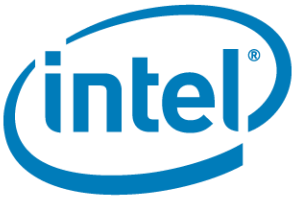


IOSF SBC Base Endpoint Unit

INTEGRATION GUIDE

ALL RTL 1.0-PICr35
January 2021

Intel Top Secret

Contents

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 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

1.3 Related Documents

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

- IOSF Specification 1.0
- SIP Converged IOSF Trunk Clock Gating Methodology (included in the 'doc' directory for this release)

1.4 Contact Information

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

Table 1. Contact Information

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

1.5 Document Revision History

Table 2. Revision History

Revision	Date	Description
0.9	5/14/2010	Initial publication Addition of asynchronous FIFO timing diagrams.
2010ww41	9/30/2010	Updated VISA information to match new struct outputs. Added Target Register Access limitations section.

Revision	Date	Description
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.
2012ww51 (addendum)	1/23/2012	Added documentation for side_ism_lock_b.
2013ww18	05/2013	Edited and added to standard SIP template.
	04/24/2014	Updated jtag behavior to match the current design.
2014WW22	05/29/2014	Updated clock request behavior to inform agents how to handle the clock requests. Added unique extended header information for unsupported response message generation. Updated universal synchronizer behavior and diagram.
2014WW30	07/14/2014	Added new parameters, output and descriptions for expected completion logic additions to the endpoint. Identified parameters that the users may need to pass up to the top level so that full chip architecture may set them as appropriate.
2014ww39	9/24/2014	Added parity ports and parity-enable parameter
2015ww30	7/21/2015	Updates for 2015ww30 SBE release (CLAIM_DELAY)
	9/30/15	Updated Table 17 in section 16.8, Configuration and Override Inputs, to indicate that all config/override inputs are now synchronized to side_clk, with the exception of jta_force_clkreq.
2019ww12	3/20/19	Fixed N/A sections for new Integration Guide template.

2 Quick Start

Note: Make sure everyone can run listed scripts. Specify if any special access is needed.

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 4.3, HIP Libraries Included in Release.

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.

2.5 IP/IPSS Straps

Not applicable to this IP

2.6 Integrity Checks for Standalone IP

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
sbebase_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
   sbebase_runSynth
   ```

6. Run CDC:

   ```
   cd $IP_ROOT/tools
   sbebase_runCDC / sbebase_runACECDC
   ```

7. Run LEC:

   ```
   cd $IP_ROOT/tools
   sbebase_runLEC
   ```

8. Run Scan Insertion:

   ```
   cd $IP_ROOT/tools
   sbebase_runATPG
   ```

3 Overview

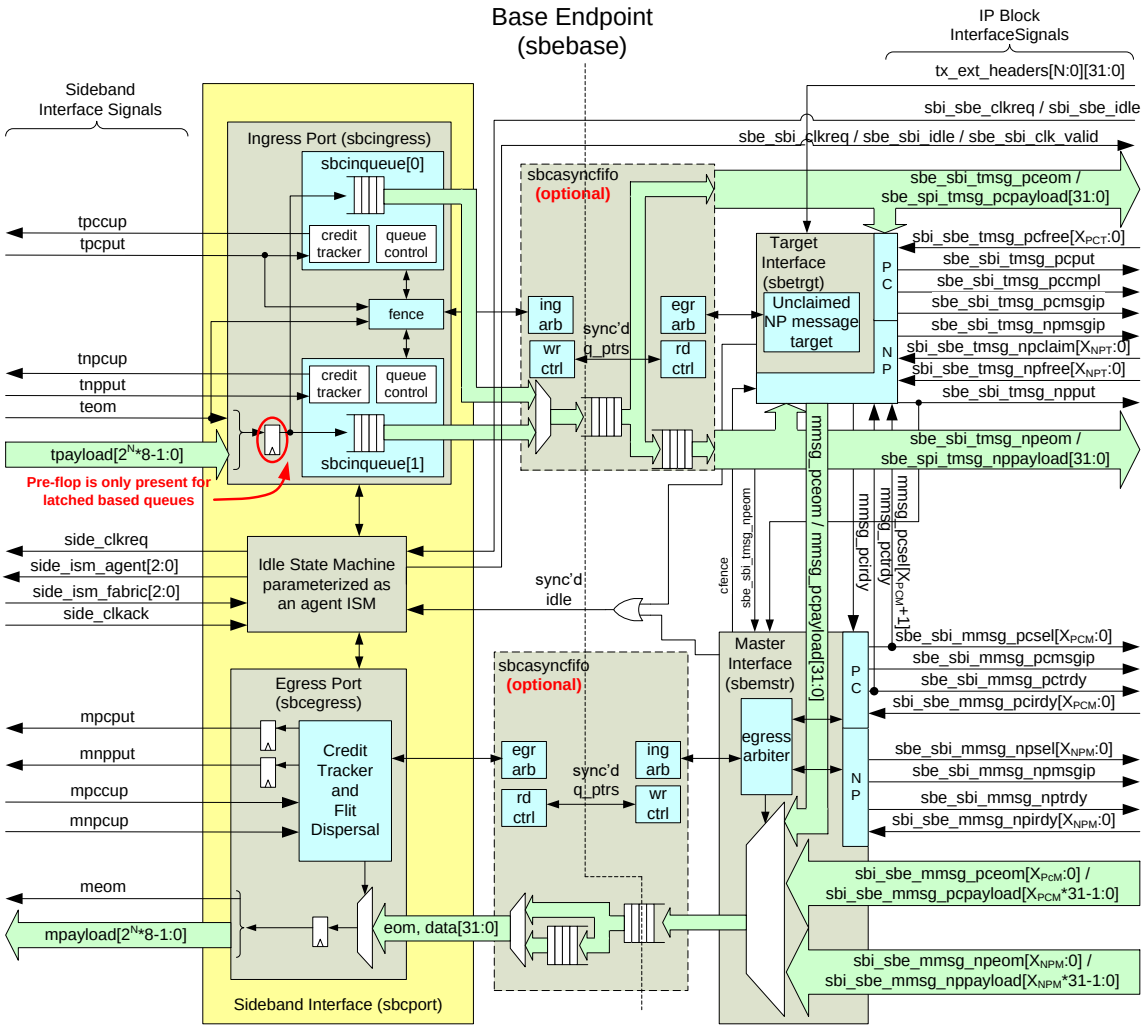
3.1 Block Diagram

The sideband interface, base endpoint contains at least three blocks as shown in Table 3.

Table 3. Block Diagram

Block	Description
SBCPORT	The sideband interface port that contains the ingress port, the egress port and the clock gating ISM
SBETRGT	The block that controls the Target Interface that connects the base endpoint to the message target agents within the IP block.
SBEMSTR	The block that controls the Master Interface that connects the base endpoint to the message master agents within the IP block.
SBCASYNCFIFO	Optionally instantiated to create asynchronous endpoint.

Figure 1. Base Endpoint Block Diagram



3.2 Top-level Signals / Functional Ports

For a list of top-level signals, see section 13.2.1, Signals List.

3.2.1 Signal List File

A complete list of top-level signal is found in the `source/rtl/iosfsbc/endpoint` folder.

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/IPSS

3.2.4 Control Signals

See `SBE Integration Guide` for all control signals (included in the 'doc' directory for this release).

3.2.5 Reset Ports

See `SBE Integration Guide` for all reset ports (included in the 'doc' directory for this release).

3.2.6 Naming Scheme

See `SBE Integration Guide` (included in the 'doc' directory for this release).

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 following table are required by the IOSF Sideband environment. They are initialized by sourcing the setup script in `$IP_ROOT/scripts`

Table 4. Environment Variables

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/IPSS

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 sbebase_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/sbebase/`

4.8 Synthesis

All synthesis related scripts reside in:

`$IP_ROOT/tools/syn/syn_<PROCESSVERSION>/sbebase/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, sbebase_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`
- `sbebase_clocks.tcl`
- `sbebase_io_constraints.tcl`
- `sbebase_scan_config.tcl`

4.8.1 Clocks

The primary clocks are listed in Table 5.

Table 5. 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 Diagram

See Figure 1, Base Endpoint Block Diagram.

4.8.3 Constraint Files

All constraints files are available under:

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

4.8.4 BKMs

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

4.8.5 Scan Insertion

This IP does not have special requirements for scan insertion.

4.8.6 Latches

The IP provides an option for a latch-based implementation of the FIFOs (please refer to the description of the LATCHQUEUES parameter in Table 10, Parameters.

4.9 Formal Verification

After the setup script is run for the given customer configuration and synthesis results are available, the helper script sbebase_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 sbebase_runCDC, and sbebase_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 sbebase_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 sbebase configurations.

4.12 DFD Triggers and Actions

Not applicable to this IP

4.13 HDK Tool Runs

Follow the steps below to run the tools in an HDK environment using the MAT/TSA versions of the tools. Start from the SBEP_ROOT directory -the Endpoint tarball root directory.

```
cd SBEP_ROOT
```

The total number of groups should be less than 16 for the environment to be setup correctly

```
wash -n users fabric soc socenv dk1273 siphdk hdk10nm dk10nm
hdk10nmproc coe73 coe73lay soc73 datools hdk22nm hdk22nmproc
```

Source HDK and set the model root

```
source /p/hdk/rtl/hdk.rc -cfg sip –reentrant
setenv MODEL_ROOT $cwd
```

Generate the helper scripts (as described in section 4.8, Synthesis.

```
cd $MODEL_ROOT/scripts/qa
runFullConfig -endpoint
```

Run Simbuild and the desired tools (using the run*_HDK<> helper scripts)

```
simbuild -dut sbe -1c -CUST <p1273 or p1274> -1c-
cd $MODEL_ROOT/tools
```

<sbendpoint/sbebase_config>_run<toolname>_HDK<p1273 or p1274>

E.g.

```
sbendpoint_runACECDC_HDKp1273
```

Other helper scripts with "_HDK" in their run names can be run similarly. Currently supported tools are Lintra, ACECDC, EFFM, and VISA.

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 customer deifne their specific connectivity.

5.3 Embedded Building Blocks/Custom Logic

Not applicable to this IP/IPSS

5.4 Macros used by IP/IPSS

Not applicable to this IP/IPSS

5.5 RTL Configuration Parameters and Overrides

The RTL configuration parameters for this IP are listed in Table 10, (Parameters), in section 13.2.1, Signals List.

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

5.9 Fuses

Not applicable to this IP

5.9.1 Fuse RDL Files

Not applicable to this IP

5.10 Clock, Power, and Reset Domains

See sections 14 (Clock and Reset Scheme), 16.9, (Design for Clock Gating), 16.10, (Design for Deterministic Clock Crossing), and 16.11, (Design for Power Gating).

5.10.1 Clock Domain Diagram

This IP has two clocks, side_clk, and agent_clk. If these clocks are different, the user should instantiate the optional sbcasyncfifo module as shown in Figure 1, Base 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; this Ip doesn't have PGCB or level shifters.

5.10.4 Power Domain Mapping

IP power port names used in the upf file are parameterized and can be maaped to soc power supply rails. For example:

```
Set SOC_GROUND_PORT vss
```

5.11 Power Information

5.11.1 Power Supply

Not applicable to this IP

5.11.2 Static Clock Gating

See section 16.9, Design for Clock Gating.

5.11.3 Power Gating

See section 16.11, Design for Power Gating.

5.11.4 Bumps and Their Power Domains

Not applicable to this IP/IPSS

5.11.5 Voltage Rail Requirements (HIP only)

Not applicable to this IP/IPSS

5.12 System Startup

5.12.1 Power-up Requirements

Not applicable to this IP/IPSS

5.12.2 Power-up Sequence

Not applicable to this IP/IPSS

5.12.3 Initialization Sequence

Not applicable to this IP/IPSS

5.12.4 Device Configuration

Not applicable to this IP/IPSS

5.12.5 Header for Windows Boot

Not applicable to this IP/IPSS

5.13 DFx Considerations

The IP is compliant with the Chassis DFx Gen2 standard.

5.13.1 DFx Top-Level Signals

See Table 11, Clock, Reset, and DFx Signals.

5.13.2 DFx Clock Definition

Not applicable to this IP/IPSS

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 16.7, VISA / DFx Structures.

5.13.5 DFD Triggers and Actions

Triggerging can happen based off DSOs instantiated at soc or through NPK, via VISA signal capture.

5.13.6 Debug Registers

Not applicable to this IP/IPSS

5.13.7 Scan – Clock Gating in RTL

See Table 11, Clock, Reset, and DFx Signals, for scan signals.

5.13.8 Scan – Reset Override

Scan reset-override is external to this IP.

5.13.9 Scan – Constraints and Coverage

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

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/IPSS

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/IPSS

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 10, Parameters, in section 13.2.1, Signals List.

5.14.2 Validation Requirements

Not applicable to this IP/IPSS

5.14.3 Interface Signals Implemented for Security

See Table 11, Clock, Reset, and DFx 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

  Not applicable to this IP
- IP_SoC_**SW**_Systematic_FuSa_Requirement_Guideline

  Not applicable to this IP

- 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 6. Compliance Statuses

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 7. Values

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/IPSS

5.18 IP META Data

For this IP customer define their configration 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

The estimated area is 2k gates.

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 Process Parametric Sampling and Monitoring (PPSM)

6.3.1 PPSM Requirements Comprehended

Per PLC requirement, at the close of the SiE MFG TR (POPL2) milestone, the IP teams should have reviewed and comprehended IP and SoC level PPSM requirements outlined in the specification denoted at http://goto/trspecs with a design commit matrix populated. Usually the chief technologist (CT) ensures early review with commit matrix has completed and that the PPSM team has had opportunity either through invitation or other means to review the commit.

6.3.2 Final Review of PPSM Requirements

By closure of SoC RTL 0.8 milestone, a second review of the implementation of the PPSM requirements (denoted and committed to in 6.3.1 with PPSM team participation) is required to ensure issues, gaps, misses, or concerns are addressed in sufficient time to mitigate. Again, usually the chief technologist (CT) and/or chief engineer (CE) or product execution lead (PXL) for the product coordinates this review.

6.4 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

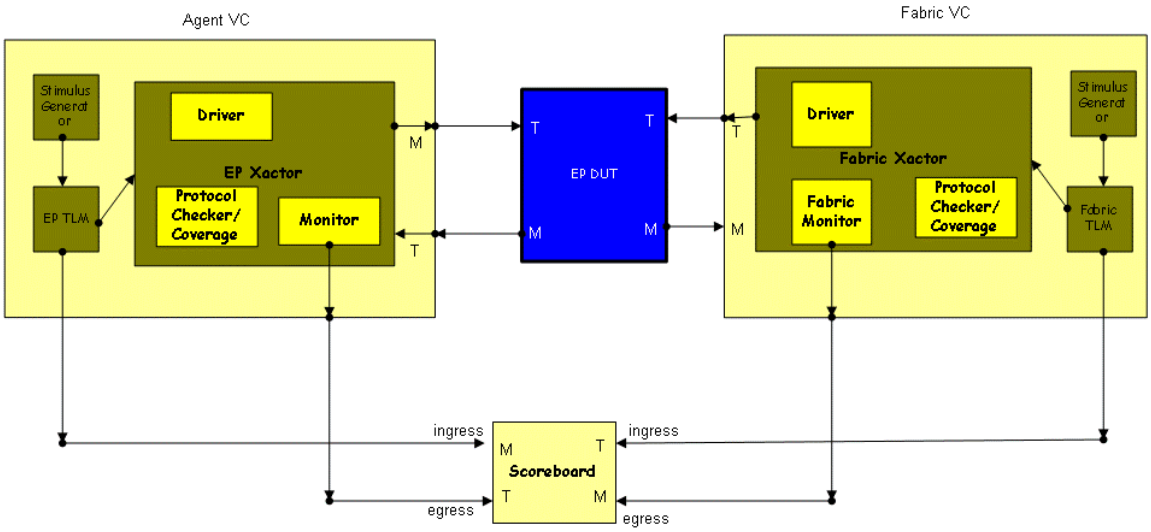
This IP has following details for the verification plan.

For additional details, see the complementary SIIP Verification Plan (included in the 'doc' directory for this release).

8.1 Testbench Overview

The IP testbench environment is shown in Figure 2.

Figure 2. IP Testbench Environment



8.2 SoC-specific Validation

This IP verification environment provides comprehensive testing for 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

The following table 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:

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

Table 8. Testbench 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.

Parameter Name	Range	Default	Descriptions (including interdependencies)
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 the pipeinps parameter.
pipeinps	0, 1	0	Set to 1 to pipeline all fabric side inputs (except ISM Inputs). Must be equal to the pipeisms parameter.
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.
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.
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

8.5.1 Testbench Directory Structure

The verification IPs used in the testbench are in `$IP_ROOT/verif/bfm`

Testbench is located here: `verif/tb/top/ep_tb/tb_top`

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.

8.5.6 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

8.5.9 Collage or Sandbox Files

Not applicable to this IP

8.5.10 IP Environment

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

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

8.5.10.1 Configuring the IP Environment

```
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
```

8.5.10.2 Saola Environment Walkthrough

Not applicable to this IP

8.5.10.3 System Manager

Not applicable to this IP

8.5.10.4 Fuse

Not applicable to this IP

8.5.11 Sequences

All sideband sequences are located in sideband BFM:
`<IP>/verif/bfm/sideband_vc/tb/seq_lib/`

8.5.11.1 Sequence for Bringing up the IP

Not applicable to this IP

8.5.11.2 BFM Sequences

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

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

8.5.11.3 IOSF Primary/Sideband BFM Sequences

Not applicable to this IP

8.5.11.4 Other Reusable Sequences

Not applicable to this IP

8.5.11.5 IP Test Sequences

Not applicable to this IP

8.5.11.6 SoC Requirements for Sequence Reuse

SoC could re-use the iosf_sb_seq sequence.

8.5.11.7 Sequence File Dependencies

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

All sequences included in sequence library package `iosfsbm_seq_pkg`

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

8.5.12 Miscellaneous

8.5.12.1 Using the Runtime or Post-Processing Checkers

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

8.5.12.2 Environment Files

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

8.5.12.3 Coverage

The IP uses functional and code coverage.

8.6 Collage or Sandbox Files

Not applicable to this IP

8.7 IP-Level Information Required for Sequence Writing

IPs could re-use the iosf_sb_seq sequence.

8.8 Environment Settings and Files

8.8.1 Base Test

The base test is located at `<IP>/verif/tests/ep_tests/base_test.svh`. It instantiates the env_ip environment, and configures the components listed in section 8.5.10, IP Environment.

8.8.2 Configuration Object

The configuration objects are listed in section 8.5.10, IP Environment.

8.8.3 API

Not applicable to this IP

8.9 Description of Reusable Tests

Not applicable to this IP

8.10 Description of Reusable Automation Scripts

Not applicable to this IP

8.11 Supported Compiler Options for Simulation

Not applicable to this IP

8.12 Reusable Simulation RUNMODEs

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

10 Workarounds, Waivers, and Disabled Assertions

Not applicable to this IP; the integration of this IP does not require any workarounds or waivers, or disabling of assertions.

10.1 Disabled Assertions

Not applicable to this IP; 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

Not applicable to this IP

12 Overview – from Old Version

The IOSF Sideband Channel Base Endpoint provides a connection to the IOSF Sideband Channel and provides very basic master/target services on the IP block-side interface to the endpoint.

This allows the sideband channel to be used as a multi-purpose chip-wide (out-of-band) communication fabric for some specifically defined services and for any implementation specific services.

The types of services provided are:

- Access to PCIe config, memory, or I/O mapped address space.
- Access to privately mapped configuration registers (implementation specific).
- INTx assert / de-assert messages sent out-of-band.
- General virtual wire messages (implementation specific).
- Distribution of fuse values (implementation specific).
- Power management broadcast messages (implementation specific).

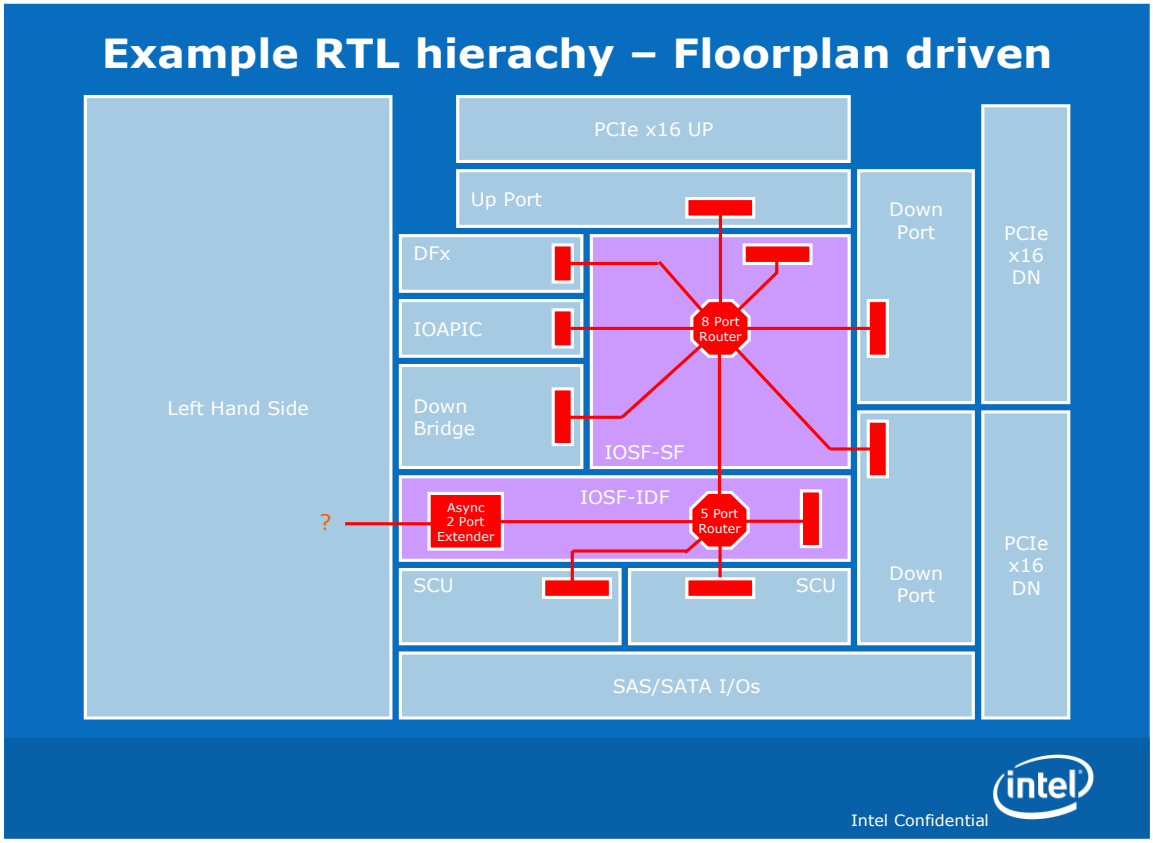
Usage of the term sideband *channel* instead of sideband *interface* is acknowledged as the incorrect terminology; however, there are no plans to replace all channel references with interface. The older versions of the IOSF spec initially used channel, which was adopted and used prevalently throughout the MAS and RTL/CVS module hierarchy.

12.1 Full-Chip Context

The placement of the Sideband Channel Endpoints is determined by the sideband services required by each IP Block.

An example floorplan showing potential placement of the Sideband Channel Endpoints is shown in Figure 3, Base Endpoint Block Diagram. The endpoints are denoted by the (unlabeled) red rectangles.

Figure 3. Full-Chip Example Location of Endpoints



12.2 Operating Conditions

The clock frequency supported by the base endpoint should be able to reach 500 MHz. However, this is dependent upon the process and floorplan distribution of endpoints and routers. Using process 1271 (low leakage), 500 MHz appears to be the upper bound for the sideband collateral without requiring a more complex pipelined implementation.

12.3 Definitions

Table 9. Definitions

Acronyms	Description
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

12.4 Feature List

- Parameterized ingress queue depth.
- Parameterized sideband payload width: Support payload widths of 1, 2, or 4 bytes.

- Parameterized number of posted/non-posted master/targets.
- Optional asynchronous FIFOs for clock synchronization.
- Parameterized support for extended headers.
- Parameterized support for completion fencing to ensure IOSF completion ordering compliance.

13 Block-Level Micro-Architecture

13.1 Block Descriptions

13.1.1 Sideband Interface Port (SBCPORT)

This block contains all of the blocks common to the endpoints and routers, which contains the clock gating ISM (SBCGCGU/SBCISM) (section 13.1.1.1, Idle State Machine (SBCGCGU)), the ingress port (SBCINGRESS) (section 13.1.1.2, Ingress Port (SBCINGRESS), and the egress port (SBCEGRESS) (section 13.1.1.3, Egress Port (SBCEGRESS)).

13.1.1.1 Idle State Machine (SBCGCGU)

The idle state machine provides a synchronous mechanism for allowing the sideband endpoint and fabric to enter/exit an idle state for clock gating.

When in the idle state, the sideband channel clock in the endpoint can be gated to save power.

When in the active state, the endpoint is allowed to send and receive messages on the sideband channel.

The agent ISM is reset to the IDLE state and proceeds to transition through the credit initialization process after reset de-asserts. The ISM transitions to ACTIVE and begin to transfer data if the IP has messages to transmit to the router or the router has messages to transmit to the IP.

The ISM state and transitions are largely transparent to the IP. Pipelined or non-pipelined fabric ISM signals are available as inputs to the agent ISM based on parameterized options. These options can cause issues with the current IOSF compliance monitor due to state transitions that are now supported by IOSF 1.0 spec.

In support of dynamic power gating, the endpoint accepts the ism_lock_req_b as an input from the agent. Integration details related to ism_lock_req_b are in section 16.11, Design for Power Gating.

13.1.1.2 Ingress Port (SBCINGRESS)

The ingress port blocks contain queues for non-posted and posted/completion messages.

The queue depth can be configured to allow the project to optimize bandwidth and latency vs. power and area.

A significant feature of the Ingress Ports is to maintain the ordering rules. Sideband ordering rules are defined in Section 3.3.2.2 of “IOSF Specification 1.0”.

More specifically, non-posted messages are not allowed to pass posted messages or completions; even more specifically speaking, the first byte of a non-posted message is not allowed to pass the first byte of a posted/completion message. This is achieved by a fencing mechanism.

There is a posted/completion message counter that keeps track of the number of messages within the ingress block (either in the ingress queue or in the accumulator flops).

When the start of a non-posted message is loaded into the non-posted accumulator output flop, the current count of posted/completion messages is loaded into the fencing counter, which is decremented to zero when each successive posted/completion message leaves (eom)

the ingress block. When the counter reaches zero, it is then safe for the non-posted message to be allowed to leave the ingress block.

Note: This fencing mechanism is stronger than it needs be which further encourages posted/completion messages to pass non-posted messages.

The fencing mechanism has been simplified to minimize the complexity and number of gates required for this feature.

13.1.1.3 Egress Port (SBCEGRESS)

The egress port contains credit counters for both types of message flits: posted/completion and non-posted. When there are no available non-posted credits (for example, np counter is equal to zero), the egress port asserts the npstall signal to the master interface block so that it can make the appropriate decision to allow a posted/completion message to pass a stalled non-posted message.

The credit counters increment at the end of the cycle when the m*cup signal is asserted. The credit counter decrements at the beginning of the cycle when the m*put signal is asserted. The egress port also contains a flit disperser (which is the logical opposite of the ingress accumulator).

The flit disperser sequences through all of the flits presented from the master interface, which could be either 1, 2 or 4 flits depending upon the payload width ratio between the internal data path width of the endpoint (32 bits) and the external sideband payload width (8, 16 or 32 bits). The flits are captured into the mpayload/meom output flops (assuming that there is at least one credit available and the ISM is in the ACTIVE state) and the appropriate m*put signal is asserted.

Note: The master interface block is only allowed to assert the posted/completion irdy signal or the non-posted irdy signal to the egress block, or neither.

It never asserts both, which could cause a collision between the two messages. The appropriate trdy signal is asserted during the cycle that the last flit is captured from the master interface.

13.1.2 Master Interface (SBEMSTR)

This block provides an interface to the IP block for mastering messages on the sideband interface. Messages can be either posted or non-posted.

The master interface allows for a parameterized number of posted/completion master agents and a parameterized number of non-posted master agents within the IP block (outside of the base endpoint). Messages are transferred into this block over multiple cycles, the first cycle containing the standard message header common to all messages (4 bytes, source, and destination port ID, the opcode and the transaction tag). Additional cycles can be used to transfer subsequent dwords of the message.

All messages contain an integer number of dwords; simple messages and completions without data are a single dword, all other messages contain two or more dwords.

The master interface contains separate posted/completion and non-posted message interfaces, allowing for posted/completion messages to pass non-posted messages when necessary.

Both the IP block and the base endpoint are responsible for ensuring that non-posted messages do not pass posted/completion messages.

If both posted/completion and non-posted irdy signals are asserted, this means that the IP block has determined that it is safe to send the non-posted before the posted/completion. However, if the endpoint begins sending a posted/completion message (mmsg_pcirdy asserted, mmsg_pcmsgip de-asserted and the internal PC/NP message arbiter is selecting posted/completion message), it does not initiate a non-posted message until after the end of the posted/completion message is sent.

Therefore, the IP block has (potential) ordering responsibilities that must be met before asserting a non-posted irdy signal and the base endpoint has ordering responsibilities that must be met before initiating a new non-posted message.

This block contains a (fixed) equal weighted round-robin arbiter for posted/completion and non-posted messages. Each is given the same fixed equal weighted chance for mastering the next message.

When a posted/completion message is selected, it completes without allowing a non-posted message to be initiated. When a non-posted message is selected, it starts and is given the highest priority for the egress port as long as it has sufficient credits (for example, it is not stalled). If the non-posted message stalls, then the next posted/completion message is initiated and is interspersed with the non-posted message.

However, as soon as the credits are available for the non-posted, the arbiter switches back to the non-posted message until it completes.

13.1.3 Target Interface (SBETRGT)

This block provides an interface to the IP block for receiving messages as a target.

The interface provides for separate busses for NP and PC messages, because PC and NP messages can be received interleaved with each other. This allows posted/completion messages to be guaranteed forward progress, by allowing them to pass the non-posted messages. See IOSF ordering requirement as specified in Section 3.3.2.2 of "*IOSF Specification 1.0"*.

The target interface contains a free[N-1:0]/put protocol rather than the standard irdy/trdy protocol used on the master interface. This is needed to support multiple targets.

The transfer occurs on the target interface when the next 4 bytes of a message is available from the ingress port (internal irdy asserted) and ALL targets are ready for the transfer (all free signals asserted), then put is asserted. The put is a combinatorial AND of the internal irdy and all of the free signals.

Because of the combinational logic in the endpoint, the targets agents must not combinatorially generate the free signals (input to the base endpoint) from any outputs of the base endpoint or risk combinatorial loops.

The free signals must be solely based upon the current internal state of the target agent. The target agent must decide if it can receive the next dword message flit from the base endpoint, regardless of the contents of the message (such as opcode or source port ID).

The 0.8 endpoint RTL and newer revision added two tmsg_*valid outputs from this block, which are subsequently routed to primary outputs of the base endpoint.

These outputs are provided as a convenience to the attached agent IP design and are particularly useful at the beginning of NP transactions. When these signals are asserted, it indicates that the associated DW on the tmsg_*payload output is valid and ready for transfer.

The agent might inspect the destination, source, opcode, or any other attribute within the first DW of the transaction to determine whether it should claim the transaction. The agent can

implement a pipelined npclaim function for single DW NP transactions, whereas before the npclaim timing had to be coincident with the free signal.

Note: The free / put handshake still needs to occur in order to continue forward progress of the transaction.

For the first dword transfer of a new message, the targets assert free only if they have the resources available to receive this first 4-byte transfer. At this point, all targets must assume that they are the target of the message (without viewing the message header contents).

After one or more transfers, each target can realize that they are not the target of the message*. If so, the target must assert the free signal because it has no reason to stall the message*. All other targets assert their free signals if they have the internal resources to continue receiving the message. Once a target decides that it is the target of a non-posted message, it asserts the npclaim signal some time before or at the same time as the final 4-byte transfer of the message (EOM).

Posted/completion messages do not require a claim signal because the posted/completion messages that are not claimed are silently dropped.

If the agent does not claim a non-posted message, the sbetrgt block synthesizes a UR completion and return to the initiator.

The SBETRGT captures the destination port ID, source port ID and tag of the non-posted message in order to generate an "unsuccessful / not supported" completion for any unclaimed messages. This is the subtractive decode agent function of the SBETRGT block and helps to guarantee that all non-posted messages have a corresponding completion.

If a target claims a non-posted message, then it is responsible for generating the appropriate completion for that non-posted message and transmitting it back to the originating port ID by initiating a PC transaction with the master interface (described above). Also, the IP block must ensure that at most one target claims a non-posted message and/or has logic to ensure that only one target generates the required completion.

Note: *It is the responsibility of the IP block to ensure that all non-posted messages that are delivered to the IP block from the target interface and claimed by a target to have a corresponding completion that is mastered on the master interface*. Failure to do this, results in a non-functional sideband interface due to either a master agent in another endpoint waiting for a completion or a router waiting for a broad/multicast completion that never occurs.

This block also contains masking logic to ignore the internal non-posted irdy signal when the non-posted fence signal is active, which ensures that a non-posted message does not pass a posted/completion message.

With the IOSF 0.9 and newer revision of the endpoint, an optional completion-fencing feature was added between the sbemstr and sbetrgt blocks. The purpose of this fencing feature is to ensure that the completions returned by the attached IP block satisfy the IOSF requirement that completions be returned in requested order.

Upon receipt of a non-posted transaction, sbetrgt is blocked from receiving any further non-posted transaction until a completion is returned through the sbemstr block. This feature can be disabled per parameter for complex IP designs, which need to support multiple outstanding NP requests and have the appropriate completion ordering mechanism in place.

13.1.4 Asynchronous FIFO (SBCASYNCFIFO)

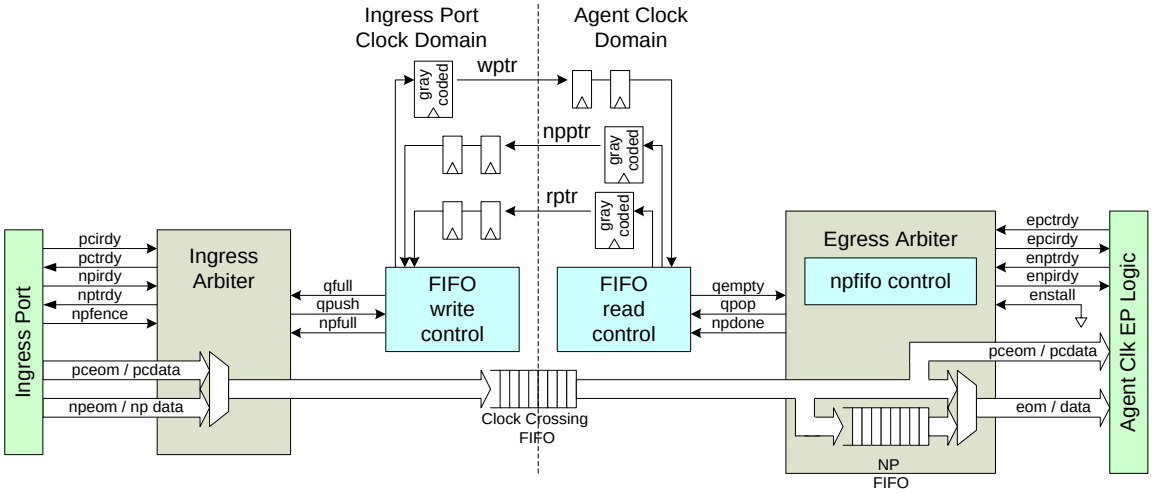
The endpoint optionally implements an asynchronous FIFO (aFIFO) to pass sideband message flits from one clock domain to another clock domain. This block is optionally instantiated

between the SBCPORT block and the rest of the blocks within the endpoint, one instantiation for the ingress data path and one instantiation for the egress data path.

When the input side of the aFIFO is connected to an ingress port, it arbitrates between the posted/completion and non-posted messages, always giving priority to the non-posted ingress queue.

The Ingress Port contains a fencing mechanism to ensure that non-posted messages do not pass posted/completion messages.

Figure 4. Asynchronous FIFO Architecture



As shown above in Figure 4, the construction of the FIFO is somewhat unique, in that there is a single clock-crossing FIFO and then a separate NP FIFO on the egress side. This separate NP FIFO is implemented to fulfill the IOSF requirement that PC transactions pass stalled NP transactions. Had only a clock crossing FIFO been implemented, a stalled NP transaction could stall the entire flow of transactions. Having separate clock crossing FIFOs for NP and PC transaction was also a possibility but that architecture was not chosen due to additional complexity with ordering across separate clock crossing FIFOs.

Posted/completion flits can be loaded into the FIFO as long as the clock crossing FIFO is not full. PC flits are also directly read off of the clock crossing FIFO and into the egress clock domain. NP flits are handled somewhat differently. Non-posted flits have an additional restriction; not only must there be space available in the clock crossing FIFO, but there also must be space available for the non-posted flit in the non-posted egress FIFO before the flit is loaded into the clock crossing FIFO. This is because NP flits are loaded from the clock crossing FIFO into the NP FIFO if the endpoint stalls reception of the NP flit. If this NP FIFO was full, the NP flit would cause the clock crossing FIFO to stall and prevent PC flits from making progress.

Low latency across the aFIFO is achieved by passing gray-coded queue pointers across the clock domain boundaries. The queue pointers are true gray-coded values, meaning that they are a reflective encoding. The queue depth is parameterizable, from 2 to 32 entries, and must contain an even number of entries. The queue is implemented as a 2-dimensional packed array: width by depth. With this scheme, a gray-to-binary conversion is needed to create a non-power of two sequential ranges of values, which uniquely access each element in the array without exceeding the bounds of the array.

13.2 Block Interface

13.2.1 Signals List

Table 10. 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. If streaming is required, this should be set to 4.
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.
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 and this may be deprecated in a future release.
MAXNPTRGT	0		The maximum bit of the sbi_sbe_tmsg_nptrdy signal, which defines the number of non-posted targets in the IP block.
MAXPCTRGT	0		The maximum bit of the sbi_sbe_tmsg_pctrdy signal, which defines the number of posted/completion targets in the IP block.
MAXNPMSTR	0		The maximum bit of the sbi_sbe_mmsg_npirdy signal, which defines the number of non-posted masters in the IP block.
MAXPCMSTR	0		The maximum bit of the sbi_sbe_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.

Parameter Name	Default Value	Expose at Top Level?	Description
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.
DISABLE_COMPLETION_FENCING	0		Set to non-zero value to disable the completion fencing algorithm.
RST_PRESYNC	0		Do not change this value from its default used by the endpoint wrapper sbendpoint.
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.
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.
SBR_VISA_ID_PARAM	11		Identifier for this SBR in the VISA network.
NUMBER_OF_BITS_PER_LANE	8		Parameters to control VISA lanes and mux structure. Please refer to VISA tool documentation for full implications.
NUMBER_OF_VISAMUX_MODULES	1		
NUMBER_OF_OUTPUT_LANES	2		
USYNC_ENABLE	0		Parameter to enable deterministic clock crossing in an asynchronous endpoint. The feature is disabled by default; set parameter to 1 to enable. Deterministic clock crossing can be dynamically disabled using the usyncselect input. When enabled, side_usync, agent_usync, and usyncselect signals must be driven appropriately. Scan chain insertion struggles to add this automatically and appropriate RTL changes for this will be made in a future release.
SIDE_USYNC_DELAY	1	When exposing USYNC_ENABLE	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. This value must be set less than the period, which the usync GAL will occur; otherwise, data will be acquired near a clock event causing metastability in the receiving clock domain.

Parameter Name	Default Value	Expose at Top Level?	Description
AGENT_USYNC_DELAY	1	When exposing USYNC_ENABLE	Used to set a counter within the universal synchronizer circuit to delay the transfer from side_clk to agent_clk. This value must at least 1. This value must be set less than the period, which the usync GAL will occur; otherwise, data will be acquired near a clock event causing metastability in the receiving clock domain.
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'd or dropped bits in this field then the agent must size it appropriately.
EXPECTED_COMPLETIONS_COUNTER	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	Y	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.
SIDE_CLKREQ_HYST_CNT	15	Y	Used to program the delay between the clocking units assertion of idle to the de-assertion of side_clkreq. The internal counter will resize as appropriate. It is advised to leave this parameter default value. This value must be greater than 0.
SB_PARITY_REQUIRED	0	Y	When set to 1, enables parity support on the Base Endpoint

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 (please see …/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.
CLKREQDEFAULT	0	Y	IOSF Spec 1.2 states that the reset value (when reset is in effect) of side_clkreq from EP to fabric is 0. But the RTL so far had been compliant to earlier versions of the IOSF Spec, setting the reset value to 1. This was to support early boot IPs. To avoid this discrepancy between the spec and RTL, at the same time to support IP's who have already set up their clock requesting logic to be consistent with the RTL/SPEC, a new parameter CLKREQDEFAULT has been added. Not setting (or setting this to 0), will result in the legacy behavior (reset value of 1 on side_clkreq). When this parameter is set to 1, side_clkreq does not have a default reset value. Instead, EP relies on the parent/agent IP to wake up the fabric clock through IP's clkreq to EP (agent_clkreq or sbi_sbe_clkreq). The IOSF spec will also reflect this change in upcoming revisions. Early boot IPs typically drive agent_clkreq to 1 during reset, whereas non-early boot IPs should have an agent_clkreq reset value of 0.

Parameter Name	Default Value	Expose at Top Level?	Description
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.

Table 11. Clock, Reset, and DFx 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 strapped.
agent_side_rst_b_sync	Input	For internal use with sbendpoint only. Tie this input low.
sbi_sbe_clkreq	Input	See section 16.9, Design for Clock Gating.
sbi_sbe_idle	Input	See section 16.9, Design for Clock Gating.
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_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 details, see section 16.11, Design for Power Gating.
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 Specification for clock requests.
sbe_sbi_clkreq	Output	For details, see section 16.11, Design for Power Gating.
sbe_sbi_idle	Output	
sbe_sbi_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.

Signal Name	Type	Description
sbe_sbi_comp_exp	Output	Indicates when the IOSF interface has outstanding completions yet to be returned from the agent. The counter will max out at 31 outstanding completions before saturating and does not track based on message contents. Disabled when EXPECTED_COMPLETIONS_COUNTER is 1.
sbe_sbi_parity_err_out	Output	Parity check error
cgctrl_idlecnt[7:0]	Input	For details, see section 16.8, Configuration and Override Inputs.
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 usyncselect is 1 (and the USYNC_ENABLE parameter is set to 1), 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. Agent clock should be free running and not gated, when EP is in USYNC MODE. For details, see section 16.10, Design for Deterministic Clock Crossing.
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. This input is ignored when UNIQUE_EXT_HEADERS is set to 1.
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.
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.

Signal Name	Type	Description
visa_*	Output	VISA candidate signals. See "VISA/DFx Structures" in section 16.7, VISA / DFx Structures.
jta_clkgate_ovrd	Input	See "Configuration and Override Inputs" in Section 16.8, Configuration and Override Inputs.
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	In	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]	In	Unused signals, provided as placeholders for VISA insertion flow. Drive them using 'd0.
avisa_data_out[N:0] avisa_clk_out[N:0]	Out	Unused signals, provided as placeholders for VISA insertion flow.

Table 12. Sideband Channel Target / Master Port Signals

Signal Name	Type	Description
tpayload[2N*8-1:0]	Input	The target port message flit.
teom	Input	The target port end-of-message signal
tpcput	Input	The target port posted/completion put signal
tnpput	Input	The target port non-posted put signal
tpccup	Output	The target port posted/completion credit update signal
tnpcup	Output	The target port non-posted credit update signal
mpayload[2N*8-1:0]	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
mnpcup	Input	The master port non-posted credit update signal
tparity	Input	Parity bit associated with the target payload data, tpayload This input is used only when parameter SB_PARITY_REQUIRED=1.

Signal Name	Type	Description
mparity	Output	Parity bit associated with the master payload data, mpayload This output should be used only when parameter SB_PARITY_REQUIRED=1.

Table 13. Target Interface Signals

Signal Name	Type	Description
sbi_sbe_tmsg_pcfree [MAXPCTRGT:0]	Input	When asserted this indicates that the IP block is ready to receive another 4-byte transfer 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.
sbi_sbe_tmsg_npfree [MAXNPTRGT:0]	Input	When asserted, this indicates that the IP block is ready to receive another 4 byte transfer 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.
sbi_sbe_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. 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. 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.
sbe_sbi_tmsg_pcput	Output	When asserted, this indicates a valid 4 byte flit transfer of a posted/completion message from the base endpoint to the IP block on the target interface. The pcput signal only is asserted in response to all pcfree signals asserted and the internal pcirdy signal asserted from the ingress port.
sbe_sbi_tmsg_npput	Output	When asserted, this indicates a valid 4 byte flit transfer of a non-posted message from the base endpoint to the IP block on the target interface. The npput signal only is asserted in response to all npfree signals asserted and the internal npirdy signal asserted and internal npfence signal de-asserted from the ingress port.
sbe_sbi_tmsg_pcvalid sbe_sbi_tmsg_npvalid	Output	When asserted, indicates the respective sbe_sbi_tmsg_*payload is valid. This is provided to allow attached IP to begin processing message before asserting free. Free/put protocol must still be observed.

Signal Name	Type	Description
sbe_sbi_tmsg_npmsgip sbe_sbi_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 respective tmsg_*put is asserted and tmsg_*eom is de-asserted. The tmsg_*msgip signal de-asserts 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.
sbe_sbi_tmsg_pceom	Output	End of message indicator. When asserted, this indicates the last 4 byte transfer of a posted/completion message on the target interface, and is only valid when tmsg_pcput is asserted.
sbe_sbi_tmsg_npeom	Output	End of message indicator. When asserted this indicates the last 4 byte transfer of a non-posted message on the target interface, and is only valid when tmsg_npput is asserted.
sbe_sbi_tmsg_pcpayload [31:0]	Output	Posted/completion message payload; the next 4 bytes of a posted/completion message on the target interface, which is only valid when tmsg_pcput is asserted.
sbe_sbi_tmsg_nppayload [31:0]	Output	Non-posted message payload; the next 4 bytes of a non-posted message on the target interface, which is only valid when tmsg_npput is asserted.
sbe_sbi_tmsg_pccmpl	Output	This signal is asserted when a completion opcode (0x20 or 0x21) is decoded on the tmsg_pcpayload[23:16] and is only valid when tmsg_pcput is asserted. This signal only is asserted on the first 4 byte transfer of a posted/completion message, which is the only time that tmsg_pcpayload[23:16] contains the message opcode. 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.

Table 14. Master Interface Signals

Signal Name	Type	Description
sbi_sbe_mmsg_pceom [MAXPCMSTR:0]	Input	End of message indicator, one bit per master. When asserted this indicates the last 4 byte transfer of a posted/completion message on the master interface, and is only valid when mmsg_pcirdy is asserted.
sbi_sbe_mmsg_npeom [MAXNPMSTR:0]	Input	End of message indicator, one bit per master. When asserted this indicates the last 4 byte transfer of a non-posted message on the master interface, and is only valid when mmsg_npirdy is asserted.
sbi_sbe_mmsg_pcpayload [32*MAXPCMSTR+31:0]	Input	Posted/completion message payload, 32 bits per master; the next 4 bytes of a posted/completion message on the master interface, which is only valid when mmsg_pcirdy is asserted.
sbi_sbe_mmsg_nppayload [32*MAXNPMSTR+31:0]	Input	Non-posted message payload, 32 bits per master; the next 4 bytes of a non-posted message on the master interface, which is only valid when mmsg_npirdy is asserted.

Signal Name	Type	Description
sbi_sbe_mmsg_pcirdy [MAXPCMSTR:0]	Input	When asserted this indicates that the IP block is ready to deliver the next 4 bytes 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 de-assert the cycle after 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_pctrdy is asserted.
sbi_sbe_mmsg_npirdy [MAXNPMSTR:0]	Input	When asserted this indicates that the IP block is ready to deliver the next 4 bytes 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 de-assert the cycle after 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.
sbe_sbi_mmsg_pctrdy	Output	When asserted this indicates that the endpoint is ready to receive another 4 byte transfer of posted/completion message data from the IP block on the master interface.
sbe_sbi_mmsg_nptrdy	Output	When asserted this indicates that the endpoint is ready to receive another 4 byte transfer of non-posted posted message data from the IP block on the master interface.
sbe_sbi_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 de-asserted. The mmsg_pcmsgip signal de-asserts the cycle after the last dword transfer of a posted/completion message on the master interface. This de-assertion occurs the cycle after mmsg_pcsel[x], mmsg_pcirdy[x], mmsg_pctrdy and mmsg_pceom are all asserted.
sbe_sbi_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 de-asserted. The mmsg_npmsgip signal de-asserts the cycle after the last dword transfer of a non-posted message on the master interface. This de-assertion occurs the cycle after mmsg_npsel[x], mmsg_npirdy[x], mmsg_nptrdy and mmsg_npeom are all asserted.
sbe_sbi_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.
sbe_sbi_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.

13.2.2 Block Interface Timing Waveforms

Note: The following diagrams illustrate the interface timing with respect to side_clk. If a clock-crossing endpoint is used, the reference clock is agent_clk.

13.2.2.1 Master Interface – Posted Message with Data

The timing diagram below shows an 8 byte posted message being transferred from the master interface through the endpoint to the sideband channel.

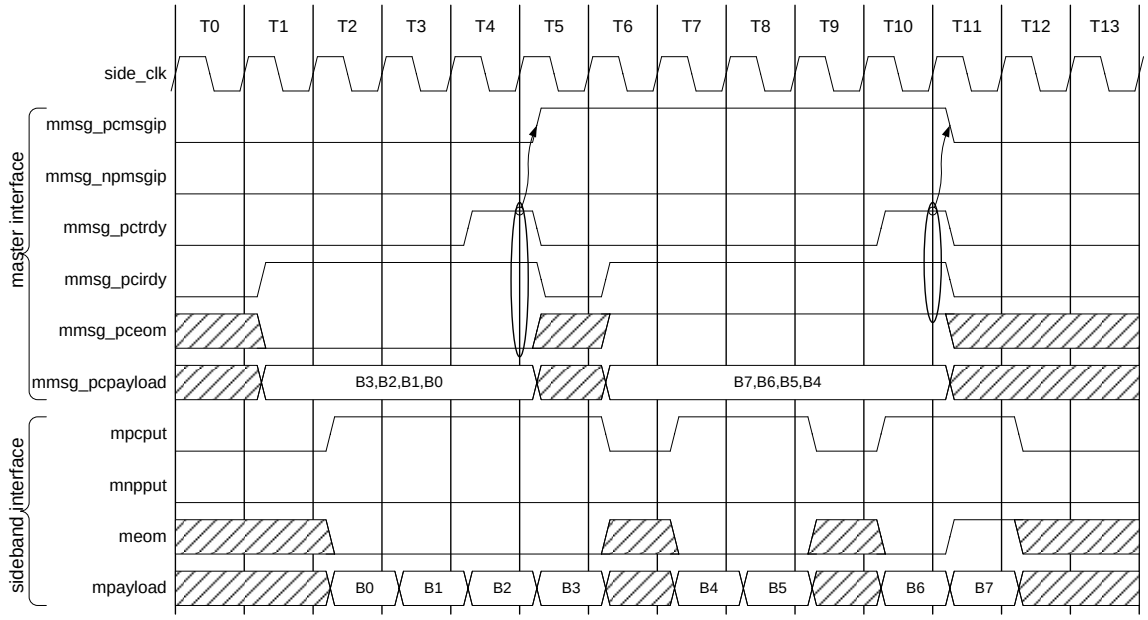
- Cycle T5 shows the message transfer being throttled by the IP block due to the de-assertion of mmsg_irdy.
- Cycle T9 shows the message being throttled by the sideband channel due to insufficient credits.

The minimum latency is one cycle from the first byte of the message being available on the master message interface to the transfer of the first byte of the message on the sideband channel.

The latency could be larger due to message arbitration in the egress port.

Note: This timing diagram illustrates a base endpoint with a single master in the IP block, so the multi-master arbiter is not shown. This could also represent a multi-master interface when the arbiter was already parked on the master issuing the simple message. The multi-master operation is illustrated in section 13.2.2.3, Master Interface – Multi-Master Operation.

Figure 5. Timing Diagram—Master Interface, Posted Message with Data



13.2.2.2 Master Interface – Posted Passing a Non-Posted

In the next timing diagram, a non-posted simple message (4 bytes) is initiated on the master interface. One cycle later, the IP block initiates a posted simple message by asserting the mmsg_pcirdy signal.

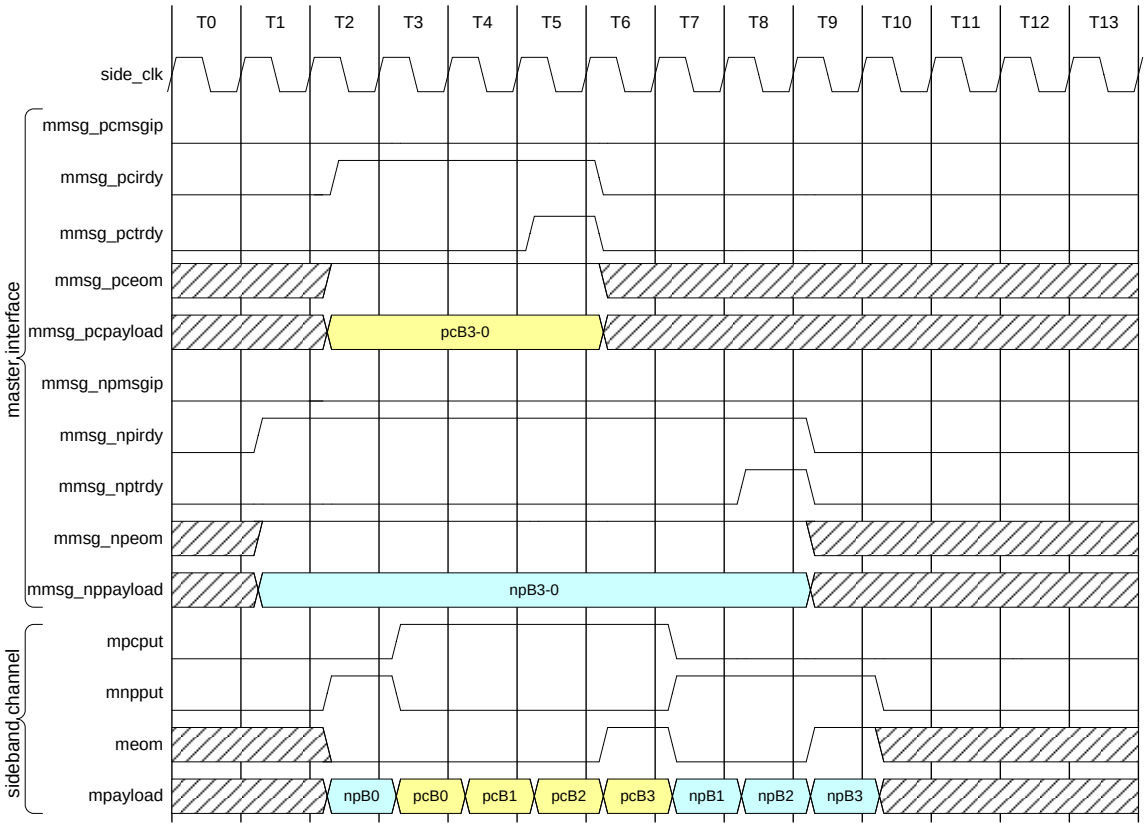
Due to insufficient non-posted credits, the posted message passes the non-posted message in the base endpoint. Once the posted message completes, the endpoint continues to send flits for the non-posted message until it completes.

Note: The posted message was not required to finish before the non-posted message was allowed to continue.

The non-posted message has highest priority because it was initiated before the posted message. The arbitration decision is only based upon available credits. If the non-posted had available credits, then it would have higher priority for the egress port, otherwise the posted is allowed to pass.

Also shown in this timing diagram are the msgip (message in-progress) signals, which remain de-asserted. The signals only assert for messages that are longer than a dword.

Figure 6. Timing Diagram—Master Interface, Posted Passing a Non-Posted



13.2.2.3 Master Interface – Multi-Master Operation

The next timing diagram illustrates the multi-master operation on the master interface.

Figure 7 shows three non-posted masters all requesting to issue a non-posted simple message on the sideband interface.

The egress arbiter is initially selecting master 0 (mmsg_npsel = 001).

In T0, master 1 and 2 both assert their mmsg_irdy signals.

In T1, the arbiter transitions to select master 1, and in T1 thru T4 the non-posted simple message from master 1 is transferred to the egress port and is sent out on the sideband interface during T2 thru T5.

In T4, the egress arbiter in the master interface block receives the nptrdy from the egress port, which is also sent directly to the IP block on the master interface.

Master 1 is the only master monitoring the mmsg_nptrdy signal because master 1 is the selected master (mmsg_npsel = 010).

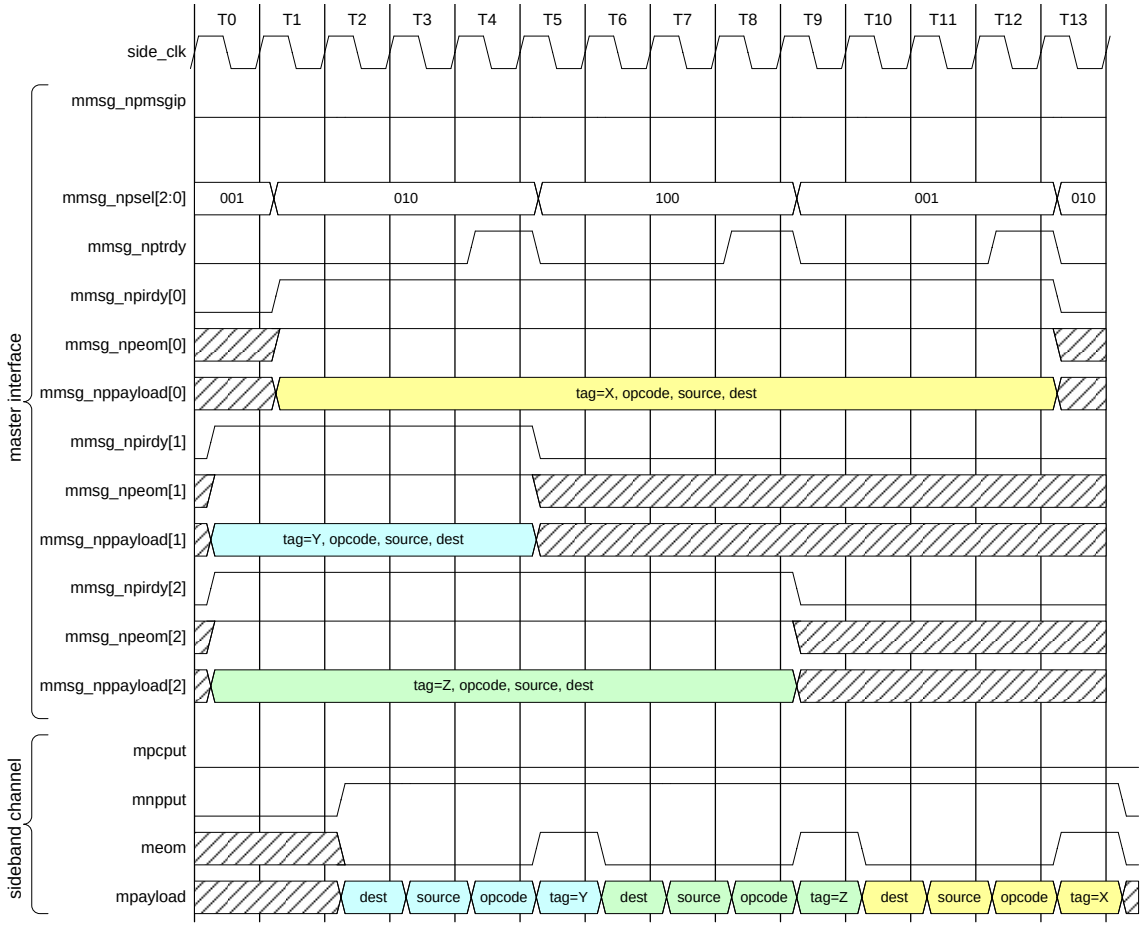
The arbiter transitions to select master 2 in T5, because the end of message transfer occurred at the end of T4.

If master 1 was sending a longer message (not eom), then the arbiter would remain selecting master 1 and the message in-progress signal (mmsg_npmsgip) would have asserted in T5.

This is a requirement because messages of the same type (PC or NP) cannot be interleaved.

Note: In T1 master 0 asserted its mmsg_npirdy signal, which was 1 cycle too late to be visible to the arbiter before switching to the next master. So, master 0 has to wait until both master 1 and master 2 have completed sending their messages.

Figure 7. Timing Diagram—Master Interface, Multi-Master Operation



13.2.2.4 Target Interface—Multi-Target Operation

This timing diagram illustrates the operation of the target interface to the IP block with multiple target agents. In this example, there are two target agents. The first message received (completion with a single dword of data) is claimed by target 0 and the second message (completion without data) is claimed by target 1.

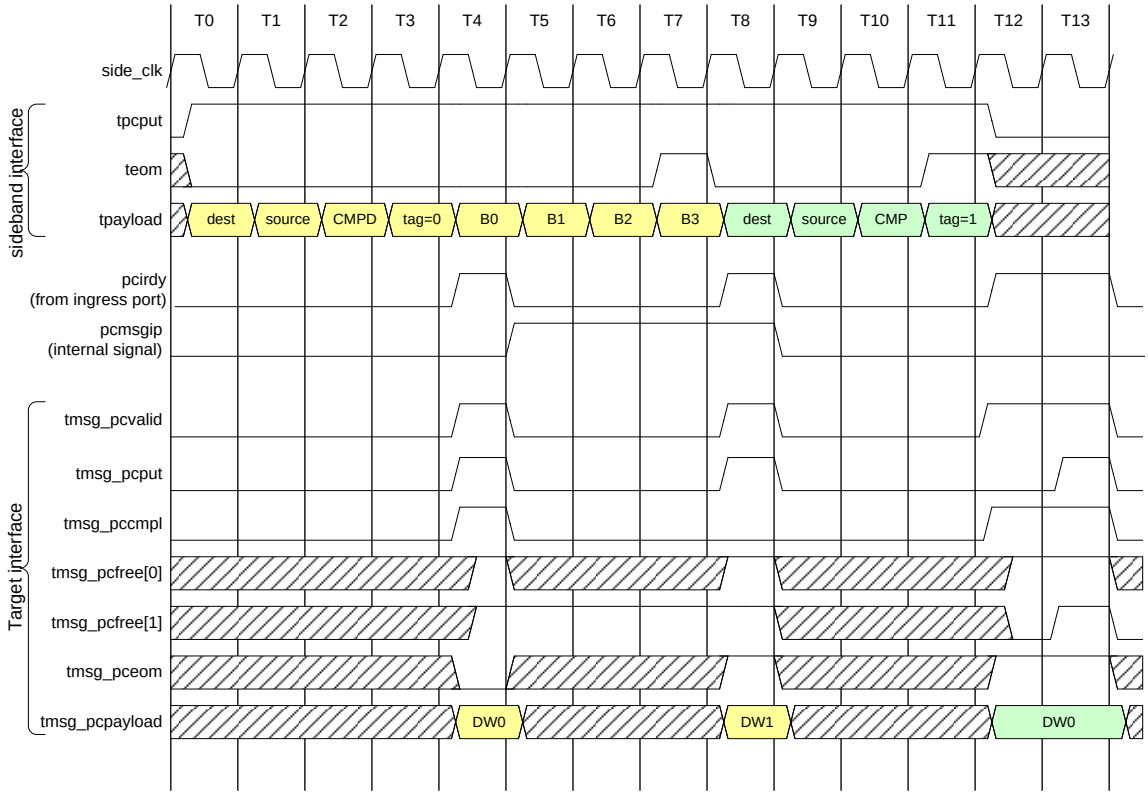
Note: Both targets have to asserted tmsg_pcfree in order for the transfer to take place on the target interface.

In T4, both targets are ready to receive the start of a new posted/completion message, and the pcput is asserted because pcirdy (from the ingress port) and all pcfree signals are asserted. In T5, target 1 asserts tmsg_pcfree[1] until the end of the message, because it has determined that it is not the target of the message. This leaves target 0 in control of the target interface.

In T8, the final dword of the message is transferred (pcput asserted). The second message is transferred in T13 with a 1 cycle stall due to target 1; the internal pcirdy from the ingress port was asserted in T12, but pcfree from target 1 was not asserted until T13.

For posted/completion messages, there is no way for the base endpoint to know which target is claiming the messages. It is possible that no targets claim the message, but all targets are responsible for ensuring that the target interface makes forward progress by eventually asserting their pcfree signals. Unclaimed posted/completion messages are silently dropped on the target interface.

Figure 8. Timing Diagram—Target Interface, Multi-Target Operation



13.2.2.5 Non-posted Target Message with a Master Completion

This timing diagram illustrates the responsibilities of the IP block target agent during the reception of a non-posted message and the generation of a completion. The timing diagram shows a 2 dword non-posted message received as a target and a completion without data being generated as a master. The message is claimed by the target in T8.

The claim signal is sampled by the endpoint whenever tmsg_npput is asserted at the positive clock edge.

The target agent must store (at least) the destination port ID, the source port ID and the tag from the non-posted message in order to generate the completion.

When the completion is mastered, the target agent simply swaps the destination and source port IDs, using the destination port ID received in the non-posted message as the source port ID of the completion and the source port ID received in the non-posted message as the destination port ID of the completion. The tag returned with the completion must be equal to the tag received in the non-posted message.

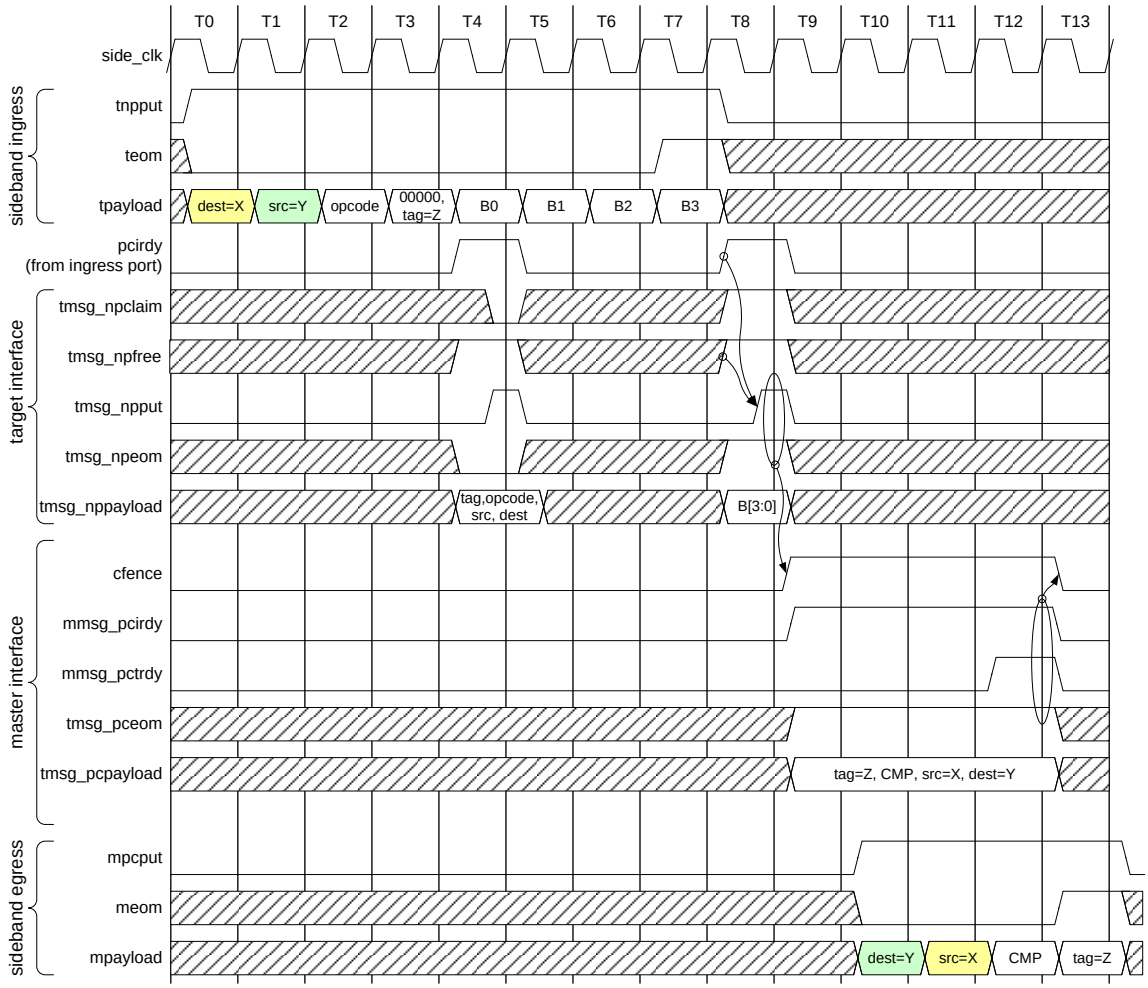
In addition, the target agent must wait until the final 4 bytes of the non-posted message are consumed on the target interface before initiating the completion on the master interface. The timing diagram below shows this occurring in the cycle after the EOM transfer (tmsg_npput/npeom both asserted).

Note: This timing diagram looks identical for unclaimed non-posted messages, but for that case, the completion is generated internally from the target interface block (SBCTRGT) to the master interface block (SBCMSTR).

Of note in this diagram is the behavior of the optional completion fence feature, which is illustrated by the cfence signal. The fence is asserted whenever the EOM payload of a non-posted transaction is accepted by the attached IP.

This fence internally gates the npfree/npput handshake from transferring any subsequent nppayload until the EOM transfer of a completion is accepted by the master interface. The fencing signal is not visible to the attached IP but is included here for illustrative purposes.

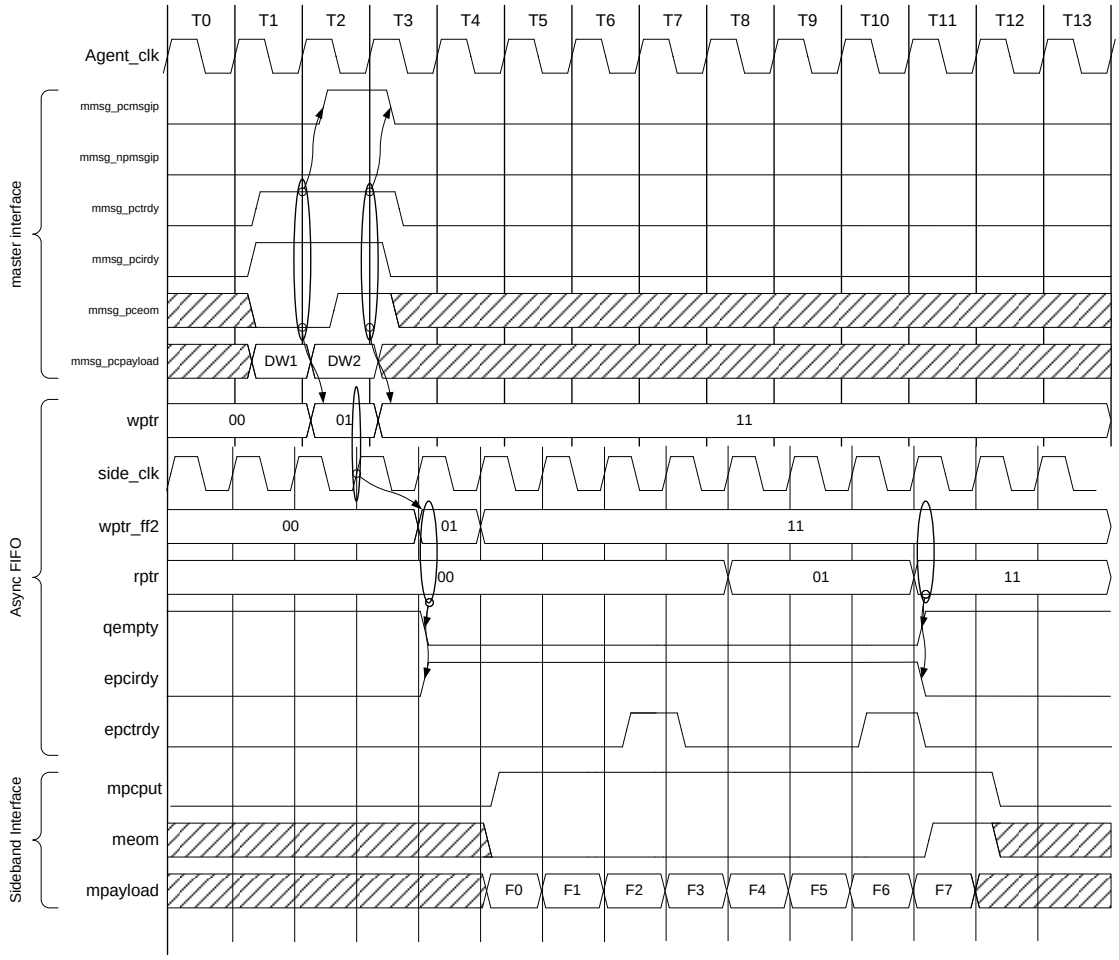
Figure 9. Timing Diagram—Non-posted Target Message with a Master Completion



13.2.2.6 Master Interface—Posted Message with Asynchronous Crossing

This diagram illustrates the impact of clock crossing on master transactions. In this scenario, transactions from a slower agent_clk transition to a faster side_clk which is running at a noninteger frequency multiplier. The diagram assumes there is no metastability encountered in the crossing of FIFO pointers from one domain to the other.

Figure 10. Timing Diagram—Master Interface, Posted message with Asynchronous Crossing



Determining the correct asynchronous queue size for a particular application is not trivial. A number of considerations must be taken into account:

1. Ratio of agent clock to side clock frequencies.
2. IOSF payload size.
3. Ingress/Egress bandwidth requirement.
4. Transaction type and length to be optimized.
5. Fabric-wide implications of agent ingress/egress bandwidth restrictions.

If bandwidth is not a concern, then a minimum size FIFO (two entries) is sufficient to enable clock crossing at the endpoint level.

The problem is: depending on the situation, this can have far-ranging implications for the entire fabric. For example, if the IOSF clock is running much faster (4x or higher) than the agent clock, any transaction sent by the fabric to the agent which is greater than the port

ingress queue and 2DW FIFO queue is blocked while waiting for the slow agent clock to acknowledge and accept the transactions.

During this time, due to the fact that transactions of the same flow class cannot be interleaved in sideband, the initiating endpoint is blocked from sending any other transaction of the same flow class to a different endpoint. If the transaction is posted, it can also be blocked from sending non-posted transactions to any endpoint due to the NP fencing mechanism.

Furthermore, if the endpoints are connected across a fabric that is comprised of multiple routers, the router to router connections might also be blocked from sending other transactions of the same flow class to different endpoints. This is an example of the fabric-wide implications of an agent’s bandwidth restriction.

14 Clock and Reset Scheme

The IOSF sideband endpoint collateral contains its own unique (active low) reset signal which is named side_rst_b. This signal is only used by the sideband channel routers and the endpoint. At some point within the IP block, the logic might cross from the sideband clock/reset domain into a different clock/reset domain. The sideband endpoint is expected to function normally while the IP block is in reset and/or clock gated.

This condition should not prevent forward progress of messages on the sideband interface. Non-posted messages targeting the IP block that is in reset should not prevent the completion from occurring or there must be some mechanism that prevents the non-posted messages from targeting the IP blocks in reset. Posted messages must also make forward progress while the IP block is in reset (either dropped or not allowed to occur). The mechanism to guarantee this is design specific and the responsibility of the IP designer. *In general, tying the appropriate free signals high and the npclaim signal low during agent reset accomplishes this requirement.*

When instantiating the endpoint with clock crossing enabled (ASYNCENDPT=1), care must be taken with regard to the reset structure of the IP and endpoint. In this scenario, the endpoint continues to have a single reset input, side_rst_b. Triggering this signal asynchronously resets both the side_clk and agent_clk portions of the endpoint. The de-assertion of side_rst_b within the agent_clk domain is synchronized to agent_clk within the endpoint.

Generally speaking, resetting a sideband endpoint must be considered within the context of the entire sideband fabric. Resetting an endpoint independently of the fabric can pose a number of potentially unrecoverable problems, particularly if the endpoint or fabric is in the middle of sending or receiving transactions. The same situation can occur with a clock crossing endpoint wherein the IP's logic attached to the agent_clk interfaces is reset in the midst of receiving or transmitting a transaction.

If independent reset of endpoints or IP logic attached to the endpoint is required, care must be taken to avoid doing so during a transaction. The overall resolution of reset structure for any SoC is beyond the scope of this document and beyond the scope of the endpoint design to address.

15 Area Estimates

Note: Data shown for IOSF payload width of 8 bits. Differing IOSF payload widths yield different results. These values are also out of date and should only be used for rough estimates.

Parameters Used	Default Values
MAXPLDBIT	7
NPQUEUEDEPTH	4
PCQUEUEDEPTH	4
LATCHQUEUES	0
MAXPCTRGT	0
MAXNPTRGT	0
MAXPCMSTR	0
MAXNPTRGT	0
ASYNCENDPT	0
RX_EXT_HEADER_SUPPORT	0
TX_EXT_HEADER_SUPPORT	0
DISABLE_COMPLETION_FENCING	0

Disclaimer

While these numbers can provide the reader a rough estimate for the area required by a 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), 250 MHz side_clk, 35% period I/O delay, *full scan insertion*

Estimated "NAND2 equivalent" area: 4007 gates

Estimated Total Cell Count: 1461

Note:

1. Altering QUEUEDEPTH has a moderate impact to gate count, as each increment in queuedepth adds (IOSF payload * 1) bits total to the queues, or ~80 gates for an 8 bit IOSF payload endpoint.

2. Enabling RX_EXT_HEADER_SUPPORT eliminates ~64 flops, or ~750 gates.

3. Enabling ASYNCENDPT adds ~64 flops for each queue entry for both ingress and egress queue; for example, the minimum increment is 64 flops * 2 (minimum FIFO depth) * 2 fifos =256 flops, plus associated queue management logic.

16 Integration Details

16.1 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 by two new parameters: PIPEISMS for fabric ISM inputs and PIPEINPS for non-ISM inputs (tpayload, t*put, teom, and m*cup). Setting these parameters to1 adds a pipeline stage to the specific inputs and increases the latency by one cycle at target and master interfaces. Also, pipeline stages should be factored into streaming requirements, which may require increasing NP/PC Ingress Queue depths to avoid bubble between transactions.

Note: The pipeline option for non-ISM inputs can only be added if ISM inputs are enabled as well. IOSF compliance monitors do not like the pipeline stages in the IP and the options should be avoided entirely. If possible, the pipeline stages should be added between the endpoint and router at the chip level.

16.2 Ingress Latency

These calculations assume that there is no credit shortfall and that the attached agent does not delay accepting the incoming data. For determining the correct QUEUEDEPTH parameter to ensure there is no credit shortfall, see section 16.4, QUEUEDEPTH Parameter Setting and Bandwidth.

For a synchronous endpoint, using no pipeline stage on the fabric inputs (parameters PIPEISMS and PIPEINPS are set to ’d0), the ingress latency calculation is simply a matter of determining how many IOSF sideband flits are required to generate a full 32-bit agent-side payload flit.

Figure 8 and Figure 9 both illustrate the best-case latency for an endpoint with an 8-bit IOSF payload. Considering the clock cycle in which the first flit is presented to the IOSF target interface (t*put asserted) as clock 0, the full 32 bit agent-side payload is available in clock 4. Using a 16 bit IOSF payload reduces latency by a factor of two. For a synchronous endpoint using pipeline stage at fabric inputs (parameters PIPEISMS and PIPEINPS are set to ’d1), one more clock cycle is added to the overall latency.

For a clock crossing endpoint using no pipeline stage at fabric inputs, the ingress latency is increased due to the clock-crossing FIFOs between the side_clk and agent_clk domain. Starting from the same reference point as above, the full 32-bit payload becomes available to the asynchronous ingress FIFO at clock 4. The payload is stored in the FIFO during clock 5; this also alters the FIFO write pointers.

At this point, the new FIFO write pointer must be crossed to the agent clock domain, which takes a minimum of 2 agent clocks. Once the new pointer has crossed to the agent clock domain, the payload is immediately available for the free/put handshake and transfer. Therefore the latency in this case is 5 side clocks plus at least 2 agent clocks. For a clock crossing endpoint pipeline stage at fabric inputs, the latency is (5+1) side clocks plus at least 2 agent clocks.

16.3 Egress Latency

Just as for ingress latency, these calculations assume there is no credit shortfall. For multi-master configurations, the calculations also assume there is no contention for the egress

resources with other masters and that the egress arbiter is selecting the master which is sending the transaction. If the arbiter is not selecting the master, there is at least one extra clock of latency to these calculations for re-arbitration; for synchronous endpoints, this is side_clk, for clock crossing, it is agent_clk.

Transaction egress latency with a synchronous endpoint is a straightforward calculation because the egress data is directed through a single flop output stage before it is available on the IOSF interface.

Figure 5 shows the associated timing diagram. There is only one side clock of latency from the assertion of irdy to the first flit output to the sideband fabric.

Transaction egress latency with a clock crossing endpoint is almost as straightforward as the synchronous example. In this case, the agent's assertion of irdy loads the payload into the FIFO at the end of clock 0. This also changes the FIFO write pointers, which requires at least two side clocks to transfer to the side clock domain. After this is accomplished, there is one side clock required to load the egress flop. In total, this is one agent clock plus at least three side clocks.

Egress latency is not sensitive to IOSF payload size.

16.4 QUEUEDEPTH Parameter Setting and Bandwidth

The NP/PCQUEUEDEPTH parameters are provided to allow the designer integrating the endpoint to prioritize area vs. bandwidth. In many cases, ingress bandwidth is not a significant concern, so some area can be saved by using a smaller ingress queue depth. Each increment to the queue depth increases the size of the endpoint by 1 IOSF payload flops (typically 8), plus a small amount of queue management control logic. The endpoint allows for independent sizing of NP and PC queues in order to optimize area and bandwidth for particular applications and traffic patterns.

The number of credits advertised to the fabric is always the same as *QUEUEDEPTH. However, due to the payload width conversion from IOSF payload to agent payload, the ingress queue typically has more storage elements than indicated by the queue depth. For example, an 8-bit, IOSF payload endpoint has QUEUEDEPTH+3 flits of storage. The extra 3 flits are required to accumulate at least 32 bits to match the agent payload size. These extra storage elements are called the accumulator.

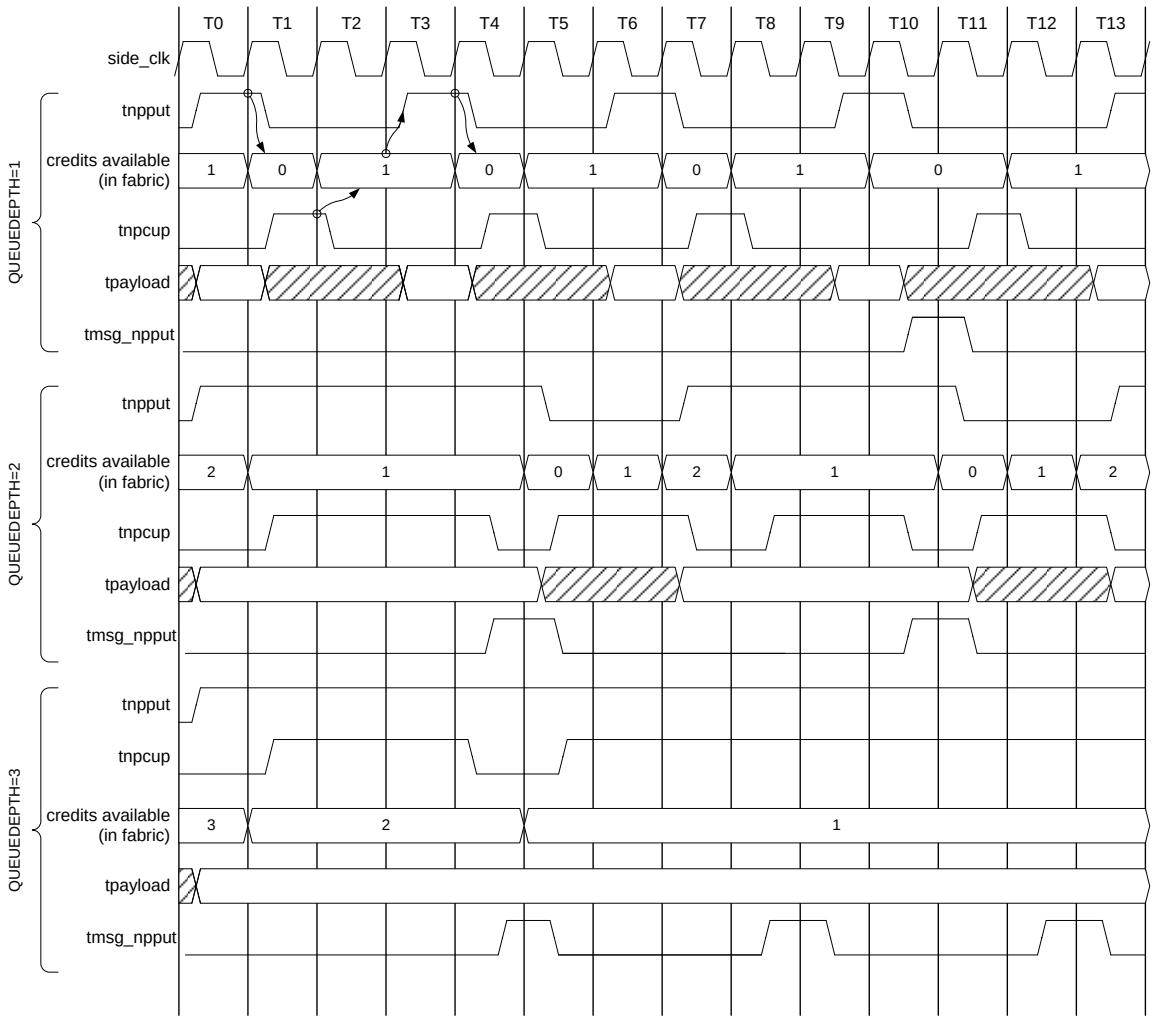
As flits are sent to the endpoint, they are loaded into the accumulator first. Because the accumulator storage elements were not part of the advertised credits, a credit is returned on the clock subsequent to an accumulator element is consumed. Once the accumulator is full, credits are returned only after the payload is transferred out of the ingress queue and to the agent.

In addition to endpoint queue and credit-handling architecture, assuring bandwidth is also dependent upon the fabric credit management architecture. Per IOSF spec, a put in clock N must be contingent upon a credit in clock N-1 or before. There cannot be a combinational path such that a credit update in clock N can allow a put in clock N. However, even using this one cycle delay can lead to long timing paths in the fabric by requiring the use of a forwarded credit counter value and not a flopped credit counter value. The typical fabrics delivered by SEG are configured to use only the flopped credit counter state in order to avoid this timing path. Therefore, an agent's credit update in clock N would be available for use by the fabric in clock N+2.

Figure 10 illustrates the implications of QUEUEDEPTH on bandwidth assuming that the attached fabric uses the flopped credit counter state and that the agent is always asserting the free signals. This diagram is valid only for synchronous instantiations (for example, no clock crossing) and no pipeline stages are added to fabric inputs to endpoint side.

Note: Because fabric can assert 'put' only after two cycle later of agent's credit update, (1+2) NP/PC Ingress Queue Depth is required to maintain put to be asserted all times; for example, for continuous data streaming. If a synchronous endpoint has pipelined fabric inputs, NP/PC Ingress Queue Depth is increased to (3+1) to store payload/eom one more cycle to maintain streaming.

Figure 11. QUEUEDEPTH Effect on Bandwidth



16.5 Endpoint Interface Completion Fencing Logic

The IOSF 1.0 endpoint release has existing completion fencing feature (was added on IOSF 0.9) to the endpoint. The purpose of this fence is to ensure that all completions are returned in the correct order. Without the fencing logic, the endpoint might violate the IOSF completion ordering rules; for example, if two NP transactions are received with the agent claiming the first but not claiming the second, the automatically generated UR completion can be sent before the agent sends the completion for the first NP transaction. There are many other similar scenarios possible.

The endpoint asserts the completion fence upon sending the EOM payload of a non-posted transaction to the agent (tmsg_npput and tmsg_eom asserted). At this point, no further

npputs are possible until the EOM payload of a transaction with the completion opcode is accepted.

The endpoint tracks the opcode of all new posted/completion transactions to detect the completion opcode, then clears the fence when the EOM payload is accepted (mmsg_pcput and mmsg_eom asserted along with the 'completion in progress' internal status). The same mechanism applies to UR completions generated for unclaimed transactions.

16.6 Extended Error Handling

This feature effectively limits the agent to process only one NP transaction at a time. For most agents this is not an issue, however, there can be complex agents that need to queue many NP transactions. In this case, the agent designer can disable the fence by setting the DISABLE_COMPLETION_FENCING parameter to 1. Whenever this parameter is set to a nonzero value, the agent *must* claim all NP transactions, even those which eventually result in a UR completion, and correctly order the completions. Relying on the unclaimed UR logic within the endpoint in this scenario can lead to completion ordering violations.

16.6.1 Unique Extended Header Support

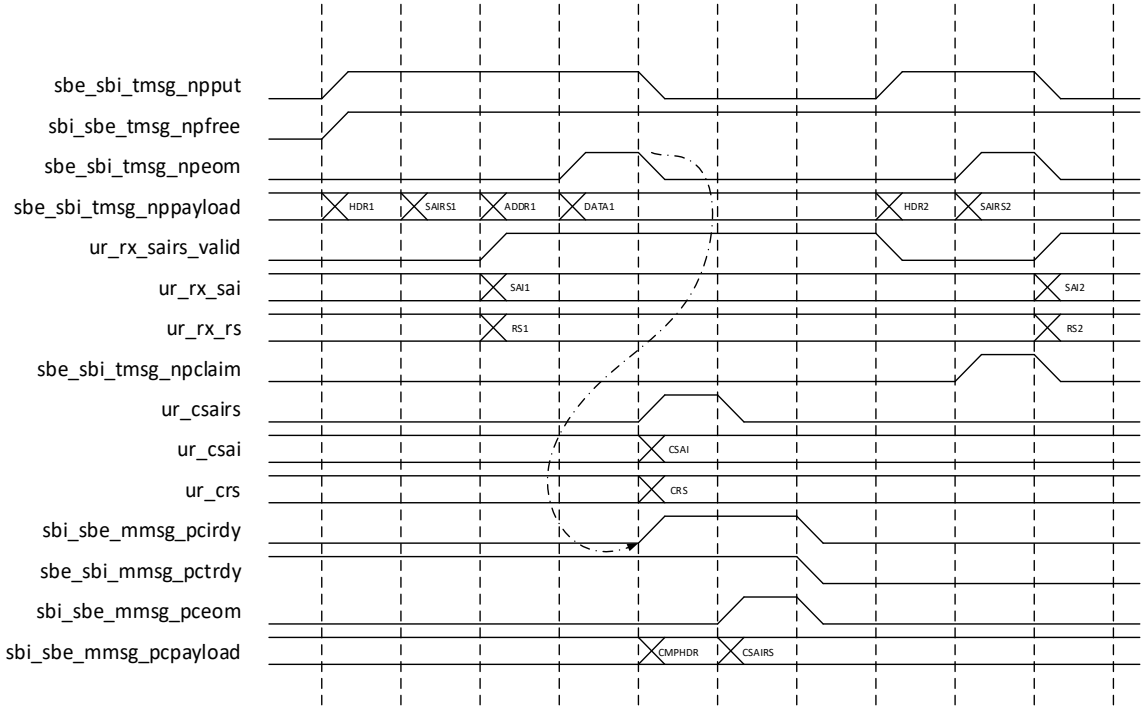
To address issues with design limitations and implementation confusion, each instance in the endpoint that is involved in extended headers now has its own, unique, extended header structure. With this structure there will be a valid bit associated with each sets of fields for each extended header type. All fields will only be valid when UNIQUE_EXT_HEADERS is a one indicating unique extended header mode is desired. While in this state, tx_ext_headers and rx_ext_headers will be ignored or driven to 0.

For example, the SAIRS extended header field is broken out into two parts: sai and rs. A valid flag would be used to indicate when the sai and rs fields are valid. If used to generate a message the valid flag will indicate when an extended header should be inserted. If used to receive a message, the valid flag indicates when an SAIRS extended header was received and the fields are valid.

Figure 12, Unsupported Response Unique Extended Headers, illustrates how the target module provides extended header information about the in-progress non-posted message and how extended headers are added to the generated unsupported response message. The message ingressing non-posted message at time slices 1 through 5 is an unclaimed message and is completed by the unsupported response at time slices 5 through 7. The SAIRS extended header starting at time 2 is latches in and made available at time 3. The SAIRS valid flag is not dropped until time 8 where a new ingressing non-posted message is started. Time 10 shows that the SAIRS field is always captured the cycle after and reguardless if the message was actually claimed.

The generated unsupported response message captures the valid bits the cycle following EOM at time 6. The valid bit is no longer required after this cycle but the ur_csai and ur_crs fields must remain stable until the message has completed. It is safest to assume that the beginning of the new non-posted message means that the target message modules has completed the unclaimed non-posted message.

Figure 12. Unsupported Response Unique Extended Headers



16.6.2 Legacy Extended Header Support

The IOSF 0.9 and newer specification adds the concept of extended header transactions to the IOSF protocol. This optional feature is implemented by setting the corresponding "EH" bit within the transaction header and appending N number of "extended headers" to all sideband transactions. For complete details about these transactions, see Section 3.3.2.2 in the IOSF Specification 1.0.

There is a complication with this change, in that the additional DW appended to the transaction makes the extended header transactions appear to be invalid transactions to all agents designed to the IOSF 0.83 and previous specifications.

To resolve this interoperability problem, IOSF 0.9 and newer endpoint designs implement an "RX_EXT_HEADER_SUPPORT" parameter. If the attached agent design is not aware of extended header transactions, this parameter should be set to 0. If the agent is aware of these transactions, the parameter should be set to 1. The function of this parameter with varying received messages is described in Table 15, Received Transactions and Descriptions.

Table 15. Received Transactions and Descriptions

RX_EXT_ HEADER_SUPPORT Setting	Received Transaction	Description
0	EH NOT Present	Transaction is passed to agent without modification.
0	EH Present	All extended headers dropped from message. The EH bit is cleared from the standard header. No other modification is made.
1	EH NOT Present	Transaction is passed to agent without modification.
1	EH Present	Transaction is passed to agent without modification.

In addition to the above, there is a TX_EXT_HEADER_SUPPORT parameter that has an effect on endpoint-generated transactions. *For the Base Endpoint, this is limited to the completions generated for an unclaimed NP transaction*.

If TX_EXT_HEADER_SUPPORT = 0, the generated completion is a standard 1DW completion without data with the completion status set to UR. The EH bit in the header is set to 0.

If TX_EXT_HEADER_SUPPORT = 1, the generated completion has N number of extended headers added, where N is determined by the NUM_TX_EXT_HEADERS parameter. The headers attached come from the tx_ext_headers[NUM_TX_EXT_HEADERS:0] input.

Extended headers are specified by an input instead of a parameter in order to provide additional flexibility to the agent design for cases in which the header value needs to change. If this is not required, the input might be tied to a specific value.

If the value can be changed, care must be taken to ensure that the input is not changed during the transmission of a transaction. Accomplishing this is design specific, however, one potential solution is to ensure sbe_sbi_idle is asserted before changing this value.

If RX_EXT_HEADER = 0, implying the receiving agents do not support extended headers, the fabric should not send any payload with extended header bit set. This is because, inside the target module drops the payload because the RX_EXT_HEADER parameter is set to 0, which misaligns the put/eom and payload.

Note: For all transactions initiated on the master message interface by the agent, the agent design is responsible for setting the EH bit and, if appropriate, driving the additional extended headers. The endpoint does not append the extended headers to these messages under any circumstance.

16.7 VISA / DFx Structures

The IOSF 1.0 endpoint release alters the debug outputs that were available in previous endpoints. The purpose of this is to provide a prioritized grouping of bits for easier selection of debug signals in cases where debug resources are limited. Another goal is to provide a consistent assignment of debug bits regardless of parameters – previously, some of the debug bits changed role depending upon the parameters used.

The VISA candidate signals are organized into two tiers; tier 1 are high-priority debug signals, while tier 2 are lower priority. Users are responsible for determining what level of VISA debug capability is required and selecting signals from these tiers as appropriate. It is recommended to include at least the high priority signals (tier1). If debug resources are available, inclusion of tier2 signals should be considered.

The signals are further categorized into their source clock domain. Outputs with the "_sb" suffix are synchronous to the IOSF side_clk. Outputs with the "_ag" suffix are synchronous to the agent clock. When the endpoint is instantiated without clock crossing, these two clocks are both side_clk.

Table 16. Output/Clock Domain

Output	Clock Domain	Notes
visa_port_tier1_sb	side_clk	
visa_port_tier2_sb		
visa_fifo_tier1_sb		Used only with ASYNCENDPT = 1
visa_fifo_tier2_sb		Used only with ASYNCENDPT = 1
visa_agent_tier1_ag		

Output	Clock Domain	Notes
visa_agent_tier2_ag	agent_clk (if clock crossing), otherwise side_clk	
visa_fifo_tier1_ag		Used only with ASYNCENDPT = 1
visa_fifo_tier2_ag		Used only with ASYNCENDPT = 1

If 8 bit VISA lanes are desired, these outputs are organized such that they should be split symmetrically; for example, bits [15:8] should go to one lane assignment while bits [7:0] should go to the other. If 32 bit VISA lanes are desired, tier1 and tier2 signals can be concatenated into one 32 bit grouping.

16.8 Configuration and Override Inputs

The endpoint provides a number of configuration and override inputs which are primarily used to control the internal clock gating and ISM behavior. These are provided to enhance post-silicon debug, particularly if clock gating or ISM issues are suspected of breaking sideband functionality. Each of these signals is described in Table 17.

These signals are intended to be either tied to specific values or driven by a configuration / JTAG register. Some signals are re-synchronized to the side_clk domain as shown in Table 17. It is recommended to tie off the values unless there are chip specific requirements to make these inputs accessible. The signals are expected to be synchronous to the side_clk domain. If synchronization is required then these signals must be synchronized before entering the endpoint.

Generally these values can be changed at any time; however, it is safest to do so when the ISM is idle. Changing the idle count value if the counter is already incrementing can result in setting a new value below the current count, in which case, the counter must overflow and count up to the new trip point in order to transition to IDLE_REQ.

Table 17. Inputs

Input	Resynchronized to side_clk?	Description
cgctrl_idlecnt[7:0]	Yes	Idle count limit for ISM which is determines when the block should transition to IDLE_REQ. Recommended value = 8'd16. A value of zero breaks the design resulting in an unresponsive endpoint. **Note:** Connect to TAP at SOC
cgctrl_clkgaten	Yes	=1 clock gate enable; =0 clock gate disable and hold ISM in ACTIVE. **Note:** To disable local clock gating with no side effects, use the cgctrl_clkgatedef input **Note:** When set to 0, the IOSF Rule 2 for entering IDLE is violated ("The agent ISM must transition from ACTIVE to IDLE_REQ between clock 17 and 32…"), and a compliance error is triggered in simulation When set to 1, enables the ISM to leave ACTIVE and gates the side_clk if in the IDLE state. When set to 0, the ISM never leaves ACTIVE once it gets to that state and the clock is never gated. Recommended value = 1. **Note:** Connect to TAP at SOC

Input	Resynchronized to side_clk?	Description
cgctrl_clkgatedef	Yes	=1 disable/defeature local clock gating; =0 enable local clock gating **Note:** To disable local clock gating in conjuction with holding the ISM in ACTIVE, use the cgctrl_clkgaten input When set to 1, disables side_clk gating within the endpoint when ISM is IDLE. Set to 0 for normal operation. Recommended value = 0. **Note:** Connect to TAP at SOC
jta_clkgate_ovrd	Yes	When set to 1, forces internal gating of side_clk (stops it) Set to 0 for normal operation. **Note:** In order to defeature clock gating, use signal cgctrl_clkgatedef
jta_force_clkreq	No	When set to 1, forces clkreq output signal to be asserted. Set to '0 for normal operation.
jta_force_idle	Yes	When set to 1, forces ISM to consider the agent as idle regardless of other inputs. This forces the ISM to transition to IDLE (following the correct transition path) and stay until the force is released. Set to '0 for normal operation.
jta_force_notidle	Yes	When set to 1, forces ISM to consider the agent as not idle (overrides jta_force_idle). This forces the ISM to transition to ACTIVE (following the correct transition path) and stay until the force is removed. Set to '0 for normal operation.
jta_force_creditreq	Yes	When set to 1, causes ISM to transition into CREDIT_REQ state from IDLE. Set to 0 for normal operation.

16.9 Design for Clock Gating

The endpoint provides a number of IO signals intended for use in trunk (side_clk) and agent clock gating.

It is required that agents have no one going glitches on sbi_sbe_clkreq. It is highly suggested that agents use a flopped version of sbi_sbe_clkreq to avoid intensive logic on reset issues in the backend. All agents must ensure that sbi_sbe_clkreq is asserted before mastering any messages into the endpoint.In other words, Sbi_sbe_clkreq must be held high for at least one clock cycle before mastering messages. sbi_sbe_clkreq must not be asserted unless sbe_sbi_clkvalid is already 0 which means there is not active clock request. Or when sbe_sbi_clkvalid is 1 and sbe_sbi_idle is 0 there is a message being mastered into the agent and it is safe to assert sbi_sbe_clkreq without causing any glitches if sbe_sbi_clkvalid is in the process of falling after side_clkreq was going to be dropped. Sbi_sbe_clkreq should not be de-asserted until after sbe_sbi_clkvalid is 1 and sbe_sbi_idle is 0 to ensure that the message has entered into the egress path of the endpoint to avoid glitches and undetected clock requests.

It is required agents with fully synchronous endpoints have no zero going glitches on sbi_sbe_idle. It is highly suggested that these agents use a flopped version of sbi_sbe_idle. If at all possible sbi_sbe_idle should be a constant one if eairly ISM IDLE => ACTIVE_REQ => ACTIVE is not required. sbi_sbe_idle enters an unprotected path of the clock request logic which could be prone to creating a glitch if not handled appropriately. This path exists for legacy agent support when sbi_sbe_clkreq is not used. sbe_sbi_idle must not be asserted unless sbe_sbi_clkvalid is 0 or when sbe_sbi_clkvalid is 1 and sbe_sbi_idle is 0 or when sbi_sbe_clkreq is 1 and sbe_clk_valid is 1. Because this signal bypasses all protections an internal glitch can be created if these rules are not followed.

It is highly suggsted that agents use a flopped version of *irdy on all mastering interfaces. These inputs will be used to request the clock in-case sbi_sbe_clkreq is missed by any IPs, especially in legacy cases. These signals cannot have any one-going glitches. Glitches can be masked if sbi_sbe_clkreq is asserted first.

For the IOSF 1.0 endpoint, if the fabric initiates a transaction to a synchronous agent that responds very slowly to incoming transactions, the endpoint puts the message in the ingress queue while waiting IP accepting message. If there is no activity on interface during this period, the agent ISM can move from ACTIVE => IDLE_REQ => IDLE as long as side_clk remains enabled during this time until the transaction is complete.

Note: Because the agent is responsible for the delay, it is the agent's responsibly to continue requesting the clock until the transaction completes.The sbi_sbe_idle signal may be driven high while waiting for the agent to complete the transaction to prevent keeping the ISM in IDLE to allow clock gating in the fabric.

Note: For customers that use the base endpoint and use the target register access module used in the endpoint wrapper (sbendpoint); it is expected that the inverse of the register access module's idle signal be OR'd with sbi_sbe_clkreq.

This needs to be done for slow synchronous agent using customer specific Register Access Target Agent to ensure that side_clk is available to Register Access Target Agent to complete the transaction. Here, target agent IP gets effectively 14 clock cycles to assert sbi_sbe_clkreq (2 cycles are for synchronizing asynchronous sbi_sbe_ckreq signal) after fabric initiates a transaction for target agent. In case of a very slow asynchronous agent, agent_clk should be available at the period specified above (even agent ISM moves to IDLE) to complete transaction either from Target Interface (SBETRGT) or Register Access Target Agent (SBETRGTREG).

If the master agent is to initiate its own transactions, it must assert the sbi_sbe_clkreq signal and de-assert sbi_sbe_idle signal whenever it has a transaction to be sent. Asserting sbi_sbe_clkreq causes side_clkreq to assert which eventually enables the side_clk. De-asserting sbi_sbe_idle indicates that the agent is no longer idle and it intends to send a transaction. The endpoint's agent ISM responds with a transition from IDLE to ACTIVE_REQ.

The signals associated with trunk clock gating are shown in Table 18.

Note: The signal sbi_sbe_idle can be generated by inverting the mmsg_*irdy signals to wake up the agent ISM through the sequence (IDLE=>ACTIVE_REQ=>ACTIVE) if needed. Also, in this case, this signal is driven low only when the agent IP intends to initiate a transaction to send.

Table 18. Trunk Clock Gating Signals

Signal	Clock Domain		Purpose
—	**ASYNCENDPT=0**	**ASYNCENDPT=1**	—
sbi_sbe_ 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. clkreq should be de-asserted after the base endpoint reports that it is no longer idle. This is to avoid metastability issues.
sbe_sbi_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.

Signal	Clock Domain		Purpose
sbi_sbe_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. Idle should be de-asserted after the agent has sent in the message or will cause thrashing on the agent ISM.

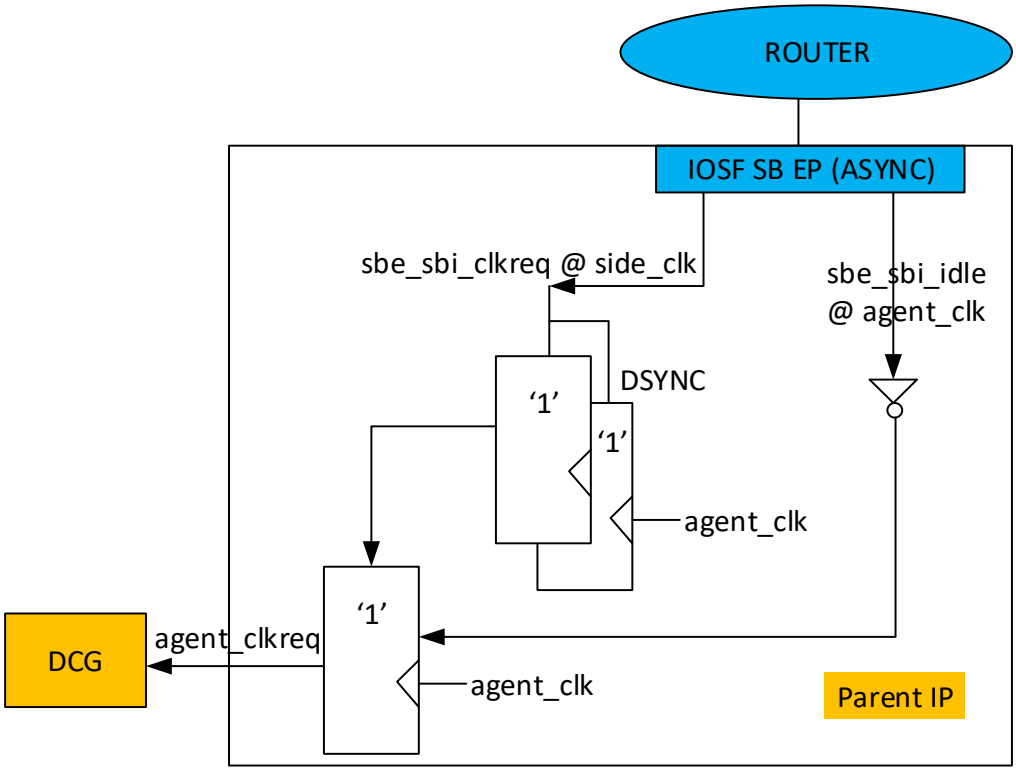
For asynchronous agents that allow gating the agent_clk, the endpoint provides two outputs which are used to control the agent clock gating logic. The signals associated with agent_clk gating are shown below in Table 19.

Table 19. Agent Clock Gating Signals

Signal	Clock Domain		Purpose
—	**ASYNCENDPT=0**	**ASYNCENDPT=1**	—
sbe_sbi_clkreq	side_clk		Clock request signal to agent_clk gating logic. When asserted, any agent_clk gating should be removed.
sbe_sbi_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.

At a high level, once agent clock that's provided to the endpoint starts running, it can be gated only if sbe_sbi_clkreq == 0 and sbe_sbi_idle == 1.

Figure 13. Agent Clock Gating



Note: The full equation for agent_clk gating must take into account both sbe_sbi_clkreq and sbe_sbi_idle. These are provided on separate outputs because they potentially contain elements from different clock domains. The agent design needs to take care of synchronizing these signals if appropriate before using within the agent clock gating logic.

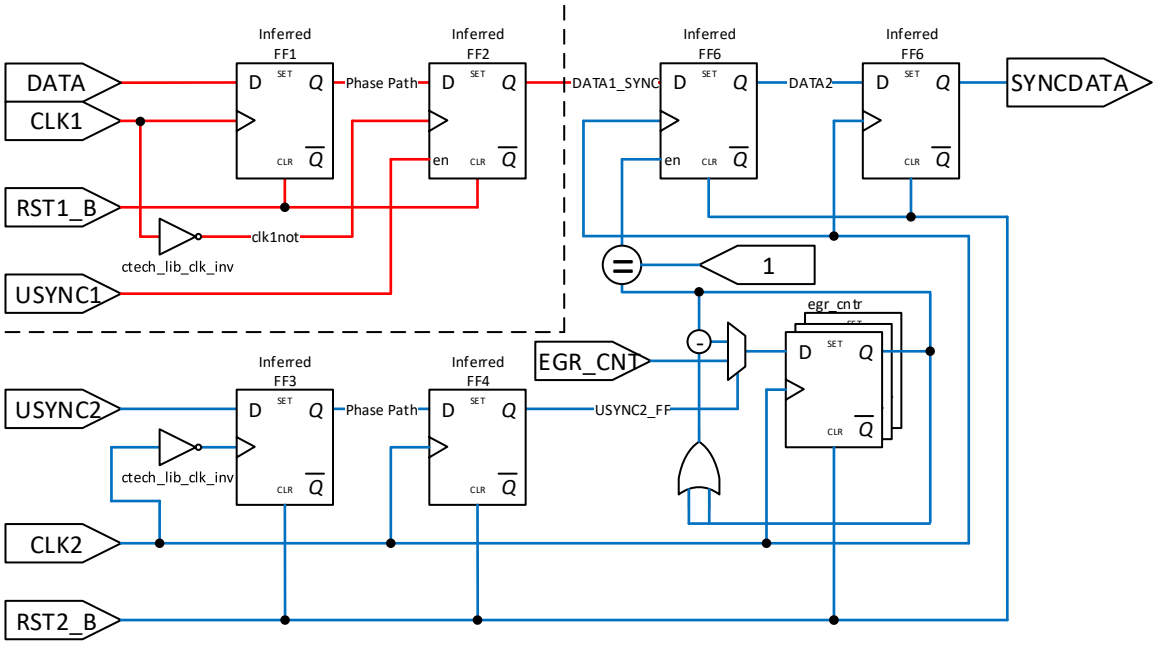
16.10 Design for Deterministic Clock Crossing

The functionality of an asynchronous Endpoint can be made deterministic (for example, cycle accurate repeatable) by enabling deterministic clock domain crossing in the asynchronous FIFOs sitting between the fabric and agent clock domains in the Endpoint.

The circuit used to implement deterministic clock domain crossing is shown in Figure 14.

The parameters SIDE_USYNC_DELAY and AGENT_USYNC_DELAY can be used to delay the egress usync to extend the setup time at the receiving flop by the indicated number of cycles into the corresponding clock domain. If a delay is used then the driver of the USYNC signals must ensure that USYNC1 and USYNC2 are delayed until the counter has expired and data has successfully transfered. Otherwise, there is a risk that metastable data could be received into the receiving clock domain.

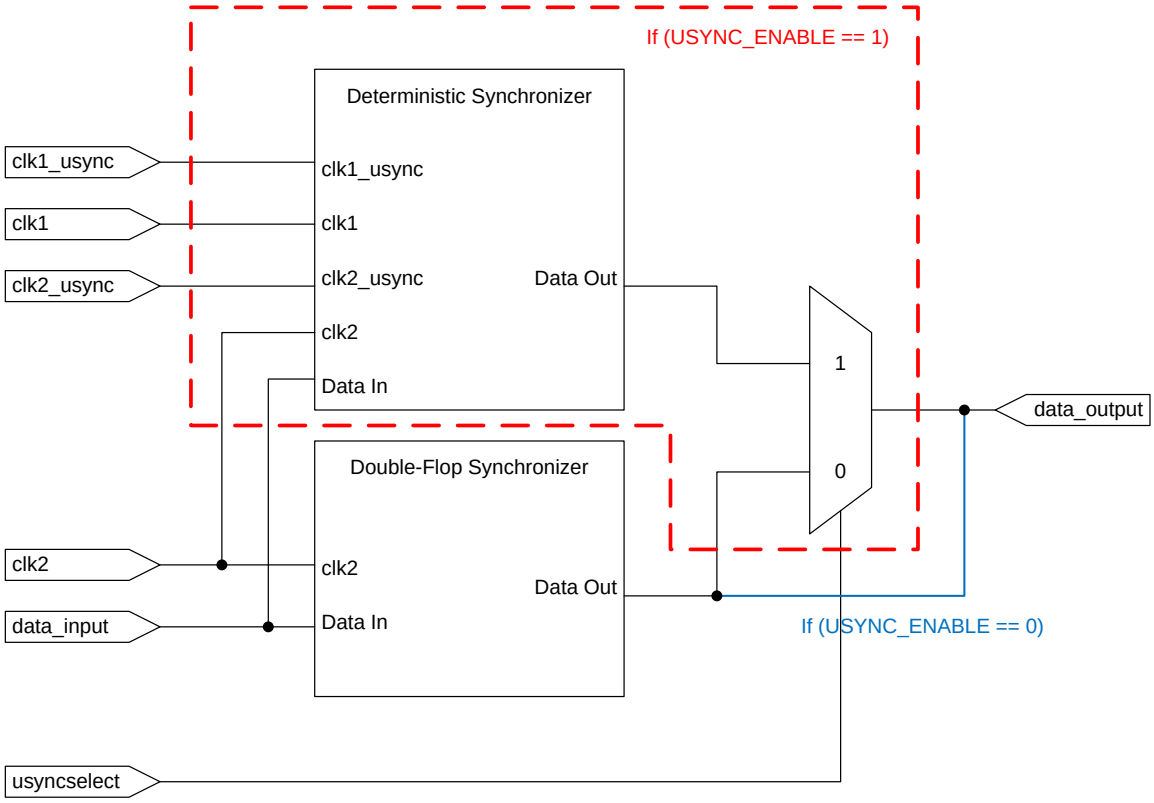
Figure 14. Deterministic Clock Crossing Circuit



The circuit in Figure 14 is placed in parallel with the double-flop synchronizers as shown in Figure 15. If the USYNC_ENABLE parameter is set to 1, the deterministic synchronizer and mux to select between the two synchronizers are implemented in the design. If the USYNC_ENABLE parameter is set to 0, the deterministic synchronizer is not implemented and the synchronized output is always driven from the output of the double-flop synchronizer. The usyncselect signal selects which synchronizer to use. If usyncselect is equal to 1, the deterministic synchronizer is used. If usyncselect is set to 0, the double-flop synchronizer is used.

Note: The usyncselect mux control must remain static during operation of the endpoint. It should only be changed while Endpoint is in reset. The agent clock input to endpoint should be free running and not gated.

Figure 15. Dynamic Selection of Clock Crossing Strategy



Deterministic clock crossing is implemented in both the ingress and egress directions. The clk1_usync, clk2_usync, and usyncselect lines are run from the top of the Endpoint HDL hierarchy. The usync signals are expected to behave according to the timing diagram of Figure 16.

Figure 16. Timing Diagram for the side_usync and agent_usync Signals

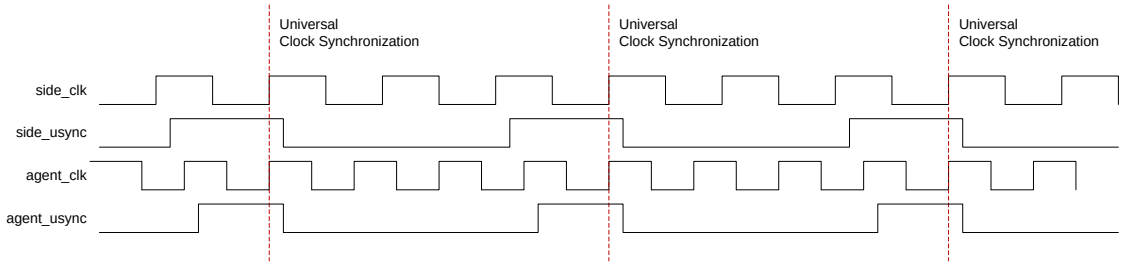


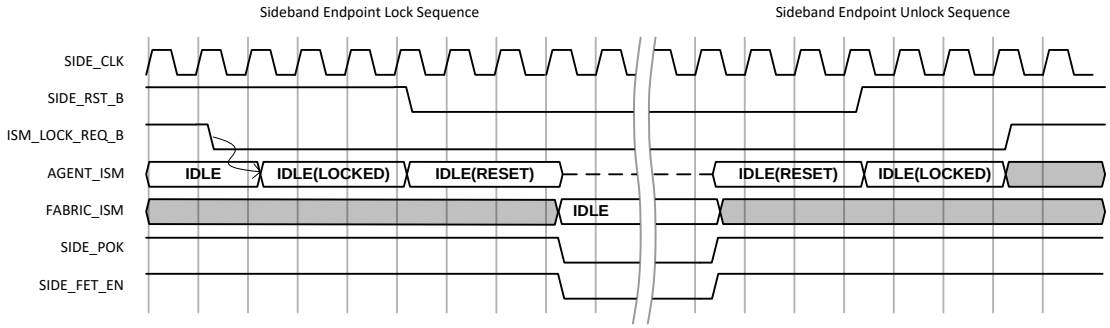
Figure 16 shows that universal synchronization is achieved once every three-side_clk periods and once every four-agent_clk periods. The side_usync signal is driven from the side_clk domain and is asserted for all positive edges of side_clk that are universally synchronized. Similarly, the agent_usync signal is driven from the agent_clk domain and is asserted for all positive edges of agent_clk that are universally synchronized. There is no requirement on relative frequency between the two clocks.

16.11 Design for Power Gating

16.11.1 Use of ism_lock_req_b for Power Gating

To maintain IOSF compliance (and to prevent loss of data) during a power cycle of the endpoint, the endpoint's agent ISM must be in the IDLE state when power is turned off. To accomplish this, the agent must assert ism_lock_req_b and then wait for the agent ISM to go IDLE before gating power the endpoint. Assertion of ism_lock_req_b does not force the agent ISM into the idle state, but it prevents it from leaving, effectively blocking the fabric from sending messages over the IOSF interface that might be lost as the endpoint is powered down. The ism_lock_req_b must be driven synchronously with to side_clk. A timing diagram showing an example use of the ism_lock_req_b is provided in Figure 17.

Figure 17. Timing Diagram for ism_lock_req_b



16.11.2 IOSF Interface Completion Fencing Logic

To help agents with power gating additional fencing logic was inserted into the IOSF interface. When enabled, a counter is inserted into the IOSF port to track when non-posted messages ingress the agent logic and completion messages egress the agent logic. As long as sbe_sbi_comp_exp is high the agent must not power gate. Otherwise, the expecting agent would need to be reset to recover from the missing completion message.

To increment, the ingress module detects when the first flit of a non-posted message leaves the ingress queue and enter into the agent portion of the endpoint. To decrement, the egress module detects when a message containing a completion opcode has entered the egress output flops. This will happen on the flit containing the opcode.

There is additional logic to prevent the IOSF ISM from entering IDLE_REQ so long as the counter is non-zero.

Note: This will enforce power management through the ISM at the cost of keeping the corresponding fabric awake and keeping the sideband clock running. This would have a high power cost, especially for agents with clocks that are opperating much slower than the IOSF fabric.