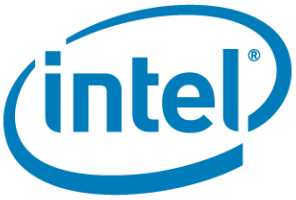


IOSF SBC Endpoint Unit

INTEGRATION GUIDE

ALL RTL 1.0-PICr35
January 2021

Intel Top Secret

Contents

1 Introduction 10
1.1 Audience 10
1.2 References 10
1.3 Contact Information 10
1.4 Terminology 10
1.5 Document Revision History 10
2 Quick Start 12
2.1 Downloading Sub IP 12
2.2 Firmware Version 12
2.3 Design Libraries 12
2.4 Defines and Variables 12
2.5 IP/IPSS Straps 12
2.6 Integrity Checks 13
3 Overview 15
3.1 Block Diagram 15
3.2 Functional Top-Level Signals 16
3.2.1 Signal List File 16
3.2.2 Tie-offs, Opens, and Gates 16
3.2.3 “Ad Hoc” Pins, Non-standard, or Non-compliant 16
3.2.4 Control Signals 16
3.2.5 Reset Ports 16
3.2.6 Naming Scheme 16
3.2.7 End Points 16
3.3 Forces 16
3.3.1 Temporary Design Forces 16
3.3.2 Other Forces 16
4 Tools and Methodology for Integration 17
4.1 Configuration Updates 17
4.2 Supported Tools 17
4.3 Environment Variables 17
4.4 HIP Libraries Included in Release 19
4.4.1 HIP IPToolData.pm Change 19
4.4.2 Register Files or SRAM 19
4.4.3 M-PHY and Related Libraries 19
4.5 Directory Structure for Tools 19
4.6 Ace 20

4.7 Lintra 20
4.8 Synthesis 20
4.8.1 Clocks 21
4.8.2 Clock Propagation Specification 21
4.8.3 Clock Diagram 21
4.8.4 Constraint Files 21
4.8.5 Synthesis Best Known Methods (BKMs) 21
4.8.6 Latches 21
4.8.7 BKMs 21
4.8.8 Gray Pointer Clock Domain Crossings on Async Endpoints 22
4.9 Formal Verification 22
4.10 CDC 22
4.11 Scan 22
4.12 DFD Triggers and Actions 22
5 Design Information for SoC Integration 23
5.1 RTL Directory Structure 23
5.2 Building Blocks for SoC Integration 23
5.3 Embedded Building Blocks/Custom Logic 23
5.4 Macros used by IP/IPSS 23
5.5 RTL Configuration Parameters and Overrides 23
5.5.1 Mandatory Parameters 24
5.5.2 Test Data Register Parameters 24
5.6 Integration Dependencies 24
5.7 RTL Uniquification 24
5.8 Registers 24
5.9 Fuses 24
5.9.1 Fuse RDL Files 24
5.10 Clock, Power, and Reset Domains 24
5.10.1 Clock Domain Diagram 24
5.10.2 Clock Requirements (HIP only) 24
5.10.3 Power Domains 25
5.10.4 Power Domain Mapping 25
5.11 System Startup 25
5.11.1 Power-up Sequence 25
5.11.2 Initialization Sequence 25
5.11.3 Device Configuration 25
5.11.4 Header for Windows Boot 25

5.11.5 Voltage Rail Requirements (HIP only) 25
5.12 System Startup 25
5.12.1 Power-up Requirements 25
5.12.2 Power-up Sequence 25
5.12.3 Initialization Sequence 25
5.12.4 Device Configuration 25
5.12.5 Header for Windows Boot 26
5.13 DFx Considerations 26
5.13.1 DFx Top-Level Signals 26
5.13.2 DFx Clock Definition 26
5.13.3 Clock Crossings 26
5.13.4 VISA 26
5.13.5 DFD Triggers and Actions 26
5.13.6 Debug Registers 26
5.13.7 Scan – Clock Gating in RTL 26
5.13.8 Scan – Reset Override 26
5.13.9 Scan – Constraints and Coverage 26
5.13.10 TAP and Associated Registers 27
5.13.11 Boundary Scan Parameters 27
5.13.12 Intra-Die Variation (IDV) - for HIP only 27
5.14 Security 27
5.14.1 SAI Width 27
5.14.2 Validation Requirements 27
5.14.3 Interface Signals Implemented for Security 27
5.15 Safety Support 27
5.15.1 Safety HW Compliance 28
5.16 Emulation Support 29
5.17 Other Design Considerations 29
5.18 IP META Data 29
6 Physical Integration 30
6.1 Memory Requirements 30
6.2 Subsystem Requirements 30
6.2.1 Hierarchy Details 30
6.2.2 Area and Floorplan Details 30
6.2.3 Fabric Convergence Requirements and Confidence 30
6.2.4 OTH Straps (Logical) 30
6.2.5 Clocking Domains and Entry/Exit Points 30

6.2.6 Global Elements 30
6.2.7 Power Domains and Power Grid Interface Requirements 31
6.2.8 Timing Considerations 31
6.2.9 Power 31
6.2.10 QRE-related Information 31
6.3 Open Issues 31
7 Connectivity Chains and IP/IPSS-specific Features (HIP only) 32
8 Verification Environment for Integration 33
8.1 IP Testbench Overview 33
8.2 SoC-specific Validation 33
8.3 Validation Parameters 33
8.3.1 #defines 33
8.3.2 +define Command Arguments 33
8.3.3 Parameters 33
8.3.4 Variables 35
8.3.5 Overrides 35
8.3.6 Plusargs 35
8.3.7 Pre- and Post- TI Imports 36
8.4 Verification Libraries 36
8.5 Testbench Components and Connectivity 36
8.5.1 Testbench Directory Structure 36
8.5.2 Test Islands 40
8.5.3 Test Island Interfaces 40
8.5.4 Checkers and Trackers 40
8.5.5 Monitors 40
8.5.6 Scoreboards 40
8.5.7 BFMs 41
8.5.8 Other Structures 41
8.6 Collage or Sandbox Files 41
9 IP2SOC Handoff Information 42
9.1 <Feature 1> 42
9.1.1 Testbench Block Diagram 42
9.1.2 BFM 42
9.1.3 Tracker Checker 42
9.1.4 Fast Sim Mode 42
9.1.5 Generator 42
9.1.6 Power Flow 42

9.2 <Feature 2> 42
10 Workarounds, Waivers, and Disabled Assertions 43
10.1 Disabled Assertions 43
10.2 Waivers 43
10.3 Other Workarounds 43
11 Integration Checks and Tests 44
12 Block Level Micro-Architecture 45
12.1 Block Descriptions 45
12.1.1 Sideband Base Endpoint (SBEBASE) 45
12.1.2 Register Access Master Agent (SBEMSTRREG) 45
12.1.3 Register Access Target Agent (SBETRGTREG) 45
12.1.4 DO_SERR Master Widget (DOSERRMSTR) 46
10.1.5 Bulk Read/Write Register Widget (BULKRDWR) 46
12.2 Block Interface 47
12.2.1 Module Parameter/Port List 47
12.2.2 Block Interface Timing Waveforms 61
13 Clock and Reset 70
14 Area Estimates 71
15 Integration Details 72
15.1 Sideband Fabric Inputs and Pipeline Stages 72
15.2 Ingress Latency 72
15.3 QUEUEDEPTH Parameter Setting and Bandwidth 72
15.4 MAXTRGTADDR Parameter 72
15.5 Parity Handling 73
15.5.1 IPs Integrating EP's Target Register Widget 73
15.5.2 IPs with Custom TMSG Block 74
15.5.3 IP with Custom Master to EP 74
15.5.4 Parity Error Reporting - DO_SERR Message 74
15.5.5 Endpoint Internal Target Path Parity 75
15.5.6 Endpoint Internal Master Path Parity 76
15.5.7 Parity Error Reporting - DO_SERR Message 76
15.6 Extended Header Handling 77
15.6.1 Unique Extended Header Support 77
15.6.2 Legacy Extended Header Support 77
15.7 VISA/DFx Features 78
15.8 Target Register Access Interface Limitations 78
15.9 Design for Clock Gating 79

15.10 UPF—Isolation and Retention 79

About This Template

How to Use This Template

Do not remove any headings from this document. If you do not need the headings to describe your IP, enter "Not applicable" under the heading. This lets the reader know that you did not overlook this topic.

In the main document that follows, add new headings that you need to fully describe the integration of this IP. Add them in the appropriate chapters.

Most red text in this document contains instructions for filling out the section where it appears. The tag for most of this red text is called "Gaps." You should replace this text with the content appropriate for that section, ensuring that the text is tagged appropriately (for example, with the BodyText or List Bullet style). If a section is not relevant, do not remove it; instead just replace the "Gap" text with "Not applicable" and apply the BodyText style.

Goal of This Document

This document should contain all information an integration team would need to accomplish the task without needing to seek help from another source. Try not to refer to other documents for required information; do so only if you include specific instructions for obtaining those documents, and only if you are sure your audience has access to them. Verify all links. This should be a self-contained guide for integration.

1 Introduction

1.1 Audience

The information in this document is intended for an integration team that is integrating this IP into an SoC.

1.2 References

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

Document	Location (if not in tarball)
IOSF SBC Base Endpoint Integration Guide	In 'doc' directory of release.

1.3 Contact Information

To contact one of the authors of this IP, use the information below.

Name	Function	Email
Manuel Savin	IP Project Contact	emanuel.savin@intel.com
Soe Myint	IP Project Contact	soe.myint@intel.com
Susann Flowers	Spec Template Owner	susann.flowers@intel.com

1.4 Terminology

The table below defines uncommon terms used in this document.

Term	Definition
IOSF	Intel On-Chip System Fabric
NP	Non-Posted message
PC	Posted / Completion message
IP/ IP Block	The design which instantiates the endpoint design for communication over IOSF Sideband network

1.5 Document Revision History

Revision	Date	Description
0.9	5/14/2010	Initial publication.
2010ww41	9/30/2010	Updated VISA information to match new struct outputs. Added Target Register Access limitations section.
1.0	3/21/2012	Updated information based on Sideband Endpoint Release for IOSF 1.0 specifications and resolving several compliance/ timing-closure/DFx-VISA related issues.
2012ww16	4/20/2012	Updated for minor corrections and additional clock gating information.
2012ww51	12/21/2012	Updated with information on deterministic clock crossing and power aware simulation.

Revision	Date	Description
2012ww51 (addendum)	1/23/2012	Added documentation for side_ism_lock_b.
2013ww18	2/5/2012	Updated documentation on power aware simulation.
2013ww18	05/2013	Edited and added to standard SIP template.
2014WW22	04/25/2013	Updated information about clock request related signals. Updated information about the next extended header handler. Updated input/output list.
2014ww39	9/24/2014	Added parity ports and parity-enable parameter
2015ww30	7/21/2015	Updates for 2015ww30 SBE release (MATCHED_INTERNAL_WIDTH, CLAIM_DELAY)
2015ww47	11/12/2015	Updates for 2015ww47 SBE release, related to parity handling: DO_SERR_MSTR parameter and widget
2019ww12	3/20/2019	Fixed N/A sections for new Integration Guide Template

2 Quick Start

2.1 Downloading Sub IP

Not applicable to this IP

2.2 Firmware Version

Not applicable to this IP

2.3 Design Libraries

CTECH library (version as per the TSA MAT version described in the release notes).

Also see section 0, HIP Libraries Included in Release, and section 0, Verification Libraries.

Library	Version	Special usage
CTECH	v14ww09e	N/A

2.4 Defines and Variables

Not applicable to this IP (all necessary defines are within the IP).

See also section 0, RTL Configuration Parameters and Overrides.

2.5 IP/IPSS Straps

Table 1. Table NN IP Straps

Signal Name	Type	Description
do_serr_hier_dstid_strap	Input	When EP is mastering DO_SERR messages to the fabric (and when EP is in GLOBAL mode either GLOBAL_EP_STRAP or GLOBAL_EP is 1), these strap pins are used to create the hierarchical header DW. Can be driven to 0 otherwise.
do_serr_srcid_strap	Input	Strap values used to populate a DO_SERR error message that is sent out upon parity error detection
do_serr_dstid_strap	Input	
do_serr_tag_strap	Input	
do_serr_sairs_valid	Input	
do_serr_sai[SAIWIDTH:0]	Input	
do_serr_rs[RSWIDTH:0]	Input	

Signal Name	Type	Description
global_ep_strap	Input	This pin (similar to the GLOBAL_EP parameter), determines if the endpoint is a global endpoint with hierarchical headers. This signal is only used if the GLOBAL_EP_IS_STRAP parameter is set to 1. When driven to 1, the endpoint acts as a global endpoint and supports hierarchical headers. When driven to 0, the endpoint acts as a local endpoint without hierarchical header support (legacy behavior or if endpoint is in a local subnet). If not used, this pin must be driven to 0.

See also section 0, Fuses, for fuse-related straps.

2.6 Integrity Checks

Following are steps for running standalone integrity checks of this IP. It is assumed that the environment variable `IP_ROOT` is set to the path of the IP collateral.

1. Run the environment script:

```
setenv IP_ROOT <release_path>
cd $IP_ROOT/scripts
source setup -x ep_<variation> (e.g., source setup -x ep_bxt)
cd $IP_ROOT/scripts/qa
runFullConfig -endpoint
```

2. Run Lintra:

```
cd $IP_ROOT/tools
sbendpoint_runLintra
```

3. Compile the model:

```
cd $IP_ROOT/verif/sim
ace -cc
```

4. Run a simple test.

   To run basic test:

```
cd $IP_ROOT/verif/sim
ace -x
```

   To run the sample IP-level test "test01" (included in the release):

```
ace -x -t test01
```

   To run the test interactively:

```
ace -x -t test01 -sd gui
```

5. Run Synthesis:

```
cd $IP_ROOT/tools
sbendpoint_runSynth
```

6. Run CDC:

```
cd $IP_ROOT/tools
sbendpoint_runCDC / sbendpoint_runACECDC
```

7. Run LEC:

```
cd $IP_ROOT/tools
sbendpoint_runLEC
```

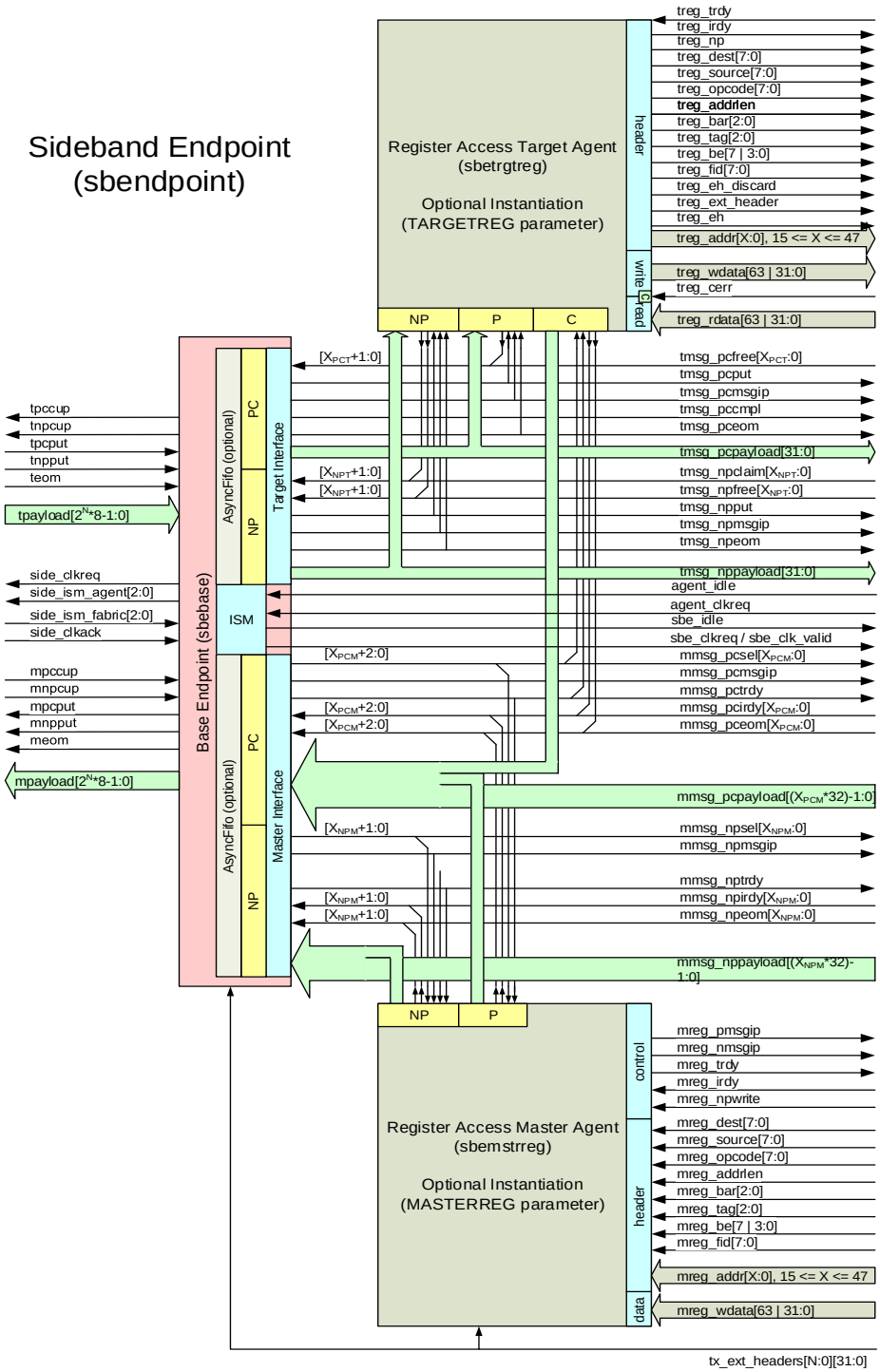
8. Run Scan Insertion:

```
cd $IP_ROOT/tools
sbendpoint_runATPG
```

3 Overview

3.1 Block Diagram

Figure 1. Sideband Endpoint Block Diagram



3.2 Functional Top-Level Signals

For a list of top-level signals, see section 12.2.1, Module Parameter/Port List.

The template for this document, SIP_signalList_template, is located in the central doc repository:

`https://sharepoint.amr.ith.intel.com/sites/SourceDocs/Templates/Forms/AllItems.aspx`

3.2.1 Signal List File

List of top level is generated at `source/rtl/iosfsbc/endpoint`

3.2.2 Tie-offs, Opens, and Gates

Not applicable to this IP

3.2.3 "Ad Hoc" Pins, Non-standard, or Non-compliant

Not applicable to this IP.

3.2.4 Control Signals

Not applicable to this IP

3.2.5 Reset Ports

Not applicable to this IP

3.2.6 Naming Scheme

Not applicable to this IP

3.2.7 End Points

Not applicable to this IP

3.3 Forces

3.3.1 Temporary Design Forces

Not applicable to this IP

3.3.2 Other Forces

Not applicable to this IP

4 Tools and Methodology for Integration

4.1 Configuration Updates

Not applicable to this IP/IPSS

4.2 Supported Tools

Note: Tool versions are listed in Release Notes, not in this document.

The following tools are used in the integration of this IP. For versions supported by each release, see Release Notes in the "doc" directory of the release package.

- VCSMX
- OVM
- Ace
- Saola
- Nebulon
- Blueprint
- Lintra
- Design Compiler
- Conformal
- 0-In
- VISA insertion

4.3 Environment Variables

The environment variables listed in the table below are required by the IOSF Sideband environment. They are initialized by sourcing the setup script in $IP_ROOT/scripts.

Variable	Description
ACE_ENG	ACE path to the head of the current IP root directory. Setup by post-setup Consumed by ACE
ACE_PROJECT	Tells the ACE environment that the current project is the IOSF Sideband Router. Setup by post-setup. Consumed by ACE
ACE_PWA_DIR	Name of the sideband root directory Setup by post-setup. Consumed by ACE
ACE_RC	ACE configuration file for IOSF Sideband Router. Setup by post-setup. Consumed by ACE

Variable	Description
LD_LIBRARY_PATH	Path to the IOSF Sideband VC library. Setup by post-setup. Consumed by ACE
DEBUSSY_HOME	Pointer to Verdi. Setup in post-setup. Consumed by ACE
MGLS_LICENSE_FILE	CDC License Servers. Setup in post-setup Consumed by Questa Zeroin
IP_ROOT	Defines the top level of the router design for backend tools. Setup in custom_pre.setup. Consumed by ACE, custom_post, runSpyglassLP, runVisa, block_setup.tcl, router.upf, ccu_vc, and pgcb_vc.
SIP_PROJECT	Defines the project name for the ccu bfm. Setup in custom_pre.setup. Consumed by ccu_vc.
SBC_SETUP_HAS_BEEN_CALLED	Enforcement of calling setup files within the IP directory prior to running SIP provided backend scripts. Setup in toolfile.dat Consumed by runPre before every backend script is called.
SIP_TOOL_VARIATION	Informs what customer toolfile.dat will be used when making calls to source setup -x <customer>. Setup in toolfile.dat. Consumed by runSynth.
KIT_PATH	Path to the KIT that is going to be used by the SIP provided backend tools. These should be in the /p/kits/intel/ directory for predictability and reliability of the scripts. Setup in toolfile.dat. Consumed by runSynth, runCDC, runATPG, runLEC.
KIT_HDL	hdl_spec file for the standard cells. Setup in toolfile.dat, and consumed in simulation and runACECDC.
SB_STDCELLS_HDL	Probed in the ace environment, in order to specify whether or not the current tool requires the KIT_HDL file. Setup in toolfile.dat, and consumed in all tool runs, as follows: simulation and runACECDC require that the SB_STDCELLS_HDL variable is set to "SB_STDCELLS_HDL" (default value in toolfile.dat), while runLintra, runCDC, runSynth, runATPG, runLEC, and runACEEmulation, require that this variable is unset/different than "SB_STDCELLS_HDL".
SIP_PROCESS_DIR	Legacy variable, not currently used.
SIP_LIBRARY_TYPE	Defines what library type should be used for the current KIT_PATH. Setup in toolfile.dat Consumed by runATPG, runCDC, runSynth.
SIP_LIBRARY_VOLTAGE	Defines what library voltage type should be used for the current library type. Setup in toolfile.dat Consumed by the synthesis tcl file additional_upf_setup.tcl.
LINTRARULES	Defines what Lintra rules to run the source code against in Lintra. Setup in toolfile.dat Consumed by runLintra.

4.4 HIP Libraries Included in Release

Not applicable to this IP

4.4.1 HIP IPToolData.pm Change

Not applicable to this IP

4.4.2 Register Files or SRAM

Not applicable to this IP

4.4.3 M-PHY and Related Libraries

Not applicable to this IP

4.5 Directory Structure for Tools

```
-- <UNTARRED_RELEASE_DIR>
    |-- ace
    |   `-- lib
    |       `-- iosf_sbc
    |-- doc
    |-- scripts
    |-- source
    |   `-- rtl
    |       `-- iosfsbc
    |           |-- common
    |           `-- router
    |-- tools
    |   |-- atpg
    |   |   `-- atpg_<PROCESS>
    |   |       `-- sbebase
    |   |       `-- sbendpoint
    |   |-- cdc
    |   |   |-- tests
    |   |   |   `-- sbebase
    |   |   |   `-- sbendpoint
    |   |-- emulation
    |   |   |-- fpga
    |   |-- lec
    |   |   `-- lec_<PROCESS>
    |   |       `-- sbebase
    |   |       `-- sbendpoint
    |   |-- lintra
    |   |   `-- sbebase
    |   |   `-- sbendpoint
    |   |-- spyglassdft
    |   |-- spyglasslp
    |   |-- syn
    |   |   `-- syn_<PROCESS>
    |   |       `-- sbebase
    |   |           `-- rdt
    |   |               `-- scripts
    |   |       `-- sbendpoint
    |   |           `-- rdt
    |   |               `-- scripts
    |   |-- upf
    |   |   `-- sbendpoint
    |   |-- verdi
    |   |-- visa
    |   |   `-- sbebase
    |   |   `-- sbendpoint
```

```
|    `-- zirconqa
|-- unsupported
 `-- verif
    |-- bfm
    |   |-- ccu_vc
    |   |-- pgcb_vc
    |   |-- psmi_oob
    |   `-- sideband_vc
    |-- sim
    |   |-- log
    |   |-- misc
    |-- tb
    `-- tests
         `-- ep_tests
```

4.6 Ace

Paths to acerc: `$IP_ROOT/ace`

Location of udf file: `$IP_ROOT/ace`

Model to use when importing libraries: IosfSbEpTestbench

Elaboration options not exported: N/A

Required content in `sip_shared_libs`: `sip_shared_libs` are imported

4.7 Lintra

After the setup script is run for the given customer toolset, the helper script sbendpoint_runLintra can be run out of the $IP_ROOT/tools/ directory.

Lintra Version: Defined in customer toolfiles

Lintra location: Defined in customer toolfiles

Location of waiver files: `$IP_ROOT/tools/lintra/iosf_sbc_ep_waivers.xml`

See also the in-line pragma waivers.

Location of Lintra patches & configuration: runLintra uses the configuration file `$LINTRARULES/cfg/LintraSoCConfig.xml`

Location of Lintra report file for warnings and errors:Reports are dumped out to `$IP_ROOT/tools/lintra/sbendpoint/`

4.8 Synthesis

All synthesis related scripts reside in:
`$IP_ROOT/tools/syn/syn_<PROCESSVERSION>/sbendpoint/rdt/scripts/`

Use the runFullConfig script in $IP_ROOT/scripts/qa, to generate helper scripts for all tools:

```
> cd $IP_ROOT/scripts
> source setup -x ep_<variation>   (e.g., source setup -x ep_bxt)
> cd $IP_ROOT/scripts/qa
> runFullConfig -endpoint
```

The synthesis helper script, sbendpoint_runSynth, can be run out of the `$IP_ROOT/tools/` directory.

The scripts directory contains the following standard flow configuration files:

- `block_setup.tcl`
- `global_constraints.tcl`
- `global_cross_clk_exceptions.tcl`
- `rtl_list.tcl`
- `sbendpoint_clocks.tcl`
- `sbendpoint_io_constraints.tcl`
- `sbendpoint_scan_config.tcl`

4.8.1 Clocks

The primary clocks are listed in Table 2.

Table 2. Primary Clocks

No.	Clock Name	Clock Period	Clock Waveform	Clock Source
1	side_clk	User defined	User defined	Primary clock input
2	agent_clk	User defined	User defined	Primary clock input

This IP does not have generated clocks, and does not require clock balancing during CTS.

4.8.2 Clock Propagation Specification

Not applicable to this IP

4.8.3 Clock Diagram

See Figure 1, Sideband Endpoint Block Diagram.

4.8.4 Constraint Files

All constraints files are available under:

`$IP_ROOT/tools/syn/syn_<PROCESSVERSION>/sbendpoint/rdt/scripts/`

4.8.5 Synthesis Best Known Methods (BKMs)

Not applicable to this IP

4.8.6 Latches

The IP provides an option for a latch-based implementation of the FIFOs (refer to the description of the LATCHQUEUES parameter in Table 10, Clock, Reset, and DFx/VISA Signals.

4.8.7 BKMs

The parameter settings for the IP should be carefully customized to the specific requirements of the agent.

4.8.8 Gray Pointer Clock Domain Crossings on Async Endpoints

- `*sbebase/*sbcasyncingress/gray_rptr* -> *sbebase/*sbcasyncingress/syncrptr*/q`
- `*sbebase/*sbcasyncingress/gray_nprptr* -> *sbebase/*sbcasyncingress/syncnprptr*/q`
- `*sbebase/*sbcasyncingress/gray_wptr* -> *sbebase/*sbcasyncingress/syncwptr*/q`
- `*sbebase/*sbcasyncegress/gray_rptr* -> *sbebase/*sbcasyncegress/syncrptr*/q`
- `*sbebase/*sbcasyncegress/gray_nprptr* -> *sbebase/*sbcasyncegress/syncnprptr*/q`
- `*sbebase/*sbcasyncegress/gray_wptr* -> *sbebase/*sbcasyncegress/syncwptr*/q`

4.9 Formal Verification

After the setup script is run for the given customer configuration and synthesis results are available, the helper script sbendpoint_runLEC can be run out of the `$IP_ROOT/tools` directory.

4.10 CDC

After the setup script is run for the given customer configuration, the helper scripts sbendpoint_runCDC, and sbendpoint_runACECDC can be run out of the `$IP_ROOT/tools` directory.

4.11 Scan

After the setup script is run for the given customer configuration, and synthesis results are available, the helper script sbendpoint_runATPG can be run out of the `$IP_ROOT/tools` directory.

Scan coverage is highly dependent on configuration.

Configurations should be above 90% based on past sbendpoint configurations.

4.12 DFD Triggers and Actions

Not applicable to this IP

5 Design Information for SoC Integration

This chapter is primarily targeted to the integration team responsible for integrating this IP/IPSS into an SoC. For subsystems, it describes the main SS and co-IPs.

5.1 RTL Directory Structure

```
-- <UNTARRED_RELEASE_DIR>
    |-- source
    |   |-- rtl
    |       |-- Makefile
    |       `-- iosfsbc
    |           |-- common
    |           |   |-- sbc_map.sv
    |           |   |-- sbcasyncclkreq.sv
    |           |   |-- sbcasyncfifo.sv
    |           |   |-- sbccomp.sv
    |           |   |-- sbcegress.sv
    |           |   |-- sbcfifo.sv
    |           |   |-- sbcfunc.vm
    |           |   |-- sbcgcgu.sv
    |           |   |-- sbcglobal_params.vm
    |           |   |-- sbcingress.sv
    |           |   |-- sbcinqueue.sv
    |           |   |-- sbcism.sv
    |           |   |-- sbcport.sv
    |           |   |-- sbcstruct_local.vm
    |           |   |-- sbcusync.sv
    |           |   |-- sbcvram2.sv
    |           |   `-- sbff.sv
    |           `-- endpoint
    |               |-- sbebase.sv
    |               |-- sbemstr.sv
    |               |-- sbemstrreg.sv
    |               |-- sbendpoint.sv
    |               |-- sbetrgt.sv
    |               `-- sbetrgtreg.sv
```

5.2 Building Blocks for SoC Integration

Not applicable to this IP; the SoC customers define their own specific fabric connectivity for the SBR IP.

5.3 Embedded Building Blocks/Custom Logic

Not applicable to this IP

5.4 Macros used by IP/IPSS

Not applicable to this IP

5.5 RTL Configuration Parameters and Overrides

The RTL configuration parameters for this IP are listed in Table 9, Parameters, in section 12.2.1, Module Parameter/Port List. If the parameter is derived, it must not be changed by the user.

5.5.1 Mandatory Parameters

Not applicable to this IP

5.5.2 Test Data Register Parameters

Not applicable to this IP

5.6 Integration Dependencies

Users should ensure that the settings for the following VISA parameters are consistent with their project requirements: SBE_VISA_ID_PARAM, and NUMBER_OF_VISAMUX_MODULES.

5.7 RTL Uniquification

The Base Endpoint RTL can now be uniquified using the "uniquifying" script that is delivered along with the IP. The executable resides in the script folder within the IP_ROOT and requires a "liblist" file (also available in the scripts area). "Uniquifyme" (within the delivered script) changes all the RTL and only certain ACE files/libraries to be prepended with the desired prefix. The remaining TB and ACE UDF/ACERC files are post-edited by the script. So to rerun uniquifcation, undo all the previous changes and redo the steps again (only for EP).

Steps to uniquify EP:

```
cd $IP_ROOT
source /p/hdk/rtl/hdk.rc -cfg sip -reentrant
./scripts/uniquifyme <prefix>
```

5.8 Registers

Not applicable to this IP/IPSS; Sideband Endpoint does not contain any registers.

5.9 Fuses

Not applicable to this IP/IPSS

5.9.1 Fuse RDL Files

Not applicable to this IP/IPSS ; Sideband Endpoint does not pull Fuses.

5.10 Clock, Power, and Reset Domains

See sections 13 (Clock and Reset), 15.9 (Design for Clock Gating), and 15.10, (UPF—Isolation and Retention).

5.10.1 Clock Domain Diagram

This IP has two clocks: side_clk, and agent_clk. If these clocks are different, instantiate the optional AsyncFifo module, as shown in Figure 1, Sideband Endpoint Block Diagram.

5.10.2 Clock Requirements (HIP only)

Not applicable to this IP

5.10.3 Power Domains

Not applicable to this IP

5.10.4 Power Domain Mapping

The IP power port names used in the UPF files are parameterized, and, therefore, can be mapped to SoC power supply rails as needed, e.g.:

```
set SOC_GROUND_PORT vss
```

5.11 System Startup

5.11.1 Power-up Sequence

Not applicable to this IP

5.11.2 Initialization Sequence

Not applicable to this IP

5.11.3 Device Configuration

Not applicable to this IP

5.11.4 Header for Windows Boot

Not applicable to this IP

5.11.5 Voltage Rail Requirements (HIP only)

Not applicable to this IP

5.12 System Startup

5.12.1 Power-up Requirements

Not applicable to this IP

5.12.2 Power-up Sequence

Not applicable to this IP

5.12.3 Initialization Sequence

Not applicable to this IP

5.12.4 Device Configuration

Not applicable to this IP

5.12.5 Header for Windows Boot

Not applicable to this IP

5.13 DFx Considerations

The IP is compliant with the Chassis DFx Gen2 standard.

5.13.1 DFx Top-Level Signals

See Table 10, Clock, Reset, and DFx/VISA Signals, in section 12.2.1, Module Parameter/Port List.

5.13.2 DFx Clock Definition

Not applicable to this IP

5.13.3 Clock Crossings

Synchronous endpoints (ASYNCENDPT=0) have no clock crossings. On the other hand, async endpoints (ASYNCENDPT=1) have clock crossings between side_clk and agent_clk

5.13.4 VISA

See section 15.7, VISA/DFx Features.

5.13.5 DFD Triggers and Actions

Triggering can happen based off of Sideband DSOs instantiated at SOC or through NPK ODLA, via VISA signal capture.

5.13.6 Debug Registers

Not applicable to this IP/IPSS

5.13.7 Scan – Clock Gating in RTL

Refer to Table 14 for the scan signals.

5.13.8 Scan – Reset Override

Scan reset-override is external to this IP.

5.13.9 Scan – Constraints and Coverage

Scan Constraints

There are no scan constraints. All flops/modules/instances are scan testable.

Stuck At and At-Speed Test Mode Procedures

The scan coverage varies with EP configuration. For one such configuration:STUCKAT: 99.36% ATSPEED: 98.5%

5.13.10 TAP and Associated Registers

Not applicable to this IP

5.13.11 Boundary Scan Parameters

Not applicable to this IP

5.13.12 Intra-Die Variation (IDV) - for HIP only

Not applicable to this IP

5.14 Security

See the Security Development Lifecycle site for information on security-related design practices:

https://sp2010.amr.ith.intel.com/sites/sdl/SitePages/Home.aspx

5.14.1 SAI Width

See parameter SAIWIDTH in Table 9, Parameters, in Section 12.2.1, Module Parameter/Port List.

5.14.2 Validation Requirements

Not applicable to this IP.

5.14.3 Interface Signals Implemented for Security

See Table 10, Clock, Reset, and DFx/VISA Signals, for a description of the security-related I/Os: ur_rx_sairs_valid, ur_rx_sai, ur_rx_rs, ur_csairs_valid, ur_csai, and ur_crs.

5.15 Safety Support

Reference Spec:

- IP_Chassis_FuSa_HW_Architecture_Guideline

  goto/docktracker

  Search For "Functional Safety (FuSa) - Chassis 2.2 "
- IP_SoC_HW_Systematic_FuSa_Requirement_Guideline

  IP_SoC_Development_FuSa_Guidance_Link
- IP_Chassis_FuSa_**SW**_Architecture_Guideline

  TBD (WIP)
- IP_SoC_**SW**_Systematic_FuSa_Requirement_Guideline

  TBD (WIP)
- INTEL Functional Safety LifeCycle (FSLC)

  http://goto.intel.com/FuSaLifecycle

or

https://sharepoint.amr.ith.intel.com/sites/FUSA_GlobalDomain/FuSaPublic/Published%20FuSa%20Lifecycle/Forms/AllItems.aspx

5.15.1 Safety HW Compliance

The following table shows compliance status for several areas, based on INTEL IP Hardware (HW) Functional Safety Integrity Related Enabling Requirements.

Table 3.

Safety HW Integrity Standard Compliance		IP Support Value
Coding Protection on Arrays (General IP)	ARYP.1 - ARYP.*	ARYP.1 - ARYP.10
Error Reporting (General IP)	ER.1 - ER.*	ER.1 - ER.5
In-Field Test For Logic (Specific IP)	IFLB.1 - IFLB.*	IFLB.1 - IFLB.3
E2E Coding Protection on Fabrics (General IP)	EEP.1 - EEP.*	EEP.1 - EEP.5
Configuration Data Protection	CDP.1 - CDP.*	CDP.1 - CDP.3
Voltage Monitoring (Specific IP)	CFD.1 - VFD.*	CFD.1 - VFD.3
Clock Monitoring (Specific IP)	CFD.1 - CFD.*	CFD.1 - CFD.4
Power Mgmt Components (Specific IP)	PM.1 - PM.*	PM.1 - PM.4
Safety-Relevant Microcontrollers (Specific IP)	UC.1 – UC.*	UC.1

The following table shows the values for several areas, based on IP HW Development Quality Measurement Systems.

Table 4.

HW Systematic Standard Type		IP Support Value
Specification Development Cycle	SPE.1 – SPE.*	SPE.1
Design Entry	DE.1 - DE.*	DE.1 - DE.6
Functional Verification	VE.1 - VE.*	VE.1 - VE.5
Physical Design (Synthesis)	SYN.1 - SYN.*	SYN.1 - SYN.7
Test (DFx)	TST.1 - TST.*	TST.1 - TST.4
Placement & Routing	PRL.1 - PRL.*	PRL.1 - PRL.7
Infrastructure	INFR.1 - INFR.*	INFR.1 - INFR.2

The following links show Dashboard of IP Development Quality Level, based on IP HW Development Quality Measurement Systems.

Example:

PLC Quality Measurement Score Dashboard Link

IPDS Quality Measurement Score Dashboard Link

IP HW Validation Quality Measurement Score Dashboard Link (QoV)

Other Quality Measurement Dashboard Link

……..

5.16 Emulation Support

The EFFM emulation flows are fully enabled in the project environment, as can be determined by running the "ace -sh EMULATION" command.

5.17 Other Design Considerations

Not applicable to this IP

5.18 IP META Data

For this IP, customer define their specific configuration meta data.

6 Physical Integration

6.1 Memory Requirements

Array type and number of instances	N/A
Functional usage (how many bits are used)	N/A
Highest functional clock frequency	N/A
Floorplan details	N/A
Security requirements	N/A
IP/IPSS power draw limitations for array testing	N/A

6.2 Subsystem Requirements

6.2.1 Hierarchy Details

Not applicable to this IP

6.2.2 Area and Floorplan Details

Not applicable to this IP

6.2.2.1 Boundary

Not applicable to this IP

6.2.2.2 Area Allocations and Confidence

Not applicable to this IP

6.2.2.3 Interface Requirements and Confidence

Not applicable to this IP

6.2.3 Fabric Convergence Requirements and Confidence

Not applicable to this IP

6.2.4 OTH Straps (Logical)

Not applicable to this IP

6.2.5 Clocking Domains and Entry/Exit Points

Not applicable to this IP

6.2.6 Global Elements

Not applicable to this IP

6.2.7 Power Domains and Power Grid Interface Requirements

Not applicable to this IP

6.2.8 Timing Considerations

Not applicable to this IP

6.2.9 Power

Not applicable to this IP

6.2.10 QRE-related Information

Not applicable to this IP

6.3 Open Issues

Not applicable to this IP

7 Connectivity Chains and IP/IPSS-specific Features (HIP only)

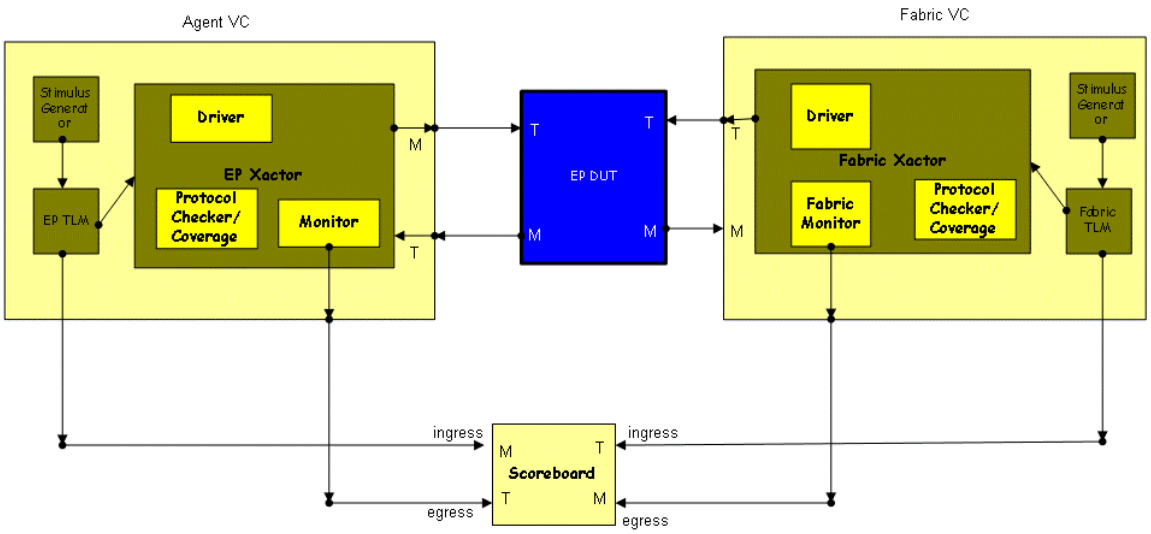
Not applicable to this IP

8 Verification Environment for Integration

8.1 IP Testbench Overview

The IP testbench environment is shown in Figure 2.

Figure 2. IP Testbench Environment



8.2 SoC-specific Validation

The IOSF sideband endpoint environment provides comprehensive testing of all features.

8.3 Validation Parameters

8.3.1 #defines

Not applicable to this IP

8.3.2 +define Command Arguments

Not applicable to this IP

8.3.3 Parameters

Table 5 lists all testbench configuration parameters for this IP.

The testbench parameters can be passed into a test by using a "-prescript_args" switch, in the ace command line, for example:

```
ace -x -test test02 -prescript_args "-targetreg 1 -asyncendpt 1 -usync_enable
0"
```

Alternatively, the desired parameter values can also be defined in a config file (for example, `$IP_ROOT/source/cfg/endpoint/csv/sbendpoint_unique_eh.csv`), and passed into the test as follows:

```
    ace -x -test test02 -prescript_args "-csv
$IP_ROOT/source/cfg/endpoint/csv/sbendpoint_unique_eh.csv "
```

Table 5. Testbench Configuration Parameters

Parameter Name	Range	Default	Descriptions (including interdependencies)
maxpldbit	7, 15, 31	7	IOSF interface payload width
npqueuedepth	1,…,31	3	Non-posted ingress queue depth
pcqueuedepth	1,…,31	3	Posted ingress queue depth
maxnptrgt	0,…,4	0	Maximum number of non-posted target widgets
maxpctrgt	0,…,4	0	Maximum number of posted/completion target widgets
maxnpmstr	0,…,4	0	Maximum number of non-posted mster widgets
maxpcmstr	0,…,4	0	Maximum number of posted/completion master widgets
asyncendpt	0, 1	0	Asynchronous endpoint when this value is 1
asynciqdepth	1,…,31	2	Queue depth of the asynchronous ingress FIFO
asynceqdepth	1,…,31	2	Queue depth of the asynchronous egress FIFO
targetreg	0, 1	1	Target register module inserted when value is 1
masterreg	0, 1	1	Master register module inserted when value is 1
asyncendpt	0, 1	0	Asynchronous endpoint when this value is 1
asynciqdepth	1,…,31	2	Queue depth of the asynchronous ingress FIFO
asynceqdepth	1,…,31	2	Queue depth of the asynchronous egress FIFO
targetreg	0, 1	1	Target register module inserted when value is 1
masterreg	0, 1	1	Master register module inserted when value is 1
maxtrgtaddr	15, 31, 47	31	Maximum address bits for target register module
maxtrgtdata	31, 63	63	Maximum data bits for target register module
maxmstraddr	15, 31, 47	31	Maximum address bits for master register module
maxmstrdata	31, 63	63	Maximum data bits for master register module
rx_ext_header_support	0, 1	0	Set to 1 if the endpoint should be able to receive transactions with extended headers.
rx_ext_header_ids	0,…,127	0	Indicates the extended header IDs the agent understands
num_rx_ext_headers	0	0	A zero based number indicating how many extended headers the endpoint can receive; for example, to receive one extended header, set to 0. Value is ignored if RX_EXT_HEADER_SUPPORT is 0.
tx_ext_header_support	0, 1	0	Set to 1 if the endpoint should be able to send transactions with extended headers.
num_tx_ext_headers	0	0	A zero based number indicating how many extended headers the endpoint can send; for example, to send one extended header, set to 0. Value is ignored if TX_EXT_HEADER_SUPPORT is 0.
disable_completion_fencing	0, 1	0	Set to 1 to disable the completion fencing algorithm
pipeisms	0, 1	0	Set to 1 to pipeline the ISM inputs on the fabric side. Must be equal to PIPEINPS.

Parameter Name	Range	Default	Descriptions (including interdependencies)
pipeinps	0, 1	0	Set to 1 to pipeline all fabric side inputs (except ISM Inputs). Must be equal to PIPEISMS.
usync_enable	0, 1	0	Parameter to enable deterministic clock crossing in an asynchronous endpoint. The feature is disabled by default; set parameter to 1 to enable.
unique_ext_headers	0, 1	0	Set to a value of 1 to have the endpoint sequence in the extended header inputs and generate extended headers as needed. The tx_ext_headers input will be ignored in this mode and the rx extended header outputs will become active.
saiwidth	0,…,15	15	Indicates the upper bound of the SAI field used across all modules within the endpoint.
rswidth	0,…,3	3	Indicates the upper bound of the SAI field used across all modules within the endpoint.
agt_clk_period	250,…,10000	10000	250 = 4GHz, 10000=100MHz
fab_clk_period	250,…,10000	10000	250 = 4GHz, 10000=100MHz
agent_usync_delay	1,…,10	1	Used to set a counter within the universal synchronizer circuit to delay the transfer from agent_clk to side_clk. This value must at least 1. If usyncselect is 1 this value must be 1.
side_usync_delay	1,…,10	1	Used to set a counter within the universal synchronizer circuit to delay the transfer from side_clk to agent_clk. This value must be at least 1. If usyncselect is 1 this value must be 1.
expected_completions_counter	0, 1	0	Used to enable a counter within the IOSF port to track the number of ingressing non-posted messages to track how many completions should come back. Defaults to ISM_COMPLETION_FENCING.
ism_completion_fencing	0, 1	0	When EXPECTED_COMPLETIONS_COUNTER and ISM_COMPLETION_FENCING are equal to 1 the IOSF ISM will remain in the ACTIVE state until all completions have egressed the IOSF interface; for example, when the expected completions counter has reached zero.

8.3.4 Variables

Not applicable to this IP

8.3.5 Overrides

Not applicable to this IP

8.3.6 Plusargs

Not applicable to this IP

8.3.7 Pre- and Post- TI Imports

Not applicable to this IP

8.4 Verification Libraries

Not applicable to this IP

8.5 Testbench Components and Connectivity

In the following subsections, list major validation structures included in drop and how they connect to the DUT.

8.5.1 Testbench Directory Structure

Here is a short description of the VC files. As a general rule, each class has its own file and each file is named the same as the class it contains.

Table 6. Description of Each File used in Val Environment

iosf-sideband-fabric	
|	
tb	
|	
|--|--fabric_vc	- Directory of fabric VC files
| |-- iosfsbm_fbrc_pkg.sv	- VC package to compile
| |-- iosfsbm_fbrcvc.svh	- VC top level
| |-- fabric_ism.svh	- TLM Fabric ISM
| |-- fabric_tlm.svh	- TLM driver
| |-- fbrcvc_api.svh	- Contains Fabric VC APIs
| |-- fbrcvc_cfg.svh	- Contains Fabric VC Config Desciptor
|--|-- common	- Directory of shared files
| |-- iosfsbm_cm_pkg.sv	- Common files package to compile used by VCs
| |-- checks_control_cfg.svh	- Config descriptor for controlling checks
| |-- common_cfg.svh	- Config descriptor for common configuration
| |-- ep_cfg.svh	- Config descriptor for TLM driver
| |-- defs.svh	- enum types used by VCs
| |-- xaction.svh	- Transaction class base class
| |-- msgd_xaction.svh	- Transaction class message with data
| |-- regio_xaction.svh	- Transaction class regio message
| |-- simple_xaction.svh	- Transaction class simple message
| |-- comp_xaction.svh	- Transaction class completion message
| |-- opcode_cb.svh	- completion generation callback base class
| |-- default_msgd_cb.svh	- completion generation callback msg with data

iosf-sideband-fabric	
| |-- default_regio_cb	- completion generation callback regio message
| |-- default_simple_cb.svh	- completion generation callback simple message
| |-- default_comp_cb.svh	- completion generation callback completion message
| |-- xactor_base.svh	- Base class for xactor
| |-- xactor.svh	- Xactor class
| |-- xactor_passive.svh	- Passive xator class
| |-- responder.svh	- Responder master or target bus
| |-- driver.svh	- Driver master bus
| |-- ism.svh	- ISM base class master bus
| |-- iosfsb_intf_wrapper.svh	- iosf_sbc_intf wrapper class
| |-- iosfsb_macros.svh/iosfsb_params.svh	- Default parameters
| |-- xaction_counters.svh	- Global transaction counter class
| |-- queue_wrapper.svh	- SV queue wrapper class
| |-- queue_wrapper_assoc.svh	- SV assoc queue wrapper class
| |-- xaction_cov.svh	- Cover Group transaction
| |-- plus_args.svh	- plus args parsing class
| |-- tlm_pc.svh	- TLM protocol checker
| |-- memory_model.svh	- Memory model
| |-- utils.svh	- Utils class
| |-- iosfsbc_sequencer.svh	- Sequencer Class
| |-- sai_footer.svh	- Sai footer object
| |-- pass_thru_seq.svh	- Pass Thru Sequence
|--|-- agent_vc	- Directory of agent VC files
| |-- iosfsbm_agent_pkg.sv	- VC package to compile
| |-- iosfsbm_agtvc.svh	- VC top level
| |-- agent_ism.svh	- TLM Agent ISM
| |-- ep_tlm.svh	- TLM driver
| |-- pwr_dmn_cfg.svh	- Config descriptor power wells
| |-- pwr_manager.svh	- TLM power manager
| |-- agtvc_api.svh	- Contains Agent VC APIs
| |-- agtcvc_cfg.svh	- Contains Agent VC Config Desciptor
|--|-- env	- Directory of monitor files
| |-- iosfsbm_agent_fbrc_env_pkg.sv	- Package to import
| |-- env_fabric_agent.svh	- Environment object StandAlone tb
|--|--intf	- Directory of interfaces
| |-- iosf_sbc_intf.sv	- Sideband interface
| |-- comm_vintf.sv	- VC Communication interface

iosf-sideband-fabric	
| |-- clk_rst_intf.sv	- Clock,Reset interface
| |-- connect_sbc_intf.sv	- Connect two sideband interface
| |-- power_intf.sv	- Power interface
| |-- connect_sbc_intf.sv	- Connect two sideband interface
| |-- iosf_sbc_chk_crd.sv	- Assertion properties for credits
| |-- iosf_sbc_chk_ism.sv	- Assertion properties for ISM
| |-- iosf_sbc_cov_props.sv	- Cover properties
| |-- iosf_sbc_coverage.sv	- Cover Groups master target bus
| |-- connect_sbc_intf.sv	- Connect two sideband interface
|--|--sequences	- Directory of Sequences
| |-- unicast_rnd_seq.svh	- Trans Generator Random unicast
| |-- directed_seq.svh	- Trans Generator Directed
| |-- base_seq.svh	- Base sequence class
| |-- exhaustive_seq.svh	- Trans Generator exhaustive
| |-- loopback_seq.svh	- Trans Generator Loopback
| |-- rnd_bcast_mcast_seq.svh	- Trans Generator bcast_mcast
| |-- single_bcast_mcast_seq.svh	- Trans Generator single bcast_mcast
| |-- unicast_rnd_invalid_opcode_seq.svh	- Trans Generator invalid opcode
| |-- unicast_rnd_seq_posted_only.svh	- Trans Generator unicast posted
| |-- unicast_rnd_seq_nonposted_only.svh	- Trans Generator Unicast nonposted
| |-- unsupported_pid_seq.svh	- Trans Generator unsupported pid
| |-- rnd_seq.svh	- Trans Generator random
| |-- simple_seq.svh	- Trans Generator to send simple messages
| |-- msgd_seq.svh	- Trans Generator to send msgd messages
| |-- regio_seq.svh	- Trans Generator to send regio messages
| |-- vintf_seq.svh	- Trans Generator Directed with access to virtual Interface
| |-- polling_seq.svh	- Trans Generator Directed read with waiting for completion
| |-- pass_over_seq.svh	- Trans Generator Random with send_xaction API
| |-- iosf_sb_seq.svh	- Trans Generator Random with send_xaction API, also supports ovm_do_on_with macro
|	
tests	- Directory of tests, sequences
| |-- back2back	- Directory of Stand-Alone tb tests
| |-- base_test.svh	- Base Test Stand-alone TB
| |-- test01.svh	- Directed Test + credit update, completion delay API and loop for getting completion back from sequencer
| |-- test02.svh	- Random Test + tasks to get rx and tx messages from VC+ credit reinit API
| |-- test03.svh	- Directed Test + Sending xactions using ep specific opcodes + set response type for particular opcode + use of

iosf-sideband-fabric	
| |-- test04.svh	- Directed Test to send write and read messages and use of memory to store/retrieve completion data
| |-- test05.svh	- Directed Test to send couple of write followed by read to get completion back using ovm REQ/RSP channel
| |-- test06.svh	- Directed test to send xactions without using pass_over_seq
| |-- test07.svh	- Use of ovm REQ/RSP channel for simple, msgd xactions using polling_simple_seq and polling_msgd_seq, use of ovm_do_with macro
| |-- test08.svh	- load_compl_data API used to store completion data into memory
| |-- test09.svh	- Use of iosf_sb_seq to send all types of xactions using send_xaction API and ovm_do_on_with macro
| |-- test10.svh	- Use of regio_seq with compare_completion field set.
| |-- iosftest_pkg.sv	- Package to import
|	
Scripts	- Directory of acripts
| |-- all.f	- List of files for fabric vc
| |-- clean_logs	- Script to clean log files
| |-- clean_sim	- script to clean sim files
| |-- run_test	- script to run fbrc vc test using vcs

Table 7. User Guide and Script to Launch Regression

| |-- tool_setup	**- script to setup tools**
| |-- vcs_args.f	- script for vcs arguments
| |-- minigress	- script to run all tests
| |-- all_compliance.f	- List of files for Compliance monitor
| |-- minigress_list	- List of tests for miniregress script
| |-- run_doxygen.pl	- Script to run doxygen, creats html document
|	
sim	- Directory from which to run Router TB test
|-- README.txt	- Readme file
|	
doc	- Document
|-- User_Guides	- VC User Guide/Quick start guide
| |-- IOSF_Sideband_Message_VC_User_Guide.doc	
| |-- IOSF_Sideband_Message_VC_User_Guide.pdf	
| |-- IOSF_Sideband_Message_VC_Quick_start Guide	
| |-- IOSF_Sideband_Message_VC_Quick_Start_Guide	
|-- TestPlans	- IOSF Sideband TestPlan
| |-- IOSF_Sideband_TestPlan.doc	
| |-- IOSF_Sideband_TestPlan.pdf	

| |-- tool_setup	**- script to setup tools**
svlib	- Directory for SV utilities
ovmlib	- Directory for ovm libraries

8.5.2 Test Islands

Not applicable to this IP

8.5.3 Test Island Interfaces

Not applicable to this IP

8.5.4 Checkers and Trackers

Not applicable to this IP

8.5.5 Monitors

Monitors are used to capture transaction for scorebaording and responding go the transaction. IOSF Compliant monitor is also checking the protocol level correctness. All of these are enabled and reusable in SoC.

8.5.5.1 Compliance Monitor

Compliance monitor is provided by architecture group, which is instantiated in the iosf_sbc_intf interface. User can choose to disable this using +define+ DISABLE_IOSFSB_COMPLIANCE argument or can use disable_compmon config parameter.

It is 0.83 compliant, for 0.81 interfaces user can use DISABLE_CLKACK parameter in the iosf_sb_ifc_compliance module to disable clkack related assertions.

Compliance monitor has Agent_ISM_Reset_IDLE, Fabric_ISM_Reset_IDLE, eh_support, ext_header_support straps which are driven by Fabric and Agent VCs based on the spec version.

It also have mcast_num_pids strap to indicate which pid belongs to mcast group, this strap is driven by VC as well.

8.5.5.2 Monitor

Monitor communicates with rest of the components using analysis ports. It assembles fabric and agent xactions by looking at master and target interfaces and these xactions are available through different analysis ports. fab_agt_pc_ap, fab_agt_np_ap and fab_agt_ap sends fabric to agent posted/nonposted/all xactions. agt_fab_pc_ap, agt_fab_np_ap, and agt_fab_ap sends agent to fabric posted/non-posted/all xactions.

8.5.6 Scoreboards

Checkers such as ip_vc Scoreboard, and Compliance Monitorfor for interface protocol checks, are part of the SVC deliverable provided with the IP release

8.5.7 BFMs

IP is delivered with specific IOSF_SVC version, with all necessary BFM's. (version described in release notes).

8.5.8 Other Structures

Not applicable to this IP

8.6 Collage or Sandbox Files

Not applicable to this IP

9 IP2SOC Handoff Information

Not applicable to this IP

9.1 <Feature 1>

9.1.1 Testbench Block Diagram

Not applicable to this IP

9.1.2 BFM

Not applicable to this IP

9.1.3 Tracker Checker

Not applicable to this IP

9.1.4 Fast Sim Mode

Not applicable to this IP

9.1.5 Generator

Not applicable to this IP

9.1.6 Power Flow

Not applicable to this IP

9.2 <Feature 2>

Not applicable to this IP

10 Workarounds, Waivers, and Disabled Assertions

The integration of this IP does not require any workarounds or waivers, or disabling of assertions.

10.1 Disabled Assertions

The integration of this IP does not require any disabling of assertions.

10.2 Waivers

Not applicable to this IP

10.3 Other Workarounds

Not applicable to this IP

11 Integration Checks and Tests

Prior to integrating this IP, the user should ensure correct functionality of the endpoint, by running the unit tests described in the following Table. It is recommended that agent validation owners create agent-level versions for the unit-level test listed in Table 8, Unit-Level Tests.

Table 8. Unit-Level Tests

Test name	Test Description
test01	Directed test for basic flow
test02	Random test for basic flow with sparse xactions
test03	Sends directed xactions only from agent side
test04	Sends directed xactions only from agent side, and disables clock gating; completion delay set to 60 for fabric
test05	Sends directed xactions, randomly claims all messages, randomly disables clock gating, async_reset with reset_test, separate reset for agent and fabric, asserts/de-asserts agent and fabric resets
test06	Disables clock gating, sends directed xactions only from fabric
test07	Disables clock gating, sends directed xactions from agent and fabric to cover all xaction level cg related to xaction_type
test08	Test to cover agent's ACTIVE_REQ to CREDIT_REQ ism transition, fabric's creditack ism delay set to 10, async_test, reset_test
test09	Sets clkack deassert delay to 100, agent claims all xactions, and agent's completion message for np is delayed by 300 cycles.
test10	Agent randonly claims all xactions, pc and np crd init delay set to 5 and 4, sends random xactions from agent and fabric
test11	Sends random xactions, sets crd init delay and crd updated delay for fabric
test12	Agent claims all xactions, sets np completion delay to be 0, and sets flag in the agent to send completion message after 12 posted messages are received.
test13	Sets crd update delay for fabric, disables clock gating, sends directed xactions from agent and fabric for covegare
test14	Sends random unicast invalid messages from agent and fabric
test15	Sets activereq delay for fabric , asserts fabric's reset to hit agent's activereq_req to credit_req ism
test16	Sets activereq delay to be 10 for fabric
test17	Sends directed xactions from fabrics
test18	Randomly claims all messages, asserts fabric/agent's reset and sends directed xactions
test19	Set not to claim any xactions, sends invalid messages from fabric and agent
test20	Sends directed xactions from agent/fabric

12 Block Level Micro-Architecture

12.1 Block Descriptions

12.1.1 Sideband Base Endpoint (SBEBASE)

This block contains the Sideband Base Endpoint. For more details, see the IOSF SBC Base Endpoint Integration Guide (included in the 'doc' directory for this release).

12.1.2 Register Access Master Agent (SBEMSTRREG)

This block provides an interface to the IP block for initiating (mastering) register access messages, either posted or non-posted. All inputs are asserted in a single cycle including the destination port ID, source port ID, opcode, address and address length, and 64-bit write data. If the request is a read, the write data is not used and the read data is returned on the Target Interface (tmsg_pc*; see the IOSF SBC Base Endpoint Integration Guide which is included in the 'doc' directory for this release). The endpoint has no knowledge of the assigned port ID of the IP block, so this must be provided by the IP block.

Also, there are reasons why the source port ID can vary, such as the IP block could support more than one port ID. In addition, the broad/multicast non-posted masters need to send 0xFE for the source port ID in order to request that the router(s) return a single coalesced completion rather than receiving a completion from every endpoint the received the broad/multicast non-posted message.

This block contains minimal logic to mux from the large parallel interface down to the dword payload width of the master interface of the base endpoint.

Also contained in this block are dword counters needed to control the mux selection and basic interface protocol logic for the IP block and the internal base endpoint.

12.1.3 Register Access Target Agent (SBETRGTREG)

This block provides an interface to the IP block for receiving target register access read/write messages and for initiating their associated completions.

Write requests are transferred to the IP block (when both endpoint and IP block are ready) in a single cycle including the full message header, the address and address length, and 64-bit write data.

Read requests are presented to the IP block when the endpoint is ready, holding the outputs to a constant state allowing the IP block to complete the read in a variable (yet finite) number of cycles.

The completion for non-posted messages is created within this block from the flops that hold the non-posted message and the flops that capture the read data and completion status.

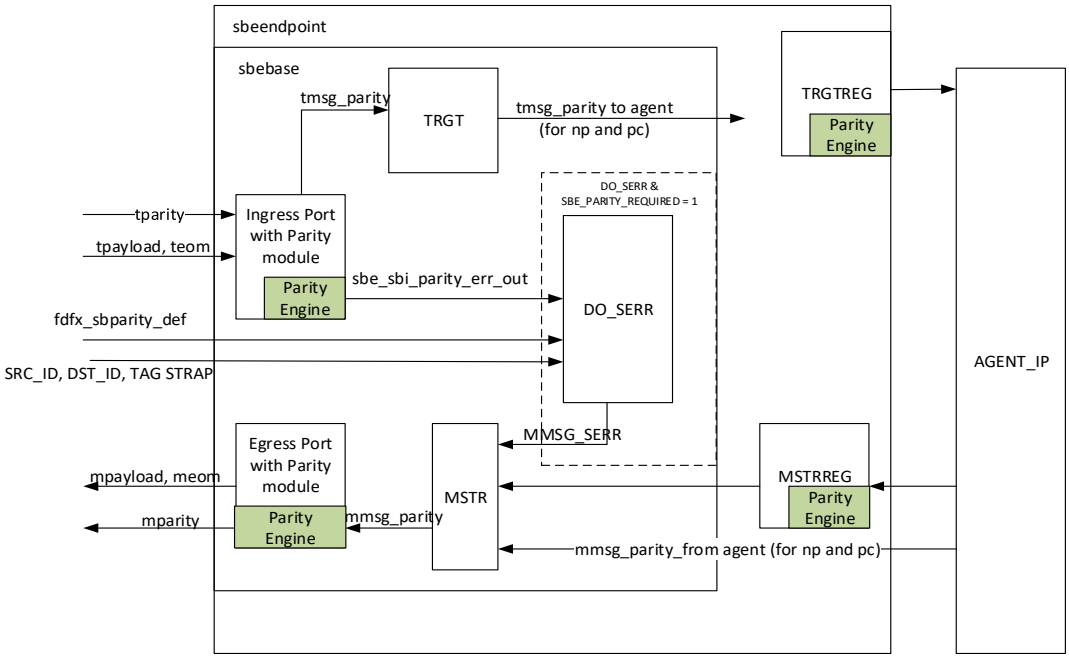
Note: The flops that are used to hold the write data are also used to capture the read data.

Note: If the SBETRGTREG is instantiated in the endpoint (parameter TARGETREG=1), then no target agents instantiated in the IP block (outside of the endpoint) are allowed to claim any register access messages as a target. The SBTRGTREG claims all of the defined global register access opcodes as defined in the IOSF specification.

12.1.4 DO_SERR Master Widget (DOSERRMSTR)

A DOSERRMSTR widget is instantiated in the Endpoint as illustrated in Figure 3, when the SB_PARITY_REQUIRED and DO_SERR_MSTR parameters are both set. When a parity error is detected, this module generates a DO_SERR error message (opcode 0x88), by employing the source/dest/tag information supplied through top-level straps (i.e., sbi_sbe_srcid_strap, sbi_sbe_dstid_strap, and sbi_sbe_tag_strap). When TX_EXT_HEADER_SUPPORT is also enabled, and the sbi_sbe_sairs_valid_strap is set, the strap values for SAI (sbi_sbe_sai_strap) and RS (sbi_sbe_rs_strap) are also packaged as a 2nd DW word in the DO_SERR message.

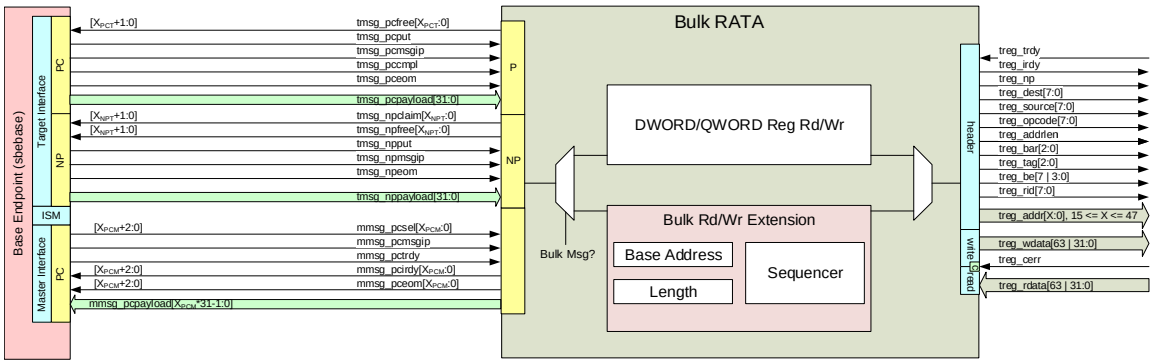
Figure 3. Instantiation of the DOSERRMSTR Widget



10.1.5 Bulk Read/Write Register Widget (BULKRDWR)

Setting the BULKRDWR parameter to 1 will instantiate a new target register widget, in place of the legacy RATA (enabled by setting TARGETREG to 1). Setting both the parameters will result in the bulk widget instantiation. Bulk widget combines the functionality of the legacy RATA, supporting single register operations together with the new bulk register feature. When EP recieves a message with the bulk opcode, the widget begins to sequence the addresses and generate separate reads/writes for each incremental address. More details of the implementation can be found in the Bulk HAS here:
https://sharepoint.amr.ith.intel.com/sites/docs_chassis/IOSFSb/IOSFSbDev/SBEP_RATA_Bulk_Extension_0p8.docx

Figure 4. Instantiation of the BULKRDWR Widget



12.2 Block Interface

12.2.1 Module Parameter/Port List

Table 9. Parameters

Parameter Name	Default Value	Expose at Top Level?	Description
MAXPLDBIT	7	Y	Defines the most significant bit of the sideband payload, which must be either 7, 15, or 31.
NP/PCQUEUEDEPTH	4	Y	The advertised depth of the NP/PC ingress queues, which must be in the range 1-31. In the absence of any performance requirements, this should be set to 1 to minimize the area.
RX_EXT_HEADER_SUPPORT	0		Set to 1 if the attached agent is able to receive transactions with extended headers. Set to 0 for EH unaware or legacy agents. When set to 0, the endpoint transparently removes the EH from any incoming transaction.set to 1 if GLOBAL_EP is set to 1.
NUM_RX_EXT_HEADERS	0		A zero based number indicating how many extended headers the endpoint can receive; for example, to receive one extended header, set to 0. Value is ignored if RX_EXT_HEADER_SUPPORT is 0.
LATCHQUEUES	0		When set to 1 the ingress queues are latched based queues, otherwise they are standard flop based queues. It is suggested to set this value to zero. There is no backend support for latches from the Sideband team and this may be deprecated in a future release.
MAXNPTRGT	0		The maximum bit of the tmsg_nptrdy signal, which defines the number of non-posted targets in the IP block.
MAXPCTRGT	0		The maximum bit of the tmsg_pctrdy signal, which defines the number of posted/completion targets in the IP block.
MAXNPMSTR	0		The maximum bit of the mmsg_npirdy signal, which defines the number of non-posted masters in the IP block.

Parameter Name	Default Value	Expose at Top Level?	Description
MAXPCMSTR	0		The maximum bit of the mmsg_pcirdy signal, which defines the number of posted/completion masters in the IP block.
ASYNCENDPT	0		Asynchronous Endpoint = 1, otherwise 0
ASYNCIQDEPTH	2	Y	Asynchronous ingress FIFO queue depth. Ignored if ASYNCENDPT = 0
ASYNCEQDEPTH	2	Y	Asynchronous egress FIFO queue depth. Ignored if ASYNCENDPT = 0
CUP2PUT1CYC	1		Deprecated. Any setting of this parameter is ignored.
VALONLYMODEL	0		Deprecated. Any setting of this parameter is ignored.
DUMMY_CLKBUF	0		Set to 1 to insert a dummy clock buffer at the root of the endpoint clock distribution tree. Required for some CPU DFT flows. Set to 0 for typical use cases.
TX_EXT_HEADER_SUPPORT	0		Set to 1 to indicate that the attached agent is capable of sending extended headers. This is required to append extended headers on endpoint-generated transactions such as completions for unclaimed NP transactions. Set to 0 if the agent does not transmit extended headers.
NUM_TX_EXT_HEADERS	0		A zero based number indicating how many extended headers to transmit; for example, to transmit one extended header, set to 0. Value is ignored if TX_EXT_HEADER_SUPPORT is 0.
UNIQUE_EXT_HEADERS	0		Set to a value of 1 to have the endpoint sequence in the extended header inputs and generate extended headers as needed. The tx_ext_headers input will be ignored in this mode and the rx extended header outputs will become active.
SAIWIDTH	15		Indicates the upper bound of the SAI field used across all modules within the endpoint. All other bits will either be appended with 0's or have bits dropped depending on the scenario. If intolerant of 0's or dropped bits in this field then the agent must size it appropriately.
RSWIDTH	3		Indicates the upper bound of the RS field used across all modules within the endpoint. All other bits will either be appended with 0's or have bits dropped depending on the scenario. If intolerant of 0's or dropped bits in this field then the agent must size it appropriately.
DISABLE_COMPLETION_ FENCING	0		Set to non-zero value to disable the completion fencing algorithm.
TARGETREG	1		When set to 1 the register access target agent (SBETRGTREG) is instantiated in the endpoint. This adds one additional PC and NP target and one additional PC master to the base endpoint. This causes the following parameters to be incremented by 1 inside the endpoint before passing the resultant value into the base endpoint: MAXNPTRGT, MAXPCTRGT and MAXPCMSTR.

Parameter Name	Default Value	Expose at Top Level?	Description
MASTERREG	1		When set to 1 the register access master agent (SBEMSTRREG) is instantiated in the endpoint. This adds one additional PC and NP master to the base endpoint. This causes the following parameters to be incremented by 1 inside the endpoint before passing the resultant value into the base endpoint: MAXPCMSTR and MAXNPMSTR. The completions are received on the Target Interface rather than re-flopping the read data and then presenting it to the IP block. This would be a waste of cells/area/power.
MAXTRGTADDR	31		This parameter selects the maximum address bit that the register-access target agent supports. The minimum value is 15; the maximum value is 47 and the default is 31. The IOSF spec allows for 2 different address lengths 16 and 48 bits, but the IP block can support something in between. If a message is received with an address larger than the IP block supports, then the message is not claimed, resulting in a posted message being silently dropped and a non-posted message can receive an unsuccessful / not supported completion generated by the internal unclaimed non-posted message target within the base endpoint. In either case, the message cannot be presented to the IP block on the target register access interface (treg_*).
MAXTRGTDATA	63		This parameter selects the maximum data bit that the register access target agent supports. The valid values are 31 or 63 and the default is 63. This parameter also controls the maximum bit of the treg_be signal, which is either 3 or 7 (respectively). If this parameter is set to 31 and a register access message is received with a non-zero second byte enable (sbe), then the message is handled in the same fashion as an addressing error described above in the MAXTRGTADDR parameter.
MAXMSTRADDR	31		This parameter selects the maximum address bit that the register-access master agent supports. The minimum value is 15; the maximum value is 47 and the default is 31. The IOSF spec allows for 2 different address lengths 16 and 48 bits, but the IP block can support something in between.
MAXMSTRDATA	63		This parameter selects the maximum data bit that the register access master agent supports. The valid values are 31 or 63 and the default is 63.
RX_EXT_HEADER_IDS [NUM_RX_EXT_HEADERS:0]	0		Indicates the extended header IDs the agent understands.
SKIP_ACTIVEREQ	1		Set to 1 value to skip ACTIVE_REQ of agent ism (per IOSF 1.0 specification). Should always be set to 1.

Parameter Name	Default Value	Expose at Top Level?	Description
PIPEISMS	0	Y	Set to 1 value to add pipeline of ISM inputs in the fabric side. Must be equal to PIPEINPS. The IOSF compliance monitor that exists today will have issues with state transitions that are now valid in IOSF 1.1.
PIPEINPS	0	Y	Set to 1 value to add pipeline of all fabric side inputs (except ISM Inputs). Must be equal to PIPEISMS.
SBE_VISA_ID_PARAM	11		Identifier for the VISA ULM, if inserted within the endpoint.
NUMBER_OF_BITS_PER_ LANE	8		Parameters to control VISA lanes and mux structure. For details, see the VISA IT documentation at https://intelpedia.intel.com/Visa_IT.
NUMBER_OF_VISAMUX_ MODULES	1		
NUMBER_OF_OUTPUT_ LANES	2		
USYNC_ENABLE	0		For details, see the IOSF SBC Base Endpoint Integration Guide (included in the 'doc' directory for this release).
SIDE_USYNC_DELAY	1	When exposing USYNC_ENABLE	
AGENT_USYNC_DELAY	1	When exposing USYNC_ENABLE	
EXPECTED_COMPLETIONS_COUNTER	0		
ISM_COMPLETION_FENCING	0	When exposing EXPECTED_COMPLETIONS_COUNTER	
SIDE_CLKREQ_HYST_CNT	15	Y	
SB_PARITY_REQUIRED	0	Y	When set to 1, enables parity support on the Endpoint
DO_SERR_MASTER	0	Y	When set to 1, and SB_PARITY_REQUIRED is also 1, a DOSERRMSTR master widget is instantiated. Upon the detection of a parity error, the DOSERRMSTR widget sends out a DO_SERR error message. No DO_SERR widget is instantiated when DO_SERR_MASTER=0, or SB_PARITY_REQUIRED=0.
MATCHED_INTERNAL_WIDTH	0		When set to 1, the internal payload width is the same as the external payload width, as specified through MAXPLDBIT (i.e., 7, 15, or 31). When the MATCHED_INTERNAL_WIDTH parameter is set to 0, the internal payload width is always 31, regardless of the value of MAXPLDBIT.

Parameter Name	Default Value	Expose at Top Level?	Description
CLAIM_DELAY	0		This parameter allows the agent to add delay on the tmsg_npclaim indication to the endpoint relative to tmsg_npeom. An agent might choose to use this feature due to repeater flops on tmsg_npclaim, instantiation of tmsg/mmsg buffers between sbebase and the TREG/MREG widgets, or any microarchitecture pipelining that delays the worst case assertion of the tmsg_npclaim signal relative to tmsg_npeom. When enabling this feature (i.e., CLAIM_DELAY >= 1), the user should also ensure that DISABLE_COMPLETION_FENCING=0. The CLAIM_DELAY parameter should be set to a value that is equal to ("number of pipeline delays on tmsg_npclaim" + "3x the number of tmsg/mmsg buffers between sbebase and the TREG/MREG widgets"). For instance, when the agent instantiates one pipeline flop on tmsg_npclaim, and no tmsg/mmsg buffers, CLAIM_DELAY must be at least 1. On the other hand, when the agent instantiates one tmsg/mmsg buffer stage, with no pipepline delay on tmsg_npclaim, CLAIM_DELAY must be at least 3. When this feature is not enabled (i.e., CLAIM_DELAY = 0), the endpoint samples the tmsg_npclaim indication from the agent only while the NP message is in flight, until the end-of-message boundary. On the other hand, when CLAIM_DELAY >= 1, the window when the endpoint samples the tmsg_npclaim from the agent is extended past the end-of-message flit until tmsg_npfree is asserted for CLAIM_DELAY number of cycles. In order to facilitate use of the CLAIM_DELAY feature, a reference design for a tmsg/mmsg buffer module is provided with the endpoint release. It can be found in the subIP folder of the endpoint release area (refer to …/subIP/reference_code/sberepeater.sv). Note that the claim delay math above assumes the microarchitecture of this reference design. A different buffer module with a different microarchitecture might have a multiplicative factor different than 3x.
GLOBAL_EP	0		This parameter enables the endpoint in a transparent bridge network which supports hierarchical header. When set to 1, the endpoint will be able to handle 1 additional DW in payload and treat it as a hierarchical header to and from the fabric and the agent IP (both as a master and as a target). When set to 0, the endpoint will operate as in local mode (or legacy).
GLOBAL_EP_IS_STRAP	0		This determines whether the hierarchical header feature, for global endpoints, is enabled through the parameter (GLOBAL_EP) or though the strap pin (global_ep_strap). The default value 0, indicates the feature is controlled by the parameter GLOBAL_EP. By setting it to 1, the global_ep_strap input determines if the endpoint is a global endpoint (hierarchical header).
RELATIVE_PLACEMENT_EN	0	Y	Enables the usage of "genram" modules instead of fifo. The "genram" modules are a subIP in sideband provided by the RP team which enable more efficient placement of storage structures.

Parameter Name	Default Value	Expose at Top Level?	Description
BULKRDWR	0	Y	Enables the Bulk Save and Restore feature, that instantiates a bulk target register widget (instead of the legacy widget) that can generate sequential address reads/writes from a given payload. See section 10.1.5 for more details.
BULK_PERF	0	Y	Used only when BULKRDWR is set to 1. Users can chose to either continuously stream read/write requests without any bubbles (BULK_PERF = 1) or take a performance hit, but run at higher frequencies because of pipelined stages (BULK_PERF =0).
CLKREQDEFAULT	0	Y	IOSF spec changed the default value of the output side_clkreq form 1 to 0. Since this would affect all IP's especially early boot IP's, CLKREQDEFAULT was added. Setting this to 1 would mean the side_clkreq would take the value of wake when in reset and would resume the old clkreq behavior after the reset. Setting CLKREQDEFAULT to 0 would be the default previous behavior of clkreq being 1 during reset.

Table 10. Clock, Reset, and DFx/VISA Signals

Signal Name	Type	Description
side_clk	Input	The endpoint clock input
gated_side_clk	Output	The gated side clock output. For agents, the output clock is active when the agent ISM is in ACTIVE, the agent and fabric ISM are in ACTIVE_REQ, or the agent ISM is in IDLE_REQ and the fabric is not IDLE. Also subject to cfg* overrides specified below.
side_rst_b	Input	The asynchronous active low endpoint reset. This reset is expected to be synchronized and have scan bypasses inserted prior to the agent sending it into the endpoint. This is for power and area savings to not resynchronize the reset again for each agent block on the chip.
agent_clk	Input	The IP block clock input. Required for asynchronous endpoints (ASYNCENDPT=1), otherwise it can be tied to 0.
agent_ clkreq	Input	These ports are mapped to sbi_sbe_clkreq and sbi_sbe_idle ports in Base Endpoint. For a functional description of these ports, see section 15.9, Design for Clock Gating.
agent_idle	Input	
side_clkreq	Output	Clock request signal output to the sideband fabric. Follows IOSF 1.0 Specifications for clock requests.
side_ism_fabric[2:0]	Input	Sideband fabric clock gating idle state machine (ISM) from the connected router port. This input is used directly by the idle state machine and does not go through a flop unless PIPEISMS=1.
side_ism_agent[2:0]	Output	Sideband endpoint clock gating idle state machine (ISM) that is output to the sideband fabric.
side_clkack	Input	Clock acknowledge from sideband fabric. Used with side_clkreq and fabric ISM state to determine when clock is valid. Follows IOSF 1.0 Specifications for clock requests.

Signal Name	Type	Description
side_ism_lock_b	Input	Assertion of this signal prevents agent ISM from transitioning out of the idle state. The signal must be driven synchronously to side_clk. For documentation about these signals, see the IOSF SBC Base Endpoint Integration Guide which is included in the 'doc' directory for this release.
sbe_clkreq	Output	These ports are mapped to sbe_sbi_clkreq and sbe_sbi_idle ports in Base Endpoint. For a functional description of these ports, see section 15.9, Design for Clock Gating.
sbe_idle	Output	
sbe_clk_valid	Output	Indicates the side clock is valid. Asserted if clkreq and clkack are both asserted or the fabric ISM state is not IDLE.
sbe_comp_exp	Output	For details about sbe_sbi_comp_exp, see the IOSF SBC Base Endpoint Integration Guide which is included in the 'doc' directory for this release.
sbe_parity_err_out	Output	Parity error indication
ext_parity_err_detected	Input	Allows external widgets to send a DO_SERR error message via the DOSERRMSTR module instantiated in the Endpoint and/or trigger a parity error condition in the endpoint
cgctrl_idlecnt[7:0]	Input	These ports are mapped to same named ports in Base Endpoint. For a functional description of these ports, review section 15.9, Design for Clock Gating, in the Integration Guide for IOSF SBC Base Endpoint Unit.
cgctrl_clkgaten	Input	
cgctrl_clkgatedef	Input	
usyncselect	Input	These three signals are used to enable and implement deterministic clock crossing in an asynchronous endpoint. If usyncselec is 1 (and the USYNC_ENABLE parameter is set to 1), then deterministic clock crossing is implemented. side_usync and agent_usync qualify the side_clk and agent_clk clocks, respectively. These signals must be asserted one cycle prior to the global synchronization of all clocks. For documentation about these signals, see the IOSF SBC Base Endpoint Integration Guide which is included in the 'doc' directory for this release.
side_usync	Input	
agent_usync	Input	
tx_ext_headers [NUM_TX_EXT_HEADERS:0]	Input	Extended headers to be used for endpoint-generated messages. Ignored if TX_EXT_HEADER_SUPPORT is set to zero.
ur_rx_sairs_valid	Output	A value of 1 indicates that the outputs ur_rx_sai and ur_rx_rs contain valid values. This output is not driven when UNIQUE_EXT_HEADERS is set to 0
ur_rx_sai[SAIWIDTH:0]	Output	When ur_rx_sairs_valid is a 1, this output reflects the SAI of the currently in progress non-posted message. This value will remain static until the start of a new non-posted message. This output is not driven when UNIQUE_EXT_HEADERS is set to 0
ur_rx_rs[RSWIDTH:0]	Output	When ur_rx_sairs_valid is a 1, this output reflects the RS of the currently in progress non-posted message. This value will remain static until the start of a new non-posted message. This output is not driven when UNIQUE_EXT_HEADERS is set to 0
ur_csairs_valid	Input	A value of 1 indicates that the inputs ur_csai and ur_crs contain valid values and the SAIRS extended header will be inserted into the generated message. This input is not used when UNIQUE_EXT_HEADERS is set to 0. This input must be held stable by the EOM of the ingressing, non-posted message as it will be used without flops.

Signal Name	Type	Description
ur_csai[SAIWIDTH:0]	Input	When ur_cairs_valid is a 1, this input will be inserted into the generated SAIRS extended header for the unsupported response completion message. This input is not used when UNIQUE_EXT_HEADERS is set to 0 This input must be held stable by the EOM of the ingressing, non-posted message as it will be used without flops.
ur_crs[RSWIDTH:0]	Input	When ur_csairs_valid is a 1, this input will be inserted into the generated SAIRS extended header for the unsupported response completion message. This input is not used when UNIQUE_EXT_HEADERS is set to 0 This input must be held stable by the EOM of the ingressing, non-posted message as it will be used without flops.
treg_csairs_valid	Input	The target register module valid indication signal from the agent IP. When a value of 1 is indicated, the SAIRS extended header will be inserted and populated with treg_csai and treg_crs values. A value of 0 will not generate a SAIRS extended header. This signal will be flopped when treg_trdy is asserted.
treg_csai[SAIWIDTH:0]	Input	The target register module SAI value used for the generated completion messages SAIRS extended header. If SAIWIDTH is less than 15, the difference will be zero padded in the most significant bits of the SAI field. This signal will be flopped when treg_trdy and treg_np is asserted.
treg_crs[RSWIDTH:0]	Input	The target register module RS value used for the generated completion messages SAIRS extended header. This signal will be flopped when treg_csairs_valid and treg_trdy has asserted. If RSWIDTH is less than 3, the difference will be zero padded in the most significant bits of the RS field. This signal will be flopped when treg_trdy and treg_np is asserted.
treg_rx_sairs_valid	Output	The target register module valid indication signal to the agent IP. When a value of 1 is indicated, the SAIRS extended header has been successfully captured.
treg_rx_sai[SAIWIDTH:0]	Output	The target register module SAI value received from the SAIRS extended header in the received packet.
treg_rx_rs[RSWIDTH:0]	Output	The target register module RS value received from the SAIRS extended header in the received packet.
mreg_sairs_valid	Input	The master register module valid indication signal from the agent IP. This signal MUST be held stable from the assertion of mreg_irdy until mreg_trdy is asserted. When this is a value of 1, the extended header field will be populated with mreg_sai and mreg_rs. A value of 0 will not generate a SAIRS extended headers. This input will be flopped when mmsg_[pc|np]irdy, mmsg_[pc|np]sel indicate the flit has been seen by the mmsg interface for the purpose of keeping track of what unique extended headers have to be sent. This signal should still be held until mreg_trdy is asserted as it will be used as a pass through signal.
mreg_sai[SAIWIDTH:0]	Input	The master register module SAI value used for the generated messages extended header. This signal must be held stable from the assertion of mreg_irdy until mreg_trdy is asserted. If SAIWIDTH is less than 15, the difference will be zero padded in the most significant bits of the SAI field. This signal is a pass through and must be held constant until mreg_trdy has asserted.

Signal Name	Type	Description
mreg_rs[RSWIDTH:0]	Input	The master register module RS value used for the generated messages extended header. This signal must be held stable from the assertion of mreg_irdy until mreg_trdy is asserted. If RSWIDTH is less than 3, the difference will be zero padded in the most significant bits of the RS field. This signal is a pass through and must be held constant until mreg_trdy has asserted.
visa_*	Output	VISA candidate signals. See section 15.7, VISA/DFx Features.
jta_clkgate_ovrd	Input	These ports are mapped to same named ports in Base Endpoint. For a functional description of these ports, see section 15.9, Design for Clock Gating, in the IOSF SBC Base Endpoint Integration Guide which is included in the 'doc' directory for this release.
jta_force_clkreq	Input	
jta_force_idle	Input	
jta_force_notidle	Input	
jta_force_creditreq	Input	
fscan_latchopen	Input	SCAN support signals need to test the latch-based queues. These signals are only used if the LATCHQUEUES parameter is set to 1.
fscan_latchclosed_b	Input	
fscan_clkungate	Input	Scan mode clock gate override. Set to 1 to enable clocks during scan shift mode, 0 for normal operation.
fscan_rstbypen	Input	Scan mode reset bypass enable. Set to 1 to bypass internally generated resets during scan testing. Set to 0 for normal operation.
fscan_byprst_b	Input	Scan mode reset. When fscan_rstbypen is set to 1, this input controls the internally generated resets.
fscan_mode fscan_clkungate_syn	Input	Unused signals, provided for pin compatibility with IOSF DFX requirements.
fscan_shiften	Input	Shift enable for scan chains. To be connected to scan logic in a post-scan inserted netlist.
visa_all_disable visa_customer_disable visa_ser_cfg_in[2:0]	Input	Unused signals, provided as placeholders for VISA insertion flow. Drive them using 'd0 if not used.
avisa_data_out[N:0] avisa_clk_out[N:0]	Output	Unused signals, provided as placeholders for VISA insertion flow.
sbe_visa_serial_rd_out	Output	This signal is intended to be driven by the serial_rd_data_out port of the top level VISA mux in the IP, enabling serial read of control registers values in the IP muxes. Not connecting it will disable this functionality, even if the IP enables it internally. If the IP is using VISA <4 muxes, then this port cannot be driven anyway, and can be left unused/disconnected
sbe_visa_bypass_cr_out	Output	This vector port is intended for driving the "bypass_cr_in" of any repeater the integrator may want to place right at the output of the top-level VISA mux of the IP. If your top level mux is VISA <2 (i.e., its module does not have a suffix of _4_s or _cro_s), then you cannot drive this port, and should leave it unused/disconnected.
fdfx_rst_b	Input	Reset signal for the VISA ULM. This signal is stitched to the reset input on the VISA ULM during VISA insertion.
fdfx_sbparity_def	Input	When set to 1, it disables the checking of parity errors. The parity bit continues to be generated and propagated as per normal operation when SB_PARITY_REQUIRED is set.

Table 11. Sideband Channel Target / Master Port Signals

Signal Name	Type	Description
tpayload[MAXPLDBIT:0]	Input	The target port message flit. **Note:** This input is used directly by the ingress queue and does not go through a flop unless PIPEINPS=1.
teom	Input	The target port end-of-message signal. **Note:** This input is used directly by the ingress queue and does not go through a flop unless PIPEINPS=1.
tpcput	Input	The target port posted/completion put signal. **Note:** This input is used directly by the ingress queue and does not go through a flop unless PIPEINPS=1.
tnpput	Input	The target port non-posted put signal. **Note:** This input is used directly by the ingress queue and does not go through a flop unless PIPEINPS=1.
tpccup	Output	The target port posted/completion credit update signal.
tnpcup	Output	The target port non-posted credit update signal
mpayload[MAXPLDBIT:0]	Output	The master port message packet.
meom	Output	The master port end-of-message signal.
mpcput	Output	The master port posted/completion put signal.
mnpput	Output	The master port non-posted put signal.
mpccup	Input	The master port posted/completion credit update signal. **Note:** This input is used directly by the egress queue and does not go through a flop unless PIPEINPS=1.
mnpcup	Input	The master port non-posted credit update signal. **Note:** This input is used directly by the egress queue and does not go through a flop unless PIPEINPS=1.
tparity	Input	Parity bit associated with the target payload data, tpayload **Note:** This input is used only when parameter SB_PARITY_REQUIRED=1.
mparity	Output	Parity bit associated with the master payload data, mpayload **Note:** This output should be used only when parameter SB_PARITY_REQUIRED=1.

Table 12. Target Interface Signals

Note: * A flit refers to a 4 byte transfer when MATCHED_INTERNAL_WIDTH=0 (flit_size=32), or to a transfer consisting of (MAXPLDBIT+1)/8 bytes, when MATCHED_INTERNAL_WIDTH=1 (flit_size=MAXPLDBIT+1).

Signal Name	Type	Description
tmsg_pcfree [MAXPCTRGT:0]	Input	When asserted this indicates that the IP block is ready to receive another flit* of posted/completion message payload from the endpoint on the target interface. This signal should be generated only from the internal state of the target agent within IP block (one bit per target agent). This input to the Base Endpoint should not be combinatorially generated from any outputs from the Base Endpoint.

Signal Name	Type	Description
tmsg_npfree [MAXNPTRGT:0]	Input	When asserted, this indicates that the IP block is ready to receive another flit* of non-posted message payload from the endpoint on the target interface. This signal should be generated only from the internal state of the target agent within IP block (one bit per target agent). This input to the Base Endpoint should not be combinatorially generated from any outputs from the Base Endpoint.
tmsg_npclaim [MAXNPTRGT:0]	Input	When an npclaim bit is asserted, a target has claimed the non-posted message. This means that the target that claims the message is responsible for generating the appropriate completion for the non-posted message. When CLAIM_DELAY=0, the npclaim signal can be asserted (at the earliest) in the same cycle as the first npput, and (at the latest) in the same cycle as the last npput of the non-posted message in order to successfully claim the message as a target. On the other hand, when CLAIM_DELAY >= 1, the window when the endpoint samples the tmsg_npclaim from the agent is extended past the end-of-message flit until tmsg_npfree is asserted for CLAIM_DELAY number of cycles. **Note:** A target should never assert the npclaim signal unless it is absolutely sure that it is generating the completion for the non-posted message. There is no mechanism for a target to relinquish a claim once it has been asserted for a given message. After a target asserts the npclaim signal, then all of the npclaim signals from all of the targets are "don't care" in all subsequent cycles up to the end-of-message transfer (npput/npeom both asserted) for the message that is in-progress.
tmsg_pcput	Output	When asserted, this indicates a valid flit* of a posted/completion message from the Base Endpoint to the IP block on the target interface. The pcput signal can only be asserted in response to all pcfree signals asserted and the internal pcirdy signal asserted from the ingress port.
tmsg_npput	Output	When asserted this indicates a valid flit* of a non-posted message from the Base Endpoint to the IP block on the target interface. The npput signal can only be asserted in response to all npfree signals asserted and the internal npirdy signal asserted and internal npfence signal deasserted from the ingress port.
tmsg_pcvalid tmsg_npvalid	Output	In sync mode, When asserted, indicates the respective tmsg_*payload is valid. This is provided to allow attached IP to begin processing message before asserting free. But in async mode free should be asserted first. **Note:** Free/put protocol must still be observed.
tmsg_npmsgip tmsg_pcmsgip	Output	These signals assert after the first dword transfer of a message on the target interface, if the message is longer than one dword. This assertion occurs the cycle after the tmsg_*put for the last transfer of the first dword is asserted, and tmsg_*eom is deasserted. The tmsg_*msgip signal deasserts the cycle after the last dword transfer of a message on the target interface. These signals can be used by the targets to differentiate between the first dword of the message (including opcode) and the rest of the message.
tmsg_pceom	Output	End of message indicator. When asserted, this indicates the last flit* of a posted/completion message on the target interface, and is only valid when tmsg_pcput is asserted.
tmsg_npeom	Output	End of message indicator. When asserted this indicates the last flit* of a non-posted message on the target interface, and is only valid when tmsg_npput is asserted.
tmsg_pcpayload [flit_size-1:0]*	Output	Posted/completion message payload; the next flit of a posted/completion message on the target interface, which is only valid when tmsg_pcput is asserted.
tmsg_nppayload [flit_size-1:0]*	Output	Non-posted message payload; the next flit of a non-posted message on the target interface, which is only valid when tmsg_npput is asserted.

Signal Name	Type	Description
tmsg_pccmpl	Output	This signal is asserted when a completion opcode (0x20 or 0x21) is decoded on the flit that has the opcode on tmsg_pcpayload (e.g., when MATCHED_INTERNAL_WIDTH=0, the first flit has the opcode on bits [23:16]; on the other hand, when MATCHED_INTERNAL_WIDTH=1, the opcode flit could be the 1st, 2nd, or 3rd flit, when MAXPLDBIT is, respectively, 31, 15, and 7). This signal is valid only when tmsg_pcput is asserted. **Note:** The non-posted message master agents in the IP block that are waiting for completions as a target can use this signal instead of duplicating the same opcode decode logic.
tmsg_pcparity	Output	Parity output to agent, calculated across tmsg_pcpayload and tmsg_pceom.
tmsg_npparity	Output	Parity output to agent, calculated across tmsg_nppayload and tmsg_npeom.

Note: All `tmsg_*` ports are mapped to `sbi_sbe_tmsg_*/sbe_sbi_tmsg_*` ports in Base Endpoint.

Table 13. Target Register Access Interface Signals

Signal Name	Type	Description
treg_trdy	Input	When asserted this indicates that the IP block is ready to receive, consume and complete another register access message. If the IP block cannot complete a register access message in a single cycle, then it deasserts trdy when waiting for the next message to be received. It then asserts trdy in response to an irdy assertion, once it is ready to complete the message on the target register access interface.
treg_cerr	Input	This signal is valid when trdy is asserted and indicates whether or not the register access message was successfully completed in the IP block.
treg_rdata [MAXTRGTDATA:0]	Input	This is the read data provided by the IP block to the endpoint for register access read messages. The width of the data is either a dword or a qword.
treg_irdy	Output	When asserted, this signal indicates that the endpoint has a register access message available for the IP block.
treg_np	Output	When asserted the message on the target register access interface is non-posted, otherwise it is posted. This signal is only valid when treg_irdy is asserted.
treg_dest[7:0]	Output	Destination port ID of the message.
treg_source[7:0]	Output	Source port ID of the message.
treg_opcode[7:0]	Output	Opcode of the message.
treg_addrlen	Output	Address length of the message (0 = 16-bit, 1 = 48-bit).
treg_bar[2:0]	Output	The BAR number which is applicable only for memory mapped or I/O mapped space.
treg_tag[2:0]	Output	Request identifier.
treg_be[X:0] (X=3 or 7)	Output	Byte enables: Bits 3:0 are the first dword byte enables and (if present) bits 7:4 are the second dword byte enables. The width of this signal is dependent upon the MAXTRGTDATA parameter.
treg_fid[7:0]	Output	Routing ID, which is applicable only for memory mapped, I/O mapped and config space.
treg_addr [MAXTRGTADDR:0]	Output	Address of the register access message.
treg_wdata [MAXTRGTDATA:0]	Output	The write data that is provided by the endpoint to the IP block.
treg_eh	Output	The “EH” bit from the standard IOSF 1.0 header

Signal Name	Type	Description
treg_eh_discard	Output	Asserted if an extended header not listed in RX_EXT_HEADER_IDS is received by the register access block.
treg_ext_header [NUM_RX_EXT_ HEADERS:0]	Output	All received extended headers
treg_hier_srcid	Output	These are part of the transparent bridge PCR. The first DW in the payload in an endpoint with GLOBAL_EP set to 1, contains the hierarchical source and destination port ID's. This is sent to the target register block on these signals. When GLOBAL_EP is 0, these are driven to 0.
treg_hier_destid	Output	

Table 14. Master Interface Signals

Note: * A flit refers to a 4 byte transfer when MATCHED_INTERNAL_WIDTH=0 (flit_size=32), or to a transfer consisting of (MAXPLDBIT+1)/8 bytes, when MATCHED_INTERNAL_WIDTH=1 (flit_size=MAXPLDBIT+1).

Signal Name	Type	Description
mmsg_pceom [MAXPCMSTR:0]	Input	End of message indicator, one bit per master. When asserted this indicates the last flit* of a posted/completion message on the master interface, and is only valid when mmsg_pcirdy is asserted.
mmsg_npeom [MAXNPMSTR:0]	Input	End of message indicator, one bit per master. When asserted this indicates the last flit* of a non-posted message on the master interface, and is only valid when mmsg_npirdy is asserted.
mmsg_pcpayload [flit_size*MAXPCMSTR+ (flit_size-1):0]*	Input	Posted/completion message payload, flit_size* bits per master; the next flit of a posted/completion message on the master interface, which is only valid when mmsg_pcirdy is asserted.
mmsg_nppayload [flit_size*MAXNPMSTR+ (flit_size-1):0]	Input	Non-posted message payload, flit_size* bits per master; the next flit of a non-posted message on the master interface, which is only valid when mmsg_npirdy is asserted.
mmsg_pcirdy [MAXPCMSTR:0]	Input	When asserted this indicates that the IP block is ready to deliver the next flit* of a posted/completion message to the endpoint on the master interface. The transfer occurs when mmsg_pcirdy and mmsg_pctrdy are both asserted, and the corresponding mmsg_pcsel signal is asserted. This signal must deassert the cycle after the last flit* of the lasy dword transfer of a message, unless the master has another message to send. Also, once this signal is asserted, the signal must remain asserted until mmsg_pctrdy is asserted.
mmsg_npirdy [MAXNPMSTR:0]	Input	When asserted this indicates that the IP block is ready to deliver the next flit* of a non-posted message to the endpoint on the master interface. The transfer occurs when mmsg_npirdy and mmsg_nptrdy are both asserted, and the corresponding mmsg_npsel signal is asserted. This signal must deassert the cycle after the last flit* of the last dword transfer of a message, unless the master has another message to send. Also, once this signal is asserted, the signal must remain asserted until mmsg_nptrdy is asserted.
mmsg_pctrdy	Output	When asserted this indicates that the endpoint is ready to receive another flit* of posted/completion message data from the IP block on the master interface.
mmsg_nptrdy	Output	When asserted this indicates that the endpoint is ready to receive another flit* of non-posted posted message data from the IP block on the master interface.

Signal Name	Type	Description
mmsg_pcmsgip	Output	This signal asserts after the first dword transfer of a posted/completion message on the master interface, if the message is longer than one dword. This assertion occurs the cycle after mmsg_pcsel[x], mmsg_pcirdy[x] and mmsg_pctrdy are all asserted, and mmsg_pceom is deasserted. The mmsg_pcmsgip signal deasserts the cycle after the last dword transfer of a posted/completion message on the master interface. This deassertion occurs the cycle after mmsg_pcsel[x], mmsg_pcirdy[x], mmsg_pctrdy and mmsg_pceom are all asserted.
mmsg_npmsgip	Output	This signal asserts after the first dword transfer of a non-posted message on the master interface, if the message is longer than one dword. This assertion occurs the cycle after mmsg_npsel[x], mmsg_npirdy[x] and mmsg_nptrdy are all asserted and mmsg_npeom is deasserted. The mmsg_npmsgip signal deasserts the cycle after the last dword transfer of a non-posted message on the master interface. This deassertion occurs the cycle after mmsg_npsel[x], mmsg_npirdy[x], mmsg_nptrdy and mmsg_npeom are all asserted.
mmsg_pcsel [MAXPCMSTR:0]	Output	This is a one-hot encoded vector that indicates which posted/completion master is selected by the master interface arbiter. Each bit of the vector is used by an IP block posted/completion master to qualify the mmsg_pcmsgip and the mmsg_pctrdy signals.
mmsg_npsel [MAXNPMSTR:0]	Output	This is a one-hot encoded vector that indicates which non-posted master is selected by the master interface arbiter. Each bit of the vector is used by an IP block non-posted master to qualify the mmsg_npmsgip and the mmsg_nptrdy signals.
mmsg_pcparity	Input	Parity input from agent, calculated across mmsg_pcpayload and mmsg_pceom.
mmsg_npparity	Input	Parity input from agent, calculated across mmsg_nppayload and mmsg_npeom.

Note: All `mmsg_*` ports are mapped to `sbi_sbe_mmsg_*/sbe_sbi_mmsg_*` ports in Base Endpoint.

Table 15. Master Register Access Interface Signals

Signal Name	Type	Description
mreg_irdy	Input	When asserted this indicates that the IP block is ready to deliver another register access message (posted or non-posted).
mreg_npwrite	Input	Indicates if the register access write message is non-posted (npwrite=1) or posted (npwrite=0). This signal is only used when the opcode selects a register access write. Register access reads are always sent as a non-posted message.
mreg_dest[7:0]	Input	Destination port ID of the message.
mreg_source[7:0]	Input	Source port ID of the message.
mreg_opcode[7:0]	Input	Opcode of the message.
mreg_addrlen	Input	Address length of the message (0 = 16-bit, 1 = 48-bit).
mreg_bar[2:0]	Input	The BAR number which is applicable only for memory mapped or I/O mapped space.
mreg_tag[2:0]	Input	Request identifier, which is only used when the master is initiating a completion message.

Signal Name	Type	Description
mreg_be[X:0] (X=3 or 7)	Input	Byte enables: Bits 3:0 are the first dword byte enables and (if present) bits 7:4 are the second dword byte enables. The width of this signal is dependent upon the MAXMSTRDATA parameter.
mreg_fid[7:0]	Input	Routing ID, which is applicable only for memory mapped, I/O mapped and config space.
mreg_addr [MAXMSTRADDR:0]	Input	Address of the register access, which depends upon the address space defined by the opcode.
mreg_wdata [MAXMSTRDATA:0]	Input	The write data that is provided by the IP block to the endpoint.
mreg_hier_srcid	Input	When the endpoint is part of the global network supporting transparent bridge (GLOBAL_EP is set to 1), these inputs need to be connected to the hierarchical source and destination ID's that will eventually appear as the first DW in the payload from the endpoint to the fabric. When GLOBAL_EP is 0, these inputs can be ignored.
mreg_hier_destid	Input	
mreg_trdy	Output	When asserted this indicates that the endpoint is ready to receive another 4 byte transfer of the message presented on the master register access interface.
mreg_pmsgip	Output	When asserted there is a posted message currently in-progress on the master register access interface. This means that the endpoint has starting sending the posted register access message thru the egress port.
mreg_nmsgip	Output	When asserted there is a non-posted message currently in-progress on the master register access interface.

12.2.2 Block Interface Timing Waveforms

Credits are not shown for the diagrams in this section. It is assumed that there are no credit stalls for any transactions. It is also assumed that the agent does not support extended headers and that no extended headers are sent or received.

12.2.2.1 Master Register Access Interface—Posted Write

The timing diagrams in this section, show a posted register access write message initiated on the master register access interface. When MATCHED_INTERNAL_WITDH=0 (Figure 5), the width of mmsg_pcpayload is always 32 bit, regardless of the setting of MAXPLDBIT, which specifies the width of mpayload (assumed to be 8 bits in Figure 5, with MAXPLDBIT=7). On the other hand, when MATCHED_INTERNAL_WIDTH=1, the width of pcpayload matches the width of mpayload, i.e.: 1) 32 bits in Figure 6, with MAXPLDBIT=31, 2) 16 bits in Figure 7, with MAXPLDBIT=15, and 3) 8 bits in Figure 8, with MAXPLDBIT=7. The register access write uses a 16-bit address length and a single dword of write data. This type of register access message requires a 12-byte message on the sideband channel.

At the same time that this transfer is occurring on the sideband channel, the master register access interface remains unchanged with the irdy asserted from the IP block, while the master register access service block (SBEMSTRREG) and the Base Endpoint (SBEBASE) sequence through the necessary dwords of the messages presented on the interface.

Note: When MATCHED_INTERNAL_WIDTH=0, the function of the SBEMSTRREG block is to create 4 byte message flits, which are sent to the egress block using the master message interface protocol. On the other hand, when MATCHED_INTERNAL_WIDTH=1, the SBEMSTRREG block creates MAXPLDBIT+1 byte message flits.

Note: The effect of employing the MATCHED_INTERNAL_WIDTH parameter is illustrated only in this section, as the behavior is similar in all other scenarios.

Figure 5. Timing Diagram—Master Register Access Interface, Posted Write (MATCHED_INTERNAL_WIDTH=0)

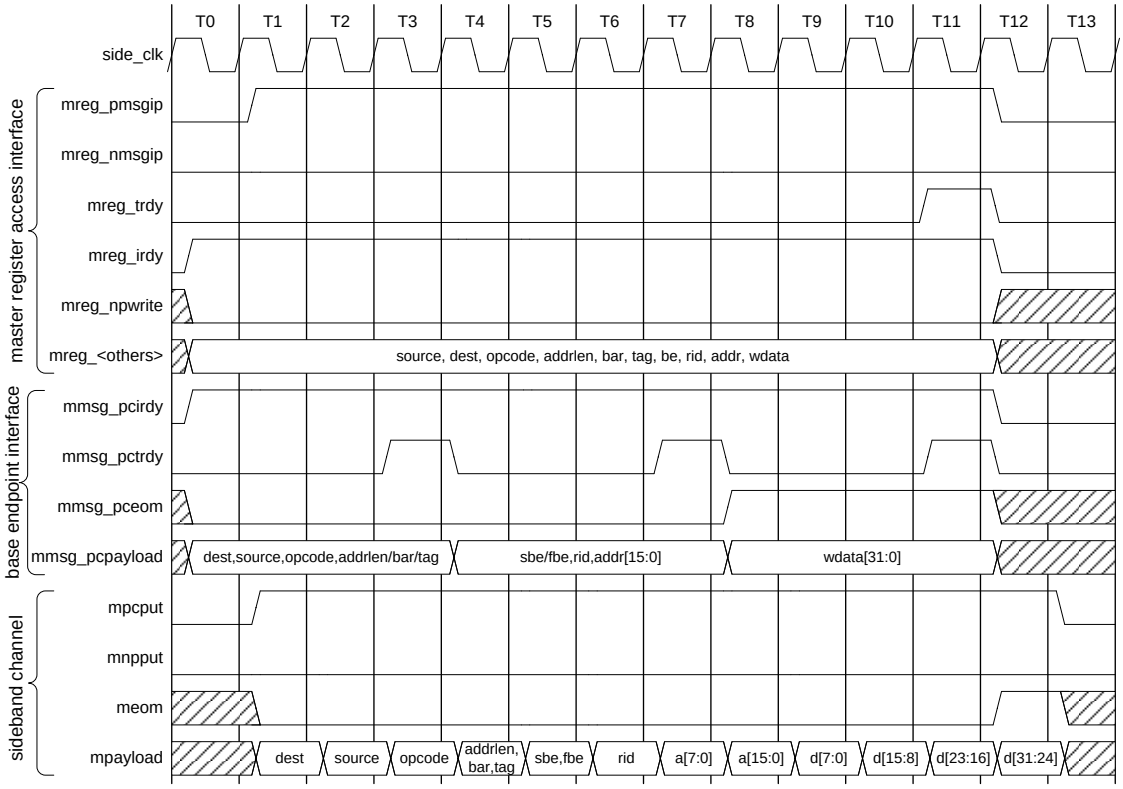


Figure 6. Timing Diagram—Master Register Access Interface, Posted Write (MATCHED_INTERNAL_WIDTH=1, and MAXPLDBIT=7)

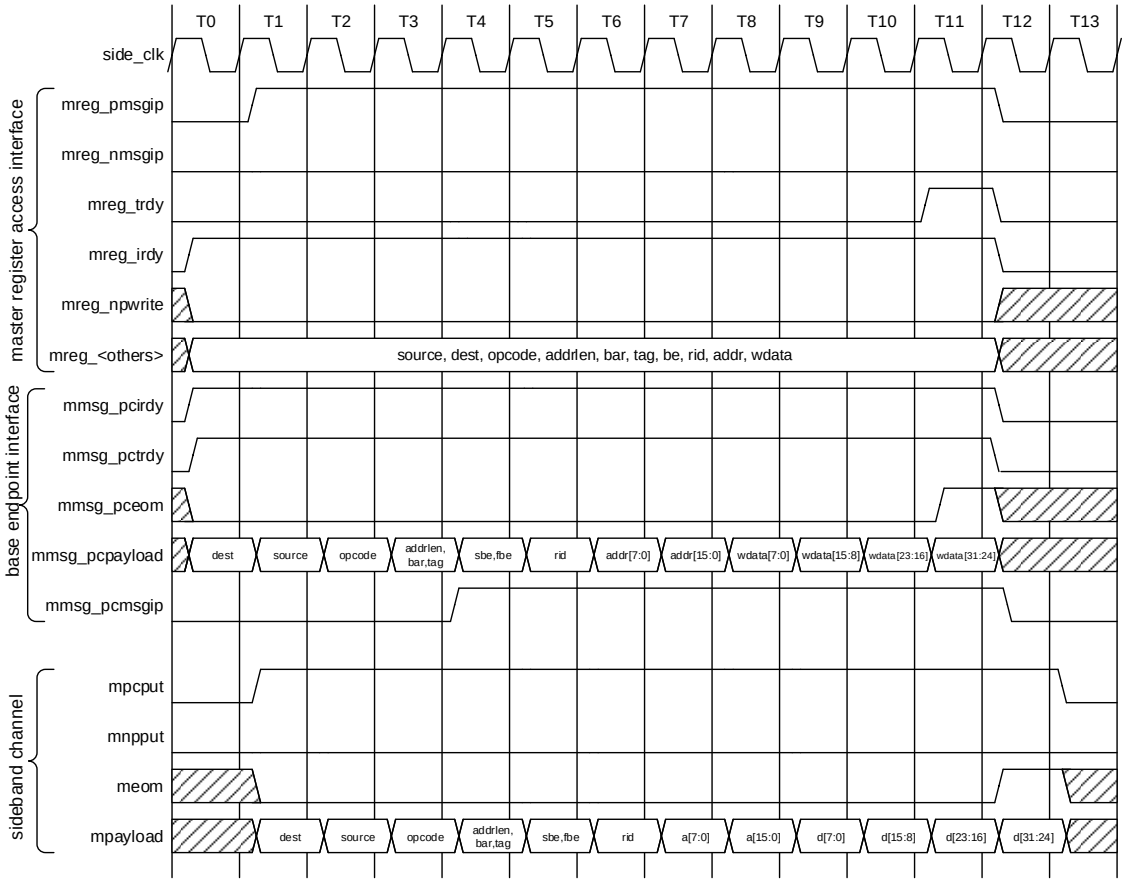


Figure 7. Timing Diagram—Master Register Access Interface, Posted Write (MATCHED_INTERNAL_WIDTH=1, and MAXPLDBIT=15)

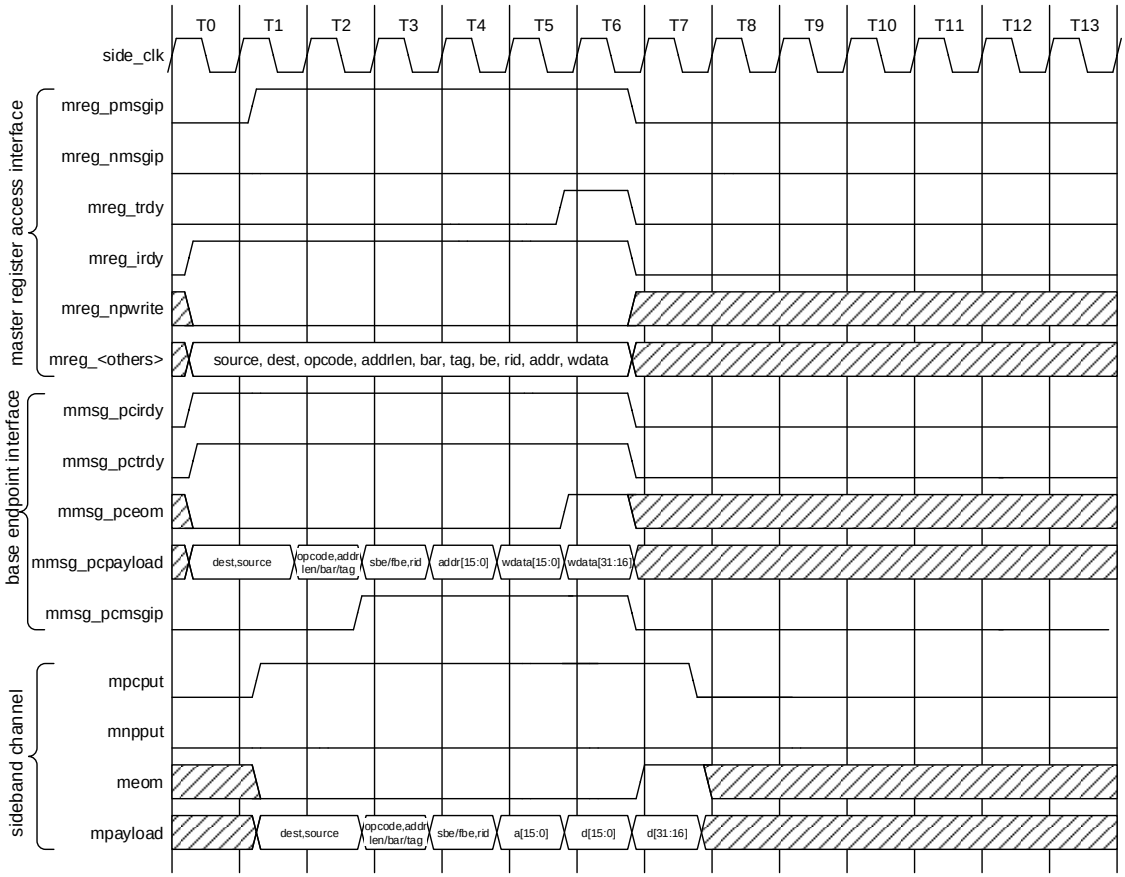
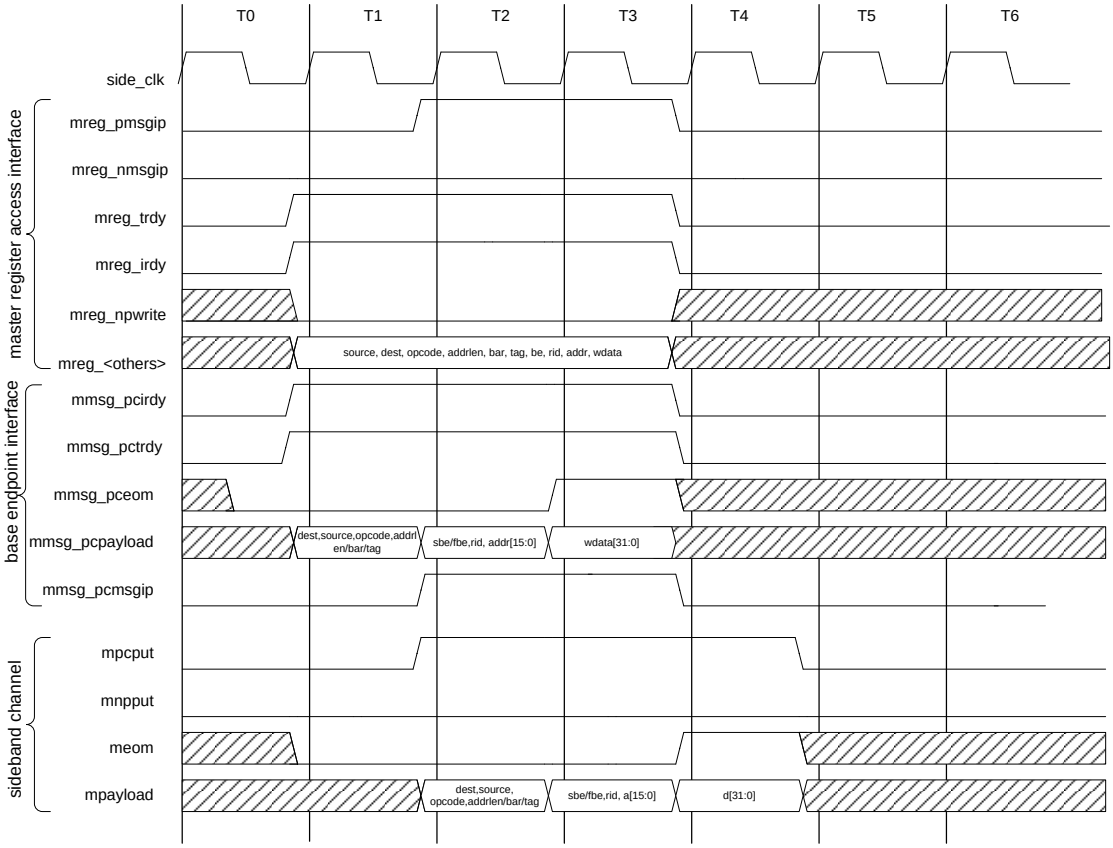


Figure 8. Timing Diagram—Master Register Access Interface, Posted Write (MATCHED_INTERNAL_WIDTH=1, and MAXPLDBIT=31)



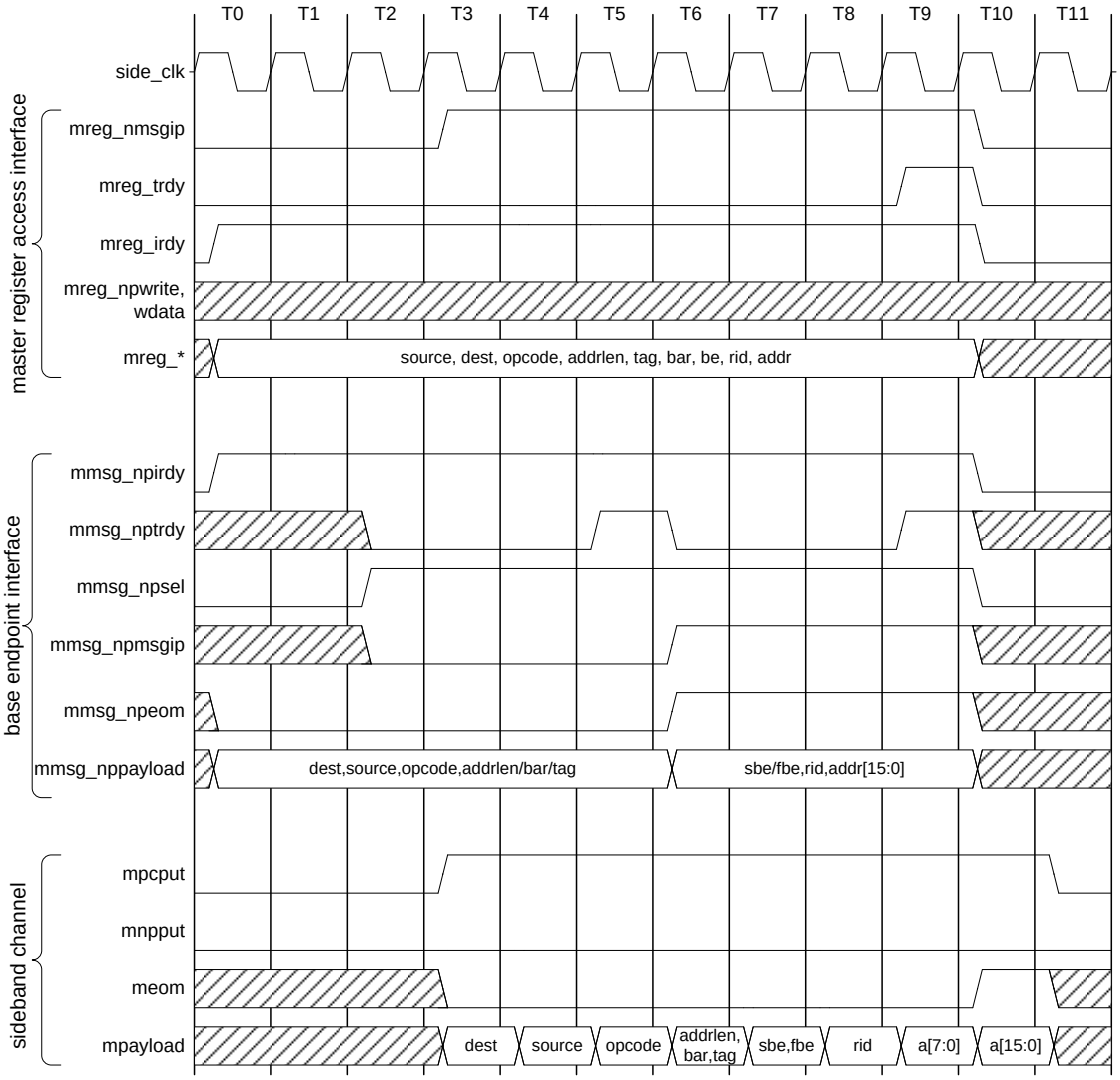
12.2.2.2 Master Register Access Interface – Non-posted Read

This timing diagram shows a non-posted register access read message initiated on the master register access interface. The register access write uses a 16-bit address length. This type of register access message requires an 8-byte message on the sideband channel.

In this diagram, the register access message is stalled by 2 cycles until the internal Base Endpoint arbiter selects the register access non-posted master. Until the npsel signal is asserted, this block ignores the nptrdy and npmsgip signals on the internal master interface.

Once the npsel signal is asserted (in T2), the mreg_nmsgip signal is asserted to the IP block indicating that the arbiter has selected the message to be transferred and might have started to send the message on the sideband interface.

Figure 9. Timing Diagram—Master Register Access Interface, Non-Posted Read



12.2.2.3 Target Register Access Interface—Read with Successful Completion

This timing diagram illustrates a non-posted register access read (CfgRd) and the completion for the read. The CfdRd message length is 8 bytes in this example because the chosen address length is 16 bits.

In cycles T4 and T8, a 4 byte flit of the message is transferred from the Base Endpoint (SBEBASE) to the SBETRGTREG block, capturing the flits in flops which are subsequently presented to the target register access interface (as long as the interface is idle; not driving a valid posted register access message).

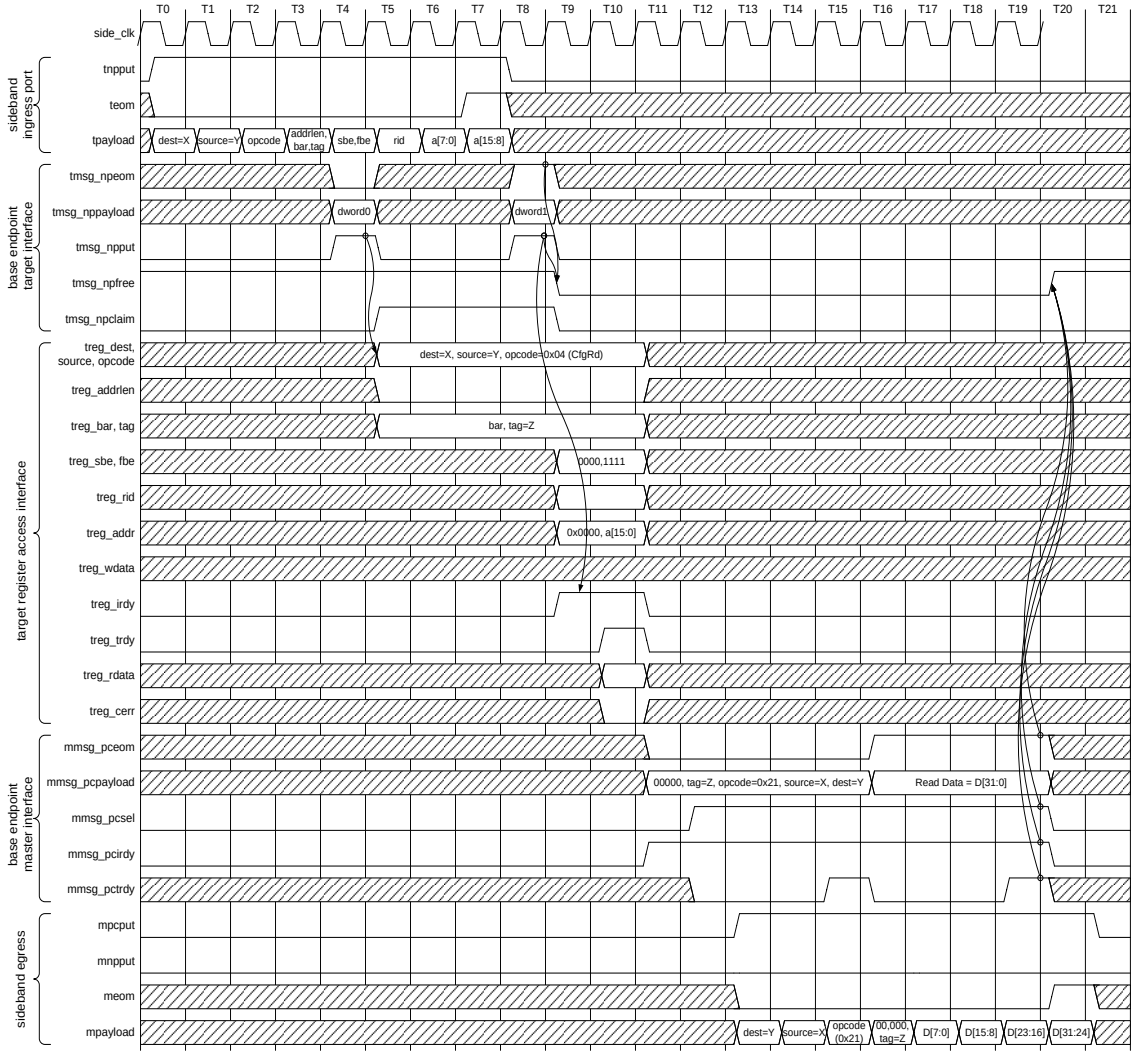
In T9, the entire message has been flopped and the irdy signal (treg_irdy) is asserted. In this example, the IP block completes the read 1 cycle later by asserting the trdy signal (treg_trdy), driving the appropriate read data on treg_rdata and the appropriate value on the completion error signal (treg_cerr).

In this example, the read completes without and error (treg_cerr=0 in T10), so a completion with data is generated (opcode = 0x21). The completion message in this example is 8 bytes, because the second byte enables were all zero.

Note: The SBETRGTREG block cannot accept another non-posted message until the completion finishes at the egress port.

Note: The IP block is really only involved during cycles T9 and T10 (treg_irdy/trdy cycles), everything else is handled by the sideband endpoint.

Figure 10. Timing Diagram—Target Register Access Interface, Read with Successful Completion



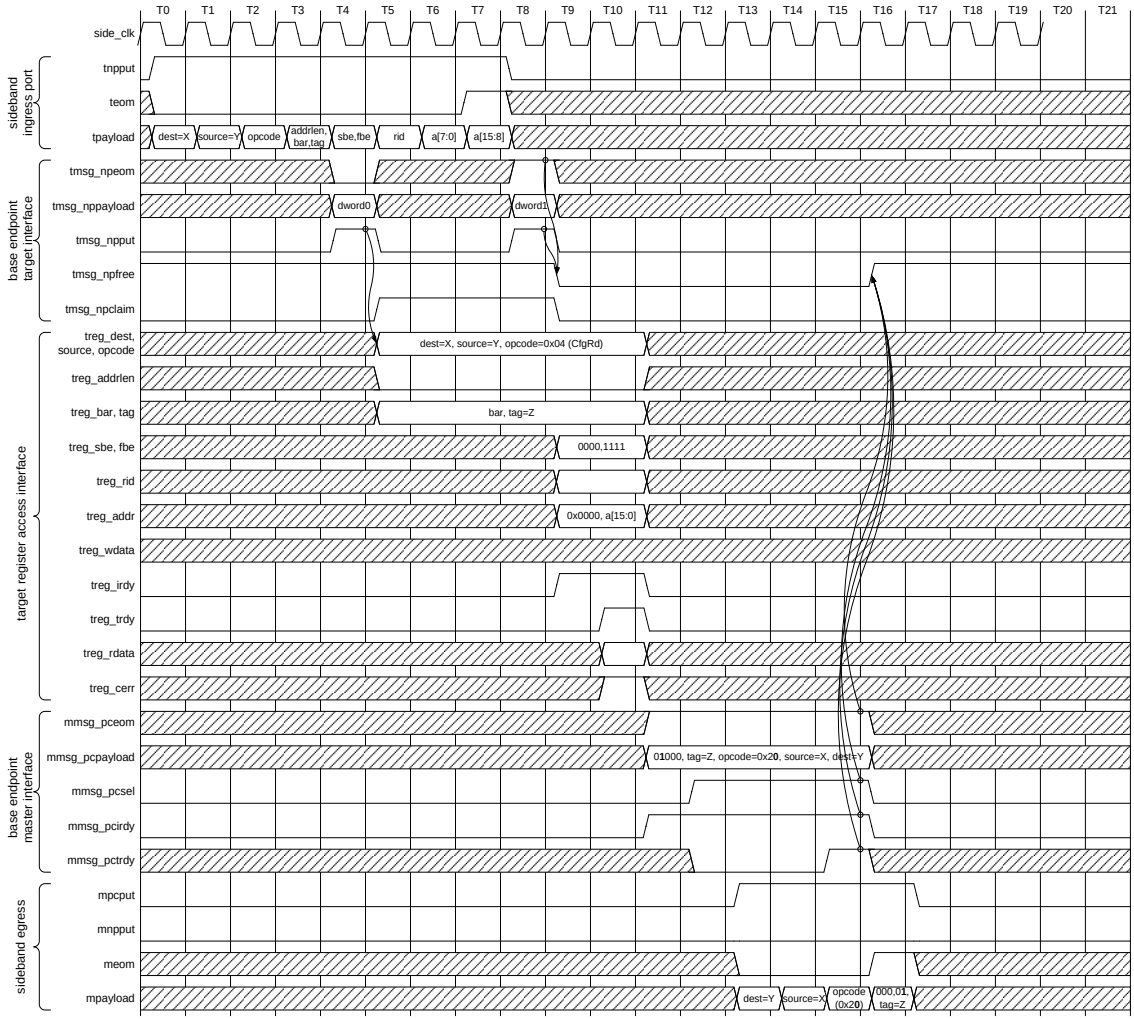
12.2.2.4 Target Register Access Interface—Read with Unsuccessful Completion

This timing diagram is very similar to the previous timing diagram, except in this case the IP block indicates an error has occurred (treg_cerr asserted in cycle T10).

In this case, the read data input (treg_rdata) to the endpoint is not used and the completion message sent on the sideband (egress) channel changes from a 8 byte completion with data to a 4 byte completion without data.

Note: The completion response field change from 00 (successful completion) to 01 (unsuccessful / not supported completion)

Figure 11. Timing Diagram—Target Register Access Interface, Read with Error Completion



12.2.2.5 Target Register Access Interface – Write (Posted)

This timing diagram illustrates a posted register access write that is received on the sideband (ingress) channel and forwarded to the master register access interface.

This is an example of the largest possible register access message, which is 20 bytes.

Note: This requires 5 transfers from the Base Endpoint port (SBEBASE) to the SBETRGTREG block.

The final (5th) dword transfer occurs in T20 (tmsg_pceom, pcirdy and pctrdy all asserted). As each dword is transferred from the Base Endpoint to the register access target block, the corresponding flops capture the message and then drive the target register access interface to the IP block.

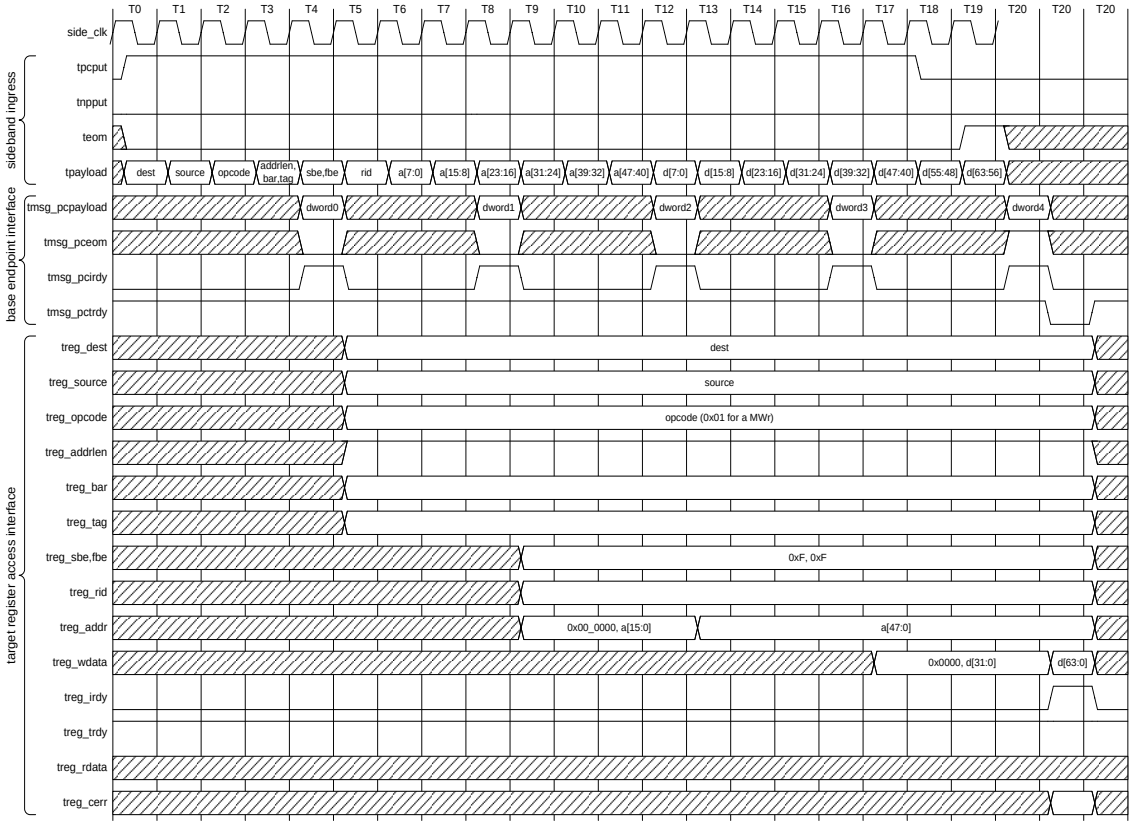
The tmsg_pctrdy signal remains asserted until the end of the message is received. It then deasserts (T20) and remains deasserted until the message is completed on the register access interface. In this example, this only takes 1 cycle.

In this example, the IP block design is always able to accept register access messages and has strapped the treg_trdy signal active. This would require that both read and writes would have to be able to be completed in a single cycle.

The actual IP block design could have an arbitrarily long irdy to trdy delay required to complete the register access message.

Note: The treg_cerr signal is always valid (either 0 for successful or 1 for error) when the trdy signal is asserted. If the message is posted, then the cerr signal is not used by the endpoint.

Figure 12. Timing Diagram—Target Register Access Interface, Write (Posted)



13 Clock and Reset

For reset details, see the IOSF SBC Base Endpoint Integration Guide (included in the 'doc' directory for this release).

14 Area Estimates

The data shown is for an endpoint configured with an IOSF payload width of 8 bits. Differing IOSF payload widths yield different results.

Table 16. Parameter Values Associated with Area Estimates

Parameters Used	Default Values	Parameters Used	Default Values
MAXPLDBIT	7	DISABLE_COMPLETION_FENCING	0
QUEUEDEPTH	4	TARGETREG	1
LATCHQUEUES	0	MASTERREG	1
MAXPCTRGT	0	MAXTRGTADDR	31
MAXNPTRGT	0	MAXTRGTDATA	63
MAXPCMSTR	0	MAXMSTRADDR	31
MAXNPTRGT	0	MAXMSTRDATA	63
ASYNCENDPT	0	RX_EXT_HEADER_IDS	‘0
RX_EXT_HEADER_SUPPORT	0	N/A	N/A
TX_EXT_HEADER_SUPPORT	0	N/A	N/A

Disclaimer

While these numbers can provide the reader, a rough estimate for the area required by an IOSF Sideband Endpoint, they remain estimates and should be treated as such. The area numbers have not been validated for the latest RTL, or the latest tool versions.

Conditions

P1271 (b12_wn_p1271_1x1r2_tttt_0.7v_70.00c_core), 250MHz side_clk, 35% period I/O delay, *full scan insertion*

Estimated “NAND2 equivalent” area: ~10092 gates

Estimated Total Cell Count = ~3052

Notes

- The minimum value for MAXTRGTADDR is 15. Each increment adds two flops, or ~20 gates.
- Using a MAXTRGTDATA value of 31 saves 64 flops, or ~750 gates.
- Altering MAXMSTR* parameters have negligible effect on design area.
- A ‘minimum’ configuration with MAXTRGTADDR=15 and MAXTRGTDATA=31 yields a design of approximately 8000 NAND2 gates.

15 Integration Details

Most integration details are covered in the IOSF SBC Base Endpoint Integration Guide, which is included in the 'doc' directory for this release. The reader should consider this integration guide as an addendum to the base document, filling in details that are unique to the 'full' Endpoint design (for example, the sbbase plus sbemstrreg and sbetrgtreg).

15.1 Sideband Fabric Inputs and Pipeline Stages

The IOSF 1.0 endpoint release adds an optional feature for inserting pipeline stages at both ISM fabric inputs and other sideband fabric inputs. This optional feature is added to improve timing closure in cases where suboptimal wire delays are present between endpoint and router.

This feature is achieved through the addition of two new parameters: PIPEISMS for fabric ISM inputs and PIPEINPS for non-ISM inputs (tpayload, t*put, teom, and m*cup). Setting 'd1 to these parameters adds pipeline stage to the associated inputs and adds an extra cycle of latency to target and master transactions.

It may be necessary to increase the NP/PC Ingress Queue depths to avoid stalling the bus between transactions.

Note: The pipeline stage can be added to the fabric ISM inputs without adding pipeline to non-ISM inputs but the opposite is not true.

15.2 Ingress Latency

The ingress latency for message interface transactions is unchanged from the Base Endpoint. For a detailed description, see the IOSF SBC Base Endpoint Integration Guide which is included in the 'doc' directory for this release.

The target register access interface adds one clock of latency (side_clk if no clock crossing, agent_clk if using clock crossing) compared to transactions handled directly by the message interface.

15.3 QUEUEDEPTH Parameter Setting and Bandwidth

The queue depth requirement to enable streaming is unchanged from the Base Endpoint. For a detailed description, see the IOSF Sideband Base Endpoint Integration Guide which is included in the 'doc' directory for this release.

15.4 MAXTRGTADDR Parameter

The IOSF specification allows for two addressing schemes for register IO transactions: 16 or 48 bit. However, there are many cases where the agent needs more than a 16b address but does not require all 48b. In these cases, it is possible to instantiate the endpoint with MAXTRGTADDR set to a different number, such as 31.

This creates a 32b address output from the target register block, and, importantly, only infers a 32b flop bank to store the address. The fabric still needs to send an IOSF standard 48b transaction to this address space, however, the endpoint verifies that the unused upper bits are all zero before discarding them.

If any of the upper bits in a transaction are non-zero, the transaction is dropped by the TREG widget. An UR is returned from SBEBASE, if it is a non-posted payload for the errorneous transaction. The endpoint handles both scenarios without any involvement by the agent.

15.5 Parity Handling

Agents not using the Parity feature set the SB_PARITY_REQUIRED and DO_SERR_MASTER parameter to 0.

Agent using the Parity feature:

1. Expose SB_PARITY_REQUIRED (set it to 1), fdfx_sbparity_def and ext_parity_err_detected as per descriptions.
2. Expose the new interface pins – tparity and mparity on the fabric side of EP.
3. IPs as master should drive the master parity interface (mmsg_pc/np_parity).
4. IPs as regular TMSG target consume tmsg_pc/np_parity.
5. IPs which have to send an error message, set DO_SERR parameter and connect the strap pins.
6. IPs will have to provide a sticky storage for the sbe_parity_err_out output

Checking of parity can be disabled by setting the fdfx_sbparity_def pin, which must eventually be connected at the SOC top level. Once parity error is detected and sbe_parity_err_out is sent out of EP, asserting the defeature pin will not revert the checks or de-assert sbe_parity_err_out. In other words, once the EP steps into the error state, asserting defeature will not take any effect.

Agents can trigger the sending the DO_SERR message and push the EP to the error state (Parity Handling Rules Section 3.4.3 in IOSF spec1.2) through ext_parity_err_detected. This input to EP from the agent should be an "or" of all parity checks internal to the agent and must be driven off a flop in agent_clk. This is because once the last put happens on the regular target TMSG – target register interface, the register access widget will immediately send out the payload. Ext_parity_err_detected must be a level signal and cannot toggle once set. Any payload received before this pin is set, will be sent out of EP. This input also allows external widgets to generate DO_SERR message (when DO_SERR_MASTER parameter is set) and is internally connected to the parity error detected within the EP.

When IPs detect a parity error, it is their responsibility to assert the treg_tmsg_parity_err_det to the base endpoint. Treg_tmsg_parity_err_det is a level signal and must remain asserted once set. The status of the tmsg_*free signals do not matter after treg_tmsg_parity_err_det is set, as any further NP and PC puts into your logic will be masked and dropped by the base endpoint.

Once IP's detect a parity error, it must ensure containment of the parity error (i.e a current message in progress must not continue further into the agent).

The endpoint takes care of the rest of the IOSF spec requirements so long as these rules are followed.

15.5.1 IPs Integrating EP's Target Register Widget

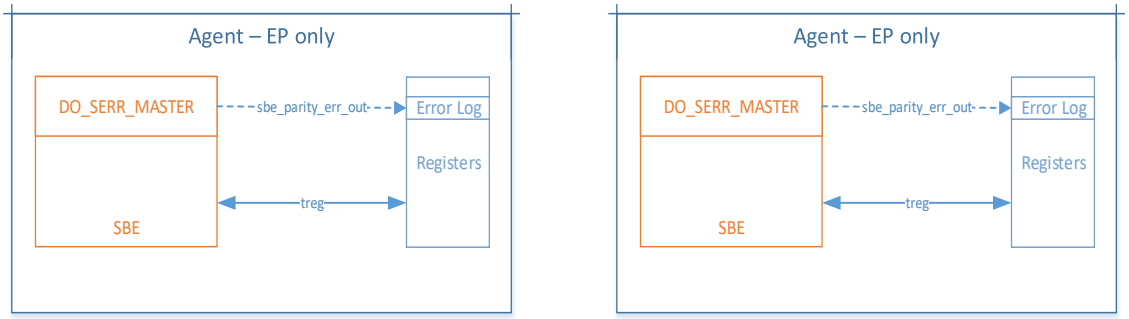
The register widget provided by EP, consumes the tmsg_pc/np_parity outputs from the EP's target block and checks for parity error before presenting on the widget output. The IPs will

need to implement a sticky storage for the sbe_parity_err_out (final parity error output) from EP. Once sbe_parity_err_out is set, it can only be reset by a warm reset.

15.5.2 IPs with Custom TMSG Block

IPs using their custom target widgets, will need to implement the checking by themselves. The parity pins, together with payload and eom will be available on the regular target interface to IP. They will have to set the appropriate parameters for parity in EP and wire the custom widgets parity error out pin into EP's ext_parity_err_detected (following the conditions for ext_parity_err_detected). This will trigger the error packet to be sent on the fabric output (shown as send_do_serr) and also asserts the sbe_parity_err_out (shown as parity_err). The custom widget will have to take in the fdfx_sbparity_def to disable parity checking within IP's widget. IPs will need to implement the sticky storage for sbe_parity_err_out.

Figure 13. IP Using EP's Target Widget and IP with Custom TMSG



15.5.3 IP with Custom Master to EP

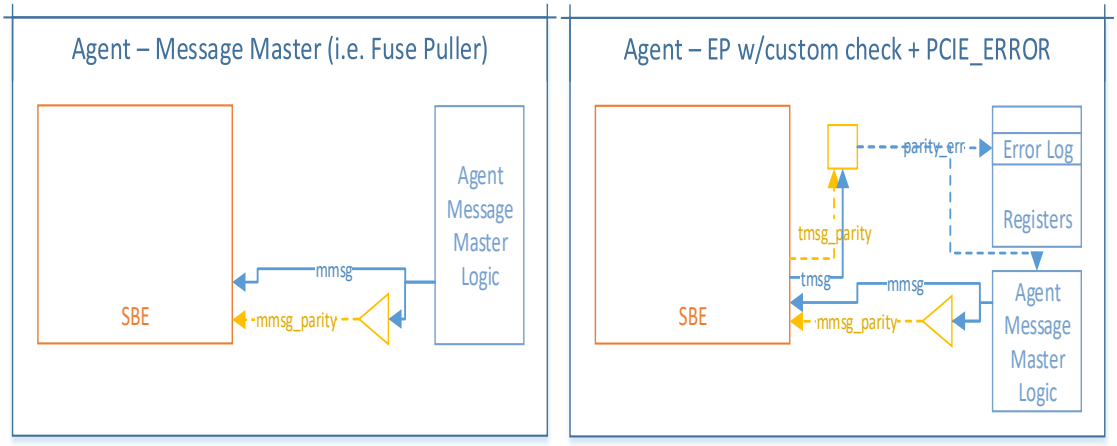
All of IP's master register widgets will need to drive correct parity on their respective mmsg_pc/np_parity inputs to EP. EP will feed through these parity inputs onto the SB fabric interface. If any of the master widgets do not support parity, then the IP will either have to turn off SB_PARITY_REQUIRED in EP or generate mmsg_pc/np_parity as an integration requirement. Generating parity is a combinatorial XOR of the respective payload and eom. fdfx_sbparity_def MUST NOT affect this parity generation.

15.5.4 Parity Error Reporting - DO_SERR Message

IPs that send PCIE_ERROR message, will need to ensure the sbe_parity_err_out from EP, triggers the fatal message through the PCIE_ERROR in its master logic, instead of the DO_SERR within EP. Such IPs should disable DO_SERR_MASTER when enabling parity in endpoint.

Agents (excluding IPs that generate PCI_ERROR) will have to set the DO_SERR_MASTER to 1. DO_SERR is required to generate the error message to the fabric, when a parity error is detected Refer IOSF spec section 3.4.4.1 Agent Parity. The strap pins will directly affect the content of do_serr message and will be IP specific drivers. These strap pins may be tied off if desired.

Figure 14. IP with Custom Master to EP and IP with Custom PCIE+ERROR Generation



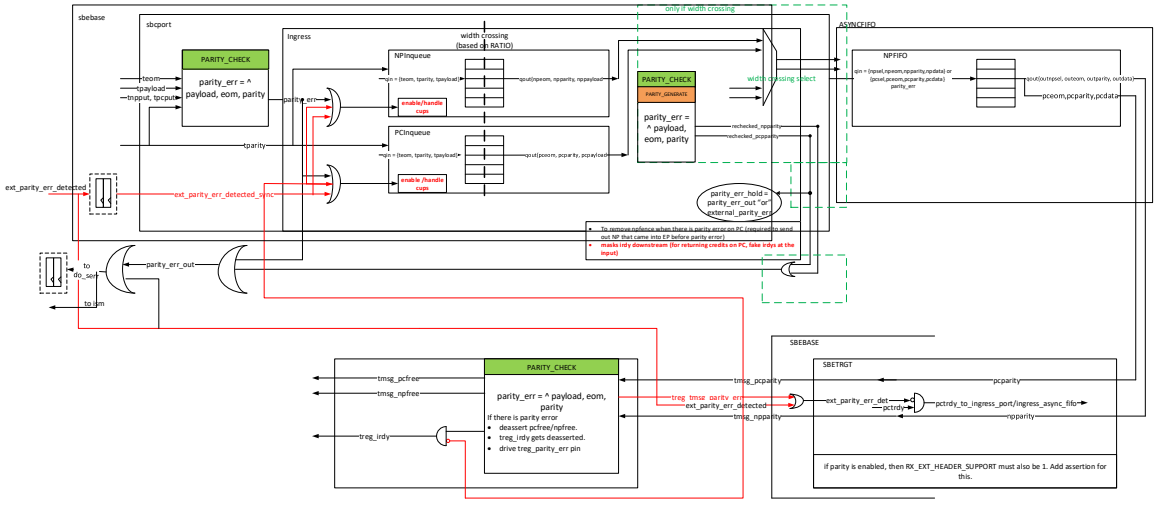
15.5.5 Endpoint Internal Target Path Parity

The endpoint has parity input pins (tparity) from fabric and propagate it to the agents connected as targets on the respective tmsg_pcparity and tmsg_npparity pins. Parity is checked at 3 instances within the EP – ingress port, ingress fifo and at the target register widget (if enabled). Each of this checking engines, check for even parity and the resulting parity error is OR' d together (sbe_parity_err_out) and used to trigger the DO_SERR message with error opcode to be sent with hardwired source/dest/tag input straps.

If there was no parity error within the endpoint, but an agent detected parity error on its internal checks, the agent can notify the endpoint through the external_parity_err_detected input. This external parity input is also a trigger for DO_SERR message and is functionally similar to the other 3 parity check output signals. The external parity err input is double sync'd to side clock in sbebase (when EP is used in ASYNC mode).

At the ingress fifo, parity is checked and regenerated only when there is a width crossing on payload. If the internal and external payload widths are the same, the parity is simply fed through.

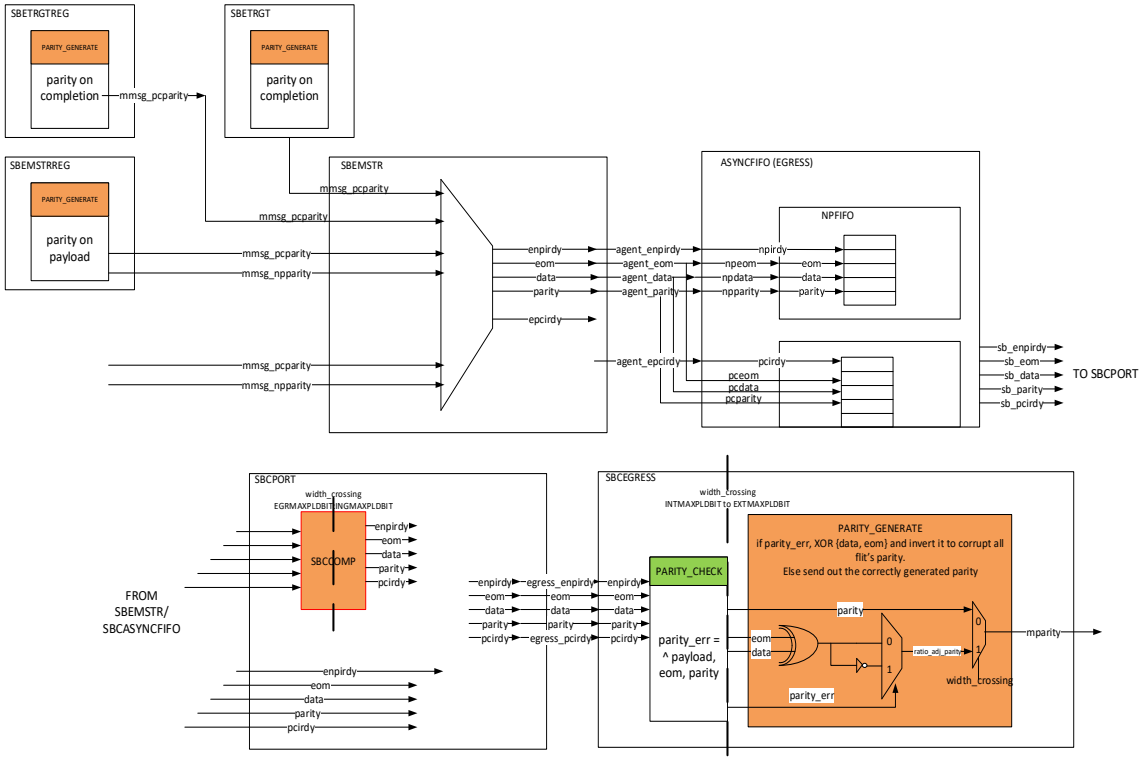
Figure 15. Propagation of Parity on Target Channel



15.5.6 Endpoint Internal Master Path Parity

EP acts a feed through stage when the agent IP master payload with parity on the mmsg_pcparity and mmsc_npparity inputs. Parity is checked only at the egress port (where there could be payload width crossing). If there is a parity error at this check on any flit of the message and if there is width crossing, the entire message (upto EOM) is corrupted even if the message extends upto multiple flits. If there is no width crossing (when both internal and external payload widths are equal), the input parity and simply fed through, bypassing the "intentional corruption of flits" logic.

Figure 16. Propagation of Parity on Master Channel



15.5.7 Parity Error Reporting - DO_SERR Message

IPs that send PCIE_ERROR message, will need to ensure the sbe_parity_err_out from EP, triggers the fatal message through the PCIE_ERROR in its master logic, instead of the DO_SERR within EP. Such IPs should disable DO_SERR_MASTER when enabling parity in endpoint.

All other agents will have to set the DO_SERR_MASTER to 1. DO_SERR is required to generate the error message to the fabric, when a parity error is detected (Refer IOSF spec section 3.4.4.1). The strap pins will directly affect the content of do_serr message and will be IP specific drivers. These strap pins may be tied off if desired.

We intentionally poison the np completion for a bulk read once a parity error occurs in the request. This will create a hung in the router and the DO_SERR message will get stuck but that's fine, since the error will still propogate via the parity_err wire.

15.6 Extended Header Handling

15.6.1 Unique Extended Header Support

More information on extended headers can be found in the IOSF SBC Base Endpoint Integration Guide, which is included in the 'doc' directory for this release. The Base Endpoint integration guide also provides details on the target message interface unsupported response message handling.

Each interface will now have its own separate extended header inputs to aid agents that master register type messages at the same time as letting the endpoint handle completion messages for unsupported messages and targeted register access messages.

The master register module only masters messages and is not a target so there are no output extended headers only input. This interface is a pass through type interface so it is expected that the extended header inputs will be handled similarly to the rest of the mreg_* interface. The valid bits are captured to aid in sequencing each extended header input from the agent.

The target register module will master completion messages and receive posted and non-posted register requests. The treg_* interface has been updated to match the other interfaces and treg_ext_headers will no longer be supported and replaced by a similar interface to the target message interface with treg_rx_*. The new treg_c* interface has been updated to register the extended header information similar to the data at the assertion treg_trdy.

15.6.2 Legacy Extended Header Support

The Endpoint handles extended headers in a similar manner to the Base Endpoint. The exception is the target register interface. The target register interface converts the serialized IOSF messages into a fully parallel transfer, which means that flops are instantiated to store every bit of the transaction. Since there can be a variable number of extended headers attached to a message, it is necessary to bound the supported number of extended headers and, therefore, to bound the physical resources required to store extended headers.

To limit the area required to support extended headers in the target register access block, two more parameters are added: NUM_RX_EXT_HEADERS and [NUM_RX_EXT_HEADERS:0][6:0] RX_EXT_HEADER_IDS. The first parameter is used to determine the number of flops that should be inferred to store the extended headers. The second parameter indicates the valid header IDs that the agent is capable of processing.

If the endpoint receives a transaction with an extended header ID that is not listed in the RX_EXT_HEADER_IDS list, it is dropped transparently. In this case, the treg_eh_discard output is asserted so that the agent can take any appropriate action for receiving a transaction with an unknown extended header. This action is design specific.

One implication of this is that it becomes difficult for the agent to determine exactly how many extended headers have been received and which of those are valid. To assist in this, the 7th bit (treg_ext_headers[N][7]), which is the 'extended header continues' bit within the IOSF spec, is overridden. If this bit is set, it indicates that this extended header was received correctly. If the bit is zero, it indicates that no extended header was received.

For example, consider an agent, which is parameterized to accept up to two extended headers. The treg_ext_headers output is sized accordingly, as [1:0][31:0]. If a transaction with only one extended header is received, it is loaded into treg_ext_header[0] and treg_ext_header[1] is all zero. How can the agent determine if the content of treg_ext_header[1] is a real header received with all zero data, or simply not received at all? In this case, if treg_ext_header[0] was an exact transcription of the data received, it might be possible to determine this by examining bit 7 (EHs continue bit) to ensure it is 0.

However, this cannot be generally relied upon as an indication of a valid header. As another example, consider an endpoint configured to receive two extended headers. It receives a transaction with two headers, the first is a header it supports, while the second is an unsupported header. In this case, if treg_ext_header[0] is loaded with an exact transcription of the data received, bit 7 is set to 1. treg_ext_header[1] is zero. This signature is ambiguous: certainly, treg_ext_header[0] is a valid header, but how does one determine if treg_ext_header[0] is also a header or not? Examining bit 7 directly from the transaction cannot reliably determine this.

To resolve this ambiguity, bit 7 of treg_ext_header is overridden in the register block. If bit 7 is set to 1, it indicates that this entire 32 bit header is valid, regardless of whether any additional headers were received. If it is set to 0, it indicates that no header was received for this treg_ext_header index.

Note: Agent designers must take care to note that there is no specific ordering requirement in the IOSF spec for transmitting header IDs.
The agent design must be capable of accepting headers received in any arbitrary order.

15.7 VISA/DFx Features

The VISA strategy outlined in the IOSF SBC Base Endpoint Integration Guide (included in the ‘doc’ directory for this release) is maintained for the 'full' endpoint. Two additional lanes are included to assist in debug of target register interface.

Table 17. Endpoint Specific Visa Lanes

Output	Clock Domain	Notes
visa_reg_tier1_ag	agent_clk (if clock crossing), otherwise side_clk	
visa_reg_tier2_ag		

15.8 Target Register Access Interface Limitations

The target register access interface is intended to simplify register accesses for the agent; for example, accesses where there are no dependencies on other transactions. It is forbidden to use this interface if asserting treg_trdy for a non-posted register access transaction if it is dependent upon the endpoint finishing a posted-posted handshake with another endpoint.

If assertion of treg_trdy is dependent upon the endpoint sending or receiving any other transaction, it is not appropriate to use this interface. The user must design their own register access logic and design it to avoid deadlock or in/out dependencies.

15.9 Design for Clock Gating

For a detailed discussion about clock gating, see the IOSF SBC Base Endpoint Integration Guide which is included in the ‘doc’ directory for this release. The relevant clock gating signals have different names at the top of the 'full' endpoint as compared to the Base Endpoint. The name changes are documented in Table 18 and Table 19.

Table 18. Trunk Clock Gating Signals

Signal	Clock Domain		Purpose
—	**ASYNCENDPT=0**	**ASYNCENDPT=1**	—
agent_ clkreq	Async		Input from the IP block that indicates that the side_clock is requested. This signal is asynchronously OR’d with internal signals to generate the side_clkreq output signal that is sent to the sideband fabric. This signal is equivalent to sbi_sbe_clkreq in the Base Endpoint. This signal should only asserted when sbe_clk_valid == 1 & sbe_idle == 0 OR when sbe_clk_valid == 0.
sbe_clk_valid	side_clk		Indicates the side clock is valid. Asserted if side_clkreq and side_clkack are both asserted or the fabric ISM state is not IDLE. This signal is equivalent to sbe_sbi_clkvalid in the Base Endpoint.
agent_idle	side_clk	agent_clk	Input from the IP block that indicates the IP has no further messages to send. This is used to move agent ISM from IDLE to ACTIVE_REQ in case if agent is in IDLE. Generally, it is recommended to tie this to the inverse of the mmsg_*irdy signals. This signal is equivalent to sbi_sbe_idle in the Base Endpoint. If the master register module is inserted the mreg_idle signal will be AND'ed together with the agent_idle signal.

Table 19. Agent Clock Gating Signals

Signal	Clock Domain		Purpose
—	**ASYNCENDPT=0**	**ASYNCENDPT=1**	—
sbe_clkreq	side_clk	side_clk/ agent_clk	Clock request signal to agent_clk gating logic. When asserted, any agent_clk gating should be removed. This signal is equivalent to sbe_sbi_clkreq in the Base Endpoint. If the target register module is used then the treg_idle signal will be included into sbe_clkreq.
sbe_idle	side_clk	agent_clk	Idle signal to the IP block’s clock gating logic. When asserted, indicates that the master/target and any asynchronous FIFOs are idle and agent_clk can be gated. This signal is equivalent to sbe_sbi_idle in the Base Endpoint.

15.10 UPF—Isolation and Retention

The UPF files listed in Table 20 are included with the endpoint sub-IP to enable the definition of power intent for agents employing power gating. The files are located under

tools/upf/sbendpoint in the directory tree of the endpoint release. The TCL variables required by the sbe_top.upf file are listed in Table 21.

Table 20. List of UPF Files

File	Description
sbe_create_power_domains.upf	This file creates the power domains (both always on and gated) required to test the sbe_isolation.upf and sbe_retention.upf files.
sbe_isolation.upf	This file identifies the outputs of the endpoint and defines clamp values for each. The file is intended for use as a reference. The endpoint is not expected to sit in its own power domain; therefore, not all endpoint outputs require isolation.
sbe_retention.upf	This file identifies synthesizable elements that require state retention, but also test-only logic that requires state retention during simulation. State retention is applied to the test-only logic only if the upf_for_simulation TCL variable is set to 1.
sbe_top.upf	This top-level UPF file sets up several variables (upf_for_simulation, pg_domain_sbebase, path_to_pg_domain, and path_to_ep_base) and then sources soc_upf.cfg, sbe_create_pwr_domains.upf, sbe_isolation.upf, and sbe_retention.upf files. While this file provides a valid UPF example (it can be synthesized, simulated, and linted), it is intended for test purposes only.
../soc_upf.cfg	This file is a place holder for an equivalent file to be provided by the SoC. The file defines a list of variables, which are referenced throughout the sideband UPF files. This file should be modified (or replaced) during integration into the IP/SoC.

Table 21. TCL Variables Set by `sbe_top.upf`

Variable	Description
upf_for_simulation	If 1, simulation-only logic is state retained. If 0, simulation only logic is not state retained.
pg_domain_sbebase	If 0, the power gated domain is set to sbebase. If 1, the power gated domain is set to the top of sbendpoint.
path_to_pg_domain	This variable defines the hierarchical path to the top of the power gated domain.
path_to_ep_base	This variable defines the hierarchical path to the instance of sbebase.

The IP owner must complete these steps to integrate the `sbe_retention.upf` and `sbe_isolation.upf` files into an agent's UPF script.

1. Determine which endpoint outputs are on the edge of the agent's power domain.
2. Modify sbe_isolation.upf to clamp only these outputs.
3. In the agent's top-level UPF script, set the variables from Table 21 appropriately.
4. Modify/replace soc_upf.cfg to define the necessary power domain names, state retention cells, isolation cells, and others.
5. In the agent's top-level UPF script, source sbe_isolation.upf and sbe_retention.upf after the necessary power domains, supply ports, and supply nets have been created.

The UPF files can be incorporated into a unit-level power-aware simulation of the endpoint by performing the following steps.

1. Change directory to the root of the endpoint release directory tree.
2. Source the endpoint setup file by typing:

```
cd $IP_ROOT/scripts
source setup -x spt
```

3. Call ACE by typing:

```
ace -cc -x -epi -epa –t test01 -model Ep_pa
```

4. To bring up the DVE GUI, include '-sd gui' to the ace call.