect parasitics and in the timing analysis algorithms.

- **Manufacturing variations:** Uncertainty in the parameters of a fabricated devices and interconnects from die-to-die and within a particular die.

- **Operating context variations:** Uncertainty in the operating environment of a particular device during its lifetime, such as temperature, supply voltage, mode of operation and lifetime wear-out.

For instance, the STA tool might utilize a conservative delay noise algorithm resulting in certain paths operating faster than expected. Environmental uncertainty and uncertainty due to modeling and analysis errors are typically modeled using worst-case margins, whereas uncertainty in process is generally treated statistically.

Taxonomy of Process Variations

As process geometries continue to shrink, the ability to control critical device parameters is becoming increasingly difficult and significant variations in device length, doping concentrations and oxide thicknesses have resulted [9]. These process variations pose a significant problem for timing yield prediction and require that static timing analysis models the circuit delay not as a deterministic value, but as a random variable.

Process variations can either systematic or random.

- **Systematic variation:** Systematic variations are deterministic in nature and are caused by the structure of a particular gate and its topological environment. The systematic variations are the component of variation that can be attributed to a layout or manufacturing equipment related effects. They generally show spatial correlation behavior.

- **Random variation:** Random or non-systematic variations are unpredictable in nature and include random variations in the device length, discrete doping fluctuations and oxide thickness variations. Random variations cannot be attributed to a specific repeatable governing principle. The radius of this variation is comparable to the sizes of individual devices, so each device can vary independently.

Process variations can classified as follow:

- **Inter-die variation or die-to-die:** Inter-chip variations are variations that occur from one die to next, meaning that the same device on a chip has different features among different die of one wafer, from wafer to wafer and from wafer lot to wafer lot. Die-to-die variations have a variation radius larger than the die size including within wafer, wafer to wafer, lot to lot and fab to fab variations [12].

- **Intra-die or within-die variation**: Intra-die variations are the variations in device features that are present within a single chip, meaning that a device feature varies between different locations on the same die. Intra-chip variations exhibit spatial correlations and structural correlations.

- **Front-end variation:** Front-end variations mainly refer to the variations present at the transistor level. The primary components of the front end variations entail transistor gate length and gate width, gate oxide thickness, and doping related variations. These physical variations cause changes in the electrical characteristics of the transistors which eventually lead to the variability in the circuit performance.

- **Back-end variation:** Back-end variations refer to the variations on various levels of interconnecting metal and dielectric layers used to connect numerous devices to form the required logic gates.

In practice, device features vary among the devices on a chip and the likelihood that all devices have a worst-case feature is extremely small. With increasing awareness of process variation, a number of techniques have been developed which model random delay variations and perform STA. These can be classified into f**ull-chip analysis** and **path-based analysis** approaches.

Full Chip Analysis

Full-chip analysis models the delay of a circuit as a random variable and endeavors to compute its probability distribution. The proposed methods are heuristic in nature and have a very high worst-case computational complexity. They are also based on very simple delay models, where the dependence of gate delay due to slope variation at the input of the gate and load variation at the output of the gate is not modeled. When run time and accuracy are considered, full chip STA is not yet practical for industrial designs.

Path Based STA

Path based STA provides statistical information on a path-by-path basis. It accounts for intra-die process variations and hence eliminates the pessimism in deterministic timing analysis, based on case files. It is a more accurate measure of which paths are critical under process variability, allowing more correct optimization of the circuit. This approach does not include the load dependence of the gate delay due to variability of fan out gates and does not address spatial correlations of intra-die variability.

To compute the intra-die path delay component of process variability, first the sensitivity of gate delay, output slope and input load with respect to slope, output load and device length are computed. Finally, when considering sequential circuits, the delay variation in the buffered clock tree must be considered.

In general, the fully correlated assumptions will under-estimate the variation in the arrival times at the leaf nodes of the clock tree which will tend to overestimate circuit performance.

- **PVT, Derarting and STA**

  Where do you get the WLM's? Do you create WLM's? How do you specify?

  - Wire Load Models (WLM) are available from the library vendors.
  - We dont create WLM.
  - WLMs can be specified depending on the area

  What is the derate value that can be used?

  - For setup check derate data path by 8% to 15%, no derate in the clock path.
  - For hold check derate clock path by 8% to 15%, no derate in the data path.

  What are the corners you check for timing sign-off? Is there any changes in the derate value for each corner?

- Corners: Worst, Best, Typical.
- Same derating value for best and worst.For typical it can be less.

Write Setup and Hold equtions?

- Setup equation: Tlaunch clock + Tclk-q_max + Tcombo_max <= Tcapute clock - (Tsetup+skew)
- Hold equation: Tlaunch clock + Tclk-q_min + Tcombo_min >= Tcapture clock + (Thold-skew)
- .

Where do you get the derating value? What are the factors that decide the derating factor?

- Based on the guidelines and suggestions from the library vendor and previous design experience derating value is decided.
- PVT variation is the factor that decides the derating factor.

What factors decides the setup time of flip-flop?

- D- pin transition and clock transition.

Why dont you derate the clock path by -10% for worst corner analysis?

- We can do. But it may not be accurate as the data path derate.

What is latency? Give the types?

- **Source Latency**
- It is known as source latency also. It is defined as "the delay from the clock origin point to the clock definition point in the design".
- Delay from clock source to beginning of clock tree (i.e. clock definition point).
- The time a clock signal takes to propagate from its ideal waveform origin point to the clock definition point in the design.

- **Network latency**
- It is also known as Insertion delay or Network latency. It is defined as "the delay from the clock definition point to the clock pin of the register".
- The time clock signal (rise or fall) takes to propagate from the clock definition point to a register clock pin.

What are the different types of delays in ASIC or VLSI design?

Different Types of Delays in ASIC or VLSI design

- **Source Delay/Latency**
- **Network Delay/Latency**
- **Insertion Delay**
- **Transition Delay/Slew: Rise time, fall time**
- **Path Delay**
- **Net delay, wire delay, interconnect delay**
- **Propagation Delay**
- **Phase Delay**
- **Cell Delay**
-
- **Intrinsic Delay**
- **Extrinsic Delay**
- **Input Delay**
- **Output Delay**
- **Exit Delay**
- **Latency (Pre/post CTS)**
- **Uncertainty (Pre/Post CTS)**
- **Unateness: Positive unateness, negative unateness**
- **Jitter: PLL jitter, clock jitter**

Gate delay

- Transistors within a gate take a finite time to switch. This means that a change on the input of a gate takes a finite time to cause a change on the output.[Magma]
- Gate delay =function of(i/p transition time, Cnet+Cpin).
- Cell delay is also same as Gate delay.

Source Delay (or Source Latency)

- It is known as source latency also. It is defined as "the delay from the clock origin point to the clock definition point in the design".
- Delay from clock source to beginning of clock tree (i.e. clock definition point).
- The time a clock signal takes to propagate from its ideal waveform origin point to the clock definition point in the design.

Network Delay(latency)

- It is also known as Insertion delay or Network latency. It is defined as "the delay from the clock definition point to the clock pin of the register".
- The time clock signal (rise or fall) takes to propagate from the clock definition point to a register clock pin.

Insertion delay

- The delay from the clock definition point to the clock pin of the register.

Transition delay

- It is also known as "Slew". It is defined as the time taken to change the state of the signal. Time taken for the transition from logic 0 to logic 1 and vice versa . or Time taken by the input signal to rise from 10%(20%) to the 90%(80%) and vice versa.

- Transition is the time it takes for the pin to change state.

Slew

- Rate of change of logic.See Transition delay.

- Slew rate is the speed of transition measured in volt / ns.

Rise Time

- Rise time is the difference between the time when the signal crosses a low threshold to the time when the signal crosses the high threshold. It can be absolute or percent.

- Low and high thresholds are fixed voltage levels around the mid voltage level or it can be either 10% and 90% respectively or 20% and 80% respectively. The percent levels are converted to absolute voltage levels at the time of measurement by calculating percentages from the difference between the starting voltage level and the final settled voltage level.

Fall Time

- Fall time is the difference between the time when the signal crosses a high threshold to the time when the signal crosses the low threshold.

- The low and high thresholds are fixed voltage levels around the mid voltage level or it can be either 10% and 90% respectively or 20% and 80% respectively. The percent levels are converted to absolute voltage levels at the time of measurement by calculating percentages from the difference between the starting voltage level and the final settled voltage level.

- For an ideal square wave with 50% duty cycle, the rise time will be 0.For a symmetric triangular wave, this is reduced to just 50%.

Definition of Rise Time

Rise time is the difference between the time when the signal crosses a low threshold to the time when the signal crosses the high threshold. It can be absolute or percent.

Absolute Rise Time

In absolute rise time, the low and high thresholds are fixed voltage levels around the mid voltage level.

Percent Rise Time

In percent rise time, the low and high thresholds are percent levels, and are usually either 10% and 90% respectively or 20% and 80% respectively. The percent levels are converted to absolute voltage levels at the time of measurement by calculating percentages from the difference between the starting voltage level and the final settled voltage level.

Definition of Fall Time

Fall time is the difference between the time when the signal crosses a high threshold to the time when the signal crosses the low threshold. It can be absolute or percent.

Absolute Fall Time

In absolute fall time, the low and high thresholds are fixed voltage levels around the mid voltage level.

Percent Fall Time

In percent fall time, the low and high thresholds are percent levels, and are usually either 10% and 90% respectively or 20% and 80% respectively. The percent levels are converted to absolute voltage levels at the time of measurement by calculating percentages from the difference between the starting voltage level and the final settled voltage level.

Significance of Rise Time & Fall Time

This is best explained by comparing a square wave with a triangular wave. In an ideal square wave with 50% duty cycle, the rise time will be 0 and the signal will be above threshold for 100% of the half period time. In a symmetric triangular wave, this is reduced to just 50%. More severely affected is the total area above the threshold, which is reduced to 25% of that of square wave's. Though the information about loss of time above threshold is conveyed by many other parameters, the information about loss of area above threshold is only conveyed by rise and fall times.

Rise Time & Fall Time Requirements

The rise time & fall time should be small compared to the clock period. A factor of 10 is considered good. Very large rise or fall times have the risk of the cycles going undetected. Also, large rise or fall times mean that the signal will be hovering around mid level for too long, making the system highly susceptible to noise and multiple triggering if there is not enough hysteresis.

This might make you think that the faster rise & fall times are, the better the system is. Not really. Very fast rise or fall times are not free from trouble. They might cause severe ringing at the receiver resulting in reduction in voltage & timing margins or even double triggering. Or the fast edges can & will get coupled to the adjacent signal lines causing false triggering on them or reducing the voltage margins.

Measurement of Rise Time & Fall Time

The measurement system should have enough analog bandwidth to measure the edge times faithfully. This means that the oscilloscope and the probe together should have at least twice the bandwidth of the fastest edge rates to be measured. To estimate the highest frequencies of significant level in an edge, divide 0.35 with an estimate of the edge time (i.e. rise time or fall time).

If a digital oscilloscope is being used, an edge should get at least 5 samples for the measurements to be reasonably accurate. Normally, the lesser bandwidth the measuring system has, the larger the measured rise or fall time will be from the actual value. This means that the measured value for rise & fall rates will always be larger than the actual value. However, this is not always true. If the probe used for measurement is highly under-damped, it will result in ringing causing the measured signal to have faster rise & fall times than the actual signal.

You have to use cursors to measure the rise & fall times in analog oscilloscopes. In modern digital oscilloscopes, the measurements are automatically done by the oscilloscopes. However, all digital oscilloscopes measure only one edge in the whole acquisition (normally first). To have a higher confidence in the measurement, multiple measurements must be made. One way is to let the oscilloscope run in continuous run mode and let the statistics accumulate over time. This approach may be time consuming and will not measure consecutive edges.

The better way is to use a post processing software. These software are very fast and can acquire data from the oscilloscope automatically, process millions of data edges in just a few seconds, display the results graphically in time domain as well as in frequency domain, with statistics along with the time stamp of max and min values. You can even save waveforms from the oscilloscope for post processing by the software

- The rise/fall definition is set on the meter to 10% and 90% based on the linear power in Watts. These points translate into the -10 dB and -0.5 dB points in log mode (10 log 0.1) and (10 log 0.9). The rise/fall time values of 10% and 90% are calculated based on an algorithm, which looks at the mean power above and below the 50% points of the rise/fall times. Click here to see more.

Path delay

- Path delay is also known as pin to pin delay. It is the delay from the input pin of the cell to the output pin of the cell.

Net Delay (or wire delay)

- The difference between the time a signal is first applied to the net and the time it reaches other devices connected to that net.
- It is due to the finite resistance and capacitance of the net.It is also known as wire delay.
- Wire delay =fn(Rnet , Cnet+Cpin)

Propagation delay

- For any gate it is measured between 50% of input transition to the corresponding 50% of output transition.
- This is the time required for a signal to propagate through a gate or net. For gates it is the time it takes for a event at the gate input to affect the gate output.
- For net it is the delay between the time a signal is first applied to the net and the time it reaches other devices connected to that net.
- It is taken as the average of rise time and fall time i.e. Tpd= (Tphl+Tplh)/2.

Phase delay

- Same as insertion delay

Cell delay

- For any gate it is measured between 50% of input transition to the corresponding 50% of output transition.

Intrinsic delay

- Intrinsic delay is the delay internal to the gate. Input pin of the cell to output pin of the cell.
- It is defined as the delay between an input and output pair of a cell, when a near zero slew is applied to the input pin and the output does not see any load condition.It is predominantly caused by the internal capacitance associated with its transistor.
- This delay is largely independent of the size of the transistors forming the gate because increasing size of transistors increase internal capacitors.

Extrinsic delay

- Same as wire delay, net delay, interconnect delay, flight time.

- Extrinsic delay is the delay effect that associated to with interconnect. output pin of the cell to the input pin of the next cell.

Input delay

- Input delay is the time at which the data arrives at the input pin of the block from external circuit with respect to reference clock.

Output delay

- Output delay is time required by the external circuit before which the data has to arrive at the output pin of the block with respect to reference clock.

Exit delay

- It is defined as the delay in the longest path (critical path) between clock pad input and an output. It determines the maximum operating frequency of the design.

Latency (pre/post cts)

- Latency is the summation of the Source latency and the Network latency. Pre CTS estimated latency will be considered during the synthesis and after CTS propagated latency is considered.

Uncertainty (pre/post cts)

- Uncertainty is the amount of skew and the variation in the arrival clock edge. Pre CTS uncertainty is clock skew and clock Jitter. After CTS we can have some margin of skew + Jitter.

Unateness

- A function is said to be unate if the rise transition on the positive unate input variable causes the ouput to rise or no change and vice versa.

- Negative unateness means cell output logic is inverted version of input logic. eg. In inverter having input A and output Y, Y is -ve unate w.r.to A. Positive unate means cell output logic is same as that of input.

- These +ve ad -ve unateness are constraints defined in library file and are defined for output pin w.r.to some input pin.

- A clock signal is positive unate if a rising edge at the clock source can only cause a rising edge at the register clock pin, and a falling edge at the clock source can only cause a falling edge at the register clock pin.

- A clock signal is negative unate□ if a rising edge at the clock source can only cause a falling edge at the register clock pin, and a falling edge at the clock source can only cause a rising edge at the register clock pin. In other words, the clock signal is inverted.

- A clock signal is not unate if the clock sense is ambiguous as a result of non-unate timing arcs in the clock path. For example, a clock that passes through an XOR gate is not unate because there are nonunate arcs in the gate. The clock sense could be either positive or negative, depending on the state of the other input to the XOR gate.

Jitter

- The short-term variations of a signal with respect to its ideal position in time.

- Jitter is the variation of the clock period from edge to edge. It can varry +/- jitter value.

- From cycle to cycle the period and duty cycle can change slightly due to the clock generation circuitry. This can be modeled by adding uncertainty regions around the rising and falling edges of the clock waveform.

Sources of Jitter

- Internal circuitry of the phase-locked loop (PLL)

- Random thermal noise from a crystal

- Other resonating devices

- Random mechanical noise from crystal vibration

- Signal transmitters

- Traces and cables

- Connectors

- Receivers

Skew

- The difference in the arrival of clock signal at the clock pin of different flops.
- Two types of skews are defined: Local skew and Global skew.

Local skew

- The difference in the arrival of clock signal at the clock pin of related flops.

Global skew

- The difference in the arrival of clock signal at the clock pin of non related flops.
- Skew can be positive or negative.
- When data and clock are routed in same direction then it is **Positive skew.**
- When data and clock are routed in opposite then it is **negative skew.**

Recovery Time

- Recovery specifies the minimum time that an asynchronous control input pin must be held stable after being de-asserted and before the next clock (active-edge) transition.
- Recovery time specifies the time the inactive edge of the asynchronous signal has to arrive before the closing edge of the clock.
- Recovery time is the minimum length of time an asynchronous control signal (eg.preset) must be stable before the next active clock edge. The recovery slack time calculation is similar to the clock setup slack time calculation, but it applies asynchronous control signals.

Equation 1:

- Recovery Slack Time = Data Required Time â€“ Data Arrival Time
- Data Arrival Time = Launch Edge + Clock Network Delay to Source Register + Tclkq+ Register to Register Delay
- Data Required Time = Latch Edge + Clock Network Delay to Destination Register =Tsetup

If the asynchronous control is not registered, equations shown in Equation 2 is used to calculate the recovery slack time.

Equation 2:

- Recovery Slack Time = Data Required Time â€“ Data Arrival Time

- Data Arrival Time = Launch Edge + Maximum Input Delay + Port to Register Delay

- Data Required Time = Latch Edge + Clock Network Delay to Destination Register Delay+Tsetup

- If the asynchronous reset signal is from a port (device I/O), you must make an Input Maximum Delay assignment to the asynchronous reset pin to perform recovery analysis on that path.

Removal Time

- Removal specifies the minimum time that an asynchronous control input pin must be held stable before being de-asserted and after the previous clock (active-edge) transition.

- Removal time specifies the length of time the active phase of the asynchronous signal has to be held after the closing edge of clock.

- Removal time is the minimum length of time an asynchronous control signal must be stable after the active clock edge. Calculation is similar to the clock hold slack calculation, but it applies asynchronous control signals. If the asynchronous control is registered, equations shown in Equation 3 is used to calculate the removal slack time.

- If the recovery or removal minimum time requirement is violated, the output of the sequential cell becomes uncertain. The uncertainty can be caused by the value set by the resetbar signal or the value clocked into the sequential cell from the data input.

Equation 3

- Removal Slack Time = Data Arrival Time â€“ Data Required Time

- Data Arrival Time = Launch Edge + Clock Network Delay to Source Register + Tclkq of Source Register + Register to Register Delay

- Data Required Time = Latch Edge + Clock Network Delay to Destination Register + Thold

- If the asynchronous control is not registered, equations shown in Equation 4 is used to calculate the removal slack time.

Equation 4

- Removal Slack Time = Data Arrival Time â€“ Data Required Time

- Data Arrival Time = Launch Edge + Input Minimum Delay of Pin + Minimum Pin to Register Delay

- Data Required Time = Latch Edge + Clock Network Delay to Destination Register +Thold
- If the asynchronous reset signal is from a device pin, you must specify the Input Minimum Delay constraint to the asynchronous reset pin to perform a removal analysis on this path.

21. **What are Wire-load models?**
22. **What are multi-cycle paths and false paths in an SDC file?**
23. **If Flip flops are having 1000 Drive strength of cells, How to do cloning for the FF?**
24. **If Combinational circuit is having 1000 Drive strength of cells, How to do cloning?**
25. **How are nets crossing multi-clock domain handled?**
26. **What is difference between buffering and cloning?**

How do we eliminate slack if it occurs during First optimization stage (trial routing)?

9. Design Rule Check – DRC

1. **What are the violations are solved in DRC?**

includes the following 65 and 90nm design rules:

• Fat metal width spacing rule
• Fat metal extension spacing rule
• Maximum number minimum edge rule
• Metal density rule (requires a Hercules license)
• Via density rule (requires a Hercules license)
• Fat metal connect rule
• Via corner spacing rule
• Minimum length rule
• Via farm rule
• Enclosed via spacing rule
• Minimum enclosed spacing rule
• Fat poly contact rule
• extendMacroPinToBlockage (new parameter)
• Special end-of-line spacing rule
• Special notch rule
• U-shaped metal spacing rule
• Maximum stack level for via (for array)
• Stud spacing
• Multiple fat spacing
• Enclosure

2. **What is the Difference between Magma and Caliber for solving the Problem of DRC LVS?**

Magma is an implementation tool, this does only metal level DRC, But Caliber is a sign off tool, It does DRC in POLY and Diffusion level.

In a reg to reg path if you have setup problem where will you insert buffer-near to launching flop or capture flop? Why?

- (buffers are inserted for fixing fanout voilations and hence they reduce setup voilation; otherwise we try to fix setup voilation with the sizing of cells; now just assume that you must insert buffer !)
- Near to capture path.
- Because there may be other paths passing through or originating from the flop nearer to lauch flop. Hence buffer insertion may affect other paths also. It may improve all those paths or degarde. If all those paths have voilation then you may insert buffer nearer to launch flop provided it improves slack.

10. Layout Versus Schematic – LVS

1. **What are the violations are solved in LVS?**
   - Open Error - Short Error
   - Device Mismatch - Port Mismatch
   - Instance Mismatch - Net Mismatch
   - Floating Nets

11. Parasitic Extraction – XRC

1. **What is the significance of RC extraction?**
2. **What is the Difference between 2.5D and 3D parasitic Extraction?**
3. **What is LPE? Explain?**
4. **Why Parasitic Extraction for only R and C, why not L(inductor) ?**
5. **Why filler cell are used?**
   For P-Well and N-Well Connectivity

12. Power Analysis

1. **Why is IR drop analysis done?**
2. **How to do IR Drop analysis? What kind of information does it needs?**

During power analysis, if you are facing IR drop problem, then how did you avoid?

a. Increase power metal layer width.
b. Go for higher metal layer.
c. Spread macros or standard cells.
d. Provide more straps.

What sort of general information is available about the commands involved in the overall flow through Astro-Rail?

Answer:

The following provides a list of facts about the Astro-Rail analysis flow. We will first break the flow into stages and then provide details about each individual stage. This division is as follows:

1. Data preparation
2. Loading power-related information
3. Power analysis
4. Power/ground net extraction
5. Rail analysis
6. Viewing the violations
7. White box and gray box creation
8. Use of white box and gray box though the Astro-Rail flow

===================
1. Data preparation
===================

All CEL and FRAM views MUST be in the same library.

Technology file:
- Check if the technology section has the unitVoltageName/Precision, unitCurrentName/Precision, and unitPowerName/Precision values defined for IR analysis.
- Check if the metal and via layer sections have the maxCurrentDensity value defined for IR analysis.

Library views:
- Check if the TIM views and PWR views exist in the library.
- Recreate the PWR views if you do not trust those created by loading the CLF files that are generated by converting a .lib file.
  - Alternatively, starting from Astro-Rail version U-2003.09, you can use LM views for power information.
- Check if the PWR views contain one or all of the following information, depending on what was in the .lib file:
  - Short-circuit power
  - Leakage power
  - Internal power

All hard macros (including I/O pad cells) need the CONN views and current source files (CSF) if you wish to see the PG mesh inside them.

DO NOT make CONN views for standard cells or soft macros.

====================================
2. Loading power-related information
====================================

Scheme-based net switching:

- Do not use pattern matching unless necessary because it takes longer than specifying the net names. It is suggested that, however, you do pattern matching once and then write the net switching information out to an output file with the poDumpPowerInfo command. This output file contains a list of the nets and their switching activity in the Scheme format, so the file can be loaded directly back to the tool using the poLoadNetSwitchingInfo command.

- Verify the activity value calculations.

Example:

The netSwitching value is the average toggling counts of the net per unitTimeName. If the clock frequency is at 100 MHz, then the clock signal toggles twice in 10 ns, once up and once down. Thus the netSwitching value is 2/10 = 0.2. If a given net was to be switching at 20% of the clock frequency, you would specify a value of 0.04 (20% x 0.2) for that given net.

VCD-based net switching:
- Verify the header to be removed (scope-upscope) correctly. Here you are removing the additional hierarchy that may exist in the VCD file but not in the verilog netlist.
- Verify that the VCD file is for the netlist that was loaded into Astro-Rail.
  If you have performed optimizations on the original netlist in Astro, then this is NOT the same netlist because the optimization introduces new buffers and nets to the design.
- When doing frame-by-frame, the width must be a multiple of what is in the VCD's time steps.

Load Transition Times:
- Use this to debug when you get the message, "ERROR: Fail to get delay/trans thresholds," in the Power Analysis stage.
- Use this to save time. You do not have to run the STA in power analysis when selecting the General Power Model option.
- Remember to DESELECT the Purge Power Info Before Loading option.
- Remember that you need to select the Load Transition Time Only option in the Power Analysis (poPowerAnalysis) dialog box.
- The syntax used in the text file:

```
  definePortInstanceTransition cellID "cellInstName" "portName" transTime
```

=================
3. Power analysis
=================

Power analysis is a GIGO process. The accuracy of the power calculation results depends on,
- Accurate net switching activities
- Accurate .lib cell characterization for timing in TIM or LM views
- Correct signal parasitics

Power Models:

  Switching Power Only = 0.5(C*V^2)freq

General Power Model = Switching + Leakage + Internal + Short-circuit Power

The Astro STA engine will be invoked if the General Power Model option is selected. This is because the internal and short-circuit power are dependent on the rise/fall time.

To see all four different power types, enter the following:

```
  define poPrintCellPower 1
```

This will produce the following syntax in the log file:

```
   InstanceName SwithcingPower Short-circuitPower LeakagePower InternalPower
       TotalPower
```

Be careful that setting poPrintCellPower prints a line for every cell in the design and thus results in a pretty massive log file.

Defining Cell instance power:
- Do NOT select the Load Cell Instance Power Only option in general because this will ONLY load the values specified in the file that is loaded to the tool by specifying the file name in the Cell Instance Power File text field.

- Do specify all the hard macro, I/O, gray box and white box power values in an ASCII file and load it via the Cell Instance Power File option field. The syntax used in this file is defineCellInstancePower.

- If no power consumption value is specified for the white box or gray box modeled cells, the value zero will be assigned.
- The defineCellInstancePower values specified are used to scale power
  Consumption values of white box and gray box models and the CONN+CSF data Running Astro LPE by using the poPowerAnalysis command does not create the PARA view data.

The poSignalExtraction command runs LPE hierarchically and then saves the PARA view data for soft macros.

Common Errors:

```
  ERROR: Fail to get delay/trans thresholds
  Error: fail to load CG on cell BLOCK
  Error: fail to prepare top-level timing info
  Error: power analysis failed
```

These errors are generated because STA failed and Astro-Rail tried to run the astReportTiming command.
The cause of the failure may be one of the following:
- The subcell in the block in question is not present (missing placement in LEF/DEF flow)
- The TIM view for the block in question is corrupted
- Missing reference libraries for the block (analog block, like PLL). You can remove the CEL view of the block that causes the problem and leave the FRAM view, so the PLL block is treated as a black box.

Log file:

Following explains the power analysis data processing recorded in the log file.

- Every cell in the design is treated as though it was a standard cell.

```
 Power analysis succeeded.
 Log file extract:
 IO net switching power = 0 mW Static cell-inst power calculation:
    total switching power = 160.562 mW (23.4036 %)
    total short-circuit power = 77.7475 mW (11.3325 %)
    total internal power = 446.576 mW (65.093 %)
    total leakage power = 1.17244 mW (0.170895 %)
    total power = 686.057 mW                  <====== TOTAL POWER
```

- Then the user-specified values overwrite the previously calculated values.

```
 Loading cell instance power...
 Log file extract:
 Cell instance power has been loaded and updated.
 Total static cell-inst power and current:
  total power and current at power net A_VDD1 = 0.0865513 mW, 0.0721261 mA
  total power and current at power net A_VDD2 = 0.0865513 mW, 0.0721261 mA
  total power and current at power net D_VDD1 = 75 mW, 62.5 mA
  total power and current at power net D_VDD2 = 143.4 mW, 119.5 mA
  total power and current at power net D_VDD3 = 394.092 mW, 328.41 mA
  total power and current at power net D_VDD4 = 325.6 mW, 271.334 mA
  total power = 938.266 mW                    <====== TOTAL POWER
```

The log file does state the final power values for each macro cell instances. Look for "Calculated power values for macro cell instances" and then the "Final power values for macro cell instances".

==============================
4. Power/ground net extraction
==============================

You must do extraction on each power and ground net, like VSS and VDD. There MUST be a FRAM or CONN view present in the same library as the CEL view that is being extracted.

Common Error:

If any error occurs due to zero boundary nodes, resistors or current sources, do the following:

- Check the block in question to see what is present in the FRAM, CEL, or CONN view for the net.
- Determine which block to investigate errors by checking the log file. For example, if the log file contains the following information, the block in question to investigate is "core".

```
 Extracting net VDD of cell core from library corelib ...
```

Common log message:

```
  WARNING : no intersection between PAD location and layout
```

- This warning is not a concern if the top-block being extracted contains pad cells.
- The poPGExtraction command recognizes pin objects during net extraction.
  If no pins but only pads exist at the top level, then this warning makes sense and you need to investigate the block-level.

The investigation uses grep "Design Hierarchy" table in the log file reported by the tool. Check for the following:

- Missing macros?
  - Missing reference library
  - FRAM view not in same library as CEL/CONN view

Use the poCleanupExtraction command to control the power and ground contents saved in the PARA view. Be careful of the default settings which remove all the data.

================

5. Rail analysis

================

Performing rail analysis is usually time-consuming. Rail Analysis combines the power information (current consumption) of the cells and the PG resistive mesh in a matrix to solve. The log file reports the following:

```
 Setup Compressed Matrix, number of equations #
 Matrix solver: Total number of elements #
```

The number of equations and elements along with how "meshy" the resistive network is gives an indication to runtime and memory usage.

Common Error:

```
 ERROR : All matrix nodes are floating
```

This message indicates that the sources of ideal voltage do not overlap with the actual material of the net in question. Verify if the user-defined taps or pin objects are overlapping with the power or ground net. Check if the tap is not located over a pin.

```
 ERROR : fail to scan cell instance power id
```

This is primarily because a cell in the design has not been annotated with a power consumption value. You can check on this by using poDumpPowerInfo or poQueryPowerInfo to write the power-related information out to a file or by using the poDisplayPowerMap to see this graphically.

========================

6. Viewing the violations

========================

After rail analysis is complete, you can view the voltage drop map by using poDisplayVoltageDropMap. Here you can investigate the maximum voltage drop and select the Step, Metal, Via Bounds option to isolate areas and layers of interest. To investigate a spot on the layout, click the Query button and a "spot value" is returned. Following is an example of querying voltage drop values:

```
  R = 261e-3, layer: METAL2, bounding box: (295.290, 17.290) -
          (304.538,19.790) Current direction: LEFT Voltage Drop = (-2.815)
```

You can also produce error cells and reports for locations where user-defined limits have been violated.

You can investigate violations within white box models by using the poDispalyParasitic command. The command displays the PG resistive network from the PARA view in the CEL view. In the parasitic map of the PG mesh, you will be able to see,

Yellow crossed box: ideal voltage source (boundary node)
Blue crossed box: current source location

===================================
7. White box and gray box creation
===================================

The steps of creating white box and gray box for hard macros are different from the steps for soft macros. In Astro-Rail, a hard Macro is usually an CEL that has been streamed in from GDSII, such as a RAM. A soft macro is a block that has been placed and routed in Astro.

A. White box and gray box creation for hard macros
Starting with a hard macro's CONN view and a current source file (CSF).
i. Run poLoadPowerSupply to define power supplies.
ii. Run poPGExtraction to do P/G net extraction. This automatically generates a white box model.
iii. Run poGenMacroModel to create a gray box (macro) model.

B. White box and gray box creation for soft macros Starting with a soft macro's CEL view. Leaf cells and interconnect routing.
i. Run poLoadPowerSupply to define power supplies.
ii. Run poLoadNetSwitchingInfo to load net switching activities.
iii. Run poPowerAnalysis to do power analysis.
iv. Run poPGExtraction to do P/G net extraction
v. Run poGenWhiteBox to create a white box model (optional).
vi. Run poGenMacroModel to create a gray box (macro) model.

===
8. Use of white box and gray box though the Astro-Rail flow
===

A. Power analysis

If there are CEL views of sub-blocks and the Flatten Hierarchical Cells option is selected, the power analysis will dive down into the CEL views, even if there is a white box or gray box.

The default power value for a white box and a gray box is ZERO even if you created a white box and it has the model power. White box and gray box power instance value MUST be specified in an ASCII file and load it to the tool by specifying the file name in the Cell Instance Power File text field in the Power Analysis dialog box.

The power analysis results are NOT automatically saved, so you MUST do a save. The power info will then be saved in the CEL view.

B. Power/Ground net extraction

If there are white box or gray box models for macros (subblocks) and the Flatten Hierarchical Cells option is selected, the order of preference for a soft macro is:

Gray box, White box, CEL (Soft Macro), FRAM (black box)

And the order of preference for a hard macro is:

Gray box, White box, CONN (Hard Macro), FRAM (black box)

Currently no method is provided to select which collection of data to be used for a child cell of the cell being analyzed. The order of precedence for power and ground net information is as follows:

```
  Hard Macro: Gray-box model, then White-box model,
              then CONN view and then FRAM view (BBox)
  Soft Macro: Gray-box model, then White-box model,
              then CEL view and then FRAM view (BBox)
```

The results of the power and ground net extraction are automatically saved into
the PARA view.

C. Rail analysis

If there are white box or gray box models for macros (subblocks) and the Flatten Hierarchical Cells is selected, the order of preference is:

Hard Macro:
Gray box, White box, CONN (Hard Macro), FRAM (BBox)

Soft Macro:
Gray box, White box, CEL (Soft Macro), FRAM (BBox)

The poRailAnalysis command has also been enhanced with another option for the treatment of hierarchical cells. As of version U-2003.09, you can use the hierarchy browser to select which of the available data that is present will be used for the rail analysis.

====================================

More detailed Astro-Rail information
====================================
Astro-Rail User Guide
SolvNet - Application Notes (search criteria: Astro-Rail)
SolvNet - Q & A

============================
TLAs (Three Letter Acronyms)
============================

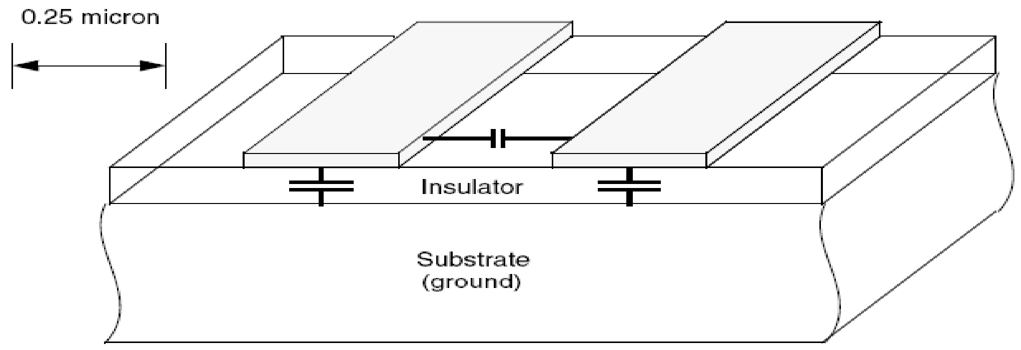
CEL	: CEL view in Milkyway
FRAM	: FRAM view in Milkyway. An abstract view that contains just enough information for place and route tool.
CONN	: CONN view in Milkyway, the connectivity view of the power and ground nets
CSF	: Current Source file. A text file that contains current source and location information
PG	: Power and ground
TIM	: Timing Model view in Milkyway, containing timing arcs description of cells.
LM	: Logic Model view in Milkyway
PWR	: Power view in Milkyway
Gray box	: Grey Box Model
White box	: White Box Model
BBox	: Black Box Model

13. Cross Talk Analysis

Signal integrity is the ability of an electrical signal to carry information reliably and resist the effects of high-frequency *electromagnetic interference* from nearby signals.

Crosstalk is the undesirable electrical interaction between two or more physically adjacent nets due to capacitive cross-coupling.

As integrated circuit technologies advance toward smaller geometries, crosstalk effects become increasingly important compared to cell delays and net delays.



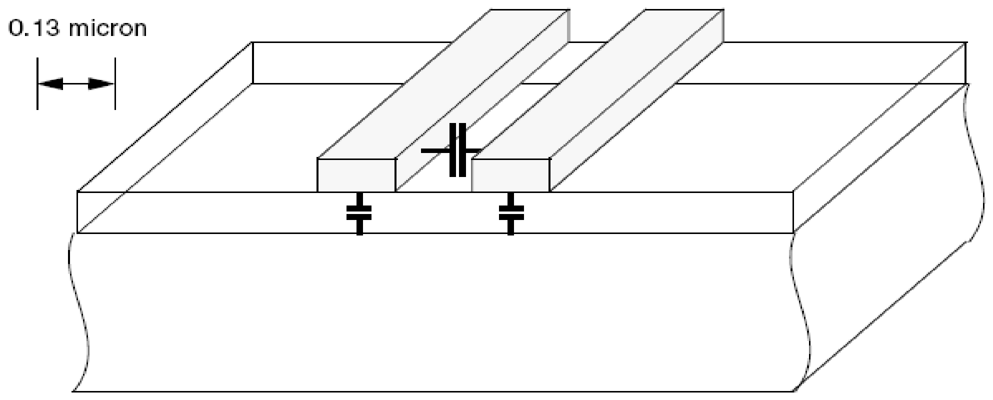


Figure1-1 shows an enlarged view of two parallel metal interconnections in an integrated circuit, first for a 0.25-micron technology and then for a 0.13-micron technology.

As circuit geometries become smaller, **wire** interconnections **become closer** together and **taller**, thus increasing the cross-coupling capacitance between nets. At the same time, parasitic capacitance to the substrate becomes less as interconnections become narrower, and cell delays are reduced as transistors become smaller.

With circuit geometries at 0.25 micron and above, substrate capacitance is usually the dominant effect. However, with geometries at 0.18 micron and below, the coupling capacitance between nets becomes significant, making crosstalk analysis increasingly important for accurate timing analysis.

The Miller Effect

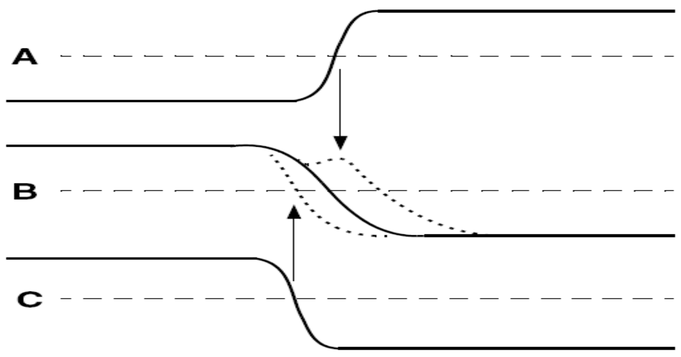
The Miller effect describes the extent to which sudden changes in voltage level of adjacent nets cause the coupling portion of the capacitive load to increase or decrease as discerned by the victim nets driving gate. The positive or negative impact of the Miller effect is dependent on the relative switching directions of aggressor and victim nets.

• When aggressor nets switch in the **opposite** direction from a victim net, a *positive* **Miller effect increases the capacitance** between them and magnifies delay. If aggressor and victim slew rates are equal, the late timing (setup) coupling capacitance doubles because the voltage difference is doubled.

• When aggressor nets switch in the **same** direction as a victim net, a *negative* **Miller effect reduces the capacitance** between them and accelerates timing. If aggressor and victim slew rates are equal, the early timing (hold) coupling capacitance is negated.

Crosstalk Delay Effects

Crosstalk can affect signal delays by changing the times at which signal transitions occur. For example, consider the signal waveforms on the cross-coupled nets A, B, and C in Figure 1-2



Because of capacitive cross-coupling, the transitions on net A and net C can affect the time at which the transition occurs on net B. A **rising-edge transition** on net A at the time shown in Figure1-2 can cause the transition to occur later on net B, possibly contributing to a **setup violation** for a path containing B. Similarly, a **falling-edge transition** on net C can cause the transition to occur earlier on net B, possibly contributing to a **hold violation** for a path containing B.

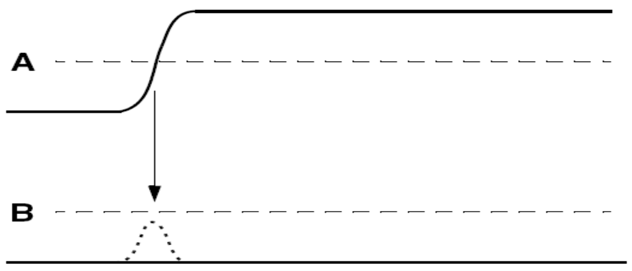
A Miller factor (a number typically between 0 and 2) is used to scale coupling capacitance for wire load correction. When there is a positive Miller effect between nets (because they switch in opposite directions), delay is scaled up by multiplying the coupling capacitance between the nets by the maximum Miller factor. The typical value is 2, assuming equal slew rates.

• When there is a negative Miller effect between nets (because they switch in the same direction), hold time is scaled down by multiplying the coupling capacitance between them by the minimum Miller factor. When aggressor and victim slew rates are equal, the typical value is 0. This implies that the effect of coupling capacitance is negated.

• When adjacent aggressor signals are not switching at the same time as their victim net, delay may not be affected. The coupling capacitance between the nets is multiplied by a nominal Miller factor, typically 1.

Crosstalk Noise Effects

Figure 1-3 Glitch Due to Crosstalk



In Figure 1-3 Net B should be constant at logic zero, but the rising edge on net A causes a noise bump or glitch on net B. If the bump is sufficiently large and wide, it can cause an incorrect logic value to be propagated to the next gate in the path containing net B.

Aggressor and Victim Nets
A **net** that **receives** undesirable cross-coupling effects from a nearby net is called a **victim net**. A **net** that **causes** these effects in a victim net is called an **aggressor net**. Note that an aggressor net can itself be a victim net; and a victim net can also be an aggressor net. The terms aggressor and victim refer to the relationship between two nets being analyzed.

The timing impact of an aggressor net on a victim net depends on several factors:
• The amount of **cross-coupled capacitance**
• The **relative times and slew rates** of the signal transitions
• The **switching directions** (rising, falling)
• The **combination of** effects from **multiple aggressor** nets on a single victim net

Figure 1-4 Effects of Crosstalk at Different Arrival Times

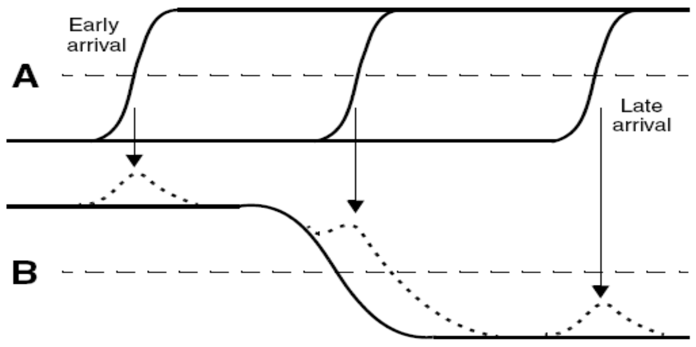


Figure 1-4 illustrates the importance of timing considerations for calculating crosstalk effects. The aggressor signal A has a range of possible arrival times, from early to late.

If the transition on A occurs at about the same time as the transition on B, it could cause the transition on B to occur later as shown in the figure, possibly contributing to a setup violation; or it could cause the transition to occur earlier, possibly contributing to a hold violation.

If the transition on A occurs at an early time, it induces an upward bump or glitch on net B before the transition on B, which has no effect on the timing of signal B. However, a sufficiently large bump can cause unintended current flow by forward-biasing a pass transistor.

Similarly, if the transition on A occurs at a late time, it induces a bump on B after the transition on B, also with no effect on the timing of signal B. However, a sufficiently large bump can cause a change in the logic value of the net, which can be propagated down the timing path.

Timing Windows and Crosstalk Delay Analysis
There are three analysis modes with respect to operating conditions: single, best_case/worst_ case, and on-chip variation. PT SI uses the on-chip variation mode to derive the timing window relationships between aggressor nets and victim nets.

Using the on-chip variation mode, PT SI finds the earliest and the latest arrival times for each victim net and aggressor net. **The range of switching times, from earliest to latest arrival, defines a timing window for the victim net, and defines another timing window for the**

aggressor net. Crosstalk timing effects can occur only when the victim and aggressor timing windows overlap.

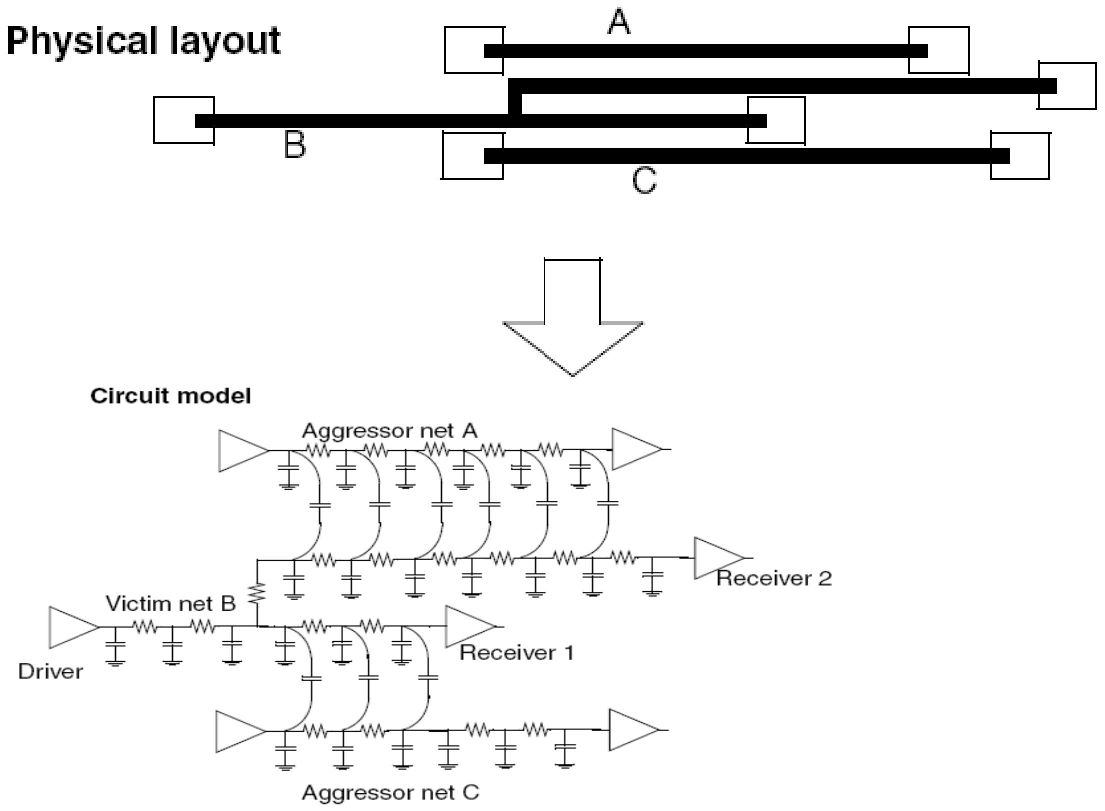
PT SI performs crosstalk analysis using multiple iterations. For the first iteration, it ignores the timing windows and assumes that all transitions can occur at any time. This results in a pessimistic but fast analysis that gives approximate crosstalk delay values.

In subsequent analysis iterations, PT SI considers the timing windows and eliminates some victim-aggressor relationships from consideration, based on the lack of overlap between the applicable timing windows.

When an overlap occurs, PT SI calculates the effect of a transition occurring on the aggressor net at the same time as a transition on the victim net. The analysis takes into account the drive strengths and coupling characteristics of the two nets.

Cross-Coupling Models

Figure 1-5 shows the physical layout for a small portion of an integrated circuit, together with a detailed model of the circuit that includes cross-coupled capacitance. Each physical interconnection in the design has some distributed resistance along the conductor and some parasitic capacitance to the substrate (ground) and to adjacent nets. The model divides each net into subnets and represents the distributed resistance and capacitance as a set of discrete resistors and capacitors.



A detailed model such as this can provide a very accurate prediction of crosstalk effects in simulation. For an actual integrated circuit, however, a model might have too many circuit

elements to process in a practical amount of time. Given a reasonably accurate (but sufficiently simple) network of cross-coupled capacitors from an external tool, PT SI
can obtain accurate crosstalk analysis results in a reasonable amount of time.

If you need to account for possible timing effects of crosstalk, using PT SI is much easier, faster, and more thorough than using a circuit simulator such as SPICE. Instead of analyzing just a single path or a few paths for crosstalk effects, PT SI lets you analyze the whole circuit using the familiar PT analysis flow.

To use PT SI with PT, you only need to do the following additional steps:

• Enable PT SI by setting the **si_enable_analysis** variable to true.

• Back-annotate the design with **cross-coupling capacitor** information, as specified in a Standard Parasitic Exchange Format (SPEF) or Detailed Standard Parasitic Format (DSPF) file.

• Specify the parameters that determine the accuracy and speed of the crosstalk analysis effort, such as the number of analysis iterations and the capacitance values that can be safely ignored.

Usage Flow

Here is an example of a script that uses crosstalk analysis, with the crosstalk-specific items shown in boldface:

```
set_operating_conditions -analysis_type on_chip_variation
set si_enable_analysis TRUE
read_db ./test1.db
current_design test1
link_design
read_parasitics -keep_capacitive_coupling SPEF.spf
create_clock -period 5.0 clock
check_timing -include { no_driving_cell ideal_clocks \
partial_input_delay unexpandable_clocks }
report_timing
report_si_bottleneck
report_delay_calculation -crosstalk -from pin -to pin
```

The set_operating_conditions command sets the analysis type to on_chip_variation, which is necessary to allow PT SI to correctly handle the
min-max timing window relationships. If you do not specify this analysis type explicitly,
PT SI automatically switches to that mode.

Setting si_enable_analysis to true enables crosstalk analysis using PT SI.
The -keep_capacitive_coupling option in the read_parasitics command is
necessary to maintain the cross-coupling status of capacitors that have been read into
PT.

In addition to IEEE 1481 Standard Parasitic Exchange Format (SPEF), PT SI can also read parasitic data in Synopsys Binary Parasitic Format (SBPF) or Detailed Standard Parasitic Format (DSPF). To read data in this format, use a command similar to the following:

pt_shell> **read_parasitics -keep_capacitive_coupling -format SBPF mydata.sbpf**

How PT SI Operates

Electrical filtering:
This means removing from consideration the aggressor nets whose effects are too small to be significant, based on the calculated sizes of bump voltages on the victim nets. You can specify the threshold level that determines which aggressor nets are filtered.

When filtering occurs, the aggressor net and the coupling capacitors connect to it are not considered for analysis between that victim net and aggressor net. If the bump height contribution of an aggressor on its victim net is very small (less than 0.00001 of the victim's nominal voltage), this aggressor is automatically filtered.

Filtering eliminates aggressors based on the size of the voltage bump induced on the victim net by the aggressor net. The bump sizes depend on the cross-coupling capacitance values, drive strengths, and resistance values in the nets.

An aggressor net is filtered if the peak voltage of the noise bump induced on the victim net, divided by Vdd (the power supplyvoltage), is less than the value specified by this variable: By default, this variable is set to 0.01

si_filter_per_aggr_noise_peak_ratio

Initial Net selection:
After filtering, PT SI selects the initial set of nets to be analyzed for crosstalk effects from those not already eliminated by filtering. You can optionally specify that certain nets be included in, or excluded from, this initial selection set.

Delay Calculation:
The next step is to perform delay calculation, taking into account the crosstalk effects on the selected nets. This step is just like ordinary timing analysis, but with the addition of crosstalk considerations.

PT SI Variables
si_analysis_logical_correlation_mode true
For a faster but more pessimistic analysis, you can disable logical correlation consideration. To do so, set the variable si_analysis_logical_correlation_mode to false. This variable is set to true by default.

si_ccs_aggressor_alignment_mode lookahead
si_ccs_use_gate_level_simulation true
si_enable_analysis false
si_filter_accum_aggr_noise_peak_ratio 0.03
si_filter_per_aggr_noise_peak_ratio 0.01
si_noise_composite_aggr_mode disabled
si_noise_slack_skip_disabled_arcs false
si_xtalk_composite_aggr_mode disabled
si_xtalk_composite_aggr_noise_peak_ratio 0.01

```
si_xtalk_composite_aggr_quantile_high_pct 99.73
si_xtalk_delay_analysis_mode all_paths
si_xtalk_double_switching_mode disabled
si_xtalk_exit_on_max_iteration_count 2
si_xtalk_exit_on_max_iteration_count_incr 2
si_xtalk_reselect_clock_network true
```

if a combination of smaller aggressors is below a different, larger threshold, all of those smaller aggressors are filtered. This threshold is set by the variable si_filter_accum_aggr_noise_peak_ratio. If the combined height of smaller noise bumps, divided by Vdd, is less than this variable setting, all of those aggressors are removed from consideration for that set of analysis conditions. The default setting for this variable is 0.03.

Capacitive Coupling Data
Make sure that your parasitic capacitance extraction tool generates an IEEE-standard Standard Parasitic Exchange Format (SPEF) or Detailed Standard Parasitic Format (DSPF) file with coupling capacitors, and that the tool settings generate a reasonable number of capacitors.

If your extraction tool supports the filtering of small capacitors based on a threshold, it might be more efficient to let the extraction tool rather than PT SI do electrical filtering.

To further trade accuracy for simplicity, you might consider limiting the number of coupling capacitors per aggressor-victim relationship.

PT SI ignores any cross-coupling capacitor between a net and itself. If possible, configure your extraction tool to suppress generation of such self-coupling capacitors.

When you read in the capacitive coupling data with the **read_parasitics** command, remember to use the **-keep_capacitive_coupling** option to retain the data.

check_timing
The check_timing command can check for several conditions related to crosstalk analysis, making it easier to detect conditions that can lead to inaccurate crosstalk analysis results. It is recommended that you run check_timing after you set the constraints and before you start an analysis with update_timing or report_timing.

There are four types of checking that are specific to crosstalk analysis

• **no_driving_cell** –Any input port that does not have a driving cell and does not have case analysis set on it. When no driving cell is specified, that net is assigned a strong driver for modeling aggressor effects, which can be pessimistic.

• **ideal_clocks** – Any clock networks that are ideal (not propagated). For accurate determination of crosstalk effects, the design should have a valid clock tree and the clocks should be propagated.

• **partial_input_delay** – Any inputs that have only the minimum or only the maximum delay defined with set_input_delay. To accurately determine timing windows, PT SI needs both the earliest and latest arrival times at the inputs.

• **unexpandable_clocks** – Any clocks that have not been expanded to a common time base. For accurate alignment of arrival windows, all of the synchronous and active clocks of different frequencies must be expanded to a common time base.

Initial Crosstalk Analysis Run
For the first analysis run with crosstalk analysis, it is a good idea to use the default settings for the crosstalk variables so that you can obtain results quickly. For example:

```
pt_shell> set si_enable_analysis TRUE
pt_shell> report_timing
```

With the default variable settings, PT SI does the crosstalk analysis using two delay calculation iterations. In the first iteration, PT SI ignores the timing windows so that it can quickly get an estimate of crosstalk delay effects. In the second and final iteration, PT SI reselects only the nets in the critical path of each path group, and then does a detailed analysis of those nets considering the timing windows and transition types (rising or falling).

Additional Crosstalk Analysis Runs

There are many ways to modify the PT SI variable settings to obtain a more thorough and accurate analysis, at the cost of more execution time. A good starting point is to try the following:

```
pt_shell> set si_xtalk_reselect_min_mode_slack 0
pt_shell> set si_xtalk_reselect_max_mode_slack 0
pt_shell> report_timing
```

if you know that the clock is designed in such a way that it cannot be a victim net, then you can disable net reselection based on the change in delay calculated in the previous iteration.

To do so, set the delta delay reselection parameters to very large numbers. For example:

```
pt_shell> set si_xtalk_reselect_delta_delay 100000
pt_shell> set si_xtalk_reselect_delta_delay_ratio 100000
```

As a result of these settings, even if the clock net has a large change in calculated delay due to crosstalk, it is not reselected for analysis; only data path nets that contribute to slack violations are reselected. This can speed up the analysis considerably because of the resources that would otherwise be used for analyzing the clock net.

However, if you need to analyze clock paths in the detailed analysis, you can set either one of the delta delay variables to some meaningful value, appropriate to your technology, so that nets are reselected for analysis if they have a large enough change in calculated delay due to crosstalk.

By selecting a combination of the critical slack range and delta delay threshold, you have the ability to trade off accuracy against runtime.

Virtual Attacker:
Noise Analysis with Composite Aggressors
Some complex designs require an efficient method of analyzing multiple aggressors to a single victim net. In general, such designs have the following characteristics:

• A large number of aggressors per victim net
• Little or no filtering of aggressors
• Noise calculations are performed in high effort mode

If your design has these characteristics, you may wish to use composite aggressor analysis. Composite aggressor analysis aggregates the effects of multiple aggressors into a single "**virtual" aggressor**, thereby reducing the computational complexity cost and improving the runtime and memory utilization without impacting accuracy.

To enable this mode, set the si_noise_composite_aggr_mode variable to normal or statistical. The statistical mode applies statistical analysis to the composite aggressor to reduce its overall impact on the victim net. By default, the variable is set to disabled, which disables composite aggressor analysis.

Derating Noise Results
The set_noise_derate command modifies the calculated noise bump sizes by a specified factor or absolute amount. It works like the set_timing_derate command, except that it modifies noise bump sizes rather than path delays. The derating factors affect the reported bump sizes and noise slacks. In the command, you specify the types of noise bumps (above/below high/low), the parameters to be modified (height offset, height factor, and width factor), and the factor or absolute amount of modification. For details, see the man page for the command.

Reading and Writing Parasitic Data

The read_parasitics command reads parasitic data from a file and annotates the information on the nets of the current design.

PT SI supports the reading of parasitic data in Standard Parasitic Exchange Format (SPEF), Detailed Standard Parasitic Format (DSPF), and Synopsys Binary Parasitic Format (SBPF). Data in SBPF format occupies less disk space and can be read much faster than the same data stored in the other formats.

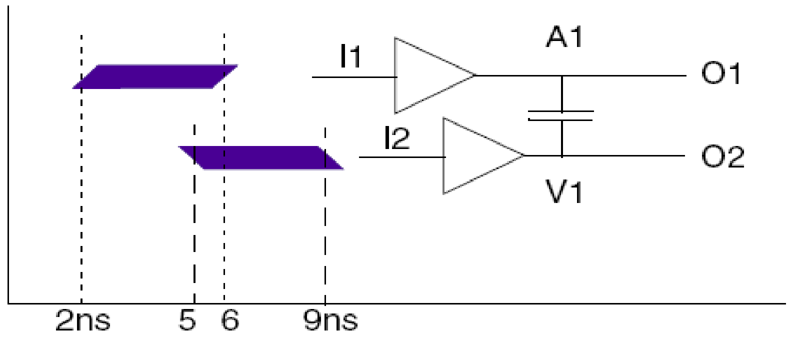
```
pt_shell> read_parasitics -keep_capacitive_coupling file1.spef
```

Timing Window Overlap Analysis
The crosstalk effect on a net depends on the alignment of the victim and aggressor switching time.

Depending on how the victim and aggressor switching times align, a net could become slower or faster depending on the switching directions. PT SI calculates the crosstalk effect based on the **timing window** of the aggressors and victim. This process is referred as **timing window overlap** analysis or aggressor alignment.

During timing window overlap analysis, PT SI calculates the crosstalk delta delay per load pin of a net. For this purpose, the timing arrival windows are used by PT SI, because it encapsulates all the timing paths passing through the net. If the aggressor partially overlaps with the victim's timing window, the partial effect (smaller delta delay) is considered.

Figure 2-4 illustrates how timing windows can overlap.

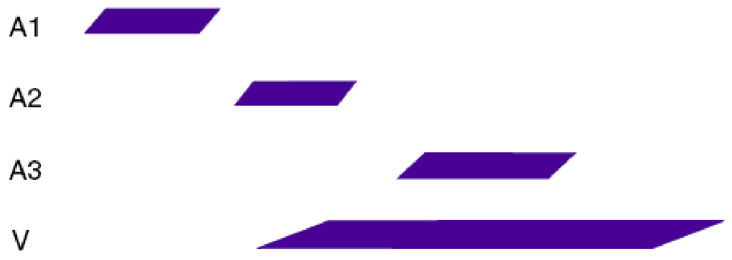


In this example, victim V1 is coupled with aggressor A1. The timing arrival windows are 2ns to 6ns for the aggressor, and 5ns to 9ns for the victim. Since the victim timing window overlaps with the aggressor's timing window, the signal integrity engine calculates the crosstalk delta delay due to this aggressor.

Sometimes, when timing windows from different clocks exist on a victim and aggressor net, PT SI considers the different combinations of these clocks with respect to the clock periods.

Multiple Aggressors. When there are multiple aggressors in the design, the signal integrity engine finds the combination of aggressors that could produce the worst crosstalk effect and calculates the crosstalk delta delay for this combination.

Figure 2-5 Multiple Aggressor Alignment



In this example, the victim has three aggressors: A1 is stronger than A2, and A2 is stronger than A3. Since aggressor A1's window does not overlap with the victim's window, and A2 is stronger than A3, the signal integrity engine will calculate the crosstalk delay due to aggressor A2. The A1 and A3 will not be considered for delta delay calculation.

The report_delay_calculation -crosstalk command will report the attributes for aggressors A1 and A3 as follows:

N - aggressor does not overlap for the worst case alignment

Asynchronous Clocks. If the victim timing window clock and the aggressor timing window clocks are asynchronous, they have no fixed timing relationship with each other. The aggressor will be treated as infinite window with respect to the victim. The report_delay_calculation -crosstalk command will report this as follows:

I - aggressor has Infinite arrival with respect to the victim

For multiple aggressors, if the aggressor clocks are synchronous with each other, but asynchronous with the victim, the timing relationships between the aggressors are respected, but they are still treated as infinite windows with respect to the victim.

Crosstalk Delay Analysis for All Paths
In the default mode, PT SI calculates the maximum possible delta delay (worst crosstalk effect) for the victim and aggressor arrival windows. This ensures that crosstalk delta delay is considered for all paths passing through the victim net, and that the maximum delta delay value is applied on that net. This guarantees that all the paths going through the victim net are conservative.

To use this type of analysis, set the si_xtalk_delay_analysis_mode variable to all_paths mode. The disadvantage of this approach is that the largest crosstalk delta delay value is applied to the critical paths, making them pessimistic. When a path is recalculated using path-based analysis, this pessimism is removed. You can also remove this pessimism by using other available crosstalk delay analysis modes.

Preventing and Correcting Crosstalk With Astro-Xtalk

Question:

When should I use Astro-Xtalk to prevent and correct crosstalk?

Answer:

Crosstalk prevention and correction can and should be performed throughout the flow by using Astro-Xtalk in the following stages:

* **Post-Placement Optimization Phase 1** (astPostPS1)

Select "Prevent Xtalk" to use cell sizing and buffer insertion on potential crosstalk problems.
Potential crosstalk problems are identified by estimating future coupling capacitances and considering the drive strengths of the driving cell of a net.

*** Global Route (axgSetXtalkRouteOptions)**

Set the following option in Crosstalk Route Options before performing Global Route. Select "Reduce potential noise & crosstalk-induced delay" under Global Routing to add extra cost to the router's cost function on potential crosstalk affected nets. The router will avoid assigning too many nets with overlapping timing windows to the same G-cell.

*** Track Assign (axgSetXtalkRouteOptions)**

Set the following option in Crosstalk Route Option before performing Track Assign. Select "Minimize noise & crosstalk-induced delay" under Track Assign to avoid putting nets with overlapping timing windows on adjacent tracks, and to estimate potential noise using the timed-peak noise model and reassign wires to reduce potential noise.

* At this stage, detailed crosstalk analysis needs to be performed.

See the *Synopsys Astro User Guide* for details.

*** In Route Optimization (axgAdvRouteOpt)**

Select "Crosstalk Noise Constraints" and "Crosstalk-induced Delay" to optimize by changing route topology, cell sizing, and buffer insertion. Parasitic, timing, and crosstalk effects are updated through the process.

Postroute Flow For Fixing Timing, Antenna, and Crosstalk

Question:

What is the best postroute flow for fixing timing, antenna, and crosstalk?

Answer:

The recommended order for fixing timing, antenna, and crosstalk is to first make sure your design has either routed cleanly or is close to routing cleanly.

Close Timing

Use the astPostRouteOpt command to improve the postroute timing result (if it is bad). Skip this step if the timing is already met.

Close Antenna Violations

Load in your antenna rules.

Select the high performance options with the SetHPORouteOptions dialog box.Use the axgSearchRepair command to fix the antennas.Select the "Insert Diode with Checking" option in the axgInsertDiode command dialog box to fix any remaining antenna violations.

Close Crosstalk

Optionally, run Astro crosstalk analysis. A noise threshold of anywhere between 0.25 and 0.45 is valid. The default value of 0.45 is often pessimistic enough. Crosstalk analysis reports the number of violations and generates crosstalk constraints for nets that almost fail crosstalk analysis, as well as for nets that have failed.

The noise threshold should also be set on the Xtalk panel of the Timing Setup dialog box. This value will be used as the default noise constraint for all nets. In this way, no previous crosstalk analysis needs to be run because the noise is recalculated against this threshold each time the timer is updated. This helps prevent new nets from becoming noise critical.

Use the ataIncludeXtalk command to take crosstalk effects into account during static timing analysis.

Use the astPostRouteOpt command to fix timing, antenna ,and crosstalk at the same time. The Crosstalk Noise Constraints and Crosstalk-Induced Delay options must be selected at this point.

If crosstalk issues persist, switch from Static to Switching Noise by selecting the Xtalk tab of the Timing Setup form and re-run the astPostRouteOpt command once more.

See the *Astro User Guide* for more information.

CRPR Value in PT SI Analysis?

Question:

I am performing signal integrity analysis with the timing_remove_clock_reconvergence_pessimism variable set to true. When I generate the report, I see a very small value for CRPR. When I issue the report_crpr command, I find two CRPR values reported, one with arrival times without crosstalk and another with arrival times with crosstalk.

My value for clock reconvergence pessimism matches with the value of CRP reported without crosstalk. Since I am doing signal integrity analysis shouldn't it have picked CRP value with crosstalk?

Answer:

The manner in which PT SI accounts for crosstalk delta delays during CRPR analysis has changed starting with the 2003.03 release to get better accuracy.

During clock reconvergence pessimism analysis, crosstalk delta delays are considered only if the aggressor switching affects both the launch and capture signals at the same time.

For a given timing check, any clock reconvergence pessimism resulting from crosstalk delta delays is removed only if precisely the same clock edge drives both the launch and capture devices. Such checks are broadly classified as zero-cycle checks.

CRP value with crosstalk will be used only with zero cycle check timing reports.

Do on-chip variation and crosstalk analysis affect SDF delay figures?

Question:

Are the delay values written by the write_sdf command affected by enabling on-chip variation or by enabling crosstalk analysis with PT SI?

Answer:

Yes, enabling on-chip variation analysis or crosstalk analysis affects the delay values written by the write_sdf command. Note that on-chip variation analysis is implicitly enabled and soon and crosstalk analysis is enabled. The min/max on-chip variation timing and the min/max delta delays are used to compute the numbers placed in the min and max SDF triplet values.

Using on-chip variation analysis or crosstalk analysis results in a more conservative SDF file. The fast delays are faster and the slow delays are slower.

With crosstalk analysis enabled, not only are the arrival windows considered, but also the coupling capacitance and victim aggressor relationship. The delay values of interconnects definitely depend on the switching of aggressor nets.If the aggressor and victim nets are changing in the same direction, it reduces RC value and thereby reduces the delay values. As a result, the net is sped up, and these delays are used for min delays. If the aggressor and victim nets are switching in opposite directions, then the RC value increases, which induces more delay. These delays are used for max delays.

You can dump out the SDF with and without enabling crosstalk to find out how the interconnect delay values are different.

Power Supply Definitions for Crosstalk

Question:

Do I need to load a power supply before running crosstalk analysis? How does crosstalk analysis work with different voltages for the same layout?

Answer:

You do not need to specify a supply voltage to run crosstalk analysis when you have only one voltage. The Astro tool uses the global pruning threshold as the constraint and reports the nets exceeding the constrained (the default on the dialog box is 0.45) noise voltage.

However, when you have multiple voltages in the design, you should load that information prior to running crosstalk analysis.

The value specified as the global noise threshold is a %Vdd. During crosstalk analysis, the noise is also reported in terms of % of Vdd.
See the following formula:

Vp = Vdd * (1/(1 + Cv/Cx + Ra/Rv * (1 + Ca/Cx)))

If you have multiple voltages, it will change as follows:

Vp = Vddvictim * (Vddagg/Vddvictim) * (1/(1 + Cv/Cx + Ra/Rv * (1 + Ca/Cx)))

Crosstalk analysis does not actually multiply the noise value with the Vddvictim
value. It basically calculates only the

(Vddagg/Vddvictim) * (1/(1 + Cv/Cx + Ra/Rv * (1 + Ca/Cx)))

part.

The ratio of **Vddagg/Vddvictim** would be 1 if you had a single supply voltage, but you would need the actual values if they were not equal.

How does PT handle SPEF cross-coupling capacitance?

Question:

My extraction tool generates SPEF parasistics with coupling capacitors between nets. But I am only using PT, without the PT SI crosstalk analysis option. What is the effect of coupling capacitance on delay calculation?

Answer:

If you are using only PT (without PT SI crosstalk analysis), the coupling capacitance is tied to ground for each coupled net. There is a switch in the read_parastics command, -coupling_reduction_factor, which you can use to scale the amount of coupling capacitance that is tied to ground. The default value is 1.0. To make your timing results more conservative, you can use a value of 2.0 for max anlaysis and 0.0 for min analysis.

What is a tap cell and how do I use it in Apollo?

Question:

I would like to know:

Answer:

1. What is a 'tap cell'

The basic idea with a global substrate tap methodology is to omit internal substrate taps from the standard cells and instead to sprinkle dedicated tap cells throughout the P&R layout. The process design rules require that no piece of source/drain diffusion (inside a standard cell) be more than some maximum distance away from the tap diffusion inside at least one of the sprinkled tap cells.

If we assume (for the time being at least) that we don't have visibility into the cells showing us the exact locations of source/drain diffusion regions and/or tap diffusion regions, the tap to diffusion spacing constraints outlined above translate into a maximum allowed distance between any part of a standard cell and the edge of a dedicated tap cell.

2. What is a 'tap layer'

A tap layer is just an extra drawing layer defined in your technology file. It is not used in a fab process and will not be streamed out.

3. Why I would want a 'tap cell'

You would want a tap cell to satisfy the design rules of your technology.

4. Why does Apollo need a 'tap layer'

Apollo can not distinguish N-tap from P-tap and therefore needs an extra layer to represent the tap layer. Apollo will calculate max distance from this tap layer to source/drain diffusion, and if needed, insert the special tap cell with the tap layer.

5. How to implement the 'tap layer' and add the tap cells

The tap layer must be drawn coincident with diffusion in the tap cell. Simply open the CEL view of your tap cell and look for the diffusion layer. Now draw RECTANGLES in the tap layer over the diffusion. Do not draw polygons. The tap cell placer cannot see tap polygons. After the tap layer is added, save the cell. Use axgAddTapCell to place these tap cells in your design.

6. Are there any hidden switches associated with the tap cell insertion function

Currently, there are two in Apollo:
a. When inserting tap cells, you will need to put a value in the field setFormField "Add Tap Cell" "Tap Distance Limit" "2x" where x is fab max tap distance. It will be required that a tap cell be a maximun of x from any given point in the core; therefore, if you place tap cells 2x apart, this condition will be satisfied. This causes tap cells to be placed 2x from the edge of the core, though, which violates the max distance requirement. To get tap cells x from the edge of the core, enter the following before running axgAddTapCell:

```
define TAPCELL_ADD_ROWENDS 1
```

b. axgAddTapCell looks for the CEL view of the tap cell by default. If the CEL view is not available you can use the FRAM view. First set the following:

```
define TAPCELL_USE_FRAMEVIEW 1
```

Remember that as with the CEL view, tap layer must be present in FRAM view for the function to work.

Check and Add Missing Tap Cell By Distance

Certification: Synopsys tested

Script Documentation

Tap cells are a special non-logic cell with well and substrate ties. These cells are typically used when most or all of the standard cells in the library contain no substrate or well taps. Generally, the design rules specify the maximum distance allowed between every transistor in a standard cell and a well or the substrate ties.

Tap cells are usually added at regular intervals in the design prior to placement of standard cells. After timing optimization, there might be some areas that are not covered within a certain distance of tap cells and that need additional tap cells to avoid latch up issues.

To check for missing tap cells, the tap cells defined in the library need to have various attributes, layer, and so forth, defined. This might require modification of the technology file and library in which the tap cell resides.
However, it might not be feasible to do so.

The attached script checks for missing tap cells in the design, based on distance. The script takes the tap distance and tap MASTER name as input. It will check for missing tap cells on the left and right side by the specified distance. It will also add these missing tap cells and legalize the design and ECO route the broken connections.

Note:
You might have to run this script multiple times because legalize_placement might move the standard cells to adjacent rows where the tap cell might not be present within the specified distance.

Inserting Well Taps Early

Question:

I am currently using the axgAddTapCell command to insert taps late in my flow, but I am concerned about this. If a logic cell has to be moved to meet well tap DRC rules, then timing might get worse. Can I insert well taps early in the flow before placement?

Answer:

After the chip or block is floorplanned, just before the first placement command is run, use the axgArrayTapCell command to to place tap cells at consistently spaced intervals throughout the placement area.

Here is an example of the axgArrayTapCell command with some options selected:

```
axgArrayTapCell
setFormField "Array Tap Cell" "Tap Master Name" "welltie_hivt_3"
setFormField "Array Tap Cell" "Pattern" "Stagger Every Other Row"
setFormField "Array Tap Cell" "Tap Cell Distance in Array" "56"
formOK "Array Tap Cell"
```

Be aware of the relationship between spacing and design rules for well taps:
The spacing is 2X the design rule. If a well tap is required to be within 28 microns of every standard cell, well taps need to be placed every 56 microns.

Here is an example of a required **tap cell layer in a .tf (techfile)**:

```
Layer       "tap" {
            layerNumber             = 100
            maskName                = "tap"
            isDefaultLayer          = 1
            visible             = 0
            selectable            = 1
            blink               = 0
            color               = "white"
            lineStyle             = "solid"
            pattern              = "blank"
            pitch               = 0
            defaultWidth            = 0
            minWidth               = 0
            minSpacing              = 0
```

}

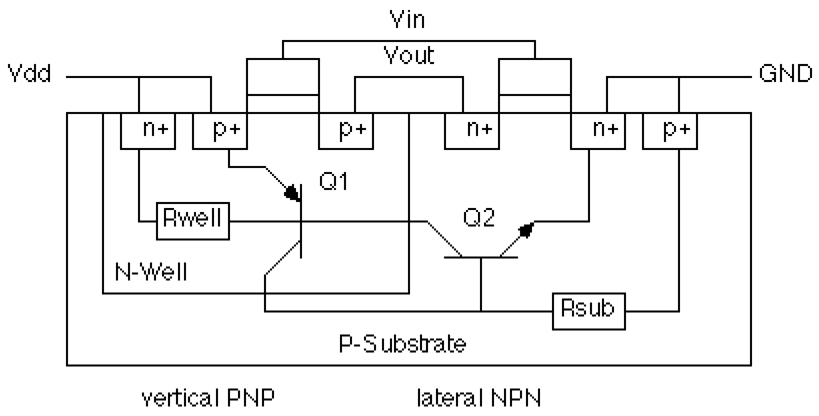
Need For Filler Cell

The purpose of filler cells is to maintain continuity in the rows by adding vdd! and gnd! lines and an n-well. The filler cells also contain substrate connections to improve substrate biasing. Typically, filler cells are used to fill any spaces between regular library cells to avoid planarity problems and provide electrical continuity for power and ground.

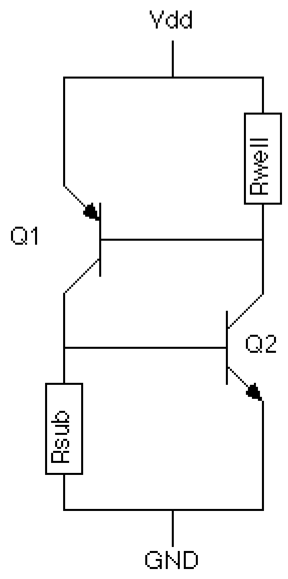
While we are into 90 nanometer technology (or lower), you must use this switch "define poIncludeFiller 1" before extraction to consider filler cells. Then you can proceed to power analysis and rail analysis.

Latchup in Bulk CMOS

A byproduct of the Bulk CMOS structure is a pair of parasitic bipolar transistors. The collector of each BJT is connected to the base of the other transistor in a positive feedback structure. A phenomenon called latch-up can occur when (1) both BJT's conduct, creating a low resistance path between Vdd and GND **and** (2) the product of the gains of the two transistors in the feedback loop, b1 x b2, is greater than one. The result of latch-up is at the minimum a circuit malfunction, and in the worst case, the destruction of the device.



Cross section of parasitic transistors in Bulk CMOS



Equivalent Circuit

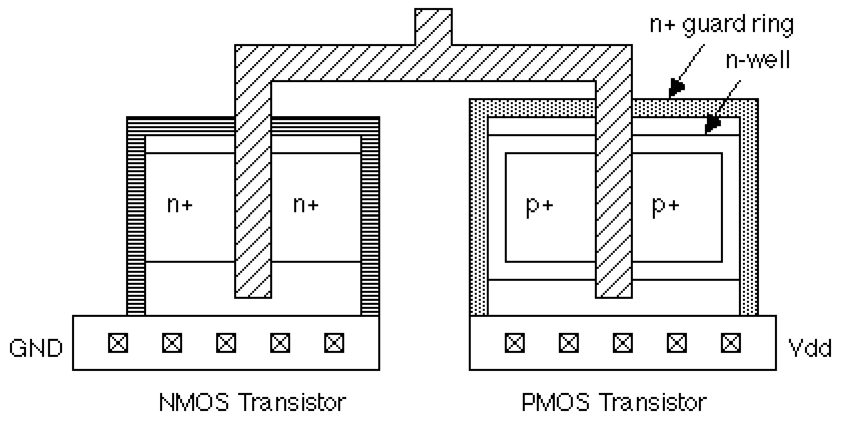
Latchup may begin when Vout drops below GND due to a noise spike or an improper circuit hookup (Vout is the base of the lateral NPN Q2). If sufficient current flows through Rsub to turn on Q2 (I Rsub > 0.7 V), this will draw current through Rwell. If the voltage drop across Rwell is high enough, Q1 will also turn on, and a self-sustaining low resistance path between the power rails is formed. If the gains are such that b1 x b2 > 1, latchup may occur. Once latchup has begun, the only way to stop it is to reduce the current below a critical level, usually by removing power from the circuit.

The most likely place for latch-up to occur is in pad drivers, where large voltage transients and large currents are present.

Preventing latchup

Fab/Design Approaches

1. Reduce the gain product b1 x b1
   - move n-well and n+ source/drain farther apart increases width of the base of Q2 and reduces gain beta2 > also reduces circuit density
   - buried n+ layer in well reduces gain of Q1
2. Reduce the well and substrate resistances, producing lower voltage drops
   - higher substrate doping level reduces Rsub
   - reduce Rwell by making low resistance contact to GND
   - guard rings around p- and/or n-well, with frequent contacts to the rings, reduces the parasitic resistances.



CMOS transistors with guard rings

Systems Approaches

1. Make sure power supplies are off before plugging a board. A "hot plug in" of an unpowered circuit board or module may cause signal pins to see surge voltages greater than 0.7 V higher than Vdd, which rises more slowly to is peak value. When the chip comes up to full power, sections of it could be latched.
2. Carefully protect electrostatic protection devices associated with I/O pads with guard rings. Electrostatic discharge can trigger latchup. ESD enters the circuit through an I/O pad, where it is clamped to one of the rails by the ESD protection circuit. Devices in the protection circuit can inject minority carriers in the substrate or well, potentially triggering latchup.
3. Radiation, including x-rays, cosmic, or alpha rays, can generate electron-hole pairs as they penetrate the chip. These carriers can contribute to well or substrate currents.
4. Sudden transients on the power or ground bus, which may occur if large numbers of transistors switch simultaneously, can drive the circuit into latchup. Whether this is possible should be checked through simulation.

1. **How to reduce the power / ground Bounce?**

The Power/Ground bounce can be avoided by designing the Power Network considering (Simultaneous switching operation) SSO and Simultaneous switching noise (SSN).
Where will you get EM violations? How will you resolve EM violations on the power network, near power pads or hard macros or on the vias?
Description
===========
Sometimes EM violations are flagged on the power network near power pads, hard macros or on the vias. The following article offers a method to analyze these violations and fix them

(1) Violations near the hard macros. These violations could be on the
   (a) VIAS that connect to the power pins of the macros

(b) on a strap near the macro
(c) "inside" the hard macro on a strap or via that is internal to hard macro

(2) Violations near power pads Violation here, could be on the core power ring or on the straps/vias near the power pad

Analyzing the EM violations
==========================

(1) Analyzing EM violations near hard macro
(a) First identify on which layer the EM violation is occurring. It could happen on the VIA layers as well.
(b) Next check the width of the wire on which EM violation is occurring
(c) Next identify all the nearby paths through which current could flow to the point where violation is occurring.
(d) While identifying paths, find the paths that come through the biggest wires as those offer the smallest resistance.
(e) Compare the widths of these biggest wires in the area to the width of the wire on the EM violation is occurring.

If the violation is occurring on a via layer
(a) Identify all the via layer stack.
for example, Metal5 strap connecting to Metal2 power pin/strap would need 2-3, 305 and 4-5 via layers

(b) Using layer visibility control of the layout tool check the via arrays created to see if they are full or partial. Or any missing via layer. partial via array is something that does not have all the possible via cuts in the array. This could happen due to some blockage of hard macro nearby that prevents the tool to put full via array. Sometime the blockage could prevent the tool from putting the via array altogether. Only by turning on/off the via layer visibility, it can be easily seen if any one of the via array is missing in the stack.

(2) Analyzing EM violations near the power pads The analysis of these violations is similar to violations near hard macros. Follow the current from the power pad to the violation location. Checking for any partial or missing vias and noting the width of the wires in the path.

Solution/Workaround
==================

(1) Resolving EM violations near hard macros REMOVE the vias on the biggest wires near the violation location.

The reason is this. Almost always, these EM violations occur due to smaller width wires, drawing too much current from the nearby larger width wires.

The larger width wires can carry a lot of current, the smaller cannot.
Thus causing EM violation on the smaller wire.

By REMOVING the via too much current is not drawn by the smaller wire and the violation is removed.

This could sound counter-intuitive as IR drop now increases due to missing vias. Adding more vias and additional straps near the EM violation area might agree with the logic of reducing resistance and providing more paths.

That is difficult to control. And the maximum current to the small violating wire is always provided by the biggest wire in the area, due to path of least resistance.

Also violations inside the macro on a different layer, say Metal2, is harder to control by adding straps over or near the macro on a different layer.

REMOVING vias is a very effective way of resolving EM violations at the slight cost of reducing IR drop.

Violations on the via layer can be fixed by replacing partial via array by full via arrays or by REMOVING the entire via stack. These partial via arrays create a bottleneck for the current causing the EM violation.

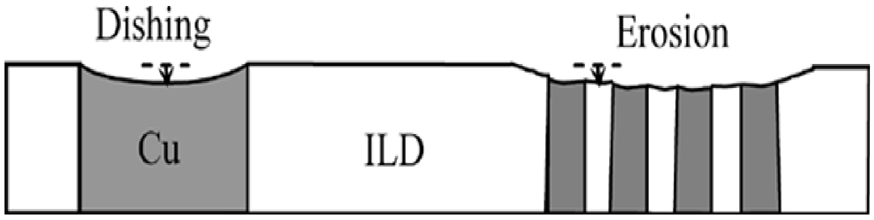
(2) Resolving EM violations near power pads. These are expected as maximum current is present on the wires near the power pads. Core ring wires near power pad, Straps near power pad and the vias in the power pad proximity carry the maximum current in the whole design.

To reduce EM or to remove these violations, make sure no partial via arrays area present, especially the connection from the power pad to core ring.

Give more IO2Core spacing to facilitate full via arrays. Most power pads have power pins on several layers. Provide stacked parallel wires if possible from these power pins to the core ring. This reduces R significantly.

You can add vias on the parallel wires to further reduce Resistance and
Current density. Adding additional horizontal and vertical straps near the power pad could also reduce current density on the straps. Widening these straps also is a good way to reduce R and current density.

Dishing Effect:
For sub-90-nm CMOS technology, chemical-mechanical polishing (CMP) is widely used as the primary technique to planarize the interlayer dielectric (ILD) and metal surface. It has been the enabling technology for the copper damascene process with a high metal removal rate in this kind of trench-first
integration [1]. However, the CMP damascene process also introduces undesirable side effects, including dielectric erosion and metal dishing. The Figure below illustrates oxide erosion and copper dishing their influence on metal line cross sections after CMP



When using the CMP procedure to remove unwanted portions of metal layer to leave thin lines of metals as interconnects, problems such as dishing may occur, which leads to a non-planar surface. The dishing effect is particularly serious when the polishing needs to be carried out until metal is left exclusively in previously etched lines without any metal on the surface of a dielectric layer. It has been found that a significant overpolish is typically needed, which results in erosion of the dielectric layer and dishing of metal below the surface of the dielectric layer. As a result, the thickness of the interconnects in overpolished areas is severely reduced, resulting in an increased sheet resistance when compared to interconnects in other areas of the semiconductor wafer. Additionally,
an uneven topography is in reduced on the surface of the semiconductor

What is cross talk?

- Switching of the signal in one net can interfere neigbouring net due to cross coupling capacitance.This affect is known as cros talk. Cross talk may lead setup or hold voilation.

How can you avoid cross talk?

- -Double spacing=>more spacing=>less capacitance=>less cross talk
- -Multiple vias=>less resistance=>less RC delay
- -Shielding=> constant cross coupling capacitance =>known value of crosstalk
- -Buffer insertion=>boost the victim strength

How shielding avoids crosstalk problem? What exactly happens there?

- -High frequency noise (or glitch)is coupled to VSS (or VDD) since shilded layers are connected to either VDD or VSS.
- Coupling capacitance remains constant with VDD or VSS.

How spacing helps in reducing crosstalk noise?

- width is more=>more spacing between two conductors=>cross coupling capacitance is less=>less cross talk

Why double spacing and multiple vias are used related to clock?

- Why clock?-- because it is the one signal which changes it's state regularly and more compared to any other signal. If any other signal switches fast then also we can use double space.
- Double spacing=>width is more=>capacitance is less=>less cross talk
- Multiple vias=>resistance in parellel=>less resistance=>less RC delay

How buffer can be used in victim to avoid crosstalk?

- Buffer increase victims signal strength; buffers break the net length=>victims are more tolerant to coupled signal from aggressor.

2. **What is Crosstalk Delay and Crosstalk Noise? Describe that?**
3. **What are the SI issues faced in 65nm? How did you solve?**
4. **Pin placement/reordering for handing crosstalk issues.**
5. **What is power integrity and signal integrity?**

14. Electron Migration

What is EM and it effects?

- o Due to high current flow in the metal atoms of the metal can displaced from its origial place. When it happens in larger amount the metal can open or bulging of metal layer can happen. This effect is known as Electro Migration.
- o Affects: Either short or open of the signal line or power line.

Special requirements for EM verification
All the physical verifications and parasitic extractions are based on the design layout. The layout is a two-dimensional view, say X-Y dimensions. The other dimension of the design, say the vertical or Z dimension, is solely determined by the chip manufacturing process. The parameters, such as the metal thickness of a specific metal layer, the via height of a specific via layer, etc., are the same for all the designs using the same manufacturing process. Therefore, the EM specifications for metal layers are typically expressed in current per unit width, not in current per unit area. For a specific manufacturing process, different metal layers may have different values of thickness. More importantly, the interconnect metal pieces of different metal layers are connected through so-called vias. Because metal layers and vias have different EM specifications to comply with, the resistors belonging to different metal or via layers in the extracted parasitic netlist must not be merged.

Power and ground nets may be extracted separately with R-only option
Current flows on power and ground nets are mainly in one direction, so the average current values are typically used for the EM check. Thus the capacitance impact to the average current values is small enough to be ignored. Considering that the number of parasitic capacitors on power and ground nets is close to that of parasitic resistors, it is a very significant simplification to eliminate the capacitance extraction of power and ground nets.

"LAYER" reduction option should be used for all EM verification extractions
StarRCXT provides a parasitic reduction option choice called "LAYER", which means that no merge can be done if two connected resistors belong to different physical layers, and connected resistors can be merged if they belong to the same physical layer. The "LAYER" reduction option should be used for all the EM verification extraction in order to meet the special EM check requirement and to reduce the netlist as much as possible.

An Example StarRCXT command file for EM verification extraction

```
* Database Options
BLOCK: ADC08B3000_ALL_A_Z1
MILKYWAY_DATABASE: ./ADC08B3000_ALL_A_Z1
MILKYWAY_EXTRACT_VIEW: YES
REMOVE_DANGLING_NETS: YES
REMOVE_FLOATING_NETS: YES
TRANSLATE_RETAIN_BULK_LAYERS: YES
*
EXTRACTION: RC
POWER_EXTRACT: YES
POWER_REDUCTION: LAYER
NETLIST_NODE_SECTION: YES
* Processing Options. Use ! to skip the nets
POWER_NETS: pwpr VDD* vddo gnd* pwrn VSS*
NETS: *
NETLIST_SELECT_NETS: *
NETLIST_TYPE: Cg !VDD* !pwpr !vddo !VA *
NETLIST_TYPE: R VDD* pwpr vddo VA
*
* For EM analysis purpose, the following two options should be
set
* in such a way as
NETLIST_TAIL_COMMENTS: YES
REDUCTION: LAYER
*
TCAD_GRD_FILE: CMOS9_5lm_bm_xtor.nxtgrd
MAPPING_FILE: cmos9_5lm_bm_xtor.mapfile
XREF: YES
* Transistor-level extraction.
SKIP_CELLS: !*
* Use STAR format to make the netlist smaller
NETLIST_FILE: adc08b3000_all_a_z1_vdd.spf
NETLIST_FORMAT: star
```

1. What is electro-migration? How is this problem addressed in backend?

Chip Finishing (Closure)

Post-Route: Timing & DRC clean design

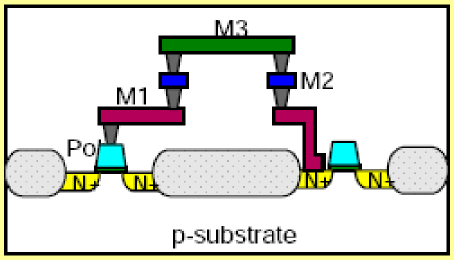
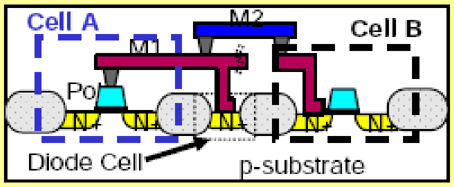
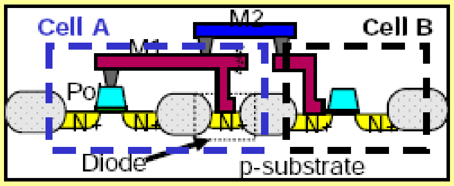
I. Antenna Fixing
II. Wire Spreading
III. Double via Insertion
IV. Filler Cell Insertion
V. Metal Fill Insertion

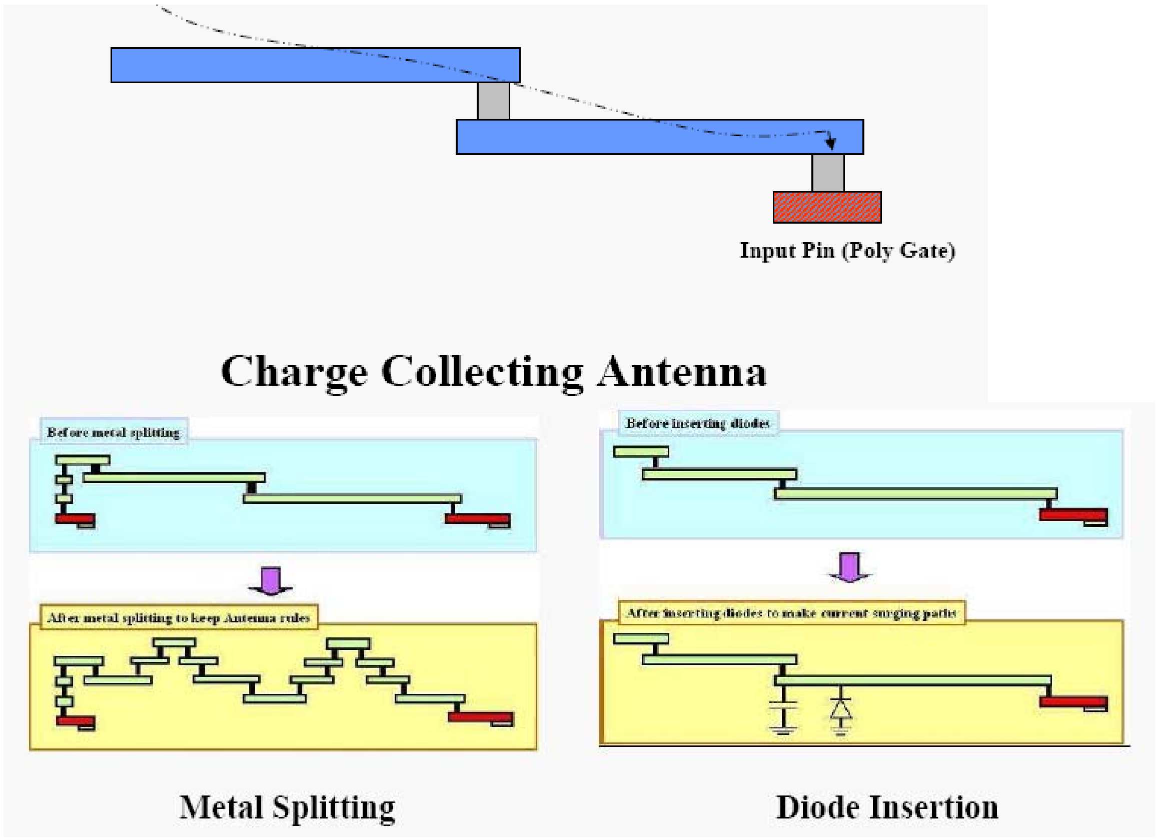
1. What does antenna rules signify related to ASIC backend?How are these violations handled?

In general, fixing antenna problems is quite expensive. Therefore, before fixing antenna violations, the routing should be completed with very few or 0 DRC violations.
Antenna fixing can be done either before or after 'Optimize Routing'. Running 'Optimize Routing' after antenna-fix can generate a good layout first. However, in most cases, running 'Optimize Routing' first can shorten
the overall turn-around time if both steps are required.

2. What is Antenna effect and antenna ratio? How to eliminate this? why it occurs only in Deep sub-micron technology ?
Antenna Ratio:

The antenna effect can occur during the chip manufacturing process and render a die useless. During metallization (when metal wires are laid across devices), some wires connected to the polysilicon gates of transistors may be left floating (unconnected) until the upper metal layers are deposited. A long floating interconnect can act as a temporary capacitor, collecting charges during fabrication steps, such as plasma etching. If the energy built-up on the floating node is suddenly discharged, the logic gate could suffer permanent damage due to transistor gate oxide breakdown. This is termed as Antenna Effect.
Because in nanotechnology the oxide thickness under the transistor gate is very thin. This problem was not there in 0.35u technology. Even if this charge does not get discharge to body. It stays in oxide as hot carrier. there by changing threshold, a big problem.
Antenna Elimination and its effect:

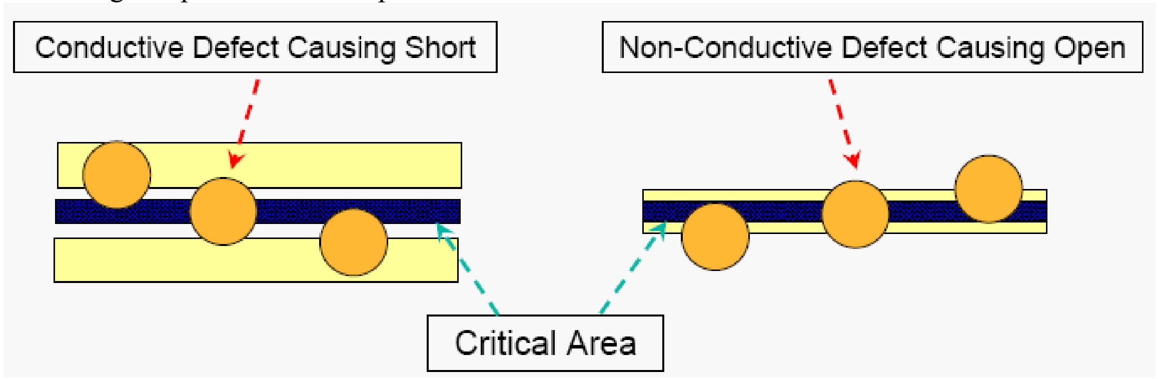
Approach	Rationale	Impact	
Layer Hoping	Top metal completes connection	Longer routes More RCs More congestion	
Net diode insertion	Discharge path through diode	Added capacitance Area penalty LVS (new cell) longer routes(Place and route to diode)	
Diode in cell	Cell protected by construction	Added capacitance, Area penalty	


Input Pin (Poly Gate)
Charge Collecting Antenna
Before metal splitting
After metal splitting to keep Antenna rules
Before inserting diodes
After inserting diodes to make current surging paths
Metal Splitting
Diode Insertion

Antenna Ratio:
Antenna ratio is defined as the ratio between the physical area of the conductors making to the total gate oxide area to which the antenna is electrically connected. A higher ratio implies a greater propensity to fail due to the antenna effect.
Antenna ratio for metal = 500
Antenna ratio for metal = 1100

Why Wire Spreading?
• Random particle defects during manufacturing may cause shorts/opens on routes resulting in loss of yield

• Such regions prune to shorts/opens are referred as 'critical area'


Conductive Defect Causing Short
Non-Conductive Defect Causing Open
Critical Area

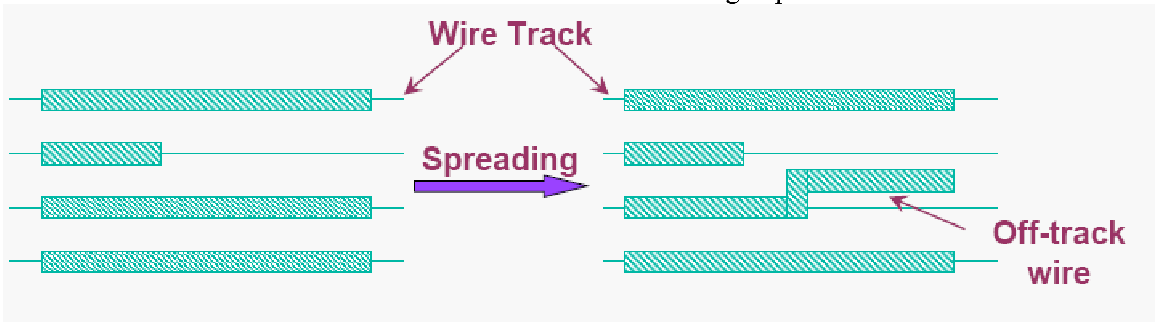
To improve yield against the random particle Defects

Wire spreading resulting in more evenly distributed wires

A probability distribution of various defect sizes to calculate critical area
– The distribution function varies for different fabrication processes

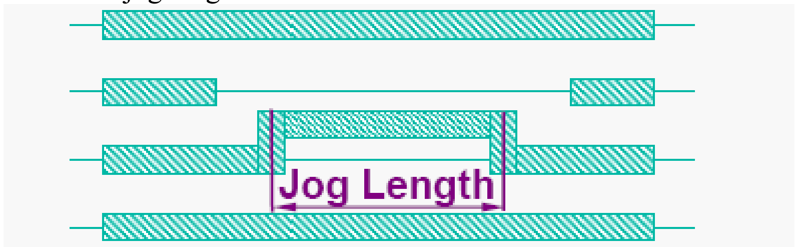
Push routes off-track by ½ pitch

– May result in increase of "open" critical area even though reducing "short" critical area
 – Can choose to widen wires thus not increasing "open" critical area



– Will not push the frozen nets

Push routes off-track by ½ pitch where resulting length of the jog metal can be more than minimum jog length



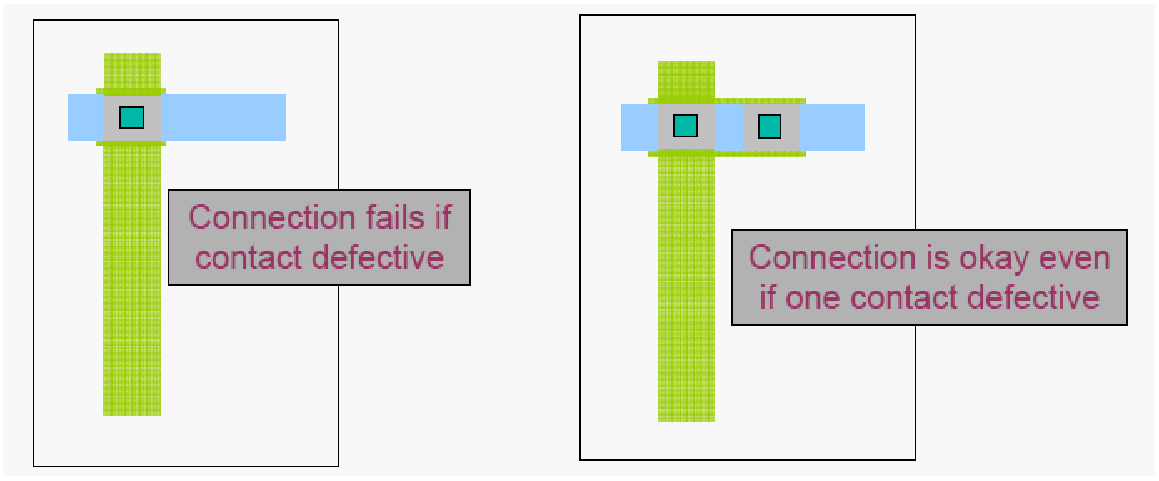
Why NOT run wire spreading before Antenna fixing?
 – Not recommended, Antenna has the most priority after DRC
 – Wire spreading (by pushing off-track) might not leave enough resource to fix antenna

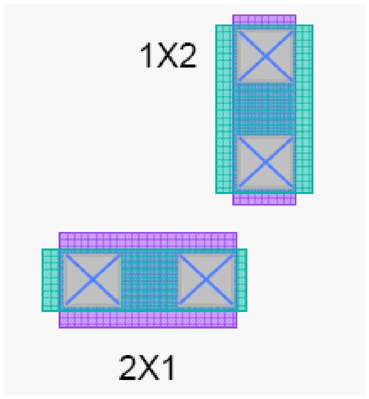
• Will wire spreading switch layers?
– Pushes wires off the track if space available, doesn't switch layers to allow pushing (spreading)
– However, Search & Repair run after wire spreading may result in minimal changes to resolve DRC

• Will wire spreading introduce Antenna violations?
– Antenna ratio dependant on Antenna length may change slightly by wire spreading
 – In most cases should NOT introduce new Antenna violations

What/Why Double Via Insertion?
• Voids in vias is a serious issue in manufacturing
• To handle this issue,

– Reduce via count : Optimize via techniques are employed in ***route_opt***
 – Add backup vias : known as double vias


Connection fails if contact defective
Connection is okay even if one contact defective



Double via count
Good : ~ 95% on all top level layers & ~70% on via1
Rolls back double via to avoid antenna violations
– DRC violations
Double via introduces drc in congested designs

Will double via increase critical area?
– Yes, Any increase in metal (route/cut layer) on layout is bound to introduce critical area. But it will be very little.
• **Will double via introduce new Antenna violations?**
– Vias are doubled with out disturbing the route pattern
– However, Search & Repair run later to resolve DRC might cause antenna violations
– **Turning ON antenna checking will roll back double to single vias reducing antennas**

Why Filler Cell Insertion?
For better yield, density of the chip needs to be uniform
• Some placement sites remains empty on some Rows

– Accepts two lists of filler cells : with/without metal
– Cells without metal are inserted without checking drc's
You need to provide cells without any metal
– Cells with metal are inserted only if no DRC violation
Results
• Filler cell Insertion
– Inserts the cells in the order specified
Larger to smaller is recommended

- By default, honors the hard/soft placement blockages

Why Metal Fill Insertion?
Non uniform metal density causes problems during
manufacturing

- Especially., Chemical mechanical polishing

Extraction considering Metal fill environment
– Extraction does NOT consider metal fill in FILL view
– Extraction does NOT correctly consider fill in CELL view

- Timing analysis does NOT consider fill

15. Chip Closure or DFM or DFY:

Metal Fill:
Once the low-density regions have been identified, they must be repaired. The most common approach is to insert additional metal, not for electrical purposes, but simply to help keep the planarity constant. There are three generally accepted methods of creating this metal fill.
1. By hand through a layout editor; this is time consuming and error prone.

2. To use automatic insertion methods in layout or place-and-route tools. Although it's certainly faster and easier, grid restrictions and other issues often limit the amount of metal fill that can be placed.

3. To do automatic metal fill insertion at the time of physical verification. This simplifies the task because density checking and repair can be done in a single iteration

If the metal fill is left 'floating', it can affect the timing and signal integrity of neighboring signal lines.

Metal Slotting:

Another factor that affects yield is metal slotting. There are many physical reasons that make the inclusion of holes or slots in the metal desirable.

For **aluminium processes**, metal slotting can help reduce **electromigration**. The first, most commonly associated with aluminium processes, involves the creation of **rectangular slots**.

In **copper processes**, metal slotting gives the copper something to '**stick to'**, which helps offset the **sagging** effect. Most commonly associated with copper processes, involves the insertion of simple **square holes**.

To minimise current blockage with rectangular slots, the slots must be aligned in the same direction as the current and staggered for better electron flow. Problems occur at T-junctions or similar structures where working out the current flow is more difficult.

Some layout related tools offer methods to automatically slot wide metal lines upon creation but they can have limitations. Metal slotting, like metal fill insertion, is often easier to implement at the **physical verification stage using a DRC engine**, especially in the **less constrained case** of **square slots**. Employing the same algorithms used in metal fill insertion, a physical verification tool can place automatic slots in wide copper lines.

Antenna Effect:

Define antenna problem and how did you resolve these problem?

In aluminum processing, an ion etch is used to remove excess metal underneath a mask layer, allowing for the accumulation of charge along the sidewalls of the metal. Once excess metal is removed, the mask layer is removed with yet another ion etch. This creates a charge that accumulates along the top of the metal line. Once the metal layer is fully processed and an oxide layer is built on top of it, all the charge that had been associated with the metal will seek a means to dissipate

- Increased net length can accumulate more charges while manufacturing of the device due to ionization process. If this net is connected to gate of the MOSFET it can damage dielectric property of the gate and gate may conduct causing damage to the MOSFET. This is antenna problem.

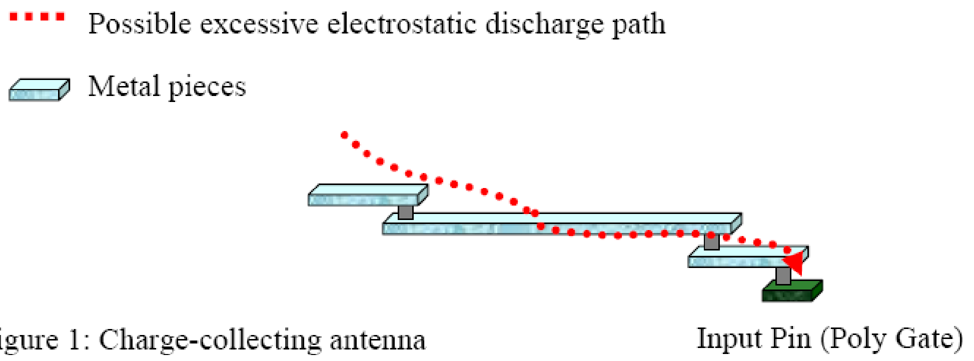


Figure 1: Charge-collecting antenna

This static charge collected on the wires during the multilevel metallization process can destroy the device and make the chip functionless. This is usually referred to as "the charge-collecting antenna problem" or simply "the antenna problem."

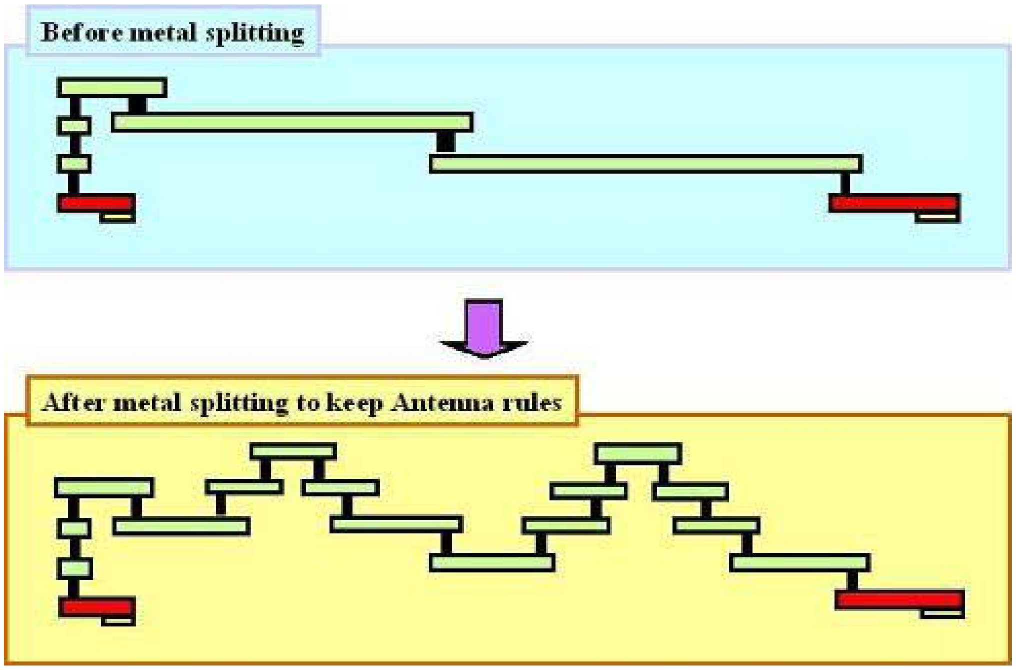
To protect the MOS device from being damaged, the area of the metal, which is seen by the input pin during a single metal mask stage, must be limited to a certain value. When the metal area exceeds this value, a reverse-biased diode can be used to provide a discharging path to protect the input pin.

(antenna-area) / (gate-area) < (max-antenna-ratio)

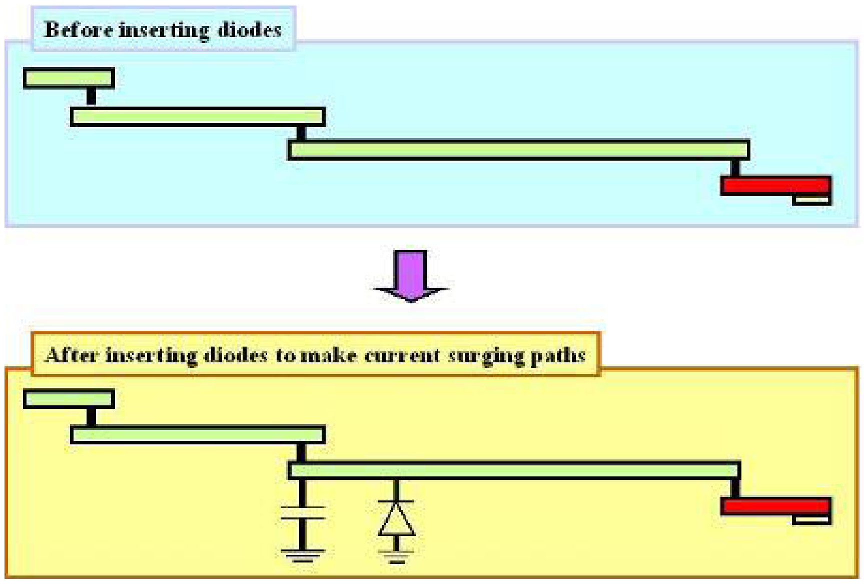
It means that the ratio of metal (antenna-area) to input pin (gate area) must be less than the max-antenna-ratio given by foundry.

Decrease the length of the net by providing more vias and layer jumping (Jogging).

- Metal splitting layer jumping (Jogging). Minimizing the antenna area by breaking the wires with more contacts and wires in other layers



- **Antenna diode Insertion** : Inserting extra diode to provide the discharge path



Synopsys:

One way to protect gates from antenna effects is to provide a discharge path for the accumulated charge to leave the net. However, the discharge path should not allow current to flow during normal chip operation. Discharging can be accomplished by inserting a reverse-biased diode on the net somewhere close to the gate that is being protected. In normal operation, the reverse bias would prevent the current flowing through the diode. When the voltage across the diode exceeds its breakdown voltage (**which is much less than the voltage drop when the gate fails**), the current is discharged through the diode to the substrate. This discharging technique limits the maximum voltage that the gate ever encounters.

Mentor:

At high temperatures, such as those used during IC fabrication, a reverse-biased diode is capable of draining a significant amount of charge, preventing damage to neighbouring transistors. In **normal conditions, these high temperatures are not reached** so diodes will not affect the signals significantly, other than occasional timing effects due to added parasitic capacitances. This does not apply only to specifically designed diodes. During manufacture, any standard NP junction can act as a diode, including source and drain regions.

Why should I avoid slotting? Is wire splitting or multiple rings a good alternative for power rings?

Answer:

Description
===========
Most foundries now require that any wide wire that is greater than a certain width, for example, 10um, has to be slotted.

The slotting rule states that wide wires greater than a certain width must have rectangles of holes in the middle of the wires to avoid dishing effects.

Wide wires are often used for core power rings that go around the core in the device's peripheral IOs.

Slotting has severe negative impact:

(A) Poor Current flow.
The slots (rectangular holes) on the wide wires are usually not aligned with each other. This causes the current to go around the slots in a round about fashion.

(B) Increased resistance due to current crowding
Due to current crowding around the slots, there is an increased resistance causing increased IR drop in the power network.

(C) Low Reliability Due to Electromigration
Due to severe current crowding around the slots, electromigration causes the slots to grow in time until the wide wire completely breaks. This wide wire being the main core power ring will cause the device to fail.

(D) Extraction Issues
Extraction tools cannot properly account for the current crowding effects.

For resistance extraction, extraction tools have to break the wide wire into several polygons to cut around the slots and then apply the unit resistance which reduces accuracy.

Not taking current crowding into effect will severely limit the accuracy and any rail analysis done on such a network is inaccurate.

Capacitance extraction, if done, on these wide wires is also inaccurate
because extraction tools do not have patterns of slotted wires to verify and also cannot take into account the intra-wire capacitance within a slot.

Wire Splitting or Multiple Rings

Wire splitting (also called multiple rings) is a very simple technique to avoid slotting and can be easily implemented in IC Compiler.

Foundries indicate that any wire over 10um has to be slotted and the core power ring has to be at least 100um to carry all the current required by the device.

Instead of one 100um wire, put ten 10um wires spaced apart at minimum distance.

The power wires from the IO POWER PAD will connect to ALL the 10 wires.
The horizontal and vertical power straps (or stripes) that go over the core will also connect to ALL the 10 wires.

The same thing is done for VSS as well.

The only downside to this is the extra space needed to put in ten 10um wires at minimum distance apart instead of one 100um wire (which could be just a few microns).

The current and resistance distribution is uniform and smooth.

Wire/VIA hookups to multiple rings
==================================
Vias will also hook up appropriately to all the 10 wires.

Any wire (strap) connecting to these ten 10um wires will connect to ALL
through VIAS; hence there is no extra load on any single wire.

This technique does not have any of the drawbacks mentioned for the slotted wires.

Wire splitting or multiple rings in IC Compiler can be achieved by simply specifying VDD 10 times in the Nets Tab and specifying the min space, resulting in 10 wires.

You can also interlace, if desired, by specifying "VDD VSS VDD VSS," which will put 4 wires in that order with min spacing between them.

Wire splitting can be done in almost all cases where slotting is required.

16. Low Power Design techniques

1. **What is the significance of clock gating?**
   Clock Gating is a power reduction technique, As clock is a highly toggling signal, It consumes more power, When a potion of design not using the clock, Clock passing to that particular portion is stopped by using clock gating technique.
2. **What are the Power Management Techniques?**

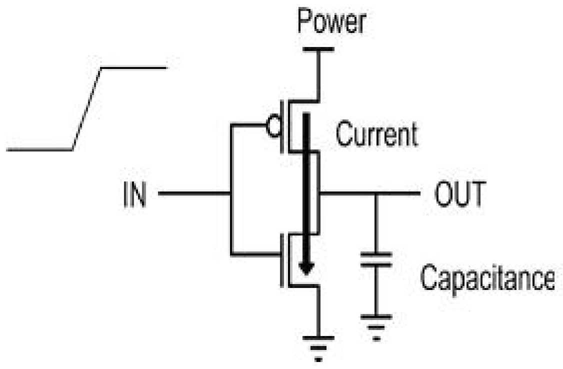
- Multi Voltage Libraries can be used
- Multi threshold Cells can be used
- Optimized Coarse grained switching cells is used to reduce and fast wake-up time.

3. **Why are multi Vt libraries?**
4. **What are the challenges being faced as we move to deep sub micron technology?**

   1 Explain short circuit current.

Short Circuit Power

When dynamic power is analyzed the switching component of power consumption, an instantaneous rise time was assumed, which insures that only one of the transistors is ON. In practice, finite rise and fall times results in a direct current path between the supply and ground, GND, this exists for a short period of time during switching.



Short circuit power [3]

Consider an example of inverter. During switching both NMOS and PMOS transistors in the circuit conduct simultaneously for a short amount of time. Specifically, when the condition, VTn (lesser than) Vin (lesser than) Vdd - |VTp| holds for the input voltage, where VTn and VTp are NMOS and PMOS thresholds, there will be a conductive path open between Vdd and GND because both the NMOS and PMOS devices will be simultaneously on. This forms direct current path between the power supply and the ground. This current has no contribution towards charging of the output capacitance of the logic gate.

When the input rising voltage exceeds the threshold voltage of NMOS transistor, it starts conducting. Similarly until input voltage reaches Vdd-|Vt,p| PMOS transistor remains ON. Thus for some time both transistors are ON. Similar event causes short circuit current to flow when signal is falling. Short circuit current terminates when transition is completed.

Assuming symmetric inverter with Kn=Kp=K and Vt,n=|Vt,p|=Vt and very small capacitive load and both rise and fall times are same we can write,

P_{avg}(short circuit) = 1/12.k.τ.F_{clk}.(Vdd-2Vt)^3 [1]

Thus short circuit power is directly proportional to rise time, fall time and k. Therefore reducing the input transition times will decrease the short circuit current component. But propagation delay requirements have to be considered while doing so.

Short circuit currents are significant when the rise/fall time at the input of a gate is much larger than the output rise/ fall time. This is because the short-circuit path will be active for a longer period of time. To minimize the total average short-circuits current, it is desirable to have equal input and output edge times [2]. In this case, the power consumed by the short-circuit currents is typically less than 10% of the total dynamic power. An important point to note is that if the supply is lowered to be below the sum of the thresholds of the transistors, Vdd (lesser than) VTn + |VTp|, the short-circuit currents can be eliminated because both devices will not be on at the same time for any value of input voltage.

2 What are pros/cons of using low Vt, high Vt cells?

Multi Threshold (MVT) Voltage Technique

Multiple threshold voltage techniques use both Low Vt and High Vt cells. Use lower threshold gates on critical path while higher threshold gates off the critical path. This methodology improves performance without an increase in power. Flip side of this technique is that Multi Vt cells increase fabrication complexity. It also lengthens the design time. Improper optimization of the design may utilize more Low Vt cells and hence could end up with increased power!

Ruchir Puri et al. [2] have discussed the design issues related with multiple supply voltages and multiple threshold voltages in the optimization of dynamic and static power. They noted several advantages of Multi Vt optimization. Multi Vt optimization is placement non disturbing transformation. Footprint and area of low Vt and high Vt cells are same as that of nominal Vt cells. This enables time critical paths to be swapped by low Vt cells easily.

Frank Sill et al. [3] have proposed a new method for assignment of devices with different Vth in a double Vth process. They developed mixed Vth gates. They showed leakage reduction of 25%. They created a library of LVT, mixed Vt, HVT and Multi Vt. They compared simulation results with a LVT version of each design. Leakage power dissipation decreased by average 65% with mixed Vth technique compared to the LVT implementation.

Meeta Srivatsav et al. [4] have explored various ways of reducing leakage power and recommended Multi Vt approach. They have carried out analysis using 130 nm and 90 nm technology. They synthesized design with different combination of target library. The combinations were Low Vt cells only, High Vt cells only, High Vt cells with incremental compile using Low Vt library, nominal (or regular) Vt cell and Multi Vt targeting Hvt and Lvt in one go. With only Low Vt highest leakage power of 469 µw was obtained. With only High Vt cells leakage power consumption was minimum but timing was not met (-1.13 of slack). With nominal Vt moderate leakage power value of 263 µw was obtained. Best results (54 µw with timing met) obtained for synthesis targeting Hvt library and incremental compile using Lvt library.

Different low leakage synthesis flows are carried out by Xiaodong Zhang [1] using Synopsys EDA tools are listed below:

- **Low-Vt --> Multi-Vt flow**: This produces least cell count and least dynamic power. But produce highest leakage power. It takes very low runtime. Good for a design with very tight timing constraints

- **Multi-Vt one pass flow**: It takes longest runtime and can be used in most of designs.

- **High-Vt --> Multi-Vt flow**: Produce least leakage power consumption but has high cell count and dynamic power. This methodology is good for leakage power critical design.

- **High-Vt --> Multi-Vt with different timing constraints flow**: This is a well balanced flow and produces second least leakage power. This has smaller cell count, area and dynamic power and shorter runtime. This design is also good for most of designs.

Optimization Strategies

The tradeoffs between the different Vt cells to achieve optimal performance are especially beneficial during synthesis technology gate mapping and placement optimization. The logic synthesis, or gate mapping phase of the optimization process is implemented by synthesis tool, and placement optimization is handled physical implementation tool.

Synthesis

During logic synthesis, the design is mapped to technology gates. At this point in the process optimal logic architectures are selected, mapped to technology cells, and optimized for specific design goals. Since a range of Vt libraries are now available and choices have to be made across architectures with different Vt cells, logic synthesis is the ideal place to start deploying a mix of different Vt cells into the design.

Single-Pass vs. Two-Pass Synthesis –with multiple threshold libraries

Multiple libraries are currently available with different performance, area and power utilization characteristics, and synthesis optimization can be achieved using either one or more libraries concurrently. In a single-pass flow, multiple libraries can be loaded into synthesis tool prior to synthesis optimization. In a two-pass flow, the design is initially optimized using one library, and then an incremental optimization is carried out using additional libraries.

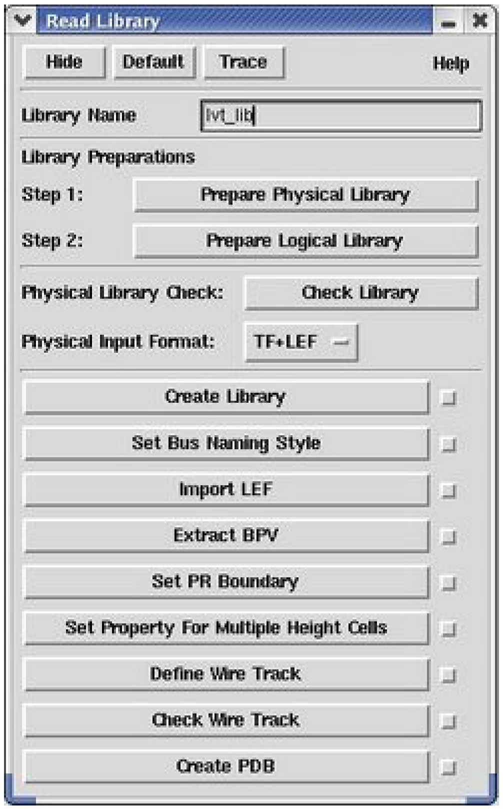
About multi vt optimization in his paper Ruchir Puri[2] says: "The multi-threshold optimization algorithm implemented in physical synthesis is capable of optimizing several Vt levels at the same time. Initially, the design is optimized using the higher threshold voltage library only. Then, the Multi-Vt optimization computes the power-performance tradeoff curve up to the maximum allowable leakage power limit for the next lower threshold voltage library. Subsequently, the optimization starts from the most critical slack end of this power-performance curve and switches the most critical gate to next equivalent lower-Vt version. This will increase the leakage in the design beyond the maximum permissible leakage power. To compensate for this, the algorithm picks the least critical gate from the other end of the power-performance curve and substitutes it back with its higher-Vt version. If this does not bring the leakage power below the allowed limit, it traverses further from the curve (from least critical towards more critical) substituting gates with higher-Vt gates, until the leakage limit is satisfied. Then we jump back to the second most critical cell and switch it to the lower-Vt version. This iteration continues until we can no longer switch any gate with the lower vt version without violating the leakage power limit."

But Amit Agarwal et al. [5] have warned about the yield loss possibilities due to dual Vt flows. They showed that in nano-scale regime, conventional dual Vt design suffers from yield loss due to process variation and vastly overestimates leakage savings since it does not consider junction BTBT (Band To Band Tunneling) leakage into account. Their analysis showed the importance of considering device based analysis while designing low power schemes like dual Vt. Their research also showed that in scaled technology, statistical information of both leakage and delay helps in minimizing total leakage while ensuring yield with respect to target delay in dual Vt designs. However, nonscalability of the present way of realizing high Vt, requires the use of different process options such as metal gate work function engineering in future technologies.

Issues with Multi Height Cell Placement in Multi Vt Flow

Creating the reference libraries

There are two reference libraries required. One is low Vt cell library and another is high Vt cell library. These libraries have two different height cells. Reference libraries are created as per the standard synopsys flow. Library creation flow is given in Figure 1. Read_lib command is used for this purpose. As TF and LEF files are available TF+LEF option is chosen for library creation. After the completion of the physical library preparation steps, logical libraries are prepared.



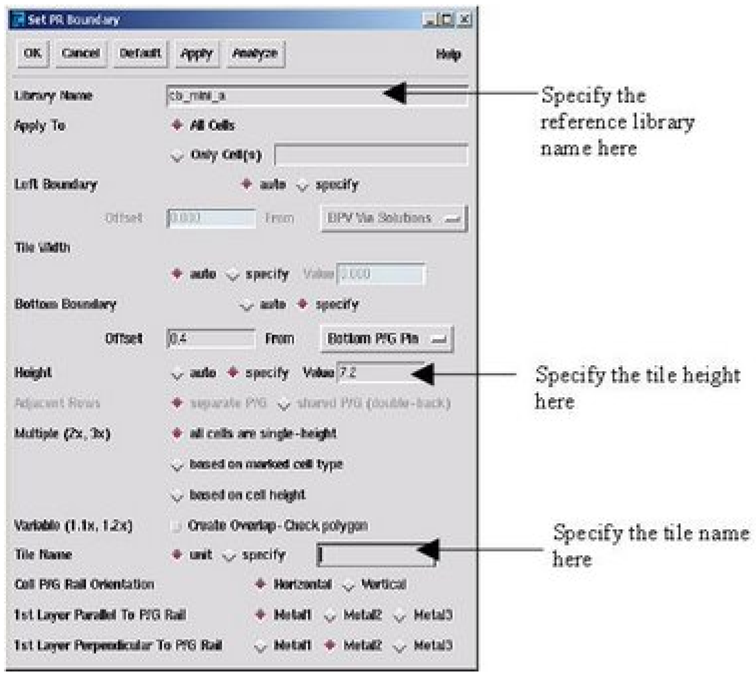
Library Name: lvt_lib

Physical Input Format: TF+LEF

Figure 1 Library preparation command window

Different Unit Tile Creation

The unit tile height of lvt cells is 2.52 µ and hvt cells are 1.96 µ. Hence two separate unit tiles have to be created and should be added in the technology file. Hvt reference library is created with the unit tile name “unit” and lvt reference library is created with unit tile name “lvt_unit”. By default “unit” tile is defined in technology file and the other unit tile “lvt_unit” is also added to the technology file.


Set PR Boundary
OK
Cancel
Default
Apply
Analyze
Help
Specify the reference library name here
Specify the tile height here
Specify the tile name here

Library Name: cb_mini_a
Apply To: [✓ All Cells] Only Cell(s)
Left Boundary: [✓ auto] specify
Offset 0.000 From DPV Via Solutions
Tile Width: [✓ auto] specify Value 0.000
Bottom Boundary: auto [✓ specify]
Offset 0.4 From Bottom P/G Pin
Height: auto [✓ specify] Value 7.2
Adjacent Rows: [✓ separate P/G] shared P/G (double-back)
Multiple (2x, 3x): [✓ all cells are single-height] based on marked cell type / based on cell height
Variable (1.1x, 1.2x): Create Overlap-Check polygon
Tile Name: [✓ unit] specify
Cell P/G Rail Orientation: [✓ Horizontal] Vertical
1st Layer Parallel To P/G Rail: [✓ Metal1] Metal2 Metal3
1st Layer Perpendicular To P/G Rail: Metal1 [✓ Metal2] Metal3

Figure 2. Tile height specifications in library preparation

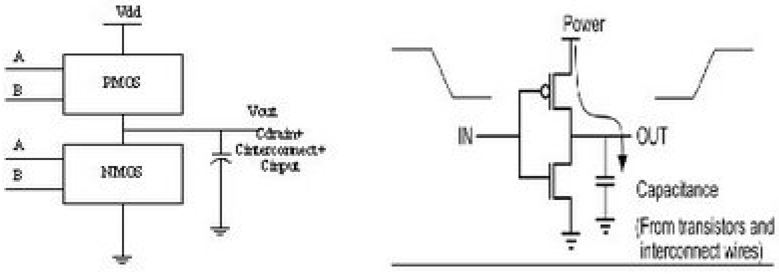
2 What are power dissipation components? How do you reduce them?

Dynamic Power

As the name indicates it occurs when signals which go through the CMOS circuits change their logic state. At this moment energy is drawn from the power supply to charge up the output node capacitance. Charging up of the output capacitance causes transition from 0V to Vdd. Considering an inverter example power drawn from the power supply is dissipated as heat in pMOS transistor. On the other hand charge down process causes NMOS transistor to dissipate heat.

Output capacitance of the CMOS logic gate consists of below components:

1) **Output node capacitance of the logic gate:** This is due to the drain diffusion region.
2) **Total interconnects capacitance:** This has higher effect as technology node shrinks.
3) **Input node capacitance of the driven gate:** This is due to the gate oxide capacitance.



Dynamic power dissipation in CMOS inverter [1]

The average power dissipation of the CMOS logic circuit can be mathematically expressed [2]. Integrating the instantaneous power over the period of interest, the energy E_{VDD} taken from the supply during the transition is given by

$E_{VDD}= 0\text{->}\infty\int I.\ V_{DD}(t).V_{DD}.dt$

$=V_{DD}.\ 0\text{->}\infty\int C_L.(dvout/dt).dt$

$= C_L.V_{DD}.\ 0\text{->}V_{DD}\int.dvout$

$= C_L.V_{DD}^2$

Similarly integrating the instantaneous power over the period of interest, the energy Ec stored in the capacitor at the end of transition is given by,

$Ec = 0\text{->}\infty\int I.\ V_{DD}(t).Vout.dt$

$= 0\text{->}\infty\int C_L.(dvout/dt).vout.dt$

$= C_L.$(integration from 0 to VDD)$.Vout.dvout$

$= (C_L.V_{DD}^2)/2$

Therefore energy stored in capacitor is= $C_L.V_{DD}^2 / 2$.

This implies that half of the energy supplied by the power source is stored in C_L. The other half has been dissipated by the PMOS devices. This energy dissipation is independent of the size of the PMOS device. During the discharge phase the charge is removed from the capacitor, and its energy is dissipated in the NMOS device.

Each switching cycle takes a fixed amount of energy = $C_L.\ V_{DD}^2$.

If a gate is switched on and off 'fn' times / second, then Pdynamic = $C_L.\ V_{DD}^2.\ fn$.

Where fn □ frequency of energy consuming transitions. This is also called "switching activity".

In general we can write,

Pdynamic = $Ceff.V_{DD}^2.f$

Where f □ maximum switching activity possible i.e. clock rate.

Hence,

$P_{avg}= 1/T\ [0\text{->}T/2\int Vout\ (-C_{load}.dVout/dt)dt+T/2\text{->}T\int(V_{DD}-Vout)(C_{load}.dVout/dt)\ dt]$

i.e. $P_{avg}=1/T\ C_{load}.V_{DD}^2$
i.e. $P_{avg}=C_{load}.V_{DD}^2.F_{clk}$

Here energy required to charge up the output node to Vdd and charge down the total output load capacitance to ground level is integrated. Applied input periodic waveform having its period T is assumed to be having zero rise and fall time. Note that average power is independent of transistor size and characteristics.

Internal power

This is the power consumed by the cell when an input changes, but output does not change [3]. In logic gates not every change of the current running through an input cell necessarily leads to a change in the state of the output net. Also internal node voltage swing can be only Vi which can be smaller than the full voltage swing of Vdd leading to the partial voltage swing.

Below mentioned steps can be taken to reduce dynamic power

1) Reduce power supply voltage Vdd
2) Reduce voltage swing in all nodes
3) Reduce the switching probability (transition factor)
4) Reduce load capacitance

Leakage Power Trends

Development of the digital integrated circuits is challenged by higher power consumption. The combination of higher clock speeds, greater functional integration, and smaller process geometries has contributed to significant growth in power density. At 90 nm and below, leakage power management is essential in the ASIC design process. As voltages scale downward with the geometries threshold voltages must also decrease to gain the performance advantages of the new technology but leakage current increases exponentially. Thinner gate oxides have led to an increase in gate leakage current.

Scaling improves transistor density and functionality on a chip. Scaling helps to increase speed and frequency of operation and hence higher performance. At the same time power dissipation increases. To counteract increase in active and leakage power Vth should also be scaled. Leakage power is catching up with the dynamic power in VDSM CMOS circuits as shown in Figure 1.

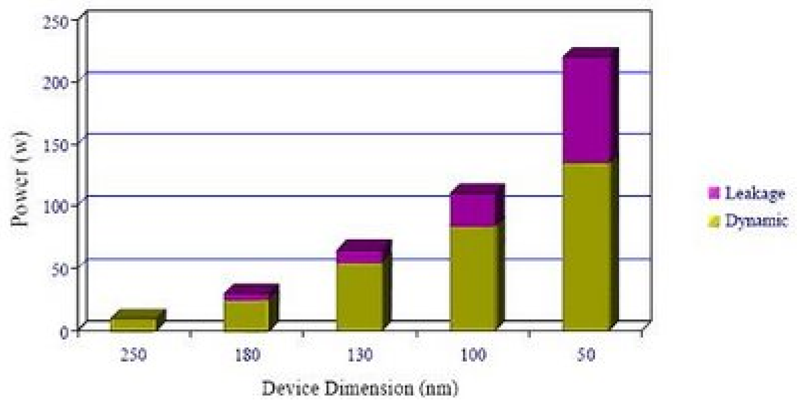


Figure 1. Leakage vs.Dynamic power [3]

According to Sung Mo Kang et al.[1] and Anantha P. Chandrakasan et al.[2] power consumption in a circuit can be divided into 3 different components. They are:

1) dynamic

2) static (or leakage) and

3) Short circuit power consumption.

Dynamic (or switching) power consumption occurs when signals which go through the CMOS circuits change their logic state charging and discharging of output node capacitor.

Leakage power consumption is the power consumed by the sub threshold currents and by reverse biased diodes in a CMOS transistor.

Short circuit power consumption occurs during switching of both NMOS and PMOS transistors in the circuit and they conduct simultaneously for a short amount of time.

Leakage Power

The power consumed by the sub threshold currents and by reverse biased diodes in a CMOS transistor is considered as leakage power. The leakage power of a CMOS logic gate does not depend on input transition or load capacitance and hence it remains constant for a logic cell.

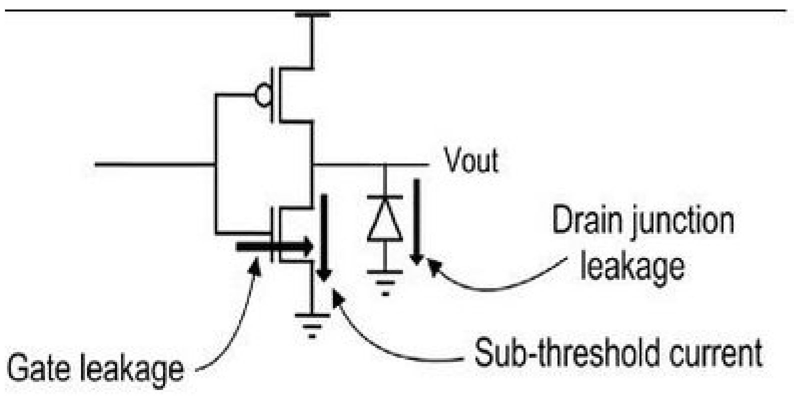


Figure 2. Leakage power components in an inverter [5]

Leakage Components in Bulk CMOS

Different leakage power components are classified are as follows and are shown in Figure 3.

- Diode reverse bias current or Reverse-biased, drain- and source-substrate junction band-to-band-tunneling (BTBT) –I1
- Sub threshold current – I2

- Gate induced drain leakage – I3

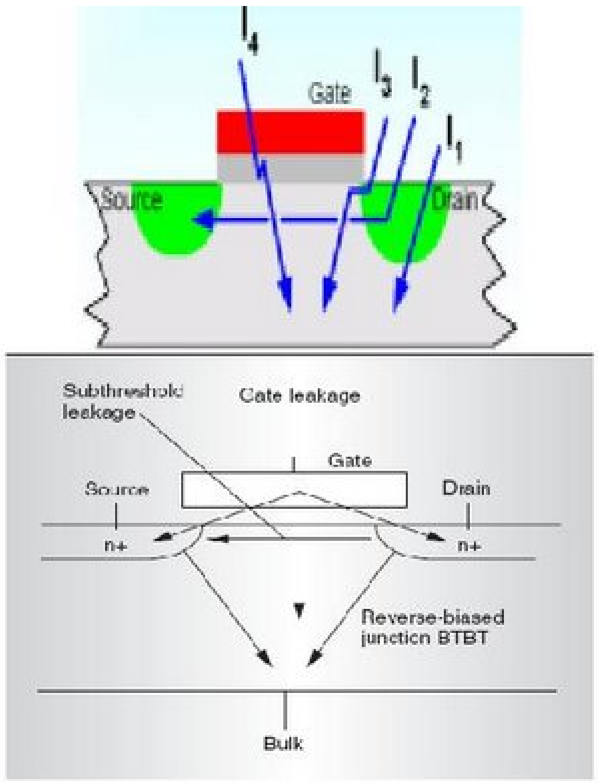


Figure 3. Major leakage components in a transistor [2] [3]

As technology node shrinks towards 45 nm and below gate leakage (i.e. leakage current due to direct tunneling) increases owing to the increased electric field. This is the reason why voltage is scaled down to around 1V. Improvements in the manufacturing process and material have helped to control other leakage components such as sub threshold leakage, GIDL and junction reverse bias leakage. A comparative graphical representation of different leakage currents in different technology nodes is shown in Figure 4.

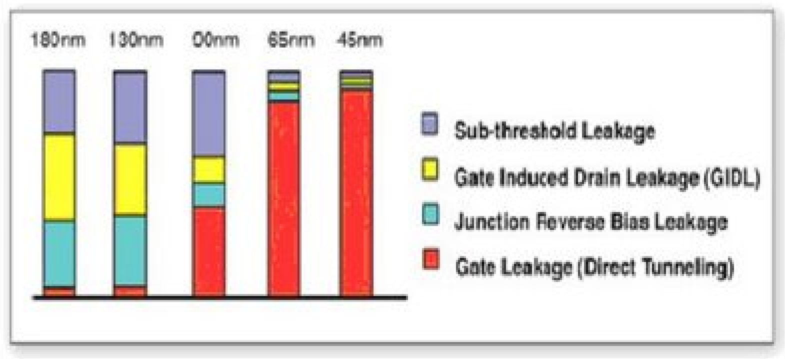
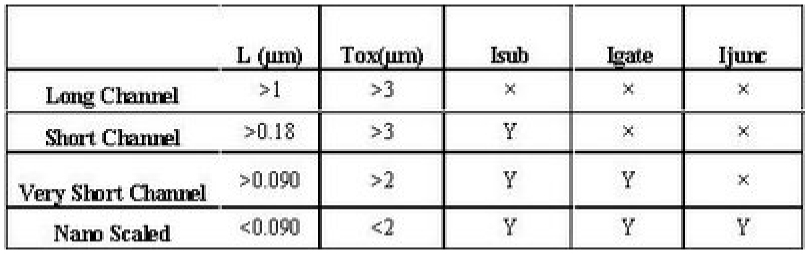


Figure 4. Technology shrinking vs. Leakage components

Sub threshold leakage is controlled by having more control over threshold voltage. Olden process technologies are causing up to 50 % of threshold voltage variation but newer technologies produce very low threshold voltage deviation, 30 mV being maximum value. Decrease in junction area and voltage automatically decreases junction reverse bias leakage and GIDL respectively. But the tunneling effect is threatening further decrease in device dimension. Reducing the GIDL, reverse bias leakage and gate leakage due to tunneling is directly related to the improvements in fabrication chemistry of the device whereas designer has a little control over threshold voltage. The other way by which a designer can have control over these leakage components is to switch off the device itself in controlled fashion! Low power techniques like “power gating” does this effectively and “back bias” technique controls threshold voltage.

Xiaodong Zhang [3] has studied impact of dynamic and leakage power as technology node reaches deep submicron level. Their summary of the result and leakage trends studied by Massoud Pedram [4] is shown below in Table 1.



	L (µm)	Tox(µm)	Isub	Igate	Ijunc
Long Channel	>1	>3	×	×	×
Short Channel	>0.18	>3	Y	×	×
Very Short Channel	>0.090	>2	Y	Y	×
Nano Scaled	<0.090	<2	Y	Y	Y

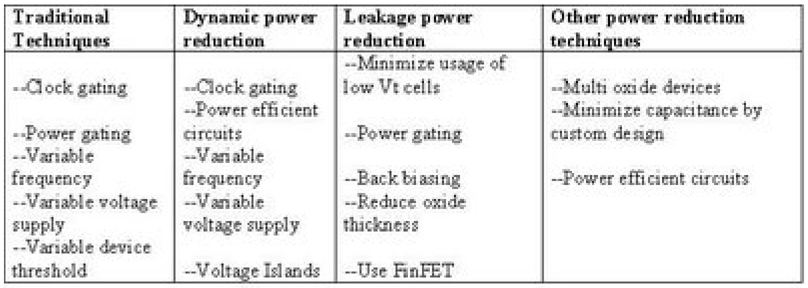
Table 1. Leakage power trends

Wide variety of techniques have been developed to address the various aspects of the power problem and to meet power specifications. These techniques include clock gating, multi-threshold (multi-Vt) voltage cells, multiple-voltage domains, substrate biasing, dynamic voltage and frequency scaling (DVFS), power gating. [1]

Low Power Design Techniques

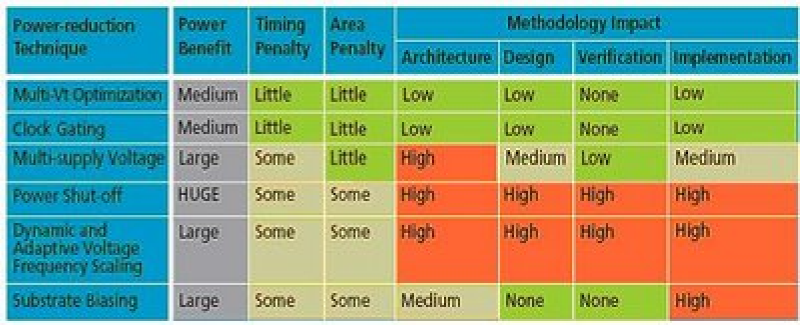
Michael Keating et al. [1] lists several low power techniques to tackle the dynamic and static power consumption in modern SoC designs. Dynamic power control techniques include clock gating, multi voltage, variable frequency, and efficient circuits. Leakage power control techniques include power gating, multi Vt cells. Common methods supported by EDA tools include clock gating, gate sizing, low power placement, register clustering, low power CTS, multi Vt optimization.

Some of the low power techniques in use today are listed in below table.



Traditional Techniques	Dynamic power reduction	Leakage power reduction	Other power reduction techniques
--Clock gating --Power gating --Variable frequency --Variable voltage supply --Variable device threshold	--Clock gating --Power efficient circuits --Variable frequency --Variable voltage supply --Voltage Islands	--Minimize usage of low Vt cells --Power gating --Back biasing --Reduce oxide thickness --Use FinFET	--Multi oxide devices --Minimize capacitance by custom design --Power efficient circuits

Different Low Power Techniques [3]



Power-reduction Technique	Power Benefit	Timing Penalty	Area Penalty	Methodology Impact			
				Architecture	Design	Verification	Implementation
Multi-Vt Optimization	Medium	Little	Little	Low	Low	None	Low
Clock Gating	Medium	Little	Little	Low	Low	None	Low
Multi-supply Voltage	Large	Some	Little	High	Medium	Low	Medium
Power Shut-off	HUGE	Some	Some	High	High	High	High
Dynamic and Adaptive Voltage Frequency Scaling	Large	Some	Some	High	High	High	High
Substrate Biasing	Large	Some	Some	Medium	None	None	High

Trade-offs associated with the various power management techniques [2]

Above table summarizes trade-offs associated with different power management techniques. Power gating and DVFS demand large methodology change whereas multi vt and clock gating affect least. Unless large leakage optimization is not necessary it is always beneficial to go with either multi vt or clock gating techniques. Based on the design complexity and requirements combination of any low power techniques can be adopted. Multi vt optimization along with the power gating is found to be efficient in some of the complex designs. Advanced improvements in the implementation (i.e. fabrication) technology has allowed substrate biasing techniques to be used heavily as it does not pose any architectural and design verification challenges and also provides high leakage reduction

Do you know about input vector controlled method of leakage reduction?

- Leakage current of a gate is dependant on its inputs also. Hence find the set of inputs which gives least leakage. By applyig this minimum leakage vector to a circuit it is possible to decrease the leakage current of the circuit when it is in the standby mode. This method is known as input vector controlled method of leakage reduction.

How can you reduce dynamic power?

- -Reduce switching activity by designing good RTL
- -Clock gating
- -Architectural improvements
- -Reduce supply voltage
- -Use multiple voltage domains-Multi vdd

What are the vectors of dynamic power?

- Voltage and Current

17. Generic Questions

1. Why is pre-simulation and post-Synthesis and simulation done?
Pre-synthesis simulation is called as Functional simulation, This is done to check the Functionality of design.

2. What Is Partitioning?

Partitioning is the process of splitting a design into manageable pieces. The purpose of partitioning is to divide complex design components into manageable pieces for ease of implementation. During this step, models for timing and physical implementation are defined. The floorplan defined during prototyping is pushed down into lower level blocks, thus preserving placement, power routing, and obstructions related to placement and routing. Feed-through might also be assigned for nets routed over-the-block and buffered by inserting hole-punch buffers or by modifying the block netlist Logical partitioning is not required for flat physical implementations.

Partitioning splits design for logical and physical implementation. For hierarchical physical implementations, logical partitioning directly impacts the physical implementation phase.

Partitioning is a method to manage functional complexity from a logical design perspective

- ♦ Partitioning allows multiple design teams to proceed in parallel
- ♦ The bridges between flat and hierarchical physical implementations are:
- Creation of timing budgets
- Pin optimization
- Feed-through or hole-punch buffer assignment
- Floorplan push-down (Obstructions, Power routes)
- Advanced netlist optimizations: timing, clock, power, and signal integrity

3. Compare the hierarchical and flattened design approaches related to ASIC design?

Flat Design

Advantages

- A flat design methodology ensures that there are no problems with boundary constraints between the different levels of hierarchy.
- You have the ability to analyze I/O and hard macro to block paths. You have more accurate timing analysis, because no block modeling is needed.

Disadvantages:

- Large data sizes
- Potentially long run times

Hierarchical Design

Advantages:

- You can save time by closing timing at the top level and at the block level in parallel.
- Generate early top-level timing analysis.
- Smaller data sets lead to faster run times.
- You can reuse blocks once they are implemented.
- If the design uses an IP block, it is easier to insert it into a hierarchical modular design than to try and fit it into a flat design.

Disadvantages:

- Preliminary block characterization is inaccurate. and can yield false top and block-level timing violations as well as mask timing violations that appear to meet timing.
- You need to update block timing models frequently when the blocks change.

- Details are hidden or lost due to modeling at boundaries.

What parameters (or aspects) differentiate Chip Design and Block level design?

- Chip design has I/O pads; block design has pins.
- Chip design uses all metal layes available; block design may not use all metal layers.
- Chip is generally rectangular in shape; blocks can be rectangular, rectilinear.
- Chip design requires several packaging; block design ends in a macro.

4. What does the word "0.18 micron technology" signify? Does scaling theory really work as we move into DSM technology?
5. Write the optimized layout of a 3 input NAND gate?
6. What does the technology rules file contain?
7. What is GDS2 file? What is the layer mapping file?
8. What are the commonly pre-routed blocks?
9. What is the difference between a LEF and a GDSII file related to a backend library?
LEF is a Abstract view of std Cells and Macros, But GDSii is a complete layout of blocks.

10. Which is preferred gate NAND or NOR?
11. What is meta file and macro file?

Input Files

Library Exchange Format Files (LEF):

A Library Exchange Format (LEF) file contains library information for a class of designs. Library data includes layer, via, placement site type, and macro cell definitions. The LEF file is an ASCII representation using the syntax conventions

You can define all of your library information in a single LEF file; or you can divide the information into two files, a "technology" LEF file and a "cell library" LEF file. It contain the following information's,

1. Version
2. Bus Bit Characters
3. Divider Character
4. Units
5. Manufacturing Grid
6. Use Min Spacing
7. Property Definitions
8. Clearance Measure
9. Extensions
10. Layer (Cut)
11. Layer (Implant)
12. Layer (Masterslice or Overlap)
13. Layer (Routing)

14. Macro
    Layer Geometries (Size)
    Macro Obstruction (Blockages)
    Macro Pin Statement
15. Maximum Via Stack
16. Names Case Sensitive
17. No Wire Extension
18. Same-Net Spacing
19. Via
20. Via Rule
21. Via Rule Generate
22. Nondefault Rule
23. Site

Managing LEF Files

A technology LEF file contains all of the LEF technology information for a design, such as placement and routing design rules, and process information for layers. A technology LEF file can include all of the following LEF statements:

```
VERSION statement
 NAMESCASESENSITIVE statement
 BUSBITCHARS statement
 DIVIDERCHAR statement
 [ UNITS statement ]
 [ MANUFACTURINGGRID statement ]
 [ NOWIREEXTENSIONATPIN statement ]
 [ USEMINSPACING statement ]
 [ CLEARANCEMEASURE statement ]
 [ PROPERTYDEFINITIONS statement ]
 { LAYER (Nonrouting) statement
   | LAYER (Routing) statement } ...
 [ MAXVIASTACK statement ]
 { VIA statement } ...
 { VIARULE statement
   | VIARULE GENERATE statement } ...
 [ NONDEFAULTRULE statement ] ...
 [ SPACING statement ]
 { SITE statement} ...
 [ BEGINEXT statement ] ...
```

A cell library LEF file contains the macro and standard cell information for a design. A library LEF file can include the following statements:

```
VERSION statement
 NAMESCASESENSITIVE statement
 BUSBITCHARS statement
 DIVIDERCHAR statement
 [ SITE statement ]
 { MACRO statement
    [ PIN statement ] ...
    [ OBS statement ...] } ...
```

```
      [ BEGINEXT statement ] ...
```

When reading in the LEF files, always read in the technology LEF file first.

What is Site in LEF?

Site

```
SITE siteName
      CLASS {PAD | CORE} ;
      [SYMMETRY {X | Y | R90} ... ;]
      SIZE width BY height ;
END siteName
```

Defines a placement site in the design. A placement site gives the placement grid for a family of macros, such as I/O, core, block, analog, digital, short, tall, and so forth.

`CLASS {PAD | CORE}`
  Specifies whether the site is an I/O pad site or a core site.

`SITE siteName`
  Specifies the name for the placement site.

`SIZE width BY height`
  Specify the dimensions of the site in normal (or north) orientation, in microns.

`SYMMETRY {X | Y | R90}`
  Indicates which site orientations are equivalent. The sites in a given row all have the same orientation as the row. Generally, site symmetry should be used to control the flipping allowed inside the rows.

What is nondefault rule in LEF?

Non-default Rule

Defines the wiring width, design rule spacing, and via size for regular (signal) nets. You do not need to define cut layers for the non-default rule.

```
[NONDEFAULTRULE rulename1
      {LAYER layerName
         WIDTH width ;
         SPACING minSpacing ;
         [WIREEXTENSION value ;]
         [RESISTANCE RPERSQ value ;]
         [CAPACITANCE CPERSQDIST value ;]
         [EDGECAPACITANCE value ;]
      END layerName} ...
      {VIA viaStatement} ...
      [SPACING
         [SAMENET layerName layerName minSpace [STACK] ;] ...
      END SPACING]
      [PROPERTY propName propValue ;] ...
END]
```

Design Exchange Format Files (DEF)

A Design Exchange Format (DEF) file contains the design-specific information of a circuit and is a representation of the design at any point during the layout process. The DEF file is an ASCII representation using the syntax conventions

DEF conveys logical design data to, and physical design data from, place-and-route tools. Logical design data can include internal connectivity (represented by a netlist), grouping information, and physical constraints. Physical data includes placement locations and orientations, routing geometry data, and logical design changes for backannotation. Place-and-route tools also can read physical design data, for example, to perform ECO changes.

For standard-cell-based/ASIC flow tools, floorplanning is part of the design flow. You typically use the various floorplanning commands to interactively create a floorplan. This data then becomes part of the physical data output for the design using the `ROWS`, `TRACKS`, `GCELLGRID`, and `DIEAREA` statements. You also can manually enter this data into DEF to create the floorplan.

General Rules

Note the following information about creating DEF files:

- Lines in the DEF file are limited to 2,048 characters (extra characters are truncated on input).
- Net names and cell names also are limited to 2,048 characters.
- DEF statements end with a semicolon (`;`). You *must* leave a space before the semicolon.
- Each section can be specified only once. Sections end with `END` *SECTION*.
- You must define all objects before you reference them except for the `+ ORIGINAL` argument in the `NETS` section.
- No regular expressions or wildcard characters are recognized except for `( *` *pinName* `)` in the `SPECIALNETS` section.

DEF contains the following informations,

1. `Version Namescasesensitive`
2. `Dividerchar`
3. `Busbitchars`
4. `Design`
5. `Technology`

6. `Units`
7. `History`
8. `Propertydefinitions`
9. `Diearea`
10. `Rows`
11. `Tracks`
12. `Gcellgrid`
13. `Vias`
14. `Regions`
15. `Components`
16. `Pins`
17. `Pinproperties`
18. `Blockages`
19. `Slots`
20. `Fills`
21. `Specialnets`
22. `Nets`
23. `Scanchains`
24. `Groups`
25. `Beginext`
26. `End design`

Order of DEF Statements

Standard DEF files can contain the following statements and sections. You must specify the statements and sections in the following order.

```
VERSION statement
 NAMESCASESENSITIVE statement
 DIVIDERCHAR statement
 BUSBITCHARS statement
 DESIGN statement
 [ TECHNOLOGY statement ]
 [ UNITS statement ]
 [ HISTORY statement ] ...
 [ PROPERTYDEFINITIONS section ]
 [ DIEAREA statement ]
 [ ROWS statement ] ...
 [ TRACKS statement ] ...
 [ GCELLGRID statement ] ...
 [ VIAS statement ]
 [ REGIONS statement ]
 COMPONENTS section
 [ PINS section ]
 [ PINPROPERTIES section ]
 [ BLOCKAGES section ]
 [ SLOTS section ]
 [ FILLS section ]
 [ SPECIALNETS section ]
 NETS section
 [ SCANCHAINS section ]
 [ GROUPS section ]
 [ BEGINEXT section ] ...
 END DESIGN statement
```

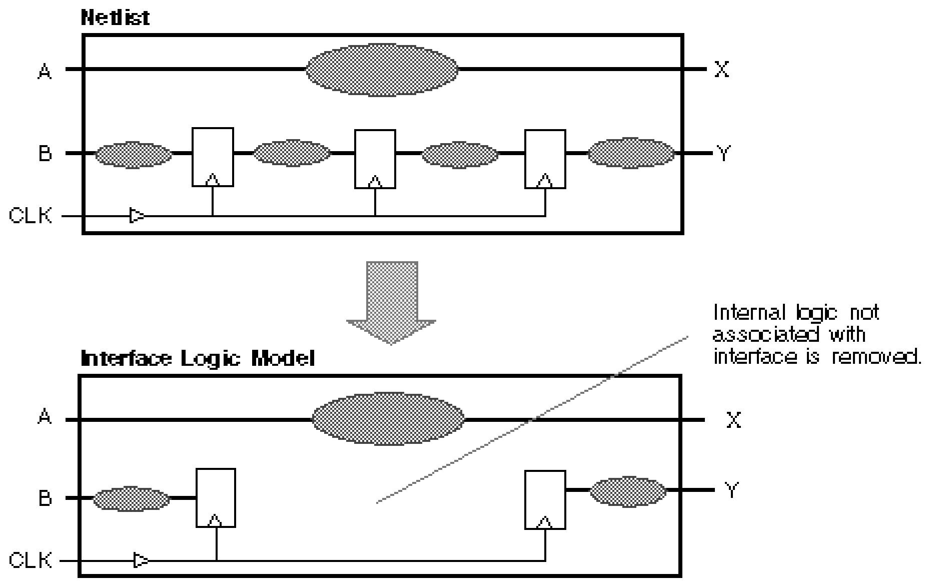
Note: You can include BEGINEXT sections at any point.

Interface Logic Modeling

An ILM is a partial netlist that retains the combinational logic from each input port to the first stage of sequential elements of the block. It is also the combinational logic from the last stage of sequential elements to each output port of the block. The clock paths to these sequential elements are also retained. Combinational paths from the input ports that do not encounter a sequential element and pass directly to an output port are also retained in the ILM.

The intent of an ILM is to produce a model that closely resembles the timing of the interface to that of the block-level netlist. The ILM generator takes as input the gate-level netlist for a block and generates a flattened Verilog netlist that contains only the interface logic, without the internal register-to-register logic, as shown in Figure 3-1.

Figure 3-1 Generated Interface Logic Model



Benefits, Flow, and Limitations of ILM

An ILM partial netlist can provide a smaller memory footprint, faster runtimes, and an extremely accurate timing of the interface logic of a block.

You can use ILMs in Standard Delay Format (SDF) or Standard Parasitic Exchange Format (SPEF) based flows. In addition, you can use ILMs in PT SI using both crosstalk and noise analysis.

To use the model in a hierarchical static timing analysis methodology, you must validate an ILM. This validation requires that the timing and scope match the block-level netlist usage within the design. A certain amount of time is needed to validate an ILM.

Specific Setups with ILM

High Fanout Ports

When generating an ILM, each input port that is not defined as a clock is considered as a data port. When the tool generates an ILM, it traces each input port to the first level of the sequential element. For input ports, such as scan_enable, this means that each flip-flop in the block netlist is kept.

Latches

For designs with latches, you should define the number of levels of borrowing for the latches at the interface. You should set the value of the `-latch_level` option of the `create_ilm` command to match the number of levels of latch borrowing.

Extracted Timing Models

Model Extraction Overview

The `extract_model` command generates a static timing model for the current design from its gate-level netlist. The generated model has the same timing behavior as the original netlist, and can be used in place of the original netlist in a hierarchical timing analysis.

Using an extracted timing model has these advantages:

- The generated model is usually much smaller than the original netlist. When you use extracted models in place of netlists in PT, you can significantly reduce the time needed to analyze a large design.
- Using a model in place of a netlist prevents a user from seeing the contents of the block, allowing the block to be shared while protecting the intellectual property of the block creator.

Extraction Requirements

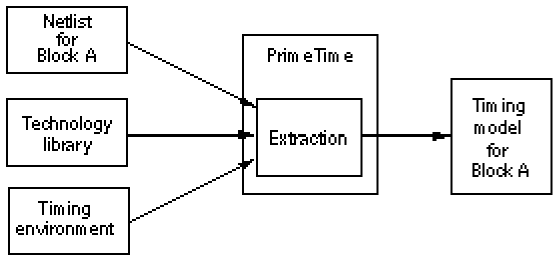
To generate an extracted timing model, you need the following:

- Block netlist
- Technology library
- Timing environment of the block (such as clocks and operating conditions)

Note:

Ensure that the netlist contains no timing violations before you generate a model.

Figure 4-1 Extraction Process



Extraction Process

Timing model extraction creates a timing arc for each path in the design from an input port to a register, an input port to an output port, and from a register to an output port.

Figure 4-2 Gate-Level Netlist

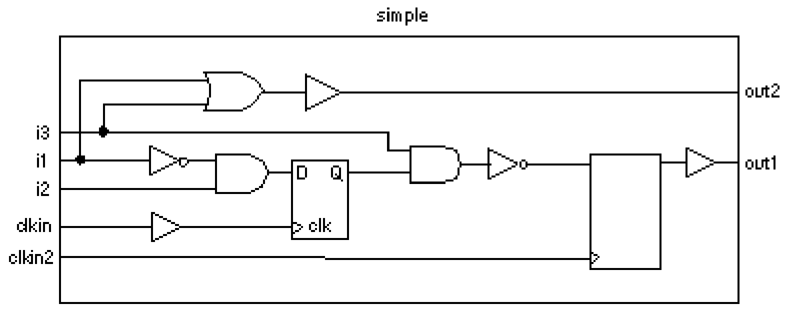
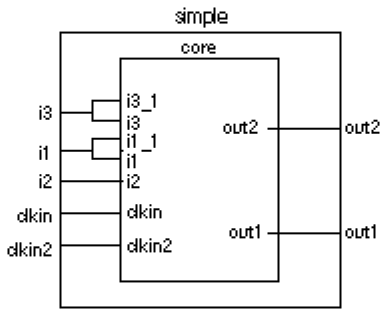
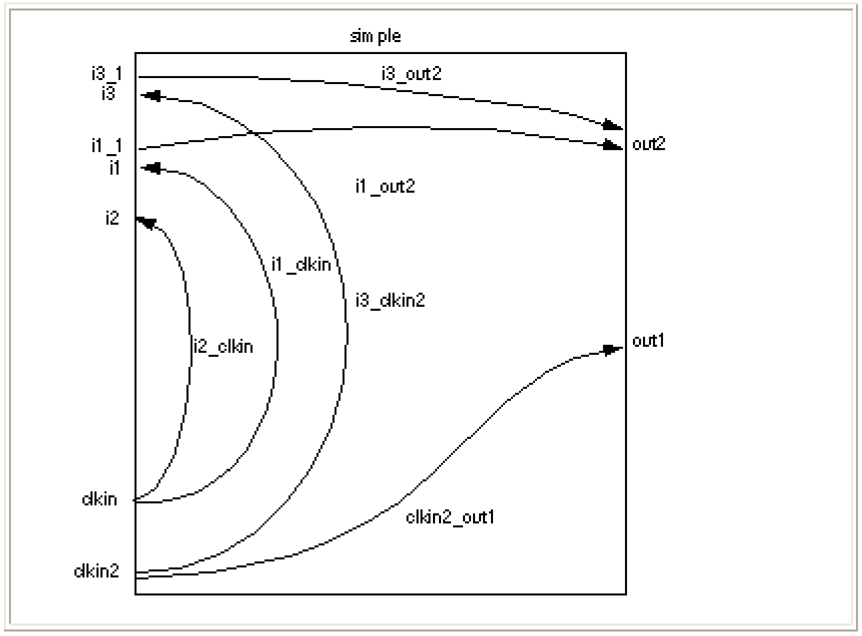


Figure 4-3 Netlist of Extracted Model



The generated timing model is another design containing a single leaf cell, as shown in Figure 4-3. The core cell is connected directly to input and output ports of the model design. This cell contains the pin-to-pin timing arcs of the extracted model. Figure 4-4 shows the timing arcs of the core cell. These arcs are extracted from the timing paths of the original design.

Figure 4-4 Timing Arcs of Core Cell



The delay data in the timing arcs is accurate for a range of operating environments. The extracted delay data does not depend on the specific values from input transition times, output capacitive loads, input arrival times, output required times, and so on. When the model is used in a design, the arc delays vary with the input transition times and output capacitive loads. This is called a "context-independent" model because it works correctly in a variety of contexts.

The characteristics of the extracted model depend on the operating conditions and wire load model in effect at the time of extraction. However, clocking conditions and external constraints do not affect the model extraction process. Commands such as `create_clock`, `set_clock_latency`, `set_clock_uncertainty`, `set_input_delay`, and `set_output_delay` do not affect the model extraction process, but the extracted model, when used for timing analysis, is sensitive to those commands.

Extracting Timing Paths

The following sections describe the extraction process for nets, paths, interface latch structures, minimum pulse width and period, false paths, clock-gating checks, and back-annotated delays.

Boundary Nets

The extracted model preserves the boundary nets of the original design, thereby maintaining accuracy for computing net delays after the model design is instantiated in another design.

Internal Nets

When performing extraction, if the internal nets are not annotated with detailed parasitics, PT computes the capacitance and resistance of internal nets. The delay values of the arcs in the model are accurate for the wire load model you set on the design before extraction. To get a model that is accurate for a different set of wire load models, set these models on the design and perform a new extraction. If internal nets are annotated with detailed parasitics (DSPF, Reduced Standard Parasitic Format (RSPF), or SPEF), PT does not use a wire load model to calculate the net capacitance and resistance. Instead, it takes the values from the detailed parasitics.

If the internal nets are back-annotated with delay or capacitance information, the back-annotated values are used instead of the computed values. For more information, see “Back-Annotated Delays”.

Paths From Inputs to Registers

A path from an input to a register is extracted into an equivalent setup arc and a hold arc between the input pin and the register’s clock. The setup arc captures the delay of the longest path from the particular input to all registers clocked by the same clock, plus the setup time of the register library cell. The hold arc represents the delay of the shortest path from the particular input to all register clocks, including the hold times of the register library cell.

The register’s setup and hold values are incorporated into the arc values. The setup and hold value of each arc is a function of the transition time of the input signal and the transition time of the clock signal. If an input pin fans out to registers clocked by different clocks, separate setup and hold arcs are extracted between the input pin and each clock.

Paths From Inputs to Outputs

A path from an input to an output is extracted into two delay arcs. One of the arcs represents the delay of the longest path between the input pin and the related output pin. The other arc represents the delay of the shortest path between the two pins.

The delay values for these arcs are functions of the input signal transition time and output capacitive load. The extracted arc is context-independent, resulting in different delays when used in different environments.

Paths From Registers to Outputs

A path from a register to an output is extracted into two delay arcs. One arc represents the longest path delay between the register’s clock pin and the output pin, and the other arc represents the shortest path delay between those pins. The clock-to-data output delay is included in the arc value. Different clock edges result in different extracted arcs.

Similar to input-to-output arcs, the delays of register-to-output arcs are functions of input transition times and output load capacitance. The arc delays differ depending on the environment in which they are used.

Paths From Registers to Registers

Paths from registers to registers are not extracted. For timing arc extraction, PT only traverses the interface logic.

Clock Paths

Delays and transition times of clock networks are reflected in the extracted model. However, clock latency is not included in the model. Therefore, the source latency and network latency must be specified when the timing model is used.

Interface Latch Structures

The `extract_model` command supports extraction of simple latch structures on the interface of the design (latches with fanin from a primary input or fanout to a primary output). The extracted model only preserves the borrowing behavior specified at the time of model extraction, not all possible borrowing behaviors.

Specifying Latch Borrowing

The `extract_model` command has two options that affect the extraction of latch behavior: `-context_borrow` and `-latch_level`. The `-context_borrow` option is the default behavior, which you can override by using the `-latch_level` option.

When you use `-context_borrow`, the model extractor identifies latches on the interface that borrow and traces through them, and stops each trace at a latch that does not borrow. On the other hand, using the `-latch_level` option specifies that all latch chains at the interface of the design have a specified length and definitely will borrow. Using the `-latch_level` option is recommended only when you are certain of the borrowing behavior of latches at the interface of the design, and you are able to specify a single latch chain length for the entire design.

The total amount of borrowing should not exceed one clock cycle. If it does, you need to manually set multicycle paths in the extracted model (using the `set_multicycle_path` command) to make the model timing match the netlist timing.

How the Model Extractor Handles Latches

When you use `the extract_model` command, PT handles the latches in the design as follows:

- It traces through borrowing latches and stops when it encounters a flip-flop, port, or nonborrowing latch.
- It extracts a setup arc when an input port goes through borrowing latches to a flip-flop or nonborrowing latch.
- It extracts a delay arc when an input port, or a clock connected to a flip-flop or nonborrowing latch, goes through borrowing latches to an output port.
- Neither the `-context_borrow` nor `-latch_level` option to `extract_model` causes the actual time borrowed (by a latch) to be factored into the generated setup or delay values.
- The timing of the extracted model matches that of the original netlist as long as the specified borrowing behavior at the time of model generation remains valid for the arrival times defined on input ports and clocks when the model is used. The actual time

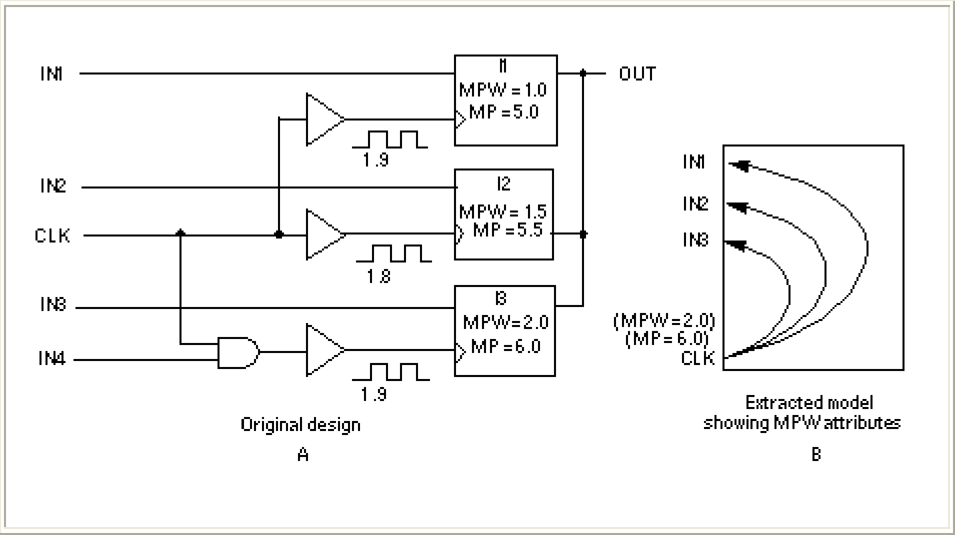
borrowed on a latch does not need to remain the same. Only the borrowing status (borrowing or not borrowing) must remain the same.

The model validation commands `write_interface_timing` and `compare_interface_timing` are useful for checking the model timing against the netlist timing, especially when there are latches on the interface. The `write_interface_timing` command traverses any borrowing latch in the netlist, which is necessary for matching the numbers reported for the model and for the netlist.

Minimum Pulse Width and Minimum Period

Minimum pulse width (MPW) and minimum period (MP) constraints on cell pins are propagated as MPW and MP attributes on the clock pins that are present in the extracted model, as shown in Figure 4-5.

Figure 4-5 Minimum Pulse Width and Minimum Period Extraction



In the original design A, all latches have MPW and MP attributes on their clock pins. In the extracted model B, the MPW attributes are translated into attributes on the clock port. Only the maximum of the MPW and MP values are used. CLK has the MPW_HIGH = 2.0 attribute set on it. In the original design A, the input pin IN4 does not have MPW or MP attributes, even though it is in the fanin of the clock pin of latch I3.

The extracted timing model only looks at the MPW attributes of the library pin, not the user MPW constraints applied with the `set_min_pulse_width` command. When the extracted timing model is instantiated, you can still include the user-defined MPW and MP constraints in the scripts at a higher level in the hierarchy.

The delay effects (nonsymmetrical rise or fall) and slew propagation along the path are not taken into consideration. Violations can occur when the clock pulse width decreases to less than the required minimum pulse width during the course of its propagation toward the latch clock pin. These violations are not reported for the model.

For example, in Figure 4-5 the clock waveform at the input latch I3 has a width of 1.9, which is less than the required 2.0. Using 2.0 for the clock port attribute without considering the slew effects causes this MPW violation to be missed.

False Paths

The model extraction algorithm recognizes and correctly handles false paths declared with the `set_false_path` command.

Clock-Gating Checks

If your design has clock-gating logic, the model contains clock-gating checks. Depending on your environment variable settings, these checks are extracted as a pair of setup and hold arcs between the gating signal and the clock, or a pair of no-change arcs between the gating signal and the clock. For more information, see "Extracting Clock-Gating Checks".

Back-Annotated Delays

If the design to be extracted is back-annotated with delay information, the model reflects these values in the extracted timing arcs. Transition time information for each arc is extracted in the same way as if the design were not back-annotated.

The delays of boundary nets, including any back-annotated delay values, are extracted if the `-library_cell` option is used. If the `-library_cell` option is not used, delays of boundary nets are not extracted, whether or not they are back-annotated. These delays are computed (or can be back-annotated) after the model is instantiated in a design and placed in context.

If the delays of boundary cells (cells connected directly to input or output ports) are back-annotated, the delays of the generated model are no longer context-independent. For example, the delays of paths from inputs are the same for different transition times of input signals. In addition, the delays of paths to outputs are not affected by the input transition time and output load.

Note:

Do not back-annotate the delays of output boundary cells if you want the output delays to change as a function of output load. Similarly, do not back-annotate the delays of input boundary cells if you expect the arc delays to depend on input transition times. To remove already-annotated information, use the `remove_annotated_delay` command.

Block Scope

An extracted timing model is context-independent (valid in different contexts) as long as the external timing conditions are within the ranges specified for block-level analysis. When the timing model is used at a higher level of hierarchy, it is no longer possible to check the internal clock-to-clock timing of the block. To ensure that the block context is valid for the original block-level timing analysis, you can check the block "scope" at the higher level.

When you generate the extracted model, you can also generate a “block scope” file containing information about the ranges of timing conditions used to verify the timing of the block. When you use the timing model in a chip-level analysis, you can have PT check the actual chip-level conditions for the block instance and verify that these conditions are within the ranges recorded in the scope file. This process ensures the validity of the original block-level analysis.

To generate the block scope file, use the `-block_scope` option of the `extract_model` command. To check the scope of the timing model during chip-level analysis, use the `check_block_scope` command. For more information, see Chapter 6, “Hierarchical Scope Checking.”

Noise Characteristics

PT SI users can analyze designs for crosstalk noise effects. An extracted timing model supports noise analysis by maintaining the noise immunity characteristics at the inputs of the extracted model, and by maintaining the steady-state resistance or I-V characteristics at the outputs of the extracted model. However, an extracted model does not maintain the noise propagation characteristics of the module. Also, any cross-coupling capacitors between the module and the external circuit are split to ground. Therefore, PT SI does not analyze any crosstalk effects between the extracted model and the external circuit.

Note:

To create timing models that can handle crosstalk analysis between modules and between different levels of hierarchy, use interface timing models instead of extracted timing models. See “Hierarchical Crosstalk Analysis”.

To include noise characteristics in the extracted model, use the `-noise` option of the `extract_model` command. Otherwise, by default, no noise information is included in the extracted model. If you use the `-noise` option, but no noise information is available in the module netlist, then the only noise characteristics included in the extracted model are the steady-state I-V characteristics of the outputs. In that case, the model extractor estimates the output resistance from the slew and timing characteristics.

An extracted timing model can be created in two forms: a wrapper plus core or a library cell. A wrapper-plus-core model preserves the input nets, the noise immunity curves of all first-stage cells at the inputs, the output nets, and all last-stage driver cells at the outputs (including their I-V characteristics). For a library cell, the model extractor must combine parallel first-stage cells at each input to make a single noise immunity curve, and must combine parallel last-stage cells at each output to make a single driver. For this reason, a wrapper-plus-core model provides better accuracy for noise analysis than a library cell model.

An extracted timing model with noise can be created in two forms: wrapper-plus-core or library cell. A wrapper-plus-core model preserves the input nets, the noise immunity curves of all first-stage cells at the inputs, the output nets, the noise I-V curve or steady-state resistance at the outputs, and the accumulated noise at the outputs. For a library cell, the model extractor combines the inputs nets (including coupling) and all leaf-cell noise immunity curves or DC margins into a single, worst-case noise immunity curve. Also, the library cell combines the worst-case resistance of last-stage cells and net resistance at each output to make a single I-V curve or resistance for each noise region. For this reason, a wrapper-plus-core model provides better accuracy for noise analysis than a library cell model.

To create a library cell model, the model extractor considers the noise immunity curves of individual cells connected in parallel within the module netlist. It considers the worst-case (lowest) data points of multiple curves and combines them into a single worst-case curve for the library cell input. For general information about static noise analysis, see the “Static Noise Analysis” chapter of the *PT SI User Guide*.

Other Extracted Information

In addition to the timing paths, the model extraction algorithm extracts other types of information:

Mode information

> Timing modes define specific modes of operation for a block, such as the read and write mode. PT extracts modes of the original design as modes for the timing arcs in the extracted model.
>
> For a description of how to define mode information for a design, see the *PT User Guide: Advanced Timing Analysis*.

Generated clocks

> Generated clocks you specify in the original design become generated clocks in the extracted model. For more information about generated clocks, see the *PT User Guide: Advanced Timing Analysis*.

Capacitance

> PT transfers the capacitance of input and output ports of the original design to the model’s extracted ports.

Design rules

> PT extracts the maximum transition at input and output ports. PT reflects maximum capacitance at output ports within the design in the extracted model.

Operating conditions

> The values of timing arcs in the extracted model depend on the operating conditions you set on the design when you perform the extraction.

Design name and date of the extraction

> PT preserves the design name and the date of extraction in the extracted model.

Multicycle paths

PT extracts multicycle paths and writes that information to a script containing a set of `set_multicycle_path commands` that can be applied to the extracted model.

Three-state arcs

Three-state arcs ending at output ports are included in the extracted model.

Preset and clear delay arcs

Preset and clear arcs are extracted if enabled. For more information, see "Extraction Variables".

Clock latency arcs

Clock latency arcs are extracted if enabled (see "Extraction Variables"). Clock latency arcs are used by tools such as Astro and Physical Compiler to compensate and balance clock tree skews at chip level. They are also reported by the `report_clock_timing` command in PT.

Limitations of Model Extraction

This section provides general guidelines and lists the general limitations of model extraction in PT, the limitations of using extracted models in Design Compiler, and the limitations of extracted models in .lib format.

Observe the following model extraction guidelines:

- Complex latch structures on the design interface are not supported. For more information, see "Interface Latch Structures".
- Clocks with multiple-source pins or ports are not supported.
- Avoid using the `-remove_internal_arcs` and `-latch_level` options.

The following environment variables affect model extraction:

```
extract_model_enable_report_delay_calculation
extract_model_gating_as_nochange
extract_model_status_level
extract_model_num_capacitance_points
extract_model_num_clock_transition_points
extract_model_num_data_transition_points
extract_model_capacitance_limit
extract_model_clock_transition_limit
extract_model_data_transition_limit
extract_model_with_clock_latency_arcs
timing_clock_gating_propagate_enable
```

```
timing_disable_clock_gating_checks
timing_enable_preset_clear_arcs
```

See the man page of each variable for a description of its effects on the extracted model.

The following limitations apply to model extraction:

Minimum and maximum delay constraints

Paths in the original design that have a minimum or a maximum delay constraint set on them are treated by extraction as follows:

- If you set the constraint on a timing path, PT ignores the constraint in the model.
- If you set the constraint on a pin of a combinational cell along a path, PT ignores the timing paths that pass through this pin. These paths do not exist in the extracted model.

For these reasons, it is recommended that you remove all minimum and maximum delay constraints from the design before you perform extraction.

Input and output delays

PT ignores input and output delays set on design ports when it extracts a model, unless your design contains transparent latches and you specify the `-context_borrow` option of the `extract_model` command.

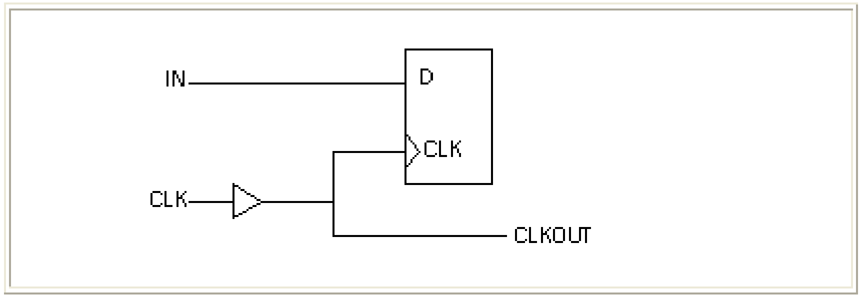
Input or output delay values set on pins of combinational cells along a timing path cause the paths that pass through these pins to be ignored in the extracted model. It is recommended that you remove input and output delay values set on internal paths (or set on objects other than ports) from the design before performing an extraction.

Extraction of load-dependent clock networks

If the delay of a clock to register path in the original design depends on the capacitive load on an output port, the delay of the extracted setup arc to the register in the model is independent of the output load.

For example, consider the design shown in Figure 4-6. In this design, the setup time from the IN input relative to the CLK clock depends on the load on the CLKOUT output. This occurs because the delay of the clock network depends on the load on the CLKOUT output and because the transition time at the register’s clock pin depends on the output load.

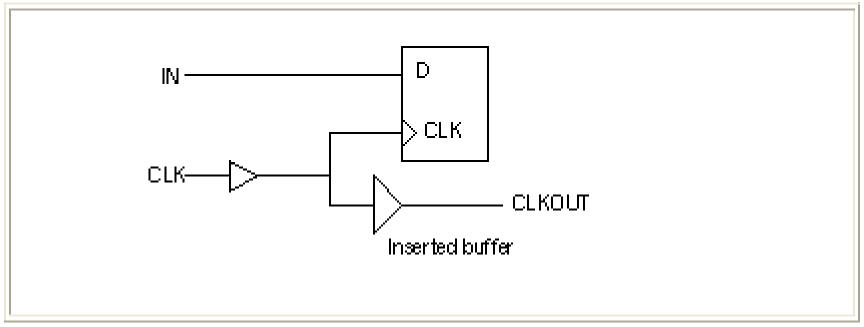
Figure 4-6 A Load-Dependent Clock Network



In the extracted model, the setup time from input IN to input CLK is independent of the load on output CLKOUT. However, the extracted CLK delay from input to CLKOUT remains load dependent.

To avoid inaccuracies in the extracted model, isolate the delay of the clock network inside the design to be modeled from the changes in the environment by inserting an output buffer, as shown in Figure 4-7.

Figure 4-7 Isolating the Delay of the Clock Network



Limitations of Extracted Models in Design Compiler

Design Compiler does not support all the features of extracted timing models. Specifically, Design Compiler ignores mode information. All modes are considered enabled.

Limitations of Extracted Models in .lib Format

You can optionally specify .lib format for the extracted model. This format is useful for exporting the timing model to an external tool. The following limitations apply to a model extracted in this format:

- Like extracted models in other formats, the .lib model captures only the timing behavior, not the logic, of the original netlist.

Extracted Model Types

The `extract_model` command can create two different types of timing models: a library cell or a wrapper with a core cell inside. A library cell serves as a port-to-port replacement for the design being modeled, with the boundary net capacitances approximated inside the model. A wrapper-and-core model consists of a central core cell surrounded by a wrapper design that preserves the original boundary nets.

The type of model created by `extract_model` depends on the `-library_cell` switch. If the `-library_cell` switch is present, PT creates a library cell. The name assigned to the new cell is the same as the design name.

If the `-library_cell` switch is absent, PT creates a wrapper design with a core cell. The `-format` option controls the output format of the core cell. However, PT always writes out the wrapper design in .db format, irrespective of the `-format` setting. The name of the core cell is *design_name*_core. The name of the wrapper design is *output*.db, where *output* is the value specified by the `-output` option.

You can optionally create a test design containing an instance of the extracted library cell. To do this, use the `-test_design` switch along with `-library_cell` in the `extract_model` command. The test design is named *design_name*_test and the output file is named *output*_test.db. PT does not write parasitics to the test design.

Extraction Variables

Several environment variables control the model accuracy and other aspects of timing model extraction.To attain the level of accuracy and complexity you want, consider each variable setting.

Table 4-1 lists the environment variables and summarizes how they affect the `extract_model` command. For more information on any particular variable, see the man page for that variable.

Table 4-1 Extraction Environment Variables

Extraction environment variables	**Effect on extract_model command**
`extract_model_enable_report_delay_calculation`	Specifies whether to allow the final user of the extracted model to get timing information using the `report_delay_calculation` command. Set this variable to false if the timing calculations for the model are proprietary.
`extract_model_capacitance_limit`	Specifies the maximum load

	capacitance on outputs. Model extraction characterizes the timing within this specified limit.
`extract_model_clock_ transition_ limit`	For clock input ports, specifies the maximum input port transition time. Model extraction characterizes the timing within this specified limit.
`extract_model_data_ transition_ limit`	For data input ports, specifies the maximum input port transition time. Model extraction characterizes the timing within this specified limit.
`extract_model_num_capacitance_ points`	Specifies the number of capacitance data points used in the delay table for each arc.
`extract_model_num_clock_transition_points`	Specifies the number of transition time data points used to make the delay table for each clock arc.
`extract_model_num_data_transition_ points`	Specifies the number of transition data points used to make the delay table for each data arc.
`extract_model_with_clock_latency_ arcs`	Specifies whether to enable modeling of clock tree insertion delay timing arcs in the extracted model.
`extract_model_status_level`	Specifies the amount of detail provided in model extraction progress messages.
`timing_enable_preset_clear_arcs`	Specifies whether to enable or disable preset and clear timing arcs.
`extract_model_gating_as_nochange`	Extracts clock-gating setup and hold checks as corresponding no-change arcs.
`timing_disable_clock_gating_checks`	When set to true, disables all clock-gating checks and does not extract them to the model.
`timing_clock_gating_propagate_ enable`	Determines whether data signals that gate a clock are propagated past the gating logic.

Variable Setting Guidelines

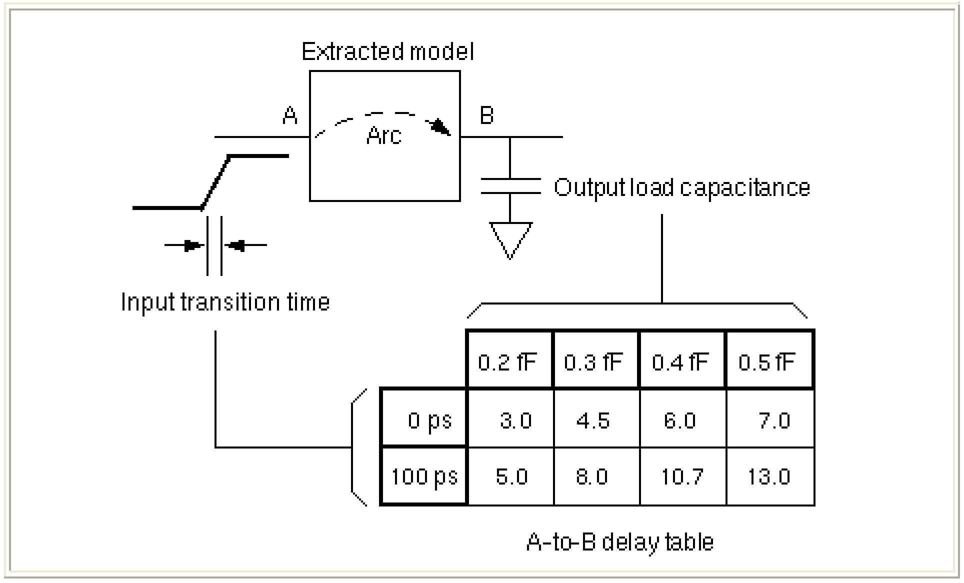
To generate accurate timing models, observe these guidelines:

- If the design contains latches on the interface, run `extract_model` with the `-context_borrow` option.
- Avoid the `-remove_internal_arcs` and `-latch_level` options of the `extract_model` command because they compromise accuracy.

Delay Table Generation

When you use `extract_model`, PT extracts a set of timing arcs and creates a delay table for each arc. A delay table is a matrix that specifies the arc delay value as a function of the input transition time and the output load capacitance. For example, the delay arc shown in Figure 4-8 uses a delay table like the 2-by-4 table shown in the figure. The amount of delay for the arc depends on the input transition time and output load capacitance.

Figure 4-8 Delay Table Example



	0.2 fF	0.3 fF	0.4 fF	0.5 fF
0 ps	3.0	4.5	6.0	7.0
100 ps	5.0	8.0	10.7	13.0

When you use the extracted model in a design, PT calculates the arc delay according to the context where the model is being used. The delay table makes the model accurate for the range of input transition times and output loads defined in the table.

For transition times or output loads at intermediate values within the table, PT uses interpolation between the table points to estimate the delay. A table with a larger number of entries (for example, a 10-by-10 matrix) provides better accuracy at the cost of more memory and CPU time for model generation.

For transition times and output loads outside of the ranges defined in the table, PT uses extrapolation to estimate the delay. A table that spans a larger range of transition time or load capacitance provides better accuracy for a larger range of conditions, but wastes memory and CPU resources if the range is larger than necessary to accommodate actual usage conditions.

You can control the number of data points and the range of conditions specified in delay tables of models generated by model extraction. To do so, set the following variables:

```
extract_model_num_capacitance_points
extract_model_num_clock_transition_points
extract_model_num_data_transition_points

extract_model_capacitance_limit
extract_model_clock_transition_limit
extract_model_data_transition_limit
```

The “number or points” variables control the number of data points in the generated delay tables. By default, these variables are set to 5, resulting in 5-by-5 delay tables. The “limit” variables control the range of capacitance and transition times covered in the generated delay tables. The default settings are 64 pf for the capacitance limit and 5 ns for the transition time limits. For more information, see the man page for each variable.

Preparing for Extraction

To prepare your design for extraction, consider the following steps:

- Set the model extraction environment variables
- Set the operating conditions for the extracted model
- Define the wire load models
- Identify pins being used as clocks in your design
- Define the clocks in the current design
- Prepare for extraction of latch behavior
- Identify false paths in your design
- Remove input or output delays set on objects that are not input or output ports
- Set the design modes, if any, for extraction
- Check the timing
- Prepare for crosstalk analysis

The sections that follow describe how to perform these tasks.

Setting the Extraction Variables

Several environment variables affect the extraction process, as explained in "Extraction Variables". Set them as needed by using one of these methods:

- Specify variables and their corresponding values individually. For example, enter
  ```
  pt_shell> set extract_model_data_transition_limit 2.0
  pt_shell> set extract_model_capacitance_limit 5.0
  ```
- Specify a variable and its value in a script file, then source the batch file. For example, enter
  ```
  pt_shell> source script_file
  ```
- Specify the variable and its value in the .synopsys_pt.setup (PT setup) file.

Setting Operating Conditions

The operating conditions for vendor libraries vary. For example, some conditions might be nominal, worst-case commercial (WCCOM), and best-case commercial (BCCOM).

Model extraction uses the current single operating condition or current minimum and maximum operating conditions. For example, for a single operating condition, enter:

```
pt_shell> set_operating_conditions WCCOM
pt_shell> extract_model -output model
```

To create a single model that has setup and hold arcs using maximum and minimum operating conditions, use commands like the following:

```
pt_shell> set_operating_conditions -min BEST_COND \
-max WORST_COND -lib your_lib -analysis_type bc_wc
pt_shell> extract_model -output file_name -format db
```

To create a single conservative model using on-chip variation to calculate the timing arcs, set the `-analysis_type` option to `on_chip_variation` instead of `bc_wc`.

You can also extract two different models at two different operating conditions, and then invoke the two models for min-max analysis. For example, you can use a script similar to the following:

```
set link_path "* your_library.db"
read_db BLOCK.db
link BLOCK
source constraint.pt
read_parasitics ...
read_sdf ...
set_operating conditions WORST_COND -lib your_library.db
extract_model -format db -output CORE_max
set_operating conditions BEST_COND -lib your_library.db
```

```
extract_model -format db -output CORE_min
remove_design -all
```

Then you can use instances of the core wrapper CORE_max.db in your design, and invoke the two models for min-max analysis using a script similar to the following:

```
set link_path "* CORE_max_lib.db your_library.db"
read_db CHIP.v
set_min_library CORE_max_lib.db -min_version \
  CORE_min_lib.db
link CHIP
...
```

Defining Wire Load Models

Defining the wire load models is a prerequisite for extraction unless the delays of all internal nets in the design have been back-annotated. The `extract_model` command uses wire load models to calculate the net capacitance and net delays in the extracted paths.

Use the `set_wire_load_model` and `set_wire_load_mode` commands to define one or more wire load models.

For example, enter the following commands to specify a 20-by-20 wire load model for the top design and a 05-by-05 wire load model for block u4.

```
pt_shell> set_wire_load_mode enclosed
pt_shell> set_wire_load_model -name 20x20 -lib class
pt_shell> set_wire_load_model -name 05x05 -lib class U4
```

The delay values of the arcs in the model are accurate for the wire load model you set on the design before extraction. To get an accurate model for a different set of wire load models, set wire load models on the design and perform a new extraction.

Defining Clocks

Defining all clocks in the current design is a prerequisite for extraction. You identify pins as clocks by using the `create_clock` or `create_generated_clock` command.

The `extract_model` command uses the clock period and input delay information under the following conditions:

- If you use the `-context_borrow` option, the model extractor uses the clock information to determine which transparent latches are borrowing.
- If you set a multicycle path definition at a point that the model extractor cannot easily move to the design boundary, the clock period determines which path is considered the critical path.

Typically, you place the created clocks on input ports of your design. Clocks created with the `create_clock` command on pins inside the design produce internal clocks in the generated model that require special attention. That is, you must specify the clocks on the pins in the model again; however, generated clocks referenced to boundary ports are extracted as expected and require no extra setup when using the extracted model.

The model extractor does not support multiple-source clocks. A multiple-source clock is a clock that is applied to more than one point in the design. For example, model extraction cannot work with a clock defined as follows:

```
pt_shell>    create_clock    -name    CLK    -period    10    \
[get_pins {Clk1 Clk2}]
```

The model extractor also does not support multiple clocks applied to the same point in the design. For example, model extraction cannot work with two clocks defined as follows:

```
pt_shell> create_clock -name CK1 -period 10 [get_pins Clk1]
pt_shell> create_clock -name CK2 -period 12 [get_pins Clk1]
```

Setting Up Latch Extraction

If the design you want to extract contains transparent latches and you want the generated model to exhibit time-borrowing behavior, you can use the `-context_borrow` and `-latch_level` options of the `extract_model` command to specify the method of latch extraction.

Observe the following guidelines when you use the `-context_borrow` and `-latch_level` options:

- If your design has no transparent latches or has transparent latches with no time borrowing, these option settings have no effect on the results.
- With a latch-based design, do not use the `-latch_level` option together with the `-context_borrow` option.
- Using `-context_borrow` is preferred.
- If the extracted model is to be used in an environment where the clock waveforms or input arrival times (or both) are expected to change drastically, do not use `extract_model`. Instead, generate an interface logic model as described in Chapter 3, "Interface Logic Models."

For more information on extraction of latch behavior, see "Interface Latch Structures".

Identifying False Paths

Before you perform model extraction, define false paths in your design by using the `set_false_path` command. Paths defined as false are not extracted into timing arcs by the `extract_model` command. For example, enter the following command to define the path between input IN4 and output OUT1 as a false path.

```
pt_shell> set_false_path -from IN4 -to OUT1
```

For more information about the `set_false_path` command, see the specifying timing exceptions details in the *PT User Guide: Fundamentals*.

Removing Internal Input and Output Delays

Remove input or output delays set on design objects that are not input or output ports. These objects are usually input or output pins of cells in the design. To remove delays, remove the commands from the original script that set the delays.

Controlling Mode Information in the Extracted Model

A complex design might have several modes of operation with different timing characteristics for each mode. For example, a microcontroller might be in setup mode or in monitor mode. A complex design might also have components that themselves have different modes of operation. A common example is an on-chip RAM, which has a read mode and a write mode.

The `extract_model` command creates a model that reflects the current mode settings of the design. The generated model itself does not have modes, but contains timing arcs for all the paths that `report_timing` detects with the modes at their current settings.

If you extract multiple timing models from a design operating under different modes, different conditions set with case analysis, or different exception settings, you can merge those models into a single model that has operating modes. For more information, see “Merging Extracted Models”.

Performing Timing Checks

To check your design for potential timing problems before you extract, use the `check_timing` command. Because problems reported by the `check_timing` command affect the generated model, fix these problems before you generate the timing model. For more information about the `check_timing` command, see the *PT User Guide: Fundamentals*.

Preparing for Crosstalk Analysis

PT SI is an optional tool that adds crosstalk analysis capabilities to PT. If you are a PT SI user and crosstalk analysis is enabled, you can have the `extract_model` command consider crosstalk effects when it calculates the timing arcs for the extracted model.

The model extractor cannot calculate the crosstalk-induced delays for all combinations of input transitions times for victim and aggressor nets. Instead, it accounts for crosstalk effects in a conservative manner. You specify a range of input delays and slews for the model prior to extraction. When you do `update_timing`, PT SI uses this information to calculate the worst-case changes in delay and slew that can result from crosstalk. The model extractor then adds the calculated changes to the extracted timing arcs.

This is the procedure for including crosstalk effects during model extraction:

1. Use the `set_input_delay` command with both the `-min` and `-max` options to specify the worst-case timing window for each input, given the current clock configuration.

2. Use the `set_input_transition` command with both the `-min` and `-max` options to specify the worst-case slew change for each input.

3. With PT SI crosstalk analysis enabled (by setting the `si_enable_analyis` variable to true) and with the design back-annotated with cross-coupling capacitors, perform a crosstalk analysis with `update_timing`. PT SI calculates the worst-case delay changes and slew changes under the specified conditions using on-chip variation analysis.

4. Run the `extract_model` command in the usual manner. PT performs model extraction without crosstalk analysis, then adds the fixed delta delay and delta slew values to the resulting timing arc values.

In step 4, PT only adds crosstalk values that make the model more conservative. For example, it adds a delta delay value to a maximum-delay arc only if the delta delay is positive, or to a minimum-delay arc only if the delta delay is negative.

The transition times you define with the `set_input_transition` command are used to calculate the delta delay values. These transition times are preserved in the extracted model as a set of design rules. When you use the extracted model, if a transition time at an input port of the model is outside of the defined range, you receive a warning message.

Model Extraction Options

The `extract_model` command has a `-format` option, which can be set to any combination of `db`or `lib` to generate models in .db format or .lib format. The .db format can be used directly by most Synopsys tools. Use the .lib format when you want to be able to read and understand the timing arcs contained in the model, or for compatibility with a third-party tool that can read .lib files. The .lib model can be compiled by Library Compiler to get a .db file. The `extract_model` command provides two command options that control the level of detail and accuracy of the extracted model:

- `-library_cell`, which generates the model as a library cell rather than a wrapper and core

- `-remove_internal_arcs`, which generates the model with internal arcs and timing points removed

By default, the generated model is a design containing a single core cell. The model preserves all the boundary nets in the original design and is the more accurate type of model.

The `-library_cell` option causes `extract_model` to generate the model as a library cell instead of a wrapper design and core. In this case, all boundary net delays are added to the arcs in the model.

The library cell model is simpler than the detailed model. However, the boundary net delays are not as accurate. In the case where outputs are shorted, the dependence of output delay on the capacitance of other outputs is lost.

You can use the `-test_design` option with the `-library_cell` option to have the model extractor generate a test design that instantiates the model. You can link the model to this design and obtain timing reports using the `report_timing` command. This design is not part of the model. It is provided as a convenience for testing the model after extraction.

If you have a PT PX license, you can use the `-power` option to generate a power model. By using this option, you can store power data for IPs and large blocks in a model that you can instantiate. This option performs an implicit `update_power`, if necessary; therefore, you must set the `power_enable_analysis` variable to `true` before using the `extract_model -power` option. For more information about this option, see the *PT PX User Guide* and the `extract_model` command man page.

Extracted Model Examples

Figure 4-9 shows a gate-level netlist for a design called simple.

Figure 4-9 Gate-Level Netlist

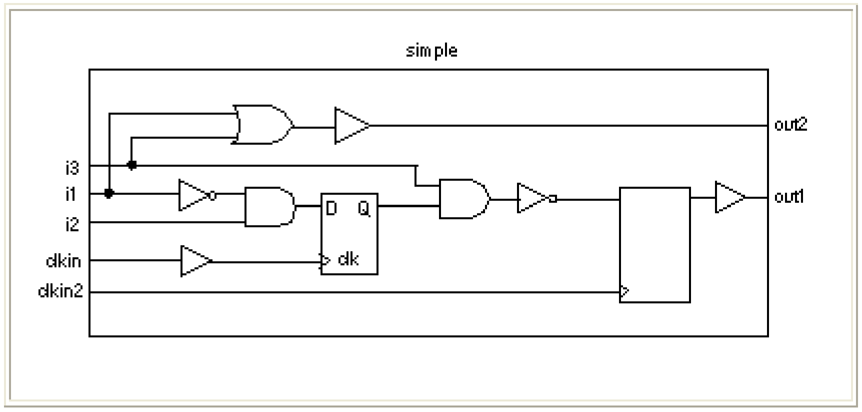


Figure 4-10 shows the extracted model when you do not use the `-library_cell` option. The generated model is a design called simple. This design contains an instance of a single cell called simple_core, which contains all the timing arcs in the model.

Figure 4-10 Extracted Model Without -library_cell

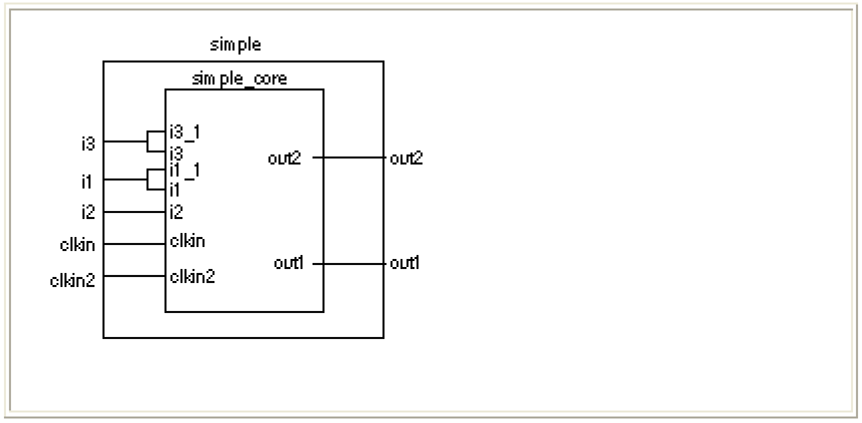


Figure 4-11 shows the core cell and its timing arcs.

Figure 4-11 Core Cell and Timing Arcs

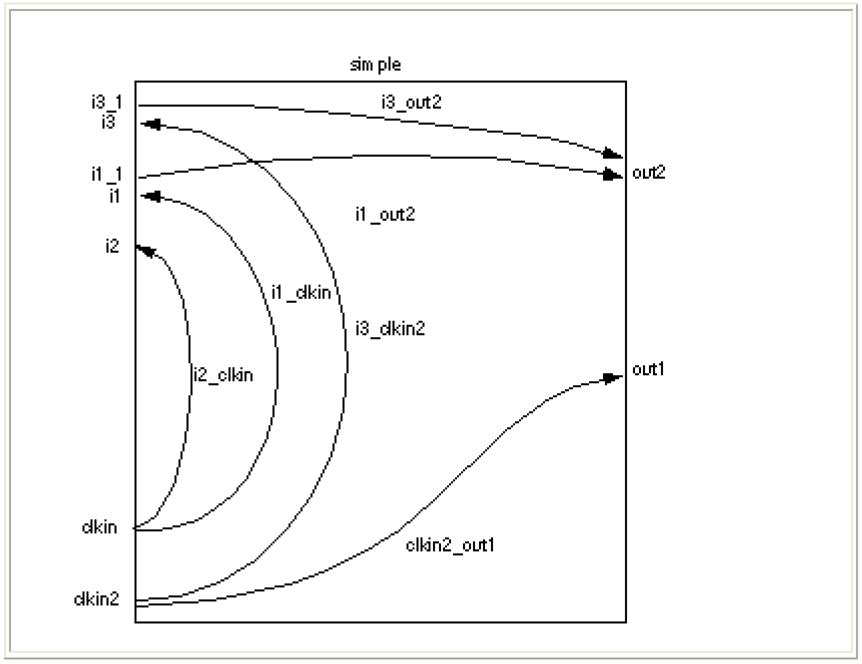


Figure 4-12 shows the model generated when you specify the `-library_cell` option. The model is a library cell called simple. This cell has the same input ports and output ports as the model. It also contains all the timing arcs. This model has no boundary nets. All boundary net delays are lumped into the generated model.

Figure 4-12 Model Extracted With -library_cell

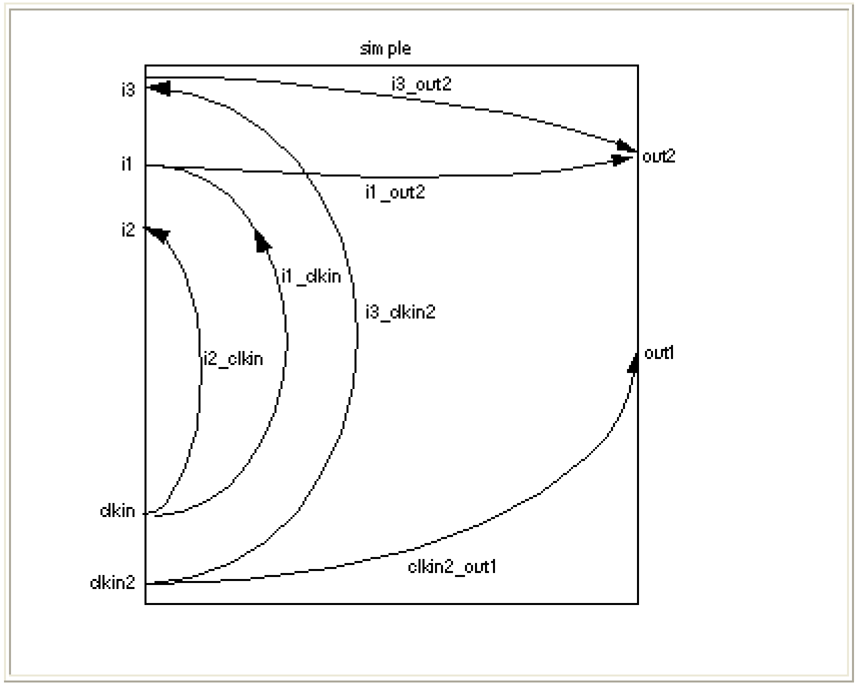
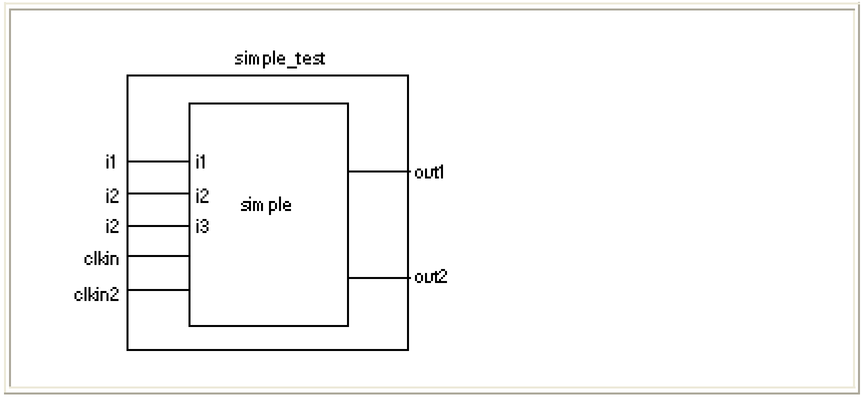


Figure 4-13 shows a test design generated when you specify the `-test_design` option. The design is called simple_test, which instantiates the model. This design is generated for your convenience in obtaining model and timing reports for the model if the model is generated as a library cell.

Figure 4-13 Test Design That Instantiates the Model



Extracting Clock-Gating Checks

The clock-gating setup and hold constraints, which are between the clock-gating pin and the clock, are converted to setup and hold checks between the primary input pin and input clock pin, and then written out to the model.

The following items can affect the way clock-gating checks are extracted:

Clock propagation

You can select whether to enable or disable clock delay propagation by using the `set_propagated_clock` and `remove_propagated_clock` commands. For more information, see the *PT User Guide: Advanced Timing Analysis*.

Clock-gating checks

You can use the `set_clock_gating_check` command to add clock-gating setup or hold checks of any desired value to any design object. See the information about clock-gating setup and hold checks in the *PT User Guide: Advanced Timing Analysis*.

Enabling or disabling gating checks

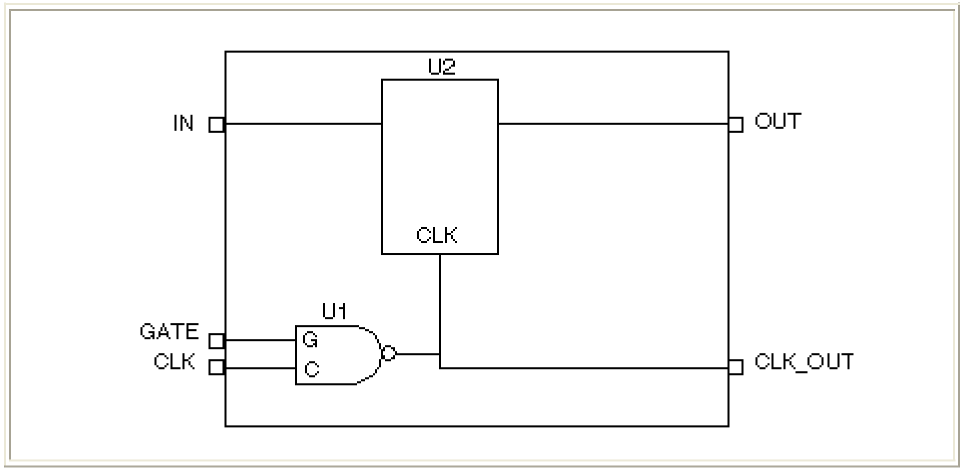
Extraction of clock-gating checks can be enabled or disabled with the `timing_disable_clock_gating_checks` variable prior to extraction.

Extraction of Combinational Paths Through Gating Logic

Combinational paths that start at input or inout ports, go through clock-gating logic, and end at output or input ports are extracted in the model.

In Figure 4-14, the path from GATE through U1/G to CLK_OUT is a combinational path in the clock-gating network and is extracted if the `timing_clock_gating_propagate_enable` variable is set to true.

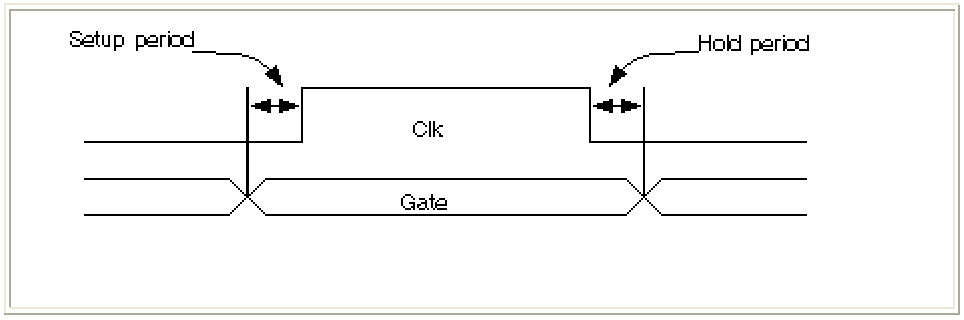
Figure 4-14 Combinational Path Through Clock-Gating Logic



Extraction of Clock-Gating Checks As No-Change Arcs

Clock-gating setup and hold checks can be grouped and merged to form a no-change constraint arc. The no-change arc is a signal check relative to the width of the clock pulse. PT establishes a setup period before the start of the clock pulse and a hold period after the clock pulse (see Figure 4-15).

Figure 4-15 No-Change Arc



A clock-gating setup check can be combined with its corresponding clock-gating hold check to produce two no-change arcs. One arc is the no-change condition with the clock-gating signal low during the clock pulse, the other is the gating signal high during the clock pulse.

The `extract_model_gating_as_nochange` variable, when set to true, causes the model extractor to convert all clock-gating setup and hold arcs into no-change arcs. When set to false (the default), the clock-gating checks are represented as a clock-gating constraint.

For example, if the variable is set to true, and if the clock pulse leading edge is rising and the trailing edge is falling, the no-change arcs produced are as follows:

- NOCHANGE_HIGH_HIGH Rise table taken from the setup arc rise table. Fall table taken from the hold arc fall table. See Figure 4-16.
- NOCHANGE_LOW_HIGH Rise table taken from the hold arc rise table. Fall table taken from the setup arc fall table. See Figure 4-17.

Figure 4-16 NOCHANGE_HIGH_HIGH Arc

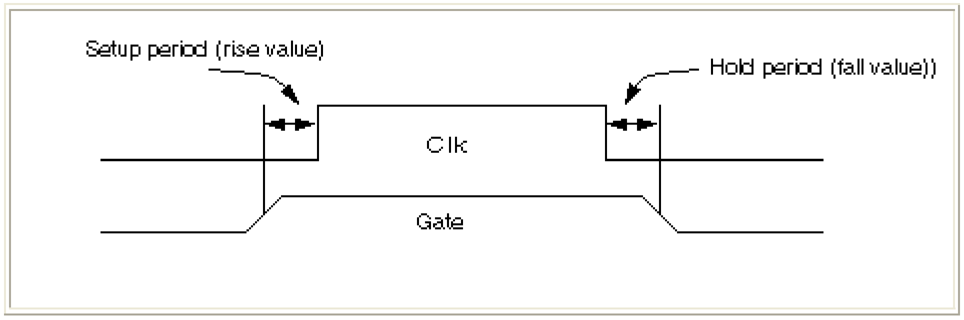
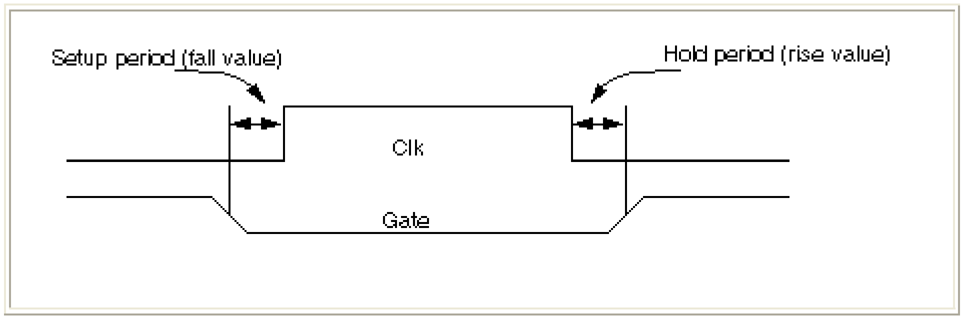


Figure 4-17 NOCHANGE_LOW_HIGH Arc



Extracting Constant Values

If the output ports are driven to a constant value, this value is preserved in the model. If the extracted model is used in a design, the constant value is propagated into its fanout logic.

Extracting Timing Exceptions

When you specify a timing exception for a path, PT does not treat that path as an ordinary single-cycle path. The most common type of exception is a false path. Other exception types are multicycle, `max_delay`, and `min_delay` paths. You can also use mode analysis (`define_design_mode`, `set_mode`) to control the timing analysis applied to certain defined paths.

The following rules apply to conflicts between timing exceptions and mode analysis:

- If a conflict arises between a specified false path and a moded endpoint, the false path specification prevails.
- If a conflict arises between a specified multicycle path and a moded endpoint, the mode is propagated.

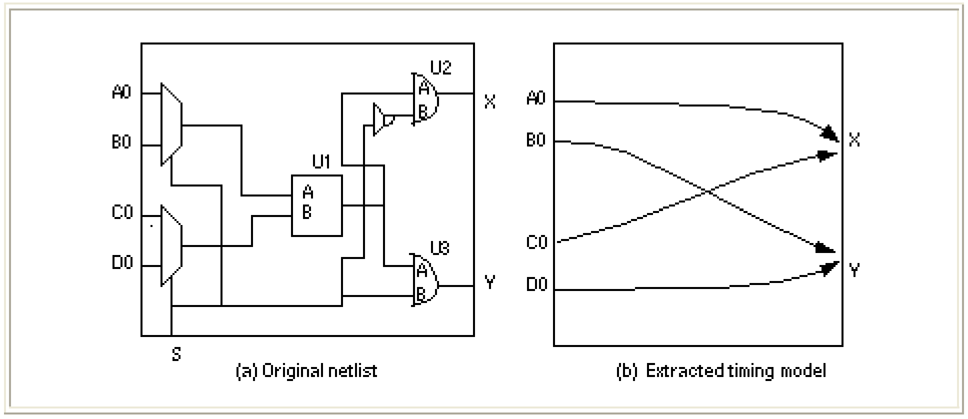
PT extracts paths with timing exceptions automatically, including exceptions you specify using the `-through` option of the exception-setting commands.

PT supports the following exceptions:

- False paths
- Multicycle paths
- Moded paths

Figure 4-18 shows an original design and an extracted timing model when false paths are present.

Figure 4-18 Original Design With False Paths and Extracted Model

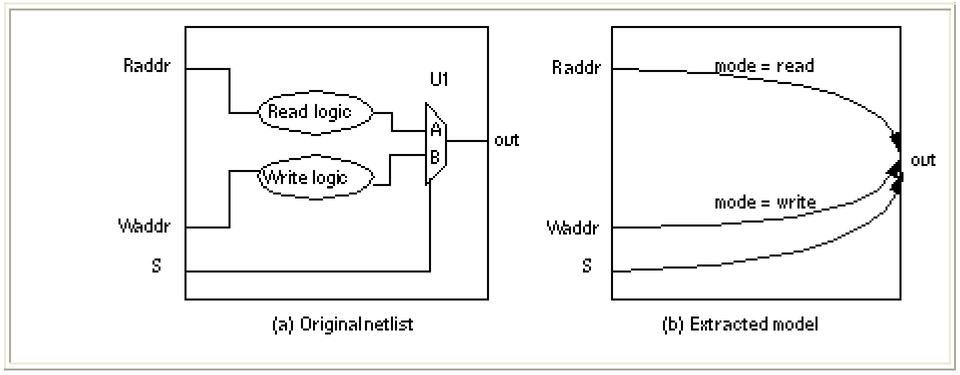


If you set the following false paths in the design shown in (a), the extracted model represented by (b) is the result.

```
pt_shell> set_false_path -from {A0 C0} -through U3/A
pt_shell> set_false_path -from {B0 D0} -through U2/A
```

There are timing arcs from A0 and C0 to X, but none from A0 and C0 to Y. Similarly, there are timing arcs from B0 and D0 to Y, but none from B0 and D0 to X. Figure 4-19 shows a design in which modes are defined with the `-through` option, together with the extracted timing model.

Figure 4-19 Extracting Modes With the -through Option



If you define the following modes for the netlist shown in (a), the result is the set of arcs represented in (b).

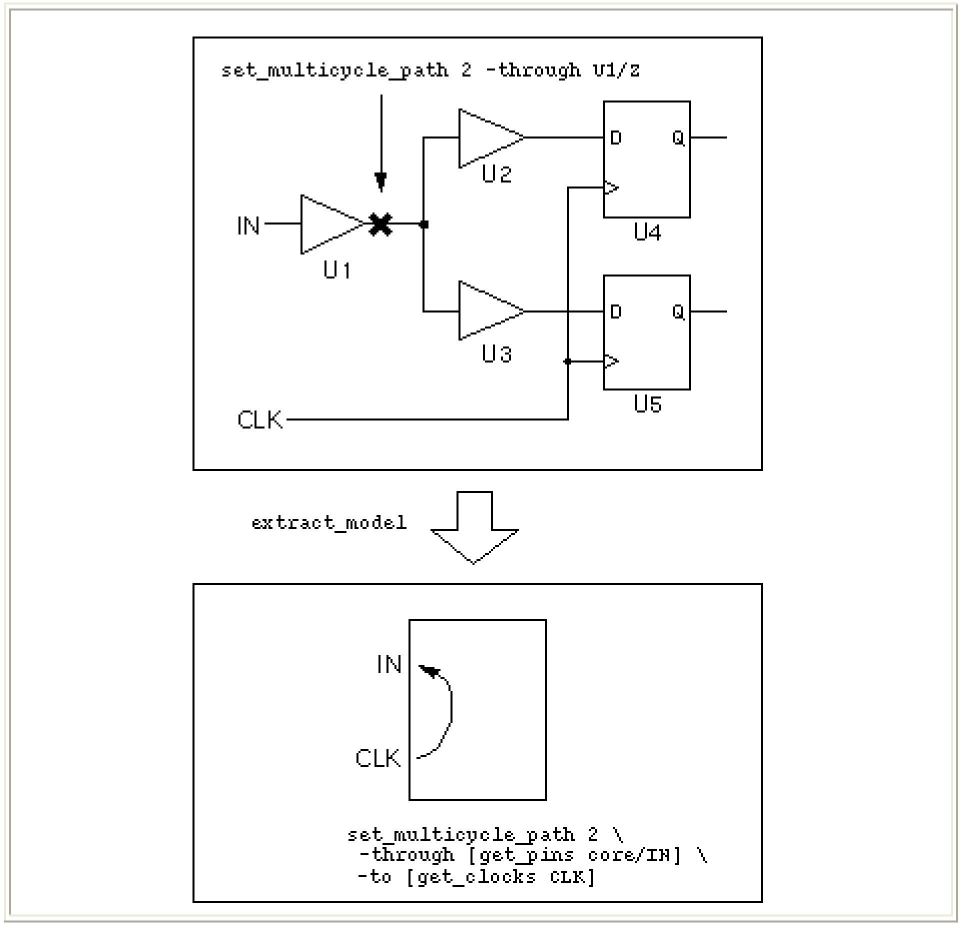
- Raddr to out in read mode
- Waddr to out in write mode
- An unmoded arc from S to out

```
pt_shell> set_mode -from Raddr -through U1/A read
pt_shell> set_mode -from Waddr -through U1/B write
```

If the original design has multicycle paths, the model extractor analyzes the multicycle paths and then writes out a script containing a set of equivalent `set_multicycle_path` commands that can be applied to the extracted model. When necessary, it also adjusts the setup, hold, and delay values for the timing arcs to correctly model the multicycle paths.

The name of the script file containing the multicycle path definitions is *output*_constr.pt, where *output* is the name specified by the `-output` option of the `extract_model` command. You need to apply this script when you use the extracted timing model. For example, Figure 4-20 shows a design in which a multicycle path has been defined at a point just inside the interface logic. The model extractor creates a setup arc from CLK to IN and writes the `set_multicycle_path` command shown in Figure 4-20, which must be applied to the extracted model for accurate results.

Figure 4-20   Multicycle Path Extraction With an output_mcp.pt File

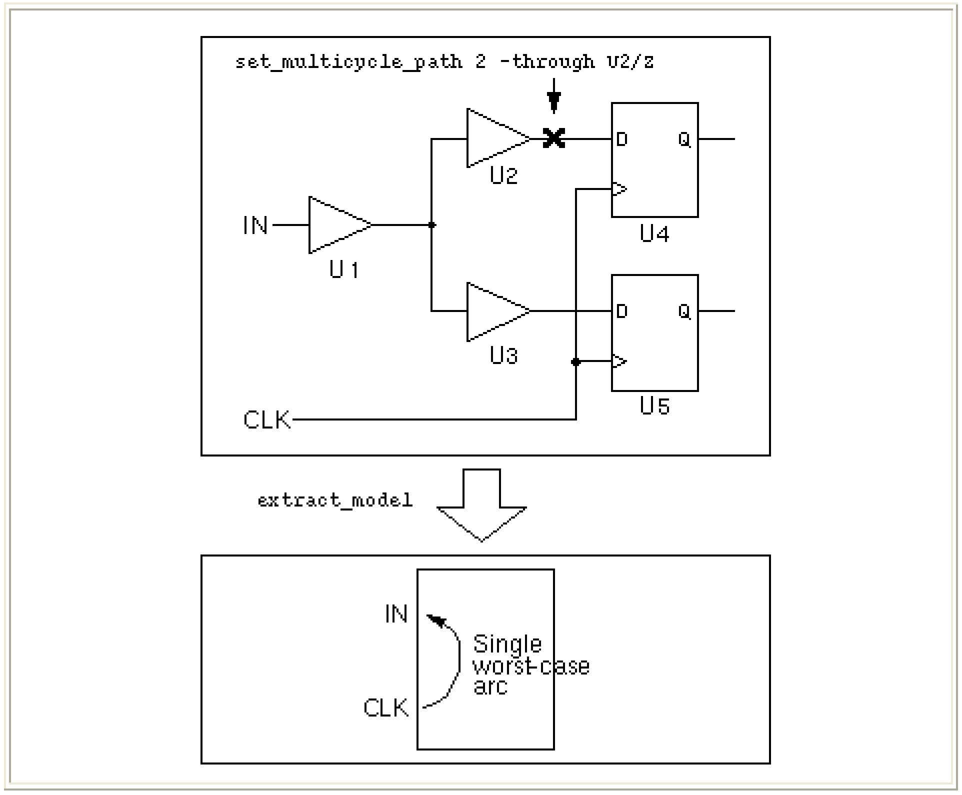


```
set_multicycle_path 2 \
 -through [get_pins core/IN] \
 -to [get_clocks CLK]
```

A timing exception that only applies to clocks (for example, a false path between to clocks) is written to the same constraint file as the multicycle paths, *output*_constr.pt. You need to apply this script when you use the extracted timing model.

If a multicycle path has been defined that cannot be moved to the boundary of the design, the model extractor does not write out a multicycle path definition. Instead, it creates worst-case setup, hold, and delay arcs using the clock period defined at the time of model extraction, as demonstrated in Figure 4-21.

Figure 4-21 Multicycle Path Extraction Without an output_mcp.pt File



Restricting the Types of Arcs Extracted

By default, the `extract_model` command extracts a full set of timing arcs for the model, including the following types: minimum sequential delay, maximum sequential delay, minimum combinational delay, maximum combinational delay, setup, hold, recovery, removal, clock gating, and pulse width arcs.

You can optionally extract only certain arc types for a model. This feature can be useful for debugging purposes when you are only interested in one type of arc, and you want to run multiple extractions under different conditions. Model extraction runs faster with this option because PT only spends time extracting the requested arcs.

To restrict the types of arcs extracted, use the `-arc_types` option of the `extract_model` command, and specify the types of arcs you want to extract. These are the allowed arc type settings:

- `min_seq_delay`: minimum-delay sequential arcs

- `max_seq_delay`: maximum-delay sequential arcs
- `min_combo_deay`: minimum-delay combinational arcs
- `max_combo_delay`: maximum-delay combinational arcs
- `setup`: setup arcs
- `hold`: hold arcs
- `recovery`: recovery arcs
- `removal`: removal arcs
- `pulse_width`: pulse width arcs

Back-Annotated Delay and Layout Information

You can back-annotate two types of information to a design in PT:

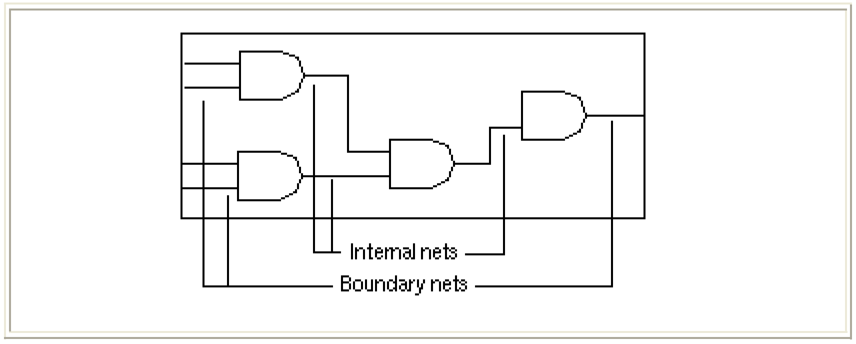
- Layout information, including lumped net capacitance or resistance and detailed RC information using standard parasitic exchange formats.
- Delay information, including net delays, cell delays, or both, using SDF. These delays are calculated for a given transition time.

In the absence of both types of information, PT calculates arc delays in the model using the cell libraries and calculates net delays using the wire load model set on the design.

Back-Annotated Delay on Nets and Cells

PT handles back-annotation of delays on internal nets and boundary nets in different ways. Figure 4-22 shows the locations of boundary nets and internal nets in a simple design.

Figure 4-22 Boundary Nets and Internal Nets



Internal net delays are included in the extracted arcs of the model. Boundary net delays cannot be computed until the model is used because the delays depend on the connection of the model to the outside world, unless the `-library_cell` option is used for model extraction, in which case boundary net delays are included in the extracted timing arcs.

Internal Nets

When the internal nets in the design are back-annotated with lumped capacitance, PT computes the net delays, cell delays, and transition times at cell outputs for model generation using this capacitance instead of the wire load models.

When SDF is back-annotated to internal nets, PT uses the back-annotated net delays during model extraction. For the transition times at cell outputs to be accurately computed, the net capacitance must also be back-annotated.

The extractor also supports extraction of designs in which internal nets are back-annotated with detailed parasitics in DSPF, RSPF, or SPEF.

Boundary Nets

By default, PT writes all annotated parasitics of the boundary nets to the extracted timing model, including both detailed and lumped RC information. This preserves the dependence of the net delays on the environment when the model is used. However, using the `ignore_boundary_parasitics` option of the `extract_model` command causes PT to ignore the boundary parasitics for timing model extraction, including both detailed and lumped RC information.

When the extracted model type is a library cell (created by using the `-library_cell` option of `extract_model`), PT lumps all of the boundary net capacitors onto the ports of the model. For primary input nets, it adds the wire delay resulting from using the driving cell present on the input port at the time of model extraction. For primary output ports, it calculates the wire delay by using the range of external loads characterized for the driving device.

Cells With Annotated Delays

If the cells in the design are back-annotated with SDF delay information, PT uses this information when it extracts arc delays in the model. Because the SDF information is computed at a particular transition time, the delays of arcs in the model are accurate for this transition time. The model can be inaccurate for other transition times.

To ensure that the model is context independent, do not annotate SDF to cell delays. If you need to annotate cell delays, the extracted model is accurate for the transition time at which the SDF is generated. To create some dependence on transition time and capacitive load, you can remove the annotations on boundary cells by using the `remove_annotated_delay` and `remove_annotated_check` command before you extract the model.

Guidelines for Back-Annotation

When you perform model extraction, keep in mind the following guidelines for various types of layout and delay data back-annotation:

Internal nets

Use detailed RC data if available. This data generates the most accurate context-independent model.

- Use both SDF and lumped RC data if both are available.
- Use lumped RC data if only this data is available.
- Use SDF data alone if only this data is available. In this case, the extracted model is accurate only for the transition times at which the SDF is generated. However, using SDF data alone is not recommended.

Cell delays

To ensure that the model is context-independent, do not annotate SDF to cell delays. If you need to annotate cell delays, the extracted model is accurate only for the transition times at which the SDF is generated. To create some dependence on transition time, you can remove the annotations on boundary cells by using the `remove_annotated_delay` command and the `remove_annotated_check` command before you extract the model.

Boundary nets

If you have the detailed parasitics for the boundary nets, annotate the information before you extract.

Performing Model Extraction

After you understand the tasks necessary to set up your modeling environment (see “Preparing for Extraction”), you can extract a timing model.

The following sections explain how to do this:

- Loading and Linking the Design
- Preparing to Extract the Model
- Generating the Model

Loading and Linking the Design

Load and link your design into PT by following these steps:

1. Set the search path of your library. For example, enter
   ```
   pt_shell> set search_path \
   ". /remote/release/v2000.11/libraries/syn"
   ```
2. Set the link path of your library:
   ```
   pt_shell> set link_path "* class.db"
   ```
3. Read your design into PT:
   ```
   pt_shell> read_db design.db
   ```
4. Link your design:
   ```
   pt_shell> link_design design
   ```

Preparing to Extract the Model

Preparing a model for extraction includes the following steps:

1. Define wire load models for your design. Enter
   ```
   pt_shell> set_wireload_model -name wire_model_name
   ```
2. Define the clocks in your model. For example, enter
   ```
   pt_shell> create_clock -period 10 CLK1
   pt_shell> create_clock -period 20 CLK2
   ```
3. Set the false paths in the design:
   ```
   pt_shell> set_false_path -from IN4 -to OUT1
   ```
4. Check your design for potential timing problems:
   ```
   pt_shell> check_timing
   ```
5. Verify that the design meets timing requirements:
   ```
   pt_shell> report_timing
   ```

6. Set the model extraction variables, if applicable. PT uses the default values for any variables you do not set. For example, enter

```
pt_shell> set extract_model_capacitance_limit 5.0
pt_shell> set extract_model_transition_limit 2.0
```

Generating the Model

Determine the options you want to use for the `extract_model` command and issue the command. For example, enter the following command to extract a timing model for the current operating conditions and create a .db model file and a design in .db format:

```
pt_shell> extract_model -output example_model \
-format {db}
```

PT displays a report similar to this:

```
Warning: Environment variable
'extract_model_min_resolution' is
not defined.
Using default value '0.1'. (MEXT-6)
Using default value '0.45'. (MEXT-6)
Wrote model library core to './example_model_lib.db'
Wrote model to './example_model.db'
```

If you have generated a .lib version of the timing model, you may notice it contains user-defined attributes. For a description of these attributes, see "PT User-Defined Attributes in .lib".

PT User-Defined Attributes in .lib

User-defined attributes are used by PT and can be used by other Synopsys tools. These attributes do not affect timing analysis when using the model.

- `min_delay_flag` - Used to help `merge_models` separate minimum and maximum arcs between a pair of pins with the same sense. This attribute is used to properly merge the arcs and maintain the arc configuration across different modes.
- `original_pin` - Used to help maintain the mapping of extracted time model cell pins to the original pin names that were defined in the block netlist prior to extraction.

Merging Extracted Models

A module can have different operating modes, such as test mode and normal operating mode. The timing requirements for a module can be quite different for different operating modes. In these cases, you need to extract a separate model for each mode. For each extraction, place the module into the applicable mode by setting the instance or design modes, by using case analysis, and by setting the applicable timing exceptions.

To use different models extracted under different operating modes, you can swap each model into the higher-level design by using `swap_cell`. However, instead of keeping and using multiple extracted models, you can optionally merge them into a single, comprehensive model that has different timing arcs enabled for different operating modes. Then you can use this single model and change its behavior by setting it into different modes.

This is the basic procedure for creating and merging timing models for different operating modes:

1. Load and link the module netlist.
2. Back-annotate the SDF data or detailed parasitics.
3. Apply the constraints for the first operating mode.
4. Extract a model in .lib format for the operating mode.
   ```
   pt_shell> extract_model -format lib -output model_1
   ```
5. Remove all constraints on the design.
6. Repeat steps 3, 4, and 5 for each additional operating mode.
7. Merge the generated .lib models
   ```
   pt_shell> merge_models \
   -model_files {model_1.lib model_2.lib ...} \
   -mode_names {mode_1 mode_2 ...} \
   -group_name etm_modes \
   -output my_model \
   -formats {db lib} \
   -tolerance 0.1
   ```

Each `extract_model` command generates a .lib file and a .data file for the extracted model. The `merge_models` command merges the extracted models into a single model that has different operating modes. You specify the operating mode when you use the model.

In the `merge_models` command, you specify the .lib files, the names of the modes corresponding to those models, the name of the new model being generated, the model formats to be generated (.db and .lib), an optional mode group name, and an optional tolerance value. For more information about mode groups, see *PT User Guide: Fundamentals*.

The tolerance value specifies how far apart two timing values can be to merge them into a single timing arc. If the corresponding arc delays in two timing models are within this tolerance value, the two arcs are considered the same and are merged into a single arc that applies to both operating modes. The default tolerance value is 0.04 time units.

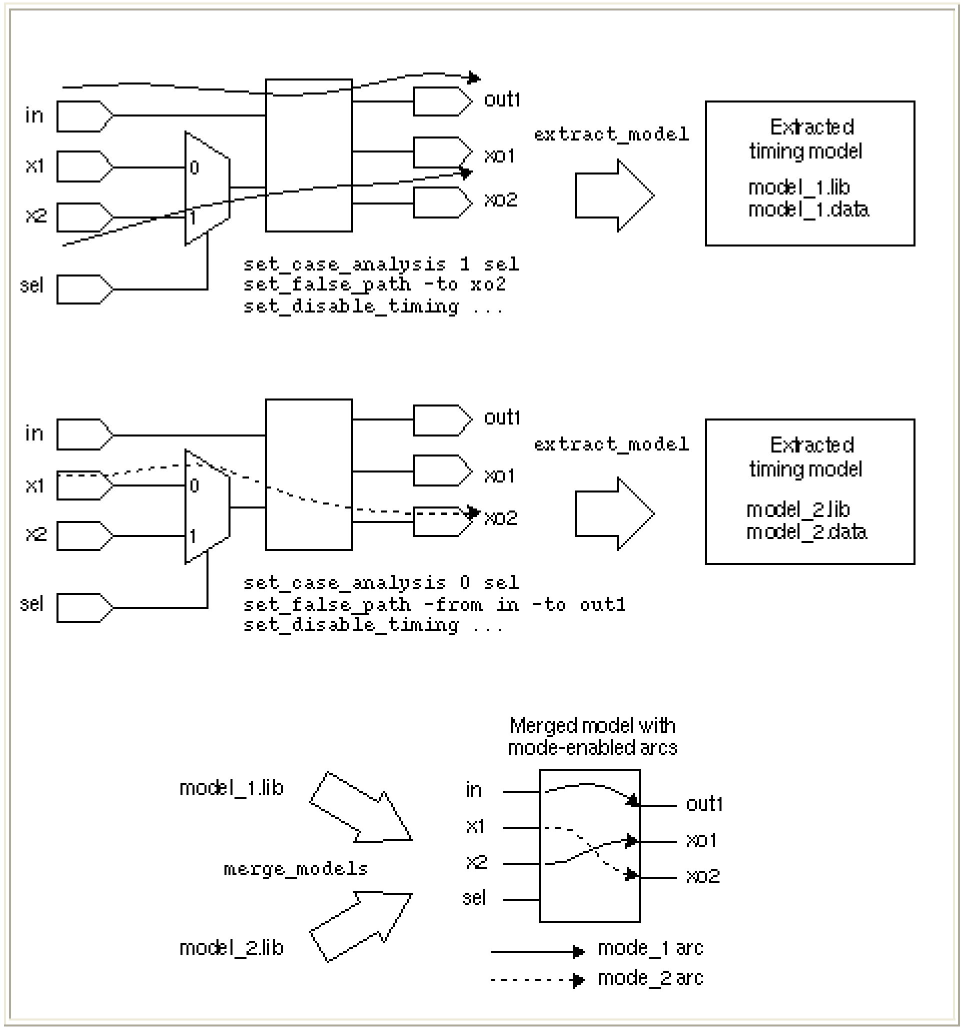
PT can handle more than one mode per timing arc, but some tools cannot. To generate a merged model that can work with these tools, use the `-single_mode` option in the `merge_models` command. This forces the merged model to have no more than one mode per timing arc, resulting in a larger, less compact timing model.

It might be necessary to disable all arc merging. For instance, if you are performing two independent merges and the resulting merged models must have the same number of arcs. This occurs when one merged model has the maximum and another has the minimum operating condition. To use these two models in the `set_min_library` command, they must have exactly the same number of arcs. To disable all arc merging, you can use the `-keep_all_arcs` option of the `merge_models` command.

The models being merged must be consistent, with the same I/O pins, operating conditions (process/voltage/temperature), and so on. Timing arcs that are different are assigned to the modes specified in the `merge_models` command. Design rule constraints (DRC), such as minimum and maximum capacitance and transition time, are allowed to be different. In those cases, the more restrictive value is retained in the merged model.

Figure 4-23 shows an example of a module from which two timing models are extracted, where the extracted models are merged into a single timing model having two operating modes.

Figure 4-23 Model Extraction and Merging Example



To use a merged timing model with modes:

1. Set the `link_path` variable to include the path to the merged model in .db format.
2. Load and link the top-level design.
3. Apply the constraints and back-annotation on the design.
4. Using the `set_mode` command, set the mode on the module instance to the desired mode.
5. Run the timing analysis.
6. Repeat steps 4 and 5 for each mode that you want to analyze.

Here are some points to consider for model merging:

- To retain in the merged model, case values propagated to the output pins of the models being merged must be the same in all models. If there are mismatching case values, PT ignores them and issues a warning message.
- Any model that already has modes or moded arcs defined in it cannot be merged.
- Any generated clocks in the models to be merged must be exactly the same.

For more information, see the man page for the `merge_models` command.

What are all the SDC Constraints and related commands?
Operating conditions
set_operating_conditions
Wire load models

set_wire_load_min_block_size	set_wire_load_mode
set_wire_load_model	set_wire_load_selection_group

System interface

set_drive	set_driving_cell
set_fanout_load	set_input_transition
set_load	set_port_fanout_number

Design rule constraints

set_max_capacitance	set_max_fanout
set_max_transition	set_min_capacitance

Timing constraints

create_clock	create_generated_clock
set_clock_gating_check	set_clock_latency
set_clock_transition	set_clock_uncertainty
set_data_check	set_disable_timing
set_input_delay	set_max_time_borrow

set_output_delay	set_propagated_clock
set_resistance	set_timing_derate
Timing exceptions	
set_false_path	set_max_delay
set_min_delay	set_multicycle_path
Area constraints	
set_max_area	
Power constraints	
set_max_dynamic_power	
set_max_leakage_power	
Logic assignments	
set_case_analysis	set_logic_dc
set_logic_one	set_logic_zero
Multivoltage constraints	
create_voltage_area	
set_level_shifter_strategy	
set_level_shifter_threshold	

design	current_design	A container for cells. A block.
clock1	get_clocks	A clock in a design.
	all_clocks	All clocks in a design.
port	get_ports	An entry point to or exit point from a design
	all_inputs	All entry points to a design.
	all_outputs	All exit points from a design
cell	get_cells	An instance of a design or library cell.
pin	get_pins	An instance of a design port or library cell pin.
net g	get_nets	A connection between cell pins and design ports.
library	get_libs	A container for library cells.
lib_cell	get_lib_cells	A primitive logic element.
lib_pin	get_lib_pins	An entry point to or exit point from a lib_cell.

18. Designs Executed (Project Related)

Details	**DDR**	**DXO**	**PT2**	**PT1**
Type	Block Design	Block Design	Full chip (Flat Design)	Full chip (Flat Design)
Technology	65 nm	130 nm	130nm	130 nm
Tech Lib Name	065lv	CL013G	CL013lv	CL013lv
Operating Frequency	133 MHz Half Cycle Pah	195 MHz	125 MHz/500 MHz (Derived clock)	
No of Clocks	3 Clocks	1 Clocks	9 Clocks	

Area	4 Sq.mm size 2 x 2 mm	8 Sq.mm size 2.82 x 2.82 mm	15 Sq.mm size 5 x 3mm	
No of Macros	16	61	13 Macro and 1 PLL	
No of IO Pins	473	656	Total 189 120 Signal, 60 Power Pads	
No of Layers used	6 layers used out of 8	7 layers used out of 8	8 layers used out of 8	
Std Cell Instances	260 k gates	200 k gates	520 k gates	
Utilization	85 %	94 %	83	
Operating Voltage	Core 1.0 V, IO 2.5v	1.0 V / 3.3 V	1.0 V / 3.3 V	
Congestion Number				
Area of NAND2		5 Sq.um	5 Sq.um	
Gate Density	854 kgates/Sq.mm	219 kgates/Sq.mm	219 kgates/Sq.mm	
Std Cell Height		3.6 um / 9 tracks		

TSMC Advanced Technology Overview

Parameters	**65 nm**	**90 nm**	**130 nm**	**180 nm**
Voltage Range	1.0 to 1.2 V	1.0 to 1.2 V	1.0 to 1.5 V	1.2 to 1.8 V
IO Voltage	1.8, 2.5 / 3.3V	1.8, 2.5 /3.3 V	2.5 and 3.3, 3.5 V	
Gate Density	854 kgates/Sq.mm	436 kgates/Sq.mm	219 kgates/Sq.mm	110 kgates/Sq.mm
SRAM Memory Density	0.97 Sq.um(8T) 0.499 Sq.um(6T)	1.99 Sq.um(8T) 0.99 Sq.um(6T)		

1. **To get the guidelines to be followed for doing sanity tests on the inputs received from the front-end.**
2. **If Five Memories are placed closed to each other and placed in the Top corner of the Design. What are the issues are faced in the design?**
3. **Evaluating the hard macro for suitability towards any tool.**
4. **Memory requirements needed for any block backend implementation to be worked out.**
5. **Scan chain reordering concepts at backend level.Selection of IO pad**
6. **How are nets crossing multi-voltage domain handled?**

Standared Low Power Methods:
Therea are number of power reduction methods,

1. Clock gating
2. Gate level Power Optimization
3. Multi-VDD
4. Multi-Vt

1. Clock Gating:
Up to 50% or even more of the dynamic power can be spent in the clock buffers. This result makes intuitive sense since the buffers have the highest toggle rate in the system, there are lots of them, and they often have the high drive strength to minimize the clock delay. In addition, the flops receiving the clock dissipate some dynamic power even if the input and output remain the same.

The most common way to reduce this power is to turn clocks off when they are not required. This approach is known as clock gating.

From some analysis and experiments, It is decided to use clock gating only on registers with a bit-width of at least three. The clock gating on one-bit registers was not power or area efficient.

2. **Gate Power Optimization:**
   There are a number of logic optimizations that the tools can perform to minimize dynamic power. The examples of gate level power optimization include cell sizing and buffer insertion. In cell sizing, the tool can selectively increase cell drive strength throughout the critical path to achieve timing and then reduce dynamic power to minimum.

   In buffer insertion, the tool can insert buffers rather than increasing the drive strength of the fate itself. If done in the right situation, this can result in low power.

3. **Multi Vdd:**
   Since dynamic power is proportion to the square of VDD, lowering VDD on selected blocks helps reduce power significantly. Unfortunately, lowering the voltage also increases the delay of the gates in the design.

   Mixing blocks at different VDD supplies adds some complexity to the design – not only do we need to add IO pins to supply the different power rails, but we need a more complex power grid and level shifters on signals running between blocks.

4. **Multi – Threshold Logic:**

As geometries have shrunk to 130nm, 90nm, and below, using libraries with multiple Vt has become a common way of reducing leakage current. The sub-threshold leakage depends exponentially on Vt delay has much weaker dependence on Vt.
May libraries today offer two or three versions of their cells : Low vt, Standard Vt, and High Vt. The implementation tools can take adventage of these libraries to optimize timing and power simultaneously.

Thus, each major compenent of the system is running at the lowest voltage consistent with meeting system timing. This approach can provide significant savings in power.

It is now quite common to use a "Dual Vt" flow during synthesis. The goal of this approach is to minimize the total number of fast, leaky low Vt transistors vy deploying them a primary library followed by an optimization step targeting one(or more) additional libraries with differing thresholds.

Usually there is a minimum performance which must be met before optimizing power. In practice this usually means synthesizing with the high performance , high leakage library first and then relaxing back any cells not on the critical path by swapping them for their lower performing, lower leakage equivalents.

What is the need Well tap cells?

An integrated circuit includes a set of standard cells, referred to as tap cells, each having a well tap and a substrate tap for coupling a well region and a substrate region to a power source and ground, respectively. The tap cells are disposed at intervals that do not exceed a maximum allowable distance as specified by a set of design rules associated with the integrated circuit. A method for designing the integrated circuit includes determining the positions at which the tap cells will be fixed and creating a design layout for the integrated circuit using a place and route tool that incorporates the positions.

Undesired bipolar transistors are inherently formed in integrated circuits that are manufactured using conventional integrated circuit manufacturing techniques. These undesired bipolar transistors may cause the integrated circuit to have robustness problems including, for example, a phenomenon known as latch-up. Latch-up occurs when the undesired bipolar transistors in combination with desired transistors create a positive feedback circuit in which the current flowing through the circuit increases to a magnitude that exceeds the current capacity of the integrated circuit. The excess current causes the integrated circuit to become defective and thus, unusable.

As is well known in the art, latch-up is prevented by placing well taps and substrate taps at positions in the integrated circuit that are located appropriate distances from one another. Each well tap is an electrically conductive lead that couples a well region of the integrated circuit to a power source and each substrate tap is an electrically conductive lead that couples a substrate region of the integrated circuit to ground. Coupling the well and substrate regions to power and ground, respectively, reduces the substrate resistance, thus causing the positive feedback to be removed.

In particular, the taps must be positioned so that the distance between any two well taps and any two substrate taps does not exceed a maximum allowable distance that is obtained using a set of design rules associated with the integrated circuit. As will be understood by one having ordinary skill in the art, the design rules typically specify that the distance from any point in either the substrate or well regions must not be located farther than a maximum distance from the nearest substrate tap or well tap, respectively. Thus, the maximum allowable distance specified in the design rules is not defined as being equal to the maximum allowable distance as that term is used for purposes of this discussion, i.e., the maximum allowable distance between tap cells. Rather, the maximum allowable distance between tap cells is equal to the maximum distance specified in the design rules multiplied by a factor of two. As is well known in the art of integrated circuit design, the design rules may also specify various other physical parameters necessary for the proper construction of the integrated circuit such as, for example, the minimum allowable distance between wires or conducting paths disposed in the integrated circuit and the minimum allowable width of such wires.

Currently, methods for positioning well and substrate taps in an integrated circuit are performed during the circuit design process which begins with defining the desired functions or computing tasks to be performed by the integrated circuit. These functions, once defined, are described in a hardware description language that is then translated by hand or using a computerized synthesis tool, into a netlist. As will be understood by one having ordinary skill in the art, a netlist defines a set of logic gates and the connectivity between the logic gates needed to implement the functions described in the hardware description language. For example, the netlist may include a list of the logic gates, wires and input/output ports needed to implement the integrated circuit. After the netlist has been created, the integrated circuit designer provides the netlist and a floorplan as data input to a computerized "place and route" tool that creates a layout associated with netlist. The floorplan specifies various physical constraints associated with the integrated circuit design including, for example, the location of the power grid, the locations of input and output ports and the locations where the various wires or conducting paths will be located. The "place and route" tool uses the netlist and floorplan to determine the physical design of the integrated circuit from which the integrated circuit will ultimately be fabricated. Each logic gate is represented in the netlist by a logic cell and thus, the resulting layout is comprised of a set of logic cells that, when formed in the integrated circuit, will implement the logic dictated by the corresponding logic gates. To reduce the time required to perform the design process, cell libraries have been created wherein standard cell designs are available. Of course, there are applications that may require one or more specialized cells in which case the designer will either create a custom cell for the layout or alter a library cell in a manner required by the desired design. Once complete, the resulting layout is used to manufacture the desired integrated circuit.

Two well-known methods for positioning taps in an integrated circuit occur at different stages of the circuit design process. In a first tap positioning method, the positions of the taps were determined by virtue of using a standard cell library for cell selection. More particularly, each standard cell, also referred to as a library cell, was designed to include at least one well tap and one substrate tap. Thus, when the "place and route" tool used the netlist to extract standard library cells to use for the layout, the tap locations were defined in the resulting layout by default. This technique of placing at least one of each type of tap per library cell was effective for earlier-generation integrated circuits because earlier integrated circuits and the cells associated with these integrated circuits were physically larger than the integrated circuits of today. More particularly, at least one well tap and one substrate tap had to be placed in each library cell to satisfy the design rule related to the maximum allowable distance between taps. In some instances, the dimensions of a cell must be increased in order to fit the taps thereby causing an undesired increase in the size of the integrated circuit.

In a second tap positioning method, a layout was prepared using cells that are not designed to include taps and then proper locations for the taps were determined by engineers. Although this second method yielded the optimal result for layout density, it was extremely costly in terms of engineering hours. Using this method, an engineer determined appropriate locations for a set of taps and then used a design rule checker to determine whether the taps were indeed appropriately positioned, i.e., are in compliance with the design rules. As is well known in the art, a design rule checker is a computer-aided design tool that determines whether the resulting layout complies with the design rules associated with the integrated circuit. These steps were then repeated in an iterative fashion until the resulting measurements indicated that the tap locations had been properly positioned. Unfortunately, the iterative and manual nature of the process caused it to be a time consuming and thus costly task.

However, due to continuing developments in the design and manufacture of integrated circuits, cell dimensions are shrinking. As a result, a well tap and a substrate tap are no longer required in each cell and the engineering hours spent positioning taps in library cells are needlessly spent. Moreover, the positioning of unnecessary taps needlessly consumes the area of the cell that may otherwise be available for other cell circuitry and further consumes the area available for wires in the cell.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1A is a plan view of an integrated circuit having tap cells positioned in accordance with the present invention;

FIG. 1B is a plan view of the integrated circuit of FIG. 1A with portions of a set of layers of the integrated circuit removed to show features associated with the layers;

FIG. 2 is a plan view of a tap cell in accordance with the present invention; and,

FIG. 3 is a flow chart that illustrates a method for positioning tap cells in an integrated circuit in accordance with the present invention.

SUMMARY OF THE INVENTION

The present invention is directed to an integrated circuit having standard cells that contain well and substrate taps and that are located at equally spaced intervals that do not exceed a maximum allowable distance. The tap cells each include a well tap and a substrate tap and are positioned at generally equally spaced intervals to prevent the formation of positive feedback circuits which may cause latch-up. A method for designing an integrated circuit having tap cells located at generally equally spaced intervals includes determining a set of positions at which the tap cells are to be located and then creating a layout for the integrated circuit that incorporates the positions.

DETAILED DESCRIPTION

Referring now to FIG. 1A, an integrated circuit having a set of tap cells 10 positioned in accordance with the present invention is indicated generally at 12. The integrated circuit 12 includes a placeable area 14 in which logic cells 11 and wiring (not shown) may be placed and that is partitioned into a set of rows 16 of equal height. The height of each row 16 is also equal to the height of a standard library cell, and thus, each row 16 is dimensioned to support a plurality of standard library cells 11. The width of each standard library cell 11 may vary, and as a result, the number of standard library cells disposed in any given row 16 will vary depending on the widths of the cells disposed therein.

As will be appreciated by one having ordinary skill in the art, integrated circuits 12 are constructed by forming layers comprised of a semiconductor material, such as silicon, an insulating material and a metal. For illustrative purposes, the detail associated with various layers of the integrated circuit 12 are represented in the sectionalized view of FIG. 1B. Specifically, the placeable area 14, which extends through all layers of the integrated circuit, is shown in a section 13 of FIG. 1B. In addition, the rows 16 of the integrated circuit 12 are associated with all layers of the integrated circuit and are shown in a section 15 of FIG. 1B. In a section 17, layers of the integrated circuit 12 having a set of well regions 18 and substrate regions 20 are shown. In particular, each row 16 of the integrated circuit 12 includes a well region 18 and a substrate region 20. The well and substrate regions 18 and 20 are formed by a process, referred to as doping, which is performed by adding impurities to the various layers of silicon. Adding impurities causes the electrical qualities of the silicon to change such that the silicon becomes either more or less electrically conductive depending on the type of material added. More specifically, the substrate region 20 is formed using a layer of silicon that spans the entire placeable area 14 and that may be doped with, for example, a P-type dopant and the well regions 18 are formed by adding a layer of N-type dopant on top of select areas of the layer that forms the substrate region 20. Thus, the well regions 18 are defined as the select areas that are doped with the N-type dopant and the substrate regions 20 are, by default, the portions of the placeable area 14 that are not occupied by well regions 18. The well regions 18 and substrate regions 20 are positioned so that each row 16, except the first and last rows, shares a well region 18 with an adjacent row 16 and each row 16 shares a substrate region 20 with an adjacent row 16. A section 19, shows a layer of the integrated circuit 12 at which at set of power rails 22 and a set of ground rails 24 are disposed. The power and ground rails 22, 24 span the width of the placeable area 14 and are positioned so that each power rail 22, except for the power rail 22 disposed in the top row 16, straddles two adjacent rows 16 and each ground rail 24, except the ground rail 24 disposed in the bottom row 16, straddles two adjacent rows 16. The ground rail 24 is connected to ground potential and the power rail 22 is coupled to a power source. In a section 21, the position of the tap cells 10 relative to the rows 16, the well and substrate regions 18, 20 and the power and ground rails 22, 24 is shown. Of course, integrated circuits may be configured in any number of ways and the configuration of the integrated circuit 12 of FIGS. 1A and 1B is merely intended to be representative of a typical integrated circuit design. For example, the power and ground rails 22 and 24, which are shown as being disposed in the same layer of the integrated circuit 12, may be disposed in any layer of the integrated circuit 12 and both rails 22 and 24 need not be disposed in the same layer of the integrated circuit 12. Likewise, the features associated with the other layers of the integrated circuit 12 may be reconfigured in any of a number of possible ways.

Referring now to FIGS. 1 and 2, tap cells 10 are preferably, though not necessarily, positioned at equally spaced intervals 26 in alternating rows 16 of the integrated circuit 12. A well tap 28 is disposed near a first end 30 of each tap cell 10 and a substrate tap 32 is disposed near a second end 34 of each tap cell 10. Each well and substrate tap 28, 32 acts as a lead by which the well and substrate regions 18, 20 are coupled to the power and ground rails 22, 24 respectively. More particularly, each well tap 28 extends from a layer of the circuit 12 at which the well region 18 is disposed to a layer at which the power rail 22 is disposed. In addition, each well tap 28 is doped with a dopant that causes the well tap 28 to be electrically conductive so that the well region 18 is thereby electrically coupled to the power rail 22. Due to the nature of the doping process, the doping pattern is more diffused in an outer region 36 of the well tap 28 and becomes more concentrated and thus more conductive in an inner region 38 of the well tap 28. Similarly, each substrate tap 32 extends from a layer of the circuit 12 at which the substrate region 20 is disposed to a layer at which the ground rail 24 is disposed. In addition, each substrate tap 32 is doped with a dopant that causes the substrate tap 32 to be electrically conductive so that the substrate region 20 is thereby electrically coupled to the ground rail 24. Again, the doping process causes the doping pattern of the substrate tap 32 to be more diffused in an outer region 40 of the substrate tap 32 and more concentrated and thus more conductive in an inner region 42 of the substrate tap 32.

The interval 26, or distance between tap cells 10 does not exceed the maximum allowable distance obtained using the design rules (not shown) that are associated with the integrated circuit 12. Specifically, as described above, the design rules specify the maximum distance from any point in the substrate or well region to the nearest substrate or well tap, respectively. Further, the distance specified in the design rules is multiplied by a factor of two to obtain the maximum allowable distance between tap cells. Thus, in order to satisfy the design rules related to tap positioning and thereby prevent latch-up, every substrate region 20 is populated with a set of substrate taps 32 that are separated by an interval 26 that does not exceed the maximum allowable distance and every well region 18 is populated with a set of well taps 28 that are separated by an interval 26 that does not exceed the maximum allowable distance.

Although in many instances, the maximum allowable distance between substrate taps 32 will be equal to the maximum allowable distance between well taps 28, there may be instances in which the distances are not equal. When the distances are not equal, the length of the interval 26 must be designed so that it does not exceed the smaller of the two maximum allowable distances. In addition, because each well and substrate region 18, 20 is disposed in two adjacent rows 16, the tap cells 10 in which the well and substrate taps 28, 32 are disposed need only be located in every other row 16 to satisfy the design rules. Of course, the dimensions of the tap cells 10 will affect the length of the interval 26 separating the tap cells 10. For example, if the width 44 of the tap cell 10 exceeds the width 46 of the well or substrate tap 28, 32 disposed therein, then the length of the interval 26 must be reduced to take into account the distance 48 between the edge of the tap 28, 32 and the edge of the tap cell 10. Specifically, the length of the interval 26 plus the length of the distance 48 between the edge of the tap 28, 32 and the edge of the tap cell 10 must not exceed the maximum allowable distance. As will be appreciated by one having ordinary skill in the art, the exact positions at which the tap cells 10 are located may vary, provided that the intervals 26 do not exceed the maximum allowable distance. In addition, although the intervals 26 are preferably spaced at equal distances, the distance between the intervals 26 may vary provided, of course, that the largest distance between any two tap cells 10 does not exceed the maximum allowable distance associated with the integrated circuit. Further, FIG. 1 is only intended to provide sufficient detail to indicate the positions at which the tap cells 10 are located relative to each other and relative to the well and substrate regions 18, 20 and the power and ground rails 22, 24. As a result, only two of the plurality of logic cells 11 that are used to implement the desired logic of the integrated circuit 12 and that would typically be disposed in the portions of the rows 16 that are not occupied by the tap cells 10, are shown in FIG. 1.

Referring now to FIG. 3, a flowchart illustrating a method of designing an integrated circuit that incorporates a set of steps for determining the positions at which a set of tap cells 10 will be located begins at a step 50 where the desired functionality of the integrated circuit 12 is defined. For example, if the integrated circuit 12 is going to be used as a microprocessor, then the desired functionality of the microprocessor is defined. If, instead the integrated circuit 12 is going to be used as an application specific integrated circuit (ASIC) then the desired functionality of the ASIC is defined. Next, at a step 60, the desired functionality of the integrated circuit 12 is described using a hardware description language such as, for example, Verilog or VHDL.

Continuing at a step 70, the hardware description language is converted into a netlist using any of a number of synthesis tools such as, for example, Design Compiler by Synopsys.RTM. or Build Gates by Cadence.RTM.. As described above, the netlist is a listing of the logic gates, the connections between the gates, wiring and input/output ports required to implement the functionality of the integrated circuit as described in the hardware description language. In particular, each of the logic gates required to implement the integrated circuit is represented in the netlist by a logic cell which defines the circuitry required to implement the logic gate. As described above, the logic cells listed in the netlist may be standard library cells or may instead be customized cells having application-specific functions. Also, at the step 70, the netlist version of the integrated circuit design may be used to perform computer simulations to test the integrated circuit design for defects.

Next, at a step 80, the dimensions of a set of tap cells 10 are defined and a set of positions at which the tap cells will be located in the placeable area 14 of the integrated circuit 12 are determined. More particularly, a floorplan associated with the integrated circuit is used to provide the physical dimensions of the basic structure of the integrated circuit. As will be understood by one having ordinary skill in the art, the floorplan defines the physical constraints of the integrated circuit, including, for example, the location of a power grid, the location of input/output ports, the dimensions of the integrated circuit block and the areas of the integrated circuit in which wires associated with the power grid and other pre-existing circuitry are disposed. Specifically, the designer obtains the orientation, number and dimensions of the rows 16 of the integrated circuit 12 in which the logic cells 11 and tap cells 10 will be located. In addition, the designer obtains the maximum allowable distance between tap cells 10 from the design rules associated with the integrated circuit 12 and defines the dimensions of the tap cells 10. Using the row information, the tap cell dimensions and the maximum allowable distance, the designer calculates the number of tap cells 10 that will be placed in each row 16 and the length of the interval 26 that will be interposed between each tap cell 10 located in a row 16 such that the positions of the tap cells 10 are located at equally spaced intervals 26 and such that the interval 26 between the tap cells 10 located in a given row 16 does not exceed the maximum allowable distance. Of course, as described above, the designer must take into account the distance 48 between the edge of the tap 28 or 32 and the edge of the tap cell 10 when determining the length of the interval 26.

Once the dimensions and positions of the tap cells 10 have been defined, at a step 90, the floorplan associated with the integrated circuit design is modified to include the dimensions and positions of the tap cells 10 at a step 90. Typically, the floorplan exists in a computer file format written in a physical netlist syntax such as Design Exchange Format ("DEF") by Cadence.RTM. that is easily modifiable to include the dimensions and positions of the tap cells 10. After the netlist version of the integrated circuit 12 has been completely tested and debugged, at the step 70, and after the floorplan has been modified to include the dimensions/positions of the tap cells, at the step 90, the netlist and the floorplan are provided as data input to a computer-automated design tool referred to as a "place and route" tool such as, Silicon Ensemble by Cadence.RTM., at a step 100. The "place and route" tool then uses the netlist and the floorplan to design a layout for the integrated circuit 12. As will be understood by one having ordinary skill in the art, the layout is a representation of the integrated circuit 12 that includes the physical dimensions and configuration of the integrated circuit components and serves as a blueprint from which the integrated circuit 12 may be manufactured. For example, the "place and route" tool determines where the logic cells listed in the netlist are to be located in the placeable area 14 of the integrated circuit 12 and further determines the routing of any wires needed to support the logic cells. In addition, the "place and route" tool automatically incorporates the information provided in the floorplan such that the resulting layout incorporates the positions at which the tap cells 10 are located.

Finally, at a step 110, the layout may be used to manufacture the desired integrated circuit. As will be appreciated by one having ordinary skill in the art, the step 70 of determining positions where the tap cells 10 will be located may be done before, during or after the step 60 of converting the hardware description language into the netlist. In addition, the design method of FIG. 3 is not limited to design processes that use a "place and route" tool. In fact, any layout tool may be used to create the integrated circuit layout. In addition, the layout may even be created using a process that includes steps that are performed manually by a designer who, for example, incorporates the positions of the tap cells determined at the step 80.

From the foregoing description, it should be understood that an improved integrated circuit with tap cells fixed at equally spaced intervals and a method for positioning the tap cells in the integrated circuit have been shown and described, both of which have many desirable attributes and advantages. The integrated circuit having tap cells fixed at equally spaced intervals is less costly to design and the method of designing integrated circuits that include tap cells located at equally space intervals will eliminate the need to place well and substrate taps into each and every library cell, thereby potentially reducing library cell dimensions. Moreover, the method of determining the positions at which the tap cells shall be located is compatible with existing design processes, including design processes that use a computerized layout tool such as a "place and route" tool.

While various embodiments of the present invention have been shown and described, it should be understood that other modifications, substitutions and alternatives are apparent to one of ordinary skill in the art. Such modifications, substitutions and alternatives can be made without departing from the spirit and scope of the invention, which should be determined from the appended claims.

Various features of the invention are set forth in the appended claims.

Timing Analysis Conditions

Input Delays
Excluding Clocks

Output Delays

Drive Characteristics at Input Ports
Setting the Port Driving Cell
Setting the Port Drive Resistance
Setting a Fixed Port Transition Time
Displaying Drive Information
Removing Drive Information From Ports

Port Capacitance

Wire Load Models
Setting Wire Load Models Manually
Automatic Wire Load Model Selection
Setting the Wire Load Mode
Reporting Wire Load Models

Operating Conditions
Interconnect Model Types
Setting Operating Conditions
Creating Operating Conditions
Operating Condition Information

Slew Propagation

Design Rules
Maximum Transition Time
Handling Slew in Multiple Threshold and Derate Environment
Conversion of single-float slews between thresholds and derates:
Maximum Transition Constraint Storage
Evaluation of Max Transition Constraint
Example 1
Example 2 - Scaling the Max Transition with Different Library Slew Threshold
Example 3 - Scaling the Max Transition with Slew Derate Factor Specified in Library
Minimum and Maximum Net Capacitance
Maximum Fanout Load
Fanout Load Values for Output Ports

A timing analysis by Timing Engine depends on the conditions that you specify. These can include such conditions as input delays, output delays, port capacitance, wire load models, and operating conditions.

The specification and usage of these conditions are described in the following sections:

- Input Delays
- Output Delays
- Drive Characteristics at Input Ports
- Port Capacitance
- Wire Load Models
- Operating Conditions
- Slew Propagation
- Design Rules

Input Delays
Specifies the amount of delay from an edge of the CLK clock to the arrival of data on input pins. Shown in figure 7.0a

An input delay is the specification of an arrival time at an input port relative to a clock edge. The **set_input_delay** command specifies the timing of external paths leading to an input port..

Specifies the drive characteristics of the external cell presumed to be driving all the input ports of the design.

Driving-cell delay

The ports have a cell delay that is the load-dependent value of the external driving-cell delay. To prevent this delay from being counted twice, estimate the load-dependent delay of the driving cell, then subtract that amount from the input delays on the port. Shown in figure 7.0b. This can be Applied using set_drive_resistance or set_driving_cell commands to the port.

The input delay should equal the path length from the clock pin of the source flip-flop to the output pin of the driving cell, minus the load-dependent portion of the driving cell's delay. For example, see the external path for the L1 clock port to port IN1 in Figure 7-1.

Figure 7.0a Input and Output Delays

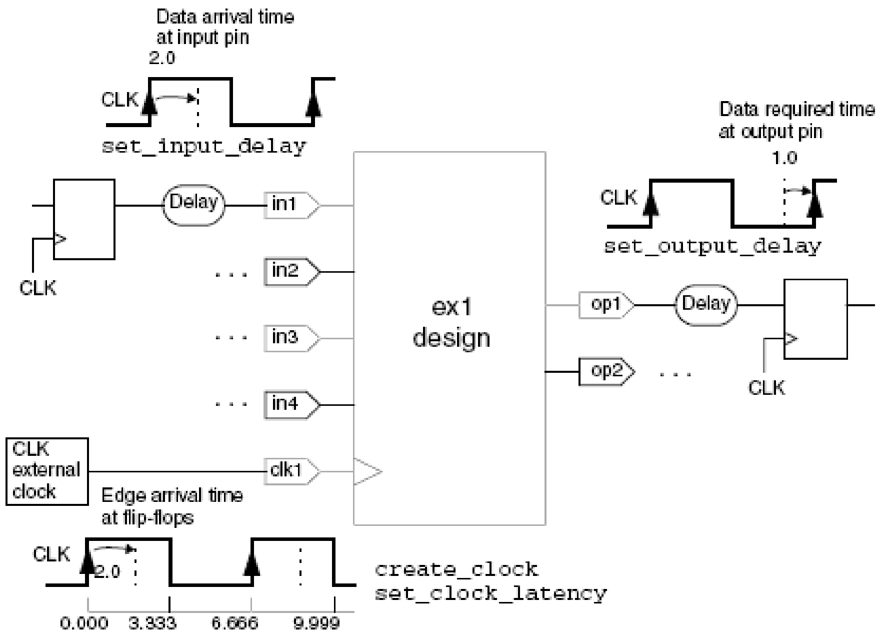


Figure 7.0b Driving Cell

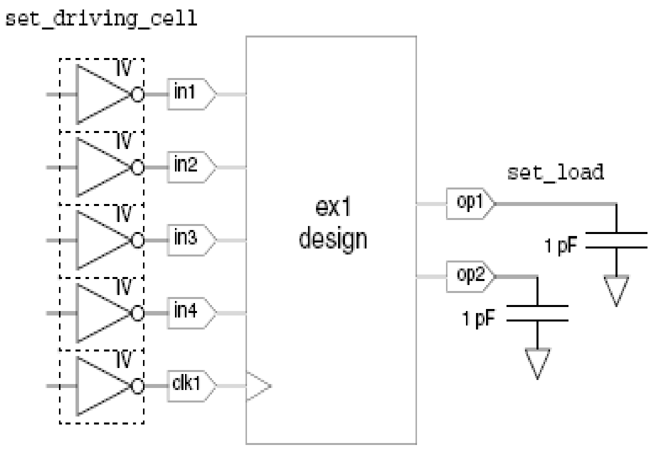
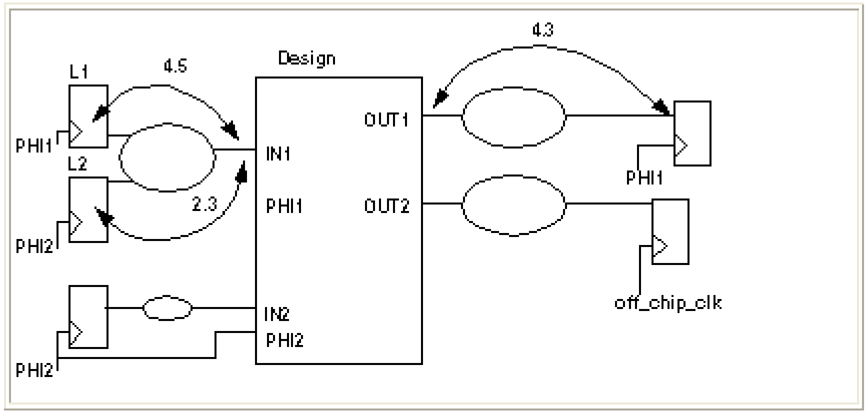


Figure 7-1 Two-Phase Clocking Example for Setting Port Delays



When you use the set_input_delay command, you can specify whether the delay value includes the network latency or source latency. For details, see the set_input_delay man page.

Example 1

If the delay from L1 clock port to IN1 (minus the load-dependent delay of the driving cell) is 4.5, this set_input_delay command applies:

set_input_delay 4.5 -clock PHI1 {IN1}

Example 2

If paths from multiple clocks or edges reach the same port, specify each one using the -add_delay option. If you omit the -add_delay option, existing data is removed. For example:

set_input_delay 2.3 -clock PHI2 -add_delay {IN1}

If the source of the delay is a level-sensitive latch, use the -level_sensitive option. This allows Timing Engine to determine the correct single-cycle timing constraint for paths from this port. Use the -clock_fall option to denote a negative level-sensitive latch; otherwise the -level_sensitive option implies a positive level-sensitive latch.

To see input delays on ports, use report_port -input_delay.

To remove input delay information from ports or pins in the current design set using the set_input_delay command, use remove_input_delay. The default is to remove all input delay information in the port_pin_list.

Excluding Clocks

Timing Engine considers the input delay on clock source ports or pins as source latency if the clock is propagated.

The input delay can be relative to no clock, or to the clock of that source. The source latency value is added to the clock network delay to determine total latency.

A common **mistake** is to use the following command, which sets delay on all the inputs, including the clock input:

```
 set_input_delay 2 -clock CLK [all_inputs]
```

Instead, use this command:

```
set_input_delay 2 -clock CLK \
    [remove_from_collection [all_inputs] [get_port CLK]]
```

Use the set_clock_latency command with the -source option to define the actual source latency, if any.

Output Delays

Specifies the amount of delay from each output to the external device, that captures the output data with respect to the CLK. Shown in figure 7.0a

An output delay represents an external timing path from an output port to a register. The maximum output delay value should be equal to the length of the longest path to the register data pin, plus the setup time of the register. The minimum output delay value should be equal to the length of the shortest path to the register data pin, minus the hold time.

The set_output_delay command specifies output delays.

To show output delays associated with ports, use report_port -output_delay.

To remove output delay from output ports or pins set through set_output_delay, use remove_output_delay. By default, all output delays on each object in the port or pin list are removed. To restrict the removed output delay values, use -clock, -clock_fall, -min, -max, -rise, or -fall.

Example

To set an output delay of 4.3 relative to the rising edge of clock PHI1 on port OUT1 (see Figure 7-1):

```
 set_output_delay 4.3 -clock PHI1 {OUT1}
```

Drive Characteristics at Input Ports

You need to define the drive capability of the external cell driving each input port. Timing Engine uses this information to calculate the load-dependent cell delay for the port and to produce an accurate transition time for calculating cell delays and transition times for the following logic stages.

The set_driving_cell command can specify a library cell arc for the driving cell so that timing calculations are accurate even if the capacitance changes. This command causes the port to have the transition time calculated as if the given library cell were driving the port.

For less precise calculations, you can use the set_drive or set_input_transition command. The most recent drive command has precedence. If you issue the set_drive command on a port delay and then use the set_driving_cell command on the same port, information from the set_drive command is removed.

Setting the Port Driving Cell

The set_driving_cell command directs Timing Engine to calculate delays as though the port were an instance of a specified library cell. The port delay is calculated as a cell delay that consists of only the load-dependent portion of the port.

The transition time for the port is also calculated as though an instance of that library cell were driving the net. The delay calculated for a port with information from the driving_cell command takes advantage of the actual delay model for that library cell, whether it is nonlinear or linear. The input delay specified for a port with a driving cell or drive resistance should not include the load-dependent delay of the port.

To display port transition or drive capability information, use the report_port command with the -drive option.

With the set_driving_cell command you can specify the input rise and fall transition times for the input of a driving cell using the -input_transition_rise or the -input_transition_fall option. If no input transition is specified, the default is 0.

For more information, see the set_driving_cell man page.

Setting the Port Drive Resistance

The set_drive command models the external driver as a linear resistance, as in the linear generic_cmos delay model. The port has a transition time equal to Rdriver * Ctotal. Timing Engine uses this transition time in calculating the delays of subsequent logic stages. Timing Engine performs detailed delay calculation based on a 0 input transition to the driving cell, the total net cap, and the drive resistance specified using this command.The cell delay of the port is also equal to half of the Rdriver * Ctotal.

The set_drive command is useful when you cannot specify a library cell (for example, when the driver is a custom block not modeled as a Synopsys library cell).

For more information, see the set_drive man page.

Setting a Fixed Port Transition Time

The set_input_transition command defines a fixed transition time for input ports. The port has zero cell delay. Timing Engine uses the specified transition time only in calculating the delays of logic driven by the port.

A fixed transition time setting is useful for

- Comparing the timing of a design to another tool that supports only input transition time.
- Defining transition for ports at the top level of a chip, where a large external driver and a large external capacitance exist. In this case, the transition time is relatively independent of capacitance in the current design.

For more information, see the set_input_transition man page.

Port Capacitance

The set_load command sets load capacitance on ports. To accurately time a design, you need to describe the external load capacitance of nets connected to top-level ports, including pin capacitance and wire capacitance.

For more information, see the set_load man page.

Example 1

To specify the external pin capacitance of ports, enter

```
set_load -pin_load 3.5 {IN1 OUT1 OUT2}
```

You also need to account for wire capacitance outside the port. For prelayout, specify the external wire load model and the number of external fanout points. This process is described in "Setting Wire Load Models Manually".

Example 2

For post-layout, specify the external annotated wire capacitance as wire capacitance on the port. For example, enter

```
set_load -wire_load 5.0 {OUT3}
```

To remove port capacitance values, use the remove_capacitance command.

Wire Load Models

A wire load model attempts to predict the capacitance and resistance of nets in the absence of placement and routing information.

The estimated net capacitance and resistance are used for delay calculation. Technology library vendors supply statistical wire load models to support estimation of wire loads based on the number of fanout pins on a net.

You can set wire load models manually or automatically.

Setting Wire Load Models Manually

The set_wire_load_model command manually sets a named wire load model on a design, instance, list of cells, or list of ports.

For example, consider this design hierarchy:

```
 TOP
   MID (instance u1)
     BOTTOM (instance u5)
   MID (instance u2)
     BOTTOM (instance u5)
```

To set a model called 10x10 on instances of BOTTOM, a model called 20x20 on instances of MID, and a model called 30x30 on nets at the top level, use these commands:

```
set_wire_load_mode enclosed
set_wire_load_model -name 10x10 all_instances BOTTOM]
set_wire_load_model -name 20x20 all_instances MID]
set_wire_load_model -name 30x30
```

To capture the external part of a net that is connected to a port, you can set an external wire load model and a number of fanout points. For example, to do this for port Z of the current design:

```
set_wire_load_model -name 70x70 [get_ports Z]
set_port_fanout_number 3 Z
```

To calculate delays, Timing Engine assumes that port Z has a fanout of 3 and uses the 70x70 wire load model.

To see wire load model settings for the current design or instance, use the report_wire_load command. To see wire load information for ports, use the report_port command with -wire_load option. To remove user-specified wire load model information, use the remove_wire_load_model command.

Automatic Wire Load Model Selection

Timing Engine can set wire loads automatically when you update the timing information for your design. If you do not specify a wire load model for a design or block, Timing Engine automatically selects the models based on the wire load selection group, if specified.

If you do not apply a wire load model or selection group but the library defines a default_wire_load model, Timing Engine applies the library-defined model to the design. Otherwise, the values for wire resistance, capacitance, length, and area are all 0.

Automatic wire load selection is controlled by selection groups, which map the block sizes of the cells to wire load models. If you specify the set_wire_load_selection_group command on the top design, or if the main technology library defines a default_wire_load_selection_group, Timing Engine automatically enables wire load selection.

For more information, see the set_wire_load_selection_group man page.

When wire load selection is enabled, the wire load is automatically selected for hierarchical blocks larger than the minimum cell area, based on the cell area of the block.

Example

To set the minimum block size for automatic wire load selection, enter:

set_wire_load_min_block_size ***size***

In this command, *size* is the minimum block size for automatic wire load selection, in library cell area units. The specified size must be greater than or equal to 0.

The auto_wire_load_selection variable specifies automatic wire load selection. The default setting is true, enabling automatic wire load selection if a selection group is associated with the design. To disable automatic wire load selection, enter

set auto_wire_load_selection false

To remove the wire load selection group setting, use remove_wire_load_selection_group.

Setting the Wire Load Mode

The current wire load mode setting, which can be set with the set_wire_load_mode command, determines the wire load models used at different levels of the design hierarchy. There are three possible mode settings: top, enclosed, or segmented.

If the mode for the top-level design is **top**, the top-level wire load model is used to compute wire capacitance for all nets within the design, at all levels of hierarchy. If the mode for the top-level design is either enclosed or segmented, wire load models on hierarchical cells are used to calculate wire capacitance, resistance, and area for nets inside these blocks.

If the **enclosed** mode is set, Timing Engine determines net values using the wire load model of the hierarchical cell that fully encloses the net. If the **segmented** mode is set, Timing Engine separately determines net values for segments of the net in each level of hierarchy, and then obtains the total net value from the sum of all segments of the net.

If you do not specify a mode, the default_wire_load_mode setting of the main technology library is used. The enclosed mode is often the most accurate. However, your ASIC vendor might recommend a specific mode.

For more information, see the set_wire_load_mode man page.

Example

To set the wire load mode to enclosed on the current design, enter

set_wire_load_mode enclosed

Example

The design in Figure 7-2 has the following wire load models set:

set_wire_load_model -name Big (the current design)
set_wire_load_model -name Medium (instances U1 and U2)
set_wire_load_model -name Small (instance U2/U1)

Figure 7-2 Example Design for Wire Load Mode Settings

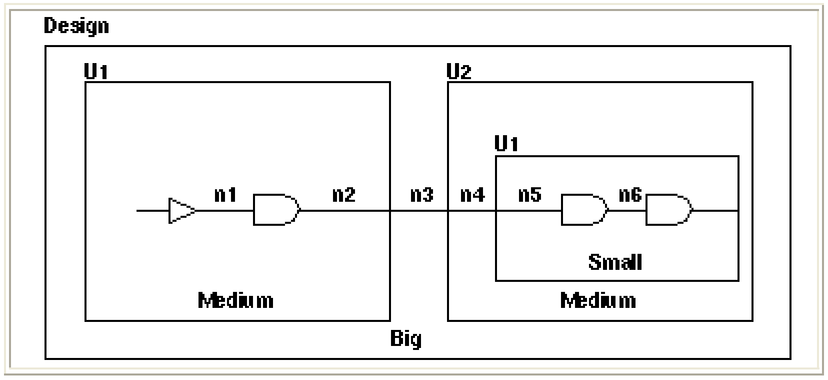


Table 7-2 lists the resulting wire load modes and models that apply to the nets in Figure 7-2.

Table 7-2 Wire Load Models

Wire load setting	**Wire load model**	**Applies to these nets**
top	Big	All nets
enclosed	Big	n3, U1/n2, U2/n4, U2/U1/n5
	Medium	U1/n1
	Small	U2/U1/n6
segmented	Big	n3
	Medium	U1/n1, U1/n2, U2/n4
	Small	U2/U1/n5, U2/U1/n6

Operating Conditions

Integrated circuits exhibit different performance characteristics for different operating conditions: fabrication process variations, power supply voltage, and temperature. The technology library defines nominal values for these parameters and specifies delay information under those conditions.

An set of operating conditions contains the following values:

Process derating factor

> This value is related to the scaling of device parameters resulting from variations in the fabrication process. A process number less than the nominal value usually results in smaller delays.

Ambient temperature

> The chip temperature affects device delays. The temperature of the chip depends on several factors, including ambient air temperature, power consumption, package type, and cooling method.

Supply voltage

> A higher supply voltage usually results in smaller delays.

Interconnect model type

> This value defines an RC tree topology that Timing Engine uses to estimate net capacitance and resistance during prelayout analysis.

The delay information for each timing arc is specified at nominal process, temperature, and voltage conditions. If your operating conditions are different from this, Timing Engine applies scaling factors to account for the variations in these conditions. Many libraries use linear scaling for process, temperature, and voltage.

If the technology library contains scaled cell information, you can include the exact delay tables or coefficients for specific operating conditions. This method can be very accurate for library cells that do not scale linearly. For more information, see the Library Compiler and Design Compiler reference manuals.

You can use a single set of operating conditions to do analysis (for setup and hold) or you can specify minimum and maximum conditions. If you do not set operating conditions on your design, Timing Engine uses the default set of operating conditions if the main library contains them, or the nominal values of the main library.

Interconnect Model Types

Timing Engine uses interconnect model information when it calculates net delays for prelayout designs, when annotated net delays and parasitic information are not available. Two nets with the same total resistance and capacitance, but different RC tree topologies, can have different pin-to-pin delays.

This section provides background information about interconnect model types. You cannot modify the types in Timing Engine. For more information, see the Library Compiler reference manuals.

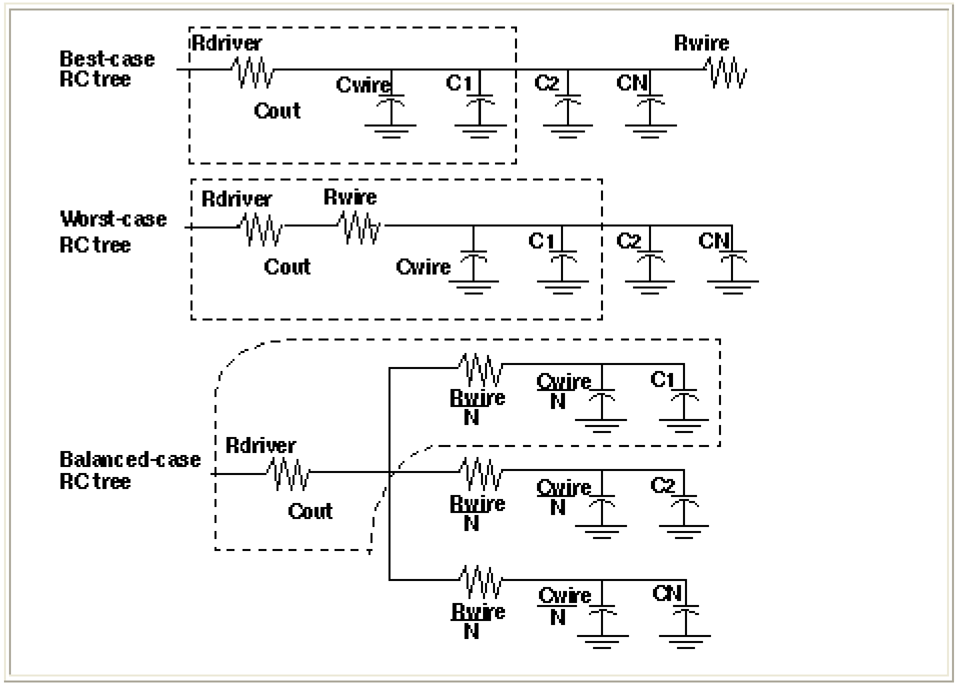
The interconnect model is defined by the tree_type specification in each technology library's set of operating conditions. A tree_type specification indicates the type of wire resistance and capacitance topology: best_case_tree, worst_case_tree, or balanced_tree. For example:

```
operating_conditions(BEST) {
    process     : 1.1;
    temperature : 11.0;
    voltage     : 4.6;
    tree_type   : ”best_case_tree”;
}
operating_conditions(TYPICAL) {
    process     : 1.3;
    temperature : 31.0;
    voltage     : 4.6;
    tree_type   : ”balanced_tree”;
}
operating_conditions(WORST) {
   process      : 1.7;
   temperature  : 55.0;
   voltage      : 4.2;
   tree_type    : ”worst_case_tree”;
}
```

If the technology library does not define the tree type, Timing Engine uses the balanced_tree model.

Figure 7-3 shows the tree type model networks.

Figure 7-3 RC Interconnect Topologies for Fanout of N



Setting Operating Conditions

The set_operating_conditions command sets process, temperature, and voltage conditions for timing analysis.

The operating conditions you specify must be defined in a specified library or a library in the link path. You can create custom operating conditions for a library with the create_operating_conditions command. Use the report_lib command to get a list of the available operating conditions in a technology library before you use the set_operating_conditions command.

For more information, see the set_operating_conditions man page.

Example 1

To set WCCOM from the tech_lib library as a single operating condition, enter

```
set_operating_conditions WCCOM -library tech_lib
```

Example 2

To set WCCOM as the maximum condition and BCCOM as the minimum condition, enter

```
set_operating_conditions -analysis_type bc_wc \
        -min BCCOM -max WCCOM
```

Because you do not specify a library, Timing Engine searches all libraries in the link path. After you set the operating conditions, you can report or remove operating conditions.

Creating Operating Conditions

A technology library contains a fixed set of operating conditions. You can use the create_operating_conditions command to create new operating conditions in a library you specify and use these custom operating conditions to analyze your design during the current session.

You cannot write these operating conditions to a library .db file.

To see the operating conditions defined for a library, use the report_lib command.

To set operating conditions on the current design, use the set_operating_conditions command.

Example

To create a new operating condition called WC_CUSTOM in the library tech_lib, enter

```
create_operating_conditions -name WC_CUSTOM \
          -library tech_lib -process 1.2 \
          -temperature 30.0 -voltage 2.8 \
          -tree_type worst_case_tree
```

Operating Condition Information

These commands report, remove, or reset operating condition information.

To do this	Use this
List the operating condition settings for the design.	report_design
Remove operating conditions from the current design.	remove_operating_conditions
Reset operating conditions to the default and remove all user-specified data, such as clocks, input and output delays.	reset_design

Slew Propagation

The timing_slew_propagation_mode variable allows you specify how Timing Engine propagates slew through the circuit. You can set this variable to worst_slew (the default), which gives a possibly pessimistic but safe analysis; or worst_arrival, which gives a more accurate but possibly optimistic analysis. The method of slew propagation affects the delay of a cell arc or net.

In the worst_slew (default) mode, at a pin where multiple timing arcs meet (or merge), Timing Engine computes the slew per driving arc at the pin, then selects the worst slew value at the pin to propagate along. Note that the slew selected might not be from the input that contributes to the worst path, so the calculated delay from the merge pin could be pessimistic.

In the worst_arrival mode, Timing Engine selects and propagates the slew of the input with the worst arrival time, selecting from multiple inputs propagated from the same clock domain. Timing Engine selects and propagates different slews from different clock domains.

Each slew propagation method, worst-arrival or worst-slew, has its own advantages. The worst-arrival method generally provides more accurate analysis for the worst path for each timing endpoint. However, it can potentially produce optimistic results that miss timing violations. The worst-slew method performs a pessimistic analysis that can be relied upon for timing signoff.

You can use both methods to benefit from their respective advantages. If the two results are nearly the same, it suggests that both reports are reasonably accurate. However, if the worst-arrival method shows significantly improved slack, more analysis is necessary to make sure that there is no optimism in the results.

Minimum slew propagation is similar to maximum slew propagation. In other words, Timing Engine selects minimum slew based on the input with the best delay at the merge point. With the worst-arrival method, if the operating condition was originally set to single (set_operating_conditions -analysis_type single), Timing Engine automatically switches the operating condition to on_chip_variation to ensure that both minimum and maximum slews are propagated, and also issues a message to indicate the new analysis mode.

Associated with the worst-arrival mode are two options to the set_input_transistion and set_driving_cell commands, -clock *clock_name* and -clock_fall. To set the input transition time and delay relative to a clock, use the -clock option. This means the transition applies only to external paths driven by the clock. By default, the transition time and delay are set relative to the rising edge of the clock. To use the falling edge instead, use the -clock_fall option.

Using the -clock or -clock_fall option is meaningful only in the worst-arrival slew propagation mode. In the worst-slew (default) mode, Timing Engine takes the worst value over all clocks specified with the command.

Design Rules

Timing Engine checks for violations of electrical rules that are defined in the library or by Timing Engine commands. These rules include

- Maximum limit for transition time
- Maximum and minimum limits for capacitance
- Maximum limit for fanout

To get a report on design rule violations in a design, use the report_constraint command.

Maximum Transition Time

The maximum transition time in Timing Engine is treated in a similar fashion as the slew. First we will discuss slew in the context of thresholds and derate and extend the discussion to maximum transition time.

A spice waveform can be represented as a floating point number in Liberty tables, which we will illustrate with an example Figure 7-4:

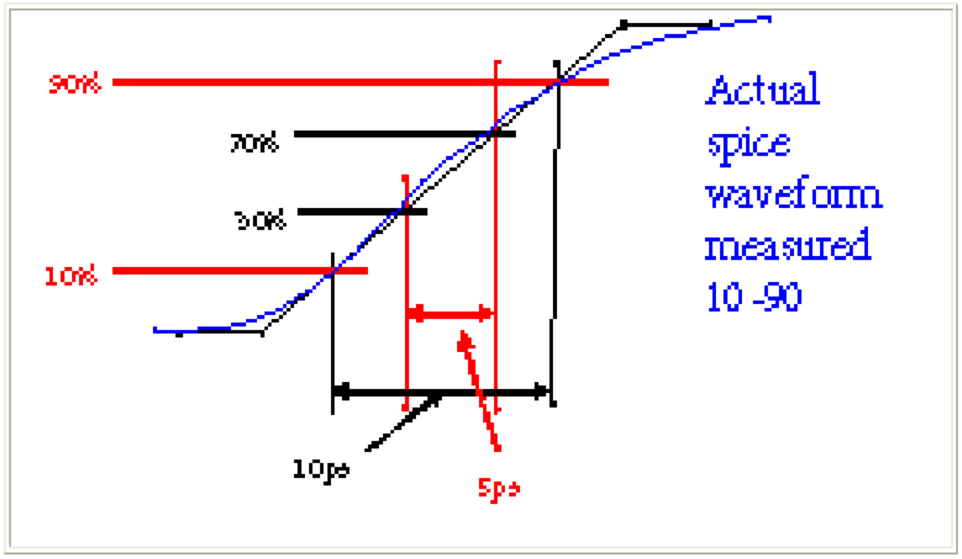
- Spice waveform is measured 10-90
- Spice waveform is showing in blue line
- Spice measured transition time with slew threshold 10-90 is 10 ps
- Linearized waveform is measuring transition time with slew threshold 0-100 is 12.5 ps = 10 * (100-0) / (90-10)
- Representation as single float with slew threshold 30-70 is 5 ps = 12.5 * (70-30) / (100-0)

The spice waveform can thus be represented as a floating point number in Liberty tables or in Timing Engine report as follows:

- A -> 10 ps slew threshold as 10-90 with slew derate 1.0
- B -> 12.5 ps slew threshold as 10-90 with slew derate 0.8
- C -> 5 ps slew threshold as 10-90 with derate 2.0

Note that slew threshold in library are always the threshold used for spice measurement. The case ‘A’ is the native representation, measured in the Liberty. The ‘B’ and ‘C’ are rescaled to the slew threshold 0-100 and 30-70, respectively.

Figure 7-4 Spice Waveform



Handling Slew in Multiple Threshold and Derate Environment

Liberty allows arbitrary slew threshold to minimize the error due to slew linearization for various process technologies. Therefore whenever slew information from library interacts with another library (or even a pin of the same library with different slew threshold), a conversion to common base is required.

Conversion of single-float slews between thresholds and derates:

Assume that you have a library L1 with thresholds TL1-TH1, derate SD1, and L2 with TL2-TH2, derate SD2. Note that the L1, L2 are just entities that have their own local thresholds, such as library, design, library pin.

Assume S1 - slew in local thresholds and derate of L1, S2 - slew in local thresholds and derate of L2. Same conversion rule will apply if we considered maximum transition instead of slews.

Then we can obtain slews expressed in the local thresholds and derate of the other object as follows:

Equivalent slew S2_1 which is slew S2 expressed in local derate/threshold of L1:

S2_1 = S2 * (SD2 / (TH2 - TL2)) * ((TH1 - TL1)/ SD1)

The meaning of S2_1 is a float number that represents a waveform of slew S2 but measured in the context of L1. The S1 and S2_1 can be directly compared. The S1 and S2 cannot.

Note that in presence of detailed RC, slew is computed appropriately in the context of threshold and derate.

Next we will discuss the how max transition constraint is handled in the context of thresholds and derate.

Maximum Transition Constraint Storage

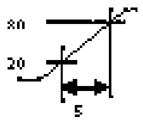
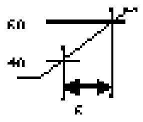
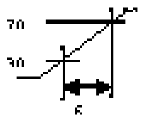
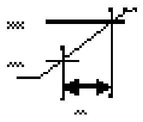
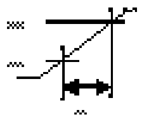
Max transition constraint can come from user, library and library pin. User max transition constraint is expressed with main library derate and slew threshold of Timing Engine.

The set_max_transition command sets a maximum limit on transition time for the nets attached to specified ports, for a whole design, or for pins within a specified clock domain. The set_max_transition command places the max_transition attribute (a design rule constraint) on the specified objects. Setting a maximum transition limit on a port applies to the net connected to that port. Setting the limit on a design sets the default maximum transition time for all nets in the design. Setting the limit on a clock domain applies to all pins within that clock domain. Within a clock domain, you can optionally restrict the constraint further to clock paths or data paths only, and to rising or falling transitions only. Where there is a conflict between design, port, or clock constraints, the most restrictive constraint (the smaller value) applies. This is also true where multiple clocks launch the same path. In Timing Engine the slews and max transition constraint attributes are reported in local threshold and derate of each pin or library. For example, maximum transition time set in the context of design threshold and derate will be scaled to that of the design pin's threshold and derate. The scaling of transition time for slew threshold is on by default.

The following table describes the conversion of slews between different slew threshold and derates:

Table 7-3 Conversion of Slews

Object	**Main Library (L1)**	**Local Library**	**Library Pin**	**Design**	**Cell Instance pin**
Threshold and Derate Source	Yes	Yes	Yes for slew threshold. Inherits derate of the library	Uses main library derate and slew threshold	The more local trip points have higher precedence that the more general ones
Limit (max transition)	Yes	Yes	Yes	Yes - in main lib threshold and derate	The most restrictive limit applies
Thresholds	20/80 (rise/fall)	40/60 (rise/fall)	30/70 (rise/fall)	Uses main library threshold	30/70 (rise/fall)
Derate	0.5 (defined)	0.6 (defined)	0.6 (inherited)	0.5 (inherited)	0.6 (inherited)
Example					

Timing Engine or Liberty value (slew or limit)	10 ps	10 ps	10 ps	10 ps	10 ps
Linearized spice waveform: (rise) Similar for fall					
Expressed in derate and threshold of the main library	10 ps	10 * (0.6 / (0.6 – 0.4)) * ((0.8 – 0.2)/0.5)	10 * (0.6 / (0.7 – 0.3)) * ((0.8 – 0.2)/ / 0.5)	10 * 0.5 / (0.8 – 0.2) * ((0.8 – 0.2) / 0.5)	10 * 0.6 / (0.7 – 0.3) * ((0.8 – 0.2) / 0.5)
Timing Engine or Liberty value (slew or limit):	10 ps	-	-	-	-
Main lib limit expressed in local threshold and derate	10 ps	10 * (0.5/(0.8 -.02)) * ((.6 - .4)/.6)	10 * (0.5/(0.8 – 0.2)) * ((0.7 – 0.3) / 0.6)	10*(0.5/((0.8 – 0.2)) * ((0.8 – 0.2)/0.5)	10 * (0.5/(0.8 – 0.2)) * ((0.7 – 0.3)/0.6)

Evaluation of Max Transition Constraint

To see the maximum transition constraint evaluations, use the report_constraint -max_transition command. Timing Engine reports all constraints and slews in threshold and derate of the pin of cell instance and the violations are sorted based on the absolute values (that is, they are not expressed in that of design threshold and derate).

To see the port maximum transition limit, use the report_port -design_rule command. To see the default maximum transition setting for the current design, use the report_design command. To undo maximum transition limits previously set on ports, designs, or clocks, use remove_max_transition.

Example 1

To set a maximum transition limit of 2.0 units on the ports of OUT*, enter:

```
set_max_transition 2.0 [get_ports "OUT*"]
```

To set the default maximum transition limit of 5.0 units on the current design, enter:

```
 set_max_transition 5.0 [current_design]
```

To set the maximum transition limit of 4.0 units on all pins the CLK1 clock domain, for rising transitions in data paths only, enter:

```
 set_max_transition 4.0 [get_clocks CLK1] \ -type
data_path -rise
```

For more information, see the set_max_transition man page.

Example 2 - Scaling the Max Transition with Different Library Slew Threshold

Consider library lib1 having slew threshold 10/90. Design has max transition limit set by the user at 0.3 ns. The main library slew threshold for rise and fall is 30/70. By using the report_constraints -max_transition -all_violators -sig 4 command:

```
            Required     Actual
Pin           Transition   Transition      Slack
---------------------------------------------------------
FF1/D         0.6000       0.4000          0.2000
```

Example 3 - Scaling the Max Transition with Slew Derate Factor Specified in Library

Library lib1 having slew derate factor 0.5 and slew threshold for rise and fall 30/70. User sets a max transition on the design at 0.3 ns. The main library slew threshold for rise and fall is 30/70 and slew derate is 1.0. By using the report_constraints -max_transition -all_violators -sig 4 command:

```
            Required     Actual
Pin          Transition   Transition      Slack
----------------------------------------------------------
FF1/D         0.6000       0.4000         0.2000
```

Minimum and Maximum Net Capacitance

The set_min_capacitance command sets a minimum capacitance limit for the nets attached to specified ports or for the whole design. The set_min_capacitance command places the min_capacitance attribute (a design rule constraint) on the specified objects.

Similarly, the set_max_capacitance command sets a maximum limit on total capacitance for each net.

Setting a capacitance limit on a port applies to the net connected to that port. Setting a capacitance limit on a design sets the default capacitance limit for all nets in the design. In case of conflict between these two values, the more restrictive value applies.

To see the capacitance constraint evaluations, use the report_constraint -min_capacitance or -max_capacitance command. To see port capacitance limits, use the report_port -design_rule command. To see the default capacitance settings for the current design, use the report_design command.

To undo capacitance limits you set on ports or designs, use remove_min_capacitance and remove_max_capacitance.

Examples

To set a minimum capacitance limit of 0.2 units on the ports of OUT*, enter

```
set_min_capacitance 0.2 [get_ports "OUT*"]
```

To set the default minimum capacitance limit of 0.1 units on the current design, enter

```
set_min_capacitance 0.1 [current_design]
```

For more information, see the *Design Compiler Reference Manual: Constraints and Timing*.

Maximum Fanout Load

The set_max_fanout command sets a maximum fanout load for specified output ports or designs. The command sets the max_fanout attribute (a design rule constraint) on the specified objects.

Setting a maximum fanout load on a port applies to the net connected to that port. Setting a maximum fanout load on a design sets the default maximum for all nets in the design. In case of conflict between these two values, the more restrictive value applies.

Library cell pins can have a max_fanout value specified. Timing Engine uses the more restrictive of the limit you set or the limit specified in the library.

To see maximum fanout constraint evaluations, use the report_constraint -max_fanout command. To see port maximum fanout limits, use the report_port -design_rule command. To see the default maximum fanout setting for the current design, use the report_design command.

To undo maximum fanout limits you set on ports or designs, use the remove_max_fanout command.

Examples

To set a maximum fanout limit of 2.0 units on the ports IN*, enter

```
set_max_fanout 2.0 [get_ports "IN*"]
```

To set the default maximum fanout limit of 5.0 units on the current design, enter

```
set_max_fanout 5.0 [current_design]
```

Fanout Load Values for Output Ports

The fanout load for a net is the sum of fanout_load attributes for the input pins and output ports connected to the net. Output pins can have maximum fanout limits, specified in the library or set through the set_max_fanout command. By default, ports are considered to have a fanout load of 0.0.

The set_fanout_load command specifies the expected fanout load for output ports in the current design.

Example

To set the fanout load on ports matching OUT* to 3.0, enter

```
set_fanout_load 3.0 "OUT*"
```

For more information, see the *Design Compiler Reference Manual: Constraints and Timing*.

Timing Exceptions

By default, Timing Engine assumes that data launched at a path startpoint ought to be captured at the path endpoint by the very next occurrence of a clock edge at the endpoint. For paths that are not intended to operate in this manner, you need to specify a timing exception. Otherwise, the timing analysis will not match the behavior of the real circuit.

This chapter covers the following timing exception topics:

- Timing Exception Overview
- Single-Cycle (Default) Path Delay Constraints
- Setting False Paths
- Setting Maximum and Minimum Path Delays
- Setting Multicycle Paths
- Specifying Exceptions Efficiently
- Exception Order of Precedence
- Reporting Exceptions
- Checking Ignored Exceptions
- Removing Exceptions
- Transforming Exceptions

Timing Exception Overview

Timing Engine supports the following ways to specify timing exceptions:

- Setting false paths. Use the set_false_path command to specify a logic path that exists in the design but should not be analyzed. Setting a false path removes the timing constraints on the path.
- Setting minimum and maximum path delay values. Use the set_max_delay and set_min_delay commands to override the default setup and hold constraints with specific maximum and minimum time values.
- Setting multicycle paths. Use the set_multicycle_path command to specify the number of clock cycles required to propagate data from the start to the end of the path.

Each timing exception command can apply to a single path or to a group of related paths, such as all paths from one clock domain to another, or all paths that pass through a specified point in the design.

When you specify a path by its startpoint and endpoint, be sure to specify a timing path that is valid in Timing Engine. A path startpoint must be either a register clock pin or an input port. A path endpoint must be either a register data input pin or an output port.

To view a list of timing exceptions that have been applied to a design, use the report_exceptions command. To restore the default timing constraints on a path, use the reset_path command.

Single-Cycle (Default) Path Delay Constraints

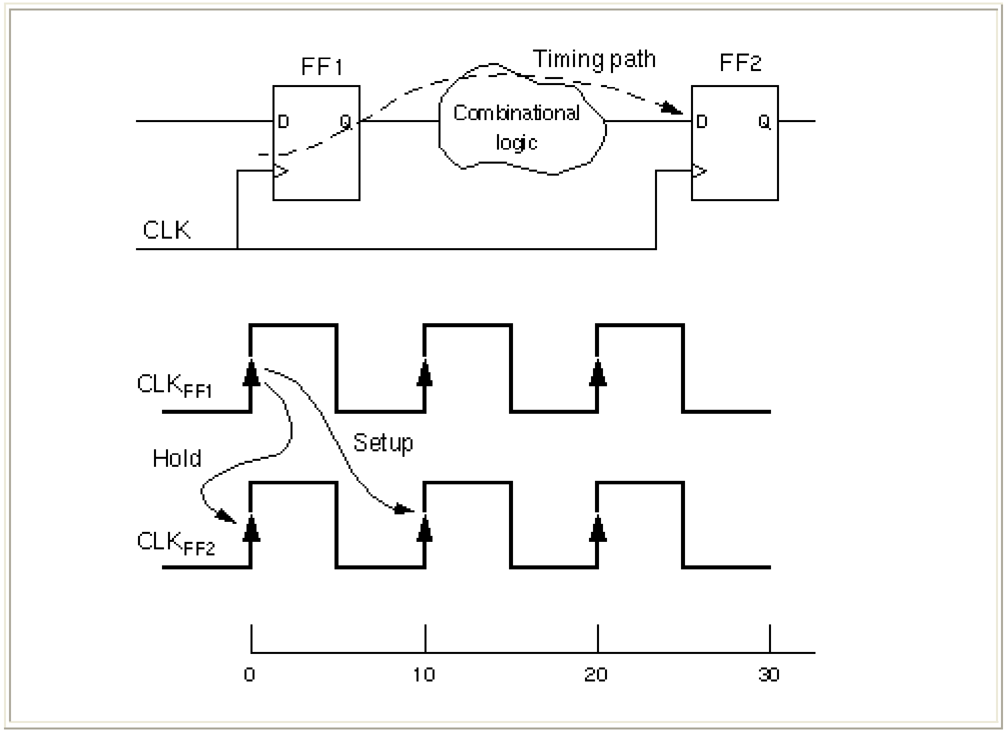
To determine whether you need to set a timing exception, you need to understand how Timing Engine calculates maximum and minimum delay times for paths, and how this calculation is like (or is not like) the operation of the design being analyzed.

Path Delay for Flip-Flops Using a Single Clock

Figure 8-1 shows how Timing Engine determines the setup and hold constraints for a path that begins and ends on rising-edge-triggered flip-flops. In this simple example, the two flip-flops are triggered by the same clock. The clock period is 10 ns.

Timing Engine performs a setup check to verify that the data launched from FF1 at time=0 arrives at the D input of FF2 in time for the capture edge at time=10. If the data takes too long to arrive, it is reported as a setup violation.

Figure 8-1 Single-Cycle Setup and Hold for Flip-Flops



Similarly, Timing Engine performs a hold check to verify that the data launched from FF1 at time 0 does not get propagated so soon that it gets captured at FF2 at the clock edge at time 0. If the data arrives too soon, it is reported as a hold violation.

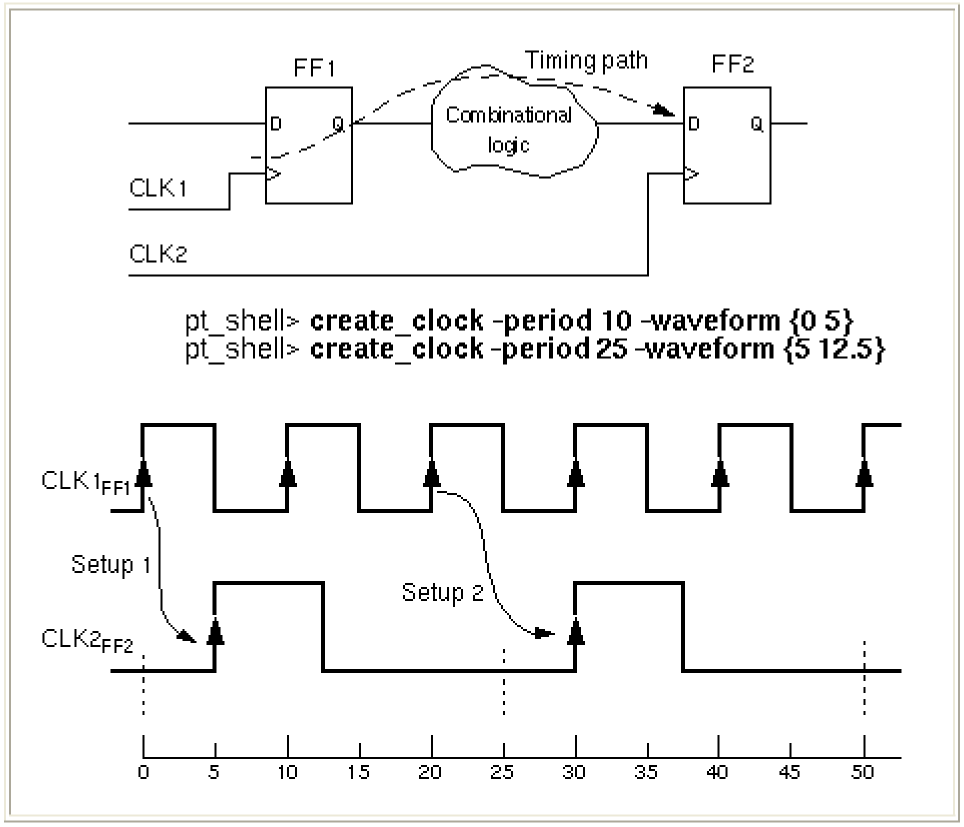
The setup and hold requirements determined by Timing Engine for sequential elements take into account all relevant parameters such as the delay of the combinational logic elements in the path, the setup and hold requirements of the flip-flop elements as defined in the technology library, and the net delays between the flip-flops.

Path Delay for Flip-Flops Using Different Clocks

The algorithm for calculating path delays is more complicated when the launch and capture flip-flops belong to different clock domains.

Consider the circuit shown in Figure 8-2. The flip-flops at the beginning and end of the timing path are clocked by different clocks, CLK1 and CLK2. The two clocks are declared by the create_clock commands shown in the figure.

Figure 8-2 Setup Constraints for Flip-Flops With Different Clocks



```
pt_shell> create_clock -period 10 -waveform {0 5}
pt_shell> create_clock -period 25 -waveform {5 12.5}
```

By default, Timing Engine assumes that the two clocks are synchronous to each other, with a fixed phase relationship. If this is not the case for the real circuit (for example, because the two clocks are never active at the same time or because they operate asynchronously), then you need to declare this fact by using any of several methods. Otherwise, Timing Engine will spend time checking constraints and reporting violations that do not exist in the actual circuit.

Setup Analysis

Timing Engine looks at the relationship between the active clock edges over a full repeating cycle, equal to the least common multiple of the two clock periods. For each capture (latch) edge at the destination flip-flop, Timing Engine assumes that the corresponding launch edge is the nearest source clock edge occurring before the capture edge

In Figure 8-2, there are two capture edges in the period under consideration. For the capture edge at time=5, the nearest preceding launch edge is at time=0. The corresponding setup relationship is labeled Setup 1.

For the capture edge at time=30, the nearest preceding launch edge is at time=20. This setup relationship is labeled Setup 2. The source clock edge at time=30 occurs at the same time as the capture edge, not earlier, so it is not considered the corresponding launch edge.

Setup 1 allows less time between launch and capture, so it is the more restrictive constraint. It determines the maximum allowed delay for the path, which is 5 ns for this example.

Hold Analysis

The hold relationships checked by Timing Engine are based on the clock edges adjacent to those used to determine the setup relationships. To determine the most restrictive hold relationship, Timing Engine considers all valid setup relationships, including both Setup 1 and Setup 2 in Figure 8-2.

For each setup relationship, Timing Engine performs two different hold checks:

- The data launched by the setup launch edge must not be captured by the *previous* capture edge.
- The data launched by the *next* launch edge must not be captured by the setup capture edge.

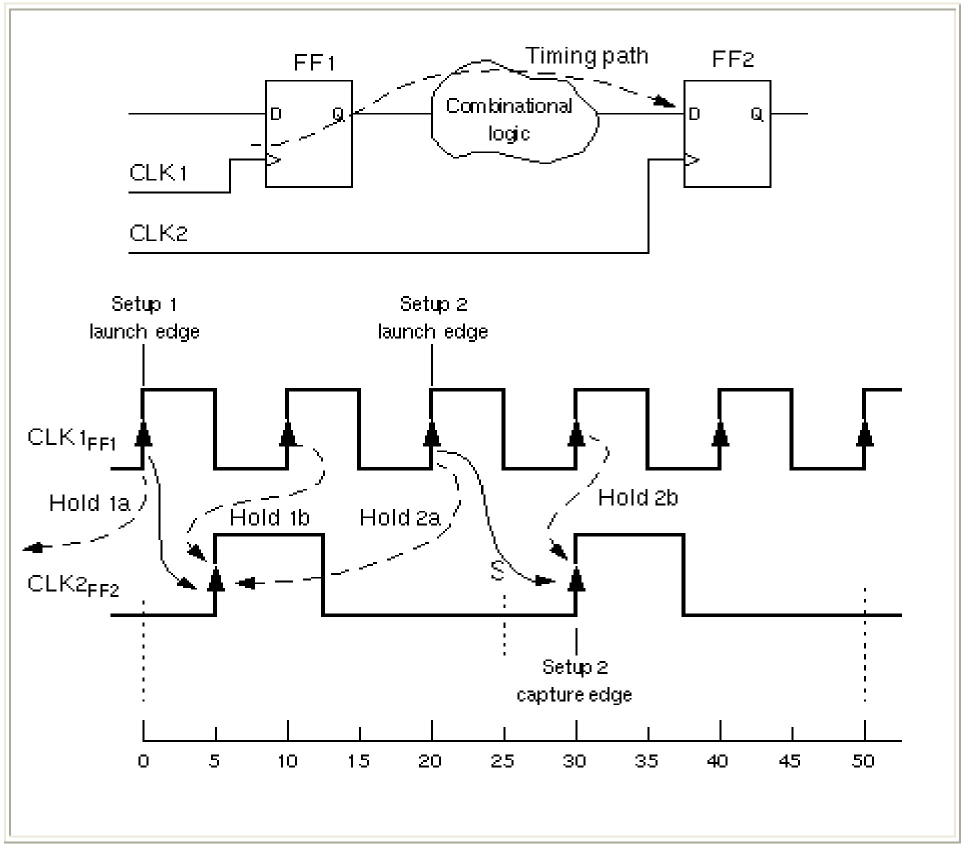
Figure 8-3 shows the hold checks performed by Timing Engine for the current example. First consider the setup relationship labeled Setup 2. Timing Engine confirms that the data launched by the setup launch edge is not captured by the previous capture edge; this relationship is labeled Hold 2a. It also confirms that the data launched by the next clock edge at the source is not captured by the setup capture edge; this relationship is labeled Hold 2b.

Timing Engine similarly checks the hold relationships relative to the clock edges of Setup 1, as indicated in the figure. The most restrictive hold check is the one where the capture edge occurs latest relative to the launch edge, which is Hold 2b in this example. Therefore, Hold 2b determines the minimum allowed delay for this path, 0 ns.

Single-Cycle Path Analysis Examples

The following examples further illustrate how Timing Engine calculates the delay requirements for edge-triggered flip-flops in the absence of timing exceptions.

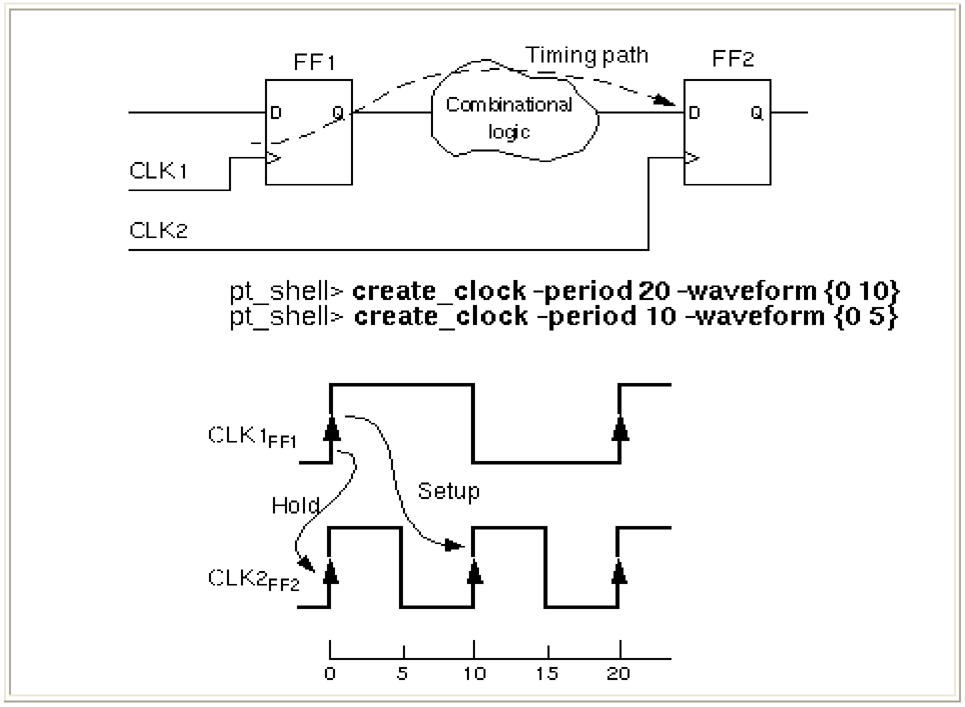
Figure 8-3 Hold Constraints for Flip-Flops With Different Clocks



In Figure 8-4, CLK1 has a period of 20 ns and CLK2 has a period of 10 ns. The most restrictive setup relationship is the launch edge at time=0 to the capture edge at time=10. The most restrictive hold relationship is the launch edge at time=0 to the capture edge at time=0

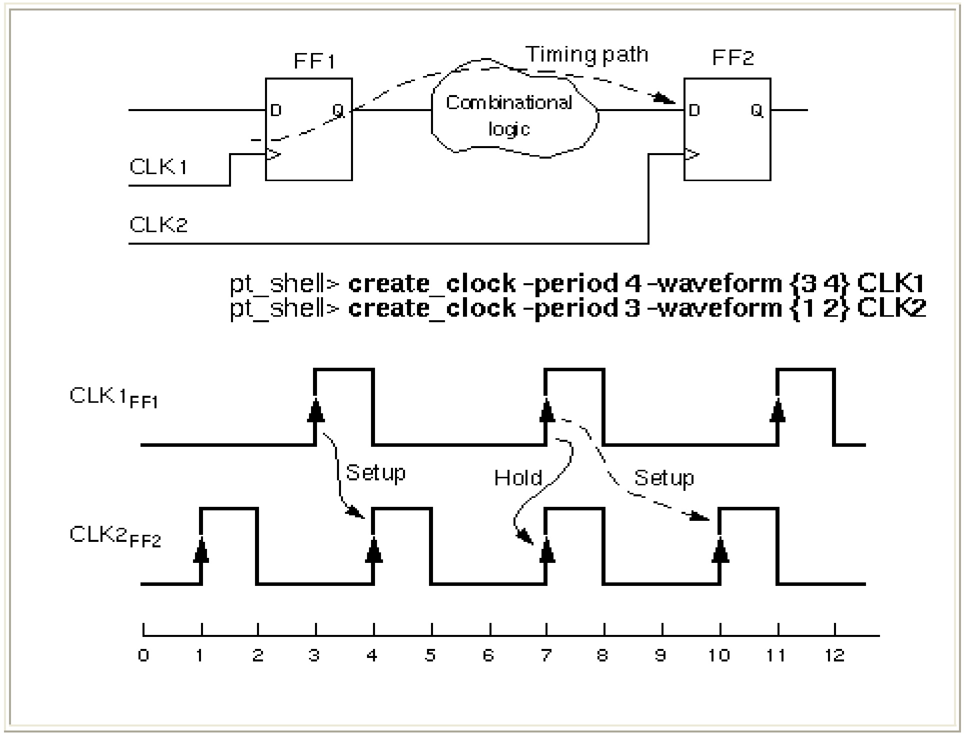
In Figure 8-5, CLK1 has a period of 4 ns and CLK2 has a period of 3 ns. The most restrictive setup relationship is the launch edge at time=3 to the capture edge at time=4. The most restrictive hold relationship is the launch edge at time=7 to the capture edge at time=7, based on the setup relationship shown by the dashed-line arrow in the timing diagram.

Figure 8-4 Delay Requirements With 20ns/10ns Clocks



```
pt_shell> create_clock -period 20 -waveform {0 10}
pt_shell> create_clock -period 10 -waveform {0 5}
```

Figure 8-5 Delay Requirements With 4ns/3ns Clocks



```
pt_shell> create_clock -period 4 -waveform {3 4} CLK1
pt_shell> create_clock -period 3 -waveform {1 2} CLK2
```

Setting False Paths

A false path is a logic path in the design that exists but should not be analyzed for timing. For example, a path can exist between two multiplexed logic blocks that are never selected at the same time, so that path is not valid for timing analysis.

For example, to declare a false path from pin FFB1/CP to pin FFB2/D:

set_false_path -from [get_pins FFB1/CP] -to [get_pins FFB2/D]

Declaring a path to be false removes all timing constraints from the path. Timing Engine still calculates the path delay, but does not report it to be an error, no matter how long or short the delay.

Setting a false path is a point-to-point timing exception. This is different from using the set_disable_timing command, which disables timing analysis for a specified pin, cell, or port. Using the set_disable_timing command removes the affected objects from timing analysis, rather than simply removing the timing constraints from the paths. If all paths through a pin are false, using set_disable_timing [get_pins *pin_name*] is more efficient than using set_false_path -through [get_pins *pin_name*].

Another example of a false path is a path between flip-flops belonging to two clock domains that are asynchronous with respect to each other.

To declare all paths between two clock domains to be false, you can use a set of two commands such as the following:

set_false_path -from [get_clocks ck1] -to [get_clocks ck2]
set_false_path -from [get_clocks ck2] -to [get_clocks ck1]

For efficiency, be sure to specify each clock by its clock name, not by the pin name (use get_clocks, not get_pins).

An alternative is to use the set_clock_groups command to exclude paths from consideration that are launched by one clock and captured by another. Although this has the same effect as declaring a false path between the two clocks, it is not considered a timing exception and is not reported by the report_exceptions command.

Timing Engine has the ability to detect the presence of certain types of false paths in the design when you use case analysis. To do this, use the report_timing command with the -justify option. For more information, see the section *Timing Engine User Guide: Advanced Timing Analysis* manual.

Setting Maximum and Minimum Path Delays

By default, Timing Engine calculates the maximum and minimum path delays by considering the clock edge times. To override the default maximum or minimum time with your own specific time value, use the set_max_delay or set_min_delay command.

For example, to set the maximum path delay between registers REGA and REGB to 12, use the following command:

```
set_max_delay 12 -from [get_cells REGA] -to [get_cells REGB]
```

With this timing exception, Timing Engine ignores the clock relationships. A path delay between these registers that exceeds 12 time units minus the setup requirement of the endpoint register is reported as a timing violation.

Similarly, to set the minimum path delay between registers REGA and REGB to 2, use the following command:

```
set_min_delay 2.0 -from [get_cells REGA] -to [get_cells REGB]
```

Again, Timing Engine ignores the clock relationships. A path delay between these registers that is less than 2 time units plus the hold requirement of the endpoint register is reported as a timing violation.

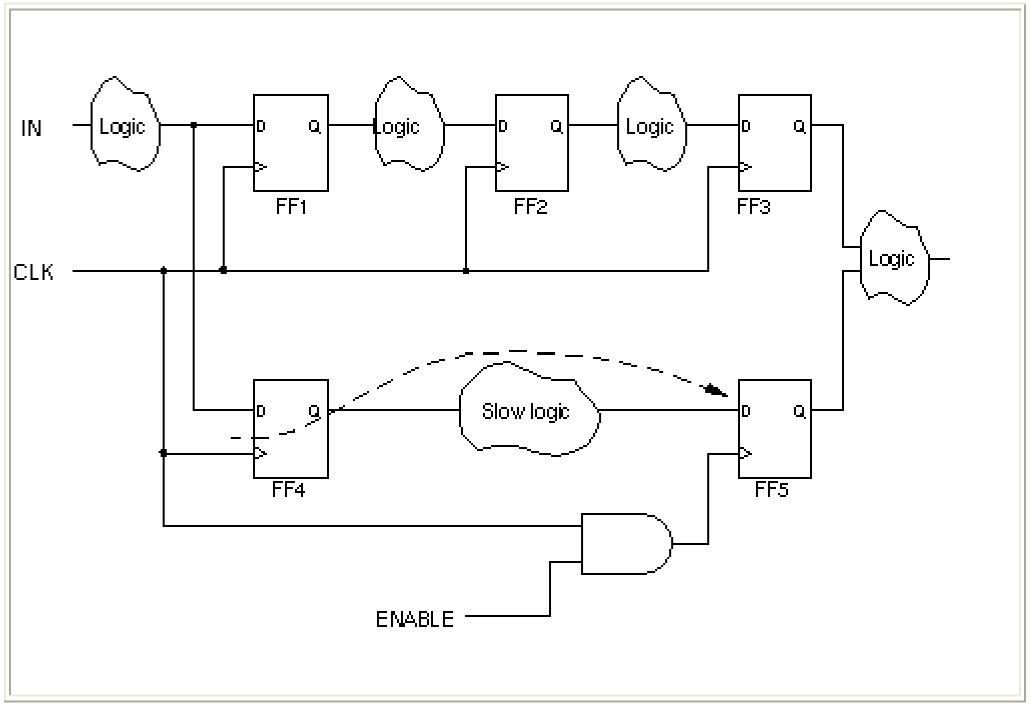
You can optionally specify that the delay value apply only to rising edges or only to falling edges at the endpoint.

Setting Multicycle Paths

The set_multicycle_path command specifies the number of clock cycles required to propagate data from the start of a path to the end of the path. Timing Engine calculates the setup or hold constraint according to the specified number of cycles.

For example, consider the circuit shown in Figure 8-6. The path from FF4 to FF5 is designed to take two clock cycles rather than one. However, by default, Timing Engine assumes single-cycle timing for all paths. Therefore, you need to specify a timing exception for this path.

Figure 8-6 Multicycle Path Example



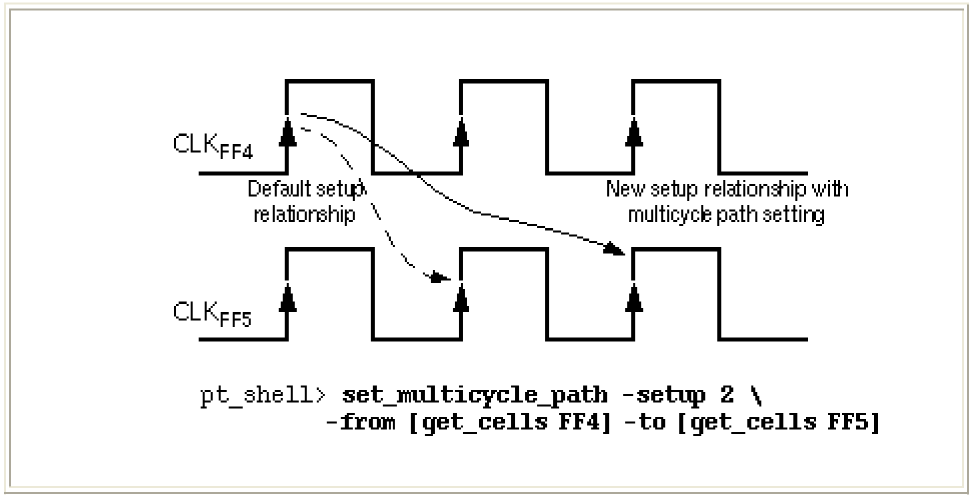
One timing exception method is to specify an explicit maximum delay value with the set_max_delay command. However, you might want to use set_multicycle_path instead because the maximum delay value is automatically adjusted when you change the clock period.

To set the multicycle path for the design shown in Figure 8-6, you can use the following command:

set_multicycle_path -setup 2 -from [get_cells FF4] -to [get_cells FF5]

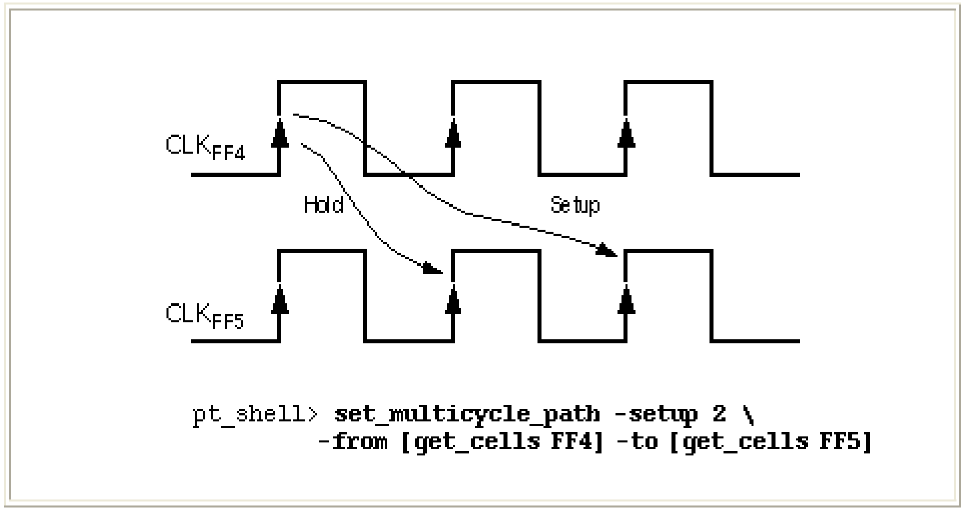
The first command tells Timing Engine that the path takes two clock cycles rather than one, establishing the new setup relationship shown in Figure 8-7. The second capture edge (rather than the first) following the launch edge becomes the applicable edge for the end of the path.

Figure 8-7 Multicycle Path Setup


CLKFF4
Default setup relationship
New setup relationship with multicycle path setting
CLKFF5

```
pt_shell> set_multicycle_path -setup 2 \
          -from [get_cells FF4] -to [get_cells FF5]
```

Figure 8-8 Multicycle Path Hold Based on New Setup



```
pt_shell> set_multicycle_path -setup 2 \
          -from [get_cells FF4] -to [get_cells FF5]
```

The hold relationship shown in Figure 8-8 is probably not the correct relationship for the design. If FF4 does not need to hold the data beyond the first clock edge, you need to specify another timing exception.

Although you could use the set_min_delay command to specify a particular hold time, it is better to use another set_multicycle_path command to move the capture edge for the hold relationship backward by one clock cycle. For example:

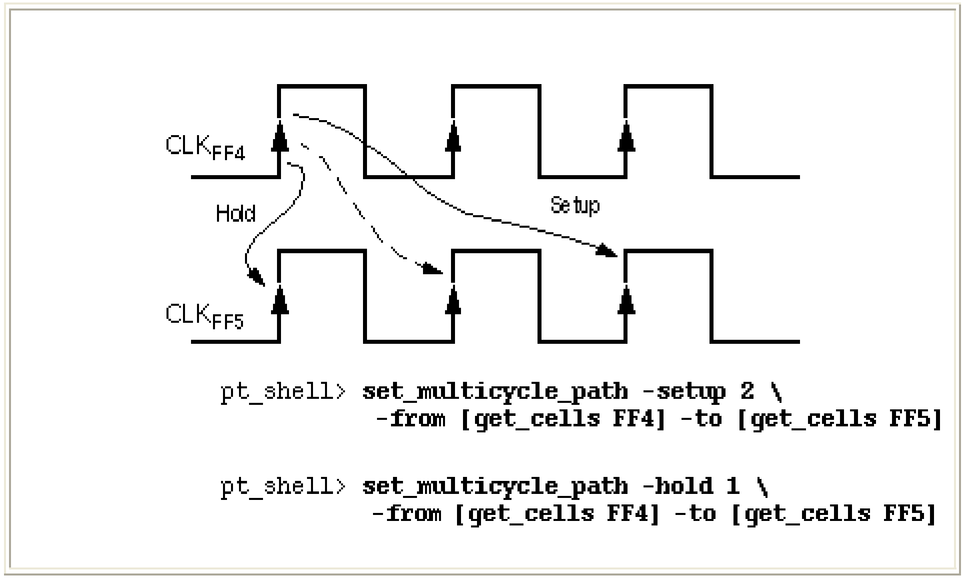
set_multicycle_path -setup 2 -from [get_cells FF4] -to [get_cells FF5]
set_min_delay 0 -from [get_cells FF4] -to [get_cells FF5]

or preferably:

set_multicycle_path -setup 2 -from [get_cells FF4] -to [get_cells FF5]
set_multicycle_path -hold 1 -from [get_cells FF4] -to [get_cells FF5]

Figure 8-9 shows the setup and hold relationships set correctly with two set_multicycle_path commands. The second set_multicycle_path command moves the capture edge of the hold relationship backward by one clock cycle, from the dashed-line arrow to the solid-line arrow.

Figure 8-9 Multicycle Path Hold Set Correctly



```
pt_shell> set_multicycle_path -setup 2 \
           -from [get_cells FF4] -to [get_cells FF5]

pt_shell> set_multicycle_path -hold 1 \
           -from [get_cells FF4] -to [get_cells FF5]
```

With set_multicycle_path -setup 5, the setup relationship spans five clock cycles rather than one, from time=0 to time=50, as shown by the long solid-line arrow. This implicitly changes the hold relationship to the prior capture edge at time=40, as shown by the long dashed-line arrow.

To move the capture edge for the hold relationship back to time=0, you need to use set_multicycle_path -hold 4 to move the capture edge back by four clock cycles.

To summarize, Timing Engine determines the number of hold cycles as follows:

(hold cycles) = (setup option value) – 1 – (hold option value)

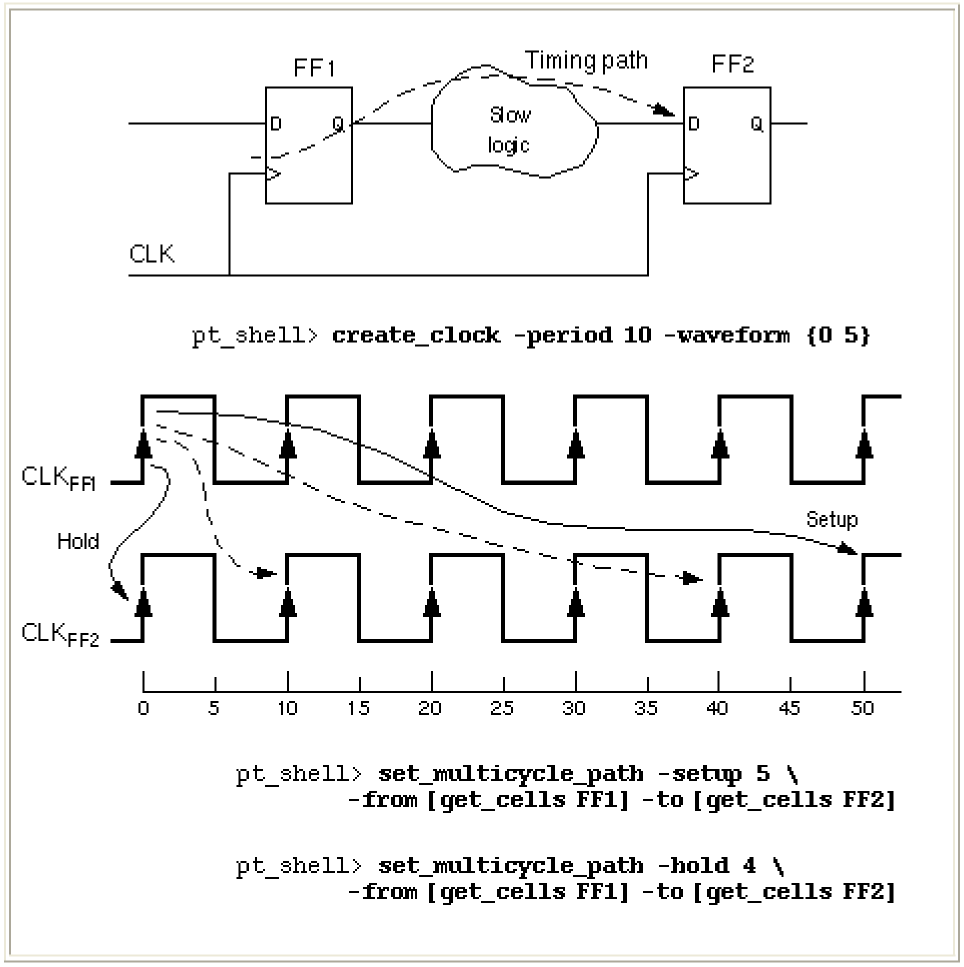
By default, hold cycles = 1 – 1 – 0 = 0.

For Figure 8-9, hold cycles = 2 – 1 – 1 = 0.

For Figure 8-10, hold cycles = 5 – 1 – 4 = 0.

You can optionally specify that the multicycle path exception apply only to rising edges or only with falling edges at the path endpoint. If the startpoint and endpoint are clocked by different clocks, you can specify which of the two clocks is considered for adjusting the number of clock cycles for the path.

Figure 8-10 Multicycle Path Taking Five Clock Cycles



```
pt_shell> create_clock -period 10 -waveform {0 5}
```

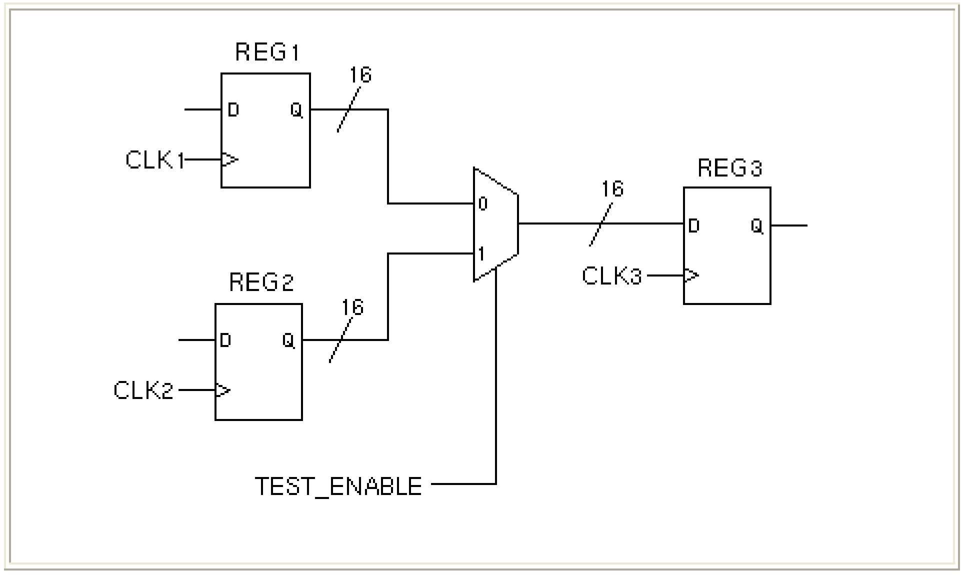
```
pt_shell> set_multicycle_path -setup 5 \
          -from [get_cells FF1] -to [get_cells FF2]

pt_shell> set_multicycle_path -hold 4 \
          -from [get_cells FF1] -to [get_cells FF2]
```

Specifying Exceptions Efficiently

In many cases, you can specify the exception paths many different ways. Choosing an efficient method can reduce the analysis runtime.

For example, consider the circuit shown in Figure 8-11. The three 16-bit registers are clocked by three different clocks. Each register represents 16 flip-flops. Register REG2 is only used for test purposes, so all paths from REG2 to REG3 are false paths during normal operation of the circuit.

Figure 8-11 Multiplexed Register Paths



To prevent analysis of timing from REG2 to REG3, you can use any of the following methods:

- Use case analysis to consider the case when the test enable signal is 0
- Set an exclusive relationship between the CLK1 and CLK2 clock domains.
- Declare the paths between clock domains CLK2 and CLK3 to be false.

set_false_path -from [get_clocks CLK2] -to [get_clocks CLK3]

This method is an efficient way to specify the false paths because Timing Engine only needs to keep track of the specified clock domains. It does not have to keep track of exceptions on registers, pins, nets, and so on.

- Declare the 16 individual paths from REG2 to REG3 to be false.

set_false_path -from [get_pins REG2[0]/CP] -to [get_pins REG3[0]/D]
set_false_path -from [get_pins REG2[1]/CP] -to [get_pins REG3[1]/D]

This method is less efficient because Timing Engine must keep track of timing exceptions for 16 different paths.

- Declare all paths from REG2 to REG3 to be false.

set_false_path -from [get_pins REG2[*]/CP] -to [get_pins REG3[*]/D]

This method is even less efficient because Timing Engine must keep track of paths from each clock pin of REG2 to each data pin of REG3, a total of 256 paths.

- Declare all paths from REG2 to be false.

set_false_path -from [get_pins REG2[*]/CP]

This method is similar to the previous one. Timing Engine must keep track of all paths originating from each clock pin of REG2, a total of 256 paths.

In summary, look at the root cause that is making the exceptions necessary and find the simplest way to control the timing analysis for the affected paths. Before using false paths, consider using case analysis (set_case_analysis), declaring an exclusive relationship between clocks (set_clock_groups), or disabling analysis of part of the design (set_disable_timing). These alternatives can be more efficient than using set_false_path.

If you must set false paths, avoid specifying a large number of paths using the -through argument, by using wildcards, or by listing the paths one at a time. After you set false paths and other timing exceptions, you might be able to simplify the set of exception-setting commands by using the transform_exceptions command

Exception Order of Precedence

If different timing exception commands are in conflict for a particular path, the exception Timing Engine uses for that path depends on the exception types or the path specification methods used in the conflicting commands. A set of rules establishes the order of priority for different exception-setting situations.

Note that Timing Engine applies the exception precedence rules independently on each path (not each command). For example, suppose that you use the following commands:

set_max_delay -from A 5.1
set_false_path -to B

The set_false_path command has priority over the set_max_delay command, so any paths that begin at A and end at B are false paths. However, the set_max_delay command still applies to paths that begin at A but do not end at B.

Exception Type Priority

The following pairs of timing exception types are not considered to be in conflict, so both settings can be valid for a path:

- Two set_false_path settings
- set_min_delay and set_max_delay settings
- set_multicycle_path -setup and -hold settings

In case of conflicting exceptions for a particular path, the timing exception types have the following order of priority, from highest to lowest:

1. set_false_path
2. set_max_delay and set_min_delay
3. set_multicycle_path

For example, if you declare a path to be false and also set its maximum delay to some value, the false path declaration has priority. The maximum delay setting is ignored. You can list ignored timing exceptions by using report_exceptions -ignored.

Path Specification Priority

If you apply the same type of timing exception using commands with different path specifications, the more specific command has priority over the more general one. For example:

```
set_max_delay 12 -from [get_clocks CLK1]

set_max_delay 15 -from [get_clocks CLK1] \
          -to [get_clocks CLK2]
```

The first command sets the maximum delay of all paths starting from CLK1. However, the second command is more specific, so it overrides the first command for paths starting at CLK1 and ending at CLK2. The remaining paths starting from CLK1 are still controlled by the first command.

The various -from/-to path specification methods have the following order of priority, from highest to lowest:

1. -from *pin*, -rise_from *pin*, -fall_from *pin*
2. -to *pin*, -rise_to *pin*, -fall_to *pin*
3. -through, -rise_through, -fall_through
4. -from *clock*, -rise_from *clock*, -fall_from *clock*
5. -to *clock*, -rise_to *clock*, -fall_to *clock*

Use the preceding list to determine which of two conflicting timing exception commands has priority (for example, two set_max_delay commands). Starting from the top of the list:

1. A command containing -from *pin*, -rise_from *pin*, or -fall_from *pin* has priority over a command that does not contain -from *pin*, -rise_from *pin*, or -fall_from *pin*.
2. A command containing -to *pin*, -rise_to *pin*, or -fall_to *pin* has priority over a command that does not contain -to *pin*, -rise_to *pin*, or -fall_to *pin*.

... and so on down the list until the priority is resolved.

Here are some possible path specification combinations, listed in order of priority from highest to lowest, according to the preceding priority rules:

1. -from *pin* -to *pin*
2. -from *pin* -to *clock*
3. -from *pin*
4. -from *clock* -to *pin*
5. -to *pin*
6. -from *clock* -to *clock*
7. -from *clock*
8. -to *clock*

Reporting Exceptions

To get a report on timing exceptions that have been set, use the report_exceptions command. You can reduce the scope of the report by using the path specification arguments -from, -to, -through, -rise_from, -fall_to, and so on, to match the path specifiers used when the original exceptions were created.

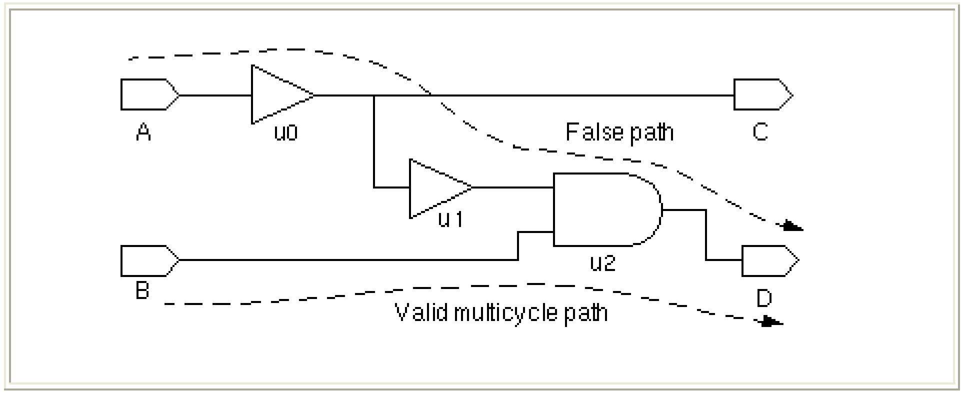
The report_exceptions command causes a complete timing update, so be sure to use it only after you have set up all timing assertions and you are ready to receive the report.

Exception-setting commands are sometimes partially or fully ignored for a number of reasons. For example, a command that specifies a broad range of paths can be partially overridden by another exception-setting command that specifies a subset of those paths. For a partially ignored command, the exception report shows the reason that some part of the command is being ignored.

By default, a command that is fully ignored is not reported by the report_exceptions command. To get a report on just the commands that are fully ignored, use the -ignored option with the report_exceptions command.

Consider the simple design shown in Figure 8-12 and the following exception-setting and exception-reporting commands.

Figure 8-12 Design to Demonstrate Ignored Exceptions



```
set_false_path -from A -to D
set_multicycle_path 2 -from A -to D
set_multicycle_path 2 -from {A B C} -to D
report_exceptions
...
Reasons:
f - invalid startpoint(s)
t - invalid endpoint(s)
p - non-existent paths
o - overridden paths

From       To     Setup     Hold      Ignored
-------------------------------------------------
A          D      FALSE     FALSE
{ A B C }   D      2         *         f,o
```

```
report_exceptions -ignored
...
From       To     Setup     Hold      Ignored
-------------------------------------------------
A          D      2         *         o
```

The first exception-setting command sets a false path from port A to port D. This is the highest-priority command, so it is fully enforced by Timing Engine.

The second exception-setting command attempts to set a multicycle path from port A to port D. It is fully ignored because it is overridden by the higher-priority false path command. Because this command is fully ignored, it is reported only when you use the -ignored option of the report_exceptions command. In the report, the letter code "o" in the "Ignored" column shows the reason the command is being ignored.

The third exception-setting command attempts to set multicycle paths from ports A to D, B to D, and C to D. It is partially valid (not fully ignored), so it is reported by the report_exceptions command without the -ignored option. The path from A to D is ignored because it is overridden by the false path command. The path from B to D is valid, so that multicycle path is enforced by Timing Engine. The path from C to D is invalid because port C is not a valid startpoint for a path. In the report, the letter codes in the "Ignored" column indicate the reasons that some of the paths specified in the command are being ignored.

Checking Ignored Exceptions

A timing exception entered by the user but not accepted by Timing Engine is called an ignored exception. There are several reasons an exception can be ignored:

- The specified startpoint or endpoint is not valid. The startpoint must be a register clock pin or input port. The endpoint must be a register data input pin or output port.
- The specified path is not constrained (for example, the startpoint is an input port with no input delay set, or the capture flip-flop at the endpoint is not clocked by a defined clock signal).
- The path is invalid because of set_disable_timing, constant propagation, loop breaking, or case analysis.
- The exception has a lower priority than another exception applied to the same path.

The report_exceptions command reports exceptions that are fully and partially valid. To get a report on all fully ignored exceptions, use report_exceptions -ignored. It is a good idea to examine these reports to confirm that you have specified all exceptions correctly.

If ignored exceptions are reported, you should determine the cause and correct them by changing the path specifications or removing the exception-setting commands. Large numbers of ignored exceptions can increase memory usage and analysis runtime.

If the reason that an exception is partially or fully ignored is not immediately apparent, check the reasons listed in the "Ignored" column of the exception report and consider the possible causes listed above. It might be helpful to get more information using the following commands:

```
transform_exceptions -dry_run
report_exceptions -ignored
```

To find out if there is a logical path between two points, use:

```
all_fanout -from point_a -endpoints_only
all_fanin -to point_b -startpoints_only
```

After a timing update, to examine the path timing, you can use:

```
report_timing -from point_a -to point_b
```

To convert the current set of exception-setting commands to a simpler, equivalent set of commands, use the transform_exceptions command.

For an example of a report on ignored exceptions, see Figure 8-12.

Removing Exceptions

To remove a timing exception previously set with set_false_path, set_max_delay, set_min_delay, or set_multicycle_path, use the reset_path command.

You control the scope of exception removal in the reset_path command by specifying -from, -to, -through, -rise_from, -fall_to, and so on. The path specification and object (such as pin, port, or clock) must match the original path specification and object used to set the exception. Otherwise, the reset_path command will have no effect. For example:

set_false_path -from [get_clocks CLK]
reset_path -from [get_pins ff1/CP] # ff1 clocked
by CLK

set_false_path -through [get_pins {d/Z g/Z}]
reset_path -through [get_pins a/Z] # where a fans
out to d

In each of these two examples, the object in the reset_path command does not match the original object, so the paths are not reset, even though they might have exceptions applied.

You can use the -reset_path option in an exception-setting command to reset all of the exceptions set on a path before the new exception is applied. For example:

set_false_path -through [get_pins d/Z] -reset_path

This example first resets all timing exceptions previously applied to the paths through the specified point, then applies the false path exception to these paths.

To remove all exceptions from the design, you can use the reset_design command, which removes all user-specified clocks, path groups, exceptions, and attributes (except those defined with set_attribute).

Transforming Exceptions

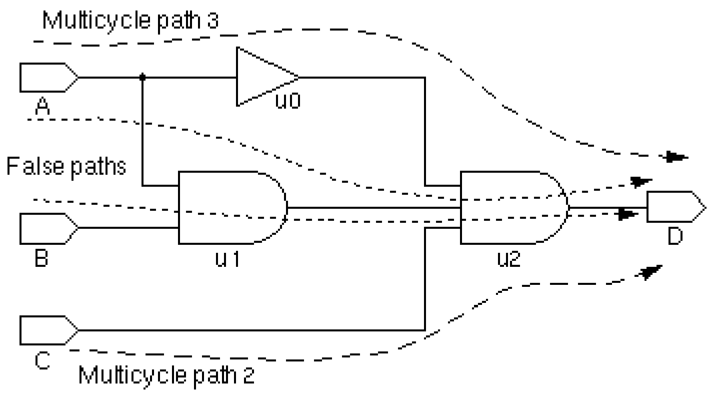
Exceptions set by the user are sometimes partially or completely ignored for various reasons. Multiple exception-setting commands with overlapping path specifications can make it difficult to understand the scope of each command.

To gain a better understanding of these exceptions, you can use the report_exceptions command. The generated report indicates the reasons that paths were ignored for each exception-setting command, based on the exceptions originally entered by the user. However, the report does not always make it clear what exceptions apply to a particular path.

The transform_exceptions command has the ability to transform the current set of exceptions into a new, simpler set of equivalent exception-setting commands. The command can be used to get information about the exceptions that apply to a particular path or group of related paths, or to transform a complex set of exception-setting commands into a simpler, equivalent set of commands.

The simple design and exception-setting commands shown in Figure 8-13 demonstrate some of the principles of exception transformation.

Figure 8-13 Design to Demonstrate Transformed Exceptions



```
pt_shell> set_false_path -through [get_pins u1/Z]

pt_shell> set_multicycle_path 2 \
            -from [get_ports A] -to [get_ports D]

pt_shell> set_multicycle_path 3 -from [get_ports {B C}]
            -through [get_pins u2/Z] -to [get_ports D]
```

The false path through u1/Z sets the two false paths shown in the figure, starting at ports A and B and both ending at port D.

The set_multicycle_path 2 command specifies two paths that start at port A and end at port D: one through u0 and the other through u1. However, the path through u1 is overridden by the set_false_path command, which has a higher priority.

The set_multicycle_path 3 command specifies two paths: one starting at port B and the other starting at port C, with both ending at port D. However, the path starting at port B is overridden by the set_false_path command.

There are many ways you could specify this same set of exceptions. For example, you could simplify the last command as follows:

```
set_multicycle_path 3 \
        -from [get_ports C] -to [get_ports D]
```

After you set the exceptions on a design, you can use the transform_exceptions command to automatically eliminate invalid, redundant, and overridden exceptions and find a simpler, equivalent set of commands. You can then write out the new set of commands with write_sdc or write_script. A simpler set of commands is easier to understand and is easier for back-end tools to use. To write out just the exception-setting commands (without the other timing context commands), use -include {exceptions} in the write_sdc or write_script command.

By default, transform_exceptions transforms all exceptions throughout the design. For example, using the command on the design shown in Figure 8-13 gives the following results:

```
transform_exceptions
...
Transformations Summary   false  multicycle  delay  Total
---------------------------------------------------------
  non-existent paths     0     1          0     1
---------------------------------------------------------
Total transformations                          5

Exceptions Summary       false  multicycle  delay  Total
---------------------------------------------------------
  Modified             0     1          0     1
  Analyzed             1     2          0     3
```

To restrict the scope of the design on which the command operates, you can use the options -from, -through, -to, -rise_from, and so on. In that case, Timing Engine only transforms exception commands that operate on at least one of the specified paths.

To generate a transformation report without actually performing the transformation, use the -dry_run option. The generated report can help you understand why certain paths are being ignored in the original exception-setting commands.

Exception transformation depends on the design context as well as the set of exception-setting commands that have been used. If you change the design, the transformed exceptions might not be equivalent to the original exceptions. To keep a record of the current set of exception-setting commands, use write_sdc or write_script before doing the transformation.

Exception Removal

A major effect of exception transformation is the removal of invalid, redundant, and overridden paths from the original exception-setting commands. To find out the reasons that paths are being eliminated, use the -verbose option in the transform_exceptions command. For example:

```
transform_exceptions -verbose
...
From        Through       To       Setup        Hold
---------------------------------------------------------
*           u1/Z          *        FALSE        FALSE

A           *             D        cycles=2     cycles=0

{ B C }     u2/C          D        cycles=3     cycles=0
```

```
 NON_EXISTENT_PATH
    -from   B
    -through     u2/C
    -to     D
...
```

Each time Timing Engine eliminates paths from the exception-setting commands, it reports one of the following reasons:

- Non-existent path: The path does not exist in the design or has been disabled (for example, by set_disable_timing, set_false_path, or case analysis).
- Overridden: The exception applied to the path is overridden by another exception of higher priority.
- Invalid startpoint: The path startpoint is not a primary input or a clock pin of a sequential element.
- Invalid endpoint: The path endpoint is not a primary output or a data input pin of a sequential element.
- Unconstrained path: The path is not constrained (for example, a primary input with no input delay specified).
- Clock network path: The path is part of a clock network, which by default is unconstrained.
- Single-cycle MCP: The exception is a multicycle path with the number of cycles set to the default value (1 for a setup check or 0 for a hold check).
- Mode analysis: The path has been invalidated by mode analysis (for example, a path that involves writing to RAM with the RAM module set to read mode).
- Exclusive domains: The path crosses between clock domains that have been defined to be exclusive by the set_clock_groups command.

By default, transform_exceptions removes exceptions for all of the reasons listed above. To selectively restrict exception removal to certain reasons, use the -remove_ignored option and specify the types of ignored exceptions to remove. For example:

transform_exceptions -remove_ignored overridden

These are the possible settings for the -remove_ignored option:

- invalid_specifiers: paths specified with an invalid startpoint or endpoint, single-cycle multicycle paths

- no_path: paths that do not exist or are inactive due to case analysis, set_disable_timing, or other reasons
- overridden: paths overridden by other exceptions or mode analysis; and paths between exclusive clock domains
- clock_path: clock network paths
- unconstrained_path: paths that are not constrained

Exception Flattening

The transform_exceptions command performs flattening when you use the -flatten option. Flattening separates multiple -from, -through, or -to objects in a single command into multiple commands. For example:

```
set_false_path -through {u0/Z u1/Z}
transform_exceptions -flatten
...
write_script -include {exceptions}
```

The script written by the write_script command contains the result of flattening the original commands:

```
set_false_path -through [get_pins {u0/Z}]
set_false_path -through [get_pins {u1/Z}]
```

The single set_false_path command is flattened into two commands by separating the two -through objects.

19. Multi-Voltage Designs

The multivoltage design steps are

- Insert buffer-type level shifters.
- Create voltage areas and associate or align the logic hierarchies
- Perform placement in the voltage areas
- Optimize the design

Using Buffer-Type Level Shifters

General Properties of Buffer-Type Level Shifters. buffer-type level-shifter cell is modeled as a buffer with the following properties:

- Two power supplies: a master (or primary) supply and a secondary supply
  Input and output pin voltages that are different

- PVT operating conditions set on the output side of the cell (otherwise, tool creates new operating conditions under-the-hood that are *not* written out and passed to the rest of the flow)

Buffer-type level shifters are used to connect drive and load pins operating at significantly different voltages. The following commands are used to manage the level shifters in your design:

- ***Check level shifters*** – Checks the design for existing level shifters, including enable-type level shifters

- ***Insert level shifters*** – Inserts *only* buffer-type level shifters to accommodate voltage mismatches on nets, according to level shifter strategy and threshold settings.

- ***Level shifter threshold*** – Sets the minimum allowed voltage difference between a source and sink, beyond which voltage differences are adjusted by insertion of level shifters. (0 threshold voltage difference and 0 percentage difference).

Both voltage step-up and voltage step-down buffer-type level shifters are used. For example, in the case where design blocks are working on two different voltage rails, at least two types of level shifters are required:

(i) shifts the level mismatch from the lower voltage to the higher voltage (step-up).
(ii) shifts from the higher voltage to the lower voltage (step-down).

Buffer-Type Level-Shifter Insertion Flow

The flow has the following steps:

1. Check design for buffer-type level shifters and level-shifter violations
2. Insert buffer-type level shifters
3. Import voltage area information from Tool create the voltage areas in Physical Compiler and associate logic modules with the voltage areas.
4. Place cells and level shifters (both types), and optimize the design

These steps are described in greater detail in the following sections.

Optionally Setting Buffer-Type Level-Shifter Strategy

If you do not explicitly set a level-shifter strategy before inserting or checking buffer-type level shifters, the default strategy is -all, which allows the use of both step-up and step-down buffer-type level shifters.

Level-Shifter Threshold

Determining Level-Shifter Threshold

Determine level-shifter thresholds if the target library cells have the following attributes (usually described by equations involving the rail voltages, for example, VDD1+0.5):

- V_{ih} – Lowest input voltage for logic 1
- V_{il} – Highest input voltage for logic 0
- V_{imax} – Maximum input voltage for logic 1
- V_{imin} – Minimum input voltage for logic 0
- V_{oh} – Lowest output voltage for logic 1
- V_{ol} – Highest output voltage for logic 0
- V_{omax} – Maximum output voltage for logic 1
- V_{omin} – Minimum output voltage for logic 0

These voltage parameters are used to determine output and input voltage bands (ranges) for both logic 1 and logic 0. For logic 1, the input voltage band is (V_{imax}-V_{ih}) and the output voltage band is (V_{omax}-V_{oh}). Similarly, for logic 0, the input voltage band is (V_{il}-V_{imin}) and the output voltage band is (V_{ol}-V_{omin}).

Therefore, for any pair of driver and load pins, if the output voltage band of the driver pin is completely overlapped (contained within) the input voltage band of the load pin, no level shifter is required. This criteria holds for both logic 1 and logic 0.

Conversely, a sufficient voltage difference between the driver and load pins exists and buffer-type level-shifter insertion is required if any of the following conditions hold:

- Driver pin V_{omax} > Load pin V_{imax}
- Driver pin V_{omin} < Load pin V_{imin}
- Driver pin V_{oh} < Load pin V_{ih}
- Driver pin V_{ol} > Load pin V_{il}

For any of these four conditions, the driver voltage band does not fall completely within the load voltage band, and buffer-type level shifters are needed.

User-Specified Level-Shifter Threshold

You can explicitly specify the minimum voltage difference beyond which the voltage is to be adjusted by use of a buffer-type level shifter.

Checking the Design for Level Shifters and Level-Shifter Violations

- ***Incorrect operating condition*** – The level shifter does not have the correct annotated operating condition.
- **Wrong level shifter** – The given level shifter cannot be used to shift the voltage levels as needed.

- ***Multiple fanin voltages*** – The fanins of the level shifter are not all at the same voltage level.
- ***Multiple fanout voltages*** – The fanouts of the level shifter are not all at the same voltage level.
- ***Not required*** – The level shifter is not required according to the defined strategy and/or threshold.

All nets that cross boundary hierarchies with different operating conditions are checked for supply voltage, level-shifter strategy, and level-shifter threshold.

Inserting Buffer-Type Level Shifters

Level shifters are inserted on nets where significant voltage differences occur between drive pins and load pins. The voltage differences are determined either automatically by the tool or according to voltage threshold conditions you define. Also, you can accept the default level-shifter strategy, which allows both step-up and step-down buffer-type level shifters.

Although you can insert buffer-type level shifters at a number of points in the logic synthesis-physical synthesis flow, it is recommended that you insert them early in the flow—preferably before logic synthesis.

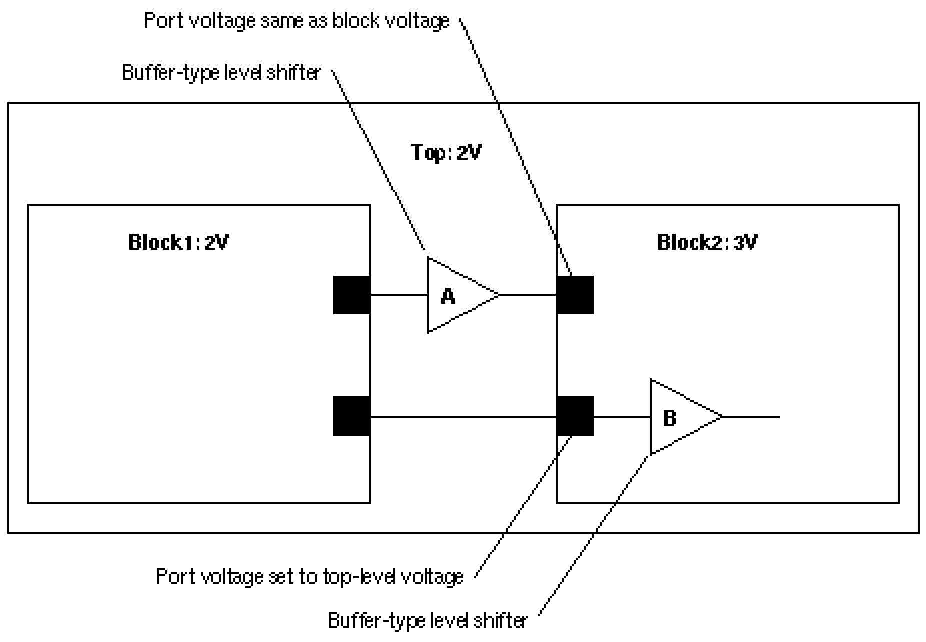
By default, level shifters are not inserted on clock nets. if you want level shifters inserted on clock nets.

The information reported includes library name and type, level-shifter cell name, operating condition(s), input and output voltages, process ID,temperature, and tree type. Also, the number of inserted level shifters is reported.

In general, the buffer-type level shifters are inserted between the blocks. This is the case when the operating voltage is set on the instance and not on any of its ports. However, if an operating voltage different from the block operating voltage is set on a port of the block, the buffer-type level shifter for that port is inserted inside the block.

Figure shows an example of buffer-type level-shifter insertion at both the top level for a block port that has the default block voltage and within a block for a port that is set at the top-level voltage.

Figure Buffer-Type Level Shifter Insertion at the Top Level and Block Level


Port voltage same as block voltage
Buffer-type level shifter
Top: 2V
Block1: 2V
Block2: 3V
A
B
Port voltage set to top-level voltage
Buffer-type level shifter

Placing Level Shifters

Level shifters (both types) can be placed at regular sites (single-height cells) or at special sites (multiheight cells) with multiple power rails. For regular sites, you are responsible for connecting secondary power to the level shifters.

If special sites are needed for level shifters, these sites must be instantiated in the floor plan at locations where these level shifters could be placed. Such sites should usually be defined along the voltage area boundaries. They can overlap with the base site array.

Creating Voltage Areas

A voltage area is a placement area for one or more logic partitions operating at the same voltage. The cells of the logic partition are associated with the partition's voltage area and are constrained to placement within that area.

A nested voltage area is a logical construct. The physical floorplan does not support physically nested voltage areas. Therefore, for logically nested voltage areas, you must define the physical voltage areas using disjoint rectangular and rectilinear shapes in such a way that no physical shape is embedded within or overlapping another physical shape.

In addition, Physical Compiler checks the operating condition voltage of each module against the voltage area specification to ensure that all modules inside the voltage area operate at the correct voltage.

Voltage areas are specified manually or derived automatically by Physical Compiler on the basis of row types and site types. Both rectangular and rectilinear voltage areas are allowed. Voltage areas can consist of disjoint rectilinear and rectangular shapes, but it is recommended that you avoid using disjoint structures – except in the case of logically nested voltage areas – because of possible QoR impact.

The logic hierarchies are assigned to specific voltage areas when you create or update these voltage areas. Care must be taken to align the hierarchies correctly.

Physical Compiler derives a default voltage area for placement of cells not assigned to other voltage areas. The default voltage area can contain top-level leaf cells and buffer-type level shifters, as well as the blocks.

Level shifters and isolation cells can have site heights that differ from the standard cell site height by either integer multiples or noninteger multiples. Physical Compiler supports the placement of both types. Note that you need to define appropriate site arrays for these nonstandard cell heights.

In this design, the floorplan has four voltage areas: Ra, Rb, Rc, and Rd. The default voltage area is Rd. Each voltage area has a corresponding logic partition, A, B, C, or D, that aligns with its respective voltage areas. A hard bound can be defined inside a voltage area, but not across voltage areas. In this example, a hard bound, BB, is defined inside voltage area Ra.

Optionally, guard bands can be specified for some or all voltage areas to prevent shorts. Guard bands define hard keepout margins surrounding the voltage areas in which no cells, including level shifters and isolation cells, can be placed. Guard bands can be defined when creating voltage areas automatically or when updating the voltage areas

. Figure3-2 shows a floorplan similar to Figure3-1 but with guard bands protecting voltage areas Ra and Rc.

Figure 3-1 Multivoltage Design With Several Voltage Areas

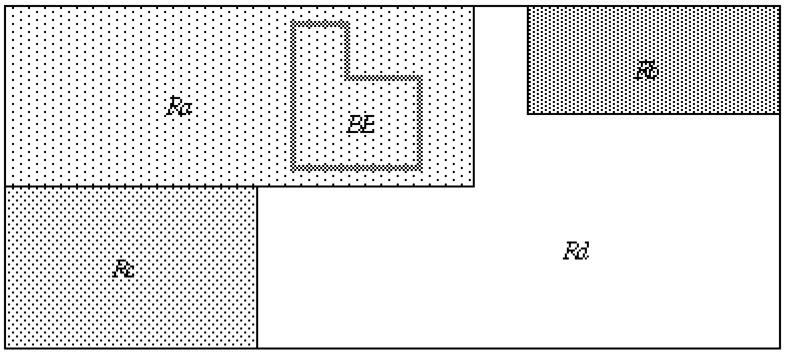
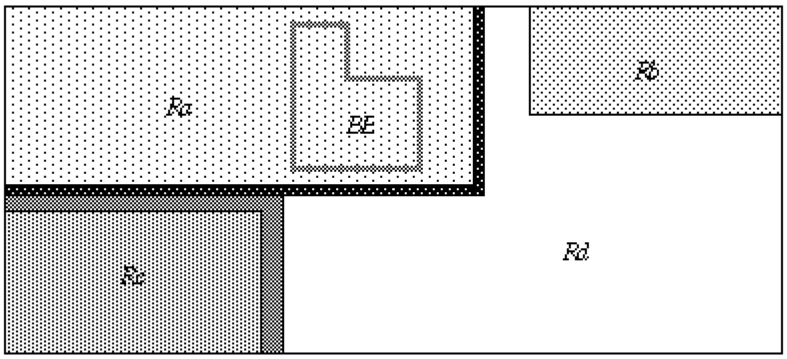


Figure 3-2 Multivoltage Design With Several Voltage Areas and Guard Bands



Creating Voltage Areas Manually

Voltage areas can be manually by specifying the exact coordinates of the voltage area. You also specify the list of cells to be associated with a given voltage area.

Note the following limitations:

- Overlapping voltage areas are not supported.
- All voltage area geometries must be mutually exclusive.

- An error occurs if the voltage area defined when you use the -coordinate option is too small for all the cells to fit into it.

- It is recommended that each derived voltage area not be disjoint. The size of a voltage area must be greater than the sum of the areas of all the placed cells in the hierarchy.

Updating Voltage Areas in a Design

Once you have created a voltage area and associated hierarchical cells with the voltage area, However, you might prefer to change the size of the voltage areas and keep the utilization the same.

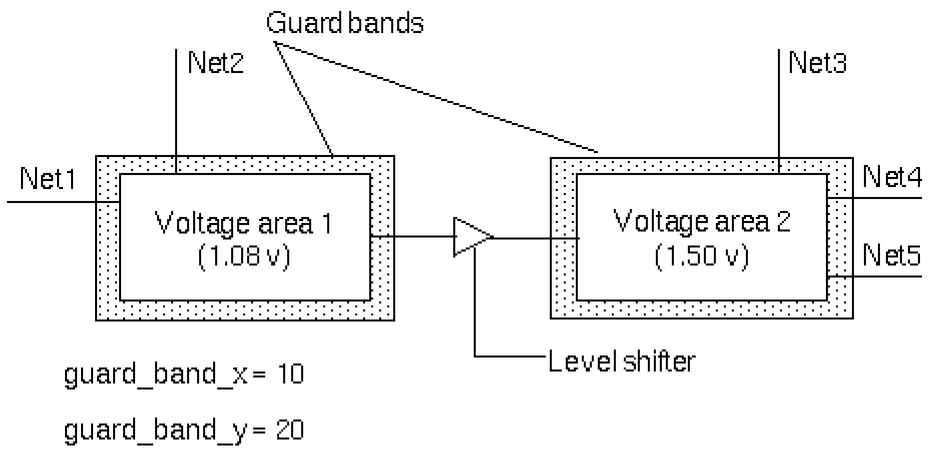
Including Guard Bands With the Voltage Areas

You can use guard bands to ensure that no shorts occur at the boundaries of the voltage areas. Guard bands define hard keepout margins surrounding the voltage areas. No cells, including level shifters and isolation cells, can be placed within the guard band margins.

For example, using guard bands is recommended in the case when the rows are the same for all the voltage areas. The guard bands guarantee the cells in different voltage areas are separated so that power planning does not introduce shorts.

Figure3-3 shows two voltage areas with guard bands of different widths. For both voltage areas, the x-width is 10 and the y-width is 20. Notice the x-width defines the width of the vertical part of the guard band, and the y-width defines the width of the horizontal part.

Figure 3-3 Two Voltage Areas With Guard Bands



When generating guard bands you should be aware of the following conditions:

- If some coordinates of a voltage area are defined outside the core area, the tool does not generate the voltage area and issues warning and error messages.
- If a voltage overlaps an existing one, the tool does not generate the voltage area and issues error messages.
- If a guard band of a voltage area overlaps another voltage area, the command generates the guard band and issues warning message
- The x (y) guard band width applies to both the left and right (top and bottom) side of a voltage area.
- Overlapping guard bands are allowed and do not generate warning messages.
- A guard band that lies outside the core area does not generate a warning message.

Handling Logically Nested Voltage Areas

Voltage areas can be nested with respect to the logic hierarchy, but they cannot be physically nested. You resolve logically nested voltage areas into separate physical areas by using the methods described in the preceding sections to create several voltage areas, some of which would consist of abutted rectangular or rectilinear shapes, as needed. Each abutted shape must not overlap, intersect, or be embedded within any other shape of the voltage area that it belongs to, nor with the shapes of any other voltage area. A composite voltage area can appear to violate this rule, when it is, in fact, a set of properly abutted, separate rectangular or rectilinear shapes.

Figure3-4 shows two examples of logically nested voltage areas and shows how you might resolve them into physically abutted, separate areas. In these two examples, the logically nested voltage areas are labeled VA1 and VA2.

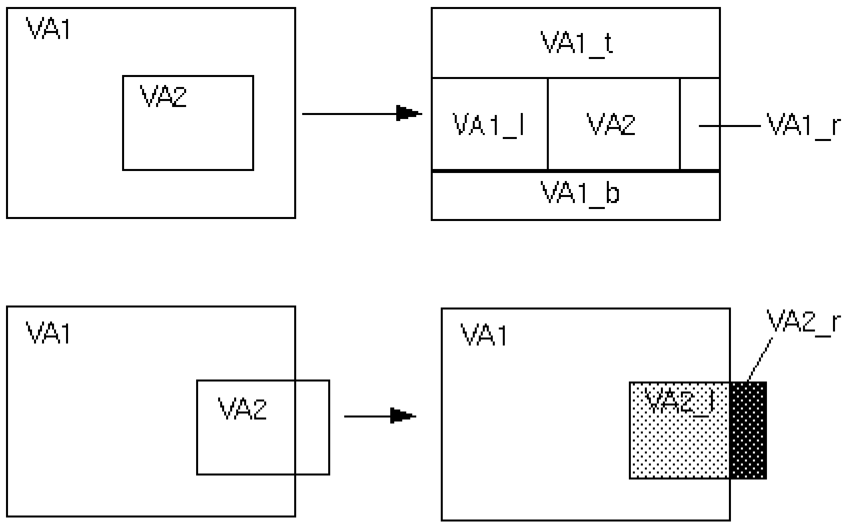
In the first example, you create a voltage area VA1 that consists of the four sequentially abutted rectangles VA1_b, VA1_l, VA1_t, and VA1_r with a central, rectangular gap. Voltage area VA2 is a simple rectangle that "fits" into this gap. The following commands would create the two voltage areas:

```
create_volatage_area -name VA1 -coord {
   10  10 150 30
   10 30 70 80
   10 80 150 120
   140 30 150 80}
create_voltage_area -name VA2 -coord {
   70 30 140 80}
```

In the second example, you create two voltage areas. VA1 is a single, eight-sided rectilinear shape, and VA2 consists of two disjoint rectangles, VA2_l and VA2_r.

These examples do not represent the only way you might resolve the logically nested voltage areas into unnested physical areas

Figure 3-4 Examples of Physically Resolved Nested Voltage Areas



.

Removing Voltage Areas From a Design

If needed you can remove all voltage areas or specified voltage areas from the design. Any move bound belonging to a removed voltage area is also removed. If you remove a logically nested voltage area, the subhierarchy logic inherits the voltage area properties of the parent hierarchy.

Checking Isolation Cells

check for the presence of isolation cells in the mapped logic hierarchies (blocks) or the voltage areas associated with the power domains of the design.

Voltage Area-Aware Capabilities

which means that it can carry out placement and optimization steps in multivoltage designs. Important capabilities include

- Automatic high-fanout synthesis
- Virtual hierarchy routing
- Maximum net length optimization
- Voltage area-based optimization

Buffer trees are built with dedicated subtrees for the fanouts in each voltage area. In particular, automatic high-fanout synthesis (AHFS) selects buffers according to the associated operating condition. Buffers are inserted and placed correctly.

Virtual Hierarchy Routing

The virtual route estimator takes the presence of voltage areas into account

Automatic High-Fanout Synthesis

When this feature is enabled, the virtual route estimator observes the following constraints:

- Virtual routes for nets connecting cells within a voltage area do not go outside the voltage area.
- Virtual routes for nets connecting cells outside a voltage area detour around the voltage area.
- Virtual routes for nets crossing a voltage area do not zig-zag in and out of the voltage area (the number of boundary crossings are minimized).

Also, routing driven synthesis (for example, max_net_length) use voltage area-aware routing.

Multivoltage Relative Placement

You can use relative placement with multivoltage designs. Relative placement is most often applied to physical datapaths. It provides a way for you to create structures in which you specify the relative column and row positions of instances with respect to each other. During placement and legalization, these structures are preserved and the cells in each structure are placed as a single entity.

After you group the relevant cells of a voltage area into an relative placement grouping, you can place them as a single structure within the voltage area. The flow is identical to a typical relative placement flow, and the entire relative placement feature set is available.

However, you should be aware of these limitations:

- An relative placement group should not span voltage areas. If it is necessary to span voltage areas, break the relative placement group into multiple relative placement groups, each contained within its voltage area.

- In situations where a relative placement group is placed very close to a voltage area boundary, legalization might place part of the group outside the multivoltage area. When this happens, the group is broken in a way that respects the voltage area constraint.

Maximum Net Length Optimization

The maximum net length optimization carried out into account the presence of voltage areas, basically routing around them. Routes for nets connecting two cells within a voltage area stay inside the voltage area, and routes for nets crossing into a voltage area do not excessively zig-zag in and out of the voltage area.

Voltage Area-Based Optimization

Cell placement and buffer optimization is one-pass voltage area aware. When the tool buffers a net that crosses voltage areas, the net is divided into multiple segments, each of which is confined within a single voltage area, and buffering is confined to the segments.

Tie-high and tie-low cells are handled consistently. Constant signals are fed to the correct cells for a given voltage area.

Placing and Optimizing the Design

The last step in the multivoltage design flow is to place the cells and optimize the design. The tool places level shifters and isolation cells near the boundaries of defined voltage areas

No buffering is done between the placed level shifter and the edge of the voltage area.

Subsystem Multi-VDDt

Tool provides the following support for multi-VDD designs throughout the Tool design flow:

Common graph and hierarchical common graph

The common graph stores the following information:

- A pointer for each cell to the voltage area index
- A pointer for each cell to the minimum-and-maximum operating condition pair
- The can mapping table specified
- The hierarchical common graph stores the voltage area and operating condition of each

logic hierarchy.

Timer
A timing arc in the cell library contains multiple delay tables, indexed by a set of operating conditions. It is the operating condition of a cell, from the design, that decides which delay table is to be used. Interpolation among different timing tables is not supported.
In the case of multiple NLDM, all .db files need to be searched and preprocessed to extract the matching criteria.

Placement and overlap removal
The boundaries that define a voltage area form a rigid boundary, which works to confine cells to their voltage domains. All cells must stay within their corresponding voltage areas during global placement and overlap removal. To be accepted, any new cells created during these processes must have an appropriate voltage area associated with them before optimization.
Region and plan group constraints are also honored. The placer fails when voltage area constraints are inconsistent with region and plan group constraints. For example, if a cell belongs to Region R1 and voltage area V1 but R1 and V1 have no overlap, it is impossible for the placer to find any legal location for the cell.

Filler cell insertion
Filler cell insertion can be restricted to a named voltage area.

Optimization
Major optimization operations (such as buffer insertion and collapse, and remapping) are not applied across different voltage domains. Sizing and moving must stay within the original voltage area of the cell. Optimization does not add or delete level-shifter cells.

Virtual routing
Virtual routing behaves like global routing. It takes account of voltage areas and constructs the appropriate routing obstructions and detours.

Scan chain
Flip-flops in different voltage domains should belong to different scan groups when level shifting is required. Make sure you define the proper scan groupings, so that they match the voltage domains. Violations are reported by the scan chain subsystem.

Clock tree synthesis
- During clock tree synthesis, Tool queries the overlap removal engine to retrieve the voltage area's physical information. It uses this information to modify groups of sink pins according to voltage areas. That is, clock tree synthesis generates a subtree for each group of sink pins in a voltage area. Then clock tree synthesis assigns the buffers in the subtree to the voltage area. (The overlap removal engine provides the legal location of the buffers, based on voltage area.)

- The same rules for assigning operating conditions to new cells in the common graph are also applied during clock tree synthesis. The timer computes delay, based on the assigned operating conditions. This means that skew computation reflects the delay differences caused by operating voltages.

- Voltage area constraints (when present) are automatically detected by clock tree synthesis. The command usage for performing clock tree synthesis is the same with or without multiple operating voltages.

Global routing

- During global routing, Tool queries the overlap removal engine to retrieve the voltage area boundaries. The voltage area of each cell is also stored. For nets that are in the same voltage area, the global router generates temporary routing guides that work to restrict the routes to the voltage area.

- When a net spans several voltage areas, the global router decomposes it into subnets, where each subnet is in one voltage area. The global router applies the single-voltage area algorithm to each subnet and then routes the remaining “border crossing” parts of the net without temporary route guides.

- Voltage area constraints, if any, are automatically detected by the global router. The command usage for performing global routing is the same with or without multiple operating voltages.

- For consistency with the database, which allows multiple polygons to be defined for one voltage area, the global router recognizes all the polygons that belong to the same voltage area.

- ***Diode insertion***
  The diode insertion operation takes account of voltage areas when inserting diode cells for antenna fixing.

- ***Rail analysis***
  Each voltage has a scaling factor that is specified by the operating condition. The scaling factor is used for rail analysis.

Multivoltage Threshold Filler Cells

For a design with multivoltage threshold cells, the filler cell insertion process must take into account the voltage threshold type of adjacent cells, so that tool can select the appropriate filler cell to insert in the gap between two cells. Specifically, the requirements are these:

- Two adjacent cells of different voltage threshold types must be separated by a special interface cell, known as a filler cell.

- The filler cell to be inserted depends on the voltage threshold types of the two adjacent cells.

- A gap between two adjacent cells of the same voltage threshold type must be filled by a filler cell of that voltage threshold type.

The flow for inserting multivoltage threshold filler cells includes these major steps:

1. Mark the voltage threshold type of all the available standard cells that are relevant.
2. Assign special filler cells for filling the gaps between cells of certain voltage threshold types

3. Insert filler cell

Marking the Voltage Threshold Type for Cells

specify the cells for each voltage threshold type in your design. Attach the voltage threshold type property to each instance of a named cell master or to all the cells in a named library, respectively.

You can also do the following:

- Write the voltage threshold type information for a cell master to an output file.
- Remove the voltage threshold type property from the cell master

Marking All Cells in a Library

You can also do the following:

- Write the voltage threshold type information of each cell in the named cell library to an output file.
- Remove the voltage threshold type property from all cells in the named cell library

Mapping Filler Cells to Voltage Threshold Cells

creates the voltage threshold rule table information, which maps the filler cells to various combinations of different voltage threshold cells. You apply the information in the table to your design.

allows a filler cell to be placed to the left or right of a voltage threshold cell for the following conditions:

- End of a row
- Beginning of a row
- Space between a nonstandard cell (a macro cell) and a standard cell (a voltage threshold cell)
- Space between a placement blockage and a standard cell (a voltage threshold cell)

You can also do the following:

- Write the filler cell list that is stored for the two specified voltage threshold types to an output file
- Remove the filler cell list that is stored to be used for filling between cells of two specified voltage threshold types.

- Write all special filler cells and their voltage threshold type filling information to an output file

Voltage Threshold Rule Table Information

The voltage threshold rule table information allows filler cells to honor one or more voltage threshold rules. Support for multiple rules, in the form of a two-dimensional table, is useful when your design requires that filler cells be inserted between multiple voltage threshold types

Here is a sample two-dimensional table:

	<default>	VTType1	VTType2
<default>	Filler5	Filler3	Filler4
VTType1	X	Filler1,Filler2	Filler2
VTType2	X	X	Filler1

Reference Library Handling

Generally, you have only read permission to the reference library (changes to the reference library should be avoided). However, you should be aware of the following points relating to reference library handling during filler cell insertion.

- Voltage threshold rule information is now stored in the open design view, but for backward compatibility Tool honors voltage threshold rules defined in the reference library. If there are conflicting rules, Tool uses the rules defined in the open design view and issues a warning message to this effect.

- Check whether there are voltage threshold rules stored in the reference library and removes them when found.
- Lists the voltage threshold rules that are defined in the open design view. It will continue to list the voltage threshold rules from each reference library.

Inserting Multivoltage Threshold Filler Cells

1. Specify the filler cell lists to be loaded for use during insertion. You can do one or more of the following:
    - o Enter the names of the master cells.

    - o Specify whether all or only specific voltage threshold filler cells are to be loaded. To specify that all threshold filler cells are to be loaded, select “include all VT filler” (the default).
    - o When left-and-right rules and voltage threshold rules are defined for a design, and the filler cell names in the rules are also entered in either the Master Cell Name(s), Tool inserts filler cells based on left-and-right rules first, followed by cells based on voltage threshold rules, and then fills the remaining empty spaces in the design with regular filler cells listed in the master cell names boxes.

2. Select other options, depending on your requirements. You can
    - Specify that filler cells be placed in named voltage areas only by entering the voltage area names.

    - check that fillers are inserted correctly, based on the specified requirements. check that
      The left-and-right filler cells are inserted as specified
      The filler cells do not overlap with neighboring cells
      The filler cells are not outside of the cell row
      The filler cell's power and ground pins are connected to the specified power and ground nets
      The filler cells are assigned the correct voltage area in multivoltage designs
      The correct voltage threshold filler cells are inserted between voltage threshold cells
      There are no empty spaces big enough for a filler cell
      A filler cell name has the correct hierarchy and user-specified name identifier

Optimization and Voltage Areas

The optimization processes use hierarchy-based voltage areas instead of connectivity-based voltage areas in the hierarchical flow (block level). This allows a hierarchical timing view (HTV) model for a block to have a different voltage area than the top level. SDC constraints must specify an operating condition for the target HTV block. Also, there cannot be multiple voltage areas inside the HTV block.

During optimization, nets crossing different voltage areas are treated differently, depending on whether level-shifter cells or isolation cells are present. For example, consider a net with cell instances A and B that are assigned voltage areas VA_a and VA_b, respectively. During buffer insertion, Tool treats the different cases as follows:

- If neither cell instance A or B is an isolation or level-shifter cell, buffers can be added to VA_a, VA_b, or both.

- If only cell instance A is an isolation or level-shifter cell, buffers can be added only to VA_b.

- If only cell instance B is an isolation or level-shifter cell, buffers can be added only to VA_a.

- If both cell instances A and B are isolation or level-shifter cells, no buffers can be added to the net.

To summarize the rules, when a net is connected to an isolation or level-shifter cell in voltage area V, buffering of the net cannot occur at voltage area V (see Figure 4-3). This rule preserves the integrity of the isolation or level-shifting capability of the netlist. If this rule is violated, the isolation or level-shifting capability might fail. The same rule is applied to any optimization operations that involve netlist changes. If all cell instances of a net belong to an identical voltage area, these rules do not apply.

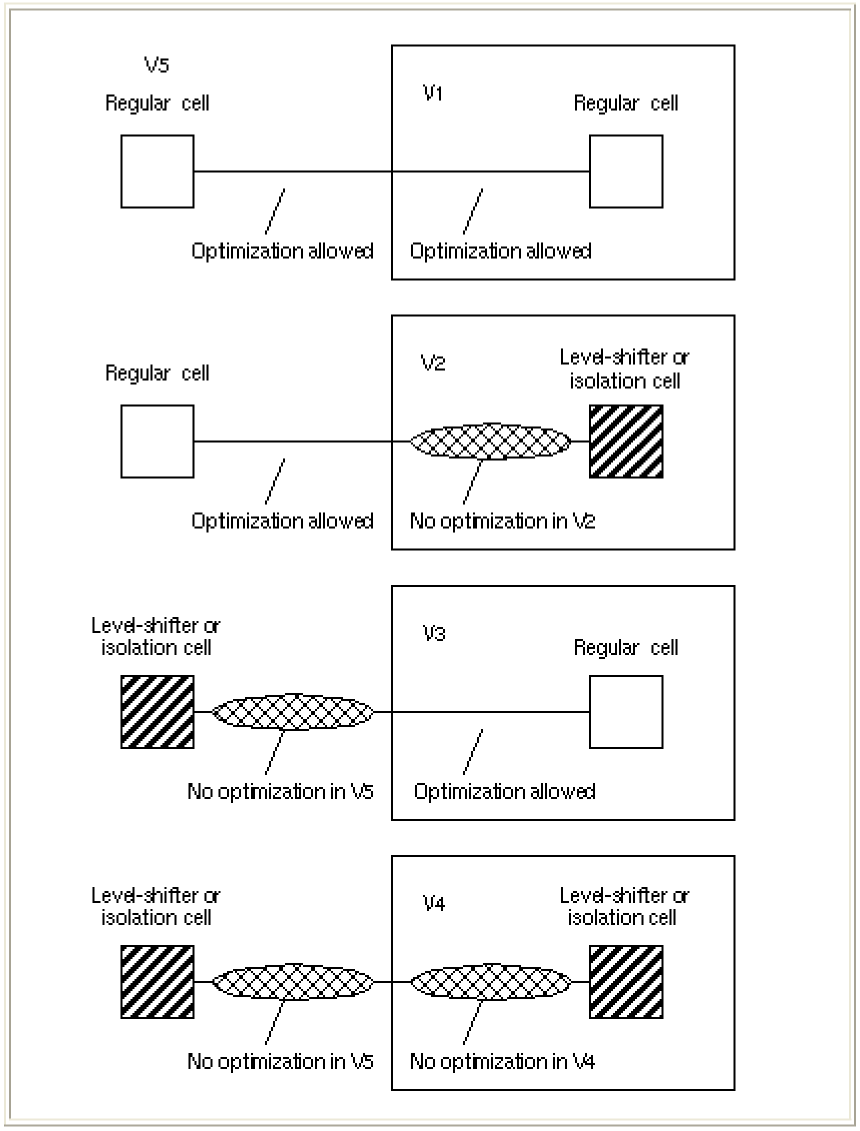
Clock Tree Synthesis and Voltage Areas

When all the flip-flops and roots of a clock net belong to the same voltage area, the clock tree generated during clock tree synthesis is placed in that voltage area. Complications arise when the clock net spans multiple voltage areas. In such a case, Tool attempts to minimize the number of clock nets that cross different voltage areas and tries to keep all sink pins of the same voltage area within one clock subtree. To do so, Tool does the following during clock tree synthesis:

1. Groups the sink pins on the net by voltage area.

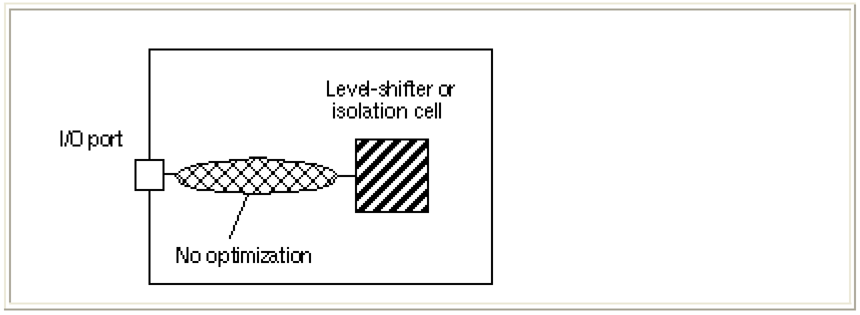
2. Inserts a subtree for each voltage area. From the guide buffer, Tool creates a subtree for each voltage area. All buffers in the subtree are assigned and placed in the same voltage area.
3. Inserts the necessary buffers between the clock root and the subtrees. These buffers are assigned and placed in the same voltage area as the clock root

Figure 4-3 Optimization for Nets Crossing Multiple Voltage Areas



A special case occurs when a level-shifter cell or isolation cell is connected to a boundary port (see Figure 4-4). In this case, the net is not optimized regardless of the existence of voltage areas. Clock tree synthesis also observes the same rule, and buffers will not be added in the net marked "no optimization."

Figure 4-4 No Optimization Between I/O Port and Level-Shifter or Isolation Cell



These steps should minimize the number of clock nets that cross different voltage areas and should allow these nets to be near the root of the clock tree. Note that at step 2, the lower-level subtrees are contained within their voltage area boundaries. The guide buffer on the clock net is not inserted when a level-shifter or isolation cell is already present in the voltage area, as shown in Figure 4-3.

When a buffer is inserted during clock tree synthesis, its voltage area is determined, based on the bottom-up construction principle described in the previous steps. The operating condition of the buffer is also assigned, based on the mapping information.

Clock tree synthesis inserts guide buffers that are based on the operating condition as well as the voltage area, which helps make hierarchy maintenance easier. That is, the hierarchy updating that occurs at the end of the clock tree synthesis operation will take into account the cell's voltage area and operating condition, as assigned by clock tree synthesis. This style of insertion and hierarchy updating is consistent.

For multivoltage designs, clock tree synthesis performs the following steps:

1. Groups the sink pins on the clock net by using a combination of voltage area and operation condition criteria.

2. Inserts a guide buffer for each group. The guide buffer should be assigned the same voltage area and operating condition as the group.
   The guide buffer is also used to create a subtree for each group. All buffers in the subtree are assigned to and placed in the same voltage area and operating condition.

3. Inserts the buffers, as needed, between the clock root and the subtree root of different groups. These buffers are assigned to and placed in the same group as the clock root.

After these steps are completed, the hierarchy is updated, at which time the buffers added during clock tree synthesis are inserted in the proper hierarchy.

For multi-VDD designs, the hierarchy repair process takes the cell voltage areas assigned by clock tree synthesis into account. If a buffer is assigned VA_1 and all its sinks also reside in VA_1, the buffer needs to be pushed down to a lower logic hierarchy that operates at a matching voltage area VA_1. If this adjustment is not performed, the cell might experience inconsistent voltage area and logic hierarchy assignment after clock tree synthesis.

Global Routing Within Voltage Areas

An important multi-VDD routing requirement is to minimize the number of “border crossing” routes. That is, when all the cells of a net are in a single voltage area, it is desirable to keep the routes within the boundaries of the voltage area. This is especially important when buffering decisions are based on the routing results.

Tool meets this requirement by automatically generating routing guides, using voltage area information, in a three-step process:

1. Tool divides the nets that span multi-VDD areas into subnets, so that each subnet connects only one voltage area. This minimizes border crossing.

2. Tool generates temporary routing guides (routing blockages) for each subnet and routes the subnets. This keeps each subnet inside its voltage area boundary.

3. Tool removes the temporary routing guides so that it can complete the routing between all the subnets.

For multi-VDD designs, the global router follows these constraints:

- When all the ports of a net are in the same voltage area, the routing for the net is restricted to the voltage area.
  This is a hard constraint. The global router routes outside the voltage area only when it fails to find a routing solution, due to blockages; it does not go outside because of congestion.
  The main reason for this constraint is to allow better QOR for buffer insertion.

- When a net connects across more than one voltage area, the global router tries to minimize the number of intersections between the routing and the voltage area boundaries.

Power Net Connections

Create and modify power net connections. Because this command takes voltage area constraints into account, it can connect or reconnect power and ground nets that are based on the voltage area of a cell instance.

Often you might generate a Verilog netlist with pins connected to 1’b1 and 1’b0. When reading in a Verilog netlist, Tool requires you to specify global net names for the pins connected to 1’b1 and 1’b0. When you run the aprPGConnect command, which take voltage into account, the connections to 1’b1 and 1’b0 are based on the voltage area of the cell instance. Only cell instances matching the specified voltage area are processed. Cell instances not matching the specified voltage area are ignored.

When a new cell is added to a voltage area, power and ground nets are automatically connected whenever possible. When there is a unique power net in a voltage area, the new cell’s power net is automatically connected. The same rule applies to ground nets. When a given voltage area has multiple power and ground nets, such automatic connection is not possible, and Tool issues a warning. In this case, you must connect the power nets of the new cells that were added during optimization. Automatic connection of power and ground nets is also not performed in non-multi-VDD designs.

Multicorner and Multimode Analysis and Optimization

Multicorner and multimode scenarios provide for analysis and optimization of designs at various process corners, voltages, and temperatures in different functional modes.

This chapter contains the following sections:

- Analyzing With Sequential Multiple Corner Cases
- Analyzing With Sequential Multiple Modes
- Analyzing With Sequential Multiple Corner Cases in Multiple Modes
- Working With Multicorner TLUPlus

Analyzing With Sequential Multiple Corner Cases

In deep submicron processes, the behavior of cell delay and wire delay is different depending on the process variations. As a result, you need to do analysis and optimization at more than the current single minimum corner and single maximum corner. The multicorner capability enables you to analyze and optimize at all these corner cases.

To do multicorner analysis and optimization, you need to

- Set up the environment
- Define the scenarios
- Load the SDC file
- Analyze the timing reports from multiple scenarios
- Determine which scenario to optimize

For each corner case, you need to provide one or more separate .db files with process, temperature, and voltage information.
You also need to provide one SDC constraints file (.sdc) that applies for all the process corner cases.

Creating the Scenarios for Multicorner Analysis
Use the scaling options to define the cell delay and wire delay scaling factors.

Note:

Consider the following when using the parasitic scaling factor

- When you specify both the parasitic multipliers and the parasitic scaling factor ,tool scales parasitics based only on the parasitic scaling factor.
- When you specify both the parasitic scaling options and the wire delay scaling options, tool calculates the initial delay based on the scaled parasitics and then scales the resultant delay by the delay scaling factor.

20. Physical Design Objective Type of Questions and Answers

- **1) Chip utilization depends on ___.**

a. Only on standard cells
b. Standard cells and macros
c. Only on macros
d. Standard cells macros and IO pads

- **2) In Soft blockages ____ cells are placed.**

a. Only sequential cells
b. No cells
c. Only Buffers and Inverters
d. Any cells

- **3) Why we have to remove scan chains before placement?**

a. Because scan chains are group of flip flop
b. It does not have timing critical path
c. It is series of flip flop connected in FIFO
d. None

- **4) Delay between shortest path and longest path in the clock is called ____.**

a. Useful skew
b. Local skew
c. Global skew
d. Slack

- **5) Cross talk can be avoided by ___.**

a. Decreasing the spacing between the metal layers
b. Shielding the nets
c. Using lower metal layers
d. Using long nets

- **6) Prerouting means routing of _____.**

a. Clock nets
b. Signal nets
c. IO nets
d. PG nets

- **7) Which of the following metal layer has Maximum resistance?**

a. Metal1
b. Metal2
c. Metal3
d. Metal4

- **8) What is the goal of CTS?**

a. Minimum IR Drop
b. Minimum EM
c. Minimum Skew
d. Minimum Slack

- **9) Usually Hold is fixed ___.**

a. Before Placement
b. After Placement
c. Before CTS
d. After CTS

- **10) To achieve better timing ____ cells are placed in the critical path.**

a. HVT
b. LVT
c. RVT
d. SVT

- **11) Leakage power is inversely proportional to ___.**

a. Frequency
b. Load Capacitance
c. Supply voltage
d. Threshold Voltage

- **12) Filler cells are added ___.**

a. Before Placement of std cells
b. After Placement of Std Cells
c. Before Floor planning
d. Before Detail Routing

- **13) Search and Repair is used for ___.**

a. Reducing IR Drop
b. Reducing DRC
c. Reducing EM violations
d. None

- **14) Maximum current density of a metal is available in ___.**

a. .lib
b. .v
c. .tf
d. .sdc

- **15) More IR drop is due to ___.**

a. Increase in metal width
b. Increase in metal length
c. Decrease in metal length
d. Lot of metal layers

- **16) The minimum height and width a cell can occupy in the design is called as ___.**

a. Unit Tile cell
b. Multi heighten cell
c. LVT cell
d. HVT cell

- **17) CRPR stands for ___.**

a. Cell Convergence Pessimism Removal
b. Cell Convergence Preset Removal
c. Clock Convergence Pessimism Removal
d. Clock Convergence Preset Removal

- **18) In OCV timing check, for setup time, ___.**

a. Max delay is used for launch path and Min delay for capture path
b. Min delay is used for launch path and Max delay for capture path
c. Both Max delay is used for launch and Capture path
d. Both Min delay is used for both Capture and Launch paths

- **19) "Total metal area and(or) perimeter of conducting layer / gate to gate area" is called ___.**

a. Utilization	c. OCV
b. Aspect Ratio	d. Antenna Ratio

- **20) The Solution for Antenna effect is ___.**

a. Diode insertion	c. Buffer insertion
b. Shielding	d. Double spacing

- **21) To avoid cross talk, the shielded net is usually connected to ___.**

a. VDD	c. Both VDD and VSS
b. VSS	d. Clock

- **22) If the data is faster than the clock in Reg to Reg path ___ violation may come.**

a. Setup	c. Both
b. Hold	d. None

- **23) Hold violations are preferred to fix ___.**

a. Before placement	c. Before CTS
b. After placement	d. After CTS

- **24) Which of the following is not present in SDC ___?**

a. Max tran	c. Max fanout
b. Max cap	d. Max current density

- **25) Timing sanity check means (with respect to PD)___.**

a. Checking timing of routed design with out net delays
b. Checking Timing of placed design with net delays
c. Checking Timing of unplaced design without net delays
d. Checking Timing of routed design with net delays

- **26) Which of the following is having highest priority at final stage (post routed) of the design ___?**

a. Setup violation	c. Skew
b. Hold violation	d. None

- **27) Which of the following is best suited for CTS?**

a. CLKBUF	c. INV
b. BUF	d. CLKINV

- **28) Max voltage drop will be there at(with out macros) ___.**

a. Left and Right sides
b. Bottom and Top sides
c. Middle
d. None

- **29) Which of the following is preferred while placing macros ___?**

a. Macros placed center of the die
b. Macros placed left and right side of die
c. Macros placed bottom and top sides of die
d. Macros placed based on connectivity of the I/O

- **30) Routing congestion can be avoided by ___.**

a. placing cells closer
b. Placing cells at corners
c. Distributing cells
d. None

- **31) Pitch of the wire is ___.**

a. Min width
b. Min spacing
c. Min width - min spacing
d. Min width + min spacing

- **32) In Physical Design following step is not there ___.**

a. Floorplaning
b. Placement
c. Design Synthesis
d. CTS

- **33) In technology file if 7 metals are there then which metals you will use for power?**

a. Metal1 and metal2
b. Metal3 and metal4
c. Metal5 and metal6
d. Metal6 and metal7

- **34) If metal6 and metal7 are used for the power in 7 metal layer process design then which metals you will use for clock ?**

a. Metal1 and metal2
b. Metal3 and metal4
c. Metal4 and metal5
d. Metal6 and metal7

- **35) In a reg to reg timing path Tclocktoq delay is 0.5ns and TCombo delay is 5ns and Tsetup is 0.5ns then the clock period should be ___.**

a. 1ns
b. 3ns
c. 5ns
d. 6ns

- **36) Difference between Clock buff/inverters and normal buff/inverters is __.**

a. Clock buff/inverters are faster than normal buff/inverters
b. Clock buff/inverters are slower than normal buff/inverters
c. Clock buff/inverters are having equal rise and fall times with high drive strengths compare to normal buff/inverters
d. Normal buff/inverters are having equal rise and fall times with high drive strengths compare to Clock buff/inverters.

- **37) Which configuration is more preferred during floorplaning ?**

a. Double back with flipped rows
b. Double back with non flipped rows

c. With channel spacing between rows and no double back
d. With channel spacing between rows and double back

- **38) What is the effect of high drive strength buffer when added in long net ?**

a. Delay on the net increases
b. Capacitance on the net increases
c. Delay on the net decreases
d. Resistance on the net increases

.

- **39) Delay of a cell depends on which factors ?**

a. Output transition and input load
b. Input transition and Output load
c. Input transition and Output transition
d. Input load and Output Load.

- **40) After the final routing the violations in the design ___.**

a. There can be no setup, no hold violations
b. There can be only setup violation but no hold
c. There can be only hold violation not Setup violation
d. There can be both violations.

- **41) Utilisation of the chip after placement optimisation will be ___.**

a. Constant
b. Decrease
c. Increase
d. None of the above

- **42) What is routing congestion in the design?**

a. Ratio of required routing tracks to available routing tracks
b. Ratio of available routing tracks to required routing tracks
c. Depends on the routing layers available
d. None of the above

- **43) What are preroutes in your design?**

a. Power routing
b. Signal routing
c. Power and Signal routing
d. None of the above.

- **44) Clock tree doesn't contain following cell ___.**

a. Clock buffer
b. Clock Inverter
c. AOI cell
d. None of the above

- **Answers:**

1)b	2)c	3)b	4)c	5)b	6)d	7)a	8)c	9)d	10)b	11)d	12)d
13)b	14)c	15)b	16)a	17)c	18)a	19)d	20)a	21)b	22)b	23)d	24)d
25)c	26)b	27)a	28)c	29)d	30)c	31)d	32)c	33)d	34)c	35)d	36)c
37)a	38)c	39)b	40)d	41)c	42)a	43)a	44)c				

What is the most challenging job in P&R flow?

- -It may be power planning- because you found more IR drop
- -It may be low power target-because you had more dynamic and leakage power
- -It may be macro placement-because it had more connection with standard cells or macros
- -It may be CTS-because you needed to handle multiple clocks and clock domain crossings
- -It may be timing-because sizing cells in ECO flow is not meeting timing
- -It may be library preparation-because you found some inconsistency in libraries.
- -It may be DRC-because you faced thousands of violations

21. ASIC Design Check List

Silicon Process and Library Characteristics

1. What exact process are you using?
2. How many layers can be used for this design?
3. Are the Cross talk Noise constraints, Xtalk Analysis configuration, Cell EM & Wire EM available?

Design Characteristics

1. What is the design application?
2. Number of cells (placeable objects)?
3. Is the design Verilog or VHDL?
4. Is the netlist flat or hierarchical?
5. Is there RTL available?
6. Is there any datapath logic using special datapath tools?
7. Is the DFT to be considered?
8. Can scan chains be reordered?
9. Is memory BIST, boundary scan used on this design?
10. Are static timing analysis constraints available in SDC format?

Clock Characteristics

1. How many clock domains are in the design?
2. What are the clock frequencies?
3. Is there a target clock skew, latency or other clock requirements?
4. Does the design have a PLL?
5. If so, is it used to remove clock latency?
6. Is there any I/O cell in the feedback path?
7. Is the PLL used for frequency multipliers?
8. Are there derived clocks or complex clock generation circuitry?
9. Are there any gated clocks?
10. If yes, do they use simple gating elements?
11. Is the gate clock used for timing or power?
12. For gated clocks, can the gating elements be sized for timing?
13. Are you muxing in a test clock or using a JTAG clock?
14. Available cells for clock tree?
15. Are there any special clock repeaters in the library?

16. Are there any EM, slew or capacitance limits on these repeaters?
17. How many drive strengths are available in the standard buffers and inverters?
18. Do any of the buffers have balanced rise and fall delays?
19. Any there special requirements for clock distribution?
20. Will the clock tree be shielded? If so, what are the shielding requirements?

Floorplan and Package Characteristics

1. Target die area?
2. Does the area estimate include power/signal routing?
3. What gates/mm2 has been assumed?
4. Number of routing layers?
5. Any special power routing requirements?
6. Number of digital I/O pins/pads?
7. Number of analog signal pins/pads?
8. Number of power/ground pins/pads?
9. Total number of pins/pads and Location?
10. Will this chip use a wire bond package?
11. Will this chip use a flip-chip package?
12. If Yes, is it I/O bump pitch? Rows of bumps? Bump allocation?Bump pad layout guide?
13. Have you already done floor-planning for this design?
14. If yes, is conformance to the existing floor-plan required?
15. What is the target die size?
16. What is the expected utilization?
17. Please draw the overall floorplan ?
18. Is there an existing floorplan available in DEF?
19. What are the number and type of macros (memory, PLL, etc.)?
20. Are there any analog blocks in the design?
21. What kind of packaging is used? Flipchip?
22. Are the I/Os periphery I/O or area I/O?
23. How many I/Os?
24. Is the design pad limited?
25. Power planning and Power analysis for this design?
26. Are layout databases available for hard macros ?
27. Timing analysis and correlation?
28. Physical verification ?

Data Input

1. Library information for new library
2. .lib for timing information
3. GDSII or LEF for library cells including any RAMs
4. RTL in Verilog/VHDL format
5. Number of logical blocks in the RTL
6. Constraints for the block in SDC
7. Floorplan information in DEF
8. I/O pin location
9. Macro locations

22. Companies Interview Questions

Intel

1. Why power stripes routed in the top metal layers?
2. Why do you use alternate routing approach HVH/VHV (Horizontal-Vertical-Horizontal/ Vertical-Horizontal-Vertical)?
3. What are several factors to improve propagation delay of standard cell?
4. How do you compute net delay (interconnect delay) / decode RC values present in tech file?
5. What are various ways of timing optimization in synthesis tools?
6. What would you do in order not to use certain cells from the library?
7. How delays are characterized using WLM (Wire Load Model)?
8. What are various techniques to resolve congestion/noise?
9. Let's say there enough routing resources available, timing is fine, can you increase clock buffers in clock network? If so will there be any impact on other parameters?
10. How do you optimize skew/insertion delays in CTS (Clock Tree Synthesis)?
11. What are pros/cons of latch/FF (Flip Flop)?

12. How you go about fixing timing violations for latch- latch paths?

13. As an engineer, let's say your manager comes to you and asks for next project die size estimation/projection, giving data on RTL size, performance requirements. How do you go about the figuring out and come up with die size considering physical aspects?

14. How will you design inserting voltage island scheme between macro pins crossing core and are at different power wells? What is the optimal resource solution?

15. What are various formal verification issues you faced and how did you resolve?

16. How do you calculate maximum frequency given setup, hold, clock and clock skew?

17. What are effects of meta-stability?

ST Microelectronics

1 What are the challenges you faced in place and route, FV (Formal Verification), ECO (Engineering Change Order) areas?
2 How long the design cycle for your designs?
3 What part are your areas of interest in physical design?
4 Explain ECO (Engineering Change Order) methodology.
5 Explain CTS (Clock Tree Synthesis) flow.

Answer: Clock Tree Synthesis

6 What kind of routing issues you faced?

7 **How does STA (Static Timing Analysis) in OCV (On Chip Variation) conditions done? How do you set OCV (On Chip Variation) in IC compiler? How is timing correlation done before and after place and route?**
8 If there are too many pins of the logic cells in one place within core, what kind of issues would you face and how will you resolve?
9 Define hash/ @array in perl.
10 Using TCL (Tool Command Language, Tickle) how do you set variables?
11 What is ICC (IC Compiler) command for setting derate factor/ command to perform physical synthesis?
12 What are nanoroute options for search and repair?
13 What were your design skew/insertion delay targets?
14 How is IR drop analysis done? What are various statistics available in reports?
15 Explain pin density/ cell density issues, hotspots?
16 How will you relate routing grid with manufacturing grid and judge if the routing grid is set correctly?
17 What is the command for setting multi cycle path?
18 If hold violation exists in design, is it OK to sign off design? If not, why?

Texas Instruments (TI)

1 How are timing constraints developed?
2 Explain timing closure flow/methodology/issues/fixes.
3 Explain SDF (Standard Delay Format) back annotation/ SPEF (Standard Parasitic Exchange Format) timing correlation flow.
4 Given a timing path in multi-mode multi-corner, how is STA (Static Timing Analysis) performed in order to meet timing in both modes and corners, how are PVT (Process-Voltage-Temperature)/derate factors decided and set in the Primetime flow?
5 With respect to clock gate, what are various issues you faced at various stages in the physical design flow?
6 What are synthesis strategies to optimize timing?
7 Explain ECO (Engineering Change Order) implementation flow. Given post routed database and functional fixes, how will you take it to implement ECO (Engineering Change Order) and what physical and functional checks you need to perform?

Qualcomm

3 In building the timing constraints, do you need to constrain all IO (Input-Output) ports?
4 Can a single port have multi-clocked? How do you set delays for such ports?
5 How is scan DEF (Design Exchange Format) generated?
6 What is purpose of lockup latch in scan chain?
7 How do you set inter clock uncertainty?
8 In DC (Design Compiler), how do you constrain clocks, IO (Input-Output) ports, maxcap, max tran?
9 What are differences in clock constraints from pre CTS (Clock Tree Synthesis) to post CTS (Clock Tree Synthesis)?
10 How is clock gating done?

Answer: Clock Gating

11 what constraints you add in CTS (Clock Tree Synthesis) for clock gates?
12 What is tradeoff between dynamic power (current) and leakage power (current)?

Answer:
Leakage Power Trends
Dynamic Power

13 How do you reduce standby (leakage) power?

Answer: Low Power Design Techniques

14 Explain top level pin placement flow? What are parameters to decide?
15 Given block level netlists, timing constraints, libraries, macro LEFs (Layout Exchange Format/Library Exchange Format), how will you start floor planning?
16 With net length of 1000um how will you compute RC values, using equations/tech file info?
17 What do noise reports represent?
18 What does glitch reports contain?
19 What are CTS (Clock Tree Synthesis) steps in IC compiler?
20 What do clock constraints file contain?
21 How to analyze clock tree reports?
22 What do IR drop Voltagestorm reports represent?
23 Where /when do you use DeCAP (Decoupling Capacitor) cells?
24 What are various power reduction techniques?

Answer: Low Power Design Techniques

Hughes Networks

3 What is setup/hold? What are setup and hold time impacts on timing? How will you fix setup and hold violations?
4 Explain function of Muxed FF (Multiplexed Flip Flop) /scan FF (Scal Flip Flop).
5 What are tested in DFT (Design for Testability)?
6 In equivalence checking, how do you handle scanen signal?
7 In terms of CMOS (Complimentary Metal Oxide Semiconductor), explain physical parameters that affect the propagation delay?
8 Why is power signal routed in top metal layers?

Top Metal Layers has Less Resistance and High Current Density.

Avago Technologies (former HP group)

1 How do you minimize clock skew/ balance clock tree?
2 Given 11 minterms and asked to derive the logic function.
3 Given C1= 10pf, C2=1pf connected in series with a switch in between, at t=0 switch is open and one end having 5v and other end zero voltage; compute the voltage across C2 when the switch is closed?
4 Explain the modes of operation of CMOS (Complimentary Metal Oxide Semiconductor) inverter? Show IO (Input-Output) characteristics curve.
5 Implement a ring oscillator.

6 How to slow down ring oscillator?

Hynix Semiconductor

1 How do you optimize power at various stages in the physical design flow?
2 What timing optimization strategies you employ in pre-layout /post-layout stages?
3 What are process technology challenges in physical design?
4 Design divide by 2, divide by 3, and divide by 1.5 counters. Draw timing diagrams.
5 What are multi-cycle paths, false paths? How to resolve multi-cycle and false paths?
6 Given a flop to flop path with combo delay in between and output of the second flop fed back to combo logic. Which path is fastest path to have hold violation and how will you resolve?
7 What are RTL (Register Transfer Level) coding styles to adapt to yield optimal backend design?
8 Draw timing diagrams to represent the propagation delay, set up, hold, recovery, removal, minimum pulse width.