Fusion Compiler™ Power Analysis User Guide

Version T-2022.03-SP1, April 2022

SYNOPSYS®

Copyright and Proprietary Information Notice

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html.
All other product or company names may be trademarks of their respective owners.

Free and Open-Source Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

www.synopsys.com

Feedback

Contents

About This User Guide

The Synopsys Fusion Compiler tool provides a complete netlist-to-GDSII or netlist-to-OASIS® design solution, which combines proprietary design planning, physical synthesis, clock tree synthesis, and routing for logical and physical design implementation throughout the design flow.

This guide describes the Fusion Compiler power analysis features. For more information about the Fusion Compiler tool, see the following companion volumes:

- *Library Manager User Guide*
- *Fusion Compiler Design Planning User Guide*
- *Fusion Compiler Data Model User Guide*
- *Fusion Compiler Timing Analysis User Guide*
- *Fusion Compiler Graphical User Interface User Guide*
- *Fusion Compiler Multivoltage User Guide*

This user guide is for design engineers who use the Fusion Compiler tool to perform power analysis.

This preface includes the following sections:

New in This Release

Information about new features, enhancements, and changes, known limitations, and resolved Synopsys Technical Action Requests (STARs) is available in the Fusion Compiler Release Notes on the SolvNetPlus site.

Related Products, Publications, and Trademarks

For additional information about the Fusion Compiler tool, see the documentation on the Synopsys SolvNetPlus support site at the following address:

https://solvnetplus.synopsys.com

You might also want to see the documentation for the following related Synopsys products:

- PrimePower

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
`Courier`	Indicates syntax, such as `write_file`.
`Courier italic`	Indicates a user-defined value in syntax, such as `write_file` *`design_list`*
`Courier bold`	Indicates user input—text you type verbatim—in examples, such as `prompt> write_file top`
`Purple`	• Within an example, indicates information of special interest. • Within a command-syntax section, indicates a default, such as `include_enclosing = true | false`
[ ]	Denotes optional arguments in syntax, such as `write_file [-format fmt]`
...	Indicates that arguments can be repeated as many times as needed, such as *`pin1 pin2 ... pinN`*.
|	Indicates a choice among alternatives, such as `low | medium | high`
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Bold	Indicates a graphical user interface (GUI) element that has an action associated with it.
Edit > Copy	Indicates a path to a menu command, such as opening the **Edit** menu and choosing **Copy**.

Convention	Description
Ctrl+C	Indicates a keyboard combination, such as holding down the Ctrl key and pressing C.

Customer Support

Customer support is available through SolvNetPlus.

Accessing SolvNetPlus

The SolvNetPlus site includes a knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. The SolvNetPlus site also gives you access to a wide range of Synopsys online services including software downloads, documentation, and technical support.

To access the SolvNetPlus site, go to the following address:

https://solvnetplus.synopsys.com

If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to sign up for an account.

If you need help using the SolvNetPlus site, click REGISTRATION HELP in the top-right menu bar.

Contacting Customer Support

To contact Customer Support, go to https://solvnetplus.synopsys.com.

1

Power Modeling and Calculation

As you create a design, it moves from a high level of abstraction to its final implementation at the gate level. The Fusion Compiler tool supports power analysis and optimization throughout the design cycle, from RTL to the gate level.

This topic contains the following sections:

Power Types

The power dissipated in a circuit falls into two broad categories:

- Static power, which is power dissipated by a gate when it is inactive

  The largest percentage of static power results from source-to-drain subthreshold leakage, which is caused by reduced threshold voltages that prevent the gate from completely turning off. Static power is also dissipated when current leaks between the diffusion layers and the substrate. For this reason, static power is often called *leakage power*.

- Dynamic power, which is power dissipated when the circuit is active

  A circuit is active any time the voltage on a net changes due to some stimulus applied to the circuit. Because voltage on an input net can change without necessarily resulting in a logic transition on the output, dynamic power can be dissipated even when an output net does not change its logic state.

Dynamic power includes switching power and internal power.

- The switching power of a driving cell is the power dissipated by the charging and discharging of the load capacitance at the output of the cell. The total load capacitance at the output of a driving cell is the sum of the net and gate capacitances on the driving output.

  Because load charging and discharging are the result of the logic transitions at the output of the cell, switching power increases as the frequency of logic transitions

Feedback

increases. Therefore, the switching power of a cell is a function of both the total load capacitance at the cell output and the rate of logic transitions.

- Internal power is any power dissipated within the boundary of a cell. During switching, a circuit dissipates internal power by the charging or discharging of any existing capacitances internal to the cell. Internal power includes power dissipated by a momentary short circuit between the P and N transistors of a gate, called short-circuit power.

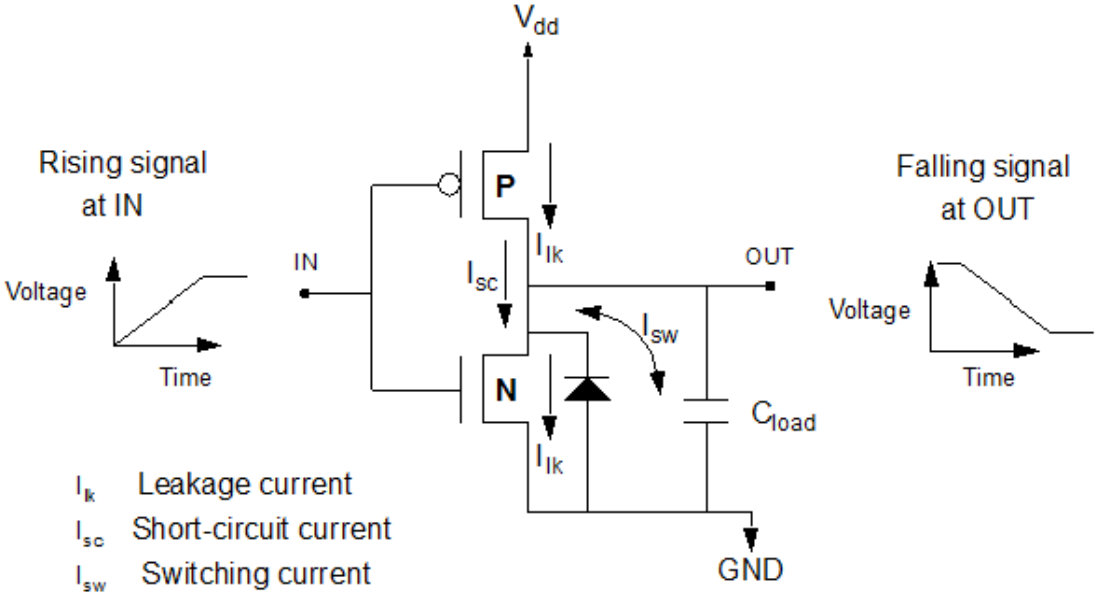
  For circuits with fast transition times, short-circuit power can be small. However, for circuits with slow transition times, short-circuit power can account for 30 percent of the total power dissipated by the gate. Short-circuit power is affected by the dimensions of the transistors and the load capacitance at the gate's output.

  In most simple library cells, internal power is due mostly to short-circuit power. For more complex cells, the charging and discharging of internal capacitance might be the dominant source of internal power.

  Library developers can model internal power by using the internal power library group. For more information about modeling internal power, see the *Library Compiler User Guide*.

Figure 1 shows a simple gate and illustrates where static and dynamic power are dissipated.

Figure 1 Components of Power Dissipation



To illustrate the cause of short-circuit power, consider the simple gate shown in Figure 1. A rising signal is applied at IN. As the signal transitions from low to high, the N type

transistor turns on and the P type transistor turns off. However, for a short time during signal transition, both the P and N type transistors can be on simultaneously. During this time, current I_{sc} flows from V_{dd} to GND, causing the dissipation of short-circuit power (P_{sc}).

Calculating Power

This topic describes how the tool performs power analysis. The tool uses power information in the specified logic library to evaluate the power of a design. For more information about modeling power in library cells, see the *Library Compiler User Guide*.

Note:

The power calculations described apply only to NLPM power calculations.

This topic includes information about library modeling and power equations:

- Calculating Static (Leakage) Power
- Calculating Dynamic Power
- Calculating Internal Power
- Calculating Power Consumption of NLDM Models
- Internal Power Calculation
- Calculating Switching Power
- Calculating Power for Multirail Cells

Calculating Static (Leakage) Power

The tool analysis computes the total leakage power of a design by summing the leakage power of the design's library cells, as follows:

$$P_{LeakageTotal} = \sum_{\forall cells(i)} P_{CellLeakage_i}$$

Where:

PLeakageTotal = Total leakage power dissipation of the design

$PCellLeakage_i$ = Leakage power dissipation of each cell i

Library developers annotate the library cells with the total leakage power dissipated by each library cell. The model can contain a default for leakage power by using the

Feedback

`default_cell_leakage_power` attribute. Alternatively, the model can provide the leakage power per cell with the `cell_leakage_power` attribute.

If the `cell_leakage_power` attribute is missing or negative, the tool assigns the value of the `default_cell_leakage_power` attribute. If both values are missing, the tool assigns a value of 0.

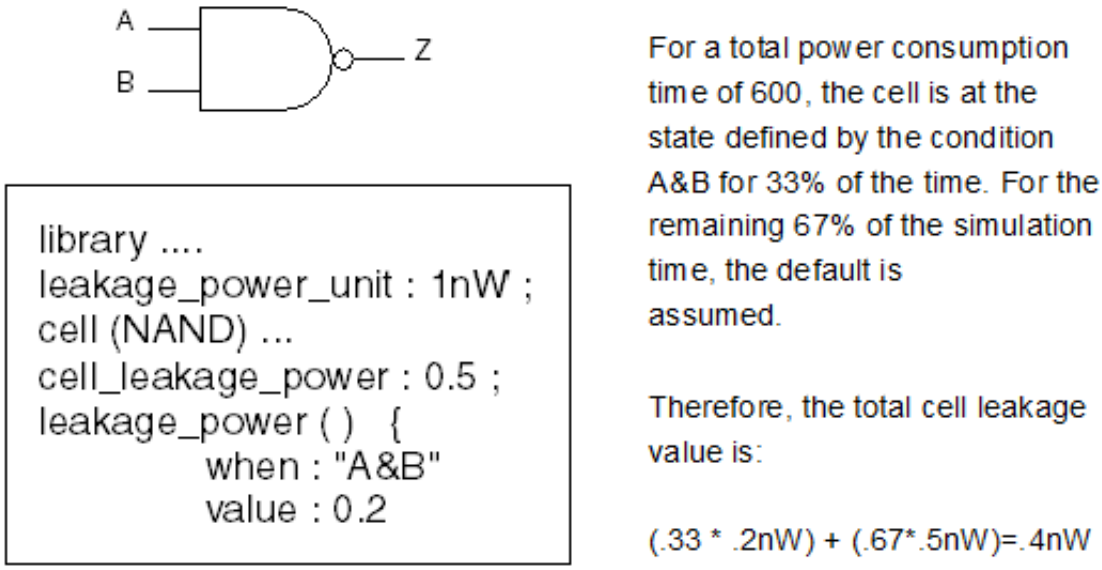
The `leakage_power` attribute models state-dependent leakage. You can also use Boolean expressions to define the conditions for different cell leakage power values.

To calculate cell leakage, the tool determines the units based on the `leakage_power_unit` attribute. It checks for the `leakage_power` attribute first. The leakage value for each state is multiplied by the percentage of the total simulation time at that state and then summed to provide the total leakage power per cell.

If the state is not defined in the `leakage_power` attribute, the value of the `cell_leakage_power` attribute is used to obtain the contribution of the leakage power at the undefined state.

Figure 2 shows the leakage power calculation performed on a NAND gate with state-dependent values.

Figure 2 Leakage Power Calculation for a NAND Gate With State Dependent Values



Feedback

Multiple Threshold Voltage Libraries

Static power dissipation has an exponential dependence on transistor threshold voltage. In order to address low-power designs, foundries offer technologies that enable multiple threshold voltages.

Each type of logic gate is available in two or more threshold voltage groups. The threshold voltage determines the speed and the leakage characteristics of the cell. Cells with low-threshold transistors switch quickly, but have higher leakage and consume more power. Cells with high-threshold transistors have lower leakage and consume less power, but switch more slowly.

For leakage power optimization, the tool supports can swap cells with different threshold voltages appropriately, based on power and timing requirements.

Calculating Dynamic Power

Dynamic power is the power dissipated when a circuit is active and is the sum of switching power and internal power:

Dynamic power = Switching power + Internal power

The tool calculates the unit for dynamic power based on the `capacitive_load_unit`, `voltage_unit`, and `time_unit` values in the library cell, as follows:

1. Find the starting value.

```
capacitive_load_unit (0.35, ff);
voltage_unit: "1V"
time_unit: "1ns";

starting value = capacitive_load_unit*voltage_unit2/time_unit
starting value = .35e-15*(1^2)/1e-9
starting value = 3.5e-7W
```

   The starting value consists of a base unit (1e-7W) and a multiplier (3.5).

2. Select an MKS base unit that converts the multiplier of the starting value found in step 1 to an integer. The MKS base unit that meets this requirement in this example is nano [1e-9]. The starting value of 3.5e-7W expressed in nW becomes 350nW.

```
converted value = 350e-9W
converted value multiplier = 350
base unit = 1e9W = 1nW
```

3. Determine the base unit multiplier by selecting a power of 10 integer (for example, 1, 10, 100, ...) closest in magnitude to the converted value multiplier found in step 2.

```
converted value multiplier = 350 (from step 2)
base unit multiplier = 100
```

Feedback

4. Combine the base unit multiplier obtained in step 3 and the base unit obtained in step 2 to obtain the dynamic power unit.

```
base unit = 1nW (from step 2)
base unit multiplier = 100 (from step 3)
dynamic power unit = (100) 1nW = 100nW
```

   In this example, each cell's dynamic power calculated by the tool is multiplied by 100nW.

Dynamic Power Unit Derivation

The unit for switching power and the values in the `internal_power` table is a derived unit. It is derived from the following function:

`(capacitive_load_unit * voltage_unit`^{2}`)/time_unit`

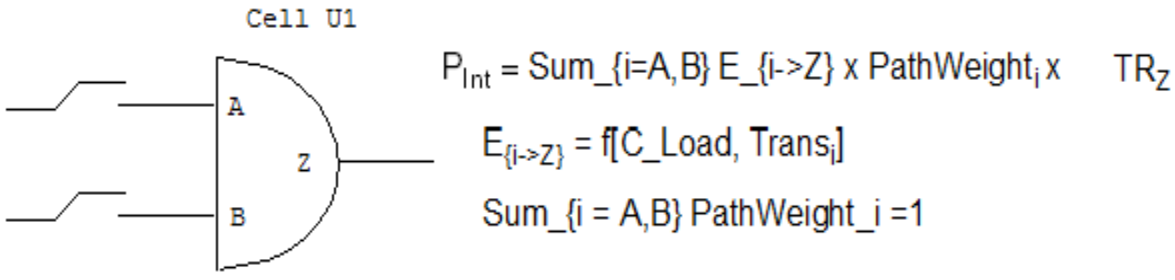
The function's parameters are defined in the library. The result is scaled to the closest MKS unit: micro, nano, femto, or pico. This dynamic power unit scaling effect must be taken into account by library developers when generating energy values for the internal power table.

Calculating Internal Power

When computing internal power, power analysis uses information characterized in the logic library. The `internal_power` library group and its associated attributes and groups define scaling factors and a default for internal power. Library developers can use the internal power table to model internal power on any pin of the library cell.

A cell's internal power is the sum of the internal power of all of the cell's inputs and outputs as modeled in the logic library. Figure 3 shows how the tool calculates the internal power for a simple combinational cell with path-dependent internal power modeling.

Figure 3 Internal Power Model (Combinational)



P_{Int}	Total internal power of the cell$_E$

Feedback

E_z	Internal energy for output Z as a function of input transitions, output load, and voltage
TR_z	Toggle rate of output pin Z, transitions per second
TR_i	Toggle rate of input pin i, transitions per second
$Trans_i$	Transition time of input i
$WeightAvg_{(Trans)}$	Weighted average transition time for output Z

The tool calculates the input path weights based on the input toggle rates, transition times, and functionality of the cell. The tool supports NLDM (table-based) models.

See Also

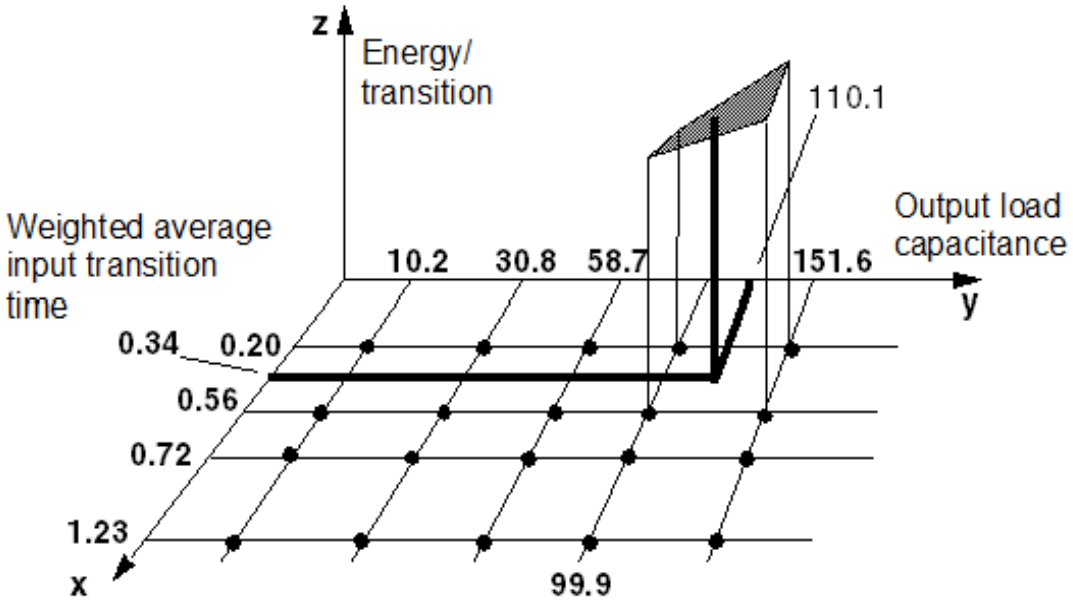
- Calculating Power Consumption of NLDM Models
- Calculating Dynamic Power

Calculating Power Consumption of NLDM Models

To compute the internal power consumption of NLDM models, the Fusion Compiler tool uses the weighted average transition time as an index to the internal power associated with the output pin. The second index to the two-dimensional lookup table is the output load capacitance, as shown in Figure 4.

Feedback

Figure 4 Two-Dimensional Lookup Table



For cells in which output pins have equal or opposite logic values, the tool can use a three-dimensional lookup table. The tool indexes the three-dimensional table by using input transition time and both output capacitances of the equal (or opposite) pins. For example, the three-dimensional table is well suited to describing the flip-flop, which has Q and Q-bar outputs of opposite value.

The `internal_power` library group supports a one-, two-, or three-dimensional lookup table. Table 1 shows the types of lookup tables, whether they are appropriate to inputs or outputs, and how they are indexed.

Table 1 Lookup Tables

Lookup table	Defined on	Indexed by
One-dimensional	Input	Input transition
	Output	Output load capacitance
Two-dimensional	Output	Input transition and output load capacitance
Three-dimensional	Output	Input transition and output load capacitances of two outputs that have equal or opposite logic values

Feedback

For more information about modeling internal power and library modeling syntax and methodology, see the *Library Compiler User Guide*.

For different operating conditions, the table model supports scaling factors for the internal power calculation, as follows:

- `k_process_internal_power`
- `k_temp_internal_power`
- `k_volt_internal_power`

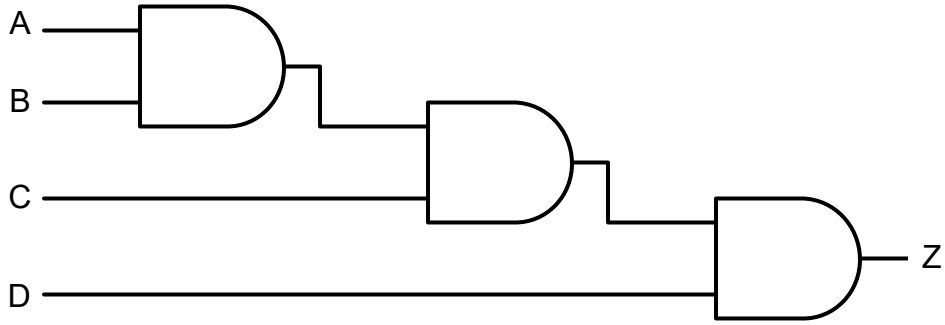
However, these factors cannot accurately model the nonlinear effects of the operating conditions. Most vendors generate separate table-based libraries for different operating conditions.

Internal Power Calculation

Internal power is often calculated using the state- and path-dependent power values of a cell, along with the signal transitions.

Cells consume different amounts of internal power, depending on the input pin transitions or the state of the cell.

To demonstrate path-dependent internal power, consider the following simple library cell, which has several levels of logic and a number of input pins:



Inputs A and D can each cause an output transition at Z. However, input D affects only one level of logic, whereas input A affects three levels. An output transition at Z consumes more internal power when it results from an input transition at A than when it results from an input transition at D. You can specify multiple lookup tables for outputs to capture the dependency on the input transitions.

The Fusion Compiler tool chooses the appropriate path-dependent internal power table for an output by checking the `related_pin` attribute in the library. Based on the percentage of toggles on each input pin, the total power due to transitions on the output pin is calculated

by using the correct table or equation for each related pin and applying the percentage contribution per input pin.

An example of a cell with state-dependent internal power is a RAM cell. It consumes a different amount of internal power depending on whether it is in read or write mode. You can specify separate tables or equations depending on the state or mode of the cell.

If the toggle rate information is provided for each state defined in the power model, the tool accesses the appropriate information. If only the input or output toggle information is available, the tool averages the tables for the different states to compute the internal power of the cell.

The internal power for a rising transition might be different from the internal power for a falling transition. A library model can optionally provide separate power values for rising and falling transitions.

Calculating Switching Power

The tool analysis calculates switching power as follows:

$$P_c = \frac{V_{dd}^2}{2} \sum_{\forall nets(i)} \left(C_{Load_i} \times TR_i \right)$$

Where:

P_c Switching power of the design

TR_i Toggle rate of net i, transitions per second

V_{dd} Supply voltage

C_{Loadi} is the total capacitive load of net i, including parasitic capacitance, gate capacitance, and drain capacitance of all the pins connected to the net i.

The tool obtains C_{Loadi} from the wire load model for the net and from the logic library information for the gates connected to the net. You can also back-annotate capacitance information after physical design.

See Also

- Calculating Dynamic Power
- Calculating Internal Power

Calculating Power for Multirail Cells

The Fusion Compiler tool supports the power analysis of libraries which contain cells with multiple rails for which power values are defined per voltage rail.

For multivoltage cells which contain separate power tables for each power level, the tool determines the internal and leakage power contribution for each power rail and sums it to report the total power consumption.

For more information about defining per-rail power tables, see the *Library Compiler User Guide*.

2

Preparing for Power Analysis

To perform power analysis, the tool uses information about the switching activity of your design. You can use simulation tools such as VCS to generate switching activity information and save it in a Switching Activity Interchange Format (SAIF) file. After you generate the SAIF file, you can annotate your design using the switching activity provided by the SAIF file.

For more information, see the following topics:

- About SAIF Files
- Generating SAIF Files

About SAIF Files

Accurate power calculations depend on accurate switching activity data. SAIF (Switching Activity Interchange Format) is an ASCII format that facilitates the interchange of switching information between Synopsys tools. This data is generated using RTL simulation or gate-level simulation and is stored in a SAIF file. You should use the SAIF file to annotate switching activity information on the design objects before you perform power optimization and analysis.

For more information about SAIF file contents, see the *IEEE 1801 Standard for Design and Verification of Low Power Integrated Circuits*, Annex I.

Use the `read_saif` command to read the SAIF file and the `write_saif` command to write out the SAIF file. For more information, see the command man pages.

Early in the design cycle, you can use RTL simulation to determine the high-level switching and power characteristics of the design. Later in the design cycle, you can use gate-level simulation to get more detailed switching data to annotate your design. The detailed switching data increases the accuracy of the power optimization and power analysis.

Table 2 summarizes the methods of generating SAIF files.

Feedback

Table 2 Comparing Methods of Capturing Switching Activity

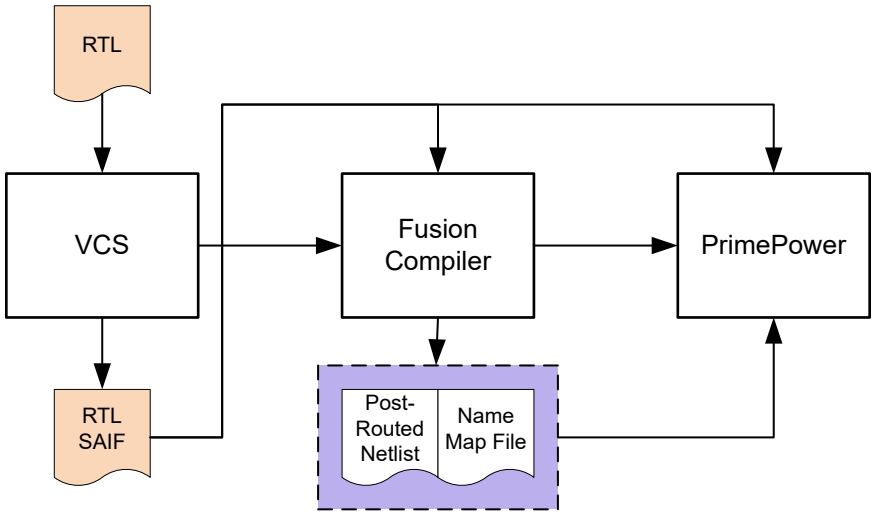
Simulation	Captured	Not captured	Trade-offs
RTL	Synthesis-invariant elements	1. Internal nodes 2. Correlation of non-synthesis-invariant elements 3. Glitching 4. State and path dependencies	Fast runtime at expense of some accuracy
Zero-delay and unit-delay gate-level	1. Synthesis-invariant elements 2. Internal nodes 3. Correlation 4. State dependencies 5. Some path dependencies	1. Some path dependencies 2. Glitching	More accurate than RTL simulation, but significantly higher runtime
Full-timing gate-level	1. All elements of design 2. Correlation 3. State and path dependencies	Correlation between primary inputs	Highest accuracy, but runtime might be very long

Using SAIF Files With Other Tools

Figure 5 illustrates SAIF file generation and its use with other Synopsys tools. Using the RTL SAIF file generated by VCS, the tool implements power optimization and generates a post-routed netlist and a SAIF name map file that associates the names in the routed netlist to the original RTL objects.

Feedback

Figure 5 Using the SAIF File With Other Synopsys Tools



See Also

- Name Mapping and Tracking

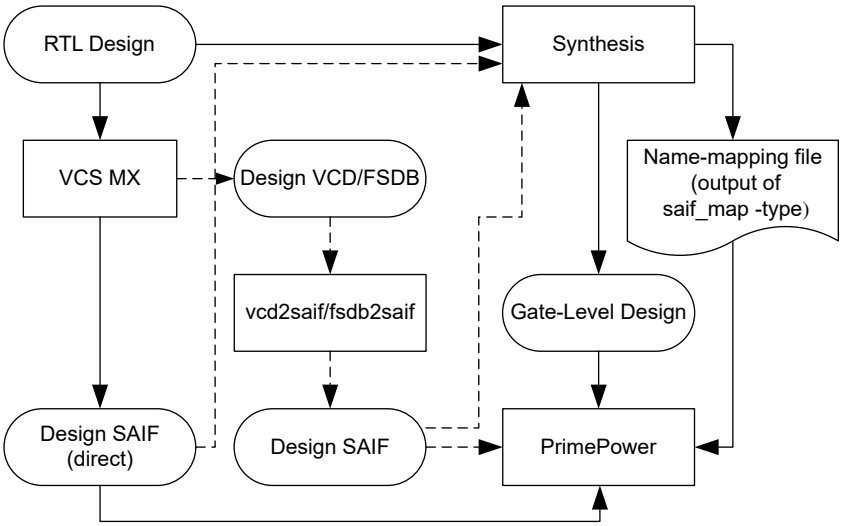
Generating SAIF Files

You can generate a SAIF file from either RTL simulation or gate-level simulation using the Synopsys VCS tool. VCS supports Verilog, SystemVerilog, and VHDL formats.

Figure 6 shows two ways of generating a SAIF file. The solid lines indicate the suggested SAIF flow while the dotted lines indicate the alternative SAIF flow using other Synopsys tools.

Feedback

Figure 6 SAIF File Generation and Usage With Synopsys Tools



You can generate the SAIF file using the following:

- Simulation
- VCD files
- FSDB output files

After you generate a SAIF file, you can load the file into the Fusion Compiler tool and generate a mapping file for all the name changes of the nodes. Then, you supply the name-mapping file and the synthesized gate-level netlist to the PrimePower tool to perform averaged power analysis.

Topics covered in this section:

- Generating SAIF Files From Simulation
- Generating SAIF Files From VCD Files
- Generating SAIF Files From FSDB Output Files
- Verilog Switching Activity Examples
- VHDL Switching Activity Example

Generating SAIF Files From Simulation

VCS MX can generate the SAIF file directly from simulation. This direct SAIF file is smaller than VCD or FSDB files. Your input design for simulation can be an RTL or gate-level design. The design can be in Verilog, SystemVerilog, VHDL, or mixed HDL formats. When

Feedback

your design is in Verilog or SystemVerilog formats, you must specify system tasks to VCS MX using toggle commands. If your design is in VHDL format, use the power command as described in Generating SAIF Files From VHDL Simulation.

For more information about the supported formats and mixed language formats, see the *VCS MX User Guide*.

When generating the SAIF file during simulation, use the default monitoring policy (see the *VCS MX User Guide* for more information). This monitoring captures the switching activity of only the synthesis-invariant objects such as ports, tristate cells, black box cells, flip-flops, latches, retention registers, and hierarchical cells other than clock-gating cells. Integrated clock-gating cells and latch-based isolation cells are synthesis-dependent objects and are therefore not captured.

If the library forward SAIF file contains details of state and path dependencies, the backward SAIF file generated also contains these details. For more information, see Capturing State- and Path-Dependent Switching Activity.

The steps to generate SAIF files from simulation are discussed in the following sections:

- Generating SAIF Files From SystemVerilog or Verilog Simulations
- Generating SAIF Files From VHDL Simulation
- VCS MX Toggle Commands

Generating SAIF Files From SystemVerilog or Verilog Simulations

Using VCS MX, you can generate SAIF files from both RTL and gate-level Verilog designs. When your design is in Verilog format, you must specify system tasks to VCS MX. These system tasks are also known as toggle commands. The system tasks specify the module for which switching activity is to be recorded and reported in the SAIF file. They also control the toggle monitoring during simulation.

For details about the toggle commands, see VCS MX Toggle Commands.

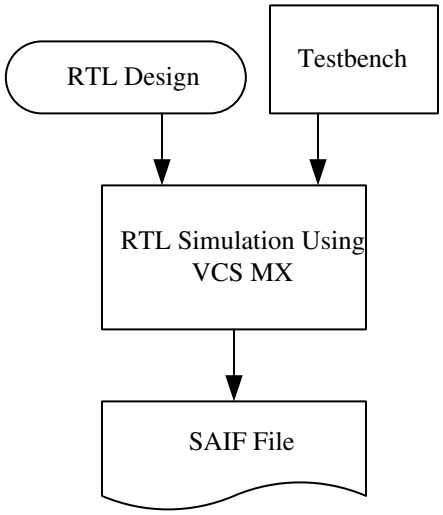
See Also

- VCS MX Toggle Commands

Generating SAIF Files From RTL Simulation

Figure 7 shows the methodology to capture switching activity using RTL simulation. RTL simulation captures the switching activity of primary inputs, primary outputs, and other synthesis-invariant elements.

Figure 7 RTL Simulation Using VCS MX



To capture the switching activity using RTL simulation, specify the appropriate testbench and run the simulation.

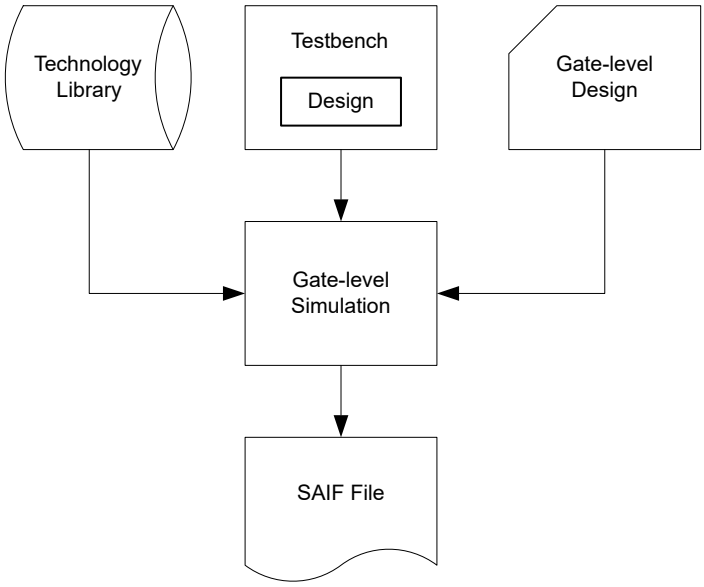
The SAIF file contains the switching activity information of the synthesis-invariant elements in your design. To use the SAIF file for synthesis in the Fusion Compiler tool, annotate the switching activity, as described in Annotating the Switching Activity Using RTL SAIF Files.

Generating SAIF Files From Gate-Level Simulation

Figure 8 illustrates the methodology to capture switching activity using gate-level simulation. Gate-level simulation captures switching activity of pins, ports, and nets in your design.

Feedback

Figure 8 Gate-Level Simulation Using VCS MX



To capture switching activity using gate-level simulation, specify the appropriate toggle commands in the testbench and run the simulation.

The SAIF file contains information about the switching activity of the pins, ports, and nets in your design. It can represent the pin-switching activity, based on rise and fall values, if your logic library has separate rise and fall power tables.

To use the SAIF file for synthesis in the Fusion Compiler tool, annotate the switching activity as described in Annotating Switching Activity Using Gate-Level SAIF Files.

Generating SAIF Files From VHDL Simulation

You can use VCS MX to generate SAIF files from RTL or gate-level simulation of VHDL designs. The methodology to generate the SAIF file is similar to the methodology used for Verilog designs. However, you cannot use the toggle commands to specify the system tasks to the simulator.

For RTL-level VHDL files, variables are not supported by the simulator for monitoring. However, VHDL constructs such as generates, enumerated types, records, and arrays of arrays are supported by VCS MX for simulation.

The use model to generate a SAIF file from VHDL simulation consists of using the `power` command at the VCS MX command line interface, simv. The syntax of the `power` command is as follows:

```
power
 -enable
 -disable
 -reset
```

Feedback

```
-report file_name synthesis_time_unit scope
-rtl_saif file_name
[test_bench_path_name]
-gate_level on| off | rtl_on
region_signal_variable
```

- The `-enable` option enables the monitoring of the switching activity.
- The `-disable` option disables the monitoring of the switching activity.
- The `-reset` option resets the toggle counter.
- The `-report` option reports the switching activity to an output SAIF file.
- The `-rtl_saif` option reads the RTL forward-SAIF file.
- You can use `on`, `off`, or `rtl_on` with the `-gate_level` option. Table 3 summarizes the monitoring policy for VHDL simulation.

Table 3 Monitoring Policy for VHDL Simulation

Monitoring policy	Ports	Signals	Variables
on	Yes	Yes	No
off	No	No	No
rtl_on	Yes	Yes	No

- You can specify either the hierarchical path to the signal name or the toggle region and its children to be considered for monitoring.

```
power test1
power -enable
run 10000
power -disable
power -report vhdl.saif 1e-09 test
quit
```

System Task List for SAIF File Generation From VHDL Simulation

The following example script shows a task list that you specify to the simulator to generate a SAIF file. The design name in the example is test1. You can either specify each of these commands at the VCS MX command prompt or run the file that contains these commands.

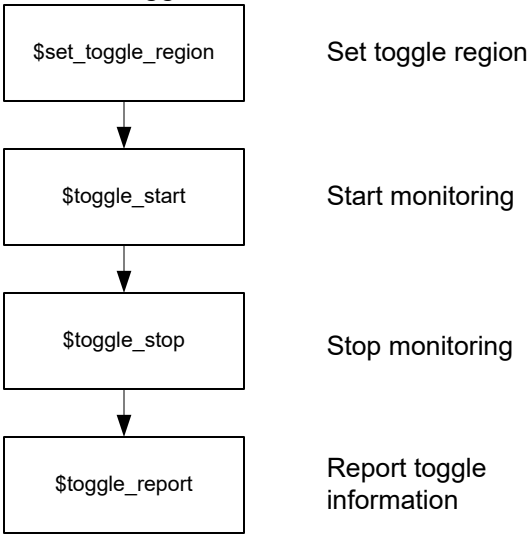
VCS MX Toggle Commands

To generate the SAIF file from RTL or gate-level Verilog of SystemVerilog, use toggle commands to specify system tasks to VCS MX. Using the toggle commands, you can

specify the subblock for toggle counting and define specific periods for toggle counting during simulation. You can also control the start and stop of toggle counting.

Figure 9 presents an overview of the toggle commands in your testbench file. Each toggle command starts with the $ symbol. For simplicity, the figure does not show optional commands.

Figure 9 Toggle Command Flow



The system tasks that you specify to VCS MX using the toggle commands are:

1. Define the toggle region

   The `$set_toggle_region` command specifies the module instance for which the simulator records the switching activity in the generated SAIF file. The syntax of this command is as follows:

   ```
   $set_toggle_region(instance [, instance]);
   ```

   When you explicitly mention one or more module instances as the toggle region, the simulator registers these objects and monitors them during simulation.

   Note:

   For gate-level simulation, if the logic library cell pins have rise and fall power values, their switching activity is monitored and reported for rise and fall separately.

2. Begin toggle monitoring

Feedback

Use the `$toggle_start` command to instruct the simulator to start monitoring the switching activity. The syntax of this command is as follows:

```
$toggle_start();
```

During simulation, the tool starts monitoring the switching activity of the module instances that are defined in the toggle region.

3. End toggle monitoring

   Use the `$toggle_stop` command to instruct the simulator to stop monitoring the switching activity.

4. Report toggle information in an output file

   Use the `$toggle_report` command to write monitored gate and net switching activity to an output file. You can invoke `$toggle_report` any number of times using different parameters. For more details and examples of SAIF files, see Verilog Switching Activity Examples.

   The syntax for the `$toggle_report` command is as follows:

   ```
   $toggle_report (filename,[synthesis_time_unit],instance_name_string);
   ```

   The values for the options and parameters are as follows:

   - *filename*

     This is the name of the switching activity output file.

   - *synthesis_time_unit*

     This optional parameter is the time unit of your synthesis library, in seconds. For example, if the time unit in your synthesis library is 10 picoseconds, specify 1.0e-11.The `$toggle_report` command uses this number to convert simulation time units to synthesis time units. Fusion Compiler obtains the simulation time unit from simulation. If you don’t specify the synthesis time unit parameter, the default is 1 ns (1.0e-9).

   - *instance_name_string*

     This required parameter is the full instance path name of the block from the top of your simulation environment down to the name of the block instance to be reported.

```
$toggle_reset();
```

Use the `$toggle_reset` command only after you have written out the previous results with the `$toggle_report` command.

Feedback

See Also

- Capturing State- and Path-Dependent Switching Activity
- Overriding Default Registration of Internal Nets

Example

```
$toggle_report ("file.saif", 1.0e-11, "test.DUT");
```

In this example, the file written out is file.saif, the synthesis time unit is 10 picoseconds, and the name of the monitored instance is test.DUT. The output file format is SAIF, which is the default.

Resetting the Toggle Counter

Use the `$toggle_reset` command to set the toggle counter to 0 for all the nets in the current toggle region. This command starts a new toggle monitoring period in a simulation session.

For example, when using `$toggle_start`, `$toggle_stop`, or `$toggle_reset` with the `$toggle_report`, you can create SAIF output files for specific periods during simulation. The syntax of this command is as follows:

Capturing State- and Path-Dependent Switching Activity

By default, the tool estimates the state- and path-dependent power information that is required for power calculations. However, if you want to obtain this information through simulation, you can use the Library Compiler tool's `lib2saif` command before simulation. In this case, for a given a logic library, you can run the utility to obtain a library SAIF file that contains the directives for generating state- and path-dependent switching information. This file is called the library forward SAIF file. This file becomes an input to gate-level simulation.

The library forward SAIF file contains information from the logic library about cells that have state and path dependencies. It can have rise and fall information if the library has separate rise and fall power tables.

To read the library forward SAIF file into the simulator, use the `$read_lib_saif` command. This command registers the state- and path-dependent information for monitoring during simulation.

The syntax of the `$read_lib_saif` command is as follows:

```
$read_lib_saif(input_file);
```

For gate-level simulation, you must use the `$read_lib_saif` command to register state- and path-dependent cells and, by default, all internal nets in the design. The command registers state-dependent and path-dependent cells by reading the library forward SAIF file. In addition, you must also set the toggle region for monitoring. If you do not use the

`$read_lib_saif` command, the simulator registers all internal nets for monitoring by default.

You can use the `$read_lib_saif` command as often as you require during simulation; however, you must use this command before defining the toggle region using the `$set_toggle_region` command. When you define the toggle region, the `$set_toggle_region` command checks for the presence or absence of a previous `$read_lib_saif` command and registers internal nets accordingly.

Example 1 shows an example of a library forward SAIF file generated by the `lib2saif` command.

Example 1 File Generated by the Library Compiler Tool lib2saif Command

```
(SAIFILE
(SAIFVERSION "2.0" "lib")
(DIRECTION "forward")
(DESIGN )
(DATE ...)
(VENDOR "Synopsys, Inc")
(PROGRAM_NAME "... lib2saif") (VERSION ...) (DIVIDER / ) (LIBRARY
 "example"
  (MODULE "AND2"
    (PORT
      (Z
        (IOPATH A IOPATH B)
      )
    )
  )
  (MODULE "DFF1"
    (LEAKAGE
     (COND Q
      COND !Q
      COND_DEFAULT)
    )
  )
  (MODULE "EXOR3"
    (PORT
      (Z
        (COND ((!B * !A) | (B * A)) RISE_FALL (IOPATH C)
         COND ((!B * A) | (B * !A)) RISE_FALL (IOPATH C)
         COND ((!C * !A) | (C * A)) RISE_FALL (IOPATH B)
         COND ((!C * A) | (C * !A)) RISE_FALL (IOPATH B)
         COND ((!C * !B) | (C * B)) RISE_FALL (IOPATH A)
         COND ((!C * B) | (C * !B)) RISE_FALL (IOPATH A)
         COND_DEFAULT RISE_FALL (IOPATH A IOPATH B IOPATH C))
      )
    )
  )
  (MODULE "MUX21"
    (PORT
      (Z
```

Feedback

```
          (COND (B * !A) RISE_FALL (IOPATH S)
           COND (!B * A) RISE_FALL (IOPATH S)
           COND_DEFAULT RISE_FALL (IOPATH A IOPATH B IOPATH S))
        )
      )
      (LEAKAGE
       (COND (B * S * A)
        COND (!B * S * A)
        COND (!B * !S * A)
        COND (!A * S * B)
        COND (!A * !S * B)
        COND_DEFAULT)
      )
    )
    (MODULE "NAND2"
      (PORT
        (Z
          (IOPATH A IOPATH B)
        )
      )
    )
    (MODULE "OR2"
      (PORT
        (Z
          (IOPATH A IOPATH B)
        )
      )
    )
....
   (MODULE "iopad6"
      (PORT
        (PAD
          (COND !TS RISE_FALL (IOPATH DI)
           COND_DEFAULT RISE_FALL)
        )
        (DI
          (COND TS RISE_FALL
           COND_DEFAULT RISE_FALL)
        )
        (DO
          (IOPATH PAD IOPATH_DEFAULT)
        )
      )
    )
)
```

Overriding Default Registration of Internal Nets

After you have run the `$read_lib_saif` command in the testbench, you can override the default net monitoring behavior using the `$set_gate_level_monitoring` command. This command turns on or turns off the registration of internal nets.

The following is the syntax of the `$set_gate_level_monitoring` command:

```
$set_gate_level_monitoring ("on" | "rtl_on", "mda" | "sv");
```

- `"on"`

  This string explicitly registers all internal nets for simulation. Thus, simulation monitors any internal net in the region defined by the `$set_toggle_region` command.

- `"rtl_on"`

  The registers in the toggle region are monitored and the nets in the toggle region are not monitored during simulation.

- `"mda"`

  Use this argument for Verilog memories and multidimensional arrays.

- `"sv"`

  Use this argument for SystemVerilog data objects.

The `$set_gate_level_monitoring` command is optional. If you use it, you must do so before invoking the `$set_toggle_region` command.

Generating SAIF Files From VCD Files

You can generate SAIF files from VCD files. To generate a SAIF from a VCD file generated by the VCS tool, use the `vcd2saif` utility. Follow these steps to generate the SAIF file and to annotate the switching activity:

1. Run the simulation to generate VCD file.
2. Use the `vcd2saif` utility to convert the VCD file to a SAIF file.
3. Annotate the switching activity within the SAIF file as described in Annotating the Switching Activity Using RTL SAIF Files.

The disadvantage of using this method is that VCD files can be very large, especially for gate-level simulation, requiring more time for processing. Also, the SAIF file generated by the `vcd2saif` utility lacks state-dependent and path-dependent information.

Converting a VCD File to a SAIF File

The `vcd2saif` utility converts the RTL or gate-level VCD file generated by VCS into a SAIF file. This utility has limited capability when the VCD is generated from the SystemVerilog simulation as described in Generating SAIF Files From VCD Files.

Feedback

The `vcd2saif` utility is platform-specific and is located in *install_dir*/$ARCH/syn/bin. The $ARCH environment variable represents the specific platform (architecture) of your installation, such as linux.

You can use compressed VCD files (.Z) and gzipped VCD files (.gz). In addition, for VPD files, you can use the utility located at $VCS_HOME/bin/vpd2vcd, and for FSDB files, you can use the utility located at $SYNOPSYS/bin/fsdb2vcd.

The `vcd2saif` utility does not support state-dependent and path-dependent switching activity. For information about each option, use the `vcd2saif -help` command.

Limited SystemVerilog Support in the vcd2saif Utility

The `vcd2saif` utility supports only a limited set of SystemVerilog constructs for VCD files generated from SystemVerilog simulation. Table 4 lists the SystemVerilog constructs supported by the `vcd2saif` utility.

Table 4 SystemVerilog Constructs Supported by the vcd2saif Utility

char	int	shortint	longint	bit	byte	logic
shortreal	void	enum	typedef	struct	union	arrays (packed and unpacked)

Generating SAIF Files From FSDB Output Files

To generate a SAIF file from an FSDB file, use the `fsdb2saif` utility.

For more information about the FSDB utilities, see the *Verdi3 and Siloti Command Reference Manual*. After generating the SAIF file, annotate the switching activity from the SAIF file as described in Annotating the Switching Activity Using RTL SAIF Files.

Verilog Switching Activity Examples

This section contains examples of the RTL and gate-level descriptions with Verilog-generated switching activity data.

Topics in this section:

- RTL Example
- Gate-Level Example

Feedback

RTL Example

This Verilog RTL example includes the following elements:

- RTL design description
- RTL testbench
- SAIF output file from simulation

Verilog Design Description

Example 2 shows the description for a state machine called test.

Example 2 RTL Verilog Design Description

```
`timescale 1 ns / 1 ns
module test (data, clock, reset, d_out);

input [1:0] data;
input clock;
input reset;
output d_out;

wire d_out;

wire [1:0] NEXT_STATE;
reg [1:0] PRES_STATE;

parameter s0 = 2'b00;
parameter s5 = 2'b01;
parameter s10 = 2'b10;
parameter s15 = 2'b11;

function [2:0] fsm;
 input [1:0] fsm_data;
 input [1:0] fsm_PRES_STATE;
 reg fsm_d_out;
 reg [1:0] fsm_NEXT_STATE;

begin
  case (fsm_PRES_STATE)
  s0: //state = s0
  begin
    if (fsm_data == 2'b10)
    begin
       fsm_d_out = 1'b0;
       fsm_NEXT_STATE = s10;
    end
    else if (fsm_data == 2'b01)
       //....
  end
```

```
    s5: //state = s5
    begin
      // ...
    end
    s10: //state = s10
    begin
      // ...
    end

    s15: //state 15
    begin
      // ...
    end
    endcase

    fsm = {fsm_d_out, fsm_NEXT_STATE};
  end

  endfunction

  assign {d_out, NEXT_STATE} = fsm(data, PRES_STATE);

  always @(posedge clock)
  begin
    if (reset == 1'b1)
    begin
        PRES_STATE = s0;
    end
    else
    begin
        PRES_STATE= NEXT_STATE;
    end
  end
  endmodule
```

RTL Testbench

The Verilog testbench in Example 3 simulates the design test described in Example 2. The testbench instantiates the design test as U1.

Example 3 RTL Testbench

```
  `timescale 1 ns / 1 ns

  module stimulus;

  reg clock;
  reg [1:0] data;
  reg reset;
  wire d_out;
  test U1 (data,clock, reset, d_out);

  always
```

```
  begin
    #10 clock = ~clock;
  end

initial
begin
$set_toggle_region(stimulus.U1);
$toggle_start();
clock = 1'b0;
data = 2'b00;
reset = 1'b1;
#50 reset = 0;
#25 data = 3; #20 data = 0;
#20 data = 1; #20 data = 2;
// ...
$toggle_stop();
$toggle_report("my_rtl_saif", 1.0e-12, "stimulus");
#80 $finish;
end
```

RTL SAIF File

The RTL SAIF file is the output of RTL simulation and contains information about the switching activity of synthesis-invariant elements. The `$toggle_report` command creates this file.

Example 4 is a SAIF file created for the RTL Verilog description that is also shown in Example 2 and for the testbench shown in Example 3.

Example 4 RTL SAIF File

```
/** There is no explicit set_gate_level_monitoring command, **/
/** and the default behavior is to monitor internal nets  **/
(SAIFILE
(SAIFVERSION "2.0")
(DIRECTION "backward")
(DESIGN )
(DATE "Fri Feb  6 14:21:20 2015")
(VENDOR "Synopsys, Inc")
(PROGRAM_NAME "VCS I-2014.03-SP1")
(VERSION "1.0")
(DIVIDER / )
(TIMESCALE 1 ps)
(DURATION 135000.00)
(INSTANCE stimulus
   (INSTANCE U1
      (NET
         (data\[1\]
            (T0 115000) (T1 20000) (TX 0)
            (TC 2) (IG 0)
         )
         (data\[0\]
            (T0 95000) (T1 40000) (TX 0)
```

```
            (TC 3) (IG 0)
         )
         (clock
            (T0 70000) (T1 65000) (TX 0)
            (TC 13) (IG 0)
         )
         (reset
            (T0 85000) (T1 50000) (TX 0)
            (TC 1) (IG 0)
         )
         (d_out
            (T0 0) (T1 0) (TX 135000)
            (TC 0) (IG 0)
         )
         (NEXT_STATE\[1\]
            (T0 0) (T1 0) (TX 135000)
            (TC 0) (IG 0)
         )
         (NEXT_STATE\[0\]
            (T0 0) (T1 0) (TX 135000)
            (TC 0) (IG 0)
         )
      )
   )
)
)
```

Understanding the SAIF File

Table 5 summarizes the definitions for SAIF file terms.

Table 5 Definitions of SAIF File Terminology

T0	Duration of time found in logic 0 state.
T1	Duration of time found in logic 1 state.
TX	Duration of time found in unknown “X” state.
TC	The sum of the rise (0-to-1) and fall (1-to-0) transitions that are captured during monitoring.
IG	Number of 0 - X - 0 and 1 - X - 1 glitches captured during monitoring.
RISE	Rise transitions in a given state.
FALL	Fall transitions in a given state.

Feedback

Duration refers to the time span between `$toggle_start` and `$toggle_stop` commands in the testbench during simulation. During this time span, ports, pins, and nets are monitored for toggle activity. For more information on the terminology of the SAIF file, see the *IEEE 1801 Standard, Annex J*.

Gate-Level Example

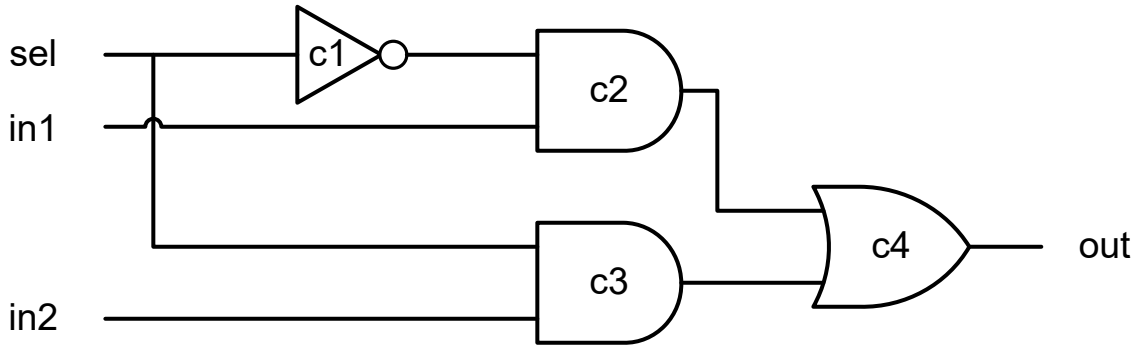
This Verilog gate-level example illustrates the following elements:

- Verilog cell description and schematic
- Verilog testbench
- SAIF output file from simulation

Gate-Level Verilog Module

Figure 10 shows the schematic for a simple multiplexer.

Figure 10 Schematic of Multiplexer Circuit: MUX21



Example 5 is the Verilog module that describes the MUX21 design.

Example 5 Verilog Module of Multiplexer Circuit: MUX21

```
/*`timescale 10ps/ 1ps
*/
module MUX21(out,d1,d2,sel);
input d1, d2, sel;
output out;
    IV c1(.Z(sel_),.A(sel));
    AN2 c2(.Z(d1m),.A(d1),.B(sel_));
    AN2 c3(.Z(d2m),.A(d2),.B(sel));
    OR2 c4(.Z(out),.A(d1m),.B(d2m));
endmodule
```

Verilog Testbench

The Verilog testbench in Example 6 tests the MUX21 design by simulating it and monitoring the signals.

Feedback

Example 6 Verilog Testbench for MUX21

```
            /* Begin test.v */
    `timescale 1ns/ 10ps
    module top;
         reg in1, in2, sel;
         parameter hazrate = 0.99;
         parameter haztime = 0.23;

         MUX21 m1(out,in1,in2,sel);

         initial
         begin
               // start monitoring
               $monitor($time,,,"in1=%b in2=%b sel=%b
               out=%b",in1,in2,sel,out);
               // read SAIF file of state or path dependent information
               $read_lib_saif (cell.saif);
               // define the monitoring scope
               $set_toggle_region (m1);
               $toggle_start;

               // test first data line passing 0
               sel = 0;
               in1 = 0;
               in2 = 0;
               // test first data line passing 1
               #10 in1 = 1;

               #10 sel = 1;

               // test second data line passing 1
               #10 in2 = 1;
               $toggle_stop;
               $toggle_report("my_1st", 1.0e-9,"top.m1", hazrate, haztime);

               // exit simulation
               $finish(2);
    end
    endmodule
```

The `$set_toggle_region` command sets the monitoring scope in module m1 (the testbench instantiation of MUX21). All subsequent toggle commands affect only registered design objects and designs instantiated in registered objects. Thus, under m1, simulation monitors internal nets and state- and path-dependent cells (in this simple example, however, there are no subdesigns in m1).

The testbench example invokes `$toggle_report` command before exiting the simulation. Make sure that you declare any parameters you use for `$toggle_report` command in your testbench. These parameters appear at the top of the testbench in Example 6.

Feedback

Gate-Level SAIF File

Example 7 shows a SAIF file generated from gate-level simulation of MUX21.

Example 7 $toggle_report Output File in SAIF

```
(SAIFILE
(SAIFVERSION "2.0")
(DIRECTION "backward")
(DESIGN )
(DATE "Fri Oct  6 18:58:58 2000")
(VENDOR "Synopsys, Inc")
(PROGRAM_NAME "VCS-MX ...")
(VERSION "3.3")
(DIVIDER / )
(TIMESCALE 1 ns)
(DURATION 99999.00)
(INSTANCE tb
   (INSTANCE dut
      (NET
         (n12159
             (T0 99529) (T1 470) (TX 1)
             (TC 46) (IG 0)
         )
         (n12480
             (T0 0) (T1 99998) (TX 0)
             (TC 0) (IG 0)
         )
         (n12117
             (T0 61) (T1 99938) (TX 0)
             (TC 26) (IG 0)
         )
      )
      (INSTANCE U12053
         (PORT
            (Z
               (T0 10) (T1 99989) (TX 0)
               (COND A (RISE)
                  (IOPATH B (TC 0) (IG 0)
                  )
               COND A (FALL)
                  (IOPATH B (TC 0) (IG 0)
                  )
               COND B (RISE)
                  (IOPATH A (TC 0) (IG 0)
                  )
               COND B (FALL)
                  (IOPATH A (TC 1) (IG 0)
                  )
               COND_DEFAULT (TC 1) (IG 0)
               )
            )
         )
```

```
        )
      )
)
)
```

VHDL Switching Activity Example

This VHDL RTL example includes the following elements:

- RTL design description
- RTL testbench
- SAIF output file from simulation

VHDL Design Description

Example 8 shows the description for a design called ABC.

Example 8 RTL VHDL Design Description

```
library ieee;
use ieee.std_logic_1164.all;
entity ABC is
architecture beh of ABC is
 signal clk: std_logic := '0';
begin
 clk <= not clk after 5 ns;
end beh;
```

RTL Testbench

The RTL testbench in Example 9 simulates the design test described in Example 8. The testbench instantiates the design ABC as ABC_ins.

Example 9 RTL Testbench

```
library ieee;
use ieee.std_logic_1164.all;
entity test is
end entity
architecture testbench of test is
 component ABC is
 end component;
begin
 ABC_ins: ABC;
end testbench;
```

Feedback

RTL SAIF File

This RTL SAIF file is the output of RTL simulation and contains information about the switching activity of synthesis-invariant elements. The `power -report` command creates this file.

Example 10 is a SAIF file for the RTL VHDL description that is shown in Example 8.

Example 10 RTL SAIF File

```
/** There is no explicit set_gate_level_monitoring command, **/
/** and the default behavior is to monitor internal nets  **/
(SAIFILE
(SAIFVERSION "2.0")
(DIRECTION "backward")
(DESIGN )
(DATE "Tue May  5 05:56:35 2009")
(VENDOR "Synopsys, Inc")
(PROGRAM_NAME "VCS-Scirocco-MX ...")
(VERSION "1.0")
(DIVIDER / )
(TIMESCALE 1 ns)
(DURATION 10000.00)
(INSTANCE TEST
   (INSTANCE DUMMY_INS
      (NET
         (CLK
            (T0 5000) (T1 5000) (TX 0)
            (TC 1999) (IG 0)
         )
      )
   )
)
)
```

3

Switching Activity

Switching activity accounts for a large percentage of dynamic power consumption, which is a large percentage of the power consumption of a design. The tool models switching activity based on the following information:

- Static probability

  This is the fraction of time that a signal is at the logic 1 state. For example, a static probability of 0.8 means that the signal is in a logic 1 state 80 percent of the time and at a logic 0 20 percent of the time.

- Toggle rate

  The rate at which a signal changes from 0 to 1 and from 1 to 0, in number of transitions per time unit.

Topics covered in this section are:

Power Analysis Flows

Performing power analysis depends on the how the switching activity is defined for the design. The Fusion Compiler tool supports the following flows:

- RTL SAIF file flow
- Gate-level SAIF file flow
- No SAIF file flow

Feedback

RTL SAIF File Flow

When you use a SAIF file from an RTL simulation, the Fusion Compiler tool creates a gate-level name-mapping file using the `saif_map` command.

The RTL SAIF flow is the most accurate when comparing results to simulation power analysis. However, this flow is sensitive to how well the vectors from the RTL SAIF file are mapped to the gate-level netlist.

See Also

- Annotating the Switching Activity Using RTL SAIF Files

Gate-Level SAIF File Flow

If you use a gate-level SAIF file, the tool does not need to use a mapping file because all the objects in the SAIF file are already mapped correctly to the netlist objects. While this flow is easier to use because there is no need for a mapping file, it is less accurate when comparing power because it relies on the activity being maintained and propagated during synthesis and optimization.

See Also

- Annotating Switching Activity Using Gate-Level SAIF Files

No SAIF File Flow

If you do not use a SAIF file, the tool estimates switching activity based on default switching activity values and propagated switching activity from known nets. If you do not specify any activity from a SAIF file, from the `set_switching_activity` command, or from SDC constraints such as the `create_clock` or `set_case_analysis` commands, the tool propagates the default switching activity values.

To explicitly set activity constraints, use the `set_switching_activity` command.

The following is an example of a tool script:

```
# read design
...
saif_map -start
create_clock clk -period 10
set_case_analysis 0 ScanEnable
# set up corner/mode/scenario
create_scenario ...
set_switching_activity -static_probability 0 \
   -toggle_rate 0 [get_ports Slee]
...
```

Feedback

When you set explicit constraints, you should set the same equivalent commands in the PrimePower tool or export the SAIF activity from the Fusion Compiler tool. This ensures consistent power analysis results between the tools.

See Also

- Design Objects Without Annotated Switching Activity
- Annotating Switching Activity Using the set_switching_activity Command
- Inferring Switching Activity

Name Mapping and Tracking

To correctly annotate activity from SAIF files onto objects in the design, the tool must accurately locate the netlist object that corresponds to the SAIF object. If the SAIF file was generated from the current design, then the object names match exactly. However, if the SAIF was generated from an earlier version of the design, for example, before optimization, then the tool must understand how the object might have changed.

The following types of changes can occur to objects:

- Renaming
- Cloning
- Removal
- Pin swapping
- Cell remapping

These types of changes are tracked by the Fusion Compiler tool and saved in a name mapping database that can be written to a name mapping file to pass to downstream tools.

Topics covered in this section:

- Starting Name Tracking
- Modifying the Name Mapping Database
- Retrieving Name Mappings
- Removing Manually-Added Name Mappings
- Removing All Name Mapping
- Reporting Manually-Added Name Mappings
- Writing Name-Mapping Information for PrimePower

Feedback

Starting Name Tracking

SAIF name tracking starts when you do *either* of the following:

- Run the `saif_map -start` command just one time during the flow
- Import an existing name mapping file using the `saif_map -read_map` command

After you run either of these commands, the tool tracks name changes to all objects in the design. You should run these commands before any netlist changes are made.

The name mapping database is persistent so you can continue to track names changes in different sessions without saving or restoring the database.

Modifying the Name Mapping Database

You can modify the name mapping database by creating, adding, changing, and removing mappings.

Topics covered in this section:

- Replacing Name Mappings
- Adding Name Mappings
- Changing a Name Mapping

Replacing Name Mappings

To define a mapping, use the `saif_map -set_name` command, which replaces any existing mapping for an object in the database. If the object shares the same name mapping with other objects, the other objects are also updated.

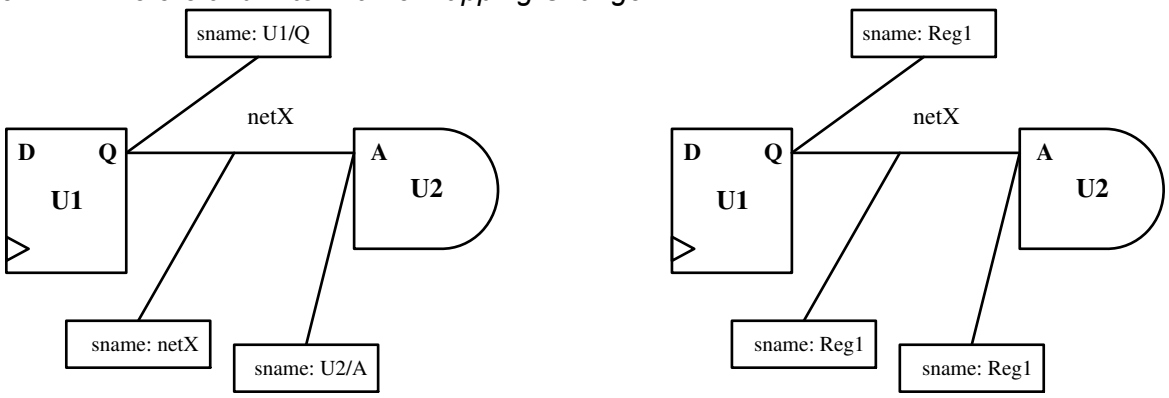
For example,

```
fc_shell> saif_map -set_name Reg1 [get_nets netX]
```

Figure 11 shows the changed names of the objects with the same name mapping as netX.

Feedback

Figure 11 Before and After Name Mapping Change



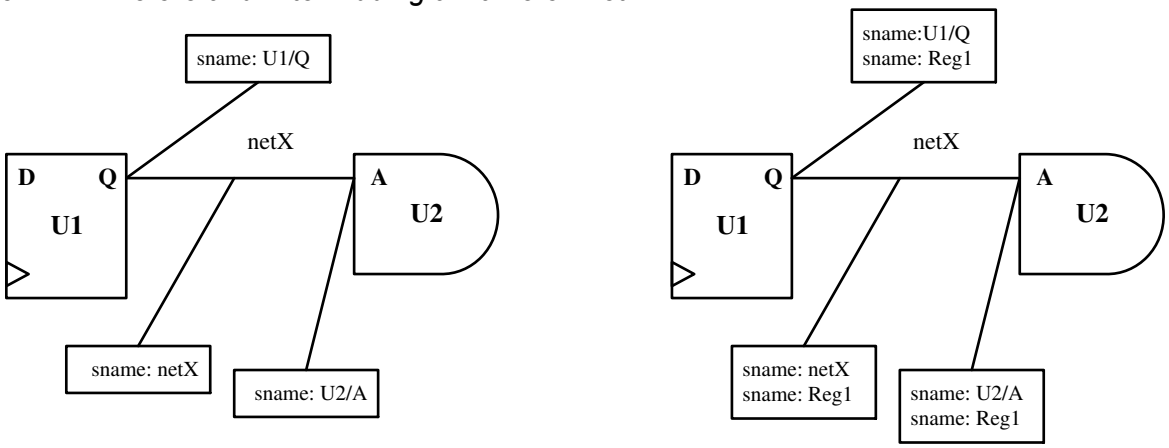
Adding Name Mappings

Use the `saif_map -add_name` command to add a name mapping to an existing entry in the database. The `-add_name` option adds the specified name to the specified object in the database. For example,

```
fc_shell> saif_map -add_name Reg1 [get_nets netX]
```

Figure 12 shows the added name Reg1 on the netX object.

Figure 12 Before and After Adding a Name on netX



When you add a name to a SAIF object, any existing mapping is maintained and additional mappings accumulate.

Changing a Name Mapping

To process a SAIF name before storing it to the name mapping database, use the `saif_map -change_name` command. The `-change_name` option specifies that a SAIF

name should be interpreted as a different name when processing the SAIF file. For example,

```
fc_shell> saif_map -change_name Reg1 netX
```

All instances of Reg1 are interpreted as netX; however,no existing mapping is changed in the name mapping database. If multiple `saif_map -change_name` commands are specified for the same SAIF name, only the last one applies. For example,

```
fc_shell> saif_map -change_name AAA BBB
fc_shell> saif_map -change_name AAA CCC
```

Only the mapping rule from AAA to CCC applies.

Retrieving Name Mappings

To find out the current name mapping for a design object, use the `saif_map -get_saif_names` command. For each of the specified objects, the command returns the set of SAIF names that map to the object. The reported SAIF names include both rules set by the `-change_name` option and name mapping information.

To find which design object matches to a SAIF name, use the `-get_object_names` option.

Removing Manually-Added Name Mappings

To remove a name mapping rule that was applied by using the `saif_map -set_name` or `-add_name` options, use the `saif_map -remove_name` command.

The `-remove_name` option works only on the name mapping database. Rules that are specified using the `-change_name` option cannot be removed this way because they do not change the database. To remove rules set by the `-change_name` option, use the `saif_map -reset` command. To apply new rules, reissue the `saif_map -change_name` command.

The `saif_map -reset` command reverts the name mapping database to the last state as defined by the last `saif_map -read_map` command or the last `saif_map -start` command.

Removing All Name Mapping

The `saif_map -reset` command only resets user manual mappings. On the other hand, the `saif_map -stop` command removes the SAIF map database.

To create a new SAIF map database, use the following command:

```
saif_map -start
```

Feedback

The SAIF map database is based on RTL or gate-level names.

To continue with the flow, use the `compile_fusion` command.

To stop the SAIF map database, use the following command:

```
saif_map -stop
```

Note:

To create a totally new different SAIF map database, use the `saif_map -start` command.

Reporting Manually-Added Name Mappings

To report manually added name-mapping information and rules added by using the `-change_name` option, use the `saif_map -report` command.

Writing Name-Mapping Information for PrimePower

The PrimePower tool uses the `set_rtl_to_gate_map` command to map the names of RTL to gate-level objects.

Use the `-type ptpx` or `-type primepower` option of the `set_rtl_to_gate_map` command to write the name-mapping information.

The `-essential` option (recommended) writes only the essential points in the PrimePower mapping file including the following objects:

- All primary input ports that existed at the time of executing the `saif_map -start` command
- All register output pins that existed at the time of executing the `saif_map -start` command
- All macro output pins (black boxes and memories)
- All preexisting RTL integrated clock-gating (ICG) cells
- All memory inputs pins
- All hard macros inputs pins

Annotating Switching Activity

Switching activity is required for accurate power calculations. This section explains the different types of switching activity information and illustrates how you can annotate switching activity on gate-level design objects.

Feedback

This chapter contains the following sections:

Types of Switching Activity to Annotate

The power of a design depends on the switching activity of the nets and cell pins. The switching activity is used for power calculation during power analysis and optimization.

The following types of switching activity can be annotated on design objects:

- Simple switching activity on design nets, ports, and cell pins. Simple switching activity consists of the static probability and the toggle rate. The static probability is the fraction of the time that the object is at logic 1. The toggle rate is the rate at which the design object switches between logic 0 and logic 1.
- State-dependent toggle rates on input pins of leaf cells. The internal power characterization of an input pin of a library cell can be state dependent. The input pins of instances of such cells can be annotated with state dependent toggle rates.
- State-dependent and path-dependent toggle rates on output pins of leaf cells. The internal power characterization of output pins can be state dependent and path dependent. Output pins of cells with state- and path-dependent characterization can be annotated with state- and path-dependent toggle rates.
- State-dependent static probability on leaf cells. Cell leakage power can be characterized using state dependent leakage power tables. Such cells can be annotated with state-dependent static probability.

Feedback

Annotating the Switching Activity Using RTL SAIF Files

Optimal power analysis and optimization results occur when switching activities reported in the RTL SAIF file are accurately associated with the correct design objects in the gate-level netlist. For this to occur, the RTL names must map correctly to their gate-level counterparts.

During synthesis, mapping inaccuracies can occur that can affect your annotation. To ensure proper name mapping and annotation of RTL SAIF files,

1. Before analyzing HDL files, set `hdlin.naming.upf_compatible` to `true`.

   This application variable enables an HDL elaboration using port, net, and cell object names similar to SAIF names.

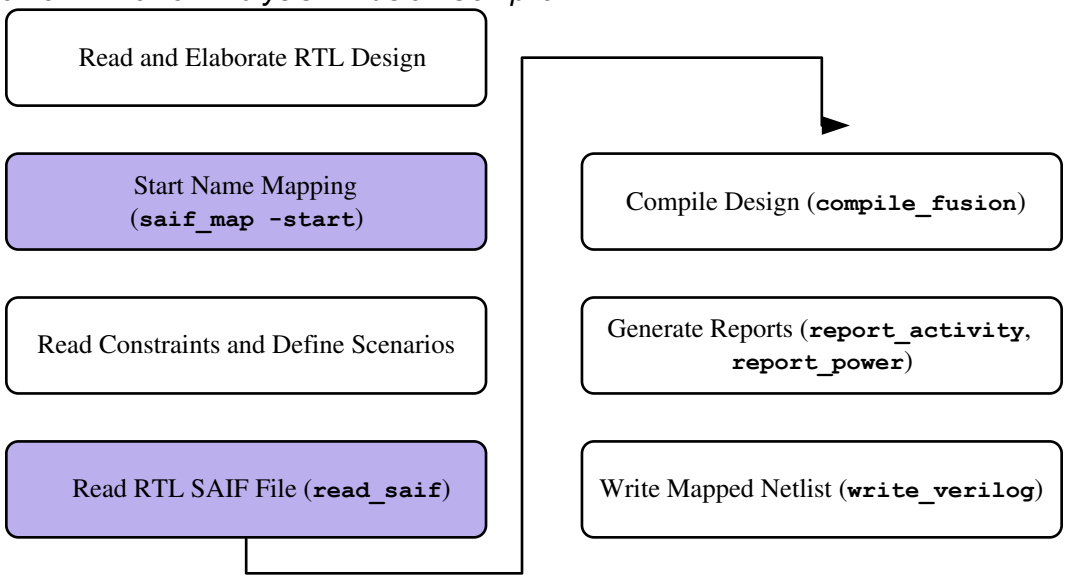
2. After reading and elaborating your design, use the `saif_map -start` command.

   This creates a name-mapping database during synthesis that the tool then uses for power analysis and optimization.

3. Before compiling, use the `read_saif` command to read and annotate RTL SAIF files.

Figure 13 shows the steps for annotating your design using an RTL SAIF file.

Figure 13 Power Analysis in Fusion Compiler



The following is an example of annotation using an RTL SAIF file in the Fusion Compiler tool:

```
# read and elaborate the design
# read the RTL SAIF file from simulation
read_saif -strip_path tu/gut RTL.saif
saif_map -start
compile_fusion
# Optional step - write out the name-mapping file
saif_map -write_map gate_level.map
# Write out the netlist
write_verilog gate_netlist.v
```

Integrating With the PrimePower Tool

The PrimePower tool requires accurate RTL-to-gate name-mapping correspondence to perform accurate power analysis. Use the Fusion Compiler tool to generate the name-mapping files that the PrimePower tool uses for mapping RTL names to gate-level names.

After the `read_saif` command, use the `saif_map -write_map -type primepower` command to generate a name-mapping file that can be read directly into the PrimePower tool.

You can reduce the number of objects in the mapping file by using the `-essential` option. This option is valid only when used with one of the `-type primepower` or `-type ptpx` options. The `-essential` option ensures that all mappings necessary for the power analysis tool are preserved.

The following example illustrates using an RTL SAIF file to ensure optimal power analysis during synthesis and to save the essential name mapping for later use by the PrimePower tool.

Example 11 Annotating Switching Activity Using an RTL SAIF File

```
read_verilog rtl_design.v
saif_map -start
read_saif ../sim/rtl.saif -strip_path tb/dut
report_activity -driver
compile_fusion
report_activity -driver
write_verilog -hierarchy all -output mapped_design.v
saif_map -write_map saifmap.pp.tcl -type primepower -essential
```

See Also

- Correlating Power Analysis Results With PrimePower

Feedback

Annotating Switching Activity Using Gate-Level SAIF Files

To annotate switching activity using gate-level SAIF files, use the `read_saif` command. Because the SAIF file should match the netlist objects, you do not need a mapping file. However, you should still enable name tracking using the `saif_map -start` command if you intend to read the gate.saif file at the end of the flow or in PrimePower.

The following example shows how to enable name tracking:

```
read_verilog gate_design.v
link_block
saif_map -start
read_saif ../sim/gate.saif -strip tb/dut
report_activity
```

If the flow starts from the RTL and you run the `saif_map -start` command before the `compile_fusion` command, you cannot read a gate-level SAIF file correctly because the `read_saif` command uses the SAIF map database by default. However, the SAIF map database is based on RTL names.

Therefore, use the `-ignore_name_mapping` option with the `read_saif` command as follows:

```
analyze
elaborate
saif_map -start
read_saif ../sim/rtl.saif
compile_fusion -to initial_opto
write_verilog initial_opto.v
read_saif gate_initial_opto.saif -ignore_name_mapping
```

Reading SAIF Files Using the read_saif Command

Use the `read_saif` command to read a SAIF file and annotate switching activity onto the design.

For example,

```
fc_shell> read_saif myfile.saif -path T1/DUT/U1
```

In this example, the `read_saif` command annotates the information in the input file named `myfile.saif` onto the current gate-level design, `T1/DUT/U1`. The `-path` option specifies the hierarchy to start applying the SAIF information.

The input file specified can be a text file or a compressed gzip file with a .gzip extension. For example,

```
fc_shell> read_saif myfile.gzip -path T1/DUT/U1
```

Feedback

A SAIF file is usually generated in the HDL simulation flow, where a simulation testbench instantiates the design being simulated and provides simulation vectors. The generated SAIF file contains the switching activity information organized in a hierarchical fashion, where the hierarchy of the SAIF file reflects the hierarchy of the simulation testbench. If a design is instantiated in the testbench (tb) as the instance i, then the SAIF file contains the switching activity information for the design under the hierarchy tb/i. In this case, specify the `tb/i` path to the `-strip_path` option when reading the SAIF file, as follows:

```
fc_shell> read_saif des.saif -strip_path tb/i
```

This strips the `tb/i` from all hierarchical instances in the SAIF file. If you specify an invalid path, the SAIF file is read incorrectly, which results in incorrect switching information. An error message is printed if none of the information stored in the SAIF file is read by the `read_saif` command.

When reading the SAIF file, the `report_lib` command gives the time units specified in a logic library. The `report_power` command gives the synthesis library time units used during power calculations.

For information about the command options, see the `read_saif` command man page.

Merging SAIF Files With the read_saif Command

The Fusion Compiler tool can read and merge multiple SAIF files to annotate switching activity on the design. If you read multiple SAIF files using a single invocation of the `read_saif` command, you can either specify a weight and scaling factor for each SAIF file or specify to use the highest toggle rate.

State-dependent and path-dependent (SDPD) information from different SAIF files is verified during merging to determine if the SDPD information is valid with respect to the library SDPD information. If part of the SDPD data for a cell is not valid, all SDPD data for that cell is dropped. If a SAIF file has missing SDPD information for a cell, the SDPD from other SAIF files are still merged.

Specifying SAIF File Weights and Scaling Ratios

The weight and scaling factor are floating point values. Specifying a weight for a SAIF file affects the toggle rate and the probability for objects in that SAIF file.

To determine the final activity, the tool uses the following equation to calculate the static probability (Pr):

$$Pr = \frac{\sum_{n=1}^{\infty}(pr_n * weight_n)}{W}$$

The probabilities for an object from different SAIF files are multiplied by the weight specified for each SAIF file and then added. The result is divided by W where W is the normalization variable.

$$W = 1 \; or \; W = \sum_{n=1}^{\infty} weight_n$$

The value of W depends on the `-normalize` option. If the value is `false`, W =1. Otherwise, W is the sum of the weights specified for each SAIF file.

When an object is found in multiple SAIF files, the toggle rate is multiplied by the weight and scaling ratio from each SAIF file, then summed over all files and divided by W.

$$Tr = \frac{\sum_{n=1}^{\infty}(tr_n * scale_n * weight_n)}{W}$$

Any object that occurs in at least one SAIF file receives a new activity value using these calculations. Objects that do not appear in any SAIF file are untouched and retain their original values.

Merging Using the Highest Toggle Rate

Alternatively, you can also specify that the tool should select the activity with the highest toggle rate from all of the SAIF files in the argument list. For example:

```
fc_shell> read_saif -use_highest_toggle_rate \
   {{test1.saif} {test2.saif}}
```

You must provide more than one SAIF file with the `-use_highest_toggle_rate` option. This option cannot be used with the `-weight` or `-normalize` options.

Merging Partial SAIF Files

A partial SAIF file contains activity for some, but not all objects. A complete SAIF file contains activity for all objects. When the tool merges partial SAIF files, it propagates the switching activity before the activity is merged.

For example, if you have three partial SAIF files named A, B, and C, the tool does the following:

1. SAIF file A is loaded and propagated, the activity is saved and then reset.
2. SAIF file B is loaded and propagated, then merged with the saved A activity, saved, and reset.
3. SAIF file C is loaded and propagated, then merged with the saved A and B activity.

Feedback

When the tool merges SAIF files, any switching activity present on the design is reset before a SAIF file is read and propagated. This ensures that your SAIF file is independent of the merge result. Therefore, any activity set by the `set_switching_activity` command is deleted during merging.

Merging SDPD activity information between different SAIF files is optional. You can disable this by using the `-exclude_sdpd` option with the `read_saif` command.

Example of Merging SAIF Files

For example, consider the following simulations that produce two different SAIF files:

- Mode1 uses a clock frequency of 500 MHz and the generated SAIF file is named sim1.saif.
- Mode2 uses a clock frequency of 1GHz and the generated SAIF file is named sim2.saif.

In addition, you have the following requirements:

- The block runs for 70% of the time in mode1 and 30% of the time in mode2.
- The simulation frequency is 2 GHz.

Issue the following command to achieve the objective:

```
fc_shell> read_saif -strip_path tb/dut \
          {sim1.saif -weight 0.7 -scaling_ratio 4.0} \
          {sim2.saif -weight 0.3 -scaling_ratio 2.0}
```

Annotating Switching Activity Using the set_switching_activity Command

The `set_switching_activity` command annotates switching activity on design objects such as pins, ports, nets, and cells. The types of activity that you can annotate include state- and path-dependent toggle rates and state-dependent static probabilities. Table 6 describes some of the options for this command.

Table 6 Options for the set_switching_activity Command

Option	Description
`-static_probability`	Floating point number between 0.0 and 1.0. Static probability is the fraction of time that the signal is at logic 1.

Feedback

Table 6 Options for the set_switching_activity Command (Continued)

Option	Description
`-toggle_rate`	Floating point number. Toggle rate is the number of low-to-high or high-to-low transitions made by the signal during a period of time. This value is different from the toggle rate used for modeling switching activity. The actual toggle rate value set in the objects depends on the usage of the `-base_clock` or `-period` option.
`-base_clock`	Specifies a clock to which the toggle rate value is related to. This clock is referred to as the object's related clock. When this option is used, the value of toggle rate specified by the `-toggle_rate` option is divided by the clock period. This option and the `-period` option are mutually exclusive.
`-period`	Specifies the time period for which the number of transitions given in the `toggle_rate_value` occur. The time units for this value are specified in the main logic library. If the `-period` option is not specified, a default `period_value` of `1.0` is assumed. This option and the `-base_clock` option are mutually exclusive.
`-state_condition`	Use this to annotate state-dependent toggle rates on pins or state-dependent static probabilities on cells. State-dependent toggle rates can be annotated only if the library is characterized with state-dependent power tables for internal power for the pins of the library cell. Similarly, state-dependent static probabilities can be annotated only if the library is characterized with state-dependent power tables for leakage power for the library cells.
`-path_sources`	Specifies path-dependent toggle rates.

Examples Using the set_switching_activity Command

The following example specifies that net `net1` is at logic `1` for 20 percent of the time, and transitions between the logic values `0` and `1` are at an average of 10 times in 1000 time units, that is, a toggle rate of 10/1000, that is 0.01.

```
fc_shell> set_switching_activity [get_nets net1] \
        -static_probability 0.2 -toggle_rate 10 -period 1000
```

The time unit used for the toggle rate is the time unit defined in the target library. The `-period` option is optional and defaults to `1`, when not specified.

Use the `-base_clock *` option to set the toggle rate based on the related clock of the object.

Feedback

The following example uses the `set_switching_activity` command to set the toggle rate 30/100, that is 0.3, as the period of the related clock is 100.

```
create_clock -period 100 [get_port clock]
set_switching_activity -toggle_rate 30 -static_probability 0.5 [get_pin
 data_reg/Q] -base_clock *
```

The following example shows how to use the `-state_condition` option to annotate the state-dependent toggle rates on pins. It specifies that the pin `ff1/Q` toggles 0.01 times when the pin D is at logic `1`, and 0.03 times when the `D` pin is at logic `0`:

```
fc_shell> set_switching_activity [get_pins ff1/Q] -toggle_rate 0.01 \
        -state_condition "D"
fc_shell> set_switching_activity [get_pins ff1/Q] -toggle_rate 0.03 \
-state_condition "!D"
```

The following example specifies that the `and1/Y` pin toggles 0.02 times because of a toggle on the input pin `A`, but never toggles because of a toggle on the `B` pin.

```
fc_shell> set_switching_activity [get_pins and1/Y] -toggle_rate 0.02 \
       -path_sources "A"
fc_shell> set_switching_activity [get_pins and1/Y] -toggle_rate 0.00 \
        -path_sources "B"
```

State- and path-dependent toggle rates are specified using the `-state_condition` and `-path_sources` options. State-dependent static probabilities can be annotated using the `-state_condition` option.

The following example specifies that the `AND1` cell is at the `A & B` state for 10 percent of the time, at the `A & !B` state for 70 percent of the time, and at the `!A` state for 20 percent of the time.

```
fc_shell> set_switching_activity [get_cells AND1] \
        -static_probability 0.1 -state_condition "A & B"
fc_shell> set_switching_activity [get_cells AND1] \
        -static_probability 0.7 -state_condition "A & !B"
fc_shell> set_switching_activity [get_cells AND1] \
        -static_probability 0.2 -state_condition "!A"
```

When you use the `set_switching_activity` command to annotate switching activity on all inputs, this includes the clock inputs as well. This results in overriding the switching activity on the clock inputs.

To avoid overriding switching activity on clock inputs, specify all inputs except the clock inputs, as shown in the following example:

```
fc_shell> set_switching_activity [remove_from_collection \
        [all_inputs] clk] \
        -static_probability sp_value -toggle_rate tr_value \
        -period period_value
```

For a complete list of options, see the `set_switching_activity` command man page.

Specifying Switching Probability for Supply Nets

The leakage power of a block is scaled based on the switching probability of its supply nets, which represents the fraction of time a supply net is in the *on* state. You can specify the switching probability for one or more supply nets by using the `set_supply_net_probability -static_probability` command. By default, the switching probability is applied to the current scenario and the corresponding mode and corner. To specify modes, corners, or scenarios in which to apply the setting, use the `-modes`, `-corners`, or `-scenarios` option. To get the switching probability of a supply net, use the `get_supply_net_probability` command, and to reset the value, use the `reset_supply_net_probability` command.

By default, the tool propagates supply net activity through power switches and determines the static probability of the switched supply net based on the UPF power switch constraint. For example, consider the following UPF power switch constraint:

```
create_power_switch my_switch \
   -output_supply_port {vout VDDS} \
   -input_supply_port {vin VDD} \
   -control_port {ms_sel ctrl1} \
   -control_port {ms_ctrl ctrl2} \
   -on_state {on vin {ms_ctrl && !ms_sel}}
```

The tool derives the static probability of the supply net named VDDS, which is connected to the output of the power switch, based on the probability of the power switch being *on*. This is derived based on the following:

- The Boolean function specified with the `-on_state` option, which is ms_ctrl && ! ms_sel, and the switching activity (static probability) of the nets connected to the corresponding control ports, which are nets named ctrl1 and ctrl2.
- The switching probability of the supply net connected to the input supply port specified with the `-on_state` option, which is the supply net named VDD.

The following application options control whether dynamic and leakage power are scaled based on the supply switching activity:

- The `power.scale_dynamic_power_at_power_off` option controls whether dynamic power is scaled. The default is `false` (no scaling).
- The `power.scale_leakage_power_at_power_off` option controls whether leakage power is scaled. The default is `true` (scaling is performed).

Feedback

Fully Versus Partially Annotating the Design

For the highest accuracy of power analysis, annotate all the elements in your design. To annotate all design elements, you must use gate-level simulation to monitor all the nodes of the design.

Using gate-level simulation, you can perform the following activities:

- Capture state- and path-dependent switching activity
- Capture switching activity that considers glitching (full-timing gate-level simulation only)

After layout, you can increase accuracy further by annotating wire loads with more accurate net capacitance values. However, if the design layout is performed at the foundry, you might not have access to the post-layout information.

If you annotate some design elements, the tool uses an internal zero-delay simulation to propagate switching activity through nonannotated nets in your design. The tool uses internal simulation anytime it encounters nonannotated nets during power analysis.

During switching activity propagation, the Fusion Compiler tool tracks which design elements are user-annotated with the `set_switching_activity` command and which are not. In calculating power, the tool does not overwrite user-annotated switching activity with propagated switching activity.

Power analysis and optimization require that you annotate at least the following:

- Primary inputs
- Outputs of synthesis-invariant elements such as black box cells
- Three-state devices
- Sequential elements
- Hierarchical ports

Note:

When performing power analysis on a partially annotated design, it is recommended that you specify a clock before running the `report_power` command. The internal zero-delay simulation requires a real or virtual clock to properly compute and propagate switching activity through the design. Use the `create_clock` command to create a clock. If no clock is available, you get a PWR-80 warning message. This does not stop propagation but the estimated switching activity might not be accurate.

Feedback

Removing the Switching Activity Annotation

Switching activity annotation can be removed from all current design objects using the `reset_switching_activity` command. This command removes all the simple and state- and path-dependent switching activity information.

In the following example, the RTL SAIF file is read before a design is compiled with power constraints and then a more accurate gate-level SAIF file is used to generate power reports:

```
read_saif rtl.back.saif -strip_path tb_rtl/i
compile_fusion
...
reset_switching_activity
read_saif gate.back.saif -strip_path tb_gate/i
report_power
```

Note that in this example, the SAIF map is already initialized.

You can selectively remove the switching activity information from individual design objects using the following command:

```
fc_shell> reset_switching_activity objects
```

When removing non-essential activity, you can expect the RTL simulated activity of the following objects to be removed:

- Hierarchical pins
- Primary output ports
- Combinational output ports

ICG output pin switching activity is no longer removed using the `reset_switching_activity -non_essential` command.

Retrieving Switching Activity Using the get_switching_activity Command

To query the activity on a pin or net, use the `get_switching_activity` command. You can use this command to select the type of activity to report based on modes, scenarios, corners, and so on. Use the `-related_clock` option to report which clock was used to calculate the toggle rate if default activity was applied to the object.

For example,

```
fc_shell> get_switching_activity APSHOLD_15/Z -related_clock
(mode = func, corner = best, probability = 0.173116,
toggle_rate = 0.167328, type = propagated, related_clock = Rclk)
```

Feedback

For this repeater output pin, the mode, corner, probability, toggle rate, and related clock are returned. Additionally, the tool derived this activity by propagating activity along the logic path rather than using activity annotated from constraints or from a SAIF file. The related clock is for this pin is Rclk, which is the clock that the tool would use to derive default activity for this pin if it did not propagate activity.

You can get state-dependent, path-dependent data on a cell that was either specified by a SAIF file or using the `set_switching_activity` command. The following example specifies a static condition using the set_switching_activity command for the FF1 cell:

```
fc_shell> set_switching_activity FF1 \
   -static_probability 0.3 \
   -state_condition D&!SI&!SE&!CP

fc_shell> get_switching_activity FF1
scenario_1 {(LEAKAGE (COND (D&!SI&!SE&!CP) (T0 0.7) (T1 0.3)))}
```

To change the output format of this command, use the `power.get_switching_activity_format` application option. If you set this application option to `list`, you can report multiple pins of a cell in one command.

Design Objects Without Annotated Switching Activity

To calculate power, the Fusion Compiler tool needs switching activity information for all design nets and state- and path-dependent information for all design cells and pins. Switching activity that is not user annotated is estimated automatically before power is calculated. This is performed in the following stages:

- The user-annotated and default-annotated switching activities are used to derive the simple static probability and toggle rate information for the rest of the design nets.
- The simple switching activity information (user-annotated or estimated) is used to derive the nonannotated state- and path-dependent switching activity.

Default Switching Activity Values

The following types of nets are automatically annotated with switching activity based on the logic of the design:

- Nets driven by constants: A toggle rate value of 0.0 is used. A static probability value of 0.0 is used for logic 0 constants, while a value of 1.0 is used for logic 1 constants.
- Nets driven by clocks: The toggle rate and static probability are derived from the clock waveform.

Feedback

- Nets driving or driven by buffers: The switching activity of a nonannotated buffer input or output is set to match the switching activity already determined for the other side of the buffer.
- Nets driving or driven by inverters: The switching activity of the inverter input or output is based on the switching activity already determined for the other side of the inverter. The toggle rate is the same and the static probability is complementary.
- Flip-flop outputs: If a flip-flop cell has both Q and QN output ports and only one of the outputs is annotated, then the other output is assigned the same toggle rate and the complementary static probability.
- Inputs and outputs of black box cells: The switching activity cannot be propagated through a black box. Therefore, the default switching activity is annotated on the outputs of a black box.

The default switching activity depends on the value of the `power.default_static_probability` and `power.default_toggle_rate` application options. The default static probability is 0.5. To specify a different value, set the `power.default_static_probability` application option.

See Also

- Analyzing Power With Partially Annotated Designs

Switching Activity From Timing Constraints

Switching activity can also be derived from timing constraints using the `create_clock` command.

Toggle rate is calculated assuming that two toggles occur per clock period and the waveform is used to calculate the static probability as follows:

```
create_clock -period 10 [get_port clk] -waveform {0 5}
toggle rate = 2/10
static_probability 5/10 = 0.5
```

In the following example, the toggle rate is `0` and static probability is set depending on the input.

```
set_case_analysis 0 or 1
```

Propagating the Switching Activity

For nets that are not user-annotated and not assigned switching activity information by default, the tool uses a zero-delay simulator to propagate switching activity from known nets. Random simulation vectors are generated for the user-annotated and default-

Feedback

annotated nets depending on the annotated toggle rate and static probability values. The zero-delay simulator uses the functionality of the design cells and the random vectors to obtain the switching activity on nonannotated cell outputs.

The number of simulation steps performed by this mechanism depends on the analysis effort option applied to the `report_power` command. User-annotated and default-annotated switching activity values are never overwritten by values derived by the propagation mechanism.

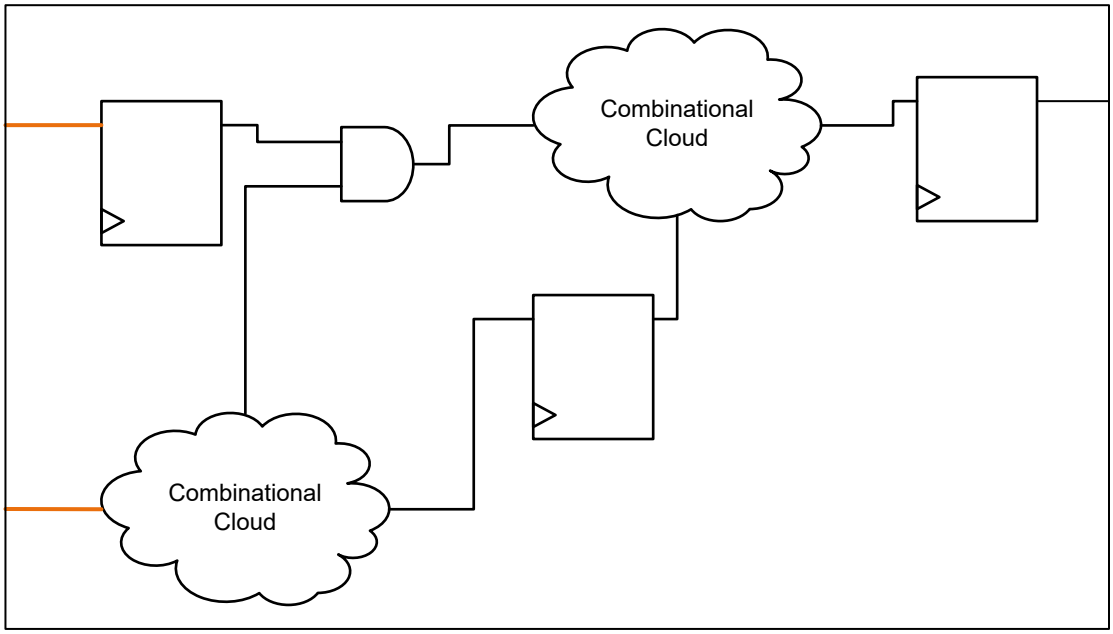
However, if a design net is not annotated with both toggle rate and static probability values, then the switching activity on this net cannot be used by the propagation mechanism. For such nets, the propagation mechanism estimates the nonannotated value.

Activity propagation can be triggered by the `report_power`, `propagate_switching_activity`, and `infer_switching_activity` commands or by optimizations when required. Commands such as `report_activity` or `get_switching_activity` do not update switching activity.

Propagating Default Activity

If you do not explicitly provide switching activity by using a SAIF file, the `set_switching_activity` command, or SDC constraints (such as the `create_clock` and `set_case_analysis` commands), the tool propagates the defaults. The default activity is applied to primary net drivers such as the top-level input ports as shown in orange in Figure 14.

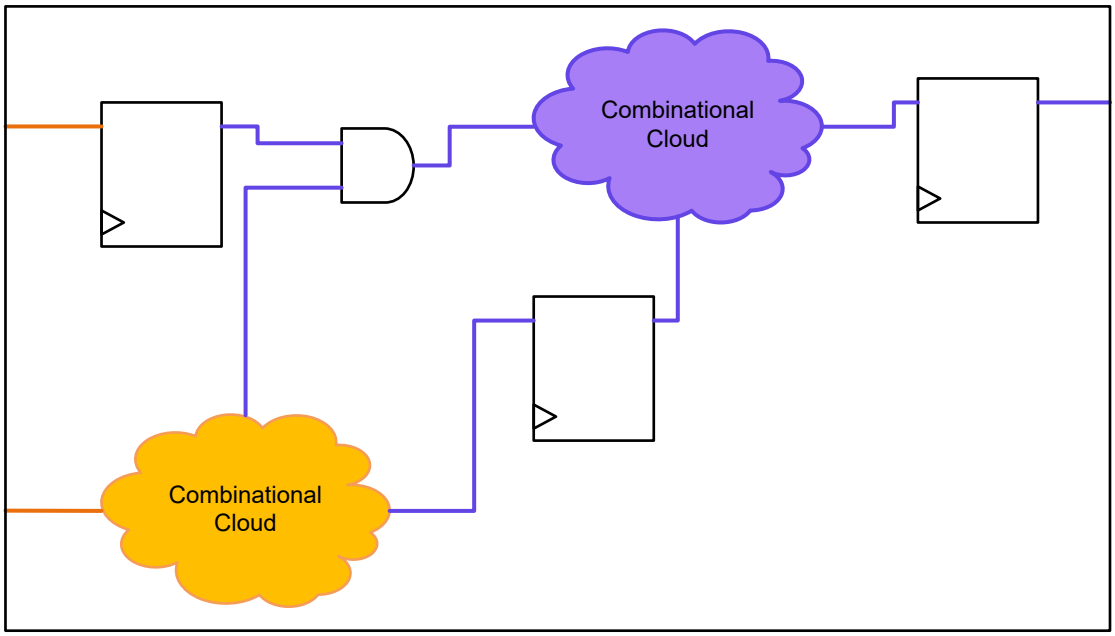
Figure 14 Default Activity Applied to Primary Net Drivers



Feedback

You can then propagate the activity manually (using the `propagate_switching_activity` command) or automatically (using the `report_power` command) to the remaining nets as shown in purple in Figure 15.

Figure 15 Activity Propagated to Remaining Nets

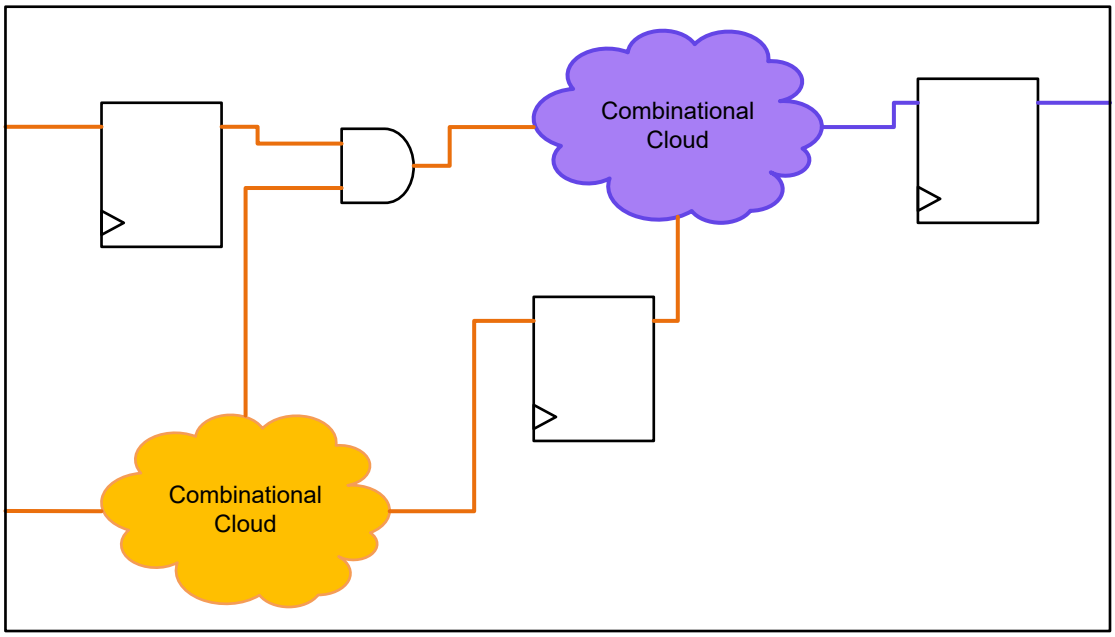


Propagating Partial Activity

If your design is only partially annotated with activity as shown in orange in Figure 16, you need to propagate the activity (purple) to determine the static probability and toggle rate for the remaining logic.

Feedback

Figure 16 Propagating Partial Activity



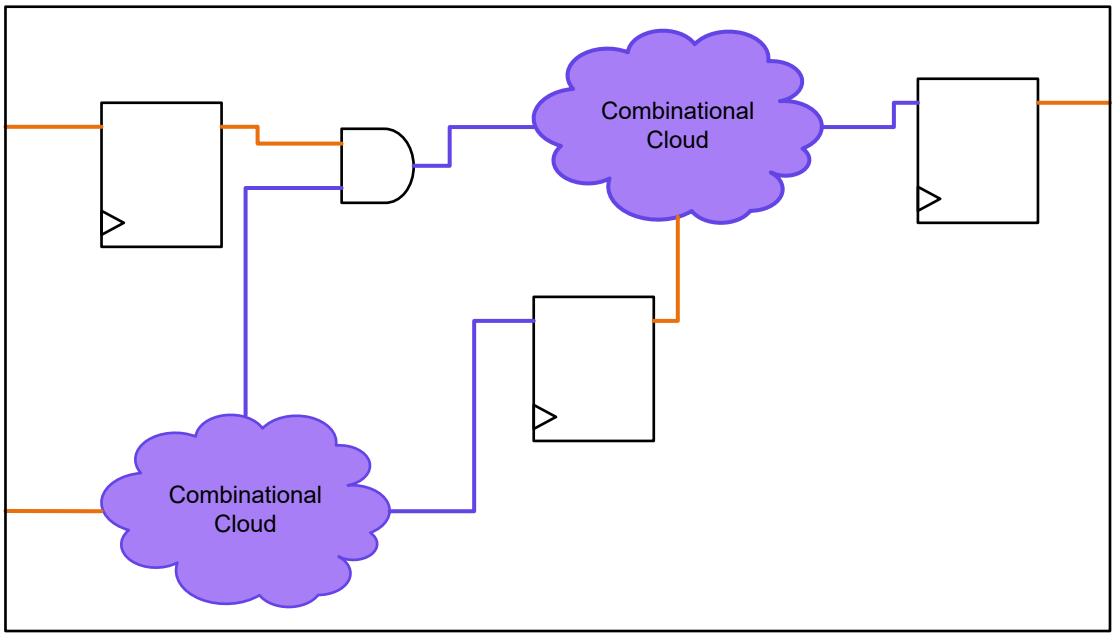
Propagating Driver-Only (Invariant Objects) Activity

When you use an RTL SAIF file, the file usually specifies switching activity only for synthesis invariant objects. These objects are typically ports and sequential elements. This activity is often referred to as RTL activity, driver activity, or essential point activity.

In Figure 17, switching activity is applied only to the drivers of nets (shown in orange). The switching activity then needs to be propagated through the design to derive the activity for the remaining nets (shown in purple).

Feedback

Figure 17 Propagating Activity for Invariant Objects



Propagating Activity for Self-Gated Registers

When a register is self-gated during optimization, the annotated activity is no longer valid. The tool calculates new switching activity and propagates the new values, replacing the annotated or simulated values on the sequential output pins of the registers.

You can expect the number of sequential elements with simulated switching activity to drop because the self-gated registers have propagated activity instead of simulated activity.

Deriving the State- and Path-Dependent Switching Activity

If an RTL SAIF file or a gate-level SAIF file without state- and path-dependent switching information is used to annotate the design switching activity, Fusion Compiler needs to estimate the required state- and path-dependent switching activity information. After obtaining the simple switching activity (from user annotation, or by switching activity propagation), Fusion Compiler estimates the state-dependent static probability information for every cell, and the state- and path-dependent toggle rate information for every cell pin. This information is obtained from the switching activities of each cell input and output pins. Although the state- and path-dependent estimation mechanism produces accurate power calculations, for best power results, use the gate-level SAIF files with state- and path-dependent information.

Feedback

Note:

State-dependent and path-dependent activity is used for power analysis only. It is not used during activity propagation.

Inferring Switching Activity

If switching activity is not specified, the tool obtains values from the `power.default_static_probability` and `power.default_toggle_rate` application options. However, these values do not provide useful results when applied to essential points (such as a sequential output port) that drive control points. A control point is an input to a pin such as a reset or scan enable pin. To infer more suitable activity values to control points, use the `infer_switching_activity` command.

When you infer switching activity, the tool identifies essential points that drive control points and determines a constant activity value to annotate on the essential point. Table 7 lists the different types of essential points that are supported.

Table 7 Definitions of Supported Essential Points

Essential Points	Note
Primary input	
Primary inout	Only the input part of the port
Sequential output	Does not include sequential integrated clock-gating cells
Tri-state output	
No-func	An output pin without a function such as a black box
No-driver	A net without a driver

The `infer_switching_activity` command can also apply default activity to essential points that do not drive control inputs. Note that the command does not apply inferred activity unless you specify the `-apply` option. If you do not specify the `-apply` option, the command reports the inferred activity.

Table 8 shows the tool support for different types of control points.

Table 8 Special Control Point Support

Special Control Point	Supported
Reset	Y
Set	Y

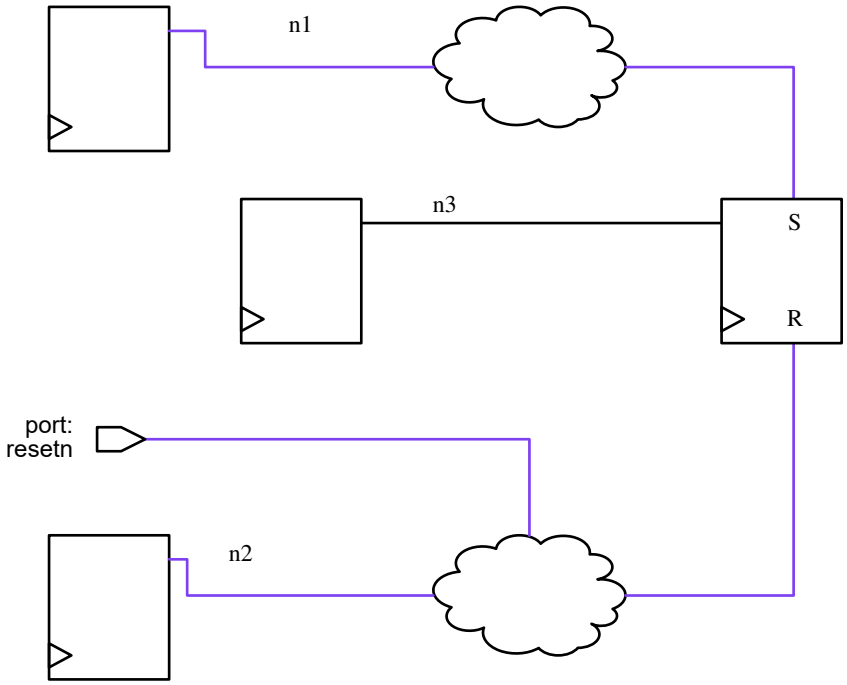
Feedback

Table 8 Special Control Point Support (Continued)

Special Control Point	Supported
Scan Enable	Y
Isolation control	Y
Power switch control	Y
Integrated clock gate enable	N
Retention control	N

For the example shown in Figure 18, the control paths highlighted in purple drive control points.

Figure 18 Example of Control Paths



Path n3 does not drive any control points, so the tool can annotate default activity on this path when it infers switching activity.

Feedback

Scaling Switching Activity

If the clock frequency used in the SAIF file is different from the clock frequency used in the tool, you can scale the switching activity using the `set_power_clock_scaling` or `read_saif -scaling_ratio` command.

Alternatively, you can scale the switching activity by performing the following steps:

1. Read the SAIF file by using the `read_saif` command.
2. Enable scaling by setting the `power.use_generated_clock_scaling_factor` application option to `true`.
3. Scale the activity by using the `set_power_clock_scaling` command.

   When you use the `set_power_clock_scaling` command, the tool scales the switching activity applied with the `read_saif` command. It does not scale switching activity applied using the `set_switching_activity` command or switching activity within block abstracts.

The scaled switching activity is persistent in the design. You can save this by using the `write_saif` command.

Saving the Switching Activity

By default, the switching activity is persistent and does not need to be annotated or propagated again between sessions of the tool.

The switching activity for your design is automatically updated during any optimization stages so the tool can track power usage throughout the flow.

You can control how often switching activity is saved and reloaded by setting the `power.enable_activity_persistency` application option. By default, it is set to `lazy` which means the activity is stored and reloaded only when needed.

To disable persistence, set the application option to `off`. In this case, you need to reload the activity between sessions.

Updating Activity With PrimePower In-Design

PrimePower In-Design in the Fusion Compiler tool uses input waveform information from the RTL FSDB file and performs time-based analysis to update the activity for enhanced power estimation accuracy. For the best correlation with standalone PrimePower time-based analysis results, run the PrimePower tool after the `route_opt` command.

Feedback

Use the `set_indesign_primepower_options` command to set PrimePower analysis options for the In-Design flow. You must specify the RTL FSDB activity file with the `-fsdbs` option and the path to the PrimePower executable with the `-pwr_shell` option. Other options allow you to specify details about the FSDB file, settings for distributed or concurrent processing, settings for scenarios and libraries, and output files.

Use the `update_indesign_activity` command to perform the analysis, which includes the following tasks:

- Invoke the PrimePower tool and read the design data which includes netlists, libraries, PVT settings, constraints, essential name mapping data, UPF, and so on. Library details come from the design database stored in the Fusion Compiler tool memory.
- Read the RTL FSDB file into the PrimePower tool and perform time-based analysis to generate a gate-level SAIF.
- Read the gate-level SAIF back into the Fusion Compiler tool.

Set the power scenario as the current scenario before starting analysis with the `update_indesign_activity` command. Activity is refreshed for the current scenario.

If you specify multiple scenarios using the `-scenarios` option of the `set_indesign_primepower_options` command, the refreshed gate-level SAIF from the PrimePower tool is read back into the Fusion Compiler tool for all the specified scenarios.

PrimePower In-Design requires the following licenses:

- Fusion-Compiler-NX
- Fusion-Compiler-BE-NX
- PP-Base
- PP-Elite

By default, the `PP-Elite` license is checked out to use advanced cycle power analysis with scalable distributed netlist partitioning and fast cycle power to accelerate SoC-level gate-level power analysis. If it is not available, PrimePower In-Design uses PrimePower cycle accurate power analysis (CAPP) for time-based power analysis.

To run PrimePower In-Design, ensure that the `saif_map` command is run before the `compile_fusion` command and immediately after the `elaborate` command and there are no unmapped objects in your design. Specify the location of the PrimePower executable and FSDB file, the HDL language in the `-format` option, and the time window (in nanoseconds) with the `-time` option if required.

```
elaborate
set_top_module top
saif_map -start
source sdc.tcl
compile_fusion -to logic_opto
```

Feedback

```
set FSDB_FILE ../simulation/top_rtl.fsdb
set STRIP_PATH "top_tb/top"
set PWR_SHELL /tools/pt_2021.06-SP3/bin/pwr_shell
set_indesign_primepower_options \
    -fsdbs {{ $FSDB_FILE \
     -strip_path $STRIP_PATH \
 -format systemverilog \
-time {10 120}
     }} \
    -output_dir PP_WORKSPACE \
    -max_cores 4 \
    -pwr_shell $PWR_SHELL

update_indesign_activity
```

You must ensure that the specified executable is compatible with the Fusion Compiler tool.

Table 9 lists the commonly used options for the `set_indesign_primepower_options` command. For a detailed description of each option, see the man page.

Table 9 Commonly Used Options for the set_indesign_primepower_options Command

Option	Description
`-fsdbs fsdb_file_list { }`	Specifies a list of FSDB file options to annotate switching activities. Each list item consists of the FSDB file name and optionally the strip path, path, time, scaling ratio, format, analyze scenarios, back-annotate scenarios, and weight.
`fsdb_file_name`	Specifies the name of the FSDB file.
`-strip_path strip_path_string`	Specifies the name of the instance of the current design as it appears in the FSDBs.
`-path path_string`	Specifies the path of the instance from the top of the design.
`-time list`	Specifies the time window (in ns) in which the activities are counted for power calculation.
`-scaling_ratio float`	Specifies the scaling ratio for scaling toggle rates. The default is 1.
`-format string`	Specifies the language format (Verilog, VHDL, or SystemVerilog) for the names in FSDB. The default is Verilog.
`-pwr_shell executable_path`	Specifies the PrimePower executable.

Feedback

Table 9 Commonly Used Options for the set_indesign_primepower_options Command (Continued)

Option	Description
`-output_dir out_dir_name`	Specifies the name of the output directory that stores all the generated files including design data. By default, the output directory gets deleted after the `update_indesign_activity` command is executed.

Reviewing the PrimePower In-Design Run

By default, the PrimePower In-Design output directory gets deleted after the `update_indesign_activity` stage. Use the `-keep` option to specify those files to be retained during the In-Design PrimePower flow. The valid values are as follows:

- `none` - Deletes all the files generated during the PrimePower In-Design flow
- `all` - Retains all the generated files
- `saif` - Retains all the generated gate-level SAIF files; all other files are deleted

When the `-keep` option is set to `all`, PrimePower In-Design generates the report_switching_activity.rpt file in its working directory. The report_switching_activity.rpt file includes the output of the PrimePower `report_switching_activity -list_not_annotated -show_pin -include_only sequential` command. Check whether essential points in PrimePower have a high annotation rate before continuing with the implementation flow.

```
****************************************
Report : Switching Activity
    -list_not_annotated
    -show_pin
Design : top
Version: S-2021.06-SP3
Date   : Tue Apr  5 12:32:22 2022
****************************************

 Switching Activity Overview Statistics for "top"
---------------------------------- ... ------------------------------------------------------------------------
                  From Activity    ...                                          Not
Object Type       File (%)         ...     Propagated(%)   Implied(%)      Annotated(%)  Total
                                   ...
---------------------------------- ... ------------------------------------------------------------------------
 Nets             168(100.00%)     ...     0(0.00%)        0(0.00%)        0(0.00%)      168
---------------------------------- ... ------------------------------------------------------------------------
Nets Driven by
---------------------------------- ... ------------------------------------------------------------------------
Primary Input     0(0%)            ...     0(0%)           0(0%)           0(0%)         0
Tri-State         0(0%)            ...     0(0%)           0(0%)           0(0%)         0
Black Box         0(0%)            ...     0(0%)           0(0%)           0(0%)         0
Sequential        168(100.00%)     ...     0(0.00%)        0(0.00%)        0(0.00%)      168
Combinational     0(0%)            ...     0(0%)           0(0%)           0(0%)         0
Memory            0(0%)            ...     0(0%)           0(0%)           0(0%)         0
Clock Gate        0(0%)            ...     0(0%)           0(0%)           0(0%)         0
---------------------------------- ... ------------------------------------------------------------------------
```

Feedback

Performing Glitch Estimation

When you use the `-delay_shifted_event_analysis` option with the `set_indesign_primepower_options` command, PrimePower uses delay-shifted power analysis to shift all the events by the delay factor associated with the corresponding pin. With delay shifting, you can model glitch power consumption using RTL vectors without running gate-level simulation.

PrimePower In-Design generates a gate-level SAIF containing the inertial glitch (IG) count and transport glitch (TG) count for all nets. Activity-based glitch power analysis (`power.saif_glitch_handling`) is automatically enabled in the Fusion Compiler tool.

Glitch power analysis is performed using glitch annotations specified in the PrimePower In-Design SAIF input and glitch power numbers are included in the `report_power` command report.

```
set_indesign_primepower_options \
-fsdbs {...} \
-delay_shifted_event_analysis
update_indesign_activity
report_power
```

```
   Cell Internal Power    = 3.52e+07 pW ( 77.5%)
   Net Switching Power    = 1.02e+07 pW ( 22.5%)
   Annotated Glitch Power = 5.73e+03 pW (  0.0%)
   Estimated Glitch Power = 1.26e-09 pW (  0.0%)
Total Dynamic Power       = 4.54e+07 pW (100.0%)

Cell Leakage Power        = 8.76e+09 pW

Note: internal and switching power includes power due to functional toggles only (excluding glitch).

  Attributes
  ----------
      u  -  User defined power group
      a  -  Annotated glitch power present
      e  -  Estimated glitch power present

Power Group     Internal Power    Switching Power      Glitch Power      Leakage Power      Total Power    (   %  )    Attrs
--------------------------------------------------------------------------------------------------------------------------
io_pad               0.00e+00          0.00e+00          0.00e+00          0.00e+00          0.00e+00    (  0.0%)
memory               0.00e+00          0.00e+00          0.00e+00          0.00e+00          0.00e+00    (  0.0%)
black_box            0.00e+00          0.00e+00          0.00e+00          0.00e+00          0.00e+00    (  0.0%)
clock_network        4.69e+07          0.00e+00          0.00e+00          2.99e+08          3.46e+08    (  3.9%)
register            -2.29e+07          3.18e+06          0.00e+00          5.07e+09          5.05e+09    ( 57.4%)
sequential           0.00e+00          0.00e+00          0.00e+00          0.00e+00          0.00e+00    (  0.0%)
combinational        1.12e+07          7.02e+06          5.73e+03          3.39e+09          3.41e+09    ( 38.7%)       ea
--------------------------------------------------------------------------------------------------------------------------
Total                3.52e+07 pW       1.02e+07 pW       5.73e+03 pW       8.76e+09 pW       8.80e+09 pW
```

Performing PrimePower In-Design Flow for MultiCorner-MultiMode Designs

For multicorner-multimode designs, PrimePower In-Design allows you to refresh activity using multiple FSDBs with multiple analyze scenarios. To do so, specify the necessary settings with the `set_indesign_primepower_options` command.

The usage of the `set_indesign_primepower_options` command is as follows:

```
set_indesign_primepower_options -fsdbs
{ \
{ \fsdb_file_name1 \
[-strip_path strip_path] \
[-path path] \
[-time time_window] \
[-scaling_ratio scaling_ratio] \
[-format format_type] \
[-analyze_scenarios scenarios_list] \
[-back_annotate_scenarios scenarios_list] \
[-weight value] \
} \
{ \fsdb_file_name2 \
[-strip_path strip_path] \
[-path path] \
[-time time_window] \
[-scaling_ratio scaling_ratio] \
[-format format_type] \
[-analyze_scenarios scenarios_list] \
[-back_annotate_scenarios scenarios_list] \
[-weight value] \ }
\}
```

Performing Distributed Analysis

For faster runtime, use distributed analysis, which is disabled by default in the PrimePower tool.

To enable distributed analysis in the PrimePower tool, specify the following options with the `set_indesign_primepower_options` command:

- `-num_processes`: Use this option to specify the number of hosts to launch. The value must be greater than 1.
- `-submit_command` or `-host_names`: Use either of these options to specify the configuration of host machines.
- `-max_cores`: Use this option to specify the maximum number of cores. The default is 4.

If valid options are specified, the tool issues the following message:

```
Information: PrimePower is being called in distributed mode
```

4

Power Analysis

After capturing switching activity, mapping your design to gates, and annotating your design, use the `report_power` command to report the power consumption of the elements of the design.

The tool creates power reports for

- Design
- Modules
- Nets
- Cells or groups of cells of specific type
- Scenarios, in case of multicorner-multimode designs

The information in this section describes the Fusion Compiler tool's power analysis engine and how to perform power analysis. It contains the following topics:

Identifying Power and Accuracy

The Fusion Compiler tool uses different methods to compute the power of your design. The tool considers the type and amount of switching activity annotated on your design and

Feedback

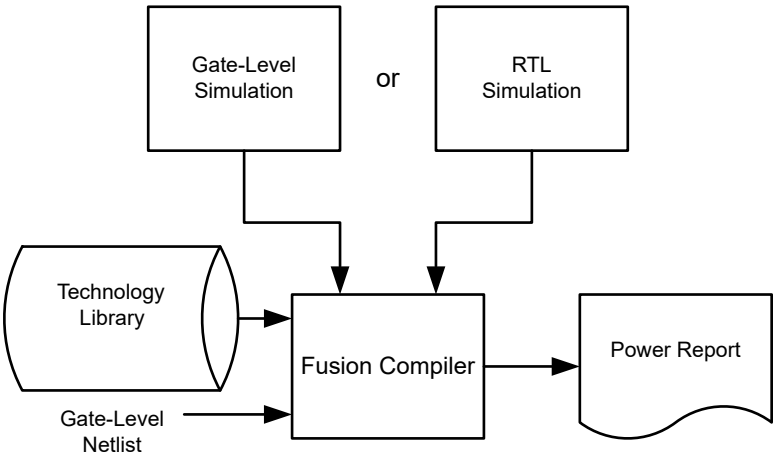
chooses the most accurate method to compute your design's power. The method used depends on whether you annotate some or all of the elements in your design.

To analyze your gate-level design, the following inputs are required:

- Switching activity
- Logic library
- Gate-level netlist

Figure 19 shows the inputs to Fusion Compiler.

Figure 19 Inputs to Fusion Compiler



For best results, use the logic libraries characterized with power information. If the library has only pin capacitance and voltage, but no power information, only the switching power of the net is reported. The switching power is a function of the pin capacitance, voltage, and toggle frequency. You can generate the report by using the `report_power` command. The power number reported corresponds to the switching power of the net, which is a function of the pin capacitance, voltage, and toggle frequency.

Factors Affecting the Accuracy of Power Analysis

The following factors can affect the accuracy of power analysis:

- Switching activity annotation
- Delay model
- Correlation
- Clock tree buffers
- Complex cells

Feedback

Switching Activity Annotation

Annotating switching activity relies on the ability to map the names of the synthesis invariant objects in the RTL source to the equivalent object names in the gate-level netlist. Mapping inconsistencies can cause the SAIF file to be incorrectly or incompletely annotated, which can affect the power analysis results. In turn, the quality of these results affects the results of power optimizations that rely on the annotation. For more information, see Annotating the Switching Activity Using RTL SAIF Files.

Clock Frequency Scaling

If a design is synthesized at a frequency that is different from the frequency the simulation is run, the SAIF file generated from the simulation reflects this difference. This causes a mismatch in timing and affects dynamic power analysis.

To correct this problem, the tool allows you to scale the clock frequency, resulting in a more accurate dynamic power analysis.

For more information, see Scaling Switching Activity.

Delay Model

The Fusion Compiler tool uses a zero-delay model for internal simulation and for propagation of switching activity during power analysis. This zero-delay model assumes that the signal propagates instantly through a gate with no elapsed time.

The zero-delay model has the advantage of enabling fast and relatively accurate estimation of power dissipation. The zero-delay model does not include the power dissipated due to glitching. If your power analysis must consider glitching, use power analysis after annotating switching activity from full-timing gate-level simulation. As mentioned previously, the internal simulation is used only for nodes that do not have user-annotated switching activity.

Switching Activity Propagation and Accuracy

While propagating switching activity through the design, the logic states of inputs of the gates' can have interdependencies that affect the accuracy of any statistical model. Such interdependency of inputs is called correlation. Correlation affects the accuracy of propagation of toggle rates. Because accurate analysis depends on accurate toggle rates, correlation also affects the accuracy of power analysis.

For more information, see Propagating the Switching Activity.

Correlation

The Fusion Compiler tool considers correlation within combinational and sequential logic, resulting in more accurate analysis of switching activity for many types of designs. The types of circuits that exhibit high internal correlation are designs with reconvergent fanouts, multipliers, and parity trees. However, Fusion Compiler has no access to

information about correlation external to the design. If correlation exists between the primary inputs of the design, the tool does not recognize the correlation.

The Fusion Compiler tool considers correlation only within certain memory and CPU thresholds, beyond which correlation is ignored. As the design size increases, the tool reaches its memory limit and is not able to fully consider all internal correlation.

As an example of correlation, consider a 4-bit arithmetic logic unit (ALU) that performs five instructions. The data bus is 4-bits wide, and the instruction opcode lines are 3-bits wide. The assumption of uncorrelated inputs holds up well for the data bus lines inputs but fails for the opcode inputs if some instructions are used more often.

If your design has black boxes, such as complex cells, RAM, ROM, or macro cells you can annotate switching activity at the outputs of these elements.

Configuring Scenario Settings for Power Analysis

Before you perform power analysis, set your scenario status. A scenario is a combination of a corner and a mode used to perform timing analysis and optimization. By default, power analysis and optimization is performed only for scenarios that have leakage or dynamic power enabled.

To enable power analysis and optimization, configure at least one scenario for power analysis. For example:

```
fc_shell> set_scenario_status func::RCworst \
    -leakage_power true \
    -dynamic_power true -active true
```

You can use the `report_power -force` command to run power analysis on scenarios that are not enabled for power optimization.

Power analysis also requires annotated activity on the design. This includes simulated activity and manual activity constraints. Constraints can also be derived from SDC constraints such as the `create_clock` and `set_case_analysis` commands.

Performing Gate-Level Power Analysis

After annotating your design with switching activity, use the `report_power` command to report the power of your gate-level design.

To perform power analysis on a partially annotated design, specify a clock before invoking the `report_power` command. The internal zero-delay simulation requires a real or virtual clock to properly compute switching activity. Use the `create_clock` command to create the clock.

Feedback

Using the report_power Command

The `report_power` command calculates and reports power for a design. If you do not annotate switching activity on the nets, the command performs zero-delay simulation to propagate switching activity for the nets. To compute the switching activity for internal nets, the command uses the switching activity for startpoint nets (if available). The nets that are annotated using the `set_switching_activity` or the `read_saif` command are not overwritten during the switching activity propagation.

To ensure that the design contains fully annotated switching activity, the `report_power` command propagates switching activity automatically.

If you annotate switching activity on all the elements of the design, the tool does not propagate any switching activity through the design. Instead, power analysis uses the annotated gate-level switching activity.

Command options enable you to print with different sorting modes and with verbose and cumulative options. The default operation is to print a power summary for the instance's subdesign (in the context of the higher-level design).

Power analysis uses any net loads during the power calculation. If you have annotated any cluster information about the design using Synopsys Floorplan Manager, the tool uses the improved capacitance estimates from the cluster's wire loads.

Power analysis uses the RC loads extracted from the design during the power calculation. If this extraction was not already performed, the tool runs it before generating the report. For nets that do not have back-annotated capacitance, the tool uses the estimated or virtual route information annotated during extraction.

The `report_power` command calculates and reports static and dynamic power for the current design. It uses the user-annotated switching activity to calculate the net switching power, cell internal power, and cell leakage power. When you do not specify any option, by default, the `report_power` command displays the summary of power values only for the current design (see Figure 20). If you specify a cell instance, the command reports the

summary power values for the specified instance. The command supports several options, for you to specify cells, nets, scenarios, include or exclude boundary nets, and so on.

Figure 20 Example of the report_power Command

```
  Cell Internal Power    = 1.80118e+07 nW ( 66.4%)
  Net Switching Power    = 9.10007e+06 nW ( 33.6%)
Total Dynamic Power      = 2.71119e+07 nW (100.0%)

Cell Leakage Power       = 1.00272e+07 nW


  Attributes
  ----------
      u  -  User defined power group

Power Group         Internal Power        Switching Power          Leakage Power            Total Power    (   %  )    Attrs
--------------------------------------------------------------------------------------------------------------------------
io_pad                0.00000e+00           0.00000e+00           0.00000e+00           0.00000e+00    (  0.0%)
memory                0.00000e+00           0.00000e+00           0.00000e+00           0.00000e+00    (  0.0%)
black_box             0.00000e+00           0.00000e+00           0.00000e+00           0.00000e+00    (  0.0%)
clock_network         1.07049e+07           2.49115e+06           7.03857e+04           1.32665e+07    ( 35.7%)
register              2.29443e+06           1.54092e+06           2.03614e+06           5.87149e+06    ( 15.8%)
sequential            0.00000e+00           0.00000e+00           0.00000e+00           0.00000e+00    (  0.0%)
combinational         5.01248e+06           5.06800e+06           7.92071e+06           1.80012e+07    ( 48.5%)
--------------------------------------------------------------------------------------------------------------------------
Total                 1.80118e+07 nW        9.10007e+06 nW        1.00272e+07 nW        3.71392e+07 nW
```

For more information, see the `report_power` command man page.

Using the report_power_calculation Command

The Fusion Compiler tool uses a complex mechanism to calculate dynamic and leakage power. The dynamic power consists of internal power on pins and switching power on nets. Both internal and leakage power might be state dependent.

The `report_power_calculation` command shows how the reported power numbers are derived from the inputs such as library, simulation data, netlist, and parasitics. You can use this information for debugging or verifying power data in a logic library or the power calculation for a design object. This command does not work on libraries that have built-in security to protect the power table numbers. This restriction does not apply for switching power.

Note:

The `report_power_calculation` command is also available in the PrimePower tool to allow you to compare how the power is derived for a design object.

For more information, see the `report_power_calculation` command man page.

Feedback

Analyzing Power With Partially Annotated Designs

If you have nets that have not been annotated, simulated, propagated, or derived from static timing analysis, the tool uses the following defaults for the primary inputs of your design:

- P_1 = 0.5 (the signal is in the 1 state 50 percent of the time)

  P_1 is the probability that input P is at logic state 1. For definitions of static probability, P_1, and toggle rate (TR), see Types of Switching Activity to Annotate.

- TR = $TR_{default}$ * f_{clk} (the signal switches one time, every 10 clock cycles)

  $TR_{default}$ is the value of the `power.default_toggle_rate` application option. By default, the value is 0.1.

  f_{clk} is the frequency of the input's related clock in the design, as defined by the `set_switching_activity` command. You can specify the related clock explicitly with its clock name or implicitly as "*". In the latter case, the tool infers a related clock automatically. If the input port does not have a related clock, the tool uses the value of the `power.default_toggle_rate_reference_clock` application option, which specifies whether to use the `fastest` or `related` clock.

Using the defaults for static probability and toggle rate can be reasonable for data bus lines. However, the defaults might be unacceptable for some signals, such as a reset or a test-enable signal. For more information on how to annotate these nets, see Inferring Switching Activity.

If you do not annotate toggle information on primary inputs, these inputs assume the default toggle value. If the input or logic connected to this input is heavily loaded, the results could be significantly different from what you expect.

To change the default for switching activity and static probability, set the following application options:

- `power.default_static_probability`

  This variable sets the default for static probability.

- `power.default_toggle_rate`

  This variable sets the default for toggle rate.

- `power.default_toggle_rate_reference_clock`

  The default is `fastest`, which causes the tool to calculate the default toggle rate by multiplying the fastest clock frequency with the default toggle rate. Set this variable to `related` to determine the behavior using the clock that is associated with the object.

The variables remain in effect throughout the shell session in which you set them.

Feedback

The following example sets the default static probability to 0.3:

```
fc_shell> set_app_options power.default_static_probability 0.3
```

The following example sets the default toggle rate to 0.4 of the toggle rate of the highest-frequency clock:

```
fc_shell> set_app_options power_default_toggle_rate 0.4
```

Analyzing Switching Activity Annotation

To check the annotation rate, use the `-driver` option with the `report_activity` command. It is not recommended to refer to the report issued after the `read_saif` command to check the annotation rate.

The following example shows how to check the annotation rate:

```
fc_shell> report_activity -driver -scenario $power_scenario
****************************************
Report : activity
         -driver
Design : top_implementation
Version: Q-2019.12-SP5
Date   : Sat Oct 31 13:50:48 2020
****************************************

Scenario 'power_scenario' (mode power', corner power')

Activity Type  primary-input   seq-pin      comb-pin    no-driver  total
--------------------------------------------------------------------------------
simulated       5 (62.5%)   152 (90.5%)     51 (2.8%)    2 (3.3%)    210
annotated       0 ( 0.0%)     0 (0.0%)       0 (0.0%)    0 (0.0%)      0
inferred        0 ( 0.0%)     0 (0.0%)       0 (0.0%)    0 (0.0%)      0
derived         0 ( 0.0%)     0 (0.0%)   276 (15.4%)  56 (93.3%)    332
calculated      0 ( 0.0%)     0 (0.0%)       0 (0.0%)    0 (0.0%)      0
```

```
default          3 (37.5%)      16 (9.5%)   1465 (81.8%)     2 (3.3%)    1486
-------------------------------------------------------------------------------
total            8 (100.0%)   168 (100.0%) 1792 (100.0%)   60 (100.0%)   2028
```

For reporting purposes, the switching activity is classified by a major and minor activity type. By default, the tool only shows the major activity types. However, you can view the minor classification types using the `-verbose` option. Figure 21 shows an example of the `report_activity` command in verbose mode.

Figure 21 Example of a Verbose Report From the report_activity Command

Scenario 'FUNC_0.85V_SETUP' (mode 'FUNC_085', corner 'FUNC_0.85V_SETUP')

Activity Type	port	net	leaf-pin	block-pin	hier-pin	cell-SDPD
simulated						
saif	25 (83.3%)	257 (64.9%)	1148 (46.2%)	0 (0.0%)	225 (84.6%)	0 (0.0%)
vcd	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
annotated						
set_switching_activity	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
abstract	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
derived						
create_clock	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
create_generated_clock	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
set_case_analysis	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
timer_implied	0 (0.0%)	6 (1.5%)	16 (0.6%)	0 (0.0%)	2 (0.8%)	0 (0.0%)
logic_constant	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
supply_constant	5 (16.7%)	5 (1.3%)	797 (32.1%)	0 (0.0%)	25 (9.4%)	0 (0.0%)
calculated						
implied	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
propagated	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
estimated	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
default	0 (0.0%)	128 (32.3%)	523 (21.1%)	0 (0.0%)	14 (5.3%)	347 (100.0%)
total	30 (100.0%)	396 (100.0%)	2484 (100.0%)	0 (0.0%)	266 (100.0%)	347 (100.0%)

For more information on how activity types are classified, see Classification of Switching Activity Types.

To report which objects have a certain activity type applied, use the `-print_objects` option. The arguments to the `-print_objects` option depend on the reporting mode (`rtl`, `driver`, or default) specified. You can also retrieve activity information from net, pin, and cell objects using their attributes. For more information, see Types of Power Attributes.

The following example shows the usage of the `-print_objects` option:

```
report_activity -driver -print_objects {{activity_type_list}
 {object_type_list}}
```

For example, the following commands show how to check the sequential output pins and primary input ports not annotated by SAIF activity:

```
read_saif rtl.saif
report_activity -driver -print_objects {{calculated default}
 {primary-input seq-pin}}

List of requested objects
```

Feedback

```
------------------------
propagated seq-pin BPM/control_reg[1]/Q
propagated seq-pin BPM/control_reg[2]/Q
propagated seq-pin BPM/control_reg[3]/Q
```

Classification of Switching Activity Types

Table 10 lists the different major switching activity types and their minor types. Refer to this table when using the `report_activity -verbose` command.

Table 10 Switching Activity Types

Major Activity Type	Minor Activity Type	Activity Set By
Simulated	simulated	SAIF file (read_saif)
Annotated	set_switching_activity abstract	Explicit `set_switching_activity` constraint `create_abstract`
Inferred		`infer_switching_activity -apply`

Derived	create_clock
	create_generated_clock
	set_case_analysis
	timer_implied
	logic_constant
	supply_constant
		`create_clock`
		`create_generated_clock`
		`set_case_analysis`
		Timer propagated set_case_analysis
		Verilog/VHDL netlist constant (1'b0, 1'b1)
		Connected to supply net

Calculated	implied propagated estimated	Forward or backward implication during propagation Calculated by propagation engine Quick estimation during optimization
Default	default	Fall back to default activity as specified by application options

Types of Power Attributes

Table 11 lists the different types of power attributes that apply to a net, pin, or cell object.

Table 11 Power Attributes

Attribute Name	Object Type	Description	Example
activity_type	Pin, Net	The major activity type	simulated

Feedback

Table 11 Power Attributes (Continued)

Attribute Name	Object Type	Description	Example
minor_activity_type	Pin, Net	The minor activity type	`read_saif`
toggle_rate	Pin, Net	The toggle rate	0.01
static_probability	Pin, Net	The static probability	0.5
power_group	Cell	The report_activity group	combinational

Performing a Quick Power Analysis

Follow these steps to get quick results from gate-level power analysis:

1. Create a SAIF file.

   This step requires RTL simulation. For information, see Generating SAIF Files.

2. Compile the design to gates, using suitable options.

3. Annotate switching activity on primary inputs and other synthesis-invariant elements of the gate-level design.

   For information about using SAIF files from RTL simulation to annotate switching activity, see Annotating the Switching Activity Using RTL SAIF Files.

4. Use the `report_power` command to analyze your design's power.

   The tool uses an internal zero-delay simulation to propagate switching activity through nonannotated elements of the design.

5. Repeat steps 1 through 4 for other architectures and coding styles.

   Quick gate-level power analysis enables you to see the results of changes in your RTL design.

Reporting and Analyzing Clock Power

Clock network power can be reported by the `report_power` command, under the `clock_network` power group. By default, the internal power of the register's clock pin is included in the `clock_network` power group, too. When the `power.clock_network_include_clock_sink_pin_power` application option is set to `off`, the internal sink power of the register's clock pin is reported in the `register` power group.

Feedback

To report a summary of clock power, use the `report_clock_qor -type power` command. This command reports leakage, internal, and switching power for each clock across scenarios.

For a detailed analysis of clock power, use the `report_clock_power` command. This command reports the power of the clock tree segments and subtrees. By reporting the power for each segment or subtree, you can quickly identify which sections of the clock tree to focus on for power optimization. The report shows related details such as the toggle rate and wire or pin capacitance to help you identify areas of higher power consumption.

Reporting Power Analysis Results

To report power analysis results, use the following commands:

- `report_activity`
- `report_power`
- `report_power_calculation`

Reporting Essential Points

To find out which essential points drive a particular control point, use the `report_essential_points` command. This information can help you determine what activity should be annotated on the design.

The following example shows how to find the essential points driving the reset pin of a register:

```
fc_shell> report_essential_points H6_reg_reg_1/RD

****************************************
Report : report_essential_points
Design : init
Version: ...
****************************************
Inferring activity for the scenario: virtual_scenario
Mode         : virtual_scenario
Corner       : virtual_scenario
Time Unit    : 1ns
Fastest clock: clk with period 0.8
------------------------------------------------
Object Type                : port
Object Name                : reset_n
Current Static Probability : 0.5
Current Toggle Rate        : 0.125
```

```
Receiver Type: reset | Receiver Count: 1045
-----------------------------------------------
```

The reset_n port drives 1045 reset pins and has an activity rate of 0.5 probability and a 0.125 toggle rate. This port is an active-low port and typically needs to be set to a static value of 1 and a toggle rate of 0 to ensure that the design is not being reset during power analysis.

5

Power Analysis With Block Abstracts

In the hierarchical flow, you can use block abstracts to increase the capacity and improve the runtime of the Fusion Compiler tool when working at the top level. All of the logic is retained in the block abstract, but only the interface logic is loaded. While you can abstract and delete the details of the block, you might keep some objects in the abstract for more accurate modeling or interface optimization.

For accurate power analysis results, the tool can store and report power analysis information for a block abstract.

This section contains the following topics:

Using Block Abstracts With Power Information

Power information is stored only for objects that are removed during abstraction. To store power information, you must use timing abstracts. Power information is not stored in placement abstracts.

If a design is bound to an abstract that contains power information, the power information is automatically visible for power analysis and reporting.

Cells that remain in the abstract have their own activity and the power information is calculated based on the top-level context. If the activity or boundary information is different between the block and the instantiation of the block at the top level, it is possible that the power reported at the top level differs from the power reported in the block.

Feedback

Creating Block Abstracts With Power Information

By default, power information is not stored in a block abstract. To create and store power information in an abstract,

1. Enable storing power information in the abstract.

```
fc_shell> set_app_options -name abstract.annotate_power -value true
```

2. Enable leakage or dynamic power analysis for your active scenarios.

```
fc_shell> set_scenario_status -leakage_power true -dynamic_power
 true ...
```

3. Create the block abstract using the `create_abstract` command.

   For cells that are abstracted, the power information is stored in the abstract. If switching activity is not available when you run the `create_abstract` command, the activity is automatically generated and propagated. For more information, see Annotating Switching Activity.

The following is an example script for creating an block abstract with power analysis information:

```
set_app_options -name abstract.annotate_power -value true
set_scenario_status -leakage_power true \
                    -dynamic_power true \
                    [get_scenarios -filter active]
create_abstract
```

For more information about creating timing block abstracts, see the *Fusion Compiler Design Planning User Guide*.

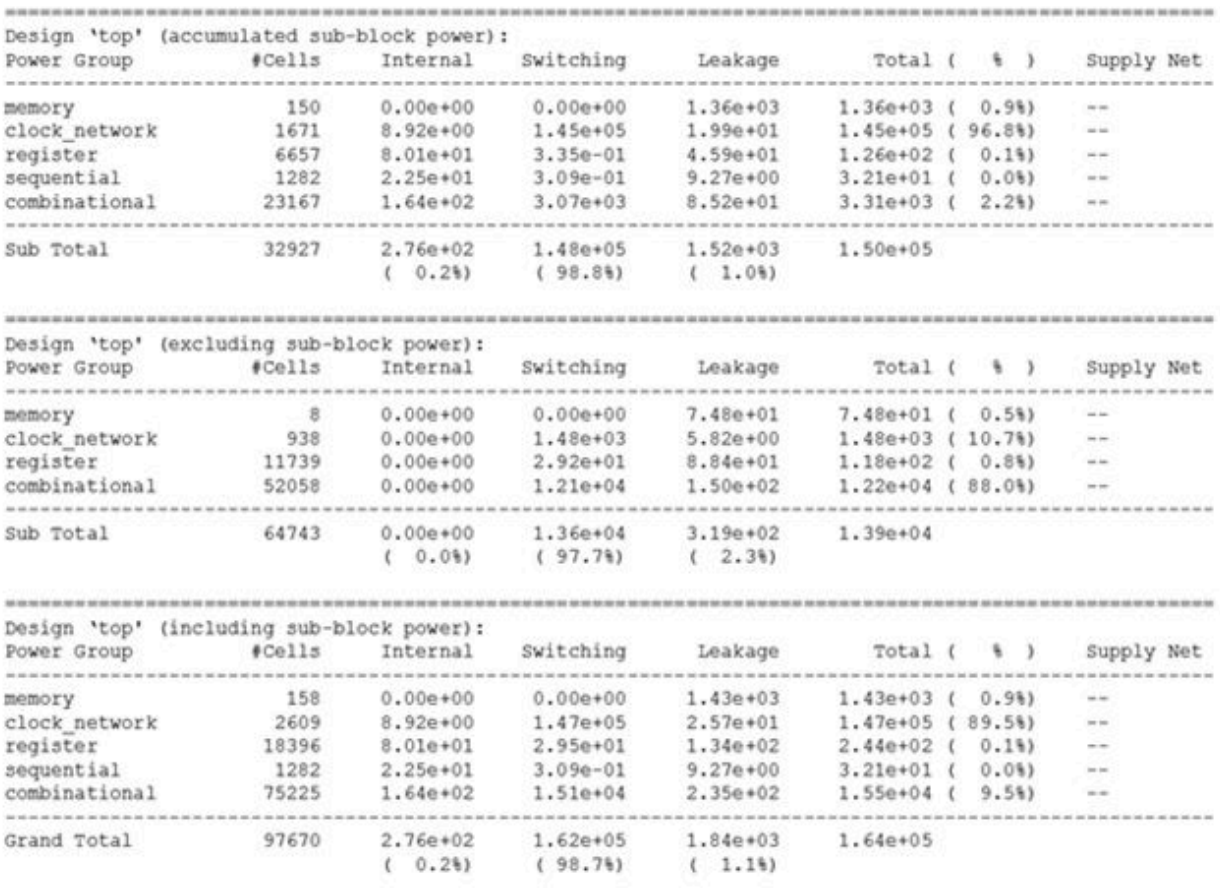
Reporting Power in Block Abstracts

To report the power information of a block abstract, use the `-blocks` option with the `report_power` command. The `-blocks` option generates a block-oriented power report. Use the `-levels` option to limit the number of levels of physical blocks to report.

Figure 22 shows an example of the `report_power -blocks` command.

Feedback

Figure 22 Output From the report_power -blocks Command



```
****************************************************************************************************
Design 'top' (accumulated sub-block power):
Power Group          #Cells     Internal    Switching      Leakage        Total (   %  )    Supply Net
----------------------------------------------------------------------------------------------------
memory                  150     0.00e+00     0.00e+00     1.36e+03     1.36e+03 (  0.9%)     --
clock_network          1671     8.92e+00     1.45e+05     1.99e+01     1.45e+05 ( 96.8%)     --
register               6657     8.01e+01     3.35e-01     4.59e+01     1.26e+02 (  0.1%)     --
sequential             1282     2.25e+01     3.09e-01     9.27e+00     3.21e+01 (  0.0%)     --
combinational         23167     1.64e+02     3.07e+03     8.52e+01     3.31e+03 (  2.2%)     --
----------------------------------------------------------------------------------------------------
Sub Total             32927     2.76e+02     1.48e+05     1.52e+03     1.50e+05
                                ( 0.2%)      ( 98.8%)     ( 1.0%)

****************************************************************************************************
Design 'top' (excluding sub-block power):
Power Group          #Cells     Internal    Switching      Leakage        Total (   %  )    Supply Net
----------------------------------------------------------------------------------------------------
memory                    8     0.00e+00     0.00e+00     7.48e+01     7.48e+01 (  0.5%)     --
clock_network           938     0.00e+00     1.48e+03     5.82e+00     1.48e+03 ( 10.7%)     --
register              11739     0.00e+00     2.92e+01     8.84e+01     1.18e+02 (  0.8%)     --
combinational         52058     0.00e+00     1.21e+04     1.50e+02     1.22e+04 ( 88.0%)     --
----------------------------------------------------------------------------------------------------
Sub Total             64743     0.00e+00     1.36e+04     3.19e+02     1.39e+04
                                ( 0.0%)      ( 97.7%)     ( 2.3%)

****************************************************************************************************
Design 'top' (including sub-block power):
Power Group          #Cells     Internal    Switching      Leakage        Total (   %  )    Supply Net
----------------------------------------------------------------------------------------------------
memory                  158     0.00e+00     0.00e+00     1.43e+03     1.43e+03 (  0.9%)     --
clock_network          2609     8.92e+00     1.47e+05     2.57e+01     1.47e+05 ( 89.5%)     --
register              18396     8.01e+01     2.95e+01     1.34e+02     2.44e+02 (  0.1%)     --
sequential             1282     2.25e+01     3.09e-01     9.27e+00     3.21e+01 (  0.0%)     --
combinational         75225     1.64e+02     1.51e+04     2.35e+02     1.55e+04 (  9.5%)     --
----------------------------------------------------------------------------------------------------
Grand Total           97670     2.76e+02     1.62e+05     1.84e+03     1.64e+05
                                ( 0.2%)      ( 98.7%)     ( 1.1%)
```

6

Budgeting Power

After you perform a full power analysis on your design, you can perform exploratory analysis by setting power budgets on the blocks or cells you want to explore. The tool computes a scaling factor based on the budgeted power and the actual total power for these blocks or cells. The tool applies the scaling factor on the affected cells or all the child cells in the block, while leaving the rest of the design intact.

For accurate power analysis results, the tool can store and report power analysis information for a block abstract.

This section contains the following topics:

Power Analysis and Derating Factors

Power derating factors affect power values shown in power reports. When calculating power, the derating factor is used as a scalar multiplicative factor. Power analysis results are multiplied by the derating factors. If no power derating factors are specified, the tool assumes a derating factor of 1.0.

The commands to set, reset, and display derating factors are:

- `set_power_derate`
- `reset_power_derate`
- `get_power_derate`
- `report_power_derate`

Feedback

Setting the Power Budget

To set a power budget for a hierarchical cell or collection of cells, use the `set_power_budget` command.

This command stores the power budget attributes and calculates total power for the specified cells and their child cells. Timing updates and activity propagation are also performed to ensure that the total power calculation is accurate.

The total power budget set on a hierarchical instance cannot be smaller than the actual total leakage power for the instance calculated by the tool.

When you set the budget across hierarchy levels, the sum of the budget for the lower-level blocks and the leakage power for the rest of the cells in the block cannot be more than the budget set on the higher-level block.

If you set the power budget on the same cell multiple times, the last `set_power_budget` command overrides the earlier settings. If you do not set a power budget on a cell, the cell inherits the derating factor from its parent.

If you set the budget on a cell and subsequently set a budget on its parent, the child cell's power budget remains the same.

The following topics provide more information:

- Power Budgets and the Derating Factor
- Power Budget Example

Power Budgets and the Derating Factor

Whenever you set a power budget, the tool computes a new derating factor by using the unscaled total power and the specified power budget. That is,

```
derating_factor = power_budget / total_power
```

If you do not set a power budget on a cell, the cell inherits the derating factor from its parent.

If you previously set a derating factor, that derating factor is cleared and recalculated after you set a power budget. If you remove all the power budgets and want to return to your original derating factor, you must reset it using the `set_power_derate` command.

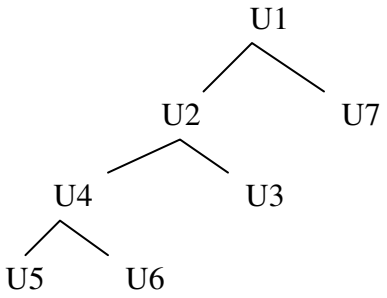
If you set the derating factor after you use the `set_power_budget` command, the power budget value is not affected. To apply a new derating value, you must use the `reset_power_budget` command for the specified cell or block, then set the derating factor with the `set_power_derate` command.

Feedback

Power Budget Example

For the power budget examples in this topic, refer to Figure 23.

Figure 23 Hierarchical Cells and Power Budgets



For the example, assume the following:

- The power of all leaf-level cells is initially set to 10

  U5 = 10, U6 = 10, U3 = 10,
- U4 = 20
- U2 = 30
- U7 = 10
- U1 = 40

If you set the following power budget:

```
fc_shell> set_power_budget -cell {U4} 40
```

The derating (or scaling) factor is 40/(10+10) = 2. The derating factor of 2 is applied on U5 and U6.

The new power is:

- U6 = 10 x 2 = 20
- U5 = 10 x 2 = 20
- U3 = 10 (same)
- U4 = 40 (budgeted)
- U2 = 40 + 10 = 50

- U7 = 10 (same)
- U1 = 50 + 10 = 60

Querying and Reporting Power Budget Values

You can get power budget information that is set on the cells or design in two ways:

- `get_power_budget`

  This command returns the budget set for the design or for specified cells

- `report_power_budget`

  This command displays a report for the power budget set on design objects

Removing Power Budget Values

To remove the power budget on a design or hierarchical instances in the design, use the `reset_power_budget` command. This resets all the stored power budgets, as specified.

For example, to reset the power budget set globally and those set on specific instances in the design for scenario SC1:

```
fc_shell> reset_power_budget -scenario SC1
```

The following command resets the power budget set on cell H2:

```
fc_shell> reset_power_budget -cell [get_cell H2]
```

When you reset a power budget, the calculated derating factor is also removed. To return to a previous derating factor, you have to set it again using the `set_power_derate` command.

A

Miscellaneous

This topic includes the following topic:

See Also

- Correlating Power Analysis Results With PrimePower

Correlating Power Analysis Results With PrimePower

For power analysis to be consistent between the Fusion Compiler tool and the PrimePower tool, you need the following:

- The same design conditions (PVT)
- The same switching activity
- Correct application option settings

For more information about how to set up the design in the PrimePower tool for power analysis, see the *PrimePower User Guide*.

This section covers the following topics:

- Checking Correlation Settings
- Correlation With PrimePower

Checking Correlation Settings

For power analysis correlation, you can check the application options between the Fusion Compiler and PrimePower tools by using the `set_consistency_settings_options` and `check_consistency_settings` commands.

The following example shows how you can set up and perform the check:

```
fc_shell> set_consistency_settings_options -tool PrimePower \
   -exec_path /global/apps/pt_2017.06/bin/pt_shell \
   -script pt.tcl
fc_shell> check_consistency_settings -tool PrimePower
CORR: Launching executable '/global/apps/pt_2017.06/bin/pt_shell'...
```

```
CORR: Collecting correlation settings from executable...
CORR: Processing correlation settings...
################################################
### Report generated by - check_consistency_settings
################################################
Error: The power_enable_analysis PrimeTime-PX setting should be set to
 true. (CORR-800)
Error: The power_scale_internal_arc PrimeTime-PX setting should be set to
 true. (CORR-800)
Error: The power.default_toggle_rate_reference_clock Fusion Compiler
 setting should be set to related. (CORR-812)
Error: The power.table_extrapolation Fusion Compiler setting should be
 set to on. (CORR-812)
Error: The power.scale_leakage_power_at_power_off Fusion Compiler setting
 should be set to true. (CORR-812)
```

Correlation With PrimePower

When you correlate power analysis results with the PrimePower tool, there are three possible ways to get switching activity:

- RTL SAIF file

  PrimePower must read a name-mapping file from the Fusion Compiler tool with tracked netlist name changes.

- Gate-level SAIF file

  If you have a gate-level SAIF file, you do not need a name-mapping file because all the SAIF file objects map to actual netlist objects.

- No SAIF file

  If there is no SAIF file, make sure you set the same user-defined switching activity on both tools using the `set_case_analysis`, `set_switching_activity`, and `infer_switching_activity` commands.

The following is an example of a script correlating power analysis results with the PrimePower tool in the Fusion Compiler tool:

```
# set up design and run power analysis
...
saif_map -write_map output.map -type primepower
write_verilog -include physical_only_cells output.v
# create gate-level SAIF file if needed
write_saif output.saif
```

In the PrimePower tool:

```
# set up design from output.v
set DESIGN [get_object_name [current_design]]
```

```
if {$USE_RTL_SAIF} {
   # load mapping file
     source output.map
   # read RTL SAIF file
     read_saif RTL.saif -strip_path /tb/gut/$DESIGN
} elseif {$USE_GATE_SAIF} {
   # read gate-level SAIF file from IC Compiler II/Fusion Compiler
     read_saif output.saif -strip_path $DESIGN
}
update_power
report_power
```

For more information about performing power analysis, see the *PrimePower User Guide*.