Feedback

If you start a new session after applying the fixes,

1. Source the sdc_runtime_<timestamp>.log.tcl file during the next compile.

   Note:

   The `compile_ultra` command identifies and lists the SDC-related runtime issues during the first session. Therefore, during the consecutive sessions when sourcing the log file, you can set the `sdc_runtime_analysis_enable` variable to `false`.

2. Verify whether the runtime improves after sourcing the Tcl file.

Command	Description	Recommendations
`check_timing -sdc_runtime`	Reporting command only. Identifies and lists SDC-related runtime issues with less runtime, as it does not perform any optimization.	Any mapped netlists or netlists from older Design Compiler versions
`sdc_runtime_analysis_enable true` with `compile_ultra` or `compile_ultra -incremental`	Identifies and lists SDC-related runtime issues with longer runtime, as it performs all optimization with the `compile_ultra` flow.	New designs

The Design Compiler NXT and Design Compiler Graphical tools write information messages related to SDC constraints in the sdc_runtime<time>.log file. For example, sdc_runtime_24_10_2019_02_11_20.log. To specify a different log file name or a standard output file, use the `sdc_runtime_analysis_log_file` variable:

```
dc_shell-topo> set_app_var sdc_runtime_analysis_log_file
 \ my_log_file.log
```

Table 14 lists the variables that affect the contents of the sdc_runtime<time>.log file.

Table 14 SDC Variables to Report Runtime Issues

Variable Name	Default
`sdc_runtime_hier_block_pins_timing_path_threshold`	`500`
`sdc_runtime_hier_block_pins_top_timing_paths`	`100`
`sdc_runtime_nets_missing_exceptions_fanout_threshold`	`500`
`sdc_runtime_paths_missing_inter_clock_constraints`	`5`
`sdc_runtime_port_clock_constraint_threshold`	`20`

Feedback

Table 14 SDC Variables to Report Runtime Issues (Continued)

Variable Name	Default
sdc_runtime_tightly_constrained_path_group_slack_percentage	75
sdc_runtime_tightly_constrained_same_clock_path_groups	50
sdc_runtime_top_fanout_nets_missing_exceptions	100
sdc_runtime_unused_clocks_threshold	5

Table 15 describes information messages related to SDC constraints.

Table 15 TIM Messages Controlled by the SDC Variables

Information Message	Description
TIM-601	If you have set the sdc_runtime_unused_clocks_threshold variable to true, the tool issues this message when there are more than five unused clocks in the design.
TIM-602	The tool reports tightly constrained paths with a critical slack of more than 75 percent of the clock period. Use the sdc_runtime_tightly_constrained_path_group_slack_percentage variable to set the threshold for reporting the paths. The variable does not impact optimization. For example, • By default, if the clock period is 2.0 ns, the paths with a critical slack more than 1.5 ns are reported. • If you set the variable to 50, the paths with a critical slack more than 1 ns are reported. Note that the QoR remains the same in both the instances.
TIM-603	The tool reports tightly constrained paths when more than 50 path groups are created from a single clock. Use the sdc_runtime_tightly_constrained_same_clock_path_groups variable to set the threshold to identify whether there are too many path groups using the same clock.
TIM-604	The tool issues this message when there are path groups with infeasible paths at different stages in the flow.
TIM-605	Multiple clocks reaching a pin might cause long runtime. This can be due to missing false paths, multicycle paths, or timing exception constraints on a path. In this case, the tool issues TIM-605 messages to identify paths with missing constraints between clock pins.
TIM-606	The tool reports the top 100 tightly constrained paths with a fanout more than 500 when high-fanout nets are not set with the set_ideal_network or set_dont_touch command. Use the sdc_runtime_nets_missing_exceptions_fanout_threshold and sdc_runtime_top_fanout_nets_missing_exceptions variables to set the fanout threshold and the number of nets to display. The defaults are 500 and 100, respectively.

Feedback

Characterizing Subdesigns

When you compile subdesigns separately, boundary conditions such as the input drive strengths, input signal delays (arrival times), and output loads can be derived from the parent design and set on each subdesign. You can do this in the following ways:

- Manually

  Use the `set_drive`, `set_driving_cell`, `set_input_delay`, `set_output_delay`, and `set_load` commands.

- Automatically

  Use the `characterize` command.

Using the characterize Command

The `characterize` command places on a design the information and attributes that characterize its environment in the context of a specified instantiation in the top-level design.

The primary purpose of `characterize` is to capture the timing environment of the subdesign. This occurs when you use `characterize` with no arguments or when you use its `-constraints`, `-connections`, or `-power` options.

The `characterize` command derives and asserts the following information and attributes on the design to which the instance is linked:

- Unless the `-no_timing` option is specified, the `characterize` command places on the subdesigns any timing characteristics previously set by the following commands:

`create_clock`	`set_load`
`group_path`	`set_max_delay`
`read_timing`	`set_max_time_borrow`
`set_annotated_check`	`set_min_delay`
`set_annotated_delay`	`set_multicycle_path`
`set_auto_disable_drc_nets`	`set_operating_conditions`
`set_drive`	`set_output_delay`
`set_driving_cell`	`set_resistance`
`set_false_path`	`set_timing_ranges`
`set_ideal_net`	`set_wire_load_model`

made on the subdesign. In this case, you must explicitly remove annotations from the subdesign (using `reset_design`) before you run the `characterize` command again.

Optimizing Bottom Up Versus Optimizing Top Down

During optimization, you can use `characterize` with `set_dont_touch` to maintain hierarchy. This is known as bottom-up optimization, which you can apply by using a golden instance or a uniquify approach (either manually with the `uniquify` command or automatically as part of the `compile` command). An alternative to bottom-up optimization is top-down optimization, also called hierarchical compile. During top-down optimization, the tool automatically performs characterization and optimization for subdesigns.

Deriving the Boundary Conditions

The `characterize` command automatically derives the boundary conditions of a subdesign based on its context in a parent design. It examines an instance's surroundings to obtain actual drive, load, and timing parameters and computes the following types of boundary conditions:

- Timing conditions

  Expected signal delays at input ports.

- Constraints

  Inherited requirements from the parent design, such as maximum delay.

- Connection relations

  Logical relationships between ports or between ports and power or ground, such as always logic 0, logical opposite of another port, or unconnected.

The `characterize` command summarizes the boundary conditions for one instance of a subdesign in one invocation. The result is applied to the reference.

Figure 36 shows instances and references.