



# **Power Aware Verification with ZeBu™ ZeBu Application Note**

**Document revision – a –**

October 2013

**Version 7\_1\_0**

AN034-ZS2

## **Copyright Notice Proprietary Information**

© 2013 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

### **Right to Copy Documentation**

The license agreement with Synopsys permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any. Licensee must assign sequential numbers to all copies. These copies shall contain the following legend on the cover page:

“This document is duplicated with the permission of Synopsys, Inc., for the exclusive use of \_\_\_\_\_ and its employees. This is copy number\_\_\_\_\_.”

### **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.

### **Registered Trademarks (®)**

Synopsys, AMPS, Cadabra, CATS, CRITIC, CSim, Design Compiler, DesignPower, DesignWare, EPIC, Formality, HSIM, HSPICE, iN-Phase, in-Sync, Leda, MAST, ModelTools, NanoSim, OpenVera, PathMill, Photolynx, Physical Compiler, PrimeTime, SiVL, SNUG, SolvNet, System Compiler, TetraMAX, VCS, Vera, and YIELDirector are registered trademarks of Synopsys, Inc.

### **Trademarks (™)**

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at <http://www.synopsys.com/Company/Pages/Trademarks.aspx>.

All other product or company names may be trademarks of their respective owners.

Synopsys, Inc.  
700 E. Middlefield Road  
Mountain View, CA 94043  
[www.synopsys.com](http://www.synopsys.com)

# Table of Contents

<b>ABOUT THIS MANUAL .....</b>	<b>7</b>
OVERVIEW .....	7
INTENDED AUDIENCE .....	7
HISTORY .....	7
RELATED DOCUMENTATION .....	7
<b>1 INTRODUCTION .....</b>	<b>8</b>
1.1 OVERVIEW .....	8
1.2 COMPLIANCE WITH STANDARDS .....	8
1.3 REQUIREMENTS .....	8
1.3.1 <i>License Feature</i> .....	8
1.3.2 <i>ZeBu Compatibility</i> .....	8
1.4 INTERACTIONS BETWEEN POWER-AWARE VERIFICATION AND OTHER FEATURES .	9
1.4.1 <i>SVAs and DPI Calls</i> .....	9
1.4.2 <i>Force and Injection of Signals at Runtime</i> .....	9
1.4.2.1 <i>Force Implementation</i> .....	9
1.4.2.2 <i>Injection Implementation</i> .....	9
1.5 HARDWARE IMPACT OF ISOLATION AND RETENTION FEATURES .....	10
1.5.1 <i>Isolation Implementation in ZeBu</i> .....	10
1.5.2 <i>Retention Implementation in ZeBu</i> .....	10
1.6 LIMITATIONS .....	10
<b>2 SOURCE FILES .....</b>	<b>11</b>
2.1 RTL SOURCE FILES .....	11
2.2 WRITING THE POWER MANAGEMENT SCRIPT FOR ZEBU .....	11
2.2.1 <i>List of Supported UPF Commands for ZeBu</i> .....	11
2.2.2 <i>Detailed Syntax</i> .....	12
2.2.2.1 <i>set_design_top</i> .....	12
2.2.2.2 <i>set_scope</i> .....	12
2.2.2.3 <i>create_power_domain</i> .....	13
2.2.2.4 <i>create_supply_net</i> .....	13
2.2.2.5 <i>create_supply_set</i> .....	14
2.2.2.6 <i>create_supply_port</i> .....	14
2.2.2.7 <i>connect_supply_net</i> .....	14
2.2.2.8 <i>set_domain_supply_net</i> .....	15
2.2.2.9 <i>create_power_switch</i> .....	15
2.2.2.10 <i>set_isolation</i> .....	17
2.2.2.11 <i>set_isolation_control</i> .....	18
2.2.2.12 <i>set_retention</i> .....	19
2.2.2.13 <i>set_retention_control</i> .....	20
2.2.2.14 <i>load_upf</i> .....	20
2.2.2.15 <i>query_isolation</i> .....	21
2.2.2.16 <i>query_isolation_control</i> .....	21
2.2.2.17 <i>query_retention</i> .....	21
2.2.2.18 <i>query_retention_control</i> .....	21
2.2.2.19 <i>query_power_domain</i> .....	22

2.2.3	<i>Additional Command for Attributes Modification</i> .....	22
2.2.4	<i>UPF File Example</i> .....	24
2.3	OPTIONAL RTL-BASED COMPILATION SCRIPT .....	26
<b>3</b>	<b>COMPILING THE DESIGN</b> .....	<b>27</b>
3.1	ZeBU COMPILATION SETTINGS .....	27
3.1.1	<i>Declaring the Source Files and the Power Management Script</i> .....	27
3.1.2	<i>Options for Synthesis</i> .....	28
3.2	COMPILATION FOLLOW-UP .....	28
<b>4</b>	<b>EMULATION RUNTIME</b> .....	<b>29</b>
4.1	INTRODUCTION .....	29
4.2	C++ INTERFACE .....	29
4.2.1	<i>Starting Emulation Runtime with Power Aware Verification</i> .....	30
4.2.2	<i>Initializing the Environment</i> .....	31
4.2.2.1	<i>initializeRandomizer</i> .....	31
4.2.2.2	<i>performRandomizer</i> .....	31
4.2.2.3	<i>performRandomizerDomain</i> .....	31
4.2.3	<i>Getting Information</i> .....	32
4.2.3.1	<i>isPowerManagementEnabled</i> .....	32
4.2.3.2	<i>getListOfDomains</i> .....	32
4.2.3.3	<i>getPowerDomainState</i> .....	32
4.2.3.4	<i>getLastTriggeredDomain</i> .....	32
4.2.4	<i>Managing Retention Strategies</i> .....	33
4.2.4.1	<i>enableRetentionStrategy</i> .....	33
4.2.4.2	<i>disableRetentionStrategy</i> .....	33
4.2.4.3	<i>isRetentionStrategyEnabled</i> .....	33
4.2.4.4	<i>getListOfRetentionStrategies</i> .....	33
4.2.5	<i>Controlling Power Domains</i> .....	34
4.2.5.1	<i>setPowerDomainOn</i> .....	34
4.2.5.2	<i>setPowerDomainOff</i> .....	34
4.2.5.3	<i>setPowerDomainState</i> .....	34
4.2.5.4	<i>releasePowerDomain</i> .....	34
4.2.6	<i>Declaring User Callbacks</i> .....	35
4.2.6.1	<i>setPowerDomainOffCallback</i> .....	35
4.2.6.2	<i>setPowerDomainOnCallback</i> .....	35
4.2.7	<i>Managing Forces and Injections</i> .....	36
4.2.7.1	<i>setForceMode</i> .....	36
4.2.8	<i>C++ Example</i> .....	36
4.3	PLAIN C LANGUAGE INTERFACE .....	37
<b>5</b>	<b>USE MODELS FOR DEBUGGING</b> .....	<b>39</b>
5.1	CHECKING ISOLATION BETWEEN POWER DOMAINS .....	39
5.2	CHECKING THE STATE OF INSTANCES .....	40
5.3	CHECKING THE STATE OF POWER DOMAINS .....	40
5.4	CONTROL SIGNALS FOR RETENTION STRATEGIES .....	41
<b>6</b>	<b>TROUBLESHOOTING</b> .....	<b>42</b>
6.1	INCORRECT ORDER WHEN CALLING POWERMGT : : INIT .....	42

---

6.2	DEPRECATED EMULATION RUNTIME CONTROL METHODS .....	42
6.3	FAILURE WHEN CALLING ANY POWER AWARE METHOD .....	42
6.4	FAILURE WHEN CALLING ANY RETENTION-RELATED METHOD.....	43

## Figures

Figure 1: Signal Behavior with/without the <code>-no_power</code> Option .....	26
Figure 2: Declaration of UPF Scripts in <code>zCui</code> .....	27
Figure 3: <code>ZEBU_POWER_ON</code> Signal in <code>zSelectProbes</code> .....	40
Figure 4: Retention Strategies Instances in <code>zSelectProbes</code> .....	41
Figure 5: Signals of a Retention Strategy in <code>zSelectProbes</code> .....	41

## Tables

Table 1: Supported UPF Commands for ZeBu .....	11
Table 2: C++ API to Control Power Aware .....	29
Table 3: C++ API / C API Equivalence .....	37

# About This Manual

## Overview

This document describes how to use Power Aware Verification in ZeBu environment, from the source files to runtime.

This document is applicable for ZeBu software version 7\_1\_0.

## Intended Audience

This document is intended for experienced ZeBu users who:

- Are familiar with the UPF 1.0 and UPF 2.0 (Unified Power Format) which is described in IEEE 1801-2009 standard.
- Already know how to compile and run a design with a C++ or C testbench.

## History

This table gives information about the content of each revision of this manual, with indication of specific applicable version:

Doc Revision	ZeBu Version	Date	Evolution
a	7_1_0	Oct 2013	First Edition

## Related Documentation

Relevant information is available in the following documents:

- *[ZeBu Compilation Manual](#)*
- *[ZeBu C++ Co-Simulation Manual](#)*
- *[ZeBu C++ API Reference Manual](#)*
- *[ZeBu C API Reference Manual](#)*

# 1 Introduction

## 1.1 Overview

ZeBu now offers the capability to evaluate, during emulation runtime, the impact of powering up and down some parts of the design. This Power Aware Verification is necessary for modern System-On-Chip (SOC) designs, especially for battery-operated devices for which power consumption is a critical point.

Power Aware Verification information is provided in a dedicated set of files describing how the physical power is implemented by the ASIC back-end tools.

At runtime, the activation of the different power domains can be controlled either by the design or from the testbench in order to evaluate:

- Instant impact:  
What is happening in the design when the registers/ports of a power domain are switched ON or OFF, taking into account the impact of multi-cycle operations
- Long-term impact:  
What is happening in the design when a power domain is OFF for a long time.

When a power domain is switched OFF, the ZeBu software provides different ways to force output ports and resume registers.

Isolation and/or retention strategies are also supported to improve the ZeBu verification environment for these power-related features.

## 1.2 Compliance with Standards

Power Aware Verification with ZeBu is based on the UPF 1.0 and UPF 2.0 (Unified Power Format) which is described in IEEE 1801-2009 standard.

## 1.3 Requirements

### 1.3.1 License Feature

Power Aware Verification requires specific licenses features for compilation and runtime.

### 1.3.2 ZeBu Compatibility

Power Aware Verification as described in this manual is applicable for ZeBu software version 7\_1\_0.



## 1.4 Interactions between Power-Aware Verification and Other Features

### 1.4.1 SVAs and DPI Calls

System Verilog Assertions (SVAs) and Direct Programming Interface (DPI) calls are switched off at the same time as the power domain where they are instantiated. This prevents the detection of inappropriate conditions on their inputs.

If you have SVAs or DPI calls connected through hierarchical references, you may have those references coming from an inactive power domain. When these SVAs or DPI calls are in an active power domain, this may lead to an inconsistent behavior. In that case, you may disable this SVAs or DPI calls manually at emulation time.

### 1.4.2 Force and Injection of Signals at Runtime

When forces and injections are declared for compilation with `zforce`, `zinject` or `force_dyn` commands, their behavior in a multi-domain design can be configured at runtime: you can choose to take into account or not the power domains when controlling the corresponding signals. The corresponding method of the API is described in Section 4.2.7.

#### 1.4.2.1 Force Implementation

Implementation of forces (via `zforce` and `force_dyn` options at compilation and `ZEBU::Signal::Force` at emulation runtime) with Power Aware Verification supports the following behaviors:

- When it is performed in an active power domain, the forces are correctly implemented.
- When the power domain is turned OFF, the forces become corrupted.
- When the power domain is turned back ON, the forces are restored.

#### 1.4.2.2 Injection Implementation

Implementation of injections (via the `zinject` option at compilation and `ZEBU::Signal::Inject` at emulation runtime) with Power Aware Verification supports the following behaviors:

- When it is performed in an active power domain, the injections are correctly implemented.
- When the power domain is turned OFF, the injections are cancelled.
- When it is performed in an inactive power domain, an error message as the one below is displayed:

```
Cannot inject signal 'top.my.signal': power domain 'MY_POWER_DOMAIN' is off
```

## 1.5 Hardware impact of Isolation and Retention Features

The implementation of isolation and retention features impacts the size of the design in terms of necessary FPGA resources.

### 1.5.1 Isolation Implementation in ZeBu

Each port of the design for which isolation is applicable requires 1 additional LUT and optionally 1 RAMLUT.

### 1.5.2 Retention Implementation in ZeBu

Each register for which retention is applicable requires an additional set of 3 to 5 registers.

## 1.6 Limitations

The following low-power related features are not supported in the present version of Power Aware Verification with ZeBu:

- Tension level shifting
- Force and injection on retained domains when registers are being restored
- Randomization on registers in case of incorrect configuration of the retention control signals
- Power-related attributes in the RTL source files
- UPF commands related to power state tables are parsed but not interpreted
- Retention on memory mapped as **zMem**
- LowConn signals in isolation
- Definitions of tuples (triplets of isolation supply, isolation signal and sense and isolation clamp arguments in an isolation strategy)
- It is not possible to map the design on FPGAs connected to RLDRAM.
- In case of simultaneous use of Power Aware Verification and Power Estimation with Flexible Probes use-model, the Power Estimation statistics data files do not take into account the power domains that are inactive: events on the pseudo random values are counted in the statistics data file.

## 2 Source Files

### 2.1 RTL Source Files

Power Aware Verification with ZeBu starts from the same RTL source files as the ones used for emulation with ZeBu.

### 2.2 Writing the Power Management Script for ZeBu

The Power Management Script is a Tcl script which describes the topology of the power network in the design. This script is an input file of the ZeBu compiler.

The Power Management script is compliant with the UPF 1.0 and UPF 2.0 (Unified Power Format) which is described in IEEE 1801-2009 standard and supports both hierarchical and non-hierarchical UPF.

In this script, RTL hierarchical paths can be declared and the usual Tcl rules are applicable, in particular to source an additional file, for special characters and to use the # character for comments.

#### 2.2.1 List of Supported UPF Commands for ZeBu

All the UPF commands and options in the Power Management Script are parsed by the ZeBu compiler:

- The subset of UPF commands listed in the table below is interpreted by the ZeBu software and impacts emulation time.
- The UPF commands which are not listed in this table are parsed by the ZeBu compiler without returning error messages and they are ignored by the ZeBu runtime software.

**Table 1: Supported UPF Commands for ZeBu**

Commands	Comment	See Section
set_design_top	Declaration of the name of the top-level module of the design. This command is expected only once in the Power Management Script.	§ 2.2.2.1
set_scope	Changes the active scope to another instance.	§ 2.2.2.2
create_power_domain	Creates a power domain by declaring the hierarchical paths to the instances of its extent.	§ 2.2.2.3
create_supply_net	Creates a net that provides power in the scope of the power domain where it is defined.	§ 2.2.2.4
create_supply_set	Creates a supply set in the active scope.	§ 2.2.2.5
create_supply_port	Declares a port that provides power to one or several domains.	§ 2.2.2.6
connect_supply_net	Connects a previously created supply port to a previously created supply net.	§ 2.2.2.7

Commands	Comment	See Section
set_domain_supply_net	For each power domain, declares the power and ground nets actually used.	§ 2.2.2.8
create_power_switch	Defines the switch instance (input supply nets, output supply net, conditions, etc.) for a given power domain.	§ 2.2.2.9
set_isolation	Defines an isolation strategy for a power domain	§ 2.2.2.10
set_isolation_control	Defines the control signals of an isolation strategy	§ 2.2.2.11
set_retention	Defines a retention strategy	§ 2.2.2.12
set_retention_control	Defines the control signals of a retention strategy	§ 2.2.2.13
load_upf	Loads an additional Power Management Script in a specified scope.	§ 2.2.2.14
query_isolation	Gets information on an isolation strategy	§ 2.2.2.15
query_isolation_control	Gets control information on an isolation strategy	§ 2.2.2.16
query_retention	Gets information on a retention strategy	§ 2.2.2.17
query_retention_control	Gets control information on a retention strategy	§ 2.2.2.18
query_power_domain	Gets information on the power domain	§ 2.2.2.19

## 2.2.2 Detailed Syntax

This section provides information on parameters and options for the commands listed in Table 1.

**Note** All these parameters and options comply with the UPF specifications and standards. Refer to IEEE documentation for more information.

### 2.2.2.1 set\_design\_top

This command must be the first one in the Power Management Script and must be used only once.

```
| set_design_top <top_module_name>
```

Where <top\_module\_name> is the name of the top-level module of the design.

#### **Example:**

```
| set_design_top top
```

### 2.2.2.2 set\_scope

Changes the active UPF scope to another instance.

```
| set_scope <path>
```

Where <path> is a hierarchical path to an instance in the design

- '..' to move the scope one level up
- '/' to move to root scope

**Note** set\_scope with no argument is not supported.

If successful, this command returns the active scope of the command as a full path relative to the active design top prior to execution. If it fails, a `TCL_ERROR` is raised.

**Example:**

```
set_scope {i0/\gen.i1 /i2}
```

#### 2.2.2.3 `create_power_domain`

Creates a power domain either by declaring the hierarchical paths to the instances of its extent or by selecting the whole active scope.

```
create_power_domain <domain_name> -elements <path_list>  
[-include_scope]
```

Where:

- `<domain_name>` is the user-defined name of the power domain
- The `-elements` option defines a Tcl list of hierarchical paths to instances in the design
- The `-include_scope` option selects the active scope and all its instances.

The `-elements` and `-include_scope` options cannot be used at the same time.

**Example:**

```
create_power_domain VCC0 -elements {top/child0}
```

#### 2.2.2.4 `create_supply_net`

Creates a net that provides power in the scope of the power domain where it is defined.

```
create_supply_net <net_name> -domain <domain_name> [-reuse]
```

Where:

- `<net_name>` is the name of the created supply net
- `<domain_name>` is the name of the domain to which the supply net is attached
- The `-reuse` option creates several supply nets with the same name for different power domains without creating conflicts. The `-reuse` option can be omitted for the first `create_supply_net` command with a given `<net_name>` but any subsequent command with the same `<net_name>` causes an error if `-reuse` is not set.

**Note** No signal with the same `<net_name>` must have been previously defined in the RTL design.

**Examples:**

```
create_supply_net VCC_TOP_net -domain TOP  
create_supply_net VCC_TOP_net -domain VCC0 -reuse
```

#### 2.2.2.5 create\_supply\_set

Creates in the active scope a supply set, which is a collection of supply nets associated with a given function name.

---

```
create_supply_set <set_name> -function <function> [-update]
```

---

Where:

- <set\_name> is the name for the supply set in the current scope
- <function> is a pair of a function name and a supply net name
- The -update option updates an already existing supply set

#### **Examples:**

Creating a new supply set called TOP\_SET:

---

```
create_supply_set TOP_SET -function {power VCC_net} -function  
{ground VSS_net}
```

---

Updating a primary supply set of power-domain called PDTOP:

---

```
create_supply_set PDTOP.primary -function {ground VSS_net} -update
```

---

#### 2.2.2.6 create\_supply\_port

Declares a port that provides power to one or several domains. This port can be created in the active scope or, if specified, in the scope of the power domain.

---

```
create_supply_port <port_name> -domain <domain_name>
```

---

Where:

- <port\_name> is the name of the port created at the top of the design;
- <domain\_name> is the name of the domain to which the supply net is attached. <domain\_name> is optional if you want to declare it in the active scope. To declare it in the scope of a given power domain, you must explicitly specify it.

---

**Note** No port with the same <port\_name> must have been previously defined in the RTL design.

---

#### **Example:**

---

```
create_supply_port VCC_TOP_port -domain TOP
```

---

#### 2.2.2.7 connect\_supply\_net

Connects a previously created supply port to a previously created supply net.

---

```
connect_supply_net <net_name> -ports <ports_list>
```

---

Where:

- <net\_name> is the name of the supply net previously created with the create\_supply\_net command;
- <ports\_list> is the Tcl list of port names previously created with create\_supply\_port commands.

**Example:**

```
connect_supply_net VCC_TOP_net -ports VCC_TOP_port
```

**2.2.2.8 set\_domain\_supply\_net**

Declares the power and ground nets actually used for each power domain.

```
set_domain_supply_net <domain_name> -primary_power_net  
<power_net_name> -primary_ground_net <ground_net_name>
```

Where:

- <power\_net\_name> is the name of the net actually providing power supply for <domain\_name>, previously created with a create\_supply\_net command for <domain\_name>;
- <ground\_net\_name> is the name of the net actually providing ground for <domain\_name>, previously created with a create\_supply\_net command for <domain\_name>.

**Note** Declaring the same net as a power supply net AND as a ground net causes an error.

**Example:**

```
set_domain_supply_net TOP -primary_power_net VCC_TOP_net \  
-primary_ground_net VSS_net
```

**2.2.2.9 create\_power\_switch**

Defines the switch instance (input supply nets, output supply net, conditions, etc.) for a given power domain. Please refer to the IEEE 1801-2009 standard for further details.

```
create_power_switch <switch_name>  
  -domain <domain_name>  
  -input_supply_port {<switch_input_port> <supply_net>}  
  -output_supply_port {<switch_output_port> <supply_net>}  
  -control_port {<switch_control_port> <control_signal>}  
  -on_state {<on_state_name> <input_supply_port> <condition>}  
  [-off_state {<off_state_name> <condition>}]
```

Where:

- <switch\_name> is the name of the switch instance.
- <domain\_name> is the name of the power domain for which the switch instance is created.
- -input\_supply\_port defines which <supply\_net> in the UPF file is connected to an input supply port of the switch instance (<switch\_input\_port>). This line can be repeated several times as you can specify several input supply ports for the switch instance (but only one <supply\_net> for each <switch\_input\_port>).



- -output\_supply\_port defines which <supply\_net> in the UPF file is connected to an output port of the switch instance (<switch\_output\_port>). There is only one output supply port per switch instance.
- -control\_port specifies which <control\_signal> in the design is used as a control port (<switch\_control\_port>) to activate the switch instance. This line can be repeated several times as you can specify several control ports (but only one <control\_signal> for each <switch\_control\_port>).
- -on\_state specifies which <input\_supply\_port> is driven to the output supply port of the switch instance when <condition> is valid. It also specifies an associated alias for this state (<on\_state\_name>).
- -off\_state is optional. It specifies which <condition> drives a 0 on the output supply port of the switch instance. It also specifies an associated alias for this state (<off\_state\_name>).

If any of the <control\_signal> is not found in the design's RTL source files, an error is thrown.

**Example:**

As long as the top/switch\_ctrl\_2\_reg design's controlling signal is 0, VCC2\_SW outgoing power net is not powered. It is powered when the controlling signal switches to 1 and VCC2\_SW is in the same state as VCC\_TOP\_net incoming power net.

```
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 icontrol/switch_ctrl_2_reg} \  
-on_state {VCCG_ON VCC_TOP_port {ctrl_sig} } \  
-off_state {VCCG_OFF {!ctrl_sig} }
```



2.2.2.10 set\_isolation

Defines an isolation strategy for a power domain.

```
set_isolation <isolation_name>
  -domain <domain_name>
  -elements <path_list>
  -source <supply_set_name>
  -sink <supply_set_name>
  -applies_to <target>
  -isolation_power_net <power_net_name>
  -isolation_ground_net <ground_net_name>
  -isolation_supply_set <supply_set_name>
  -isolation_signal <control_signal>
  -isolation_sense <high|low>
  -clamp_value <clamp>
  -location <automatic|self|parent>
  -diff_supply_only {TRUE|FALSE}
  -update
  -name_prefix <prefix>
  -name_suffix <suffix>
```

Where:

- <isolation\_name> is the name of the isolation strategy that is being defined
- <domain\_name> is the name of the power-domain for which the isolation strategy is applicable
- <path\_list> is a Tcl list of hierarchical paths to the instances or ports in the design upon which isolation instrumentation will be performed.
- <supply\_set\_name> is the name of the supply set whose <-sink> and <-source> options allow to filter element lists.  
For the -isolation\_supply\_set option, it allows to specify the power and ground nets of the isolation strategy.
- <target> filters ports that need to be isolated according to their direction. Possible values are inputs, outputs and both
- <power\_net\_name> is the name of the supply net used as power in the isolation strategy
- <ground\_net\_name> is the name of the supply net used as ground in the isolation strategy
- <control\_signal> is the name of the isolation signal
- The -isolation\_sense option defines the active level of the isolation signal defined in the -control\_signal. Possible values are high or low
- <clamp> defines the value of an isolated port. Possible values are any (default value), 0, 1 or Z.
- The -location option defines where the isolation cells are placed in the logic hierarchy. Possible values are automatic (default value), self or parent
- The -diff\_supply\_only option defines the isolation behavior between driver and receiver supply sets. Possible values are TRUE or FALSE

- The -update option updates an already existing isolation strategy
- <prefix> and <suffix> are the prefix and suffix for the names of elements added at isolation implementation

**Example:**

The following command defines an isolation strategy named VCC0\_isolation for the VCC0 power domain. Isolation is enabled when signal switch\_ctrl\_1\_reg switches to 0. In that case, the pins impacted by the isolation strategy will be driven to 0.

```
set_isolation VCC0_isolation -domain VCC0 \  
-isolation_power_net VCC_TOP_net \  
-isolation_ground_net VSS_net \  
-isolation_signal switch_ctrl_1_reg \  
-isolation_sense low \  
-clamp_value 0
```

**2.2.2.11 set\_isolation\_control**

Defines the control signals of an isolation strategy

```
set_isolation_control <isolation_name>  
-domain <domain_name>  
-isolation_signal <control_signal>  
-isolation_sense <high|low>  
-location <self|parent>
```

Where:

- <isolation\_name> is the name of the isolation strategy for which the control signals are defined.
- <domain\_name> is the name of the power-domain for which the isolation strategy is applicable
- <control\_signal> is the name of the isolation signal
- The -isolation\_sense option defines the active level of the isolation signal defined in the -control\_signal. Possible values are high or low.
- The -location option defines where the isolation cells are placed in the logic hierarchy. Possible values are automatic (default value), self or parent.

**Example:**

The following command defines the control parameters of the ISO1 isolation strategy.

```
set_isolation ISO1 -elements ...  
set_isolation_control ISO1 -domain PD0 -isolation_signal  
i0.\i1.gen0 .iso_1 -isolation_sense high -location self
```

#### 2.2.2.12 set\_retention

Defines a retention strategy for a power domain.

```
set_retention <retention_name>  
  -domain <domain_name>  
  -elements <path_list>  
  -retention_power_net <power_net_name>  
  -retention_ground_net <ground_net_name>  
  -retention_supply_set <supply_set_name>  
  -save_signal {<signal_name> <high|low|posedge|negedge>}  
  -restore_signal <signal_name> <high|low|posedge|negedge>}  
  -save_condition  
  -restore_condition  
  -update
```

Where:

- <retention\_name> is the name of the retention strategy that is being defined
- <domain\_name> is the name of the power-domain where the retention strategy will be applied
- <path\_list> is a Tcl list of hierarchical paths to the instances or signals in the design
- <power\_net\_name> is the name of the supply net used as power in the retention strategy
- <ground\_net\_name> is the name of the supply net used as ground in the retention strategy
- <supply\_set\_name> allows to specify the power and ground nets of retention strategy
- The -save\_signal option specifies a hierarchical path to the save signal and its sensitivity.
- The -restore\_signal option specifies a hierarchical path to the restore signal and its sensitivity.
- The -save\_condition option is a Boolean condition on hierarchical paths to the signal of the design. When this condition is satisfied, save can happen.
- The -restore\_condition option is a Boolean condition on hierarchical paths to the signal of the design. When this condition is satisfied, restore can happen.
- The -retention\_condition option is a Boolean condition
- The -update option updates an already existing retention strategy.

#### **Example:**

```
set_retention RET_PD \  
-domain PD \  
-retention_power_net VCC_TOP_net \  
-retention_ground_net VSS_net \  
-save_signal {save_signal high } \  
-restore_signal {restore_signal high }
```

#### 2.2.2.13 set\_retention\_control

Defines a retention strategy for a power domain.

```
set_retention_control <retention_name>  
    -domain <domain_name>  
    -save_signal {<signal_name> <high|low|posedge|negedge>}  
    -restore_signal <signal_name> <high|low|posedge|negedge>}
```

Where:

- <retention\_name> is the name of the retention strategy that is being defined
- <domain\_name> is the name of the power-domain where the retention strategy will be applied
- The -save\_signal parameter specifies a hierarchical path to the save signal and its sensitivity
- The -restore\_signal parameter specifies a hierarchical path to the restore signal and its sensitivity

#### Example:

The following command defines a retention strategy named RET\_PD\_latch that applies to all the elements that are synchronous with the PD\_latch power domain. The retention cells of this strategy are powered by the VCC\_TOP\_net and VSS\_net supply nets. The save event for this strategy occurs when the signal save\_cond\_reg=1 and posedge of save\_sig\_reg. The restore event for this strategy occurs when the signal restore\_cond\_reg=1 and posedge of restore\_sig\_reg.

```
set_retention RET_PD_latch \  
    -domain PD_latch \  
    -retention_power_net VCC_TOP_net \  
    -retention_ground_net VSS_net \  
    -save_condition      { save_cond_reg }          \  
    -restore_condition { restore_cond_reg } \  
    -retention_condition { retention_cond_reg } \  
set_retention_control RET_PD_FF \  
    -domain PD_FF \  
    -save_signal          { save_sig_reg posedge } \  
    -restore_signal       { restore_sig_reg posedge }
```

#### 2.2.2.14 load\_upf

Loads a new Power Management Script in a specified scope.

```
load_upf <file_name> [-scope]
```

Where:

- <file\_name> is the name and absolute path of the UPF file.
- The -scope option defines the name of the scope where the UPF file must be loaded

#### 2.2.2.15 query\_isolation

Gets information on isolation.

---

```
query_isolation <isolation_name> -domain <domain_name> [-detailed]
```

---

Where:

- <isolation\_name> is the name of the isolation strategy to query
- <domain\_name> is the name of the power-domain for which the isolation strategy has been defined
- The -detailed option returns all the strategy information

#### 2.2.2.16 query\_isolation\_control

Gets information on isolation control signals.

---

```
query_isolation_control <isolation_name> -domain <domain_name> [-detailed]
```

---

Where:

- <isolation\_name> is the name of the isolation strategy to query
- <domain\_name> is the name of the power-domain for which the isolation strategy has been defined
- The -detailed option returns all the strategy information

#### 2.2.2.17 query\_retention

Gets information on retention.

---

```
query_retention <retention_name> -domain <domain_name> [-detailed]
```

---

Where:

- <retention\_name> is the name of the retention strategy to query
- <domain\_name> is the name of the power-domain for which the retention strategy has been defined
- The -detailed option returns all the strategy information

#### 2.2.2.18 query\_retention\_control

Gets information on retention control signals.

---

```
query_retention_control <retention_name> -domain <domain_name> [-detailed]
```

---

Where:

- <retention\_name> is the name of the retention strategy to query
- <domain\_name> is the name of the power-domain for which the retention strategy has been defined
- The -detailed option returns all the strategy information

### 2.2.2.19 `query_power_domain`

Gets information on the power domain.

```
query_power_domain <domain_name> [-non_leaf|-all|-no_elements] [-detailed]
```

Where:

- <domain\_name> is the name of the power-domain to query
- The -non\_leaf|-all|-no\_elements options filter or exclude elements in the query
- The -detailed option returns all the power domain information

### 2.2.3 Additional Command for Attributes Modification

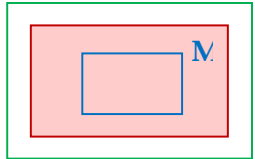

The following command is available to modify some attributes for Power Aware Verification:

```
set_design_attributes -attribute {<attr_name> <attr_value>} [-models {<module_list>}]
```

Where:

- <attr\_name> attributes are set to <attr\_value> (-attribute uses a Tcl list syntax to modify several attributes at once);
- <module\_list> is a list of RTL names for modules in the design for applicability of the attributes.  
(if -models is not present, the attributes are applicable for the complete design).

The following table lists the attributes that you may modify:

Attribute Name	Description	Supported Values (and default value)
UPF_dont_touch (*)	<p>When this attribute is set on a module and:</p> <ul style="list-style-type: none"> <li>• &lt;attr_value&gt; = TRUE: the sub-instances of the module are impacted.</li> <li>• &lt;attr_value&gt; = NOREC: the sub-instances of the module are not impacted. They inherit the power domain information from the hierarchy above module instantiation.</li> <li>• &lt;attr_value&gt; is FALSE: none of the instances of that module in the design are processed regarding the power domain information.</li> </ul>	<p>TRUE/NOREC/FALSE</p> <div style="text-align: center;"> <p>TRUE</p>  <p>NOREC</p>  </div>

Attribute Name	Description	Supported Values (and default value)
UPF_Glob_Matching	When set to TRUE, pattern matching and wilddcarding to facilitate usability and multiple object specification are made available for the following elements: <ul style="list-style-type: none"> <li>Power domain elements</li> <li>Isolation elements</li> <li>Retention elements</li> </ul>	TRUE/FALSE
SNPS_reinit	When this attribute is set on a module and its value is TRUE, the initial blocks of the RTL source files are re-evaluated at power ON of the domain to which this module belongs. When set to FALSE, no re-evaluation is performed. This is also the default behavior when this attribute is not specified.	TRUE/FALSE
ZEBU_err_power_domain_elements	Defines the error policy to apply to elements of the power-domain definition.	<ul style="list-style-type: none"> <li>warning: Only issues a warning when elements are missing from the design</li> <li>fatal_if_empty: Does not throw an error unless all elements are missing from the design (not supported by ZEBU_err_retention)</li> <li>fatal: Any missing element returns a fatal error</li> </ul>
ZEBU_err_isolation_signals	Defines the error policy to apply to elements of the isolation strategies.	
ZEBU_err_retention	Defines the error policy to apply to elements of the retention strategies.	

(\*) Declaring the same module for this attribute and in `create_power_switch` causes an error.

## Examples:

To have instances and signals below M0 and M1 modules not impacted by power domain information:

```
set_design_attributes -attribute {UPF_dont_touch TRUE} -models {M0 M1}
```

To have errors regarding the definition of the power domain return a warning:

```
set_design_attributes -attribute {ZEBU_err_power_domain_elements warning}
```

## 2.2.4 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 \
  -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  
  
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}
```

## 2.3 Optional RTL-Based Compilation Script

Constant signals of the RTL source code (in particular debugging signals such as the scan chain) can be available with the same state in several power domains with a reduced amount of logic for power-aware verification.

For that purpose, an RTL-based Compilation Script must be created with a specific syntax for the force command (`-no_power` option) on these nets:

```
force -rtlname <net> -no_power [-fnmatch]
```

Or

```
force -rtlname <port> -no_power [-fnmatch]
```

### Example:

This example forces the signal `top.ins.test_scan_enable1` to 0. This constant signal is propagated through the power domain boundaries. The amount of logic is minimized:

```
force -rtlname top.ins.my_test_scan_enable -value 0 -no_power
```

On the other hand, the signal `top.ins.test_scan_enable2` is forced to 0 without the `-no_power` option. This constant signal does not cross the boundaries of the power domain and is randomized beyond the domain.

```
force -rtlname top.ins.my_test_scan_enable -value 0
```

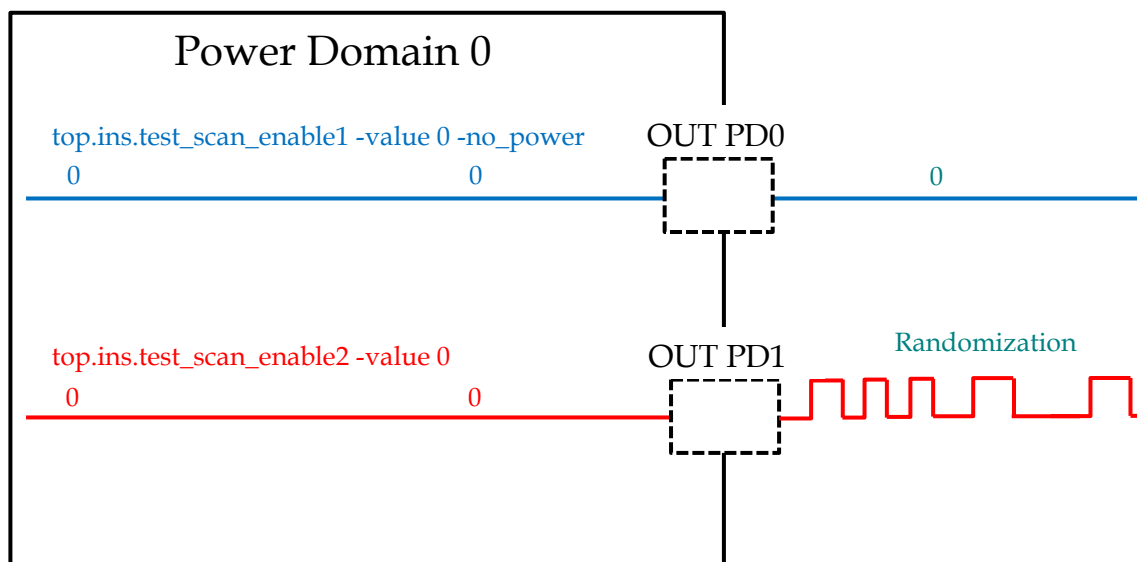



Figure 1: Signal Behavior with/without the `-no_power` Option

## 3 Compiling the Design

### 3.1 ZeBu Compilation Settings

This section lists the specific settings to compile a design for power-aware emulation. It does not provide all the options to compile the design for ZeBu: please refer to the [ZeBu-Server-2 Compilation Manual](#) before compiling the design.

#### 3.1.1 Declaring the Source Files and the Power Management Script

1. In **zCui**, declare the RTL source files and the optional RTL-based compilation script as described in the [ZeBu-Server-2 Compilation Manual](#).
2. In the **Design** workspace, select **Power Management Scripts** item in the left-hand pane.
3. Click the  control-button. **zCui** opens a file selector to add the UPF file.

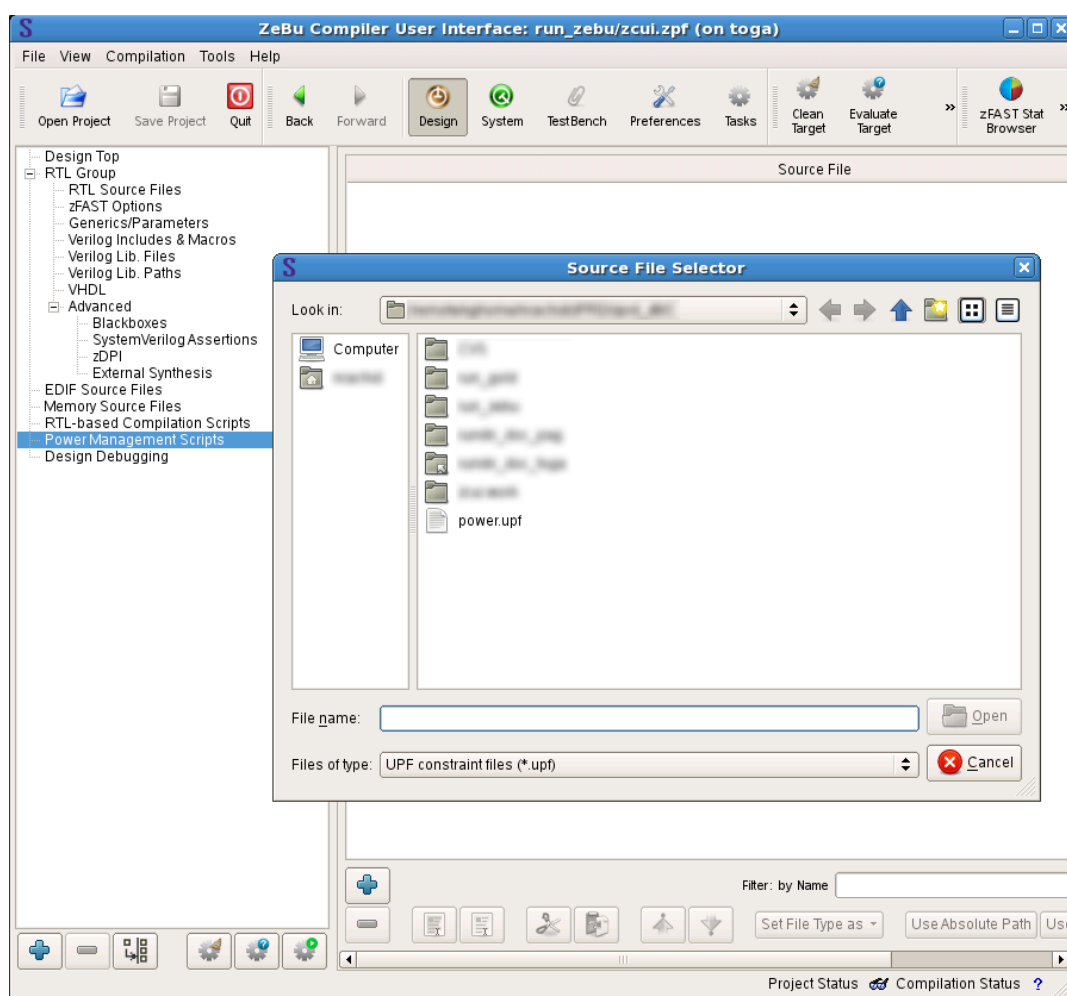


Figure 2: Declaration of UPF Scripts in zCui

### 3.1.2 Options for Synthesis

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

Other synthesizers are not supported for Power Aware Verification.

Besides, please note that it is possible to synthesize multi-RTL groups design when some Power Management Scripts are declared in the top group of the design only; Power Management Scripts cannot be declared in more than one RTL group.

## 3.2 Compilation Follow-Up

In **zCui**, error and warning messages concerning the UPF file processing are listed in the **Make RTL Front End** task.

## 4 Emulation Runtime

### 4.1 Introduction

When the design is compiled with UPF files, emulation runtime can still be done with the usual test environment if no Power Aware Verification is expected.

### 4.2 C++ Interface

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

However, for easier implementation, this header file is automatically available when declaring the `libZebu.hh` header file.

The API provides a set of methods:

- To start emulation runtime with Power Aware Verification. See Section 4.2.1.
- To initialize the environment. See Section 4.2.2.
- To get information about the design. See Section 4.2.3.
- To get information about retention. See Section 4.2.4.
- To control the power domains. See Section 4.2.5.
- To declare user callbacks. See Section 4.2.6.
- To manage forces and injections. See Section 4.2.7.

A full featured example of a testbench for Power Aware Verification is also available in Section 4.2.8.

All methods described below throw an exception if the pointer to the ZeBu board is not correct.

For other failures, these methods return a Boolean value:

- `true` for a correct processing
- `false` in case of error

**Table 2: C++ API to Control Power Aware**

C++ Methods	Description	See Section
<code>init</code>	Starts Power Aware Verification.	4.2.1
<code>enable</code>	Enables Power Aware Verification.	4.2.1
<code>initializeRandomizer</code>	Initializes the generator which will later apply values to registers, ports and memories in one or all power domains.	4.2.2.1
<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>	4.2.2.2

C++ Methods	Description	See Section
performRandomizerDomain	Set all elements of a domain to pseudo-random values, or 0 or 1 according to the mode selected in <code>initializeRandomizer</code> .	4.2.2.3
isPowerManagementEnabled	Checks if the design has been compiled to support Power Aware Verification.	4.2.3.1
getListOfDomains	Returns a list of all power domains declared in the Power Management Script.	4.2.3.2
getPowerDomainState	Provides the state of a power domain.	4.2.3.3
getLastTriggeredDomain	Returns the list of power domains for which the state changed (ON→OFF or OFF→ON), using a dedicated trigger mechanism.	4.2.3.4
enableRetentionStrategy	Enables a retention strategy.	4.2.4.1
disableRetentionStrategy	Disables a retention strategy.	4.2.4.2
isRetentionStrategyEnabled	Gets information about the retention strategy.	4.2.4.3
getListOfRetentionStrategies	Gets the list of available retention strategies	4.2.4.4
setPowerDomainOn	Switches a power domain to ON state	4.2.5.1
setPowerDomainOff	Switches a power domain to OFF state.	4.2.5.2
setPowerDomainState	Switches a power domain to the state declared in the parameters.	4.2.5.3
releasePowerDomain	Allows a domain to be no longer controlled from the testbench but by the RTL design.	4.2.5.4
setPowerDomainOffCallback	Declares a replacement function for the default behavior of the software when a power domain is switched OFF.	4.2.6.1
setPowerDomainOnCallback	Declares a replacement function for the default behavior of the software when a power domain is switched ON.	4.2.6.2
setForceMode	Defines the behavior of forces and injections regarding power domain states.	4.2.7.1

**Note** For legibility purposes, <method\_name> is often used in this chapter in replacement of `PowerMgt::<method_name>`.

#### 4.2.1 Starting Emulation Runtime with Power Aware Verification

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

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

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

**Note** If your design has been compiled with a Power Management script but you do not target power aware verification during runtime, the `init` and the `enable` methods can be omitted in the testbench.

**Example:**

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

## 4.2.2 Initializing the Environment

### 4.2.2.1 initializeRandomizer

Initializes the generator which will later apply values to registers, ports and memories in one or all power domains. Three different modes are available: pseudo-random values to produce a result similar to 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 using the result of a pseudo-random generator.
  - seedValue: integer value for initialization of the pseudo-random generator.

---

**Note** All-0 and all-1 values are only applicable to state elements such as registers, latches and memories. They do not apply to interfaces of the power domain.

---

### 4.2.2.2 performRandomizer

All elements of the design, whatever their power domain, are forced to pseudo-random values, or 0 or 1 according to the mode selected in `initializeRandomizer`. If the power domain has been set to ON or is controlled by the design, this method 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 such as registers, latches and memories. They do not apply to interfaces of the power domain.

---

### 4.2.2.3 performRandomizerDomain

All elements of a domain are forced to pseudo-random values, or 0 or 1 according to the mode selected in `initializeRandomizer`. If the power domain has been set to ON or is controlled by the design, this 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 such as registers, latches and memories. They do not apply to interfaces of the power domain.

---

### 4.2.3 Getting Information

#### 4.2.3.1 isPowerManagementEnabled

Checks if the design has been compiled to support Power Aware Verification, as described in Section 3.1.

```
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).

#### 4.2.3.2 getListOfDomains

Returns a list of all power domains declared in 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 object).

#### 4.2.3.3 getPowerDomainState

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 / 0 stands for OFF.

#### 4.2.3.4 getLastTriggeredDomain

Returns the list of power domains for which the state changed (ON→OFF or OFF→ON), using a dedicated trigger mechanism.

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

---



Where:

- board: pointer to the ZeBu::Board object.
- triggerChangedDomain: domain names for which the power state changed (reference to object).

## 4.2.4 Managing Retention Strategies

### 4.2.4.1 enableRetentionStrategy

Enables a 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.

### 4.2.4.2 disableRetentionStrategy

Disables a 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.

### 4.2.4.3 isRetentionStrategyEnabled

Gets 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.

### 4.2.4.4 getListOfRetentionStrategies

Gets 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.

## 4.2.5 Controlling Power Domains

### 4.2.5.1 setPowerDomainOn

Switches a power domain to ON state (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)

### 4.2.5.2 setPowerDomainOff

Switches a power domain to OFF state (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)

### 4.2.5.3 setPowerDomainState

Switches a power domain to the state declared in the parameters (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 which gives the expected state (0 for OFF, any positive value for ON).

---

**Note** setPowerDomainState (board, domainname, 1) is equivalent to setPowerDomainOn(board, domainname).  
setPowerDomainState (board, domainname, 0) is equivalent to setPowerDomainOff(board, domainname).

---

### 4.2.5.4 releasePowerDomain

A domain is no longer controlled from the testbench but by the RTL design.

```
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)

## 4.2.6 Declaring User Callbacks

### 4.2.6.1 setPowerDomainOffCallback

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

This method can be used to declare a replacement function for the default behavior of the software when a power domain is switched OFF (note that 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()` is not present in the testbench or when it is present with `setPowerDomainOffCallback(board, NULL, NULL)`.

### 4.2.6.2 setPowerDomainOnCallback

By default, the ZeBu software sets all registers/memories of the power domain to the pseudo-random values to be sure that the restart of the domain does not match a possible state of its outputs.

This method can be used to declare a replacement function for the default behavior of the software 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 present in the testbench or when it is present with `setPowerDomainOnCallback(board, NULL, NULL)`.

## 4.2.7 Managing Forces and Injections

### 4.2.7.1 setForceMode

Defines the behavior of forces and injections regarding power domain states, as described in Section 1.4.2:

- If set to 0 (default mode), the power domain states are taken into account when the forces and injections are applied.
- If set to 1, forces and injections are applied without taking the power domain states into account.

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

Where board is the pointer to the ZeBu::Board object.

## 4.2.8 C++ Example

The following example shows 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);

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

// forces 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);
// displays the list of power domains for which the state has
changed
std::set<std::string> domains;
PowerMgt::getLastTriggeredDomain(_zebu, domains);

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

### 4.3 Plain C Language Interface

When the testbench is written in plain C, equivalent functions are available to match the functional description of the C++ methods described above.

The corresponding header file is `ZeBu_PowerMgt.h`.

Except if stated otherwise in the description in the header file, all functions of the C API return:

- 0 in case of success
- A non-zero value is returned in case of an error.

**Table 3: C++ API/ C API Equivalence**

C++ API	Plain C API
<code>initializeRandomizer</code>	<code>ZEBU_PowerMgt_initializeRandomizer</code>
<code>setPowerDomainOnCallback</code>	<code>ZEBU_PowerMgt_setPowerDomainOnCallback</code>
<code>setPowerDomainOffCallback</code>	<code>ZEBU_PowerMgt_setPowerDomainOffCallback</code>
<code>setPowerDomainOn</code>	<code>ZEBU_PowerMgt_setPowerDomainOn</code>
<code>setPowerDomainOff</code>	<code>ZEBU_PowerMgt_setPowerDomainOff</code>
<code>setPowerDomainState</code>	<code>ZEBU_PowerMgt_setPowerDomainState</code>
<code>releasePowerDomain</code>	<code>ZEBU_PowerMgt_releasePowerDomain</code>
<code>performRandomizer</code>	<code>ZEBU_PowerMgt_performRandomizer</code>
<code>performRandomizerDomain</code>	<code>ZEBU_PowerMgt_performRandomizerDomain</code>
<code>isPowerManagementEnabled</code>	<code>ZEBU_PowerMgt_isPowerManagementEnabled</code>
<code>getPowerDomainState</code>	<code>ZEBU_PowerMgt_getPowerDomainState</code>
<code>getLastTriggeredDomain</code>	<code>ZEBU_PowerMgt_getLastTriggeredDomain</code>
<code>getListOfDomains</code>	<code>ZEBU_PowerMgt_getListOfDomains</code>
<code>init</code>	<code>ZEBU_PowerMgt_init</code>
<code>Enable</code>	<code>ZEBU_PowerMgt_enable</code>
<code>enableRetentionStrategy</code>	<code>ZEBU_PowerMgt_enableRetentionStrategy</code>
<code>disableRetentionStrategy</code>	<code>ZEBU_PowerMgt_disableRetentionStrategy</code>
<code>isRetentionStrategyEnabled</code>	<code>ZEBU_PowerMgt_isRetentionStrategyEnabled</code>
<code>getListOfRetentionStrategies</code>	<code>ZEBU_PowerMgt_getListOfRetentionStrategies</code>
<code>setForceMode</code>	<code>ZEBU_PowerMgt_setForceMode</code>

#### Example:

The following example shows a C testbench for Power Aware Verification after initialization of the ZeBu board:

```
// Initialize power engine with parameters for pseudo-random number
// generator
ZEBU_PowerMgt_initializeRandomizer(zebu, "MODE_RND", 42);

// Runs the testbench with power methods activated
ZEBU_Driver_run(top_ccosim, cycle);

// displays the list of power domains for which the state has
// changed
char **triggerChangedDomain = NULL;
unsigned int cnt = ZEBU_PowerMgt_getLastTriggeredDomain(ZEBU_Board
*board, &triggerChangedDomain)
```

```
for (unsigned int i = 0; i < cnt; ++i)
{
    printf("domain %s has triggered.", triggerChangedDomain[i]);
}
```

---

## 5 Use Models for Debugging

ZeBu provides different means to investigate the specific issues in the design with Power Aware Verification when the design controls the power domains using the information of the UPF file.

### 5.1 Checking Isolation between Power Domains

When the waveforms dumped during emulation runtime with Power Aware Verification show some unexpected behaviors, these issues may be caused by an isolation issue between power domains in the design. In particular the pseudo-random values set on the output interface of an OFF power domain may cause unexpected values on inputs of other domains if isolation is not correct (for example, if isolation is not enabled or supply of isolation is down).

These unexpected values in waveforms should be investigated upstream to locate the connection with the power domain which is switched OFF.

Once a specific power domain is pointed out as a possible root cause for isolation issues, it can be switched OFF manually from the ZeBu API and waveforms of the inputs of the other active domains can be dumped.

The ZeBu C++ API for power aware verification offers specific methods to manually control the activation of power domains, as described in Section 4.2.5:

C++ Method	Description
setPowerDomainOn	Switches a power domain to ON state (the power domain is no longer controlled by the design).
setPowerDomainOff	Switches a power domain to OFF state (the power domain is no longer controlled by the design).
setPowerDomainState	Switches a power domain to the state declared in the parameters (the power domain is no longer controlled by the design).
releasePowerDomain	A domain is no longer controlled from the testbench but by the RTL design.

## 5.2 Checking the State of Instances

You can check if any instance in the design is ON or OFF by adding a dynamic probe with **zSelectProbes/zDbPostProc** on the ZEBU\_POWER\_ON signal. This signal is automatically available for all instances in the design when compiled for power aware verification:

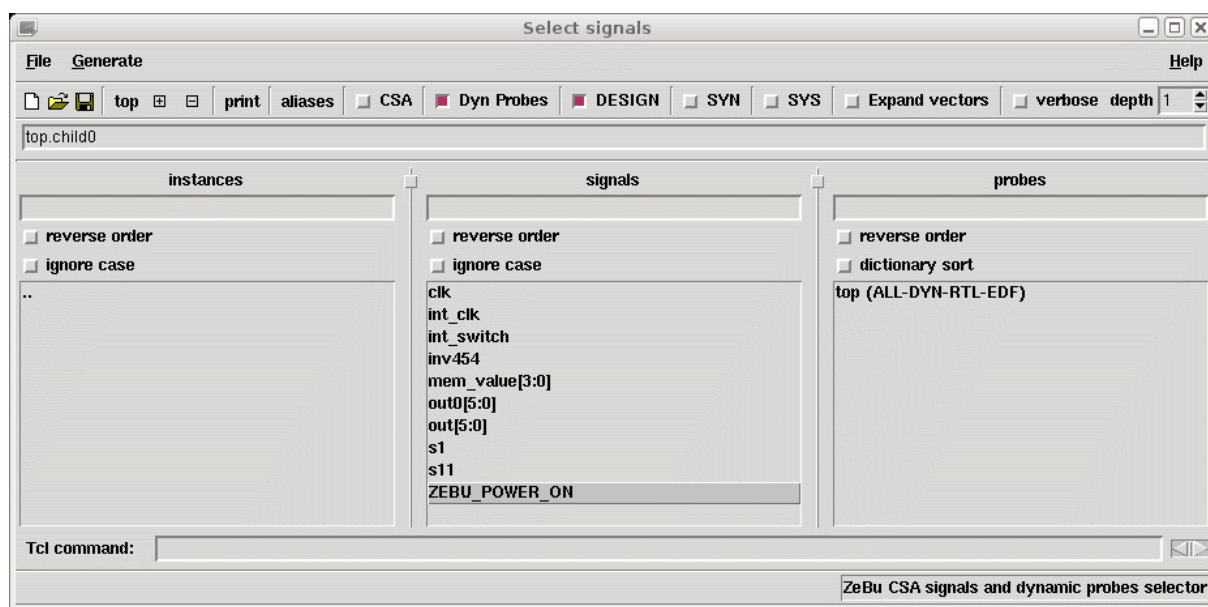


Figure 3: ZEBU\_POWER\_ON Signal in zSelectProbes

## 5.3 Checking the State of Power Domains

The ZeBu C++ API for Power Aware Verification offers specific methods to check properties and state of a power domain, as described in Section 4.2.5:

C++ Method	Description
isPowerManagementEnabled	Checks if the design has been compiled to support Power Aware Verification
getListOfDomains	Returns a list of all power domains declared in the Power Management Script.
getPowerDomainState	Provides the state of a power domain.
getLastTriggeredDomain	Returns the list of power domains for which the state changed (ON→OFF or OFF→ON), using a dedicated trigger mechanism.

Messages when a power domain changes states (ON/OFF) are reported in the runtime logs.



## 5.4 Control Signals for Retention Strategies

Retention strategies can be switched ON/OFF in the testbench via the `enableRetentionStrategy` and `disableRetentionStrategy` methods, as described in Section 4.2.4.

For debugging purposes, retention strategies can be controlled through dynamic probes. The corresponding instances and condition signals in `zSelectProbes` can be seen in the following figures:



Figure 4: Retention Strategies Instances in zSelectProbes

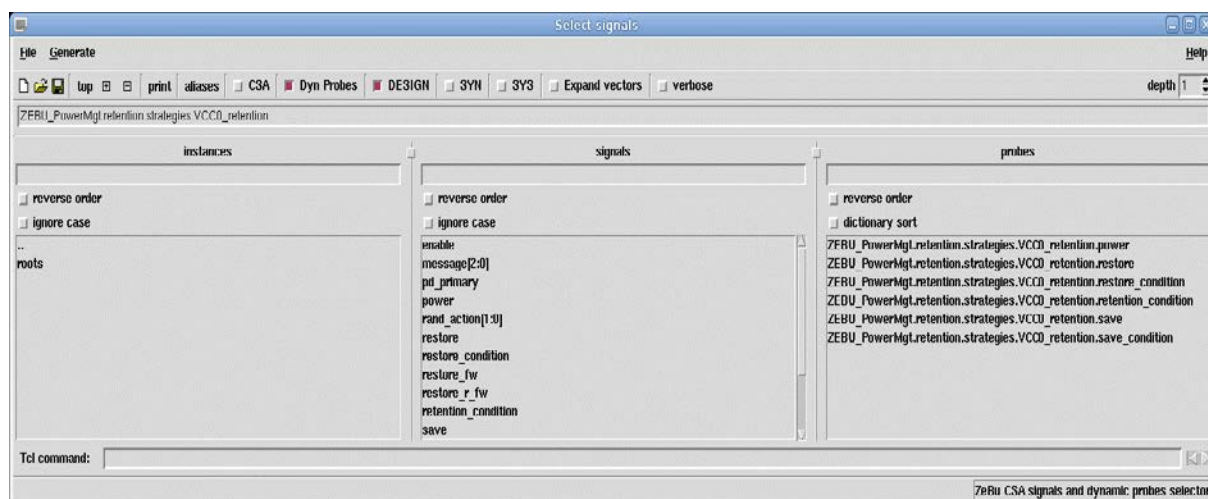


Figure 5: Signals of a Retention Strategy in zSelectProbes

## 6 Troubleshooting

The following sections show the most frequent error messages that may be reported in the log file of the testbench during the Power Aware Verification session and describe the solution to solve each problem.

### 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`)

Please contact Synopsys support at [zebu\\_support@synopsys.com](mailto: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 (as described in Section 4.2) without prior compilation of 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 recompile your design with the Power Aware Verification feature before using it, as described in Chapter 3.

## 6.4 Failure when Calling any Retention-Related Method

If you call any of the methods related to retention strategies (as described in Section 4.2.4) without prior definition of retention commands 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 contain retention-related commands if you want to call any retention-related method.