IOSF SBC Endpoint Unit

VERIFICATION PLAN

ALL RTL 1.0-PICr35
January 2021

Contents

1 About This Document

1.1 Audience

The information in this document is intended to describe the verification strategy and execution plans for this IP. It is written as a Test Plan for the IP verification team, describing how the IP *is going to be* tested. The same document is published as a Verification Reference for a SoC team to see how the IP *was* tested.

1.2 References

If you need more information on this IP, you may find these documents helpful:

- SIP_IOSF_Sideband_Endpoint_productBrief
- SIP_IOSF_Sideband_Endpoint_integrationGuide
- Title of IP HSD link with applicable BUG/ECO/Issue queries
- Companion Verification Plans for the IP

1.3 Contact Information

Table 1. Contact Information for this IP

Name	Function	Email
Julian Aung	Verification Engineer/Manager	mailto:julian.aung@intel.com
Shwetha Bandari	Verification Engineer	shwetha.bandari@intel.com
Susann Flowers	Spec Template Owner	susann.flowers@intel.com

1.4 Terminology

Term	Definition

1.5 Document Revision History

Fill in the revision dates below for each revision level. Add a revision level if needed.

Table 2. Revision History of This Document

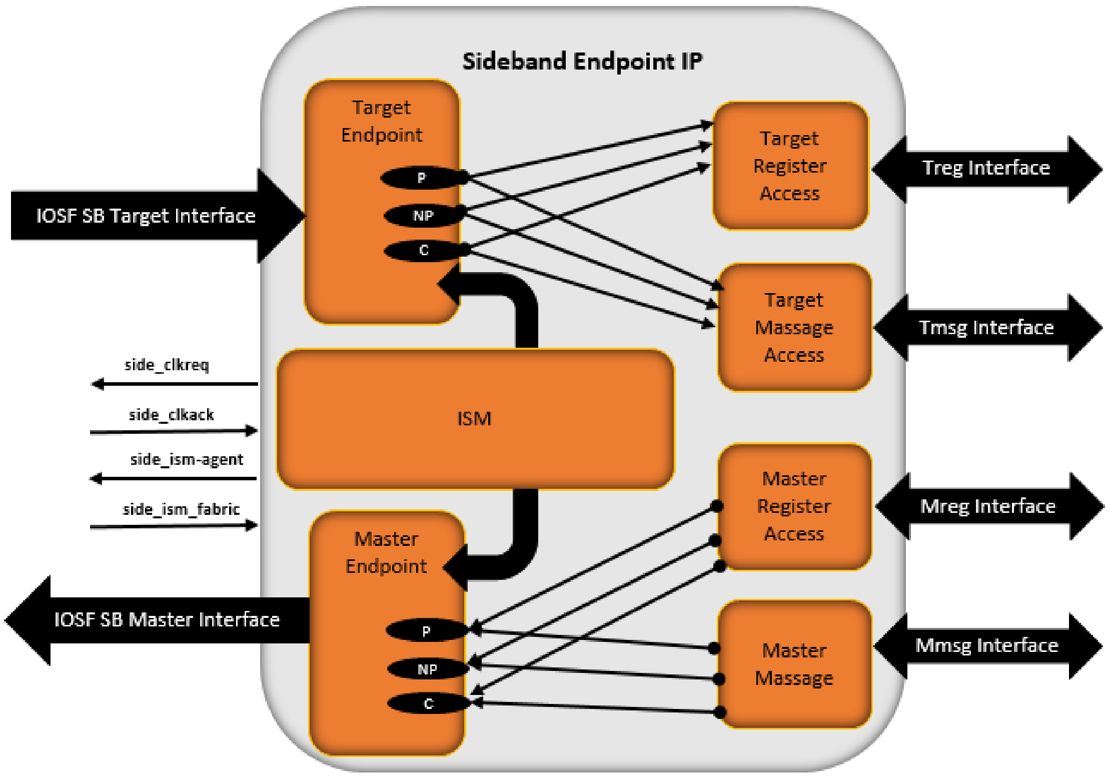
Author	Revision No.	Description	Revision Date
Dhwani Daftary	0	Initial Version	10 Nov 2009
Dhwani Daftary	1	Section 4	18 Dec 2009
Dhwani Daftary	2	Test Requirements updated for 0.9 spec	22 Nov 2010
Dipesh chauhan	3	All sections	4th May 2016
Julian Aung	4	Updated missing sections per Template Review	17 July 2020

2 Overview

2.1 IP Description

System or SOC contain multiple IP's, which uses IOSF Sideband Fabric to communicate with each other and IOSF Sideband Endpoint IP connect IOSF Sideband Fabric to multiple IPs. This IOSF Sideband Endpoint IP connected to Fabric using IOSF Sideband Massage interface and on other side it connected to Multiple IPs using Tmsg/Mmsg interface.It also provides optional register access target and master services in order to ease handling of the register access messages defined in the IOSF specification using Treg/Mreg interface.

Figure 1. IOSF Endpoint IP



This Endpoint IP has configurable payload width and configurable ingress queue depth.IOSF Sideband massage interface shown on left side contain unidirectional master and target pin interface to drive and receive fransaction from Fabric.Fabric uses Target pin interface to drives posted, non-posted transaction to target and also drive completion massage in response to Non-posted massage received on master interface. While Endpoint uses Master interface to drive posted, non-posted and completion massage to Fabric/Router.

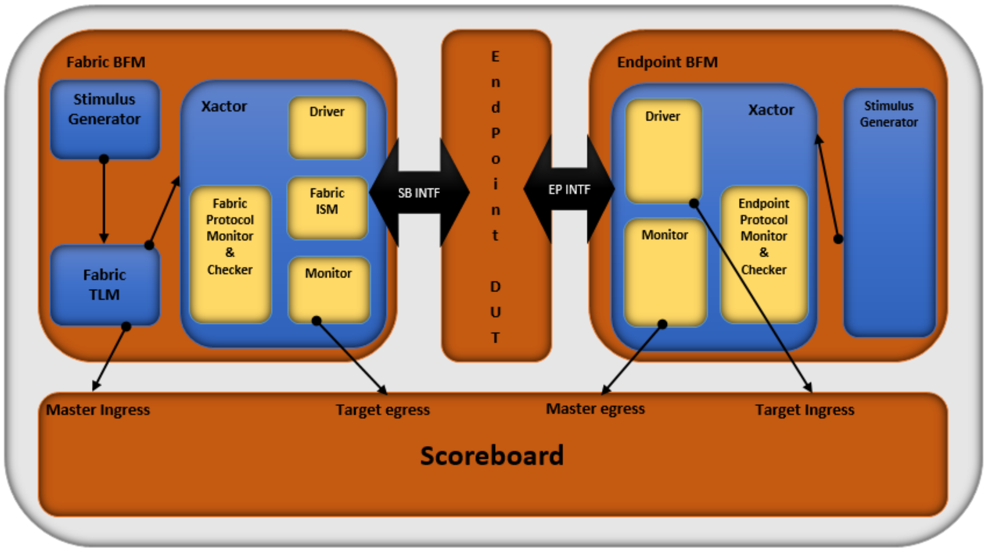
Apart from that there is credit signals, ISM sync up, power getting signals are on interface to communicate between fabric and endpoint.

While Right side on interface contain massage and register interface to communicate with individual IPs.

2.2 Testbench Description

This is a SystemVerilog testbench with re-usable OVM 2.0 base classes for the purpose of verifying the functionality of an IOSF Sideband Endpoint. The testbench features sequence generators, capable of driving constrained random, as well as directed stimuli, checkers of various types (Assertions, Protocol Checkers, Scoreboard), which verify that the DUT produces the correct response to the stimuli, and functional coverage which will ensure that all the desired features and conditions were stressed.This Testbench also include signal-level assertions, transaction-level interface checkers, and cross-fabric scoreboards. Configuration parameters control which IOSF options are stimulated, checked, and covered.

Figure 2. IOSF Endpoint Testbench



Fabric BFM is connected to Endpoint DUT though IOSF Sideband interface (SB INTF) which contain master and target port connections. Stimulus generator randomly generate Sideband massage or Register access operation command and drives to Fabric TLM. Fabric TML is responsible to drive transaction when ISM is in active state and same transaction is also driven to scoreboard ingress port. Based on credit available Driver drives Transaction on SB Interface while monitor samples driven and received transaction on SB interface. Monitor drives sampled transaction to scoreboard, protocol checker and functional coverage block.

Endpoint BFM is connected to Endpoint DUT on IOSF Sideband massage interface(EP INTF). Stimulus generator here generate posted/non-posted transaction and respond with complete transaction. Monitor samples driven and received transaction from EP interface. Using analysis port monitor drive relevant transaction to scoreboard, protocol checker and functional coverage block.

2.3 Verification Scope

Endpoint features are randomly configured and tested in addition to a set of existing configurations.

2.4 Dependencies/Assumptions

Sideband Endpoint verification environment uses the SVC and CCU BFMs.

3 Verification Environment

The IOSF Sideband Endpoint instance is connected to the test environment via test island of IOSF Sideband Fabric VC (SVC) on Fabric side and IP VC on Agent side. The test environment and RTL are configured by parameters which is specific to each project. Scoreboard compares egress message against ingress message for both master and target directions. The IOSF Sideband transactions are monitored and checked protocol by IOSF Sideband compliant monitor.

3.1 SoC-Specific Validation

IP specific usage model and different power state entry/exit validations are to be validated at SoC environment.

3.2 Validation Parameters

This section is documented in the Integration Guide (located in the 'doc' directory for this release).

3.3 Verification Libraries

This section is documented in the Integration Guide (located in the 'doc' directory for this release).

3.4 Testbench Components and Connectivity

This section discusses major validation structures included in drop, how they connect to the DUT, and how they're used.

The following subsections are documented in the Integration Guide (located in the 'doc' directory for this release):

- Testbench Directory Structure
- Test Islands
- Test Island Interfaces
- Checkers and Trackers
- Monitors
- Scoreboards
- BFMs
- Collage or Sandbox Files

3.5 IP Environment

The test bench follows the OVM methodology. The following agents are used in the IP environment.

OVM Component	Parameter Associated	Description
iosfsbm_fbrcvc	N/A	Iosf_sideband fabric VC

OVM Component	Parameter Associated	Description
iosfsbm_epvc	N/A	Endpoint VC
ccu_vc	N/A	CCU agent
ep_sb	N/A	Endpoint scoreboard, used to check endpoint ingress and egress traffic.

3.5.1 Environment Files

The sideband environment uses the following file:
`<IP>verif/bfm/sideband_vc/tb/env_ip.svh`

3.5.2 Configuring the IP Environment

Users can create an endpoint environment as follows:

```
//Create Endpoint env
env_i = iosfsbm_ip_env::env::type_id::create("env_i");
```

3.5.3 Saola Environment Walkthrough

Not applicable to this IP

3.5.4 Saola/RAL Components

Not applicable to this IP

3.5.5 System Manager

Not applicable to this IP

3.5.6 Fuse

Not applicable to this IP

3.6 Sequences

All sideband sequences are located in sideband BFM:

`<IP>/verif/bfm/sideband_vc/tb/seq_lib/`

3.6.1 Sequence for Bringing up the IP

Not applicable to this IP

3.6.2 BFM Sequences

The following table describes all Sideband sequences that can be used to generate specific types of transactions.

Sequence Name	Description	Parameters	Saola Phase
Base_seq	Base sequence, all sequence to target vc components should extend this sequence	n/a	Any phase apart from Power-On
Directed_seq	Enables sending of specific xactions through send_xaction API		
Rnd_seq	Sends generic constrained random xactions		
Simple_seq	Transaction Generator for simple xactions using set_fields API		
Msgd_seq	Transaction Generator for msgd xactions using set_fields API		
Regio_seq	Transaction Generator for regio xactions using set_fields API		
Polling_seq	Enables sending of xaction to get completion back using ovm request/response channel.		
Iosf_sb_seq	Enables sending any kind of xaction through send_xaction API		
Unicast_rnd_seq	This sequence send random unicast transactions.		
Loopback_seq	This sequence sends loopback transactions.		
Rnd_bcast_mcast_seq	This sequence sends broadcast and multicast transactions.		

3.6.3 IOSF Primary/Sideband BFM Sequences

Not applicable to this IP

3.6.4 Other Reusable Sequences

Not applicable to this IP

3.6.5 IP Test Sequences

Not applicable to this IP

3.6.6 SoC Requirements for Sequence Reuse

SoC could re-use the iosf_sb_seq sequence.

3.6.7 Sequence File Dependencies

Base sequence is located at `<IP>/verif/bfm/sideband_vc/tb/seq_lib/base_seq.svh`. All other sequence are derived from base sequence.

All sequences included in sequence library package `iosfsbm_seq_pkg`.

`<IP>/verif/bfm/sideband_vc/tb/seq_lib/iosfsbm_seq_pkg.sv`

3.6.8 Sequence Writing

IPs could re-use the iosf_sb_seq sequence.

3.7 Using the Runtime or Post-Processing Checkers

Sideband does not have any post processing checkers. The Sideband BFM has a Compliance monitor on the sideband interface to check behavior. The Sideband also has protocol-level checks inside the BFM. The Sideband monitors and scoreboard implements run-time checks.

3.8 Environment Settings and Files

3.8.1 Base Test

The base test is located at <IP>/verif/tests/ep_tests/base_test.svh. It instantiates the env_ip environment, and configures the components listed in section 3.5, IP Environment.

3.8.2 Configuration Object

The configuration objects are listed in section 3.5, IP Environment.

3.8.3 API

Not applicable to this IP

3.9 Description of Reusable Tests

Not applicable to this IP

3.10 Description of Reusable Automation Scripts

Not applicable to this IP

3.11 Supported Compiler Options for Simulation

Not applicable to this IP

3.12 Reusable Simulation RUNMODEs

Not applicable to this IP

3.13 Testbench Tools

Table 3. Testbench Tools

Tool	Version	Notes
VCSMX	J-2014.12-SP3-7	Simulation
IOSF Sideband SVC	2016WW10	BFM
Ace	2.01.23	
OVM	2.1.1_2	
questasim	10.0c	
Verdi	2012.01p2	

3.14 Testbench Utilities for Address, IP State, and Memory

There is no address, memory or IP State maintained in testbench.

3.15 Simulation Stages

During simulation, different seqs are chosen based on the test scenarios to send traffic on fabric side as well as ip side in a single phase. There is no hook or control provided to the phase since tests are not reusable to external.

3.16 Environment Setup and Test Run—Example

3.16.1 Environment Setup

First step to use VC is to instantiate the VC. In non-AVM/AVM environments, user needs to create environment object by extending ovm_env. Top level VC object need to be instantiated, configured by the environment object. In case there is AVM VC in the user testbench, there will be avm_env object as well. Simulators support multiple top level environment objects existence in the testbench. Here is an example code, which is a template for top level environment with only Fabric VC instantiated (tb/env/env_fabric_agent.svh):

Fabric VC is instantiated under ovm_env , VC configuration objects are declared as fields. Build function will create Fabric VC . Some of the test running setting are done in ovm_env as well, like stop simulation after getting x number of errors, message reporting verbosity level , watchdog timer for OVM.

3.16.2 Test Setup and Run

To start the testbench global task run_test() is called(line 169) in the previous code. Call to this task does following

- Determines which "test" to run – i.e. which class to instantiate as top level class
- Starts executing each component's phase functions
- After all the phases are completed, prints summary of OVM messages that were generated
- If ovm_top.finish_on_completion is set to 1 (default value), calls $finish.

In order to determine which class to instantiate as the top level class, run_test() picks from the following options. This top level class essentially is the test which will be run

- If a plusarg OVM_TESTNAME string is provided on the simulator cmd line, class type specified by OVM_TESTNAME is created(using factory)
- Else if the run_test() argument testname is not null, class type specified by the argument is created(using the factory)
- Else no class object is created having ovm_test_top global handle to be null

3.17 Testbench Output

3.17.1 Tracker

VC supports two tracker formats.

If MOAT compatible format is required then tracker_format needs to be passed to this API. User need to use open_tracker_file("file_name", "format", "agent_name", "Fabric_name") API from the test to enable the sideband VC tracker.

Usage of this is at `tests/back2back/test01.svh`

How to Open tracker file

- env_i.iosf_sbc_fabric_vc_i.open_tracker_file("IOSF_SB_AGT_TRK.log", SIP_FMT);
- env_i.iosf_sbc_agent_vc_i.open_tracker_file("IOSF_SB_FAB_TRK.log", MOAT_FMT);

3.17.2 OVM Arguments

Users can control some aspects of the VC, simulation by using vcs compile args, which need to be provided at the run-time. These options are as follows:

OVM_VERBOSITY:VCS uses standard OVM reporting infrastructure for reporting errors, warning, fatal, debug messages. Messaging severity is controlled through

"+OVM_VERBOSITY" switch. If OVM_DEBUG is used, all debug messages will be printed.

- OVM_NONE enables VERBOSITY_FATAL, VERBOSITY_ERROR and VERBOSITY_WARNING
- OVM_LOW enables OVM_NONE ,VERBOSITY_PATH,VERBOSE_TX_RX and VERBOSE_PROGRESS
- OVM_MEDIUM enables OVM_LOW and VERBOSITY_DEBUG_1
- OVM_HIGH enables OVM_MEDIUM and VERBOSITY_DEBUG_2
- OVM_FULL enables OVM_HIGH and VERBOSITY_ALL
- OVM_DEBUG enables OVM_FULL and OVM debug messages

4 Verification Strategy

The verification goal is to do pre Si testing on the IOSF Sideband Endpoint by the attached BFMs. There is a random regression consisting of semi random suite of configs and tests that targets every functional area of the IOSF Sideband Endpoint.

4.1 High-Level Verification Strategy

4.1.1 Methodology

The verification test bench method used is OVM.

4.1.2 Stimulus Strategy

Sequences and tests are used to verify Endpoint. There are some hooks config object to configure sequences. The stimulus is semi random. The sequences are tests are manually coded. There is no collateral required to generate stimulus other than the ones mentioned above.

4.1.3 Coverage Strategy

Describe the coverage strategy at a high level. Specify each type of coverage involved in the verification strategy (such as functional coverage, code coverage, and the like), along with any tools or infrastructure used to collect, store, or analyze coverage. For functional coverage, specify whether coverage will be collected on interfaces and/or internal design structures, and what types of metrics are used as part of the coverage methodology (for example, single event coverage, cross-condition coverage, frequency coverage, and the like.). If coverage is not used for feedback (such as with a directed-test methodology), specify that here as well.

4.1.4 Checking Strategy

Describe the high-level checking strategy used to determine the functional correctness of the IP. If applicable, include answers to questions such as:

- Is dynamic/runtime checking done or is it a post process?
- Are scoreboards or other testbench checking components used to verify correctness?
- Are assertions used in the RTL to detect and flag illegal behavior?
- Will external checking collateral be leveraged to complement the checking solution (that is, BFM checking or compliance monitors)?
- Are there any other mechanisms used for checking?

The main checking entities in the IOSF testbench are layered in four classes:

- Signal Level Checks – these are simple timing or value-matching checks, which will be implemented using System Verilog Assertions. These can be classified as related to:
- Endpooint Interface
- IOSF sideband Interface
- Transaction Level Checks – these are mainly protocol checks which make use of the transaction-level data provided by the monitor(s) on a IOSF interface and endpoint Interface:
    - Agent protocol checker

- Fabric protocol checker

4.1.4.1 Signal Level Checkers

SVA assertions and integrity checking.

4.1.4.2 Protocol Checkers

This is a class of elements that perform black-box verification only on a specific interface, without any knowledge of the higher-level picture.

Protocol checker basically checks for transaction integiry based on IOSF spec.

4.1.4.3 System Level Checkers

4.1.4.3.1 Scoreboard

Main task of scoreboard is to check IOSF Endpoint data-flow handling. Scoreboard connects with one monitor on Sideband Fabric Interface and Endpoint Intrface. It will receive packet transactions by TLM channels and keep it in master/target ingress fifos. Once transaction is received on the egress side, it will be compared with the transaction in the ingress fifo. If transaction is found it prints delets that message from the ingress fifo else prints error information.

At the end of the test it prints transaction summary, total number transaction processed by master and target interface and number of match/mis-match transactions.

4.1.5 Formal Verification Strategy

IOSF SB SVC provide compliance monitor which verify all rules stated in IOSF Sideband specificaions. Rules being categorise as transaction level, protocol monitor, power management, State machines, clock and reset etc. Assertions are being implemented to check rules on both side of iosf sideband massage interface, fabric and agent for credit complience, flow compliance, ISM complience, massage complience and clock getting complience.

4.1.6 Debug Strategy

IOSF Sideband SVC provide tracker files which gives details of transaction being driven and received by Fabric VC to RTL. It helps to find transactin sequence and response received over the period of time.

It also provide log information according to set verbosity to track down error, warnings, info massages and debug massages to trace.

4.1.7 Security Strategy

SVC provides API's to insert parity error, stall driver, inout depedancy etc to perform negative testing. This API being used by test case writer to enable and disable focused scenario or error insertions in their test sequence.

4.2 Verification Strategy for Areas of Special Emphasis

4.2.1 Reset Verification

All Rules stated in IOSF Sideband specification related to reset asserting and de-asserting and coreponding signal values being checked using formal verification.

4.2.2 Control Register and Fuse Verification

Not applicable to this IP/IPSS

4.2.3 Power Management Verification

Power aware simulation is being run making sure no transactions are dropped or corrupted.

4.2.4 Mixed Signal Verification

Not applicable to this IP/IPSS

4.2.5 Performance Verification

Transactions are streaming through while monitor is making sure no bubble between the flits.

4.2.6 Security Verification

Not applicable to this IP/IPSS

4.2.7 Safety Verification

4.2.8 Error Scenario Verification

Trasnaction stimulus items are driven with errors (parity error etc) and API's which help to create error conditions.

4.2.9 Design for Test (DFT) Verification

Not applicable to this IP/IPSS

4.2.10 Design for Validation (DFV) Verification

Not applicable to this IP/IPSS

4.2.11 Firmware Verification

Not applicable to this IP/IPSS

4.2.12 Software / Driver Verification

Not applicable to this IP/IPSS

4.2.13 Timer / Counter Verification

Not applicable to this IP/IPSS

4.3 Reuse Strategy

4.3.1 Local Reuse for IP Verification

IOSF Sideband Endpoint Verification component is highly configured SVC. Stand alone Endpoint IP uses Fabric VC and EP/IP VC on both side with all posible configurations.EP VC can have multiple instances. This Sideband verification component (SVC) can be used as passive component which montor interface and perform protocol checks.

4.3.2 Reuse of IP Verification collateral for SoC

IOSF Sideband SVC can be configured as Agent or Fabric and can be an active or passive mode.you can instatiate it multiple times in SOC according to requirment

5 Flows

There are no reusable validation collaterals from IOSF Sideband Endpoint verification environment at SoC level. Test Islands, interfaces and monitors (VCs in passive monitor mode) are reusable at SoC level. Refer to IOSF Sideband Verification Component (SVC) User Guide.

5.1 Bring-up Flow Details

Not applicalbe to this IP; see the description above.

5.2 Linkup/Down Flow Details

Not applicalbe to this IP; see the description above.

5.3 Reset Flow Details

Not applicalbe to this IP; see the description above.

5.4 Upstream/Downstream Traffic Flow Details

Not applicalbe to this IP; see the description above.

5.5 PM Entry/Exit Flow Details

Not applicalbe to this IP; see the description above.

5.6 Safety Entry/Exit Flow Details

Not applicalbe to this IP; see the description above.

5.7 Other Flow Details

Not applicalbe to this IP; see the description above.

6 Test Scenarios

Not applicable to this IP. There are no reusable tests from IOSF Sideband Endpoint verification environment at SoC level. Test Islands, interfaces and monitors (VCs in passive monitor mode) are reusable at SoC level. Refer to IOSF Sideband Verification Component (SVC) User Guide.

6.1 IP Integration "First Bring-up/Debug" Test

Not applicalbe to this IP; see the description above.

6.2 Register Access through RAL

Not applicalbe to this IP; see the description above.

6.3 PCIE Configuration Space

Not applicalbe to this IP; see the description above.

6.4 Register Access Policies and Attributes

Not applicalbe to this IP; see the description above.

6.5 Security Features

Not applicalbe to this IP; see the description above.

6.6 Safety Features

Not applicalbe to this IP; see the description above.

6.7 Datapaths

Not applicalbe to this IP; see the description above.

6.7.1 Upstream/Downstream Traffic

Not applicalbe to this IP; see the description above.

6.7.2 Various Transfer Rates

Not applicalbe to this IP; see the description above.

6.7.3 Other

Not applicalbe to this IP; see the description above.

6.8 Interrupt Verifications

Not applicalbe to this IP; see the description above.

6.9 Straps

Not applicalbe to this IP; see the description above.

6.10 Fuses

Not applicalbe to this IP; see the description above.

6.11 IP-specific Clocks and Reset Tests

Not applicalbe to this IP; see the description above.

6.12 Device and Function Disablement/Enablement

Not applicalbe to this IP; see the description above.

6.13 Sequences to Support SoC Power Gating and Flows

Not applicalbe to this IP; see the description above.

6.14 Sequences to Support SoC Reset Flows

Not applicalbe to this IP; see the description above.

6.15 Sequences to Support SoC Performance

Not applicalbe to this IP; see the description above.

6.16 Linkup/Down

Not applicalbe to this IP; see the description above.

6.17 Co-IP Validation Test Scenarios

Not applicalbe to this IP; see the description above.

6.17.1 Register Access through RAL to Co-IPs

Not applicalbe to this IP; see the description above.

6.17.2 Security Features and SAI Validation of Co-IPs

Not applicalbe to this IP; see the description above.

6.17.3 Safety Features and Validation of Co-IPs

Not applicalbe to this IP; see the description above.

6.17.4 Straps, Fuses, and Configuration of Co-IPs

Not applicalbe to this IP; see the description above.

6.17.5 Power Gating Verification of Co-IPs

Not applicalbe to this IP; see the description above.

6.17.6 Other Test Scenarios

Not applicalbe to this IP; see the description above.

6.18 Other Specific Functional Scenarios

Not applicalbe to this IP; see the description above.

7 Stimulus Details

7.1 IP Power Up and Reset

IOSF Sideband VCs need to be configured through Application Programming Interface (API) and using OVM config utility.

This section describes functions, procedure for configuring VC components, tlm port access for sending, retrieving xactions to/from the Fabric VC that can be called by tests or testbenches. Most functions are intended to be called once at the beginning of a test. As SystemVerilog functions, they execute in zero simulation time.

7.1.1 IOSF Spec Version

Since BFM supports multiple IOSF spec versions (0.81, 0.82, 0.83, 0.90, 1.0), user need to configure the BFM with the IOSF version they want to use to run the tests.. Configuration needs to be done through API from the test just once at the beginning. This is must otherwise BFM will use default spec version which is 0.82, Configuration API is:

```
fabric_cfg_i.set_iosfspec_ver(iosfsbm_cm::IOSF_082);
agent_cfg_i.set_iosfspec_ver(iosfsbm_cm::IOSF_082);
```

"Where iosfspec_ver value should be IOSF_081, IOSF_082, IOSF_083, IOSF_090 or IOSF_1”

7.1.2 Sideband Network Topology

VC needs to be configured with some information about the sideband network topology being verified. Sub-components of the VC use this information to generate legal sideband messages, build functional coverage database, and execute transaction level checks. OVM config utility is used here.

7.2 Initial IP Configuration—Example

Following are a description of config fields which are needed at configuration.

Config Field	Type	Description
payload_width	int	Payload bus width in Agent/fabric VC
my_ports[$]	bit[7:0]	Queue of pids associated with the Agent/Fabric VC
other_ports[$]	bit[7:0]	Queue of pids can be issued from the Agent/Fabric VC
mcast_ports[$]	bit[7:0]	Queue of mcast pids can be issued from the Agent/Fabric VC
supported_opcodes	bit[7:0]	Queue of opcodes can be issued from the Agent/Fabric VC

7.3 Dynamic Configurations and Injectors / User APIs

User can alter default VC configuration using APIs or OVM config utility as explained earlier. User can also update config fields at run time (During ovm run() phase).

Here is the list of config members that can be changed at run time.

Table 4. Config Fields that can be Changed during Run Time

Config Field	Type	Description
np_crd_buffer	int	Non-posted credit buffer size in Agent/fabric VC
pc_crd_buffer	int	Posted credit buffer size in Agent/fabric VC
crd_update_delay	int	Delay for the agent/fabric responder to update the credits
compl_delay	int	Delay for the agent/fabric driver to drive the completion
creditreq_delay	int	Delay to specify for how long agent/fabric can stay in credit_req state
creditinit_delay	int	Delay to specify for how long agent/fabric can stay in credit_init state
activereq_delay	int	Delay to specify for how long agent/fabric can stay in active_req state
creditack_delay	int	Delay to specify for how long fabric can stay in credit_ack state (Used only for fabric VC)
clkack_assert_delay	int	Delay before asserting clkack signal (only for fabric vc)

7.3.1 set_ep_cfg API (API of the agtvc_cfg and fbrcvc_cfg objects)

User will have to use this API if they change any of the config members (specified in Table 12) during OVM run() phase. Applicable only for agent and fabric config object.

<VC Config>.set_ep_cfg();

Note: Use this API only if config members specified in Table 12 has changed during run time.

7.3.2 set_agt_cfg API (API of the ep_cfg objects)

User will have to use this API if they change any of the config members (Specified in Table 12) during OVM run() phase. Applicable only for ep config object.

<VC Config>.set_agt_cfg();

Note: NOTE: We do not recommend use of this API.

7.3.3 set_fbrc_cfg API (API of the ep_cfg objects)

User will have to use this API if they change any of the config members during OVM run() phase. Applicable only for ep config object.

<VC Config>.set_fbrc_cfg();

Note: We do not recommend use of this API.

7.3.4 set_crd_update_delay / set_compl_delay API (Agent and Fabric VC)

User can change credit update delay, completion delay through API set_crd_update_delay and set_compl_delay.

Usage of this APIs is shown in the test is at tests/back2back/test02.svh.

User can also use set_cfg_crd_update/set_cfg_compl_delay API to set these delay values, these APIs are in the agtvc_cfg/fbrcvc_cfg objects.

7.3.5 set_compl_rsp API (Agent and Fabric VC)

User can also specify the completion response for the given opcode. set_compl_rsp API can be used to do this.

7.3.6 register_cb API (Agent and Fabric VC)

User can also register additional callbacks for other non-global opcodes using register_cb API.

7.3.7 register_user_cb API (Agent and Fabric VC)

User can also register user defined callbacks for global or non-global opcodes using register_cb API.

7.3.8 register_posted_cb API (Agent and Fabric VC)

User can also register user defined callbacks for global or non-global posted messages using register_posted_cb API.

7.3.9 enable_rnd_crd_reinit API (Agent and Fabric VC)

User can enable/disable random credit reinit using enable_rnd_crd_reinit API.

7.3.10 do_crd_reinit API (Agent and Fabric VC)

User can also initiate credit reinit using do_crd_reinit API.

7.3.11 load_compl_data API (Agent and Fabric VC)

User can also load memory with completion data and VC will use this data to send response for regio xactions. User will also need to set use_mem bit to 1 when fabric_cfg/agent_cfg is configured.

This API can be used from the test (Refer to test08) using
env_i.iosf_sbc_fabric_vc_i.load_compl_data(input iosfsbm_cm::pid_t dest_pid,

input iosfsbm_cm::flit_t cmpl_address[], input iosfsbm_cm::flit_t cmpl_data[]);

7.3.12 xaction_delay field usage (field of xaction)

User can specify delay between xactions by setting xaction_delay field before sending xactions.

Usage for this can be found under tests/back2back/test03.svh

7.3.13 expect_rsp field usage (field of xaction)

User can use expect_rsp field in the xaction to get response back in the test.

Usage for this can be found under tests/back2back/test05.svh

7.3.14 compare_completion field usage (field of xaction)

User can use compare_completion field in the xaction and can provide expected data along with the non_posted xaction regio xactions for VC to check completion data with the expected data.

Usage for this can be found under tests/back2back/test10.svh

7.3.15 set_pc_crd_init_delay and set_np_crd_init_delay API (Agent and Fabric VC)

User can use these APIs to set delay during credit initialization.

Usage for this can be found under tests/back2back/test04.svh

7.3.16 active_if_outstanding_np API (Agent and Fabric VC)

User can use this API to configure Agent VC such that agent will not move to idle if it has outstanding np messages.

Usage for this can be found under tests/back2back/test02.svh

7.3.17 set_creditack_delay API (Fabric VC)

User can use this API to indicate how long fabric ism can stay into credit_ack state before moving to credit_init state.

Usage for this can be found under tests/back2back/test01.svh

User can also use creditack_delay fabric config descriptor to set this delay or can use set_cfg_creditack_delay API to set credi_ack ism delay using fbrcvc_cfg object.

7.3.18 set_creditinit_delay API (Agent and Fabric VC)

User can use this API to indicate how long fabric/agent ism can stay into credit_init state before moving to IDLE/credit_done state.

Usage for this can be found under tests/back2back/test01.svh

User can also use creditinit_delay fabric/agent config descriptor to set this delay or can use set_cfg_creditinit_delay API to set credit_init ism delay using fbrcvc_cfg/agtvc_cfg object.

7.3.19 set_activereq_delay API (Agent and Fabric VC)

User can use this API to indicate how long fabric/agent ism can stay into active_req state before moving to active state.

Usage for this can be found under tests/back2back/test01.svh

User can also use activereq_delay fabric/agent config descriptor to set this delay or can use set_cfg_activereq_delay API to set active_req ism delay using fbrcvc_cfg/agtvc_cfg object.

7.3.20 set_creditreq_delay API (Agent and Fabric VC)

User can use this API to indicate how long fabric ism can stay into credit_req state before moving to credit_init state.

Usage for this can be found under `tests/back2back/test01.svh`

User can also use creditreq_delay fabric/agent config descriptor to set this delay or can use set_cfg_creditreq_delay API to set credit_req ism delay using fbrcvc_cfg/agtvc_cfg object.

7.3.21 set_np_crd_buffer_reinit / set_pc_crd_buffer_reinit API (Agent and Fabric VC)

User can use this API to change agent's credit buffer size during credit reinit. User need to user this API along with do_crd_reinit or enable_rnd_crd_reinit API

Usage for this can be found under `tests/back2back/test02.svh`

7.3.22 open_tracker_file API (Agent and Fabric VC)

VC supports two tracker formats.

If MOAT compatible format is required then tracker_format needs to be passed to this API.

User need to use open_tracker_file("file_name",

"format",

"agent_name", "Fabric_name", timescale_info, print_reset_state,

print_clock_state) API from the test to enable the sideband VC tracker.

Usage of this is at tests/back2back/test01.svh.

If user wants VC to print Clock and reset related information in the tracker then they need to set print_reset_state and print_clock_state inputs of this API.

7.3.23 Generate completion from test

Users who want to VC not to generate completion for non_posted opcode and want to generate completion from the test, they need to register that opcode to the VC indicating it is COMP callback, and then users can use iosf_sb_seq to send the completion for that non_posted message.

Usage of this is at tests/back2back/test11.svh.

For example, here VC will not generate completion for opcodes = 'hb9, 'h12 and 'h60. So not it is user's responsibility to generate completion for these opcodes (using same tag as the np request).

7.3.24 Use of ctrl_ext_header_support

For fabric_vc, users can send message with/without sai_header, in order to do that they need to set this ctrl_ext_header_support bit. This bit is added in case fabric receives message with/without sai_header from different agents. This bit will also customize the completion packets generated by VC.

If VC receives message with sai_header, VC will generate completion with sai_support and if VC receives message without sai_header, it will generate completion without sai_support.

7.3.25 Use of ctrl_rsp_per_opcode

For Agent/fabric_vc, If VC is set to use memory, then VC reads completion data from the memory and writes data into memory with read/write xactions. If VC is able to successfully read the completion data from the memory then it set completion response as successful for the completion message. If user wants to override this response field then they will have to use set_compl_rsp API to set the response they want to use and also will have to set ctrl_rsp_per_opcode config field.

So In the example above, Vc will generate completion for read messages based on the

response unsuccessful even if VC was successfully able to read the data from the memory.

7.3.26 Use of set_clkack_assert_delay (iosfsbm_fbrcvc API)

For fabric_vc, User can use this API to set delay before clkack is asserted. IF API is not used then VC will use random value generator by config descriptor.

There is similar API in the fbrcvc_cfg to set clkack_assert delay, set_cfg_clkack_assert_delay which user can use it from test at run time to set this delay.

7.3.27 Use of ext_headers_per_txn

For VC, Users can send message with/without sai_header from the test, to do this this config field needs to be set along with ext_header_support field.

When set users can use iosf_sb_seq to send messages with/without ext_headers. Refer to tests/back2back/test12.svh file for usage.

7.3.28 Use of set_compl_data_and_sai

For VC, Users can store completion data, completion response and sai header for each opcode for each port.

When this API is used, VC will use this completion data and sai when it sends completion back for non_psoted messages. This completion data , response and sai header is specified per opcode and per dest_pid.

Refer to the `tests/back2back/test12.svh` file for usage.

7.3.29 set_compl_sai_per_pid

For VC, Users can store completion sai header for for each port.

When this API is used, VC will use this completion sai when it sends completion back for non_psoted messages. This completion sai header is specified per dest_pid.

Refer to the `tests/back2back/test17.svh` file for usage.

7.3.30 get_mem_data

For VC, Users can store completion data for regio message into memory using load_compl_data or by sending write message.

If user wants to read this data back from the memory then this get_mem_data API can be used for that.

Refer to the `tests/back2back/test08.svh` file for usage. Users will need to provide dest_pid for which it wants to read the data back, address and be enable bits. VC will read the data back and will assign rsp field appropriately.

7.3.31 disable_compmon_assertion (for Agent/Fabric VC)

For VC, Users can selectively disable/enable compliance monitor assertion errors. Refer to the `tests/back2back/test12.svh` file for usage information.

Note: Here is the list of assertions that can be disabled /enabled.

ISMPM_046_AGENTMUSTENTER_IDLE_REQ, SBMI_096_100_MASTERHASSAISUPPORT
SBMI_SOMENUM_PMISSIZEVALID SBMI_060_MESSAGEUSESALLOWEDOPCODES
SBMI_062_TEOMVALIDFROMRESET SBMI_062_MEOMVALIDFROMRESET
ISMPM_002_STATETRANSITIONFROM_AGENT_ACTIVE_REQ_1
ISMPM_002_STATETRANSITIONFROM_AGENT_IDLE_2
ISMPM_002_ISM_INITIALIZATION_WITH_AGENT_IDLE
ISMPM_002_STATETRANSITIONFROM_AGENT_ACTIVE_2
ISMPM_002_ISM_INITIALIZATION_WITH_AGENT_CREDIT_REQ
ISMPM_002_STATETRANSITIONFROM_AGENT_ACTIVE_REQ_3
ISMPM_002_STATETRANSITIONFROM_AGENT_IDLE_REQ_3
ISMPM_002_STATETRANSITIONFROM_AGENT_ACTIVE_1
ISMPM_002_ISM_INITIALIZATION_WITH_FABRIC_IDLE
ISMPM_002_STATETRANSITIONFROM_FABRIC_IDLE_2
ISMPM_002_STATETRANSITIONFROM_FABRIC_ACTIVE_2
ISMPM_002_STATETRANSITIONFROM_FABRIC_ACTIVE_REQ_3
ISMPM_002_ISM_INITIALIZATION_WITH_FABRIC_CREDIT_REQ
ISMPM_SBMI_062_PRI_157_STATEINITIALIZATION_CLKREQ
ISMPM_015_CLKREQDEASSERTSONLYWHENAGENTISMISINIDLESTATE
SBMI_062_CLKREQVALIDFROMRESET

7.3.32 load_opcode_name API

Users can provide their own names for opcodes and VC will use that name to print in the tracker output.

<sideband vc instance>.load_opcode_name(.opcode(my_opcode), .name("my_name"));

7.3.33 waitForComplete and getData API

Users can use VC's iosf_sb_base_rsp_seq to use waitForComplete and getData xaction API. For posted messages, when a xaction is driven on the bus the complete bit of the xaction will be set. And for non_posted messages, when a completion is received by the VC for pending np message, the complete bit of the xaction will be set.

7.4 Dynamic Injectors—Example

See section 7.3, Dynamic Configurations and Injectors / User APIs.

7.5 Stimulus Generation—Example

Stimulus is generated through custom made list of sequences which can be found under seq_lib

7.6 Transaction Classes / Sequence Items—Example

This section describes objects used for building data structure which holds sideband messages to be driven by the VC to the DUT. Helper fields are used by the random constraints to generate legal transaction.

7.6.1 Xaction

This is the base transaction class. Transaction parameters can be used by the test writer. Each transaction can be of a type Simple, Register Access, Message with data and completions with data or without data belongs to POSTED or NON_POSTED type. By default, VC supports all global opcodes. Message types supported by VC are as follows:

Table 5. Transaction Types

Name	Type	Description
REGIO	GLOBAL : [0000_0000 : 0000_1111]	REGISTER ACCESS
	ENDPOINT-SPECIFIC: [0001_0000 : 0001_1111]	
COMP	GLOBAL : [0010_0000 : 0010_1111]	COMPLETION
	ENDPOINT–SPECIFIC: [0011_0000 : 0011_1111]	
MSGD	GLOBAL : [0100_0000 : 0101_1111]	MESSAGE WITH DATA
	ENDPOINT-SPECIFIC: [0110_0000 : 0111_1111]	
SIMPLE	GLOBAL: [1000_0000 : 1001_1111]	SIMPLE
	ENPOINT-SPECIFIC: [1010_0000 : 1111_1111]	

7.6.1.1 Configurable Parameters

Table 6 shows the configurable parameters which are common to all xaction types.

Table 6. Transaction Cinfigurable Paramters

Name	Type	Description
MSG[ ]	BIT[7:0]	INDICATES DATA CARRIED BY THE TRANSACTION
SRC_PID	BIT[7:0]	IT INDICATES THE PORT IDENTIFIER FOR THE SOURCE OF THE MESSAGE
DEST_PID	BIT[7:0]	IT INDICATES THE PORT IDENTIFIER FOR THE DESTINATION OF THE MESSAGE
OPCODE	BIT[7:0]	SPECIFIES OPERATION TO BE PERFORMED.
TAG	BIT[2:0]	SPECIFIES TAG FIELD OF THE TRANSACTION. IT IS USED AS AN IDENTIFIER OF THE REQUEST. IT IS USED ONLY BY THE SENDER TO ASSOCIATE COMPLETIONS WITH REQUESTS.
XACTION_DELAY	BIT[3:0]	SPECIFIES DELAY BETWEEN XACTIONS.
SP	BIT	SAI PRESENT BIT
SAI_FOOTER[]	BIT[7:0]	INDICATES 32 BIT SAI FOOTER

7.6.1.2 Helper Fields

Table 7. xaction Helper Fields

Variable	Type	Description
xaction_type	Xaction_type_e	Defines Transaction type of the xactions. Ex. Simple, register access, completion or message with data
xaction_class	Xaction_class_e	Defines transaction class of the xactions. Ex. POSTED or NONPOSTED
last_active	Time	Used by TLM_PC and Fabric Scoreboard for stale xaction detection
comp_count	Int	Used by TLM_PC to find errors. It indicates total number of received completions
exp_comp_count	Int	Used by TLM_PC to find errors. It indicates total number of expected completions
xact_id	Int	Set by test and used by tlm_pc to route completion to correct sequence using ovm standard response channel
expect_rsp	Bit	Set by test to wait for completion to return using ovm standard response channel
start_time	Time	Set by monitor when it sees the xaction on the bus
end_time	Time	Set by monitor when the xaction ends on the bus

7.6.2 regio_xaction

This is the register access transaction class which is extended from base xaction class representing regio write or read request sideband message. Supports generation of POSTED and NON-POSTED based on request type.

7.6.2.1 Configurable Fields

These fields can be configured through the random constraint.

Table 8. RegIO xaction Configuration Fields

Name	Type	Description
SRC_PID	BIT[7:0]	It indicates the port identifier for the source of the message.
DEST_PID	BIT[7:0]	It indicates the port identifier for the destination of the message.
OPCODE	BIT[7:0]	Specifies operation to be performed.
TAG	BIT[2:0]	Specified tag field of the transaction. It is used as an identifier of the request. It is used only by the sender to associate completions with requests.
BAR	BIT[2:0]	Specified bar field of the transaction. It is applicable only when accessing memory mapped or io mapped space. Value '110' and '111' are reserved and should not be used.
ADDRLEN	BIT[1:0]	Specified address length field of the transaction. Transaction can have 16 or 48 bit of address. Value '10' and '11' are reserved and should not be used.
FBE	BIT[3:0]	Specified byte enable for the first dw to be read.
SBE	BIT[3:0]	Specified byte enable for the second dw to be read.

Name	Type	Description
RID	BIT[7:0]	Specifies routing id field of the transaction. It is applicable only when accessing memory mapped, io mapped or configuration space.
ADDR[ ]	BIT[7:0]	Specifies address field of the transaction. It defines the internal address of the register within the device and related to bar#
DATA[ ]	BIT[7:0]	Specifies data carried by the transaction. It is valid only for the write register transactions. It can be of 32 or 64 bit long and decided by end of message signal.

7.6.2.2 Commands Supported

The opcode field is configured from the valid Global opcodes to generate memory, io or config write/read requests. Table 9 lists the commands that are supported.

Table 9. Commands Supported by regio_xaction

Command	Opcode bit[7:0]	Description
OP_MRD	0000_0000	Read Memory Mapped Register
OP_MWR	0000_0001	Write Memory Mapped Register
OP_IORD	0000_0010	Read IO Mapped Register
OP_IOWR	0000_0011	Write IO Mapped Register
OP_CFGRD	0000_0100	Read PCI Configuration Register
OP_CFGWR	0000_0101	Write PCI Configuration Register
OP_CRRD	0000_0110	Read Private Control Register
OP_CRWR	0000_0111	Write Private Control Register

7.6.2.3 Helper Fields

Helper fields help random constraints to generate valid transaction.

Table 10. regio_xaction Helper Fields

Variable	Type	Description
data_size_dw	Int	Indicates size of the data
addr_size_dw	Int	Indicates size of the address field

7.6.3 comp_xaction

This is the completion transaction class which is extended from base xaction class. All non-posted messages require completions. Completions are further classified into two classes – those that contain a data payload and those that do not.

7.6.3.1 Configurable Fields

Table 11. comp_xaction Configurable Fields

Name	Type	Description
SRC_PID	BIT[7:0]	It indicates the port identifier for the source of the message
DEST_PID	BIT[7:0]	It indicates the port identifier for the destination of the message

Name	Type	Description
OPCODE	BIT[7:0]	Specifies operation to be performed.
RESERVED	BIT[2:0]	Specifies reserved field of the transaction.
RSP	BIT[1:0]	Indicates the status of the completion, ex: successful, unsuccessful, powered down and multicast mixed status.
DATA[ ]	BIT[7:0]	Specifies data carried by the transaction. It is valid only for the completion with data transactions. It can be of 32 or 64 bit long and decided by end of message signal.

7.6.3.2 Commands Supported

Completion message related commands supported are: list in Table 12.

Table 12. Commands Supported by comp_xaction

Command	Opcode bit[7:0]	Description
OP_CMP	0010_0000	Completion without Data
OP_CMPD	0010_0001	Completion with Data

7.6.3.3 Helper Fields

Table 13. comp_xaction Helper Fields

Variable	Type	Description
data_size_dw	Int	Indicates size of the data

7.6.4 msgd_xaction

This is the message with data transaction class which is extended from base xaction class. Message with data can be wither posted or non-posted. All non-posted messages with data requests must be completed with a completion without data messages.

7.6.4.1 Configurable Fields

Table 14. msgd_xaction Configurable Fields

NAME	TYPE	DESCRIPTION
SRC_PID	BIT[7:0]	IT INDICATES THE PORT IDENTIFIER FOR THE SOURCE OF THE MESSAGE
DEST_PID	BIT[7:0]	IT INDICATES THE PORT IDENTIFIER FOR THE DESTINATION OF THE MESSAGE
OPCODE	BIT[7:0]	SPECIFIES OPERATION TO BE PERFORMED.
TAG	BIT[2:0]	SPECIFIED TAG FIELD OF THE TRANSACTION. IT IS USED ONLY FOR THE NON-POSTED MESSAGES AND USED AS AN IDENTIFIER OF THE REQUEST. IT IS USED ONLY BY THE SENDER TO ASSOCIATE COMPLETIONS WITH REQUESTS.
RESERVED	BIT[4:0]	SPECIFIES RESERVED FIELD OF THE TRANSACTION.
DATA[ ]	BIT[7:0]	SPECIFIES DATA CARRIED BY THE TRANSACTION. IT CAN HAVE N DWORDS OF DATA.

7.6.4.2 Commands Supported

All global opcode related commands are supported for message with data message.

Table 15. Commands Supported by msgd_xaction

Command	Opcode bit[7:0]	Description
OP_PM_REQ	0100_0000	PMU requests agent to initiate a state transition.
OP_PM_DMD	0100_0001	Agent indicates a request for a specified power state or provides its latency tolerance parameters to the PMU.
OP_PM_RSP	0100_0010	Agent response to PM_REQ.
OP_LTR	0100_0011	This message is issued by a PMU to set agent PM configuration parameters. (for 0.82 or 1.0 Spec)
OP_PCI_PM	0100_1000	PCI_PM message
OP_PCI_ERROR	0100_1001	Indicates a PCI-e error
OP_CRRD	0000_0110	Read Private Control Register
OP_CRWR	0000_0111	Write Private Control Register

7.6.4.3 Helper Fields

Table 16. msgd_xaction Helper Fields

Variable	Type	Description
data_size_dw	Int	Indicates size of the data

7.6.5 simple_xaction

This is the simple transaction class which is extended from base xaction class. Simple messages may be either posted or non-posted. Simple non-posted messages are completed with a completion without data messages.

7.6.5.1 Configurable Parameters

Table 17. simple_xaction Configurable Fields

Name	Type	Description
SRC_PID	BIT[7:0]	It indicates the port identifier for the source of the message
DEST_PID	BIT[7:0]	It indicates the port identifier for the destination of the message
OPCODE	BIT[7:0]	Specifies operation to be performed.
TAG	BIT[2:0]	Specified tag field of the transaction. It is used only with the non-posted messages and used as an identifier of the request. It is used only by the sender to associate completions with requests.
RESERVED	BIT[4:0]	Specifies reserved field of the transaction.

7.6.5.2 Commands Supported

All global opcode supported commands are supported.

Table 18. Commands supported by simple_xaction

Command	Opcode bit[7:0]	Description
OP_ASSERT_INTA	1000_0000	Assert INTA virtual wire
OP_ASSERT_INTB	1000_0001	Assert INTB virtual wire
OP_ASSERT_INTB	1000_0010	Assert INTC virtual wire
OP_ASSERT_INTB	1000_0011	Assert INTD virtual wire
OP_DEASSERT_INTA	1000_0100	De-assert INTA virtual wire
OP_DEASSERT_INTB	1000_0101	De-assert INTB virtual wire
OP_DEASSERT_INTC	1000_0110	De-assert INTC virtual wire
OP_DEASSERT_INTD	1000_0111	De-assert INTD virtual wire
OP_DO_SERR	1000_1000	DO SERR opcode (only for 1.0 Spec)

7.7 Sequencers / Sequence Drivers—Example

VC uses OVM Sequence interface to get transactions to be driven on the IOSF Sideband bus. VC has following sequences in the sequence library.

VC has iosfsbc_sequencer which extends ovm_sequencer with additional functionality to connect it to external sequencer.

User can also use external sequencer and connect it to internal iosfsbc_sequencer to send xactions.

There is one instance of the sequencer per TLM Driver component in the VC. Sequencer arbitrates between the all the sequences it is connected to and communicates with the TLM driver to send a transaction generated by a given sequence.

In case user wants to send xactions through external sequencer, pass_thru_seq is used to get the xaction from the external sequencer and send it to internal sequencer.

In case the user wants to send xactions through external TLM, then this can be done using the ovm_put_export defined inside the iosfsbc_sequencer.

7.8 Sequences / Sequence Libraries—Example

Table 19. Re-usable Test Sequences

Name	Description	MS	Status
base_seq	This class contains common members and APIs for all VC sequences. All other sequences are extended from this class		
simple_seq	Sends simple message transaction. API set_fields provides user setting of fields such as dest, src,opcode, tag, message type(posted,non-posted). If API not used, then this sequence will send default simple message transaction.		
msgd_seq	Same attributes as simple_seq except sends message with data message transaction.`		
regio_seq	Same attributes as simple_seq except sends message with data message transaction.		

Name	Description	MS	Status
exhaustive_seq	Sends random transactions.		
loopback_seq	Sends loopback message(same src_pid, dest_pid). API set_count controls how many messages to send.		
rnd_bcast_mcast_seq	Sends random pattern of non_posted/posted broadcast/multi-cast messages based on the count set by the test. It randomly selects one of the pattern based on the count and sends that many bcast/mcast xactions. Test can use this sequence to send multiple xaction from any one randomly selected agent. Done_send API can be used by the test to send another set of bcast/mcast xactions from the other Agent. At a time only one agent will send bcast/mcast xactions. Patterns are defined such ways that it ends with the NON_POSTED xaction in order to maintain posted/non-posted bcast/mcast rules.		
single_bcast_mcast_seq	This sequence will send only one bcast/mcast xaction from the randomly selected by the Test.		
unicast_rnd_invalid_opcode_seq	This sequence will randomly send xaction with invalid opcode(non-global opcodes).		
unicast_rnd_seq	This sequence will randomly send unicast xaction. It will not select dest pid inside bcast or mcast ports.		
unicast_rnd_seq_nonposted_only	This sequence will randomly send only non-posted unicast messages.		
unicast_rnd_seq_posted_only	This sequence will randomly send only posted unicast messages.		
unsupported_pid_seq	This sequence will generate message with invalid pid (pid not part of the sideband network).		
pass _thru_seq	This Sequence gets xactions from Upper level Sequencer and sends them to lower level Sequencer.		
hammer_seq	This sequence will generate messages using unicast_rnd_seq, loopback_seq and unsupported_pid_seq.		
polling_seq	This sequence will generate read xaction, if expect_rsp field of the xaction is set then it will wait for completion to return.		
vintf_seq	Sequence to send xaction without using set_cfg API and with interface access.		
Iosf_sb_seq	Sequence to send xactions using send_xaction API or with ovm_do_on_with macro		

7.9 Tests and Test Templates—Example

Recommended OVM methodology is to instantiate environment object under the top level class which is ovm_test. All the tests should be extended from ovm_test which instatiate ovm_env. Configuration objects are instantiated and built. Top level env object passes configuration objects to the Fabric VC, thus doing hierarchical configuration. Configuration object shows port_ids, credit buffer related configuration data which gets passed to the Fabric VC.

For building additional tests, base_test should be extended. Example shown at `tests/back2back/base_test.svh`. Fabric VC's default configuration can be changed from test using OVM config utility.

```
source scripts/setup -x ep_default
Go to verif/sim
ace -c -x -t $TEST_NAME
```

Table 20. Test Cases

Name	Description	MS	Status
base_test	Directed test for basic initialization		passing
test01	Directed test for credit update, completion delay API and loop for getting completion back from sequencer		passing
test02	Random Test + tasks to get rx and tx messages from VC+ credit reinit API		passing
test03	Directed Test for Sending xactions using ep specific opcodes and set response type for particular opcode		passing
test04	Directed Test to send write and read messages and use of memory to store/retrieve completion data		passing
test05	Directed Test to send couple of write followed by read to get completion back using ovm REQ/RSP channel		passing
test06	Directed test to send xactions without using pass_over_seq		passing
test07	Use of ovm REQ/RSP channel for simple, msgd xactions using polling_simple_seq and polling_msgd_seq, use of ovm_do_with macro		passing
test08	load_compl_data API used to store completion data into memory		passing
test09	Use of iosf_sb_seq to send all types of xactions using send_xaction API and ovm_do_on_with macro		passing
test10	Use of regio_seq with compare_completion field set		passing
test11-test37	Random test with different configurations and APIs		passing

7.10 Test Lists and Regressions—Example

Table 21. Regression List

Name	Description	MS	Status
iosf_sbc_ep_full.list	all ep test, each test with 5 seeds		Coded
iosf_sbc_ep_full.list	21 ep test for sanity check		Coded

7.11 Transaction Constraints

There are some constraints inside the transaction class. In case user need to have create customized transaction, can add new constraints inside the sequence generator before calling randomize () function on the transaction.

7.11.1 xaction Class Constraints

Table 22 lists all of the constraints defined in the base transaction class.

Table 22. xaction Constraints

Constraint	Description
src_pid_field	This constraint maps src_pid to second flit in the message array.
dest_pid_field	This constraint maps dest_pid to first flit in the message array.

Constraint	Description
opcode_field	This constraint maps opcode field to third flit in the message array.
tag_field	This constraint maps tag field to [4][2:0] bits in the message array
msg_size_range	Default message size of the transaction which is >0 and <10
dest_pid_range	This constraint defines destination port id range to be inside all the pid defined in the system including 8'hff and 8'hfe but excluding ports mapped to this nid.
src_pid_range	This constraint defines source port id range to be inside all the pid defined for a particular EP including 8'hfe
opcode_range	This constraint makes sure that all the opcodes are inside the range specified in the spec related to different transaction types.
supported_opcode_for_dest_pid	This constraint is used to define all the supported opcodes for each dest_pids
dest_pid_neq_src_pid	This constraint makes sure that destination and source pids are not equal
no_tag_for_posted	This constraint makes tag filed to be all zero for posted region, simple and msgd transactions.
fe_with_mcast_bcast_nposted	This constraint makes sure that for source pid fe is used for non- posted transactions and dest_pids are inside multicast_ports and 8'hff.
xaction_delay_field	It sets xaction delay to 0
set_expect_rsp	It sets expect_rsp field to 0
fe_with_compl	This constraint makes sure that dest_pid = fe is not used with completion transactions.
msg_size_array	It makes sure that msg array is consistent with msg_size_dw

7.11.2 regio_xaction class Constraints

Table 23 lists all of the constraints defined for the register access transactions.

Table 23. regio_xaction Constraints List

Constraint	Description
bar_range	This constraint makes sure that bar is inside the range specified in the spec and not the reserved values.
addrlen_range	This constraint makes sure that 16 or 48 bit address is used with memory mapped and Private control register access transaction types and 16 bit address is used with IO mapped and configuration register access transaction types.
addr_size_range	This constrain makes sure that for 16 bit of address or sbe all zero, address field size is 2 bytes and for 48 bit of address or sbe not zero, address field size is 6 bytes.
data_size_range	This constraint makes sure that data size is 0 DW for read register,1 or 2DW for write register transactions and 1DW for write register with sbe all zero.
addr_value	This constraint makes sure that the reserved bit of the address fields are set to zero.
fid_value	This sets fid value to 0 for xaction with CRRD and CRWR opcodes
addr_value_083	This sets address bit values to 0 for memory and IO read/write xactions for 0.83 spec

Constraint	Description
set_sp_field	This constraint sets sp field to 0/1 based on sai_aware config field

7.11.3 comp_xaction Class Constraints

Table 24 lists all of the constraints defined for the completion transactions.

Table 24. comp_xaction Constraints List

Constraint	Description
reserved_field_value	This constraint sets reserved filed to be all zero.
data_size_range	This constraint makes sure that data size is 0 DW for completion without data xactions and 1 or 2DW for completion with data xactions.
posted_comp	This constraint sets xaction_class to posted for all the completions xactions.

7.11.4 msgd_xaction Class Constrains

Table 25 lists all of the constraints defined for the message with data transactions.

Table 25. msgd_xaction Constraints List

Constraint	Description
data_size_range	This constraint makes sure that data size is >=1 and <9
reserved_field_value	This constraint sets reserved filed to be all zero.
class_value	This constraint sets xaction_class to be posted for PM_REQ, PM_RSP, PM_DMD, PCI_PM, PCI_ERROR messages
opcode_value_083	This constraints is defined so that tests do not send xactions with PM_CFG opcodes when spec version is set to 0.83
set_sp_field	This constraint sets sp field to 0/1 based on sai_aware config field

7.11.5 simple_xaction Constraints

Table 26 lists all of the constraints defined for the simple transactions.

Table 26. simple_xaction Constraints List

Constraint	Description
reserved_field_value	This constraint sets reserved filed to be all zero.
set_sp_field	This constraint sets sp field to 0/1 based on sai_aware config field
set_xaction_class_083	This constraint insures that xaction class is not posted for simple global messages when spec version is set to 0.83

7.12 VC Configurations

In order to generate transaction with valid fields, objects ep_cfg, common_cfg need to be configured inside the VC using Fabric/Agent config objects. Configuration is done during build() phase. Here is an example of agtvc_cfg and fbrcvc_cfg object configuration.

Agents will use agtvc_cfg to configure the VC. There are some constraints defined in the config field which use can use to configure config descriptor.

Table 27. Configuration Descriptor - Attributes

Attribute	Description	Default Values
my_ports[$]	Ports mapped to the Agent/Fabric	'h11, 'h22, 'h33
supported_opcodes[$]	Opcodes supported by the Agent/fabric	DEFAULT_OPCODE S
np_crd_buffer	Non-posted credit buffer size in Agent/fabric VC	Randomized using constraint
pc_crd_buffer	Posted credit buffer size in Agent/fabric VC	Randomized using constraint
other_ports[$]	Ports mapped to the Agent/Fabric	‘hAA, ‘hBB
mcast_ports[$]	Multicast ports mapped to the Agent/Fabric	‘h20
payload_width	Payload width of the Agent/Fabric	Randomized using constraint
no_agents	Total number of agents in the system(fabric+agent)	2 (For back2back)
no_mcast_agents	Total number of mcast port for the Agent/fabric	1
layered_pass_thru	Set this bit to 1, if external sequencer is used	0
extern_stimgen_mode	Set this bit to 1, if external stim gen is used	0
compl_delay	Delay for the agent/fabric driver to drive the completion	Randomized using constraint
crd_update_delay	Delay for the agent/fabric responder to update the credits	Randomized using constraint
creditreq_delay	Delay to specify for how long agent/fabric can stay in credit_req state	Randomized using constraint
creditinit_delay	Delay to specify for how long agent/fabric can stay in credit init state	Randomized using constraint
creditack_delay	Delay to specify for how long fabric can stay in credit_ack state (Used only for fabric VC)	Randomized using constraint
clkack_assert_delay	Delay before asserting clkack signal (only for fabric vc)	
is_active	Flag to indicate whether the fabric/agent VC will work in active or passive mode	OVM_ACTIVE
chk_enabled	Flag to indicate whether the checking is enabled or not	1
mon_enabled	Flag to indicate whether the fabric/agent Vc’s Monitor is enabled or not.	1
cov_enabled	Flag to indicate whether the fabric/agent VC coverage is enabled or not	1
use_mem	Set this to 1 when user wants to use shadow memory for write/read xactions (default is byte addressing mode)	0
mem_dw_addr_mode	Set this to 1 when user wants to use dw addressing mode for the memory (use_mem needs to be set to 1)	0
mem_be_support	Set this to 1, when user wants to use BE values for storing/updating memory data (use_mem needs to be set to1)	0
enable_crd_reinit	Set this to 0 to disable credit reinit	1
enable_rnd_crd_reinit	Set this to 1 to enable random credit reinit	0
intf_name	Specify interface name used with setData API for virtual interface bundle	fabric_intf

Attribute	Description	Default Values
connection_type	Used only for fabric testbench	RTR_2_EP
ip_ism_intf	Used only for fabric testbench	1'b1
disable_rtr_2_rtr_ism	Used only for Haswell router testbench	1'b0
set_iosfspec_ver API	Use this API from the test to set iosf spec version	
iosfsb_spec_rev	Used for specifying spec revision	IOSF_082
ep_cfg	Ep config descriptor which is build based on fields defined above and is being passed to all the other components(xactor/ism/monitor/tlm_driver/tlm_ pc)	
set_np_crd_buffer	Constraint to set np credit buffer size	Randomized to 1,2, or 3
set_pc_crd_buffer	Constraint to set pc credit buffer size	Randomized to 1,2, or 3
set_crd_update_delay	Constraint to set credit update delay	Randomized to be between 0,1,2,3
set_compl_delay	Constraint to set completion delay	Randomized to be between 0,1,2,3
set_creditreq_delay	Constraint to set credit request delay	Randomized to be between 0, and 5
set_creditack_delay	Constraint to set credit_ack fabric_ism delay	Randomized to be between 0 and 5
set_rnd_crd_reinit_cnt	Constraint to set random credit reinit count(how may times agent can initiate credit reinit)	Randomized to be between 1,2,3.4
set_crd_reinit_trigger_cnt	Constraint to set credit reinit trigger count	Randomized to be between 1,2,3.4
set_pload_width	Constraint to set payload_width	Randomized to be between 8,16 or 32
rtr_mode	Used by Fabric Router TLM Testbench	0
disable_compmon	Used by VC to disable compliance monitor	0
ext_header_support	User can set this to indicate whether agent/fabric support sending ext_header along with message or not (Only of 0.9 spec)	0
ext_headers	User can set this field to indicate ext_header value	Random with header_id set to 7'h00
num_tx_ext_headers	User can set this to indicate number of ext_headers when ext_header_support is set to 1 (Set it to 1 always, other values are not supported by IOSF 0.9 spec)	Randomized to be 1 or 2
turn_off_txn_constraints	Users can use this to turn off xaction level constraints related to src, dest and opcode	0
ctrl_rsp_per_opcode	User can use this config field, if user wants to override memory response	0
ctrl_ext_header_support	This field is used only for fabric_vc, where user wants to send messages with/without ext_headers, completion generated will have ext_header based on message received by VC. When VC receives message with ext_header, it will generate completion with ext_header and when it receives message without ext_header, it will generate completion without ext_header.	0

Attribute	Description	Default Values
ext_headers_per_txn	User can set this field in the config if user wants to send sai header for each message using iosf_sb_seq	0
agt_ext_header_support	This field is used only for fabric_vc, when set indicates that connected agent is sai aware, this needs to be driven to correct value so that VC can appropriately drive compmon straps	0
turn_off_bar_constraint	When set, VC will not turn off bar related constraint , user now can use any Bar value with messages	0
loopback_support	When set, users can use iosf_sb_seq to send loopback xactions (src_pid = dest_pid)	0
agt_skip_active_req	Agent VC can be configured to skip Active_req state and can move to active state when fabric is in active_req state. Valid only if Spec version is	0
	set to 1.0	
enable_credit_init_check	User can choose to check credit initialized during credit init loop by using this flag	0
auto_pm_rsp	When set, Vc will automatically sends pm_rsp message for any received pm_req message	0
rand_idle_nak_support	Applicable only for Fabric VC, when set fabric VC will randomly nak Agent's IDLE request even if fabric can move to IDLE	0
independent_reset	Applicable only for Fabric VC, when set fabric VC will consider agent_rst_b interface port for compliance monitor checks	0
unpack_xaction_check_off	When set, VC will turn off unpacking of xaction related checks	0
credit_check_off	When set, VC will turn off credit counter related monitor checks	0
creditdone_delay	Applicable only for agent VC. Sets credit_done state delay for agent ism	Randomized to be between 0 and 5
active_req_delay	Applicable only for agent VC. Sets active_req state delay for agent ism	Randomized to be between 0 and 5
idlereq_delay	Applicable only for agent VC. Sets idlereq_delay state delay for agent ism(IDLEREQ->ACTIVE) state	Randomized to be between 0 and 5
debug_name	Provide debug_name to control debug messages thru plusarg (+IOSF_SB_DEBUG=<debug_name>	"iosf_sb"

NOTES:

1. np_crd_buffer, pc_crd_buffer attributes are used by the VC to issue the credit updates to the Agent or router DUT. If credit buffer size is not configured, VC will randomly generate credit buffer size.
2. Also VC will use DEFAULT_OPCODES as supported opcodes by default, if user wants to send xactions with ep-specific opcode then they will have configured this. Refer to Code:2 setting up ovm_test for details.
3. VC needs to know the payload width of the sideband message link in order to drive flits, which is configured during OVM build() phase using payload_width config attribute above. By default VC will use payload_width=8bits.
4. User can disable compliance monitor by setting disable_compmon field to 1. By default compliance monitor is enabled. Compliance monitor doesn't have any switch to setup different spec version, and it is 0.83 compliant by default.

8 Coverage Details

8.1 Interface Cover Properties

Table 28. Assertion Cover Points

Error Code	Description
VR.SBC.0059	Agent can initiate exit from IDLE to send a transaction or credit update
VR.SBC.0085	Fabric can initiate exit from IDLE to send a transaction or credit update
VR.SBC.0148	Cover back to back posted messages from Node A to Node B, pcput goes high continue till eom, then pcput goes high right after that whenever credit is available
VR.SBC.0149	Cover back to back non-posted messages From Node A to Node B, npput goes high continue till eom, then npput goes high right after that whenever credit is available
VR.SBC.0150	Posted , Non-Posted messages are interleaved from Node A to Node B
VR.SBC.0159	Cover a case where we have p and np multicast/broadcast coexisting at the same time from the same source
VR.SBC.0059	Agent can initiate exit from IDLE to send a transaction or credit update
VR.SBC.0085	Fabric can initiate exit from IDLE to send a transaction or credit update
VR.SBC.0148	Cover back to back posted messages from Node A to Node B, pcput goes high continue till eom, then pcput goes high right after that whenever credit is available
VR.SBC.0149	Cover back to back non-posted messages From Node A to Node B, npput goes high continue till eom, then npput goes high right after that whenever credit is available
VR.SBC.0150	Posted , Non-Posted messages are interleaved from Node A to Node B
VR.SBC.0159	Cover a case where we have p and np multicast/broadcast coexisting at the same time from the same source
VR.SBC.0251	Two messages finished in consecutive clock cycles

8.2 Functional Coverage — Signal level

Table 29. Signal Level Cover Groups

Cover Points	Bins and Ranges	Description
master_vr_sbc_0143 triggering when master puts or cups changes		Cover Master NP credit 1, 1, >1, 255
mnp_crd_cntr	0	Cover Master NP credit 1, 1, >1, 255
	1	
	[2:254]	
master_vr_sbc_0144 triggering when master puts or cups changes		Cover Master PC credit 1, 1, >1, 255
mpc_crd_cntr	0	Cover Master PC credit 1, 1, >1, 255
	1	
	[2:254]	
master_vr_sbc_0145 triggering when master puts or cups changes		Cover Master npput = 0/1 while npcup = (0/1)
mnpcup	0	mnpcup = (0/1)

Cover Points	Bins and Ranges	Description
	1	
mnpput	0	mnpput = 0/1
	1	
master_vr_sbc_0146 triggering when master puts or cups changes		Cover Master pcput = 0/1 while pccup = (0/1)
mpccup	0	mpccup = (0/1)
	1	
mpcput	0	mpcput = 0/1
	1	
master_vr_sbc_0252 triggering when master cups changes		Cover Master pccup and npcup asserted at the same time
mpccup	1	mpccup = (1)
mnpnup	1	mnpcup = 1
target_vr_sbc_0143 triggering when target puts or cups changes		Cover Target NP credit 1, 1, >1, 255
tnp_crd_cntr	0	Cover Target NP credit 1, 1, >1, 255
	1	
	[2:254]	
target_vr_sbc_0144 triggering when target puts or cups changes		Cover Target PC credit 1, 1, >1, 255
tpc_crd_cntr	0	Cover Target PC credit 1, 1, >1, 255
	1	
	[2:254]	
target_vr_sbc_0145 triggering when target puts or cups changes		Cover target npput = 0/1 while npcup = (0/1)
tnpcup	0	tnpcup = (0/1)
	1	
tnpput	0	tnpput = 0/1
	1	
target _vr_sbc_0146 triggering when target puts or cups changes		Cover target pcput = 0/1 while pccup = (0/1)
tpccup	0	tpccup = (0/1)
	1	
tpcput	0	tpcput = 0/1
	1	
target _vr_sbc_0252 triggering when target cups changes		Cover target pccup and npcup asserted at the same time
tpccup	1	tpccup = (1)
tnpnup	1	tnpcup = 1
vr_sbc_0202_1 triggering when reset is high		Cover different flist size
WIDTH	1	WIDTH=1

Cover Points	Bins and Ranges	Description
vr_sbc_0202_2 triggering when reset is high		Cover different flist size
WIDTH	2	WIDTH=2
vr_sbc_0202_4 triggering when reset is high		Cover different flist size
WIDTH	4	WIDTH=4
vr_sbc_0236 triggering when agent ism is in IDLE state		clkreq may go low when the idle sm is in idle and bus is inactive
side_clkreq	0	
	1	
vr_sbc_0219_fabric triggering whenever fabric ism state changes		Cover all fabric_ism_state transitions
side_ism_fabric	bin per state transition	
vr_sbc_0219_fabric_states triggering whenever fabric ism state changes		Cover all fabric_ism_states
side_ism_fabric	bin per state	Cover each state entry
vr_sbc_0217_agent triggering whenever agent ism state changes		Cover all agent_ism_state transitions
side_ism_agent	bin per state transition	
vr_sbc_0217_agent_states triggering whenever agent ism state changes		Cover all agent_ism_states
side_ism_agent	bin per state	
vr_sbc_0221_tpcput_ism_agent triggering whenever agent ism state changes and tpcput changes		Fabric PC message put during valid agent ISM states
side_ism_agent	ACTIVE	
	IDLE_REQ	
tpcput	0	
	1	
cross	Side_ism_agent and tpcput	
vr_sbc_0221_tnpput_ism_agent triggering whenever agent ism state changes and tnpput changes		Fabric NP message put during valid agent ISM states
side_ism_agent	ACTIVE	
	IDLE_REQ	
tnpput	0	
	1	
cross	Side_ism_agent and tnpput	
vr_sbc_0224_mnpput_ism_fabric triggering whenever fabric ism state changes and mnpput changes		Agent NP message put during valid fabric ISM states
side_ism_fabric	ACTIVE_REQ	
	ACTIVE	
	IDLE_NAK	
mnpput	0	
	1	

Cover Points	Bins and Ranges	Description
cross	side_ism_fabric and mnpput	
vr_sbc_0224_mpcput_ism_fabric triggering whenever fabric ism state changes and mpcput changes		Agent PC message put during valid fabric ISM states
side_ism_fabric	ACTIVE_REQ	
	ACTIVE	
	IDLE_NAK	
mpcput	0	
	1	
cross	side_ism_fabric and mpcput	
vr_sbc_0221_tpccup_ism_agent triggering whenever agent ism state changes and tpccup changes		Fabric PC message cup during valid agent ISM states
side_ism_agent	ACTIVE	
	IDLE_REQ	
	CREDIT_INIT	
	CREDIT_DONE	
tpccup	0	
	1	
cross	Side_ism_agent and tpccup	
vr_sbc_0221_tnpcup_ism_agent triggering whenever agent ism state changes and tnpcup changes		Fabric NP message cup during valid agent ISM states
side_ism_agent	ACTIVE	
	IDLE_REQ	
	CREDIT_INIT	
	CREDIT_DONE	
tnpcup	0	
	1	
cross	Side_ism_agent and tnpcup	
vr_sbc_0224_mnpcup_ism_fabric triggering whenever fabric ism state changes and mnpcup changes		Agent NP message cup during valid fabric ISM states
side_ism_fabric	ACTIVE_REQ	
	ACTIVE	
	IDLE_NAK	
	CREDIT_INIT	
mnpcup	0	
	1	
cross	side_ism_fabric and mnpcup	
vr_sbc_0224_mpccup_ism_fabric triggering whenever fabric ism state changes and mpccup changes		Agent PC message cup during valid fabric ISM states
side_ism_fabric	ACTIVE_REQ	
	ACTIVE	

Cover Points	Bins and Ranges	Description
	IDLE_NAK	
	CREDIT_INIT	
mpccup	0	
	1	
cross	side_ism_fabric and mpcput	

8.3 Functional Coverage — Transactions

Table 30. Transaction Level Cover Groups

Cover Points	Bins and Ranges	Description
VR_SBC_0161 triggering for completion transactions		Cover a case where completion status is not supported
opcode	OP_CMP	With and without data completion type.
	OP_CMPD	
rsp	RSP_SUCCESSFUL	Response types
	RSP_NOTSUPPORTED	
	RSP_MCASTMIXED	
	RSP_POWEREDDOWN	
not_supported	Cross opcode and rsp	Ignore_bins = binsof(opcode) intersect {OP_CMPD} && binsof(rsp) intersect {RSP_POWEREDDOWN, RSP_NOTSUPPORTED, RSP_MCASTMIXED }
VR_SBC_0169 triggering for all xactions		Cover that all transaction types are exercised on each EP
opcode	bin per opcode	All the global opcodes.
	OP_IORD	
	OP_CFGRD	
	OP_CRRD	
VR_SBC_0171 triggering for MSGD xactions		Cover data files = 1DW and >1DW for message with data
xaction_type	MSGD	Message with data transaction type
data_size	{4}	Legal bins of 4 or more
	{[5:$]}	
	Illegal_bins {[0:3]}	
trans	Cross xaction_type and data_size	
VR_SBC_0173 triggering for REGIO xactions		Cover register access posted/nonposted transactions
xaction_class	NON_POSTED	Posted, non-posted xaction classes
	POSTED	
read Opcode	bin for each global read msg opcode	Read register global opcodes
write opcode	bin for each global write msg opcode	Write register global opcodes
read msg_size_dw	2	Message size for read register

Cover Points	Bins and Ranges	Description
	3	
write msg_size_dw	3	Message size for write register
	4	
	5	
read	Cross of xaction_class, read_opcode and read_msg_size_dw	Illegal_bins = binsof(trans) intersect {POSTED} || ((binsof(length_r) intersect {3}) && (binsof(reg_read) intersect {OP_IORD,OP_CFGRD}))
write	Cross of xaction_class, write_opcode and write_msg_size_dw	Illegal_bins = ((binsof(length_w) intersect {5}) && (binsof(reg_write) intersect {OP_IOWR,OP_CFGWR}))
VR_SBC_0174 triggering for REGIO xactions		Cover read register transaction type
opcode	bin for each global read msg opcode	Read register transaction types
VR_SBC_0176 triggering for REGIO xactions		Cover write register transaction type
opcode	bin for each global write msg opcode	Write register transaction types
VR_SBC_0178		Cover addrlen 16 and 48 bit for read and write register access transactions
opcode	Bin for each global regio message opcode	All register access opcodes
addrlen	ADDR_16_bit	16 and 48 bit address length
	ADDR_48_bit	
all_addr	Cross of opcode and addrlen	Ignore_bins = binsof(opcode) intersect with IO amd config register && binsof(addrlen) intersect with ADDR_48_bit
VR_SBC_0180 triggering for REGIO xactions		Cover all BAR values for register access transactions
opcode	OP_MRD	All memory mapped and IO mapped register access opcodes
	OP_MWR	
	OP_IORD	
	OP_IOWR	
bar	bin for each valid bar value	All Bar values
	Illegal bin BAR_RESERVED_1	
	Illegal bin BAR_RESERVED_2	
all_bar	Cross opcode and bar	
VR_SBC_0181 triggering for REGIO xactions		Cover posted and nonposted transaction type for write register
opcode	Bin for each global reg write message	All write register transactions
xaction_class	POSTED	Posted and non-posted xaction class
	NON_POSTED	
write	Cross of opcode and xaction_class	
VR_SBC_0182 triggering for REGIO xactions		Cover all sbe values for register access xactions

Cover Points	Bins and Ranges	Description
opcode	bin for each global regio message opcode	All register access transaction types
sbe	Sbe	Sbe value
all_sbe	Cross of opcode and sbe	
VR_SBC_0183 triggering for REGIO xactions		Cover all fbe values for register access xactions
opcode	bin for each global regio message opcode	All register access transaction types
	OP_IOWR	
	OP_CFGWR	
	OP_CRWR	
	OP_MRD	
	OP_IORD	
	OP_CFGRD	
	OP_CRRD	
fbe	fbe	fbe value
all_fbe	Cross of opcode and fbe	
VR_SBC_0184 triggering for REGIO xactions		Cover all routing ids for register access xactions
opcode	bin for each global regio message opcode	All register access transaction types
Fid	Fid	Function IDs
all_rid	Cross of opcode and rid	
VR_SBC_0185 triggering for REGIO xactions		Cover address length, bar and rid for register access xactions
opcode	bin for each global regio message opcode	All register access transaction types
addrlen	ADDR_16_bit	Address length modes
	ADDR_48_bit	
rid	[8'h00:8'h40]	Routing ids
	[8'h41:8'h80]	
	[8'h81:8'hC0]	
	[8'hC1:8'hFF]	
bar	bin for each valid BAR value	All Bar values
	Illegal_bins BAR_RESERVED_1	
	Illegal_bins BAR_RESERVED_2	

Cover Points	Bins and Ranges	Description
all_addr	Cross opcode, addrlen, rid, bar	ignore_bins ((binsof(reg_access) intersect {OP_IORD,OP_CFGRD,OP_IOWR,OP_CFGWR}) && (binsof(address_lengths) intersect {ADDR_48_bit})) ((binsof(reg_access) intersect {OP_CFGRD,OP_CRRD,OP_CFGWR,OP_CRWR}) && binsof(bar)) ((binsof(reg_access) intersect {OP_CRRD,OP_CRWR}) && binsof(rid))
VR_SBC_0187 triggering for REGIO xactions		Cover a case where sbe =0 and fbe != 0 for read register transactions
opcode	bin per global reg read msg opcode	All read register access transaction types
xaction_type	REGIO	
sbe	0	Zero and non-zero sbe values
	[1:$]	
fbe	0	Zero and non-zero fbe values
	[1:$]	
dw_access	Cross opcode. Xaction_type,sbe,fbe	Dw_access = binsof(sbe.zero) && binsof(fbe.non_zero)
		illegal_bins = ((binsof(sbe) intersect {[4'b0001:4'b1111]}) && (binsof(reg_read) intersect {OP_IORD,OP_CFGRD}))
VR_SBC_0188 triggering for REGIO xactions		Cover a case where sbe !=0 and fbe = 0 for read register transactions
opcode	bin per global reg read msg opcode	All read register access transaction types
xaction_type	REGIO	
sbe	0	Zero and non-zero sbe values
	[1:$]	
fbe	0	Zero and non-zero fbe values
	[1:$]	
dw_access	Cross opcode, xaction_type, sbe, fbe, dw_access	Dw_access = binsof(sbe.non_zero) && binsof(fbe.zero)
VR_SBC_0190 triggering for REGIO xactions		Cover write register trasnaction lengths 3, 4 and 5 DW
opcode	bin per global reg write msg opcod	All write register transactions
msg_size_dw	3	Message size 3, 4 and 5 DW
	4	
	5	
write	Cross opcode. Msg_size_dw	illegal_bins = ((binsof(length_w) intersect {5}) && (binsof(reg_write) intersect {OP_IOWR,OP_CFGWR}))
VR_SBC_0191 triggering for REGIO xactions		Cover a case of 1 DW requests with no bytes enables for write register transactions

Cover Points	Bins and Ranges	Description
opcode	bin per global reg write msg opcode	All write register transactions
msg_size_dw	1	Message size of 1DW
sbe	{0}	Sbe = 0
fbe	{0}	Fbe = 0
write	Cross opcode, message_size, sbe, fbe	
VR_SBC_0192 triggering for REGIO xactions		Cover both posted and non-posted for write register transactions
xaction_class	POSTED	Posted, non-posted xaction classes
	NON_POSTED	
opcode	bin per global reg write msg opcode	All write register transactions
write	Cross opcode, xaction_class	
VR_SBC_0193 triggering for COMP xactions		Cover both completion with data and completion without data
opcode	OP_CMP	Completion with and without data xactions
	OP_CMPD	
VR_SBC_0194 triggering for COMP xactions		Cover all completion response fields for completion with and without data
opcode	OP_CMP	Completion with and without data xactions
	OP_CMPD	
rsp	RSP_NOTSUPPORTED	All response fields
	RSP_SUCCESSFUL	
	RSP_POWEREDDOWN	
	RSP_MCASTMIXED	
all	Cross opcode, rsp	ignore_bins = ((binsof(rsp) intersect {RSP_NOTSUPPORTED,RSP_POWEREDDOWN,RSP _MCASTMIXED}) && (binsof(comp_type) intersect {OP_CMPD}))
VR_SBC_0195 triggering for COMP xactions		Cover both completion with data sizes 2DW and 3DW
opcode	OP_CMPD	Completion with data xactions
msg_size_dw	2	Message size 2 and 3DW
	3	
cmpd_access	Cross opcode, msg_size_dw	
VR_SBC_0196 triggering for all xactions		Cover all global opcode values
opcode	bin per global opcode for each message type	All global opcodes.
VR_SBC_0197 triggering for simple xactions		Cover all opcodes in the range {1000_000:1000_0111}
opcode	bin per global simple msg opcode	All the simple global opcodes.

Cover Points	Bins and Ranges	Description
xaction_class	POSTED	Posted, non-posted xaction classes
	NON_POSTED	
simple_msg_all	Cross opcode, xaction_class	
VR_SBC_0198 trigering for message with data		Cover all opcodes in the range {0100_0000:0100_1001}
opcode	bin per global msg with data message opcode	All the message with data global opcodes.
xaction_class	POSTED	Posted, non-posted xaction classes
	NON_POSTED	
msg_w_data_all	Cross opcode, xaction_class	
VR_SBC_0199 triggering for COMP xactions		Cover all opcodes in the range {0010_0000 : 0010_0001}
opcode	OP_CMP	All the completion global opcodes.
	OP_CMPD	
xaction_class	POSTED	Posted, non-posted xaction classes
	Illegal bins NON_POSTED	
cmp_all	Cross opcode, xaction_class	
VR_SBC_0200 triggering for mREGIO xactions		Cover all opcodes in the range {0000_0000 : 0000_0111}
Opcode	bin per global regio message opcode	All read register opcodes
Xaction_class	POSTED	Posted, non-posted xaction classes
	NON_POSTED	
Read	Cross opcode(read), xaction_class	illegal_bins = binsof(trans) intersect {POSTED}
Write	Cross opcode(write), xaction_class	
VR_SBC_0136 triggering for MSGD xactions		Cover payload corner cases for message with data xactions where data size is 1DW
data[0]	8'hAA, 'hFF, 'h55, 'h0	Corner cases payload
data[1]	8'hAA, 'hFF, 'h55, 'h0	Corner cases payload
data[2]	8'hAA, 'hFF, 'h55, 'h0	Corner cases payload
data[3]	8'hAA, 'hFF, 'h55, 'h0	Corner cases payload
VR_SBC_0136 triggering for MSGD xactions		Cover payload corner cases for message with data xactions where data size is 2DW
data[0]	8'hAA, 'hFF, 'h55, 'h0	Corner cases payload
data[1]	8'hAA, 'hFF, 'h55, 'h0	Corner cases payload
data[2]	8'hAA, 'hFF, 'h55, 'h0	Corner cases payload
data[3]	8'hAA, 'hFF, 'h55, 'h0	Corner cases payload
data[4]	8'hAA, 'hFF, 'h55, 'h0	Corner cases payload

Cover Points	Bins and Ranges	Description
data[5]	8'hAA, 'hFF, 'h55, 'h0	Corner cases payload
data[6]	8'hAA, 'hFF, 'h55, 'h0	Corner cases payload
data[7]	8'hAA, 'hFF, 'h55, 'h0	Corner cases payload
VR_SBC_0136 triggering for REGIO xactions		Cover payload corner cases for regio xactions where data size is 1DW
data[0]	8'hAA, 'hFF, 'h55, 'h0	Corner cases payload
data[1]	8'hAA, 'hFF, 'h55, 'h0	Corner cases payload
data[2]	8'hAA, 'hFF, 'h55, 'h0	Corner cases payload
data[3]	8'hAA, 'hFF, 'h55, 'h0	Corner cases payload

8.4 Checking Details

Table 31. Checkers

Name	Description	MS	Status
IOSF Compliance	Passive verification module delivered with IOSF BFM that does dynamic checking of various IOSF interface rules. Will be enabled by default. May only be disabled during compile with +NO_IOSF_CHK. Checker assertions may be ignored if specific assertions are added to $IP_ROOT/sva_control/ignore.sva	0.5	Enabled
Input Scoreboard	Main dynamic scoreboard checker used to detect dropped or corrupted transactions. Will be enabled by default. Can be disabled from test run commandline using -NO_INP_SB_CHK	0.5	Enabled
Rcomp Checker	Dynamic checks for a variety of rcomp rules / protocols. Will be enabled by default. Can be disabled from test run commandline using -NO_RCOMP_CHK	0.8	Planned
Data Checker	End-of-test dynamic check to ensure contents of local memory match expected values from the specified test's mem.out file	0.8	Coded
Power Flow Checker	Post-process power flow checker (inherited) that operates on power.log. Will only be enabled on power regression, disabled by default. Enable using =PWR_CHK on test commandline.	0.8	Coded
Performance Checks	Performance checks will be owned by the performance team.	N/A	N/A
Misc SVAs	Embedded SVAs used throughout the design to detect illegal conditions. Will be enabled by default. Failing assertions can be ignored if specific assertions are added to $IP_ROOT/sva_control/ignore.sva	0.5+	Ongoing
Test Self-Check	Note to highlight that many directed tests do self-checks as part of the test.	0.5+	N/A

8.5 Saftey Feature Traceability Indicators

Not applicalbe to this IP

9 Checking Details

Chekers such as compliance monitor, ip_vc scoreboard along with interface protocol check are part of the environment.

Table 32. Checkers—Example

Name	Description	MS	Status
IOSF Compliance	Passive verification module delivered with IOSF BFM that does dynamic checking of various IOSF interface rules. Will be enabled by default. May only be disabled during compile with +NO_IOSF_CHK. Checker assertions may be ignored if specific assertions are added to $IP_ROOT/sva_control/ignore.sva	0.5	Enabled
ip_vc Scoreboard	scoreboard checker used to detect dropped or corrupted transactions. Will be enabled by default.	0.5	Enabled

9.1 Scoreboard and Checker

Check Point	Objective	MS	Status
IOSF protocol	Checking IOSF protocol correctness	0.5	Enabled
Tmsg/msg/trge/mreg protocol	Checking protocol at IP interface	0.5	Enabled
Ip_vc scoreboard	Checking transactions are coming out correctly	0.5	Enabled

10 Debug Details

Table 33. OVM Log Messages

OVM_INFO	Information Message
OVM_ERROR	Error Message
OVM_FATAL	Fatal Error Message

Table 34. Message from Components

EP TX	Message from Endpoint
DRV RX	Message from driver
DRV TX	Message from Driver
RSP RX	Message from Responder
TLM PC	Message from Transaction Level Protocol Checker
RTR SB	Message from Scoreboard
ISM	Message from ISM
PM	Message from Power Manager
OPCODE CB	Message from Opcode Callback
PWR MGR	Message from Power Manager
DRV DEBUG	Debug message from driver
RSP DEBUG	Debug message from Responder

- [EP TX] Sending Xaction - This indicates that ep_tlm is sending the xaction.
- [EP TX] Sending Xaction posted/non-posted to put_port/np_put_port - This indicates that ep_tlm has written the received xaction onto pc/np port.
- [EP TX] Sending Completion Xaction - This indicates that ep_tlm is sending completion xaction
- [EP TX] Sending Completion Xaction to put_port - This indicates that ep_tlm has written the completion xaction onto pc/np port.
- [DRV RX] Xaction - This indicates that Driver has received the xaction.
- [DRV TX] Xaction – This indicates that Driver has transmitted the xaction.
- [ISM] Sending * command – This indicates that ism is sending * command
- [PM] Received * command – This indicates that power manager has received this command
- [TLM PC] vr_sbc_0* - This indicates that check for verification requirement vr_sbc_0* has failed.
- [TLM PC] AGING: non-posted * transaction timeout – This indicates that at non-posted messages is sitting in the queue for time more than TIMEOUT_DELAY set during the test.
- [TLM PC] For * link: Completion w/o matching non-posted – This indicates that tlm_pc has received completion without matching non-posted xaction sent.
- [TLM PC] For * link: vr_sbc_0248: Duplicate Tag – This indicates that there are multiple xaction exist in the system.

11 Formal Verification Details

Table 35. SystemVerilog Assertions on interface

Error Code	Description
VR.SBC.0086	A message flit cannot be put unless there is a corresponding credit for that flit type
VR.SBC.0091	When encounter a npput, decrease internal np credit counter and check against model counter (if access is available)
VR.SBC.0092	When encounter a pcput, decrease internal pc credit counter and check against model counter (if access is available)
VR.SBC.0097	The EOM signal is only valid if a put is asserted, and is undefined otherwise
VR.SBC.0098	EOM lasts only for one cycle, 32-bit flit size being exception where EOM can be active for more than one cycle
VR.SBC.0105	For each stream of put signals, EOM should be asserted at the end
VR.SBC.0119	When encounter npcup, increase internal np credit counter and check against model counter (if access is available)
VR.SBC.0120	When encounter pccup, increase internal pc credit counter and check against model counter (if access is available)
VR.SBC.0126	When side_rst_b is asserted, the credit tracking registers in agents are reset to zero.
VR.SBC.0271	npput should be valid at start of message, EOM or message in progress
VR.SBC.0272	pcput should be valid at start of message, EOM or message in progress
VR.SBC.0257	Assert error if, Agent credits are initialized during states other than CREDIT_INIT state
VR.SBC.0259	Assert error if, at least one non-posted credit is not advertized during CREDIT INIT state
VR.SBC.0260	Assert error if, atleast one posted credit is not advertized during CREDIT INIT state
VR.SBC.0204	Assert error if, Credit counter exceeds value advertized during CREDIT INIT state
VR.SBC.0206	Assert error if, Agent credits are re-initialized during states other than CREDIT_INIT state
VR.SBC.0216	Check legal agent ISM state transition
VR.SBC.0218	Check legal fabric ISM state transition
VR.SBC.0247	ISM state is valid at posedge of clk outside of reset
VR.SBC.0220	Assert error if agent receives puts during agent states other than ACTIVE and IDLE REQ and credit updates during agent states other than CREDIT INIT,

12 IP or Subsystem Verification Milestones

12.1 Milestones

IP milestone is based on PIC release. Every PIC release might have PCR request, fixed HSD tickets.

12.2 Other Indicators

Note: Other Indicators used to track the progress and quality of IP is reviewing/closing HSD tickets for bug and enhancement.

13 Validation Risks

Not applicable to this IP/IPSS