

Clock Tree Synthesis

- [The Clock Tree Synthesis Engines](#)
- [Overview](#)
- [Flow and Quick Start](#)
 - [Quick Start Example](#)
- [Early Clock Flow](#)
 - [Use Model](#)
- [Configuration and Method](#)
 - [Properties System](#)
 - [Route Types](#)
 - [Library Cells](#)
 - [Transition Target](#)
 - [Skew Target](#)
 - [Creating the Clock Tree Specification](#)
 - [Configuration Check](#)
 - [CCOpt Effort](#)
 - [Create Preferred Cells Stripes to Control IR Drop](#)
 - [EM Avoidance Function](#)
 - [Common Specification Modifications](#)
 - [Restricting CCOpt Skew Scheduling](#)
 - [Method](#)
- [Concepts and Clock Tree Specification](#)
 - [Graph-Based CTS](#)
 - [Clock Trees and Skew Groups](#)
 - [Automatic Clock Tree Specification Creation](#)
 - [Manual Setup and Adjustment of the Clock Specification](#)
 - [Deleting the Clock Tree Specification](#)
 - [Chains](#)
- [Reporting](#)
 - [Get Commands](#)
 - [Skew Groups](#)
 - [Clock Trees](#)
 - [Clock Tree Network Structure](#)
 - [Timing Data for Reports](#)
 - [Worst Chain](#)
 - [Halo Violations](#)
 - [Cell Name Information](#)
 - [Clock Tree Convergence](#)
- [CCOpt Clock Tree Debugger](#)
 - [Launching the CCOpt CTD](#)
 - [CCOpt CTD Interface](#)
 - [Key Features of the CTD](#)
- [Additional Topics](#)
 - [Source Latency Update](#)

- [Cell Halos](#)
- [Power Management](#)
- [Shared Clock and Data Concerns](#)
- [CCOpt Property System](#)
 - [Setting Properties](#)
 - [Getting Properties](#)
- [Migrating from FE-CTS](#)
 - [Specification Translation](#)
 - [Concept Mapping](#)

The Clock Tree Synthesis Engines

The Innovus™ Implementation System (Innovus) product family offers the following types of clock tree synthesis (CTS):

- **CCOpt** – Full Clock Concurrent Optimization (CCOpt). The command [ccopt_design](#) , without the `-cts` parameter, is used to invoke this.
- **CCOpt-CTS** – Global skew balanced CTS using the CCOpt CTS engine. In the 14.2 and later versions of the software, this is the default CTS engine when the [clockDesign](#) command is invoked. Existing users of CCOpt-CTS are familiar with the [ccopt_design -cts](#) command. CCOpt-CTS can be used with a clock tree specification generated from SDC timing constraints, or for backward compatibility from a FE-CTS clock specification.

The table below summarizes ways in which the different CTS engines can be invoked. The [clockDesign](#) command's default behavior is changed in the 14.2 release of the software. By default, the 14.2 and later versions of the software will use the CCOpt-CTS engine.

Command	setCTSMODE -engine Setting	Engine Used
ccopt_design	ignored	CCOpt: Full clock concurrent optimization is performed. Use create_ccopt_clock_tree_spec to create a clock tree specification from SDC constraints.
<code>ccopt_design -cts</code>	ignored	CCOpt-CTS : Global skew balancing using CCOpt-CTS engine. Use create_ccopt_clock_tree_spec to create a clock tree specification from SDC constraints.
<code>ccopt_design -cts -ckSpec</code>	ignored	CCOpt-CTS : Global skew balancing using CCOpt-CTS engine driven by a FE-CTS specification for backward compatibility.
clockDesign	auto (default)	If an FE-CTS specification is not loaded then the engine used is as per the <code>ccopt</code> engine setting. If an FE-CTS specification is loaded then the engine used is as per <code>ccopt_from_edt_spec</code> engine setting.
<code>clockDesign</code>	<code>ccopt</code>	Same as <code>ccopt_design -cts</code> , but automatically invokes create_ccopt_clock_tree_spec first to

create a clock tree specification from SDC constraints.

`clockDesign ccopt_from_edi_spec` Same as `ccopt_design -cts -ckSpec`.

To obtain the `clockDesign` behavior of the software version 14.1 and earlier, use the following commands:

```
setCTSMode -engine ck  
clockDesign ...
```

Icon

The `setCTSMode -engine ck` option is part of a limited-access feature in this release. It is enabled by a variable specified using the [setLimitedAccessFeature](#) command. To use this feature, contact your Cadence representative to explain your usage requirements.

Overview

CCOpt-CTS is the CTS engine underpinning CCOpt. The benefits of CCOpt-CTS include the following:

- Automatic creation of the clock tree specification from multi-mode SDC constraints via the `create_ccopt_clock_tree_spec` command. The skew group data model permits complex balancing relationships to be captured.
- The use of a graph-based algorithm avoids the need for sequential CTS between modes and avoids over balancing in complex multi-mode clock networks.
- Graphical CCOpt Clock Tree Debugger to visualize and debug clock trees.

For an introduction to CCOpt concepts including clock trees and skew groups, see the [Concepts and Clock Tree Specification](#) section.

CCOpt extends CCOpt-CTS to replace traditional global skew balancing with a combination of CTS, timing driven useful skew, and datapath optimization. In traditional CTS flows, an ideal clock model is used before CTS to simplify clock timing analysis. With the ideal clock model, launch and capture clock paths are assumed to have the same delay. After CTS, the ideal clock model is replaced by a propagated clock model that takes account of actual delays along clock launch and capture paths. In traditional CTS, global skew balancing attempts to make the propagated clock timing match the ideal mode clock timing by balancing the insertion delay (clock latency) between all sinks. However, a number of factors combine such that skew balancing does not lead to timing closure. These include:

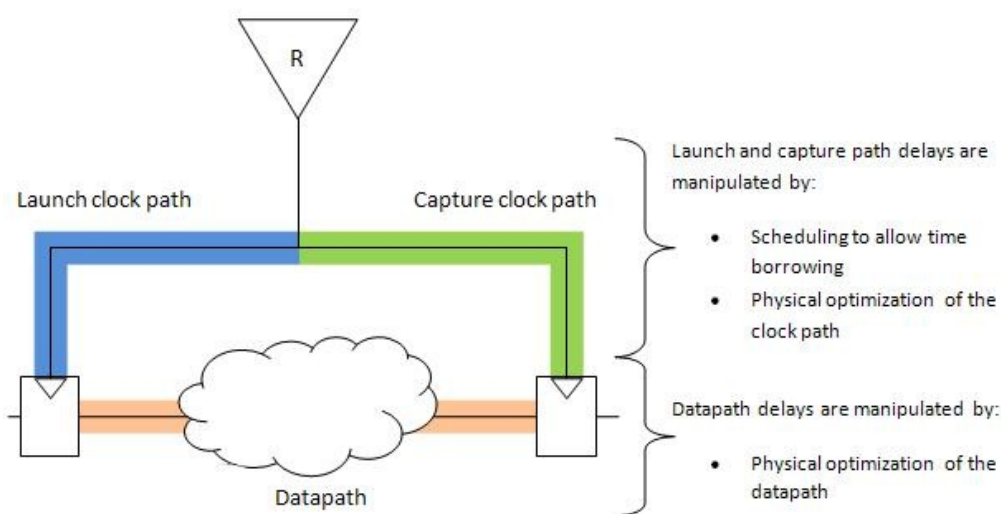
- **OCV** – On-chip variation means that skew, measured using a single metric such as the ‘late’ configuration of a delay corner, no longer directly corresponds to timing impact because launch and capture paths have differing timing derates. In addition, Common Path Pessimism Removal (CPPR) and per-library cell timing derates mean that it is not possible to accurately estimate clock or datapath timing without synthesizing a clock tree. Advanced OCV (AOCV) further complicates this by adding path and bounding box dependent factors.
- **Clock gating** – Clock gating uses datapath signals to inhibit or permit clock edges to propagate from a clock source to clock sinks. The clock arrival time at a clock gating cell is unknown prior to CTS and this arrival time determines the required time for the datapath control signal to reach the clock gating cell enable input. Therefore, the setup slack at a clock gating enable input is hard to predict preCTS. In addition, clock gating cells have an earlier clock arrival time than regular sinks and are, therefore, often timing critical. Typically, the fan-in registers controlling clock gating may need to have an earlier clock arrival time than regular sinks in order to avoid a clock gating slack violation – which means the fan-in

registers need to be skewed early.

- **Unequal datapath delays** – Front end logic synthesis will attempt to ensure that logic between registers is roughly delay-balanced to optimize the target clock frequency. However, with wire delay dominating many datapath stages, it is likely that after placement and preCTS optimization there will exist some combinational paths with unavoidably longer delays than others. Useful skew clock scheduling permits slack to be moved between register stages to increase clock frequency. In contrast, global skew balancing is independent of timing slack. In addition, CCOpt useful skew scheduling can avoid unnecessary balancing of sinks where there is excess slack in order to reduce clock area and clock power.

CCOpt treats clock launch, clock capture, and datapath delays as flexible parameters that can be manipulated to optimize timing. This is illustrated below.

Manipulating Clock Delays and Logic Delays



At each clock sink (flip-flop) in the design, CCOpt can adjust both datapath and clock delays in order to improve negative setup timing slack – specifically the high-effort path group(s) WNS. This is performed using the propagated clock timing model at all times. For detailed information about the concept of moving slack between register stages and the concept of worst chains, see the [Chains](#) section.

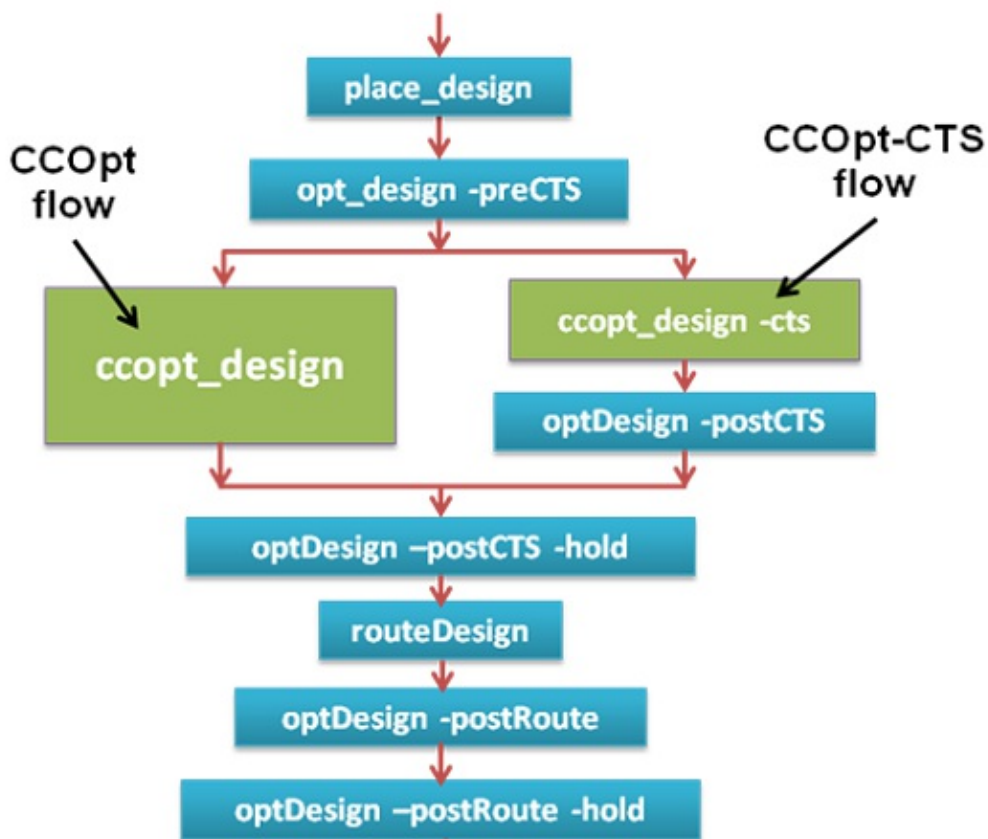
Flow and Quick Start

Icon

If you are using a FE-CTS clock specification, this guide assumes the use of either `cchopt_design` or `cchopt_design -cts`. For information on using `cchopt_design -cts -ckSpec`, including the default behavior of `clockDesign` with a FE-CTS clock specification loaded, see the [Migrating from FE-CTS](#) section.

The diagram below highlights the CCOpt and CCOpt-CTS steps as part of the standard Innovus block implementation flow. In the CCOpt flow, postCTS optimization is not required because `cchopt_design` includes postCTS style optimization both as part of clock concurrent optimization and as a further final internal optimization step.

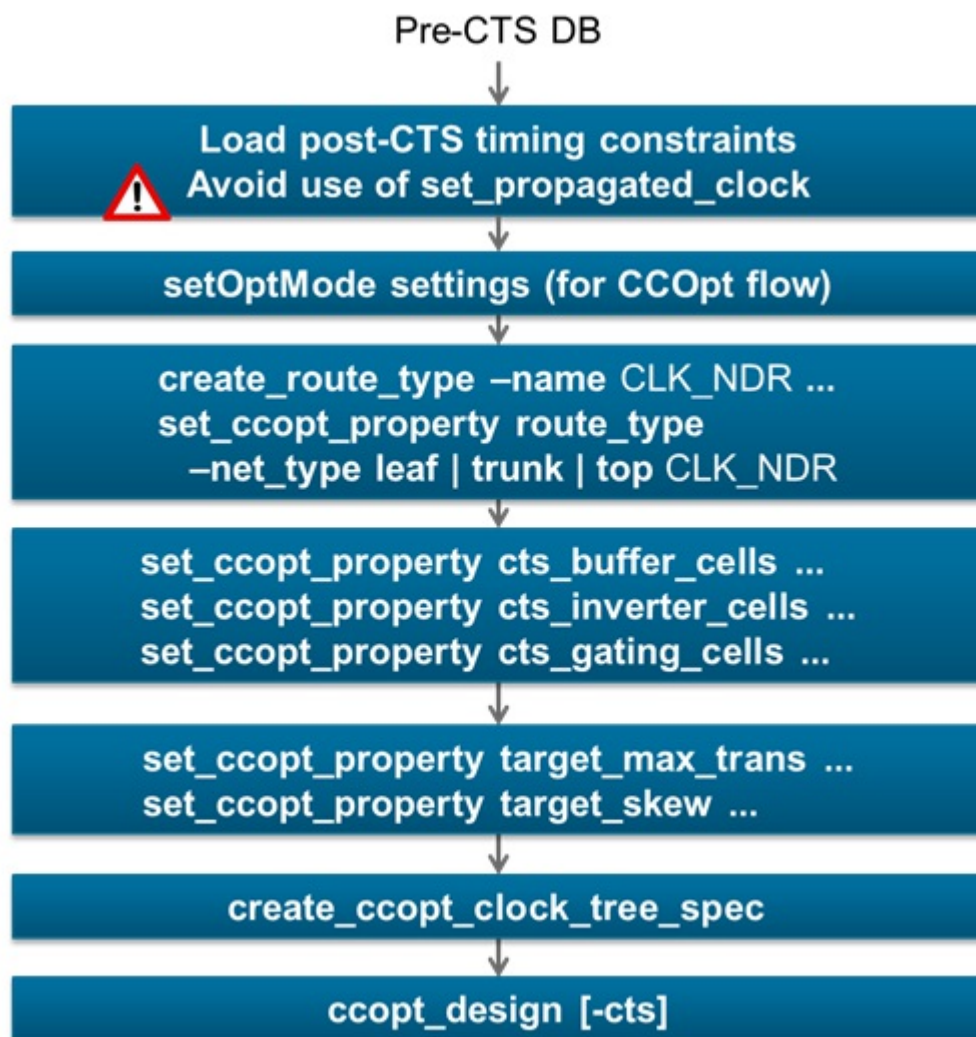
CCOpt and CCOpt-CTS in the Design Flow



An important consideration for the CCOpt and CCOpt-CTS flows is the need for high quality multi-mode timing constraints. The best strategy is to use postCTS timing constraints but with clocks in ideal clocking mode. It is recommended to avoid the use of 'CTS-specific' SDC, which may be encountered in other tool flows or even with FE-CTS based flows.

The diagram below summarizes the key configuration steps.

CCOpt and CCOpt-CTS Configuration Steps



It is important to apply the intended postCTS configuration before invoking CCOpt but with clocks still in ideal timing mode. This is also recommended for use with CCOpt-CTS.

Switch to propagated timing model – CCOpt and CCOpt-CTS switch clocks to propagated mode and update source latencies of clock root pins such that the average arrival time of clocks after CTS matches the before CTS ideal mode arrival times. This process does not happen for clocks that are initially in propagated mode. The source latency update is important so that optimization of inter-clock and I/O boundary paths during `ccopt_design` operates with correct timing. In contrast, in a traditional flow this would be done as a separate flow step. For detailed information about the source latency update scheme, see the [Source Latency Update](#) section.

PostCTS configuration – CCOpt combines CTS with datapath optimization, replacing the need for a separate postCTS setup timing optimization step. Before running CCOpt the session should be configured with postCTS uncertainties, CPPR enabled, OCV timing derates or AOCV enabled, active analysis views, and all other settings appropriate for postCTS optimization. The [update_constraint_mode](#), [setAnalysisMode](#), and [setOptMode](#) commands most commonly used for configuring relevant settings.

The example below illustrates a typical CCOpt/CCOpt-CTS configuration and run script. Some of these settings may also be configured via the Foundation Flow environment.

Quick Start Example

Load postCTS timing constraints. This example loads functional mode and scan mode constraints intended for postCTS use. The user has ensured that clocks are not switched to propagated mode in these SDC files.

```
update\_constraint\_mode -name func -sdc_files  
func_postcts_no_prop_clock.sdc
```

```
update_constraint_mode -name scan -sdc_files  
scan_postcts_no_prop_clock.sdc
```

Ensure that sufficient analysis views are active. Clock specification generation is fully multi-mode and it is important that views using both functional and scan modes are active. CCOpt is hold-slack aware when permitting sinks to be unbalanced to reduce clock area. The first defined setup view is used to determine the primary CTS delay corner – this is the delay corner used by the majority of CTS, but note that CCOpt useful skew scheduling and timing optimization considers setup slack in all active analysis views.

```
set\_analysis\_view -setup {func_max_setup scan_max_setup} -hold  
{func_min_hold scan_min_hold}
```

Define route types. A route type binds a non-default routing rule, preferred routing layers, and a shielding specification together. NDRs can be defined via LEF or using the [add_ndr](#) command.

```
create\_route\_type -name leaf_rule -non_default_rule CTS_2W1S  
-top_preferred_layer M5 -bottom_preferred_layer M4
```

```
create_route_type -name trunk_rule -non_default_rule CTS_2W2S  
-top_preferred_layer M7 -bottom_preferred_layer M6  
-shield_net VSS -bottom_shield_layer M6
```

```
create_route_type -name top_rule -non_default_rule CTS_2W2S  
-top_preferred_layer M9 -bottom_preferred_layer M8  
-shield_net VSS -bottom_shield_layer M8
```

Specify that the route types defined above will be used for leaf, trunk, and top nets, respectively. Note that top routing rules will not be used unless the `routing_top_min_fanout` property is also set.

```
set\_ccopt\_property -net_type leaf route_type leaf_rule  
set\_ccopt\_property -net_type trunk route_type trunk_rule  
set\_ccopt\_property -net_type top route_type top_rule
```

Specify that top routing rules will be used for any clock tree net with a transitive sink fanout count of over 10000.

```
set\_ccopt\_property routing_top_min_fanout 10000
```

Configure library cells for CTS to use. In this example, the `logic_cells` property is not defined so when resizing existing logic cell instances CTS will use matching family cells which are not `dont_use`. The specification of a library cell overrides any `dont_use` setting for that library cell.

```
set\_ccopt\_property buffer_cells { BUFX12 BUFX8 BUFX6 BUFX4 BUFX2 }  
set\_ccopt\_property inverter_cells { INVX12 INVX8 INVX6 INVX4 INVX2 }  
set\_ccopt\_property clock_gating_cells { PREICGX12 PREICGX8 PREICGX6  
PREICGX4 }
```

Include this setting to use inverters in preference to buffers.

```
set_ccopt_property use_inverters true
```

Configure the maximum transition target.

```
set_ccopt_property target_max_trans 100ps
```

Configure a skew target for CCOpt-CTS (`ccopt_design -cts`). This is ignored by CCOpt (`ccopt_design`).

```
set_ccopt_property target_skew 50ps
```

Create a clock tree specification by analyzing the timing graph structure of all active setup and hold analysis views. The clock tree specification contains `clock_tree`, `skew_group`, and property settings. Alternatively, the specification can be written to a file for inspection or debugging purposes and then loaded.

[create_ccopt_clock_tree_spec](#)

```
#create_ccopt_clock_tree_spec -filename ccopt.spec
```

```
#source ccopt.spec
```

Run CCOpt or CCOpt-CTS

```
ccopt_design
```

```
#ccopt_design -cts
```

Report on timing

[timeDesign](#) -postCTS -expandedViews -outDir post_cts_report_dir

Report on clock trees to check area and other statistics.

[report_ccopt_clock_trees](#) -filename clock_trees.rpt

Report on skew groups to check insertion delay and, if applicable, skew.

[report_ccopt_skew_groups](#) -filename skew_groups.rpt

Open the CCOpt Clock Tree Debugger Window. Alternatively, use the “CCOpt Clock Tree Debugger” entry in the main *Clock* menu.

[ctd_win](#)

For a more detailed explanation and recommendation on each of the above settings, see the [Configuration and Method](#) section. For details of the clock tree specification system, see the [Concepts and Clock Tree Specification](#) section.

Early Clock Flow

In this flow, CCOpt creates clock trees during [place_opt_design](#), based on an initial clustering, and annotates clock latencies for timing optimization. The flow also enables CCOpt ideal mode useful skew during the WNS/TNS fixing step inside `place_opt_design`. The high-level flow is shown in the diagram below.

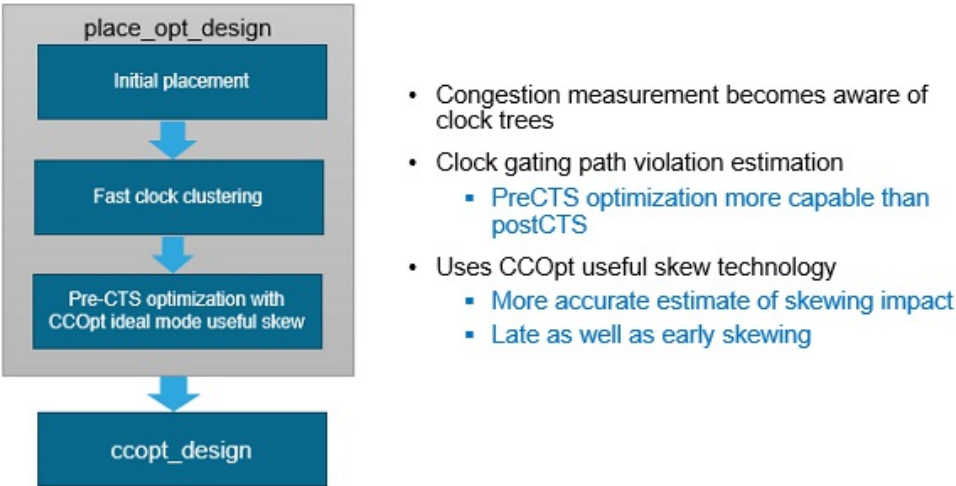
Icon

The early clock flow is a limited-access feature in this release. This feature is enabled by a variable specified using the [setLimitedAccessFeature](#) command. To use this feature, contact your Cadence

representative to explain your usage requirements, before deploying it widely.

Note: To use the early clock flow the CTS configuration must be set up before running the `place_opt_design` command. For details, see the [Flow and Quick Start](#) section.

Early Clock Flow



Use Model

Use the following commands to enable the early clock flow:

The `setDesignMode -earlyClockFlow` is set to `true`. By default it is set to `false`.

The table below outlines the combination of settings for early clock flow and useful skew CTS and the impact on `place_opt_design` behavior as a result.

earlyClockFlow	usefulSkewPreCTS	Behavior in place_opt_design
false (default)	false	No clock tree building No preCTS useful skew
false (default)	true (default)	No clock tree building PreCTS useful skew (skewClock)
true	false	Clock tree clustering No useful skew
true	true	Clock tree clustering CCOpt ideal mode useful skew

Configuration and Method

Properties System

Configuration of CCOpt-CTS and CCOpt is performed using a combination of the clock tree specification and CCOpt properties.

To set a property:

```
set_ccopt_property [-object_type <object>] <property name> <property value>
```

To get a property:

```
get_ccopt_property [-<object_type> <object>] <property name>
```

To obtain help on a property, or to find properties matching a wildcard pattern:

```
get_ccopt_property -help <property name or pattern>
```

To obtain a list of all available properties use this command:

```
get_ccopt_property -help *
```

The help for each property indicates which object type(s) the property applies to. Many properties are global properties for which an object type is not specified, but there are also properties that are applicable to specific object types including pins, skew groups, clock trees, and types of nets.

For example:

```
get_ccopt_property -help target_max_trans
```

```
...
Optional applicable arguments: "-delay_corner name", "-clock_tree
name", "-net_type name", "-early" and "-late".
```

For a summary of all parameters of the [get_ccopt_property](#) and [set_ccopt_property](#) commands, see the *Innovus Text Command Reference*. You can also use the man command, for example: `man get_ccopt_property`

Note that some properties are read-only and can not be set. For further details of property manipulation, see the [CCOpt Property System](#) section.

For descriptions of all public CCOpt properties, see the [CCOpt Properties](#) chapter.

Route Types

CCOpt-CTS and CCOpt use the concept of top, trunk, and leaf net types as illustrated below.

Clock Tree Net Types



- **Leaf nets** – Any net that is connected to one or more clock tree sinks is a leaf net. By default, CCOpt-CTS and CCOpt will insert buffers so that no buffer drives both sinks and internal nodes.
- **Trunk nets** – Any net that is not a leaf net is by default a trunk net.
- **Top nets** – If you configure the `routing_top_min_fanout` property, then any trunk net which has a transitive fanout sink count higher than the configured count threshold will instead be a top net. For example, if the property is set to 10,000, then any trunk net that is above (in the clock tree fan-in cone) 10,000 or more sinks will be a top net.

You can define route types. A route type binds together a non-default routing rule (NDR), preferred routing layers, and a shielding specification. For each net type (leaf, trunk, and optionally top) you can specify the route type that should be used. Non-default routing rules can be defined via LEF or using the [add_ndr](#) command. For example, the following command creates a route type with the name `trunk_rule` that uses the CTS_2W2S non-default rule, with preferred routing layers M6 and M7, with shielding in all layers down to M6 inclusive.

```
create_route_type -name trunk_rule -non_default_rule CTS_2W2S
-top_preferred_layer M7 -bottom_preferred_layer M6
-shield_net VSS -bottom_shield_layer M6
```

The following command specifies that the `trunk_rule` route type should be used for the trunk net type:

```
set_ccopt_property -net_type trunk route_type trunk_rule
```

While the above command might seem superfluous, note that route types can be specified per clock tree, or per clock tree and net type combination. The [Quick Start Example](#) section illustrates commands to configure and apply route types for leaf, trunk, and top nets.

Top Net Configuration

As discussed above, the `routing_top_min_fanout` property can be configured with a sink count threshold to determine which nets are considered top nets instead of trunk nets. This property can be set globally for all clock trees, or per individual clock tree. For example:

```
set_ccopt_property -clock_tree clk500m routing_top_min_fanout 10000
```

By default, each clock tree sink counts as '1' for the purposes of top net thresholds. For macro clock input pins it may be desirable to treat the clock input pin as having a higher count, for example representing the number of internal state elements. The `routing_top_fanout_count` property can be used to

configure this.

For example, to specify that the clock input `mem0/clkin` to a memory should count as 1000 sinks, use the following command:

```
set_ccopt_property -pin mem0/clkin routing_top_fanout_count 1000
```

Routing Rule Recommendations

Clock net routing rules are crucial to obtaining low insertion delay and avoiding signal integrity problems. Especially for small geometry process nodes, the following recommendations should be considered:

- Configure the trunk net type to use double width double spacing and shielding. Prefer middle to higher layers subject to the power grid pattern. Double width is recommended to reduce resistance and permit use of bar shape vias, with double spacing to reduce the capacitance impact of shielding. Shielding is essential to avoid aggressors impacting clock trunk net timing, as impact on clock trunk timing is often significant for both WNS and TNS.
- Configure the leaf net type to use double width and prefer middle layers. Double width is recommended to reduce resistance. Extra spacing is desirable, but extra spacing and/or shielding may consume too much routing resource.
- Try to arrange for each net type (leaf, trunk, top) to use a single layer pair, one horizontal and one vertical, which have the same pitch, width, and spacing. This increases the correlation accuracy between routing estimates before clock nets are routed and actual routed nets.

Library Cells

The library cells used by CCOpt-CTS and CCOpt are configured with the properties listed below. These cell lists may be configured per-power domain and per-clock tree by using the `-power_domain` and `-clock_tree` keys for these properties respectively.

<code>buffer_cells</code>	Specifies buffer, inverter, and clock gating cells. It is recommended to explicitly configure buffer, inverter, and clock gating cells.
<code>inverter_cells</code>	
<code>clock_gating_cells</code>	
<code>logic_cells</code>	Specifies logic cells. If logic cells are not specified, CTS will use any library cell which has the same logic function and is not <code>dont_use</code> when resizing existing logic cell instances.
<code>use_inverters</code>	Specifies that CTS should prefer inverters to buffers.

To view detailed information about above properties, run the following command:

```
get_ccopt_property -help propertyname
```

For example:

```
get_ccopt_property -help buffer_cells
```

A Tcl list of symmetric buffer cells. All buffers created in the clock tree under a power domain will be instances of these cells.

Set the global property to specify buffer cells for all clock trees and all power domains:

```
set_ccopt_property buffer_cells {bufA bufB bufC}
```

Set the per-clock tree/power domain property to specify buffer cells for a particular clock tree and ALL power domains:

```
set_ccopt_property buffer_cells {bufX bufY} -clock_tree clk
```

Set the per-clock tree /power domain property to specify buffer cells for a particular clock tree and power domain:

```
set_ccopt_property buffer_cells {bufX bufY} -clock_tree clk -  
power_domain pd
```

By default CCOpt automatically selects appropriate cells.

Valid values: `list lib_cell`

Default: `{}`

Optional applicable arguments: `"-clock_tree <name>"` and `"-power_domain <name>"`.

The [Quick Start Example](#) illustrates commands to configure library cells. Specifying that CTS can use a library cell overrides any user or library `dont_use` setting for that library cell.

The following are some recommendations:

- Always specify library cells for buffers, inverters and clock gating.
- Use low threshold (LVT) cells. The resulting insertion delay will be lower leading to less impact of OCV timing derates, therefore reducing the datapath dynamic and leakage power increase from timing optimization.
- For many, but not all, low geometry processes inverters result in lower insertion delay and lower power than buffers. In older technologies, buffers may be more efficient. The exact preference here is technology, library, and design target dependent.
- Permitting the largest library cells to be used may be undesirable for electromigration reasons and can increase clock power.
- Very weak cells, for example, X3 and below in many libraries, are usually undesirable due to poor cross-corner scaling characteristics and are sensitive to detailed routing jogs and changes.
- Limiting the number of library cells to no more than 5 per cell type may help reduce run time.

Note: Starting with the Innovus 16.2 version, the library cell list will be pre-screened at the start of CTS, and only those cells with the best drive strength and area characteristics will be used during CTS.

- Do not exclude small cells, such as X4, as otherwise CTS will be forced to use larger more power and area consuming cells to balance skew or implement the useful skew schedule.
- Include always-on buffers and inverters in designs with multiple power domains.

Transition Target

The maximum transition target to CCOpt is specified using the following command:

```
set\_ccopt\_property target_max_trans value
```

The value can be specified in library units, for example “100”, or specified in explicit units for example “100ps”, “0.1ns”.

The transition target can be specified by net type, clock tree, and delay corner. For example, it may be

desirable to have a tighter transition target at sink pins to improve flop CK->Q arc timing, but relax the transition target in trunk nets to reduce clock area and power. Shielding and extra spacing could be used for trunk nets to further reduce clock power whilst avoiding signal integrity problems. This example configures trunk nets to have a 150ps transition target whilst leaf nets have a 100ps transition target:

```
set_ccopt_property -net_type trunk target_max_trans 150ps
set_ccopt_property -net_type leaf target_max_trans 100ps
```

If a target max transition is not specified, CCOpt-CTS and CCOpt will examine the `target_max_trans_sdc` property (see the [SDC Transition Targets](#) section), and if that is not defined then an automatically generated target is chosen. It is recommended to set a transition target unless intentionally using per clock tree settings which are to be obtained from the SDC constraints.

Skew Target

The skew target used by CCOpt-CTS can be configured globally as follows:

```
set\_ccopt\_property target_skew value
```

Alternatively, the target skew can be specified per skew group, for example:

```
set_ccopt_property -skew_group ck200m/func target_skew 0.1ns
```

If a target max transition is not specified, CCOpt-CTS will auto-generate a target. This may not be optimal. Note that CCOpt ([ccopt_design](#) without `-cts`) ignores skew targets, except where skew groups have been explicitly configured to restrict useful skew.

Creating the Clock Tree Specification

The recommended method for generating a clock tree specification is to use the command [create_ccopt_clock_tree_spec](#). This will analyze the timing graph of all active setup and active hold analysis views and create the specification. For details on how the specification creation process operates and the commands used in the specification, see the [Concepts and Clock Tree Specification](#) section.

To write the specification to a file, instead of just applying it immediately in memory, add the `-file` parameter. This example writes the specification to a file and then loads the specification.

```
create_ccopt_clock_tree_spec -filename ccopt.spec
source ccopt.spec
```

Configuration Check

The command `ccopt_design -check_prerequisites` can be used to perform setup, library, and design validation checks without actually running CTS. It is otherwise identical to the checks normally performed near the start of `ccopt_design`.

CCOpt Effort

The `-usefulSkewCCOpt` parameter of the [setOptMode](#) command can be used to specify the optimization effort used by CCOpt. The available options are none, standard, medium, and low. The possible settings are:

- none: low effort `ccopt_design` with no skewing
- standard: low effort `ccopt_design`. This is the default setting.

- `medium`: medium effort `cchopt_design`
- `extreme`: high effort `cchopt_design`

The `-usefulSkewCCOpt extreme` option is used to run the CCOpt flow with high effort. A medium effort is used to lower the optimization effort and with this mode the timing QOR is not expected to be as good as that compared to the high-effort setting but there are run-time benefits. The `standard` or `low` effort setting runs the [cchopt_design](#) -cts and [optDesign](#) flow. This is the default setting. Note that setting the effort level changes multiple internal properties ‘under the cover’. Advanced users using internal properties advised by their support contact should be aware of this.

This parameter setting is only respected by `cchopt_design` and `optDesign -postCTS`.

Create Preferred Cells Stripes to Control IR Drop

Large cells can induce a large IR drop when they switch. In some designs, where power switches occur in regular vertical columns across the chip, it is advantageous to try and place large CTS cells, which may switch often, closer to the power switches to control the R part of the IR drop.

To do this, you can create, one or more, preferred cell stripes objects, using the [create_ccopt_preferred_cell_stripe](#) command. This command lets you define a box in which large, specified, cells should be preferentially placed within the specified stripes; the pitch, width, height, bottom left corner, and the number of stripes is also configured. When these objects are created, CTS will try to place cells in the stripes, wherever possible.

For example, if power switches are arranged in vertical columns at intervals of 50um, are 5um wide, with the first column at 20um from the left of the chip, then you might create preferred cell stripes in the following manner:

```
create_ccopt_preferred_cell_stripe -name stripes_for_IR_drop_control
-bbox {0 0 500 1000} -bottom_edge 0 -height 1000 -left_edge 15 -width
15 -pitch 50 -repeat 10 -cells {CTS_BUF_X16 CTS_BUF_X20 CTS_INV_X16
CTS_INV_X20 ICG_X16 ICG_X20}
```

Assuming that the chip bounds are {0 0 500 1000}, the above command will create a set of 10 stripes, each 15um wide, so that the large (X16 and X20) CTS cells are preferentially placed within 5um on either side of the columns of power switches.

However, this is not a hard constraint, so, if there is a scenario where in order to be DRV clean or to meet skew constraints, a cell needs to be outside the stripes, then CTS has the freedom to do that. But, wherever possible, it will use the area inside the stripes.

Note: These constraint objects should be created before running [cchopt_design](#) .

You can delete any incorrectly created preferred cell stripes objects and view a list of preferred cell stripes objects matching the supplied pattern by using the [delete_ccopt_preferred_cell_stripe](#) and [get_ccopt_preferred_cell_stripe](#) commands, respectively.

You can also specify the preferred cell stripes to which a property applies by using the `-preferred_cell_stripe` parameter of the [set_ccopt_property](#) command. For related properties, see the [CCOpt Properties](#) chapter in the *Innovus User Guide*. The properties related to preferred cell stripes objects are listed below:

- `stripe_bbox`

- `stripe_cells`

EM Avoidance Function

The EM avoidance function of CCOpt lets you find a maximum total capacitance that a cell can drive without violating EM constraints and then set that capacitance value as the target maximum capacitance value for that cell. This helps avoid EM violations when running CTS.

This function is called automatically through the `ccopt_design` command when the property, `consider_em_constraints` is set. This property sets the current waveform calculation method to be used when considering EM constraints. You can use this property to set one or more of following methods:

- `rms` to check the root-mean-square current limit
- `peak` to check the peak current limit
- `avg` to check the average current limit

This function computes the maximum capacitive load for all usable buffer, inverter, and clock gate cells to allow when considering EM constraints. This capacitance value is then set as the target maximum capacitance value for these cells. This is done at the pre-requisite check stage of CTS when no clock tree has been changed, with the trimmed routing information and trimmed usable buffer/inverter/clock gate cells. The computation uses the period of the considered clock tree to calculate currents and the corresponding EM constraints. Then, the computed target is keyed by this period. If a clock tree contains nodes with different periods, there are multiple targets computed for each period.

The targets can be viewed and modified after saving the `ccopt` configuration (`ccopt_config`). Before modifying the targets, you need to first unset the `consider_em_constraints` property. For example:

```
set_ccopt_property consider_em_constraints rms
```

```
get_ccopt_property consider_em_constraints
```

The software returns the following information:

```
rms
```

```
unset_ccopt_property consider_em_constraints
```

```
get_ccopt_property consider_em_constraints
```

The software returns nothing.

The EM avoidance function is designed to avoid most of the EM violations on clock tree nets but not to eliminate all of them entirely. Because it computes the targets based on the preferred routing layers in routing information, generally, there are some violations left on the wires below the preferred layers. However, these too can be eliminated by using the NanoRoute EM rules.

Common Specification Modifications

Stop and Ignore Pins

Stop pins and ignore pins both stop the clock tree specification process from tracing beyond a pin. A stop pin is treated as a sink to be balanced whereas an ignore pin is not balanced. For details about stop and ignore pins, see the [Clock Tree Sink Pin Types](#) section.

Macro Clock Input Pins

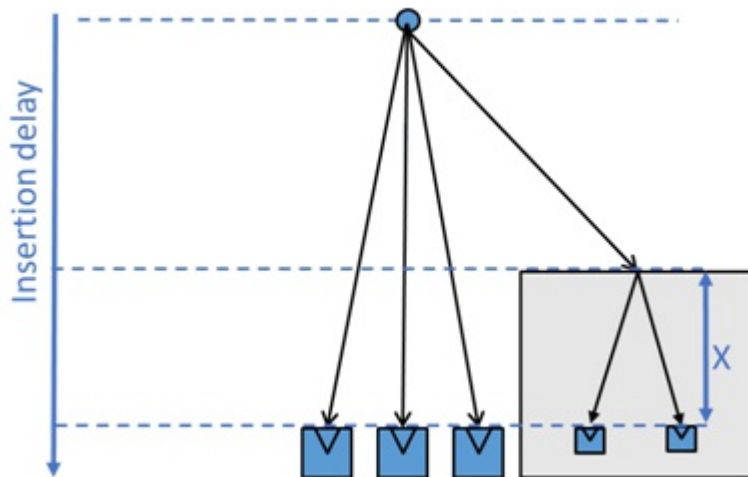
The clock input pins of macros (.lib model) must usually be earlier than other sinks, which means they will have a lesser clock arrival time to take account of the internal clock path inside the macro. If this is represented by a pin specific network latency, [set_clock_latency](#), command in the SDC timing constraints then the automatically-generated clock tree specification will take this into account. This is discussed further in the [Network Latencies](#) section.

If the clock offset at a macro clock pin is not captured in the timing constraints, then you must add this. For example:

```
set_ccopt_property -pin mem1/CK insertion_delay 1.2ns
```

Note that the property setting is the delay to be assumed inside the macro. Positive numbers will reduce the clock arrival time at the pin, negative numbers will increase it. This is illustrated in the following diagram, where X represents the property setting value.

Pin Insertion Delay



It is possible to set a pin insertion delay at any clock sink to adjust the skew of the sink relative to other sinks without such a setting. This can be used to implement manual or preCTS useful skew. Note that setting a pin insertion delay on large number of pins is not recommended and may increase clock area and power.

Architectural Clock Gates

An architectural clock gate is a clock gate typically very early (small insertion delay) in a clock tree that is used to enable and disable entire functions or logical partitions of a design. The flops controlling such a clock gate may also need to be scheduled early to avoid setup slack violations at the clock gate enable input. This can be achieved by adding an additional skew group to balance the flops with the clock gate. For an example of this, refer to the example, Balancing flops with a clock gate, in the [Modifying Skew Groups](#) section. Such additional configuration for architectural clock gates is frequently recommend with CCOpt, and will be essential for timing closure with CCOpt-CTS.

Restricting CCOpt Skew Scheduling

CCOpt will initially compute the maximum insertion delay over all skew groups. By default, CCOpt skew scheduling is restricted such that the insertion delay of any sink may not exceed some factor multiplied by the

initially computed maximum insertion delay. This factor is set by the property, `auto_limit_insertion_delay_factor`, which in the 14.2 version of the software, defaults to 1.5. This permits useful skew scheduling to increase the global maximum insertion delay by up to 50%. Useful skew scheduling is unrestricted by how much it can decrease the insertion delay to a sink.

To change this restriction on late useful skew set the property. For example to lower the restriction to a 20% insertion delay increase:

```
set_ccopt_property auto_limit_insertion_delay_factor 1.2
```

To restrict the skew of a given skew group in CCOpt set the `target_skew` property on the skew group and set the `constrains` property of the skew group to include the keyword 'ccopt'. For syntax details, see the [Defining Skew Groups](#) section. For example, to place a hard limit on the skew of all skew groups to 400ps, irrespective of the impact on timing closure, use the following commands:

```
foreach sg [get_ccopt_skew_groups *] {  
  set_ccopt_property target_skew 400ps -skew_group $sg  
  set_ccopt_property constrains ccopt -skew_group $sg  
}
```

Method

The recommended method for setting up CCOpt or CCOpt-CTS on a new design is to use the following steps:

1. Configure and create the clock tree specification as per the Quick Start Example and configuration instructions above.
2. Before invoking the [ccopt_design](#) command use the CCOpt Clock Tree Debugger in unit delay mode to inspect the clock tree. This will permit examination of the clock tree structure. For more information, see the [Unit Delay](#) section.
3. Invoke only the clustering step of CCOpt or CCOpt-CTS which performs buffering to meet design rule constraints but does not perform skew balancing or timing optimization. Check the maximum insertion delay path looks sensible in the CCOpt Clock Tree Debugger. For designs with narrow channels, many blockages, or complex power domain geometries this is a good time to check for large transition violations caused by floorplan issues. The screenshot, "Cluster Maximum Insertion Delay", below shows the placement view (left) and the CCOpt Clock Tree Debugger view (right) with the maximum insertion path delay highlighted in green.
4. For a CCOpt flow with a simple clock tree, for example a CPU core, switch to using full `ccopt_design`. For a design with a complex clocking architecture consider using trial mode, which will perform clustering and then balancing using virtual delays. The trial balancing can be inspected to look for large skew or insertion delay increases due to conflicting skew group constraints. The design can be timed using [timeDesign](#) -postCTS to check for large timing slack violations, for example, due to incorrect balancing constraints. Virtual delays will appear in timing reports as additional arrival time increments.
5. Run full `ccopt_design`. Inspect the log file for errors and warnings. For CCOpt, a summary table of timing slack and other metrics at each stage of the `ccopt_design` internal flow is reported.
6. For CCOpt, check the worst chain report in the log. Note that there may be multiple worst chain reports in the log. It is recommend to look at the worst chain report after the last occurrence of skew adjustment before any re-clustering steps in the log, this is usually the second last chain report. This report will indicate if useful skew scheduling has hit constraint limits that are limiting optimization. For more information on the worst chain report, see the [Worst Chain](#) section .

7. Report on clock trees and skew groups. For example, it is recommended to check skew group maximum insertion delay and clock tree area even if setup timing slack is closed. For more information, see the [Reporting](#) section.

As mentioned above, CCOpt and CCOpt-CTS can be configured between `cluster`, `trial`, or `full` mode. This is performed using the `cts_opt_type` mode setting, which controls both CCOpt and CCOpt-CTS.

```
set_ccopt_mode -cts_opt_type cluster | trial | full
ccopt_design -cts
```

The default is full mode. The concepts of clustering and trial virtual delay balancing are detailed further in the [Graph-Based CTS](#) section.

Cluster Maximum Insertion Delay



Concepts and Clock Tree Specification

Graph-Based CTS

CCOpt-CTS and CCOpt use a graph-based CTS algorithm to perform skew balancing (CCOpt-CTS) or initial seed balancing (CCOpt). The main internal steps in this are as follows:

- **Clustering** – This step groups nearby clock sinks into clusters and buffers clock trees to meet maximum transition, capacitance, and length constraints, such as DRV (Design Rule Violation) constraints. After clustering, the maximum insertion delay is approximately known.
- **Constraints analysis and virtual delay balancing** – Constraints analysis identifies how the balancing constraints (skew and insertion delay constraints) interact and identifies where delay should be added to the clock graph to best meet these constraints. For example, a common scenario is identifying where to add delay to balance test mode clock skew without impacting functional mode clock insertion delay. This happens automatically without any user intervention or need for user-driven sequential steps. Virtual delay balancing is simply the process of annotating clock nodes in the timing graph with additional delay that is added to the propagated clock arrival time to achieve the solution found by constraints analysis. CCOpt uses both clustering and virtual delays to initially balance clocks to obtain initial propagated mode timing and to permit run-time efficient what-if style analysis during useful skew scheduling.
- **Implementation** – This step synthesizes virtual delays using real physical cells. It is followed by a

refinement process to account for the difference between virtual delays and those achievable with physical cells. For CCOpt-CTS, this is essentially the last step. For CCOpt, further post-CTS style datapath optimization is performed.

The use of this graph-based CTS approach, combined with the multi-mode clock specification generated by the `create_ccopt_clock_tree_spec` command enables CCOpt-CTS and CCOpt to cope with large complex clock structures, typically, with zero or minimal user intervention.

Clock Trees and Skew Groups

Clock trees and skew groups are the two key object types used in the CCOpt clock specification. The term object is used here because clock tree and skew group objects can be defined, modified, and deleted using commands. For example, [create_ccopt_clock_tree](#), [create_ccopt_skew_group](#), [modify_ccopt_skew_group](#), and [delete_ccopt_skew_groups](#).

Properties can be set per skew group or clock tree instead of globally. For example, [set_ccopt_property](#) -skew_group *name* target_skew *value*. The [report_ccopt_clock_trees](#) and [report_ccopt_skew_groups](#) commands can be used to generate reports on clock trees and skew groups. For more information, see the [Reporting](#) section.

Clock Trees

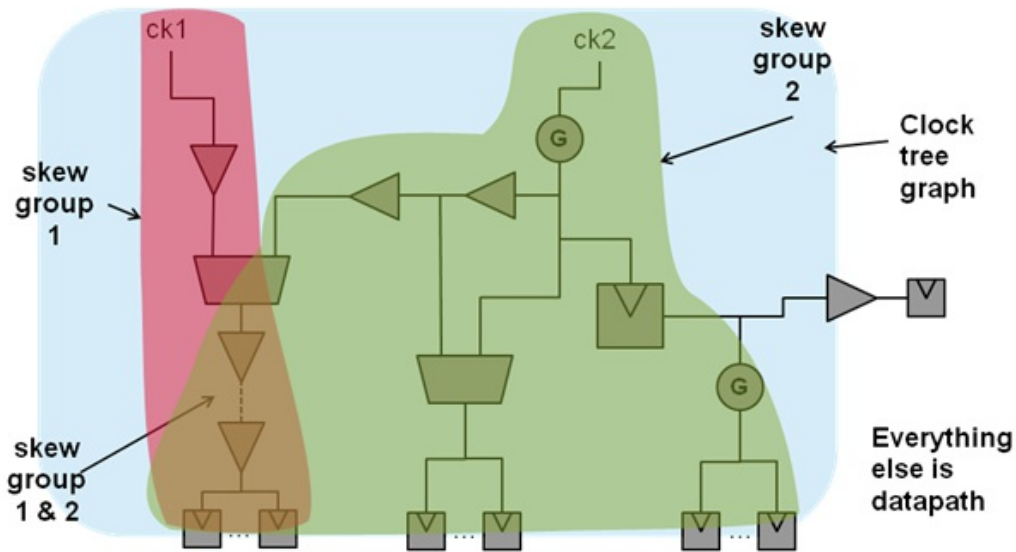
- The union of all clock trees specifies the subset of the circuit graph that CTS will buffer. The circuit subset covered by clock tree definitions is best referred to as a clock tree graph since clock trees may interact, for example via clock logic cells. The clock tree graph is a single physical graph even in a multi-mode timing environment.
- Maximum transition times, route types and other physical properties are associated with the clock tree graph or with individual trees in the clock tree graph.
- In all but rare exceptional circumstances, the clock tree definitions created by `create_ccopt_clock_tree_spec` do not require user modification.

Skew Groups

- A skew group represents a balancing constraint and is the CTS equivalent of an SDC clock. The automatically generated clock tree specification will create one skew group per SDC clock per mode.
- Each skew group has one or more sources and a number of sinks. Among other properties, a skew target and insertion delay target can be set per skew group. Any pin in the clock tree graph can be a skew group source or sink and pins can be designated a skew group specific ignore pin such that the specific skew group does not propagate beyond the pin.
- CCOpt-CTS global skew balancing aims to achieve an equal delay, subject to the skew target, from all sources to all sinks within each skew group. CCOpt virtually balances skew groups to zero skew to determine initial clock tree timing with propagated clocks before optimization starts.
- A skew group can be viewed as a subset of the clock tree graph superimposed on top of the clock tree graph. Skew groups can overlap, share sources, and/or sinks.
- In complex cases or with CCOpt-CTS where the SDC timing constraints do not fully capture the balancing requirements, user adjustment to the skew group configuration may be required and/or additional skew groups can be defined.

The diagram below illustrates the relationship between the clock tree graph and skew groups. Note the path to the data input of a flip-flop at the right hand side, the clock tree graph is 'pruned back' to exclude this path, the input to the right most buffer will be an ignore pin – clock pin types are discussed later.

Clock Tree Graph and Skew Groups



Automatic Clock Tree Specification Creation

The [`create_ccopt_clock_tree_spec`](#) command is used to automatically create clock tree and skew group definitions from analysis of the active timing constraints, typically those loaded from SDC. It is important to note that the `create_ccopt_clock_tree_spec` command does not perform a text based or any other kind of direct translation from SDC constraint commands but performs an analysis of the clock definitions and clock propagation in the timing graph. This means that whilst there is often a close correspondence between SDC commands and the clock tree specification in some areas there will not be an exact one-to-one mapping.

- **Clock trees** – The automatically generated specification will have clock trees defined for primary clocks at input ports, generated clocks at sequential flip-flop outputs, and will have ignore pins at the edge of the clock tree graph. Typically, the user can ignore the precise clock tree definitions.
- **Skew groups** – The automatically generated specification will have one skew group defined per timing clock in each constraint mode. Skew groups corresponding to generated timing clocks are set to be non-constraining, which means they are ignored by CTS but included for reporting purposes. The sinks of generated timing clocks are balanced with sinks of the appropriate master clock. This is arranged via a single skew group corresponding to the master clock that propagates through the generator. This is illustrated as part of the "Single Mode Example" below.

Skew groups are named according to the timing clock and constraint mode they correspond to. For example, the timing clock with name `clk1500m` in the `func` constraint mode would be given name `clk1500m/func`.

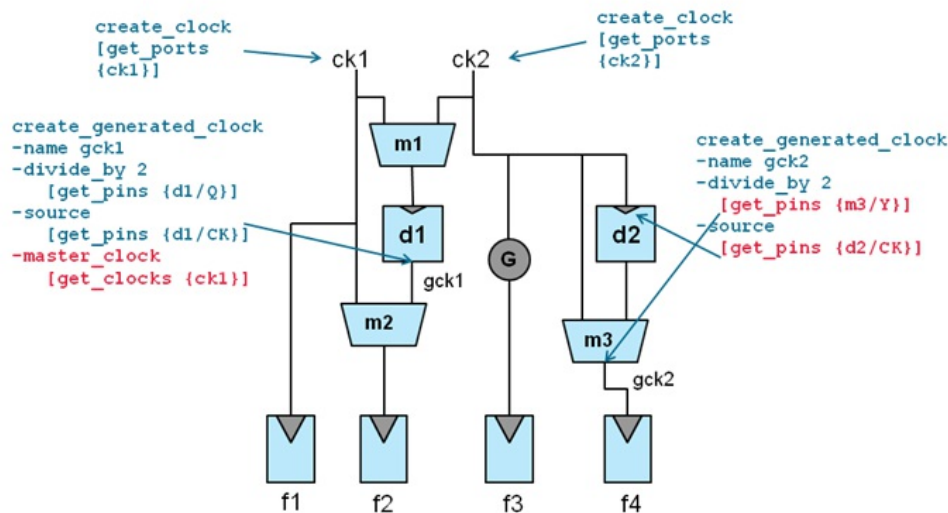
Note: In some CCOpt property names, the process of clock specification generation from SDC constraints is sometimes referred to as 'extraction' for historic reasons. This is not to be confused with parasitic extraction.

Single Mode Example

The diagram below shows a single constraint mode example with two clocks, some multiplexers, and two clock dividers. The SDC clock definitions are illustrated.

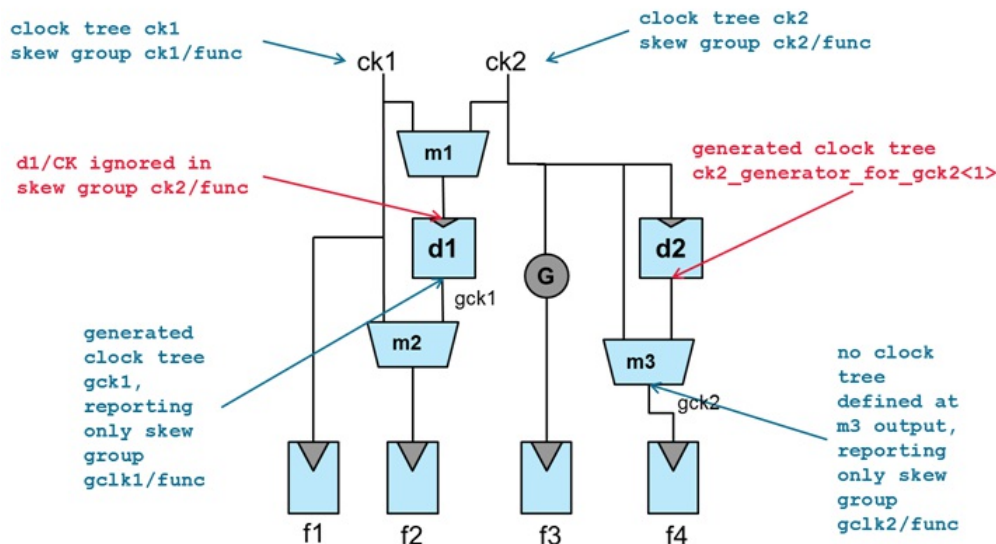
Note the precise definitions of the generated clocks carefully.

Single Mode Example – SDC Clock Definitions



On the left side, the generated clock `gck1` refers to master `ck1` such that `ck2` does not propagate to `f1` or `f2`. On the right side, the definition of `gck2` is such that the path from `d2/CK` to `m3/Y` is considered part of the clock generator circuit. Both these points have implications for the resulting clock tree specification output that is annotated in the diagram below.

Single Mode Example – Clock Tree Specification



In the generated specification, a clock tree is defined at each of the primary inputs `ck1` and `ck2`.

On the left side, a generated clock tree is defined at the output of divider `d1` to distinguish `d1` as a sequential element in the clock graph. Without this generated clock tree definition CTS would treat `d1` as a regular sink. Additionally, at the output of divider `d1` a skew group `gck1/func` is defined, but note that this skew group is non-constraining so does not influence CTS. It is present purely for reporting purposes. Sinks `f1` and `f2` are balanced together by skew group `ck1/func`. Skew group `ck2/func` is ignored at the input to `d1`, this corresponds to the `master_clock` specification in the SDC.

On the right side, no generated clock tree is defined at the output of multiplexer m3, since m3 is a combinational cell. However, a non-constraining skew group is defined at the output of multiplexer m3 for reporting purposes. So that CTS does not treat divider d2 as a regular clock sink and so that the path from d2 to m3 is included in the clock tree graph, a generated clock tree is defined at the output of d2.

Key lines from the output of `create_ccopt_clock_tree_spec -filename ccopt.spec` for the example are given below.

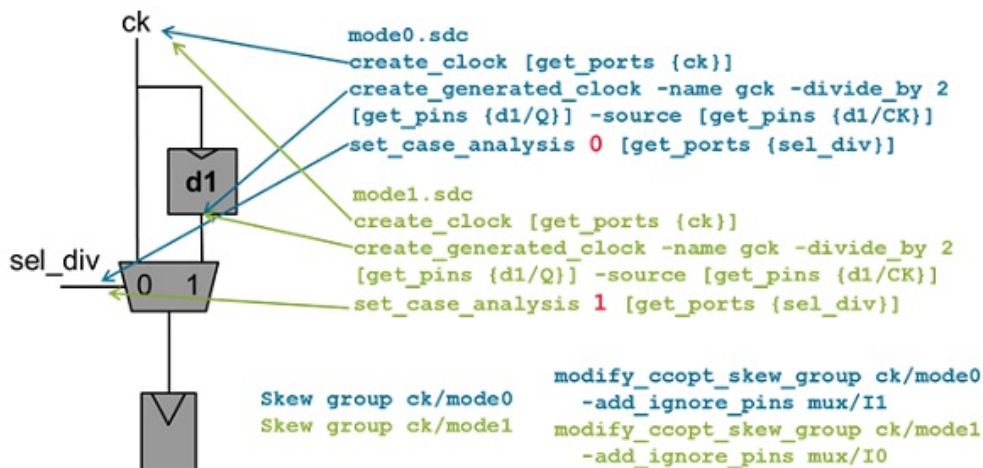
Single Mode Example – create_ccopt_clock_tree_spec Output

```
create_ccopt_clock_tree -name ck2 -source ck2 -no_skew_group
create_ccopt_generated_clock_tree -name ck2_generator_for_ck2<1> -
source d2/Q -generated_by d2/CK
create_ccopt_clock_tree -name ck1 -source ck1 -no_skew_group
create_ccopt_generated_clock_tree -name gck1 -source d1/Q -
generated_by d1/CK
create_ccopt_skew_group -name ck1/func -sources ck1 -auto_sinks
create_ccopt_skew_group -name ck2/func -sources ck2 -auto_sinks
modify_ccopt_skew_group -skew_group ck2/func -add_ignore_pins d1/CK
create_ccopt_skew_group -name gck1/func -sources d1/Q -auto_sinks
set_ccopt_property constrains -skew_group gck1/func none
create_ccopt_skew_group -name gck2/func -sources m3/Y -auto_sinks
set_ccopt_property constrains -skew_group gck2/func none
```

Multi-Mode Example

The diagram below shows a simple multi-mode example annotated with SDC constraints and skew group information.

Multi-Mode Example



The resulting specification contains the following:

- Two clock trees, ck and gck. The clock tree definitions tell CTS which parts of the circuit are included in the clock tree graph and are not mode specific.
- Two skew groups, ck/mode0 and ck/mode1. The skew groups tell CTS how to perform balancing.
- Each skew group has an ignore pin defined at the appropriate multiplexer input. This represents the fact that there is no need to balance the direct clock path with the divided clock path as the paths are never

active in the same mode at the same time.

Key commands from the specification are listed below. Some details have been omitted for clarity.

Multi-Mode Example – create_ccopt_clock_tree_spec Output

```
create_ccopt_clock_tree -name ck -source ck -no_skew_group
create_ccopt_generated_clock_tree -name gck -source d1/Q -
generated_by d1/CK
create_ccopt_skew_group -name ck/mode0 -sources ck -auto_sinks
create_ccopt_skew_group -name ck/mode1 -sources ck -auto_sinks
modify_ccopt_skew_group -skew_group ck/mode0 -add_ignore_pins mux/I1
modify_ccopt_skew_group -skew_group ck/mode1 -add_ignore_pins mux/I0
create_ccopt_skew_group -name gck/mode0 -sources d1/Q -auto_sinks
set_ccopt_property constrains -skew_group gck/mode0 none
create_ccopt_skew_group -name gck/mode1 -sources d1/Q -auto_sinks
set_ccopt_property constrains -skew_group gck/mode1 none
```

SDC Transition Targets

The `create_ccopt_clock_tree_spec` command will translate `set_max_transition` constraints. For example, consider the SDC constraint “`set_max_transition 0.123 [get_clocks {ck1}]`”.

The automatically generated specification will contain the following line:

```
set_ccopt_property target_max_trans_sdc -clock_tree ck1 0.123
```

CCOpt-CTS and CCOpt will look at the `target_max_trans` property. If this is set to the default value of `auto`, then the `target_max_trans_sdc` will be inspected. If `target_max_trans_sdc` is not set, then an automatic default will be computed.

Network Latencies

The `create_ccopt_clock_tree_spec` command will translate clock network latency settings to an insertion delay target on the corresponding skew group. For example, consider the functional mode SDC constraint, “`set_clock_latency 1.456 [get_clocks {ck1}]`”. The automatically generated specification will contain the following line:

```
set_ccopt_property target_insertion_delay -skew_group ck1/func 1.456
```

Similarly, pin network latency settings are translated to the `insertion_delay` property of a pin. This property is often referred to as a pin insertion delay. A pin insertion delay represents the delay ‘underneath’ a clock sink. For example, for a macro clock input pin, the pin insertion delay would represent the internal clock path delay inside the macro. Continuing the above example, add the constraint “`set_clock_latency 0.234 [get_pins {mem1/CK}]`”. The automatically generated specification will additionally contain the following line:

```
set_ccopt_property insertion_delay -pin mem1/CK 1.222
```

The property setting indicates that the delay internal to the macro clock input `mem1/CK` is `1.222`. The value `1.222` is computed as the difference between the clock latency of `1.456` and the pin latency of `0.234`. Note that SDC pin-specific latencies override clock latencies, which means they are not added together.

Clock Tree Convergence

In some circumstances, the clock tree graph undesirably propagates into datapath and includes what should be datapath as part of the clock tree graph. For example, this can happen due to missing `set_case_analysis` or other incorrect SDC constraints. Including significant datapath logic as part of the clock tree graph can result in excessive CCOpt or CCOpt-CTS run time due to large numbers of paths existing between a skew group source and sink due to multiple levels of re-convergent logic. Additionally, such paths would not be optimized by datapath optimization.

To help detect cases where run time would be adversely affected, the automatically generated clock tree specification includes an invocation of the [check_ccopt_clock_tree_convergence](#) command. This command traces the number of paths to every sink and issues a warning if the number of clock paths to any sink is greater than a default threshold of 100 paths. The [report_ccopt_clock_tree_convergence](#) command can be used to report sinks with large numbers of clock paths.

To remedy this situation, either correct the SDC constraints, for example by adding [set_case_analysis](#), [set_clock_sense](#) `-stop_propagation` or other suitable commands, or use a clock tree stop, ignore, or exclude pin as appropriate. These pins are described in the next section.

Clock Tree Sink Pin Types

Clock tree sink pin types can be manually overridden before invoking the `create_ccopt_clock_tree_spec` command with the following property setting:

```
set_ccopt_property -pin pin_name sink_type ignore | stop | exclude
```

Ignore pin (ignore)

An ignore pin is considered as a part of the clock tree graph. CTS will perform DRV buffering up to the pin, but the pin will not be considered as a sink in any skew group, which means the latency to an ignore pin is not important. Tracing through and beyond the pin will be disabled. Sometimes such a pin is referred to as a clock tree ignore pin. An alternative strategy to deploying an ignore pin would be to use the SDC constraint, `set_clock_sense -stop_propagation`. This may be preferable since it would keep the timing model in synchronization with the CTS configuration.

Stop pin (stop)

A stop pin is considered as a part of the clock tree graph. CTS will perform DRV buffering up to the pin and by default the pin will be considered a sink to be balanced in any skew group that reaches the stop pin. Tracing through and beyond the pin will be disabled.

Exclude pin (exclude)

An exclude pin is a pin that is not part of the clock tree graph but might still be connected to a clock net anyway if the same net has other clock fanout. Specifically, the clock tree graph must not extend beyond an exclude pin but it can be pruned back from an exclude pin. The `create_ccopt_clock_tree_spec` command will prune back from an exclude pin and, if possible, specify an ignore pin earlier in the fanin cone. The [Shared Clock and Data Concerns](#) section discusses how to add buffers to disconnect exclude pins from any clock tree nets they may be connected to. This can be important where clock and datapath overlap.

Note: In addition to the above pin types, it is possible to make any pin that is within the clock tree graph a skew group specific ignore or sink pin. This is discussed in the subsequent sections.

Manual Setup and Adjustment of the Clock Specification

Following are some important recommendations for setting up and adjusting the clock tree specification:

- It is recommended that the `create_ccopt_clock_tree_spec` command is used to create the clock tree specification.
- It is not recommended to edit the specification file generated by the `create_ccopt_clock_tree_spec -file filename` command. A more stable flow is obtainable either by adjusting the SDC, configuring CCOpt properties, setting clock tree sink pin types before generating the specification, or making skew group adjustments after loading the specification.
- Consider making adjustments to the SDC timing constraints instead of the CTS specification, if applicable. This will ensure that timing analysis uses a clock propagation model consistent with the CTS configuration. For example, setting a clock logic instance input pin to be a clock tree ignore pin will stop CTS tracing through the pin, but will not stop [report_timing](#) from propagating a clock through the pin. The `create_ccopt_clock_tree_spec` command has been engineered to create a clock tree specification consistent with the active timing constraints.

The table below shows commonly used commands for manipulating clock trees and skew groups:

Command Name	Usage
create_ccopt_clock_tree	Adds a new clock tree definition into the in memory clock tree specification.
delete_ccopt_clock_trees	Removes a clock tree definition from the in memory clock tree specification.
create_ccopt_skew_group	Adds a new skew group definition into the in memory clock tree specification.
delete_ccopt_skew_groups	Removes a skew group definition from the in memory clock tree specification.
modify_ccopt_skew_group	Permits adjustment to sinks and ignore pins of a skew group.
set_ccopt_property -skew_group <i>skew_group_name property_name value</i>	Adjusts skew group specific properties. The most commonly adjusted properties are <code>target_skew</code> and <code>target_insertion_delay</code> .

Note: The above commands do not modify the design or perform any CTS but manipulate the in-memory clock tree specification.

Defining Clock Trees

Clock trees are defined using the `create_ccopt_clock_tree` and [create_ccopt_generated_clock_tree](#) commands. For example:

```
create_ccopt_clock_tree -name ck -source ck -no_skew_group
create_ccopt_generated_clock_tree -name gck -source d1/Q -
generated_by d1/CK
```

The optional `-name` parameter can be used to specify the name of the clock tree. Alternatively, the source pin name will be used as the clock tree name. The mandatory `-source` parameter specifies the clock tree root pin from which clock tree tracing will be performed. The `-no_skew_group` parameter disables the automatic creation of a corresponding skew group, otherwise a skew group with the same name as the clock tree is automatically created. In addition, the definition of a generated clock tree requires the `-`

`generated_by` parameter to specify the input side of the clock generator, which is typically the clock input pin of a divider flip-flop.

When a clock tree is defined, CCOpt traces the circuit connectivity from the specified source pin, adding the nets and cell instances it encounters to the clock tree graph. Tracing continues until encountering a clock pin (such as a flip-flop, latch, or macro input), a user-defined stop, ignore, or exclude pin. A generated clock tree definition must normally be used at the output of a sequential cell to continue tracing.

Defining Skew Groups

Skew groups are defined using the [`create_ccopt_skew_group`](#) command. The complete syntax of this command is detailed below:

```
create_ccopt_skew_group
[-help]
-name skew_group_name
[-constrains {icts | ccopt_initial | ccopt}]
[-sinks pins | -shared_sinks pins | -exclusive_sinks pins | -
auto_sinks | -filtered_auto_sinks pins | -balance_skew_groups
skew_group_names]
[-sources pins]
[-rank rank]
[-target_insertion_delay value]
[-target_skew value]
[-from_clock clock_name]
[-from_constraint_modes constraint_mode_names]
[-from_delay_corners delay_corner_names]
```

The descriptions of the above parameters are in the table below:

Parameter Name	Description
-name <i>skew_group_name</i>	Specifies the mandatory name for the newly created skew group. The <code>create_ccopt_clock_tree_spec</code> command will use a name in the following format: clock_name/constraint_mode_name
-constrains {cts ccopt_initial ccopt}	Optionally specifies the parts of CTS that the skew group should constrain. A list of keywords is specified: CCOpt-CTS/CCOpt initial balancing (cts/ccopt_initial) or full CCOpt flow (ccopt). Note that cts and ccopt_initial are synonymous. The default is ccopt_initial such that the skew group constrains CCOpt-CTS and influences the initial virtual delay balancing step of CCOpt. Note: The initial virtual delay balancing step of CCOpt ignores skew targets. The ccopt setting can be used to limit the range of useful skew performed by CCOpt.
-sinks <i>pins</i>	Specifies sink pins to be balanced within this skew group. If the -rank parameter is not used, then this is equivalent to -shared_sinks.
-shared_sinks <i>pins</i>	Specifies sink pins to be balanced with this skew group. Sinks will be balanced in any

existing skew group that they are sinks in, and will additionally be balanced in the newly created skew group. The newly created skew group is given a rank of 0.

-exclusive_sinks *pins*

Specifies sink pins to be balanced with this skew group. The newly created skew group will have a rank number 1 above all existing skew groups. This means that sinks will only be balanced in the newly created skew group and will be ignored in existing skew groups, which by definition will have a lower rank.

-auto_sinks Automatically traces the clock tree graph from the source pins looking for sinks, for example flip-flop clock input pins.

Note: If this parameter is not specified and an explicit list of sinks is not specified then the skew group will not have any sinks.

-filtered_auto_sinks *pins*

This performs the same tracing as `-auto_sinks`, but will only select sinks that are also in the specified pins.

-balance_skew_groups *skew_group_names*

Specifies that the newly created skew group should be a merging of existing skew groups. The newly created skew group will have the union of source pins from the existing skew groups and sink pins from the existing skew groups. The effect is as if to balance existing skew groups together as a single skew group. Typically, this parameter is used to balance two skew groups that correspond to different clock trees.

-sources *pins*

Specifies the source pins for the skew group.

-rank *rank* Specifies the exclusive sinks rank for the skew group. This is discussed in a subsequent section.

-target_insertion_delay *value*

Specifies a target insertion delay for this skew group. CCOpt-CTS attempts to keep the latency of all sinks within half the target skew earlier or later than this delay. CCOpt will use this target insertion delay during initial virtual delay balancing. The default `auto` setting permits an increase of 5% from the initial clustering insertion delay to improve clock power. The special value of `min` can be used to aim for minimum insertion delay even at the expense of some clock power. Alternatively, set the `target_insertion_delay` property for the skew group using the `set_ccopt_property` command.

-target_skew *value*

Specifies a target skew for skew between sinks of this skew group overriding the global skew target. CCOpt-CTS will respect this setting. CCOpt ignores it unless the `-constrains` setting includes `ccopt`. Alternatively, set the `target_skew` property for the skew group using the `set_ccopt_property` command.

-from_*

The `-from` parameters are used by `create_ccopt_clock_tree_spec` command to record the origin of the skew group definition. These are accessible via the `extracted_from_clock_name`, `extracted_from_constraint_mode_name`, and `extracted_from_delay_corners` properties of a skew group, and can be accessed using the [get_ccopt_property](#) command. These parameters do not influence CTS.

Note: The parameters taking a list of pins operate with either a plain TCL list of hierarchical pin names or with a collection of pins obtained from the [get_pins](#) command.

Skew Group Rank

The rank of a skew group determines whether a sink pin is an active sink in that skew group or not. A pin is only an active sink in the skew group(s) with the highest rank out of all the skew groups to which the pin belongs. An active sink is a pin that will be balanced against other active sinks in the same skew group.

For example, consider the following sequence of commands:

```
create_ccopt_skew_group -name SG1 -sources get_pins top -shared_sinks
[get_pins */D]
create_ccopt_skew_group -name SG2 -sources get_pins top -
exclusive_sinks [get_pins *XYZ*/D]
create_ccopt_skew_group -name SG3 -sources get_pins top -
exclusive_sinks [get_pins *XYZ_01*/D]
```

After running the first command, a single skew group SG1 is created. Shared sinks are specified so this skew group has a rank of zero. All the "D" pins in the design are members of this skew group. In addition, all the "D" pins are active sinks in skew group SG1. This is because SG1 is the highest ranked skew group so far, even though it has a rank of zero.

The second command defines skew group SG2. Exclusive sinks are specified so this skew group has a rank of 1, which is one higher than the current highest rank of 0. Pins that match the pattern “*XYZ*/D” are now members of both SG1 and SG2. However, they are only active sinks in SG2, which is the highest ranked parent skew group so far. Pins that do not match this pattern remain active sinks in SG1.

The third command defines another exclusive skew group SG3. Exclusive sinks are specified so this skew group has a rank of 2, which is one higher than the current highest rank of 1. Pins that match the pattern “*XYZ_01*/D” are now members of skew groups SG1, SG2, and SG3. However such pins are only active sinks in SG3, which is the highest ranked parent skew group. Pins that matched the pattern “*XYZ*/D” but not “*XYZ_01*/D” are members of both SG1 and SG2 but only active sinks of SG2. Sinks that do not match the pattern *XYZ*/D are active sinks in SG1.

The rank of a skew group can be accessed via the `exclusive_sinks_rank` property.

Finding Active Skew Group Sinks

Use the following command to find all the sink members of a skew group:

```
get_ccopt_property -skew_group name sinks
```

Use the following command to find all the active sink members of a skew group:

```
get_ccopt_property -skew_group name sinks_active
```

Use the following command to find all the skew groups for which a pin is a sink member:

```
get_ccopt_property -pin name skew_groups_sink
```

Use the following command to find all the skew groups for which the pin is an active sink:

```
get_ccopt_property -pin name skew_groups_active_sink
```

Use the following command to find all the skew groups which are active at a pin, either passing through the

pin or for which the pin is an active sink:

```
get_ccopt_property -pin name skew_groups_active
```

Note: In debugging CCOpt-CTS skew or CCOpt initial balancing, the ‘active’ properties above should be used, since these reflect the constraints CTS will respect. For example, if a pin is configured as a sink of skew group but the skew group does not propagate to the pin due to a lack of connectivity, the pin will not be an active sink of the skew group. After defining skew groups or modifying existing skew groups it is recommended to invoke the [report_ccopt_skew_groups](#) or [ccopt_design](#) command to ensure that the CTS timer is updated before checking the active sinks properties.

Modifying Skew Groups

The [modify_ccopt_skew_group](#) command is used to make changes to the sink and ignore pins associated with a skew group. The syntax of the command is provided below.

```
modify_ccopt_skew_group  
[-help]  
-skew_group skew_group_name  
[-add_sinks pins | -remove_sinks pins]  
[-add_ignore_pins pins | -remove_ignore_pins pins]
```

The `-add_sinks` and `-remove_sinks` parameters are used to add and remove sinks. The `-add_ignore_pins` and `-remove_ignore_pins` parameters are used to add and remove ignore pins, and are discussed below.

The [set_ccopt_property](#) command can be used to modify properties of a skew group, including the `target_insertion_delay`, `target_skew`, and constraints properties.

Skew Group Ignore Pins

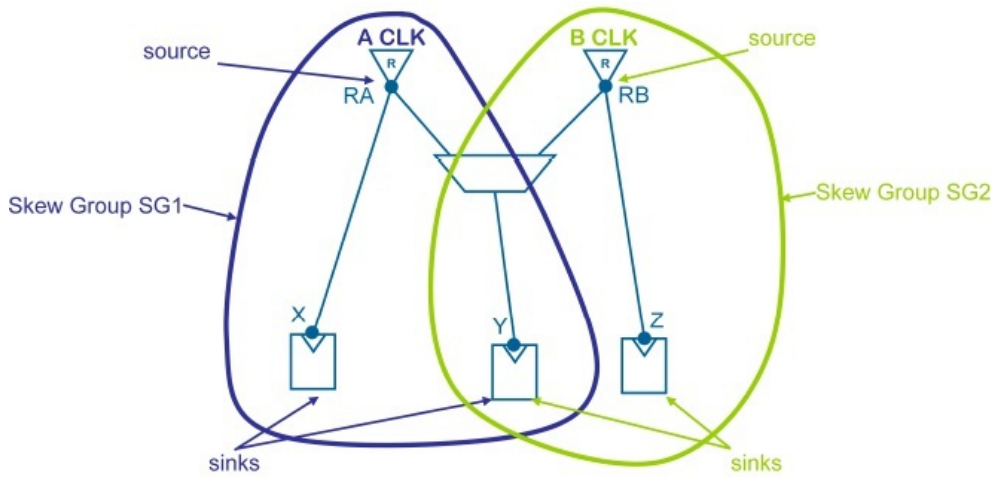
Specifying a pin as an ignore pin of a skew group stops CTS from considering the latency to that pin in that specific skew group, and stops that specific skew group propagating through and beyond that pin. Other skew groups at the pin are not affected. Skew group ignore pins are always applicable regardless of the skew group rank.

For example, if a leaf flip-flop clock pin is specified as a skew group ignore pin, CTS will not balance that flip-flop with other sinks for the same skew group. Balancing of other skew groups, possibly involving the same pins, would not be affected.

If a non-leaf pin is specified as a skew group ignore pin, for example a multiplexer input, CTS will ignore both the latency to and through that multiplexer input in the given skew group. Other skew groups passing through the same multiplexer input would not be affected. In such an example, any flip-flops in the fanout of the multiplexer would cease to be active sinks of the skew group.

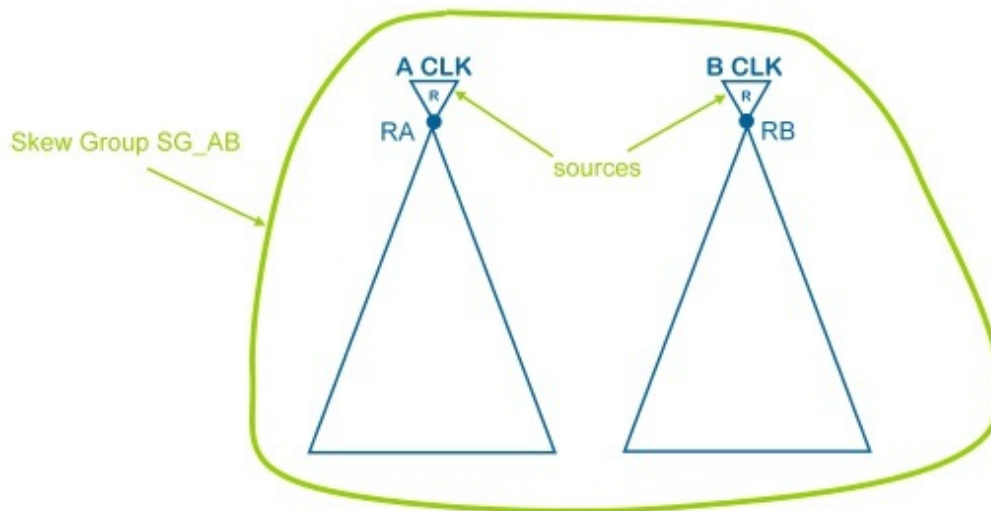
Example – Overlapping Skew Groups

The diagram below illustrates an example with two clock trees, A and B, with corresponding skew groups, SG1 and SG2. The sink Y is an active sink of both skew group SG1 and skew group SG2. Sink X and Y are balanced together and sink Y and Z are balanced together. The insertion delay difference between X and Z is not constrained. Constraint analysis during CTS will identify the most efficient place to put this delay, which in may be at the multiplexer inputs. For example, to add delay to balance the B-Y path with the B-Z path, delay can be added at the right hand multiplexer input without increasing the insertion delay of the A-Y path.



Example – Balancing Independent Clock Trees

The next example below again has two clock trees, A and B, but with a single skew group, SG_AB. The skew group has two sources and all the sinks of clock tree A and clock tree B. CTS will balance all paths from A to the sinks and all paths from B to the sinks together.



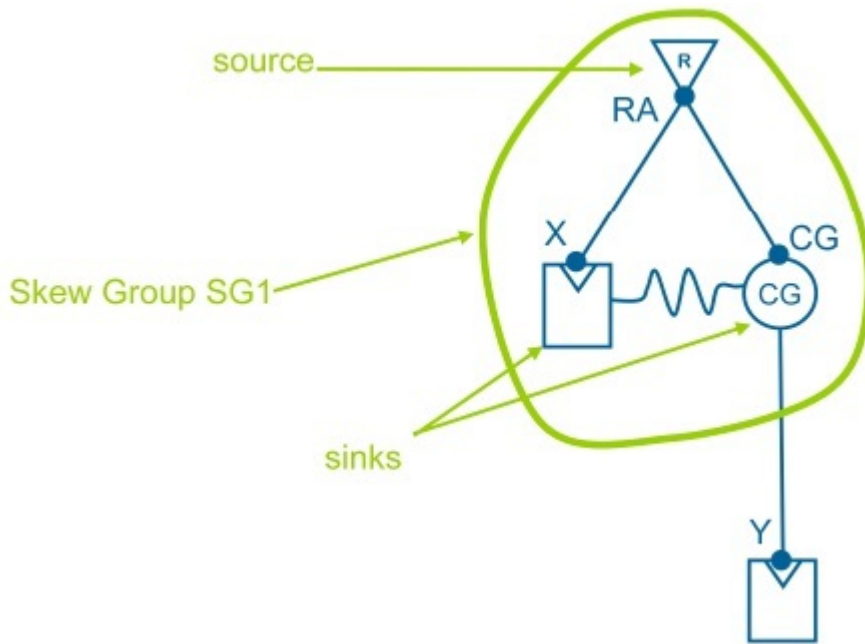
A variant of the above example would also have skew group, SG_A and skew group, SG_B corresponding to each of the two clock trees, which would be the default behavior of automatic clock tree specification. The user could then use `create_ccopt_skew_group -name SG_AB -balance_skew_group {SG_A SG_B}` to create a combined skew group.

Example – Balancing Flops with a Clock Gate

When using CCOpt-CTS it may be necessary to reduce the clock insertion delay to sinks at the source of paths to a clock gate to avoid a setup violation at the clock gate enable input. In the example below, this is done by creating an additional skew group, SG1, to balance the flip-flop pin X with the clock gate pin CG as follows:

```
create_ccopt_skew_group -name SG1 -sources RA -exclusive_sinks {X CG}
```

The pins, X and CG, are made exclusive sinks so that X is no longer an active sink in other existing skew groups.



With CCOpt, rather than CCOpt-CTS, an additional user skew group would not normally be required to do this as useful skew scheduling will automatically adjust the insertion delay of X and CG to optimize the setup slack at the clock gate enable input.

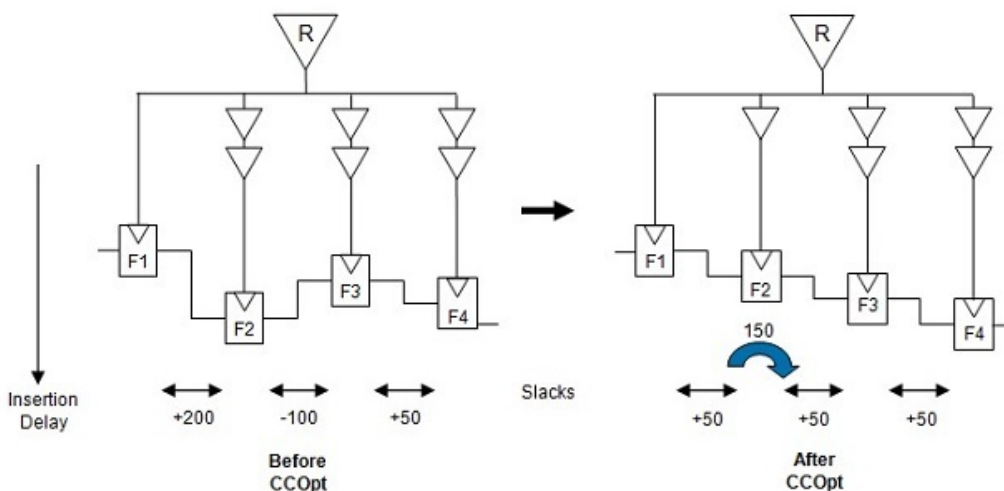
Deleting the Clock Tree Specification

The [delete_ccopt_clock_tree_spec](#) command can be used to remove all skew groups, clock trees, and associated data. However, this command does not reset property settings on pins, instances and other database entities. The [reset_ccopt_config](#) command can be used to remove both the clock tree specification and all CCOpt property settings.

Chains

CCOpt uses useful skew to adjust clock delays, therefore, moving slack between datapath stages. The limit of WNS optimization is not determined by a single flop-to-flop datapath stage but a chain of such paths. At each flop slack can be shifted from the capture or launch side as illustrated below.

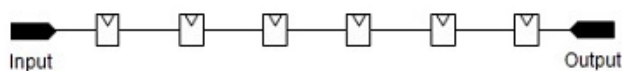
Moving Slack between Datapath Stages



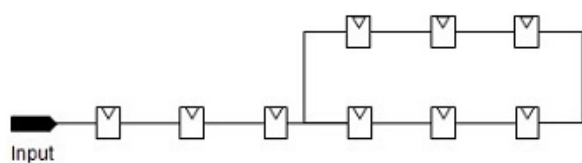
In the example above, CCOpt moves 150ps of slack from the F1 → F2 path and moves it to the F2 → F3 path to address the negative slack of -100ps. This is done by reducing the delay on the F2 clock path, illustrated by the removal of a buffer in the above simplified diagram. However, the ability to move slack between datapath stages is not unlimited. It must stop when the chain of paths either loops back on itself or reaches an input or output port. This gives rise to different types of chains:

- Input-to-output chain – a chain of flops starting at an input pin and ending at an output pin
- Input-to-loop chain – a chain of flops starting at an input pin and ending at a looped path
- Loop-to-output chain – a chain of flops starting at a looped path and ending at an output pin
- Looping chain – a chain of flops starting and ending at a looped path

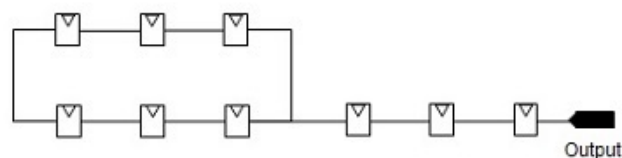
The different types of chains are shown below.



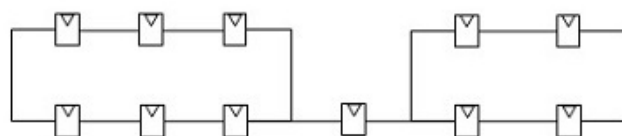
IO Chain



Input-to-loop chain



Loop-to-output chain

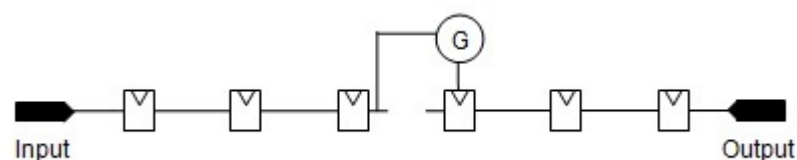


Looping chain

A variant of the input and output chains is a chain containing a flop which either does not launch or does not capture paths, for example if such paths are subject to [set_false_path](#) exception.

Chains can contain clock gates as illustrated below. CCOpt can adjust the clock insertion delay both to the clock gate and the flops during useful skew scheduling. Adjusting the clock insertion delay to a clock gate may impact the insertion delay to the gated flops, which in turn impacts the slack of timing paths launched by those flops.

Clock Gate in a Chain

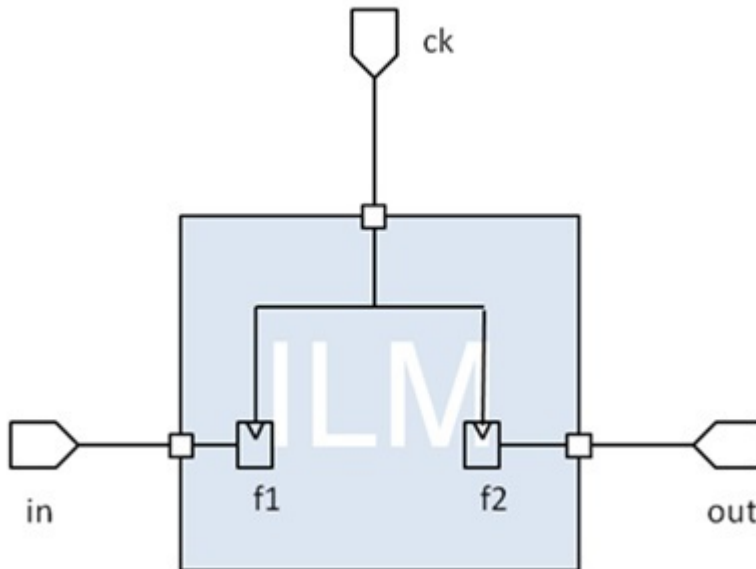


The design worst chain is the chain constructed by taking the global WNS path and expanding the chain around this path such that each path within the chain is a local WNS path. The worst chain is reported in the log during CCOpt design, and the format of this chain report is discussed further in the "Worst Chain" section.

Disjoint Chains

The worst chain may pass through an ILM partition or “.lib” macro. The example below illustrates an ILM partition in which a single clock input clocks both an input register and an output register inside the ILM. The example contains two timing paths, in-to-f1 and f2-to-out. CCOpt cannot independently adjust the insertion delay of flop f1 and flop f2 because the ILM contents are read-only.

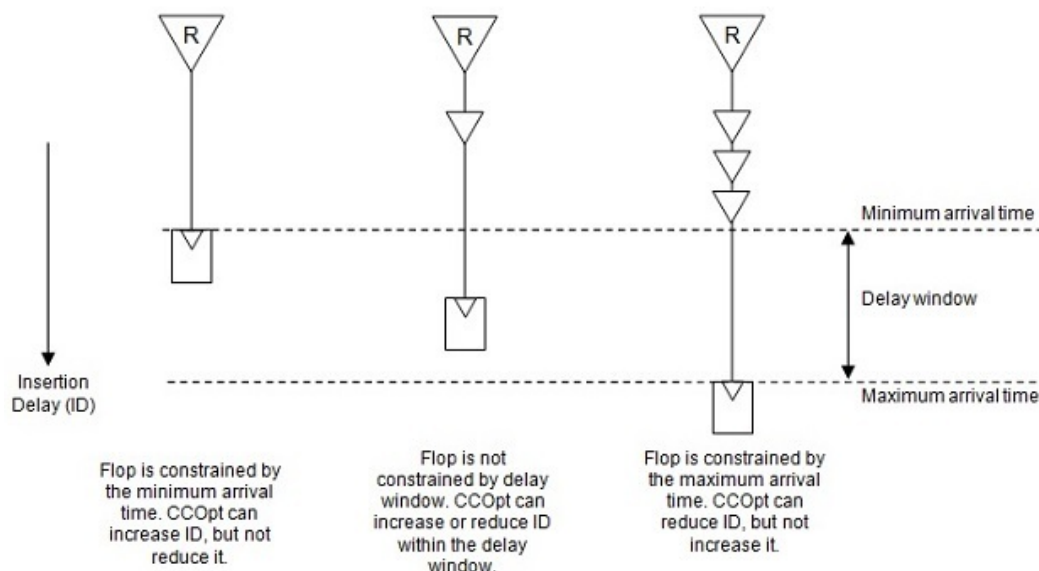
Partition or Macro in a Disjoint Chain



Constraint Windows

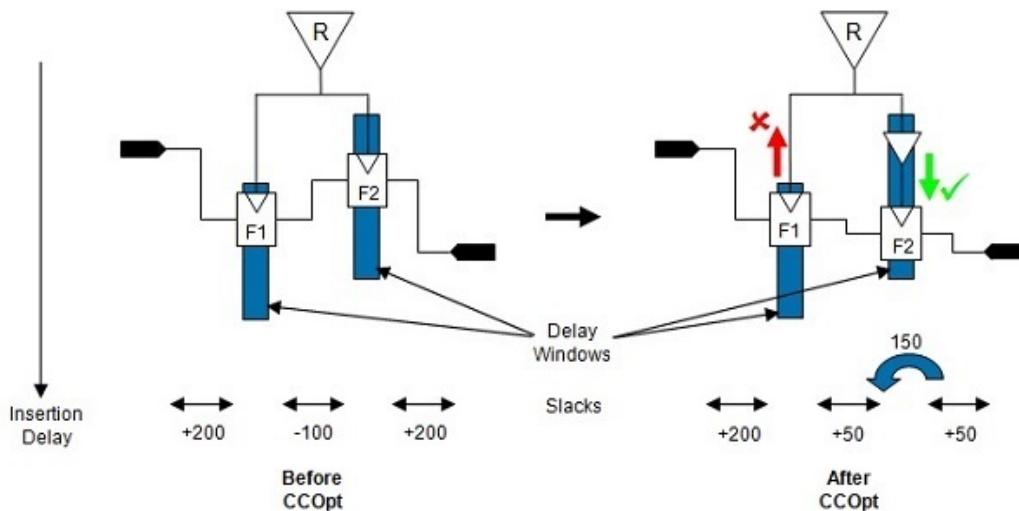
CCOpt determines a delay constraint window for every sink representing the minimum and maximum clock insertion delay (clock arrival time) for the sink. This is illustrated below.

Constraint Windows



When calculating delay constraint windows, CCOpt considers all the constraints applicable to a particular sink including physical constraints, for example the minimum buffering delay from the clustering step, skew group constraints and insertion delay limit. Useful skew can, therefore, only take place if permitted by the delay constraint window. Consider the example below.

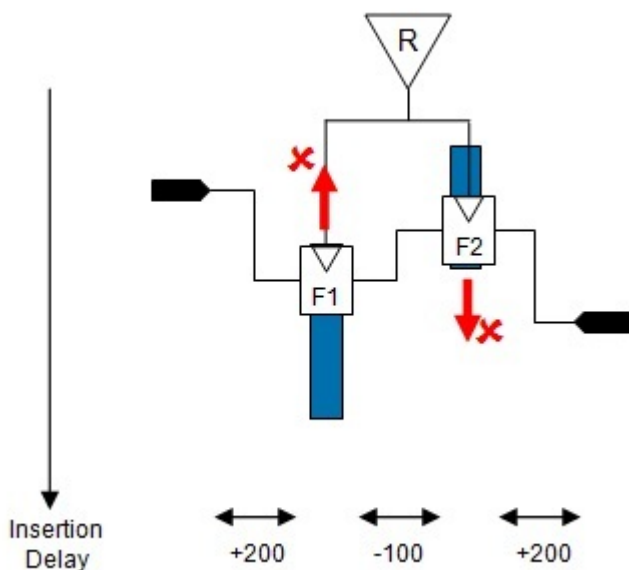
Skew Scheduling within Constraint Windows



In this example, flop F2 needs to be scheduled later than flop F1 to improve the negative slack of -100ps. To achieve this, CCOpt could decrease the insertion delay to flop F1 but F1 is already close to the top of the delay constraint window. Alternatively, increasing the insertion delay to flop F2, which is in the middle of its delay constraint window, permits the movement of 150ps of slack from launch side to the capture side of F2.

However, consider the same situation with different constraints and, therefore, different delay constraint windows. In the example below, CCOpt is unable to fix negative slack because of the delay constraint windows.

Skew Scheduling Restricted by Constraint Windows



CCOpt is unable to reduce the insertion delay to flop F1 because it is already at the top of its delay window. Similarly, CCOpt is unable to increase the insertion delay to flop F2 because it is already at the bottom of its delay window. Therefore, the negative slack between F1 and F2 cannot be fixed by useful skew. It might be possible to optimize the datapath between F1 and F2 further, but note that CCOpt will typically only skew clock sinks when datapath optimization is unable to progress.

Timing Windows

Further window types are the "chosen window" that appears in worst chain reports and "timing windows" that are viewable in the CCOpt Clock Tree Debugger. The chosen window represents an insertion delay range that useful skew scheduling would like a sink to be within, in order to progress timing optimization.

The timing window represents the final window used by the implementation step. Each sink is assigned a timing window such that so long as the sink is within the timing window the sink will not be at risk of degrading the high effort path group(s) WNS and will not adversely impact hold timing. Implementation is able to group sinks together that are physically nearby with overlapping timing windows such that clock tree area and power is reduced by avoiding the need to strictly balance less critical sinks.

For more information on worst chain reporting, both in the log and via the [report_ccopt_worst_chain](#) command, see the [Worst Chain](#) section.

Reporting

Get Commands

There are several `get_ccopt_*` commands that aid TCL scripting and combined with [get_ccopt_property](#) and other Innovus commands, for example [dbGet](#), aid the generation of custom checks and reports. The most commonly used `get_ccopt_*` commands are listed below. For each command, additional parameters may be necessary. For details of these commands, refer to the *Innovus Text Command Reference*.

Command Name	Usage
get_ccopt_clock_tree_cells	Returns all cells which are part of the clock tree graph. Cells that are leaf sinks are excluded.
get_ccopt_clock_tree_nets	Returns all nets that are part of the clock tree graph.
get_ccopt_clock_tree_sinks	Returns all clock tree sinks, that is automatically determined sinks, stop pins, ignore pins which are part of the clock tree graph.
get_ccopt_clock_trees	Returns all clock trees.
get_ccopt_effective_max_capacitance	Returns the value of the frequency-dependent effective maximum capacitance constraint that the software will apply at a given pin in the clock tree.
get_ccopt_skew_group_delay	Returns the insertion delay for a particular skew group or sink within a skew group.
get_ccopt_skew_group_path	Returns a path for a particular skew group or sink within a skew group.
get_ccopt_skew_groups	Returns all skew groups.
get_ccopt_preferred_cell_stripe	Returns a list of preferred cell stripes objects matching the supplied pattern.

Skew Groups

The [report_ccopt_skew_groups](#) -filename *file* command creates a report including a summary of all defined skew groups with insertion delay data per delay corner and the maximum and minimum insertion delay paths per skew group and delay corner. The optional `-histograms` parameter

can be used to include an insertion delay histogram per delay corner.

The main sections in the skew group report are:

- Skew group structure summary indicating number of active sinks
- Skew group summary indicating maximum and minimum insertion delay per skew group per late/early conditions of each delay corner
- Table of skew group minimum and maximum insertion delay paths per skew group and delay corner with sink pin names. Each path is given an ID number.
- Detailed path listings, using the same path ID number

An example of the skew group summary is illustrated below. A ‘*’ indicates that a target insertion delay or skew target was not met.

Skew Group Summary:

Timing Corner	Skew Group	ID Target	Min ID	Max ID	Avg ID	Skew Target Type	Skew Target	Skew
slow_max:setup.early	div_clk/functional_func_slow_max	-	0.651	0.809	0.737	ignored	-	0.158
	my_clk/functional_func_slow_max	-	0.922	1.139	1.096	ignored	-	0.217
	test_clk/functional_func_slow_max	-	0.652	0.871	0.827	ignored	-	0.218
slow_max:setup.late	div_clk/functional_func_slow_max	none	0.651	0.809	0.738	explicit	0.200	0.158
	my_clk/functional_func_slow_max	none	0.928	1.146	1.102	explicit	*0.200	0.217
	test_clk/functional_func_slow_max	none	0.663	0.880	0.836	explicit	*0.200	0.217
fast_min:hold.early	div_clk/functional_func_slow_max	-	0.212	0.284	0.256	ignored	-	0.072
	my_clk/functional_func_slow_max	-	0.310	0.417	0.392	ignored	-	0.106
	test_clk/functional_func_slow_max	-	0.228	0.335	0.310	ignored	-	0.107
fast_min:hold.late	div_clk/functional_func_slow_max	-	0.213	0.284	0.256	ignored	-	0.072
	my_clk/functional_func_slow_max	-	0.316	0.422	0.398	ignored	-	0.106
	test_clk/functional_func_slow_max	-	0.236	0.341	0.317	ignored	-	0.105

The `report_ccopt_skew_groups` command accepts various parameters to restrict the reporting to specified skew groups or delay corners, or to restrict to particular paths using `-through` and `-to` in a similar manner to the [report_timing](#) command. The `-summary` parameter can be used just to report the summary. For more details, see the *Innovus Text Command Reference*.

Note that the skew group report uses the same timing model as the CCOpt-CTS engine. To report on timing clocks, use the [report_clock_timing](#) command.

Clock Trees

The [report_ccopt_clock_trees](#) -filename *file* command creates a report including statistics per clock tree and statistics over all clock trees, including transition violations. The `-histograms` parameter can be used to enable histograms of various data and the `-list_special_pins` parameter will add a detailed listing of clock tree stop and ignore pins.

Clock Tree Network Structure

The [report_ccopt_clock_tree_structure](#) command reports the structure of the clock network as a text report. The syntax of the command is as follows:

```
[ -help ]
[ -check_type {setup | hold} ]
[ -clock_trees {string1 string2 ...} ]
[ -delay_corner corner ]
[ -delay_type {early | late} ]
[ -expand_below_generators ]
[ -expand_below_logic ]
[ -file name ]
[ -show_sinks ]
```

The parameters, `-expand_below_logic` and `-expand_below_generators` control how the report addresses clock convergence/reconvergence at clock logic cells and at clock generator paths, which are instances with more than one clock input. Such a multi-input instance will appear multiple times in the report. By default, the command aims for brevity, and will therefore emit the subtree below the multi-input instance once, at the first occurrence; and at subsequent occurrences will simply indicate that the fanout have been omitted. However, this behavior can be modified by specifying the `-expand_below_logic` and `-expand_below_generators` parameters.

When the `-expand_below_logic` parameter is specified, the subtree below each multi-input logic will be printed.

When the `-expand_below_generators` parameter is specified, each generated clock tree is expanded in full, possibly multiple times, within each parent tree(s), and is not reported at the top level.

The report can be customized to include or exclude the skew-group latencies at each sink. for this, you can use the `-show_sinks` parameter. By default, the clock sink networks, which are the non-generator flops/latches are not included in the report. Instead, a count of the number of sinks is displayed,for example. "... 209 sinks omitted". However, when this parameter is specified, per-skew group latencies for each sink are included.

The report identifies macros in the clock tree. This means that if your clock sink is a macro, the report will show "macro sink" instead of just "sink".

Sample reports are shown below.

Sample 1: Clock Tree Structure report for clock tree, `m_clk`

```
report_ccopt_clock_tree_structure -clock_trees {m_clk}
Clock tree m_clk:
TEST_CONTROL_INST/g137/ZN root output at (273.210,301.000), lib_cell
ND2D1BWP, level 1, slew 9.724ns, wire_cap 0.096pF, load_cap 0.452pF
\_ ... (21 sinks omitted)
\_ DMA_INST/CPF_LS_158_m_clk/I logic input at (169.210,300.720),
lib_cell LVLHLD2BWP, level 2, slew 9.725ns
| DMA_INST/CPF_LS_158_m_clk/Z logic output at (169.770,300.860),
lib_cell LVLHLD2BWP, level 2, slew 0.449ns, wire_cap 0.010pF,
load_cap 0.006pF
| \_ ... (8 sinks omitted)
| \_ DMA_INST/RC_CG_HIER_INST0/RC_CGIC_INST/CP cgate input at
(115.450,305.760), lib_cell CKLNQD1BWP, level 3, slew 0.449ns
| DMA_INST/RC_CG_HIER_INST0/RC_CGIC_INST/Q cgate output at
(116.290,305.900), lib_cell CKLNQD1BWP, level 3, slew 0.320ns,
wire_cap 0.004pF, load_cap 0.005pF
| \_ ... (8 sinks omitted)
\_ CONV_INST/CPF_LS_159_m_clk/I logic input at (199.170,320.880),
lib_cell LVLHLD2BWP, level 2, slew 9.724ns
| CONV_INST/CPF_LS_159_m_clk/Z logic output at (199.730,321.020),
lib_cell LVLHLD2BWP, level 2, slew 0.864ns, wire_cap 0.030pF,
load_cap 0.015pF
| \_ ... (15 sinks omitted)
| \_ CONV_INST/RC_CG_HIER_INST1/RC_CGIC_INST/CP cgate input at
(110.130,313.320), lib_cell CKLNQD1BWP, level 3, slew 0.865ns
```

```

| | CONV_INST/RC_CG_HIER_INST1/RC_CGIC_INST/Q cgate output at
(109.290,313.460), lib_cell CKLNQD1BWP, level 3, slew 0.338ns,
wire_cap 0.004pF, load_cap 0.006pF
| | \_ ... (9 sinks omitted)
| \_ RESULTS_CONV_INST/RC_CG_HIER_INST2/RC_CGIC_INST/CP cgate input
at (155.490,341.040), lib_cell CKLNQD1BWP, level 3, slew 0.865ns
| | RESULTS_CONV_INST/RC_CG_HIER_INST2/RC_CGIC_INST/Q cgate output
at (156.330,341.180), lib_cell CKLNQD1BWP, level 3, slew 0.668ns,
wire_cap 0.011pF, load_cap 0.010pF
| | \_ ... (16 sinks omitted)
| \_ CONV_INST/RC_CG_HIER_INST3/RC_CGIC_INST/CP cgate input at
(154.650,354.760), lib_cell CKLNQD1BWP, level 3, slew 0.865ns
| | CONV_INST/RC_CG_HIER_INST3/RC_CGIC_INST/Q cgate output at
(155.490,354.620), lib_cell CKLNQD1BWP, level 3, slew 0.636ns,
wire_cap 0.010pF, load_cap 0.010pF
| | \_ ... (16 sinks omitted)
| \_ CONV_INST/RC_CG_HIER_INST6/RC_CGIC_INST/CP cgate input at
(111.110,338.520), lib_cell CKLNQD1BWP, level 3, slew 0.865ns
| | CONV_INST/RC_CG_HIER_INST6/RC_CGIC_INST/Q cgate output at
(110.270,338.660), lib_cell CKLNQD1BWP, level 3, slew 0.246ns,
wire_cap 0.003pF, load_cap 0.004pF
| | \_ ... (6 sinks omitted)
| \_ CONV_INST/RC_CG_HIER_INST7/RC_CGIC_INST/CP cgate input at
(112.790,324.520), lib_cell CKLNQD1BWP, level 3, slew 0.865ns
| | CONV_INST/RC_CG_HIER_INST7/RC_CGIC_INST/Q cgate output at
(111.950,324.380), lib_cell CKLNQD1BWP, level 3, slew 0.349ns,
wire_cap 0.005pF, load_cap 0.005pF
| | \_ ... (8 sinks omitted)
\_ SPI_INST/RC_CG_HIER_INST17/RC_CGIC_INST/CP cgate input at
(133.630,125.440), lib_cell CKLNQD1BWP, level 2, slew 9.725ns
SPI_INST/RC_CG_HIER_INST17/RC_CGIC_INST/Q cgate output at
(134.470,125.300), lib_cell CKLNQD1BWP, level 2, slew 0.335ns,
wire_cap 0.010pF, load_cap 0.005pF
  \_ ... (8 sinks omitted)

```

Sample 2: Clock Tree Structure for clock tree, m_digit_clk to include skew-group latencies

```

report_ccopt_clock_tree_structure -clock_trees {m_digit_clk} -
show_sinks
Clock tree m_digit_clk:
TEST_CONTROL_INST/g141/ZN root output at (122.850,294.000), lib_cell
IOA21D1BWP, level 1, slew 0.230ns, wire_cap 0.006pF, load_cap 0.005pF
\_ REG_INST/digit_out_reg[0]/CP sink input at
(100.310,293.160),lib_cell SDFSNQD1BWP, level 2, slew 0.230ns,
skew_groups {m_digit_clk/PM_FUNC 0.001ns}
\_ REG_INST/digit_out_reg[1]/CP sink input at (98.910,294.280),
lib_cell SDFSNQD1BWP, level 2, slew 0.230ns, skew_groups
{m_digit_clk/PM_FUNC 0.001ns}
\_ REG_INST/digit_out_reg[2]/CP sink input at (95.270,295.680),
lib_cell SDFSNQD1BWP, level 2, slew 0.230ns, skew_groups
{m_digit_clk/PM_FUNC 0.001ns}
\_ REG_INST/digit_out_reg[3]/CP sink input at (94.430,294.280),

```

```
lib_cell SDFSNQD1BWP, level 2, slew 0.230ns, skew_groups
{m_digit_clk/PM_FUNC 0.001ns}
\_ REG_INST/digit_out_reg[4]/CP sink input at (92.050,293.160),
lib_cell SDFSNQD1BWP, level 2, slew 0.230ns, skew_groups
{m_digit_clk/PM_FUNC 0.001ns}
\_ REG_INST/digit_out_reg[5]/CP sink input at (92.050,296.800),
lib_cell SDFSNQD1BWP, level 2, slew 0.230ns, skew_groups
{m_digit_clk/PM_FUNC 0.001ns}
\_ REG_INST/digit_out_reg[6]/CP sink input at (92.050,298.200),
lib_cell SDFSNQD1BWP, level 2, slew 0.230ns, skew_groups
{m_digit_clk/PM_FUNC 0.001ns}
\_ REG_INST/digit_out_reg[7]/CP sink input at (92.050,300.720),
lib_cell SDFSNQD1BWP, level 2, slew 0.230ns, skew_groups
{m_digit_clk/PM_FUNC 0.001ns}
\_ REG_INST/flag_out_reg/CP sink input at (93.170,299.320), lib_cell
SDFSNQD1BWP, level 2, slew 0.230ns, skew_groups {m_digit_clk/PM_FUNC
0.001ns}
```

Timing Data for Reports

The timing engine is used to calculate the timing data that is displayed in the clock tree and skew group reports. By default, this is invalidated when you generate the reports, so all the timing data is recalculated. This increases the time taken to generate the reports. Use the `-no_invalidate` parameter of the [report_ccopt_clock_trees](#) and the [report_ccopt_skew_groups](#) commands to specify that the timing engine should not be invalidated and that the existing timing data, if any, should be used in the reports.

Worst Chain

For an explanation about the concept of chains, see the [Chains](#) section.

The worst chain is reported from time to time in the log during CCOpt and an examination of the log may help identify reasons limiting timing optimization. In addition, the [report_ccopt_worst_chain](#) command can be used to report the worst timing chain after [ccopt_design](#) has completed, but note that this will reflect the current worst chain, not the worst chain during optimization.

The illustration below shows a perfectly balanced worst chain. Each sequential element in the chain is identified by a “cell:name” line with ASCII art on the left representing the chain connectivity. In this example, there is a loop between flops A and B. The data between each sequential element summarizes the combinational path between adjacent sequential elements. For example, the timing slack is identified, and the WNS marked with “*WNS*”. In this example, the slack between each stage is identical suggesting that it is not possible to further move slack between stages. Such a chain is balanced.

Example of Worst Chain

Worst chain (Setup):

=====

Equal slacks

```
, -o cell:uls/flopA
| | pin:.../clk @+878ps constraint=(-226ps,+379ps) chosen=(-0ps,+2ps)
| | location=(910.335,1230.822) slew=(launch: 28ps, capture: 28ps)
| | slack -68ps pin:.../q -> pin:.../d (distance: 453.123um)
| | delays=(launch: 290ps, datapath: 556ps, capture: 64ps)
`-o cell:uls/flopB
| | pin:.../clk @+652ps constraint=(-50ps,+589ps) chosen=(-0ps,+0ps)
| | location=(862.335,1250.822) slew=(launch: 28ps, capture: 28ps)
| | *WNS* -68ps pin:.../q -> pin:.../d (distance: 1232.345um)
| | delays=(launch: 64ps, datapath: 782ps, capture: 290ps)
o cell:uls/flopC
| | pin:.../clk @+867ps constraint=(-226ps,+381ps) chosen=(-0ps,+3ps)
| | location=(910.335,1230.822) slew=(launch: 28ps, capture: 28ps)
| | slack -68ps pin:.../q -> pin:.../d (distance: 372.312um)
| | delays=(launch: 290ps, datapath: 556ps, capture: 64ps)
...
```

The two diagrams below label the various fields in the worst chain report.

Worst Chain Data - Diagram 1

Constraint window

Chosen window

Delay under fragment

```
, -o cell:uls/flopA
| | pin:.../clk @+878ps constraint=(-226ps,+379ps) chosen=(-0ps,+2ps)
| | location=(910.335,1230.822) slew=(launch: 28ps, capture: 28ps)
| | slack -68ps pin:.../q -> pin:.../d (distance: 453.123um)
| | delays=(launch: 290ps, datapath: 556ps, capture: 64ps)
`-o cell:uls/flopB
```

Slack

Pin location

Clock pin transition

Worst Chain Data - Diagram 2

Total path length

```
, -o PM_INST/power_switch_enable_reg_reg
| | .../CP @(unknown) constraint=(unknown) chosen=(unknown)
| | location=(209.510,145.600) slew=(launch: 0.000ns, capture: 0.000ns)
| | slack 1.036ns .../Q -> .../D (distance: 42.280um)
| | delays=(launch: 0.000ns, datapath: 0.964ns, capture: 0.000ns, adjust: 2.000ns)
| | launch/capture clock m_clk in analysis view AV_HL_FUNC_MAX_RC1
| | path group reg2reg
`-o PM_INST/power_switch_enable_reg_reg
```

Path group name

Clock launch path delay

Data path delay

Clock capture path delay

The above labels are described in detail in the table below.

Filed Name	Description
Constraint window	The constraint window is the range of permissible insertion delay modification subject to the minimum buffering delay from the clustering step, the automatic insertion delay limit and optional user skew constraints. For more information, see the Constraint Windows section. In this example, the -226ps indicates that the sink insertion delay can be scheduled up to 226ps earlier and the +379ps indicates that the sink can be scheduled up to 379ps later.
Chosen window	The chosen window represents the insertion delay range, relative to the current insertion delay, within which useful skew scheduling desires to place the sink.
Slack	The datapath slack between two sequential elements in the chain.
Delay under fragment	This is an internal number representing the delay between the non-buffer parent of a sink and the sink.
Pin location	The placement coordinate of the sink pin.
Clock Pin transition	The transition time at the sink pin, both for launch and for capture.
Clock launch path delay	The launch clock arrival time.
Data path delay	The datapath delay.
Total path length	The physical length of the path obtained by summing the distance between each pin pair along the path.
Clock capture path delay	The capture clock arrival time.
Path group name	The name of the path group for each path.

The example below illustrates a worst chain report where the slacks on either side of flopB are unequal. To further improve the WNS, it is desirable to move slack from the launch side of flopB to the capture side of flopB. To do this would require the insertion delay of flopB to be increased but this is not possible because flopB is at the bottom of the constraint window. The source of such a limit is the automatic insertion delay limit as discussed in the [Restricting CCOpt Skew Scheduling](#) section.

Example of Worst Chain – sink at bottom of constraint window

```
, -o cell:uls/flopA
| | pin:.../clk @+600ps constraint=(-226ps,+200ps) chosen=(-0ps,+2ps)
| | location=(910.335,1230.822) slew=(launch: 28ps, capture: 28ps)
| | *WNS* -68ps pin:.../q -> pin:.../d (distance: 453.123um)
| | delays=(launch: 64ps, datapath: 556ps, capture: 290ps)
`-o cell:uls/flopB
| | pin:.../clk @+800ps constraint=(-50ps,0ps) chosen=(-0ps,+0ps)
| | location=(862.335,1250.822) slew=(launch: 28ps, capture: 28ps)
| | slack -45ps pin:.../q -> pin:.../d (distance: 1232.345um)
```

next path has better slack

0ps – sink at bottom of constraint window

Halo Violations

Clock cell halo violations can be reported with the [report_ccopt_cell_halo_violations](#) command. For more information about cell halos, see the [Cell Halos](#) section.

Cell Name Information

All new cell instances created by CCOpt have an identifying code in their name to let you determine why that cell was created. For example:

cuk Cts: Unknown creator, will not appear in the netlist.
cex Cts: Existing cells in the clock tree which cannot be removed
coi Cts: Cells created as a result of cancelling out inversions
ccl Cts: Created by the clustering process to meet the slew target.....

To view all the available codes and their meanings, run the [show_ccopt_cell_name_info](#) command.

Clock Tree Convergence

The [report_ccopt_clock_tree_convergence](#) command reports statistics on the number of paths leading to clock tree sinks, and a list of the top 10 sinks with the most paths. In the example report below, 5 design IOs (primary input/outputs) have 1400 clock paths leading to them. These sinks are likely to be problematic and need further investigation. For details, see the [Clock Tree Convergence](#) section.

Convergence above clock sinks
=====

Number of paths to sink	PIOs	DFFs	Other sinks	Total

1	1	10118	0	10119
8	54	2134	183	2371
16	13	311	18	342
32	2	0	0	2
700	3	164	5	172
1400	5	0	0	5

Sinks with most paths leading to them
=====

Sink	Number of paths

Owest/nout[0]	1400
Owest/nout[1]	1400
Owest/nout[2]	1400
Owest/nout[3]	1400
Owest/nout[4]	1400

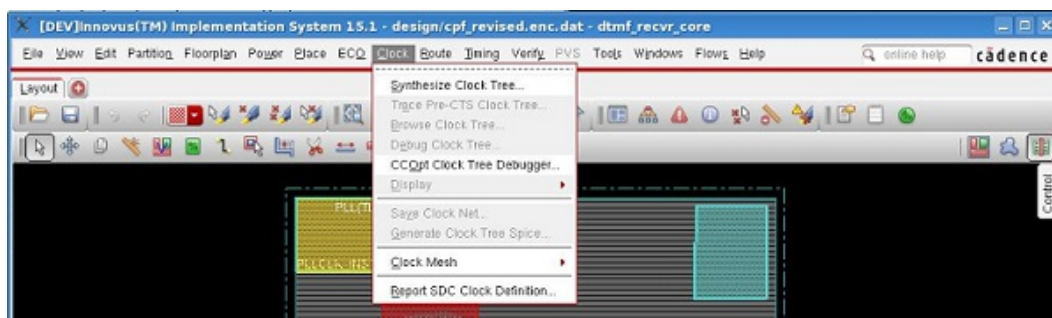
CCOpt Clock Tree Debugger

The CCOpt Clock Tree Debugger (CTD) provides a graphical interface to explore clock trees and provides debugging capabilities to aid understanding of CCOpt-CTS and CCOpt results. The interface is based on a top-down tree view of all defined clock trees with the vertical axis representing insertion delay. Cells and nets can be colored by various attributes, sections of the clock tree graph can be hidden and unhidden to facilitate navigation of complex clock architectures, and cross probing with the layout view is supported.

Many of the features of the debugger, for example, coloring are self-explanatory from exploring the interface. This section discusses some of the more in-depth features. For details of the *CCOpt Clock Tree Debugger*, see the "CCOpt Clock Tree Debugger" section in the [Clock Menu](#) chapter in *Innovus Menu Reference*.

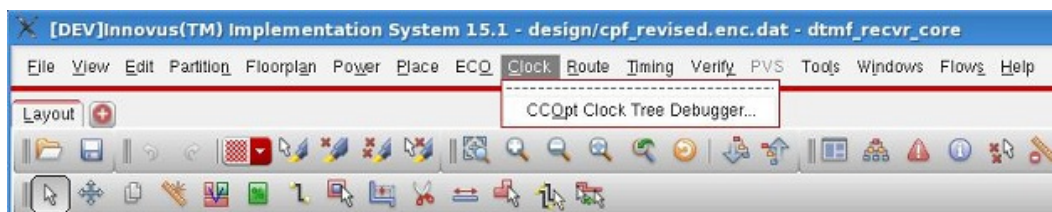
Launching the CCOpt CTD

The tool can be accessed from the *Clock Menu* of Innovus. Choose *Clock < CCOpt Clock Tree Debugger*.



Note that this menu selection will only be visible if there is a CCOpt clock tree specification loaded, specifically that one or more clock trees are defined. Alternatively, the [ctd_win](#) command can be used from a script to launch the CTD, and optional `-id` and `-title` parameters permit the window to be customized. This permits the CTD to be open and ready for use, for example at the end of an overnight run.

The complete submenu as shown above is visible only when the [setCTSMode](#) `-engine` option is set to `ck`. For all other CCOpt engine options, `-ccopt`, `auto`, and `ccopt_from_edi_spec`, only *CCOpt Clock Tree Debugger* submenu is visible. This is shown below.



Note: Deleting any clock tree or skew group definitions will automatically close all open CTD windows. Multiple windows may be opened, and the following commands permit manipulation of the CTD windows:

Command Name	Usage
ctd_win	Open a debugger window. An ID and title can be optionally specified. The ID is used to identify this particular window when using the other commands below.
close_ctd_win	Closes either the specified window by ID, all windows, or the most recently active window.
get_ctd_win_id	Get the ID of the most recently active window or the IDs of all open windows.
get_ctd_win_title	Get the title if the specified window by ID, or the titles of all open windows.
set_ctd_win_title	Set the title of the most recently active window or the specified window by ID.

[ctd_trace](#)

Highlight the path to a sink in the active debugger window.

For more information about the above commands, see the *Innovus Text Command Reference*.

CCOpt CTD Interface

The main components in the CTD window are shown below.

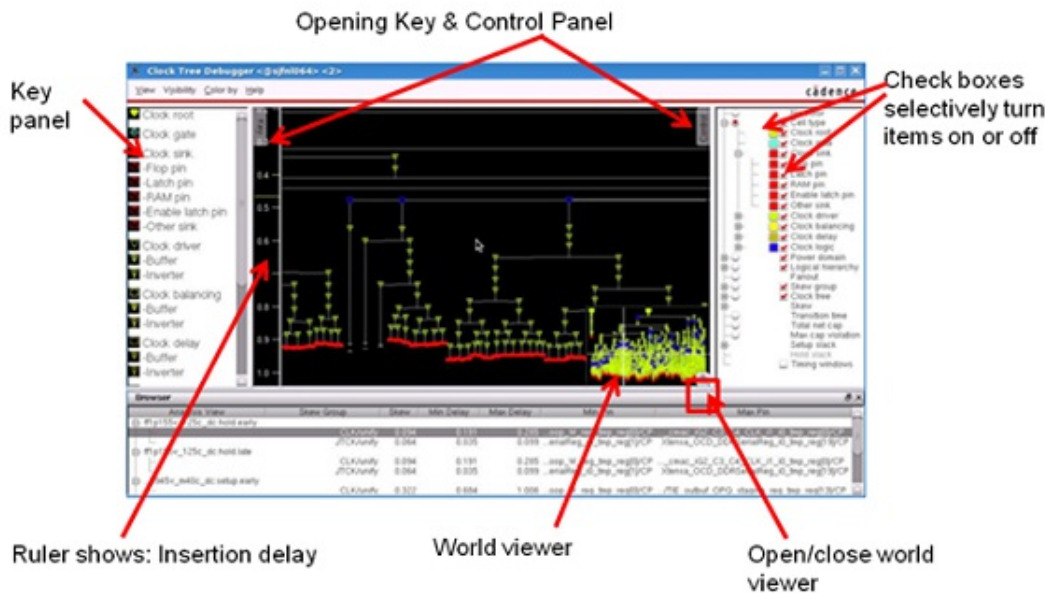
CCOpt CTD – Main Window Components



- **Clock Tree Viewer** – Displays a top-down tree view of clock trees.
- **World Viewer** – Provides an overview even whilst the Clock Tree Viewer is zoomed in on a smaller region. Clicking in the world viewer will navigate to that area.
- **Control Panel** – Contains controls to determine visibility of different object types and coloring. Additional controls are available via the *View*, *Visibility*, and *Color by* menu items.
- **Key Panel** – A reference key indicating symbols and/or coloring used within the Clock Tree Viewer.
- **Path Browser** – Displays skew group path summary data in a table. Double-clicking on a row or using the right-click context menu permits opening the *Clock Path Analyzer* window. By default, the Path Browser opens at the bottom of the window. The *Clock Path Analyzer*, when invoked, replaces the *Clock Path Browser*.

By default, the *Control Panel* and *Key Panel* are hidden. These panels can be exposed or hidden as illustrated below.

CCOpt CTD – Opening the Key and Control Panel



Key Features of the CTD

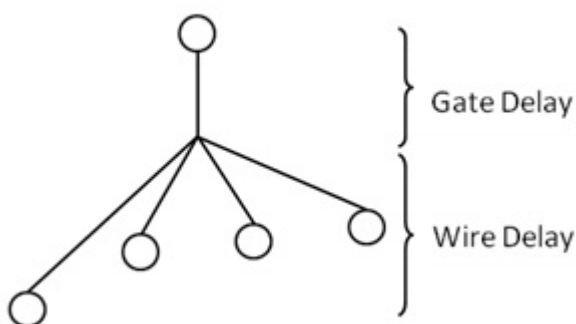
The key features of the CTD are briefly explained in the subsequent sections.

For details of the *CCOpt Clock Tree Debugger*, see the "CCOpt Clock Tree Debugger" section in the [Clock Menu](#) chapter in *Innovus Menu Reference*.

Clock Tree Representation

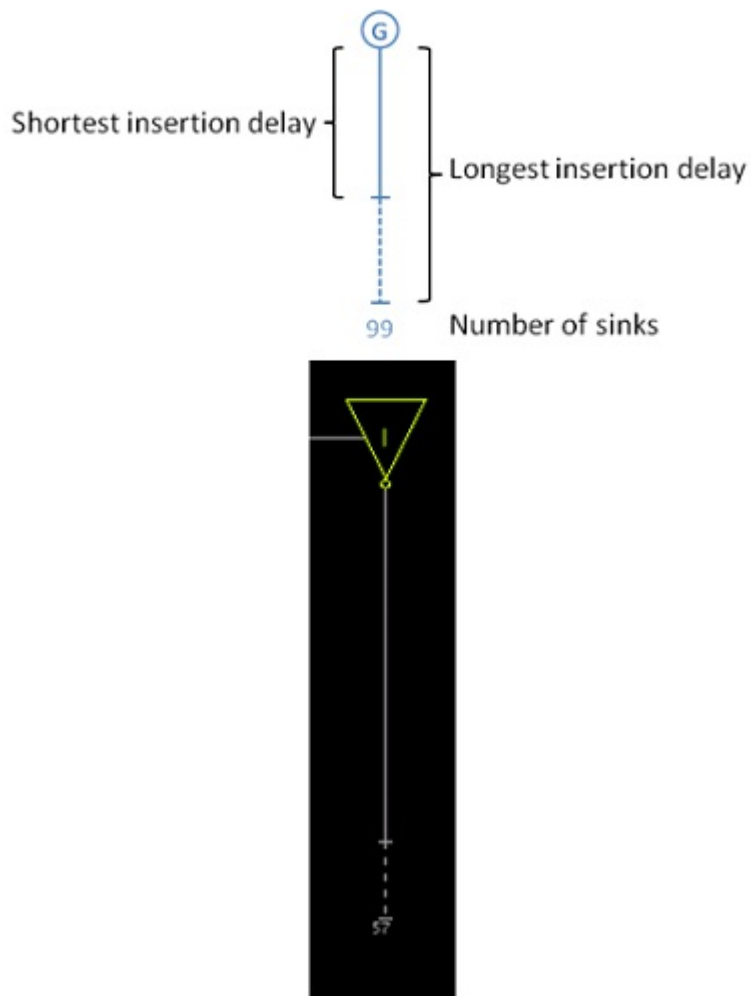
Clock trees are drawn in a top-down tree like manner with the vertical axis representing insertion delay. Different symbols are used for differing cell types, for example buffers, inverters, clock gates, logic and sinks. Gate and wire delay are separately represented, as illustrated below.

Gate and Wire Delay



Expanding and Collapsing Sub-trees

Any node in the tree may be either expanded or collapsed. The sub-trees of collapsed nodes are shown as a vertical summary bar indicating the maximum and minimum insertion delay below the node, and the number of sinks in the sub-tree, as illustrated below.

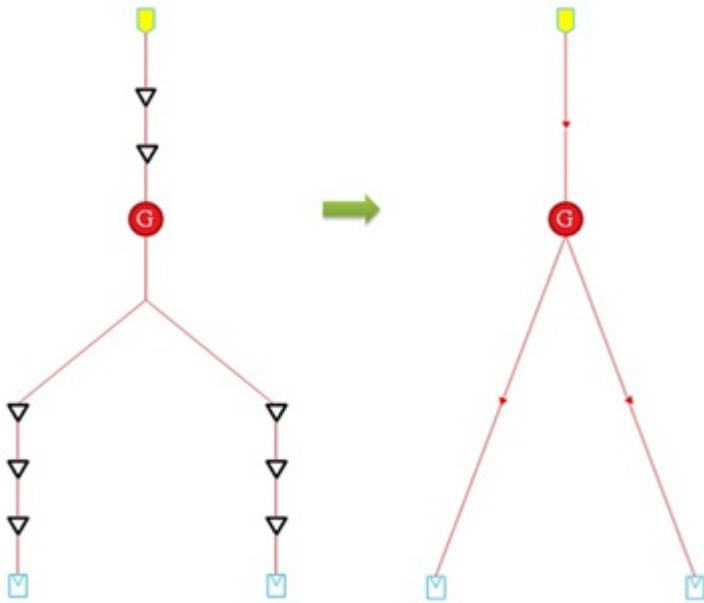


The expanded and collapsed state of a node may be toggled by double-clicking on the node, or using the Expand/Collapse item on the context menu. Additionally, a sub-tree can be marked as un-collapsible by using the context menu. An un-collapsible sub-tree will not be collapsed when its parent is collapsed.

Simplification





The *View – Simplify* option in the menu bar can be used to further manage visibility, including the following:

- *Mark All Collapsible* – Mark all nodes as collapsible.
- *Collapse all* – Collapse all sub-trees such that each clock tree is fully collapsed.
- *Expand All by Skew Group* – Expand all sub-trees that pass through the specified skew group.
- *Abstraction* – Specified cell types or cell instances can be abstracted using the *View -> Simplify -> Abstract* menu option. Abstracted cells are simply omitted, as represented in the diagram below where buffer cells, but not clock gates, have been abstracted.



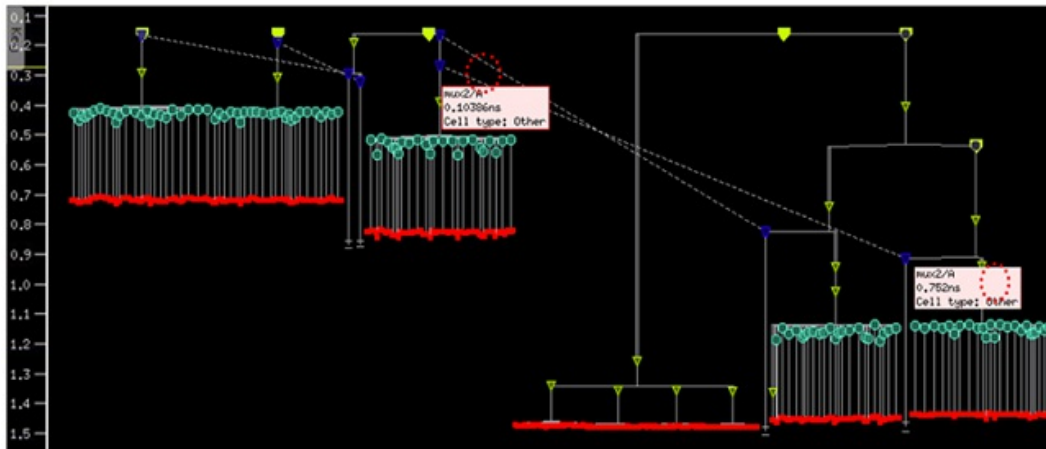
Context Menu

Right-clicking on a cell opens a context menu. This menu permits various operations to be performed, such as copying the cell name, highlighting the cell, highlighting paths to the cell, opening a schematic viewer, or opening cell properties.

<u>S</u> et Window Title	▶
Copy Name	
Highlight	▶
Dehighlight	
Highlight Clock Path	▶
Dehighlight All	
 Attribute Editor...	
 Design Browser...	
 Schematic Viewer (Cone)	▶
 Schematic Viewer (Module)	▶
Show Multiple Corner...	
<u>S</u> elect by Level	▶
Collapse Below	▶
<u>C</u> ollapse subtree	
<u>A</u> bstract this node	
<u>U</u> nabstract this node	
<u>E</u> xpand all subtrees	
<u>E</u> xpand all sinks	
<u>M</u> ark Uncollapsible	

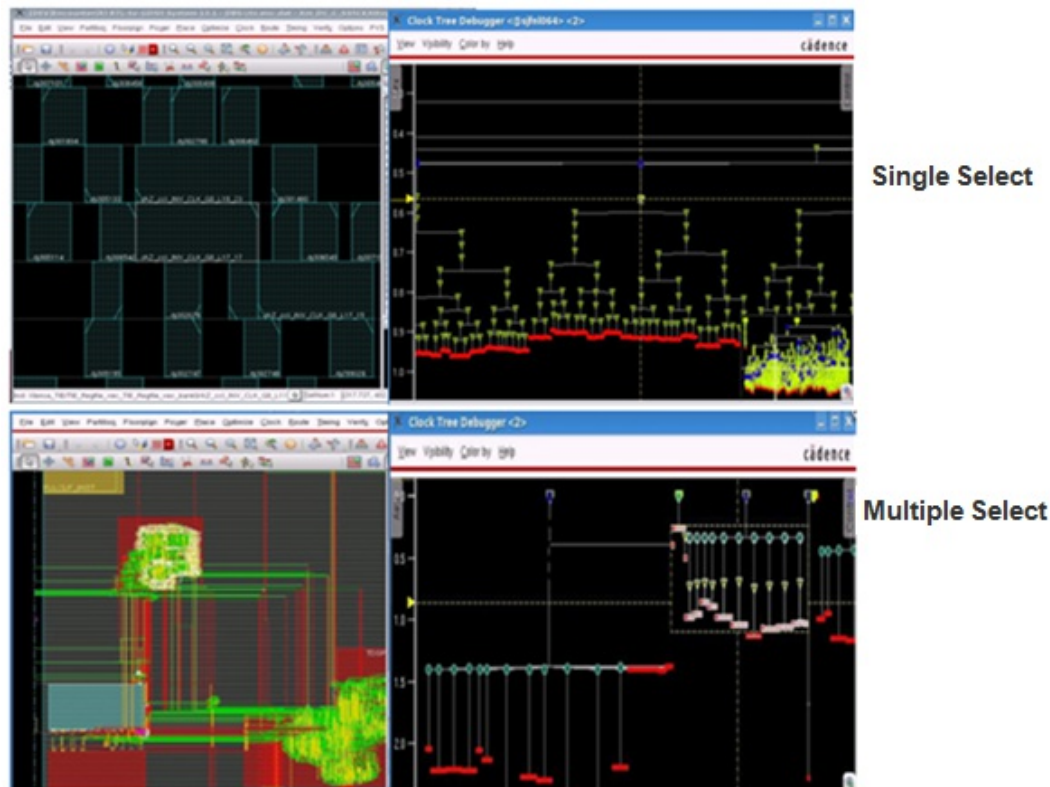
Multiple Input Cells

A multiple input cell, for example a multiplexer, can exist in more than one clock tree. Multiple input cells are shown in multiple clock trees, but the sub-tree underneath the cell can only be expanded in one clock tree at a time. A dotted line is drawn between the instances of the same cell as illustrated below.



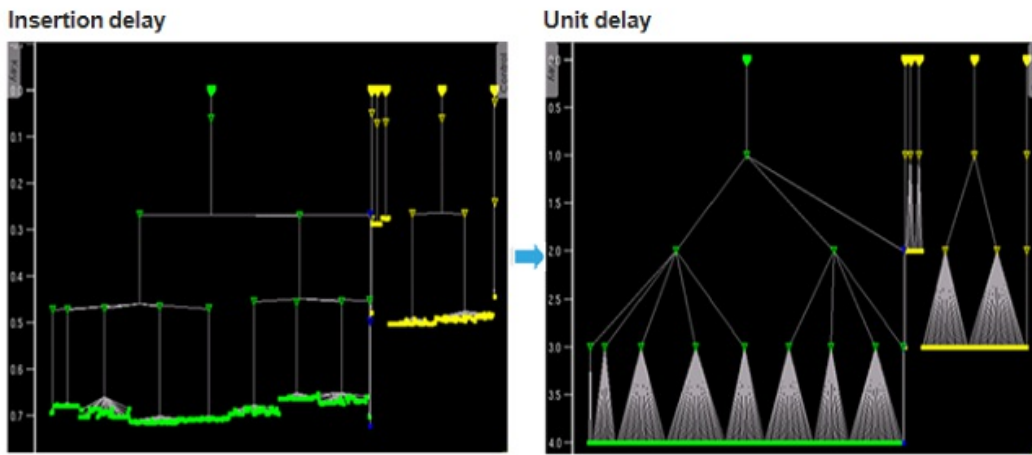
Cross Probing

When an object is selected in the main Innovus layout window it will also be selected in the CTD window. Conversely, objects selected in the debugger window will be selected in the layout window. The right-hand mouse button can be used to draw a bounding box to perform multiple selections.



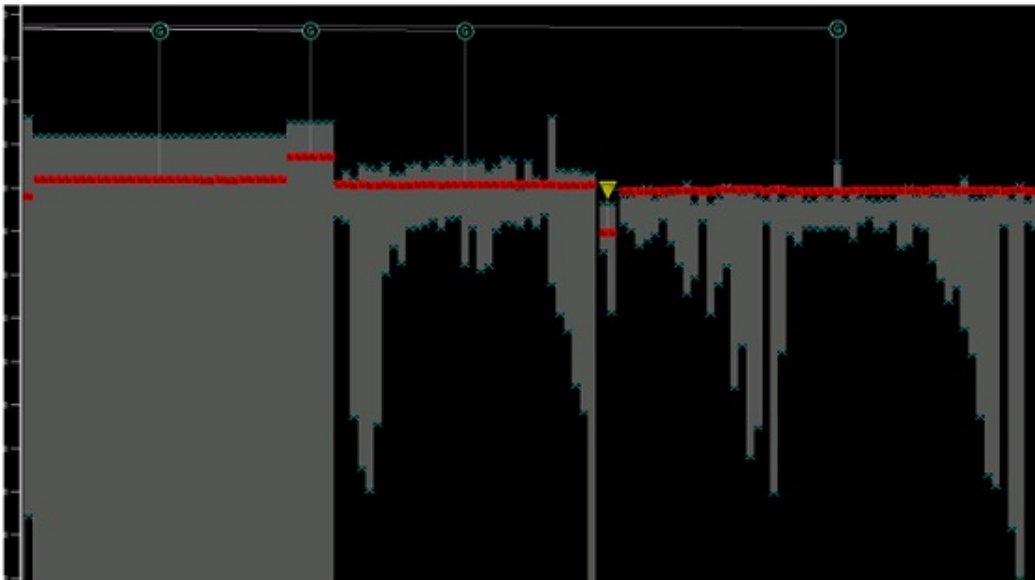
Unit Delay

The *Visibility – Unit Delay* menu option changes to a unit delay model where each cell has a delay of 0 and each wire a delay of 1.0. This mode is useful for inspecting the clock graph structure before running CCOpt or CCOpt-CTS.



Timing Windows

After running CCOpt, the timing windows can be shown by clicking on the *Timing windows* option in the *Control Panel*. For each sink, the window is drawn in a grey color as shown in the example below. Timing windows are discussed further in the Timing Windows section.



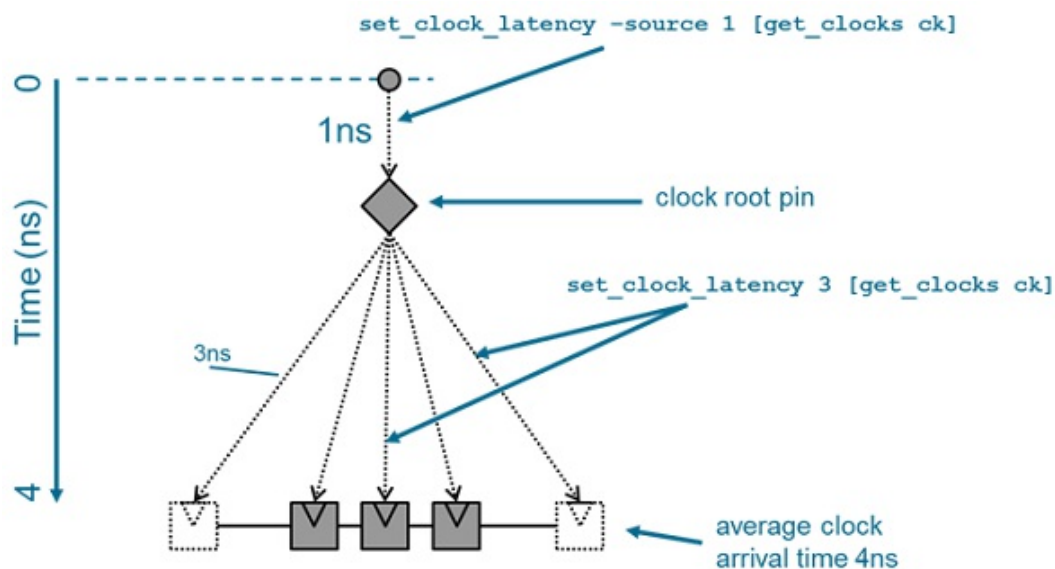
Additional Topics

Source Latency Update

Source latency update is performed to ensure that after CTS when clocks are switched to propagated mode that I/O timing and inter-clock timing is consistent with the ideal mode timing model. Stated succinctly:

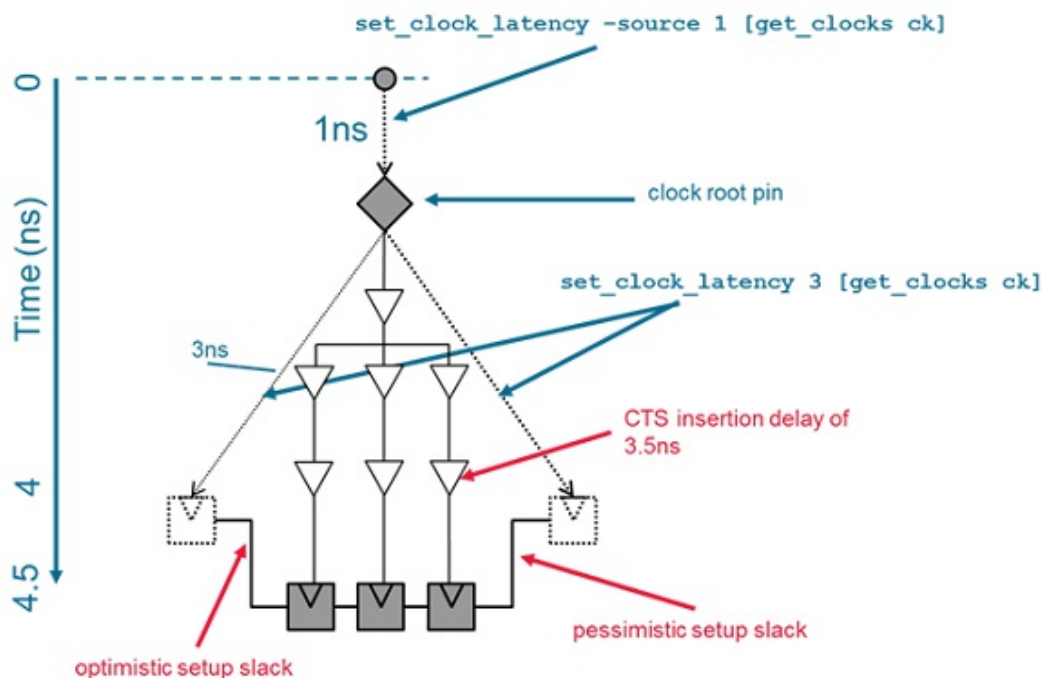
- The source latency update step updates the source latencies at clock root pins such that the per clock average clock arrival time is identical postCTS as it was preCTS.
- This mechanism is best explained using an example. The diagram below represents the before CTS ideal clock mode timing. In this example, there is a single clock with a clock source latency of 1ns and a network latency of 3ns. Therefore, the average clock arrival time at both the I/O pins (represented by dotted flops) and the real sinks is 4ns.

Source Latency Update – Before CTS



The next diagram below illustrates what would happen if CTS were performed but instead of achieving a 3ns insertion delay CTS achieves a 3.5ns insertion delay. The clock arrival time at the I/O pins is unchanged, but the clock arrival time at the real flops is now 3.5ns. This results in optimistic setup slack on input paths and pessimistic setup slack on output paths.

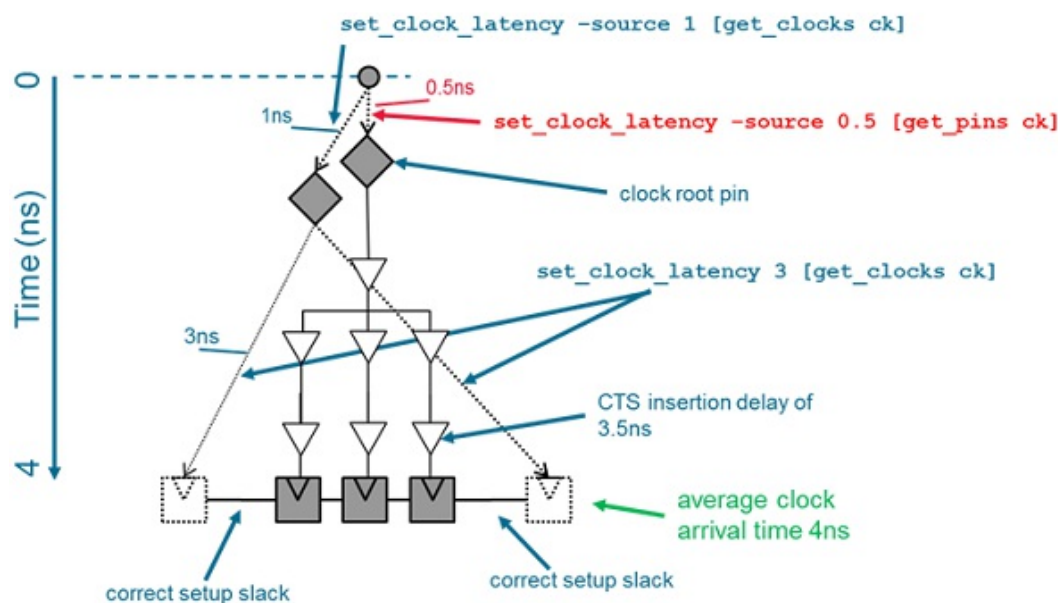
Source Latency Update – CTS without Source Latency Update



The next diagram illustrates the effect of source latency update. The clock source latency and network latency are unchanged, and the I/O pin timing is unchanged. The clock root pin has the source latency overridden to be 0.5ns instead of 1ns. This adjusts the clock arrival time at the real sinks such that it remains at 4ns. The input paths and output paths are now timed in the manner which was intended preCTS. Unlike other 'I/O latency modification' schemes, this scheme operates correctly in the presence of multiple clock domains communicating with I/O pins without any need for averages or need to match up virtual clocks with real

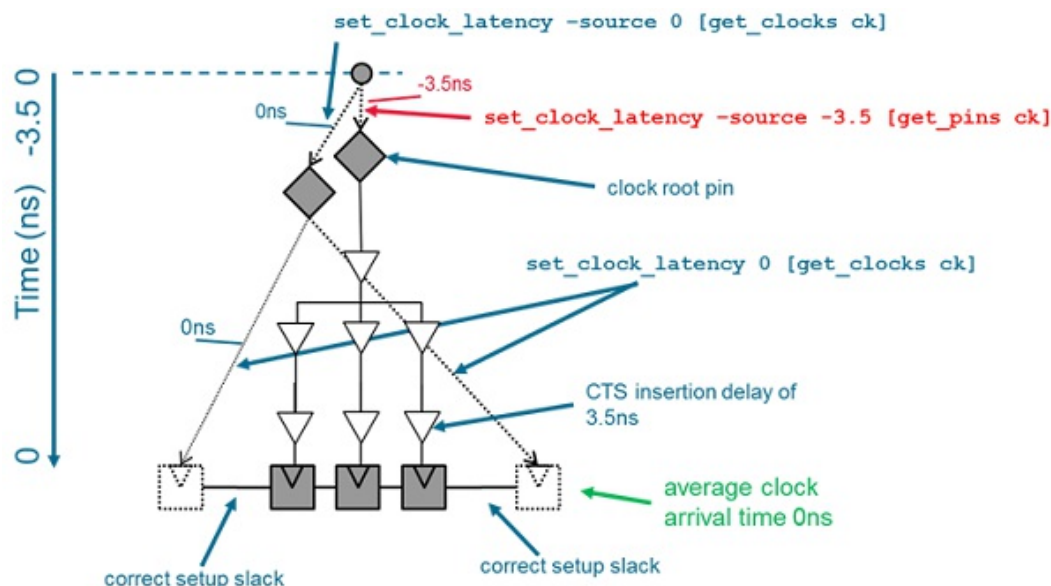
clocks. The source latency modification is performed per clock root pin per clock, such that cross-clock timing consistency is also maintained from before CTS to after CTS. Virtual clocks, if present, do not need modification.

Source Latency Update – CTS with Source Latency Update



The next diagram below is the same example but with the initial "before CTS" clock and network latencies both at zero. This gives rise to a negative source latency set at the clock root pin. Before CTS the average clock arrival time is zero and this is maintained after CTS via the source latency update.

Source Latency Update – Variation with Zero Initial Latencies



In the CCOpt-CTS flow, the source latency update step is performed near the end. In the CCOpt flow, the source latency update step is performed after initial virtual delay balancing before timing optimization and useful skew scheduling commences. The source latency updates are reflected in the Innovus timing constraints and will be saved in any saved database or exported in SDC as normal.

In a block-level design with multiple clocks it is likely that each clock will obtain a different source latency modification. For example, a small clock tree might have a source latency modification of -0.5ns while a large clock tree might have a source latency modification of -2ns. This correctly maintains the validity of the timing constraints which were present before CTS for after CTS usage. However, it means that the 1.5ns difference between the small and large clock tree needs to be synthesized at the top level, but it is usually considerably more efficient to do this at the top level than it is at a block level. Typically, at the top level, an ILM model of blocks is used, in which case CTS will be able to directly see the clock paths inside the ILM so the users need take no action to configure this 1.5ns offset.

The source latency modification scheme can be disabled using "`set_ccopt_property update_io_latency false`". The latency modification scheme should be disabled for top-level chips as balancing clocks outside the chip is unlikely to be practical or if the flow requires balancing between clocks to be performed inside the block level. When latency modification is disabled, the designer needs to correctly estimate the achievable clock tree insertion delay and configure clock network latencies accordingly to avoid a timing jump over CTS and the switch to propagated clock mode timing.

Cell Halos

Cell halos provide a means to enforce additional spacing between clock tree cell instances, specifically between non-sink cell instances and non-sink cell instances, for example between all clock tree buffers, clock gates, and clock tree logic. Halos can be configured by library cell or by clock tree.

For example, use the following commands to set halo distances by library cell:

```
set_ccopt_property cell_halo_x -cell CLKBUF2 10um
set_ccopt_property cell_halo_y -cell CLKBUF2 5um
```

Use the following commands to set halo distance by clock tree:

```
set_ccopt_property cell_halo_x -clock_tree ck1 30um
set_ccopt_property cell_halo_y -clock_tree ck1 30um
set_ccopt_property cell_halo_x -clock_tree ck2 10um
set_ccopt_property cell_halo_y -clock_tree ck2 10um
```

The `-cell` and `-clock_tree` parameters can be combined to specify halos per library cell per clock tree. The 'um' suffix is optional as micrometers are the default units.

The default value for both the `cell_halo_x` and `cell_halo_y` properties is `auto`. When set to `auto`, the `cell_density` and `adjacent_rows_legal` properties will be used instead. From the software's 14.2 release onwards, the default settings for `cell_density` and `adjacent_rows_legal` are changed to `0.75` and `false`, respectively.

Reporting of cell halo compliance (but not density or adjacent rows compliance) is available via the [report_ccopt_cell_halo_violations](#) command. For more information, see the [Halo Violations](#) section.

Power Management

CCOpt-CTS and CCOpt respect power management constraints specified via CPF or UPF. Typically, on most designs it is important to permit CTS access to "always-on" buffers that have additional non-switched power. This is important so that CTS can buffer across power domains where the primary power is switched. This is performed simply by including always-on buffers and inverters with the regular buffers and inverters in

the cell selection settings. For example:

```
set_ccopt_property inverter_cells {INVX4 INVX8 INVX12 PMINVX4
PMINVX8}
set_ccopt_property buffer_cells {BUFX4 BUFX8 BUFX12 PMBUFX4 PMBUFX8}
```

Note: The order of the cells in the lists is not important.

Just after [ccopt_design](#) is invoked, the log file will report which library cells are being used in each power domain. For example:

```
Clock tree balancer configuration for clock_tree ck:
CCOpt power management detected and enabled.
For power_domain SW and effective domain power_domain SW:
Buffers: BUFX8 BUFX4 BUFX1
Inverters: INVX8 INVX4 INVX1
For power_domain SW and effective domain power_domain A0:
Buffers: PMBUFX8 PMBUFX2
Inverters: PMINVX8 PMINVX2
For power_domain A0 and effective domain power_domain SW:
Buffers: PMBUFX8 PMBUFX2
Inverters: PMINVX8 PMINVX2
For power_domain A0 and effective domain power_domain A0:
Buffers: BUFX8 BUFX4 BUFX1
Inverters: INVX8 INVX4 INVX1
Unblocked area available for placement of any clock cells in
power_domain SW: 178511.090um^2
Unblocked area available for placement of any clock cells in
power_domain A0: 5000.000um^2
```

If CTS detects an illegal effective power domain crossing in the clock tree, it will attempt to manage the situation by temporarily overriding the effective domain of the fan-out of a violating clock tree net to match the driver's effective domain. If this happens, then in the log, there will be messages like these:

```
**ERROR: (ENCCCOPT-1044): CTS has found the clock tree is
inconsistent with the power management setup: cell buf1 (a lib_cell
BUFX2) at (10.000,0.000), in power domain PDA has power_domain PDA
and effective domain power_domain PDA but drives modb/flop2/clk which
has power_domain PDB and effective domain power_domain PDB.
**WARN: (ENCCCOPT-1110): Clock tree clk has power supply
illegalities. Attempting to manage these so that CTS can continue.
Type 'man ENCCCOPT-1110' for more detail.
**WARN: (ENCCCOPT-1110): Considering pin modb/flop1/clk to have power
context=power_domain PDA and effective domain power_domain PDA,
actual power context=power_domain PDB and effective domain
power_domain PDB
Type 'man ENCCCOPT-1110' for more detail.
```

To disable this behavior and have CTS abort with an error instead, set the `manage_power_management_illegalities` property to `false`. To disable all power management checks completely, set the `consider_power_management` property to `false`.

Unbufferable Regions

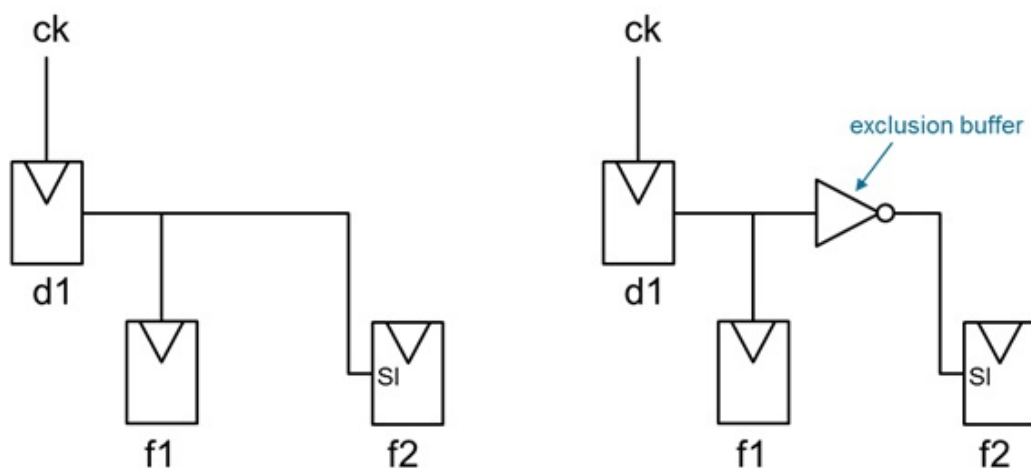
It is possible that the power management constraints prevent a particular net from being buffered. In this situation, the CTS quality is likely to be severely hampered. CTS logs the following message:
****WARN: (ENCCOPT-1042): CTS cannot select a library cell to use as a driver at one or more points of clock_tree ck.**

In order to see the detail of such messages such as the net involved, set the `cts_detailed_cell_warnings` property. This will result in many ENCCOPT-1042 messages being logged with detailed information. Such occurrences frequently result in maximum transition violations and increased insertion delay and should be investigated.

Shared Clock and Data Concerns

In some situations, a net may be used for both clock and datapath purposes. Consider the left-hand side example below with clock divider flop d1 clocking flop f1. However, the output of d1 is additionally connected to the SI input of f2 as part of the scan chain. The net driven by d1 is part of the clock tree graph and may not be operated on by datapath optimization including datapath hold fixing. This restriction prevents datapath transforms from disrupting clock timing. After CTS it is possible that there exists a hold violation at the input of f2, but datapath hold buffer insertion will not be able to insert a buffer to fix it.

Addition of an exclusion buffer



The SI input of flop f2 will by default be a clock tree exclude pin. The command, [ccopt_add_exclusion_drivers](#) can be used to add exclusion drivers to isolate exclude pins from the clock tree graph. The inputs to exclusion drivers are explicitly set to be clock tree ignore pins.

As illustrated in the right-hand side example, once the exclusion buffer is added, datapath hold fixing, or other datapath optimization is free to operate on the net between the exclusion buffer and flop f2.

CCOpt Property System

This section details the way the CCOpt property system operates behind the [set_ccopt_property](#) and [get_ccopt_property](#) commands. Note that some properties are read only and so many more properties are accessible to `get_ccopt_property` than `set_ccopt_property`.

Setting Properties

Properties have a name and are assigned a value. Properties can be global, or per object type. The property name, value, and if desired or required the object type and object pattern can be specified. The following syntax are acceptable:

1. `set_ccopt_property <name> <value> [-obj <obj_pattern>]`
2. `set_ccopt_property <name> [-obj <obj_pattern>] <value>`
3. `set_ccopt_property [-obj <obj_pattern>] <name> <value>`

Where,

name is the name of the property to be set

value is the new value to which you want to set the property

The most commonly used object types are: `-skew_group`, `-clock_tree`, `-pin`, `-inst`, `-lib_pin`, `-delay_corner`, `-net_type`.

All of the commands below are equivalent:

1. `set_ccopt_property target_skew 0.1 -skew_group sg1`
2. `set_ccopt_property target_skew -skew_group sg1 0.1`
3. `set_ccopt_property -skew_group sg1 target_skew 0.1`

The property name and value parameters are positional and must appear in order. The object type and object pattern specification can appear anywhere. If an object type is not specified, then the setting will apply to all relevant objects including ones that are created in the future.

Wildcards

The object name can involve wildcard style patterns, for example:

```
set_ccopt_property use_inverters -clock_tree test* true
```

Wildcards are resolved when the command is issued not when the property value is used. In the command shown above, if a new clock tree `test_new` was defined after the `set_ccopt_property` command was issued, it would not have the `use_inverters` property set.

Optional Object Type Switches

If the `-object_type` parameter is omitted, this implies a "forall" on that type value. The example below specifies that CTS should use inverters for clock tree `ct1`:

```
set_ccopt_property use_inverters -clock_tree ct1 true
```

The example below does not specify the `-clock_tree` object type:

```
set_ccopt_property -use_inverters true
```

And is equivalent to:

```
foreach ct [get_ccopt_clock_trees *] {  
  set_ccopt_property -use_inverters -clock_tree $ct true  
}
```

Delay Corner Special Handling

There are some exceptions to the above rule that are designed to capture common user intent. The main

exceptions are properties for which the `-delay_corner` object type and `-early/-late` parameters apply. When these parameters are omitted, the default behavior is that the property is not set across all delay corners but instead it will be set just for the primary delay corner. In the following example, the leaf-level properties are maintained internally and one or more properties can be set at a time by specifying additional information. The example illustrates the internal representation of the `target_skew` property in a design with one delay corner used for setup timing and two delay corners used for hold timing. The initial default values are shown below:

	Early	Late
SetupDC	Ignore	Auto
Hold1DC	Ignore	Ignore
Hold2DC	Ignore	Ignore

If the command “`set_ccopt_property target_skew 0.05`” is issued, the internal representation will change to become:

	Early	Late
SetupDC	Ignore	0.05
Hold1DC	Ignore	Ignore
Hold2DC	Ignore	Ignore

If the command “`set_ccopt_property target_skew -delay_corner Hold1DC 0.1`” is issued, the representation will change to:

	Early	Late
SetupDC	Ignore	0.05
Hold1DC	0.1	0.1
Hold2DC	Ignore	Ignore

Note that the specification of “Hold1DC” matches two cells in the table. To restrict further, add `-early` or `-late`.

Getting Properties

The [`get_ccopt_property`](#) command is used to retrieve the values of properties. For example, in the last table shown above, the command “`get_ccopt_property -target_skew -delay_corner SetupDC -late`” will return a value of `0.05`.

However, if some or all of the “`-key_name key_value`” switches are omitted, then multiple values may be returned in a list format. An example is shown below.

```
get_ccopt_property -target_skew -delay_corner SetupDC
```

returns the following:

```
{ \
  { -delay_corner SetupDC -early -value default } \
  { -delay_corner SetupDC -late -value 0.05 } \
}
```

If all the selected cells have the same value then a single value instead of a list will be returned. To force the

return of a fully expanded list, even if all values are the same, use the `-list` parameter. For a summary of all parameters of the `get_ccopt_property` and `set_ccopt_property` commands, see *Innovus Text Command Reference*.

Getting a List of Properties and Descriptions

To obtain help on a property, or to find properties matching a wildcard pattern:

```
get_ccopt_property -help propertyName or pattern
```

To obtain a list of all available properties:

```
get_ccopt_property -help *
```

For example:

```
get_ccopt_property -help target_max_trans
```

This specifies the target skew for clock tree balancing. This may be set to a numeric value, or one of 'auto', 'ignore' or 'default'.

If set to 'auto' this indicates that an appropriate skew target should be computed.

If set to 'ignore' this indicates that skew should not be balanced for this corner/path combination.

If unspecified then the value of this property is 'default'.

If the value of the property is 'default' the target skew for late delays in the primary delay corner is interpreted as 'auto' and as 'ignore' otherwise.

Valid values: default | auto | ignore | double

Default: default

Optional applicable arguments: "-skew_group <name>", "-delay_corner <name>", "-early" and "-late".

Migrating from FE-CTS

A list of the CTS engines and how to choose the CTS engine used is given in the The [Clock Tree Synthesis Engines](#) section. It is recommended that users migrate from using FE-CTS to using CCOpt-CTS.

Icon

The FE-CTS flow is a limited-access feature in this release. It is enabled by a variable specified using the [setLimitedAccessFeature](#) command. To use this feature, contact your Cadence representative to explain your usage requirements.

There are several possible variants:

- **Clean start** – The CCOpt-CTS clock tree specification is created using the `ccopt_clock_tree_spec` command and FE-CTS clock specification files and FE-CTS related commands are not required in the flow. This is the recommended approach.
- **Specification translation** – An existing FE-CTS clock tree specification is translated to a CCOpt-CTS clock tree specification. This is the default behavior for the automatic engine setting if a FE-CTS clock

tree specification is loaded, or if this mode is explicitly requested. A list of the CTS engines and how to choose the CTS engine used is given in the The [Clock Tree Synthesis Engines](#) section. This approach is recommended only for existing projects where a heavily manually customized FE-CTS specification is in use such that there is not time to deploy the ‘clean start’ approach. To deploy CCOpt, rather than CCOpt-CTS, this approach is not recommended.

- **Global setting translation** – This is essentially the same as ‘clean start’ except that some global settings can be configured from an FE-CTS specification file. This can be done by using the command, `set_ccopt_mode -import_edict_spec true`, which will configure [set_ccopt_mode](#) and [setCTSMode](#) settings based on the FE-CTS specification contents. This approach is not recommended since `set_ccopt_mode` support is for backward compatibility.
- **Macro model translation** – This is essentially the same as ‘clean start’ except that data for partitions or macros can be read from existing FE-CTS macro model files. The command, `set_ccopt_mode -edict_spec_for_macro_models filename` will read MacroModel and DynamicMacroModel statements from the specified file. MacroModel statements are translated to `skew_group_insertion_delay` property settings, while DynamicMacroModel statements are translated to [create_ccopt_skew_group](#) commands with `-exclusive_sinks` to balance the specified sinks.

Specification Translation

To run CCOpt-CTS using an existing FE-CTS specification the recommended flow is as follows:

```
specifyClockTree -filename fects.spec
create\_ccopt\_clock\_tree\_spec -file ccopt.spec -from_fects_spec
source ccopt.spec
ccopt_design -cts
```

Icon

The `specifyClockTree` command is part of a limited-access feature in this release. It is enabled by a variable specified using the [setLimitedAccessFeature](#) command. To use this feature, contact your Cadence representative to explain your usage requirements.

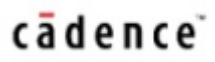
This flow loads the FE-CTS specification and `create_ccopt_clock_tree_spec` is used to translate the FE-CTS specification to a CCOpt-CTS specification. The CCOpt-CTS specification can be inspected and edited prior to loading before invoking CCOpt-CTS. By default, the [clockDesign](#) command will do the conversion and run CCOpt-CTS but without the opportunity to inspect the converted specification.

Concept Mapping

The following table provides an approximate mapping between FE-CTS specification commands and the CCOpt-CTS equivalents. Note that due to the more flexible data model used by CCOpt-CTS, which eliminates the need for sequential CTS steps, an exact 1:1 relationship does not exist.

FE-CTS	CCOpt-CTS
AutoCTSRootPin	create_ccopt_clock_tree
LeafPin/ExcludedPin	set_ccopt_property sink_type stop/ignore/exclude
SinkMaxTran/BufMaxTran	set_ccopt_property target_max_trans -net_type leaf/trunk
maxSkew	set_ccopt_property target_skew

MacroModel	set_ccopt_property insertion_delay -pin
DynamicMarcoModel	<u>create_ccopt_skew_group</u> -exclusive_sinks
clkGroup	create_ccopt_skew_group -balance_skew_groups
CellHalo	set_ccopt_property cell_halo_x/cell_halo_y
RouteTypeName	<u>create_route_type</u>



For support, see [Cadence Online Support](#) service.

Copyright © 2016, [Cadence Design Systems, Inc.](#)
All rights reserved.