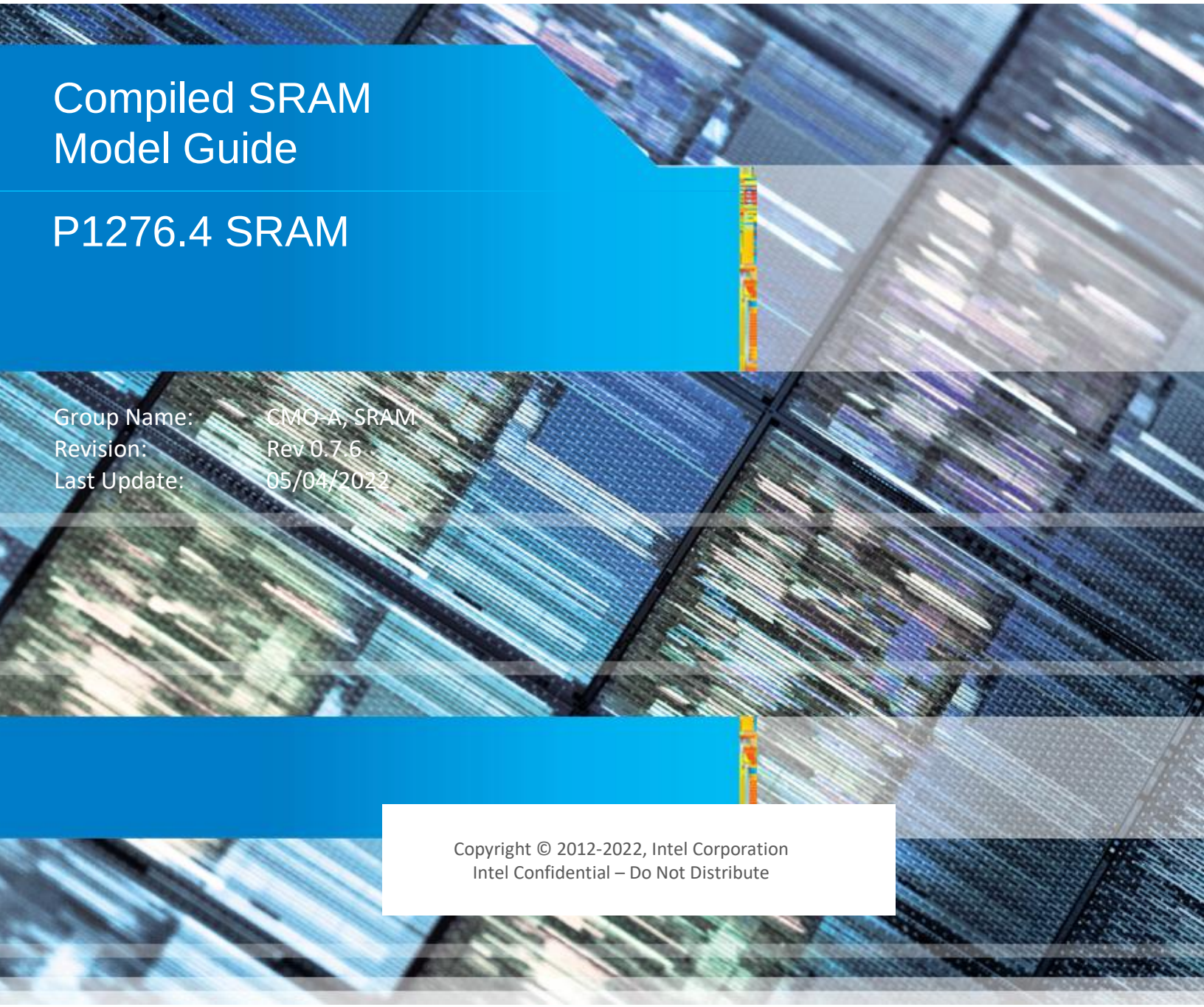


Compiled SRAM Model Guide

P1276.4 SRAM

Group Name: CMO-A, SRAM
Revision: Rev 0.7.6
Last Update: 05/04/2022

Table of Contents

List of Tables

List of Figures

Section 1 – Models

1.1 Overview

This document describes the models provided with compiled SRAM memories. Both usage and construction are covered. Not covered are performance, specifications and other hard block specific details that would be redundant if included in this guide. Please refer to the specific hard block integration guide for the memory in question for performance, physical, operational and general specifications and documentation. Table 1 shows a summary of the compilers covered by this document. Unless specifically noted, features and descriptions cover all compilers.

Table 1 *Compiler Summary (compilers currently covered by this guide)*

Compiler Name	Comments
c76hsuspsr High Speed Single Port SRAM Compiler	
c76hduspsr High Density Single Port SRAM Compiler	
c76hdusplr High Density Single Port SRAM Compiler	

1.2 Model Types

These compilers come with several models for various purposes. Table 2 shows the model types covered in this document. There are three (3) basic model types. The model types covered are the BMOD (behavioral) model, the ATPG (Automated Test Pattern Generation) models and the synthesizable DFX wrapper model, including the Tessent LVLIB. Within the BMOD description, there is also a section covering UPF (Unified Power Format) usage.

Table 2 *Model types provided with SRAM compiler outputs*

Model	Description	Purpose	Language
BMOD	Behavioral model	Represents the HB (Hard Block) of the SRAM memory. Model use cases: 1. General Verilog simulation. 2. SDF annotated simulation. 3. For power aware simulations if UPF is used (requires the UPF file, which is provided). 4. For Emulation/FPGA by setting appropriate defines.	System Verilog RTL with SVA (System Verilog Assertion) and UPF capabilities. Emulation/FPGA mode is synthesizable on the target Emulation/FPGA machines.

Model	Description	Purpose	Language
UPF	UPF model	Unified Power Format to represent the functionality of the power supplies and switches for the HB (Hard Block) of the SRAM memory. 1. Hard block upf file is loaded with BMOD RTL when simulating power-aware functionality. File is also utilized in SoC power integration flows. **2.** Additional hard block upf wrapper file is included if power integration requirements require isolation.	Intel UPF 2.1 standard (based on IEEE 1801-2013)
ATPG	Scan ATPG model	ATPG (Automated Test Pattern Generation) for Scan Testing. Model use cases: 1. Full feature (full) model including functionality for power management, output latch and redundancy. 2. Reduced (simple) model with no functionality for power management, output latch or redundancy.	Split Verilog and Fastscan core (fslib)
DFX	DFX wrapper	Synthesized to provide outer wrapper for redundancy and FHAS Gen3 capabilities. Also used for Verilog/Emulation/FPGA/ATPG Simulations.	System Verilog
LVLIB	Tessent	MBIST insertion model	Tessent

1.3 Collateral Views Provided

The internal collateral views covered by this document are listed in Table 3. In the and following table, diagrams and text, a FUB (Functional Unit Block) is the particular configuration of memory, sometimes referred to as an instance.

Table 3 *Collateral model views for Internal (Intel) deliveries*

View	Description
<fubname>.sv	BMOD w/ integral timing wrapper
<fubname>_dfx_wrapper.sv	Synthesizable Gen3 DFX wrapper RTL
<fubname>.hdl	BMOD hdl file
<fubname>.upf	UPF (Universal Power Format) file
global_upf.cfg	UPF global config file
<fubname>_dfx_wrapper.lvlib	Tessent view for DFX level
<fubname>.atpg.v	ATPG Verilog model
<fubname>.fslib	Fastscan ATPG memory core model
<fubname>_fastscan.do.cat	Fastscan constraints example file
<fubname>_REDUNDANCY_INFO.txt	File describing the redundancy solution fuse mapping

1.4 BMOD (Behavioral MODel)

The BMOD is a System Verilog model designed to emulate the functions of the corresponding hard IP memory block. This model is presented as a single file with multiple modules and macro defines included. All modules and macros included are uniquified with a memory name prefix and therefore usable only by the top level and internal modules contained within the model. This allows multiple SRAM instantiations of different configurations to be used together without conflict. The BMOD will require no external files to support its use and is therefore the complete representation of the SRAM hard block. Any similarity to macro names found within the BMOD to actual libraries is coincidental and not necessarily intended to match any real synthesis libraries.

1.4.1 BMOD features

The BMOD contains several features as follows.

1. **Zero-delay**
    a. Default model provides zero delay simulation.
2. **SDF annotation ready**
    a. SDF model with specify block and annotation buffers. Enabled with GLS compiler directive.
3. **Full emulation of all functional modes**
    a. Normal operations
        i. Read
        ii. Write, including bit write enable resolution
    b. Dual Row redundancy with enable pin per redundant row.
        i. See section on DFX wrapper model, 1.6, for more information on full redundancy implementation including column redundancy
4. **Partial emulation of test modes**
    a. Functional aspects of array sleep and periphery shutoff modes
        i. Read/write disabled in these modes

    b. Timing and voltage selection (tuning) inputs are not modeled in the Verilog model.
5. **SVA indications of Error and Warning conditions** (selectable, on by default)
    a. Read/Write (same address, same cycle) collision
    b. Multiple redundant row selects
    c. Address out of range (X address will not fire SVA)
    d. Tuning fuse inputs unknown (Z or X).
6. **Verilog messaging, for non SVA capable simulators, for Error and Warning conditions (**selectable, on by default**)**
    a. Read/Write on same cycle (same address) with X propagation
    b. Multiple redundant row selects
    c. Address out of range
7. **X generation for hazards**
    a. Read/Write (same address) collision
        i. X read data and X write location
    b. Redundant row collision
        i. X read data, X both redundant rows and target memory location
    c. Address out of range
        i. X read data
8. **X propagation in UPF enabled simulations**
    a. X propagation of appropriate internal registers with power domain shutoff (UPF only)
        i. UPF file for BMOD is supplied
9. **X propagation in PAV (Power Aware Verilog, non-UPF) enabled simulations**
    a. X propagation of appropriate internal registers with power domain shutoff.
        i. In non-UPF mode, the BMOD contains power domain inputs which control power domain features.
10. **Defines for control of simulation modes** (see section 1.4.2.1, BMOD `defines, below)
    a. Enabling/disabling of power pins in interface
    b. Enable timing wrapper for SDF annotation
11. **Tasks for array and output control**
    a. Array initialization from user defined file (readmemh).
    b. Array X for internal Array power domain gating (with and without UPF)
    c. Tasks to support memLibCertify fault injection.
12. **X-state check for input signals (INTC_MEM_<fubname>_XINPUT_CHECK)**
    a. Hazard warning message
13. **X propagation for input signals (with X behavior response)**
    a. Clocking
    b. Functional Write/Read operations (Address in/out of valid range)
    c. Redundancy conditions
    d. PME signals (output and array corruption)
14. **Hazards check for asynchronous assertion/de-assertion of power management (PME) signals during valid write/read**
    a. Output corruption
    b. Array corruption
15. **Fuse input check**
    a. Warning for unknown or unconnected fuse inputs
16. **SDF X-propagation**
    a. X-response for input setup/hold timing failures of PME inputs.
        i. Feature only exists in GLS model.
        ii. Feature can be enabled using 'INTC_MEM_ENABLE_GLS_XPROP'.

17. **Clock reset paranoia checks**
    a. Checks to look for clock reset capability independent of internal memory clock reset.
        i. Disabledd by default.
        ii. Used as an advanced paranoia check.
        iii. Can be used to make sure there is another clock reset capability in place in case needed for silicon debug.
        iv. Enabledd with the 'INTC_MEM_ENABLE_CLK_CHECKS' `define.

1.4.2 BMOD controls (defines and tasks)

The BMOD contains several different control methods for adjusting how the model is used. There are controls for turning off SVA and other messaging as well as several other features. Each sub-section below will cover each type of control in more detail.

1.4.2.1 BMOD `defines

The BMOD employs some `defines which allow user control of its modes and capabilities. The `defines can be used in a defines file, on the command line, or with a defparam command.

Table 4 *BMOD `defines*

Define name	Action if defined / Description	Comments
INTEL_NO_PWR_PINS	Removes power pins from interface (default is power pins)	
INTC_ADD_VSS	Adds vss to the power pin list	Happens when NO_PWR_PINS is not defined
INTC_MEM_GLS	Enables timing wrapper	This define enables GLS timing mode.
INTC_MEM_<fubname>_GLS		
INTC_MEM_COV_OFF	Turn off Coverage in DFX wrapper	
INTEL_SVA_OFF	Turn off SVAs	
INTC_MEM_<fubname>_SVA_OFF		
INTC_MEM_FEV	Modifies SVAs for FEV work, Filters out error messages	
INTC_MEM_LINT	Kills write logic for LINTRA checks	**User setting is not required nor recommended under normal usage, however it is required if running Lintra on the BNOD.**
INTC_MEM_TESTMODE	Turns off hazard messages	Used for CMO validation only
INTC_MEM_<fubname>_TESTMODE		
INTC_MEM_MODEL_TECH	Use "#0" instead of "final"	Used in assertions
INTC_MEM_XINPUT_CHECK	Turn on checks for X logic state on inputs	
INTC_MEM_<fubname>_XINPUT_CHECK	Turn on checks for X logic state on inputs	
INTC_MEM_NOXHANDLING	Turns off X handling/messaging	
INTC_MEM_<fubname>_NOXHANDLING	Turns off X handling/messaging	
INTC_MEM_ATPG_SIMPLE	Turn on ATPG simple mode	
INTEL_FPGA	Turn on FPGA specific options	
INTC_MEM_<fubname>_FPGA_MODE		
INTEL_EMULATION	Turn on Emulation specific options	
INTC_MEM_<fubname>_EMULATION		
INTC_MEM_<fubname>_EF_MODE		
INTC_MEM_FAST_SIM	Same as FPGA mode, but different name	Switches model to faster simulation FPGA mode, but without interfering with the INTEL_FPGA define. This mode sets the internal uniquified versions of the INTEL_SVA_OFF, INTC_MEM_TESTMODE, EF_MODE and EF_SIM defines.
INTC_MEM_EF_SIM	Enable unit delays for simulation of FPGA mode in regular Verilog simulator.	Must NOT be used for synthesis of the FPGA model.
INTC_MEM_<fubname>_EF_SIM		

Define name	Action if defined / Description	Comments
INTC_MEM_ESP	Enable ESP specifc tasks	Used when running ESP tool (CMO internal only)
INTC_MEM_<fubname>_ESP_TASKS		
INTC_MEM_PATH_SIZE	Path size for array load task	Default 256
INTC_MEM_ENABLE_GLS_XPROP	Turn on X-propagation for SDF timing failures on PME paths	Only applicable to GLS model
INTC_MEM_fault_norepair	Global enable of unrepairable fault injection	
.INTC_MEM_<fubname>_fault_norepair	Global enable of unrepairable fault injection	
INTC_MEM_fault_repair	Global enable of repairable fault injection	Faults are exhaustive of repair solutions
INTC_MEM__<fubname>_fault_repair	Global enable of repairable fault injection	Faults are exhaustive of repair solutions
INTC_MEM_fault_single	Global enable of single bit fault injection	
INTC_MEM__<fubname>_fault_single	Global enable of single bit fault injection	
INTC_<fubname>_local_fi_nrep	Instance enable of unrepairable fault injection	
INTC_<fubname>_local_fi_rep	Instance enable of repairable fault injection	Faults are exhaustive of repair solutions
INTC_<fubname>_local_fi_srep	Instance enable of single bit fault injection	
INTC_MEM_<fubname>_FI	Turns on Fault Injection code block	
<fubname>_ en_count	Enable for all count cycle defines	
INTC_MEM_async_reset_count	Defines number of cycles to count after async_reset is released	All count defines simply require setting to a value in order to turn the cycle counting feature on.
<fubname>_async_reset_cnt	Local version of previous define	Auto set by previous define
<fubname>_async_reset_count	Local version of previous define	Auto set by previous define
INTC_MEM_shutoff_count	Defines number of cycles to count after shutoff is released	
<fubname>_shutoff_cnt	Local version of previous define	Auto set by previous define
<fubname>_shutoff_count	Local version of previous define	Auto set by previous define
INTC_MEM_pshutoff_count	Defines number of cycles to count after pshutoff is released	
<fubname>_ pshutoff _cnt	Local version of previous define	Auto set by previous define
<fubname>_ pshutoff _count	Local version of previous define	Auto set by previous define
INTC_MEM_arysleep_count	Defines number of cycles to count after arysleep is released	
<fubname>_ arysleep _cnt	Local version of previous define	Auto set by previous define
<fubname>_ arysleep _count	Local version of previous define	Auto set by previous define

Table 5 *BMOD controlled/Partname Uniquified BMOD `defines*

Define name	Action if defined / Description	Comments
		These macro defines are NOT required or recommended to be set by the end user. The BMOD directly controls these based on other defines in Table 4**.**
INTC_MEM_<fubname>_SVA_OFF	Turn off SVAs	
INTC_MEM_<fubname>_TESTMODE	Turns off hazard messages	
INTC_MEM_<fubname>_FPGA_MODE	Turn on FPGA specific options	
INTC_MEM_<fubname>_EMULATION	Turn on Emulation specific options	
INTC_MEM_<fubname>_EF_MODE	Turn on FPGA/Emulation common	
INTC_MEM_<fubname>_EF_SIM	Enable unit delays in FPGA mode	
INTC_MEM_<fubname>_ESP_TASKS	Bring array tasks into EF models	
INTC_MEM_<fubname>_FI	Turns on Fault Injection code block	
<fubname>_ en_count	Enable for all count cycle defines	

1.4.2.1.1 **INTEL_NO_PWR_PINS** (aka UPF mode switch)

If INTEL_NO_PWR_PINS is defined, the power pins are left off the BMOD interface for UPF capability. If not set, the power pins are left on the BMOD interface and PAV mode is activated. The default if no define is set is to have power pins on the BMOD interface.

1.4.2.1.2 **INTC_ADD_VSS**

If INTEL_NO_PWR_PINS is NOT defined, this define will add vss to the power pin list. The purpose is to allow use in an environment where there is potential mixing of internal and external IP that has differing power pin requirements. Most external IP has vss in the power pin list in the Verilog models.

1.4.2.1.3 **INTC_MEM_GLS, INTC_MEM_<fubname>_GLS**

This define will enable all the specify block features in the GLS version of the BMOD.

1.4.2.1.4 INTEL_SVA_OFF, INTC_MEM_<fubname>_SVA_OFF

This define removes the SVA code from the model. This is used for FEV and other model validation where the SVA gets in the way, or the tool does not understand SVA. It is also used for the Verilog version of the model in order to allow it to run in a non-System Verilog simulator. The default is that the SVAs are active. It must be used to turn SVA off if running a non-System Verilog simulation.

1.4.2.1.5 INTC_MEM_FEV

This define is used in the model to disable the error reporting part of SVA and removing the effect of the "rst" input to the `defines for the assertions. This should require no user setting at this time.

1.4.2.1.6 INTC_MEM_LINT

This define is used to mask certain behavioral code from the model for the purpose of running Lintra checks. The behavioral code that is non-synthesizable or breaks other Lintra rules that are otherwise not applicable to the operation of the BMOD are masked out with this define. The user must set this define if attempting Lintra runs in order to prevent fatal stops to the Lintra run. All other Lintra errors that are waivable are waived in the BMOD with inline Lintra waivers.

1.4.2.1.1 INTC_MEM_TESTMODE, INTC_MEM_<fubname>_TESTMODE

This define disables the Verilog based hazards that are in parallel with the SVAs so that they can be turned off or left on independent of turning the SVAs on or off.

1.4.2.1.1 INTC_MEM_MODEL_TECH

This define causes the use of Use "#0" instead of "final" in SVA assertions. It is mostly used for NC-Verilog usage.

1.4.2.1.1 INTC_MEM_XINPUT_CHECK, INTC_MEM_<fubname>_XINPUT_CHECK

This define enables the checking for X logic state on inputs.

1.4.2.1.2 INTC_MEM_NOXHANDLING, INTC_MEM_<fubname>_NOXHANDLING

This define turns off the X generation for specific hazards. This define should not be used for PAV simulations as it could adversely affect the accuracy of the PAV results by not Xing some operations correctly.

1.4.2.1.3 INTC_MEM_ATPG_SIMPLE

This define turns on the SIMPLE mode for the ATPG model. It features reduced functionality relative to the full functional ATPG model.

1.4.2.1.4 INTEL_EMULATION and INTEL_FPGA (INTC_MEM_<fubname>_EMULATION and INTC_MEM_<fubname>_FPGA_MODE)

These two defines do the same thing in the SRAM BMOD. Using either define will switch the BMOD into an emulation or FPGA simulation model. The emulation and FPGA modes are checked with the EFFM tool for compatibility with all internally used Emulators and FPGA systems.

Both defines are included because SoC systems may have differences in how they respond to either the INTEL_EMULATION or INTEL_FPGA defines. Although the memory model is the same, the SoC may have major differences. Therefore the model is able to be switched into the appropriate mode without forcing the SoC to use an incorrect define.

Setting either of these defines will also internally set other defines needed to allow the model to work in emulation or FPGA modes. This provides a single define switch to place the model into the needed mode.

1.4.2.1.4.1 INTC_MEM_FAST_SIM

This define basically enables FPGA mode in order to switch the model into faster simulating code. This means that all hazards and many functional checks are disabled. The localized version of the EF_SIM define is also set automatically for this mode.

1.4.2.1.4.2 INTC_MEM_<fubname>_EF_MODE

This define is set automatically in either of the INTEL_EMULATION or INTEL_FPGA defines is used. It is not intended or recommended to be used by the end user. This defines controls internal model code that is common to both emulation and FPGA modes. This provides for a single define that can be used for emulation, FPGA, or both depending on the need. At the current time, the model does not have any differentiation between the emulation or FPGA modes, but this could change in future.

1.4.2.1.5 INTC_MEM_EF_SIM

This define enables the unit delay for the buffers in the clock pulse emulation found in the FPGA mode. This is necessary for the model to simulate properly in a Verilog simulation environment.

1.4.2.1.6 INTC_MEM_ESP

This define is automatically set by the ESP symbolic simulation equivalence tool and is used to control the BMOD for equivalence validation purposes. This define should never be set by the end user.

1.4.2.1.7 INTC_MEM_<fubname>_ESP_TASKS

This define is an internal define used by the model to respond to the setting of the INTC_MEM_ESP define. This define should never be set by the end user.

1.4.2.1.8 INTC_MEM_PATH_SIZE

This define sets the maximum path size for the pointer to the memory initialization file. The default is 256 characters and can be overridden with a defparam if needed.

1.4.2.1.9 INTC_MEM_ENABLE_GLS_XPROP

This define turns on X-propagation for SDF timing failures. This define is only applicable to GLS model.

1.4.2.1.10 FI (Fault Injection) defines

There are six (6) user controllable defines and on internal define that cause the BMOD to fault bits during a read process.

The first three (3) defines are global defines with the INTC_MEM_ prefix. These will cause all memories in the simulation to perform the requested type of fault.

The next three (3) defines are similar except they have the INTC_<fubname>_ prefix and therefore limit the fault injection to only the EBB name specified.

The final define is used internally to turn on the associated FI code if any of the other six (5) defines are used. The user must not use the enable define directly.

1.4.2.1.10.1 INTC_MEM_fault_norepair, INTC_MEM_<fubname>_fault_norepair, INTC_<fubname>_local_fi_nrep

These defines cause the BMOD to fault enough bits to represent an unrepairable fault condition.

1.4.2.1.10.2 INTC_MEM_fault_repair, INTC_MEM_<fubname>_fault_repair, INTC_<fubname>_local_fi_rep

These defines cause the BMOD to fault enough bits for a repairable and exhaustive fault condition.

1.4.2.1.10.3 INTC_MEM_fault_single, INTC_MEM_<fubname>_fault_single, INTC_<fubname>_local_fi_srep

These defines cause the BMOD to fault a single bit.

1.4.2.1.10.4 INTC_MEM_<fubname>_FI

This define is set if any of the FI modes is selected with the above FI defines This define should never be set by the end user.

1.4.2.1.11 Cycle Counting defines

The BMOD uses several defines for controlling the cycle counting feature that causes the model to X the array and outputs if the cycle count coming out of power down modes is not met before attempting memory access.

The defines come in two (2) flavors. One for global usage and the other for local (instance name specific) usage.

There is also a model controlled enable define that the end user does not need to control.

1.4.2.1.11.1 INTC_MEM_async_reset_count, <fubname>_async_reset_cnt

These defines cause the BMOD to count clock cycles after the async_reset input is released.

1.4.2.1.11.2 INTC_MEM_shutoff_count, <fubname>_shutoff_count

These defines cause the BMOD to count clock cycles after the power down input is released.

1.4.2.1.11.3 INTC_MEM_pshutoff_count, <fubname>_pshutoff_count

These defines cause the BMOD to count clock cycles after the periphery shutoff input is released.

1.4.2.1.11.4 INTC_MEM_arysleep_count, <fubname>_arysleep_count

These defines cause the BMOD to count clock cycles after the array sleep input is released.

1.4.2.1.11.5 <fubname>_ en_count

This define is set if any of the counting modes is selected with the above clock counting defines. This define should never be set by the end user.

1.4.2.2 BMOD plusarg(s)

The BMOD only contains one plusarg control, 'INTC_MEM_ENABLE_CLK_CHECKS'. Plusargs allow changing of the clock checks without requiring re-compile.

Table 6 *BMOD pluargs*

Define name	Action if defined / Description	Comments
INTC_MEM_ENABLE_CLK_CHECKS (plusarg)	Enable the clock reset checks	Clock reset checks are off by default

1.4.2.2.1 INTC_MEM_ENABLE_CLK_CHECKS plusarg

This plusarg enables the clock checking mode.

This mode disables the normal self reset feature of the memory clock and relies on external setting methods via pwrenb_in, ensleep, and/or async_reset to make sure there is an alternative clock reset path for silicon testing if needed.

This mode also checks for conditions that are likely to increase the probability to corrupt the internal clock such as assertion of pshutoff or clock high during the memory power low to high region. This helps to highlight potential connectivity issues in power on modes.

In this mode, the memory clock will not self-reset. This mode is recommended to be used early in the integration process. If any issues happen due to using this plusarg mode, the customer must contact CMO to discuss the results and determine if the issues can be safely waived. The determination to waive certain paranoia issues is dependant on memory usage in the particular SoC.

1.4.2.3 BMOD tasks

The BMOD contains three tasks used for setting the array and loading the array from a file. Two of these tasks are designed for internal BMOD usage, but can also be accessed from outside by the end user. The BMOD contains two arrays. One for the main content, and one for the redundant row elements. Most, if not all, user interaction with the array should be with the main array, not the redundant array. The Verilog array registers are named "array" and "array_r" respectively and reside in the <fubname>_array module hierarchy. Table 7 shows the tasks contain within the BMOD.

Table 7 *Array Setting and Loading Tasks*

Task name	Purpose	Usage	Location in BMOD	Comments
setarray	Set main array	setarray(val, range)	<fubname>_dfx_wrapper.<fubname> .<fubname>_bmod.<fubname>_array .setarray	val is the value to set, i.e. 1'bx range is the max address. The task starts at 0 and goes up to range. Only a single bit value is entered, all bits receive the same value.
setredarray	Set redundant row array	setredarray(val,range)	<fubname>_dfx_wrapper.<fubname> .<fubname>_bmod.<fubname>_array .setredarray	
setfiarray	Set Fault Injection array	setfiarray(va l,range)	<fubname>_dfx_wrapper.<fubname> .<fubname>_bmod.<fubname>_array .setfiarray	

Task name	Purpose	Usage	Location in BMOD	Comments
				Setfiarray is only available if a FI (Fault Injection) mode is selected.
INTC_MEM_INIT	Load from file	INTC_MEM_INIT(filename)	<fubname>_dfx_wrapper.<fubname>_upf_wrapper.<fubname>.<fubname>_bmod. INTC_MEM_INIT	The INTC_MEM_PATH_SIZE define defaults to 256 characters and can be overridden via defparam if a longer or shorter file path size is required.

1.4.3 BMOD file structure

The BMOD is contained in one single file and is composed of all necessary modules and macros needed for its function. All macros and modules are properly uniquified in order to allow multiple configuration memory models to be used at the same time without conflict.

CMO memory models are not designed to be loaded using the Verilog –y, or library, mechanism due to the nature of having multiple modules per file. This single file feature is a requirement from several memory compiler customers and cannot be changed. Make sure to load memory behavioral models directly to avoid loading issues.

The elements in the BMOD file come in the following order:

Table 8 *BMOD file element order*

Element	Description/Comments
Header comments	
The library type modules and macros	Behavioral versions only, not for synthesis
<fubname>_* modules	Used in main module
<fubname>_bmod	Main RTL module
`celldefine	All elements fall within the celldefine block at instantiation time
<fubname>	Top level module, optionally includes specify block (timing wrapper for GLS) depending on model type used. See section 1.3 for model types delivered.
`endcelldefine	End of celldefine block

1.4.4 BMOD model hierarchy structure

Figure 1 shows the hierarchy structure of the BMOD. The BMOD hierarchy has been built to present the top level module with the name of the FUB and with a consistent hierarchy mapping regardless of the model type used, section 1.3. This means that switching out between GLS versus non GLS capable views will not result in a hierarchy change that could require user code changes. I.e. the memory array hierarchy path will remain the same. The DFX wrapper, section 1.6, is wrapped around everything in Figure 1. The DFX and UPF wrapper has been included in the diagram for completeness.

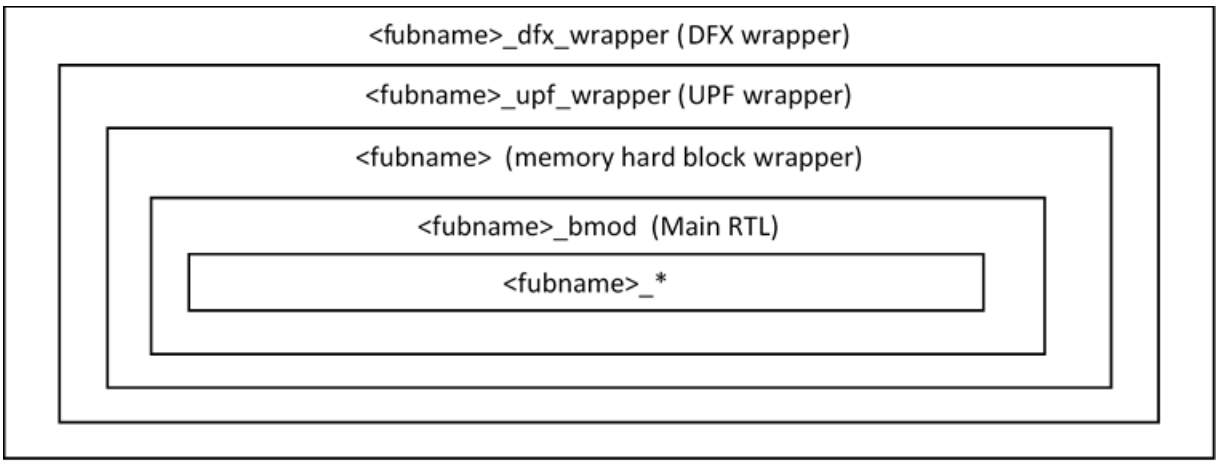


Figure 1: BMOD Hierarchy

1.4.5 BMOD Power aware simulation (UPF/PAV)

Please refer to the integration guide for all power supply connections and details. The BMOD runs in VCS NLP power aware simulations using UPF. UPF power intent files are provided for proper X-propagation with power domain shutoff conditions in a VCS NLP power aware simulation.

UPF is not supported in Gate-Level Simulations (GLS). GLS requires PAV.

Other Verilog tool simulations using UPF files are supported, but may not be fully tested due to lack of infrastructure tools. CPF (Common Power Format) is not supported.

All UPF power intent files follow the Intel internal UPF WG standards.

Information on Intel UPF power intent standards can be found at:
https://wiki.ith.intel.com/display/hdk/UPF+Standards

Single rail power gate configurations feature functional inputs on an internal gated supply. This internal gated supply does not have a supply port. Therefore, the gated supply is not available according to standard UPF integration methods – rather a virtual connection must be made. SoC integrators must handle this special case when integrating these memories. Detailed instructions from UPF WG can be found at:
https://wiki.ith.intel.com/display/P17X/UPF+Requirements+for+RFs+with+Gated+Supplies

Please note that the VCS default simulation options can affect the behavior when running power simulations. The VCS simulation options can vary across tool versions. Power simulations may also behave differently based on the IEEE 1801 UPF versions. For example, different VCS versions may have different default options that affect continuous and constant assignments in modules.

Information on the VCS simulation options can be found at:
https://soco.intel.com/docs/DOC-2450800 -> LP_switch_usage.xlsx

To use UPF in a power aware simulation, you must define the “INTEL_NO_PWR_PINS” tic define which eliminates the power pins from the interface.

The files included for UPF simulation are as follows:

1. <fubname>.upf
2. global_upf.cfg

All UPF files require the global upf config file: 'global_upf.cfg'. Per UPF WG standards, SoC should have a top-level 'global_upf.cfg' file which contains the appropriate settings for the SoC power integration. The lower-level global config files will not be loaded after the top-level file is loaded.

Variables for power supply levels are used in the UPF Power State Tables. These variables are specified in the 'global_upf.cfg' file.

The power supply level variables used are:

1) VCC_SUPPLY_HV
2) VCC_SUPPLY_LV
3) VSS_GROUND

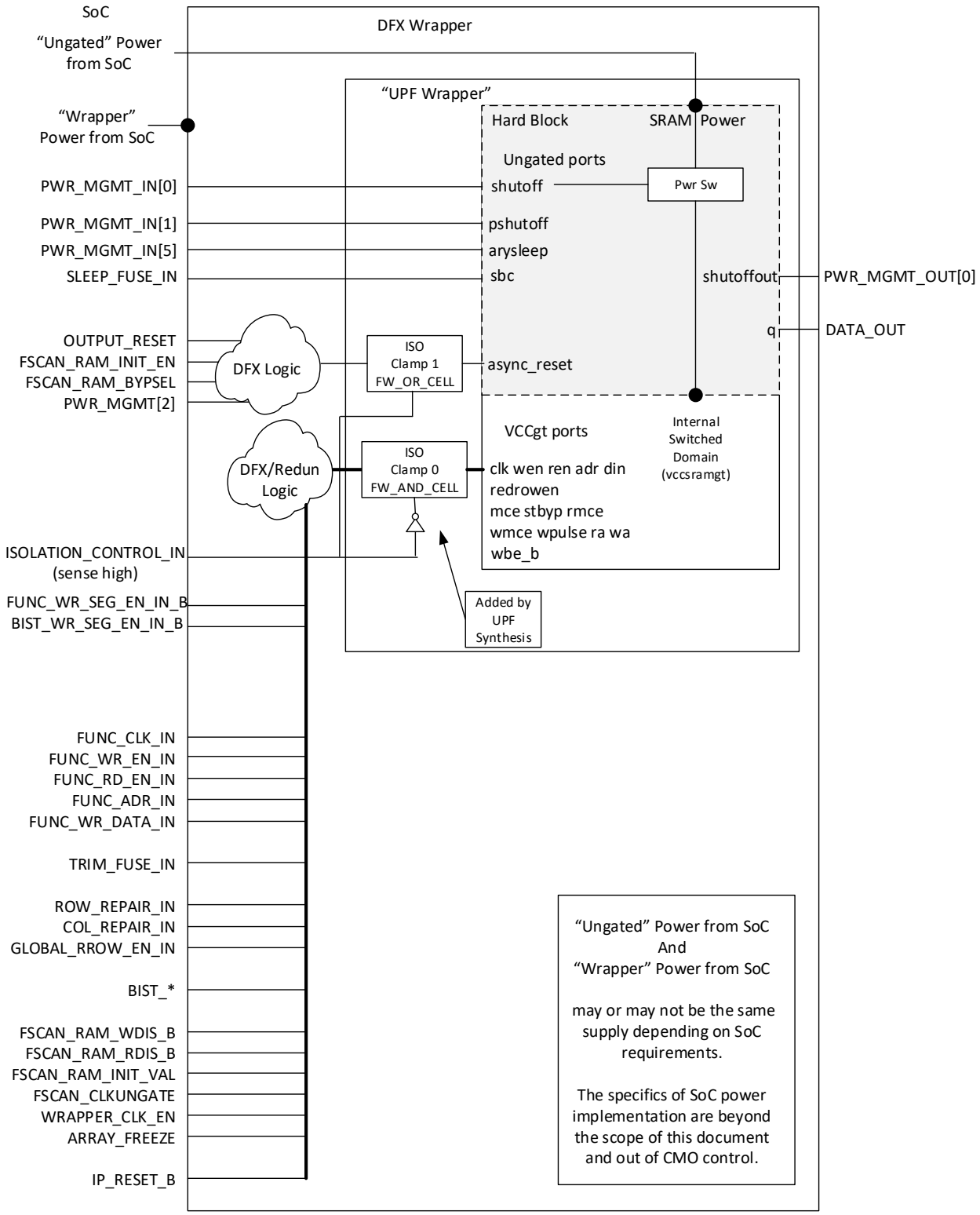
Versions of the global config files can be found at:
/nfs/site/disks/ccdo.soc.tools.000/proj_tools/power_templates/cds/<version>/

To run in a non-UPF Power Aware Verilog (PAV) simulation environment, the "INTEL_NO_PWR_PINS" tic define must not be defined to add the power pins to the model interface. If the vss power ground signal is also required to be on the interface, the "INTC_ADD_VSS" tic define must also be set. Without UPF, the Verilog BMOD connects to the power pins and will support full X-propagation for power domain switching using PAV methods.

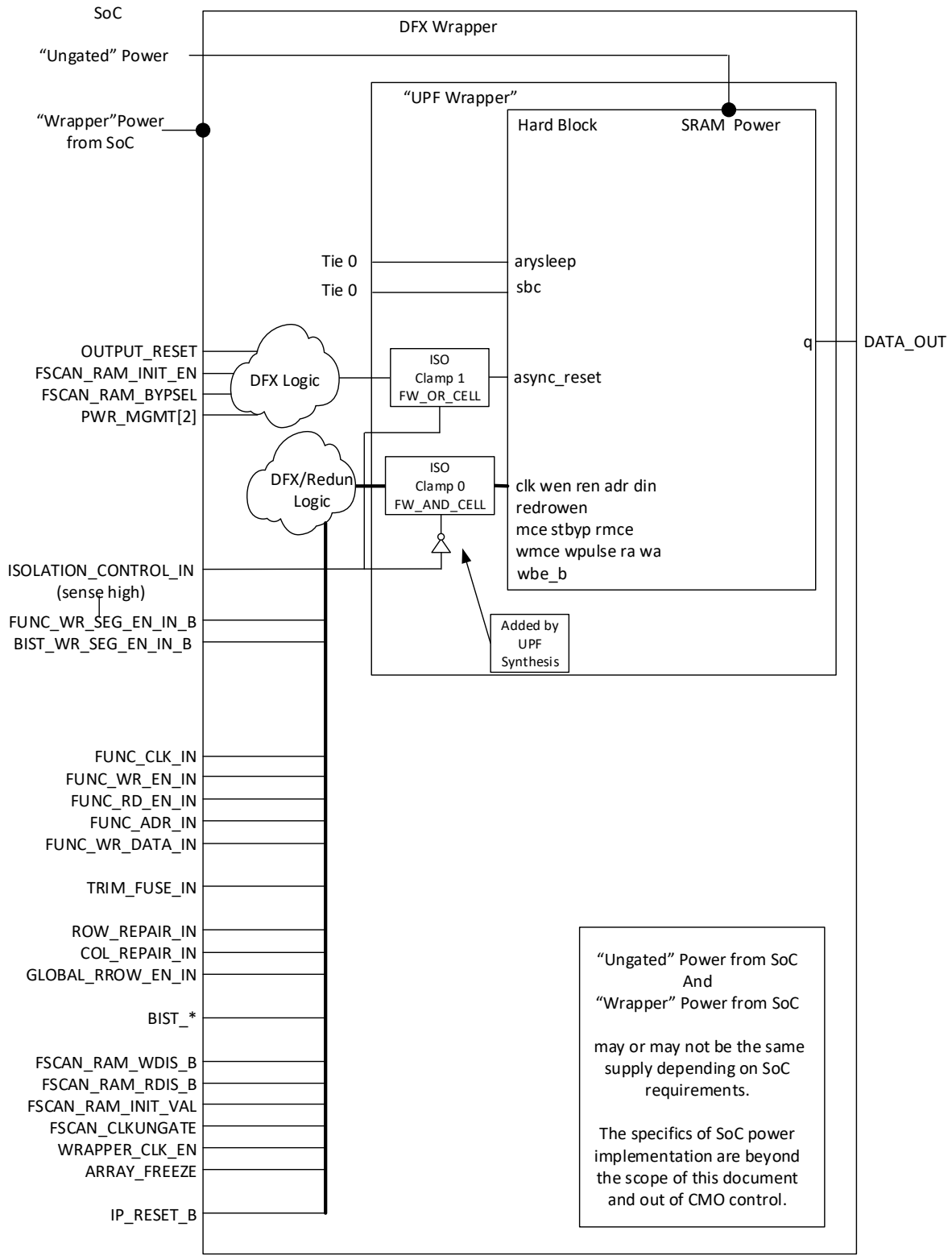
1.4.5.1 Memory Hard Block Isolation Wrapper

The following diagrams show different single power configurations with and without isolation wrapper.

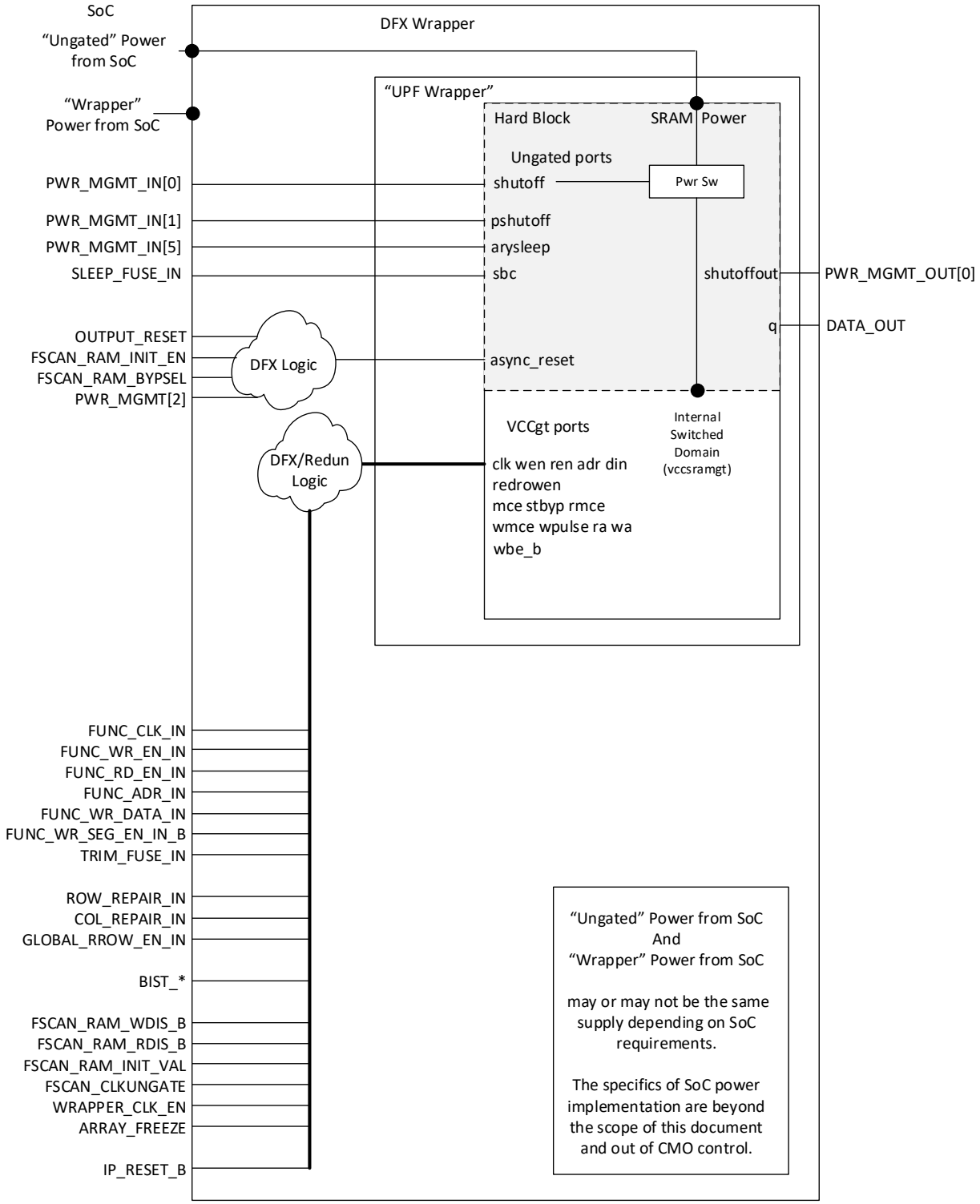
Single Rail Power Gate configuration w/Isolation



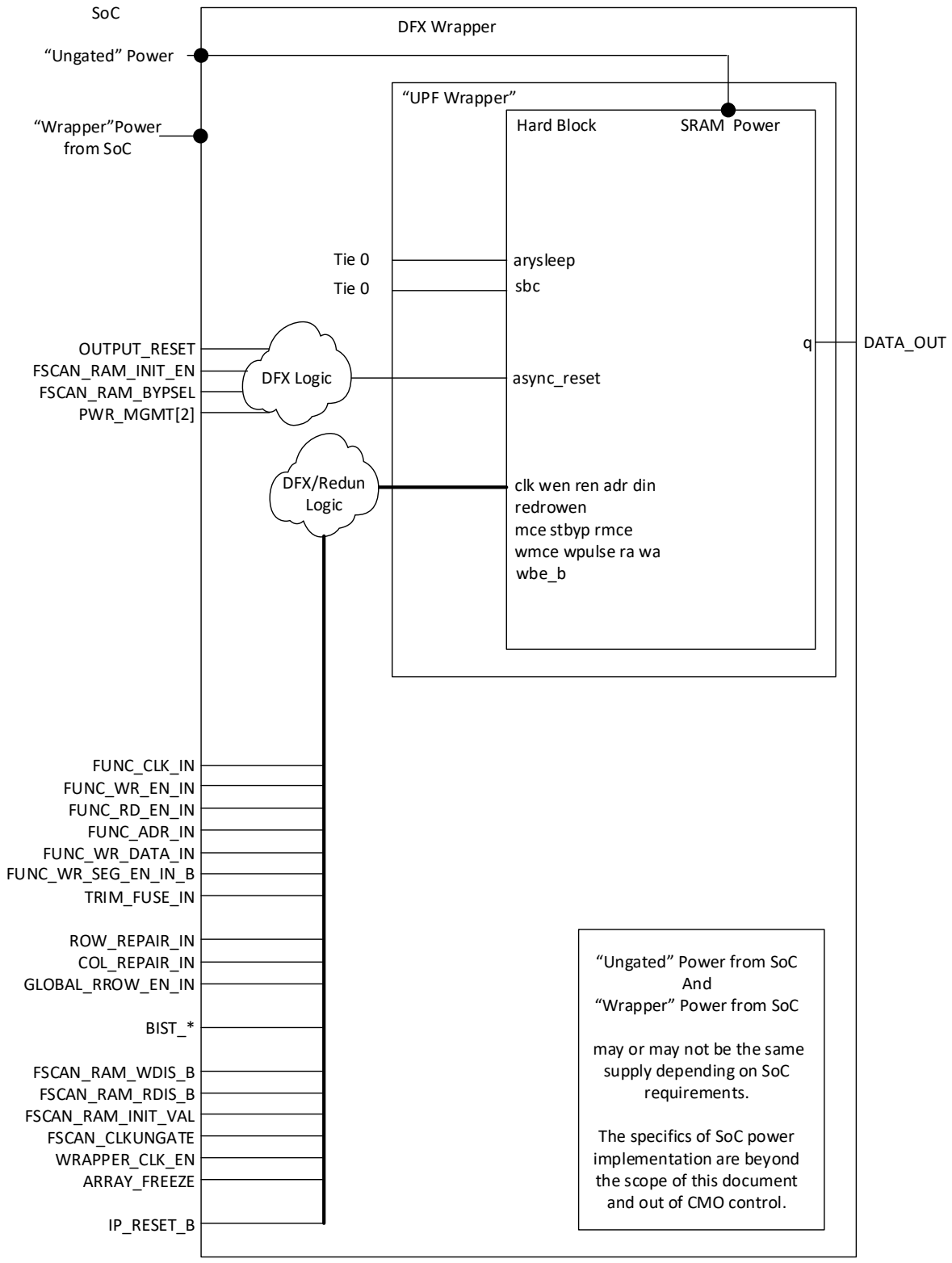
Single Rail No Power Gate configuration w/Isolation



Single Rail Power Gate configuration NO Isolation



Single Rail No Power Gate configuration NO Isolation



1.4.6 RTL based Clock Counting for power modes

The macros for using the clock counting feature are as described in section 1.4.2.1.11. They are listed here for reference.

Memory configuration unique macros:

1. <fubname}>_async_reset_count
2. <fubname}>_shutoff_count
3. <fubname}>_ pshutoff _count
4. <fubname}>_ arysleep _count

Global macros:

1. INTC_MEM_async_reset_count
2. INTC_MEM_shutoff_count
3. INTC_MEM_ pshutoff _count
4. INTC_MEM_ arysleep _count

The configuration specific (local) macro number will take precedence if the global macro is also defined. That way you can set most memories with a single value and tweak the rest as needed.

If the macros are not defined, then the code for this feature will not activate and simulation will happen as before. If used, each macro cannot be left empty, i.e. simply defined. Each macro must have an integer number assigned to it. The usage is simple in that you simply assign a number of clocks to count before access can be achieved. The error reporting is simplistic and a relatively large hammer. There will be a display error explaining the failure, and also a corruption of either the array (in case of a write), or corruption of the output (in case of a read). You can use none, one, or any number of the macros at the same time.

Example command line option to set one of the macros:
+define+INTC_MEM_async_reset_count=25

Setting any macro will enable the counting mechanism for that signal, and leaving them all undefined will disable the feature. The feature will also be disabled for EFFM and Lintra checking modes.

Setting the macros to 0 should not assert a failure, but will still activate the code for this feature and use up simulation resources. So, leaving them undefined is the best option for disabling them if they are not needed.

Please note that due to the uncertainties of Verilog results in 0 delay simulations with signals toggling at the same time that there might be a +-1 count discrepancy in pass versus fail. Therefore It is suggested to consider the individual situation carefully and increment the count to check by 1 if you want to be on the safe side. This is especially important when the number of clocks to count is reaching the low end, i.e. 2 or 1, but this general guideline applies at any number of clock counts.

1.4.7 Fast Simulation Option

In order to allow for faster simulation with reduction in hazard and other checking features, the user can set the INTC_MEM_FAST_SIM tic define. This will effectively place the BMOD in FPGA mode, which simulates faster, due to both changing the model to a posedge based design for EFFM synthesys and

removal of the otherwise costly hazard and other checking features of the mode. The following table shows some of the options and settings that can be used for various simulation objectives.

Table 9 *BMOD controls for Simulation Objectives*

	Simulation objective	What matters most	Recommended mode and defines

1	RTL development, debug	Basic memory function, fast simulation	• INTC_MEM_FAST_SIM
			• Optionally add
			  ○ INTC_MEM_NOXHANDLING
			  ○ INTC_MEM_COV_OFF
			  ○ INTEL_NO_PWR_PINS, but do not use UPF
			  ○ INTC_MEM_ENABLE_CLK_CHECKS plusarg
			
			Do not use INTC_MEM_LINT as it will break the model functionally:
			
			Notes:
			• INTEL_EMULATION and INTEL_FPGA do different things, but both set uniquified versions of INTEL_SVA_OFF **and** INTC_MEM_TESTMODE.
			• NTC_MEM_FAST_SIM will not set the uniquified versions of INTEL_SVA_OFF **or** INTC_MEM_TESTMODE **or** INTC_MEM_NOXHANDLING **or** INTC_MEM_COV_OFF like the INTEL_EMULATION and INTEL_FPGA switches do.
			• INTEL_SVA_OFF turns off the System Verilog Assertions
			• INTC_MEM_TESTMODE turns off some X generation hazards for corner cases. (These hazards are generally not Emulation/FPGA friendly.)
			• INTC_MEM_COV_OFF turns off coverage code in DFX chassis 2.1 wrapper.
			• INTEL_NO_PWR_PINS removes the power pins and also disables PAV (Power Aware Verilog). UPF would be required to obtain power simulation capability in this case. Without INTEL_NO_PWR_PINS PAV is active by default.
			• INTC_MEM_NOXHANDLING turns off Xing of the array for X propagation for X on clock, write and read inputs..
			
			Do not use INTC_MEM_LINT as it will break the model functionally:

2	Low power features: RTL development & debug	Low power modes behavior, reasonable sim speed	• INTC_MEM_FAST_SIM • Optionally add ○ INTEL_NO_PWR_PINS with UPF file

			or Leave power pins in and use PAV (PAV is required for GLS) ○ INTC_MEM_NOXHANDLING ○ INTC_MEM_COV_OFF ○ INTC_MEM_ENABLE_CLK_CHECKS plusarg Do not use INTC_MEM_LINT as it will break the model functionally:
3	Signoff RTL simulation	Signoff quality accuracy, checks	Notes: • Do not use any of the above mentioned `defines with the following exceptions. The default is a fully functional and spec compliant model. 1. One exception is INTC_MEM_GLS if you are running SDF annotation and are using the GLS version of the BMOD. 2. PAV/UPF is user choice depending on whether running GLS simulations or not. 3. Use the INTC_MEM_ENABLE_CLK_CHECKS plusarg at least once before final sighnoff and get CMO approval for any waivers.
4.a	Gate simulation (no timing)	No unreasonable X propagation	Use the “GLS” version of the BMOD, <fubname>.sv. Use • INTC_MEM_GLS (enables timing) Optionally use • INTC_MEM_COV_OFF • INTEL_SVA_OFF • INTC_MEM_TESTMODE • INTC_MEM_NOXHANDLING • INTC_MEM_ENABLE_CLK_CHECKS plusarg Notes: • Do not use INTEL_NO_PWR_PINS, PAV is on by default for GLS compatibility. • Do not use any other `defines. The default is a fully functional and spec compliant model.
4.b	Gate simulation (with timing)	Correct modeling of timing	Use the “GLS” version of the BMOD, <fubname>.sv. Use • INTC_MEM_GLS (turns on timing) Use SDF annotation. Optionally use • INTC_MEM_COV_OFF • INTEL_SVA_OFF

			• INTC_MEM_TESTMODE • INTC_MEM_NOXHANDLING • INTC_MEM_ENABLE_CLK_CHECKS plusarg Notes: • Do not use INTEL_NO_PWR_PINS, PAV is on by default for GLS compatibility. • Do not use any other `defines. The default is a fully functional and spec compliant model.
5	Verilog simulation of FPGA model or Emulation model		INTEL_FPGA **and** INTC_MEM_EF_SIM or INTEL_EMULATION Notes: • INTEL_EMULATION and INTEL_FPGA do different things, but both set uniquified versions of INTEL_SVA_OFF **and** INTC_MEM_TESTMODE. • INTC_MEM_EF_SIM enables the unit delay for the simulated self-time clock generation for the output latch. • INTEL_EMULATION does not need INTC_MEM_EF_SIM set in order to simulate. • Do NOT use INTEL_FPGA and INTEL_EMULATION at the same time. • Do NOT use INTC_MEM_EF_SIM for synthesis of the FPGA model. • Do not use the INTC_MEM_ENABLE_CLK_CHECKS plusarg. It is intended for non-EFFM simulation modes only.
6	Paranoia mode	Check for connection of test and clock reset control inputs	INTC_MEM_ENABLE_CLK_CHECKS plusarg Notes: • This is a plusarg so that re-compilaion is not needed to switch usage of the clock reset checks. This mode is only intended as a paranoia check for connectivity of reset paths for later Silicon debug, and also to check for potential clock and pshutoff activity that can be related to memory power instability during power up sequences.

1.4.8 BMOD Row Redundancy

Please refer to the Integration guide for more information on SRAM HB row redundancy in terms of timing, and physical layout. Please see the DFX model section of this guide, section 1.6, for full information on how row redundancy is controlled via the DFX wrapper. The SRAM HB itself only contains the extra row elements for repair, and redundant row enable input pins used to enable a desired repair row. Each redundant row enable pin selects a corresponding replacement row during a

read or a write access. The redundant row enable selection must be made for each access that the redundant replacement is required. Since the SRAM HB does not contain the address comparators for row redundancy, the DFX wrapper contains the address decoding for row redundancy and will provide the correct redundancy row enable inputs at the correct time to affect a row repair for the desired address. Only one select can be active at a time. If more than one redundant row enable is active for a given access time, both the output, and/or written data, should be considered undetermined.

1.4.9 BMOD validation

The BMOD is validated through two methods. The first method uses a SystemVerilog functional testbench which simulates the model in VCS and exercises the BMOD with the associated DFX wrapper through all specified functional modes of the DFX/BMOD combination. This testbench is intended to validate the BMOD's compliance to the specification.

The second method involves equivalence testing using the following tool. This method is used to validate that the BMOD and Spice netlist of the design are equivalent. Equivalence testing is also used to debug the BMOD and design in general. The BMOD is also run through ESP in the FPGA and Emulation modes.

1. ESP – Symbolic simulation/Formal equivalence

1.4.10 SDF annotation

SDF annotation is supported in the BMOD file. The BMOD file contains a wrapper with Verilog specify block and also contains buffers at the interface to provide a device for the SDF annotator to annotate. This is known as the "timing wrapper" and is sometimes referred to as the "GLS (Gate Level Simulation)" wrapper. That is because it is used mainly in gate level simulations where the entire SoC under simulation is annotated with realistic timing. The timing wrapper is enabled using the specified compiler directive.

Table 10 *BMOD files for SDF*

BMOD file used for SDF annotation	Define to turn ON SDF timing wrapper capability
<fubname>.sv	INTC_MEM_GLS

1.4.10.1 writesdf, SDF creation options

The PT_shell command and options used for validation of the ability to create a valid SDF file from the .lib file, and subsequently perform SDF annotation in a Verilog simulation, are in the following sub-section. Command and options for other tools other than the Primetime timing calculator may or may not be similar. If using another tool, care must be taken to find the right options to produce the required results. It is advisable to use pt_shell first to obtain a reference SDF file and then adjust the target tool's options to produce a similar SDF file.

Other timing calculators have been known to require greatly differing options. The discussion of the various tools and options is beyond the scope of this document.

1.4.10.1.1 Pt_shell command

The following is the command for pt_shell that writes out the SDF file. The -exclude checkpins option is optional since the .lib contains no checkpins. It is included in checks because some users report using this option. It is here to make sure the option does not cause harm. Other options will likely result in an SDF that does not properly match the specify block section of the BMOD.

> ***write_sdf -no_edge -include {SETUPHOLD RECREM} -context verilog -version 3.0 -exclude checkpins ${partname}_${PVT}.sdf***

1.4.11 Hazards (X-propagation)

The BMOD supports SVA and Verilog based hazards which also include X-propagation for those hazards. Again, SDF warnings are only available in the SDF enabled BMOD. Table 11 shows the list of hazards.

Table 11 *Hazards, X propagation and messaging*

Hazard	Functional effect / X propagation	Message Class	Messaging/Hazard Control defines/plusargs	Matching SVA
Multiple redrow enable	X read data X write data (both rows)	-E-	Disable: INTC_MEM_TESTMODE	Yes
Array collision, read/write at same time	X read data X write data	-E-	Diasble: INTC_MEM_TESTMODE	Yes
Address in memory hole	X read data	-E-	Disable: INTC_MEM_TESTMODE	Yes
Timing failure on functional inputs	None	-W- Controlled by SDF annotator	Simulator options	n/a
X-state on non-fuse inputs	Natural X propagation	-W-	Hazard messaging enabled with Enable: INTC_XINPUT_CHECK	No
X/Z on fuse inputs	None	-W-	No Disable	No
Power Domain Corruption	Handled by UPF/PAV	None	No Disable in UPF/PAV modes	No
Asynchronous PME signals and write/read operations	Output and Array Corruption	None	No Disable	No
SDF X-propagation on PME timing failures	Output and Array Corruption	-W-	Enable: INTC_MEM_ENABLE_GLS_XPROP	No

1.4.12 Known Issues

1.4.12.1 Self-time bypass test mode (BMOD RTL)

In the actual circuit, this test mode can be enabled using the ‘stbyp’ input. The self-time bypass input is considered a static input and should not toggle during regular functional operation. The test mode will disable the self-time circuitry of the internal write/read pulse which controls the wordline assertion. The internal write/read pulse will track the high phase of the clock in this mode allowing longer write/read

duration of bitcells. The memory output data will become valid after falling edge of clock in this mode. It is intended for burn-in applications.

Functionality for self-time bypass test mode is included in the BMOD. The test mode functionality in the BMOD is only used for equivalency with the circuit netlist. The only functionality modeled in the BMOD is the data output valid at falling edge of clock. Complex hazards and interactions with other modes are not modeled. Only functional hazards are modeled.

1.4.12.2 Data output X-state during PoR with async_reset initialized high (BMOD RTL)

Memory model hazard prevents data outputs from achieving logic low data state during SoC power-up where the memory 'async_reset' signal is initialized high at the start of simulation. Data outputs are held in X-state after 'async_reset' signal deasserts until next valid read.

1.4.12.3 VCLP OpenLatch violations (DFX wrapper RTL)

VCLP open latch violations can appear when trim and repair fuse latches are not bypassed. Violations can be waived since latches are modeled as transparent latches.

1.4.12.4 VCLP violations for gated power state (UPF)

UPF_CROSSOVER_NOSTATE, UPF_PORTSTATE_NOSTATE, PG_DATA_SUPPLY error codes most likely result from the lack of visibility of the memory gated power domain (vccsramgt_lv) associated with the functional inputs of single power rail power gate memory. The violations can be addressed by following UPF WG BKM referred in Section 1.4.5. Waivers for the violations are also an option if integrators can guarantee no isolation requirements exist for the functional inputs when the memory is power-gated. The potential isolation requirement covers entering, exiting and during memory power gate mode.

1.4.12.5 sglint violations for ctech cells in DFX wrapper

CMO does not use the map file for application of ctech cells. It is customer responsibility to choose correct ctech library and required cells.

```
waive -du {  {<fubname>_dfx_wrapper}  }  -msg {ctech_lib module 'ctech_lib_clk_gate_and' instantiated as '<fubname>_dfx_wrapper.clk_gate_and_en_sync' should be 'instantiated' only inside map file}  -rule  {  {60706}  }  -comment {}
```

1.4.12.6 sglint violations for bus mismatch in DFX wrapper

The bus width mismatch for the row repair bus is expected in order to meet the aliasing requirement for DFX Wrapper specification. Code is checking for valid row repair fuse specification allowed by the memory configuration.

```
waive -du {  {<fubaname>_dfx_wrapper}  }  -msg {For operator (<=), left expression:
\"row_repair_in_no_scan[(RROW_WIDTH - 1):14] \" width 12 should match right expression: \"(512 - 1)\" width 9 [Hierarchy: ':@<fubname>_dfx_wrapper:\gen_redrowen ']}  -rule {  {W362b}  }  -comment {}
```

The BYPSIZE mismatch is expected as a result of data replication (from prior compression) getting truncated when overfilling the data width bus size.

```
waive -du {  {<fubname>_dfx_wrapper}  }  -msg {For operator (<), left expression: \"BYPSIZE\" width 5 should match right expression: \"DATA_WIDTH\" width 6 [Hierarchy: ':@<fubname>_dfx_wrapper:\output_mux ']}  -rule  {  {W362b}  }  -comment {}
```

1.5 ATPG model (Fastscan)

The ATPG model is delivered in two files (split model). A top-level Verilog behavioral model file called "**<fubname>.atpg.v**" and a Fastscan core (array) model file called **"<fubname>.fslib**". The top-level model is written in standard Verilog code to model the memory functionality. The core model file is written in Fastscan model language and utilizes the _cram primitive to model the memory array. The ATPG split model can simulated in "full" functionality mode or "simple" functionality mode. A compiler directive is used to enable the "simple" mode. See compiler directives table.

The "full" model contains all behavioral functionality (except power management) of the SRAM.

The "simple" model contains limited behavioral functionality of the SRAM. The "simple" model does not contain power management, row redundancy or output latch functionality.

Customer would load both the top-level model with the fslib core model to run ATPG. The user is expected to constrain the ATPG model inputs appropriately based on the "full" or "simple" model requirements. A sample "fastscan.do.cat" file is provided in the atpg directory to show the constraints. Customer will need to unconstrain the 'redrowen' inputs in the constraints file in order to provide scan coverage for row redundancy.

Read/Write X propagation found in the BMOD is present in both ATPG model types ("full", "simple") and will therefore provide X output in Fastscan tests that attempt to read and write at the same time. This feature will allow Fastscan to avoid write/read collisions. X propagation for redundant row selection is also supported.

There is no ATPG model for the DFX wrapper since it is a synthesizable model that needs the synthesizer to perform auto scan stitching in order to obtain a gate level structural model that will be used by Fastscan directly.

1.5.1 ATPG model validation

The ATPG model is validated through two means.

The ATPG Verilog view is validated through ESP equivalence tests using the same ESP test suite used for the BMOD. In this case, the Verilog ATPG model is equivalence checked against the memory Spice netlist. This checks functional equivalence to the memory that cannot be achieved by running Fastscan only. ESP provides for %100 node coverage whereas Fastscan tests will only test what is needed to validate the stuck at coverage needed. This often leaves much of the Fastscan model uncovered.

Once the ATPG model is equivalence checked, the model is then tested in Fastscan. In Fastscan, the model is tested in four (4) different ways. The model is tested both with and without a scan capable wrapper around the memory, and in both serial mode as well as parallel mode. The result is four tests. Once Fastscan runs and creates the four corresponding Verilog testbenches, the BMOD (also equivalence checked against the Spice netlist) is run in those testbenches and checked for compliance.

Overall, this provides for a reasonably high confidence that the model is both functionally equivalent to the real memory, but will also work well in Fastscan test environments.

1.6 DFX Wrapper model

The DFX Wrapper RTL conforms to the FHAS Chassis specification known as Gen3 (Wave-4).

All information and questions should be directed to the DTEG Array WG. Contacts and links provided below.

1.6.1 Chassis DFX Resources

- DTEG Array WG
  - Contact(s)
    - Kaitlyn Chen (kaitlyn.chen@intel.com)
    - Michael Nelms (michael.r.nelms@intel.com) – DFX Wrapper
    - Pouria Bastani (pouria.bastani@intel.com)
  - Website
    - https://wiki.ith.intel.com/display/DTEG
    - Documents
      - https://wiki.ith.intel.com/display/DTEG/Docs
        - Wave4_DFX_Wrapper_FHAS.docx

- Zero-touch MBIST WG
  - Contact(s)
    - Kaitlyn Chen (kaitlyn.chen@intel.com)

- DTEG Scan WG
  - SoC Scan DFT Working Group
  - Contacts: Dale March (dale.j.march@intel.com), Jackie Cooper (jackie.m.cooper@intel.com)

- Chassis DFX WG
  - Website
    - http://goto.intel.com/ChassisDFx
    - goto/dfx

1.6.2 DFX wrapper Redundancy (Column/Row)

SRAM repair guidelines vary between projects based on a variety of factors such as unit memory size, process, operating voltage, temperature, and more and as such are beyond the scope of this document. This section briefly describes the compiler redundancy functionality for column and row. Each project should consult their QRE team for specific repair guidelines to meet yield and Vccmin distribution goals.

1.6.2.1 DFX wrapper Column Redundancy

SRAMs can have additional bits to support column redundancy. Column redundancy logic is implemented outside the SRAM using a soft wrapper; additional I/O(s) are added in the hard memory to support the soft wrapper. The bit shift logic for column redundancy implementation is shown below.

Each column redundancy implementation requires ($\log_2 D$ + 1) fuse bits; $\log_2 D$ bits represent the faulty column, these bits are decoded to generate D control signals to select the bit-shifting muxes. *D* is the number of data bits in the SRAM. The extra fuse bit is for column redundancy enable. For example a 2048x64 SRAM requires $\log_2(64) + 1 = 7$ fuse bits to implement a single column redundancy.

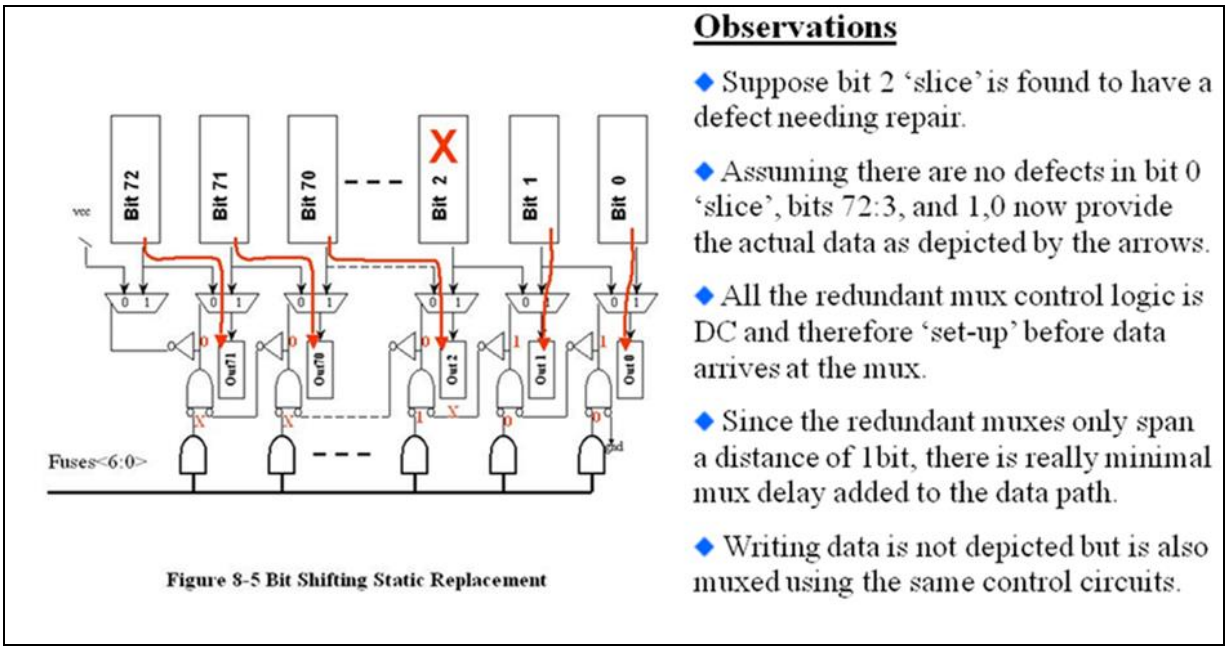


Figure 2. ***Bit Shifting Replacement Logic***

For the DFX wrapper, users can turn off column redundancy via a parameter, or order a memory without column redundancy, and implement their own logic for column redundancy outside the wrapper. The memory must be ordered as wide as needed to provide the extra column(s) worth of redundant elements that are used as redundant columns. Also, logic must be placed to implement the column redundancy scheme desired. Note that turning off the built in column redundancy logic for a memory that was ordered with column redundancy will NOT make the extra column in the HB available at the DFX wrapper interface! Also, turning on the built in column redundancy logic for a memory that was ordered without column redundancy will result in connection errors! Specifics on parameters used to control enabling of redundancy options can be found in Gen3 (Wave-4) DFX Wrapper specification (Wave4_DFX_Wrapper_FHAS.docx). One example for the need to implement column redundancy outside the DFX wrapper would be for sharing logic over many memories, resulting in a smaller logic footprint.

For unrepaired memory, all I/Os are capable of write/read operations permitting burn-in of all I/Os. For repaired memory with bit write capability, the replaced I/O will be blocked from write/read operation. For repaired memory without bit write capability, no I/O's will be blocked from write/read operations.

1.6.2.2 DFX wrapper Row Redundancy

If the SRAM supports row redundancy, a redundant row is selected, instead of the regular row, when one of the redundant row enable inputs, usually named "*redrowen[n:0]*" is active during a read or write access. The logic for generating the redundant row enable signals is implemented outside the SRAM, in the DFX wrapper, and is shown below.

The row redundancy logic in the DFX wrapper is user selectable via parameter, and therefore can be replaced with user supplied logic. Memory must be ordered with redundant rows to support this

option. One example would be for sharing logic over many memories. In this case, the DFX wrapper inputs 'GLOBAL_RROW_EN_IN' are passed directly to the memory hard block redundant row enables.

In cases where there is more than one instance of a given SRAM, more than 2 redundant rows can be supported by grouping the SRAM instances. In the example below, Figure 4, 4 rows can be repaired by grouping the 16 instances into two sets of 8 SRAMs each. Each set has double row redundancy which supports a total of 4 row repair capability. Each row redundancy requires a separate set of fuses.

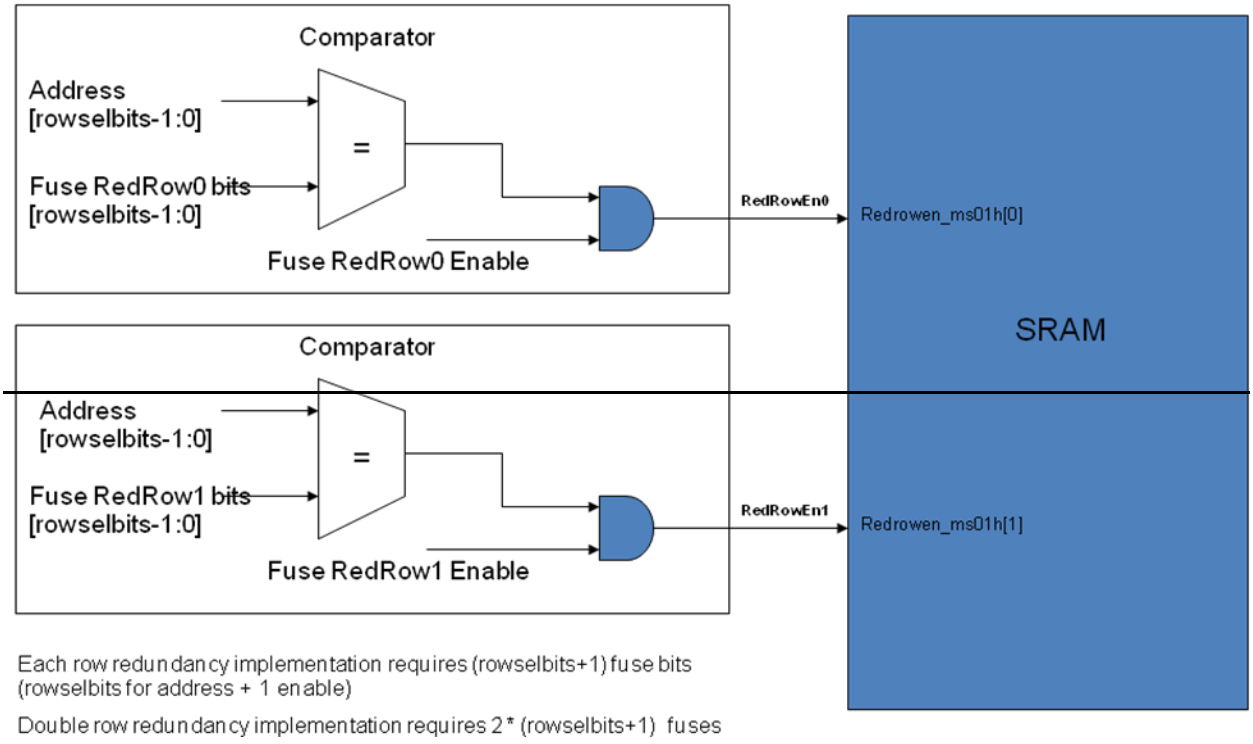


Figure 3. *Row Redundancy Implementation*

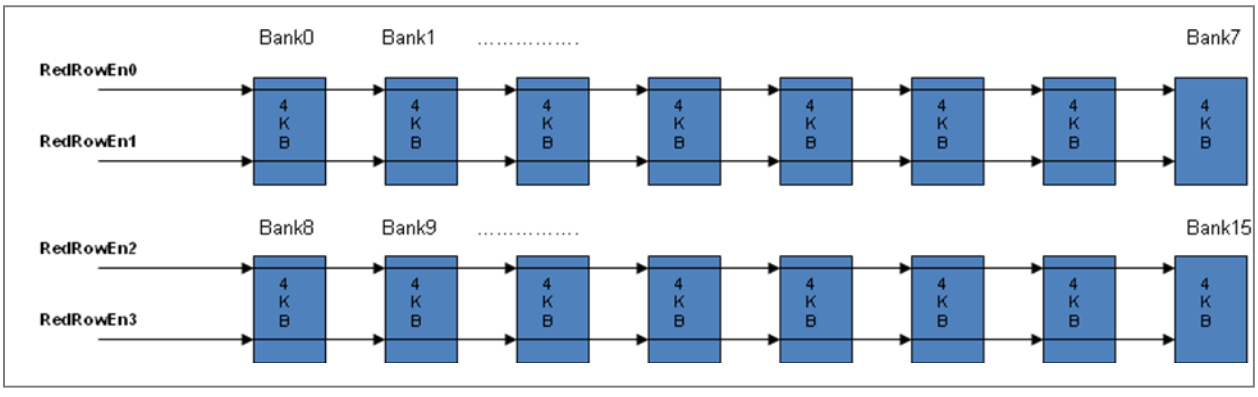


Figure 4. *Row Redundancy Implementation for multiple instances*

1.6.3 SRAM specific DFX information

This section will point out some of the specific SRAM related interface details.

1.6.3.1 DFX wrapper signals

The BMOD is instantiated by the DFX wrapper (internal UPF pin wrapper). Integrators should refer to the FHAS Wave4 DFX Wrapper specification (Wave4_DFX_Wrapper_FHAS.docx) for details on the standard DFX ports. Other signal connections that are not covered in the DFX wrapper specification are highlighted in **Table 12 *DFX Signal Mapping***.

1.6.3.2 PWR_MGMT_IN bus

The PWR_MGMT_IN bus is used for power management control. The PWR_MGMT_OUT signal is used to send the delayed power enable signal out for use in daisy chaining and staggering of the power up signal. Port options will vary by memory configuration. For example, the Non Power Gate option will lack the DFX PWR_MGMT_IN/OUT connections to the HB shutoff, shutoffout, and pshutoff inputs.

Please refer to the memory integration guide for clarification of the pin availability for the various configuration options.

Table 12 *DFX Signal Mapping*

DFX signal	BMOD signal	Comments
PWR_MGMT_IN[0]	shutoff	Array and periphery shutoff
PWR_MGMT_IN[1]	pshutoff	Periphery shutoff
PWR_MGMT_IN[2][1]	async_reset	Clock and output reset. See note 1
PWR_MGMT_IN[4:3]	unused	
PWR_MGMT_IN[5]	arysleep	Tied to 0 at BMOD instantiation [NPG fubs only]
SLEEP_FUSE_IN[1:0]	sbc	Tied to 0 at BMOD instantiation [NPG fubs only]
ROW_REPAIR_IN[25:0]	redrowen[1:0]	Row repair logic will set the redrowen inputs for memories with row redundancy when repair logic is enabled in the DFX wrapper
GLOBAL_RROW_EN_IN[1:0]	redrowen[1:0]	DFX signals will be connected directly to redrowen inputs for memories with row redundancy when the row repair logic is disabled in the DFX wrapper
TRIM_FUSE_IN[3:0][2]	rmce[3:0]	See note 2
TRIM_FUSE_IN[4][2]	stbyp	See note 2
TRIM_FUSE_IN[5][2]	mce	See note 2
TRIM_FUSE_IN[6:9][2]	***reserved***	Tie to 0 at DFX wrapper interface. See note 2
TRIM_FUSE_IN[10][2]	wpulse[2]	See note 2
TRIM_FUSE_IN[12:11][2]	wpulse[1:0]	See note 2
TRIM_FUSE_IN[14:13][2]	wmce[1:0]	See note 2
TRIM_FUSE_IN[16:15][2]	ra[1:0]	See note 2
TRIM_FUSE_IN[19:17][2]	wa[2:0]	See note 2
PWR_MGMT_OUT[0]	shutoffout	Delayed Power Enable Output

Notes:
1 – Signal is input to combinational logic.
2 - Signals could be latched based on DFX parameter setting (BYPASS_TRIM_LATCHES).

1.7 Repair information (<fubname>_REDUNDANCY_INFO.txt)

The <fubname>_REDUNDANCY_INFO.txt file contains the redundancy information and fuse mapping used for each individual SRAM memory instance.

1.8 LVLIB (Tessent MBIST) view(s)

A single LVLIB view for the DFX wrapper level is delivered with each memory. DFX wrapper level is where the BIST ports and other logic reside. The default LVLIB file matches the DFX wrapper when it is used without macro and/or parameter overrides.

1.8.1 LVLIB considerations when overriding DFX parameters

For internal deliveries, there is one consideration that must be made before using the LVLIB view. Redundancy repair logic will only be included in DFX wrapper if memory was originally ordered with those options. Repair parameters can disable repair logic only in this case. The ROW_REPAIR_IN and COL_REPAIR_IN repair busses on the DFX wrapper are always present and fixed size.

1.9 Fault Injection

The BMOD is capable of performing varoius fault on read operations which can be controlled by several defines. There are three types of define groups with a global and an IP specific version in each group. This fault injection method is also used in LVLIB redundancy mapping validation.

You can define 3 types of fault injection:

Repairable faults:

Available only for memories with Redundancy feature.
Inject faults in:
- one column only (if only column redundancy available),
- one row only (if only row redundancy available)
- or both row + column (both row and columns redundancy are present in the memory)

This fault type is consumes the total available redundancy elements in the memory

Non-repairable faults:

Available only for memories with Redundancy feature.
Inject fault in more that available redundant Rows + Columns

Single bit fault:

Available for all memories.
Inject a single bit fault into random memory address location

To inject repairable faults :

- *For specific instances :* ***+define+INTC_MEM_<ipname>_fault_repair***
- *For all memories :* ***+define+INTC_MEM_fault_repair***

To inject non-repairable faults:

- *For specific instances:* ***+define+INTC_MEM_<ipname>_fault_norepair***
- *For all memories :* ***+define+INTC_MEM_fault_norepair***

To inject single bit fault:

- *For specific instances :* ***+define+INTC_MEM_<ipname>_fault_single***

- *For all memories : +**define+INTC_MEM_fault_single***

Only one type of faulting can be active at one time. The defines are respected in the following priority order.

1. +define+INTC_MEM_<ipname>_fault_norepair (highest priority)
2. +define+INTC_MEM_fault_norepair
3. +define+INTC_MEM_<ipname>_fault_repair
4. +define+INTC_MEM_fault_repair
5. +define+INTC_MEM_<ipname>_fault_single
6. +define+INTC_MEM_fault_single (lowest priority)

A message similar to the following will be output whenever fault injection is turned on to indicate where the faults are being injected:

```
-----------------------------------------------------
-- Fault injection table for <instance path>
-- Type of fault : <Single|Repairable|Unrepairable>
-- Word <i>,   I/O <j>
-- Word <i>,   I/O <j>
…
-----------------------------------------------------
```

Section 2 – Revision History

Version	Date	Description
0.5.0	07/17/2019	Initial Version based on P1276 c76hsuspsr compiler
0.6.0	09/05/2019	Updated to include c76hduspsr compiler
0.6.1	9/05/2019	Minor text corrections
0.6.2	10/15/2019	Removed strikethrough for row redundancy
		Removed strikethrough for SDF X-propgation
0.7.0	05/18/2020	Added details for UPF 2.1.
		Clean up on multiple sections.
0.7.1	05/22/2020	Further updates wrt UV compiler release.
0.7.2	05/26/2020	Update of isolation block diagrams
		Added paranoia check sections
		General formatting cleanups
0.7.3	01/12/2021	Updated Diagrams to clarify SoC power domain connections and remove AON specifics
		Added special section related to implications of Single Rail Power gated configuration on UPF usage
0.7.4	01/23/2021	Changed details for upf isolation upf file from optional to reference only
0.7.5	07/26/2021	Updated Known Issues with four new items
0.7.6	05/04/2022	Added known issue sections 1.4.12.5 and 1.4.12.6 Removed references to the now defunct DFX and UPF wrapper RTL UPF files