Silicon Highway Technologies

Si-Time - Static Timing Analysis Engine

User Manual

Version 0.3

07/02/2025

Table of Contents

List of Figures

List of Tables

1. Introduction

This User Manual (UM) provides the information necessary for experienced digital integrated circuit designers to effectively use the Silicon Highway Technologies (SiHi) **Si-Time** EDA tool. To understand and use **Si-Time,** comprehensive knowledge on physical design flow and principles, as well as a strong understanding of UNIX and TCL programming are required.

This UM provides instructions on installing and running **Si-Time**, as well as detailed description of the key features and capabilities of the tool. **Si-Time** is fully compatible with all the tools provided by SiHi in the same user interface, making the user experience easier and reducing the tool familiarization time.

This UM is under development, so please feel free to provide feedback or/and request an updated version. Sections with [TBD] notation are supported by Si-Time. These will be completed in the next UM version.

1.1 Overview

SiHi **Si-Time** ensures that the timing requirements of the design are met. It analyzes the timing behavior of a digital circuit without considering its dynamic behavior or how it may change over time due to varying input conditions. The tool evaluates signal propagation delays, setup and hold times, clock skew, and other timing constraints to verify that the design operates reliably within specified timing margins. **Si-Time** helps designers identify and rectify timing violations early in the design process, reducing the risk of costly errors and ensuring that the final product meets performance requirements.

1.2 Si-Time Key Features and Capabilities

Si-Time offers a wide range of features and functionalities to ensure that your designs meet stringent timing requirements and performance goals. Such features are listed here:

- **Timing Verification**:
  - Evaluate setup and hold times to ensure proper data capture.
  - Analyze clock-to-clock, data-to-clock, and data-to-data timing paths.
- **Clock Domain Crossing (CDC) Analysis**:
  - Identify and analyze signals crossing between different clock domains.
  - Detect and report metastability issues and potential data loss.
- **Constraint Management**:
  - Define and manage timing constraints such as clock period, input/output delays, and maximum path delays.
  - Support for industry-standard constraint formats like SDC (Synopsys Design Constraints) and SPEF (Standard Parasitic Exchange Format).
- **Path Analysis**:
  - Trace critical timing paths from input ports to output ports.
  - Highlight timing violations and identify the slack on each path.
  - Define and manage false paths.

- **Power Analysis**:
  - Estimate power consumption based on activity factors and switching activity.
- **Delay Calculation**:
  - Calculate signal propagation delays through combinational logic and interconnect.
  - Account for process variations and environmental conditions.
- **Reporting and Visualization**:
  - Generate detailed timing reports highlighting violations and slack analysis.
  - Provide interactive timing diagrams and waveform viewers for visualization.
- **Cross-Probing with Physical Design Tools**:
  - Integrate with physical design tools to cross-probe between timing analysis results and layout views.
  - Streamline the debug process by correlating timing violations with physical implementation.
- **Scripting and Automation**:
  - Provide scripting interfaces (*e.g.*, TCL, Python) for automation and customization.
  - Allow batch mode operation for running analysis on multiple designs.
- **Interactive Results Analysis and Design Exploration**:
  - Enable interactive results analysis and design exploration features to explore timing paths and identify root causes of violations.
  - Offer capabilities for timing fixes and optimizations, as timing derates.
  - Allow exploring potential solutions by using case analysis.
- **Integration with Design Flows**:
  - Seamlessly integrate with electronic design automation (EDA) tools and design flows.
  - Provide APIs for interoperability with third-party tools and custom workflows.

1.3 Latest Version Limitations

Some limitations of the current version are listed below. **Si-Time** is continually developing and evolving, while SiHi engineers work to eliminate these limitations.

- **Clock Tree Analysis**:
  - Analyze clock distribution networks for skew and jitter.
  - Optimize clock tree for improved timing performance.
- **Multi-Voltage and Multi-Mode (MVMM) Analysis**:
  - Handle designs with multiple voltage domains and operational modes.
  - Perform timing analysis considering voltage scaling and mode transitions.
- **Advanced Analysis Techniques**:
  - Support for advanced analysis techniques like multi-corner, multi-mode (MCMM) analysis.
  - Statistical timing analysis to account for process variations.
- **Crosstalk Noise Analysis**

1.4 Conventions of this User Manual

This section describes the conventions used in SiHi User Manuals.

Table 1: Conventions List

Convention	Description
`Courier`	Indicates exact keyword value
`Courier Italic`	Indicates user defined value
<>	Indicates mandatory argument
[ ]	Indicates optional arguments
|	Indicates a choice between arguments
…	Indicates repentance of the arguments

1.5 Support

We are committed to providing excellent customer support to ensure that user experience with **Si-Time** is smooth and productive. If you encounter any issues or have questions while using the tool, our dedicated support team is here to assist you.

1.5.1 Contacting Support

If you require assistance or have any inquiries, please feel free to contact our support team via the following methods:

- **Email:** info@sihi-tech.com
- **Direct assistance for an assigned engineer.** If no engineer is assigned, please contact us at info@sihi-tech.com.

1.5.2 Providing Feedback

We value your feedback and suggestions as they help us improve our products and services. If you have any comments or ideas for enhancing SiHi **Si-Time**, please don't hesitate to share them with us. You can send your feedback to info@sihi-tech.com.

1.6 Licensing Schemes

SiHi products are available in four schemes, while the scheme is selected based on the needs and the status of each customer. The available licensing schemes are described in Table 2.

Table 2: Licensing Schemes

Scheme	Description
Freemium	Provided to customers mainly for evaluation purposes. The version of the tool has limitations, in terms of the available features, the usage duration, and/or the range of the supported designs.

Custom	Provided to customers that have specific needs. The available features are configured to meet customer needs. Features implementation is available, after agreement between the customer and SiHi.
Advanced	Provided to customers that need access to the entire capabilities of SiHi tool. New features, full support and training are including, while customer specific features are provided, after agreement between the customer and SiHi.

Note: Based on the decided scheme, the support from SiHi on bugs fixing, applied engaging, and extra features implementation, differ. The level of support can be agreed with each independent customer.

2. Getting Started

2.1 System Requirements

Suggested Operating Systems: **CentOS 7, CentOS 8, or Docker Image**

Minimum Available Disk Space: **500 MB or for Docker Image 4GB**

2.2 Validated PDKs For Si-Time

Table 1 summarizes the PDKs and technology nodes that **Si-Time** had been tested and used.

Table 3: Validated PDKs List

PDK Provider	Technology Node							
	7nm	12nm	15nm	22nm	40nm	45nm	130nm	250nm
ASU	✓							
NANGATE			✓			✓		
Global Foundries		✓		✓				
TSMC					✓			
IHP							✓	✓
SkyWater							✓	

2.3 Setting Up Si-Time

SiHi provides all the necessary files to simplify the installation experience. All the required files are shipped with a tarball file. The only action that user must perform is to extract the contents of the tarball to a path of his choice and run the setup script.

After the extraction, the directory `INSTALL` will be created. Then you have to set the `SiTime_INSTALL_DIR` to target to the `INSTALL` directory, i.e.

```
export SiTime_INSTALL_DIR=<path of INSTALL dir>
```

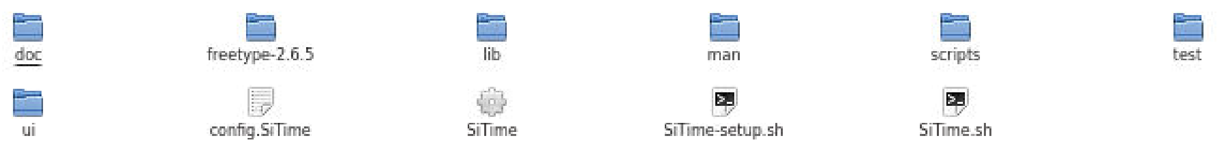
or you can include the above command to your bashrc.

After setting the above variable run the shell script named `SiTime-setup.sh`, that is inside `INSTALL` dir.

Note 1: Linux command for extracting contents of a tarball with name *`file`*: `tar xvzf <file>.tar.gz`

After setting up the **Si-Time**, the `INSTALL` folder must contain the folders and files presented in the Figure 1, while the **Si-Time** executable is the `SiTime`.

Figure 1: Installation Folder Structure



2.4 Additional Required Packages

After setting up the **Si-Time**, see Section 2.3, additional required packages must be installed. To list the required packages to run **Si-Time** you can run the following Linux command:

```
ldd SiTime
```

2.4.1 Sandia Labs Xyce Electronic Circuit Simulator

SiHi provides the ability to automatically run Xyce electronic circuit simulation, in case user needs to compare **Si-Time** timing results against SPICE level simulations performed using Sandia Labs Xyce. The Xyce executable must be included in the Linux PATH shell variable, *e.g.*

```
export PATH=/usr/local/Xyce-Release-6.12.0-OPENSOURCE/bin:$PATH
```

Details about running Xyce are provided in Section [TBD].

3. Using Si-Time

The following sub-sections provide detailed, step-by-step instructions on how to run and use the various utilities of **Si-Time**.

3.1 Launching and Exiting the Tool

Once the setup script runs, user can launch the tool by running the `SiTime` executable.

Si-Time is launched via a Linux shell prompt by typing: `./SiTime.sh`. `-no_gui`, for the command-line interface, while for the graphical user interface enter `./SiTime.sh.` **Si-Time** also supports Tool Command Language (TCL).

The available options for launching **Si-Time** are listed here:

Table 4: Launching Options

Option	Description
`-no_gui`	Run Si-Time without GUI
`-help`	Print this help message
`-f <`*`TCL script`*`>`	Source specified TCL script upon startup
`-exit_on_error`	Close Si-Time if any error occurs
`-threads <`*`number of threads`*`>`	Specify number of threads (default: 1, no less than 1 are allowed)
`-logs_prefix <`*`prefix for log files`*`>`	Specify prefix for log files (default: SiTime_history.cmd)
`-overwrite_logs`	Overwrite most recently created log files (default false)
`-no_cmd`	Don't create command log files (default false)
`-no_logs`	Don't create log files (default false)

To exit from **Si-Time,** enter `quit, exit,` or close program window.

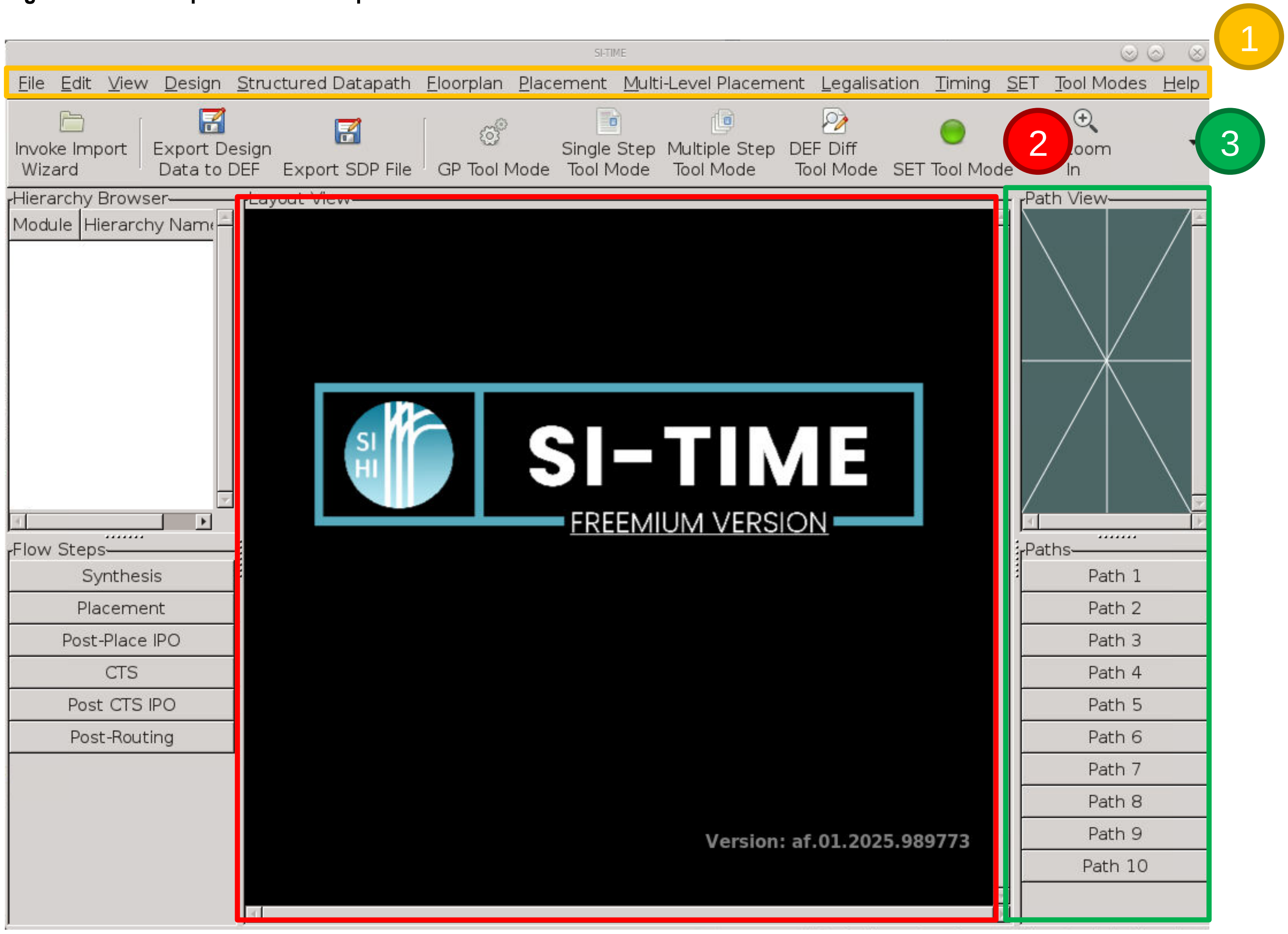
3.1.1 Graphical User Interface Overview

Error! Reference source not found. presents the main overview of the Graphical User Interphase (GUI) of **Si-Time**.

To enable and disable GUI use `show_gui` and `hide_gui` TCL commands, respectively.

1) Main tool bar
2) Physical Layout Viewer
3) Timing Analysis Paths Viewer

Figure 2: Main Graphical User Interphase



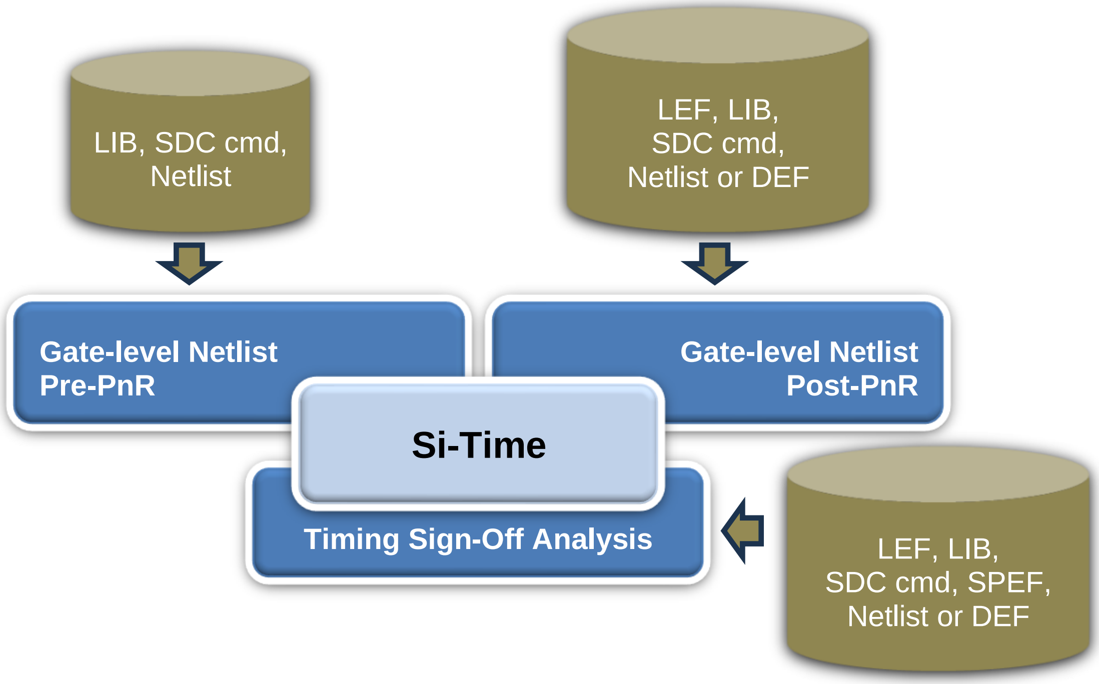
3.2 Analysis Modes

Before performing timing analysis, the designer must be aware of the available analysis modes of **Si-Time**. Figure 3 presents the three available analysis modes, *i.e.***:**

- **Gate-level Netlist Pre-PnR**
- **Gate-level Netlist Post-PnR**
- **Timing Sign-Off Analysis**

Each mode requires loading different files, while the quality and runtime of each mode differ. See Section 3.3 for more details about the necessary and optional input files of each mode.

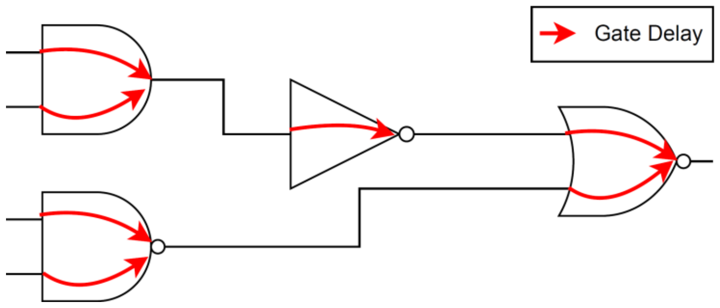
Figure 3: STA Modes and Required Files


LIB, SDC cmd, Netlist
LEF, LIB, SDC cmd, Netlist or DEF
Gate-level Netlist Pre-PnR
Gate-level Netlist Post-PnR
Si-Time
Timing Sign-Off Analysis
LEF, LIB, SDC cmd, SPEF, Netlist or DEF

3.2.1 Gate-level Netlist Pre-PnR

Fast timing analysis for the input design based on its connectivity, standard cell or macros timing delay, and input timing constraints. Gate-level Netlist Pre-PnR does not consider wires delay and physical information of chip elements, such as standard cells, macros, and I/O ports locations.

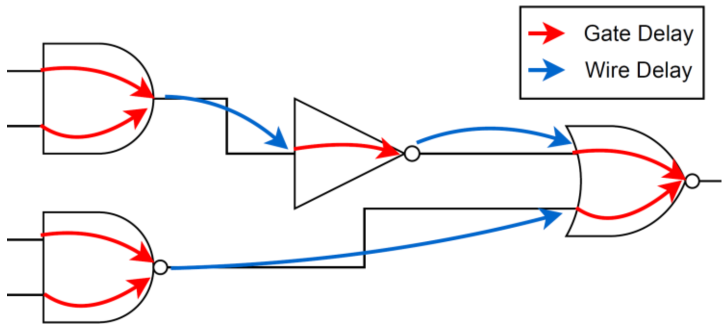
Figure 4: Gate-level Netlist Pre-PnR Considers only the Gates Delay



3.2.2 Gate-level Netlist Post-PnR

In Gate-level Netlist Post-PnR gate delays along with wires delay impact on the compotation of circuit performance. Gates physical locations impact on timing analysis results, while wire delay models contribute to the analysis.

Figure 5: Gate-level Netlist Post-PnR Considers both Gates and Wire Delays



The supported wire delay timing RC modes of **Si-Time** are:

- Elmore (Lumped RC)
- Pi-model
- DetailedRC, for Timing Sign-off (parasitics SPEF in required)

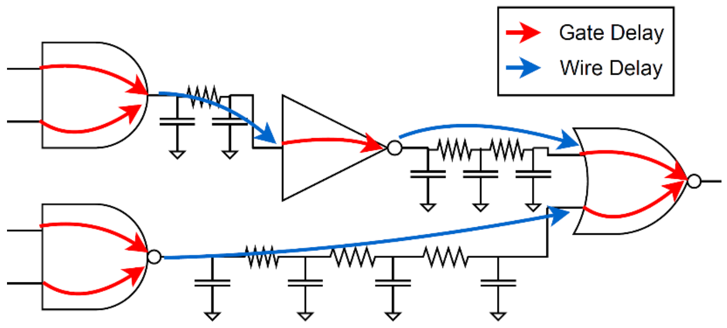
The TCL command `set_timing_RC_mode` sets the wire delay model (or RC model). The default model is the Elmore. The syntax and the available parameters of the `set_timing_RC_mode` command are:

```
set_timing_RC_mode <lumpedRC | pimodelRC | detailedRC>
```

Note: `detailedRC` mode is only supported for Timing Sign-Off flow, *i.e.* when SPEF is used, post parasitics extraction. It is NOT compatible with Gate-level Post-PNR Netlists.

3.2.3 Timing Sign-Off Analysis

The most accurate STA Sign-Off analysis is performed, when a SPEF file is provided, containing all extracted wire RC parasitics. This is known as Detailed RC mode, as explained above.



TCL command `load_spef` is used to load the SPEF file required for the Timing Sign-Off analysis mode.

```
load_spef -longest | -shortest ?-errors? {<filenames>}
```

3.3 Loading Design Files

SiHi tools support most industrial EDA file formats within the Netlist to GDSII design flow. Table 5 summarizes the supported file formats for each analysis mode.

Table 5: Required Files for the various STA Analysis Modes

File Format	Gate-level Netlist Pre-PnR	Gate-level Netlist Post-PnR	Signoff
`Netlist or DEF`	Necessary	Necessary	Necessary
`LIB`	Necessary	Necessary	Necessary
`LEF`	Optional	Necessary	Necessary
`SPEF`	Optional	Optional	Necessary

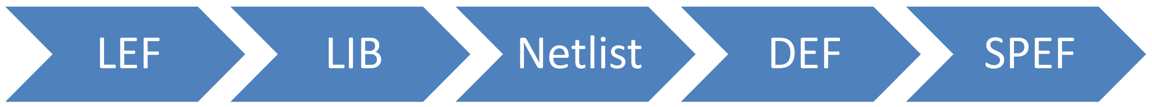
Note: Verilog Gate-Level Netlists must currently be uniquified. In addition, tri-states and other non-STA compatible standard-cell elements are NOT supported.

Note: A DEF file may be loaded instead of a Verilog Gate-Level Netlist. The design will be DEF based and DEF physical locations will be used. Alternatively, a DEF file may be loaded post Verilog in. In that case, the DEF must match the Verilog Design. DEF physical locations will be used for all relevant design elements (standard-cells, hard macros, Top-Level I/Os *etc.*).

The following TCL commands must be used to load a design's files.

```
load_verilog <Uniquified netlist file>
load_def <DEF file>
load_lef <LEF file>
load_lib <LIB file>
load_spef <SPEF file>
```

The recommended order for loading design files is (based on desired analysis mode, skip optional files):



Example Design Loading Script:

3.3.1 Loading Multiple LEF Files

To load multiple LEF files use a list of LEF files in the `load_lef` command, *e.g.*

```
load_lef {lef1 lef2 ... lefn}
```

for example: `load_lef {mock_tech.lef mock_stdcell.lef}`

3.3.2 Loading Multiple LIB Files

To load multiple LIB files, similarly to LEF files, a list of LIB files must be used in the `load_lib` command. However, as the LIB command groups LIB files per corner, double braces are required, *e.g.*

```
load_lib {{<filenames> per corner 0}{<filenames> per corner 1}{...}}
```

where the default supported corner is corner 0.

For example: `load_lib {{mock1.lib mock2.lib mock3.lib mock14.lib}}}},` loads 4 liberty files for corner 0. Additionally, standard TCL utilities can be used to load multiple files.

Note: The current version supports one corner.

For example:

```
load_lib [list [glob /home/user/PDKs/mockPDK/*.lib]]
```

3.3.3 Example Script for Loading a Design

The following script sets some TCL variables and then loads LEF, LIB files and a Verilog netlist.

```
######################################################################
set LIB /home/user/PDKs/lib/mock.lib
set DESIGN /home/user/design/mock.v

load_lef {/home/user/PDKs/lef/mock.lef}
load_lib $LIB -errors

load_verilog $DESIGN
######################################################################
```

3.4 Storing and Loading Design Data

[TBD]

3.5 Reporting Design Data

Various reporting commands are available to obtain design information through TCL commands. These include:

3.5.1 Report Design Information

- `report_design`

Reports Top-Level I/O state, Top Level Module, Special Nets, Top Module Total Area, Utilization and various other design status metrics.

3.5.2 Report Design Area

- `report_area`

Reports the Top-Level Module standard cell area, in um^2, and the Total Number of design Instances.

3.5.3 Report Top-Level I/O

- `list_toplevel_pins`

Reports the Top-Level I/O pin information.

3.5.4 Check Design Correctness

- `check_design`

This command is useful for identifying unconnected design portions.

It checks for and reports: (1) all unconnected component input or output pins, as well as (2) all unconnected Verilog module input or output pins.

3.5.5 Report Design Modules

- `list_modules`

Reports all modules of the design.

- `list_module <modulename>`

Lists specified module information.

For example:

```
> list_module fa_33
--------------------------------------
Module: fa_33, Hierarchical Name: lpffir_core/rca_inst2/fa_inst15
Module Pins (5):
a (#0, INPUT)[Corresponding Gatepin: fa_33/a], b (#1,
INPUT)[Corresponding Gatepin: fa_33/b], ci (#2, INPUT)[Corresponding
Gatepin: fa_33/ci], co (#3, OUTPUT)[Corresponding Gatepin: fa_33/co],
s (#4, OUTPUT)[Corresponding Gatepin: fa_33/s].
--------------------------------------
Module Ports (5):
b (Order Index: 1, Port Index: 1), co (Order Index: 3, Port Index: 3),
s (Order Index: 4, Port Index: 4), ci (Order Index: 2, Port Index: 2),
a (Order Index: 0, Port Index: 0).
--------------------------------------
Module Components (4):
fa_33/U1, fa_33/U2, fa_33/U3, fa_33/U4.
--------------------------------------
Module Gate Pins (20):
fa_33/a, fa_33/U3/B, fa_33/U4/A, fa_33/b, fa_33/U3/A, fa_33/U4/B,
fa_33/ci, fa_33/U1/A, fa_33/U2/A, fa_33/co, fa_33/U2/Q, fa_33/s,
fa_33/U1/Q, fa_33/1'b0, fa_33/U2/C, fa_33/U3/Q, fa_33/U1/B,
fa_33/U2/B, fa_33/U4/Q, fa_33/1'b1.
--------------------------------------
Module Component Size: 4
Module Connections Size: 20
--------------------------------------
```

3.5.6 Report Standard Cells and Macros

- `list_components`

Reports all components (std. cells and Macros) of the design.

- `list_component <componentname> ?-CCs? ?-pinsCCs? ?-cluster?`

Lists specified component (std. cells and Macros) information.

`CCs`: Reports the connected components

`pinsCCs`: Reports the gate pins connections

`cluster`: Reports the structural cluster that specified component exists.
Note: Structural clusters are not supported in this version of the tool.

For example:

```
> list_component counter_DW01_inc_0/U1_1_1
--------------------------------------
Component: counter_DW01_inc_0/U1_1_1 (16, 0)
Library Cell Type: adhalf_1
Library Cell Pins: a (Corresponding Gatepin
counter_DW01_inc_0/U1_1_1/a), b (Corresponding Gatepin
counter_DW01_inc_0/U1_1_1/b), co (Corresponding Gatepin
counter_DW01_inc_0/U1_1_1/co), s (Corresponding Gatepin
counter_DW01_inc_0/U1_1_1/s)
Component Orientation: N
Width = 1.260um, Height = 0.576um, Area = 0.726um^2, Layout Location:
(0.000, 0.000)
--------------------------------------
```

3.5.7 Report Gatepins

- `list_gatepins`

Reports all component (std. cells and Macros) pins of the design.

- `list_gatepin <gatepinname>"`

Lists specified component (std. cells and Macros) pins information.

For example:

```
> list_gatepin lpffir_core/x54/C
--------------------------------------
Gate Pin: lpffir_core/x54/C (1912, 0)
Connections: (3)
```

```
(Module lpffir_core)
Corresponds to Component/Pin lpffir_core/x54/C.
lpffir_core/clk_i (no delay info)
lpffir_core/x115C (no delay info)
lpffir_core/x114/C (no delay info)
```

3.5.8 Instance Names Note: Flat, Declared Names vs. Full Hierarchy Names

The Si-Time tool recognizes design instance identifiers using two different naming schemes, (a) declared module, cell, pin names and (b) hierarchical module, cell, pin names. The former is of the form `<declared modulename>/<portname>` or `<declared modulename>/<componentname>/<gatepin>`. The latter corresponds to the full hierarchical name, *i.e.* `<module/instance1/instance2/.../component/portname>` or `<module/instance1/instance2/.../component/gatepin>`.

Most TCL commands use flat, declared names, instead of full Verilog hierarchy names, *e.g.* most TCL list commands, *e.g.* `list_component(s)`, `list_gatepin(s)`, use flat declared names. In STA reports however, for convenience, the complete hierarchy names are used. To convert between the two, the following TCL commands are provided.

- `list_component_by_hierarchyname`

Converts the provided hierarchical component name into the flat declared component name.

For example:

```
> list_component_by_hierarchyname rca_inst2/fa_inst15/U2
---------------------------------------

Component: fa_33/U2 (168, 1)
Library Cell Type: AO21JILTX1
Library Cell Pins: A, B, C, Q, VDD!, VSS!
Component Orientation: N
Width = 3.000um, Height = 3.780um, Area = 11.340um^2, Layout Location:
(0.000, 0.000)
```

- `list_gatepin_by_hierarchyname`

Converts the provided hierarchical gatepin name into the flat declared gatepin name.

For example:

```
> list_gatepin_by_hierarchyname counter_DW01_inc_0/U1_1_1/a
---------------------------------------
Gate Pin: counter_DW01_inc_0/U1_1_1/a (16, 0)
Connections: (1)
(Module counter_DW01_inc_0)
Corresponds to Component/Pin counter_DW01_inc_0/U1_1_1/a.
counter_DW01_inc_0/A|1 (no delay info)
---------------------------------------
```

- `list_hierarchyname`

Convert provided hierarchical module name into the flat declared module name.

For example:

```
> list_hierarchyname lpffir_core/rca_inst2/fa_inst15
```

```
INFO: Hierarchical Name lpffir_core/rca_inst2/fa_inst15 corresponds to
Module fa_33
```

It should be noted that the use of flat name identifiers in TCL commands does not cause issues when saving Verilog or exporting DEF from the tool. When DEF or Verilog files are exported, full hierarchy names are used for compatibility to other EDA tools.

3.5.9 Escaped Identifiers and Escape characters in TCL

A component name may be escaped, *e.g.* `counter/\count_reg[5]`. In this case, escape characters are also needed at the TCL level, to specify such a component, or one of its gatepins.

All TCL commands which use identifiers including the special "[", "]", or "\" characters must escape such special characters using an extra backslash ("\"). The latter is absorbed by the TCL shell interpreter.

For example, to report timing information about one of the escaped components pins, `counter/\count_reg[5]/d`, the name must be escaped as `counter/\\count_reg\[5\]/d`, *i.e.* with three extra escape characters.

```
> list_gatepin counter/\\count_reg\[5\]/d
--------------------------------------
Gate Pin: counter/\count_reg[5]/d (98, 1)
Connections: (2)
(Module counter)
Corresponds to Component/Pin counter/\count_reg[5]/d.
counter/add_13/SUM|5 (no delay info)
counter_DW01_inc_0/SUM|5 (no delay info)
--------------------------------------
```

Otherwise, if escape identifiers are not used, the tool will report an error.

For example,

```
> list_component counter/\count_reg[5]
invalid command name "5"
ERROR: TCL Error for Command "list_component counter/\count_reg[5]".
```

3.5.10 Escape characters in TCL lists, double escapes

For functions which use TCL lists as inputs, when a name contains escaped identifiers, double escapes must be used. Below, we illustrate an example with an escaped component name.

The original instance name is `\reg_layer_2_w27_reg[0]`, within a module called `multiplier_16x16bit_pipelined`, as shown in the netlist below.

```
module multiplier_16x16bit_pipelined (
...
DFRRQJILTX1 \reg_layer_2_w27_reg[0] (.C(i_clk), .D(n218),
.Q(reg_layer_2_w27[0]), .RN(n333));
...
endmodule
```

The module is instantiated with the same name. To convert the hierarchical name to the flat, declared module name, as used in the tool database we use the `list_component_by_hierarchyname` TCL command as follows.

```
> list_component_by_hierarchyname
multiplier_16x16bit_pipelined/\\reg_layer_2_w27_reg\[0\]
--------------------------------------
Component: multiplier_16x16bit_pipelined/\reg_layer_2_w27_reg[0]
(1904, 0)
Library Cell Type: DFRRQJILTX1
Library Cell Pins: C, D, Q, RN, VDD, VSS
Component Orientation: N
Width = 8.500um, Height = 3.780um, Area = 32.130um^2, Layout Location:
(0.000, 0.000)
--------------------------------------
```

Note that we had to add two escape characters, in front of the component name, for TCL to accept it. The internal “flat” name, identical in this case is `multiplier_16x16bit_pipelined/\\reg_layer_2_w27_reg\[0\]`, as shown in the above report.

However, were we to use the name in a TCL command accepting lists, *e.g.* for TCL command `report_gatepin_annotated_parasitics`, we would need to use double escapes to precede the escaped name, so a total of four escape characters, as follows.

```
report_gatepin_annotated_parasitics
{multiplier_16x16bit_pipelined/\\\\reg_layer_2_w27_reg\[0\]/C} -
longest -corner 0
```

3.5.11 Verilog Modules Bussed Ports Naming

Verilog bus names of the form `portname[i]` are stored as `portname|i` in Si-Time memory, for any Verilog module within the design hierarchy. This feature does not create consistency issues when the netlist or a DEF file is written out, as names are reverted to their Verilog `portname[i]` form.

However, for Top-Level I/Os for instance, of the form `aes/key_i[126]` or `aes/data_o[76]`, the corresponding internal gatepins will be `aes/key_i|126` and `aes/data_o|76` respectively. Using brackets for non-escaped identifiers, such as these two examples will not be able to identify the gatepin using TCL commands. An example of using | for a Top-Level Input is shown below.

```
> list_gatepin aes/key_i|125
--------------------------------------
Gate Pin: aes/key_i|125 (12905, 1)
```

```
Connections: (2)
(Module aes)
(Module aes, Gatepin Validated as a Moduleport)
Corresponds to Module Port/Pin aes/key_i|125.
aes/U1422/I (no delay info)
aes/U3754/A1 (no delay info)
--------------------------------------
```

To list all Top-Level I/Os, TCL command `list_toplevel_pins` may be used.

3.6 Analyzing Design Connectivity using TCL Commands

Si-Time provides connectivity information TCL commands, which allow for interactive exploration of pin (gate pin) to pin connections or gatepin to gatepin connections.

Gate pins must be specified as flat hierarchy names in the form `<declared modulename>/<component name>/<pin name>`. Components must be specified as flat hierarchy names in the form `<declared modulename>/<component name>`.

3.6.1 Gate Pin Connectivity

Gate pin connectivity may be explored using the following TCL command:

```
list_gatepin_connections ?-allios? ?-predecessors | -successors?
<gatepinname>
```

Lists specified gatepin connections, either fan-in (predecessors) or
fan-out (successors).

example:

```
> list_gatepin_connections -allios M65C02_ALU/U878/A2

--------------------------------------

Gatepin M65C02_ALU/U878/A2 Successors:

(All Top-Level I/Os Enabled.)

M65C02_ALU/U752 (/A1), M65C02_ALU/U896 (/A2), M65C02_ALU/U1012 (/A2),
M65C02_ALU/U1124 (/A1).

--------------------------------------

> list_gatepin_connections -allios -predecessors M65C02_ALU/U878/A2

--------------------------------------

Gatepin M65C02_ALU/U878/A2 Predecessors:

(All Top-Level I/Os Enabled.)

M65C02_ALU/\BCD/rS_reg[1] (/Q).

-------------------------------------
```

3.6.2 Component Connectivity

Component connectivity may be explored using the following TCL command:

```
list_component_connections ?-predecessors | -successors?
<componentname>
```

Lists specified component connections, either fan-in (predecessors) or fan-out (successors).

example:

```
> list_component_connections M65C02_ALU/U878

--------------------------------------

Component M65C02_ALU/U878 Successors:

M65C02_ALU/U897.

--------------------------------------

> list_component_connections -predecessors M65C02_ALU/U878

--------------------------------------

Component M65C02_ALU/U878 Predecessors:

M65C02_ALU/U877, M65C02_ALU/\BCD/rS_reg[1].

--------------------------------------
```

TCL command `list_component_connections_pins_all` does the exact same thing, but also dumps the pin names. Its format is as follows:

```
list_component_connections_pins_all ?-allios? ?-predecessors | -
successors? <componentname>
```

In addition, a more direct type of component connectivity, *i.e.* CC (Connected Components) connectivity may be explored using the respective TCL command below.

```
list_components_CCs ?-componentCCsgatein | -componentCCsgateout?
```

The latter will display CCs for all design components, for only between components. Components to Top-Level I/O connections will not be included.

3.6.3 Top-Level I/Os Connectivity

Top-Level I/O Pin connectivity, also referred to as Top Module Ports connectivity may be explored using the following TCL commands.

`list_moduleports_CCs` and `list_moduleports_pins_CCs`

Example:

```
> list_moduleport_CCs CSel

--------------------------------------

Module Port: CSel (#10), Type: INPUT

CCs: M65C02_ALU/U535 (Component) [INPUT], M65C02_ALU/U707 (Component)
[INPUT]

--------------------------------------
```

3.6.4 Logic Cones Connectivity

All of the following TCL commands are related to Logic Cones connectivity.

```
list_forward_logic_cone <gatepinname> -longest | -shortest ?-
constants?
```

```
list_backward_logic_cone <gatepinname> -longest | -shortest ?-
constants?
```

```
get_forward_logic_cone <gatepinname> -longest | -shortest ?-
sortbylevel? ?-ffname <namesubstring> (when .lib is NOT loaded)? ?-
components?
```

```
get_backward_logic_cone <gatepinname> -longest | -shortest ?-
sortbylevel? ?-ffname <namesubstring> (when .lib is NOT loaded)? ?-
components?
```

All of them list component gatepins forward from a specified gatepin. Use `-longest` for the default STA graph. The `-constants` parameter is to ignore constant gatepins, *i.e.* pin tied to fixed values.

These functions stop at sequential cells. The first two detect Flip-Flops and Latches, based on the LIB file, thus require LIB to be loaded. The latter two allow for the user to specify sequential names by a prefix string, using the `-ffname` parameter. The latter two functions also return the logic cone gate pins, or components, if `-components parameter` is set, as TCL lists.

Example:

```
> get_forward_logic_cone M65C02_ALU/U878/A2 -longest
```

```
INFO: Getting Forward Logic Cone from Gatepin M65C02_ALU/U878/A2
```

```
M65C02_ALU/U878/A2 M65C02_ALU/U878/ZN M65C02_ALU/U897/A2
M65C02_ALU/U897/ZN M65C02_ALU/Out|1 M65C02_ALU/U1117/I
M65C02_ALU/U1117/ZN M65C02_ALU/U1119/A2 M65C02_ALU/U1121/A2
M65C02_ALU/U1123/A2 M65C02_ALU/U1219/A1 M65C02_ALU/U1222/A1
M65C02_ALU/U1119/ZN {M65C02_ALU/\A_reg[1]/D} M65C02_ALU/U1121/ZN
{M65C02_ALU/\X_reg[1]/D} M65C02_ALU/U1123/ZN {M65C02_ALU/\Y_reg[1]/D}
M65C02_ALU/U1219/ZN M65C02_ALU/U1220/A1 M65C02_ALU/U1222/ZN
M65C02_ALU/U1223/A2 M65C02_ALU/U1220/ZN M65C02_ALU/U1221/A4
M65C02_ALU/U1223/ZN M65C02_ALU/U1224/B M65C02_ALU/U1221/ZN
M65C02_ALU/U1222/B M65C02_ALU/U1224/ZN {M65C02_ALU/\PSW_reg[1]/D}
```

3.7 Setting Up and Reporting SDC Timing Constraints

SiHi tools also support constraints commands in the format of SDC files, that specify the desired timing behavior and operational characteristics of a digital design. SDC commands are not mandatory for timing analysis, but are recommended to be used to achieve the desired timing behavior.

3.7.1 Create Clock(s)

Creates a clock at a specified design gatepin, specifies its period, optionally its name, and a set of waveform edges. Three types of clocks are supported, user-defined (for STA), using `create_clock` command, asynchronously generated (for Asynchronous STA), and generated, *i.e.* based on other clocks.

```
create_clock <source_gatepin> -period <period_value> ?-name
<clock_name>? ?-waveform {edge_list}?
```

OPTIONS

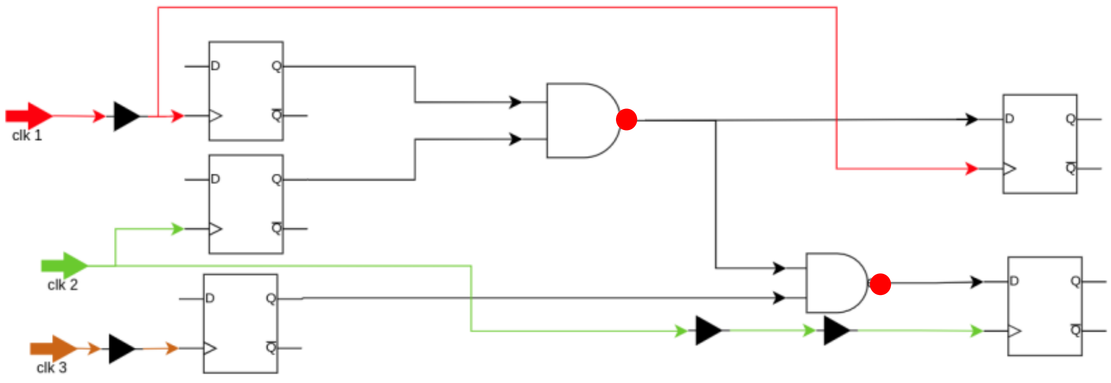
`<source_gatepin>`
the gatepin at which the clock signal is applied.

`-period <period_value>`
the clock period value, in ns.

`?-name <clock_name>?`
the clock name; this is optional.

`?-waveform {edge_list}?`
a list of waveform edges, within the clock period.

3.7.2 Multiple Clocks Domain Crossings (CDCs)



Clock Domain Crossings (CDC) occur when paths originating from 2 or more clocks converge to a common node. At least two clocks need to be created for CDCs to exist. The user needs to provide only the SDC commands to create the clocks and set input/output delays. The final timing report style remains the same as with single-clock designs, but CDC analysis is automatically used.

Si-Time supports multiple clocks. All the clocks must be created with the `create_clock` command, *e.g.*

```
create_clock clk1 -period 2 -waveform {0 1}

create_clock clk2 -period 4 -waveform {0 2}
```

3.7.3 False Paths

False paths are specific paths within a digital circuit which are to be intentionally excluded from STA and the critical timing paths. A typical example of false paths is the set of all paths starting from a circuit's reset input pin.

To set a false path use the following command.

```
set_false_path {-rise | -fall, -setup | -hold, -from {from_list} | -
rise_from {rise_from_list} | -fall_from {fall_from_list}, -through
{through_list}, -rise_through {rise_through_list}, -fall_through
{fall_through_list}, -to {to_list} | -rise_to {rise_to_list} | -
fall_to {fall_to_list}, -reset_path}
```

To report defined false paths use `list_false_path_gatepins` command.

```
list_false_path_gatepins <setup | hold>
```

3.7.4 Set Case Analysis

Specifying input constant values enables the STA engine to ignore the timing of constant logic cones. Thus, setting an input value to a constant 0 or a constant 1 produces a specific STA scenario, based on that fixed input value, aka Constant Case Analysis.

```
set_case_analysis <Boolean constant value (0 | 1 | x | z)> {<primary
input>}
```

3.7.5 Set Input Delay

To specify input delay(s) related to a specified or virtual clock, SDC command `set_input_delay` may be used.

```
set_input_delay <delay_value> ?-clock <clock_name>? ?-clock_fall? ?-
rise | -fall? ?-max | -min ?-add_delay? {port_pin_list}
```

A simple example:

```
set_input_delay 0 -clock clk1 [all_inputs]
```

3.7.6 Set Input Transition

To specify input transition time(s), SDC command `set_input_transition` may be used.

```
set_input_transition <transitionvalue> <-rise | -fall> <-min | -max>
{port_pin_list}
```

3.7.7 Set Load

To specify load(s) at a design output, SDC command `set_load` may be used.

```
set_load <loadvalue> <-min | -max> <-pin_load | -wire_load> <-
subtract_pin_load> {port_pin_list}
```

3.7.8 Set Max Fanout

To specify a maximum fanout design constraint, SDC command `set_max_fanout` may be used.

```
set_max_fanout <max fanout> <gatepinname> <buffer cell type> -
arbitrary|-capacitance|-slew
```

3.7.9 Set Output Delay

To set the output delay(s) for a specified or virtual clock, SDC command `set_output_delay` may be used.

```
set_output_delay <delay_value> ?-clock <clock_name>? ?-clock_fall? ?-
rise | -fall? ?-max | -min ?-add_delay? {port_pin_list}
```

For example:

```
set_output_delay 0.5 -clock clk1 [all_outputs]
```

3.7.10 Set Timing Derate

Derating involves multiplying STA computed delays by a user-specified factor, thereby modifying STA delay and slack values and exploring What-if delay scenarios. This command affects STA the results, *i.e.* TCL command `report_timing` output.

```
set_timing_derate <deratefactor> ?-shortest | -longest? ?-net_delay |
-cell_delay | -cell_check? ?-rise | -fall? ?-data | -clock?
{<cellnames>}
```

3.7.11 All Inputs

Creates a set of all input Top-Level ports of the current design.

```
all_inputs
```

3.7.12 All Outputs

Creates a set of all output Top-Level ports of the current design.

```
all_outputs
```

3.7.13 Current Design

Reports the name of the Top-Level module.

```
> current_design
INFO: Top-Level Module is "M65C02_ALU"
```

3.8 Other Timing Engine Parameters

3.8.1 Set Scan Pin Names, Set Special Pin Names

TCl commands set_scanpinnames/set_specialpinnames may be used used to specify library cell pin names which are to be ignored during STA and connectivity analysis, as are scan designated or special pin names, *e.g.* SI, SE, TE or SO. Scan, Special pins should be declared either pre or post Verilog in, *i.e.* before any connectivity analysis is performed.

Note that connectivity to or from Scan, Special pins is ignored. This is what distinguishes Scan, Special pins from False Paths, the connectivity information of which are still traced. The format of the two TCL commands is identical.

```
set_scanpinnames {<scanpinname1> <scanpinname2> ... <scanpinnamen>}
```

```
set_specialpinnames {<scanpinname1> <scanpinname2> ... <scanpinnamen>}
```

3.8.2 Set Timing RC Mode

Sets the Wire RC Mode as Elmore delay, Pi-Model or Sign-Off accuracy (using SPEF in).

```
set_timing_RC_mode <lumpedRC | pimodelRC | detailedRC>
```

3.8.3 Set Timing Model

Sets the delay calculation timing model used from the Liberty File, NLDM or CCS. NLDM is the default, if this command is not used.

```
set_timing_model <nldm | ccs>
```

3.8.4 Metal Layer Parameters Setup

For the Elmore and Pi-Models, LEF routing data are used to estimate the metal net length, based on BB (Bounding Box) or other approximations. To bind the timing, the lowest routing layer is used for Wire RC estimations.

To view information about the Wire RC estimations layer, use TCL command `report_lowest_layer_info`.

It is also possible to manually set Wire RC estimation parameters and override LEF data. This also makes it possible to run multiple STA analyses with different basic RC characteristics. The manual setup may be performed with a script similar to the following, where M1 is a LEF declared routing layer.

Example:

```
# Manual Metal RC data Example

# rpersq in Ohms
set_layer_rpersq M1 14

# cpersqdist in pF
set_layer_cpersqdist M1 0.09286

# lowest via R in Ohms
set_lowestviaresistance 14

#  layer edgecapacitance in pF
set_layer_edgecapacitance M1 0.0000048

# set M1 as lowest metal for STA Elmore RC Estimation
set_layer_as_lowest_metal M1

# inspect M1 layer
list_layer M1

# inspect lowest layer parameters
report_lowest_layer_info
```

3.8.5 Set Top-level I/O Pin Layers

For Top-Level I/O Pin placement, the layers are specified as horizontal and vertical using TCL command `set_toplevelpinlayers`.

```
set_toplevelpinlayers {<horizontal toplevelpinlayername> <vertical
toplevelpinlayername>}
```

3.9 Running Static Timing Analysis (STA)

3.9.1 Global, Incremental and Local STA

STA exists in three forms, (1) Global, (2) Incremental and (3) Local STA, based on the STA scope. In addition, STA corresponds to GBA (Graph-Based Analysis), unless PBA (Path-Based Analysis) is required and specifically requested.

The Global STA command is `report_timing`.

Incremental and Local STA is used for closed-loop timing constraints optimization flows, where running Global STA would simply be too costly.

Incremental Delay Analysis is performed by the `compute_delay` TCL command. Incremental Slack/RAT Analysis is performed by the `compute_slack` TCL command.

Local STA, only changes local delay values locally, 1 level back from a selected cell and n levels forward, for the interest of speed. Local STA is performed using the `get_local_timing` TCL command.

3.9.2 Report Timing TCL Command

```
report_timing ?-async? ?-corner <corner>? -delay_type <min, max> ?-
backannotate? ?-to?
```

Perform Timing Analysis on the design stored in memory. Clocked, cyclic and asynchronous designs are supported.

For information on the clocked flow, you may lookup the "STA" man page.

For cyclic and asynchronous designs, the "ASTA" man page.

The `report_timing` TCL command is indeed a wrapper, executing multiple, individual timing commands underneath, based on the design flow, *i.e*. clocked or asynchronous. The default settings, *i.e.* if the user simply types `report_timing`, the following command is executed:

```
        report_timing -longest -corner 0 -backannotate
```

For clocked circuits, `report_timing` uses the following set of individual STA commands:

`compute_delay <report_timing arguments>`, to annotate the circuit ATs and slews.

`compute_slack <report_timing arguments> -backannotate`, to annotate RATs and compute the available slack.

For cyclic and asynchronous circuits, `report_timing` uses following set of point ASTA commands:

`compute_cyclic_slews`, to annotate slews to the cyclic or asynchronous portion of the circuit; the latter may be user-specified by the `set_async_modules` command.

`async_calculate_all_eventgraph_edge_delays`, to annotate ATs, based on the annotate slews to the cyclic or asynchronous portion of the circuit.

`compute_eventgraph_period`, to compute the eventgraph period, and identify the critical cycle and event offsets.

`annotate_eventgraph_matched_gatepins_from_eventgraph_offsets`, to annotate the gatepin ATs, based on the critical cycle and event offsets. This TCL function also determines the type of design, between cyclic (1), FF-based Bundled-Data (2), Latch-based Bundled-Data (3) Mixed Latch and FF-based Bundled-Data (4).

At this point, for design type (1), no further action is needed, whereas for design type (2), i.e. FF-based Bundled-Data, which is the simplest possible flow, the following additional individual STA command are used.

`compute_delay -asynclatch`, to annotate the Bundled-Data circuit part ATs and slews.

`compute_slack`, to annotate RATs and compute the available slack.

OPTIONS

`-async`

selects the ASTA flow, instead of the default STA one.

`-corner <corner>`

selects the timing library corner, with the default corner being corner 0.

`-delay_type <min, max>`

specifies the type of analysis, as max or min, longest or shortest, late or early.

`-backannotate`

instructs the STA/ASTA engine to annotate RAT and slack for all design gatepins, not only for endpoints (FFs, Latches, POs).

`-to` for STA only; instead of displaying the critical path to the worst slack gatepin, display the critical path to the user-specified gatepin

An example run of the report_timing command, with backannotation is illustrated below

```
######################################################################
load_lef {mock_tech.lef mock.lef}
load_lib mock.lib
load_verilog mock.v
set clk_name clk
set clock_period 2
create_clock $clk_name -period $clock_period -waveform {0 1}
set_input_delay 0.0 -clock $clk_name [all_inputs]
set_output_delay 0.0 -clock $clk_name [all_outputs]
```

```
# Full STA, with Backannotation
report_timing -corner 0 -longest –backannotate
########################################################################
```

The STA critical path report produced by the above script is the following.

```
-------------------------------------
Dumping Longest Path to Gatepin keysched/\key_reg_reg[9]/D
1: clk (Delay = 0.0000000 r) (r: 0.0000000, f: 0.0000000) (Load: 6.421600) (Wireload: 0.000000)
2: sbox1/\to_invert_reg[2]/CP (dfc3d3) (Delay = 0.0000000 r) (r: 0.0000000, f: 0.0000000) (Load: 0.000000) (Wireload: 0.000000)
3: sbox1/\to_invert_reg[2]/Q (dfc3d3) (Delay = 0.1379571 r) (r: 0.0508459, f: 0.0478824) (Load: 0.020400) (Wireload: 0.000000)
4: sbox1/U201/A2 (nd21d1) (Delay = 0.1379571 r) (r: 0.0508459, f: 0.0478824) (Load: 0.000000) (Wireload: 0.000000)
5: sbox1/U201/ZN (nd21d1) (Delay = 0.2508640 f) (r: 0.1301636, f: 0.0751856) (Load: 0.011100) (Wireload: 0.000000)
6: sbox1/U189/A1 (xn21d1) (Delay = 0.2508640 f) (r: 0.1301636, f: 0.0751856) (Load: 0.000000) (Wireload: 0.000000)
7: sbox1/U189/ZN (xn21d1) (Delay = 0.6478543 r) (r: 0.3125653, f: 0.1111754) (Load: 0.013000) (Wireload: 0.000000)
8: sbox1/U227/A2 (xn21d3) (Delay = 0.6478543 r) (r: 0.3125653, f: 0.1111754) (Load: 0.000000) (Wireload: 0.000000)
9: sbox1/U227/ZN (xn21d3) (Delay = 0.8632686 r) (r: 0.1106138, f: 0.0979935) (Load: 0.016900) (Wireload: 0.000000)
10: sbox1/U226/A2 (xn21d2) (Delay = 0.8632686 r) (r: 0.1106138, f: 0.0979935) (Load: 0.000000) (Wireload: 0.000000)
11: sbox1/U226/ZN (xn21d2) (Delay = 0.9933919 f) (r: 0.0972783, f: 0.1268229) (Load: 0.018900) (Wireload: 0.000000)
12: sbox1/U236/A2 (xn21d3) (Delay = 0.9933919 f) (r: 0.0972783, f: 0.1268229) (Load: 0.000000) (Wireload: 0.000000)
13: sbox1/U236/ZN (xn21d3) (Delay = 1.1838446 f) (r: 0.1137119, f: 0.1249847) (Load: 0.017200) (Wireload: 0.000000)
14: sbox1/U151/A1 (nr21d1) (Delay = 1.1838446 f) (r: 0.1137119, f: 0.1249847) (Load: 0.000000) (Wireload: 0.000000)
15: sbox1/U151/ZN (nr21d1) (Delay = 1.4157909 r) (r: 0.1473386, f: 0.0759839) (Load: 0.005900) (Wireload: 0.000000)
16: sbox1/U150/A2 (xn21d1) (Delay = 1.4157909 r) (r: 0.1473386, f: 0.0759839) (Load: 0.000000) (Wireload: 0.000000)
17: sbox1/U150/ZN (xn21d1) (Delay = 1.6196589 r) (r: 0.1938555, f: 0.0753585) (Load: 0.005900) (Wireload: 0.000000)
18: sbox1/U149/A2 (xn21d1) (Delay = 1.6196589 r) (r: 0.1938555, f: 0.0753585) (Load: 0.000000) (Wireload: 0.000000)
19: sbox1/U149/ZN (xn21d1) (Delay = 1.7854134 f) (r: 0.3209100, f: 0.1175320) (Load: 0.013500) (Wireload: 0.000000)
20: sbox1/U235/A1 (xn21d3) (Delay = 1.7854134 f) (r: 0.3209100, f: 0.1175320) (Load: 0.000000) (Wireload: 0.000000)
21: sbox1/U235/ZN (xn21d3) (Delay = 2.0946280 f) (r: 0.1211849, f: 0.1049624) (Load: 0.021100) (Wireload: 0.000000)
22: sbox1/U177/A2 (xn21d2) (Delay = 2.0946280 f) (r: 0.1211849, f: 0.1049624) (Load: 0.000000) (Wireload: 0.000000)
23: sbox1/U177/ZN (xn21d2) (Delay = 2.3263951 f) (r: 0.0927255, f: 0.1230932) (Load: 0.016700) (Wireload: 0.000000)
24: sbox1/U125/A1 (xn21d1) (Delay = 2.3263951 f) (r: 0.0927255, f: 0.1230932) (Load: 0.000000) (Wireload: 0.000000)
25: sbox1/U125/ZN (xn21d1) (Delay = 2.7299464 r) (r: 0.3046416, f: 0.1459423) (Load: 0.012500) (Wireload: 0.000000)
26: sbox1/U100/A1 (xn21d1) (Delay = 2.7299464 r) (r: 0.3046416, f: 0.1459423) (Load: 0.000000) (Wireload: 0.000000)
27: sbox1/U100/ZN (xn21d1) (Delay = 2.8942152 f) (r: 0.2931912, f: 0.1349530) (Load: 0.011800) (Wireload: 0.000000)
28: sbox1/U99/A2 (xn21d1) (Delay = 2.8942152 f) (r: 0.2931912, f: 0.1349530) (Load: 0.000000) (Wireload: 0.000000)
29: sbox1/U99/ZN (xn21d1) (Delay = 3.2381854 r) (r: 0.2351850, f: 0.1562174) (Load: 0.008200) (Wireload: 0.000000)
30: sbox1/U232/A2 (ond1d4) (Delay = 3.2381854 r) (r: 0.2351850, f: 0.1562174) (Load: 0.000000) (Wireload: 0.000000)
31: sbox1/U232/ZN (ond1d4) (Delay = 3.3784699 f) (r: 0.1787528, f: 0.1120976) (Load: 0.032800) (Wireload: 0.000000)
32: ks1/U501/A1 (xn21d1) (Delay = 3.3784699 f) (r: 0.1787528, f: 0.1120976) (Load: 0.000000) (Wireload: 0.000000)
33: ks1/U501/ZN (xn21d1) (Delay = 3.6139741 f) (r: 0.2366760, f: 0.0969107) (Load: 0.008400) (Wireload: 0.000000)
34: ks1/U488/A2 (xn21d1) (Delay = 3.6139741 f) (r: 0.2366760, f: 0.0969107) (Load: 0.000000) (Wireload: 0.000000)
35: ks1/U488/ZN (xn21d1) (Delay = 3.9431216 r) (r: 0.2379989, f: 0.1025650) (Load: 0.008500) (Wireload: 0.000000)
36: ks1/U487/A2 (xn21d1) (Delay = 3.9431216 r) (r: 0.2379989, f: 0.1025650) (Load: 0.000000) (Wireload: 0.000000)
37: ks1/U487/ZN (xn21d1) (Delay = 4.1801361 r) (r: 0.2365391, f: 0.1023613) (Load: 0.008400) (Wireload: 0.000000)
38: ks1/U486/A2 (xn21d1) (Delay = 4.1801361 r) (r: 0.2365391, f: 0.1023613) (Load: 0.000000) (Wireload: 0.000000)
39: ks1/U486/ZN (xn21d1) (Delay = 4.3602338 r) (r: 0.1393310, f: 0.0762421) (Load: 0.002500) (Wireload: 0.000000)
```

```
40: ks1/U484/A1 (ond1d1) (Delay = 4.3602338 r) (r: 0.1393310, f: 0.0762421) (Load: 0.000000) (Wireload: 0.000000)

41: ks1/U484/ZN (ond1d1) (Delay = 4.4289040 f) (r: 0.1471163, f: 0.1298782) (Load: 0.002900) (Wireload: 0.000000)

42: ks1/\key_reg_reg[9]/D (dfc3d3) (Delay = 4.4289040 f) (r: 0.1471163, f: 0.1298782) (Load: 0.000000) (Wireload: 0.000000)
--------------------------------------
Data required arrival time: 1.7436754

Data arrival time: 4.4289040

Slack: (VIOLATED)        -2.685229

WARNING: Components with all Outputs Unconnected Identified in Netlist.
```

The STA TNS may be reported using the `report_TNS` TCL command, and its result is the following.

```
> report_TNS

INFO: Total Negative Slack (longest):  -56743.1073253
```

3.9.3 Compute Delay as a separate TCL command

Delay Computation, *i.e.* delay and slew analysis, may be performed by using the `compute_delay` TCL command. Using the `compute_delay` command directly does not perform slack analysis, so slack information will not be reported or be available. Its format is as follows:

```
compute_delay -corner <corner_number> -longest | -shortest ?-
incremental? ?-ccs?
```

where `–incremental` corresponds to performing incremental delay, slack analysis, and requires incremental updates to be specified by respective incremental STA, ECO commands, and `-ccs` corresponds to using the CCS mode.

3.9.4 Compute Slack as a separate TCL command

Slack Computation, *i.e.* endpoint slack computation and slack propagation, requires delay computation data to be available. Slack Computation may be performed using the `compute_slack` TCL command. Its format is as follows:

compute_slack -corner <corner_number> -longest | -shortest ?-backannotate? ?-to <gatepin>? ?-incremental?

where `–backannotate` corresponds to performing slack propagation, and not solely compute slack at the circuit endpoints, `–to` gatepin may be used to specify a specific gatepin for slack computation to begin from, and `–incremental` performs incremental slack analysis, again provided that incremental circuit updates were previously specified by respective incremental STA, ECO commands.

3.9.5 Relative Timing (RT) Constraints and RT Paths

Relative Timing Constraints is a way to set specific path to path constraints, in PBA fashion, pre or post P&R, *i.e.* include wire RC delays.RT Constraints are specified in two phases. RT Paths must be specified first, as a from and to gatepin specification, and then an RT constraint may be specified between sets of these defined paths.

RT Path Collections

To specify a set of RT paths, TCL command set_RT_path is used:

```
set_RT_path -(rise/fall)from <pin> -(rise/fall)to <pin> -corner <idx>
-stopatsequential 0|1 –name <collection name>
```

To list the specified RT paths, TCL command list_RT_paths may be used:

```
list_RT_paths ?-file <file name>? ?-verbose? ?-max|-min?
```

The above command lists all RT paths in ascending (-min) or descending (-max) order. This command may output to a file and displays pin delays and slews, per Path.

A specified collection may be dumped by TCL command list_RT_path:

```
list_RT_path <collection name> ?-file <file name>? ?-verbose? ?-max|-
min?
```

To remove an RT path collection, remove_RT_path may be used:

```
remove_RT_paths <path collection name> | -all
```

This command removes all paths and all associated RT constraints!

RT Constraints

To set a constraint between two paths, or path collections, TCL command set_RT_constraint is used:

```
set_RT_constraint –primary_path <path collection> -reference_path
<path collection> -margin <value> -less|-greater
```

Where the `–less` and `–more` parameters specify the RT constraint slack margin direction, as follows:

- `-less` implies (primary path delay – reference path delay) < slack margin
- `-more` implies (primary path delay – reference path delay) > slack margin

To list all specified RT constraints, TCL command list_RT_constraints may be used, which uses the following format:

```
Primary path: <path name>

Reference path: <path name>

Margin: <margin>

Constraint Type: Less Than / Greater Than
```

To remove any specified RT constraints, TCL command `remove_RT_constraint` may be used:

```
remove_RT_constraints <path name> | -all –match_primary_paths | -
match_reference_paths –match_all_paths
```

Once RT constraints have been specified, RT violations may be checked, by using TCL command `report_RT_violations`:

```
report_RT_violations ?-file <file name>
```

3.9.6 SDF Annotation and SDF Out for Verilog Simulation

SDF out is used for annotating timing delays for Verilog gate-level simulation.to output SDF. The related TCL command is `save_sdf`, and has the following format.

```
save_sdf <filename> -corner <cornerindex> -longest | -shortest
```

After the SDF file is saved, it may be used to annotate the delays for Verilog simulation purposes. Note that an SDF file contains delays only and zero slews.

3.9.7 SPEF In and SPEF Annotation

SPEF in is supported by using the `load_spef` command.

```
load_spef -longest | -shortest ?-errors? {<filenames>}
```

Any SPEF issues detected will produce tool WARNING messages. To manually inspect annotated parasitics per design gate pin, use the report_gatepin_annotated_parasitics as follows.

```
report_gatepin_annotated_parasitics {<gatepinnameslist>} ?-longest | -
shortest? -corner <cornernum>
```

The latter will produce a list of SPEF nodes per gatepin and associated SPEF nets.

Example:

```
> report_gatepin_annotated_parasitics ring/U3/ZN -longest -corner 0
--------------------------------------
SPEF Nodes (7):
0: 2:2 (Level: 0) (Node C: 0.000159)
1: 2:3 (Level: 0) (Node C: 0.000159)
2: 2:4 (Level: 0) (Node C: 0.000053)
3: 2:5 (Level: 0) (Node C: 0.000053)
4: ring/U3/ZN (Level: 0) (Node C: 0.000000) (pin C: 0.000000, pin rise C: 0.000000,
pin fall C: 0.000000)
5: 2:1 (Level: 0) (Node C: 0.000000)
6: ring/U1/A2 (Level: 0) (Node C: 0.000000) (pin C: 0.008600, pin rise C: 0.000000,
pin fall C: 0.000000)
--------------------------------------
Driver Node: 4 (ring/U3/ZN)
Is Levelised: 0
Capacitors: 4, Resistors: 6, Supplies: 0
Total Capacitance (without pin C): 0.000424
--------------------------------------
*CAP
1 2:2 0.000159
2 2:3 0.000159
3 2:4 0.000053
4 2:5 0.000053
*RES
1 2:5 ring/U3/ZN 0.001500
2 2:4 2:5 0.000059
```

```
3 2:3 2:4 0.002000
4 2:2 2:3 0.000228
5 2:1 2:2 0.002000
6 ring/U1/A2 2:1 0.001500
--------------------------------------
```

3.10 Engineering Change Order (ECO) Commands

Si-Time provides the following ECO commands.

3.10.1 Compute Cyclic Slews

`eco_compute_cyclic_slews`

Compute and propagate async slews for a list of cell pins.

3.10.2 Force Gatepin Delay Slew

`eco_force_gatepin_delay_slew`

Set rise/fall delay and slew for cell pin.

3.10.3 Move cell

`eco_move_cell`

Move <cell> to a specific location.

3.10.4 Set Component Libcell Type

`eco_set_component_libcell_type`

Set <cell> .lib cell type.

3.10.5 Insert New Component

`eco_insert_new_component`

Insert new cell to the graph.

3.10.6 Replicate Component

`eco_replicate_component`

Create a replica of <cell> with the same predecessors. Successors must be specified.

3.10.7 Swap Component Pins

`eco_swap_component_pins`

Swap connections of <cell> input pins.

3.10.8 Downsize Component

`eco_downsize_component`

Set <cell> to next smaller lib cell type in terms of area.

3.10.9 Upsize Component

```
eco_upsize_component
```

Set <cell> to next bigger lib cell type in terms of area.

Configure RC wire model.

3.10.10 Remove Buffer

```
remove_buffer
```

Remove a buffer from the circuit. This command will not work on cells with multiple inputs or outputs.

3.10.11 Estimate Pi-Model Delay and Slew

```
estimate_pi_model_delay_and_slew
```

Report delay for a pi model for a specified driver PWL.

3.10.12 Estimate Pi-Model PWL Output

```
estimate_pi_model_pwl_output
```

Create and export Piecewise Linear (PWL) for a pi-model output for plotting.

3.10.13 Estimate Detailed RC Model Delay and Slew

```
estimate_detailed_RC_model_delay_and_slew
```

Report delay for a spefnet for a specified driver PWL or slew value.

3.10.14 Estimate Spefnet PWL Output

```
estimate_spefnet_pwl_output
```

Create and export PWL for a spefnet node for plotting.

3.10.15 Estimate LEF Layer RC Delay and Slew

```
estimate_lef_layer_RC_delay_slew
```

Report wire delay and slew for specified LEF layer.

3.10.16 Set Layer as Lowest Metal

```
set_layer_as_lowest_metal
```

Set LEF layer as lowest, to be used for wire delay calculations.

3.10.17 Report From Gatepin-to-Gatepin Path

```
report_from_gatepin_to_gatepin_path
```

Report delay and slew for paths specified by –from and –to cell pins.

3.11 STA Analysis Result Commands

Si-Time provides the following commands to report timing information.

3.11.1 Report Gatepin Spef Info

```
report_gatepin_spef_info {<gatepinnameslist>} ?-longest | -shortest? -
corner <cornernum>
```

3.11.2 Report Gatepin Spefnet Spefnode Connections

```
report_gatepin_spefnet_spefnode_connections <gatepinname>
<spefnodename> ?-longest | -shortest? -corner <cornernum>
```

3.11.3 Report Global Slack

```
report_global_slack
```

3.11.4 Report CCS Timing Arc Delay

```
report_CCS_timing_arc_delay -gatepin <output_gatepin_name> -
related_pin <related_input_lib_pin_name> -corner corner_number -
input_transition <input_transition_value> -rise | -fall ?-pimodel
<cnear> <resistance> <cfar>? ?-spefnet?
```

3.11.5 Report Cell Equivalent Pulldown Resistance

```
report_cell_equivalent_pulldown_resistance
```

3.11.6 Report Cell Equivalent Pullup Resistance

```
report_cell_equivalent_pullup_resistance
```

3.11.7 Report Relative Clocks Relationships

```
report_relative_clocks_relationships
```

3.11.8 Report Cell Logic Type

```
report_cell_logic_type
```

3.11.9 Report Cell Timing Arc Containers

```
report_cell_timing_arc_containers
```

3.11.10 Report Cell Timing Constraints

```
report_cell_timing_constraints
```

3.11.11 Report Cell Timing Info

```
report_cell_timing_info
```

3.11.12 Report Clocks

```
report_clocks
```

3.11.13 Report Longest Path To

```
report_longest_path_to <pinname>, -rise | -fall
```

3.11.14 Report RT Violations

```
report_RT_violations
```

3.11.15 Report Lowest Layer Info

```
report_lowest_layer_info
```

3.11.16 Report Lowest Pin Metals

```
report_lowest_pin_metals
```

3.11.17 Report Shortest Path To

```
report_shortest_path_to <pinname>, -corner <corner_number>, -rise | -
fall
```

3.11.18 Report Specialnets Sizes

```
report_specialnets_sizes
```

3.11.19 Report Timing

```
report_timing
```

3.11.20 Report Timing Arc Constraint

```
report_timing_arc_constraint -cell <cell_name> -pin <pin_name> -
related_pin <related_pin_name> -corner corner_number -arc <arc_name> -
relatedpintransition <related_pin_transition_value> -
constrainedpintransition <constrained_pin_transition_value>
```

3.11.21 Report Critical Capacitance Ratios

```
report_critical_capacitance_ratios
```

3.11.22 Report Timing Arc Delay

```
report_timing_arc_delay -cell <cell_name> -pin <pin_name> -related_pin
<related_pin_name> -corner corner_number -input_transition
<input_transition_value> (-arc <arc_name> -output_load
<output_load_value>) | (-pimodel <cnear> <resistance> <cfar>) ?-rise |
-fall?)
```

3.11.23 Report Critical Cycle

```
report_critical_cycle
```

3.11.24 Report Timing Model

```
report_timing_model
```

3.11.25 Report Critical Cycle Period

```
report_critical_cycle_period
```

3.11.26 Report Timing RC Mode

```
report_timing_RC_mode
```

3.11.27 Report TNS

```
report_TNS
```

3.11.28 Report Delay

```
report_delay <gatepinname> -corner <corner_number> -longest | -
shortest ?-successors? ?-pimodel? ?-ccs?
```

3.11.29 Report Delay Calculation

```
report_delay_calculation -gatepin <pinname> -longest | -shortest
```

3.11.30 Report Top Longest Delay Paths

```
report_top_longest_delay_paths -paths <number>
```

3.11.31 Report Top Longest Levels Paths

```
report_top_longest_levels_paths <num>
```

3.11.32 Report Top Longest Slack Paths

```
report_top_longest_slack_paths -paths <number>
```

3.11.33 Report Top Longest Slack Paths Components

```
report_top_longest_slack_paths_components -paths <number>
```

3.11.34 Report Top Shortest Delay Paths

```
report_top_shortest_delay_paths -paths <number>
```

3.11.35 Report Top Shortest Levels Paths

```
report_top_shortest_levels_paths <num>
```

3.11.36 Report Top Shortest Slack Paths

```
report_top_shortest_slack_paths -paths <number>
```

3.11.37 Report FF Type

```
report_ff_type <ffname> -corner <cornerindex>
```

3.11.38 Report Total Negative Slack

```
report_total_negative_slack
```

3.11.39 Report From Gatepin-to-Gatepin Path

```
report_from_gatepin_to_gatepin_path (-from | -rise_from | -fall_from)
<gatepinname> (-to | -rise_to | -fall_to) <gatepinname> -corner
<cornerindex> -longest | -shortest ?-file <filename>? ?-
stopatsequential <0 | 1 (default)>? ?-min? ?-verbose? ?-gui? ?-path
<pathindex>?
```

3.11.40 Report Gatepin Annotated Parasitics

```
report_gatepin_annotated_parasitics {<gatepinnameslist>} ?-longest | -
shortest? -corner <cornernum>
```

3.11.41 Report Gatepins Capacitance Ratios

```
report_gatepins_capacitance_ratios -corner <cornerindex> {<gatepins
list>} ?-longest | -shortest?
```

3.12 Graphical User Interface (GUI)

[TBD]

4. Advanced Features

4.1 SET (Single Event Transient) Pulse STA Analysis

The STA engine supports an SET analysis layer. This may be used for (1) SET pulse generation at arbitrary circuit points and (2) SET pulse propagation from the SET Fault point forward towards circuit endpoints. SET pulse propagation is probabilistic and depends on signal 0 or 1 probabilities set by the user, to reflect testbench values. Pulse propagation will perform both logical masking and electrical masking, as well as pulses accumulation for reconvergent circuit parts and derive both the probability and pulse width reachable at circuit endpoints, *i.e.* FF inputs or Top-Level Outputs.

More information about SET Flows will be available soon.

[TBD]

4.2 SPICE Out

SPICE Out is supported for STA, SPICE correlation. This may be performed manually or automatically. For SPICE Out, the technology library standard cells must be provided as a list of SPICE subcircuits.

It is possible to output a specified Verilog module as a SPICE netlist, using the `save_module_spice_deck` TCL command. It is also possible to output a path, or a forward logic cone using SPICE, from a specified gatepin, using the `save_SET_scenario_traversed_components_to_SPICE_deck` TCL command.

```
save_module_spice_deck -setupsubcircuit {<subcircuitfiles>} -output
<outputfilename> -powernet <powernetname> -groundnet <groundnetname>
?-module <modulename>? ?-gatepins {<gatepinnames>}? ?-includewires?
```

```
save_SET_scenario_traversed_components_to_spice_deck <scenario index>
-setupsubcircuit {<subcircuitfiles>} -powernet <powernetname> -
groundnet <groundnetname> -output <output file name> -corner <corner
index> ?-include {<includefiles>}? ?-spectre? ?-pwl <"PWL(...)">? ?-
includewires?
```

More information about the manual SPICE Out will be available soon.

[TBD]

4.3 STA vs SPICE Automated Flow(s)

STA vs SPICE Automated Flows also exist. These automatically generate SPICE Out, run SPICE or SPECTRE and measure the error % between an STA result and the SPICE result.

These include (1) STA vs SPICE for circuit paths, (2) STA vs SPICE for wire RC interconnects and (3) STA vs SPICE for ASTA.

STA vs SPICE for wire RC interconnects is performed using the `estimate_pi_model_delay_and_slew` or `estimate_detailed_RC_model_delay_and_slew` TCL commands. The former corresponds to a Pi-Model of an RC interconnect, where the Cnear, Cfar and Resistance values are provided by the user to the TCL command, based on the values of the respective interconnect in the circuit layout. The latter corresponds to a SPEF (DetailedRC) interconnect, the parasitics of which are in memory.

The parameters of the two TCL commands are the following.

```
estimate_pi_model_delay_and_slew <cnear> <resistance> <cfar> {<input
timepoint1> <input voltage1> <input timepoint2> <input voltage2> ... }
?-xyce? -corner <corner index>
```

```
estimate_detailed_RC_model_delay_and_slew <gatepin name> ({<input
timepoint1> <input voltage1> <input timepoint2> <input voltage2> ... }
| -rise_slew <value> | -fall_slew <value>) (-spef_node <spef node
name> | -xyce) -corner <corner index> ?-longest (default) | -shortest?
?-addpincapacitance?
```

A user-specified waveform must be provided for both functions. This is the driver waveform seen by the wire RC interconnect. To correlate against SPICE the `-xyce` parameter must be used, as Xyce is the default SPICE Out language for this command. The Xyce simulator will be automatically invoked on the generated wire RC interconnect SPICE netlist, and the delay and slew errors between the STA waveform propagation and the measured SPICE result will be reported.

STA vs SPICE for ASTA is performed using the `run_step_simulation_eventgraph` TCL command.

```
run_step_simulation_eventgraph ?-spice -stimulusfile
<stimulusfilepath> -meascommandfile <meascommandfilepath> -
measvaluesfile <measvaluesfile> -powernet <powernetname> -voltage
<voltagevalue> -groundnet <groundnetname> -spectrescript
<spectrescriptpath>? ?-steps <stepsnum>?
```

More information about the automated SPICE Out flows will be available soon.

[TBD]

4.4 Slack Comparison against PrimeTime

It is possible to compare the slack report of Si-Time with PrimeTime global slack reports automatically. To perform the comparison, the PrimeTime global slack report should be generated, using the PrimeTime `report_global_slack` TCL command, and saved into a file.

This report may be loaded into Si-Time using the `load_primetime_global_slack_report` TCL command. Once PrimeTime data are loaded into Si-Time, TCL command `check_slack_against_primetime` reports the difference between the two tools as a percentage error. A threshold parameter may be used for comparison purposes. An example is illustrated below.

```
> load_primetime_global_slack_report
scripts/CDC/Clock_Domain_Crossings/cdc1-
experiments/NLDM/setup/$designname-global_slack-single_clock.pt

WARNING: Report Gatepin cdc1/launch1/SDN NOT Found!

WARNING: Report Gatepin cdc1/launch2/SDN NOT Found!

WARNING: Report Gatepin cdc1/capture/SDN NOT Found!

check_slack_against_primetime -threshold 0.01

Longest STA Slack Comparison

--------------------------------------

WARNING: Infinite Rise RAT at Constrained Gatepin cdc1/launch1/CP!

WARNING: Infinite Rise RAT at Constrained Gatepin cdc1/capture/CP!

WARNING: Infinite Rise RAT at Constrained Gatepin cdc1/launch2/CP!

INFO: Average Rise Slack Error: 0.00%, Average Fall Slack Error: -
0.00%

INFO: Total Number of Gatepins: 19

INFO: Infinite Rise RAT Gatepins Number: 3, Infinite Fall RAT Gatepins
Number: 0
```

The first set of WARNINGs about gatepins not being found when the PrimeTime global slack report is loaded are, due to these gatepins being unconnected. The second set of WARNINGs related to Infinite RAT times is related to RAT to clock pins, which is not defined. Both sets of WARNINGs may be ignored.

4.5 Latch Timing Analysis

Latch STA is supported for Synchronous Latch designs as well as Asynchronous Bundled-Data Latch Designs. Multiple Clock Phases and Time Borrowing are supported.

Synchonous Latch STA deviates from the report_timing STA flow reviewed earlier and requires specific Latch STA TCL commands to work properly, specifically TCL command `compute_latch_delays_fast`.

Example:

```
######################################################################

# load library #

load_lib {mock.lib}

# load design #

load_verilog design.v

# sdc commands; clock phases #
```

```
create_clock phi1 -name "phase1" -period 10 -waveform {0 5}
create_clock phi2 -name "phase2" -period 10 -waveform {5 10}
# compute delay #
compute_delay -corner 0 -longest
# Worst Slew propagation in case of Cyclic Circuit #
set_latchcyclicslews 0
# init Latch Graph #
init_latchgraph -corner 0
# Latch STA Methodology #
compute_latch_delays_fast -corner 0
# Constraint Verification
verify_latch_constraints -corner 0
# Report Command - General Timing Information #
report_latch_timing -corner 0
# Report Command - Worst Path Analysis #
report_latch_worst_path_analysis -corner 0
######################################################################
```

4.5.1 Report Latch Delay Values

```
report_latch_delay_values <latchname>
```

4.5.2 Report Latchgraph

```
report_latchgraph
```

4.5.3 Report Latchgraph Edge Delay

```
report_latchgraph_edge_delay -from <launchlatchname> -to
<capturelatchname> -corner <cornervalue>
```

4.5.4 Report Latch Slack

```
report_latch_slack -corner <cornervalue>
```

4.5.5 Report Latch Timing

```
report_latch_timing -corner <cornervalue>
```

4.5.6 Report Latch Type

```
report_latch_type <latchname> -corner <cornerindex>
```

4.5.7 Report Latch Worst Path Analysis

```
report_latch_worst_path_analysis -corner <cornerindex>
```

More information about the automated Latch STA flows will be available soon.

[TBD]

4.6 Asynchronous Static Timing Analysis (ASTA)

Si-Time supports natively both cyclic and asynchronous circuits and is compatible with both the Petrify and WorkCraft asynchronous circuits synthesis tools.

ASTA for cyclic or asynchronous circuits identifies the circuit critical cycle, *i.e.* its longest cycle, based on the worst-case slews propagated across all cycles. ASTA longest cycle analysis may take place post-synthesis, post-placement and post-routing (SPEF in). It may also be combined with the AIPO (Asynchronous In-Place Optimisation) flow, to optimise critical cycle delay.

ASTA takes place in 4 steps post design import in Si-Time.

i. **Setting of asynchronous modules**, if the design is Bundled-Data (BD), *i.e.* composed of both an asynchronous control section and a BD Datapath section; this may be performed using the set_async_modules TCL command,
ii. **Importing an STG file (.g) (Optional)**, if ASTA should model the environment of the circuit as well, and the latter is part of an STG file; this may be performed using the `load_g` TCL command. Note that if an STG file is loaded, ASTA takes place per STG event, by default, instead of per individual gatepin,
iii. **Extracting the Reduced Event Timing Graph (RETG)** is necessary before running core ASTA, to build the cyclic circuit model; this may be performed using the `extract_eventgraph` TCL command. To visually inspect the Event Graph, the `display_eventgraph` TCL command may be used,
iv. **Running ASTA Analysis is the last step**, where circuit cell and wire RC delays will be used to annotate the Event Graph and compute the longest cycle delay; this may be performed using the `report_timing -async` TCL command. ASTA analysis produces an `allcycles.temp` file in directory `/tmp/<username>/`. This file contains the complete list of all asynchronous circuit cycles and their annotated delays and slews. If `display_eventgraph` command is used post ASTA analysis, the critical cycle is highlighted on the Event Graph, to visualise it within the context of the entire circuit.

Both asynchronous and BD circuits are handled in exactly the same way, using the same TCL flow scripts. BD circuits are special, as they also require STA analysis, so they are handled an ASTA, STA combination as detailed in the relevant section below.

4.6.1 Asynchronous Control Circuit ASTA Example

Below, we present an ASTA example without a BD Datapath for simplicity.

```
set LIB mock.lib
set DESIGN mock.v
load_lef {mock.lef mock_tech.lef}
```

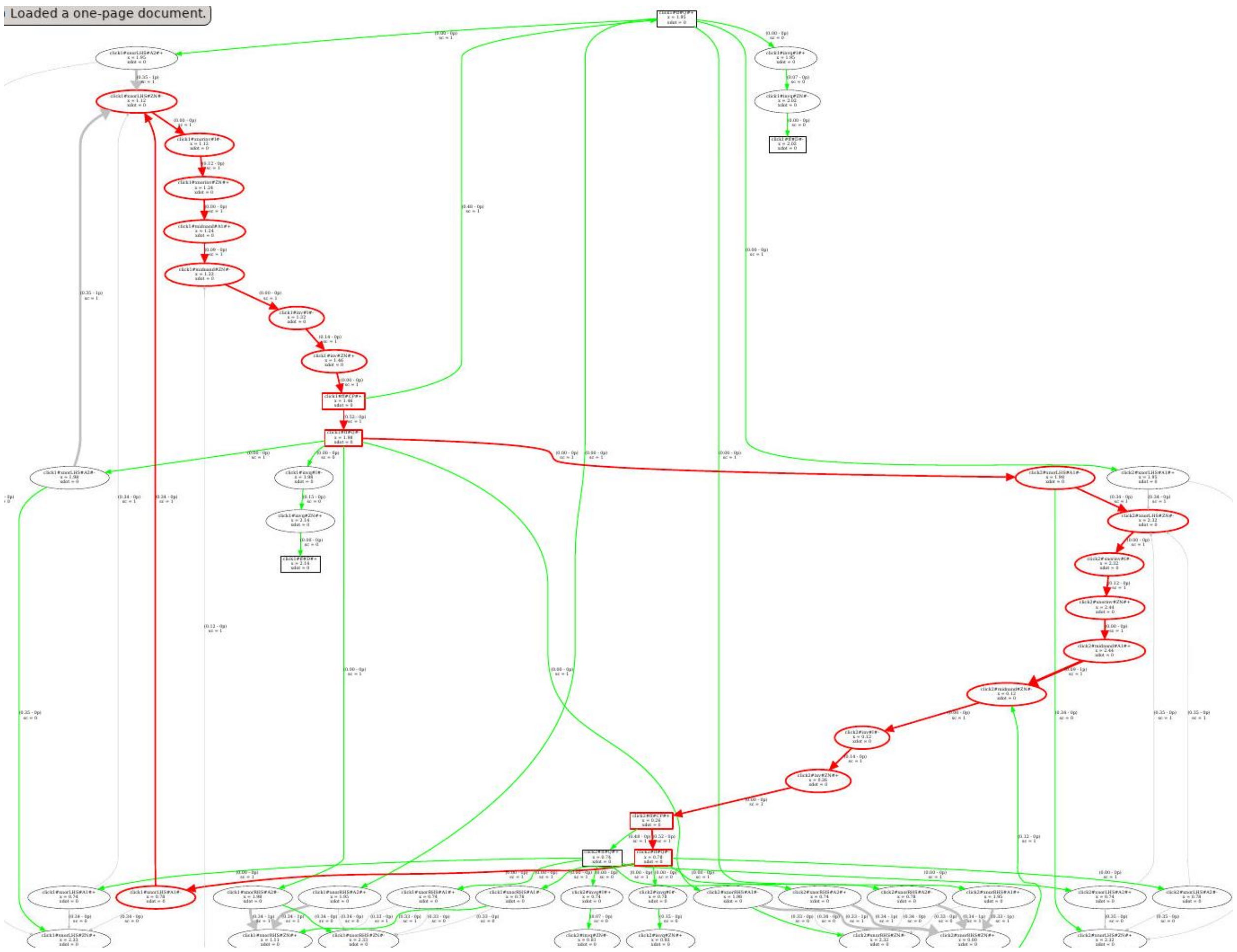
```
# load .lib file #
load_lib $LIB
# load Verilog netlist #
load_verilog $DESIGN
# extract Event Graph automatically
extract_eventgraph
report_timing -async -corner 0 -longest
```

This is the corresponding circuit critical cycle report, produced using NLDM models, post-synthesis, *i.e.* with no placement data.

```
Startpoint: click2#midnand#ZN#-

1: Node click2#inv#I#-, myclick2/inv/I (in01d1) (Delay = 0.0000000 f) (Incr = 0.0000000 f) (r:
0.0884399, f: 0.0728929) (Load: 0.000000) (Wireload: 0.000000)

2: Node click2#inv#ZN#+, myclick2/inv/ZN (in01d1) (Delay = 0.1382565 r) (Incr = 0.1382565 r) (r:
0.0794908, f: 0.0459785) (Load: 0.004000) (Wireload: 0.000000)

3: Node click2#ff#CP#+, myclick2/ff/CP (dfn3d1) (Delay = 0.1382565 r) (Incr = 0.0000000 r) (r:
0.0794908, f: 0.0459785) (Load: 0.000000) (Wireload: 0.000000)

4: Node click2#ff#Q#-, myclick2/ff/Q (dfn3d1) (Delay = 0.6610897 f) (Incr = 0.5228332 f) (r:
0.2451570, f: 0.1682000) (Load: 0.021100) (Wireload: 0.000000)

5: Node click1#xnorLHS#A1#-, myclick1/xnorLHS/A1 (xn21d1) (Delay = 0.6610897 f) (Incr = 0.0000000
f) (r: 0.2451570, f: 0.1682000) (Load: 0.000000) (Wireload: 0.000000)

6: Node click1#xnorLHS#ZN#-, myclick1/xnorLHS/ZN (xn21d1) (Delay = 0.9994012 f) (Incr = 0.3383115
f) (r: 0.2101227, f: 0.1084262) (Load: 0.002500) (Wireload: 0.000000)

7: Node click1#xnorinv#I#-, myclick1/xnorinv/I (in01d1) (Delay = 0.9994012 f) (Incr = 0.0000000
f) (r: 0.2101227, f: 0.1084262) (Load: 0.000000) (Wireload: 0.000000)

8: Node click1#xnorinv#ZN#+, myclick1/xnorinv/ZN (in01d1) (Delay = 1.1180944 r) (Incr = 0.1186932
r) (r: 0.0598848, f: 0.0576373) (Load: 0.001700) (Wireload: 0.000000)

9: Node click1#midnand#A1#+, myclick1/midnand/A1 (nd21d1) (Delay = 1.1180944 r) (Incr = 0.0000000
r) (r: 0.0598848, f: 0.0576373) (Load: 0.000000) (Wireload: 0.000000)

10: Node click1#midnand#ZN#-, myclick1/midnand/ZN (nd21d1) (Delay = 1.2061474 f) (Incr =
0.0880529 f) (r: 0.0884399, f: 0.0728929) (Load: 0.002500) (Wireload: 0.000000)

11: Node click1#inv#I#-, myclick1/inv/I (in01d1) (Delay = 1.2061474 f) (Incr = 0.0000000 f) (r:
0.0884399, f: 0.0728929) (Load: 0.000000) (Wireload: 0.000000)

12: Node click1#inv#ZN#+, myclick1/inv/ZN (in01d1) (Delay = 1.3444039 r) (Incr = 0.1382565 r) (r:
0.0794908, f: 0.0459785) (Load: 0.004000) (Wireload: 0.000000)

13: Node click1#ff#CP#+, myclick1/ff/CP (dfn3d1) (Delay = 1.3444039 r) (Incr = 0.0000000 r) (r:
0.0794908, f: 0.0459785) (Load: 0.000000) (Wireload: 0.000000)

14: Node click1#ff#Q#-, myclick1/ff/Q (dfn3d1) (Delay = 1.8672371 f) (Incr = 0.5228332 f) (r:
0.2451570, f: 0.1682000) (Load: 0.021100) (Wireload: 0.000000)

15: Node click2#xnorLHS#A1#-, myclick2/xnorLHS/A1 (xn21d1) (Delay = 1.8672371 f) (Incr =
0.0000000 f) (r: 0.2451570, f: 0.1682000) (Load: 0.000000) (Wireload: 0.000000)

16: Node click2#xnorLHS#ZN#-, myclick2/xnorLHS/ZN (xn21d1) (Delay = 2.2055485 f) (Incr =
0.3383115 f) (r: 0.2101227, f: 0.1084262) (Load: 0.002500) (Wireload: 0.000000)

17: Node click2#xnorinv#I#-, myclick2/xnorinv/I (in01d1) (Delay = 2.2055485 f) (Incr = 0.0000000
f) (r: 0.2101227, f: 0.1084262) (Load: 0.000000) (Wireload: 0.000000)

18: Node click2#xnorinv#ZN#+, myclick2/xnorinv/ZN (in01d1) (Delay = 2.3242418 r) (Incr =
0.1186932 r) (r: 0.0598848, f: 0.0576373) (Load: 0.001700) (Wireload: 0.000000)

19: Node click2#midnand#A1#+, myclick2/midnand/A1 (nd21d1) (Delay = 2.3242418 r) (Incr =
0.0000000 r) (r: 0.0598848, f: 0.0576373) (Load: 0.000000) (Wireload: 0.000000)

20 (Endpoint): Node click2#midnand#ZN#-, myclick2/midnand/A1 (nd21d1) (Delay = 2.4122947 f) (Incr
= 0.0880529 f) (r: 0.0884399, f: 0.0728929) (Load: 0.002500) (Wireload: 0.000000)
```

```
Total Cycle Delay: 2.4122947
OPT-INFO: End of Period Minimisation Process (0 reduce iterations).
INFO: Computed Event Graph Period: p = 2.41229
```

TCL command display_eventgraph produces the following illustration of the critical cycle.



4.6.2 Bundled-Data Asynchronous Circuits

As mentioned previously, BD asynchronous circuits require explicit identification of their asynchronous control portions. This is to have a clear separation between the control section and the BD section, and for the tool to be able to automatically perform ASTA for the asynchronous control, followed by STA for the BD section.

Asynchronous control section must be specified as a set of Verilog Modules, *i.e.* by the declared Verilog module names, using the `set_async_modules` command. The Event Graph (RETG) will, in this way, be extracted solely for the asynchronous control section. ASTA command `report_timing –async`, automatically detects BD connectivity to the control section. It will first automatically classify the circuit as non-BD or BD, and next as a Latch Design or FF Design. If the design is BD indeed, asynchronously generated clocks will be automatically created at the control, BD boundary signals. Their timing will be automatically derived, based on the critical cycle. More specifically, their opening clock edges depend on the critical cycle timing, whereas the closing clock edges on the shortest closing edge cycle or path. Details about their timing are displayed at the end of the ASTA run, and before STA is invoked.

STA for the BD section is performed based on the asynchronously generated clocks as the effective STA clocks.

Below, we present an ASTA example with an attached BD Datapath.

```
######################################################################
set LIB mock.lib
set DESIGN mock.v
load_lef {mock.lef mock_tech.lef}
# load .lib file #
load_lib $LIB
# load Verilog netlist #
load_verilog $DESIGN
### if the design is bundled-data, then declare asynchronous region
### by defining the list of asynchronous Verilog modules
set_async_modules {click1 click2}
### if you need to inspect the asynchronous modules you have declared
### you may use list_async_modules
list_async_modules
extract_eventgraph
display_eventgraph
list_toplevel_pins
### set BD constraints
set_input_delay 0.55 toplevel/b
set_input_delay 0.55 toplevel/a
set_input_transition 0.1 -rise toplevel/b
set_input_transition 0.1 -fall toplevel/b
set_input_transition 0.1 -rise toplevel/a
set_input_transition 0.1 -fall toplevel/a
### at this point we are ready to perform ASTA Critical Cycle
Analysis, followed by Bundled-Data STA
### this is performed with report_timing -async command
report_timing -async -longest -corner 0
######################################################################
```

This is the corresponding circuit post critical cycle report, relevant to the BD section, produced using NLDM models, post-synthesis, *i.e.* with no placement data.

```
INFO: Design is FF-Based, Bundled-Data.
INFO: Timing RC Mode is Elmore, Lumped RC.
INFO: Timing Model is NLDM.
INFO: Top-Level Module is toplevel.
INFO: Top-Level Module is toplevel.
STA Progress: 3.09%
STA Progress: 6.19%
STA Progress: 8.25%
STA Progress: 10.31%
INFO: Longest Path Level = 3, Gatepin = toplevel/d
DEBUG: Worst CDCs computed slack for gatepin datapath/ff3/D: 0.499396
--------------------------------------
Dumping Longest Path to Gatepin datapath/ff3/D
1: myclick1/inv/ZN (in01d1) (Delay = 0.0000000 r) (r: 0.1827779, f: 0.0818434) (Load: 0.012000)
(Wireload: 0.000000)

2: thedatapath/ff2/CP (dfn3d1) (Delay = 0.0000000 r) (r: 0.1827779, f: 0.0818434) (Load:
0.000000) (Wireload: 0.000000)

3: thedatapath/ff2/Q (dfn3d1) (Delay = 0.4295708 f) (r: 0.1057030, f: 0.0720394) (Load: 0.004600)
(Wireload: 0.000000)

4: thedatapath/ff3/D (dfn3d1) (Delay = 1.7643231 f) (r: 0.1057030, f: 0.0720394) (Load: 0.000000)
(Wireload: 0.000000)

--------------------------------------

Data required arrival time: 2.2637186

Data arrival time: 1.7643231

Slack: (MET)     0.499396

WARNING: Resetting RC Mode to lumpedRC, as no Placement Data Exists for the Current Flow Step!

INFO: Capture clock period, active edge: 2.625055, 1.334752

INFO: Launch clock period, active edge: 2.625055, 1.334752

DEBUG: Clocks have identical periods

DEBUG: Calculated Clock Edges ATs (capture, launch): (3.959807, 1.334752)

DEBUG: Clock Distance: 2.625055

INFO: Capture clock period, active edge: 2.625055, 1.334752

INFO: Launch clock period, active edge: 2.625055, 0.000000

DEBUG: Clocks have identical periods

DEBUG: Calculated Clock Edges ATs (capture, launch): (1.334752, 0.000000)

DEBUG: Clock Distance: 1.334752

INFO: Capture clock period, active edge: 2.625055, 0.000000

INFO: Launch clock period, active edge: 2.625055, 1.334752

DEBUG: Clocks have identical periods

DEBUG: Calculated Clock Edges ATs (capture, launch): (2.625055, 1.334752)

DEBUG: Clock Distance: 1.290303

INFO: Capture clock period, active edge: 2.625055, 0.000000

INFO: Launch clock period, active edge: 2.625055, 0.000000

DEBUG: Clocks have identical periods

DEBUG: Calculated Clock Edges ATs (capture, launch): (2.625055, 0.000000)

DEBUG: Clock Distance: 2.625055

INFO: Top-Level Module is toplevel.

INFO: Top-Level Module is toplevel.
```

```
STA Progress: 3.09%

STA Progress: 6.19%

STA Progress: 8.25%

STA Progress: 10.31%

INFO: Longest Path Level = 3, Gatepin = toplevel/d

DEBUG: Worst CDCs computed slack for gatepin datapath/ff3/D: -0.835357

--------------------------------------

Dumping Longest Path to Gatepin datapath/ff3/D

1: myclick1/inv/ZN (in01d1) (Delay = 2.0759966 r) (r: 0.1827779, f: 0.0818434) (Load: 0.012000)
(Wireload: 0.000000)

2: thedatapath/ff2/CP (dfn3d1) (Delay = 2.0759966 r) (r: 0.1827779, f: 0.0818434) (Load:
0.000000) (Wireload: 0.000000)

3: thedatapath/ff2/Q (dfn3d1) (Delay = 2.5055674 f) (r: 0.1057030, f: 0.0720394) (Load: 0.004600)
(Wireload: 0.000000)

4: thedatapath/ff3/D (dfn3d1) (Delay = 3.8403197 f) (r: 0.1057030, f: 0.0720394) (Load: 0.000000)
(Wireload: 0.000000)

--------------------------------------

Data required arrival time: 3.0049629

Data arrival time: 3.8403197

Slack: (VIOLATED)        -0.835357
```

And reporting the ASTA asynchronously generated clocks, we get the following report, using TCL command `report_clocks`. As can be seen, the two clocks are non-overlapping, as the two controllers form a loop.

```
> report_clocks

Clock (0):
        Name: click1/inv/ZN
        Source: click1/inv/ZN
        Period: 2.6250551
        Async. Offset: 0.7412443
        Duty cycle:
                Uncertainty:
                        Setup: 0.0000000
                        Hold: 0.0000000
                        Rise: 0.0000000
                        Fall: 0.0000000
                Edges:
                        Rising: 1.3347523, Falling: 1.7670594
        Type: Asynchronously Generated Clock.
Clock (1):
        Name: click2/inv/ZN
        Source: click2/inv/ZN
        Period: 2.6250551
        Async. Offset: 0.7412443
        Duty cycle:
                Uncertainty:
                        Setup: 0.0000000
                        Hold: 0.0000000
                        Rise: 0.0000000
                        Fall: 0.0000000
                Edges:
                        Rising: 0.0000000, Falling: 0.4038388
        Type: Asynchronously Generated Clock.
```

In this example, `gatepin datapath/ff3/CP` is driven by asynchronous clock `click2/inv/ZN`. This may be checked using TCL command `list_gatepin_connections` on the flat gatepin name.

```
> list_gatepin_connections -predecessors datapath/ff3/CP
--------------------------------------
Gatepin datapath/ff3/CP Predecessors:
click2/inv (/ZN).
--------------------------------------
```

This corresponds to the capture clock, whereas the launch clock is `click1/inv/ZN`. The worse BD RAT is identified at gatepin `datapath/ff3/CP` and is equal to:

```
(ASTA clock period of the capture clock – FF setup time),
```

which in this case is `2.2638ns` above. The actual AT, from the launch clock with edge at `1.3347ns` is `1.764323ns`, thus the BD slack corresponds to `0.499396ns`, as shown in the report above.

This example clearly illustrates the ASTA, STA interaction. Last, but not least as the ASTA clocks are generated they have non-zero slews. This may be verified or checked by using TCL command report_delay_calculation on them:

```
> report_delay_calculation click1/inv/ZN
--------------------------------------
Gatepin: click1/inv/ZN (11, 0)
Rise Delay: 0.0000000000000000000
Fall Delay: 0.0000000000000000000
Rise Transition: 0.1827778947368421114
Fall Transition: 0.0818433926709095788
Cell Rise Arc is Negative Unate
Cell Fall Arc is Negative Unate
Related Clock(s): click1/inv/ZN (click1/inv/ZN).
HPWL: 0.0000000000000000000
Capacitance: 0.0120000000000000002
Rise Capacitance: 0.0000000000000000000
Fall Capacitance: 0.0000000000000000000
Rise RAT: inf
Fall RAT: inf
Rise Slack: inf
Fall Slack: inf
Is Constrained Flag: -128
Is Constrained Flag: asyncgenclock
False Path Type at Gatepin: 0 = -
Constant Value at Gatepin: 1'bx
--------------------------------------
```

5. Troubleshooting & Support

[TBD]

5.1 Error Messages

[TBD]

5.2 Special Considerations

[TBD]

5.3 Support

See Section 1.5.

6. Resources

`test` folder (see Figure 1) contains mockup TCL scripts to run the tool.

More TCL scripts can be provided upon request.

Appendix A: Record of Changes

[TBD]

Table 6: Record of Changes

Version Number	Date (D/M/Y)	Description of Change
V0.1	28/05/2024	First Draft of STA Engine Manual
V0.2	29/05/2024	Provide Better Setup Instructions
V0.3	07/02/2025	Introduction of new commands

Appendix B: Glossary

Table 7: Glossary

Term	Definition
Uniquified	To avoid confusion, a uniquified netlist assigns a unique name to each signal, ensuring that there are no naming conflicts within the design. This process typically involves appending a unique identifier, such as a number or a suffix, to each signal name to distinguish it from others with the same name. Uniquified netlists are commonly used in electronic design automation (EDA) tools for tasks such as simulation, synthesis, and verification, where accurate signal tracing and analysis are essential. They help prevent errors and improve the reliability and efficiency of the design process.
`SPEF`	Standard Parasitic Exchange Format (SPEF) file is a common format used in electronic design automation (EDA) to represent parasitic information extracted from a physical layout of an integrated circuit.
`SDC`	SDC (Synopsys Design Constraints) file used in digital design to specify timing constraints for electronic design automation (EDA) tools. It contains detailed instructions and constraints that guide the synthesis, place-and-route, and static timing analysis (STA) processes. These constraints define the desired timing behavior of the design and help ensure that it meets timing requirements and performance goals.
`DEF`	DEF (Design Exchange Format) file is a standard text-based or binary file format used in electronic design automation (EDA) to describe the physical layout of an integrated circuit (IC). It contains detailed information about the placement and routing of components, metal layers, vias, and other physical design elements. DEF files are typically generated by place-and-route tools during the physical implementation stage of the IC design flow.
`LEF`	LEF (Library Exchange Format) file used in electronic design automation (EDA) to describe the physical and electrical properties of standard cell libraries in integrated circuit (IC) designs. LEF files contain information about the geometric shapes, sizes, and electrical characteristics of standard cells, including transistors, logic gates, and other building blocks used in digital designs.
`Verilog or SystemVerilog Netlist`	A netlist is a textual or binary representation of an electronic circuit or digital design, detailing the interconnections between its components. It lists the various electrical nodes and the components

	connected to each node, such as resistors, transistors, capacitors, and other electronic elements.
`LIB`	LIB (Library) file is a standard format used in electronic design automation (EDA) to store information about timing and electrical characteristics of standard cells, macros, and other library elements used in integrated circuit (IC) designs. LIB files contain essential data for cell characterization, such as timing models, power consumption, and noise margins, which are necessary for accurate simulation and analysis during the design process.
`SDF`	