IOSF Sideband Endpoint

HIGH-LEVEL ARCHITECTURE

SPECIFICATION

PIC RTL 1.0-PICr35
January 2021

Contents

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 we did not overlook this topic.

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

Note: 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.” 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 but just replace the Gap text with the phrase “Not applicable to this IP” and apply the BodyText style.

Goal of This Document

This document should contain all architecture information an SoC team would need to accomplish their design 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.

1 About this Document

1.1 Audience

The information in this document is intended for an SoC design team that is using this IP.

1.2 References

Name	File Name	Link/Location
IOSF Specification 1.3		http://goto/iosf
IOSF Sideband Usage Specification 1.1.1		http://goto/iosf_sideband_usage
SIP IOSF Sideband Base Endpoint Integration Guide		Release directory
SIP IOSF Sideband Endpoint Integration Guide		Release directory
SIP IOSF Sideband Endpoint Product Brief		Release directory
SIP IOSF Sideband Endpoint Verification Plan Reference		Release directory
SIP IOSF Sideband Endpoint Release Notes		Release directory

1.1 Contact Information

Function	Name	Email
Architect/Microarchitect	Robert Hesse	robert.hesse@intel.com
Design	Vishnu Nandakumar	vishnu.vardhan.nandakumar@intel.com
Validation	Soe Myint	soe.myint@intel.com
Spec Template Owner	Susann Flowers	susann.flowers@intel.com

1.2 Terminology

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 Document Revision History

Table 1. Revision History for this Document

Author	**Revision No.**	**Revision Description**	**Revision Date**
	0.2	Initial Release	14 June 18
	0.5		dd Month yy
	0.6		dd Month yy
	0.7		dd Month yy
	0.8		dd Month yy
	0.9		dd Month yy
	1.0		dd Month yy
Robert Hesse	Rev 1.0.1	New HAS Template + Fixes for zircon compliance	February 2019

1.3 IP Revision History

Table 2. IP Revision History

Author	**Revision No.**	**Description** *Do not remove or change the text that is here. Add rows or text to describe your IP revision*	**Corresponding RTL**	**Revision Date**
	Rev 0.5	Enable all features for Rev 0.5 RTL; full definition of an IP block including performance capabilities, allowing FED to begin Outlines an high-level description of power, signals, interfaces, and security Full Review Prior to Rev 0.5 complete DFx features/requirements are complete Summary outline available for RAS/SER & PerfMon Enables Rev 0.5 RTL Change Control start at Rev. 0.5; no new features added; Issues and BugECOs managed in HSD-ES	RTL0.5	26 August 16
	Rev 0.8	Spec Complete to enable Rev 0.8 RTL PCR Control (Product Change Request - IP Only) RAS/SER & PerfMon features detailed and specified	RTL0.8	26 August 16
	Rev 1.0	DCCB Control prior to TO, RTL Rev 0.8 (Post review) + Rev 0.8 HAS + Issues from HSD-ES + Full Review ECO Control (Engineering Change Order)	RTL1.0	26 August 16
	Rev 1.1	HAS Complete		26 August 16
	Rev 1.x	HAS Updates		26 August 16

1.4 Firmware Version

Not applicable to this IP

2 IOSF Sideband Endpoint Introduction

This is a feature HAS, which for IPs describes what the feature is and how it works. It is to be consumed primarily by the design and validation teams.

2.1 IP Description

The IOSF Sideband Endpoint is not a standalone IP. It is a subIP that gets integrated into a parent IP to provide a connection to the IOSF Sideband Fabric for the parent IP. The IOSF Sideband subIP can either be integrated as the IOSF Sideband Base Endpoint only, or as the IOSF Sideband Endpoint with additional register access capabilities.

Integration details are provided in the IOSF Sideband Endpoint integration guide document.

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

This allows the sideband fabric 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).

The IOSF Sideband Endpoint extends the IOSF Sideband Base Endpoint with optional Target and Master Register Access services. These services are intended to ease handling of the predefined Register IO transactions as defined in the IOSF specification. In effect, the endpoint translates these transactions into parallel operations instead of packet based operations. The register access modules are provided with basic features, any advance features not implemented already would need to be added by the integration teams.

2.2 Block Diagram

Figure 1 shows the sideband interface base endpoint, which contains at least three blocks as shown in Table 3. Figure 2 shows the sideband endpoint, which extends the IOSF Sideband Base Endpoint with optional Target and Master Register Access services

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.

Block	Description
SBCASYNCFIFO	Optionally instantiated to create asynchronous endpoint.

Figure 1. Base Endpoint Block Diagram

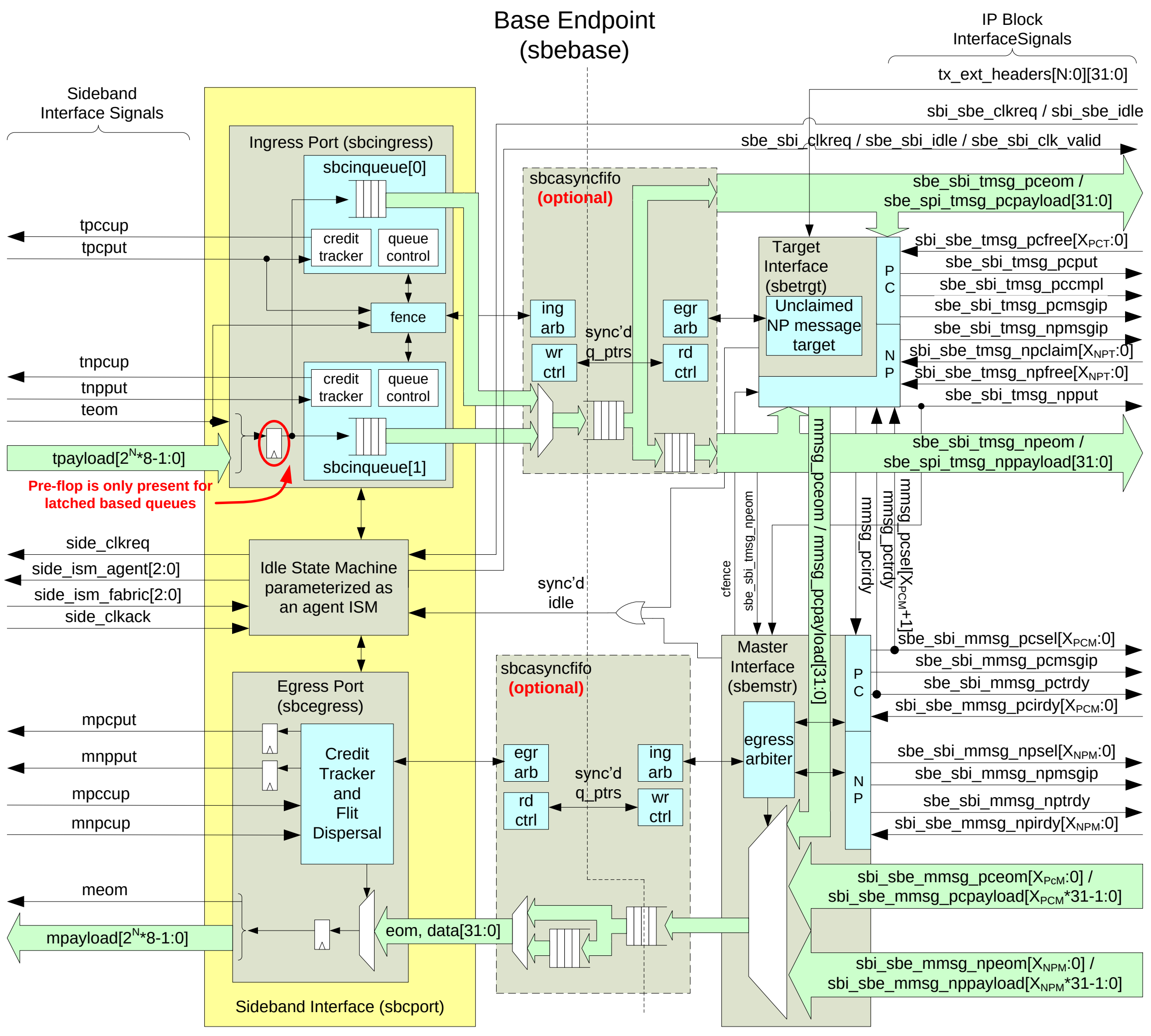
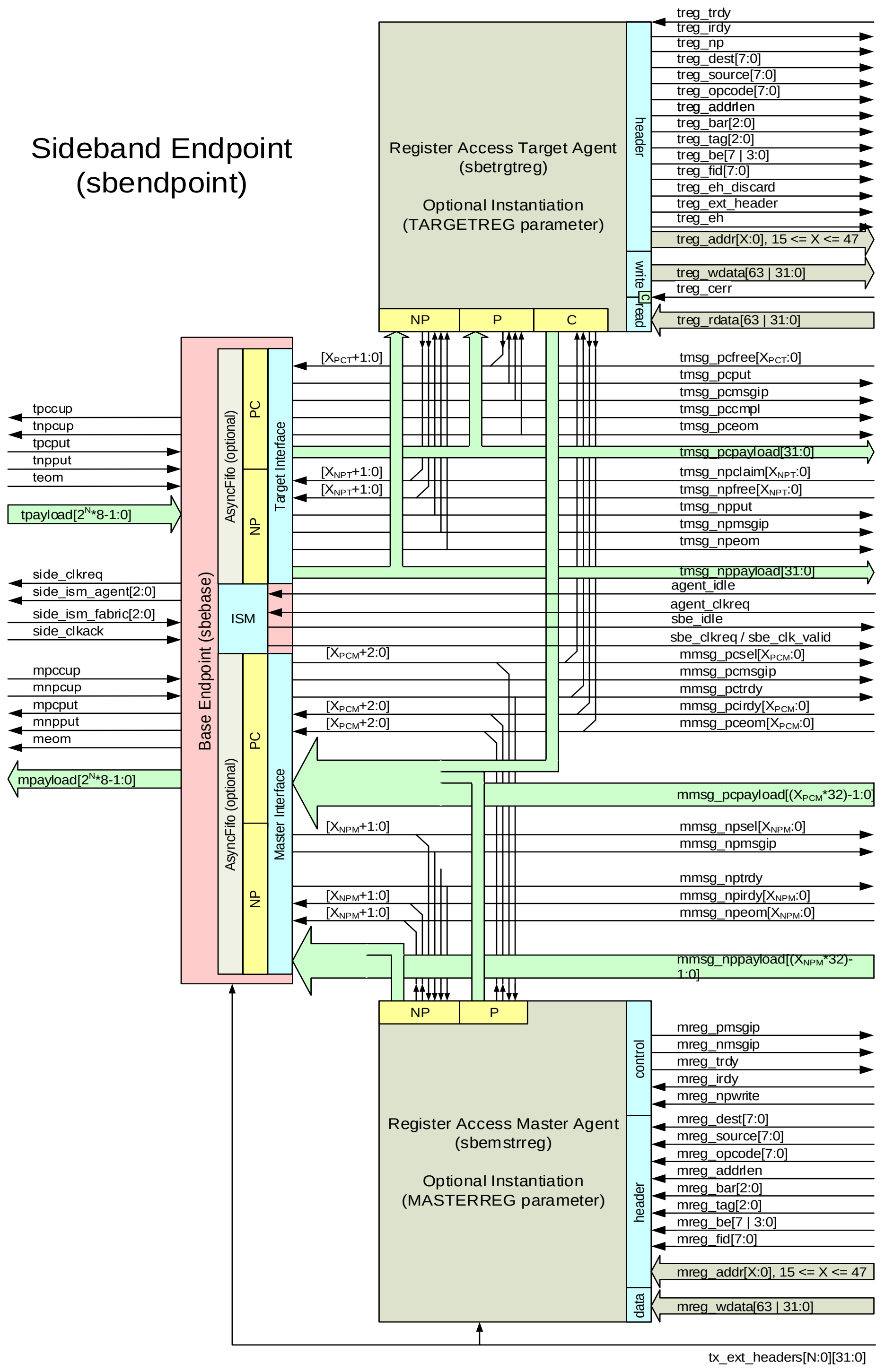

Base Endpoint (sbebase)
IP Block InterfaceSignals
Sideband Interface Signals
tx_ext_headers[N:0][31:0]
sbi_sbe_clkreq / sbi_sbe_idle
sbe_sbi_clkreq / sbe_sbi_idle / sbe_sbi_clk_valid
Ingress Port (sbcingress)
sbcinqueue[0]
credit tracker
queue control
fence
sbcinqueue[1]
tpccup
tpcput
tnpcup
tnpput
teom
tpayload[2^N*8-1:0]
Pre-flop is only present for latched based queues
sbcasyncfifo (optional)
ing arb
wr ctrl
sync'd q_ptrs
egr arb
rd ctrl
sbe_sbi_tmsg_pceom / sbe_spi_tmsg_pcpayload[31:0]
Target Interface (sbetrgt)
Unclaimed NP message target
P C
N P
sbi_sbe_tmsg_pcfree[XPCT:0]
sbe_sbi_tmsg_pcput
sbe_sbi_tmsg_pccmpl
sbe_sbi_tmsg_pcmsgip
sbe_sbi_tmsg_npmsgip
sbi_sbe_tmsg_npclaim[XNPT:0]
sbi_sbe_tmsg_npfree[XNPT:0]
sbe_sbi_tmsg_npput
sbe_sbi_tmsg_npeom / sbe_spi_tmsg_nppayload[31:0]
mmsg_pceom / mmsg_pcpayload[31:0]
mmsg_pcirdy
mmsg_pctrdy
mmsg_pcsel[XPCM+1]
cfence
sbe_sbi_tmsg_npeom
side_clkreq
side_ism_agent[2:0]
side_ism_fabric[2:0]
side_clkack
Idle State Machine parameterized as an agent ISM
sync'd idle
Master Interface (sbemstr)
sbe_sbi_mmsg_pcsel[XPCM:0]
sbe_sbi_mmsg_pcmsgip
sbe_sbi_mmsg_pctrdy
sbi_sbe_mmsg_pcirdy[XPCM:0]
sbe_sbi_mmsg_npsel[XNPM:0]
sbe_sbi_mmsg_npmsgip
sbe_sbi_mmsg_nptrdy
sbi_sbe_mmsg_npirdy[XNPM:0]
egress arbiter
sbi_sbe_mmsg_pceom[XPCM:0] / sbi_sbe_mmsg_pcpayload[XPCM*31-1:0]
sbi_sbe_mmsg_npeom[XNPM:0] / sbi_sbe_mmsg_nppayload[XNPM*31-1:0]
Egress Port (sbcegress)
Credit Tracker and Flit Dispersal
mpcput
mnpput
mpccup
mnpcup
meom
mpayload[2^N*8-1:0]
eom, data[31:0]
Sideband Interface (sbcport)

Figure 2. Sideband Endpoint Block diagram



2.3 Landing Zone

POR Features:

- 1, 2 or 4 byte payload width

  - The sideband payload width can be parameterized as either 1 byte, 2 bytes, or 4 bytes wide.
  - The payload width of the ingress port (tpayload) must equal the payload width of the egress port (mpayload).
  - Internal datapath width may differ from the IOSF interface payload width.
- Designed to support fine-grain power gating.
  - Routers and endpoints may be associated with power gated domains.
  - endpoints will error handle the transaction as appropriate (PC transactions dropped, NP transactions sent a powered-down completion).
- Parameterized ingress queue depth (1 to 3 entries)
- Full support for IOSF Sideband Parity, as required by the IOSF specification.
- 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.

2.4 Voltage and Frequency Operating Points

Not applicable to this IP/IPSS

2.5 Standards and Specification Compliance

The Sideband Endpoint subIP is compliant with the IOSF specification, version 1.3 and the IOSF-SB Usage specification version 1.0.1.

2.6 Difference from Previous Project

Not applicable to this IP/IPSS

2.7 Use Cases

Interfacing of IOSF-SB agents to the IOSF-SB fabric.

2.8 Derivative Strategy

Not applicable to this IP/IPSS

2.9 Scalability, Configurability, and Customizations

The IOSF Sideband Endpoint is highly scalable and configurable. See the landing zone or integration guide for more specific info.

1.4.1 Hardware Scalability, Configurability, and Customizations

Not applicable to this IP

1.4.2 Software Scalability, Configurability, and Customizations

Not applicable to this IP

2.10 Design Assumptions

Not applicable to this IP/IPSS

3 Functional Description

This section covers the functionality of the IOSF Sideband Endpoint.

3.1 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 3.1.1, Idle State Machine (SBCGCGU)), the ingress port (SBCINGRESS) (section 3.1.2, Ingress Port (SBCINGRESS), and the egress port (SBCEGRESS) (section 3.1.3, Egress Port (SBCEGRESS)).

3.1.1 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.

3.1.2 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.

3.1.3 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.

3.2 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.

3.3 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.

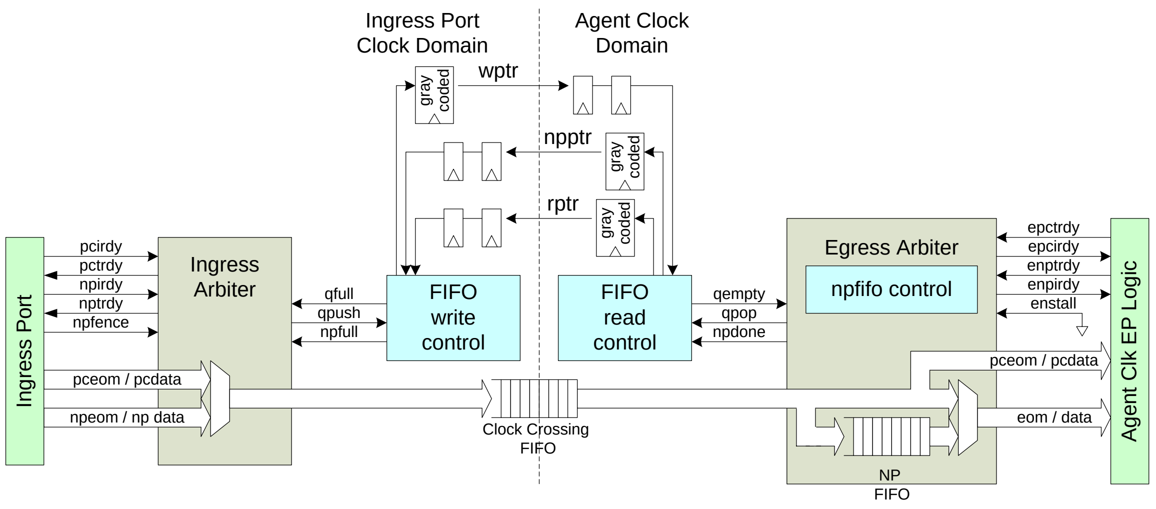
3.4 Asynchronous FIFO (SBCASYNCFIFO)

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 3. Asynchronous FIFO Architecture



As shown above in Figure 3, 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.

3.5 Register Access Master Agent (SBEMSTRREG)

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

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

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

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

3.6 Register Access Target Agent (SBETRGTREG)

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

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

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

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

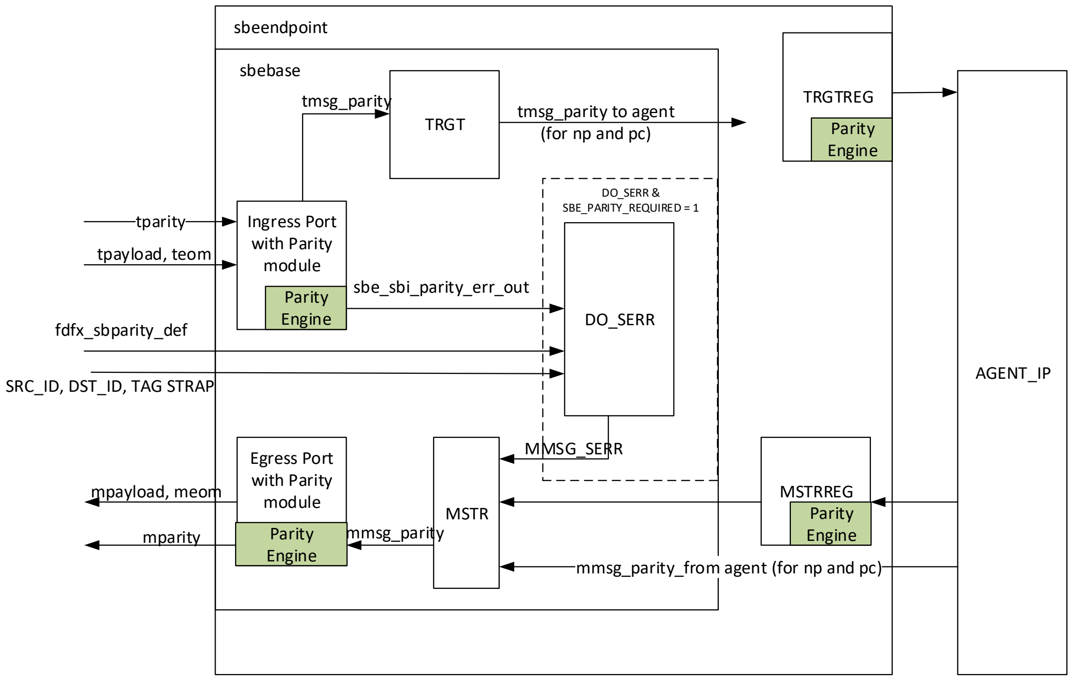
Note: The flops that are used to hold the write data are also used to capture the read data.

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

3.7 DO_SERR Master Widget (DOSERRMSTR)

A DOSERRMSTR widget is instantiated in the Endpoint as illustrated in Figure 4, when the SB_PARITY_REQUIRED and DO_SERR_MSTR parameters are both set. When a parity error is detected, this module generates a DO_SERR error message (opcode 0x88), by employing the source/dest/tag information supplied through top-level straps (i.e., sbi_sbe_srcid_strap, sbi_sbe_dstid_strap, and sbi_sbe_tag_strap). When TX_EXT_HEADER_SUPPORT is also enabled, and the sbi_sbe_sairs_valid_strap is set, the strap values for SAI (sbi_sbe_sai_strap) and RS (sbi_sbe_rs_strap) are also packaged as a 2nd DW word in the DO_SERR message.

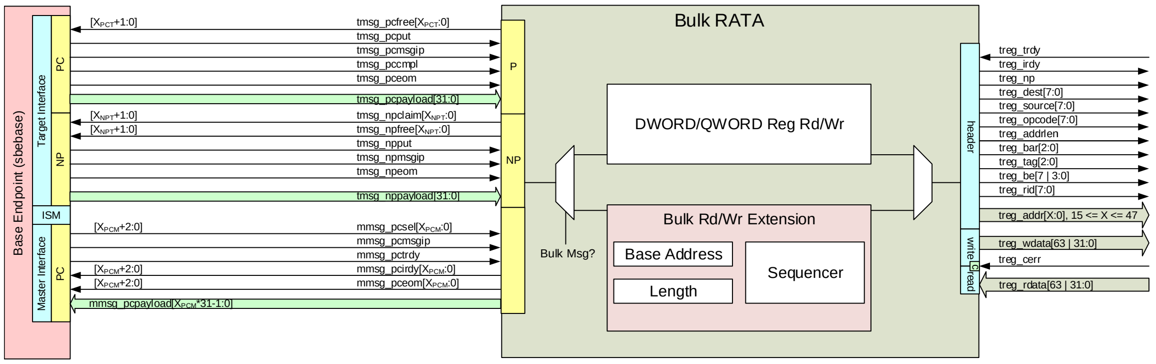
Figure 4. Instantiation of the DOSERRMSTR Widget



3.8 Bulk Read/Write Register Widget (BULKRDWR)

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

Figure 5. Instantiation of the BULKRDWR Widget



3.9 Safety <Feature Name1>

Not applicable to this IP

3.9.1 Feature Section Title

Not applicable to this IP

3.9.2 Feature Section Title

Not applicable to this IP

4 Interfaces

Details of the Sideband Endpoint interface are covered in the integration guide.

4.1 IP Interface Signal List

Signal Name	Type	Description
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.

Table 4. 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.
mparity	Output	Parity bit associated with the master payload data, mpayload This output should be used only when parameter SB_PARITY_REQUIRED=1.

Table 5. 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.

Signal Name	Type	Description
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	In sync mode, 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. In async mode free should be asserted first. Free/put protocol must still be observed.
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.

Signal Name	Type	Description
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 6. 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.
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.

Signal Name	Type	Description
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.

4.2 Functional Signals

Signal Name	Type	Description
sbi_sbe_clkreq	Input	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 deasserted after the base endpoint reports that it is no longer idle. This is to avoid metastability issues.
sbi_sbe_idle	Input	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.
side_clkreq	Output	Clock request signal output to the sideband fabric. Follows IOSF 1.0 Specifications for clock requests.
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.

Signal Name	Type	Description
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	Output from the sbe to the IP block.
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.
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	These are config register 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.
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.

Signal Name	Type	Description
ur_csai[SAIWIDTH:0]	Input	When ur_cairs_valid is a 1, this input will be inserted into the generated SAIRS extended header for the unsupported response completion message. This input is not used when UNIQUE_EXT_HEADERS is set to 0 This input must be held stable by the EOM of the ingressing, non-posted message as it will be used without flops.
ur_crs[RSWIDTH:0]	Input	When ur_csairs_valid is a 1, this input will be inserted into the generated SAIRS extended header for the unsupported response completion message. This input is not used when UNIQUE_EXT_HEADERS is set to 0 This input must be held stable by the EOM of the ingressing, non-posted message as it will be used without flops.

4.3 Power, Resets, and Firewalls

Signal Name	Type	Description
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_side_rst_b_sync	Input	For internal use with sbendpoint only. Tie this input low.

4.4 Clocks

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.
agent_clk	Input	The IP block clock input. Required for asynchronous endpoints (ASYNCENDPT=1), otherwise it can be strapped.

4.5 DFx

Signal Name	Type	Description
visa_*	Output	VISA candidate signals. See “VISA/DFx Structures”
jta_clkgate_ovrd	Input	See “Configuration and Override Inputs” in the RTL
jta_force_clkreq	Input	
jta_force_idle	Input	
jta_force_notidle	Input	
jta_force_creditreq	Input	

Signal Name	Type	Description
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.

5 Interrupts

Not applicable to this IP/IPSS

6 Clocking

6.1 Clocking Requirements

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.

6.2 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. 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 early 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.

Note: 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 7.

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 7. 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.

Signal	Clock Domain		Purpose
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.
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.

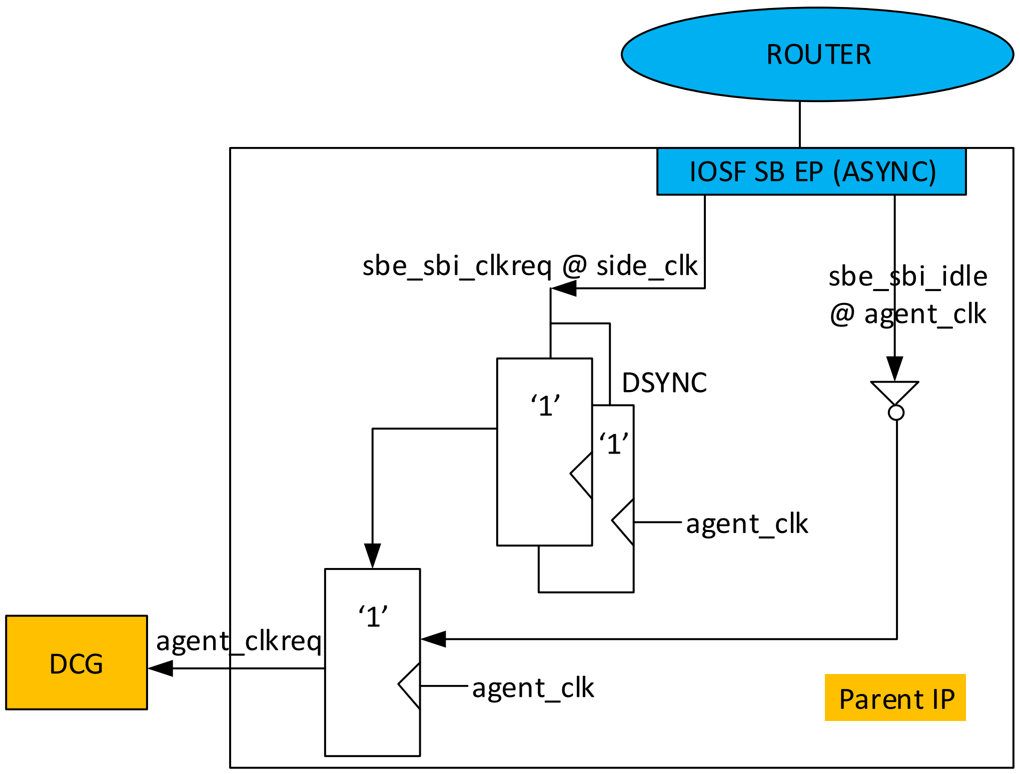
For asynchronous agents that allow gating the agent_clk, the endpoint provides two outputs which are used to control the agent clock gating logic. The signals associated with agent_clk gating are shown below in Table 8.

Table 8. 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 6. 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.

6.3 PLLs

Not applicable to this IP/IPSS

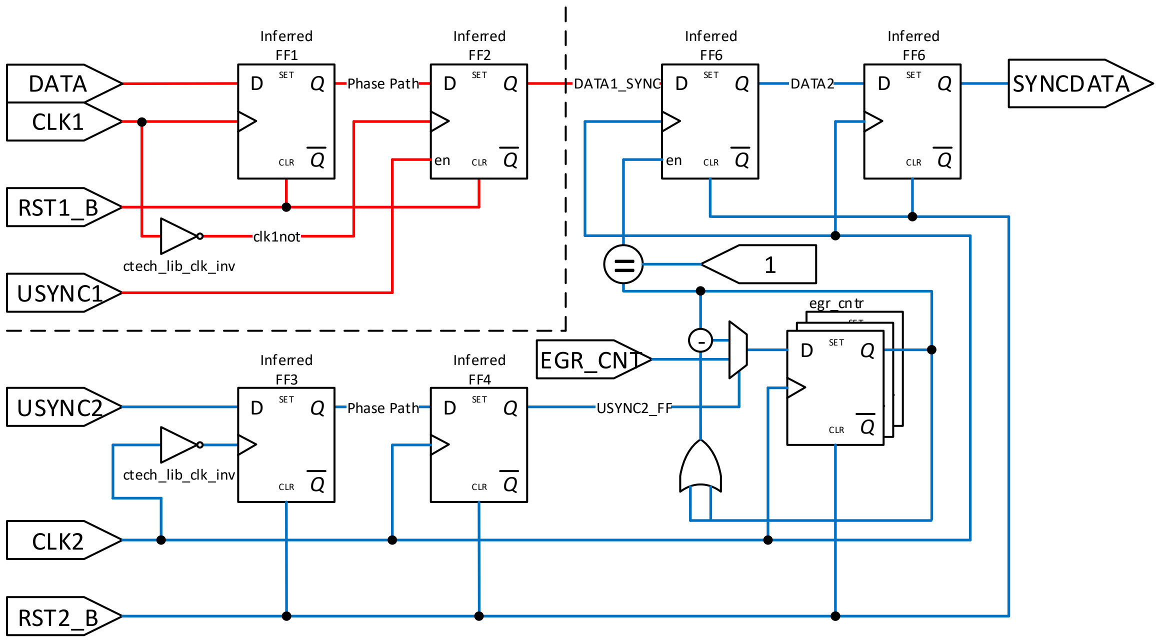
6.4 Clocking Assumptions

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 7.

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.

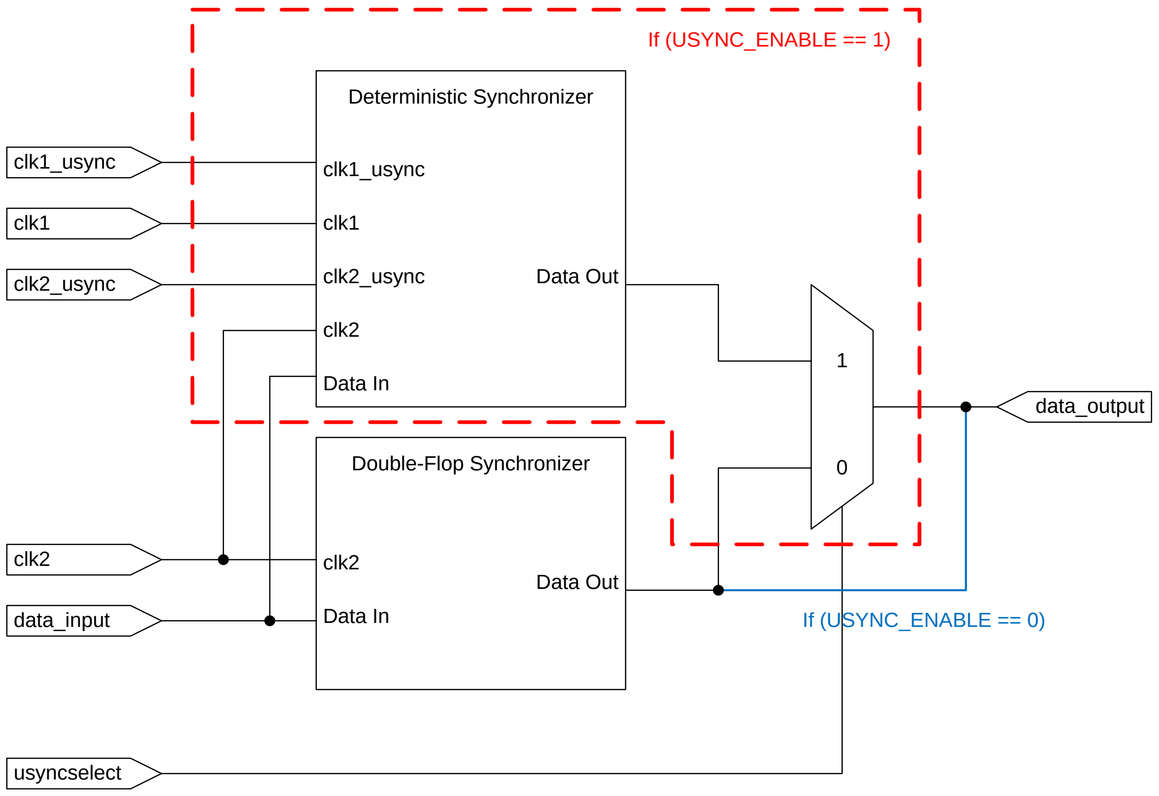
Figure 7. Deterministic Clock Crossing Circuit



The circuit in Figure 7 is placed in parallel with the double-flop synchronizers as shown in Figure 8. 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 8. 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 9.

Figure 9. Timing Diagram for the side_usync and agent_usync Signals

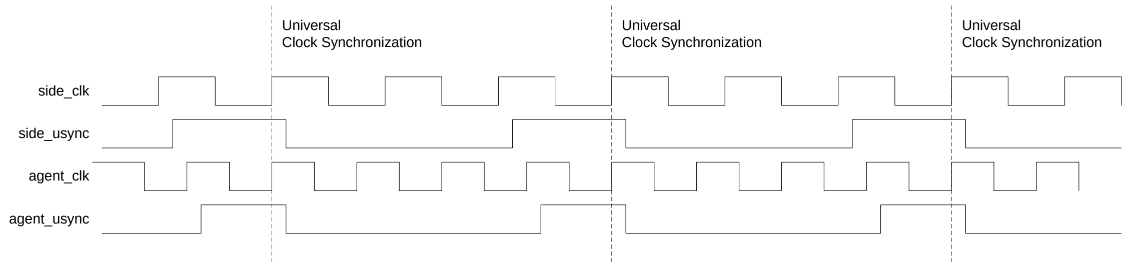


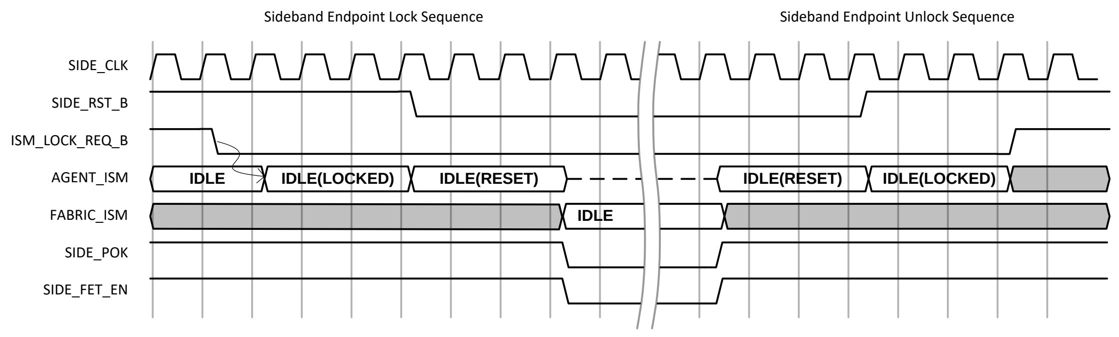
Figure 9 shows that universal synchronization is achieved once every three-side_clk periods and once every four-agent_clk periods. The side_usync signal is driven from the side_clk domain and is asserted for all positive edges of side_clk that are universally synchronized. Similarly, the agent_usync 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.

7 Power Management

7.1 Isolation Gates and 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 10.

Figure 10. Timing Diagram for ism_lock_req_b



7.2 Behavior in Various Power States

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.

8 Reset Transactions

8.1 Reset Controls

Not applicable to this IP/IPSS

8.2 Reset Initialization Flows

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.

8.3 Reset Assumptions

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.

9 Transaction Flows

9.1 Transactional Interface

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

9.1.1 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.

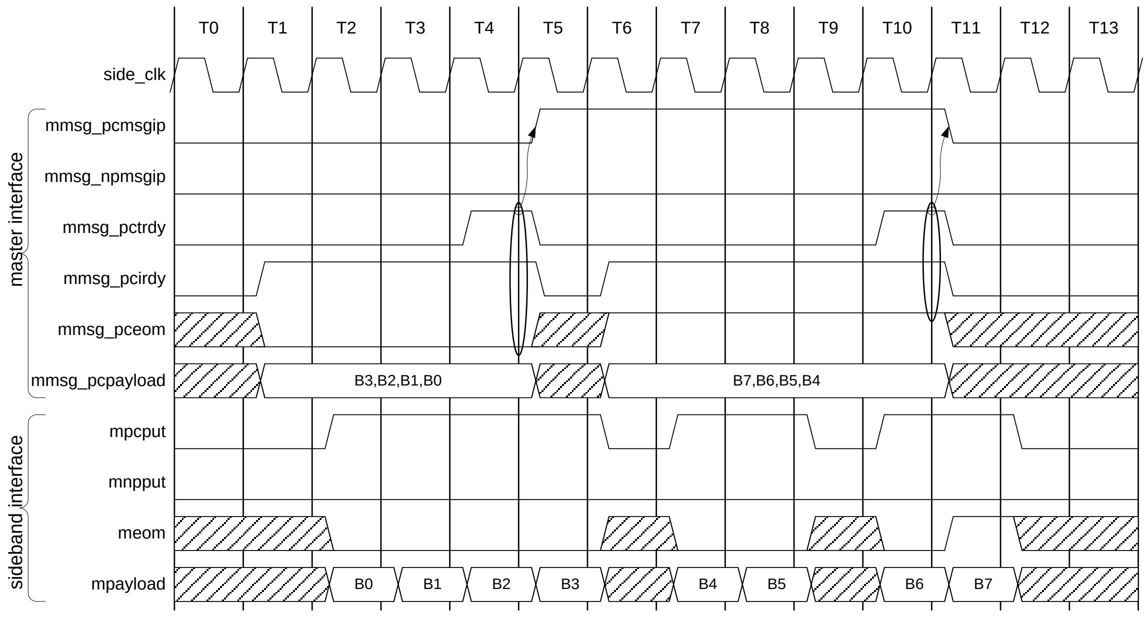
- Cycle T5 shows the message transfer being throttled by the IP block due to the de-assertion of mmsg_irdy.
- Cycle T9 shows the message being throttled by the sideband channel due to insufficient credits.

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 9.1.3, Master Interface – Multi-Master Operation.

Figure 11. Timing Diagram—Master Interface, Posted Message with Data



9.1.2 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.

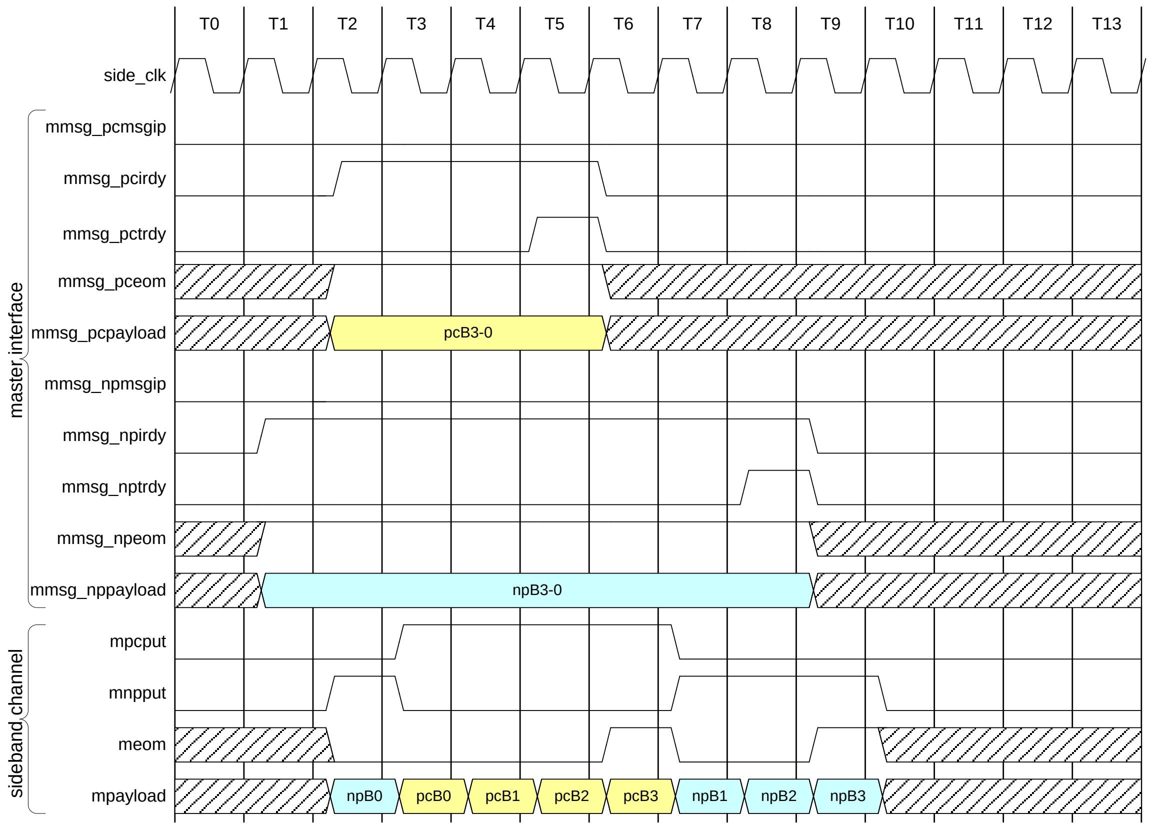
Due to insufficient non-posted credits, the posted message passes the non-posted message in the base endpoint. Once the posted message completes, the endpoint continues to send flits for the non-posted message until it completes.

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



9.1.3 Master Interface – Multi-Master Operation

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

Figure 13 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).

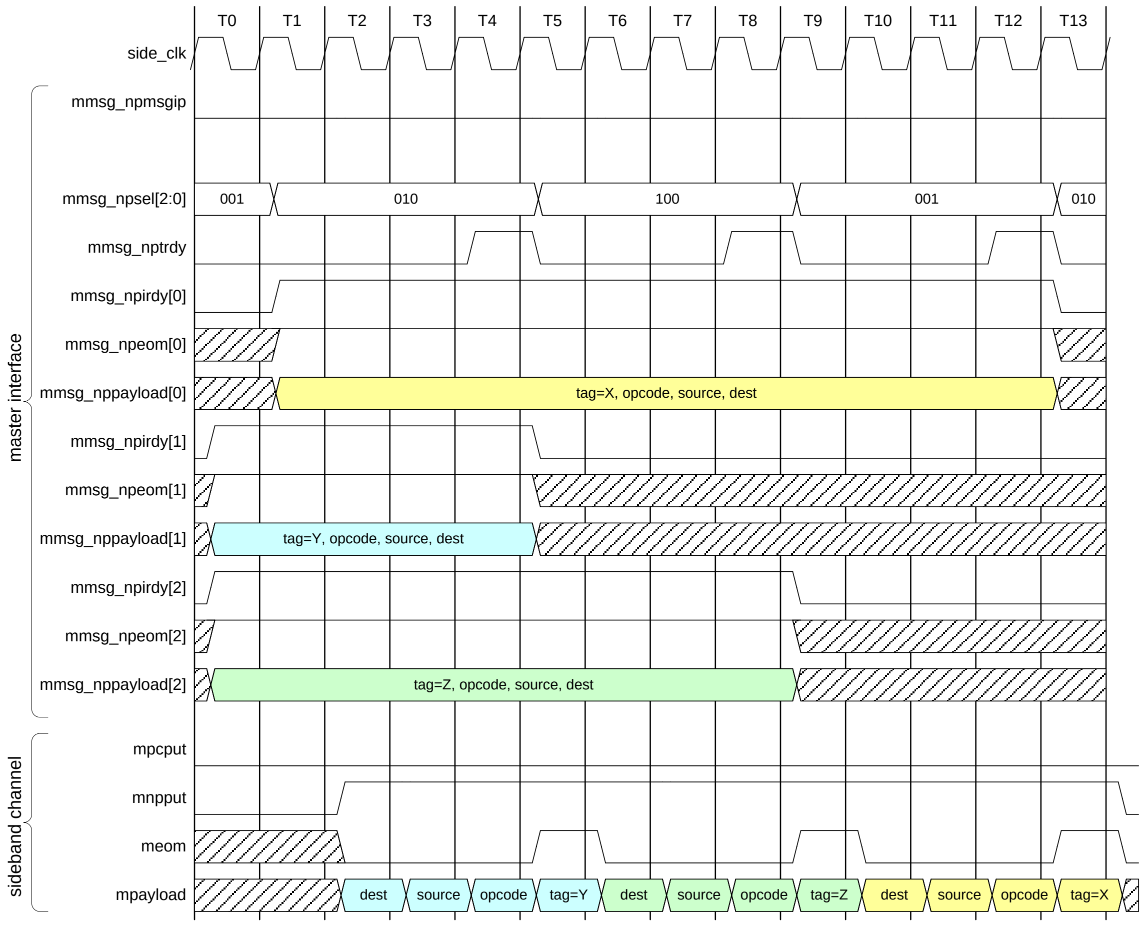
The arbiter transitions to select master 2 in T5, because the end of message transfer occurred at the end of T4.

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 13. Timing Diagram—Master Interface, Multi-Master Operation



9.1.4 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.

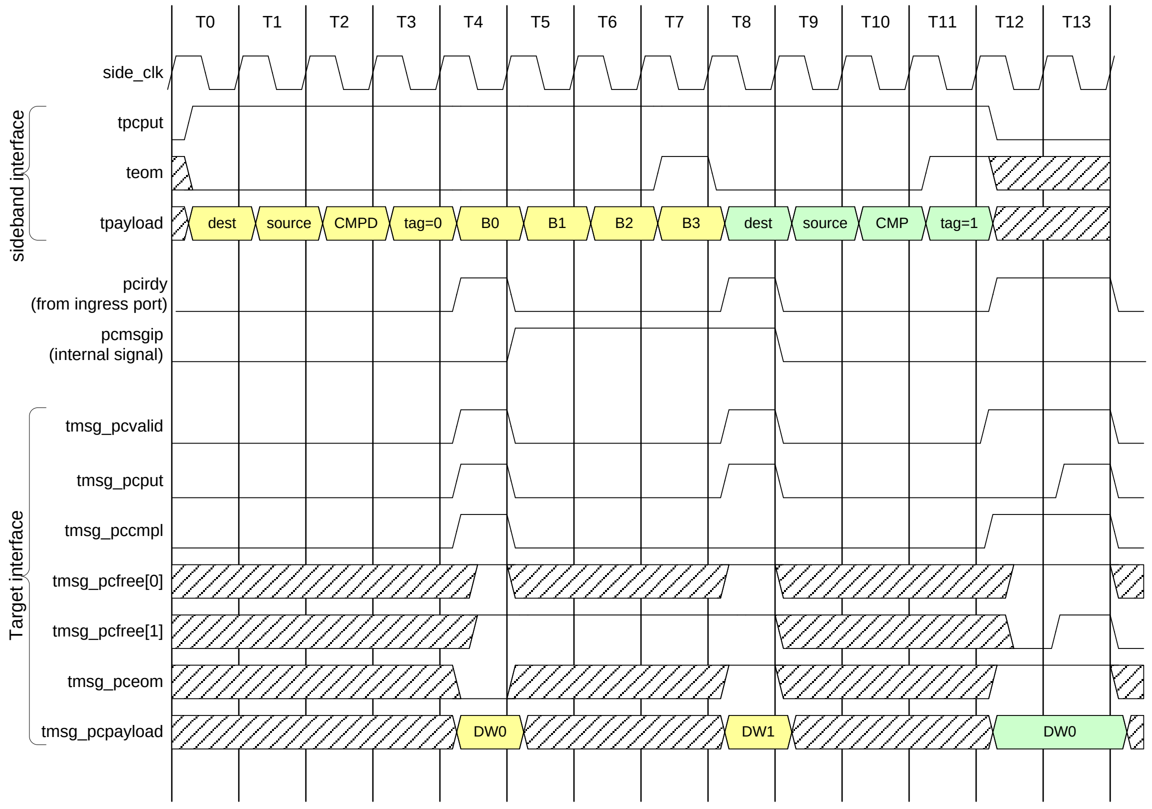
Note: Both targets have to asserted tmsg_pcfree in order for the transfer to take place on the target interface.

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 14. Timing Diagram—Target Interface, Multi-Target Operation



9.1.5 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.

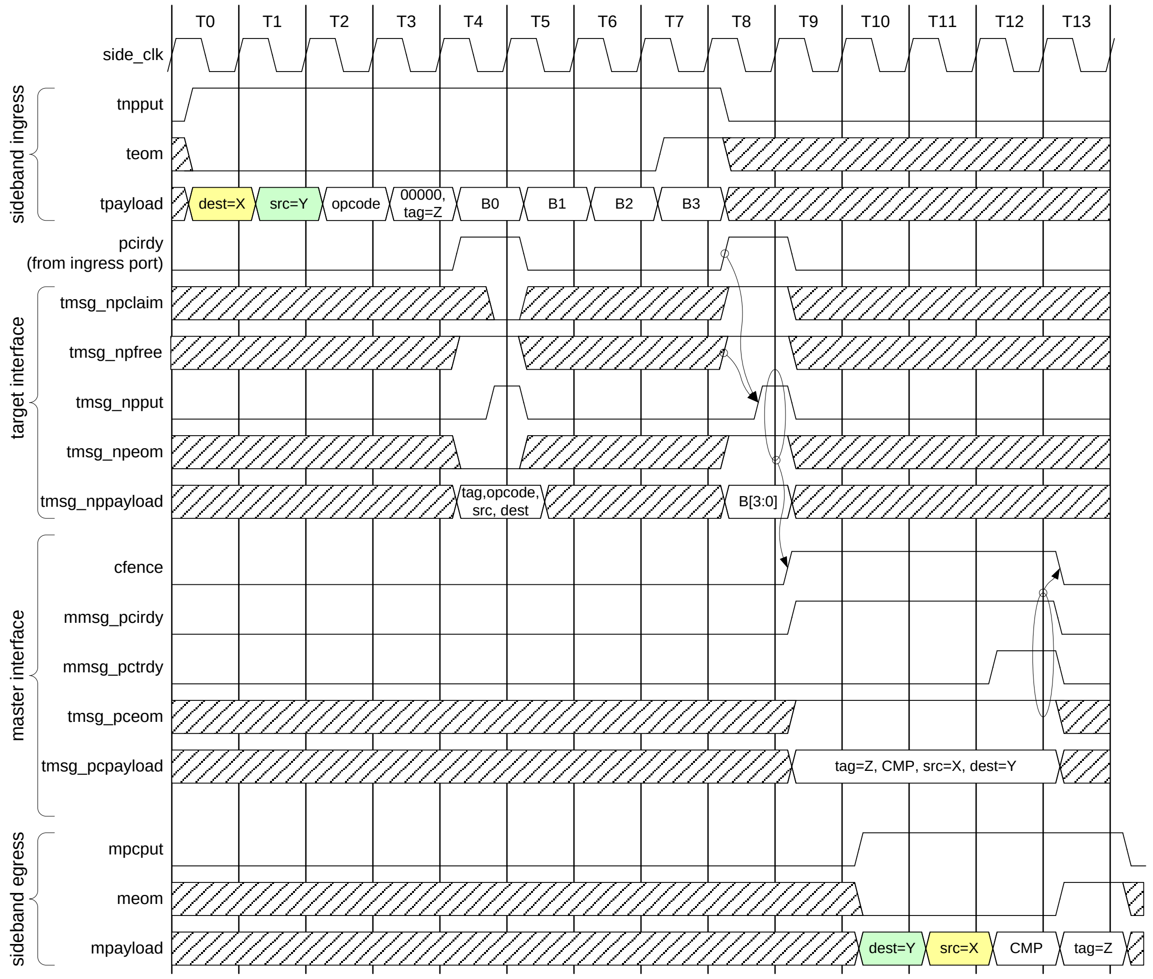
In addition, the target agent must wait until the final 4 bytes of the non-posted message are consumed on the target interface before initiating the completion on the master interface. The timing diagram below shows this occurring in the cycle after the EOM transfer (tmsg_npput/npeom both asserted).

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.

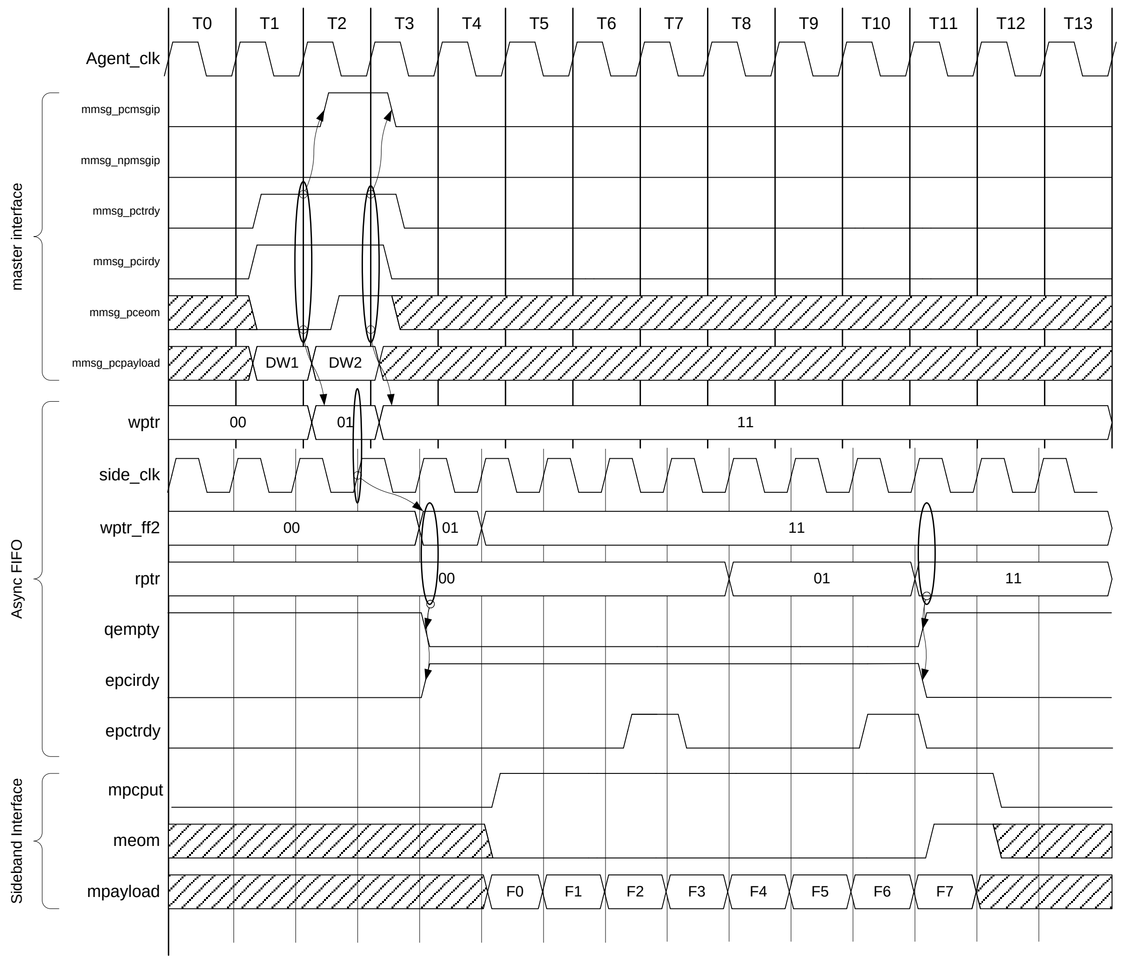
Figure 15. Timing Diagram—Non-posted Target Message with a Master Completion



9.1.6 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 16. 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.

9.2 Ordering/Coherency Rules

The Sideband Endpoint implements ordering rules as described in the IOSF Spec version 1.3.

10 Register Descriptions

Not applicable to this IP/IPSS

11 Area and Gate Count Goals

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.

12 Power Goals

Not applicable to this IP/IPSS

13 Performance Goals

When the ingress queues are configured correctly to support the pipeline delay on their interface + their own internal credit delay, The IOSF Sideband Endpoint is capable of supporting a flit per clock ingressing the endpoint.

13.1 Maximum Performance Targets

Not applicable to this IP/IPSS; the performance depends on the specific configuration of the IP.

13.2 Performance Environment

The performance environment is determined by the integrating parent IP.

14 Functional and Performance Monitoring

Not applicable to this IP/IPSS

15 Reliability, Availability, and Serviceability (RAS)

The IOSF Sideband Endpoint supports Sideband Parity as described in the IOSF Specification.

16 Security and Access Control

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

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

16.1 Security Mitigations

The security of the endpoint is responsibility of the integrating parent IP.

16.2 Security Threats and Attack Points

Not applicable to this IP/IPSS; there are no known attack points or security threats.

16.3 Architecture Review

Not applicable to this IP/IPSS

16.3.1 Assets

Not applicable to this IP/IPSS

16.3.2 Security Objectives

Not applicable to this IP/IPSS

16.4 Security Risk Assessment

Provide the results of the Security Risk Assessment (SRA) tool. Reference:

https://sharepoint.amr.ith.intel.com/sites/SeCoE/Detect/IP_SRA/Forms/AllItems.aspx?RootFolder=%2Fsites%2FSeCoE%2FDetect%2FIP%5FSRA%2FIP&FolderCTID=0x012000722D3D40DEFE3C45884C05D15ABF0A01&View={9B73CCD5-AF75-4958-BA72-A3BEEBC340AC}

16.5 Additional Security Information

Not applicable to this IP/IPSS

16.6 Corner Cases

Not applicable to this IP/IPSS

17 Safety

See the Safety Development Lifecycle site for information about safety-related design practices:

http://goto.intel.com/FuSaLifecycle

or

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

See the Chassis Safety Architecture Spec for information on Standard IP HW safety-releated design requirement:

- IP_Chassis_FuSa_HW_Architecture_Guideline

  `goto/docktracker`

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

17.1 Safety Feature Traceability Mapping

The functional features and DFD/DFT features, which are described in Functional Description Chapter (see section 3, section 19 and section 20), allow the usage of the IP in Platform/Soc FuSa (**Fu**nctional **Sa**fety) applications.

17.2 Safety Risk Assessment

Not applicable to this IP

17.3 Additional Safety Information

Not applicable to this IP

18 Validation Guidance

Validation guidance is provided in both the endpoint integration guide and the verification plan documents. Both documents are available in the release directory.

19 Test Requirements

19.1 MBIST and PBIST

Not applicable to this IP/IPSS

19.2 Array Freeze

Not applicable to this IP/IPSS

19.3 Scan

The endpoint uses scan flops. See Integration Guide for details (included in the 'doc' directory for this release).

19.4 MISR

Not applicable to this IP/IPSS

19.5 DFT Feature Definition

The sideband endpoint provides VISA. Details can be found in the Integration Guide (included in the 'doc' directory for this release).

19.6 DFT PLC Implementation Timeline Checklist

Not applicable to this IP/IPSS

19.7 SCAN System Connectivity and Features

Not applicable to this IP/IPSS

19.8 Validation Test and Environment Requirements

The details are described in the Validation Integration Guide (included in the 'doc' directory for this release).

19.9 HVM Tooling and Flows Requirements

Not applicable to this IP/IPSS

19.10 HVM Reset Requirements

Not applicable to this IP/IPSS

19.11 HVM Clocking and Determinism Requirements

Not applicable to this IP/IPSS

19.12 DFT Chassis Topology and Connectivity

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

19.13 DFT iIplementation and Usage Details

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

20 Debug Requirements

Not applicable to this IP/IPSS

20.1 Visualization of Internal Signals Architecture (VISA)

This IP has VISA bus for debug.

20.2 IP Firmware Support for Messaging to North Peak

Not applicable to this IP

20.3 Trigger Events and/or Responses

Not applicable to this IP

20.4 Debug Trace Fabric (DTF)

Not applicable to this IP

20.5 Debug Power Domains and Clocks

Not applicable to this IP

20.6 Other

Not applicable to this IP

21 Fuses

Not applicable to this IP/IPSS

22 Pin List/Ball Map

Not applicable to this IP/IPSS

23 Wish List and Recommendations

Not applicable to this IP/IPSS

24 Opens

This section is a list of opens specifically for items that are marked as Opens throughout the HAS. This list functions like a table of contents.

To mark an item as Open, highlight the text that explains the open, and then select the Opens style from the Styles pane (if your Styles pane is not visible, press Ctrl + Alt + Shift + S).

To update the list of open items in section, right-click anywhere in the list, and then select Update Field > Update entire table. The list updates with all of the items.

24.1 Opens List

The following is a list of any open items throughout this document:

Note: Be sure to right click on the list and update fields when adding Opens to this document.

List of Opens

No table of figures entries found.