

Verification Continuum™

ZeBu® Power Aware Verification User Guide

Version S-2021.09-2, October 2023



Copyright Notice and Proprietary Information

©2023 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

Contents

Preface.....	9
About This Book	9
Audience	9
Contents of This Book	9
Related Documentation	10
Typographical Conventions	11
Synopsys Statement on Inclusivity and Diversity	12
 1. UPF Commands Supported by ZeBu.....	 13
 2. ZeBu Power Aware Emulation Flow	 17
2.1. UPF Support in Unified Compile.....	18
2.1.1. Benefits	18
2.1.2. ZeBu Front-End Compilation	19
2.1.3. ZeBu Back-End Compilation Flow	20
2.2. ZeBu Power Aware Compilation	21
2.3. Corruption in ZeBu Power Aware Emulation Flow.....	21
2.3.1. Scrambling	21
2.4. UPF File Example.....	22
2.5. Low Power Optimization and Reporting Commands	26
 3. Emulation Runtime for Power Aware Verification.....	 29
3.1. C++ Interface	30
3.2. Starting Emulation Runtime with Power Aware Verification.....	34
3.3. Initializing the Environment.....	34
3.3.1. initializeRandomizer	35
3.3.2. performRandomizer	35
3.3.3. performRandomizerDomain	36
3.3.4. isPowerManagementEnabled	36
3.3.5. getListOfDomains	37
3.3.6. getPowerDomainState.....	37
3.3.7. getLastTriggeredDomain.....	37
3.4. Managing Retention Strategies	38
3.4.1. enableRetentionStrategy	38

3.4.2. disableRetentionStrategy	38
3.4.3. isRetentionStrategyEnabled	39
3.4.4. getListOfRetentionStrategies	39
3.5. Controlling Power Domains	40
3.5.1. setPowerDomainOn.....	40
3.5.2. setPowerDomainOff	40
3.5.3. setPowerDomainState	41
3.5.4. releasePowerDomain.....	41
3.6. Declaring User Callbacks	42
3.6.1. setPowerDomainOffCallback	42
3.6.2. setPowerDomainOnCallback	43
3.7. Managing Forces and Injections	44
3.8. C++ Example for Power Aware Verification.....	45
 4. Limitations of ZeBu Power Aware Verification	 47
 5. Investigating Power Bugs with Power Aware Verification.....	 49
5.1. Checking Isolation between Power Domains.....	50
5.2. Examining the Power State of Design Instances.....	51
5.3. Examining the State of Power Domains	52
5.4. Retention Control Signals	53
 6. Troubleshooting Power Aware Verification	 55
6.1. Incorrect Order when Calling PowerMgt::init	55
6.2. Deprecated Emulation Runtime Control Methods.....	55
6.3. Failure when Calling any Power Aware Method	56
6.4. Failure When Calling any Retention-Related Method	57

List of Tables

Supported UPF Commands for ZeBu 13
config_upf Command Options 27
C++ API to Control Power Aware Verification: PowerMgt class 32



List of Figures

UPF Flow in UC..... 18

Front-End Compilation Flow 20

Corruption in ZeBu Power Aware Emulation Flow 21

ZEBU_POWER_ON Signal in zSelectProbes..... 51

Retention Strategy Instances in zSelectProbes 53

Retention Strategy Signals in zSelectProbes 53

About This Book

The **ZeBu® Power Aware Verification User Guide** describes how to use Power Aware Verification in ZeBu environment, from the source files to runtime.

Audience

This guide is written for experienced ZeBu users who are familiar with the Unified Power Format (UPF) 2.0, which is described in IEEE 1801-2009 standard. Also, the ZeBu users know how to compile and run a design with a C++ testbench.

Contents of This Book

The **ZeBu® Power Aware Verification User Guide** has the following chapters:

Chapter	Describes...
UPF Commands Supported by ZeBu	List of UPF commands used for Power Aware verification.
ZeBu Power Aware Emulation Flow	ZeBu UPF compilation flow.
Emulation Runtime for Power Aware Verification	ZeBu runtime requirements and methods used for Power Aware verification.
Limitations of ZeBu Power Aware Verification	Limitations of ZeBu Power Aware verification.
Investigating Power Bugs with Power Aware Verification	Ways to investigate issues with Power Aware verification.
Troubleshooting Power Aware Verification	List of errors reported during Power Aware verification and the solution to resolve them.

Related Documentation

Document Name	Description
<i>ZeBu User Guide</i>	Provides detailed information on using ZeBu.
<i>ZeBu Debug Guide</i>	Provides information on tools you can use for debugging.
<i>ZeBu Debug Methodology Guide</i>	Provides debug methodologies that you can use for debugging.
<i>ZeBu Unified Command-Line User Guide</i>	Provides the usage of Unified Command-Line Interface (UCLI) for debugging your design.
<i>ZeBu UTF Reference Guide</i>	Describes Unified Tcl Format (UTF) commands used with ZeBu.
<i>ZeBu Power Aware Verification User Guide</i>	Describes how to use Power Aware verification in ZeBu environment, from the source files to runtime.
<i>ZeBu Functional Coverage User Guide</i>	Describes collecting functional coverage in emulation.
<i>Simulation Acceleration User Guide</i>	Provides information on how to use Simulation Acceleration to enable cosimulating SystemVerilog testbenches with the DUT
<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 Runtime Performance Analysis With zTune User Guide</i>	Provides information about runtime emulation performance analysis with zTune.
<i>ZeBu Custom DPI Based Transactors User Guide</i>	Describes ZEMI-3 that enables writing transactors for functional testing of a design.
<i>ZeBu LCA Features Guide</i>	Provides a list of Limited Customer Availability (LCA) features available with ZeBu.
<i>ZeBu Transactors Compilation Application Note</i>	Provides detailed steps to instantiate and compile a ZeBu transactor.
<i>ZeBu zManualPartitioner Application Note</i>	Describes the zManualPartitioner feature for ZeBu. It is a graphical interface to manually partition a design.
<i>ZeBu Hybrid Emulation Application Note</i>	Provides an overview of the hybrid emulation solution and its components.

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	NOTE: 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

Synopsys Statement on Inclusivity and Diversity

Synopsys is committed to creating an inclusive environment where every employee, customer, and partner feels welcomed. We are reviewing and removing exclusionary language from our products and supporting customer-facing collateral. Our effort also includes internal initiatives to remove biased language from our engineering and working environment, including terms that are embedded in our software and IPs. At the same time, we are working to ensure that our web content and software applications are usable to people of varying abilities. You may still find examples of non-inclusive language in our software or documentation as our IPs implement industry-standard specifications that are currently under review to remove exclusionary language.

1 UPF Commands Supported by ZeBu

UPF commands and options are parsed by VCS.

ZeBu supports the subset of UPF commands listed in the following table.

TABLE 1 Supported UPF Commands for ZeBu

UPF Command	Supported Options
associate_supply_set	-handle supply_set_handle
connect_supply_net	[-ports list]
create_power_domain	[-elements element_list] [-include_scope]
	[-supply {supply_set_handle [supply_set_ref]}*]
create_power_switch	-output_supply_port {port_name [supply_net_name]}
	{-input_supply_port {port_name [supply_net_name]}*}
	{-control_port {port_name [net_name]}*}
	{-on_state {state_name input_supply_port {boolean_function}}*}
	[-off_state {state_name {boolean_function}}*]
	[-domain domain_name]
create_supply_net	[-domain domain_name] [-reuse] [-resolve <>]
create_supply_port	[-domain domain_name] [-direction <>]
create_supply_set	[-function {func_name [net_name]}]*

TABLE 1 Supported UPF Commands for ZeBu

UPF Command	Supported Options
	[-update]
load_upf upf_file_name	[-scope instance_name]
name_format	[-isolation_prefix string]
	[-isolation_suffix string]
set_design_attributes	-models model_list
	[-attribute name value]*
set_design_top	
set_domain_supply_net	-primary_power_net supply_net_name
	-primary_ground_net supply_net_name
set_isolation	-domain ref_domain_name
	[-elements element_list]
	-source source_supply_ref
	-sink sink_supply_ref
	-applies_to
	[-isolation_power_net net_name]
	[-isolation_ground_net net_name]
	[-no_isolation]
	[-isolation_supply_set supply_set_list]
	[-isolation_signal signal_list <>]
	[-isolation_sense {<high low>*}]
	[-clamp_value {< >*}]
	[-location <>]
	[-diff_supply_only <TRUE FALSE>]
set_isolation_control	-domain domain_name
	-isolation_signal signal_name

TABLE 1 Supported UPF Commands for ZeBu

UPF Command	Supported Options
	<code>[-isolation_sense <high low>]</code>
	<code>[-location <>]</code>
<code>set_port_attributes</code>	<code>[-ports {port_list}]</code>
	<code>[{-elements {element_list} <>}]</code>
	<code>[-applies_to <>]</code>
	<code>[-receiver_supply supply_set_ref]</code>
	<code>[-driver_supply supply_set_ref]</code>
<code>set_retention</code>	<code>-domain domain_name</code>
	<code>[-elements element_list]</code>
	<code>[-retention_power_net net_name]</code>
	<code>[-retention_ground_net net_name]</code>
	<code>[-retention_supply_set ret_supply_set]</code>
	<code>[-save_signal {{logic_net <>}}]</code>
	<code>-restore_signal {{logic_net <>}}]</code>
	<code>[-save_condition {{boolean_function}}]</code>
	<code>[-restore_condition {{boolean_function}}]</code>
<code>set_retention_control</code>	<code>-domain domain_name</code>
	<code>-save_signal {{net_name <>}}]</code>
	<code>-restore_signal {{net_name <>}}]</code>
<code>set_scope</code>	

Note

UPF commands not listed in this table are parsed and ignored without displaying error messages.

2 ZeBu Power Aware Emulation Flow

This section describes the ZeBu UPF compilation flow. See the following subsections:

- *UPF Support in Unified Compile*
- *ZeBu Power Aware Compilation*
- *Corruption in ZeBu Power Aware Emulation Flow*
- *UPF File Example*
- *Low Power Optimization and Reporting Commands*

2.1 UPF Support in Unified Compile

The UC flow supports the same UPF syntax, UPF command support, and error messaging as VCS. The same UPF file is used for both simulation and emulation. When compiling for ZeBu, VCS parses and elaborates both the design and UPF files.

The ZeBu front-end compiler generates the EDIF files, containing design and power intent information. The EDIF files and Core Definition Files are processed by **zTopBuild** in the back-end compilation. The Xilinx Place and Route software generates the final bitstream files that are downloaded into an FPGA.

The following figure displays the UPF compilation flow.

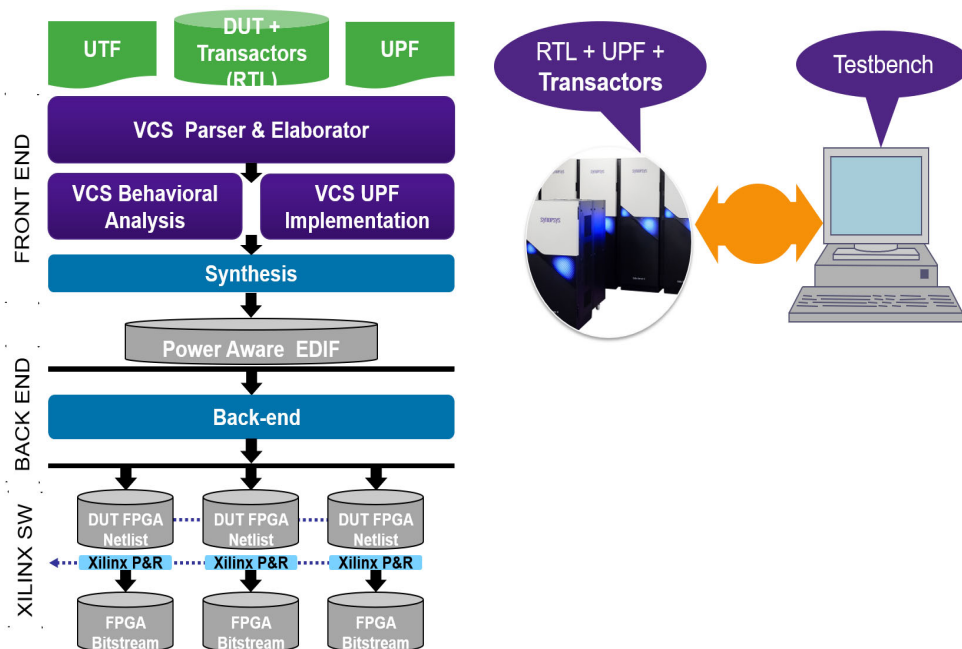


FIGURE 1. UPF Flow in UC

2.1.1 Benefits

The ZeBu UPF compilation flow provides the following benefits:

- Same semantics and analysis like VCS
- A similar set of supported UPF constructs like VCS
- Early error (RTL/UPF) flagging by VCS
- Precedence rules and implementation matching Synopsys cross-tools like, VCS, DC, and ICC
- Integrated debug capability (ZeBu/VCS/Verdi)

2.1.2 ZeBu Front-End Compilation

In the front-end VCS parses and creates the data-model comprising both functional intent and power intent. After VCS execution the design is synthesized into EDIF.

To synthesize for Power Aware Verification, you can choose **zFAST** or **zFAST** Script Mode in the Synthesizer selector of the RTL Group panel of **zCui**.

Note

Other synthesizers are not supported for Power Aware Verification. In addition, it is only possible to synthesize multi-RTL groups when the UPF Script is declared in the top group. UPF Scripts cannot be declared in more than one group.

The following figure displays the front-end compilation flow.

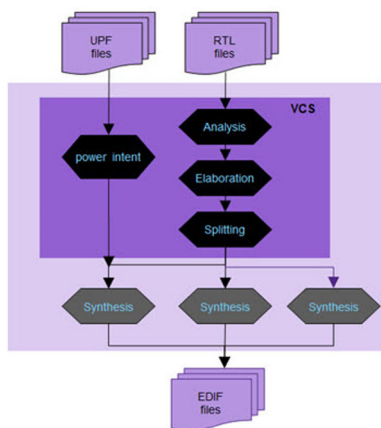


FIGURE 2. Front-End Compilation Flow

Design Files

Power Aware Verification with ZeBu uses the same design source files as used for emulation without Power Aware.

Power Management Script (UPF)

The UPF script describes the topology of the power network. This script is an input to the compiler.

ZeBu supports the following UPF (Unified Power Format) versions:

- UPF 1.0 and UPF 2.0 (IEEE 1801-2009 standard)
- Limited set of commands from UPF 2.1 (IEEE 1801-2013 standard)

ZeBu also supports both hierarchical and non-hierarchical UPF. In the UPF file, design hierarchical paths can be declared and applied using the usual Tcl rules.

2.1.3 ZeBu Back-End Compilation Flow

After front-end compilation, the next steps in ZeBu emulation are back-end compilation and FPGA Place and Route. The EDIF generated by front-end compilation and Core Definition Files are processed by **zTopBuild**. The Xilinx Place and Route software generates the final bitstream files that are downloaded into the FPGAs.

2.2 ZeBu Power Aware Compilation

The UPF file is specified in the VCS command line (or VCS script) with the `-upf` option:

```
% vcs -upf <filename.upf> <vcs_options> <design_files>
```

2.3 Corruption in ZeBu Power Aware Emulation Flow

ZeBu models corruption of shutdown domain by randomizing its outputs and its scrambling internal state elements.

Randomization

As long as a power domain is OFF, its output ports are continuously randomized with pseudo-random values, as illustrated in the following figure:

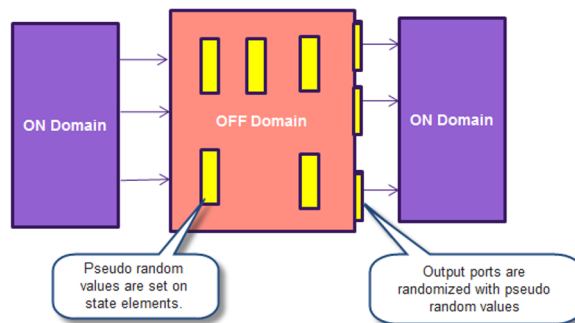


FIGURE 3. Corruption in ZeBu Power Aware Emulation Flow

2.3.1 Scrambling

When a power domain switches on or off, pseudo random values are injected into its internal state elements: registers, latches, and memories. This happens only during power switching - not continuously. Scrambled values are not injected into combinational logic.

2.4 UPF File Example

```
set_design_top top

create_power_domain TOP -elements {} -include_scope
create_power_domain VCC0 -elements {child0 adder0}
create_power_domain VCC1 -elements {child1 adder1}
create_power_domain VCC2 -elements {adder2}

# VCC
create_supply_port VCC_TOP_port -domain TOP
create_supply_net VCC_TOP_net -domain TOP
connect_supply_net VCC_TOP_net -ports VCC_TOP_port

# VSS
create_supply_port VSS -domain TOP
create_supply_net VSS_net -domain TOP
connect_supply_net VSS_net -ports VSS

## Switch output
create_supply_net VCC0_SW -domain VCC0
create_supply_net VCC1_SW -domain VCC1
create_supply_net VCC2_SW -domain VCC2
#####

## Set Domain supplies
#####
set_domain_supply_net TOP \
```

UPF File Example

```

    -primary_power_net VCC_TOP_net \
    -primary_ground_net VSS_net
## Declare that all power sets have a common ground.
create_supply_set VCC0.primary -function {ground
TOP.primary.ground} -function { power VCC0_SW} -update
create_supply_set VCC1.primary -function {ground
TOP.primary.ground} -function { power VCC1_SW} -update
create_supply_set VCC2.primary -function {ground
TOP.primary.ground} -function { power VCC2_SW} -update
create_power_switch VCC0_SWITCH \
    -domain VCC0 \
    -input_supply_port {VCC_TOP_port VCC_TOP_net} \
    -output_supply_port {VCCU_SW VCC0_SW} \
    -control_port {ctrl_sig switch_ctrl_0_reg} \
    -on_state {VCCU_ON VCC_TOP_port {ctrl_sig} } \
    -off_state {VCCU_OFF {!ctrl_sig} }
create_power_switch VCC1_SWITCH \
    -domain VCC1 \
    -input_supply_port {VCC_TOP_port VCC_TOP_net} \
    -output_supply_port {VCCG_SW VCC1_SW} \
    -control_port {ctrl_sig switch_ctrl_1_reg} \
    -on_state {VCCG_ON VCC_TOP_port {ctrl_sig} } \
    -off_state {VCCG_OFF {!ctrl_sig} }
create_power_switch VCC2_SWITCH \
    -domain VCC2 \
    -input_supply_port {VCC_TOP_port VCC_TOP_net} \
    -output_supply_port {VCCG_SW VCC2_SW} \
    -control_port {ctrl_sig switch_ctrl_2_reg} \

```

```
-on_state {VCCG_ON VCC_TOP_port {ctrl_sig} } \  
-off_state {VCCG_OFF {!ctrl_sig} } \  
#-----  
#               set isolation strategies  
#-----  
name_format -isolation_prefix "ISO_PREFIX_"  
  
set_isolation VCC0_isolation -domain VCC0 \  
-isolation_power_net VCC_TOP_net \  
-isolation_ground_net VSS_net \  
-applies_to outputs \  
-clamp_value 0  
  
set_isolation_control VCC0_isolation -domain VCC0 \  
-isolation_signal switch_ctrl_0_reg \  
-isolation_sense low \  
-location self  
  
set_isolation VCC1_isolation -domain VCC1 \  
-isolation_power_net VCC_TOP_net \  
-isolation_ground_net VSS_net \  
-clamp_value 1  
  
set_isolation_control VCC1_isolation -domain VCC1 \  
-isolation_signal switch_ctrl_1_reg \  
-isolation_sense low \  
-location self
```


UPF File Example

```

set_isolation VCC2_isolation -domain VCC2 \
    -isolation_power_net VCC_TOP_net \
    -isolation_ground_net VSS_net \
    -clamp_value Z
set_isolation_control VCC2_isolation -domain VCC2 \
    -isolation_signal switch_ctrl_2_reg \
    -isolation_sense low \
    -location self
#-----
#                               set retention strategies
#-----
set_retention VCC0_retention \
    -domain VCC0 \
    -retention_power_net VCC_TOP_net \
    -retention_ground_net VSS_net \
    -save_signal          { save_sig_0_reg high } \
    -restore_signal       { restore_sig_0_reg high } \

set_retention VCC1_retention \
    -domain VCC1 \
    -retention_power_net VCC_TOP_net \
    -retention_ground_net VSS_net \
    -save_signal          { save_sig_1_reg high } \
    -restore_signal       { restore_sig_1_reg high } \
set_retention VCC2_retention \
    -domain VCC2 \
    -retention_power_net VCC_TOP_net \
    -retention_ground_net VSS_net \

```

```

-save_signal      { save_sig_2_reg high } \
-restore_signal   { restore_sig_2_reg high } \

#-----
#
#               tools option set
#-----
set_design_attributes -attribute {SNPS_reinit TRUE}

```

2.5 Low Power Optimization and Reporting Commands

ZeBu back-end provides multiple low power optimization and reporting commands. You can enable or disable these commands depending on your low power verification requirements. This behavior is specified in the `config_upf` command using **Build System Parameters > Advanced Command File (Advanced > Custom User Files** panel, **System** workspace) option in **zCui**:

```

config_upf [-boundary_output <BOUNDARY_OUTPUT>]
# [-const_propagate_thru_iso] [-constant_vector_port] [-
firmware_lib <FIRMWARE_LIB>] [-hier_sep <HIER_SEP>]
# [-ignore_buffers <IGNORE_BUFFERS>] [-pass_through_const
<PASS_THROUGH_CONST>] [-reg_init <REG_INIT>]
# [-share_randomizers <SHARE_RANDOMIZERS>] [-
tristate_boundary_output <TRISTATE_BOUNDARY_OUTPUT>]
# [-upf_report_inferred_isolation] [-
upf_report_inferred_retention] [-upf_report_spa_srsn]

```

Note

The following options must be set to YES or NO to enable or disable them:

- 📄 ignore_buffers
- 📄 -pass_through_const
- 📄 -share_randomizers

The following table lists the `config_upf` command's options:

TABLE 1 `config_upf` Command Options

Option	Description
<code>-ignore_buffers</code>	Ignores simple continuous assignments on power domain boundary for corruption. By default, this option is enabled.
<code>-pass_through_const</code>	Does not corrupt constants on the power domain boundary. By default, this option is enabled.
<code>-shared_randomizers</code>	Allows sharing of one randomizer for four ports. The randomizer uses a 4-bit serial register. This optimal sharing of randomizer can reduce the randomizer overhead by 75%. Not having a randomizer for each boundary port may reduce the coverage (and hence quality) of corruption. By default, this option is not enabled.
<code>-boundary_output</code>	Defines the value to be driven on boundary output ports of powered-down domains. Possible values are: <ul style="list-style-type: none"> • <code>pseudo_random</code>: Pseudo-random values are continuously driven on the port (default). • <code>0</code>: is driven on the ports. • <code>1</code>: is driven on the ports. • <code>REG</code>: Configurable value to drive on the ports. The value is configured using the <code>-reg_init</code> option.
<code>-constant_vector_port</code>	Specifies to never power off a 0 constant mapped to instance ports.
<code>-firmware_lib</code>	Defines the name of the firmware library provided to the system-level compiler without using the hierarchical path.
<code>-hier_sep</code>	Defines the hierarchical separator to use when parsing requests.

TABLE 1 `config_upf` Command Options

Option	Description
<code>-reg_init</code>	If <code>-boundary_output</code> is set to REG, you can use this option to define the value to be driven on powered-down domains. Possible values are: <ul style="list-style-type: none">• 0: (Default value). This value can be changed during emulation.• 1: This value can be changed during emulation.
<code>-upf_report_inferred_isolation</code>	Creates a report listing all isolation cells.
<code>-upf_report_inferred_retention</code>	Creates a report listing all retention cells.
<code>-upf_report_spa_srsn</code>	Creates a report listing ports to which the <code>set_related_supply_net</code> command is applied.
<code>-verbose_retention</code>	Displays all retention registers.

Note

Optimization commands have a direct impact on the hardware cost.

3 Emulation Runtime for Power Aware Verification

A design compiled with UPF can be emulated at runtime with the same test environment when no Power Aware Verification is required. See the following subsection for more information:

- [*C++ Interface*](#)
- [*Starting Emulation Runtime with Power Aware Verification*](#)
- [*Initializing the Environment*](#)
- [*Managing Retention Strategies*](#)
- [*Controlling Power Domains*](#)
- [*Declaring User Callbacks*](#)
- [*Managing Forces and Injections*](#)
- [*C++ Example for Power Aware Verification*](#)

3.1 C++ Interface

The C++ API methods are provided in the `PowerMgt` class, which is described in the `$ZEBU_ROOT/include/PowerMgt.hh` header file. This API is included in the ZEBU namespace.

For easier implementation, `PowerMgt.hh` header file is automatically available when including the `libZebu.hh` header file.

The `PowerMgt` APIs must be called during emulation runtime in the following order:

1. Anytime during emulation, call `PowerMgt::isPowerManagementEnabled()` to find whether power aware verification is enabled during runtime.
2. To initialize power aware verification during runtime, call `PowerMgt::init`.
3. To enable power aware verification during runtime, call `PowerMgt::enable`.
4. The following APIs must be called before calling `PowerMgt::enable`.
 - a. `PowerMgt::initializeRandomizer`
 - b. `PowerMgt::setForceMode`
 - c. `PowerMgt::setPollingSleepTime`
5. The following APIs must be called after calling `PowerMgt::enable`:
 - a. `PowerMgt::getPowerDomainState`
 - b. `PowerMgt::releasePowerDomain`
 - c. `PowerMgt::getLastTriggeredDomain`
 - d. `PowerMgt::getSupplyState`
 - e. `PowerMgt::enableIsolation`
 - f. `PowerMgt::enableIsolationStrategy`
 - g. `PowerMgt::isIsolationStrategyEnabled`
 - h. `PowerMgt::enableRetention`
 - i. `PowerMgt::enableRetentionStrategy`
 - j. `PowerMgt::isRetentionStrategyEnabled`
 - k. `PowerMgt::setDomainOnPreCallback`
 - l. `PowerMgt::setDomainOnPostCallback`

C++ Interface

- m. `PowerMgt::setDomainOffPreCallback`
 - n. `PowerMgt::setDomainOffPostCallback`
 - o. `PowerMgt::enableSRSN`
 - p. `PowerMgt::getListOfDomains`
 - q. `PowerMgt::getListOfIsolationStrategies`
 - r. `PowerMgt::getListOfRetentionStrategies`
 - s. `PowerMgt::supplyOn`
 - t. `PowerMgt::StartWriteBack/FlushWriteBack`
6. The following APIs must be called after `PowerMgt::init` and after or before `PowerMgt::enable`:
- a. `PowerMgt::supplyOff`
 - b. `PowerMgt::setCorruptionState`
 - c. `PowerMgt::setScramblingState`
 - d. `PowerMgt::getPowerDomainName`
 - e. `PowerMgt::setMultiPowerDomainState`
 - f. `PowerMgt::setPowerDomainState`
 - g. `PowerMgt::setPowerDomainOn`
 - h. `PowerMgt::setPowerDomainOff`
 - i. `PowerMgt::disableIsolation`
 - j. `PowerMgt::disableIsolationStrategy`
 - k. `PowerMgt::disableRetention`
 - l. `PowerMgt::disableRetentionStrategy`
 - m. `PowerMgt::disableSRSN`

This API provides the following methods:



- To start emulation runtime with Power Aware Verification. See [Starting Emulation Runtime with Power Aware Verification](#).
- To initialize the environment. See [Initializing the Environment](#).
- To get information about the design. See [getListOfDomains](#).

- To get information about retention. See [Managing Retention Strategies](#).
- To control the power domains. See [Controlling Power Domains](#).
- To declare user callbacks. See [Declaring User Callbacks](#).
- To manage forces and injections. See [Managing Forces and Injections](#).

For more information on featured example of a testbench for Power Aware Verification, see [C++ Example for Power Aware Verification](#).

Note

All methods described in this section throw an exception if the pointer to the ZeBu board is incorrect. For any other failure, these methods return a Boolean value:

-  *true* for a correct processing.
-  *false* in case of error.

The following table lists the C++ API methods to control power aware verification. These methods are described in this section.

TABLE 1 C++ API to Control Power Aware Verification: `PowerMgt` class

C++ Methods	Description
<code>Init</code>	Starts Power Aware Verification.
<code>Enable</code>	Enables Power Aware Verification.
<code>initializeRandomizer</code>	Initializes the generator that later applies values to registers, ports and memories in one or all power domains.
<code>performRandomizer</code>	Set all elements of the design, whatever their power domain, to pseudo-random values, or 0 or 1 according to the mode selected in <code>initializeRandomizer</code>
<code>performRandomizerDomain</code>	Set all elements of a specific domain to pseudo-random values, or 0 or 1 according to the mode selected in <code>initializeRandomizer</code> .
<code>isPowerManagementEnabled</code>	Checks if the design is compiled to support Power Aware Verification.

TABLE 1 C++ API to Control Power Aware Verification: `PowerMgt` class

C++ Methods	Description
<code>getListOfDomains</code>	Returns a list of all power domains declared in the Power Management (UPF) Script.
<code>getPowerDomainState</code>	Returns the state of a power domain.
<code>getLastTriggeredDomain</code>	Returns the list of power domains that changed state (ON→OFF or OFF→ON).
<code>enableRetentionStrategy</code>	Enables a retention strategy.
<code>disableRetentionStrategy</code>	Disables a retention strategy.
<code>isRetentionStrategyEnabled</code>	Returns information about the retention strategy.
<code>getListOfRetentionStrategies</code>	Returns the list of available retention strategies
<code>setPowerDomainOn</code>	Switches a power domain ON.
<code>setPowerDomainOff</code>	Switches a power domain OFF.
<code>setPowerDomainState</code>	Switches a power domain to the given state.
<code>releasePowerDomain</code>	The domain is no longer controlled from the testbench, but by the design itself.
<code>setPowerDomainOffCallback</code>	Designates a replacement function for the default behavior of the software when a power domain is switched OFF.
<code>setPowerDomainOnCallback</code>	Designates a replacement function for the default behavior of the software when a power domain is switched ON.
<code>setForceMode</code>	Defines the behavior of forces and injections regarding power domain states.

Note

For legibility purposes, `<method_name>` is often used in this chapter in place of `PowerMgt::<method_name>`.

3.2 Starting Emulation Runtime with Power Aware Verification

Power Aware Verification must be started with the following methods in the following order:

1. `init(Board*)`;
2. `enable(Board*)`;

The `PowerMgt::init` method must be called after the `Board::open` and before the `Board::init` methods.

Note

If your design is compiled with a Power Management script, but you do not want to enable power aware verification during runtime, the `init` and `enable` methods can be omitted.

For example:

```
Board* zebu = Board::open(workdir);  
PowerMgt::init(zebu);  
zebu->Board::init();  
PowerMgt::enable(zebu);
```

3.3 Initializing the Environment

The environment is initialized to define the randomizer mode. This section describes the commands to initialize the environment. See the following commands:

- `initializeRandomizer`
- `performRandomizer`
- `performRandomizerDomain`
- `isPowerManagementEnabled`
- `getListOfDomains`
- `getPowerDomainState`
- `getLastTriggeredDomain`

3.3.1 initializeRandomizer

This method initializes the random generator that applies values to registers, ports, and memories in shutdown power domains. Three different modes are available: pseudo-random values (to simulate X values), all-0 values, or all-1 values.

```
bool initializeRandomizer (Board *board, const string &mode, const
unsigned int seedValue) throw(std::exception);
```

where,

- board: Pointer to the ZeBu : :Board object.
- mode: Type of randomization; the following values are supported:
 - ❑ MODE_ZERO: Forces all elements to value 0.
 - ❑ MODE_ONE: Forces all elements to value 1.
 - ❑ MODE_RND: Forces all elements to a pseudo-random value.
 - ❑ seedValue: Integer value to initialize the pseudo-random generator.

Note

All-0 and all-1 values are only applicable to state elements (registers, latches, and memories). They do not apply to the power-domain's interface ports.

3.3.2 performRandomizer

All elements in the design, irrespective of their power domain, are forced to 0, 1, or a pseudo-random value according to the mode specified in `initializeRandomizer`. If the power domain is set to ON or is controlled by the design, the `performRandomizer` has no effect.

```
bool performRandomizer (Board *board) throw(std::exception);
```

where, board is the pointer to the ZeBu : :Board object.

Note

All-0 and all-1 values are only applicable to state elements (registers, latches, and memories). They do not apply to the power-domain's interface ports.

3.3.3 performRandomizerDomain

All elements of a domain are set to pseudo-random values or 0 or 1 according to the mode selected in `initializeRandomizer`. If the power domain is set to ON or is controlled by the design, the `performRandomizer` method has no effect.

```
bool performRandomizer (Board *board, const std::string
&domainname) throw(std::exception);
```

where,

- `board`: Pointer to the `ZeBu::Board` object.
- `domainname`: Name of the domain (reference to object).

Note

All-0 and all-1 values are only applicable to state elements within the domain such as registers, latches and memories.

At runtime, you can get information about the power management status (whether it is enabled or not), current state of power domains, and so on. This section describes the methods to get this information at runtime.

3.3.4 isPowerManagementEnabled

This method checks whether the design is compiled to support Power Aware Verification.

```
bool isPowerManagementEnabled (Board *board, unsigned int
&enabled) throw(std::exception);
```

where,

- `board`: Pointer to the `ZeBu::Board` object.
- `enabled`: Capability to run with power aware verification (reference to object).

3.3.5 getListOfDomains

This method returns a list of all power domains created by the Power Management Script.

```
bool getListOfDomains(Board *board, std::set<std::string>
&domains)
```

where,

- board: Pointer to the ZeBu::Board object.
- domains: List of domains names (reference to board).

3.3.6 getPowerDomainState

This method provides the state of a power domain.

```
bool getPowerDomainState (Board *board, const std::string
&domainname, unsigned int &state) throw(std::exception);
```

where,

- board: Pointer to the ZeBu::Board object.
- domainname: Name of the power domain declared in the Power Management Script
- state: Pointer to the current state of domainname; 1 stands for ON and 0 stands for OFF.

3.3.7 getLastTriggeredDomain

This method retrieves the list of power domains for which the state changed (ON→OFF or OFF→ON).

```
bool getLastTriggeredDomain(Board *board, std::set<std::string>
&triggerChangedDomain);
```

where,

- `board`: Pointer to the `ZeBu::Board` object.
- `triggerChangedDomain`: Names of domains whose power state changed (reference to object).

3.4 Managing Retention Strategies

At runtime, you can turn retention strategies on or off for analysis or performance tuning. This section describes the following methods to manage retention strategies:

- [*`enableRetentionStrategy`*](#)
- [*`disableRetentionStrategy`*](#)
- [*`isRetentionStrategyEnabled`*](#)
- [*`getListOfRetentionStrategies`*](#)

3.4.1 `enableRetentionStrategy`

This method enables the retention strategy.

```
bool enableRetentionStrategy (Board *board, const std::string
&strategyName) throw(std::exception);
```

where,

- `board`: Pointer to the `ZeBu::Board` object.
- `strategyName`: Name of the retention strategy.

3.4.2 `disableRetentionStrategy`

This method disables the retention strategy.

```
bool disableRetentionStrategy (Board *board, const std::string
&strategyName) throw(std::exception);
```

where,

- `board`: Pointer to the `ZeBu::Board` object.
- `strategyName`: Name of the retention strategy.

3.4.3 `isRetentionStrategyEnabled`

This method retrieves information about the retention strategy.

```
bool isRetentionStrategyEnabled (Board *board, const std::string  
&strategyName, bool &enabled) throw(std::exception);
```

where,

- `board`: Pointer to the `ZeBu::Board` object.
- `strategyName`: Name of the retention strategy.
- `enabled`: Capability to see if a retention strategy is enabled.

3.4.4 `getListOfRetentionStrategies`

This method retrieves the list of available retention strategies.

```
bool getListOfRetentionStrategies (Board *board,  
std::set<std::string> &strategies) throw(std::exception);
```

where,

- `board`: Pointer to the `ZeBu::Board` object.
- `strategies`: List of retention strategies sorted in alphabetical order.

3.5 Controlling Power Domains

The methods described in this section provide runtime control for turning power domains ON or OFF. This is another way of testing the low power behavior of the design.

- [*setPowerDomainOn*](#)
- [*setPowerDomainOff*](#)
- [*setPowerDomainState*](#)
- [*releasePowerDomain*](#)

3.5.1 setPowerDomainOn

This method switches a power domain ON (the power domain is no longer controlled by the design).

```
bool setPowerDomainOn (Board *board, const std::string
&domainname) throw(std::exception);
```

where,

- `board`: Pointer to the `ZeBu::Board` object
- `domainname`: Name of the domain (reference to object)

3.5.2 setPowerDomainOff

This method switches a power domain OFF (the power domain is no longer controlled by the design).

```
bool setPowerDomainOff (Board *board, const std::string
&domainname) throw(std::exception);
```

where,

- `board`: Pointer to the `ZeBu::Board` object
- `domainname`: Name of the domain (reference to object)

3.5.3 setPowerDomainState

This method switches a power domain to the given state (the power domain is no longer controlled by the design).

```
bool setPowerDomainState (Board *board, const std::string
&domainname, const unsigned int state) throw(std::exception);
```

where,

- board: Pointer to the ZeBu::Board object
- domainname: Name of the domain (reference to object)
- state: Integer value that specifies the state (0 for OFF, any non-zero value for ON).

Note

■ setPowerDomainState (board, domainname, 1) is equivalent to setPowerDomainOn(board, domainname).

■ setPowerDomainState (board, domainname, 0) is equivalent to setPowerDomainOff(board, domainname).

3.5.4 releasePowerDomain

This method designates the domain to be controlled by design, and no longer by the testbench.

```
bool releasePowerDomain (Board *board, const std::string
&domainname) throw(std::exception);
```

where,

- board: Pointer to the ZeBu::Board object
- domainname: Name of the domain (reference to object)

3.6 Declaring User Callbacks

These methods enable you to override the default behavior of a power domain when it changes state from ON to OFF or vice-versa.

- [setPowerDomainOffCallback](#)
- [setPowerDomainOnCallback](#)

3.6.1 setPowerDomainOffCallback

By default, ZeBu sets all registers/memories and all ports of a power domain to pseudo-random values when the domain is switched OFF.

This method is used to specify a function that overrides the default behavior when a power domain is switched OFF.

Note

Pseudo-random values are applied to ports with a user-callback as well. The user-callback only applies to registers and memories.

```
bool setPowerDomainOffCallback (Board *board, const std::string
&domainname, void (*callback)(void *), void *userData)
throw(std::exception);
```

where,

- `board`: Pointer to the ZeBu::Board object
- `domainname`: Name of the domain (reference to object)
- `callback`: Pointer to the callback function declared by the user
- `userData`: Pointer to the data structure transmitted to the user callback

The default behavior applies when the `setPowerDomainOffCallback()` method is not called by the testbench or when it is called with as null function, as `setPowerDomainOffCallback(board, NULL, NULL)`.

3.6.2 setPowerDomainOnCallback

By default, ZeBu sets all registers and memories of a power domain to pseudo-random values to prevent power-on to restart with a previous (and valid) state.

This method is used to specify a function that overrides the default behavior when a power domain is switched ON.

```
bool setPowerDomainOnCallback (Board *board, const std::string
&domainname, void (*callback) (void *), void *userData)
throw(std::exception);
```

where,

- `board`: Pointer to the `ZeBu::Board` object
- `domainname`: Name of the domain (reference to object)
- `callback`: Pointer to the callback function declared by the user
- `userData`: Pointer to the data structure transmitted to the user callback

The default behavior applies when the `setPowerDomainOnCallback()` is not called by the testbench or when it is called with a null function as `setPowerDomainOnCallback(board, NULL, NULL)`.

3.7 Managing Forces and Injections

Forcing a signal means to set a user-defined value at runtime until explicitly released. Injecting a signal means to set a user-defined value at runtime, which is overwritten by the design at the next write operation. The `setForceMode` method defines the behavior of forces and injections of power domain states:

- If set to 0 (default mode), the power domain state is considered when applying forces and injections.
- If set to 1, forces and injections are applied without considering the power domain states.

```
bool setForceMode(Board *board, unsigned int mode)
throw(std::exception);
```

where, `board` is the pointer to the `ZeBu::Board` object.

3.8 C++ Example for Power Aware Verification

The following example displays a C++ testbench for Power Aware verification after initialization of the ZeBu board:

```
using namespace ZEBU;

// Initialize generator of pseudo-random values
PowerMgt::initializeRandomizer(zebu, "MODE_RND", 42);

// Run the testbench with power methods activated
top_ccosim->run(cycle);

// force the signal top.regs.d2 to the value "2"
unsigned int value2 = 0;
Signal::Force(zebu, "top.regs.d2", &value2);
[...]
Signal::Release(zebu, "top.regs.d3");
top_ccosim->run(cycle);
// display the power domains whose state has changed
std::set<std::string> domains;

PowerMgt::getLastTriggeredDomain(_zebu, domains);

for (std::set<std::string>::const_iterator dit = domains.begin();
     dit != domains.end(); ++dit)
{
    const std::string& domain = *dit;
    std::cerr << "Domain " << domain << " has switched" << std::endl;
}
```


4 Limitations of ZeBu Power Aware Verification

The following features are not supported by the current version of ZeBu Power Aware Verification with:

- Voltage-level (value) shifting - only ON/OFF is supported.
- Force and injection on retained domains when registers are restored.
- Randomization on registers in case of incorrect configuration of the retention control signals.
- Power-related attributes in the design source code.
- UPF commands related to power state tables are parsed and ignored.
- Retention on memory mapped as **zMem** (a tool that infers the synthesizable memory in ZeBu).
- Definitions of tuples (triplets of isolation supply, isolation signal and sense and isolation clamp arguments in an isolation strategy).
- Mapping the design on FPGAs connected to Reduced Latency DRAM (RLDRAM).

5 Investigating Power Bugs with Power Aware Verification

ZeBu provides several means to investigate issues with Power Aware Verification typically found when the design controls the power domains specified in UPF. See the following subsections:

- *Checking Isolation between Power Domains*
- *Examining the Power State of Design Instances*
- *Examining the State of Power Domains*
- *Retention Control Signals*

5.1 Checking Isolation between Power Domains

When an unexpected behavior is observed, the issue may be caused by isolation between power domains. In particular, the pseudo-random values applied to the output ports of an OFF power domain may cause unexpected values on other ON domains not properly isolated. For example, the isolation control is not enabled, or the isolation supply is OFF.

These unexpected values must be investigated upstream to locate the corresponding power domain that is switched OFF.

Once a particular power domain is identified as the root cause for the isolation problem, it can be manually switched OFF using the ZeBu API and then verify the inputs of the other ON domains.

The ZeBu C++ API for power aware verification offers specific methods to manually control the activation of power domains, see [Controlling Power Domains](#).

C++ Method	Description
<code>setPowerDomainOn</code>	Switches a power domain ON (the power domain is no longer controlled by the design).
<code>setPowerDomainOff</code>	Switches a power domain OFF (the power domain is no longer controlled by the design).
<code>setPowerDomainState</code>	Switches a power domain to the given state (the power domain is no longer controlled by the design).
<code>releasePowerDomain</code>	The domain is no longer controlled from the testbench but by the design itself.

5.2 Examining the Power State of Design Instances

You can check whether any design instance is ON or OFF by connecting a dynamic-probe to the ZEBU_POWER_ON signal using `zSelectProbes/zDbPostProc`. This signal is automatically available for any design instance compiled for power aware verification:

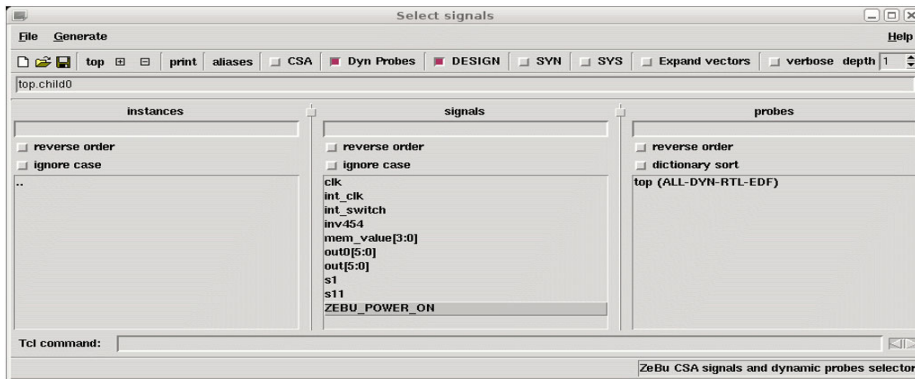


FIGURE 1. ZEBU_POWER_ON Signal in zSelectProbes

5.3 Examining the State of Power Domains

The ZeBu C++ API for Power Aware Verification offers specific methods to check the properties and state of a power domain, see [Controlling Power Domains](#).

C++ Method	Description
<code>isPowerManagementEnabled</code>	Checks whether the design has been compiled for Power Aware Verification.
<code>getListOfDomains</code>	Returns a list of all power domains declared in the UPF Script.
<code>getPowerDomainState</code>	Returns the (ON/OFF) state of a power domain.
<code>getLastTriggeredDomain</code>	Returns the list of power domains whose state changed (ON→OFF or OFF→ON).

In addition, messages are reported in the runtime logs indicating when power domains change state (ON/OFF).

5.4 Retention Control Signals

Retention strategies can be switched ON/OFF by the testbench through the `enableRetentionStrategy` and `disableRetentionStrategy` methods, see [Controlling Power Domains](#).

A dynamic-probe can be connected to a retention strategy using `zSelectProbes`. The corresponding instances and condition signals are displayed in the following figures:

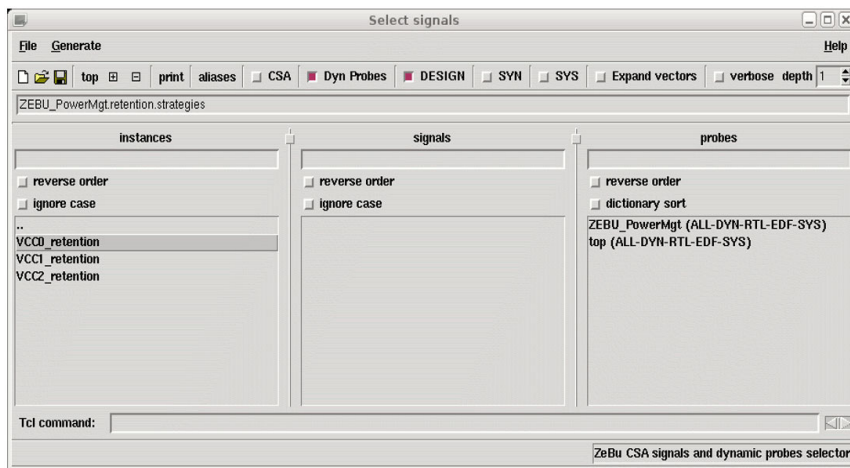


FIGURE 2. Retention Strategy Instances in zSelectProbes

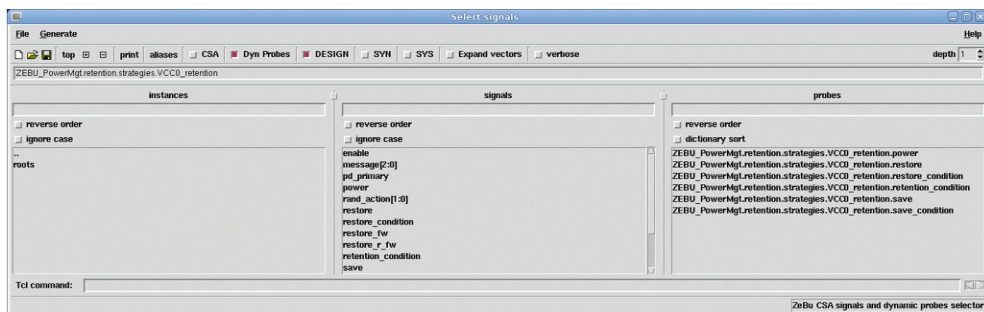


FIGURE 3. Retention Strategy Signals in zSelectProbes

6 Troubleshooting Power Aware Verification

The following sections display the error messages reported in the log file of a Power Aware Verification testbench, and solutions to solve each problem.

- [*Incorrect Order when Calling PowerMgt::init*](#)
- [*Deprecated Emulation Runtime Control Methods*](#)
- [*Failure when Calling any Power Aware Method*](#)
- [*Failure When Calling any Retention-Related Method*](#)

6.1 Incorrect Order when Calling PowerMgt::init

If you do not call the `PowerMgt::init` method before the `PowerMgt::enable` method, the testbench fails with the following error message:

```
-- ZeBu : tb : ERROR : ZHW1027E : Call 'PowerMgt::init' before any  
call to 'PowerMgt::enable'
```

6.2 Deprecated Emulation Runtime Control Methods

The following methods to control emulation runtime for Power Aware Verification are now deprecated.

- `waitDriver` (C equivalent: `ZEBU_PowerMgt_waitDriver` or `ZEBU_PowerMgt_waitDriverEX`)
- `runDriver` (C equivalent: `ZEBU_PowerMgt_runDriver`)

Contact Synopsys support at zebu_support@synopsys.com for further details, if your testbench was previously functional, but now fails with one of the following error

messages:

```
-- ZeBu : tb : ERROR : ZHW1032E : 'waitDriver' call failed: Power Awareness is not in co-simulation mode
```

Or

```
-- ZeBu : tb : ERROR : ZHW1032E : 'runDriver' call failed: Power Awareness is not in co-simulation mode
```

6.3 Failure when Calling any Power Aware Method

If you call any of the methods related to the Power Aware Verification feature (see [C++ Interface](#)) without compiling the design with this feature, the testbench fails with one of the following error messages:

```
-- ZeBu : tb : ERROR : ZHW1031E : 'init' call failed: Power Awareness feature not available
```

Or

```
-- ZeBu : tb : ERROR : ZHW1031E : 'initializeRandomizer' call failed: Power Awareness feature not available
```

It is mandatory that you compile your design with the Power Aware Verification feature before attempting to use any of its features.

6.4 Failure When Calling any Retention-Related Method

If you call any of the methods related to retention strategies without prior definition of the retention strategy in your UPF file, the testbench fails with one of the following error messages:

```
-- ZeBu: tb: ERROR : ZHW1146E : 'getStrategies' call failed. Power  
Retention is not available
```

Or

```
-- ZeBu: tb: ERROR : ZHW1143E : 'isStrategyEnabled' call failed.  
Power Retention is not available
```

Your UPF file must define the given retention strategies in order to call any retention-related method on them.

