Chassis Power Gating PGCB Verification IP

Integration Guide

**Synopsis:**

This component should be used to validate a PMCs Chassis defined power gating interface. It is a System Verilog OVM component. The user can configure the number of SIP, Fabric and delays using configuarion objects as well as contrainted-random transactions. This VC will also consists of a monitor that scoreboards can subscribe to, a checker to chek the Chassis defined power gating protocols and coverage collector.

IP Rev # 2015WW25June 19th 2015

Copyright and Disclaimer Information

Copyright © 2012, Intel Corporation. All rights reserved.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

\* Other names and brands may be claimed as the property of others.

This document contains information on products in the design phase of development.

Information in this document is provided in connection with Intel products. No license, express or implied, by estoppel or otherwise, to any Intellectual property rights is granted by this document. Except as provided in Intel’s terms and conditions of sale for such products, Intel assumes no liability whatsoever and Intel disclaims any express or implied warranty, relating to sale and/or use of Intel products including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright, or other Intellectual property right.

Unless otherwise agreed in writing by Intel, the Intel products are not designed or intended for any application in which the failure of the Intel product could create a situation where personal injury or death may occur.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked “reserved” or “undefined.” Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your Intel account manager or distributor to obtain the latest specifications and before placing your product order.

Copies of documents that have an order number and are referenced in this document or in other Intel literature can be obtained from your Intel account manager or distributor.

Contents

[1 Getting Started 4](#_Toc337819195)

[2 Setting Up a Testbench Environment 5](#_Toc337819196)

[2.1 Creating a Standalone Testbench 5](#_Toc337819197)

[2.1.1 Connecting Agents in the Testbench 6](#_Toc337819198)

[2.1.2 Naming and Instantiating Components 8](#_Toc337819199)

[2.1.3 Extending the Testbench for Test Execution 9](#_Toc337819200)

[2.1.4 Steps to compile and run test 10](#_Toc337819201)

[2.2 Creating a Cluster and Full Chip Testbench 11](#_Toc337819202)

[3 Implementing Test Scenarios 12](#_Toc337819203)

[3.1 Configuring the Agents 12](#_Toc337819204)

[3.2 Sending Specific Transaction Sequences (Using the Base Sequence) 12](#_Toc337819205)

[3.3 Extending Transaction Constraints (Controlling Timing Delays) 13](#_Toc337819206)

[4 Agent Parameters 15](#_Toc337819207)

[5 Agent Interface 16](#_Toc337819208)

[5.1 PowerGatingIF signals 16](#_Toc337819209)

# Getting Started

Here is a guide to getting started quickly explained in the following sub-sections:

See the next section for more detailed instructions

* Installing for the First Time
* Running a Standalone Demo

Example:

Cd into Chassis\_PowerGatingVC and setenv MODEL\_ROOT $cwd

source ace/ace.env

To complile and elaborate: ace -c

To run a test : ace –x –t <test\_name>

# Setting Up a Testbench Environment

This section describes how to setup and configure OVM envirnonments for this Agents. The previous section explained Agent package installation, prerequisite tools, and example Testbenches.

This section show how Agent can be connected in two types of testbench :

* Standalone Testbench
* Cluster and Chip Testbench

The Standalone testbench connects an agent to another agent and is used to model the connection of this agent to unit-level DUT. Cluster and Chip Testbench connects an agent to a unit-level DUT in Cluster and Chip environment.

The section after this one describes how configure Agents and implement test scenarios.

## Creating a Standalone Testbench

This section focuses on how to connect Agents in Standalone Testbench, with example codes.

Here is a block diagram of the standalone testbench which was used to validate the agents. The CC and PGCB agents are connected to each other



**Block diagram of standalone testbench**

Refer to each item mentioned below:

* Connecting Agents in the Testbench
* Naming and Instantiating Testbench Components
* Extending the Testbench for Test Execution
* Steps to compile and run test

### Connecting Agents in the Testbench

Shown below is an example of the test island/test-bench file.

The power gating VC need to be in passive mode at SOC. Therefore it need to be instantiated in the IP’s TI. But the power gating VC interface **does not support TB automation flow**. So please follow the guidelines below while integrating these collaterals.

IPs that have only one instance at SOC level (most IPs fall under this category) are required to instantiate the VCs in their TI and make the connections to the interface internally inside the TI using `defines. Please note that the assign statements need to done with care. Otherwise, it could cause checker/coverage to be disabled or SOC integration issues. Example:

module pmc\_ti(iosf\_sb\_intf iosf\_sb\_if);

PowerGatingIF pg\_if;

PGCBAgentTI pg\_ti(pg\_if);

generate if(IS\_ACTIVE)

assign pg\_if.ip\_pmc\_pg\_req\_b = `IP\_TOP.abc\_pmc\_pg\_req\_b;

assign `IP\_TOP.pmc\_abc\_pg\_ack\_b = pg\_if.pmc\_ip\_pg\_ack\_b;

.....

generate if(!IS\_ACTIVE)

assign pg\_if.ip\_pmc\_pg\_req\_b = `IP\_TOP.abc\_pmc\_pg\_req\_b;

assign pg\_if.pmc\_ip\_pg\_ack\_b = `IP\_TOP.pmc\_abc\_pg\_ack\_b;

.....

endmodule

IMPORTANT NOTE:

IP\_ENV\_TO\_PGCB\_AGENT\_PATH - This parameter specifies the full hierarchy of the CCAgent instance starting from the IP’s env name. The hierarchy should be specified in the form \*<Env’s OVM name>.<PGCBAgent OVM name>.

In the examples shown below, say the env is instantiated in the base test as follows

env = <GPIO env type>::type\_id::create("**pmc\_env**", this);

The CCAgent is instantiated in the env as follows

pgcbAgent = PGCBAgent::type\_id::create("**pmc\_pgcb\_agent**",this)

So the parameter should be set to - **\*.pmc\_env.pmc\_pgcb\_agent**

PowerGatingIF#(

.NUM\_SIP\_PGCB(NUM\_SIP\_PGCB),

.NUM\_FET(NUM\_FET),

.NUM\_FAB\_PGCB(NUM\_FAB\_PGCB),

.NUM\_SW\_REQ(NUM\_SW\_REQ),

.NUM\_PMC\_WAKE(NUM\_PMC\_WAKE),

.NUM\_PRIM\_EP(NUM\_PRIM\_EP),

.NUM\_SB\_EP(NUM\_SB\_EP)

) pgIF ();

generate if (!IS\_ACTIVE) begin: ASSIGN\_PASSIVE\_BLK

//This is jsut for the monitor

assign pgIF.clk = `IP\_TOP.clk;

assign pgIF.reset\_b = `IP\_TOP.reset\_b;

assign pgIF.pmc\_ip\_sw\_pg\_req\_b = `IP\_TOP.pmc\_ip\_sw\_pg\_req\_b;

assign pgIF.ip\_pmc\_pg\_req\_b = `IP\_TOP.ip\_pmc\_pg\_req\_b;

assign pgIF.pmc\_ip\_pg\_ack\_b = `IP\_TOP.pmc\_ip\_pg\_ack\_b;

assign pgIF.pmc\_ip\_pg\_wake = `IP\_TOP.pmc\_ip\_pg\_wake;

assign pgIF.pmc\_ip\_restore\_b = `IP\_TOP.pmc\_ip\_restore\_b;

assign pgIF.prim\_pok = `IP\_TOP.prim\_pok;

assign pgIF.side\_pok = `IP\_TOP.side\_pok;

assign pgIF.fab\_pmc\_idle= `IP\_TOP.fab\_pmc\_idle;

assign pgIF.pmc\_fab\_pg\_rdy\_req\_b = `IP\_TOP.pmc\_fab\_pg\_rdy\_req\_b;

assign pgIF.fab\_pmc\_pg\_rdy\_ack\_b = `IP\_TOP.fab\_pmc\_pg\_rdy\_ack\_b;

assign pgIF.fab\_pmc\_pg\_rdy\_nack\_b = `IP\_TOP.fab\_pmc\_pg\_nack;

assign pgIF.fet\_en\_b = `IP\_TOP.fet\_en\_b;

assign pgIF.fet\_en\_ack\_b = `IP\_TOP.fet\_en\_ack\_b;

end: ASSIGN\_PASSIVE\_BLK

else begin: ASSIGN\_ALL\_BLK

assign pgIF.clk = `IP\_TOP.clk;

assign pgIF.reset\_b = `IP\_TOP.reset\_b;

assign pgIF.pmc\_ip\_sw\_pg\_req\_b = `IP\_TOP.pmc\_ip\_sw\_pg\_req\_b;

assign `IP\_TOP.ip\_pmc\_pg\_req\_b = pgIF.ip\_pmc\_pg\_req\_b ;

assign pgIF.pmc\_ip\_pg\_ack\_b = `IP\_TOP.pmc\_ip\_pg\_ack\_b;

assign pgIF.pmc\_ip\_pg\_wake = `IP\_TOP.pmc\_ip\_pg\_wake;

assign pgIF.pmc\_ip\_restore\_b = `IP\_TOP.pmc\_ip\_restore\_b;

assign `IP\_TOP.prim\_pok = pgIF.prim\_pok;

assign `IP\_TOP.side\_pok = pgIF.side\_pok;

assign `IP\_TOP.side\_pok = pgIF.fab\_pmc\_idle;

assign pgIF.pmc\_fab\_pg\_rdy\_req\_b = `IP\_TOP.pmc\_fab\_pg\_rdy\_req\_b;

assign `IP\_TOP.fab\_pmc\_pg\_rdy\_ack\_b = pgIF.fab\_pmc\_pg\_rdy\_ack\_b ;

assign `IP\_TOP.fab\_pmc\_pg\_rdy\_nack\_b = pgIF.fab\_pmc\_pg\_rdy\_nack\_b;

assign pgIF.fet\_en\_b = `IP\_TOP.fet\_en\_b;

assign pgIF.fet\_en\_ack\_b = `IP\_TOP.fet\_en\_ack\_b;

end: ASSIGN\_ALL\_BLK

endgenerate

PGCBAgentTI #(

.NUM\_SIP\_PGCB(NUM\_SIP\_PGCB),

.NUM\_FET(NUM\_FET),

.NUM\_FAB\_PGCB(NUM\_FAB\_PGCB),

.NUM\_SW\_REQ(NUM\_SW\_REQ),

.NUM\_PMC\_WAKE(NUM\_PMC\_WAKE),

.NUM\_PRIM\_EP(NUM\_PRIM\_EP),

.NUM\_SB\_EP(NUM\_SB\_EP),

.IS\_ACTIVE(IS\_ACTIVE),

//.NO\_FAB\_PGCB(1),

.IP\_ENV\_TO\_PGCB\_AGENT\_PATH({IP\_ENV, “.pmc\_pgcb\_agent”})

)pgcbTI(pgIF);

#### Driving fet\_en\_ack\_b from the testbench

Note that the fet\_en\_ack\_b needs to be driven from the testbench in response to fet\_en\_b with some delay. In the real system, the ack would be driven by the last fet in the fet chain.

### Naming and Instantiating Components

Shown below is an example of how to instantiate the agents and configure it using the PowerGatingConfigObject.

Note that the arguments need to be passed by name while configuring the agent using the config object methods. See system Verilog LRM for details on passing arguments by name.

class PowerGatingSaolaEnv extends ovm\_env;

`ovm\_component\_utils\_begin(PowerGatingSaolaEnv)

`ovm\_component\_utils\_end

PGCBAgent pgcbAgent;

PowerGatingConfig cc\_cfg\_pgcb;

function new(string name, ovm\_component parent);

super.new(name, parent);

endfunction

function void build();

// Turn off all the sequencers by default

set\_config\_int("\*sequencer", "count", 0);

set\_config\_int("\*pmc\_pg\_agent", "is\_active", 1);

**set\_config\_int("\*pgcbcAgent01\*", "hasPrinter", 1);**

super.build();

//this.set\_level(SLA\_TOP);

pgcbAgent = PGCBAgent::type\_id::create("pmc\_pg\_agent", this);

cc\_cfg\_pgcb = new({"pmc\_pg\_agent","ConfigObject"});

`ovm\_info(get\_full\_name(), "Agents created", OVM\_HIGH)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PGCB

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

cc\_cfg\_pgcb.SetTestIslandName("pmc\_testisland");

cc\_cfg\_pgcb.SetTrackerName("PGCBAgentTracker");

cc\_cfg\_pgcb.AddFETBlock(.index(0), .name("FET0"));

cc\_cfg\_pgcb.AddFETBlock(.index(1), .name("FET1"));

cc\_cfg\_pgcb.AddFETBlock(.index(2), .name("FET2"));

cc\_cfg\_pgcb.AddFabricPGCB(.index(0), .name("FAB0"), .fet\_index(2));

//int .index, string .name, int .fet\_index = 0, time .hys = 0ps, int sip\_mapping{$}, int fabric\_mapping{$}, bit initial\_state

cc\_cfg\_pgcb.AddSIPPGCB(.index(0), .name("SIP0"), .fet\_index(0), .pmc\_wake\_index(0), .sw\_ent\_index(1), .SB\_array({0}));

cc\_cfg\_pgcb.AddSIPPGCB(.index(1), .name("SIP1"), .fet\_index(1), .pmc\_wake\_index(1), .sw\_ent\_index(0), .SB\_array({1}));

cc\_cfg\_pgcb.AddSIPPGCB(.index(2), .name("FSIP"), .fet\_index(2), .pmc\_wake\_index(2), .sw\_ent\_index(2), .SB\_array({2}), .fabric\_index(0));

cc\_cfg\_pgcb.AddSIP(.name("ADSP"), .sip\_type(PowerGating::HOST), .pgcb\_array({0, 1}), .AON\_prim\_array({0}));

cc\_cfg\_pgcb.AddSBEP(.index(0), .source\_id('hE8));

cc\_cfg\_pgcb.AddSBEP(.index(1), .source\_id('hB5));

cc\_cfg\_pgcb.AddSBEP(.index(2), .source\_id('hA2));

cc\_cfg\_pgcb.AddPrimEP(.index(0), .AON\_EP(1), .pmc\_wake\_index(3));

//User set config object using {<name>, “ConfigObject”}

set\_config\_object("\*", "**pmc\_pg\_agentConfigObject**",cc\_cfg\_pgcb,0);

`ovm\_info(get\_full\_name(), "Set config object for PGCBAgent", OVM\_HIGH)

### Extending the Testbench for Test Execution

Here is a list of existing tests that can be run on this standalone test-bench.

|  |  |
| --- | --- |
| Test Name | Description |
| FabricBasicTest | Fabric exit idle command  Make sure req and ack deassert  Fabric enter idle  Make sure req and ack assert |
| FabricAbortTest | Fabric exit idle command  Make sure req and ack assert  Fabric enter idle  Make sure req asserts  Fabric exit idle again before ack  Make sure the NACK asserts  Here make fabric enter idle again  Make sure the req deassert first and only then asserts again. |
| FabricAgentWakeTest | For this test the SetAgentWakeModel finction needs to be called  See section10.1 |
| FabricResponseOverrideTest | In this test, we use the OVM type overdise capabilities to override the delay constraints to >0 and <5.  The response seq item also contains a noResponse bit that can be used to prevent the agent from responding.  See section 11.2 |
| SIPBasicTest | 1. Wake up the SIP using PMC wake  2. Make sure req and ack deassert  3. Assert SW PG request  4. Make sure req and ack assert  7. Assert HW UG req  8. Make sure req and ack assert  9. Assert HW PG req  10. Make sure req and ack deassert |
| SIPAbortTest | Wake up the SIP using PMC wake  Make sure req and ack deassert  Assert HW Save req  Deassert HW Save req  Make sure the flow is aborted |
| ResetTest | Assert reset in the middle of the test |
| ArbitrationTest | In this test, the arbitration logic is tested |
| FETModeTest | This test is to validate the FET ON mode |
| WaitForComplete | This test is to validate waitforcomplete function for a pmc wake sequence |
| IPInaccTest | Tests all the state machines for IP Inacc state |
| Acc\_IPInaccTest | Tests all state machine arcs and to verify agent going from Acc PG state to In acc PG state |
| AssertionFailTest | Tests failure cases of all assertions |
| ConcurrentTest | Random test to test arbitration and make sure all requests get granted eventually. |
| ErrorPMCWakeTest | Test the error cases requested by PMC |
| FETModeTest | Tests the mode where fets are not turned off and also tests resetting the mode. |
| RestoreTest | Test basic retore flow |
| RestoreErrorTest | Test errorenous scenario where IP asserts pg\_req\_b while in the restore window. |

### Steps to compile and run test

ace –c –x -t <test\_name>

### hdl files

The hdl files are in the ace folder.

### UDF files

There are no UDF files provided but in a typical IP udf file, you would have to speficy the path to the hdl and the dependent libs as follows. Note: Post VCS version 2013.06 is causing error after removing the VCS Backward compatibility switch (BC mode switch ) XLRM PACKAGE IMPORT COMPAT = TRUE.  This switch is getting deprecated in the upcoming VCS release i.e. VCS 2014.03. Previously this error was being masked due to presence of the BC mode switch. A fix is provided in the udf file and ChassisPowerGatingVIP Pkg files. The IP will have to add a vlog\_opt to there udf file for compiling ChassisPowerGatingVIP. Refer to below for the the vlog\_opts switch “VCS\_EXPORT\_SUPPORT”.

ChassisPowerGatingVIP => {

-hdl\_spec => ['verif/lib/shared/ChassisPowerGatingVIP/ace/ChassisPowerGatingVIP.hdl],

-dependent\_libs => ['ovm\_pkg', 'sla\_pkg',],

-vlog\_opts => [

"-sverilog +define+VCS\_EXPORT\_SUPPORT",

}

## Creating a Cluster and Full Chip Testbench

Shown below is a block diagram of the cluster test environment.



In the full-chip/SOC environment, use “is\_active” config in the Agent use set the agent in passive mode.

set\_config\_int("\*pmc\_pg\_agent", "is\_active", 0);

Make sure it matches the IS\_ACTIVE in the test-island. Otherwise, you will get a warning and the CCAgent will override with IS\_ACTIVE specified in the TI.

# Implementing Test Scenarios

This section shows how to write tests. The previous section explained how to setup and configure a simulation envirnonment. In this section, we focus on features that allow the Agents to create test scenarios.

* Configuring the Agents
* Sending Specific Transaction Sequences
* Extending Transaction Constraints

This section describes how to stimulate and control a DUT, and the section that follows this one describes how to observe and verify a DUT. It discusses how configure the interface assertions, monitor, Scoreboard, and coverage collector.

## Configuring the Agents

This section shows how to configure Agents from a test or Testbench. Add in an example of how to configure Agents during the configure phase of OVM simulation using Agent configuration methods.

As shown in the previous sections, the PowerGatingConfig object can be used to configure the agent.

## Sending Specific Transaction Sequences (Using the Base Sequence)

This section shows how to create a customized Stimulus Generator to send directed transactions to the Agents.

The **PGCBAgentBaseSequence** must be used to create any sequence with the necessary constaints.

Note that the base sequence has a rand bit waitForComplete which can be set to 1 if the user wants to wait till the sequence compeltes. Please see the userguide to know what wait for compelte means for different sequences.

Example

class SIPHWSaveReqSequence extends ovm\_sequence;

//=============================================================

// PUBLIC VARIABLES

//================================================================

**PGCBAgentBaseSequence seq;**

//==================================================================

// OVM Macros for public variables

//===================================================================

`ovm\_sequence\_utils\_begin(SIPHWSaveReqSequence, PGCBAgentSequencer)

`ovm\_sequence\_utils\_end

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @brief Constructor.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function new(string name = "SIPHWSaveReqSequence");

super.new(name);

endfunction : new

task body();

`ovm\_do\_with(seq, {seq.cmd == PowerGating::SIP\_SAVE\_REQ; seq.source == 1;seq.delay == 3; **waitForComplete** == 1;})

endtask

endclass : SIPHWSaveReqSequence

## Extending Transaction Constraints (Controlling Timing Delays)

Complete list of constraints supported in this Agent is described in section 11.

The response sequence item’s constraints can be changed as shown below.

Once the new sequence item is created, the ovm factory function set\_type\_override\_by\_type can be used to override by type in the test.

class PGCBAgentResponseSeqItemNew1 extends PGCBAgentResponseSeqItem;

//=========================================================================

// OVM Macros for public variables

//=========================================================================

`ovm\_object\_utils\_begin(PGCBAgentResponseSeqItemNew1)

//`ovm\_field\_int(delay , OVM\_ALL\_ON)

`ovm\_object\_utils\_end

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @brief Constructor.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function new(string name="PGCBAgentXaction");

super.new(name);

endfunction : new

/\*constraint delay\_c {

delay == 5;

}

\*/

constraint delay\_ug\_req\_c {

delay\_ug\_req == 5;

}

//Setting noRespnse to 1. So the respnder will not send any responses.

constraint noResponse\_c {

noResponse == 1;

}

endclass: PGCBAgentResponseSeqItemNew1

Here is an example if a test where the type override is done

class ArbitrationTest extends PowerGatingBaseTest;

SIPPMCWakeAllSequence sipWakeAll;

SIPPMCWakeSequence sipPMCWake;

SIPSWPGReqSequence sipSWPGReqSeq;

SIPHWUGReqSequence sipHWUGReqSeq;

SIPHWSaveReqSequence sipHWSaveReqSeq;

SIPRandHWSaveReqSequence randsipHWSaveReqSeq;

SIPRandHWUGReqSequence randsipHWUGReqSeq;

`ovm\_component\_utils(ArbitrationTest)

function new(string name = "ArbitrationTest", ovm\_component parent = null);

super.new(name, parent);

endfunction

function void build();

set\_config\_int("\*", "count", 0);

set\_config\_int("\*","recording\_detail", OVM\_FULL);

ovm\_top.set\_report\_verbosity\_level(OVM\_FULL);

super.build();

**set\_type\_override\_by\_type(PGCBAgentResponseSeqItem::get\_type(), PGCBAgentResponseSeqItemNew1::get\_type());**

**set\_type\_override\_by\_type(CCAgentResponseSeqItem::get\_type(), CCAgentResponseSeqItemNew::get\_type());**

endfunction

# Agent Parameters

Show examples on using Agent parameters to configure the signal width, etc.

parameter int NUM\_SIP\_PGCB = 1;

parameter int NUM\_FET = 1;

parameter int NUM\_SW\_REQ = 1;

parameter int NUM\_PMC\_WAKE = 1;

parameter int NUM\_FAB\_PGCB = 1;

parameter bit NO\_SIP\_PGCB = 0;

parameter bit NO\_FAB\_PGCB = 0;

parameter int NUM\_SB\_EP = 1;

parameter int NUM\_PRIM\_EP = 1;

parameter bit NO\_PRIM\_EP = 0;

parameter bit IS\_ACTIVE = 1;

parameter string IP\_ENV\_TO\_PGCB\_AGENT\_PATH = “”;

|  |  |
| --- | --- |
| parameter | Description |
| NUM\_SIP\_PGCB | Total number of SIP PGCBs. If there are no SIP PGCBs in the test env, then set NO\_SIP to 1.  The save and pg handshakes will be one per SIP PGCB |
| NUM\_FET | This is the numbe of FET blocks. Fet\_en\_b and fet\_en\_ack\_b will be one per FET block  Using configuration object, the user can map different PGCBs to FET blocks. |
| NUM\_SW\_REQ | Number of SW PG requests |
| NUM\_PMC\_WAKE | Number is PMC wake signals |
| NUM\_FAB\_PGCB | Total number of fabric PGCBs. If there are no SIP PGCBs in the test env, then set NO\_FAB\_PGCB to 1.  The fabric pg req, ack and nack will be one per fabric PGCB |
| NO\_SIP\_PGCB | Set to 1 if there are not SIP PGCBs in the test env. |
| NO\_FAB\_PGCB | Set to 1 if there are not Fabric PGCBs in the test env. |
| IS\_ACTIVE | This parameter should be set to 0 when the agent is promoted to an environment where the actual PMC is present. This should match the Agent’s is\_active config. |
| NUM\_SB\_EP | The number of sideband endpoints |
| NUM\_PRIM\_EP | The number of primary endpoints |
| NO\_PRIM\_EP | Set this to 1 if there are no primary endpoints |
| IP\_ENV\_TO\_PGCB\_AGENT\_PATH | This is the hierarchy of the PGCBAgent instance in the IP’s env. The hierarchy should be specified in the form \*<Env OVM name>.<PGCBAgent OVM name>. Please see integration guide for more details. |
|  |  |

# Agent Interface

List the signal interface supported in this Agent.

## PowerGatingIF signals

input logic clk;

input logic reset\_b;

input logic[NUM\_SW\_REQ-1:0] pmc\_ip\_sw\_pg\_req\_b;

output logic[NUM\_SIP\_PGCB-1:0] ip\_pmc\_save\_req\_b;

input logic[NUM\_SIP\_PGCB-1:0] pmc\_ip\_save\_ack\_b;

input logic[NUM\_SIP\_PGCB-1:0] pmc\_ip\_restore\_b;

output logic[NUM\_SIP\_PGCB-1:0] ip\_pmc\_pg\_req\_b;

input logic[NUM\_SIP\_PGCB-1:0] pmc\_ip\_pg\_ack\_b;

input logic[NUM\_PMC\_WAKE-1:0] pmc\_ip\_pg\_wake;

logic[NUM\_SB\_EP-1:0] side\_pok;

logic[NUM\_PRIM\_EP-1:0] prim\_pok;

output logic[NUM\_FAB\_PGCB-1:0] fab\_pmc\_idle;

input logic[NUM\_FAB\_PGCB-1:0] pmc\_fab\_pg\_rdy\_req\_b;

output logic[NUM\_FAB\_PGCB-1:0] fab\_pmc\_pg\_rdy\_ack\_b;

output logic[NUM\_FAB\_PGCB-1:0] fab\_pmc\_pg\_rdy\_nack\_b;

input logic[NUM\_FET-1:0] fet\_en\_b;

input logic[NUM\_FET-1:0] fet\_en\_ack\_b;