

# **ZeBu<sup>®</sup> Server Power Estimation User Guide**

---

**Version O-2018.09-SP1, June 2019**



## **Copyright Notice and Proprietary Information**

©2019 Synopsys, Inc. All rights reserved. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

## **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 <http://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 Software 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](http://www.synopsys.com)

# Contents

---

<b>Preface.....</b>	<b>9</b>
About This Book .....	9
Audience .....	9
Contents of This Book .....	9
Related Documentation .....	10
Typographical Conventions .....	11
 <b>1. Introduction to Power Estimation .....</b>	 <b>13</b>
<b>1.1. Power Analysis Requirements .....</b>	<b>14</b>
<b>1.2. Software-Driven SoC Power Analysis .....</b>	<b>15</b>
<b>1.3. Power Estimation With Emulation .....</b>	<b>16</b>
 <b>2. Average Power Estimation .....</b>	 <b>19</b>
<b>2.1. Generating ZTDB Data .....</b>	<b>20</b>
2.1.1. Enabling Average Power Estimation in ZeBu Power Compilation.....	20
2.1.1.1. Enabling Word-level Synthesis (WLS) .....	20
2.1.1.2. Enabling SWave Compilation .....	21
2.1.1.3. Declaring Signals for Power Estimation .....	21
2.1.1.4. Enabling Auto-Inlining for Gate-Level Design .....	22
2.1.1.5. Reviewing Gate-Level Libraries For Synthesis.....	22
2.1.2. Emulation Runtime for Average Power Estimation .....	23
<b>2.2. Waveform Reconstruction Using SWave .....</b>	<b>24</b>
<b>2.3. Calculating Average Power.....</b>	<b>30</b>
 <b>3. Cycle Power Estimation .....</b>	 <b>31</b>
<b>3.1. Generating Power Model Using SpyGlass.....</b>	<b>34</b>
3.1.1. Inputs Required By SpyGlass .....	34
3.1.2. Running SpyGlass .....	35
<b>3.2. Enabling Cycle-Power Estimation in ZeBu Power Compilation .....</b>	<b>36</b>
<b>3.3. Emulation Runtime for Cycle Power Estimation .....</b>	<b>37</b>
<b>3.4. Generating a SAIF File for Complete Emulation Run .....</b>	<b>38</b>
<b>3.5. Generating SWave Cycle-by-Cycle Power .....</b>	<b>39</b>
<b>3.6. Viewing Cycle-Power.....</b>	<b>42</b>

<b>4. Reporting Moving Average Cycle Power .....</b>	<b>43</b>
4.1. SpyGlass Rule/Goal and License.....	44
4.2. SpyGlass Design Constraints .....	44
4.3. Calculating the Moving Average .....	45
4.4. Calculating Slope of Moving Average .....	48
4.5. Generating the Candidate Power From the Moving Average .....	49
4.6. Report Summary .....	50
<b>5. Appendix A: SWave UTF Commands .....</b>	<b>51</b>
<b>6. Appendix B: SpyGlass Example Files .....</b>	<b>53</b>
6.1. SpyGlass Project File .....	54
6.2. SpyGlass Design Constraints File.....	56
6.3. Useful Information .....	57

# List of Tables

---

**SWave Commands/Options..... 25**

**SWave Options for Cycle-by-Cycle Power..... 39**





# List of Figures

---

<b>Requirements for Power Analysis .....</b>	<b>14</b>
<b>Critical Power Windows .....</b>	<b>15</b>
<b>Generation of SAIF/FSDB/Cycle-Power .....</b>	<b>17</b>
<b>Generation of SAIF/FSDB/Cycle-Power .....</b>	<b>17</b>
<b>Stages Involved in Average Power Estimation .....</b>	<b>19</b>
<b>SWave Inputs/Outputs .....</b>	<b>24</b>
<b>Cycle-Power Profile Generation.....</b>	<b>31</b>
<b>Cycle-Power Profile.....</b>	<b>32</b>
<b>Processing a Gate-Level Design to Calculate Cycle-Power.....</b>	<b>32</b>
<b>Moving Average Report .....</b>	<b>46</b>
<b>Cycle Power Profiler.....</b>	<b>47</b>
<b>Slope of Moving Average Report .....</b>	<b>48</b>
<b>Candidate Power Report .....</b>	<b>50</b>





## About This Book

The *ZeBu® Server Power Estimation User Guide* describes the power estimation flow and the tools required to estimate the power on a System on Chip (SoC) in emulation.

## Audience

This manual is written for experienced Design engineers to assist them in power estimation using SWave.

These engineers should have experience with the Linux operating system and basic knowledge on power estimation and the components involved in power estimation. In addition, these engineers should have knowledge of the following Synopsys tools:

- ZeBu Server
- Verdi
- SpyGlass

## Contents of This Book

The *ZeBu® Server Power Estimation User Manual* has the following chapters:

Chapter	Describes...
<a href="#">Introduction to Power Estimation</a>	Power estimation and tools required to compute average/cycle-power
<a href="#">Average Power Estimation</a>	Calculating average power
<a href="#">Cycle Power Estimation</a>	Calculating cycle-power
<a href="#">Reporting Moving Average Cycle Power</a>	Calculating moving average of a toggle count
<a href="#">Appendix A: SWave UTF Commands</a>	List of UTF commands used with SWave
<a href="#">Appendix B: SpyGlass Example Files</a>	SpyGlass project example

---

## Related Documentation

Document Name	Description
<i>ZeBu Server 4 Site Planning Guide</i>	Describes planning for ZeBu Server 4 hardware installation.
<i>ZeBu Server 3 Site Planning Guide</i>	Describes planning for ZeBu Server 3 hardware installation.
<i>ZeBu Server Site Administration Guide</i>	Provides information on administration tasks for ZeBu Server 3 and ZeBu Server 4. It includes software installation.
<i>ZeBu Server Getting Started Guide</i>	Provides brief information on using ZeBu Server.
<i>ZeBu Server User Guide</i>	Provides detailed information on using ZeBu Server.
<i>ZeBu Server Debug Guide</i>	Provides information on tools you can use for debugging.
<i>ZeBu Server Debug Methodology Guide</i>	Provides debug methodologies that you can use for debugging.
<i>ZeBu Server Unified Command-Line User Guide</i>	Provides the usage of Unified Command-Line Interface (UCLI) for debugging your design.
<i>ZeBu Server Functional Coverage User Guide</i>	<p>Describes collecting functional coverage in emulation.</p> <p>For VCS and Verdi, see the following:</p> <ul style="list-style-type: none"><li>- Coverage Technology User Guide</li><li>- Coverage Technology Reference Guide</li><li>- Verification Planner User Guide</li><li>- Verdi Coverage User Guide and Tutorial</li></ul> <p>For SystemVerilog, see the following:</p> <ul style="list-style-type: none"><li>- SystemVerilog LRM (2017)</li></ul>
<i>ZeBu Server Power Estimation User Guide</i>	<p>Provides the power estimation flow and the tools required to estimate the power on a System on a Chip (SoC) in emulation.</p> <p>For SpyGlass, see the following:</p> <ul style="list-style-type: none"><li>- SpyGlass Power Estimation and Rules Reference</li><li>- SpyGlass Power Estimation Methodology Guide</li><li>- SpyGlass GuideWare2018.09 - Early-Adopter User Guide</li></ul>
<i>ZeBu Verdi Integration Guide</i>	Provides Verdi features that you can use with ZeBu. This document is available in the Verdi documentation set.
<i>ZeBu Server LCA Features Guide</i>	Provides a list of LCA features available with ZeBu Server.
<i>ZeBu Server Release Notes</i>	Provides enhancements and limitations for a specific release.

---

# Typographical Conventions

This document uses the following typographical conventions:

To indicate	Convention Used
Program code	OUT <= IN;
Object names	OUT
Variables representing objects names	<sig-name>
Message	Active low signal name '<sig-name>' must end with _X.
Message location	OUT <= IN;
Reworked example with message removed	OUT_X <= IN;
Important Information	<b>NOTE:</b> This rule...

The following table describes the syntax used in this document:

Syntax	Description
[ ] (Square brackets)	An optional entry
{ } (Curly braces)	An entry that can be specified once or multiple times
(Vertical bar)	A list of choices out of which you can choose one
. . . (Horizontal ellipsis)	Other options that you can specify



# 1 Introduction to Power Estimation

---

As IC design sizes and frequencies increase, chip power requirements go up. Few years ago, design teams were concerned about power. Now, all mobile application designs and many wall-powered application designs have to consider this constraint.

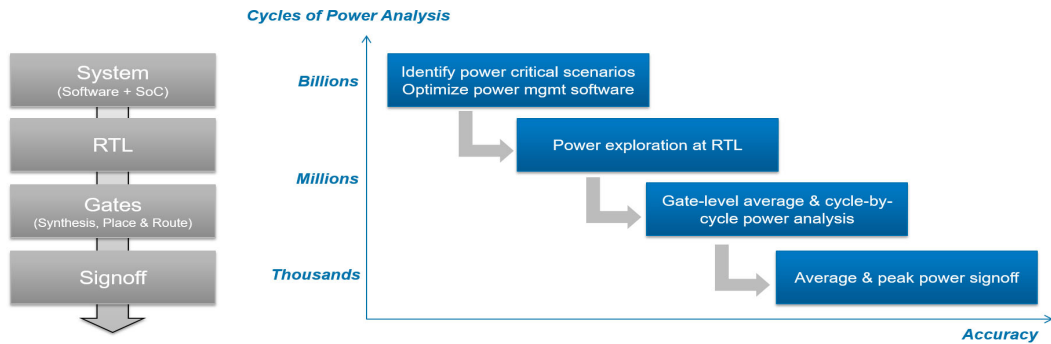
It is important to estimate the chip power utilization of a chip early in the design cycle. If the estimated power consumption is higher than the target, designers can reduce it in a timely manner. Typical power reduction techniques include the use of voltage domains, switchable power domains, and clock gating. Voltage and power domains require special logic circuits and the power grid design is more complicated. Therefore, there is a need to verify that their implementation is correct.

This section describes the following subtopics:

- *Power Analysis Requirements*
- *Software-Driven SoC Power Analysis*
- *Power Estimation With Emulation*

# 1.1 Power Analysis Requirements

There are various power analysis requirements for a System-on-Chip (SoC). The following figure lists the requirements to reduce the power at various stages from System to RTL to Gates to Signoff:

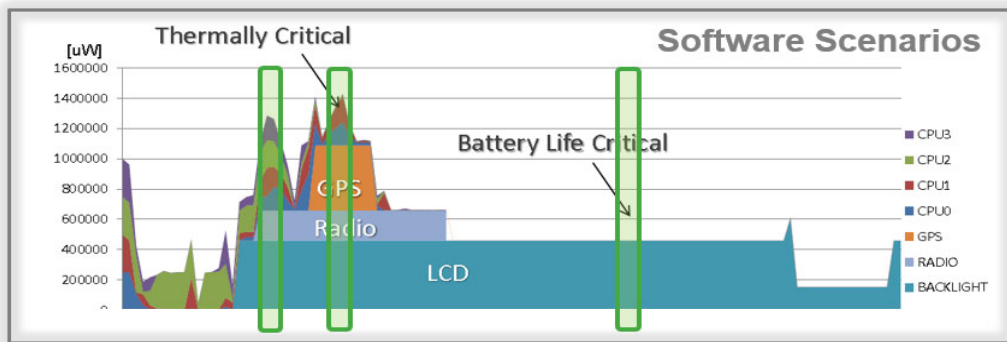


**FIGURE 1.** Requirements for Power Analysis

## 1.2 Software-Driven SoC Power Analysis

In emulation, software-driven SoC power analysis is required to ensure that all power critical modes are covered.

In the software-driven power analysis, run all the software scenarios to identify the critical windows that require detailed power analysis. The following figure shows the software scenarios for a device and highlights the power critical windows.



**FIGURE 2.** Critical Power Windows

ZeBu enables power modeling, activity, analysis, and signoff in the following stages:

- **Activity profiling:** It identifies critical windows over billions of cycles and analyzes power trends due to software workload and hardware architecture interaction.
- **Average power:** It is an average power consumed by a device or a block during a specific test. This is the total energy divided by time. Average power uses Switching Activity Interchange Format (SAIF) files and can be calculated over millions of cycles.
- **Cycle power:** It is an approximate peak power and is a graph of peak power versus time. The cycle power has some loss of accuracy versus signoff power due to the faster runtime. The cycle power can be calculated over millions of cycles.
- **RTL power:** It is the power calculation using the RTL design and target technology library. RTL power tools typically perform fast, approximate synthesis, and then calculate power on the gate-level netlist. The tools handle the mapping from RTL level to gate level. Typically, it is an average power or average power over a set of clock cycles.

- **Power signoff:** It is the power calculation on gate-level netlist with full timing (SDF) delays, and capacitance/layout information (SAIF). Average and Peak power with glitch from timing delays are calculated. Signoff is typically calculated for thousands of cycles.

## 1.3 Power Estimation With Emulation

Activity captured in SAIF or Fast Signal Database (FSDB) files is a key input to tools, such as PrimePower and SpyGlass, which can calculate power. Using an emulator running the design in real use models can be a key to improve the accuracy of these estimates.

To estimate power with accuracy, the following is mandatory:

- SAIF or FSDB file that contains all sequential, combinational, and hierarchical invariant points
- A method to obtain the SAIF or FSDB file from a ZTDB file

Waveform reconstruction is an efficient method to obtain the SAIF or FSDB file from the ZTDB file. Outputting the ZTDB file in a manner that allows for use of parallel processing of the ZTDB can also improve the efficiency.

SWave is a waveform reconstruction tool that also contains an efficient power computation engine to calculate the following:

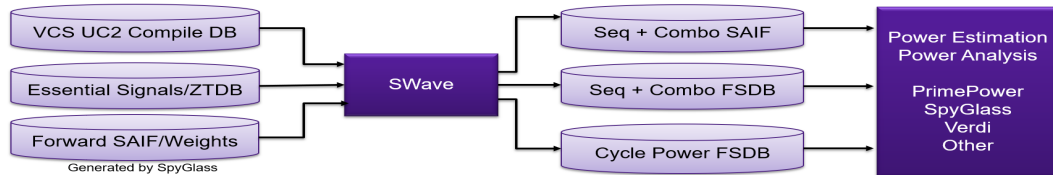
- Combinational SAIF from a sequential data dump for average power
- Combinational FSDB from a sequential data dump for peak power
- A cycle-by-cycle power graph from a sequential dump for a gate-level design

SWave is suitable for processing very large data sets from ZeBu efficiently using multi-threaded jobs on a compute grid.

SWave typically converts the ZTDB to an internal format `swdb` as part of the run. It is possible to create `swdb` separately from SWave using a program `swavedump`. You can run `swavedump` similar to SWave.



The following figure displays inputs to and outputs from SWave.

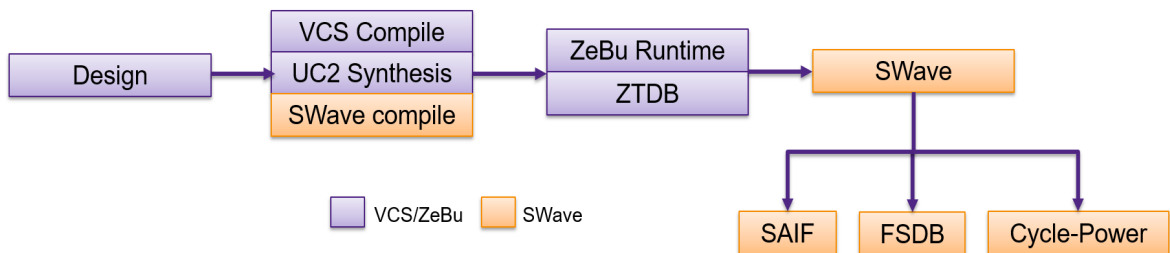


**FIGURE 3.** Generation of SAIF/FSDB/Cycle-Power

To generate SAIF or FSDB using SWave, the following must be performed in ZeBu.

- Enable Word-Level Synthesis (WLS) during compilation  
For details, see [Enabling Word-level Synthesis \(WLS\)](#).
- Enable waveform capture on a portion of the design  
For details, see [Enabling SWave Compilation](#).
- Enable ZTDB slicing during runtime  
For details, see [Emulation Runtime for Average Power Estimation](#).

The following flow chart depicts how a design is processed to generate SAIF/FSDB using SWave.

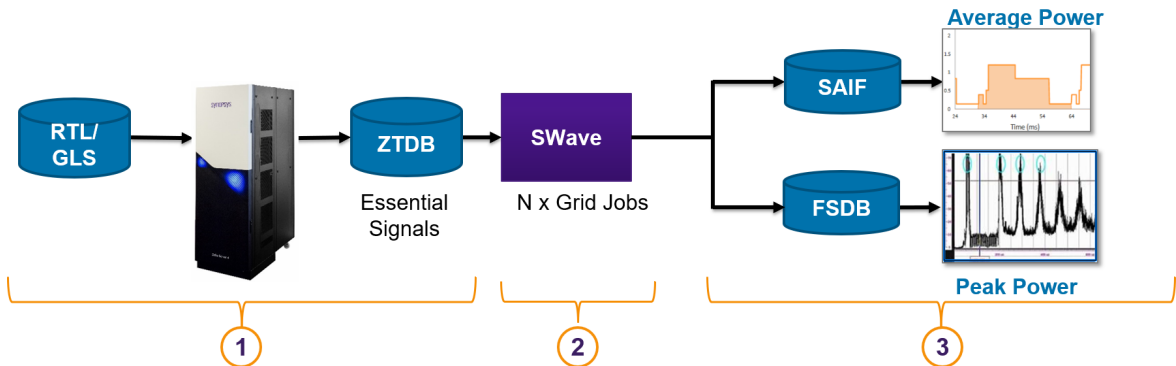


**FIGURE 4.** Generation of SAIF/FSDB/Cycle-Power



## 2 Average Power Estimation

This section describes the calculation of the average power, using RTL SAIF and Gate Level SAIF. There are three phases involved to calculate the average power as displayed in the following figure:



**FIGURE 5.** Stages Involved in Average Power Estimation

The three phases are follows:

- Phase 1: Generate essential signal data in ZTDB format using ZeBu
- Phase 2: Use SWave for waveform reconstruction for all signals and generate SAIF
- Phase 3: Calculate average power for all essential signals

This section describes the following sub-topics:

- [Generating ZTDB Data](#)
- [Waveform Reconstruction Using SWave](#)
- [Calculating Average Power](#)

## 2.1 Generating ZTDB Data

To write waveform for all power signals, perform the following steps:

- [Enabling Average Power Estimation in ZeBu Power Compilation](#): Compile RTL/gate-level design using ZeBu compiler
- [Emulation Runtime for Average Power Estimation](#): Generate ZTDB using size-slicing feature during runtime

### 2.1.1 Enabling Average Power Estimation in ZeBu Power Compilation

To enable average power estimation and compile your RTL design in ZeBu compilation, perform the following:

1. [Enabling Word-level Synthesis \(WLS\)](#)
2. [Enabling SWave Compilation](#)
3. [Declaring Signals for Power Estimation](#)

To enable average power estimation and compile your gate-level design in ZeBu compilation, perform the following steps in addition to the preceding steps:


1. [Enabling Auto-Inlining for Gate-Level Design](#)
2. [Reviewing Gate-Level Libraries For Synthesis](#)


#### 2.1.1.1 Enabling Word-level Synthesis (WLS)

To enable WLS, use the following UTF command:

```
synthesis_preferences -enable_wls true
```

#### Note

 If this feature is enabled by default, adding this command to the UTF does have not negative effects.

 If you are unsure if WLS is enabled or disabled, include this command in the UTF.

### 2.1.1.2 Enabling SWave Compilation

You can enable SWave compilation based on your design.

Power Estimation tools on gate-level designs do not require SAIF or FSDB data inside each gate-level cell. These cells typically have a `celldefine` pragma in the module definition. With the `SWAVE=gl` switch, all signals inside `celldefine` modules are omitted from FSDB or SAIF. This reduces the size of SAIF and FSDB and does not have any impact on the power calculation.

The SWave compilation occurs after the VCS stage. Any errors occurring during the SWave compilation can be found in the log files at: `zcui.work/zCui/log/*swave*`

#### Enabling SWave Compilation For RTL Designs

SWave compilation must identify all sequential elements after synthesis and before the ZeBu system-level compilation stage. To enable SWave for RTL designs, use the following UTF command:

```
debug -waveform_reconstruction_params {SWAVE=enabled}
```

#### Enabling SWave Compilation For Gate-Level Designs

The SWave compilation can identify the sequential elements in parallel with the ZeBu system-level compilation stage. To enable SWave for gate-level designs, use the following command for gate-level designs:

```
debug -waveform_reconstruction_params {SWAVE=gl}
```

### 2.1.1.3 Declaring Signals for Power Estimation

You can control a part of the design, which is written for Power Estimation using QiWC or FWC technologies, with the `$dumpvars()` constructs to generate the ZTDB. The SWave compilation and code-generation occur only for the part of the design listed in `$dumpvars()`. If `$dumpvars()` is excluded, SWave compiles and generates the code for the entire design. A typical design has `dumpvars` included in a portion of the design hierarchy.

For details on QiWC or FWC, see the *ZeBu Server Debug Guide*.

An example usage of the `$dumpvars` command is shown as follows:

```
initial begin : value_set_name_1
    (* qiwc *) $dumpvars(0, top.dut.cpu0);
end
```

**Note**

*Multiple `$dumpvars` named blocks might be added during compilation and can be controlled at runtime with Tcl or C/C++ commands.*

### 2.1.1.4 Enabling Auto-Inlining for Gate-Level Design

To improve capacity when compiling a gate-level netlist in ZeBu, it is recommended to enable auto-inlining.

To enable auto-inlining for your gate-level design, use the following UTF command:

```
# Inline small modules, needed for GLS
optimization -auto_inline_limit 30
```

### 2.1.1.5 Reviewing Gate-Level Libraries For Synthesis

Review your gate-level cell library to check for the following:

- \*IF\* lots of "X"/"Z"
- Delay code is present

If yes, generate a simple cell model from the `.lib` file using Spyglass as follows:

```
$SPYGLASS_HOME/bin/spyglass_lc \  
-gateslib typical.lib \  
-decompile_lib_models \  
-outsglib typical.sglib \  
-wdir gen_behav_model
```

## 2.1.2 Emulation Runtime for Average Power Estimation

To improve efficiency during post-processing of the ZTDB, the ZeBu runtime should be modified to generate ZTDB with size-slicing to create multiple files inside ZTDB. When the size of the uncompressed data received from ZeBu reaches a user-specified size, all data is flushed and all files in the ZTDB directory are closed. A new set of files is then opened and the emulation continues. Each file inside ZTDB has `cycle_XXX` in the filename to indicate the cycle number when this slice started.

The result of size-slicing is each set of files in ZTDB has approximately the same size, independent of the following:

- ZeBu system used during the runtime
- Speed of the host-system attached to ZeBu
- Network congestion or disk I/O speed
- Activity during the test (reset or heavy data phase)

Each size-sliced ZTDB file is performed as a separate job on the grid for waveform reconstruction, by:

- Managing the memory that each job requires
- Optimizing the grid resources and turnaround time

The size-slicing is specified in MB. The size-slicing is enabled through the following commands/functions/methods.

### zRun Tcl Command:

```
ZEBU_Fwc_setRegularZtdbDumpSizeInterval <slice size>
```

### C API:

```
$ZEBU_ROOT/include/FastWaveformCapture.h
ZEBU_FastWaveformCapture_ztdbDumpSizeInterval(<slice size>);
```

### C++ API:

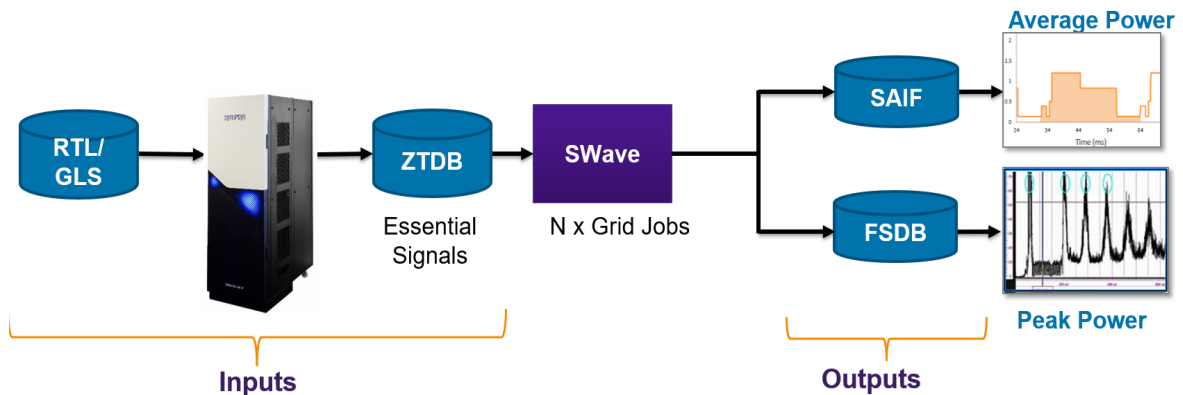
```
$ZEBU_ROOT/include/FastWaveformCapture.hh:
SetRegularZtdbDumpSizeInterval(<slice size>);
```

Start with a `<slice size>` of 2000 (MB) and then adjust up or down as needed depending on the design size and grid configuration.

## 2.2 Waveform Reconstruction Using SWave

SWave reads the ZTDB waveform data, performs waveform reconstruction to generate SAIF or FSDB files, and runs multiple jobs on the grid. By default, SWave runs one job per ZTDB slice.

The following figure displays the inputs (ZTDB signals) and outputs (SAIF or FSDB) of SWave.



**FIGURE 6.** SWave Inputs/Outputs

For more information, see the following subsections:

- [Options to Generate SAIF/FSDB Using SWave](#)
- [Error Conditions and Troubleshooting](#)

### Options to Generate SAIF/FSDB Using SWave

The following table lists the options to generate SAIF/FSDB using SWave:



**TABLE 1** SWave Commands/Options

Options	Description
<code>-work zcui.work</code>	Specifies the path to the <code>zcui.work</code> directory. A full pathname or relative pathname might be specified. <b>Default:</b> None
<code>-i &lt;name&gt;.&lt;ztdb   swdb&gt;</code>	Reads the waveform capture from the <code>.ztdb</code> directory or <code>swdb</code> from the <code>.swdb</code> directory and transmits data into SWave. The file format is determined based on the extension/contents. The valid extensions are <code>.ztdb</code> or <code>.swdb</code> . <b>Default:</b> None
<code>-fsdb filename</code>	Generates an FSDB file containing all combinational signals. A filename must be specified. If there is one ZTDB slice, a standard FSDB file is generated. If the ZTDB has multiple slice, the filename has a <code>.vtf</code> extension and it is a virtual-file with time-slice.
<code>-saif filename</code>	Generates a SAIF capture containing all combinational signals. A filename must be specified. The file format is <code>.gz</code> unless the <code>gzip</code> level is set to 0 with <code>.gz_level</code> switch. A single file is generated even if the ZTDB file has more than one slice.
<code>-fullsaifdump [ true   false ]</code>	Controls printing of TX and IG items in the SAIF file. As ZeBu does not model X, both these values can be zero in the SAIF file. <b>Note:</b> When it is set to false, it does not print TX or IG in the SAIF file. <b>Default:</b> False.
<code>-log directoryname</code>	Executes waveform reconstruction inside "directoryname", which is internally <code>split_N</code> sub-directories for each additional run that is executed. Places all SWave log files in this directory.

**TABLE 1** SWave Commands/Options

Options	Description
<code>-gziplevel 1</code>	<p>Generates the log and SAIF files with a <code>gzip</code> compression setting. If you set this option to 0, it disables <code>gzip</code> and the SAIF and log files are not compressed.</p> <p><b>Note:</b> SAIF files are significantly smaller (often 10X smaller) when <code>gzip -1</code> is used. PrimeTime and other power tools can generally read the SAIF files using the <code>.gz</code> file format.</p> <p><b>Default:</b> <code>gzip -1</code></p>
<code>-start cycle</code>	<p>Starts the waveform reconstruction at a specific start cycle.</p> <p><b>Default:</b> Begin at the first cycle of the ZTDB file.</p>
<code>-end cycle</code>	<p>Stops the waveform reconstruction at a specific end cycle.</p> <p><b>Default:</b> End at the last cycle of the ZTDB file.</p>
<code>-starttime time</code>	<p>Starts the waveform reconstruction at a specific start time. The time must include a time unit (example: 1000 ns).</p> <p><b>Default:</b> Begin at the first timestamp in the ZTDB file.</p>
<code>-endtime time</code>	<p>Stops the waveform reconstruction at a specific end time. The time must include a time unit (example: 1000 ns).</p> <p><b>Default:</b> End at the last timestamp in the ZTDB file.</p>
<code>-timescale &lt;integer&gt;&lt;timeunit&gt;</code>	<p>Specifies how many nsec/psec per ZeBu cycle if the design has a single clock and RTL clocks are not used during ZeBu compilation and runtime.</p> <p>If RTL clocks are not used, this switch is required.</p> <p>If RTL clocks are used, this switch is ignored and has no effect.</p> <p><b>Default:</b> 1 ns</p>

**TABLE 1** SWave Commands/Options

Options	Description
<code>-aligntimescale</code>	<p>Creates FSDB with 1/10/100 ns/ps/fs timescale. All waveforms/SAIF values are scaled to one of these unit values. If the half-period of a clock is 5 ns, the timescale is 1 ps and all values are scaled by 2500. This is required for power tools (including PrimePower and Power-Replay) because they only accept 1/10/100 ns/ps/fs timescale SAIF or FSDB. However, Verdi does not require this setting.</p> <p><b>Default:</b> Scaling is not applied.</p>
<code>-hierpath instance_path</code>	<p>Performs the waveform reconstruction on a part of the design, which is specified as an instance path.</p> <p><b>Default:</b> The entire design present in the ZTDB file (controlled with value-set groups or the <code>\$dumpvars</code> command during ZeBu step) is reconstructed.</p>
<code>-hierfile filename</code>	<p>Performs the waveform reconstruction on a part of the design, which is specified as a control file. The example of the control file follows:</p> <pre># Example to reconstruct part of the design # All of CPU0 except L2 cache # 2 levels of logic for CPU1 -i top.cpu0 -x top.cpu0.L2cache -i top.cpu1 -d 2</pre>
<code>-force filename</code>	<p>Forces/overrides a waveform during reconstruction. You must specify a constant value or the entire waveform. The waveform is specified in a control file, with {time, value} pairs.</p> <pre># Example: override these signal waveforms top.scan_en 0 top.reset {0 0}, {100, 1}</pre>

**TABLE 1** SWave Commands/Options

Options	Description
<code>-jobs N</code>	<p>Submits N waveform reconstruction jobs to grid.</p> <p>If the number of ZTDB slices is less than N, the number of jobs submitted is the number of slices in the ZTDB.</p> <p>If the number of ZTDB slices is greater than N, N jobs are submitted to the grid.</p> <p>Several slices can be merged to form N jobs. However, slices are not split into additional pieces/jobs.</p> <p><b>Default:</b> Launch one job per ZTDB slice</p>
<code>-grid command</code>	<p>Launches the waveform reconstruction jobs using the grid command in parallel.</p> <p>Without this switch, all jobs are executed on the local host.</p>
<code>-gridmaxjobs N</code>	<p>Launches N jobs to the grid at one time. If the number of jobs required for this run is greater than N, N jobs are launched initially and then a new job is submitted when each running job completes. This is useful if grid rules only allow a limited number of jobs per user.</p> <p><b>Default:</b> Launch all jobs at one time.</p>
<code>-griddelay N</code>	<p>Delays the submission of each grid job by N seconds. This is useful if the grid rules do not allow many jobs to be submitted at one time.</p> <p><b>Default:</b> No delay.</p>
<code>-pthreads [count   max]</code>	<p>Uses parallel threads during the waveform reconstruction. A specific number (<code>count</code>) or "<code>max</code>" launches jobs with the number of physical CPU available on the machine.</p> <p><code>Max</code> is useful for compute grids that are non-uniform (mix of 8, 16 and 32 processor machines for example).</p> <p><b>Default:</b> Use 8 threads.</p>

**TABLE 1** SWave Commands/Options

Options	Description
<code>-saifinsts inst_list.txt</code>	Generates multiple SAIF files for different hierarchies during a single SWave run. The Waveform reconstruction occurs once, and multiple SAIF files are written. One SAIF file is written for each instance/filename pair listed in the <code>inst_list.txt</code> file. An example <code>inst_list.txt</code> file is shown in the following: <code>cpu0.saif top.cluster.cpu0</code> <code>cpu1.saif top.cluster.cpu1</code>
<code>-redo</code>	Reruns any failed bundles in the latest <code>split_N</code> directory. This is useful if the grid killed any jobs or any jobs failed for an intermittent reason. Only failed bundles are relaunched. The bundles that passed are not run again.

**Error Conditions and Troubleshooting**

Each run of SWave occurs inside a `split_N` directory. Each time-slice is processed inside a `work.bundle_N` directory.

To find details of a job that is failed or an error occurred, browse to the `work.bundle_N` directory.

## 2.3 Calculating Average Power

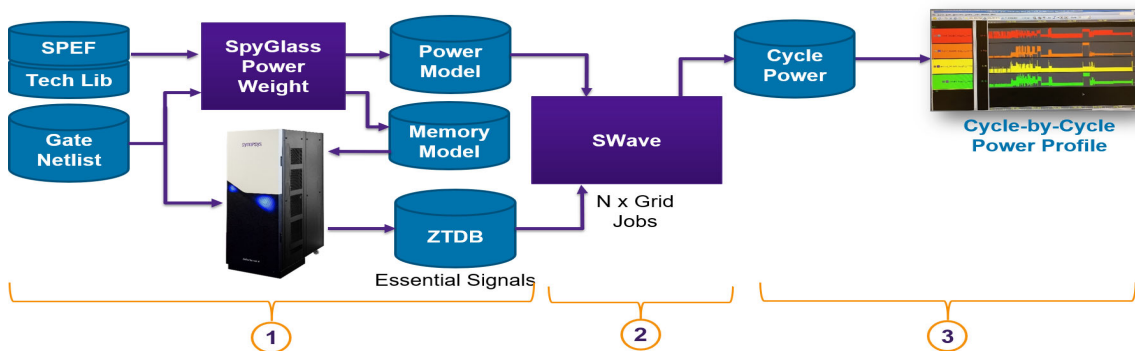
To calculate the average power, use standard SAIF file generated by SWave. The SAIF file can be read into power tools like PrimePower or SpyGlass.

### 3 Cycle Power Estimation

In emulation power analysis, analyzing the peak power is difficult because the emulator generates large amounts of activity data.

To ease the peak power analysis in emulation, generate a cycle-power profile from the ZTDB created by ZeBu using SWave. The cycle-power profile can be used to find time windows and to analyze it using power estimation tools such as PTPX or SpyGlass.

The following figure displays the phases involved in generating the cycle-power profile:

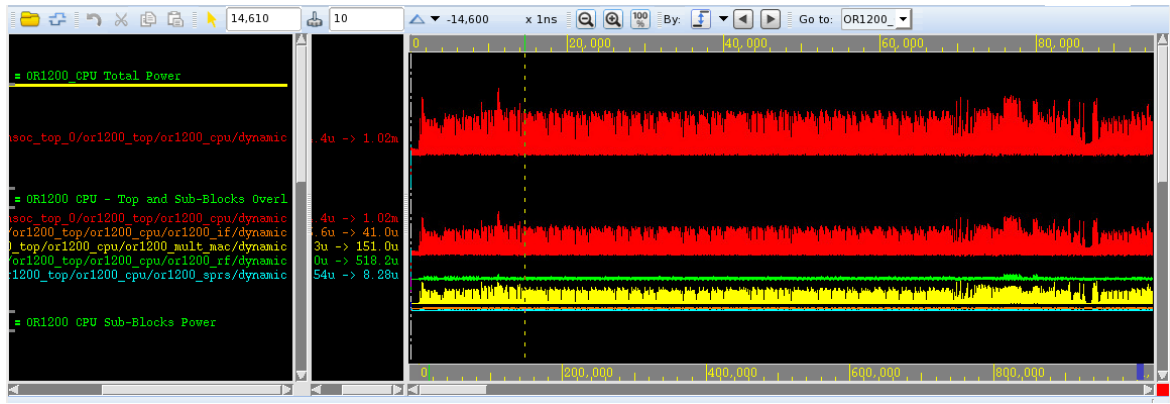


**FIGURE 7.** Cycle-Power Profile Generation

The phases are as follows:

- **Phase 1:** Compile design with SpyGlass and generate power model
- **Phase 2:** Write waveform for all power essential signals and generate ZTDB
- **Phase 3:** Use SWave for waveform reconstruction for all signals and apply power model
- **Phase 4:** Identify peak power windows for power signoff

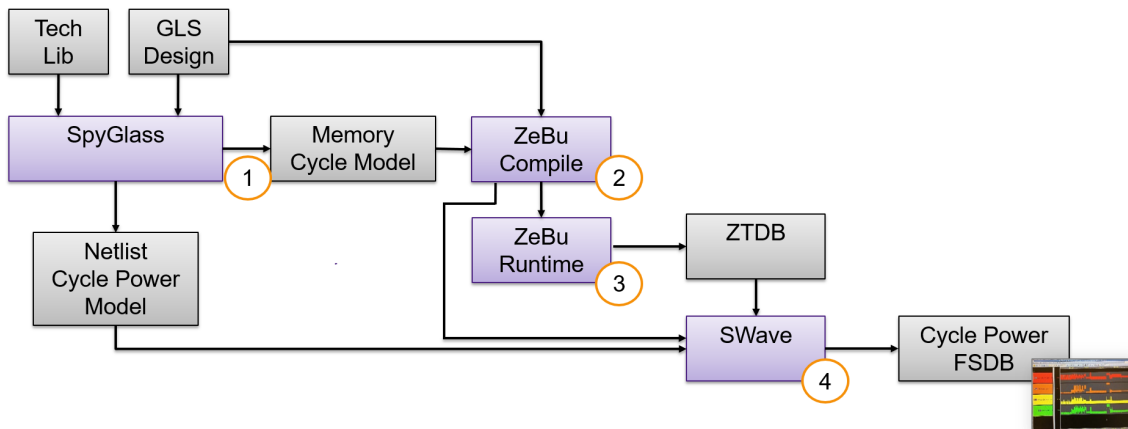
The following figure displays a graph that displays the dynamic power waveform of a design:



**FIGURE 8.** Cycle-Power Profile

In this figure, peaks in the profile represent dynamic power peaks in the design. An FSDB file containing all design nets can be extracted for a small time-range and analyzed using a power estimation tool, such as PrimePower or SpyGlass.

The following flow chart depicts how a design is processed to calculate cycle power.



**FIGURE 9.** Processing a Gate-Level Design to Calculate Cycle-Power

1. SpyGlass Model Generation
2. ZeBu Compilation



3. ZeBu Runtime
4. Power Calculation

This section describes the following subtopics:

- [\*Generating Power Model Using SpyGlass\*](#)
- [\*Enabling Cycle-Power Estimation in ZeBu Power Compilation\*](#)
- [\*Emulation Runtime for Cycle Power Estimation\*](#)
- [\*Generating a SAIF File for Complete Emulation Run\*](#)
- [\*Generating SWave Cycle-by-Cycle Power\*](#)
- [\*Viewing Cycle-Power\*](#)

## 3.1 Generating Power Model Using SpyGlass

SpyGlass is used to create a light weight power model of the design, which is used on the fly by SWave to compute power. This power model generation is split into two steps and is executed as two goal runs to reduce the overall turn around time, as follows:

1. Generate a power model for memories in your design using the `power_factor_conditions` goal.  
**NOTE:** *If the design hierarchy of interest has no memories, this step can be skipped.*
2. Generate a power model for rest of the design (except memories) using the `power_factor_values` goal and compute the power factors and memory weights during the cycle-power flow.

For details on *GuideWare/2018.09-EarlyAdaptor* methodology, see the *GuideWare2018.09 - Early-Adopter User Guide*.

### Note

*Typically, the `power/power_factor_conditions` goal is run first because this data is needed prior to the ZeBu compilation.*

*The `power/power_factor_values` goal is used during the SWave run and can run in parallel with the ZeBu compilation.*

### 3.1.1 Inputs Required By SpyGlass

You need to provide the following inputs to SpyGlass to generate the power factors and memory weights:

- The gate-level netlist used for ZeBu
- The tech libs (in liberty (.lib) format) used by backend tools for cells, memory models, and so on.
- The SPEF file for parasitic information
- Top-level clock name and period
- In the SGDC file, make sure you specify the fastest clock in the design. Specify the clock SGDC command, as shown in the following example:

```
clock -name "clk" -period 10
```

**NOTE:** *Time period is specified in nanoseconds. For details on the clock constraint specification in SpyGlass, see the Appendix:SpyGlass Design Constraints application*

*note.*

These inputs are provided to SpyGlass in a control file known as a project file or in the Tcl format. The project file selects the methodology source and goal configuration through the parameter settings.

An example of a project file is provided in [Appendix B: SpyGlass Example Files](#).

## 3.1.2 Running SpyGlass

To run SpyGlass, use the following command:

```
spyglass -project <projectfile>.prj -goal <goal_name> -batch
```

Where,

- **-project:** Defines the SpyGlass project file
- **-goal:** Specifies the goal to run (power/power\_factor\_conditions or power/power\_factor\_values)
- **-batch:** Invokes SpyGlass to run in batch mode

You can find results of the SpyGlass run in the SpyGlass results directory:

<project>/<top>/<goal>/spyglass\_reports/power\_est

After the SpyGlass run is complete, the following key files are produced:

- **pe\_power\_profiler\_memory\_conditions.db**  
This is the memory power conditions file produced by the power/power\_factor\_conditions goal. This file is used during SWave compilation (as part of ZeBu compilation).
- **pe\_power\_profiler\_weight\_file\_for\_SWave.db**  
These are the power factors produced by the power/power\_factor\_values goal. This file is used by SWave when calculating the cycle-power profile.
- **pe\_power\_profiler\_debugip\_list.txt**  
It is an optional file produced by the power/power\_factor\_values goal when the pe\_toggle\_activity\_bucket\_debugIP\_support SpyGlass parameter is set to Yes. The pe\_power\_profiler\_debugip\_list.txt file

indicates which lower-level instances can be reported in the cycle-power profile. Without this file, only the top-level instance is reported.

This file must be available in the same directory as the `pe_power_profiler_weight_file_for_swave.db` file and might be edited to adjust which instances are selected.

## 3.2 Enabling Cycle-Power Estimation in ZeBu Power Compilation

To enable cycle-power estimation during the compilation of your gate-level design in ZeBu, perform the following:

- [Adding SWave Compilation Command to UTF](#)
- [Declaring Signals for Cycle-Power Estimation Using QiWC or FWC](#)
- [Reading the Forward SAIF File](#)
- [Reading the Memory Power Models](#)

### Adding SWave Compilation Command to UTF

To enable the SWave compilation and indicate the compilation is on a gate-level design during the ZeBu Compilation, add the following command in the UTF file:

```
debug -waveform_reconstruction_params {SWAVE=gl}
```

### Declaring Signals for Cycle-Power Estimation Using QiWC or FWC

Ensure your design has QiWC or FWC included to provide the activity information required by SWave to compute the cycle-power. If it is not already present in your design, declare QiWC or FWC in your design as follows:

#### ■ QiWC declaration:

```
initial begin : name_here  
    (* qiwc *) $dumpvars(0, <top_instance_path>);
```

#### ■ FWC declaration:

## Emulation Runtime for Cycle Power Estimation

```
initial begin : name_here  
    (* fwc *) $dumpvars(0, <top_instance_path>);
```

## Reading the Forward SAIF File

Optionally, for average power, a forward SAIF file might be included for the memories. To include a forward SAIF file, add the following to your `vcs` command:

```
-swave=forwardsaiffile:<file>.saif
```

**Note**

*SWave does not support the use of forward SAIF files for combinatorial library cells.*

## Reading the Memory Power Models

The ZeBu compilation must read the memory and power models generated by SpyGlass. It is mandatory for accurate calculations when memories are present. To inform the compilation to include the memory and power models, add the following command in the UTF file:

```
-swave=spgcompilefile:<path>/pe_power_profiler_memory_conditions.db
```

# 3.3 Emulation Runtime for Cycle Power Estimation

To efficiently post-process the ZeBu ZTDB capture, it is recommended to perform size-slicing when generating the ZTDB.

For details, see [Emulation Runtime for Average Power Estimation](#) in the [Average Power Estimation](#) chapter.

## 3.4 Generating a SAIF File for Complete Emulation Run

To generate a SAIF file for a complete emulation run, use the following command:

```
% swave -work zcui.work -i <filename>.ztdb -saif <filename>.saif.gz
```

### 3.5 Generating SWave Cycle-by-Cycle Power

SWave can generate SAIF data and cycle-power data in a single run. The basic command for generating the cycle-power profile is as follows:

```
% swave -work zcui.work\  
-i <ztdb path>\   
-cyclepowerfsdb <name>\   
-cyclepowertext <filename>\   
-cyclepowercategory\  
-spgbucketfile path/pe_power_profiler_weight_file_for_swave.db
```

The following table describes the options used with SWave.

**TABLE 2** SWave Options for Cycle-by-Cycle Power

Options	Description
-work <zcui.work>	Specifies the zcui.work directory from your ZeBu compile. <b>Default:</b> None
-i <name>.<ztdb   swdb>	Reads the waveform capture (.ztdb directory) or swdb capture (.swdb directory) into SWave. The file format is determined based on extension/contents. Valid extensions are .ztdb or .swdb. <b>Default:</b> None
-cyclepowerfsdb <name>	Enables SWave to capture dynamic power versus time graph in the FSDB format. By default, a plot for a top level design is generated. You can generate more plots for sub-hierarchies or power sub-components using other switches. <b>Default:</b> No cycle power FSDB file is generated.
-cyclepowertext <name>	Generates a text report of the cycle-power profile for each slice of the ZTDB. These text reports are found at split_<n>/top/work.bundle_<n>/<name>.txt.gz. By default, do not generate the text reports. <b>Default:</b> No cycle power text file is generated.

**TABLE 2** SWave Options for Cycle-by-Cycle Power

Options	Description
-spgbucketfile <filename>	Specifies the bucket file containing power factor values for power computation. The <code>pe_power_profiler_weight_file_for_swave.db</code> file is created by running the SpyGlass <code>power/power_factor_values</code> goal and is required when generating the cycle-power graph. This is required to calculate the cycle-power profile.
-cyclepowercategory	Enables the dynamic clock power, dynamic logic power, dynamic memory power, and total dynamic power to be computed separate waveforms. The default is to only calculate the total dynamic power. <b>Default:</b> Only calculates the total dynamic power.
-debugippower	Switches to capture cycle-power profiles for sub-blocks defined and enabled in <code>pe_power_profiler_debugip_list.txt</code> generated by SpyGlass. A "1" in the first field after the instance name indicates the instance is enabled.
-scientific	Forces the cycle power text file to be written into an exponential notation. Without this switch, the values in the cycle power text file might be written in the exponential notation or floating point numbers as required for the value. <b>Default:</b> Exponential notation used only as required.



**TABLE 2** SWave Options for Cycle-by-Cycle Power

Options	Description
-cycletogglecount	Causes SWave to compute waveforms for total toggle count in the cyclepowerfsdb file. <b>Default:</b> No total toggle count waveforms are written. <b>Note:</b> This only impacts the cycle power FSDB file. The total toggle counts are not written into the cycle power text file.
-cyclepowerleakage	Enables leakage power calculation with the cycle power FDSDB format. <b>Default:</b> No leakage power is computed. <b>Note:</b> There is no leakage power generated in the cycle power text format. To calculate the leakage power, the SpyGlass parameter <code>pe_toggle_bucket_leakage_modeling</code> must be defined during the SpyGlass run of the power/power_factor_values goal. If the leakage power is not included during compilation, a warning message is reported.

## 3.6 Viewing Cycle-Power

To view the cycle-power data in the FSDB format, launch Verdi using the following command:

```
% verdi -ssf cycle_power.fsdb
```

Analog waveforms for each DebugIP and top-level are present in the waveform file. To view these waveforms, use standard Verdi commands.

# 4 Reporting Moving Average Cycle Power

---

Cycle power generates a power profile for the entire simulation. To help designers identify the window of interests, SpyGlass provides a mechanism to compute the moving average of the graph and detect windows of interest based on peak value and slope.

To report the moving average cycle power, perform the following:

- Calculate the moving average
- Calculate the slope of moving average
- Generate the candidate power from the moving average

This section describes the following subtopics:

- [\*SpyGlass Rule/Goal and License\*](#)
- [\*SpyGlass Design Constraints\*](#)
- [\*Calculating the Moving Average\*](#)
- [\*Calculating Slope of Moving Average\*](#)
- [\*Generating the Candidate Power From the Moving Average\*](#)

## 4.1 SpyGlass Rule/Goal and License

The PEPROFILER02 rule generates power profile moving average and it checks out `power_high_perf` license. The rule is part of the `power_wtc_profiler` goal.

To run this goal, use the following SpyGlass command:

```
% spyglass -project name.prj -goal power/power_wtc_profiler -batch
```

## 4.2 SpyGlass Design Constraints

You need to provide the following details using the `set_moving_average_window` SGDC constraint:

- Hierarchy
- Window size
- Offset
- Slope window size
- Number of candidate

The syntax of SGDC constraints is as follows:

```
set_moving_average_window -hier <hierarchy name> -size <size-value> -  
offset <offset-value> -window <window-value> -candidate <number-of-  
candidate>
```

where,

- `-hier <hierarchy name>`: Specifies the hierarchy name for which moving window is generated. It is a mandatory field.
- `-size <size-value>`: Specifies the window size for which average value is to be calculated. The unit for this field is ns. It is a mandatory field.
- `-offset <offset-value>`: Specifies the value for which the moving average moves. The unit for this field is ns. It is a mandatory field.
- `-window <window-value>`: Specifies the number of average value between which slope is to be calculated. It is an optional field.

## Calculating the Moving Average

- `-candidate <number-of-candidate>`: Specifies the number of power candidates to be generated between minimum and maximum moving average value. It is an optional field.

*The SGDC constraints `set_moving_average_window` should have at least one or both fields of `-window` and `-candidate`.*

In addition to the `set_moving_average_window` constraint, the FSDB file to be analyzed is provided using the `power_profile_data` SGDC constraint.

```
power_profile_data -format <activity file format> -file <activity file
name>
```

where,

- `-format <activity file format>`: is set to `fsdb` to define an FSDB file that is used or to `virt_fsdb` for a virtual FSDB file.
- `-file <activity file name>`: is the name of the FSDB or the `virtual_fsdb` file being analyzed.

## 4.3 Calculating the Moving Average

For a given window size, average cycle power of the specified hierarchy is calculated. This window can be moved using an offset value. The cycle count is converted into a time scale unit and then to nanoseconds for all calculations.

The minimum and maximum window of interest is calculated using the ceiling of start time and the floor of end time.

The following figure shows the cycle power with a time scale unit of 375ps with a window size of 50ns and an offset of 5ns.

```

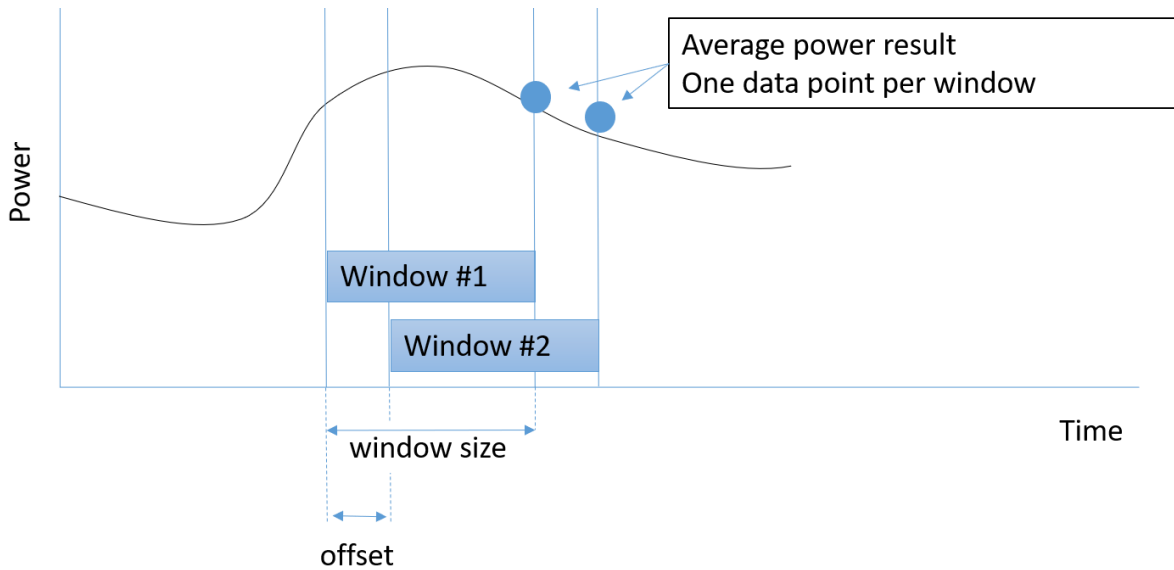
1 Time(ps),unit
2 6114262818,0.0227018
3 6114262822,0.0448769
4 6114262826,0.0445407
5 6114262830,0.0448918
6 6114262834,0.0218527
7 6114262838,0.0440025
8 6114262842,0.0460269
9 6114262846,0.0240679
10 6114262850,0.0473827
11 6114262854,0.0473685
12 6114262858,0.0612092
13 6114262862,0.0250604
14 6114262866,0.0724748
15 6114262870,0.0728709
16 6114262874,0.0511386
17 6114262878,0.0760025
18 6114262882,0.0731939
19 6114262886,0.0635522
20 6114262890,0.0253654
21 6114262894,0.0624266
22 6114262898,0.0637607
23 6114262902,0.0388164
24 6114262906,0.0627007
25 6114262910,0.062333
26 6114262914,0.0608197
27 6114262918,0.0251118
28 6114262922,0.0656011
29 6114262926,0.0677609
30 6114262930,0.0403005
31 6114262934,0.0621711
32 6114262938,0.0654849
33 6114262942,0.0597817
34 6114262946,0.0242376
35 6114262950,0.0606954

```

**FIGURE 10.** Moving Average Report

The following figure displays the cycle power and offset in a window size.

## Calculating the Moving Average

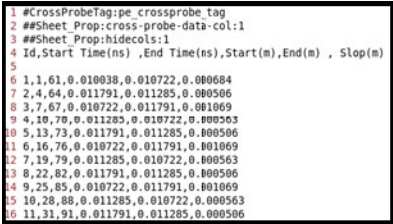
**FIGURE 11.** Cycle Power Profiler

To calculate moving average, consider [Figure 10](#) and perform the following steps:

1. Convert the start cycle count 6114262818 to the time scale unit.  
 $6114262818 * 375 = 2292848556750\text{ps}$
2. Convert from the time scale unit to nanoseconds.  
 $2292848556750 * 0.001 = 2292848556.75\text{ns}$
3. Calculate the ceiling of 2292848556.75 for start time.  
 $2292848557\text{ns}$
4. Compute the average power at 2292848557ns by adding the power between 2292848557ns and 2292848557ns + 50ns and then divide by the count of power values between the window.
5. To calculate the next average power, use the following:  
 $2292848557\text{ns} + 5\text{ns} = 2292848562\text{ns}$  for window 2292848562ns to 2292848562ns + 50ns

# 4.4 Calculating Slope of Moving Average

The slope for each moving average value with respect to a window is calculated. The window is a count specified by you. In this scenario, consider the window value as 50. The following figure report the StartTime, EndTime, Start, End, Slop, and so on.



**FIGURE 12.** Slope of Moving Average Report

The slope for first average power is the absolute difference between the first average power and the 51<sup>st</sup> average power.

The slope for second average power is the absolute difference value of second average power and the 52<sup>nd</sup> average power.

The SpyGlass spread sheet report that contains the start time, end time, start average profile number, end average profile number and absolute difference value, is generated. When you click a row in the spreadsheet, a waveform with start and end time mark opens in **nWave**.



## 4.5 Generating the Candidate Power From the Moving Average

The candidate power number is generated from the moving average number for a given candidate number.

To generate a candidate power number, perform the following steps:

1. Find minimum and maximum moving average. For example, consider the minimum as 21.713 and the maximum as 107.553.
2. To calculate the candidate value use formula:

$$\text{Min} + ((n) * (\text{Max} - \text{Min}) / (N - 1))$$

where, N is the number of candidates and n is the current candidate number.

The value of n range from 0 to N - 1.

3. Select a value from the moving average value, which is closest to the candidate value. The selected value is the sample value and the time corresponding to the selected value is the sample time.
4. Calculate the start cycle count for a candidate.

The start cycle count for a candidate is the floor of the sample time divided by the time scale value. For example, consider the time scale value is 375ps and the sample time is 2292866952ns. The start cycle count is 6114311872.00  $((2292866952 * 1000) / 375)$ .

5. Calculate the end cycle count for a candidate.

The end cycle count for a candidate is the ceil value of the sample time + average window size. For example, consider the time scale value is 375ps, the sample time is 2292866952ns and the window size is 50ns. The end cycle count is 6114312005.00  $((2292866952 + 50) * 1000) / 375)$ .

A spreadsheet report for a specified number of candidate is generated in the following format.

Index	Reference Value(m)	Selected Sample Value(m)	Selected Sample Time(ns)	Start cycle Count	End Cycle Count
1	0.010038	0.010038	1	1000	11000
2	0.010130	0.010038	1	1000	11000
3	0.010222	0.010038	1	1000	11000
4	0.010315	0.010038	1	1000	11000
5	0.010407	0.010722	7	7000	17000
6	0.010499	0.010722	7	7000	17000
7	0.010591	0.010722	7	7000	17000
8	0.010684	0.010722	7	7000	17000
9	0.010776	0.010722	1789	1789000	1799000
10	0.010868	0.010846	1792	1792000	1802000
11	0.010960	0.010846	1792	1792000	1802000
12	0.011053	0.010846	1792	1792000	1802000

**FIGURE 13.** Candidate Power Report

## 4.6 Report Summary

There are four reports generated as follows:

- `pe_moving_avg_<IP_Name>.csv`: Reports the moving average values at specified intervals along with the respective slope values. The number of slope values captured is specified using the SpyGlass parameter "pe\_profiler\_num\_violations" (default value is 50).
- `pe_moving_avg_candidate_<IP_Name>.csv`: Reports the candidate values, and the start/end times of the corresponding windows of interest.
- `pe_moving_avg_slop_<IP_Name>.rpt`: Reports the highest slope values calculated, where the number of values to be captured is specified using the SpyGlass parameter "pe\_moving\_average\_slop\_count" (default value is 5).
- `pe_moving_avg_peak_candidate_<IP_Name>.rpt`: Reports the peak moving average power of the given IP, where the number of values to be captured is specified using the SpyGlass parameter "pe\_moving\_average\_peak\_candidate\_count" (default value is 100).

---

# 5 Appendix A: SWave UTF Commands

---

This section describes examples of using SWave commands for the following:

- *Generating FSDB for Complete Emulation Run*
- *Generating FSDB and SAIF for Cycles on a Specific Instance*
- *Generating FSDB by Forcing Multiple Values on top.reset*
- *Generating FSDB from swdb*

## Generating FSDB for Complete Emulation Run

To generate an FSDB file for a complete emulation run, use the following command:

```
% swave -work zcui.work -i <filename>.ztdb -fsdb <filename>.fsdb
```

---

## Generating FSDB and SAIF for Cycles on a Specific Instance

To generate FSDB and SAIF for cycles from 1000 to 2000 on a specified instance, use the following command:

```
% swave -work zcui.work \  
-i <filename>.ztdb \  
-fsdb <filename>.fsdb \  
-saif <filename>.saif.gz \  
-start 1000 -end 2000 \  
-hierpath <instance_path>
```

---

## Generating FSDB by Forcing Multiple Values on top.reset

To generate FSDB by forcing multiple values on `top.reset`, use the following command:

```
% swave -work zcui.work -i dump.ztdb -fsdb dump.fsdb -forcesignals  
force.txt
```

---

## Force / override Waveform During Reconstruction

You must specify constant value or the entire waveform. The following is an example:

```
# Example: override these signal waveforms
    top.scan_en  0
    top.reset    {0 0}, {100, 1}
```

---

**Default:** None

## Generating FSDB from swdb

To generate FSDB from swdb using 20 GRID jobs, maximum threads per job, and 200G per job, use the following command:

```
% swavedump -help    // Details on how to convert ztdb to swdb format
% swave -work zcui.work -i output.swdb -fsdb output.vf \
    -jobs 20 \
    -grid "qrsh -P gridname -cwd -V -noshell -l mem_free=200G" \
    -pthreads max
```

---

# 6 Appendix B: SpyGlass Example Files

---

This section describes examples of SpyGlass files that can be used with emulation in the following subtopics.

- [SpyGlass Project File](#)
- [SpyGlass Design Constraints File](#)
- [Useful Information](#)

## 6.1 SpyGlass Project File

An example of a SpyGlass project file is given in the following:

```
#!/SPYGLASS_PROJECT_FILE

# Include path
set_option incdir ../verilog/or1200/include

# Analyze/Elab
read_file -type hdl ../verilog/gls_saed32hvt/or1200_cpu.v

# Constraints
read_file -type sgdc or1200_cpu.sgdc

# Tech Library
read_file -type gateslib ../verilog/gls_saed32hvt/
saed32hvt_tt0p85v125c.lib.gz

# Define top-level of Synthesizable DUT here
set_option top or1200_cpu

##Common Options Section
# enable SystemVerilog if required
set_option enableSV yes
# extract additional data required for power from the tech libraries
set_option include_opt_data
# Tell SpyGlass this is a gate level netlist
set_option netlist yes
set_option projectwdir .

# Define where the goals to run come from
current_methodology $SPYGLASS_HOME/GuideWare/2017.03-EarlyAdopter/block/
rtl_handoff
# Configure the Spyglass/SWave Cycle Power Goals

# Goal for calculating Memory items for swave-cycle-cycle-power
current_goal power/power_factor_conditions
set_parameter pe_report_type scientific
```

## SpyGlass Project File

```
# Goal for calculating Design/Gate-Level Design items for swave-cycle-  
cycle-power  
current_goal power/power_factor_values  
set_parameter pe_report_type scientific  
set_parameter pe_include_clkterm_in_clkpower yes  
set_parameter pe_infer_high_fanout_net_bufs no  
### settings for debugIP  
### enable debugIP in selection  
set_parameter pe_toggle_activity_bucket_debugIP_support yes  
### set top level to ZeBu for the debugIP  
set_parameter pe_tb_top_inst_name  
top.dut.minsoc_top_0.or1200_top.or1200_cpu  
### smart selection of ip to include. Without it just include  
instance_trace ip  
set_parameter pe_toggle_activity_bucket_debugIP_support_for_swave_smart  
yes  
### Setting for leakage modeling  
set_parameter pe_toggle_activity_bucket_leakage_modeling yes
```

## 6.2 SpyGlass Design Constraints File

An example of a SpyGlass design constraints file is given in the following:

```
#
# Spyglass Design Constraints (or1200_cpu.sgdc)
#
#=====
# Define design and clock
#=====
current_design or1200_cpu
clock -name "clk" -period 10
#=====
# Define spef file (if available)
#=====
spef_data -file or1200_cpu.spef
#=====
# DebugIP - Sub-blocks for cycle-power
#=====
instance_trace -name if -instname or1200_cpu.or1200_if
instance_trace -name mult -instname or1200_cpu.or1200_mult_mac
instance_trace -name rf -instname or1200_cpu.or1200_rf
instance_trace -name sprs -instname or1200_cpu.or1200_sprs
```



## 6.3 Useful Information

```
=====
<< SPEF >>
[PTPX] read_parasitics ../signoff_data/or1200_cpu.maxTLU_125.spef -path
or1200_cpu
[SPG] spef_data -file ../signoff_data/or1200_cpu.maxTLU_125.spef -modname
or1200_cpu

<< Library>>
[PTPX] set link_library "*" saed32hvt_tt0p85v125c.db"
[SPG] read_file -type gateslib ../verilog/gls_saed32hvt/
saed32hvt_tt0p85v125c.lib.gz

<< Verilog Netlist >>
[PTPX] read_verilog [list ../verilog/gls_saed32hvt/or1200_cpu.v]
[SPG] read_file -type hdl ../verilog/gls_saed32hvt/or1200_cpu.v
=====
```

For details, see the *PrimePower User Guide* and the SpyGlass documentation.

