SOC-15 Specification

Scalable Data Port (SDP)

Rev 1.5.0, Nov 10, 2022

AMD Confidential – Internal Use Only

Table of Contents

List of Figures

List of Tables

Revision History

Rev	Date	Changes
1.5.0	10/31/2022	• Removed mention of a "second read response" channel in Section 2.2.5 – the feature was never used and is replaced by the common method for providing more than one of any channel type. • Added WbInvBlkAll to commands that may be used in the CPU save area (DESOCARC-2126) • Transition ChgToXNR and ValBlk to using the read response channel for all responses (DESOCARC-2058, DESOCARC-2164). • Added ReqAttr encodings for ChgToX, ChgToXNR, and ValBlk (DESOCARC-2058). • Marked system address range of FFFF_FFFF_0000_0000h to FFFF_FFFF_7FFF_FFFFh as reserved in this specification – it is used in data fabric to "remap" sections of the CPU save area to support more than 256 threads in a system (FFFF_FFFD_F7xx_xxxxh is still expected to be used by the CPU in issuing these transactions and this continues to support up to 256 threads behind a single SDP port). • New PrbAction command of 5h (previously assigned to unused IcInvalidate) for Rinse added (DESOCARC-2254). • Marked system address range (ReqSpecialAddr=1) of 0000_0000_0100_0000h to 0000_0000_01FF_FFFFh as reserved for SDMA usage (CNDI bus). • Updated steering tag to 7 bits (from 5) in ReqAttr. Removed unused Processing Hint. Removed ability to set Steering=1 on caching reads, it is only available on WrSized* commands (DESOCARC-2057). • Atomics may now specify a shadow tag update exactly like a write (DESOCARC-2094). • Renamed ReqNoAlloc to ReqCacheHint and extended to 2 bits (DESOCARC-2140). • Added new atomics "Unsigned Decrement Without Wrap" and "Unsigned Conditional Subtract" (DESOCARC-2212). Also renamed "Unsigned Clamping Increment" and "Unsigned Clamping Decrement" to clarify the intent of clamping. • Allow multiple instances of a single channel type (DESOCARC-2141). • Physical Channel Multiplexing is kept, although the feature is simplified. The read response channel and write response channel no longer have RdRspChanAB and WrRspChanAB (and the credits) as all intended uses are for multiple completers that are sharing a single port (DESOCARC-2059). • Define "Data Source Feature" as an optional feature that implements RdRspSrcData and WrRspSrcData (DESOCARC-2047). • Added Exclusive Lock Request Feature (DESOCARC-2186) including RdSizedDWX and WrSizedCndlDR. • Split Enhanced RAS Feature (parity on non-data fields) into three variants. V1 covers all fields except credit and RdRspDataStatus. In order to balance the parity tree more in the request channel, two parity fields are used – one for address and meta-address fields. V2 covers data status field and V3 covers credit fields. This allows for more selective implementations. DESOCARC-2262. • Added Metadata Compression feature (DESOCARC-2154) • Specification notes: ○ Marked command encoding 1ch as being used internally by the fabric. ○ Marked PrbAction Eh as being used internally by the fabric. • Omissions and clarifications from prior specifications: ○ Corrected 3.2.1 Byte Enable Compression – the byte enables are naturally aligned to the OrigData size so that OrigData[0] always aligns to a ReqAddr with the low-order bits ($\log_2$ of the width) equal to zero. This is considered an error in prior specifications as all implementations used this definition (DESOCARC-2052). ○ In Table 37 Atomic Operations, used "||" for boolean OR. ○ Included SpecPrefCmd in section 3.2.3 Read Response with NODATA. ○ Clarified ReqLen behavior on all cache block commands. ○ Clarified that retransmitted responses do require a separate data credit. ○ Clarified that a response acknowledgement with NODATA – the acknowledgement can be sent based on the RdRspStatus=OKAY_NODATA and does not necessarily have to wait for the lagged data beat (which would contain RdRspDataStatus=NODATA). DEDFRTL-12103/DESOCARC-2150. ○ Clarified that credit handling for a read response of NODATA is not treated differently from a read response with a single beat of data. ○ Clarified that data bytes with a byte enable of 0 still contributes to data parity. The specification now recommends providing a fixed value of all zeros or all ones in order to prevent potential security leaks from uninitialized data. ○ Removed unnecessary qualifier about victims in "writeback virtual channel" in 3.9.7 Probe and Response Interlocks for Single Cache Line Address (DESPCSOC-2043). ○ Clarifications in ReqPassPW and ReqRspPassPW – most notably to indicate that the bit

		must be carried by the fabric as payload in upstream non-posted/posted VCs. o Ordering section mentioned SpecDramRd was unordered but did not include SpecPrefCmd. It should have – all speculative prefetch requests are not ordered with prior writes etc (DESOCARC-1989). o Clarified intersection of Probe Request Compression (PrbCompType) and Multiple Probable Units (multiple PrbVld signals). (DESOCARC-2024).
1.4.1	4/1/2021	• Added note to command table for encodings 35h-38h that they are used internally by fabric. • Corrected Table 47 name. • In RdRspStatus definition, the rule that Nrdstat must be six bits for caching ports is exempted for ports that issue only RdBlkS (correction to the specification, always has been true). • N_{st} was omitted in Table 32 in previous specification. • DESOCARC-1976: RdBlk* may be requested for multiple cache lines. The implementation of this feature is described as an optional “Multiple Cache Line Feature” because only certain port implementations are allowed to request more than one cache line. • DESOCARC-1970: Expand the Completer Fatal Error Notification Feature so that it can use the Read Response Channel if the Write Response Channel is not implemented. • DESOCARC-1983: Add in Floating Point Atomic Feature. • Table 77 DVM Field Assignments has been restructured due to feedback to clarify the different formats in use between core and IOMMU. There has been no functional change due to this restructure. • Updated Table 17 for the write invalidation case as a correction to the DEDFRTL-8652 update in 1.4.0. The originator does not provide the way ID of the invalidated line during a write. Instead, the shadow tag is invalidated by finding the tag at the appropriate index. • DESOCARC-2020: Revamped Table 41 (splitting into Table 42 for ChgToX* and Table 41 for all other commands). This clarifies the ChgToX states that can occur due to a change (in DF4 only) where ChgToX can optionally return clean (memory) data. • DEDFRTL-8290: VA[46] must be sign extended for 57-bit VA cores that are in an SOC that does not support 52-bit PrbAddr.
1.4.0	9/18/2020	• DESOCARC-1753: Clarified that all reserved attribute bits must be zero. • DESOCARC-1904: RdRspParity (part of the enhanced RAS feature) should not include RdRspDataUser as it does not cover any data bits. This was an error in the specification. It is also clarify that RdRspDataParity does not cover RdRspDataUser - it just covers RdRspData as specified. An implementation could include a user pin that covers any necessary RAS features on RdRspDataUser; e.g. RdRspDataUser[0] is the parity of RdRspDataUser[n:1] • DESOCARC-1910: Implementation requirements led the LVT-IBS and LVT-Performance Monitor commands to be issued as WrSizedNC; not as WrNoDataNC as specified. The destination field is “reserved” as these LVT commands are currently used for “self-interrupts” only where the destination is determined by decode of the ReqUnitID field to identify the originating thread. • Removed any specific SDP requirements on what the steering tag encoding should be. This is an SOC-specific decision outside the scope of this architecture. Put some clarification into what is meant by “steering". • DESOCARC-1905: Significant clarification on how PrbChain works to deliver a single DVMOpMsg over two separate PrbAddr fields. Clarify that it is reserved MBZ for all other types of probe packets. • DESOCARC-1920: Add a requirement that SysMgmtJoin/Leave messages do not come between the two chained probes. • DESOCARC-1862: Error in table 50 - this address starts at 0x0000_0100_0000_0000, not at address 0. PrbAddr[n:40] is 0x0000_01 in the interrupt messages. • Fixed table 17. The encoding of 101b is theoretically possible only on a ClnBlkAll because the PrbAction=Clean may pass dirty data back to be written to memory. Other than the ClnBlkAll, it is not sensical to “pass dirty” to an originator and have the system state be exclusive! However, it is warned that originators may not rely on 101b encoding of ClnBlkAll indicating the state of the line at the time the command was issued, nor does it necessarily indicate the current state of the line in the system. A fabric is not required to indicate RdRspStatus[3]=1 if the line was cleaned during the process. • Clarified that PrbRD=00b is illegal in many cases and note this in Table 38. Also clarify legal values of PrbRD for table 37. • Removed striked-out cache state transitions (that were made illegal in previous specification revisions) from table 38. • DESOCARC-1843: Corrected specification - ReqAttr on Device Peer-to-Peer messages were user-defined and carried by fabric from source to destination. • Fixed issue where SDP specification mentioned the need for address-beat ordering for read responses, but omitted mention about the same need on the write requests. It has always been a requirement that SOC-15 fabric has all data transfer for a specific transaction in “address-order” [except for RdBlk* critical-word-first support]. • Removed old ARM encodings that are not used, including IcInvalidate (probe action), IcInvBlkAll

		(request command). Also removed references to an equivalent command in ARM. • Removed RdBlkNotO command encoding • DESOCARC-1860: Added in PrbDemote action (SDP encoding Dh) (Also DECHDG-2565, DEDFRTL-7453) • Removed more references to ARM in preparation for DVMOp changes • DESOCARC-1881: Added an ability to have either a PCID or an IOMMU DomainID in the DVMOp table 70 • DESOCARC-1931: Added 57-bit VA support into DVMOperation, which includes an unexpected swizzle in formats to support 57-bit while having only 52-bits of probe channel address/physical address bits (Also DEDFRTL-7883/DEDFRTL-7884). • DESOCARC-1873: Added probe compression feature including the width of the PrbCompIndex field, which is a new port width field • Removed ReqLock and EXOKAY - features only associated with Monitor Locks which are now deprecated. Also removed the “Monitor Lock” section • Noted that the Scalable Machine Check Architecture system address range (at FFFF_FFFD_F940_xxxxh) is now currently reserved as it is no longer used as of SMCA version 3. • Documentation for “CNDI” - Command Network for Distributed IPs" ○ New commands AddrXlateRdSz and AddrXlateWrSz, documented as Address Translation Services. ○ New section for “same-destination” ordering. Prior system management spaces used this ordering model even though it wasn’t documented, so this section is not just for CNDI. • DESOCARC-1933: Added new optional credit count fields. Expected to be limited use for some portions of the design that are credit limited\). • Enhancements to SpecDramRd. The legacy SpecDramRd is supported with ReqAttr=00h. SpecPrefCmd adds new features to prefetch and allocate, retrieve, and receive a response (with or without data). The use of the SpecPrefCmd is intended for MALL where the LLC can hold data while the memory subsystem is offline (e.g. due to power management) • DESOCARC-1941: New features added to SpecDramRd encoding (now using another name for SpecPrefCmd, although SpecDramRd is still in the specification as when ReqAttr=00) • DEDFRTL-8652: Shadow tag implementation • Corrected PrbDemote encoding from C-D as agreed. • Noted that ReqNoAlloc is ignored during SpecPrefCmd to rule out possibilities that the ReqNoAlloc and the ReqAttr may disagree on whether or not a cache line should be allocated. • Further clarification in SpecPrefCmd that TRANSERR may be provided (if a response is provided) for any reason that the line could not be prefetched. • Corrected name of signal – ReqSpecialAddr (draft incorrectly called ReqSysMgmt) • For AddrXlateWr, the draft had said that the ReqLen was expected to be 8B. This is wrong. This specification does not restrict the format or length of the accompanying data. • Some SDP ports may ignore the *ClkCtl for credit release and only use it to indicate full/partial disconnect. This nuance should be considered to be a port specific feature and the specification is clarified to indicate just that the port “may” look at the ClkCtl. • Further clarification that all the fields of a port should be ignored if Valid is not asserted. • Clarifications around RdRspStatus and additional encodings for AddrXlateRd.
1.3.1	10/7/2019	• DESOCARC-1744: Noted fields in ReqAttr victims that are used internally for fabric. These fields are reserved in SDP and the note is only to remind the architecture of the usage in cHT. • DESOCARC-1739: correct the ReqVirtAddr/ReqVfid/ReqVfidValid relationship – a physical address (ReqVirtAddr=0) may still have a virtual function ID associated with the request. • DESOCARC-1736: clarify the architecture in what may be observed on the port during a bus lock • DESOCARC-1735: Correct ChgToXNR in Table 45 and further clarify the possibilities of using ValBlk or ChgToX* from invalid state. • DESOCARC-1707: Remove constraint that OrigDataError must be zero for compressed commands. • DESOCARC-1696: Add optional ReqNoAlloc field including new SDP feature “External Last Layer Cache Support”. • DESOCARC-1692: Clarify ReqLen in SpecDramRd command. • DESOCARC-1680: Restrict usage of RdSizedNoWrite to bus-lock only cases. • DESOCARC-1633: Correct relationship between two split DVMOp chained probes and other probes, especially DVMOpSync. • DESOCARC-1628: Relax WrSized with ReqAttr[2:1]=b’10. As part of this, implement an “Ordered Cacheable Command” SDP feature that is grandfathered into all GFX implementations such that RdBlkS are ordered with respect to prior writes. • DESOCARC-1627: Update SDP bus lock addresses bits 23:8 • DESOCARC-1608: Allow ReqSecLevel to be variable number of bits. • DESOCARC-1592, DESOCARC-1600 and DESOCARC-1619: Add CPU/IOMMU flags to DvmOpReq and DvmSyncReq, support for x86 TLBI, IncludeNested. • DESOCARC-1584 and DESOCARC-1589: PCIe bus segment support.

		• DESOCARC-1579, DESOCARC-1590 and DESOCARC-1765: Add interrupt system addresses. • DESOCARC-1241: Document channel to channel dependency requirements in new section (Relationships Between Channels). • DEIPCIOH-1060: Specifically allow an originator/completer to complete port disconnection while credits are outstanding (and thus may have to reconnect to return them). • DEDFRTL-4516: Clarified originator data usage for both probe response data and write data. • DESOCARC-1760: Add method for a completer/fabric to pass SysFatalErr to an originator through an unsolicited response. This is under a feature define (Completer Fatal Error Notification) as it requires both sides to agree. Will not be implemented until Stones. • DESOCARC-1756: A completer/fabric is allowed to reuse a PrbTag as soon as the probe response is received even if probe response data is still pending. This is because the Originator Data Channel is untagged (relies on order). • DESOCARC-1761: Allow ReqQosPriority to be port-defined number of bits (for example two bits or four bits – the default being four). Also allow ReqQosForward to be omitted and treated as if it is always 00b. • In WrSized ReqAttr usage, note that cache state b'11 is intended for write-through caches that have already updated the cache contents. • Clarify that ReqLen of a full cache line size is required for ChgToX* and ValBlk commands. • Update Error Handling to state that an originator must not cache a line that has gotten back an error response (SLVERR, MSTABRT, TRANSERR, PROTVIOL). • Clarify that RAR and WAR ordering, while not guaranteed, is still legal for an originator to submit. Only the ordering between the two commands is not guaranteed. • Add in a definition of "reserved".
1.3.0	9/15/2017	• Add "multiple request address" feature (DESOCARC-1570) • Removed deprecated address ranges for DVM, bus-lock, and SysMgmtJoin/SysMgmtLeave. • Removed deprecated FFh broadcast encoding from the interrupt table.
1.2.1	9/15/2017	• Updated ReqAttr definition for WrSizedNC to clarify that it is reserved for all SOC-15 fabric ports (DESSPIODDV-371). Other ports may use this as a user-defined feature. • Documented new address ranges for DVM, bus-lock, and SysMgmtJoin/SysMgmtLeave (DESOCARC-1489). • Added DVM fields that are used for x86 mode (DESOCARC-1417). • Clarified that chained probes are two probes with the same PrbAction, but that other probes (with different PrbActions) may occur between the chained probes (DESOCARC-1515). • Documented interrupt vector probe messages (DESOCARC-1579). • Included SpecDramRd in the list of "unordered" commands. Also clarify that the critical byte could change between a SpecDramRd and a RdBlk, SpecDramRd effectively only specifies a cache line address. • Corrected device peer-to-peer message bit 28 from "system dependent" to "reserved". • Correct and clarify how byte enables are transferred when byte-enable compression is implemented (DESOCARC-1471). • Clarified that WrSized* ReqAttr[7:6] specifies the number of data byte beats (regardless of beats transferring compressed byte enables). Also corrected the encoding (DESOCARC-1444). • Updated hash start address to remove the secure kernel address as part of the message (DEZNRTL-6481). • Corrected command table which incorrectly stated that WrSizedFullComp used an acknowledge. • Added FFFFFFFFh as the broadcast encoding for NMI, SMI, INIT and LINT interrupts. This value is used in all Data Fabric 3.0 products (and FFh is *not* supported). The value of FFh is supported in Data Fabric 2.0 products, except ARI which does not support broadcast of these interrupts at all. • Clarify that byte enable compression did not affect data credits. • Noted that RdRspStatus does not contain the correct PassDirty/CacheState if the response ends in error. For example, a RdBlk that ends in a protection violation has not probed peer caches. • Updated AckCancel rules on VicBlkCln used when page probe filter is used. • Noted new rules on AckCancel for a VicBlkCln that has retained a copy. This is necessary for SSP probe filter (but not used actually by the originators as they do not retain copies on VicBlkCln). • Outlaw all probe state transitions that were "optional" and ended up invalidating the line. Only invalidating probes should invalidate the cache. This is necessary for SSP probe filter. • Correct WrSizedFullComp length requirements (DESOCARC-1449) • Add in implementation notes on the non-caching read and write requirements for SOC-15 fabrics.
1.2.0	6/17/2016	• Remove OrigDataDbgMsk and RdRspDbgMsk from specification (DESOCARC-1334). • Remove implementation note that {DATERR, VALID} and {VALID, RETRANSMIT} are not used on SOC-15. SDP 1.2.0 makes these sequences legal (DESOCARC-1300). • Add Low-Power Signaling feature and the *ClkEn_m1 signals (DESOCARC-1325). • Allow WrSizedFull to be an integer number of data beats, not just cache line size (DESOCARC-1338). • Add PCOMMIT system management command (DESOCARC-1273).

		• Add byte enable compression feature (DESOCARC-1342). • Add data compression feature (DESOCARC1343). • Allow there to be multiple probe units behind a coherent agent with separate PrbVld, PrbRdy and PrbRspVld signals (DESOCARC-1345). • Use an encoding of {State, PassDirty} on invalidating/store probes to indicate a hit or not (DESOCARC-1326). • Change ChgToX cancel indication to mean that the line was I just before the response (DESOCARC-1327). • Add WBNOINVD command (DESOCARC-1272). • Add ReqAttr definition on VicBlkClnD to specify the state of the line (DESOCARC-1275). • Add ReqAttr definition on a RdBlkL to indicate that it is a refetch after a storing probe (DESOCARC-1344). • Add a RdRspStatus of OKAY_NODATA and use this for ChgToX without data (replaces the overload of PassDirty) and for responses to RdBlkL that were refetches after a storing probe (DESOCARC-1344). • PCI configuration and APIC accesses may now be up to a cache line (DESOCARC-1346). • Remove certain probe cache state transitions that were unused (DESOCARC-1339). • Add in a heads-up feature with accompanying RdRspDataVld (DESOCARC-1354). • Clarify ReqIO and allow CPU originators to have this bit set for system messages.
1.1.2	5/16/2016	• Add section 3.3.1 to document writes and atomic rules w.r.t. posted and write back channels. • Add WrSized with ReqAttr[2:1] = 10b (known invalid) to a list of commands that must be serialized in 3.7.1 Request Ordering. Also clarified throughout that when an originator must wait for a response to be returned; that this is the last beat. • Clarified wording in the definition of RdRspStatus over multiple beats. • Removed the marked out (deleted) lines in Table 47 Probes Issued in Response to Commands • In “ReqTag”, removed the words “or EARLY” because this status is not in the RdRspDataStatus, so it was inappropriate. • Fixed rule for PCI I/O (Table 76 PCI Legacy I/O Port Region) that at completers, must not cross double word boundary. • Clarified use of NODATA transaction due to escape DEFCHTA-410 • Added statement about ReqAttr[7:4] for single DW writes in ReqAttr section as well. • Clarified headings on Table 59 Message Type Interrupt Encoding. • Added an implementation note that stated SOC-15 data fabric does not allow out of order data beats • Correct error in FinalState for VicBlkCln(D) commands - 11b was for F-state, not O-state. • Include requirement that byte enables must be zero for lanes that are outside the address/length • Include statement that byte enables may be sparse and zero-byte writes are allowed. • Corrected command in Table 80 Distributed Virtual Memory Region (DESOCARC-1190) • Update Table 64 X86 Legacy Wire Region and Table 65 x86 Legacy Wire Address Definition for WBINVD. • Clarify that a port is not expected to de-assert Vld before observing Rdy except for a reset (DESOCARC-1201) • Clarified that ReqTag is not meaningful for SpecDramRd. • Change global visibility rules for Send Event (DESOCARC-1195) • CLZERO to APIC/PCI config is possible, so fabric may observe a WrSizedFull to these regions. They are not legal software though. • CPUs may give coherent commands to the APIC and PCI configuration spec, update Table 67 Secure System Management Hub Registers and Table 75 GPU Virtual Wire Region (DEDFRTL-1093) • Note the range of system addresses that has been reserved for core to L3 register communication in Table 68 Secure Core and Cache Controller Registers (DESOCARC-1203) • RETRANSMIT responses should not have dependency on probe channel (DEASRTL-8074). Also addressed a small gap in the wording for single cache line interlocks. • More clarity on when RdRspStatus[5:3] are used and remove the implication in Table 45 Originator Request Cache State Transitions that RdRspStatus[5:3] are really meaningful for RdSizedNoWriter. • Minor nit on Same-Address Ordering, as they only apply to valid reads/writes. • Corrected ReqVC, ReqPassPW and ReqRspPassPW values when the field is not implemented. • Note that DVM synchronization requires commands and possibly prior DVM ops and bus looks to complete. • Crisp up statements of which signal (RdRspVld, or RdRspDataStatus) qualifies the data sub channel. It is the RdRspVld signal (or an equivalently lagged RdRspVld signal) - DESOCARC-1280. In version 1.2.0, it will be required that originators ignore RdRspDataStatus when the bus is not active. • Add RAS section for uncorrected/poisoned data.

		• Added direly-needed definitions of the error responses. • Document with EPC that the credit release can only be requested after a full disconnect. It was never intended for one-sided disconnects. • Rewrite of the port control section to make it clearer and reduce redundancy and repeated information. • Clarify rules for full-disconnect w.r.t. quiesce (DESOCARC-1310). Clarify rules for what you are allowed to do after observing a side drop *ClkReq. Clarify that once the *ClkReq has dropped, the signaling is inactive, not just when *ClkAck has dropped. • Minor clarifications of probe state transitions. • Added SMU save/restore feature for RavenRidge (DESOCARC-1184). • Included missing {DATERR, RETRANSMIT} combination in Table 37 Read Response Data Retransmission Possibilities. Removed incorrect implementation note that {DATERR, VALID} was not used in SOC-15. • Correct and further clarify critical byte addressing (DESOCARC-1368) • Clarify that the interrupt region of system messages still follows the normal SDP rule for selecting write data bytes using ReqAddr even though ReqAddr[7:4] contains a message type for interrupts.
1.1.1	4/28/2015	• Corrected address typos in Table 66 and Table 67. • Deleted “Startup” from a list of valid message types for interrupts (Table 59). The startup encoding is used internally in the data fabric, but startup messages cannot be sent from SDP. • Corrected typo in Table 11. • Completed definition of SMIACK in Table 64. • Updates throughout to remove the undefined term “back-probe” and clarify when the originator of a request is probed or not. • Updates to Table 45. Removed unnecessary “final states” that are covered by silent drop rules. • Clarified that CMPTO and CRS are only used for PCI attached ports and not used by the SOC-15 data fabric. • Corrected Figure 18. • Clarified boundary and length crossing for PCI I/O and configuration accesses. • Clarified writeback virtual channel in Table 38. • Clarified ReqTag reuse if there is a delay between the response header and data. • Updated section 3.9.7 to state that this does not hold for victims in the writeback virtual channel (UBTS 598159) • Remove RdBlkX from an O state. An originator may not send any block read command when it has dirty data (UBTS 597720) • Highly discourage certain F->F, F->I, O->O and O->I transitions in Table 48 as they are incompatible with probe filter directories. This is in preparation to possibly drop these on revision 1.2.0. • Add implementation note for VDCI empty signals in port disconnect (UBTS 589165). • Updated legal commands for CPU save state area in Table 55 (UBTS 598799, 598925).
1.1.0	1/16/2015	• Went to three-digit revision naming (major.minor.update). • Allow Port Control signals to be asynchronous to SdpClk (UBTS 571371). This adds a feature – asynchronous port control. • Add in a new section (Probe and Response Interlocks for Single Cache Line Address) guaranteeing that an outstanding probe does not collide with a new probe or a new response to the same address (UBTS 580003). • Clarified that write responses may not be given prior to receipt of the data; even in error detection cases. • Added hash received status to Table 74. • For I/O interfaces, HASH_START and HASH_STOP transactions are changed to WrSizedNC of one doubleword. This is due to the fact that they are sourced to the LPC bus, which does not support WrNoDataNC. There is no architectural reason for them to carry data. Also corrected typo on secure kernel range (it is 64KB aligned not 32KB aligned) and clarified the two commands that are CPU only. • In AckCancel, removed incorrect stale statement that WrSized*S commands issued a response acknowledge and used AckCancel. • Corrected typo in Table 60 End of Interrupt (EOI) Range. • Updated Error Responses and Unsupported Command/Action Encodings to include DVM PrbAction. • Clarified that OrigDataVC should match ReqVC on write data and is reserved for probe data. • Correct typo on final cache state for VicBlkFull/VicBlkPtl with ReqAttr[1:0] = 11b (UBTS 578822) and added this case (which was missing) to Table 45. • Clarified what it is meant that *Rdy signals may be ignored when clock synchronization logic is done by pervasive logic outside of the IP or fabric. • Enhance Table 24 by adding in PassDirty state for clarity. • Clarified Table 45 to indicate that only WrSizedFull can allocate from I->S. • Clarify how enhanced port control feature handles credits if one side requests a reset and re-issue

		but the other side does not (UBTS 576244). • Clarify the requirements for intervening probes and AckCancel with evictions and downgrades. • Corrected typo in Table 69.
1.00b	10/29/2014	• Clarified that cache maintenance commands are examples of an operation with "write semantics" (and thus length is reserved/zero). • Corrected typo in Table 75 (UBTS 569386) • Added message type encoding (Table 59) • Indicated another 4MB region for secure system management hub registers in Table 67. • Moved 8MB address space at FFFF_FFFD_FB80_0000h from CPU (EL3 boot code), which is unused to System Management Controller Private Space. The system management controller (aka SMU) is not the same as the "system management hub" (aka FCH). UBTS 577219.
1.00	9/2/2014	• Clarified that DVM "both" means guest+HV (but not EL3). EL3 is ARMv8 only (ARM support question 592571). • Corrected typo in address for CPU save state area and typo in the size of the system address space. • Corrected typo in EOI address range • Corrected two places where ErrEvent was listed as ErrorEvent • Add note in IcInvalidate probe state transition that the state transition and the status applies only to instruction caches. Data caches can still have the line and the status does not indicate this. • Correct typo of AtomicNCNR • Clarify in writes, APIC ranges and interrupt ranges that a response is not returned until the operation has been made visible to all caches (if coherent), APIC side effects have been performed, and any interrupts have been delivered. • In Table 44, split RdBlkL and RdBlkNotO command to be clearer on which states are illegal. • Removed incorrect statement " At completer ports, the port assumes [ReqRdy is high] whenever it has asserted CompClkAck." • In "Interface Signals", clarified that requests and probe channels are also tied to the originator write/probe data channel for credits. • Added SpecDramRd to Table 44 Cache Line State Attributes and Table 45 Originator Request Cache State Transitions. • Added PSP->FCH secure address range for compatibility reasons. • Improved definition of the data parity bits. • Addressed conflicts between the reset signaling and the port control sections. • System address region is sign extended from a smaller width address to a larger width address only if bits n:34 are all ones. • Clarified that if (for example), PrbAddr[48] is used in DVM but not memory coherency probes, that bit is always zero for memory coherency probes. For system management addresses, that bit would be sign-extended as normal (system management PrbAddr[48] taken from ReqAddr[47]). • Allow VicBlkCln to addresses that were ReqIO = 1 during the fetch (UBTS 563866). • Solve the *Rdy controversy by noting that: ○ In some high-frequency implementations, *Rdy may be pipelined (provided in advance of the actual cycle). This is introduced by a port parameter. ○ Noting that when using a clock synchronizer, the originator/completer may not be able to de-assert *Rdy in order to provide back-pressure because the back-pressure is reserved for the clock synchronization logic.' • Noted that the requirement to not present data before the request/probe response is only relative to the valid signal. If the originator presents both data and request on the same cycle but the completer/port has only OrigDataRdy asserted, this is not a violation of the architecture (UBTS 560443). • Clarifications in ordering section without changing architecture/intent. Specifically noted that ReqBlockLevel=0 requests are not blocked when other requests (with non-zero ReqBlockLevel values) are blocked. Specifically mentioned that RAR and WAR hazards are not maintained by SDP. • Expanded error handling section. • Let probe response data and write data share a common pool (UBTS 561363). • OrigDataDbgMask and RdRspDbgMask must be the same for all data beats (UBTS 563121). • Removed incorrect implementation note that incorrectly stated the SOC-15 fabric restricts data interleaving at cache line boundaries. • Added table for valid interrupt destinations. • Corrected statement made in introducing system messages. • Clarified implementation statement for Mp that SOC-15 fabric does not even support this state currently (previously just said that no originators were not using it) • Removed possibility of O->I transition on Migrate probes as an optimization for data fabric. • Removed possibility of giving up writeback data on NOP, share or fetch probes as data fabric does not support this, nor is it expected to be used by any originator. • Clarified Table 47 for what the recommended cache state transition is when more than one is listed. • Clarified Table 69 that firmware-only space was just for the special firmware APIC registers.

		• Corrected typo in Table 47 – WrSized does not allow S state to maintain at probed caches – only the originator is allowed to maintain S.

Preface

About This Specification

This specification describes the Scalable Data Port (SDP) architecture.

The SDP architecture (SDP) defines an IP block attachment interface primarily intended to be utilized for intra-SOC networking between processor IP blocks (CPU, GPU, multimedia, display controllers) , system memory, and I/O hubs. Signals are grouped by function in named fields; fields are logically bundled into physical channels.

SDP specifies a split command / response transaction protocol. The protocol supports cache-coherent and non-coherent transactions. A set of commands and transactions are defined that support the requirements of x86 and graphic processors. Command attributes support PCI-e ordering rules.

Although the specification defines aspects of the link-level signaling protocol, the interface is defined at an abstraction level above that of actual digital logic design. Implementation details such as voltage levels used to represent the assertion or negation of a logical value are not specified. Link-level pacing and credit-based transport-level flow control are provided. State machines are specified at a conceptual level and will require additional implementation details supplied by the IP designer.

The SDP architecture can be applied to the design of a wide range of systems that have a single physical address space and utilize read/write semantics.

To understand how this standard is applied in the RTL-level implementation of a specific device interface, consult SOC-specific micro-architecture documentation.

Please direct comments and corrections to Eric Morton (eric.morton@amd.com). Specification bugs can be opened in Jira under project DESOCARC, Category: SDP Specification.

Audience

This specification is intended for use by SOC and IP block architects and designers; SOC emulation, test, validation, and verification engineers; and system management controller firmware engineers.

Conventions and Terminology

Conventions

The following conventions are used in this specification:

Convention	Meaning
0100b	A binary value. In this example, a 4-bit value.
128	Numbers without a suffix are assumed to be decimal, unless the context indicates otherwise.
F0EA_0BF1h	A hexadecimal value—in this example, a 32-bit value. Underscores are added for readability.
7:4	A bit range. In this example, from 7 to 4, inclusive. The high-order or leftmost is listed first followed by the low-order or rightmost. Commas may be inserted to indicate gaps.
RdRspTag	A field within a signal group or channel.
ReqAddr[39:5]	A bit range within a field. Most significant bit is listed first. Gaps are indicated by commas.
{ReqCmdExt, ReqCmd}	A field composed of the concatenation of the ReqCmdExt and ReqCmd fields.

Specialized Terminology

The following terms are used in this specification:

Term	Definition
Fabric	A communication network that routes commands from originator devices to completer devices in a system. The fabric may modify commands and can function as a proxy for a completer. The Scalable Coherent Data Fabric is an implementation of a fabric that is compliant with this specification.
Field	A group of one or more signals that share a name and encode a specific piece of information. Signals within a field are numbered according to binary significance.
Device	A logic block (IP block) that can initiate or respond to transaction-level protocol commands. A device attaches to the fabric via one or more ports.
Port	Logical definition of the interface by which devices attach to the fabric. The port definition is standardized, but is configurable in terms of the widths of various fields and capabilities to match the requirements of a particular device.
Unit	A functional block within an originator device that can act as the initiator of a transaction-level command.
SOC	System on Chip. A die or packaged component that typically incorporates functions previously implemented using

	multiple packaged logic components.
UnitID	Identifies a functional unit within an originator that is the source of a command. The transaction ID (tag) space is unique for each unit within each device.
Completer	A device that is capable of responding to a transaction protocol command. The fabric can also respond to a command as a proxy of the completer using cached information.
Originator	A device that serves the role of initiating transactions by issuing commands. A response is routed by the fabric to the originator of the command.
Reserved	A field or encoding that is unused and may be used for future enhancements. Unless otherwise specified as unpredictable, a field that is reserved is **required** to be set to zero by all originators and completers, however it is **recommended** that a receiving agent **does not depend** on the value being zero. These fields are reserved for future enhancement and backwards compatibility is maintained if and only if the originator does not depend on the field. Encodings that are listed as a reserved **must not** be used. Agents are allowed to treat receiving a reserved encoding as an error (see Error Handling).

Data Types

Data Types used in this specification follow x86 conventions:

Data Type	**Width**
Word	Two Bytes (16 bits)
Doubleword	Four Bytes (32 bits)

SDP Features

Due to differing requirements, specific devices may not require certain features of the SDP architecture. Devices that do not require the functionality of a named feature may elect to not implement the fields that support that feature.

Not implementing an optional feature may restrict the types of protocol commands the IP block can generate or features of the transaction protocol it can support. Implementations must deal with the possible signal mismatch between the device and the port that may result when optional features are not implemented. Resolving signal conflicts at the interface requires specific agreements between the IP blocks that connect to each other via the port interface and/or special provisions at the SOC-level. These agreements will be documented in SOC product micro-architectural specifications.

Signal Names

This specification assigns field names as a mnemonic device to aid in the understanding of the function of each field. Signal names used in RTL design and in simulation / verification tools to instantiate the architecturally defined fields must be more specific and generally include prefixes or suffixes that indicate the driver and the receiver as well as the model hierarchy level.

This specification uses "CamelCase" to assist comprehension of the function of a field. For example, the request address field is named "ReqAddr" which is easier to read than "reqaddr." Signal names used in implementations should include a contiguous string of characters that match the field names specified herein; however, the capitalization may differ due to syntactic rules of the logic design, simulation and verification tools utilized.

References

Table 1 References lists related documents.

Table 1 References

Document Name	**Location/Order Number**
TBD	

1 Introduction

This topic introduces the Scalable Data Port (SDP) architecture.

This introduction is presented in the following sections:

- Overview provides an overview of the architecture
- System Context establishes the system context for the architecture

1.1 Overview

The SDP architecture defines a logical signaling interface and a protocol by which devices can exchange data and control information through a variety of command and response messages. SDP defines both a link-level and a split request / response transaction-level protocol.

A port is the signaling interface, as defined by this architecture, that allows independently designed logic blocks (IP blocks) to communicate with each other in a structured way either directly or via a communication network (fabric). The definition of the port includes the enumeration and definition of required signals and the signaling regime at a link level.

The link-level protocol provides:

- Multiple physical channels for specific types of request / response messaging and data transfer
- Simple synchronous interface with link-level handshake signals for clock rate matching
- Balance between signal count and the bandwidth requirements of the physical channels
- Port clock request / acknowledge handshake allows either IP block to request the interface be brought up to an active state or quiesced for clock and power gating

The transaction-level protocol provides the following features and benefits:

- Split request / response messages improve utilization of fabric bandwidth
- Per unit transaction identification tags support multiple outstanding requests for each initiator within an attached device
- Cache-coherent memory read and write operations without requiring the originator to generate probes and consolidate probe responses
- Coherent and non-coherent reads and writes
- Support for PCIe protocols
- Support for graphics, x86, and PCIe ordering rules
- Atomic memory operations
- Virtual channels with independent credit-based flow control
- Quality of service hints
- Transaction security privilege level

Data transfer is affected through request / response message pairs. A request and its subsequent response forms a ***transaction***.

A device that initiates a transaction by making a request is called an ***originator***. A device that is capable of responding to a request is called a ***completer***. Requests are also called commands. The terms command and request are used interchangeably in this specification.

The port definition specifies the signals that provide the standardized attachment method for devices. The link-level protocol defines the signaling method used for information transfer and the link-level transfer handshake protocol. Because the scalable data port is designed primarily for use on-chip in applications where the linked IP blocks are physically close together, the interface is relatively wide and signaling is synchronous with respect to the SDP clock.

When IP blocks are not connected directly point-to-point, the fabric routes request and response messages between ports. This allows IP blocks to be less affected by the size and connectivity of the SOC as well as its placement in the SOC. In addition, the IP can also be isolated from SOC-specific clocking and tester determinism issues. **Note**: Physical implementation choices of an IP and the SOC floor plan will affect the power, area and performance of the fabric in providing this isolation.

The transaction protocol defines the requests that can be issued by an originator or by the fabric and the responses that must be returned to the originator or fabric for each request type. Responses are presented to the originator by the fabric, which serves as a proxy for the completer. Logic within the fabric implements all the messaging required to maintain cache coherency for those caches in the designated coherency domain. This allows the design of IP blocks that contain caches that participate in the system cache coherency protocol to be agnostic to the details of the cache coherency messaging protocol.

The non-coherent protocol is a subset of the coherent protocol to increase the ease of verification.

SDP supports the goal of independent IP block design and verification. Together the link and transaction-level protocols isolate IP blocks from many of the details of coherent and non-coherent data transfer over the fabric. The architecture provides a clean separation between the fabric and the IP blocks attach to it. The port interface is well defined and standardized. This allows the port interface logic to be reused by IP blocks.

Verification is simplified through the use a common set of collateral (BFMs, monitors, etc.) to verify both the fabric and IP block operation to the ports. Multiple fabric implementations can be created, not only varying the number of the ports supported, but also the coherency protocol itself (either to support differing bandwidth, area, and power requirements within a generation, or to evolve functionality over time) without impacting the design of the IP blocks.

Traditionally IP blocks have been required to implement complex protocol state machines to manage the generation and tracking of various coherency messaging packet types. The SDP architecture moves this protocol block into the fabric and makes it common among IP blocks, eliminating redundant work. IP block interface design is simplified since it only needs to comprehend the port link-level and transaction-level protocol, not the coherency protocol.

1.2 System Context

The SDP architecture is designed to provide a rich set of capabilities that supports the implementation of fully cache-coherent systems using the interface to attach multiple devices to a shared system memory.

To explain the full capabilities of SDP it is best to describe it in the context of a cache-coherent system with a mixture of types of initiating devices and responding devices. Figure 1 Scalable Data Port System Context shows the various types of devices that exchange information using the protocol defined by this architecture.

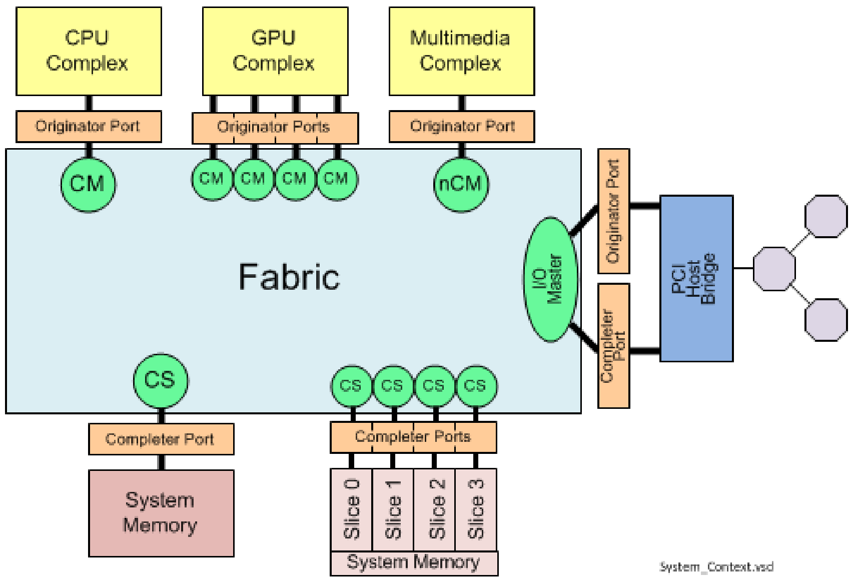

Figure 1 Scalable Data Port System Context

This example system includes four devices capable of initiating transaction requests (CPU Complex, GPU Complex, Multimedia Complex, and a PCI Host Bridge) and three devices that can be the target of requests (PCI Host Bridge and two regions of system memory). Note that the PCI Host Bridge shown in the diagram both initiates and responds to transaction requests. One of the memory devices (labeled System Memory) is attached via a single port. The second address-sliced System Memory block is included in the diagram to illustrate that memory can attach to the fabric via multiple ports to provide increased data transfer bandwidth.

Generically, devices that initiate transactions are called ***originator devices*** (or simply ***originators***). An originator device attaches to the fabric via an ***originator port***. Originators are further classified as either ***coherent*** or ***non-coherent***. A coherent originator issues cache-aware commands (such as RdBlk or VicBlkFull) and generally includes a cache that is visible to the cache coherency mechanism and participates in the system-wide cache coherency protocol. Non-coherent originators can issue coherent requests (commands that cause the probing of other caches) , but they do not contain a cache that participates in the coherency protocol and do not issue cache-fill commands.

Devices that exclusively respond to requests and do not initiate transactions (in the figure, the System Memory blocks and the completer channel of the PCI Host Bridge) are generically referred to as ***completer devices*** (or simply ***completers***). These devices attach to the fabric via a ***completer port***.

The architecture does not specify the topology of the fabric itself, but it does rely on assumptions about capabilities of the fabric to possibly alter request messages and satisfy system memory read requests using cached copies of data rather than obtaining the data from system memory.

Different devices that attach to the fabric have different roles in the system and require different types of information to be conveyed to and from the fabric via the port. To accommodate this requirement, SDP ports are defined to have optional ***features***. A feature provides a specific function or capability. When a device requires the function, it must implement ***all*** the signals that are defined for that feature. For example, some originators do not implement virtual channels. These originators are not required to implement any of the fields (such as ReqVC) that make up this feature.

All the features defined by SDP are described in Port Features.

In the figure, the PCI Host Bridge is unique in that it can both initiate and respond to requests. Such a device attaches to the fabric via a pair of ports—an originator port and a completer port.

The originator initiates a transaction by issuing a request to read or write a target location in system memory. This request is submitted to the fabric via the originator port. The fabric then takes over the responsibility to satisfy the request either by accessing a copy of the data in an internal cache, finding a copy of the requested data in one of the caches in the coherency domain, or by reading or writing system memory. As part of this process, the fabric may modify the request and generate its own requests to probe other caches within the coherency domain. The fabric consolidates probe responses and responds back to the originator to complete the transaction.

The protocol supports virtual channels (VC) and allows originators to impose specific intra-VC and inter-VC order constraints.

An originator may include more than one functional unit. The protocol supports multiple outstanding requests from each functional unit by assigning a unique UnitID to each requester within the originator.

Transactions are allowed to complete out of order. Transaction tags allow the originator to pair requests with subsequent responses. Both read and write requests require a response.

Each device attaches to an agent within the fabric. In the diagram, CM refers to an agent that proxies for coherent clients (those that contain a hardware-coherent cache), nCM refers to an agent that proxies for non-coherent clients (those that do not contain a cache or only contain software-coherent caches), and CS refers to an agent that proxies for memory that needs to be kept hardware-coherent while nCS proxies requests for memory that does not have coherency needs. One special purpose of nCM+nCS is a combined I/O agent as shown in the diagram. The fabric consolidates probe responses from system caches in the coherency domain and presents a single result to the attached originator via the coherent originator port.

This specification does not define the mechanism by which request and response messages are routed between ports over the fabric. Any translation, encoding, and encapsulation of request and response messages required to reliably and securely deliver these messages between devices is handled by lower layers of the fabric and is covered by other documents.

The SDP architecture recognizes and supports the specification of a coherency domain for cache coherent memory accesses. Each transaction request includes the specification of the domain of the caches that must be probed as a side effect of a cache coherent memory read or write request. Not all domains are applicable to every request.

The remainder of this specification is organized in the following topics:

- Port Definition describes the physical aspects of the port interface.
- Transaction Layer describes the transaction-level protocol.

2 Port Definition

The port definition specifies the name and purpose of each field within a channel, the constraints imposed by the link-level protocol on the timing of state changes of the signals, and the latitude granted to implementations in terms of the widths of specific fields.

The following topics describe the SDP physical interface:

- Port Features defines a number of optional features of the port definition.
- Interface Signals defines the fields that comprise the SDP physical interface.
- Port Parameterization defines the dependency relationships between the widths of fields that are defined to have variable widths.
- Signaling Requirements describes the constraints placed on the signals and on the timing of state transitions of these signals.
- Port Control describes the signaling protocol used to activate and disconnect the interface.

SDP comprises eight physical channels, of which each channel may have multiple instantiations. Each channel instantiation carries a specific type of information in a specific direction. For example, the Request channel carries transaction requests from the originator through the interface to the system.

The Transaction Request, Originator Data, and Read and Write Response channels support up to eight virtual channels (VCs). Virtual channels layer on top of the physical channel to provide request message passing guarantees (for deadlock prevention) and support required ordering rules. Each channel that implements flow control includes a dedicating credit release channel that carries flow control information in the opposite direction. Flow control is per virtual channel for channels that support multiple VCs.

2.1 Port Features

SDP defines the following features:

Asynchronous Port Control Feature

When implemented, OrigClkReq, CompClkReq, OrigClkAck and CompClkAck are asynchronous to SdpClk. This is recommended only for implementations that are sensitive to the latency of SDP port reconnection in low clock frequency states.

Byte Enable Compression Feature

When implemented, the originator write/probe data channel implements a single OrigDataBytEn and instead carries the byte enables on OrigData. See Byte Enable Compression.

An implementation that supports Byte Enable Compression Metadata Compression may not implement Metadata Compression.

Cache Tag Tracking Feature

When implemented, the originator provides cache way and allocation information on request channel. This allows the completer to track cached lines using “shadow tags” to limit probing. Requires implementation of the Probe Support Feature. Requires ReqStWayId, ReqStUpdate and ReqStInvalidate.

Credit Count Feature

When implemented, allows the port to have multiple credits returned at a time. Requires implementation of ReqCreditCnt, AckCreditCnt, RdRspCreditCnt, WrRspCreditCnt, PrbCreditCnt, PrbRspCreditCnt, and OrigDataCreditCnt. The width and availability of a credit count on each channel is independent.

Completer Fatal Error Notification Feature

When implemented, allows the completer to utilize a Write Response Channel (or a Read Response Channel if and only if the Write Response Channel is not implemented) for unsolicited (and credit-less) method to communicate a system fatal error.

Data Compression Feature

Adds four new commands – VicBlkFullZero, VicBlkFullComp, WrSizedFullZero, WrSizedFullComp and a method to compress the data.

Data Source Feature

When implemented, the completer utilizes RdRspSrcData and/or WrRspSrcData to communicate the source of the data or response. The two fields may be implemented independently, and it is not required that an SDP has both RdRspSrcData and WrRspSrcData.

Enhanced Port Control Feature

Requires implementation of the OrigClkCtl and CompClkCtl signals. The feature allows an agent to request that credits be re-initialized after a full disconnect. The feature also allows an agent to signal a hint for a one-sided or full-sided disconnect.

Enhanced RAS (V1, V2, V3) Feature

Enhanced RAS is split into three versions to allow independent parity on different fields. The three versions can be implemented independently and without prerequisite.

Enhanced RAS V1 Feature requires implementation of the ReqParity, ReqAddrParity, , RdRspParity, WrRspParity, , AckParity, PrbParity, PrbRspParity, and OrigDataMetaParity fields.

Enhanced RAS V2 Feature requires implementation of the RdRspDataStatusParity fiekd.

Enhanced RAS V3 Feature requires implementation of the ReqCreditParity, RdRspCreditParity, WrRspCreditParity, and OrigDataCreditParity fields.

External Last Layer Cache Support

Requires: ReqCacheHint (previously known as ReqNoAlloc).

Floating Point Atomic Feature

Enhances the Atomic* commands to include additional ReqAttr encodings for floating-point atomics.

Heads-up Feature

Requires the implementation of RdRspDataVld. Without this feature, N_{lag} must be zero.

I/O Stream Feature

Requires: ReqStreamID. Must also implement ReqBlockLevel = 01b (Stream).

I/O Space Feature

Requires: ReqIO field.

Low-Power Signaling Feature

Requires implementation of *ClkEn_m1 to be used for enabling clocks. In some implementations, clock enables may be logically combined by ORing them together.

Metadata Compression Feature

Requires implementation of ReqCompMode, ReqSpecDataFetch, RdRspDataCompMeta and additional attribute encodings on certain requests. The use of OrigDataBytEn also shifts for compressed writes. See MetaData Compression.

An implementation that supports Metadata Compression may not implement Byte Enable Compression.

Multiple Request Address Feature

Adds in a field ReqSubAddr that specifies some (but not all) address bits for an originator to “gang” two different addresses into one request.

Multiple Cache Line Feature

When implemented, the originator may issue caching read commands for multiple cache lines (or a single cache line) as specified in ReqLen. The address alignment and number of cache lines requested may be restricted by the implementation. This feature does not apply to cache update and cache eviction commands.

When this feature is not implemented, all caching read commands must be issued for a single cache line.

Ordered Cacheable Command

When implemented, the originator restricts all block read commands to just RdBlkS and requires the completer to implement read-after-write ordering for this cacheable command and prior WrSized* commands.

Physical Channel Multiplexing Feature

Requires implementation of the ReqChanAB, ReqCreditChanAB, OrigDataChanAB, and OrigDataCreditChanAB signals.

Posted Write Ordering Feature

Must implement: ReqPassPW, ReqRspPassPW, RdRspPassPW, WrRspPassPW.

Probe Chain Feature

Requires: PrbChain field and implementation of Probe Support Feature.

Probe Compression Feature

Requires: PrbCompType and PrbCompIndex and implementation of Probe Support Feature.

Probe Interrupt Delivery Feature

There are two versions of this feature. Version 1 adds in probe addresses that can be used to deliver vectored interrupts to processor threads. Version 2 removes the first set of probe addresses and adds in new addresses that can be used to delivery any type of vectored and non-vectored interrupts to processor threads. Version 2 also enhances the interrupt region to allow a processor to deliver LVTs, and adds an in-service region to allow a processor to acknowledge the previously delivered interrupt vectors. Requires implementation of the Probe Support Feature.

Probe Support Feature

Requires implementation of: Probe Request and Probe Response channels. Requires the implementation of OrigDataChan, OrigDataPrbCreditRel on the Originator (Write/Probe) Data Channel.

Read Response Feature

Used by originators that issue commands that respond in the Read Response Channel. Requires implementation of the Read Response Channel. This feature is only omitted for “write-only” ports.

Response Acknowledge Feature

Requires the implementation of all fields of the Response Acknowledge Channel and the ReqAck field.

Secure Transaction Feature:

Requires implementation of ReqSecLevel.

Trusted Memory Feature:

Requires implementation of ReqTMZ as well as the secure transaction feature.

Variable Heads-up Feature

Requires RdRspDelay field. Requires the heads-up feature as a pre-requisite.

Virtual Address Feature:

Requires implementation of ReqVirtAddr, ReqVfidValid, ReqVfid.

Virtual Channel Support Feature

Requires the implementation of: ReqVC, ReqCreditVC, ReqCreditType, RdRspVC, RdRspCreditType, RdRspCreditVC, WrRspVC, WrRspCreditVC, WrRspCreditType, OrigDataVC, OrigDataCreditType, OrigDataDreditVC.

Write Response Feature

Used by originators that issue commands that respond in a Write Response Channel. Requires implementation of the Write Response Channel. This feature is only omitted for “read-only” ports.

2.2 Interface Signals

This topic describes the signals that make up the SDP interface.

Signal Groups

The SDP interface is organized in the following groups:

- Port Control signals
- Command / Request channels
- Data channels

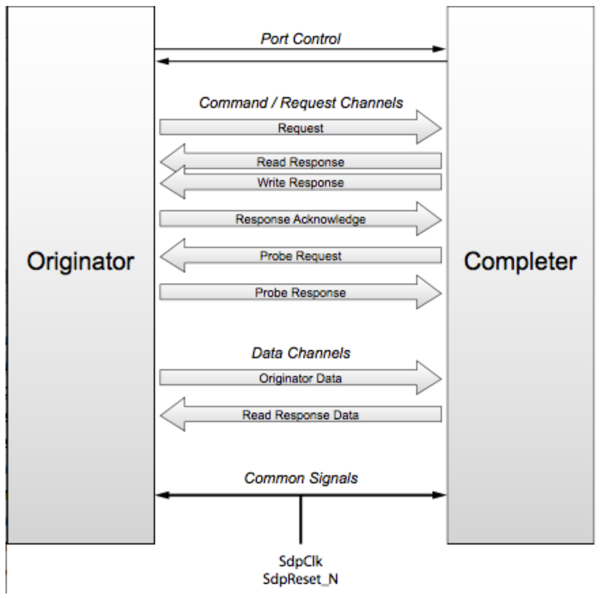

Figure 2 Scalable Data Port Interface

In Figure 2 Scalable Data Port Interface, the block labeled "Originator" represents the agent that issues commands (requests) and "Completer" represents the agent that satisfies these requests. In a typical application, the originator is not connected directly to the completer, but rather, indirectly via a fabric. At the originator port, the fabric serves as a proxy for the completer and, at the completer port, the fabric serves as a proxy for the originator.

As shown in the figure, the physical channels can be organized into three groups: port control signals, command / response channels, and data channels. A set of signals that share a name and encode specific information is called a ***field***. The ***size*** attribute of a field indicates the number of signals that make up the field. Signals within a field are numbered based on binary significance. An n-bit field is made up of the signals n-1 through 0.

The diagram shows a separate clock for each block. These clock signals are provided by SOC-pervasive logic. If these clocks are not matched in frequency with a static phase relationship, resynchronizing logic must be supplied. Synchronizing logic is accommodated, but not specified.

Note that the read response and read response data channels share the same link-level handshake signals and are described together in Read Response Channel.

Port Control

Each port includes two pairs of port control signals. These signals allow an attached device to request that the port be brought up to an active state or placed in a quiescent state to allow the device to transition to a clock-stop or power-gated state. Port control state transitions can be initiated by the port as well. These signals are described in Port Control Signals.

Command / Response Channels

The command and response channels include fields that are used to request or acknowledge the transfer of data and/or control information. These channels include:

- Originator Request Channel
- Read Response Channel
- Write Response Channel
- Response Acknowledge Channel
- Probe Request Channel
- Probe Response Channel

Data Channels

The data channels are used to transfer data from/to an originator to/from a completer in the system. These channels include:

- Originator (Write/Probe) Data Channel
- Read Response Channel

Other Signals

Each port interface includes a clock and a reset signal supplied by SOC-pervasive logic. These are described in the topic Common Signals.

All signals on the port interface are synchronous to a common clock. Signals are launched from a register on one side of the interface and are expected to reach the other side with enough time to be used in a small amount of logic (perhaps causing the de-assertion of **Vld/Rdy**, or advancing a transmit queue to the next item) before the clock edge at the receiver used to latch the information at the receiving register. In actual implementations the register sourcing a set of signals on one side of the interface may be in a different clock domain that the one receiving the signals. A mechanism not specified in this specification tracks any phase shift between the separate clocks and compensates for it.

Port Signaling

Refer to Figure 3 Port Signaling. Note that an originator device attaches to the fabric through an interface called an ***originator port*** and a completer device attaches to the fabric through an interface called a ***completer port***. The completer port is a subset of the originator port because completers do not require the same functionality as originators.

At the originator port the originator device (or simply, originator) drives request signals and the originator port receives the request signals. The fabric either satisfies the request directly or routes the request to the appropriate completer port where it is presented to a completer device. At the completer port, the port sources request signals and the completer device sinks the signals. The completer device processes the request and generates a response. The device sources the response signals and the port sinks them. Responses routed back to the originator by the fabric appear at the originator port at the response physical channel. At the originator port, the port sources the response signals and the originator sinks the signals.

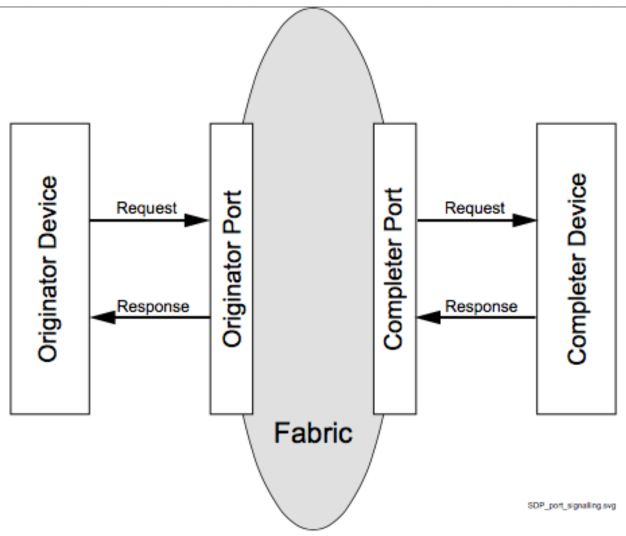

Figure 3 Port Signaling

In the following subsections, tables list the fields that belong to each channel of the SDP interface. In the tables, the driver of each field ***at the originator port*** is identified. Thus, the specified driver will either be the originator or the port. At the completer port, if the field is implemented, fields where the information flow is to the right will be driven by the port and fields where the information flow is to the left will be driven by the completer device.

Optional fields are identified based on the feature that they support. The default value for an unimplemented optional field is 0, unless specified otherwise.

When originator and completer are attached point-to-point, the fabric disappears and the originator looks like a completer port to the completer device. The completer, in turn, looks like an originator port to the completer.

2.2.2 Methods to Increase Bandwidth Within a Single Port

There are various methods to take a single port and increase bandwidth. This section describes these methods and differentiates their intended use cases. Note that these methods are interoperable, and an SDP implementation may have more than one.

Implementation of Multiple Channel Instances

An implementation may provide multiple instantiations (instances) of a single channel type. For example, an implementation may have two originator request channels and three read response channels. When referring to these channels and all fields in the channel, they are labeled as "A", "B", "C", … respectively, for example there may be a "Request Channel A" with fields ReqAVld, ReqAVc (and so on) and then there may be a "Request Channel B" with fields ReqBVld, ReqBVc (and so on), and so on up to the number of implementations (A-Z). When there is only one channel of a given type, the "A" is extraneous and may be omitted. In this specification, the field names are always used as if there is only one channel and the "A" is omitted for simplicity. There is no requirement that a port has the same number of instances for each channel type.

Within this specification, the channels are capitalized ("Originator Request Channel") when the specification refers to the plurality of all instantiated channels as a whole, such as referring to the credit or tag and it is presumed that the subject discussed could have occurred on any of the instantiated channels. The channels are not capitalized ("originator request channel" or "original request channel A") when they are referring to a single instance, such as referring to a request appearing on one of the instantiations of the originator request channel.

All instantiations of a given channel share credits, tags, and unitID encodings. There is no required association between the instantiations of different channel types. For example, a request on "originator request channel A" may use a credit that was originally transferred (from the completer) on "originator request channel B". Furthermore, the response for this request may then be transferred by the completer on "read response channel C".

The multiple instances of a given channel are considered "time-multiplexed" using their alphabetic numeral for all ordering purposes. That is, a request that appears on a given cycle on "originator request channel B" is onsidered to have been transmitted later than any request (if any was valid) on "originator request channel A" on the same cycle and is considered to have been transmitted earlier than any request (if any are valid) on "originator request channels C" through the last originator request channel instance. If there is no required ordering between the two requests, for example they are in different VCs, then this time-multiplexed relationship is inconsequential.

All fields of the channel type are replicated in every instance, including the *ClkEn_m1, *Vld, and *Rdy signals. In order to ensure that the time-multiplexed ordering relationship is maintained, any port delays or clock crossing implementations must be identical between all the instances. If original request channel A's ReqRdy signal was not active on a cycle when original request channel B's ReqRdy signal was active, then the time-multiplexed ordering requirement may be violated. Thus, the *Rdy signals, while independently instantiated and possibly generated independently must end up as identical copies of each other, regardless of whether there is a required ordering relationship in the first place.

Each channel instance has independent instances of *ClkEn_m1, allowing the port to clock gate one or more of the instances independently. There is no requirement that higher named ports are clock gated before lower named ports or vice versa. For example, "original request channel A" may be clock gated while "original request channel B" may transfer a request. When an instance deasserts *ClkEn_m1 on cycle x, then it is presumed that the other side of the port may treat the channel instance as having *Vld=0 on the cycle x+1, and this happens independently on each instance.

There is only one set of port control signals for a single port, regardless of the number of implemented channels and the number of instances of each channel type.

The intended purpose of multiple channel instances is to support a higher bandwidth at a lower clock rate. A port with two instances can be considered that it operates as if the SDPClk was twice the frequency with a single instance transferring the first channel instance in the even clock cycles and the second channel instance in the odd clock cycles.

Physical Channel Multiplexing Feature

The physical channel multiplexing feature allows two separate completer entities to share a single SDP for all data transfers, although the feature could (in a future version of the specification) to allow separate originator entities and/or extend the number beyond two. The originator is required to be aware of both completer entities and tracks separate credits for each one. For example, a credit returned on ReqCreditChanAB=0 (subchannel A) may only be used with a request that asserts ReqChanAB=0. The originator must have a method to select between the two subchannels such as by hashing the request address. The actual method is outside the scope of this specification.

The intended purpose of the physical channel multiplexing feature is to share wires (the SDP port) when two completer entities are always communicating with a single originator, such as two subchannels of what is logically a single memory channel.

2.2.3 Common Signals

Table 2 Common Signal Symbols lists the signals that make up the Common Signal group. SOC-pervasive logic supplies these to all ports.

Table 2 Common Signal Symbols

Name	Size	Driver	Description
SdpClk	1	Pervasive logic	Interface clock, sourced from the SOC pervasive logic. SDP Port signals are synchronous to this clock except for the following: • SdpReset_N assertion. • *ClkCtl, part of the enhanced port control feature, which is synchronous to *ClkReq. • *ClkReq and *ClkAck, when the asynchronous port control feature is implemented. When the asynchronous port control feature is not implemented, these four signals are synchronous to SdpClk.

SdpReset_N	1	Pervasive logic	Interface reset signal, active LOW. Reset may be asserted asynchronously to SdpClk, but the deassertion is synchronous to SdpClk.

The rising edge of SdpClk determines the timing of information transfer across the interface. In general, all signals must be stable prior to and immediately following the rising edge of SdpClk. In practice registers on each side of the port interface may be in different clock domains. *Rdy is used to ensure that there is sufficient time between the active edge of the clock of the transmitter relative to the active edge of the receiver.

SdpReset_N is a negative-active signal (that is, the signal is asserted when the voltage level of the signal is low.) and may be asserted at any time. See Signaling Requirements for more information concerning signaling requirements associated with SdpReset_N.

2.2.4 Port Control Signals

Each port provides two pairs of request / acknowledge signals that allow either the originator or the completer to request that the port be brought up to its normal active state or, once the port is in its normal active state, to request that the port be brought down and logically disconnected. In the disconnected state, there is no exchange of information across the interface. All *Vld signals are inactive and all physical channel fields are ignored. This allows either or both agents to be placed a low power clock-stopped or a power-gated state.

The enhanced port control feature adds a third pair of signals used to provide more information about the connect or disconnect request.

Table 3 Port Control Signals lists the Port Control signals.

The column labeled **Required** indicates which optional feature requires this field to be implemented. “Yes” means that the feature is required for all implementations. The column labeled **Driver** indicates the direction of the signal at an originator port.

Table 3 Port Control Signals

Name	**Size**	**Driver at Originator Port**	**Description**	**Required**
OrigClkReq	1	Originator	Originator clock request. Asserted by the originator to request that the port be brought up to an active state. When asynchronous port control feature is implemented, this signal is asynchronous to SdpClk.	Yes
CompClkAck	1	Port	Completer clock acknowledge. Asserted by the completer in response to the OrigClkReq signal. When asynchronous port control feature is implemented, this signal is asynchronous to SdpClk.	Yes
CompClkReq	1	Port	Completer clock request. Asserted by the completer to request that the port be brought up to an active state. When asynchronous port control feature is implemented, this signal is asynchronous to SdpClk.	Yes
OrigClkAck	1	Originator	Originator clock acknowledge. Asserted by the originator in response to the CompClkReq signal. When asynchronous port control feature is implemented, this signal is asynchronous to SdpClk.	Yes
OrigClkCtl	1	Originator	Originator clock control. Set to an active or inactive level by the originator prior to assertion or de-assertion of OrigClkReq to indicate additional information about the connect or disconnect request. This signal is asynchronous to SdpClk and sampled only on an OrigClkReq edge.	Enhanced port control feature
CompClkCtl	1	Port	Completer clock control. Set to an active or inactive level by the completer prior to assertion or de-assertion of CompClkReq to indicate additional information about the connect or disconnect request. This signal is asynchronous to SdpClk and sampled only on a CompClkReq edge.	Enhanced port control feature

The SDP interface is composed of multiple unidirectional physical channels. A coherent originator, for instance, has the role of transmitter on the Originator Request, Originator (Write/Probe) Data, Response Acknowledge, and Probe Response Channels. It has the role of receiver on the Read Response (which includes read data), Write Response, and Probe Request Channels. The port as a proxy for the completer has the complementary role on each of the channels.

While the interface is composed of a plurality of physical channels, there is a single signal provided for the originator (or port, at the completer port) to request the activation of the interface (OrigClkReq) and a single signal for the port (or completer) to make this request (CompClkReq). The signals CompClkAck (driven by the completer side of the interface) and OrigClkAck (driven by the originator side) are provided to complete the handshake protocol required to bring the port up to its active mode of operation. At the completion of the initialization protocol, all four signals are in their active state. This indicates that the interface is in the normal active state.

OrigClkReq

The originator (or port, at a completer port) asserts OrigClkReq to request that the port interface be brought up to an active state. This signal must remain active while the port is in its normal active mode of operation. When acknowledged by the assertion of CompClkAck, the port (or completer, at a completer port) is ready to receive information on all its inbound channels. The originator (or port, at a completer port) de-asserts this signal to request that the port be disconnected. The disconnect request is acknowledged by the port (or completer, at a completer port) by de-asserting the CompClkAck signal.

CompClkAck

Asserted by the port (or completer, at a completer port) in response to OrigClkReq indicating that the port (completer) is ready to actively receive inbound command, data, or flow-control credits on the corresponding physical channels.

CompClkReq

The port (or completer, at a completer port) asserts CompClkReq to request that the port interface be brought up to an active state. This signal must remain active while the port is in its normal active mode of operation. When acknowledged by the assertion of OrigClkAck, the port (or originator, at an originator port) is ready to receive information on all its inbound channels. The port (or completer, at a completer port) de-asserts this signal to request that the port be disconnected. The disconnect request is acknowledged by the originator (or port, at a completer port) by de-asserting the OrigClkAck signal.

OrigClkAck

Asserted by the originator (or port, at a completer port) in response to CompClkReq indicating that the completer is ready to actively receive inbound command, data, or flow-control credits on the corresponding physical channels.

OrigClkCtl

This signal may be sampled by the port (or completer, at a completer port) when a transition is detected on the OrigClkReq signal. The originator (or port, at a completer port) asserts this signal prior to asserting OrigClkReq to request that the port (completer) issue (or re-issue) flow control credits for the originator's (port's) outbound channels that utilize flow control. The originator de-asserts this signal prior to asserting OrigClkReq to explicitly request that the completer not issue these credits. The state of this signal must be held stable until the connection request is acknowledged by the port (or completer, at a completer port).

The originator (or port, at a completer port) asserts this signal prior to de-asserting OrigClkReq if it is requesting a full disconnect. The originator de-asserts this signal prior to de-asserting OrigClkReq if it is requesting a temporary (one-sided) disconnect. The state of this signal must be held stable until the disconnect request is acknowledged by the port (or completer, at a completer port).

Some SDP port implementations may derive flow control credit handling through alternate means such as firmware or reset sequencing, and may only use OrigClkReq for indicating a full disconnect.

CompClkCtl

This signal may be sampled by the originator (or port, at a completer port) when a transition is detected on the CompClkReq signal. The port (or completer, at a completer port) asserts this signal prior to asserting CompClkReq to request that the originator (or port, at a completer port) issue (or re-issue) flow control credits for the port's (completer's) outbound channels that utilize flow control. The port de-asserts this signal prior to asserting CompClkReq to explicitly request that the originator (port) not issue these credits.

The port (or completer, at a completer port) asserts this signal prior to de-asserting CompClkReq if it is requesting a full disconnect. The prot de-asserts this signal prior to de-asserting CompClkReq if it is requesting a temporary (one-sided) disconnect. The state of this signal must be held stable until the disconnect request is acknowledged by the originator (or port, at a completer port).

Some SDP port implementations may derive flow control credit handling through alternate means such as firmware or reset sequencing, and may only use CompClkReq for indicating a full disconnect.

The details of the protocol for connecting and disconnecting a port are discussed in Port Control.

Synchronization Requirements for Port Control Signals

When a port implements the asynchronous port control feature, the assertion and deassertion of OrigClkReq, CompClkReq, OrigClkAck, and CompClkAck may be asynchronous to SdpClk. It is the responsibility of the receiving logic to sample the signal within the IP clock domain using one or more latches or other means to prevent metastability. When only one side of a port implements the asynchronous port control feature, pervasive logic is required between the ports to synchronize the signals in the asynchronous to synchronous direction.

2.2.5 Relationships Between Channels

As a summary of other areas of this specification, the specification requires the following relationships between channel types:

- With the exception of Cancel, QosControl, and ErrEvent request packets, and a SYSFATALERR response; an agent **must not** issue a request, probe, response, or data packet without having a sufficient credit signaled by the other agent. The excepted case are commands or responses which are specified to not consume a credit.
- The data for a request **must not** be issued on an originator data channel before it has issued the corresponding request on a Request Channel. The data may only be issued on the cycle concurrent with the request or a later cycle.

- The data for a probe response **must not** be issued on an originator probe data channel before the originator has issued the corresponding probe response on a probe response channel. The data may only appear on the cycle concurrent with the probe response or on a later cycle.
- With the exception of an "unsolicited" response for a SYSFATALERR, a completer **must not** provide a response prior to receiving the request on a Request Channel (as matched by the ReqTag and ReqUnitID). In addition, when the request has data, the completer **must not** issue a response for a request that has data prior to receiving all beats of data on an Originator Data Channel. The excepted case is an "unsolicitated" response which is not associated with a specific request.
- An agent **must not** transfer any information, including credits or request/response/probe/data packets on any channel until it has completed the port control handshake.

The specification also requires the following deadlock-avoidance requirements between channels:

- An originator **must** be able to process a probe request, return the appropriate credit, and provide the response and data (if any) without any requirements that completer provides read or write responses to outstanding requests.
- An originator **must** be able to process a probe request, return the appropriate credit, and provide the response and data (if any) without any requirements that the completer provides request channel credits or assert ReqRdy.
- An originator that implements the Response Acknowledge Feature **must** be able to issue the response acknowledgement, once the completer has issued the completion, without having any dependencies other than having a credit on the Response Acknowledge Channel.
- Once an originator has issued a request, it **must** be able to process (or "sink") the response including the return of the read or write response credit without any requirements that the completer processes any other command including, but not limited to, other victims or writes. This requirement can also be stated as a requirement that, upon issuing a command where data may be provided on a read response channel, the originator must have a buffer to consume the read data without any dependency on the completer that it processes other requests including victims or writes. This buffer does not have to be advertised as a read response credit at the time of the request as long as the originator has a dependency-free method to sink the read response.

2.2.6 Originator Request Channel

An originator uses the originator request channel(s) to request that data be written to or read from the system memory space. Table 4 Originator Request Channel Signals the signals that make up an originator request channel.

The column labeled **Required** indicates what optional feature requires this field to be implemented. "Yes" means that the feature is required for all implementations.

The column labeled **Driver** indicates the direction of information flow within the field at an originator port.

Table 4 Originator Request Channel Signals

Name	Size	Driver	Description	Required
ReqTag	N_{rt}	Originator	Request tag. This field is used to uniquely identify each outstanding request from a unit of the originator attached via this port. See ReqTag below for more information.	Yes
ReqAddr	N_{ra}	Originator	Request address. For caching reads, this field indicates the critical byte of the requested aligned cache block. For other commands, this field gives the byte address of the data to be transferred in the beat labeled with DataOffset 0 (the lowest addressed byte of data). See ReqAddr below for more information.	Yes
ReqSubAddr	Nrsa	Originator	Request SubAddress. For non-caching accesses, this field indicates an XOR to be performed on a subset of address bits to create two logically independent transactions that are combined into a single transaction on SDP. The use of this field may be restricted. See ReqSubAddr below for more information. As an XOR, when this field is zero; ReqAddr contains the address for the entire request transaction.	Multiple Request Address feature
ReqSpecialAddr	1	Originator	System Management Request. This field indicates address space information. See ReqSpecialAddr and ReqIO below for more information. If not implemented, the completer interprets the address according to ReqIO and the system address maps.	System Management Access feature
ReqIO	1	Originator	Request I/O. This field indicates address space information. See SysMgmt and ReqIO below for more information. If not implemented, the completer interprets the address according to the system address maps.	I/O space feature

ReqLen	N_{rl}	Originator	Request length. This field indicates amount of data to be transferred by the transaction request. Requested transfer size is (ReqLen + 1) doublewords. See ReqLen below for more information.	Yes
ReqAck	1	Originator	When set to 0, indicates that a transaction response acknowledgment will not be sent on a Response Acknowledge channel when the response to this request is received. Must be set to 1 for any command that requires a response acknowledgment. If not implemented, all requests must be treated as if this field were set to 0.	Response Acknowledge feature
ReqAttr	8	Originator	Request Attributes. Extended transaction attributes to be used at the target. The encoding of this field depends on the request. See ReqAttr below for specific encodings of this field.	ReqAttr is not required for originators that generate requests without extended attributes.
ReqQosPriority	N_{pri}	Originator	Quality of service priority. 0 is the lowest and (2^N_{pri} – 1) is the highest.	QoS feature
ReqQosForward	2	Originator	Quality of Service Forwarding. Indicates which previous requests (if any) should be treated with ReqQosPriority. See ReqQosForward for specific encodings of this field. Implementations of the QoS feature are allowed to not instantiate this field and the port treats it as if the field is always 00b.	QoS feature
ReqCmd	6	Originator	Request command. This field indicates the transaction type for this request. See Commands for the encoding of this field.	Yes
ReqDomain	1	Originator	Request domain. This signal indicates the shareability domain of a request transaction.	Required in systems that implement shareability domains.
ReqVirtAddr	1	Originator	Request Virtual Address. This signal indicates that ReqAddr is a virtual address that must be translated.	Virtual Address Feature
ReqVfidValid	1	Originator	Request Virtual Function ID Valid. This signal indicates that the information in the ReqVfid field is valid. When this bit is zero, the address is for the "base" function. Note: A virtual function ID may be valid for either physical address (ReqVirtAddr=0) or virtual addresses (ReqVirtAddr=1).	Virtual Address Feature
ReqVfid	N_{vf}	Originator	Request Virtual Function ID. This signal indicates virtual function ID for this request. This field is *reserved* if ReqVfidValid = 0.	Virtual Address Feature
ReqUnitID	N_{ui}	Originator	Requester unit ID. Identifies the unit within the originator that initiated the transaction. Default value of this field is 0.	Yes, except for CS that flattens tag space.
ReqStreamID	N_{si}	Originator	Request stream ID. Identifies a particular ordered stream within a ReqUnitID. The default value of this field is 0.	I/O Stream feature
ReqSecLevel	N_{rsl}	Originator	Request security level. Indicates the security privilege level of the request.	Secure transaction feature
ReqTMZ	1	Originator	Request TMZ. Identifies the request as accessing trusted memory zone.	Trusted Memory feature
ReqPassPW	1	Originator	Request pass posted write. Applies to requests in the posted and non-posted virtual channels. • 1: This request may pass posted writes traveling the same direction. • 0: This request must not pass posted writes. If not implemented the request is handled as if this field were set to 0 when the request is in the posted or non-posted channel and 1 otherwise.	Posted write ordering feature
ReqBlockLevel	2	Originator	Request blocking level. Indicates what level of source matching is required for this request to block behind earlier requests within the same virtual channel. See ReqBlockLevel below for more information.	Request Ordering and I/O Stream features

ReqRspPassPW	1	Originator	Request response pass posted write. Applies exclusively to requests in the non-posted virtual channel. • 1: The response to this request may pass posted writes traveling in the same direction. • 0: The response to this request must not pass posted writes. If not implemented, the request is handled as if this field were set to 1.	Posted write ordering feature
ReqVC	N_{rvc}	Originator	Request virtual channel. Specifies the virtual channel of the request. If not implemented, the completer or completer-proxy determines the virtual channel of the request.	Virtual Channel Support
ReqChain	1	Originator	Request chain. This is a hint to indicate that the originator expects that it will be advantageous to keep this request grouped with subsequent requests in the same VC. It is intended to be used by arbitration logic within the fabric and at destinations to keep requests grouped together for efficiency.	Request Chain feature
ReqParity	1	Originator	Request parity. Maintains even parity across all originator-driven (at an originator port) or port-driven (at a completer port) originator request channel signals excluding ReqAddr, ReqSubAddr, ReqSpecialAddr, ReqIO, ReqVld, ReqClkEn_m1, and ReqCreditRdy.	Enhanced RAS V1 feature
ReqAddrParity	1	Originator	Request address parity. Maintains even parity across ReqAddr, ReqSubAddr, ReqSpecialAddr, and ReqIO. **Implementation Note**: The split of ReqAddrParity and ReqParity is done to reduce the size of the parity tree and should not imply that some fields are more critical than others for parity checking. All ports must implement both ReqParity and ReqAddrParity if the Enhanced RAS V1 feature is supported.	Enhanced RAS V1 feature
ReqChanAB	1	Originator	Request channel A/B select. Only defined for completer ports. • 0: Request uses channel A. • 1: Request uses channel B.	Physical Channel Multiplexing feature
ReqCacheHint	1 or 2	Originator	This field provide hints for allocation in any cache that is inside the fabric or completer on reads (caching and non-caching), writes, victims, and atomic operations. The implementation of said fabric or completer cache and the conditions upon which the originator should leverage this field are outside the scope of this specification. This field is reserved for cache maintenance, barrier, and other commands. This field is ignored when SpecPrefCmd is used as the ReqAttr specifies equivalent functions. In prior specifications, this field was only 1 bit and was named ReqNoAlloc. An implementation may still provide a single bit for this field in a backwards compatible manner and operates as if ReqCacheHint[1]=0b.	External Last Layer Cache feature
ReqStWayId	N_{st}	Originator	This field, when asserted during RdBlkS and coherent WrSized* commands, indicates the originator's cache way identifier that will be updated and/or invalidated as a result of this transaction. The field is reserved when ReqStUpdate and ReqStInvalidate are both zeros.	Cache Tag Tracking feature
ReqStUpdate	1	Originator	This field specifies the cache tag tracking update that must be performed. See ReqStWayId, ReqStUpdate, ReqStInvalidate for more information. This field is reserved, and must be zero, for all commands other than caching commands (RdBlk*) and coherent WrSized*.	Cache Tag Tracking feature
ReqStInvalidate	1	Originator	This field specifies the cache tag tracking update that must be performed. See ReqStWayId, ReqStUpdate, ReqStInvalidate for more information. This field is reserved, and must be zero, for all commands other than caching commands (RdBlk*) and coherent WrSized*.	Cache Tag Tracking feature
ReqCompMode	2	Originator	This field specifies the request compression mode. The encoding of this field is command specific. See ReqCompMode.	Metadata Compression feature
ReqSpecDataFetch	2	Originator	See ReqSpecDataFetch.	Metadata Compression feature
ReqVld	1	Originator	Request valid. This signal is used to pace the flow of requests from the originator for clock synchronization purposes.	Yes

ReqClkEn_m1	1	Originator	Request clock enable. ReqClkEn_m1 must be asserted one clock cycle before ReqVld is asserted.	Low-Power Signaling feature
ReqRdy	1	Port or pervasive logic	Request ready. This signal is used to pace the flow of requests across the interface for clock synchronization purposes. See Signaling Requirements for more information on the *Rdy / *Vld handshake.	Yes
ReqCreditChanAB	1	Port	Request credit channel A/B. Only defined for completer ports. • 0: Credit is for channel A. • 1: Credit is for channel B.	Physical Channel Multiplexing feature
ReqCreditType	1	Port	Request credit type. • 0: the credit being released is for a particular virtual channel, given by ReqCreditVC. • 1: the credit being released is a pool credit, which may be used by requests in any virtual channel. If not implemented, this field will be treated as if it were set to 0.	Virtual Channel Support
ReqCreditVC	N_{rvc}	Port	Request credit virtual channel. If ReqCreditType = 0, this field specifies the virtual channel of the credit being released. If ReqCreditType = 1, this field is ignored. If virtual channels are not implemented by this originator, this field will be treated as if it were set to 0.	Virtual Channel Support
ReqCreditParity	1	Port	Request credit parity. Maintains even parity across the ReqCredit* fields excluding ReqCreditVld, ReqCreditClkEn_m1, and ReqCreditRdy.	Enhanced RAS V3 feature
ReqCreditCnt	$N_{cc[0]}$	Port	Zero origin field of the number of credits being returned.	Credit Count Feature
ReqCreditVld	1	Port	Request credit valid. Information presented on credit release sub-channel by the port (or completer) is valid. See ReqCreditVld or more information.	Yes
ReqCreditClkEn_m1	1	Port	Request credit clock enable. ReqCreditClkEn_m1 must be asserted one clock cycle before ReqCreditVld is asserted.	Low-Power Signaling feature
ReqCreditRdy	1	Originator or pervasive logic	Request credit ready. This signal is used to pace the flow of request credit information across the interface for clock synchronization purposes. See Signaling Requirements for more information on the *Rdy / *Vld handshake.	Yes
ReqUser	—	Originator	User-defined field. If implemented, information must flow in the same direction as the request channel.	User-defined

An originator device uses an originator request channel to initiate the transfer of data to or from the system memory space. The amount of data requested along with the transfer type is indicated with the address. The smallest quantum of data that can be requested is one doubleword. Transfers of data up to the size of the data field of the physical channel (RdRspData or OrigData) carrying the information can be transferred in one clock cycle (beat). Larger amounts of data are transferred in a series of beats (a burst). See Data Transfer for a discussion of how data is transferred across the interface.

The fields of the originator request channel are further described in the following sections.

ReqTag

This field identifies each outstanding request initiated by a specific functional unit (identified by its UnitID) of an originator device. Each unit of an originator is allocated its own independent set of tags.

Each functional unit may not reuse a ReqTag until the last beat of the response (without RETRANSMIT or EARLY) has been received. When a delay exists between the response header and the response data, the functional unit may also reuse a ReqTag if the information in the response header allows it to predict that the response data status will not be RETRANSMIT without necessarily waiting for the data. With the exception of the Cancel command, when a request does not have a response (QosControl, SpecDramRd and ErrEvent), the ReqTag field is meaningless. While transactions that have an explicit acknowledgment by the originator are not complete until the originator sends a response acknowledgment with the same unit ID (AckUnitID) and tag (AckTag) as the request via the response acknowledgment channel, the originator may reuse the tag prior to sending the response acknowledgement if and only if it commits to sending this acknowledgment without any dependencies other than Response Acknowledge credits and AckRdy; and only if it commits to sending the response acknowledgements for duplicate tags in the order that it received the responses. See Response Acknowledge Channel for more information on the response acknowledgment channel signals. Table 40 Commands indicates which commands require acknowledgment by the originator.

ReqAddr

For non-caching reads, atomics, and writes (including WrNoDataNC), this field provides the byte address of the data being accessed, or, in the case of system-management commands, further details on the management function.

For caching reads, cache upgrades, cache eviction, and cache maintenance commands, ReqAddr provides the cache line address of the data being accessed. The ReqAddr bits below the cache line address are not used and are normally reserved. However, for caching reads and ChgToX, ReqAddr specifies a critical byte within the requested aligned cache block. The beat containing the critical byte should be transferred before the other beats, if possible.

Interfaces that always perform operations that are n-byte aligned may omit bits [$\log_2$n-1:0]. ReqAddr[1:0] are always optional.

Addressing is little-endian.

ReqSubAddr

For the purposes of this section, the address bit that addresses one half of a cache line size will be referred to as address bit "x" (i.e. x is bit 5 in implementations with a 64-byte cache line). In addition, this section will refer to a "ReqAddr2" that can be determined from an arithmetic operation between ReqAddr and ReqSubAddr even though ReqAddr2 is not a field on the SDP port.

When the multiple request address feature is implemented, and ReqSubAddr is non-zero, the transaction consists of two separate logical transactions with two separate addresses in one single combined SDP request. Other than the address, all other fields (e.g. ReqCmd, ReqAttr, ReqSecLevel, ReqVC, etc.) for these two logical transactions must be the same. These two separate addresses are then accessed by the completer and the response and data (for reads) is returned for both addresses.

The intent of the multiple request address feature is to optimize systems where the completer is implemented with two subchannels that operate together to perform cache line size accesses. In these implementations, the completer bandwidth is inefficiently used whenever the originator makes a sub-cache line size request. To recover this bandwidth, the originator may specify two independent half-cache line size requests - one for the first subchannel using ReqAddr and a second half-cache line size request for the second subchannel using ReqSubAddr. The originator must ensure that ReqAddr maps to the first subchannel and that ReqAddr2 (the second address) maps to the second subchannel. Unpredictable results may occur if this is not the case. Normally, this requires the originator to have a map of which address bit(s) select the subchannel.

ReqAddr2 is created by the completer with the following XOR:

- ReqAddr2[x:0] is implied based on the subchannel selection logic.
- ReqAddr2[n] = ReqAddr ^ ReqSubAddr for address bits that are specified in ReqSubAddr.
- ReqAddr2 = ReqAddr for address bits that are not specified in ReqSubAddr (except for spec x).

As only certain address bits may be different between the two logical transactions, the field ReqSubAddr is not the same width as ReqAddr. Furthermore, the specified address bits may not be contiguous. For example, ReqSubAddr[0] may map to address bit 7 and ReqSubAddr[1] may map to address bit 9. During the XOR operation to create ReqAddr2, the completer assumes that all unimplemented bits are zero with the exception of address bit "x", which is not specified but is set appropriately to select the other subchannel. Which address bits are specified in ReqSubAddr and the mapping of ReqSubAddr[a] to ReqAddr[b] is implementation specific. It is generally expected that the originator and the completer would have configuration bits that specify this mapping of ReqSubAddr to ReqAddr bits.

Note that it is not necessary that ReqAddr[x] is zero in the transaction as ReqAddr[x] may be one of the bits that selects between the two subchannels. This creates a case where the SDP transaction appears to cross a cache line if one was to not consider the multiple request address feature. The non-zero value in ReqSubAddr is indicating that there are two separate cache lines being accessed.

ReqSubAddr is reserved, and **must** be zero, in the following cases:

- Caching reads (RdBlk*), cache eviction commands (VicBlk*), SpecDramRd, and cache upgrade commands.
- For non-caching requests when ReqLen is not equal to the cache line length.
- When ReqIO=1 or ReqSpecialAddr=1.
- Depending on the implementation, for coherent reads or writes. At the time of this specification, no SOC-15 implementation allows coherent requests to use ReqSubAddr.
- Depending on the implementation, for atomic commands. At the time of this specification, no SOC-15 implementation allows atomic requests to use ReqSubAddr.
- WrNoDataNC.

When ReqAddr field is reserved or ignored (e.g. Fence, Flush, ErrEvent, QosControl), then ReqSubAddr is also reserved and must be zero.

While the two logical requests are each one-half of a cache line in length, the combined request and the ReqLen on the SDP port is for a cache line (i.e. ReqLen is 0xF for implementations with a 64-byte cache line).

The completer always returns the response in one single read completion or write completion. The data is addressed based on the subchannel – the least significant data bytes (from 0 through cache line size divided by two minus 1) are for the first transaction and first subchannel (ReqAddr). The most significant bytes (from cache line size divided by two through cache line size minus 1) are for the second transaction and second subchannel (ReqAddr2) regardless of ReqAddr[x]. Since there may be multiple address bits selecting the subchannel, the byte lanes used for the transaction data might not be the byte lane that would be used if the originator had made two separate transactions.

Any error on either of the two addresses results in an error on the entire transaction. The completer follows all other rules, including ordering, as if these requests were two separate independent requests.

ReqSpecialAddr and ReqIO

ReqSpecialAddr and ReqIO are used to determine the encoding of an address. When ReqAddr is a physical address (ReqVirtAddr=0 or the Virtual Address Feature is not present), the address may be encoding “system or device/local memory”, “IO memory”, or the address may be encoding a system management function. The fields ReqSpecialAddr and ReqIO determine the address type as shown in Table 5 ReqIO and ReqSpecialAddr Encoding. ReqSpecialAddr is required to be used when the port requires access to the system messages outside of the 12GB region at the end of physical address space. Otherwise, the “system memory” address is limited to $2^{ReqAddr}$ size minus 12GB.

On adapters, “IO memory” also includes “host memory” (all memory that is not on the adapter). The exact meaning of “system memory” and “IO memory” may be system-dependent.

For CPU originators, the I/O space feature (ReqIO) allows for an x86 architectural feature where a physical address may be mapped to either I/O or system memory space dynamically. This dynamic map is not expected to be implemented on non-x86 implementations.

Table 5 ReqIO and ReqSpecialAddr Encoding

ReqSpecialAddr	**ReqIO**	**ReqVirtAddr**	**Port Type**	**Access**
0	**0**	**0 (not implemented)**	**CPU**	Accessing the “system” memory using a physical (post-translation) address.
0	**1**			Accessing I/O using a physical (post-translation) address.
1	**0 or 1 (don’t care)**			Accessing the system message space (see System Messages).
0	**0**	**0**	**Non-CPU**	Accessing “local” memory using a physical (post-translation) address.
0	**1**	**0**		Accessing “host” memory or I/O memory using a physical (post-translation) address.
0	**1**	**0**		Accessing “local”, “host”, or I/O memory using a pre-translation address.
0	**1**	**1**		Accesses using a virtual address (ReqVirtAddr set).
1	**0**	**0**		Accessing the system message space (see System Messages).

Notes:
1) When ReqSpecialAddr or ReqVirtAddr is not instantiated on a port, the field is treated as if it is zeros.
2) When ReqSpecialAddr is not instantiated, all system messages are identified by decoding the address. Only the system addresses in the last 12GB of the system management may be used if ReqSpecialAddr is not instantiated.

ReqLen

This field specifies the number of doublewords to be transferred. The transfer size indicated is (ReqLen + 1). This size can be less than the width of the *Data field of the read response or originator data channel used to transfer the data. The width of this field (N_{rl}) is implementation-specific.

The ReqLen field is *reserved* for all commands with write semantics, but no data payload, such as VicBlkCln, WrNoDataNC, Fence, Flush, and cache maintenance commands such as WbInvBlkAll. For these commands the ReqLen field must be set to 0.

For commands that operate on a cache block, such as caching reads (RdBlk*), Cache Upgrades (ChgToX, ChgToXNR, ValBlk), cache eviction commands other than VicBlkCln (VicBlkFull*, VicBlkClnD, VicBlkPtl), and cache maintenance commands (ClnBlkAll, WbInvBlkAll, and InvBlkAll), the ReqLen is expected to indicate the size of the cache block.

ReqAck

Most requests define the value of this field, but some allow it to be cleared optionally for requests that do not need to be ordered after previous requests. See Commands for a definition of the commands which use the response acknowledge feature.

ReqAttr

Provides specific attribute information about the transaction request. Encoding depends on the type of request as shown in the following tables. ReqAttr is reserved for any command not included in this section. All reserved bits in the ReqAttr ***must*** be zero.

Table 6 WrSized, WrSizedFull, WrSizedFullZero, WrSizedFullComp, and Caching Read Requests ReqAttr Usage

Bit Range	**Sub-field (Steering =0)**		**Sub-field (Steering = 1)**
	Caching Reads	**WrSized***	**WrSized***
7:6	*Reserved*	For WrSizedFullComp: Transfer Size For all other WrSized: Metadata Compression	Steering Tag[6:0]
5	*Reserved*		
4:3	Prefetch Hint[1:0]		
2	*Reserved*	Cache State [1:0]	
1	Storing probe fetch		
0	*Reserved (0b)*	Steering	

Steering Tag

- Steering Tag is an indication or hint of a component, such as a cache in the system, where the data may be directed (writes) or sourced (reads).
- The encoding of Steering Tag is outside the scope of this document.
- It is recommended that the value of 0000000b indicates "no specific steering tag" where the completer or fabric may use an implementation specific algorithm for selecting a component to steer the data. In this use case, the Steering=1 is solely an indication that the completer or fabric may wish to steer the data.

Prefetch Hint

- 00b = No prefetch hint
- 01b, 10b, 11b = *Reserved*

Cache State (for WrSized/WrSizedFull/WrSizedFullZero/WrSizedFullComp only) indicates the cache state for coherent originators:

- 0xb = Unknown or any; originator may be probed.
- 10b = Known invalid or don't care; originator will not be probed.
- 11b = Shared; originator will not be probed. This encoding is intended for write-through caches after the originator has already updated its cache.

Transfer Size

- 00b = Reserved
- 01b = Half-size compression – i.e. 32 compressed bytes actually sent when ReqLen=0xF or 16 compressed bytes sent when ReqLen=0x7. **Note:** When byte enabled compression, the Transfer Size does not change if any beats are used for transferring byte enables.
- 1xb = Reserved

Metadata Compression

- 00b = The write data is uncompressed. ReqLen indicates the size of the uncompressed data.
- 01b = The write data is compressed. The uncompressed length is 128B (ReqLen indicates the size of the compressed data).
- 10b = The write data is compressed. The uncompressed length is 256B (ReqLen indicates the size of the compressed data).
- 11b = Reserved

The Metadata Compression field of ReqAttr is reserved and must be 00b for WrSized* commands if the Metadata Compression Feature is not implemented. See MetaData Compression.

Storing Probe Fetch

- 0b = This caching read is not the result of a storing probe. The completer does not use RdRspStatus=OKAY_NODATA in the response to these caching reads.
- 1b = This caching read was generated after a storing probe caused the CPU to fetch the line. The completer may return RdRspStatus=OKAY_NODATA in some circumstances.

Table 7 ChgToX, ChgToXNR, ValBlk Request ReqAttr Usage

Bit Range	**Sub-field**
ReqAttr[7:3]	*Reserved*
ReqAttr[2:1]	Cache State.
ReqAttr[0]	*Reserved*

Cache State indicates the cache state for coherent originators:

- 0xb = Unknown or any; the originator may be probed.
- 10b = Known invalid or don't care; the originator will not be probed.
- 11b = Shared or Owned; the originator will not be probed (the originator must maintain the cache state properly when using this encoding).

Table 8 RdSized, RdSizedNC ReqAttr Usage

Transaction Type	Bit Range	Sub-field
Device Peer-to-Peer Messages	ReqAttr[7:0]	User-defined. Carried by fabric, unmodified, from source to destination.
All Others	ReqAttr[7:4]	Byte enables, last DW. Ignored if the last DW is also the first DW.
	ReqAttr[3:0]	Byte enables, first DW
Note: Byte enables may be ignored by the completer if there is no side effect of reading more bytes than requested, such as may be the case for DRAM reads.		

Table 9 Atomic Request ReqAttr Usage

Bit Range	Sub-field
ReqAttr[7:4]	Operation Type (**OpType**). See Atomics.
ReqAttr[3:2]	Operand Size (**OpSize**). See Atomics.
ReqAttr[1]	*Reserved*
ReqAttr[0]	Operation Type Extension (OpTypeExt) when the Floating Point Atomic Feature is implemented. Reserved (must be zero) when the Floating Point Atomic Feature is not implemented. See Atomics.

Table 10 WrSizedNC ReqAttr Usage

Bit Range	Significance
ReqAttr[7:0]	Non-SOC15 fabric ports: *User-defined* SOC-15 fabric ports: ReqAttr is *user-defined* for transactions in the "Device Peer-to-Peer Message region". The fabric carries the field, unmodified, from the source to the destination. For all other transactions, including DRAM transactions, ReqAttr[7:0] is *reserved*.

Table 11 Victim Block ReqAttr Usage

Bit Range	Sub-field
ReqAttr[7:6]	TransferSize. *Field is valid for VicBlkFullComp only, reserved for all other* • 00b: *Reserved* • 01b: Half-size compression – 32 compressed bytes actually sent for 64-bytes of data. • 1xb: *Reserved* **Note:** ReqAttr[7:6] is used internally for SOC-15 fabric (for non-compressed commands).
ReqAttr[5]	*Reserved*. **Note**: ReqAttr[5] is used internally for SOC-15 fabric.
ReqAttr[4:3]	CurrentState. *Field is valid for VicBlkClnD only, reserved for VicBlkFull*, VicBlkPtl, and VicBlkCln*: • 00b: *Reserved* • 01b: Current state of the line is shared (S) state. • 10b: Current state of the line is forward (F) state. • 11b: Current state of the line being evicted is exclusive (E) state. When a cache sends out a VicBlkClnD of an exclusive line and retains a shared copy of the line, the current state must account for the downgrade. For example, a clean eviction of an exclusive (E) line where the FinalState is shared (01b) would indicate a CurrentState of Forward (10b), not Exclusive (11b).
ReqAttr[2]	NoAlloc. If NoAlloc = 1, provides a hint that the cache block being victimized should not be allocated in a higher level of the system memory cache hierarchy. Only applicable to VicBlkFull*. *Field is reserved for VicBlkCln, VicBlkClnD, and VicBlkPtl.*

ReqAttr[1:0]	FinalState: • 00b: Final state of the line is invalid (I). • 01b: Cache has retained the line in a shared (S) state. • 10b: Cache has retained the line in an exclusive (E) state. *Reserved for VicBlkCln and VicBlkClnD.* • 11b: Cache has retained the line in the forward (F) state.

Table 12 ErrEvent ReqAttr Usage

Bit Range	Sub-field
ReqAttr[7:0]	ErrorType: • 00h: *Reserved* • 01h: Fatal unrecoverable ("SYNCFLOOD") error at a system level. • 02h–FFh: *Reserved*

ReqQosForward

The encoding of this field is shown in Table 13 ReqQosForward Encoding.

Table 13 ReqQosForward Encoding

	Other Requests Affected	
ReqQosForward[1:0]	**Originator Port**	**Completer Port**
00b	Only with same tag	Escalate by tag
01b	All with the same ReqUnitID	Escalate reads in ReqVC
10b	All from this port	Escalate writes in ReqVC
11b	All in the same VC	*Reserved*

The fabric may choose to escalate more requests than necessary to simplify the tracking of escalations. A port implementation that implements the QoS Feature as a method to indicate the priority of the current request on an originator request channel does not require any forwarding of the priority. In this case, the originator would always have ReqQosForward=00b. An implementation is allowed to not instantiate the ReqQosForward field at all and have it treated as if ReqQosForward == 00b.

Table 14 AddrXlateRdSz and AddrXlateWrSiz ReqAttr Usage

Bit Range	AddrXlateRdSz Usage	AddrXlateWrSz Usage
ReqAttr[7:5]	Permissions field – {EXECUTE, WRITE, READ}. Specified the level of permissions requested for translation.	Reserved
ReqAttr[4:3]	Reserved	Reserved
ReqAttr[2:0]	Subcommand • 000b = “Address Translation” All other subcommand encodings are reserved.	Subcommand • 000b = “Address Invalidation” All other subcommand encodings are reserved.

Table 15 SpecPrefCmd ReqAttr Usage

Bit Range	Usage
ReqAttr[7:6]	PrefetchType: • 00b: No response. This type of command may be dropped anywhere. • 01b: Reserved • 10b: Return response without data • 11b: Return response with data
ReqAttr[5:4]	LlcPrefetchCmd: • 00b: Normal address and allocation policies hold • 01b: Retrieve completer cache (previously allocated data) • 10b: Prefetch and allocate in completer cache • 11b: Reserved Refer to SpecDramRd/SpecPrefCmd for more information.
ReqAttr[3:0]	Reserved

ReqCmd

This field indicates the command for this request. See Commands for a list of commands and their encodings. When ReqCmd[5] is 1b, the originator must also transfer data on an originator data channel. In addition to the Originator Request Channel flow control, one or more credits must be obtained on the Originator (Write/Probe) Data Channel as defined in Channel Flow Control.

ReqDomain

This signal indicates the shareability domain of the request. Each coherent originator port appears as a single caching entity, regardless of how many caches or levels of caches are behind it. The encoding of this field is shown in Table 16 ReqDomain Encoding.

Table 16 ReqDomain Encoding

Value	Attribute	Meaning
0	Inner Shareable	The request is coherent within a defined subset (called the shareabilty domain) of other originators for this address. Only those originators within the domain need to be probed. The determination of that subset is beyond the scope of this specification.
1	Outer Shareable	SDP originator is coherent with all other SDP originators in the system for this address, and all originators need to be probed.

Shareability domains are used to limit the scope and number of probe transactions required to maintain coherency when only a subset of originators (which would all participate in the same domain) are known to be using a region of memory. This could be used, for example, to allow a group of multimedia processors to access a specific region of memory using cache coherent accesses while limiting probes to targeting only the caches associated with that group of processors.

ReqUnitID

Requester unit ID. Identifies the unit within the originator that initiated the transaction. For requests from a CPU, the ReqUnitID identifies the thread making the request. This is for thread-specific register accesses in MSR and APIC space, security checks and debug.

ReqStreamID

Identifies a particular ordered stream within a ReqUnitID.

ReqSecLevel

Indicates the security privilege level of the request.

ReqPassPW

When ReqVC specifies a non-posted or posted virtual channel (in either upstream or downstream direction), ReqPassPW specifies (when clear) that this request must be ordered with prior posted writes (regardless of address or destination). Refer to Ordering for more information.

In the “upstream” direction, ReqPassPW ordering is only established after the upstream transaction reaches a PCIe interface and not within the fabric if the upstream posted and non-posted VCs are separate. The fabric carries the bit as a payload to the PCIe interface.

This bit is *reserved* for all other virtual channels, but it is recommended that the bit is set for these transactions (as they will be able to pass posted writes due to the different virtual channel).

ReqBlockLevel

Indicates what level of matching is required for this request to block behind an earlier request in the same virtual channel.

The encoding of this field is shown in Table 17 ReqBlockLevel Encoding.

Table 17 ReqBlockLevel Encoding

ReqBlockLevel	Blocking Type	For non-Barrier Commands	For Fence and Flush
00b	Address	Request only blocks on a write issued from the same port to the same address.	Invalid
01b	Stream	Request blocks on any write issued from the same port, ReqUnitID, and ReqStreamID	Applies to requests from same port with matching ReqUnitID and ReqStreamID.
10b	Unit	Request blocks on any write issued from the same port and ReqUnitID	Applies to requests from same port with matching ReqUnitID.
11b	Port	Request blocks on any write issued from the same port.	Applies to all requests from same port.

ReqRspPassPW

When ReqVC specifies a non-posted virtual channel (in either upstream or downstream direction), ReqRspPassPW specifies (when clear) that the response to this request must not pass writes in the posted write virtual channel that are flowing in the same direction as the response. Refer to Ordering for more information.

This bit is *reserved* for all other virtual channels, but it is recommended that the bit is set for these transactions (as their responses will be able to pass posted writes).

ReqVC

Specifies the virtual channel of the request. See Virtual Channels.

ReqChain

Optional field that provides a hint that subsequent requests in the same VC should be grouped together for routing purposes.

ReqStWayId, ReqStUpdate, ReqStInvalidate

In implementations that have a "shadow tag" of the originator's cache in order to limit probes, the fields ReqStWayId, ReqStUpdate, ReqStInvalidate together indicate the caching and replacement algorithm at the originator. The completer then updates the shadow tags to maintain an accurate state of all cache lines that are being cached coherently. These shadow tags are used to limit any probes to only the cache lines known to be in the originator's cache. Despite these shadow tags that should accurately limit probe, the originator must not make it an error to receive a probe that does not hit in the cache.

Implementation Note: It is required that the originator and completer share the same algorithm that chooses an index based on the ReqAddr, as the index is not specified on SDP.

Table 18 specifies the meaning of these bits and how they relate to the intended cache allocation or invalidation that is caused by this transaction.

These fields are reserved, and ReqStUpdate and ReqStInvalidate should be 00b, for all non-coherent commands (RdSized*NC, WrSized*NC, AtomicNC*).

Table 18 Cache Tag Tracking

Command	ReqStUpdate	ReqStInvalidate	Meaning/Result
RdBlk*, ChgToX*, ValBlk	0	1	The originator intends to cache the line non-coherently at the way specified by ReqStWayId. The completer is directed to invalidate any prior tag shadow at the appropriate index. The originator does not require future probes for this address as it does not have a coherent copy.
	1	0	The originator intends to cache the line coherently at the way specified by ReqStWayId. The completer is directed to update the tag shadow at the appropriate index with this new address. The originator will require future probes for this address.
VicBlk*	1	X	The cache state transition on victims is specified in ReqAttr.
WrSized, WrSizedFull, WrSizedFullZero, WrSizedFullComp, Atomic, AtomicNR	0	0	As part of this write, the originator does not intend to cache the line and did not have the line previously in the cache. The completer makes no update to the shadow tags. ReqStWayId is reserved. The originator does not require future probes for this address as it does not have a coherent copy.
	0	1	As part of this write, the originator has invalidated the current copy of the cache line. The completer is directed to invalidate any prior tag shadow at the appropriate index. The originator does not require future probes for this address as it does not have a copy. The way of the line is not specified by ReqStWayId, which is reserved. The completer uses a normal search of all ways at the appropriate index to invalidate any prior shadow tag.
	1	1	As part of this write, the originator has cached the line coherently at the way specified by ReqStWayId. The completer is directed to update the tag shadow at the appropriate index with this new address. The originator will require future probes for this address.
All other commands	0	0	No shadow tag update is performed. ReqStWayId is reserved.
Notes: 1) Other combinations of ReqStUpdate and ReqStInvalidate are reserved.			

ReqCacheHint (ReqNoAlloc)

This field provides temporal hints for reads, writes, victims and atomics. The field is reserved when ReqIO=1 or ReqSpecialAddr=1.

Table 19 ReqCacheHint Usage

Value	Meaning
00b	Regular temporal re-use (RT)
01b	Low (non-temporal) re-use (NT)
10b	High temporal re-use (HT)
11b	Reads: No allocation, last use (discard dirty). Write: Write-through. Atomics: Write-through, last use.

In prior specifications, ReqCacheHint was named ReqNoAlloc and was a single bit. ReqNoAlloc can be extended to ReqCacheHint by setting ReqCacheHint[1]=0b and ReqCacheHint[0]=ReqNoAlloc.

ReqCompMode

This field specifies the request compression mode for metadata compression.

Table 20 ReqCompMode Usage

Command	Value	Meaning
Atomic* and WrSized* (except WrSizedFullComp)	00b	Bypass compression. For writes, ReqAttr[7:6] must be 00b to indicate that the data is uncompressed.
	01b	Compression is enabled as indicated by ReqAttr[7:6]. The completer may also (optionally) compress the data.
	10b	Compression is enabled as indicated by ReqAttr[7:6] but any additional completer compression is disabled.
	11b	Reserved
RdSized*, RdBlk*, and SpecPrefCmd	00b	Bypass compression. Data will be returned uncompressed with RdRspDataCompMeta specifying uncompressed data. The completer may also bypass fetching the compression metadata associated with this block (the data is known to be uncompressed).
	01b	Read compressed. Data may be returned compressed or uncompressed as indicated in the metadata in RdRspDataCompMeta. When compressed, the number of data beats returned may be smaller than the block size specified by ReqLen.
	10b	Read uncompressed. Data will be returned uncompressed with RdRspDataCompMeta specifying uncompressed data. The completer uses the compression metadata to uncompress the data block.
	11b	Reserved
WrNoDataNC	11b	Force decompression. The completer will use the compression metadata to determine if the data block is compressed. If it is compressed, the completer will uncompress the data.
	00b, 01b, 11b	Reserved
All other commands	n/a	Reserved

ReqSpecDataFetch

This field provides a hint to the completer on whether it should prefetch the memory block before the contents of the compression metadata is known. This field is valid for non-caching reads, writes, atomics and caching reads. It is reserved (00b) for all other commands.

Table 21 ReqSpecDataFetch Usage

Value	Meaning
00b	Prefetch recommended based on bandwidth usage. The originator indicates that the data block should be prefetched but only if the completer is not currently bandwidth constrained.
01b	Prefetch is not recommended. The originator indicates that the data block is likely compressed and the metadata should be fetched first to determine the actual length of the compressed data.
10b	Prefetch recommended. The originator indicates that the data block should be prefetched (it is likely to be uncompressed data).
11b	Reserved

ReqVld

This signal indicates that the originator is presenting valid information on this channel.

ReqCreditType

When ReqCreditType = 1, credit being released is a pool credit, which may be used by requests in any virtual channel. When ReqCreditType = 0, the credit being released is a VC-specific credit. See Channel Flow Control for more information on credits and flow control.

ReqCreditVC

Specifies the virtual channel of the credit being released. Only valid if ReqCreditType = 0.

ReqCreditVld

When ReqCreditVld is asserted the port (or completer, at a completer port) is releasing one or more credits. When implemented, ReqCreditCnt, ReqCreditType, ReqCreditVC, and ReqCreditChanAB provide information about the type and number of credits being released. When none of the optional ReqCredit* fields are implemented, one Originator Request Channel credit is released whenever this signal is asserted on an active clock edge.

ReqUser

Optional signals that may be defined for any purpose. Information flow must be in the same direction as the channel. *ReqUser bits are not automatically copied into the response.*

2.2.7 Read Response Channel

The Read Response Channel(s) provide read request meta-data, completion status, and response data for a specific read request. In addition, if the Completer Fatal Error Notification Feature is implemented ***and*** no Write Response Channel is implemented, a completer may also send an unsolicited read response with RdRspStatus=SYSFATALERR that is not associated to any outstanding request. Refer to Using the Write Response Channel for Unsolicited Fatal Errors for more information.

A read response channel provides the requested data beginning in a subsequent clock cycle based on a specific heads-up interval. See discussion in Data Transfer.

Data is returned in the RdRspData field in one or more data transfer cycles, also known as beats. A response is matched to the request that evoked it based on the information in the RdRspUnitID and RdRspTag fields. For a read response to correspond to a read request the RdRspUnitID of the response must match the UnitID of the request and the RdRspTag must match the ReqTag.

Table 22 Read Response Channel Signals lists the signals that make up a read response channel. All the signal names use the convention for an interface with a single channel.

The column labeled **Required** indicates what optional feature requires this field to be implemented. "Yes" means that the feature is required for all implementations. The entire Read Response Channel is optional if the originator does not implement the Read Response feature and does not issue any commands that generate a read response (write-only interface).

The column labeled **Driver** indicates the direction of information flow within the field at an originator port.

Table 22 Read Response Channel Signals

Name	Size	Driver	Description	Required
RdRspTag	N_{rt}	Port	Read response tag. This field contains the tag of the request that initiated the transaction. The width of this field equals the width of the request tag.	Yes

RdRspUnitID	N_{ui}	Port	Read response UnitID. This field specifies the ReqUnitID from the request. Used by the originator to match this response to the unit that initiated it. Since tag values are allocated per unit, the unit ID qualifies the tag value. The width of this field equals the width of the request unit ID (N_{ui}).	Yes, except for a completer when the CS flattens the tag space.
RdRspVC	N_{rvc}	Port	Read response virtual channel. Specifies the virtual channel of the response. Matches the virtual channel of the initiating request.	Virtual Channel Support
RdRspPassPW	1	Port	Read response pass posted write. Determines if the read response may pass a prior posted write. • 1: Response may pass previous writes in the posted virtual channel from the same source to their destination. • 0: Response must not pass posted writes traveling in the same direction. The value of this field is normally based on the ReqRspPassPW of the request that is being completed. It is not meaningful for responses in any virtual channel other than non-posted.	Posted write ordering feature
RdRspParity	1	Port	Read Response Parity. Maintains even parity across all port-driven (or completer-driven, at a completer port) read response channel fields except RdRspVld, RdRspDataVld, RdRspData*, RdRspClkEn_m1, RdRspDataClkEn_m1, and RdRspCredit*.	Enhanced RAS V1 feature
RdRspDelay	1	Port	Read response data lag. Indicates which, of two, heads-up interval N_{lag} is being applied to this response. • 0: $N_{lag} = N_{short\_lag}$ • 1: $N_{lag} = N_{long\_lag}$ See Data Transfer for more details. If the variable heads-up feature is not implemented, the lag between the response and the data is fixed at N_{short_lag}.	Variable heads-up feature
RdRspStatus	N_{rdstat}	Port	Read response status. This field indicates the status of the read request. RdRspStatus is required to be the same across all beats of a response, including any RETRANSMIT. See RdRspStatus for more information about this field. Ports that do not support all error encodings may make this width smaller by omitting higher order bits.	Yes
RdRspOffset	N_{rdoff}	Port	Read response offset. Indicates which beat of data will be returned N_{lag} clock cycles in the future. See RdRspOffset below for more information about this field.	Yes
RdRspLast	1	Port	Read response last. This signal is asserted with the response associated with the final beat of a data transfer. See Data Transfer for more detail.	Yes

RdRspSrcData	6	Port	Provides context on the source of the data. This field is only valid for a response that does not have an error.	Data Source Feature

Encoding	**Meaning**
RdRspSrcData[5:4] – Responder Type differentiates the case when a response was returned from another cache in the system and the locality of this cache.	
00	Data returned from the device/memory.
01	Data was the result of a cache hit from a cache within the same NUMA node as this originator.
10	Data was the result of a cache hit from a cache within the same package (processor) as this originator, but different NUMA node.
11	Data was the result of a cache hit from a cache within a different package (processor) as this originator.
RdRspSrcData[3:2] – Memory Locality specifies the locality of the target (memory channel, IO device) of the request.	
00	The target for the request's address is within the same NUMA node as this originator.
01	The target for the request's address is not within the same NUMA node as this originator, but it is within the same package (processor).
10	The target for the request's address is within a different package (processor) as this originator.
11	Undetermined. This encoding may be used when the target was not involved in the access (due to a cache hit) or in cases where the NUMA locality of either the request address or the originator is not precise.
Note: IO devices may have locality as well and may respond with RdRspSrcData[3:2]=00b if the address is within the same IO-NUMA node as the originator.	
RdRspSrcData[1:0] – Memory Type specifies the type of the target (memory channel, IO device) of the request.	
00	Fast storage
01	Slow storage
10	IO
11	Reserved
Note: The definition of "fast" vs "slow" storage is outside the scope of this specification. Fast storage may refer to DRAM while slow storage may refer to NVDIMM or externally attached devices for example.	

RdRspVld	1	Port	Read response valid. When the heads-up feature is not implemented, this signal indicates that the port (or completer) is presenting valid information on this channel. When the heads-up feature is implemented, the read response channel is split into two sub-channels (response and data sub-channel) and this signal indicates that the port (or completer) is presenting valid information on the response sub-channel (all fields except RdRspDataVld, RdRspData, RdRspDataParity, RdRspDataStatus, RdRspDataStatusParity and RdRspDataUser).	Yes
RdRspClkEn_m1	1	Port	Read response clock enable. RdRspClkEn_m1 must be asserted one clock cycle before RdRspVld is asserted.	Low-Power Signaling feature
RdRspUser	—	Port	User-defined field. If implemented, information must flow in the same direction as the read response channel with the same timing as the read response.	User-defined
Data Fields (possibly lagged when the heads-up feature is implemented)				
RdRspData	N_{rd}	Port	Read response data. If the read was successful, this field carries the requested data. The transfer of a block of data may require more than one clock cycle. RdRspData provides more detail on this field.	Yes
RdRspDataParity	N_{rd} /64	Port	Read Response Data Parity. Each bit in RdRspDataParity provides even parity based error protection for 64 bits of the RdRspData field. Except for when RdRspDataStatus is NODATA, the number of set bits in RdRspDataParity[i] and RdRspData[((i+1)*64-1):(i*64)] is always even. Even parity must be provided by the port or completer, including cases when read data was masked due to the length or the address of the command, RETRANSMIT and DATERR.	Yes, unless RdRspDataUser bits implement a superior protection, such as ECC.
RdRspDataStatus	2	Port	Read response data status. This field indicates the status of the read transfer and is sent with the same timing as RdRspData. See RdRspDataStatus for the encoding of this field. It is recommended that this field is ignored when RdRspVld (or RdRspDataVld with the heads-up feature) is not asserted.	Yes
RdRspDataStatusParity	1	Port	Read response data status parity. Maintains even parity across the RdRspDataStatus field. This field is sent with the same timing as RdRspData.	Enhanced RAS V2 feature
RdRspDataCompMeta	N_{md}	Port	Read response compression metadata. If this field (except for parity) is all ones, the data is uncompressed. Otherwise, this field indicates how to uncompress RdRspData. See MetaData Compression. This field also includes a parity bit in addition to what is normally considered to be the metadata. When the size of the metadata is "n" bits, this field is "n+1" bits (N_{md} is one more than the size of the metadata). The actual metadata is in RdRspDataCompMeta[N_{md}-2:0]. RdRspDataCompMeta[N_{md}-1] is a parity bit and maintains even parity across all of RdRspDataCompMeta.	Metadata Compression feature
RdRspDataUser	—	Port	User-defined field. If implemented, information must flow in the same direction as the read response data with the same timing as RdRspData. **Note:** RdRspDataParity does not cover RdRspDataUser. Any necessary parity for RdRspDataUser should be included in the RdRspDataUser bits themselves.	User-defined
RdRspDataVld	1	Port	Read response valid. When the heads-up feature is implemented, the read response channel is split into two sub-channels (response and data sub-channel). This signal indicates that the port (or completer) is presenting valid information on the data sub-channel (RdRspData, RdRspDataParity, RdRspDataStatus, RdRspDataStatusParity and RdRspDataUser). See Read Response Data Lag and the Variable Heads-Up Feature for more details.	Heads-up feature

RdRspDataClkEn_m1	1	Port	Read response data clock enable. RdRspDataClkEn_m1 must be asserted one clock cycle before RdRspDataVld is asserted.	Low-Power Signaling feature and Heads-up feature
Credit Fields				
RdRspCreditType	1	Originator	Read response credit type. When set, credit being released is a pool credit. When clear, credit is for a read response in the virtual channel indicated by the RdRspCreditVC field.	Virtual Channel Support
RdRspCreditVC	N_{rvc}	Originator	Read response credit virtual channel. Virtual channel of the credit being released. Only valid if RdRspCreditType is 0.	Virtual Channel Support
RdRspCreditParity	1	Originator	Read response credit parity. Maintains even parity across the RdRspCredit* fields excluding RdRspCreditVld, RdRspCreditClkEn_m1, and RdRspCreditRdy.	Enhanced RAS V3 feature
RdRspCreditCnt	$N_{cc[1]}$	Originator	Zero origin field of the number of credits being returned.	Credit Count Feature
RdRspCreditVld	1	Originator	Read response credit valid. Information presented on credit release sub-channel by the originator (or port, at a completer port) is valid.	Yes
RdRspCreditClkEn_m1	1	Originator	Read response credit clock enable. RdRspCreditClkEn_m1 must be asserted one clock cycle before RdRspCreditVld is asserted.	Low-Power Signaling feature
RdRspCreditRdy	1	Port or pervasive logic	Read response credit ready. This signal is used to pace the flow of read response credit information across the interface for clock synchronization purposes. See Signaling Requirements for more information on the *Rdy / *Vld handshake.	Yes
Notes: 1. Driven by the completer device at a completer port; not supported at originator ports. 2. Driven by the port at a completer port; not supported at originator ports.				

RdRspData

The size of this field is implementation-dependent, but must be an integer multiple of 8 no greater than system cache line size. This field lags N_{lag} cycles behind the read response. See Data Transfer for more detail.

RdRspStatus

RdRspStatus provides both a success/fail indication of the read response and, in addition, may indicate a cache state.

RdRspStatus may be 3, 4, or 6 bits in width as required by the implementation and specified in the port parameterization (N_{rdstat}). The required width depends on the type of commands issued and the port type:

- N_{rdstat} must be six bits if the port may issue caching read or cache update requests (RdBlk*, ChgToX*, etc).
  - A port that may issue only RdBlkS as the only caching read and does not issue any cache update requests is exempt from this rule.
- N_{rdstat} must be at least four bits if port uses AddrXlateRd commands or if the port has PCI attachments that may communicate completion timeouts or configuration request retries.
- All other implementations require N_{rdstat} to be at least three bits.

The overall read response completion status is provided in RdRspStatus[3:0] which also indicates the cause of any failure. However, when a cache read command is issued, RdRspStatus[5:3] is used to encode a cache state and RdRspStatus[2:0] indicates the read response completion status. When RdRspStatus[3] is encoding a cache state, only RdRspStatus[2:0] indicates the read response completion failure and Table 23 should be interpreted as if RdRspStatus[3] is zero.

The encoding of RdRspStatus[3:0] is shown in Table 23 RdRspStatus[3:0] Encoding.

Table 23 RdRspStatus[3:0] Encoding

RdRspStatus[3:0]	Response	Comment
0000b	OKAY (Normal completion)	The transaction completed normally.

0001b (AddrXlateRd only)	PERMISSION_VIOLATION	For AddrXlateRd, the translation could not be performed due to a permissions violation. This encoding is reserved for all other commands. **Architecture Note:** This encoding was previously reserved for "Monitor Locks", which are now deprecated.
0010b	SLVERR (Target Abort)	Indicates that the end-target of the transaction (i.e. the I/O device or the memory device) had an error while handling the transaction or is otherwise unable to complete the transaction. This error may or may not be persistent if the transaction is performed a second time. This error response aligns with the PCI defined "target abort".
0011b	DECERR (Decode Error)	Indicates a form of address decode error, such as an invalid address or an address that does not match the command type (e.g. performing certain cacheable transactions to an address that decodes to an I/O device). This error is normally persistent - if the same transaction with the same address is performed again without any configuration changes, a DECERR would be presented again.
0100b	EARLY (Early Response)	See discussion following.
0101b	OKAY_NODATA (Normal completion with no data)	The transaction completed normally, but no data is provided. This encoding must be accompanied with RdRspDataStatus=NODATA. See Read Response with NODATA.
0110b	PROTVIOL (Protection Violation)	Indicates that certain security or protection checks caused the transaction to be aborted. This error is normally persistent - if the same transaction with the same address is performed again without any configuration changes, a PROTVIOL would be presented again.
0111b	TRANSERR (Transaction Error)	Indicates that the transaction had an error at one or more layers between the end-target (as an error at the end-target would normally return SLVERR). This error may be at the completer-proxy, in the fabric, at any probed cache, or any hub or translation point before the actual end-target. This error may or may not be persistent if the transaction is performed a second time.
1000b (RdSized* only)	CMPTO (Completion Timeout)	Only valid for RdSized* commands.
1001b (AddrXlateRd only)	UNTRANSLATED	For AddrXlateRd, the translation could not be performed for an unspecified reason and may be retried. This encoding is reserved for all other commands.
1010b (AddrXlateRd only)	TRANSLATION_FAULT	For AddrXlateRd, the translation could not be performed due to an address translation fault. This encoding is reserved for all other commands.
1011b	*Reserved*	
1100b (RdSized* only)	CRS (Configuration Request Retry)	Only valid for RdSized* commands.
1101b	SYSFATALERR (System Fatal Error)	An unsolicited (not associated to an outstanding request) indication of a system fatal error. Refer to Using the Write Response Channel for Unsolicited Fatal Errors for more information. RdRspTag may be used to encode further information on the type or reason for the fatal error. Reserved (not used) if the Completer Fatal Error Notification Feature is not implemented or if any write response channel is implemented. An implementation ***must*** use a write response channel and not a read response channel for unsolicited fatal errors if a write response channel is implemented.
1110b	*Reserved*	
1111b	*Reserved*	
Implementation Note: CMPTO and CRS are reserved only for PCI attached SDP ports. The SOC-15 data fabric does not return CMPTO or CRS.		

EARLY indicates that the consolidated system-wide cache line state for the requested data has not yet been determined and the data being returned may be stale. A second Read Response will follow with either new data or no data (NODATA) to update the consolidated system state information. Support for the EARLY read response feature is optional and an originator is not required to support it.

For caching read or cache update commands (RdBlk*, ChgToX*) that do not end in an error response (RdRspStatus[2:0] = OKAY or OKAY_NODATA), RdRspStatus[3] = PassDirty and RdRspStatus[5:4] returns a consolidated ***system state*** of the requested cache line as encoded in Table 24 RdRspStatus[5:3] Encoding. The system state does not include the originator's cache state unless a command was used that probes the originator. The system state and PassDirty bit is not meaningful for any response to a command other than caching read and cache update commands, nor is it meaningful for any response that indicates an error in RdRspStatus[2:0]. For example, a RdBlkL that ends in a protection violation has not probed peer caches. Originators are expected to not cache a line when there is an error indicated in RdRspStatus[2:0].

For non-caching read commands (RdSized*) and AddrXlateRd, RdRspStatus[3] is used to encode the extended status shown in the table above (last 8 rows). The CMPTO and CRS encodings are intended only for ports directly connected to a PCI host bridge.

For ports that do not issue caching read or cache upgrade commands (non-coherent originators) and are not connected to a PCI host bridge, RdRspStatus[5:3] is *reserved*. For ports that do not issue caching read or cache upgrade commands (non-coherent originators) and are connected to a PCI host bridge, RdRspStatus[5:4] is *reserved*. It is recommended that these ports use the configuration parameter N_{rdstat} to instantiate only the bits necessary.

Table 24 RdRspStatus[5:3] Encoding

RdRspStatus[5:4]	RdRspStatus[3]	System State	Data state	Notes
00b	0	I	Clean	
	1		Dirty	
01b	0	S	Clean	
	1		Dirty	
10b	0	X (E, M, D, or Mp)	Clean	
	1		n/a.	This encoding is possible only as a result of a ClnBlkAll and may indicate that the line was cleaned (e.g. M->E) in the process. Originators may not depend that the RdRspStatus[3] indicates this nor may it rely that the system state did not change back from E->M after the probe(s) occurred.
11b	0	O	Clean	
	1	F	Clean	

RdRspDataStatus

This field provides information about the data being presented at the port interface via the RdRspData field. This field lags N_{lag} cycles behind the read response (valid with the same timing as RdRspData). The encoding of this field is defined in Table 25 RdRspDataStatus Encoding.

Table 25 RdRspDataStatus Encoding

RdRspDataStatus	Transfer Status	Description
00b	NODATA	There is no valid data associated with this response. Contents of the RdRspData field must be ignored. The use of this status is limited to the cases outlined in Read Response with NODATA.
01b	DATERR	The data transferred is known to be in error (an uncorrectable error occurred in its storage or transmission.)
10b	RETRANSMIT	When asserted on the last beat, indicates that a transient error occurred. The data will be retransmitted. When asserted on other beats, a transient or data error may have occurred. See Read Response Data Retransmission. Support for the RETRANSMIT is optional and an originator is not required to support it.
11b	VALID	The data presented in the RdRspData field is valid for this clock cycle.

RdRspOffset

Read data is returned to the originator in the RdRspData field in one or more transfer cycles called beats. Each beat is sequentially assigned a number starting at 0 and ending with n-1 based on the order of the data in memory; n is the total number of beats required to transfer the requested data. When the transfer requires more than one beat, the completer is allowed to return the beats in an order that differs from sequential order. This field provides the number of the beat that corresponds to this response.

Beat 0 (RdRspOffset = 0) will always contain the initial (lowest addressed) byte of the requested data payload. For a RdSized* command, the initial address is the value supplied in the ReqAddr field of the read request. For a RdBlk* command, the completer will return the cache block that includes the requested data. Therefore, the initial (lowest address) of beat 0 is the value of ReqAddr aligned to the system cache line size.

The value returned in the RdRspOffset field at clock x is the offset of beat that will be presented in the RdRspData* fields at cycle x + N_{lag} in the future.

Read Response Credit Release Sub-channel

When RdRspCreditVld is asserted, the originator (or port, at a completer port) is releasing one or more credits. When implemented, RdRspCreditCnt, RdRspCreditType, and RdRspCreditVC provide information about the type and number of credits being released. One credit represents the buffer space for one read response data block. Read response data block size is implementation-dependent but must be a power-of-2 multiple of the RdRspData field size in bytes up to the system cache line size. See Data Transfer for information on channel flow control.

2.2.8 Write Response Channel

The Write Response Channel(s) provide feedback to the initiator of a write request indicating the outcome and completion of the request at the target. Responses are matched with requests based on the UnitID and request tag of the initiator. Table 26 Write Response Channel Signals lists the signals that make up a write response channel. In addition, if the Completer Fatal Error Notification Feature is implemented, a completer may also send an unsolicited write response with WrRspStatus=SYSFATALERR that is not associated to any outstanding request. Refer to Using the Write Response Channel for Unsolicited Fatal Errors for more information.

The column labeled **Required** indicates what optional feature requires this field to be implemented. "Yes" means that the feature is required for all implementations. The entire Write Response Channel is optional if the originator does not implement the Write Response feature and does not issue any commands that generate a write response (read-only interface).

The column labeled **Driver** indicates the direction of information flow within the field at an originator port.

Table 26 Write Response Channel Signals

Name	Size	Driver	Description	Required
WrRspTag	N_{rt}	Port	Write response tag. This field contains the tag of the request that initiated the transaction. The width of this field equals the width of the request tag. This field is also used for the Completer Fatal Error Notification Feature and may encode the type and/or reason of fatal error.	Yes
WrRspUnitID	N_{ui}	Port	Write response unit ID. This field specifies the ReqUnitID from the request. Used by the originator to match this response to the unit that initiated the request. Since tag values are allocated per unit, the unit ID qualifies the tag value. The width of this field equals the width of the request unit ID (N_{ui}).	Yes, except for a completer when CS flattens tag space.
WrRspStatus	4	Port	Write response status. A completer that does not implement the status encodings for which WrRspStatus[3] = 1 are free to tie this signal to 0. WrRspStatus below gives the encoding of this field.	Yes, except as noted.
WrRspVC	N_{rvc}	Port	Write Response Virtual Channel. Virtual channel of the response. Matches the virtual channel of the originating request.	Virtual Channel Support
WrRspPassPW	1	Port	Write response pass posted write. Determines if the write response may pass a prior posted write. • 1: Response may pass previous writes in the posted write virtual channel from the same source to their destination. • 0: Response must not pass posted writes traveling in the same direction. The value of this field is normally based on the ReqRspPassPW of the request that is being completed. It is not meaningful except for responses in the non-posted virtual channel and it is recommended that it be 1 for all other virtual channels.	Posted write ordering feature

WrRspSrcData	4	Port	Provides context on the source of the data. This field is only valid for a response that does not have an error.	Data Source Feature

Encoding	Meaning
WrRspSrcData[3:2] – Memory Locality specifies the locality of the target (memory channel, IO device) of the request.	
00	The target for the request's address is within the same NUMA node as this originator.
01	The target for the request's address is not within the same NUMA node as this originator, but it is within the same package (processor).
10	The target for the request's address is within a different package (processor) as this originator.
11	Undetermined. This encoding may be used when the target was not involved in the access (due to a cache hit) or in cases where the NUMA locality of either the request address or the originator is not precise.
Note: IO devices may have locality as well and may respond with WrRspSrcData[3:2]=00b if the address is within the same IO-NUMA node as the originator.	
WrRspSrcData[1:0] – Memory Type specifies the type of the target (memory channel, IO device) of the request.	
00	Fast storage
01	Slow storage
10	IO
11	Reserved
Note: The definition of "fast" vs "slow" storage is outside the scope of this specification. Fast storage may refer to DRAM while slow storage may refer to NVDIMM or externally attached devices for example.	

WrRspParity	1	Port	Write response parity. Maintains even parity across all port-driven (or completer-driven, at a completer port) write response Channel signals excluding WrRspVld, WrRspClkEn_m1, and WrRspCreditRdy.	Enhanced RAS V1 feature
WrRspVld	1	Port	Write response valid. This signal indicates that the port (or completer) is presenting valid write response information on this channel.	Yes
WrRspClkEn_m1	1	Port	Write response clock enable. WrRspClkEn_m1 must be asserted one clock cycle before WrRspVld is asserted.	Low-Power Signaling feature
WrRspRdy	1	Originator or pervasive logic	Write response ready. This signal is used to pace the flow of write response information across the interface for clock synchronization purposes. See Signaling Requirements for more information on the *Rdy / *Vld handshake.	Yes
WrRspCreditType	1	Originator	• 1: Credit being released is a pool credit, which may be used by responses in any virtual channel. • 0: Credit is for a write response in a particular virtual channel, given by WrRspCreditVC.	Virtual Channel Support
WrRspCreditVC	N_{rvc}	Originator	Virtual channel of the credit being released. Only valid if WrRspCreditType = 0.	Virtual Channel Support
WrRspCreditParity	1	Originator	Write response credit parity. Maintains even parity across the WrRspCredit* fields excluding WrRspCreditVld, WrRspCreditClkEn_m1, and WrRspCreditRdy.	Enhanced RAS V3 feature
WrRspCreditCnt	$N_{cc[2]}$	Port	Zero origin field of the number of credits being returned.	Credit Count Feature
WrRspCreditVld	1	Originator	Write response credit valid. Information presented on credit release sub-channel by the originator (or port, at a completer port) is valid.	Yes
WrRspCreditClkEn_m1	1	Originator	Write response credit clock enable. WrRspCreditClkEn_m1 must be asserted one clock cycle before WrRspCreditVld is asserted.	Low-Power Signaling feature

WrRspCreditRdy	1	Port or pervasive logic	Write response credit ready. This signal is used to pace the flow of write response credit information across the interface for clock synchronization purposes. See Signaling Requirements for more information on the *Rdy / *Vld handshake.	Yes
WrRspUser	—	Port	User-defined. If implemented, information must flow in the same direction as the write response channel.	User-defined
Notes: 1. Driven by the completer device at a completer port; not supported at originator ports. 2. Driven by the port at a completer port; not supported at originator ports.				

WrRspStatus

Table 27 WrRspStatus Encoding

WrRspStatus[3:0]	Response	Comment
0000b	OKAY (Normal completion)	The transaction completed normally.
0001b	*Reserved*	Architecture Note: This encoding was previously reserved for "Monitor Locks", which are now deprecated.
0010b	SLVERR (Target Abort)	Indicates that the end-target of the transaction (i.e. the I/O device or the memory device) had an error while handling the transaction or is otherwise unable to complete the transaction. This error may or may not be persistent if the transaction is performed a second time. This error response aligns with the PCI defined "target abort". A SLVERR is uncommon, but legal, when performing "posted" writes.
0011b	DECERR (Decode Error)	Indicates a form of address decode error, such as an invalid address or an address that does not match the command type (e.g. performing certain cacheable transactions to an address that decodes to an I/O device). This error is normally persistent - if the same transaction with the same address is performed again without any configuration changes, a DECERR would be presented again.
0100b	EARLY (Early Response)	See discussion following.
0101b	SYSFATALERR (System Fatal Error)	An unsolicited (not associated to an outstanding request) indication of a system fatal error. Refer to Using the Write Response Channel for Unsolicited Fatal Errors for more information. WrRspTag may be used to encode further information on the type or reason for the fatal error. Reserved (not used) if the Completer Fatal Error Notification Feature is not implemented.
0110b	PROTVIOL (Protection Violation)	Indicates that certain security or protection checks caused the transaction to be aborted. This error is normally persistent - if the same transaction with the same address is performed again without any configuration changes, a PROTVIOL would be presented again.
0111b	TRANSERR (Transaction Error)	Indicates that the transaction had an error at one or more layers between the end-target (as an error at the end-target would normally return SLVERR). This error may be at the completer-proxy, in the fabric, at any probed cache, or any hub or translation point before the actual end-target. This error may or may not be persistent if the transaction is performed a second time.
1000b	CMPTO (Completion Timeout)	Apply only to sized writes. **Implementation Note:** CMPTO and CRS are reserved only for PCI attached SDP ports. The SOC-15 data fabric does not return CMPTO or CRS.
1001b	*Reserved*	
1010b	*Reserved*	
1011b	*Reserved*	
1100b	CRS (Configuration Request Retry)	
1101b	*Reserved*	
1110b	*Reserved*	
1111b	*Reserved*	

The completer (or completer proxy) may not generate a write response with any status other than EARLY prior to receiving all data associated with this request.

For a WrSized*, an EARLY response does not indicate that the final response will have 'normal completion'. It is possible that the transaction may end in error, for example due to a transaction error observed during the probing. The EARLY response only indicates that the responsibility of tracking it for error reporting and/or ordering is no longer necessary. Support for the EARLY write response feature is optional and an originator is not required to support it.

For non-coherent originator ports that are not connected to a PCI host bridge, WrRspStatus[3] is *reserved*.

Write Response Credit Release Sub-channel

When WrRspCreditVld is asserted the originator (or port, at a completer port) is releasing one or more credits. When implemented, WrRspCreditCnt, WrRspCreditType, and WrRspCreditVC provide information about the type and number of credits being released. See Data Transfer for information on channel flow control.

2.2.9 Response Acknowledge Channel

The Response Acknowledge Channel(s) are used by an originator to indicate to the fabric that it has committed the results of the indicated transaction. A response acknowledgement may not be sent before the last beat of the response is received without EARLY or RETRANSMIT status. For responses with NODATA, a response acknowledgement may be sent any time after receiving the read response with RdRspStatus=OKAY_NODATA. The originator is not required to wait for the lagged RdRspDataStatus of NODATA. This channel is required for originators that issue requests that require response acknowledgment. A completer port does not include this channel. Table 28 Response Acknowledge Channel Signals lists the signals that make up a response acknowledge channel.

Table 28 Response Acknowledge Channel Signals

Name	Size	Driver	Description	Required
AckVld	1	Originator	Acknowledge valid. This signal indicates that the transaction acknowledgment information presented by the originator is valid.	Response Ack feature
AckClkEn_m1	1	Originator	Acknowledge clock enable. AckClkEn_m1 must be asserted one clock cycle before AckVld is asserted.	Low-Power Signaling feature and Response Ack feature
AckRdy	1	Port or pervasive logic	Acknowledge ready. This signal is used to pace the flow of response acknowledge information across the interface for clock synchronization purposes. See Signaling Requirements for more information on the *Rdy / *Vld handshake.	Response Ack feature
AckTag	N_{rt}	Originator	Acknowledge tag. Tag of the transaction (request / response pair) being acknowledged.	Response Ack feature
AckUnitID	N_{ui}	Originator	Acknowledge UnitID. Unit ID of the transaction being acknowledged.	Response Ack feature
AckCancel	1	Originator	Acknowledge cancel. If the original request was a VicBlk*, or ChgToX* request, refer to the command for rules on setting AckCancel due to possible intervening probe requests. *Reserved for all other request types.*	Response Ack feature
AckParity	1	Originator	Acknowledge parity. Maintains even parity across all originator-driven response acknowledge channel fields except AckVld, AckClkEn_m1, and AckCreditRdy.	Enhanced RAS V1 feature
AckCreditVld	1	Port	Acknowledge credit valid. Information presented on credit release sub-channel by the originator (or port, at a completer port) is valid	Response Ack feature
AckCreditCnt	$N_{cc[3]}$	Port	Zero origin field of the number of credits being returned.	Credit Count Feature
AckCreditClkEn_m1	1	Port	Acknowledge credit clock enable. AckCreditClkEn_m1 must be asserted one clock cycle before AckCreditVld is asserted.	Low-Power Signaling feature and Response Ack feature
AckCreditRdy	1	Originator or pervasive logic	Acknowledge credit ready. This signal is used to pace the flow of response acknowledge credit information across the interface for clock synchronization purposes. See Signaling Requirements for more information on the *Rdy / *Vld handshake.	

2.2.10 Probe Request Channel

The Probe Request channel(s) are used by the fabric to send probe requests and other system messages to coherent originators, including possibly multiple probe units behind each coherent originator. A coherent probe unit acts as two separate coherent originators where a configured bit in ReqUnitID specifies the probe unit. These probe units share a single probe request, probe response and originator data channel but have separate PrbVld and PrbRspVld signals. Since only one probe unit can logically respond to a single probe with data, there is only one OrigDataVld signal. The fabric generates probe requests as part of satisfying an originator-initiated coherent read or write requests. The probe request channel(s) are also used to send system messages, such as distributed virtual memory (DVM) and system events (interrupts) to processors that can process these messages. This channel is required for originators that include a cache that participates in the system cache coherency protocol or one that processes system messages. A completer port does not include this channel. Table 29 Probe Request Channel Signals lists the signals that make up a probe request channel.

The Probe Request Channel may optionally support an additional Probe Compression Feature. See Probe Compression Feature for more information.

Table 29 Probe Request Channel Signals

Name	**Size**	**Driver @ Originator Port**	**Description**	**Required**
PrbTag	N_{pt}	Port	Probe request tag. Used to uniquely identify the probe request while the request is outstanding and match a probe response (if any) to the probe. When a response is required, as is the case for all coherency probes and for some system management messages, the PrbTag is used to match the probe response to the probe. The completer may not reuse a PrbTag until the probe response has been received. Since the Originator Data Channel(s) rely on ordering of data and does not include a tag, a PrbTag can be reused even before any associated data has arrived. Each probe response includes the tag of the probe request it satisfies. When a response is not required, the PrbTag field is reserved. The encoding of PrbAction determines if a response is not required. **Note:** The PrbTag is not related to the ReqTag used for the original request, and there is no method to correlate the two provided in the architecture.	Probe Support feature
PrbAddr	N_{pa}	Port	Probe request address. This field indicates the address of the cache line being probed or, for DVM / system messages, message-specific information. See PrbAddr below for more information on this field.	Probe Support feature
PrbAction	4	Port	Probe Action. For probe requests, this field is used to specify the expected next state for the probed cache line to enter, based on its current state. Also used to indicate that this request is a DVM/System message rather than a probe request. See PrbAction for the encoding of this field.	Probe Support feature
PrbRD	2	Port	Probe request return data. Indicates under what circumstances a probed cache should return data. If the PrbAction is a DVM/System Message (10xxb), this field is *reserved*. See PrbRD for the encoding of this field.	Probe Support feature
PrbChain	1	Port	This field, when set to 1, indicates that this system message is paired with a subsequent probe request that has the same PrbAction. This is used exclusively for DVMOpMsg. If the PrbAction is not "DVMOpMsg, no response required" (encoding 1000b), this field is *reserved*. See PrbChain and DVM Operations for more information.	Probe Chain feature
PrbVld	N_{pu}	Port	Probe request valid. These signal(s) indicate that the probe address and control information presented by the port are valid for each of the probe units. It is legal for multiple PrbVld signals to be asserted on a given cycle, in which case the probe goes to all asserted units (as indicated by PrbRdy).	Probe Support feature
PrbClkEn_m1	1	Port	Probe clock enable. PrbClkEn _m1 must be asserted one clock cycle before any PrbVld signal is asserted.	Low-Power Signaling feature and probe support

PrbRdy	1	Originator or pervasive logic	Probe request ready. This signal is used to pace the flow of probe request information across the interface for clock synchronization purposes. See Signaling Requirements for more information on the *Rdy / *Vld handshake.	Probe Support feature
PrbParity	1	Port	Probe Parity. Maintains even parity across all port-driven probe request channel fields excluding PrbVld, PrbClkEn_m1, and PrbCreditRdy.	Enhanced RAS V1 feature
PrbCreditCnt	$N_{cc[4]}$	Port	Zero origin field of the number of credits being returned.	Credit Count Feature
PrbCreditVld	1	Originator	Probe credit valid. Information presented on credit release sub-channel by the originator is valid.	Probe Support feature
PrbCreditClkEn_m1	1	Originator	Probe clock enable. PrbCreditClkEn _m1 must be asserted one clock cycle before PrbCreditVld is asserted.	Low-Power Signaling feature and probe support
PrbCreditRdy	1	Port or pervasive logic	Probe credit ready. This signal is used to pace the flow of probe request credit information across the interface for clock synchronization purposes. See Signaling Requirements for more information on the *Rdy / *Vld handshake.	Probe Support feature
PrbCompType	2	Port	Specifies the type of probe – normal, normal with index load, or compressed. See Probe Compression Feature for more information.	Probe Compression Feature
PrbCompIndex	N_{ct}	Port	Specifies a probe compression index table. See Probe Compression Feature for more information.	Probe Compression Feature

PrbAddr

In systems that do not support DVM / system messages, the width of PrbAddr is dictated by the physical address size supported by the system. All originators that support DVM are required to implement the PrbAddr bits specified in Table 83 DVM Field Assignments. Any additional bits required to support DVM are set to zero for memory coherency probes. All originators that support storing probes (steering transactions) are required to implement the PrbAddr bits specified in Table 51 Storing Probe PrbAddr Usage. All originators that support system messages are required to implement the PrbAddr bits specified in System Messages. Originators that do not support DVM, system messages and storing probes are not required to implement PrbAddr[5:2]. PrbAddr[1:0] are *reserved* and are not implemented. For some DVM and system messages, the PrbAddr field may take on a modified encoding as described in System Messages.

PrbAction

Specifies the requested change in cache line state for probes, the DVM/system management message, and/or whether a probe response is required. The upper two bits (PrbAction[3:2]) indicate whether it is a coherency probe or a DVM operation or system message. When the PrbAction is 1001b through 10011b, the PrbAddr is decoded to determine the system management message.

Refer to Table 30 PrbAction Encoding.

Table 30 PrbAction Encoding

PrbAction[3:2]	Type	PrbAction[3:0]	Action Requested
0xb	Coherency	0000b/0h	NOP/No Change
		0001b/1h	Share
		0010b/2h	Fetch
		0011b/3h	Clean
		0100b/4h	Migrate
		0101b/5h	Rinse
		0110b/6h	Invalidate
		0111b/7h	Store. See Storing Probes for more information on storing probes.
10x	DVM or System Management	1000b/8h	DVMOpMsg, no response required
		1001b/9h	System management message - no response required. Includes DVMSyncMsg.

		1010b/Ah	*Reserved* **Architectural Note**: This PrbAction encoding is used on fabric for internal interrupt messages.
		1011b/Bh	System management message - response required
11x	Coherency	1100b/Ch	*Reserved* **Architectural Note**: This PrbAction encoding is used by the fabric as a PageInvalidate to invalidate an entire page. However, fabric converts this to multiple cache line Invalidate(6h) commands.
		1101b/Dh	Demote **Architectural Note**: This PrbAction encoding is also used by the fabric as a PageDemote to demote an entire page. However, fabric converts this to multiple cache line Demote(Dh) commands. There is no cache line demote encoding on fabric.
		1110b/Eh	*Reserved* **Architectural Note**: This PrbAction encoding is used by the fabric.
		1111b/Fh	*Reserved*

PrbRD

Table 31 PrbRD Encoding

PrbRD	Return Data Requested
00b	Return no data. This encoding is illegal for all PrbAction fields that mandate a downgrade of "dirty data".
01b	Return data if cache line is in M, D, Mp, or O states
10b	Return data if cache line is in M, D, Mp, O, E, or F states (may respond with no data if the cache line state is E or F)
11b	Return data if cache line is in M, D, Mp, O, E, S, or F states (may respond with no data if the cache line state is E, F, or S)

PrbChain

This field is only used for DVMOpMsg probe packets (PrbAction of 1000b) and indicates the first of a DVM packet pair that must be processed together. PrbChain is reserved (must be zero) for all coherency and other system management probe packets.

A DVMOpMsg places the metadata of the message in the PrbAddr fields but it contains more data than can be held in a single PrbAddr field of one packet, so the message is spread over two probe packets

Upon receiving a packet on the probe chain with PrbChain=1, the originator should retain PrbAddr until a second (later) packet on a Probe Request Channel arrives with the same PrbAction of 1000b. The second packet has PrbChain=0 and fields other than the PrbAddr such as PrbTag are the same. PrbRD is reserved for DVMOpMsg probe packets and must be zero for both packets.

Upon receipt of the second packet (with PrbAction=1000b and PrbChain=0), the originator combines the two PrbAddr fields as a single DVMOpMsg as outlined in DVM Operations. Between these two packets, there may be other coherency or system management probe packets, but there may not be:

1) A DVMSync message.
2) Another DVMOpMsg with PrbChain=1. This case is specifically reserved in case future variants of the specification require three or more probe packets for certain DVM operation messages.
3) A SysMgmtJoin message.
   or
4) A SysMgmtLeave message.

Each of the two probe packets in the DVMOp message use separate credits with independent allocations. A client that accepts DVMOpMsg on a probe request channel must either send a minimum of two credits or must release the first credit (retaining PrbAddr in a separate buffer) so that the second packet may be delivered.

See DVM Operations for more information.

Probe Compression Feature

The probe compression feature requires both sides (the originator and the port/fabric) to implement a "probe compression table" of $2^{N_{ct}}$ entries. Each entry can hold PrbAddr[n:16], where n is specified by N_{pa} (highest physical address bit in PrbAddr). The table is indexed by a probe compression index, which is specified in PrbCompIndex when PrbCompType=01b. When PrbCompType=10b, there are two compression indices – one (PrbCompIndex0) that is specified in PrbCompIndex0 and another (PrbCompIndex1) that is specified in PrbAddr. PrbCompType is ignored if no PrbVld signal is asserted and operates as if PrbCompType=00b. All system management messages must specify PrbCompType=00b.

The encoding of PrbCompType is shown in Table 32.

Table 32 PrbCompType Encoding

PrbCompType	Return Data Requested
00b	The transferred probe request and PrbAddr does not use the probe compression feature and PrbAddr is treated as uncompressed. Ports without the probe compression feature operate as if PrbCompType is always 00b.
01b	The transferred probe request and PrbAddr is a single address that is not compressed. However, the originator is directed to load the probe compression table of index PrbCompIndex with PrbAddr[n:16]. Any prior contents at this table entry are discarded. After the probe compression table is loaded, the originator processes the probe based on the remaining fields in the probe channel.
10b	This transferred probe request and PrbAddr consists of two separate addresses (while consuming a single probe credit). The two probes are processed as two completely separate and unique probes and will be referred to as “probe0” and “probe1” with all non-credit fields except for PrbVld having two variants labeled “0” and “1” (e.g. PrbAction0 for probe0 and PrbAction1 for probe1). In general, probe0 utilizes the port signals and probe1 utilizes some portions of PrbAddr instead as shown below. When multiple probable units (and multiple PrbVld signals) exist, both probes are processed by the probable units as specified by PrbVld. For the first probe:

Field	Location of field
PrbAction0	Utilizes the port’s PrbAction
PrbTag0	Utilizes the port’s PrbTag
PrbRD0	Utilizes the port’s PrbRD
PrbCompIndex0	Utilizes the port’s PrbCompIndex
PrbAddr0[n:16]	Utilizes the probe compression table, indexed by PrbCompIndex0
PrbAddr0[15:2]	Utilizes the port’s PrbAddr[15:2]

For the second probe:

Field	Location of field
PrbAction1	Utilizes the port’s PrbAddr[35+N_{pt}:32+N_{pt}] (e.g. PrbAddr[42:39] when N_{pt}=7)
PrbTag1	Utilizes the port’s PrbAddr[31+N_{pt}:32] (e.g. PrbAddr[38:32] when N_{pt}=7)
PrbRD1	Utilizes the port’s PrbAddr[31:30]
PrbCompIndex1	Utilizes the port’s PrbAddr[38+N_{pt}:36+N_{pt}] (e.g. PrbAddr[45:43] when N_{pt}=7)
PrbAddr1[n:16]	Utilizes the probe compression table, indexed by PrbCompIndex1 (i.e. from PrbAddr[45:43] when N_{pt}=7))
PrbAddr1[15:2]	Utilizes the port’s PrbAddr[29:16]

PrbCompType	Return Data Requested
11b	*Reserved*

The port (or fabric) controls the loading of the table and, once loaded (through PrbCompType=01b), can then send two compressed probes within one SDP transaction using PrbCompType=10b. In a compressed probe, since the high order bits of the probe address are specified from the probe compression table, this frees up portions of PrbAddr to hold separate copies of PrbAction, PrbTag, PrbRD, and PrbCompIndex that are used for the second of these two probes.

The probe compression feature does not alter SDP credit handling. When PrbCompType specifies two compressed probes (10b), the transaction still consumes a single probe credit even though there are two probes in one transaction. The probe compression feature also does not alter the Probe Response Channel. Other than using a single probe credit, each of the compressed probes are independent and must be responded with two unique responses. There is no requirement that the responses for these two probes are provided in any specified order.

It is illegal for the port to specify a compressed probe with a PrbCompIndex value that has not previously been loaded. The probe table resets to a state where all entries are invalid and cannot be used until loaded. In addition, the originator may clear the probe compression table during power management events. The method that the port is informed that the originator has gone through a power management is outside the scope of this specification.

All system management probe messages must specify PrbCompType=00b. The probe compression feature may only be used for memory coherency probes (when PrbAction != 10xx).

2.2.11 Probe Response Channel

The Probe Response Channel(s) are used to respond to probe requests. Each probe unit behind a coherent originator is expected to separately reply when it observes its PrbVld signal asserted on a probe. Since probe requests are initiated by the fabric, responses are returned to the fabric. This channel is required for originators that include a cache that participates in the system cache coherency protocol. A completer port does not include this channel. Table 33 Probe Response Channel Signals lists the signals that make up a probe response channel.

Table 33 Probe Response Channel Signals

Name	Size	Driver	Description	Required
PrbRspTag	N_{pt}	Originator	Probe response tag. Tag of the probe request that generated this response.	Probe Support feature
PrbRspStatus	5	Originator	Probe response status. This field indicates the response to a probe request and how it completes. This field is bit-encoded as follows: • ProbeRspStatus[0]: Data Transfer • ProbeRspStatus[1]: Transaction Error • ProbeRspStatus[2]: PassDirty • ProbeRspStatus[4:3]: State See Probed Cache State Transitions and Responses for more information. When Data Transfer is specified, the originator must also transfer data on the Originator (Write/Probe) Data Channel. In addition to the Probe Response Channel flow control, one or more credits must be obtained on the Originator Write/Probe Data Channel as defined in Channel Flow Control. Furthermore, data transfer on the Originator Probe Data Channel must follow the same order as the probe responses appear on the Probe Response Channel.	Probe Support feature
PrbRspParity	1	Originator	Probe Response Parity. Maintains even parity across all originator-driven probe response channel signals excluding PrbRspVld. PrbRspClkEn_m1, and PrbRspCreditRdy.	Enhanced RAS V1 feature
PrbRspVld	N_{pu}	Originator	Probe response valid. These signal(s) indicate that the information presented by the originator on the probe response channel is valid. Each probe unit behind the coherent originator is assigned a separate PrbRspVld signal. Each probe unit must respond separately. Only one PrbRspVld may be asserted on any given cycle. Note that a probe response from one probe unit consumes a single probe response credit even if more than one probe unit will respond (in separate cycles) to the same probe request. **Architecture Note**: Future revisions of the specification may allow multiple probe units to respond to the same probe request in a single cycle; but this is not allowed in this revision.	Probe Support feature
PrbRspClkEn_m1	1	Originator	Probe response clock enable. PrbRspClkEn_m1 must be asserted one clock cycle before PrbRspVld is asserted.	Low-Power Signaling feature and probe support feature
PrbRspRdy	1	Port or pervasive logic	Probe response ready. This signal is used to pace the flow of probe response information across the interface for clock synchronization purposes. See Signaling Requirements for more information on the *Rdy / *Vld handshake.	Probe Support feature
PrbRspCreditCnt	$N_{cc[5]}$	Port	Zero origin field of the number of credits being returned.	Credit Count Feature
PrbRspCreditVld	1	Port	Probe response credit valid. A Probe Response Channel flow control credit is released whenever this signal is asserted.	Probe Support feature
PrbRspCreditClkEn_m1	1	Port	Probe response credit clock enable. PrbRspCreditClkEn_m1 must be asserted one clock cycle before PrbRspCreditVld is asserted.	Low-Power Signaling feature and probe support feature
PrbRspCreditRdy	1	Originator or pervasive logic	Probe response credit ready. This signal is used to pace the flow of probe response credit information across the interface for clock synchronization purposes. See Signaling Requirements for more information on the *Rdy / *Vld handshake.	Probe Support feature

2.2.12 Originator (Write/Probe) Data Channel

Write data and probe response data utilize the same physical channel. The OrigDataChan field indicates whether the data is write data or probe response data. Table 34 Originator (Write/Probe) Data Channel Signals lists the signals that make up an originator data channel. Data transfer for write data and probe response data is necessarily independent so this specification may refer to the "Originator Write Data Channel" or the "Originator Probe Data Channel" when indicating a flow within one of these.

The column labeled **Required** indicates what optional feature requires this field to be implemented. "Yes" means that the feature is required for all implementations.

The column labeled **Driver** indicates the direction of information flow within the field at an originator port.

Table 34 Originator (Write/Probe) Data Channel Signals

Name	**Size**	**Driver**	**Description**	**Required**
OrigDataChan	1	Originator	If 1, channel carries probe response data. If 0, channel carries write data. This bit is not present and assumed to be 0 if the Probe Request Channel is not present.	Probe Support feature
OrigData	N_{od}	Originator	Originator data. Write request or probe response data. The size of this field is implementation-dependent, but must be an integer multiple of eight bits (one byte) up to and including the system cache line size.	Yes
OrigDataBytEn	Max(N_{be},N_{md}] if metadata compression is supported or 1 if Byte Enable Compression is supported or N_{be}	Originator	Originator data byte enable. If byte enable compression is not supported, there is OrigDataBytEn bit for each eight bits of the OrigData field. These signals indicate which byte lanes hold valid data. Valid for both write request data and probe response data. If byte enable compression is supported, there is one OrigDataBytEn bit and it is asserted on the first beat(s) of data transfer if there are byte enables being transferred. If no byte enables are transferred on the data beats, then the byte enables are all asserted. See Byte Enable Compression. If metadata compression is supported, any originator data that is associated with a WrSized* with ReqAttr[7:6] != 00b carries the metadata in the low order bits (OrigDataBytEn[N_{md}-2:0]) and even parity in OrigDataBytEn[N_{md}-1]. Any bits above N_{md} are ignored. The actual byte enables for metadata compressed writes are implied to be all ones. See MetaData Compression.	Yes
OrigDataLast	1	Originator	Originator data last. This signal indicates the last beat in an originator data transfer.	Yes
OrigDataError	1	Originator	Originator Data Error. When asserted this signal indicates that the current data beat contains a data error. Valid for both write request and probe response data.	Yes
OrigDataOffset	N_{odoff}	Originator	Originator data offset. Indicates the index of the current data beat. See OrigDataOffset below.	Yes
OrigDataVC	N_{rvc}	Originator	Originator data virtual channel. Virtual channel to be used for the data transfer. On write data, this field must match the ReqVC for the write. For probe data, this field is reserved.	Virtual Channel Support
OrigDataParity	N_{od}/64	Originator	Originator data parity. Each bit in OrigDataParity provides even-parity based error protection for the OrigData field. The number of set bits in OrigDataParity[i] and OrigData[((i+1)*64-1):(i*64)] is always even, including data bytes where the OrigDataBytEn masks some or all bytes in that parity group. One bit is provided for every 64 bits. If the OrigData field width / 64 is not an integer, a separate even parity bit should be provided for remaining (N_{rd} MOD 64) bits.	Yes, unless OrigDataUser bits implement a superior protection, such as ECC.
OrigDataMetaParity	1	Originator	Originator data meta-data parity. Maintains even parity for all originator-driven (or port-driven, at a completer port) signals except for OrigData, OrigDataParity, OrigDataVld, OrigDataClkEn_m1, and OrigDataCreditRdy.	Enhanced RAS V1 feature

OrigDataChanAB	1	N/C[1]	Originator data channel A/B select. Only defined for completer ports. • 0: Data presented is for a write request on channel A. • 1: Data presented is for a write request on channel B.	Physical Channel Multiplexing feature
OrigDataVld	1	Originator	Originator data valid. This signal indicates that information presented on the originator data channel is valid. When multiple probe units are implemented, the probe units share a single originator data channel and one OrigDataVld signal. Due to the coherency states, it is logically impossible for more than one probe unit to respond with data on any single probe so there is no need to have N_{pu} OrigDataVld signals.	Yes
OrigDataClkEn_m1	1	Originator	Originator data clock enable. OrigDataClkEn_m1 must be asserted one clock cycle before OrigDataVld is asserted.	Low-Power Signaling feature
OrigDataRdy	1	Port or pervasive logic	Originator data ready. This signal is used to pace the flow of originator data across the interface for clock synchronization purposes. See Signaling Requirements for more information on the *Rdy / *Vld handshake.	Yes
OrigDataCreditChanAB	1	N/C[2]	Originator data credit release channel A/B. Used by the completer to indicate which channel is releasing a credit. Only defined for completer ports. • 0: Credit is for channel A. • 1: Credit is for channel B.	Physical Channel Multiplexing feature
OrigDataPrbCreditRel	1	Port	Originator data probe credit release. • 0: Credit being released is for write data, or, when virtual channel support is *not* implemented, the credit being released may also be a pool credit that may be used for other write data or probe response data. • 1: Credit being released is for probe response data. When not implemented, this field is treated as if it were 0. Not implemented at completer ports or originator ports that do not support probes. See Credit Release Sub-channel for more information.	Probe Support feature
OrigDataCreditType	1	Port	Originator data credit type. When set, credit being released is a pool credit which may be used by write data in any virtual channel set or as probe response data. When clear, credit is either for write data in OrigDataCreditVC virtual channel (when OrigDataPrbCreditRel is 0) or for probe response data (when OrigDataPrbCreditRel is 1). See Credit Release Sub-channel for more information.	Virtual Channel Support
OrigDataCreditVC	N_{rvc}	Port	Originator data credit virtual channel. Virtual channel of the credit being released. Only valid if OrigDataCreditType = 0 and OrigDataPrbCreditRel = 0. See Credit Release Sub-channel for more information.	Virtual Channel Support
OrigDataCreditParity	1	Port	Originator data credit parity. Maintains even parity across the implemented OrigData*Credit* fields except OrigDataCreditVld, OrigDataCreditClkEn_m1, and OrigDataCreditRdy.	Enhanced RAS V3 feature
OrigDataCreditCnt	$N_{cc[6]}$	Port	Zero origin field of the number of credits being returned.	Credit Count Feature
OrigDataCreditVld	1	Port	Originator data credit valid. Information presented on credit release sub-channel by the port (or completer) is valid.	Yes
OrigDataCreditClkEn_m1	1	Port	Originator data credit clock enable. OrigDataCreditClkEn_m1 must be asserted one clock cycle before OrigDataCreditVld is asserted.	Low-Power Signaling feature

OrigDataCreditRdy	1	Originator or pervasive logic	Originator data credit ready. This signal is used to pace the flow of originator data credit information across the interface for clock synchronization purposes. See Signaling Requirements for more information on the *Rdy / *Vld handshake.	Yes
OrigDataUser	—	Originator	User-defined.	User-defined
Notes: 1. Driven by the port at a completer port; not supported at originator ports. 2. Driven by the completer at completer ports; not supported at originator ports.				

OrigDataOffset

Data to be written is transferred from the originator in the OrigData field in one or more cycles called beats. Each beat is sequentially assigned a number starting at 0 and ending with n-1, based on the order of the data in memory; n is the total number of beats required to transfer the data to be written. When the transfer requires more than one beat, the originator is allowed to transfer the beats in an order that differs from sequential order. This field provides the number of the beat being presented on this transfer cycle.

Beat 0 (OrigDataOffset = 0) will always contain the initial (lowest addressed) byte of the data payload.

For dual-operand vector atomics, OrigDataOffset for the second operand data is relative to the memory address plus the size of the operand.

Credit Release Sub-channel

One or more data credits are released whenever the OrigDataCreditVld signal is asserted on an active edge. The OrigDataCreditCnt, OrigDataPrbCreditRel, OrigDataPrbCreditType and OrigDataCreditVC fields specify the type and number of credits as shown in the following table. When none of the optional OrigData*Credit* fields are implemented, one write data credit is released whenever the OrigDataCreditVld signal is asserted on an active clock edge.

An originator that returns probe response data is required to reserve at least one pool credit for probe response data in order to prevent deadlock.

Table 35 Originator (Write/Probe) Data Credit Type

*Credit Type	*PrbCreditRel	*CreditVC	Virtual Channel Support	Credit Type
1	Don't care	Don't care	Don't care	Credit(s) being released may be used by the originator for either write data (in any virtual channel) or for probe response data.
0	1	Don't care	Don't care	Credit(s) being released may be used by the originator for probe response data; or returned from a prior probe response data transfer.
0	0	Any value	Yes	Credit(s) being released may be used by the originator for write data in the virtual channel specified by OrigDataCreditVC; or returned for a prior write data transfer in the virtual channel specified by OrigDataCreditVC.
		N/A	No	Credit(s) being released is either: - Pool credit(s) which may be used by the originator for either write data or for probe response data. - Credit(s) which may be used by the originator for write data in the virtual channel specified by OrigDataCreditVC; or returned for a prior write data transfer.

2.3 Port Parameterization

Scalable Data Port architecture allows for the parameterization of various field widths to support different applications and implementations. Many, but not all, of the parameters are independent. Field widths that must match share the same parameter name. For example, the number of bits in a ReqTag must be the same as the number of bits in a RdRspTag. This value is represented by the variable named N_{rt}.

Table 36 Port Field Widths lists all port parameters and the fields to which these parameters apply.

Table 36 Port Field Widths

Parameter	Port Field
N_{rt}	ReqTag, WrRspTag, RdRspTag, AckTag
N_{ui}	ReqUnitID, WrRspUnitID, RdRspUnitID, AckUnitID
N_{rdoff}	RdRspOffset
N_{odoff}	OrigDataOffset
N_{rvc}	ReqVC, ReqCreditVC, RdRspVC, RdRespCreditVC, WrRspVC, OrigDataVC, OrigDataCreditVC
N_{ra}	ReqAddr
Nrsa	ReqSubAddr
N_{pa}	PrbAddr
N_{si}	ReqStreamID
N_{vf}	ReqVfid
N_{od}	OrigData
N_{md}	RdRspDataCompMeta
N_{be}, 1, or max(N_{be}, N_{md})	OrigDataBytEn. The width of this field is dependent on the byte enable compression and the metadata compression feature support.
N_{rd}	RdRspData
N_{rdstat}	RdRspStatus
N_{pt}	PrbTag, PrbRspTag
N_{rl}	ReqLen
N_{short_lag}	The number of cycles (whole number) between the response sub-channel and the data-channel when RdRspDelay = 0 (or always if the variable heads-up feature is not implemented). This value may be programmable as long as both sides have the same value prior to port activation.
N_{long_lag}	The number of cycles (whole number) between the response sub-channel and the data-channel when RdRspDelay = 1. This value may be programmable as long as both sides have the same value prior to port activation. This parameter is used only for ports that implement the variable heads-up feature.
N_{rdy_shift}	The number of clock cycles (whole number) that the *Rdy must be flopped in order to align it to the actual cycle that it indicates. For example, if N_{rdy_shift} is 4, then *Rdy_m4 is provided to indicate four cycles in advance whether the port will be ready.
N_{pu}	The number of probe units behind a coherent originator. Each probe unit may be probed separately.
N_{rsl}	ReqSecLevel
N_{pri}	ReqQosPriority
N_{ct}	PrbCompIndex. This feature indicates $\log_2$ of the number of index tables that can be used in the Probe Compression feature.
$N_{cc[6:0]}$	ReqCreditCnt, AckCreditCnt, RdRspCreditCnt, WrRspCreditCnt, PrbCreditCnt, PrbRspCreditCnt, and OrigDataCreditCnt. A value of "-1" as a credit count indicates that there is no field for *CreditCnt and each credit is therefore implicitly one credit (as if *CreditCnt was zero).
N_{st}	ReqStWayId
Note: Parameters representing the width of user-defined fields are not included in this table. The width of a user-defined field is determined based on the application.	

2.4 Signaling Requirements

Figure 4 Port Interface Valid and Ready Signaling provides an overview the signaling requirements for the SDP interface.

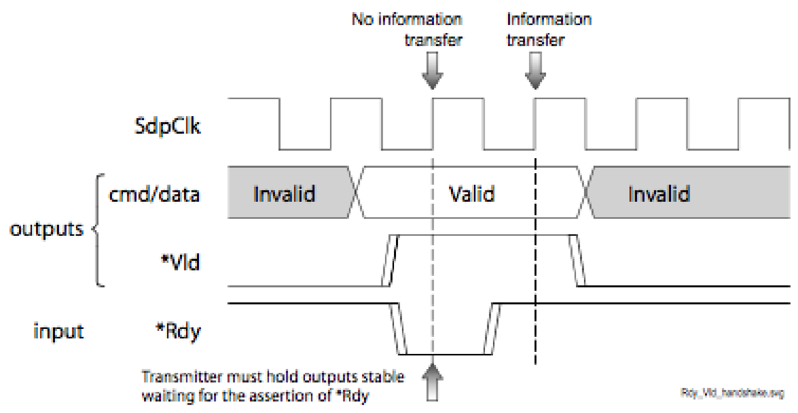

Figure 4 Port Interface Valid and Ready Signaling

Whenever an originator or a completer device or the port to which it attaches has its *ClkAck signals asserted, it must be ready to receive information on all its inbound channels for which it has also asserted *Rdy. Information is presented on the source side of the interface and transferred into latches on the receiver side.

The source asserts its *Vld signal to indicate that command and data signals are stable at the interface and the receiver asserts its *Rdy signal to indicate that it is ready to receive the command and data information. If the port implements the Low-Power Signaling feature, the source must also assert the corresponding *ClkEn_m1 signal (at least) one cycle before any cycle that it asserts the *Vld. The transfer occurs on the rising edge of the clock in which both *Vld and *Rdy are high. *Rdy may be driven by the actual originator or completer, or it may be asserted by pervasive logic outside of the actual originator or completer block to perform clock synchronization in which case any *Rdy signals from the port may be ignored and the port must always be ready any time the port is connected. Once the source asserts *Vld signal, it is not expected to de-assert it before observing *Rdy unless the port resets (deassertion of SdpReset_N).

When the source has not asserted its *Vld signal, all of the command and data information on that port is not valid and must be ignored. For example, there is no requirement that ReqCmd has a valid encoding or that the parity signals are consistent with the data.

All signals on the SDP interface except for the Port Control signals (*ClkReq, *ClkAck, *ClkCtl) and the deassertion of SdpReset_N are synchronous to a common clock (SdpClk) supplied by system pervasive logic. Source logic switching and signal propagation times are expected to be such that there is sufficient time margin to the next rising clock edge to allow simple combinatorial logic on the receiving side of the interface. This logic may perform functions such as de-asserting *Rdy or advancing a queue pointer prior to the data/control information being latched. In some high-frequency implementations, the *Rdy signals may also be provided in advance by an integer number of clocks using a port parameter called N_{rdy_shift}. Each *Rdy signal would be replaced or duplicated with a signal *Rdy_m<N_{rdy_shift}> to indicate that this signal is pipelined by that number of clocks.

Note that the logic block on the receiver side of the interface can hold off the transfer by either de-asserting or delaying the assertion of its *Rdy signal. This provides a degree of link-level pacing. However, the receiver side logic cannot make the assertion of its *Rdy signal contingent upon the assertion of the *Vld signal by the logic on the source side or upon its ability to act on or buffer the incoming data or command information. Such pacing is supplied by the higher level credit-based protocol.

Figure 5 Port Interface Reset Requirements provides information about the initialization of the interface.

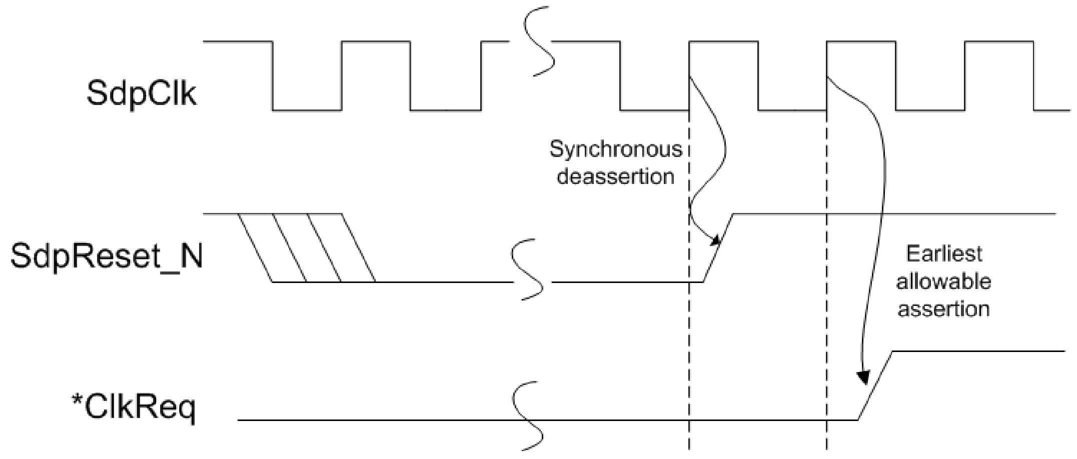

Figure 5 Port Interface Reset Requirements

The port (or system) reset signal SdpReset_N is a negative-active signal that is used to initialize the interface logic. It is expected that no state information is retained across a reset event. Any established credits are reset and any outstanding transactions are aborted and will not receive a response.

While the assertion (negative-going edge) of SdpReset_N is asynchronous to SdpClk, the de-assertion (rising edge) of SdpReset_N must be synchronous to the clock.

When SdpReset_N is active, the *Vld and *Rdy and port control *ClkReq and *ClkAck signals must be in their inactive (low) state. The state of other signals are "don't cares." No data transfer across the interface occurs. Immediately after reset, the *ClkReq signals must remain inactive until after the first rising edge of SdpClk following the de-assertion of SdpReset_N. At this time, the logic block that sources the port control *ClkReq signal may begin port activation. *ClkAck normally only asserts in response to observing the complementary *ClkReq signal assert. The receiver logic block can indicate its readiness to receive information using the *Rdy signals. *Rdy signals may assert prior to *ClkAck even though a channel cannot receive information until both *ClkAck and *Rdy are asserted.

Implementation Note: When a port involves a clock crossing boundary, the receiving logic may observe *ClkReq assert prior to the first clock or even prior to it observing reset deassert even though the source logic may not assert prior to its first clock after coming out of reset. This may occur, for example, due to a frequency delta between the two sides of the port when *ClkReq is an asynchronous signal., or due to actual differences in the clocking or reset structure outside the scope of this specification. Receivers should not depend on actually observing an edge on the incoming *ClkReq after a reset.

2.5 Port Control

SDP provides signals that allow the port to be brought up to an active state or transitioned to a disconnected state in an ordered sequence that eliminates race conditions that might corrupt the transfer of information across the interface. The handshake protocol can be orchestrated by state machines on both sides of the interface or coordinated directly by SOC-level logic and/or management firmware.

In the following diagrams and discussion, "originator" refers to the originator device at an originator port or the agent within the fabric acting as a proxy for the originator at a completer port. "Completer" refers to the completer device at a completer port or the agent within the fabric acting as a proxy for the completer at an originator port.

Port control uses the four signals: OrigClkReq, OrigClkAck, CompClkReq, and CompClkAck. Enhanced port control adds an additional two signals: OrigClkCtl and CompClkCtl.

An originator uses OrigClkReq to request the connection or disconnection of its outbound channels. The completer uses CompClkAck to acknowledge the originator's request. The originator to completer direction is active and information including flow-control credits may pass from originator to the completer (using *Vld and *Rdy handshaking protocols) only when OrigClkReq and CompClkAck are both asserted. Information may flow from the originator to the completer even if the completer to the originator direction has not (yet) been activated.

A completer uses CompClkReq to request the connection or disconnection of its outbound channels. The originator uses OrigClkAck to acknowledge the completer's request. The completer to originator direction is active and information including flow-control credits may pass from completer to the originator (using *Vld and *Rdy handshaking protocols) only when CompClkReq and OrigClkAck are both asserted. Information may flow from the completer to the originator even if the originator to completer direction has not (yet) been activated.

The port is active when both directions are active (all four signals, *ClkReq and *ClkAck are asserted).

Hereafter, this section refers to the originator or the completer as an "agent" and refers to *ClkReq and *ClkAck to generalize the two sides of the port.

The conditions that prompt an agent to initiate an activation, a reconnect or a disconnect are outside the scope of this specification. In some cases, this may involve outside firmware action and may be subject to configuration information as to whether or not a port connection or disconnection occurs.

2.5.1 Activating or Reconnecting the Port

Port ***activation*** is the process in which each side may request and establish a connection after a port reset (SdpReset_N). Port ***reconnection*** is the same process (hereafter referred to as the connect process), but one in which a side requests a connection after a disconnect (one-sided or full-disconnect). During a disconnection, an independent side may have been reset, clock-gated, or power-gated but this is separate from a port reset.

Any initialization or restoration of state within both the originator and completer must be performed prior to the connect sequence. Both agents must be powered and actively clocked for the sequence to proceed. Either non-initiating agent can hold off the completion of the connection by delaying its *ClkAck response or deferring its assertion of *ClkReq. Prior to asserting *ClkAck the acknowledging agent must be prepared to latch valid information presented by the requesting agent on any of the requesting agent's outbound channels. Once the requesting device asserts *ClkReq, it must not de-assert it until after it receives the acknowledgment (the assertion of *ClkAck).

Since the flow of transactions requires that credits are first established by a flow of information in the opposite direction, each side is required to connect the port. The completer may wait for the originator to establish the connection first, or both sides may independently activate/reconnect. It is not legal for the originator to wait for a completer to establish the connection first.

Figure 6 Originator-Initiated Activation shows the sequence of assertions in the case where the activation handshake protocol is initiated by the originator from a fully-disconnected (or reset) state. Figure 7 Completer-Initiated Activation shows the assertion sequence in the case where the activation handshake protocol is initiated by the completer from a fully-disconnected (or reset) state.

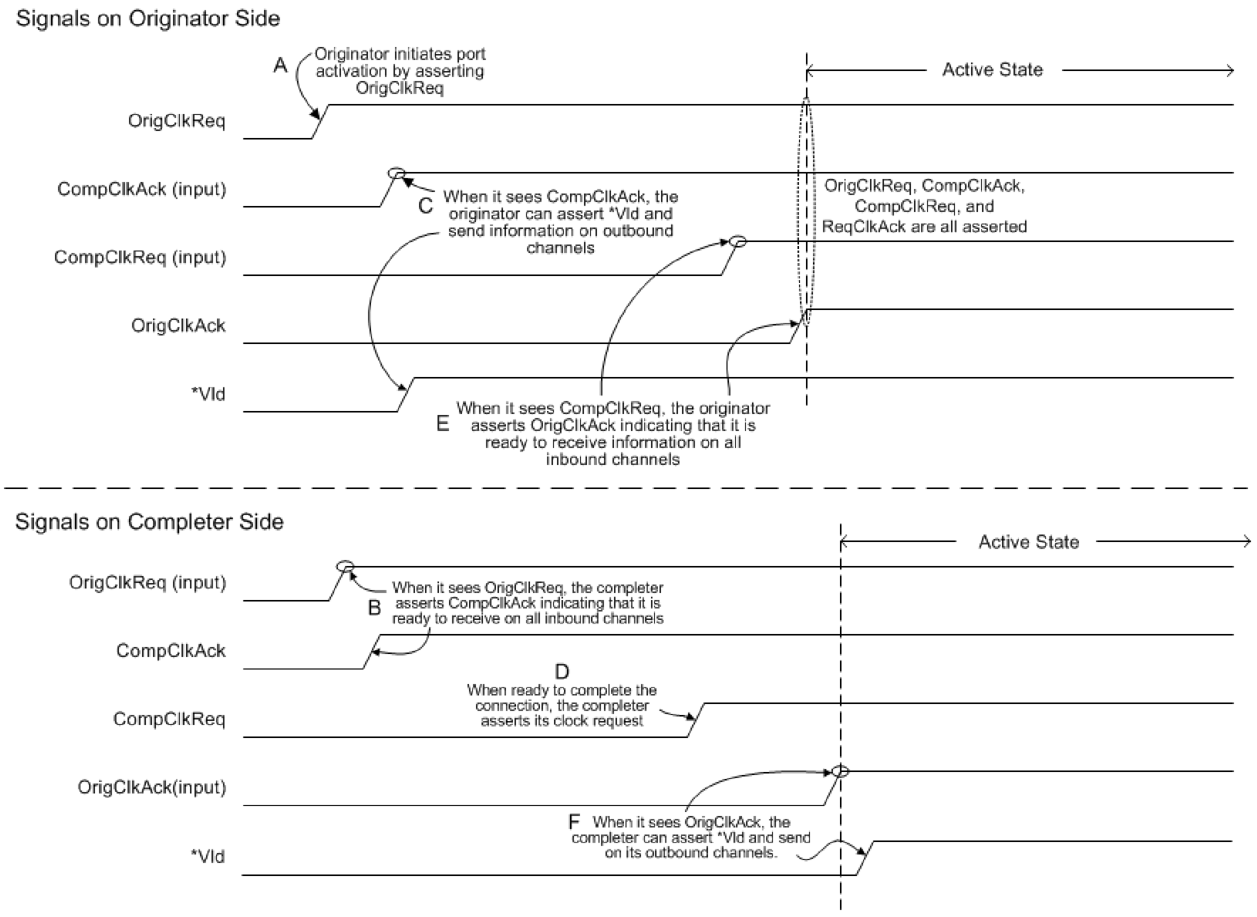
Signals on Originator Side
A Originator initiates port activation by asserting OrigClkReq
Active State
OrigClkReq
CompClkAck (input)
C When it sees CompClkAck, the originator can assert *Vld and send information on outbound channels
OrigClkReq, CompClkAck, CompClkReq, and ReqClkAck are all asserted
CompClkReq (input)
OrigClkAck
*Vld
E When it sees CompClkReq, the originator asserts OrigClkAck indicating that it is ready to receive information on all inbound channels
Signals on Completer Side
Active State
OrigClkReq (input)
B When it sees OrigClkReq, the completer asserts CompClkAck indicating that it is ready to receive on all inbound channels
CompClkAck
D When ready to complete the connection, the completer asserts its clock request
CompClkReq
OrigClkAck(input)
F When it sees OrigClkAck, the completer can assert *Vld and send on its outbound channels.
*Vld

Figure 6 Originator-Initiated Activation

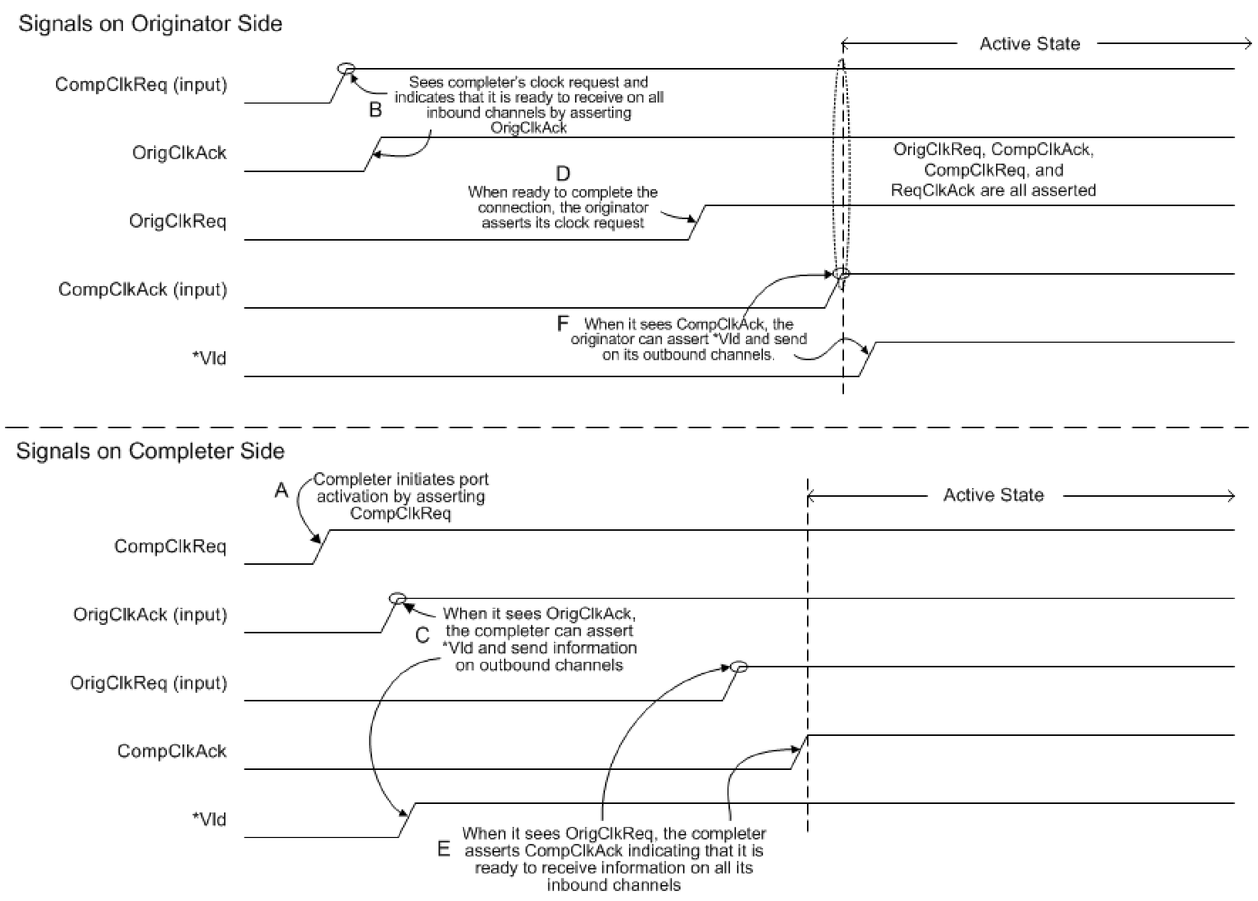
Signals on Originator Side
CompClkReq (input)
OrigClkAck
OrigClkReq
CompClkAck (input)
*Vld
Active State
B Sees completer's clock request and indicates that it is ready to receive on all inbound channels by asserting OrigClkAck
D When ready to complete the connection, the originator asserts its clock request
OrigClkReq, CompClkAck, CompClkReq, and ReqClkAck are all asserted
F When it sees CompClkAck, the originator can assert *Vld and send on its outbound channels.
Signals on Completer Side
CompClkReq
OrigClkAck (input)
OrigClkReq (input)
CompClkAck
*Vld
Active State
A Completer initiates port activation by asserting CompClkReq
C When it sees OrigClkAck, the completer can assert *Vld and send information on outbound channels
E When it sees OrigClkReq, the completer asserts CompClkAck indicating that it is ready to receive information on all its inbound channels

Figure 7 Completer-Initiated Activation

2.5.2 Disconnecting the Port – One-sided Disconnect

A ***one-sided disconnect*** means that the initiating agent has requested a disconnect (by de-asserting its *ClkReq) and the non-initiating agent has acknowledged the request (by de-asserting its *ClkAck), but has not reciprocated by de-asserting its *ClkReq.

In a one-sided disconnect state, the initiating agent cannot assert *Vld or present any information on any of its outbound channels (including the credit release sub-channels of its inbound channels). This requirement starts at the time that *ClkReq is de-asserted. The agent must remain ready to receive information on its inbound channels, including the credit release sub-channels of its outbound channels.

The initiating agent cannot gate power or clocks for any of the logic required to receive information on its inbound channels, including the logic required to track the number of credits received from the receiver on the credit release sub-channels of its outbound channels. The non-initiating agent, which continues to hold its *ClkReq active, may continue to send information on its outbound channels, including the credit release sub-channels of its inbound channels, but can gate clocks to logic required to receive information on its inbound channels including the credit release sub-channels of its outbound channels.

The non-initiating agent can hold off its acknowledgment of the initiating agent's disconnect request by delaying its *ClkAck negation response, but must not hold off indefinitely. This delay may be a time to drain clock-synchronization FIFOs or pipelining latches, but it is illegal to make this acknowledge dependent on receiving credits or completions to outstanding transactions or probes. Such a dependency would create a deadlock as the initiating agent may not send this information until after re-asserting *ClkReq and it is illegal for an initiating agent to re-assert *ClkReq before observing the non-initiating respond by de-asserting *ClkAck.

Depending on the implementation of the physical connection, the initiating or acknowledging agent may have a responsibility to wait a specified period of time for the information last sent on the port to make it through any clock-synchronization logic that may be gated by the disconnect. The implementation must establish rules, such as a minimum number of clocks, to ensure that information is not stranded in any synchronization logic between the two sides. These rules are outside the scope of this document.

Implementation Note: For SOC-15 ports with a Voltage Domain Crossing Interface (VDCI), delay the de-assertion of *ClkReq or *ClkAck until the VDCI asserts *_Empty<ch> signals on all channels.

The bus may or may not be quiesced prior to a one-sided disconnect, and the port disconnection can proceed while responses, acknowledgements, and/or credit returns are outstanding. During a one-sided disconnect, the state of the port is not altered. All outstanding transactions and credits are maintained as if the port is connected. An agent that has completed a one-sided disconnect reserves the right to issue a reconnect at any time after the disconnect request has been acknowledged by re-asserting its *ClkReq. The initiating agent is required to reconnect when it has information to send, including flow-control credits. It may put a time limit (hysteresis) before reconnection, but it may not put a threshold on the number of credits that must be sent before reconnection.

In some cases, both sides of the port may go through one-sided disconnection independently or even simultaneously. The end result is that the port is in a full disconnect state with the exception that the port may not be fully quiesced (there may be outstanding transactions or credits).

2.5.3 Disconnecting the Port – Full Disconnect

A ***full disconnect*** means that both sides of the interface have requested a disconnect by de-asserting their respective *ClkReq signals and the requests have been acknowledged by both sides. In the fully disconnected state, OrigClkReq, CompClkReq, OrigClkAck, and CompClkAck are all inactive. All *Vld signals are inactive and all channel information fields are ignored. In this state, one or both of the agents can be power or clock gated.

Before the port can be fully disconnected for power-management reasons, all request, probe, response, acknowledgements, and flow-control traffic must be quiesced. New requests must be halted and all responses returned to the requester. The issuing or return of flow control credits must be quiesced or suspended. Once all information flow has been halted, the handshake to execute the disconnect can be initiated from either side. The specification is written to assume that the disconnect is initiated from the originator side, which is more natural in most cases as it is the originator that has the state for what transactions may be outstanding and the likelihood of seeing more transactions in the near future.

Each side of a port that is going through full-disconnect has all the same requirements that each side has if it initiates two one-sided disconnects as far as the stopping of information flow after *ClkReq is dropped, acknowledging the *ClkReq, waiting for any synchronization logic, and so-forth.

The non-initiating agent is not necessarily required to reciprocate the disconnect request by de-asserting its *ClkReq. The involvement of SOC-level hardware and/or firmware (outside the scope of this document) may allow the initiating agent to know that the other side will reciprocate. The port is fully disconnected if the non-initiating agent does reciprocate and the initiating agent acknowledges the disconnect.

As with one-sided disconnection, the state of the port is expected to be maintained during the full-disconnect. The requirement to quiesce all request, probe, response, and flow-control traffic is meant to simplify the maintenance of the state. If one or both sides of the port is affected by power-gating during the time the port is disconnected, the agents are required to re-initialize or restore the state of the port prior to reconnection. The method that is used to restore this state after being power-gated is outside the scope of this specification. Agents that implement enhanced port control feature have an option to not re-initialize the credits after a full disconnect and ask for credits to be resent. If credits are not specifically requested to be resent, or if the enhanced port control feature is not implemented, the agent must restore the number and type of credits that the other side gave the agent prior to the disconnect.

Figure 8 Originator-Initiated Full Disconnect shows the disconnect sequence in the case where the handshake protocol is initiated from the originator side of the interface. Figure 9 Completer-initiated Full Disconnect shows the disconnect sequence in the case where the disconnect handshake protocol is initiated from the completer side of the interface

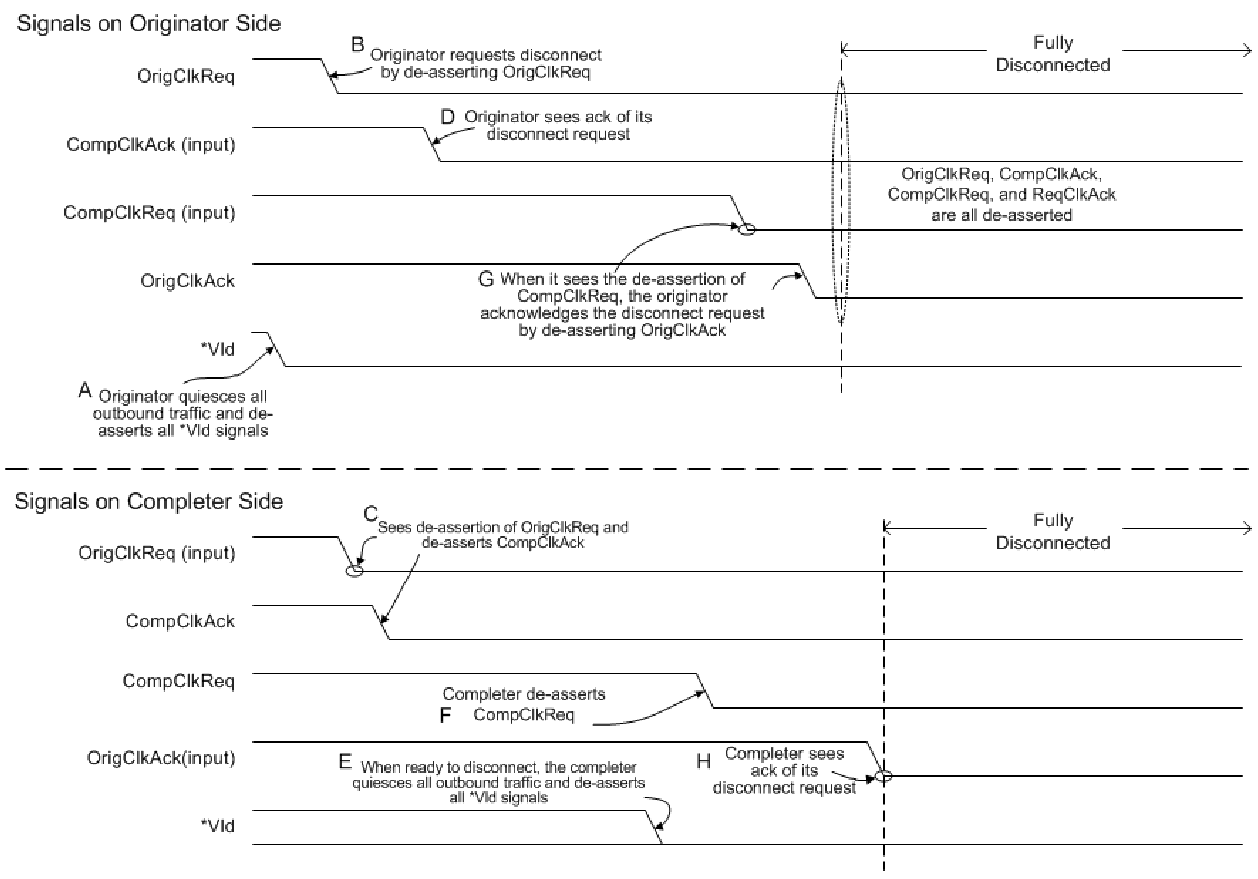
Signals on Originator Side
OrigClkReq
CompClkAck (input)
CompClkReq (input)
OrigClkAck
*Vld
B Originator requests disconnect by de-asserting OrigClkReq
D Originator sees ack of its disconnect request
G When it sees the de-assertion of CompClkReq, the originator acknowledges the disconnect request by de-asserting OrigClkAck
A Originator quiesces all outbound traffic and de-asserts all *Vld signals
Fully Disconnected
OrigClkReq, CompClkAck, CompClkReq, and ReqClkAck are all de-asserted
Signals on Completer Side
OrigClkReq (input)
CompClkAck
CompClkReq
OrigClkAck(input)
*Vld
C Sees de-assertion of OrigClkReq and de-asserts CompClkAck
Completer de-asserts F CompClkReq
E When ready to disconnect, the completer quiesces all outbound traffic and de-asserts all *Vld signals
H Completer sees ack of its disconnect request
Fully Disconnected

Figure 8 Originator-Initiated Full Disconnect

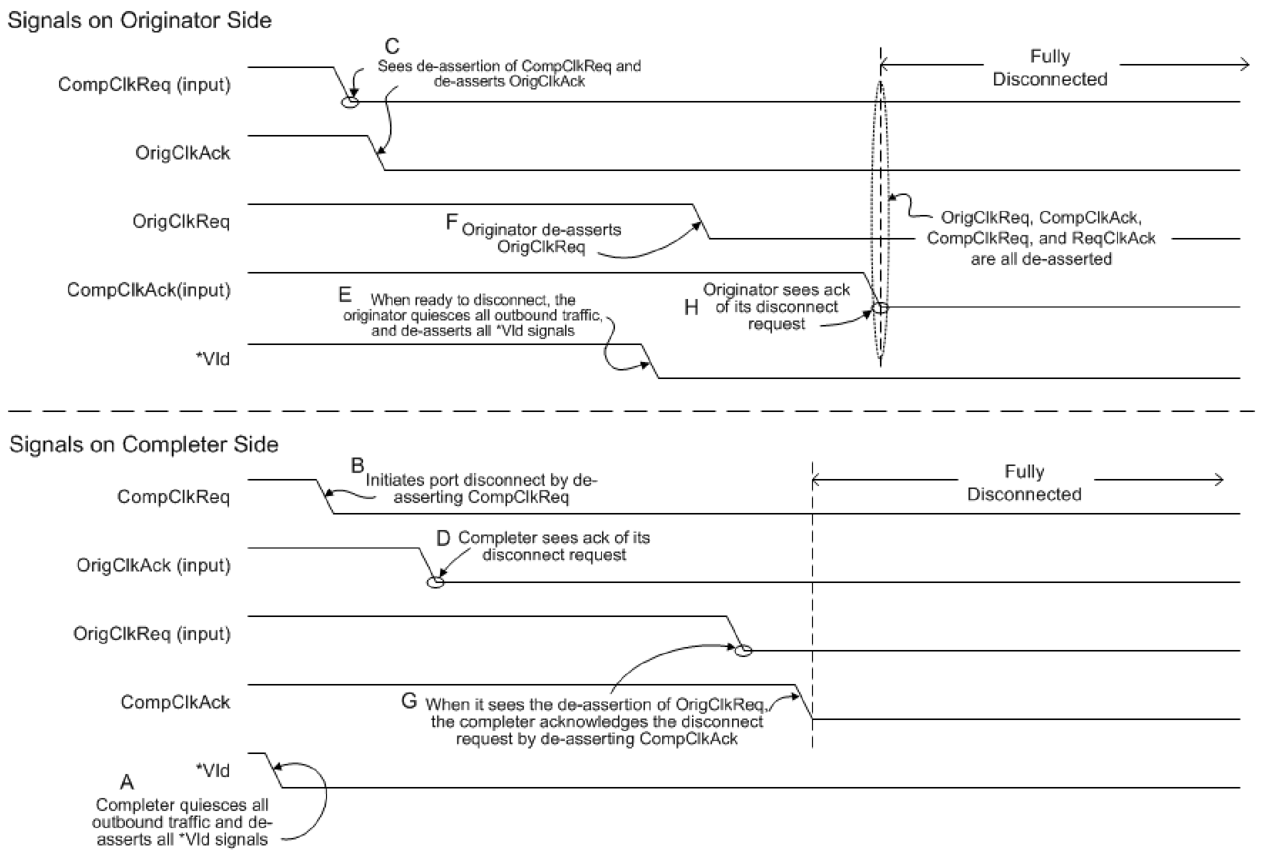
Signals on Originator Side
CompClkReq (input)
OrigClkAck
OrigClkReq
CompClkAck(input)
*Vld
C Sees de-assertion of CompClkReq and de-asserts OrigClkAck
Fully Disconnected
F Originator de-asserts OrigClkReq
OrigClkReq, CompClkAck, CompClkReq, and ReqClkAck are all de-asserted
E When ready to disconnect, the originator quiesces all outbound traffic, and de-asserts all *Vld signals
H Originator sees ack of its disconnect request
Signals on Completer Side
CompClkReq
OrigClkAck (input)
OrigClkReq (input)
CompClkAck
*Vld
B Initiates port disconnect by de-asserting CompClkReq
Fully Disconnected
D Completer sees ack of its disconnect request
G When it sees the de-assertion of OrigClkReq, the completer acknowledges the disconnect request by de-asserting CompClkAck
A Completer quiesces all outbound traffic and de-asserts all *Vld signals

Figure 9 Completer-initiated Full Disconnect

2.5.4 Enhanced Port Control Feature

The ***enhanced port control*** feature adds two signals, OrigClkCtl and CompClkCtl, which are used in both connection and disconnection of the port.

The enhanced port control feature also provides a means by which the initiating agent of a disconnect can request either a full or a one-sided disconnect. The full disconnect request is a hint that indicates that the initiating agent is prepared to enter a power gated state and will not request a reconnect for some period of time.

Connection with the Enhanced Port Control Feature

The enhanced port control feature provides a means by which the originator and/or the completer can explicitly request that the other agent issue (or re-issue) flow control credits when they reconnect after a full disconnect.

The agent can, alternately, explicitly request that flow control credits ***not*** be reissued (in the case where the agent has not lost track of the credits that were released prior to the disconnect).

In order to function properly with the enhanced port control feature, the first agent to reconnect from a full disconnect may be required to wait for the other side to establish a connection before it can issue credits.

If the enhanced port control feature is not supported, any required initialization of flow control credits during connection must be provided by a means outside the scope of this specification.

Figure 10 Enhanced Activation Protocol shows the enhanced port activation sequence in the case where the activation is initiated from the originator.

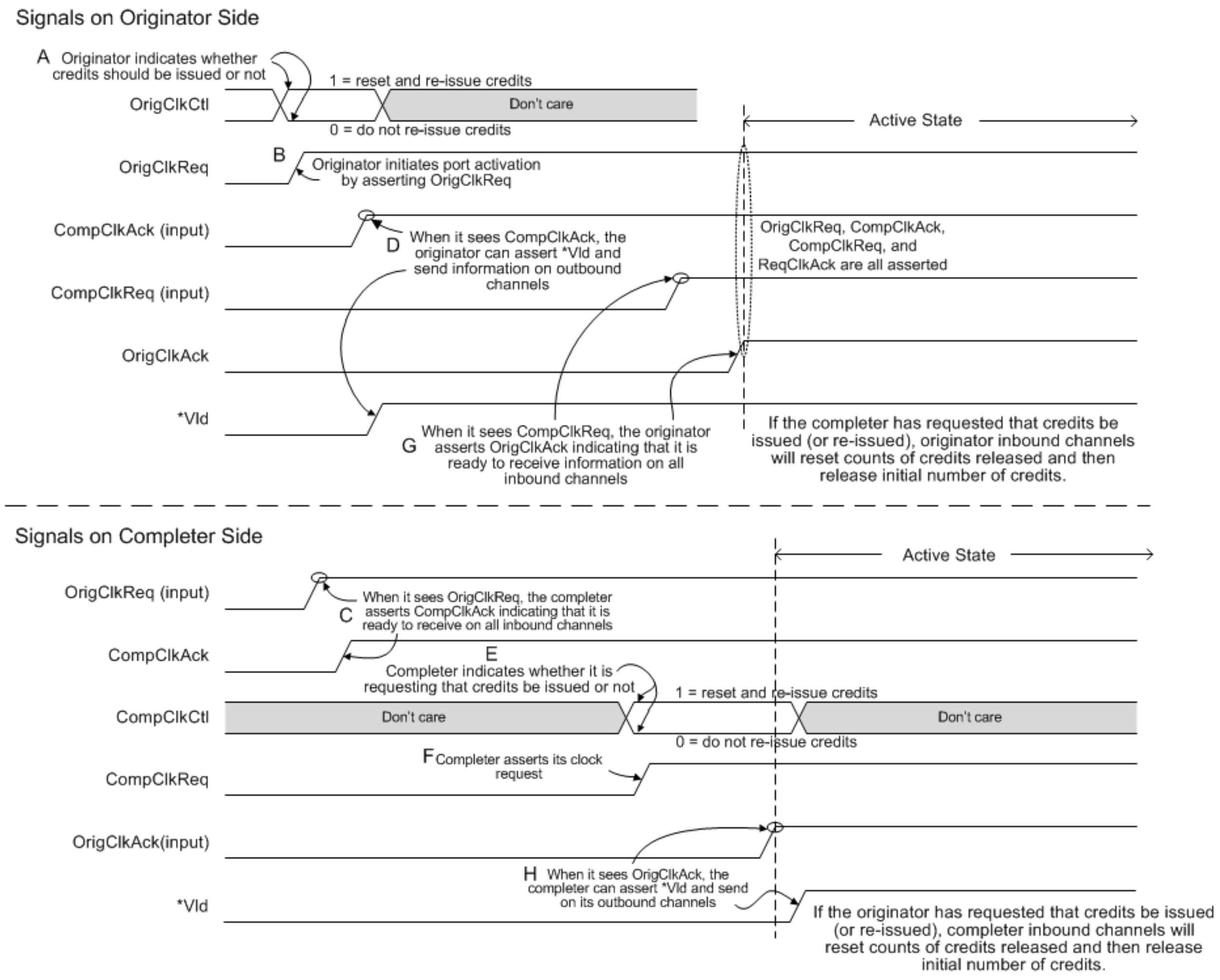
Signals on Originator Side
A Originator indicates whether credits should be issued or not
1 = reset and re-issue credits
Don't care
0 = do not re-issue credits
OrigClkCtl
OrigClkReq
B Originator initiates port activation by asserting OrigClkReq
Active State
CompClkAck (input)
D When it sees CompClkAck, the originator can assert *Vld and send information on outbound channels
OrigClkReq, CompClkAck, CompClkReq, and ReqClkAck are all asserted
CompClkReq (input)
OrigClkAck
*Vld
G When it sees CompClkReq, the originator asserts OrigClkAck indicating that it is ready to receive information on all inbound channels
If the completer has requested that credits be issued (or re-issued), originator inbound channels will reset counts of credits released and then release initial number of credits.
Signals on Completer Side
Active State
OrigClkReq (input)
C When it sees OrigClkReq, the completer asserts CompClkAck indicating that it is ready to receive on all inbound channels
CompClkAck
E Completer indicates whether it is requesting that credits be issued or not
1 = reset and re-issue credits
CompClkCtl
Don't care
Don't care
0 = do not re-issue credits
F Completer asserts its clock request
CompClkReq
OrigClkAck(input)
H When it sees OrigClkAck, the completer can assert *Vld and send on its outbound channels
*Vld
If the originator has requested that credits be issued (or re-issued), completer inbound channels will reset counts of credits released and then release initial number of credits.

Figure 10 Enhanced Activation Protocol

The originator uses the OrigClkCtl signal to request that flow control credits for its outbound channels be issued (or re-issued) by the completer or to request that flow control credits for its outbound channels not be issued (or re-issued) by the completer. It signals its request that credits be issued by asserting OrigClkCtl prior to asserting OrigClkReq and it signals its request that credits not be issued by de-asserting OrigClkCtl prior to asserting OrigClkReq.

The completer uses the CompClkCtl signal to request that flow control credits for its outbound channels be issued (or re-issued) by the originator or to request that flow control credits for its outbound channels not be issued (or re-issued) by the originator. It signals its request that credits be issued by asserting CompClkCtl prior to asserting CompClkReq and it signals its request that credits not be issued by de-asserting CompClkCtl prior to asserting CompClkReq.

Note that the requesting agent must hold the state of its *ClkCtl signal until it sees the acknowledgment of its connection request. If the requesting agent has not lost track of the credits it has received and does not want the other side to re-issue credits, it must ensure its *ClkCtl signal is inactive when asserting *ClkReq.

An agent may not request that credits be re-issued after a one-sided disconnect and must de-assert the *ClkCtl.

On the very first connection (i.e. after SdpReset_N is deasserted), it is required that all agents that implement the enhanced port control assert *ClkCtl to signal credit release but it is also allowable that the receiving agent ignore the value and assume that credits are always released after a port reset.

In the enhanced activation protocol after a full disconnect, an agent may not assume the other side will request it to reset and re-issue credits just because the first agent is requesting the other side to reset and re-issue credits. In the above figure, the originator must wait to sample CompClkCtl at the assertion of CompClkReq prior to issuing credits to the completer.

Disconnect

The enhanced port control feature allows both agents to explicitly signal their intentions relative to the currently requested disconnect. When an agent requests a full disconnect, it is providing a hint to the other agent that it has nothing to send and that it expects this condition to persist long enough that a full disconnect would be advantageous to save power. To complete the disconnect, the non-initiating agent must also request a disconnect.

If the non-initiating agent also has nothing to send and concurs that a full disconnect would be advantageous, it would reciprocate, requesting a full disconnect. If it responds by requesting a one-sided disconnect, this hint may indicate that the non-initiating agent has information that would contraindicate a full disconnect. However, since it has responded with a disconnect request, when this is acknowledged by the initiating agent, the port is in a fully disconnected state.

Figure 11 Enhanced Originator-Initiated Full Disconnect and Figure 12 Enhanced Completer-Initiated Full Disconnect show, respectively, a full disconnect initiated by the originator and a full disconnect initiated by the completer. When the *ClkCtl signal is asserted prior to de-asserting *ClkReq, the requesting agent is requesting a full disconnect. When the *ClkCtl signal is de-asserted prior to de-asserting *ClkReq, the requesting agent is requesting a one-sided disconnect. Note that the requesting agent must hold the *ClkCtl signal stable until it sees the acknowledgment of its disconnect request.

The next timing diagram shows the enhanced disconnect protocol in the case where the full disconnect request is initiated from the originator side of the interface.

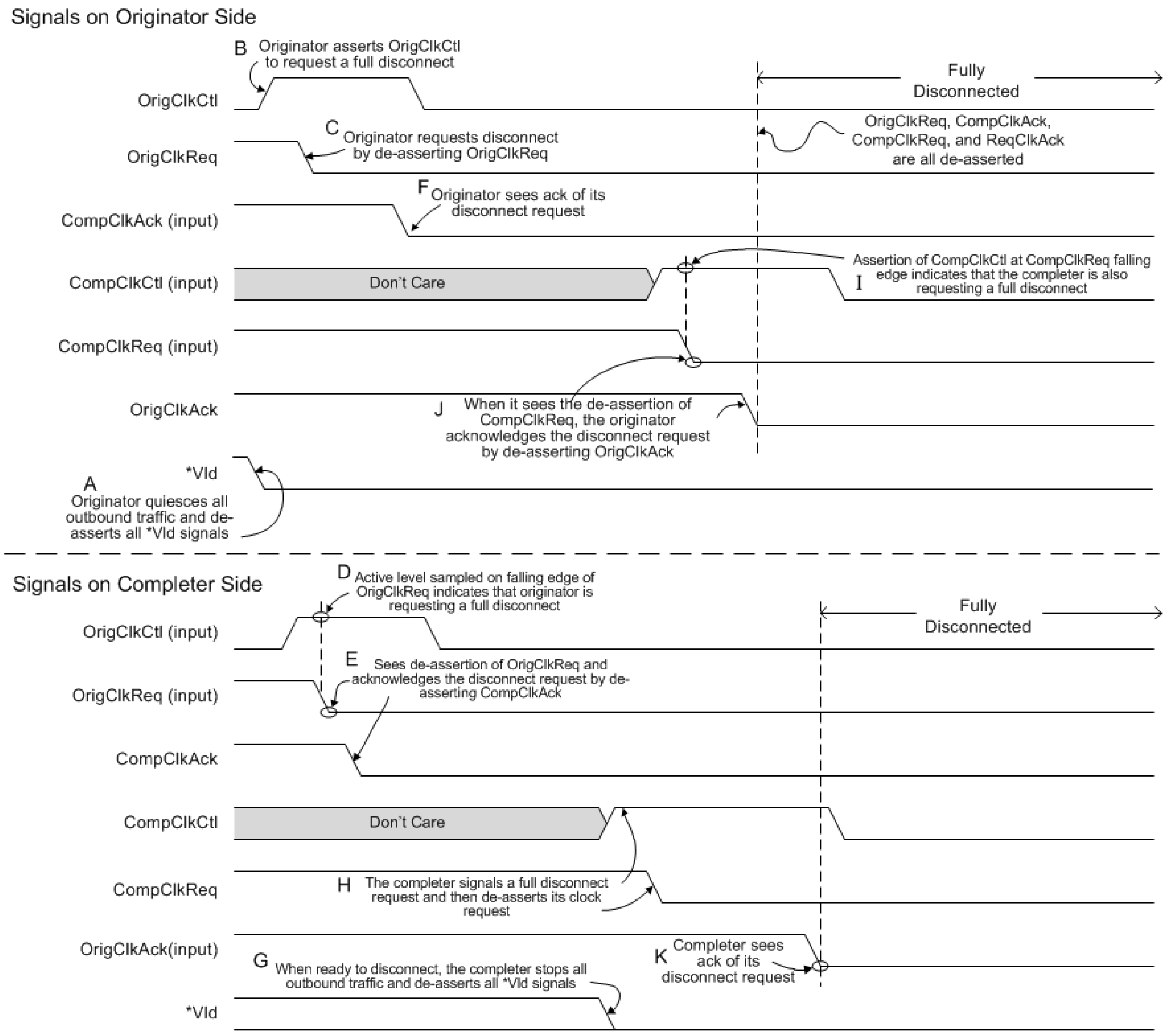
Signals on Originator Side
B Originator asserts OrigClkCtl to request a full disconnect
Fully Disconnected
OrigClkCtl
C Originator requests disconnect by de-asserting OrigClkReq
OrigClkReq, CompClkAck, CompClkReq, and ReqClkAck are all de-asserted
OrigClkReq
F Originator sees ack of its disconnect request
CompClkAck (input)
CompClkCtl (input)
Don't Care
Assertion of CompClkCtl at CompClkReq falling edge indicates that the completer is also requesting a full disconnect
I
CompClkReq (input)
OrigClkAck
J When it sees the de-assertion of CompClkReq, the originator acknowledges the disconnect request by de-asserting OrigClkAck
*Vld
A Originator quiesces all outbound traffic and de-asserts all *Vld signals
Signals on Completer Side
D Active level sampled on falling edge of OrigClkReq indicates that originator is requesting a full disconnect
Fully Disconnected
OrigClkCtl (input)
E Sees de-assertion of OrigClkReq and acknowledges the disconnect request by de-asserting CompClkAck
OrigClkReq (input)
CompClkAck
CompClkCtl
Don't Care
CompClkReq
H The completer signals a full disconnect request and then de-asserts its clock request
OrigClkAck(input)
G When ready to disconnect, the completer stops all outbound traffic and de-asserts all *Vld signals
K Completer sees ack of its disconnect request
*Vld

Figure 11 Enhanced Originator-Initiated Full Disconnect

The non-initiating agent must acknowledge the disconnect request, but can defer its own disconnect request. Normally, if the initiating agent requests a full disconnect, the non-initiating agent, when ready to complete the disconnect, would also request a full disconnect, but it is not required to do so. A means outside the scope of this specification may be provided to cancel the full disconnect request and allow the initiating agent to request a reconnect.

The next timing diagram shows the enhanced disconnect protocol in the case where the full disconnect request is initiated from the completer side of the interface.

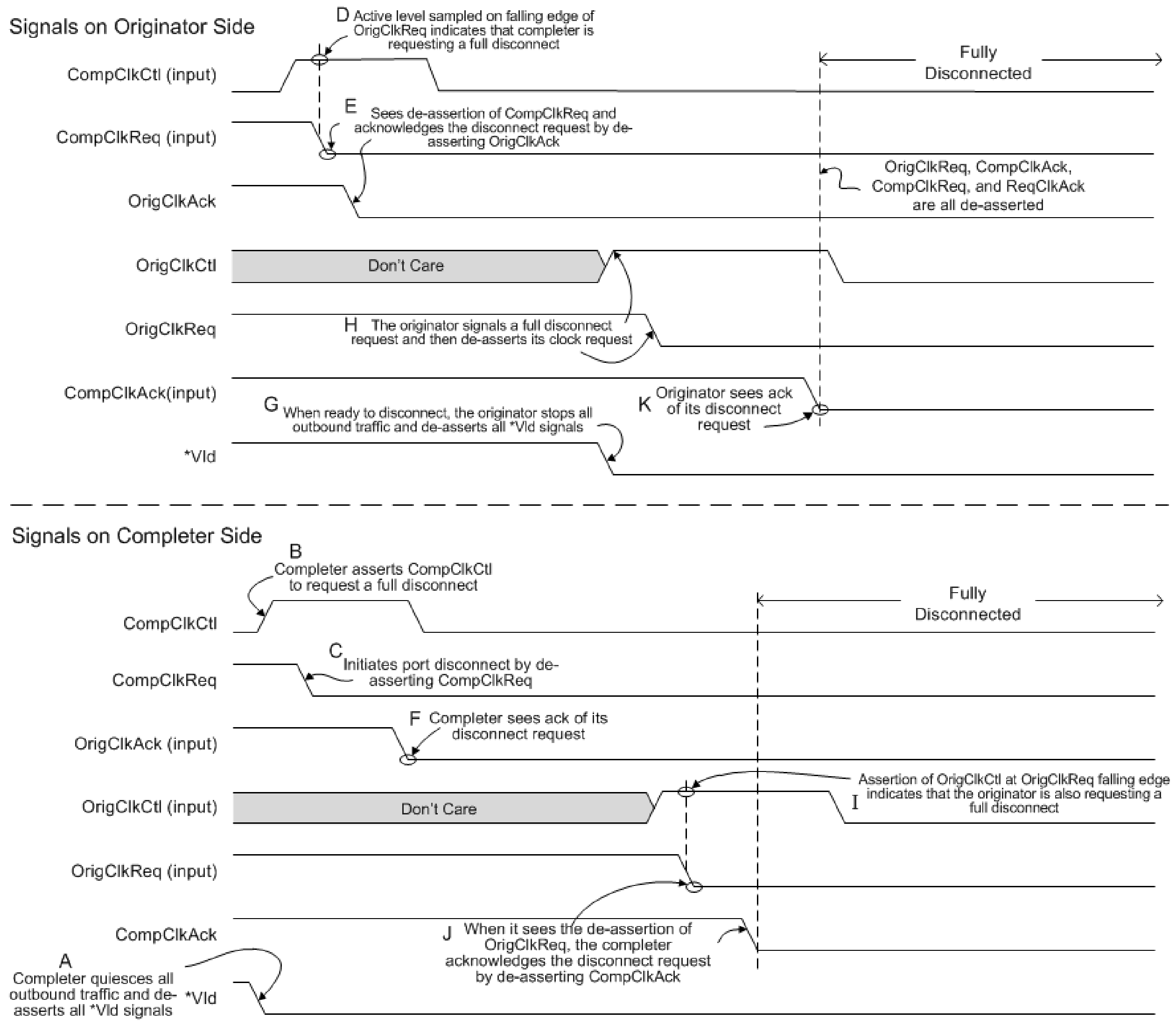
Signals on Originator Side
CompClkCtl (input)
CompClkReq (input)
OrigClkAck
OrigClkCtl
OrigClkReq
CompClkAck(input)
*Vld
D Active level sampled on falling edge of OrigClkReq indicates that completer is requesting a full disconnect
E Sees de-assertion of CompClkReq and acknowledges the disconnect request by de-asserting OrigClkAck
Fully Disconnected
OrigClkReq, CompClkAck, CompClkReq, and ReqClkAck are all de-asserted
Don't Care
H The originator signals a full disconnect request and then de-asserts its clock request
G When ready to disconnect, the originator stops all outbound traffic and de-asserts all *Vld signals
K Originator sees ack of its disconnect request
Signals on Completer Side
B Completer asserts CompClkCtl to request a full disconnect
CompClkCtl
CompClkReq
OrigClkAck (input)
OrigClkCtl (input)
OrigClkReq (input)
CompClkAck
*Vld
C Initiates port disconnect by de-asserting CompClkReq
F Completer sees ack of its disconnect request
Don't Care
Fully Disconnected
Assertion of OrigClkCtl at OrigClkReq falling edge indicates that the originator is also requesting a full disconnect
I
J When it sees the de-assertion of OrigClkReq, the completer acknowledges the disconnect request by de-asserting CompClkAck
A Completer quiesces all outbound traffic and de-asserts all *Vld signals

Figure 12 Enhanced Completer-Initiated Full Disconnect

In the enhanced port control protocol, when an initiating agent requests a one-sided disconnect, it must reconnect before it can signal a full disconnect request. If a means is provided to quiesce all information flow in both directions mutual one-sided disconnects are sufficient to transition to a fully disconnected state in which one or both agents can clock or power gate.

One-Sided Disconnect / Reconnect

Figure 13 Originator-Initiated One-Sided Disconnect shows the disconnect / reconnect protocol where the one-sided disconnect is initiated from the originator side of the interface.

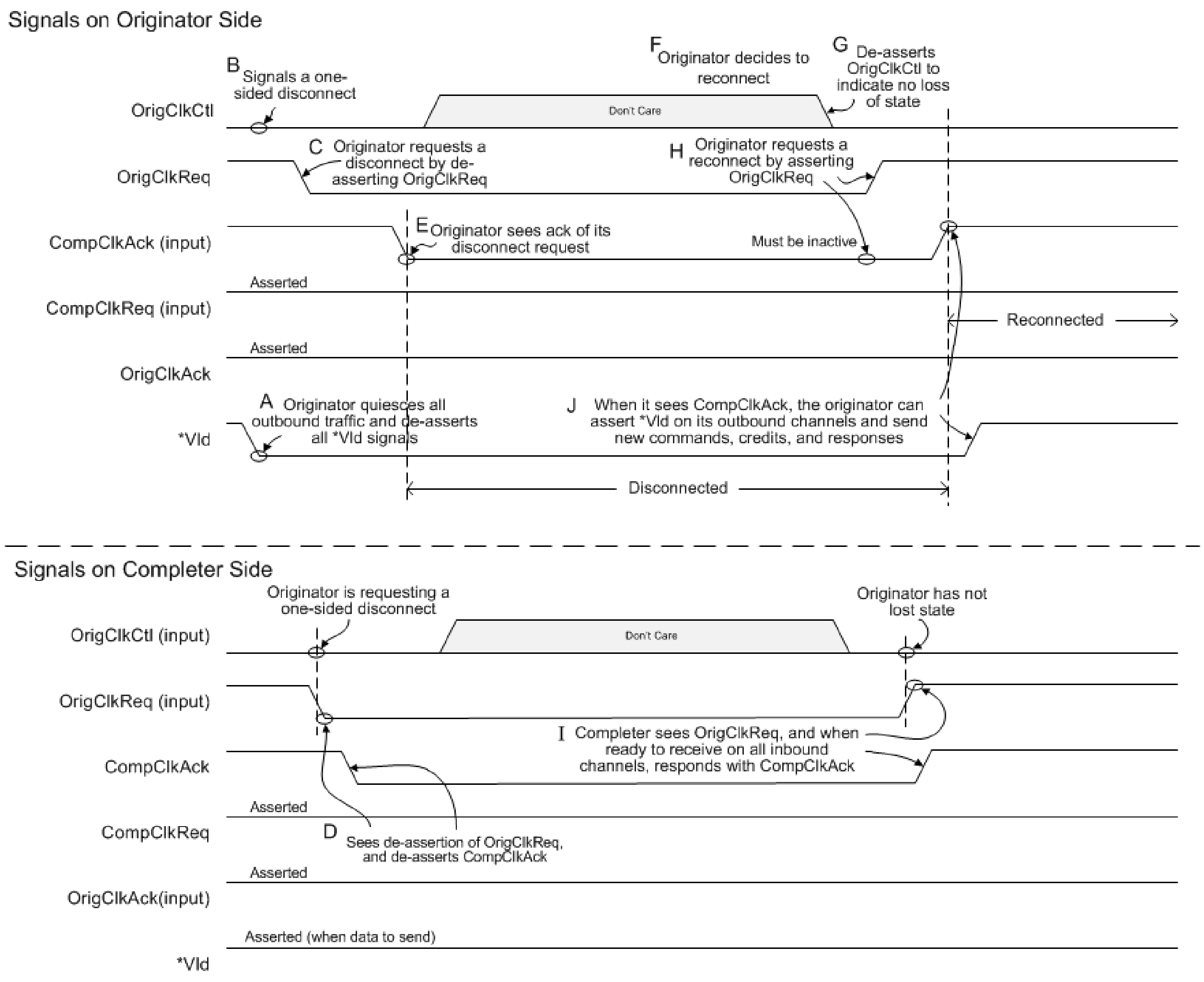
Signals on Originator Side
B Signals a one-sided disconnect
F Originator decides to reconnect
G De-asserts OrigClkCtl to indicate no loss of state
OrigClkCtl
Don't Care
C Originator requests a disconnect by de-asserting OrigClkReq
H Originator requests a reconnect by asserting OrigClkReq
OrigClkReq
CompClkAck (input)
E Originator sees ack of its disconnect request
Must be inactive
Asserted
CompClkReq (input)
Reconnected
Asserted
OrigClkAck
A Originator quiesces all outbound traffic and de-asserts all *Vld signals
J When it sees CompClkAck, the originator can assert *Vld on its outbound channels and send new commands, credits, and responses
*Vld
Disconnected
Signals on Completer Side
Originator is requesting a one-sided disconnect
Originator has not lost state
OrigClkCtl (input)
Don't Care
OrigClkReq (input)
I Completer sees OrigClkReq, and when ready to receive on all inbound channels, responds with CompClkAck
CompClkAck
Asserted
CompClkReq
D Sees de-assertion of OrigClkReq, and de-asserts CompClkAck
Asserted
OrigClkAck(input)
Asserted (when data to send)
*Vld

Figure 13 Originator-Initiated One-Sided Disconnect

Once the requesting device de-asserts *ClkReq, it must not re-assert it until after it receives the acknowledgment (the de-assertion of *ClkAck). Note that if a synchronizer in the completer outbound channel is prone to losing the state of any inflight response or request data during the time that the originator is disconnected, it should suspend accepting new outbound channel data and delay the de-assertion of CompClkAck until it has delivered all the in-flight information to the originator.

The completer-initiated one-sided disconnect / reconnect sequence is the similar to the originator-initiated one-sided disconnect shown above with the roles reversed. The completer uses CompClkCtl to signal its intent to disconnect temporarily by ensuring that the signal is inactive prior to de-asserting CompClkReq. It holds CompClkCtl inactive as it re-asserts CompClkReq to request a reconnect without the re-issuing of credits.

Explicitly signaling a one-sided disconnect provides a hint to the non-requesting agent that the requesting agent does not intend to go into a "deep-sleep" in which it might lose state information such as the count of credits that the non-requesting agent has previously released. In general, the latency and overhead of a one-sided disconnect / reconnect should be lower than a full disconnect followed by an activation.

Since the agent requesting a one-sided disconnect cannot issue credits for its inbound channels in its disconnected state, the other agent that is still connected can send messages on its outbound channels only until the credits that it currently holds run out. However, when a disconnected agent has credits to send, it must reconnect without dependency on further incoming data or command information.

3 Transaction Layer

This topic describes the Scalable Data Port (SDP) architecture at a transactional level.

3.1 Transaction Layer Overview

An SDP transaction consists of multiple phases, which are spread across multiple physical channels. A coherent originator device issues cache-aware transaction requests. When a coherent originator issues a request to read and write one or more bytes of system memory, it chooses between a number of different types of requests (also known as ***commands***) based on the expected use (for example, write, read-only, or read with intend to modify) of the block (cache line) that includes the target data element in its local cache. Cache-aware commands can have implicit or explicit effects on the cache directory of the originator and on other caches in the system.

Figure 2 Scalable Data Port Interface showed the SDP physical channels grouped by type.

In the context of SDP, the originator refers to the entity generating requests on the port, while the completer is the entity toward which the requests are directed. The originator and completer may be connected point-to-point as illustrated in the figure. More typically originator and completer are attached indirectly via a communication network (fabric). A given IP block may utilize multiple ports, and may be an originator on some and a completer on others.

The transaction flow is defined by the command channels, which are listed in the order that they are used in a transaction. Not all transactions use all channels. In general, a transaction is initiated by the originator on one of the originator request channels. If the completer is the fabric, it may then generate probe requests to various originators (including the requesting one) to collect information needed to satisfy the request. (Non-fabric completers do not generate probes.) Each probed originator responds on its Probe Response Channel(s).

The completer generates the appropriate response and sends it back to the originator on either a read response channel or on a write response channel, depending on the command. Finally, the originator may generate a response acknowledge, completing the transaction. Since a transaction in general must pass through all of these stages to complete, a stall of any channel may result in backing up and stalling all channels earlier in the transaction flow. Hence, to avoid deadlock, it is required that no channel make its ability to send data dependent on an earlier channel in the flow sending or receiving.

There is a single data channel in each direction, associated with one or more of the command channels going in the same direction. The originator data channel(s) carry both write request and probe response data. The arbitration for the originator data channel(s) must guarantee that both types of data transfer are able to make forward progress even if the other is blocked. The Read Response Channel(s) carry data for read responses as well as ChgToX, ChgToXNR, and ValBlk.

Finally, there are some additional channels that are not really part of the transaction flow. The common signals are simply the clock and reset signals shared on both sides of the port.

3.2 Data Transfer

SDP has two data channels - Originator (Write/Probe) Data Channel(s) in the originator to completer direction and Read Response Channel(s) (with the read data as part of that channel) in the completer to originator direction.

Data transfer requests move contiguous blocks of aligned doublewords in a byte-addressable system memory address space. For sized reads and writes, the initial address of this block (the address of the least-significant byte of the first doubleword) is equal to the value of the ReqAddr field of the request. For caching reads and ChgToX, the initial address of the block is equal to the value of the ReqAddr field of the request aligned to the system cache line size. For dual-operand Atomics, the initial address is irrelevant and the data transfer occurs as if the access was for a cache line (as if ReqAddr[5] was zero). For other requests, the initial address is equal to the value of the ReqAddr field of the request. Since the address is doubleword aligned, bits [1:0] of the address are always 0.

ReqLen specifies the requested number of doublewords - 1. In the following discussion the value represented by ReqSizeBytes is equal to 4* (ReqLen + 1), that is, the number of bytes requested. Note that the count of the number of bytes in a transfer includes those bytes within the contiguous block of doublewords that may be masked off. The maximum size and address alignment requirements for a single request is implementation specific and may different for reads and writes.

Transfers of individual bytes within a block of doublewords is accommodated through the use of byte enables. If a byte enable is 0, the corresponding byte is masked off (treated as non-valid). Although the actual value on the data byte is not relevant, it still contributes to any data parity. Furthermore, it is recommended that any masked off byte be “fenced” and provided as all zeros or all ones to avoid potential security leaks from uninitialized data. For read requests, the bytes transferred in one request must be contiguous and byte enables are provided for the first and last doubleword only. For write requests, write data is transferred via an Originator Data Channel, which provides a byte enable (OrigDataBytEn[i]) for each byte in the OrigData field. OrigDataBytEn[0] corresponds to byte 0, OrigDataBytEn[1] corresponds to byte 1, and so forth. The byte enables on a write may be sparse – that is, there is no requirement that byte enables must be consecutively set, nor is there a requirement that any byte enables are set for a single transaction. The originator must drive all byte enables to zero for byte lanes that are outside the ReqAddr/ReqLen boundaries, except for dual-operand Atomics. The byte enables for all byte lanes in the cache line being transferred is one for dual-operand Atomics.

In each cycle for which *Vld is asserted, N bytes are transferred across the interface, where N is less than or equal to the width of the *Data field in bytes (DataFieldWidth). Each byte of the data is placed in its naturally-aligned byte lane within the *Data field. Data transfers greater than the width of the data bus occur over multiple cycles (beats), with one beat of data being transferred each cycle. Each beat is transmitted exactly once during the transfer. If the ReqAddr is not aligned to the *Data field width, the data is shifted left so that each doubleword is positioned within the *Data field based on its address. Lanes that are vacated in this shifting are ignored during the transfer.

The wrapping of data bytes within the *Data field is not supported. As a consequence, a data transfer of ReqSizeBytes = DataFieldWidth, where the initial address is not aligned to the *Data field width, requires 2 beats.

Beats within a block of doublewords associated with a specific read or write request can be transferred across the port interface in any order except as noted below in the implementation note. The data source for the transfer chooses the order of the beats, and identifies the beat that is being transferred in a given cycle using the *Offset field. The value of *Offset for the beat that contains the initial doubleword is zero. The address of each beat is the ReqAddr of the request aligned to the *Data field plus the value of the *Offset field multiplied by the width of the field in bytes.

For transfers of size greater than the size of a cache line, all data within each aligned cache line-sized block must be transmitted before moving to the next, but the cache line-sized blocks may be transmitted in any order except as noted below in the implementation note. The last beat (even if there is only one) is indicated by the assertion of *Last. Completers are recommended, but not required, to send the beat addressed by the original request first in read responses. An implementation may guarantee and/or require a specific data beat ordering to ease originator implementation.

Implementation Note: The SOC-15 fabric restricts response data ordering and write data ordering such that the data for a single command (a unique ReqTag) is always transmitted in “address order” with the exception of caching read and ChgToX commands. Address order implies that the *Offset field starts at zero for each transaction and each successive beat increments this *Offset by one such that the data is always observed from the lowest address to the highest address. The restriction that data must be provided in address order applies to both the data fabric and all originators and completers that attach to the data fabric. As an exception to this rule, SOC-15 fabric may return the response data for caching read (RdBlk*) and ChgToX commands in *any* order, regardless of ReqAddr[5:2] and, when multiple cache lines are requested in a single RdBlk* command, may return each individual cache lines in any order.

Figure 14 Aligned Data Transfer illustrates the alignment of data bytes within the *Data field when the request address is aligned to the width of the *Data field.

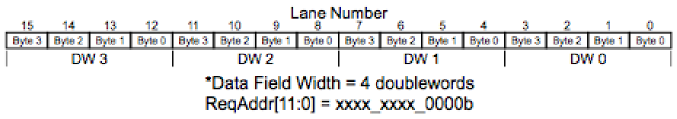
Lane Number
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Byte 3 Byte 2 Byte 1 Byte 0 Byte 3 Byte 2 Byte 1 Byte 0 Byte 3 Byte 2 Byte 1 Byte 0 Byte 3 Byte 2 Byte 1 Byte 0
DW 3
DW 2
DW 1
DW 0
*Data Field Width = 4 doublewords
ReqAddr[11:0] = xxxx_xxxx_0000b

Figure 14 Aligned Data Transfer

In this example, the width of the *Data field is 4 doublewords (128 bits) and an aligned block of four doublewords is being transferred across the interface. Doubleword 0 (DW 0) is transferred in byte lanes 0–3 (*Data[31:0]), doubleword 1 (DW 1) in byte lanes 4–7 (*Data[63:32]), doubleword 2 (DW 2) in byte lanes 8–11 (*Data[95:64]), and doubleword 3 (DW 3) in byte lanes 12–15 (*Data[127:96]). Each byte within a doubleword is transferred in a naturally-aligned position.

If the beginning address of the block of doublewords is not aligned to the *Data field, the doublewords are shifted so that each doubleword rides in its naturally-aligned doubleword position within the *Data field. Figure 15 Unaligned Data Transfer shows the transfer of four doublewords where the least significant bits of the beginning address (ReqAddr [3:0]) = 0100b.

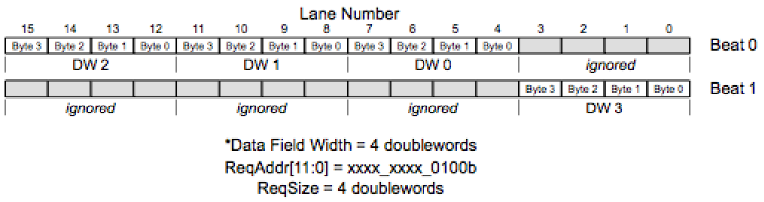
Lane Number
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Byte 3 Byte 2 Byte 1 Byte 0 Byte 3 Byte 2 Byte 1 Byte 0 Byte 3 Byte 2 Byte 1 Byte 0
Beat 0
DW 2
DW 1
DW 0
ignored
Byte 3 Byte 2 Byte 1 Byte 0
Beat 1
ignored
ignored
ignored
DW 3
*Data Field Width = 4 doublewords
ReqAddr[11:0] = xxxx_xxxx_0100b
ReqSize = 4 doublewords

Figure 15 Unaligned Data Transfer

Note that the beats shown above are not necessarily transferred on consecutive clock cycles.

Write data associated with a write request is provided on the OrigData field of an Originator Data Channel. The request that initiates the transfer is presented on an originator request channel. The use of the OrigDataBytEn signals for an aligned write transfer is illustrated in Figure 16 Aligned Write Data Transfer with Byte Enables. Note that the bytes masked off (OrigDataBytEn[i] = 0) can differ for each beat of the transfer.

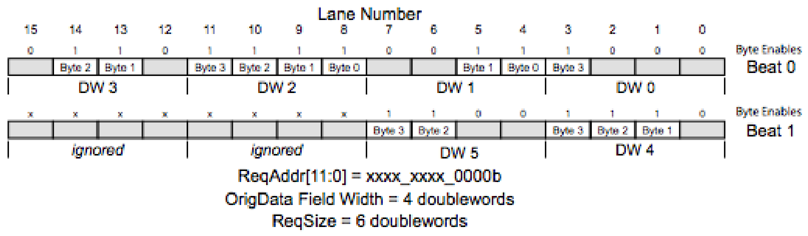
Lane Number
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 1 0 1 1 1 1 0 0 1 1 1 0 0 0
Byte Enables
Byte 2 Byte 1 Byte 3 Byte 2 Byte 1 Byte 0 Byte 1 Byte 0 Byte 3
Beat 0
DW 3
DW 2
DW 1
DW 0
x x x x x x x x 1 1 0 0 1 1 1 0
Byte Enables
Byte 3 Byte 2 Byte 3 Byte 2 Byte 1
Beat 1
ignored
ignored
DW 5
DW 4
ReqAddr[11:0] = xxxx_xxxx_0000b
OrigData Field Width = 4 doublewords
ReqSize = 6 doublewords

Figure 16 Aligned Write Data Transfer with Byte Enables

Note that the beats shown above are not necessarily transferred on consecutive clock cycles.

Write data transfers proceed in the same order on the Originator Write Data Channel within a virtual channel as the corresponding write requests on the Originator Request Channel. Write requests may be issued on more than one virtual channel. The field OrigDataVC specifies the virtual channel for each beat. Write data for different virtual channels may be interleaved. There is no requirement that the write data for two different virtual channels appear in the same order on the Originator Write Data Channel as the requests appeared on the Originator Request Channel. When multiple instances of these channels are implemented, the order that they are presented on each implementation of the Originator Write Data Channel must match the order that they are presented on the Originator Request Channel, with original request channel A considered to be before originator request channel B and originator data channel A considered to be before originator data channel B. As long as the data appears in the same logical order, there is no requirement to match the channel instance – for example, the data from a request on originator request channel B may appear on originator write data channel A.

However, within a virtual channel, write data must be supplied in the same order as the write requests without any interleaving of data beats between requests. When the Original Data Channel is shared for both write data and probe response data, the behavior of the probe response data (OrigDataChan=1) is similar to that of a separate virtual channel for write data (OrigDataChan=0) – that is, the order of probe response data must match the order that the probe responses themselves appeared but there may be interleaving and reordering between the write data and the probe response data. This interleaving may be caused by actual reordering by the originator or it may occur due to arbitrary channel delays between the Probe Response Channel and the Originator Request Channel themselves; making it appear that the data was reordered.

When presenting data for a write request or probe response, OrigDataVld can be asserted at the same time as, or later than, the assertion of ReqVld or PrbRspVld for the corresponding write command or probe response (respectively). It is not allowed to present data prior to the command or probe response. The originator need not check OrigDataRdy, ReqRdy or PrbRspRdy when complying with this requirement. The originator is allowed to have multiple data transfers pending for a single VC and/or data transfers for probe responses as long as they are presented in logical order.

Implementation Note: SDP ports that have a clock-crossing boundary where OrigDataRdy, ReqRdy or PrbRspRdy are generated by an external clock-crossing module must take steps to ensure the above cross-port requirement. One possible method may be to use a single clock-crossing module for the Originator Request Channel, the Originator (Write/Probe) Data Channel, and the Probe Response Channel.

SDP ports that implement the Physical Channel Multiplexing feature operate as two independent ports that are sharing the physical wires of one SDP port. There are no interleaving rules between these two independent channels.

Multiple independent read responses may be interleaved on the interface. Each beat includes transaction identification (combination of RdRspTag and RdRspUnitID). The completer must limit the maximum amount of response interleaving based on the capabilities of originator. This limit is configured for each originator in an implementation-specific way. The originator must have sufficient response buffering to be able to hold the total number of responses that may be interleaved.

Implementation Note: The SOC-15 fabric restricts interleaving on the Originator Data Channel(s) to cache line boundaries; that is, data for two separate writes or probe responses cannot interleave except when the associated address of the beat crosses a cache line boundary. Physical channel multiplexing occurs on a cycle to cycle basis and is not subject to any restriction.

Architecture Note: When a port implements the variable heads-up delay feature, interleaving restrictions, if any, apply to the timing of the response sub-channel.

For sized read commands, the ReqAttr field provides byte enables for the first and last doublewords of the transfer. ReqAttr[3:0] provides byte enables for bytes 3 through 0 respectively of the first doubleword and ReqAttr[7:4] provides byte enables for bytes 3 through 0 of the last doubleword. If ReqAttr[i] = 1, this indicates that the corresponding byte is being requested and will be processed when returned; if ReqAttr[i] = 0, the data in the corresponding byte will be ignored. A read of one doubleword uses ReqAttr[3:0] for byte enables and ReqAttr[7:4] is ignored.

No byte enables are provided on the read response channel. The originator must ignore data returned in the first and last doubleword of a response for those byte positions whose corresponding byte enables in ReqAttr of the request were not set.

The RdBlk* commands always operate on exactly one cache line, so the requested transfer size must equal the cache line size. ReqAddr for the request provides the address of the critical byte. If possible, the beat containing the requested critical byte should be transferred prior to any other beats of the response.

When the multiple request address feature is implemented and the data transfer is for a transaction that is made up of two independent addresses using a non-zero ReqSubAddr, the data transfer is the same as if the originator had made a single cache line size request. Specifically, OrigDataOffset and RdRspOffset are incremented without regard to the fact that these are two independent addresses.

3.2.1 Byte Enable Compression

When byte enable compression is implemented, the size of the originator data channel is reduced by using only one bit in OrigDataBytEn and using OrigData to transfer the actual contents of the byte enables. This feature is intended for SDP implementations that are pin constrained for the port. This section describes the functioning of the OrigData bus when byte enable compression is implemented.

If the byte enables are all asserted for all bytes of a data transfer – i.e. all bytes between ReqAddr to ReqAddr+ReqLen are written - or when OrigDataChan=1 (carrying probe response data of a full cache line), the originator may optionally skip transferring the byte enables. To skip transferring the byte enables, the originator de-asserts OrigDataBytEn during the entire data transfer and the number of data beats is equal to the number of bytes divided by the size of the OrigData (the same number of beats in a data transfer as if byte enable compression was not implemented).

Otherwise, the originator transfers the byte enables on OrigData. In order to perform this, the originator asserts OrigDataBytEn during the first transfer beat(s) and places the actual byte enables on OrigData. For all request data (e.g. writes and atomics), the position of the byte enables on OrigData naturally aligns to the width of OrigData. For example, if OrigData has 16 bytes, then OrigData[0] would be the the byte enable for ReqAddr[5:4,000b]. For probe response data, OrigData[0] is for the first byte of the cache line. In the event that the width of OrigData is too small to transfer the byte enables in the first beat, then additional beats are used to transfer the remaining byte enables in address order, with OrigDataBytEn asserted until the byte enable transfer is complete. In the event that the width of OrigData is larger than is necessary to transfer the byte enables, all other OrigData bits are reserved and should be set to zero. Following the byte enable transfer, the originator de-asserts OrigDataBytEn while transferring the data beats in the normal fashion.

Either no data beats are used to transfer the byte enables (for non-sparse writes or probe data), or all byte enables for every transferred data byte are sent.

OrigDataParity provides even parity of the byte enables while OrigDataBytEn is asserted.

3.2.2 Originator Data Compression

SDP ports that implement the originator data compression feature are allowed to issue four additional commands:

- VicBlkFullZero
- WrSizedFullZero
- VicBlkFullComp
- WrSizedFullComp

VicBlkFullZero and WrSizedFullZero set the appropriate write length in ReqLen but do not transfer any data. The completer acts as if the data beats were transferred with all-zero data, all byte enables set (consistent with ReqAddr/ReqLen) and OrigDataError=0.

VicBlkFullComp and WrSizedFullComp set the appropriate write length in ReqLen but may transfer fewer beats of data. The actual number of data beats is specified in ReqAttr[7:6]. Based on ReqLen, any address bits that must be zero due to the aligned nature of the command carry information on the compression type. For example, if ReqLen=0xF, the command must be an aligned 64-byte address so ReqAddr[5:2] carries a compression type. The encoding of this compression type is outside the scope of this document.

All other requirements of VicBlkFull apply to VicBlkFullZero and VicBlkFullComp. All other requirements of WrSizedFull apply to WrSizedFullZero and WrSizedFullComp.

1.1.1 MetaData Compression

SDP ports that implement the metadata compression feature use an additional extension to the data – the “metadata” that indicates both if the data is compressed and how to decompress the data.

At a high level, the metadata can be considered a “dictionary index” into either a compression methodology or a direct map of the data. The uncompressed portion of the data can only be reconstructed with both the metadata and the data bytes. The actual size and encoding of the metadata and the dictionary methodology is outside the scope of this specification but must be common for the entire SOC. However, the specification does require that a metadata of all ones (for example, FFh if the metadata is 8-bits) indicates that the data is uncompressed. The method that the completer uses to store the metadata and to look up the metadata is also outside the scope of this specification. As a hypothetical example using an 8-bit metadata field, a metadata value of 01h (with an upper ninth bit implemented to carry even parity) may indicate that the data block has every byte equal to a single hex value. That value may be in byte 0 of the transferred data byte (OrigData[7:0]). If OrigData[7:0] is 55h, then the uncompressed data block is a repeated 55555555h…. for the entirety of the block. Only two bytes needs to be stored in order to restore this data – the metadata (01h) and the value of the repeated data (55h).

Not every SDP port implementation in an SOC must implement the metadata compression feature. When there is a mix of SDP implementations, a port that does not implement the metadata compression feature (referred to as a port that supports only uncompressed data in this paragraph) is reading or writing data, the data is always uncompressed on the read response. It is system dependent whether the completer bypasses the lookup of the metadata when a port requests data without implementation of the

ReqCompMode and ReqSpecDataFetch. If the completers bypass the lookup, hardware and/or software must provide a means (also outside of the scope of this specification) to ensure that the data is uncompressed before it may be addressed by any SDP port that requires only uncompressed data. The use of WrNoDataNC with ReqCompMode=11b may be used to uncompress any data before it may be accessed using a metadata bypass. It is legal for an SDP port to use WrNoDataNC on data that is currently uncompressed. The use of ReqCompMode=00b (bypass) also requires that the hardware and/or software knows that the data must be uncompressed. There are no required consistency checks at a completer that may detect reads and/or writes that bypass the metadata lookup when the data is actually compressed, and special care must be taken to prevent incorrect data being read or written. An implementation that allows metadata bypass may require software to clear the data blocks to avoid a security leak with a malicious actor that bypasses metadata on compressed data. This is outside the scope of this specification.

When writing compressed data, it is required that the equivalent uncompressed data block is a fixed size. The current SDP specification requires the uncompressed block be either 128B or 256B (compression works on either 128B or 256B chunks) and the address must be aligned to the block size. A write with compression enabled therefore updates an aligned fixed-size block and therefore the byte enables can be presumed to be all ones. The OrigDataBytEn field is instead used to carry the metadata instead. When multiple data beats are transferred for this compressed transfer, then OrigDataBytEn is repeated on all beats. The use of OrigDataBytEn as metadata is specified by the command (WrSized* except WrSizedFullComp) and when ReqAttr[7:6] != 00b. All other commands (including all atomics), or when ReqAttr[7:6] = 00b, have uncompressed data on OrigData* and the OrigDataBytEn field is interpreted as is normal for data transfers in SDP. Compressed data is still assigned sequential OrigDataOffset values from 0 to n-1 (where n is the number of beats actually transferred) based on the order that the beat occupies in the compressed block. Any requirements to provide data in address order also holds for the compressed data.

When reading compressed data, the originator may request that the data be returned compressed or uncompressed. When requesting that the data is returned compressed, the compression algorithm may not have a method to compress the current data block contents – a metadata dictionary cannot possibly hold all arbitrary data patterns – and thus the RdRspDataCompMeta specifies the actual compression status of the block (all ones indicates the data is uncompressed). Data returned by atomics and commands that do not support metadata compression is returned with an uncompressed encoding. When metadata is bypassed, the metadata returned also indicates that the data block is uncompressed (all ones).

When transferring compressed data on an originator data channel, ReqLen indicates the size of the compressed data. ReqAttr[7:6] will indicate the uncompressed size. When returning compressed data on a read, the specified ReqLen assumes that the data block is uncompressed and thus it is the size of the data block being requested. The number of data beats returned may be smaller dependent on the actual size of the compressed data. The RdRspLast field indicates the actual size of the returned compressed data. Compressed data is still assigned sequential RdRspDataOffset values from 0 to n-1 (where n is the number of beats actually transferred) based on the order that the beat occupies in the compressed block. Any requirements to provide data in address order also holds for the compressed data.

Some dictionaries may be able to compress fixed data patterns to be contained entirely within the metadata itself (for example, all zeros may be indicated with a unique metadata encoding). In this case, it is still required that a single beat of data is transferred, if only to transfer the metadata. It is recommended that any unused bytes be “fenced” and provided as all zeros or all ones to avoid potential security leaks from uninitialized data but the completer may skip the access of the actual data block to save bandwidth. For example, the actual memory may not be written and only the metadata is updated. The data parity still maintains even parity regardless of whether or not the bytes are required to reconstruct the uncompressed block. Furthermore, data credit handling operates as normal for this single data beat even if OrigData or RdRspData is not necessary at to reconstruct the uncompressed block.

Victims and probe responses with data transfer must always provide uncompressed data as ReqAttr and ReqCompMode have no method to indicate compression status for victim and probe response data. When an SDP port uses WrNoDataNC to decompress data, any system caches are not necessarily invalidated.

Two halves of a 256B data block may be compressed independently as indicated by the metadata value. A completer may also compress a 256B data block, or may combine the compression of two 128B data blocks as long as ReqCompMode does not require the completer compression to be disabled.

3.2.3 Read Response with NODATA

RdRspDataStatus of NODATA is allowed only in the following cases:

1) When there is an idle cycle on the bus (i.e. RdRspVld or, if the heads-up feature is implemented, RdRspDataVld was not asserted). However, originators ***must*** ignore the contents of RdRspDataStatus when there is an idle cycle on the bus.
2) The completer has previously responded to the transaction using EARLY status (and data) and is now completing the transaction while indicating that the previously supplied data is the valid data.
3) The completer may use RdRspStatus=OKAY_NODATA and RdRspDataStatus=NODATA as a response to a ChgToX transaction.
4) The completer uses RdRspStatus=OKAY_NODATA and RdRspDataStatus=NODATA as a response to a ChgToXNR and ValBlk transactions.
5) The completer may respond to a transaction that has been canceled with a read response using either RdRspStatus=OKAY or OKAY_NODATA, and RdRspDataStatus=NODATA. This encoding occurs if the transaction was actually canceled. If the cancel was not performed, the transaction completes normally.
6) The completer may respond to a RdBlkL that indicated it was a storing probe refetch (ReqAttr[1:0] = 10b) with RdRspStatus=OKAY_NODATA and RdRspDataStatus=NODATA. The reason for not providing data on a storing probe fetch is outside the scope of this specification.
7) The completer uses RdRspDataStatus of NODATA with any unsolicited SYSFATALERR response.
8) The originator used certain SpecPrefCmd encodings that do not require a data response.

It is not legal to provide RdRspDataStatus of NODATA on error cases (e.g. when the RdRspStatus is DECERR, SLVERR, TRANSERR or PROTVIOL) when the transaction otherwise would return data in a non-error situation. It is recommended that completers use NODATA for error responses on a ChgToX transaction. It is also not legal to provide RdRspStatus of OKAY_NODATA and to provide a RdRspDataStatus other than NODATA.

A response with NODATA is always a single beat (RdRspLast asserted).

3.2.4 Read Response and Data Associated with Error Status

If RdRspStatus indicates an error (SLVERR, DECERR, PROTVIOL, TRANSERR, CMPTO or CRS), the completer must transfer all requested data beats including the assertion of RdRspLast, using manufactured data if there is no data available due to the error. A manufactured data pattern of all ones is recommended.

For cacheable transactions, the originator must not cache the returned data. In general, any transaction that completes with RdRspStatus or WrRspStatus indicating an error does not alter the cache state of the line.

3.2.5 Read Response Data Retransmission

If the RdRspDataStatus is RETRANSMIT on a given beat, the data transfer must complete through the beat associated with the assertion of RdRspLast. Once RETRANSMIT has been asserted, the data beat coincident with RETRANSMIT and all subsequent beats for the same response must be retained by the originator but not consumed or forwarded until the last beat has been received.

When RETRANSMIT has been asserted on the last beat, the originator must discard any beats that were retained and not consumed or forwarded. The originator must not send a response acknowledgement (if used on this response) until the data has been retransmitted. The completer will subsequently retransmit the entire response sequence including any prior beats without dependency on the Probe Request Channel. The completer may interleave other transaction responses between the RETRANSMIT status and the actual retransmission.

During a retransmission event, the originator must keep track of which data beats, if any, had already been consumed or forwarded. The completer is allowed to issue any valid combination of RETRANSMIT, VALID or DATERR during this retransmission event. The completer must provide a means to limit the number of retransmits to avoid livelock, but the originator must not have any limit.

When RETRANSMIT has been asserted on a beat other than the last beat, and RETRANSMIT is not asserted on the last beat, the RdRspDataStatus encoding from the last beat applies to all beats that asserted RETRANSMIT.

Table 37 Read Response Data Retransmission Possibilities shows all possibilities for a response that spans two data beats.

Table 37 Read Response Data Retransmission Possibilities

RdRspDataStatus for each beat	Meaning
VALID, VALID	Both beats are valid and may be consumed.
VALID, DATERR	The second beat contains an uncorrected data error.
DATERR, VALID	The first beat contains an uncorrected data error.
DATERR, DATERR	Both beats have an uncorrected data error.
DATERR, RETRANSMIT	The first beat contains an uncorrected data error. The completer will retransmit the entire response at a later time, including the first beat. On the retransmit, it is not predictable whether the first beat will have an uncorrected data error or if the data error was not reproducible.
VALID, RETRANSMIT	The first beat is valid and may be consumed. The completer will retransmit the entire response at a later time, including the first beat.
RETRANSMIT, VALID	Both beats are valid and may be consumed.
RETRANSMIT, DATERR	Both beats have an uncorrected data error.
RETRANSMIT, RETRANSMIT	Neither beat may be consumed. The completer will retransmit the response at a later time.

3.2.6 Read Response Data Lag and the Variable Heads-Up Feature

When the heads-up feature is not implemented, the read response channel is one channel and RdRspVld qualifies all signals on that channel. There is no lag or time difference between when the read response tag or status is presented and when the data is presented.

However, when the heads-up feature is implemented, the read response channel is split into two sub-channels. The first (Response sub-channel) provides advance notice of the status of a read request along with information about future data transfer cycles. The second sub-channel (Data sub-channel) provides the data (RdRspData), its parity (RdRspDataParity), and information about the data (RdRspDataStatus, RdRspDataStatusParity, and RdRspDataUser).

With this feature, the data sub-channel is produced, unconditionally, N_{lag} cycles after both RdRspVld and RdRspRdy are asserted and RdRspDataVld must be asserted on this cycle. RdRspVld qualifies only the data on the response sub-channel on any given clock cycle and RdRspVld is not necessarily asserted on the cycle that the data is presented (unless another read response is initiated at that time). The data sub-channel is qualified by a RdRspDataVld, which is an equivalently lagged version of RdRspVld. On a given clock cycle, RdRspRdy signals acceptance of both sub-channels.

The read response variable heads-up feature provides the originator of a read request with one of two intervals indicating the lag between the status of a read request and the subsequent data. Without the variable heads-up feature, the read request status and the subsequent data may only be lagged by a single interval N_{lag}.

When the variable heads-up feature is implemented, RdRspDelay indicates which of two fixed heads-up (or data lag, depending on the perspective) intervals will be applied to this response. The interval between a specific response and any data associated with the response is N_{lag} clock cycles, where N_{lag} = (RdRspDelay = 0) ? N_{short_lag} : N_{long_lag}. If programmable, the values must be static during normal operation. N_{short_lag} and N_{long_lag} may be any whole number (0, 1, …) and are not necessarily restricted to be different. When the variable heads-up feature is not implemented, N_{lag} is always a single fixed N_{short_lag} that is configured for the port.

Application Note: When the completer and the originator are actually in different clock domains, in general, the higher frequency IP block must have a method to determine which clock cycles should not be counted in determining the lag between response and response data due to the synchronization logic. Such means is outside the scope of this specification.

Implementation Note: Most implementations are expected to not implement the heads-up feature and thus align the read response data with the read response. Some implementations will implement the heads-up feature use a fixed heads-up lag (a single N_{short_lag}) and few implementations are expected to implement the variable heads-up feature. Currently, no SOC-15 IP plans to implement the variable heads-up feature.

3.2.7 Probe Data

Probe data is returned to the fabric on the OrigData field of an Originator Data Channel in response to a probe request on a Probe Request Channel. The size of the transfer is always equal to the system cache line size. Probe data transfers must proceed in the same order as the corresponding information on the Probe Response Channel. There is no interleaving within the Probe Response Channel, but transfers may be interleaved between the Originator Probe Data Channel and the Originator Write Data Channel. Data beats are identified as belonging to one or the other based on the OrigDataChan bit. When this signal is a 0, the data presented on the OrigData field is write data. When is 1, the data on the OrigData is probe response data.

3.3 Virtual Channels

Virtual channels (VCs) are supported to provide passing guarantees (for deadlock prevention). They are also used as the basis for grouping transactions for ordering purposes. VCs are implemented by reserving resources in a physical channel on a per virtual channel basis and reordering requests in different VCs, such that stalling one VC cannot cause another to be stalled. All transactions are assigned a VC identifier by the originator, and use that to select the VC on all physical channels that support multiple VCs.

VCs are not intended to indicate any type of priority among various transactions; however, cross-VC dependencies may exist that cause one VC to block behind another. A hierarchy of VCs is established with blocking (either within a VC on the same physical channel, or of incoming traffic blocking behind outgoing traffic in a different physical channel) only supported in one direction, to avoid loops leading to deadlock.

The Originator Request Channel and the Originator Write Data Channel support multiple virtual channels. IP blocks are free to not implement support for any virtual channels in which they know they will never initiate requests. The fabric and completer receives and completes the requests in one virtual channel without a dependency on other virtual channels, except for dependencies specifically requested between the posted channel and the non-posted channel by the *PassPW bits. SDP request virtual channels are, in general, capable of carrying both reads and writes. For cases where requests are required to always be able to pass previous requests (for example, posted writes passing reads in PCIe ordering), the later request must be placed in a different virtual channel. Reordering within a request virtual channel is allowed, subject to QoS and ordering rules, but cannot be guaranteed for purposes of deadlock avoidance.

The Probe Request Channel is not separated into multiple virtual channels, which means that a blocked probe request can block requests associated with any other virtual channel, creating a cross-VC dependency. In addition, it is permissible for an originator to block the incoming Probe Request Channel based on its ability to issue a cache eviction (VicBlk*) request. In order for this to be deadlock free, it is necessary that evictions be placed in a virtual channel that contains no probe-generating requests (evictions never generate probes). It is also required that the virtual channels used for evictions must ignore bus locks. Probes must complete without any other external dependencies, and are generally expected to drain quickly.

The read and write response physical channels also support virtual channels. An originator may never make its ability to receive a response contingent upon its ability to issue a request, nor may it make receiving a response contingent upon receiving a separate response. The fabric may not make its ability to receive a response contingent on its ability to issue a probe.

3.3.1 Virtual Channel Request Limitations

Virtual channel usage is implementation specific. The only requirement for interfaces that support the Posted Write Ordering feature is that ReqVC of 0 is for non-posted requests and ReqVC of 1 is for posted requests. Some virtual channels limit the use of certain commands.

Virtual channel usage for SOC-15 data fabric is shown in Table 38 Virtual Channel Usage for SOC-15 Data Fabric. Virtual channel usage for SOC-15 command processing network (for CP communication, doorbells, and address translation services) is shown in Table 39 Virtual Channel Usage for SOC-15 Command Interface Fabric.

Table 38 Virtual Channel Usage for SOC-15 Data Fabric

VC Number	Name	Usage
0	Non-posted	Non-posted ("downstream" or "peer to peer"), or default VC.
1	Posted or WriteBack[1]	Posted ("downstream" or "peer to peer"), or default writeback VC. Only sized write commands or WrNoDataNC are supported when the VC is used as the posted channel. Only WrSizedNC, WrSizedFullNC, or cache eviction commands are supported when the VC is used as the writeback channel.
2	UpRd or SysMgt	"Upstream" read. This channel is also used for DVM messages that are sent to completer ports (that do not have a Probe Request Channel).
3	UpWr or WriteBack[1]	"Upstream" write (including atomics), or alternate writeback VC (for ports using posted channel). Only sized write commands, atomic commands, or WrNoDataNC are supported when the VC is used as the posted channel. Only WrSizedNC, WrSizedFullNC, or cache eviction commands are supported when the VC is used as the writeback channel.
4	Mem	Non-real time DRAM only.
5	SRT	Soft real time.
6	HRT0	Hard real time channel 0 (higher bandwidth).
7	HRT1	Hard real time channel 1.

Notes:
1. The writeback virtual channel allows for traces and evictions that do not depend on system probes. The channel used is normally VC1 unless the posted virtual channel is already used at that port, in which case it is VC3.

Table 39 Virtual Channel Usage for SOC-15 Command Interface Fabric

VC Number	Usage
0	Address invalidations (AddrXlateWrSz)
1	Doorbells
2	Address translation services (AddrXlateRdSz)
4	Command processor communication
3, 5-7	Reserved

3.4 Channel Flow Control

Flow control at the protocol level is provided by a credit-based system. The receiver on a particular channel releases credits to the transmitter. In order for the transmitter to issue a request or transfer data, it must acquire the appropriate number of credits. The receiver releases credits to the transmitter at port reset and on an ongoing basis as buffer space frees up. The receiver on a channel is capable of releasing one credit per cycle based on its clock rate (which may differ from the transmitter's clock rate). The flow of credits is in a direction opposite to the flow of information on the channel.

The credit release signals are collectively referred to as the credit release sub-channel. The credit release sub-channel has its own *CreditRdy signal used for link-level pacing. A single credit is released by the receiver on a physical channel when *CreditVld is asserted on an active clock edge when *CreditCnt is zero or not present. The fields *CreditVC, *CreditType, if supported and implemented, specify the type of credit being released. The field *CreditCnt can be used to release more than one credit at a time. When multiple credits are released at one time, they all have the same type and VC.

If an originator implements the probe subchannel, the Originator Data Channel(s) are shared between write data and probe response data. Flow control is provided for both with a common pool. See Credit Release Sub-channel for more information. When using a common pool, the originator must ensure that at least one originator data credit is reserved for probe responses that transfer data. Implementations that support a Probe Support feature but do not support the Virtual Channel Support feature must note that the number of credits released with OrigCreditPrbCreditRel=0 may over-state the pool credits if originator data transfer coincides with initial credit release.

The Originator Data Channel optionally supports physical channel multiplexing. The physical channel multiplexing feature requires the implementation of the OrigDataCreditChanAB signal which specifies whether the write data credit being issued is associated with channel A or B.

The transmitter on a channel is expected to record credits received even when its *Vld signal for that channel is de-asserted. Counters used by the transmitter on a channel for tracking credits issued by the receiver must be implemented as saturating counters, that is, counters that do not roll over to zero when they exceed their maximum capacity.

On the Originator Request and Probe Request Channels a single credit represents the availability of a buffer in the receiver capable of storing a single request. For the Write Response Channel, a single credit represents buffer space for a single write response. For the Read Response Channel, a single credit includes both a unit of data payload buffer space and any space required to hold the response meta-data (for example, RdRspTag, RdRspUnitID, RdRspStatus, and RdRspOffset). For the Response Acknowledge Channel, a single credit represents the availability of a buffer in the receiver capable of storing a single transaction acknowledgment.

For the Originator (Write/Probe) Data Channel and the Read Response Channel, a single credit represents the availability of one naturally aligned block of buffer space. The size of this block in bytes must be an integer power-of-2 multiple of the width of the *Data field / 8. The designated size of a buffer space block on the Originator Data Channel is independent of the size of a buffer space block on the Read Response Channel. Both sides of the port must agree on the credit size, using a means that is outside the scope of this document. When byte enable compression is implemented, any byte enables transmitted on OrigData do not affect credit handling.

Implementation Note: To simplify credit return, it is recommended that each credit represent buffer space equivalent to a single beat of data. However, a port that always transfers data in fixed-sized blocks may wish to use a credit size that corresponds to this fixed block.

An originator issuing a write request or a probe response with data must acquire one credit on the Originator Request Channel or Probe Request Channel (as appropriate) and at least one credit on the Originator Data Channel before either the request/probe response, or the data is sent. If, after beginning the transfer, the originator determines that it does not have sufficient credits on the data channel to complete the transfer of the remainder of the payload, it must suspend transferring data at the point at which it has filled the buffer space equivalent to the number of credits it currently holds. It may resume the transfer when it acquires another credit.

When a read response with RETRANSMIT is used, the retransmitted response must acquire a separate credit, it may not “reuse” the credit from the transaction sent with RETRANSMIT status.

A read response with RdRspDataStatus=NODATA must acquire one credit even though there is no required buffer space for the data. This credit provides space to hold the response fields such as RdRspStatus and is not treated differently in credit handling than a read response with a single beat of data.

Implementation Note: For the SOC-15 fabric, there is an additional requirement that no data transfers can be initiated by an originator or completer unless there are at least as many data credits necessary to transfer 64-bytes (for example, two credits if the data bus width is 32-bytes), even if the transfer is smaller than 64-bytes. This requirement holds for requests with data, probe responses with data and read responses with data.

For sized transfers, data channel buffer blocks are filled starting at an offset within the buffer equivalent to the initial (lowest address) of the data payload. Data does not wrap in the buffer. Therefore, the buffer is full when the offset reaches the value of (buffer_block_size - 1). If the initial address of a block of data modulo the buffer block size is non-zero, n bytes (where n = initial_address MOD buffer_block_size) of the buffer are unused. As a consequence, even when the size of the payload is an integer multiple m of the buffer block size but is not aligned to the buffer block, m+1 credits are required to transfer the entire payload.

The Port Control Signals are not used to transfer serialized data and therefore do not require any flow control.

3.4.1 Virtual Channel Flow Control

The Request, Read Response, Write Response, and Originator Data channels support virtual channels. Virtual channel flow control utilizes two different types of credits: VC-specific credits and pool credits. When a receiver issues a VC-specific credit this means that it has freed a buffer that was utilized to hold information associated with a specific virtual channel. A pool credit is not associated with any specific virtual channel.

When the receiver on a channel issues a credit, the type of credit being released is indicated by the *CreditType field. If *CreditType = 0, the credit is VC-specific and the virtual channel number is provided by the *CreditVC field. If *CreditType = 1, the credit being issued is a pool credit and the *CreditVC field is ignored.

SDP implements a transmitter-managed buffer allocation scheme.

On initial reset, a receiver may release either all pool credits or credits for specific virtual channel(s). It is expected that a receiver would do one or the other, but not a mix. However, a transmitter should not depend on this behavior. The transmitter manages this pool of credits using them to send information on any of the implemented virtual channels. It is the responsibility of the transmitter to reserve credits for use in a particular virtual channel if this is necessary for deadlock avoidance or bandwidth guarantees in the presence of other traffic.

When the transmitter uses a pool credit to send a request, response or data beat, it indicates the virtual channel being used (e.g. ReqVC). The receiver records this virtual channel number associating it with the buffer it uses. When the receiver frees that buffer, it reflects this same virtual channel number back to the transmitter via the *CreditVC field (with *CreditType = 0). The transmitter decides whether this freed buffer should remain allocated to a specific virtual channel or returned to the pool that it is managing.

The release of a credit cannot be taken to mean that a particular transfer has been processed to an ordering point by a receiver.

When virtual channels are not implemented, all credits are effectively issued for VC0.

3.5 Transaction Identification

Commands issued on a specific port are identified by means of a transaction tag. The tag is contained within the request and returned with the response allowing each request and its corresponding response to be matched up. This is true for commands issued on either the Originator Request Channel or the Probe Request Channel and is required because responses may, in general, be returned out of order. Commands issued from originator devices also contain a UnitID field, which identifies a particular requester within the originator device sharing the port. Each independent unit has its own tag space, requiring the combination of ReqUnitID and ReqTag to be used to identify the transaction. When multiple channels of a given type are implemented (originator request channel A and originator request channel B, or probe request channel A and probe request channel B), there are not multiple tag spaces for each. The Originator Request Channel shares a common tag space amongst all implemented channels. When multiple probe request channels are implemented, the Probe Request Channel also shares a common tag space. There is no correlation between the tags on the Originator Request Channel and the Probe Request Channel.

All ports in the system have independent tag spaces. That is, there is no guarantee that the tag of a request issued by an originator will be the same at the completer port from which it emerges. The fabric is free to assign these as it will and may include the originating port and unit information in that calculation.

3.6 Commands

SDP originators initiate transactions by issuing requests (also called commands). The type of request is indicated via the ReqCmd field of the Originator Request Channel as shown in Table 40 Commands. Legal values and meanings of the various command encodings are listed below. ReqCmd[5] indicates that the command will transfer data on an Originator Data Channel.

Note the columns labeled **Orig Data**, **Resp Chan**, and **Requires Ack**. A "yes" in the **Orig Data** column means that the originator also supplies data on an originator data channel. The **Resp Chan** column indicates whether the response to the command is returned to the originator on a read response channel or a write response channel. The **Requires Ack** column indicates whether the originator is required to generate a response acknowledge for this command. A Yes/No in this column means that the value of the ReqAck field of the Originator Request Channel determines if a response acknowledgment is required.

Table 40 Commands

SDP Command	ReqCmd	Orig Data	Resp Chan	Requires Ack	Description
Reserved	00 0000b/00h	—	—	—	
RdBlkL	00 0001b/01h	No	Read	Yes	Coherent cache block load, installing in highest possible access state given the state of other caches.
RdBlkS	00 0010b/02h	No	Read	Yes/No	Coherent cache block load, installing in S state.
RdSized	00 0011b/03h	No	Read	No	Noncached coherent load, any size.
RdSizedNC	00 0100b/04h	No	Read	No	Noncached non-coherent load, any size.
RdSizedDWX	00 0101b/05h	No	Read	No	Noncached coherent load with exclusive lock request.
RdSizedNoWriter	00 0110b/06h	No	Read	No	Noncached coherent load that ensures no cached copies exist in an X state. This command is reserved only for transactions that are issued under bus lock.
RdBlkX	00 0111b/07h	No	Read	Yes	Coherent cache block load, installing in an exclusive (M, D, or E) state.
ClnBlkAll	00 1000b/08h	No	Write	No	Force all caches to write back modified copies of the line to memory.
WbInvBlkAll	00 1001b/09h	No	Write	No	Force all caches to evict the line, returning modified data to memory.

InvBlkAll	00 1010b/0Ah	No	Write	No	Force all caches to evict the line, discarding modified data.
ChgToXNR	00 1011b/0Bh	No	Read	Yes	Gain write access to a line without reading data.
ValBlk	00 1100b/0Ch	No	Read	Yes	Gain write access to a line without reading data. Originator must replace the entire cache line.
VicBlkCln	00 1101b/0Dh	No	Write	Yes/No	Indicate to the fabric that a non- modified line is being evicted.
Fence	00 1110b/0Eh	No	Write	No	Indicate an ordering requirement between transactions on each side of the fence.
Flush	00 1111b/0Fh	No	Write	No	Indicate that transactions prior to the flush have reached their destinations.
Reserved	01 0000b/10h	—	—	—	
AddrXlateRdSz	01 0001b/11h	No	Read	No	Initiates Address Translation Services where a data response is necessary, such as translating an address. **Architecture Note**: Prior to revision 1.3.2, this command was documented as RdBlkNotO but this command was never supported on any SOC and is now deprecated.
RdBlkC	01 0010b/12h	No	Read	Yes	Coherent cache block load, installing in a Clean (S/E/F) state.
RdSizedDW	01 0011b/13h	No	Read	No	Non-cached coherent load, whole DWs.
Cancel	01 0100b/14h	No	—	No	Request issued by the fabric to the memory controller to cancel an earlier read command unless it is too late to do so.
SpecDramRd or SpecPrefCmd (when ReqAttr!=00h)	01 0101b/15h	No	Read (optional based on ReqAttr)	No	When ReqAttr=00h, this is a hint to memory controller that the originator is likely to issue a read to the ReqAddr in the near future. When ReqAttr!=00h, additional features for SpecPrefCmd are also available. Depending on the encoding in ReqAttr, the command may have a read response.
QosControl	01 0110b/16h	No	—	—	Forward a QoS priority change.
ErrEvent	01 0111b/17h	No	—	—	Used to signal to the system that a non-recoverable error condition has been detected.
Reserved	01 1000b/18h	—	—	—	
VicBlkFullZero	01 1001b/19h	No	Write	Yes	Evict and write back a fully valid modified line of zero data to memory.
Reserved	01 1010b/1Ah	—	—	—	(Was IcInvBlkAll)
ChgToX	01 1011b/1Bh	No	Read	Yes	Gain write access to a line without reading data but with option to return dirty data from other caches.
Reserved	01 1100b/1Ch	—	—	—	Architecture Note: Used internally for fabric
WrNoDataNC	01 1101b/1Dh	No	Write	No	Non-coherent non-cacheable request without data transfer
WrSizedFullZero	01 1110b/1Eh	No	Write	No	Non-cached coherent store of one or more full data beats of zero data.
Reserved (Extension)	01 1111b/1Fh	—	—	—	Reserved for future extensions to ReqCmd width.
Reserved	10 0000b/20h	—	—	—	
WrSizedCndlDR	10 0001b/21h	No	Read	No	Non-cached coherent store with data return to indicate exclusive lock status.
Reserved	10 0010b/22h	—	—	—	
Reserved	10 0011b/23h	—	—	—	
Reserved	10 0100b/24h	—	—	—	
VicBlkClnD	10 0101b/25h	Yes	Write	Yes	Evict a non-modified line to the fabric.
Reserved	10 0110b/26h	—	—	—	
Reserved	10 0111b/27h	—	—	—	
WrSized	10 1000b/28h	Yes	Write	No	Non-cached coherent store, any size.
WrSizedFull	10 1001b/29h	Yes	Write	No	Non-cached coherent store of one or more full data beats.
VicBlkPtl	10 1010b/2Ah	Yes	Write	Yes	Evict and write back a partially valid modified line to memory.

VicBlkFull	10 1011b/2Bh	Yes	Write	Yes	Evict and write back a fully valid modified line to memory.
WrSizedNC	10 1100b/2Ch	Yes	Write	No	Non-cached non-coherent store, any size.
WrSizedFullNC	10 1101b/2Dh	Yes	Write	No	Non-cached non-coherent store of one or more full data beats.
VicBlkFullComp	10 1110b/2Eh	Yes	Write	Yes	Evict and write back a fully valid modified line to memory (with data compression).
WrSizedFullComp	10 1111b/2Fh	Yes	Write	No	Non-cached coherent store of one or more full data beats (with data compression).
Atomic	11 0000b/30h	Yes	Read	No	Coherent atomic read-modify-write. See Atomics below for more information.
AtomicNC	11 0001b/31h	Yes	Read	No	Non-coherent atomic read-modify-write. See Atomics below for more information.
AtomicNR	11 0010b/32h	Yes	Write	No	Coherent atomic read-modify-write with no data returned. See Atomics below for more information.
AtomicNRNC	11 0011b/33h	Yes	Write	No	Non-coherent atomic read-modify-write with no data returned. See Atomics below for more information.
AddrXlateWrSz	11 0100b/34h	Yes	Write	No	Initiates Address Translation Services where a data response is not necessary, such as requesting a completer to remove prior cached versions of virtual to physical address translation.
Reserved	11 0101b/35h	—	—	—	Note: Used internally by fabric.
Reserved	11 0110b/36h	—	—	—	Note: Used internally by fabric.
Reserved	11 0111b/37h	—	—	—	Note: Used internally by fabric.
Reserved	11 1000b/38h	—	—	—	
Reserved	11 1001b/39h	—	—	—	
Reserved	11 1010b/3Ah	—	—	—	
Reserved	11 1011b/3Bh	—	—	—	
Reserved	11 1100b/3Ch	—	—	—	
Reserved	11 1101b/3Dh	—	—	—	
Reserved	11 1110b/3Eh	—	—	—	
Reserved (ExtensionD)	11 1111b/3Fh	—	—	—	Reserved for future extensions to ReqCmd width.

The following sections provide more information about these commands organized in the following groups:

- Non-caching reads
- Writes
- Atomics
- Caching reads
- Cache upgrades
- Cache evictions
- Barriers
- Cache maintenance
- DVM Operations
- Other commands

3.6.1 Non-caching Reads

The non-caching reads include the following commands:

- RdSized
- RdSizedNC
- RdSizedDW
- RdSizedDWX
- RdSizedNoWriter

Originators perform non-caching read requests to acquire data that it does not intend to cache. The NC version is non-coherent and does not probe other originators. Any request size is legal. No response acknowledgements are provided, so the request is considered complete on SDP when the read response is returned. It is illegal for the fabric to return RdRspStatus with the PassDirty bit set. If a cache attempts to pass dirty data, the fabric must do an implicit write back to memory.

The DW version is the same format except that the byte enables for the entire length are all enabled. The DW version of the command should be used (but is not required to be used) when byte enables are all ones.

RdSizedNoWriter indicates that the originator is requesting a probe that ensures no exclusive copies of the line are kept. This operation must be used only while the originator has a bus lock and is performing a read for an atomic read-modify-write operation. The RdSizedNoWriter command is not legal at any other time.

RdSizedDWX is part of the Exclusive Lock Request Feature and may only be issued if ReqIO=0 and ReqSpecialAddr=0. In addition to acting as a RdSizedDW, the completer may track the targeted addresses (as indicated by ReqAddr and ReqLen) for any write or exclusive access until it observes a WrSizedCndlDR from the same originator and unit. A completer may have limited ability to track a certain number of cache lines and may not track this request. If the request is not tracked, then the WrSizedCndlDR will fail as indicated in the following section. RdSizedDWX is always a coherent request and will issue invalidating probes to ensure that no cache may update the line. RdSizedDWX may not cross a 64B boundary. For more information on the exclusive lock request, refer to Exclusive Access Requests.

Implementation Note: For SOC-15 ports, non-caching reads are restricted to a maximum of 256 bytes in length and must not cross a 256-byte boundary except for RdSizedDWX which is further restricted to a maximum of 64 bytes in length and must not cross a 64-byte boundary.

3.6.2 Writes

The non-caching write commands include the following commands:

- WrSized
- WrSizedNC
- WrSizedFull
- WrSizedFullNC
- WrSizedFullZero
- WrSizedFullComp
- WrSizedCndlDR

Originators perform non-caching writes to write a line without necessarily having cached the line prior to the write. Data is sent on an originator data channel with valid bytes indicated by the OrigDataBytEn bits. In general, non-caching write requests may be of any size, although there are restrictions on some variants.

The completer generates a write response when the write has been made visible to all caching agents (if a coherent version has been used) and the write data has been globally ordered at the completer so that all operations issued after the write response, even from other SDP ports, observe this write and/or appear to occur after this write. An originator does not provide a response acknowledgement for non-caching writes.

The NC flavors are non-coherent and do not probe other originators. The non-NC flavors are coherent and will cause the write data to be merged with any dirty data detected by the probes.

The Full variants provide an early hint that all bytes of the data are valid and all of the OrigDataBytEn bits will be set. The Full variants must have a size that is an integer multiple of the data beat size. WrSizedFullZero and WrSizedFullComp are exactly the same as WrSizedFull except for the compression of the data.

WrSizedCndlDR is part of the Exclusive Lock Request Feature and may only be issued if ReqIO=0 and ReqSpecialAddr=0 and if this originator and unit has previously issue a RdSizedDWX to the same cache line address. The write is performed conditionally and occurs only if the completer is (still) tracking the cache line address due to a prior RdSizedDWX from the same originator and unit. A completer stops tracking the RdSizedDWX address if it sees another atomic, write or a request to obtain exclusive access before the WrSizedCndlDR is observed. In addition, a completer may stop or never initiate tracking an address based on resource restrictions that are outside the scope of this specification including the potential that another originator or unit may be currently tracking this same address.

Whether a WrSizedCndlDR actually performed the store is indicated by a read response that either contains all zeros on the addressed bytes (success) or all ones on the address bytes (failure) along with a RdRspStatus of OKAY. Any error response on the WrSizedCndlDR also indicates that the write did not succeed but for a reason other than the exclusive access was lost. The originator may still use OrigDataBytEn to write any number of bytes within the ReqAddr/ReqLen span.

For more information on the exclusive lock request, refer to Exclusive Access Requests.

Implementation Note: For SOC-15 ports, writes are restricted to a maximum of 64 bytes in length and must not cross a 64-byte boundary. A WrSizedCndlDR must not cross a 64-byte boundary even for non SOC-15 ports.

3.6.3 Atomics

The atomics include the following commands:

- Atomic
- AtomicNC
- AtomicNR
- AtomicNRNC

Originators issue atomic commands to perform atomic (at the completer) read-modify-write operations. A number of different arithmetic and logical operations (**OpType**) are defined. In addition to the operand data located in memory at the request address, depending on OpType, the operation may require one operand (single-operand) called "Op1", or it may require two operands (dual-operand) called "Op1" and "Op2". The operand data is provided on an Originator Data Channel with byte masks to indicate which memory locations are operated upon and which bytes of the operand data are valid.

The operation may act on a single element (scalar) or it may be vector, where multiple elements are acted upon with one single SDP command. Dual-operand atomics must always be vectors although it is possible using byte masks to operate only on a single element in the vector. The size of each element (**OpSize**) is indicated by the originator and can be 32 bits (one doubleword) or 64 bits (two doublewords).

The Atomic and AtomicNC commands return the original value of the addressed location, while the AtomicNR and AtomicNRNC commands do not. The Atomic and AtomicNR commands specify that dirty data found in probed caches be merged with clean data found in memory. The *NC versions are non-coherent which do not require probing other originator caches.

For single-operand commands, the request size in bytes is a maximum of 64 and must be an integer multiple of the operand size. For dual-operand commands, the request size must be 64 bytes, indicating the transfer size of the operand data; only 32 bytes of memory data located at the request address are affected.

Single-operand commands must have a request address that is aligned to the operand size. Dual-operand commands must have a request address that is 32-byte aligned. Atomic operations cannot span a cache line. All OrigDataBytEn bits corresponding to a n-bit element must have the same value; that is, an entire location of the size and alignment given by OpSize is either being operated on or not.

For single-operand operations, when the specified request size is larger than the operand size, the data supplied on an Originator Data Channel and the memory location operated upon are vectors with the n-bit operands packed into an (N * n)-bit vector data type. The specified operation is applied to each n-bit element of the vector independently. The byte masks specify whether or not each element is operated on. The byte masks may be sparse - it is not required that sequential elements are acted upon. The size of the write data and the optional read response data is the same. The originator must ignore returned data on any element that was not acted upon.

Architecture Note: Address, length and byte masks for single-operand commands act just like addresses, lengths and byte masks for a write that is also aligned to the operand size.

For dual-operand operations, Op1 data is provided in bytes 0-31 of the originator write data and Op2 data is provided in bytes 32-63 of the originator write data. OrigDataBytEn specifies which elements are being operated upon. The byte masks may be sparse - it is not required that sequential elements are acted upon. It is legal to operate on only one element and this element does not need to occupy the least significant bytes of OrigData. OrigDataBytEn for the first 32-bytes of data must match OrigDataBytEn for the second 32-bytes of data. For purposes of generating OrigDataOffset when transmitting the data, the second operand acts as if it's memory address is immediately following the memory data being acted upon - OrigDataOffset for the second operand does not reset to zero and continues to increment.

Support of the atomic commands by any completer is optional. A completer that does not support an atomic request returns a SLVERR response.

Floating point atomic support includes two encodings for FP16 and BFLOAT-16 addition. In these commands, the OpSize is indicated as 32-bit which means the operation occurs on ***both*** 16-bit values within each 32-bit element of the Atomic command. There is no support for individual 16-bit elements.

The operation to be performed (**OpType**) is specified using either ReqAttr[7:4] of ReqAttr[0,7:4] of the Originator Request Channel. ReqAttr[0] (**OpTypeExt**) is reserved when the Floating Point Atomic Feature is not implemented. All arithmetic operations shown are unsigned and do not saturate. Unlisted encodings are *reserved*. The floating point atomic encodings are *reserved* when the Floating Point Atomic Feature is not implemented. Table 41 defines each type and specifies the result to be written to memory.

Table 41 Atomic Operations

Operation	**OpTypeExt, OpType**	**New Mem[Addr] Value**	**OpSize Restrictions**
Test[0] and OR	0,0000b (00h)	Mem[Addr][0] ? (Mem[Addr] | Op1) : Mem[Addr]	00b or 01b
Unconditional Swap	0,0001b (01h)	Op1	00b or 01b
Add	0,0010b (02h)	Mem[Addr] + Op1	00b or 01b
Subtract	0,0011b (03h)	Mem[Addr] - Op1	00b or 01b
Reserved	0,0100b (04h)		
AND	0,0101b (05h)	Mem[Addr] & Op1	00b or 01b
OR	0,0110b (06h)	Mem[Addr] | Op1	00b or 01b
XOR	0,0111b (07h)	Mem[Addr] ^ Op1	00b or 01b
Signed Min	0,1000b (08h)	(Mem[Addr] > Op1) ? Op1 : Mem[Addr]	00b or 01b

Signed Max	0,1001b (09h)	(Mem[Addr] < Op1) ? Op1 : Mem[Addr]	00b or 01b
Unsigned Min	0,1010b (0Ah)	(Mem[Addr] > Op1) ? Op1 : Mem[Addr]	00b or 01b
Unsigned Max	0,1011b (0Bh)	(Mem[Addr] < Op1) ? Op1 : Mem[Addr]	00b or 01b
Unsigned Increment With Limit[1]	0,1100b (0Ch)	(Mem[Addr] >= Op1) ? 0 : (Mem[Addr] + 1)	00b or 01b
Unsigned Decrement With Limit[1]	0,1101b (0Dh)	(Mem[Addr] == 0 || Mem[Addr] > Op1) ? Op1 : (Mem[Addr] – 1)	00b or 01b
Compare and Swap	0,1110b (0Eh)	(Mem[Addr] == Op1) ? Op2 : Mem[Addr]	00b or 01b
Test and Modify	0,1111b (0Fh)	Mem[Addr][0] ? (Mem[Addr] || Op1) & Op2 : Mem[Addr]	00b or 01b
Unsigned Subtract Without Wrap	1,0000b (10h)	(Mem[Addr] >= Op1) ? (Mem[Addr] – Op1) : 0	00b or 01b
Unsigned Conditional Subtract	1,0001b (11h)	(Mem[Addr] >= Op1) ? (Mem[Addr] – Op1) : Mem[Addr]	00b or 01b
Floating Point Min	1,0010b (12h)	(Mem[Addr] > Op1) ? Op1 : Mem[Addr]	00b or 01b
Floating Point Max	1,0011b (13h)	(Mem[Addr] < Op1) ? Op1 : Mem[Addr]	00b or 01b
Floating Point Add	1,0100b (14h)	Mem[Addr] + Op1	00b or 01b
Floating Point Add - FP16	1,0101b (15h)	Mem[Addr] + Op1	00b
Floating Point Add – BFLOAT16	1,0110b (16h)	Mem[Addr] + Op1	00b
Reserved	17h-1Fh		
Note 1: "Unsigned Increment With Limit" and "Unsigned Decrement With Limit" were previously known as "Unsigned Clamping Increment" and "Unsigned Clamping Decrement". They were renamed to clarify the intent of Op1 as a limit and to avoid confusion with "Unsigned Subtract Without Wrap". An example unsigned increment with a limit of 3 would have a series of "0, 1, 2, 3, 0, 1, 2, 3, …"			

The operand size OpSize is encoded in ReqAttr[3:2] field as shown in Table 42 Atomic OpSize Encoding.

Table 42 Atomic OpSize Encoding

OpSize	**Operand Size**
00b	32 bits
01b	64 bits
10b	*Reserved*
11b	*Reserved*

Atomic Dual-Operand Vector Example Operation

- ReqAddr[4:0] = 00000b
- ReqLen = 0Fh (16 doublewords)
- OpType (ReqAttr[7:4]) = 1111b (Test and Modify)
- OpSize (ReqAttr[3:2]) = 00b (32 bits, 1 doubleword)
- **Architecture Note:** The right column labeled "beat" is also the value on OrigDataOffset.

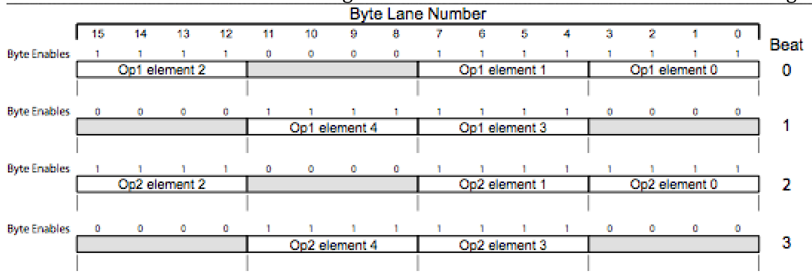

Figure 17 Example Originator Data Transfer for a Dual-Operand Vector Atomic

Atomic Single-Operand Vector Example Operation

- ReqAddr[4:0] = 01000b
- ReqLen = 5h (6 doublewords)
- OpType (ReqAttr[7:4]) = 0010b (Add)
- OpSize (ReqAttr[3:2]) = 01b (64 bits, 2 doublewords)

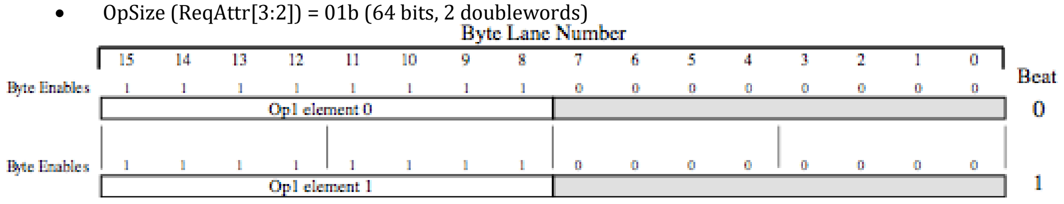

Figure 18 Example Originator Data Transfer for a Single-Operand Vector Atomic

3.6.4 Caching Reads

The caching reads include the following commands:

1) RdBlkL
2) RdBlkS
3) RdBlkC
4) RdBlkX

All the caching reads must normally be issued with a ReqLen equivalent to the size of one cache line, and return one cache line of data to the requester. The different variants indicate the desired cache state at the originator. All variants require a response acknowledge except for RdBlkS when issued by an originator that can use the old value of a line if the read collides with an invalidating probe.

Some implementations allow a caching read to be issued for multiple cache lines provided that:

1) The port implements the Multiple Cache Line Feature, and the implemented port feature allows up to the requested number of cache lines.
2) The number of cache lines is a power-of-two (i.e. one, two, four, or eight cache lines).
3) ReqAddr must be aligned to the number of cache lines requested when accessing more than one cache line. As an example, if ReqLen=0x1F (128 bytes), then ReqAddr[6:0] must be zero. Critical byte requesting is not supported unless only a single cache line is accessed.

RdBlkL (load) will allow the originator to go to the highest access permission state consistent with the state of other caches. RdBlkS indicates that the originator is only looking for sharing (and not forwarding) state, and requires the fabric to drive the system State field in the response to either S or O. RdBlkC (clean) indicates the originator is looking for a clean install state. RdBlkX (exclusive) indicates a requirement to invalidate other caches so it can be installed in an exclusive state. If RdBlkX is issued as a result of a store, the final state of the cache line is usually M, however it may also be issued speculatively, and the final state of the cache line could also be E or D.

The cache line to be accessed is specified by the ReqAddr. When ReqLen specifies a single cache line, bits [log2(cache line size)-1:0] of the address are ignored for addressing purposes but specify the critical byte that is requested to be returned first, if possible.

When multiple cache lines are requested in one command, the ordering that the cache line data is returned is not required to be in address order as long as each individual cache line is returned in the specified order.

3.6.5 Cache Upgrades

The cache upgrade commands include:

- ChgToX
- ChgToXNR
- ValBlk

The cache upgrade commands are used to get write access to a line without fetching data. They all require response acknowledgements. All cache upgrade commands must be issued with a ReqLen equivalent to the size of one cache line.

Cache upgrade commands may be used to obtain write access for lines that are present in the originator cache in a readable state. The commands may also be used to obtain write permission for lines that are not currently present in the cache (Invalid). With ValBlk, the originator ***commits*** to write the entire cache line and does not care about any current read data, allowing it to transition to the M state at the conclusion of the transaction. Any dirty data in other caches, at the point the command is ordered by the completer, will be discarded. With ChgToX and ChgToXNR, the originator does not commit to transition to the M state. Since ChgToXNR does not provide data, any dirty data in other caches may be committed to memory. It is implementation specific whether or not ChgToX commits dirty data in other caches to memory or provides the dirty data to the originator in the read response.

ChgToX, ChgToXNR, and ValBlk all use the read response bus for completion.

ChgToXNR and ValBlk never provides data on the read response bus and only uses RdRspStatus=OKAY_NODATA with RdRspDataStatus=NODATA. PassDirty is never asserted on a completion for ChgToXNR or ValBlk.

ChgToX may optionally provide data (with or without PassDirty asserted). Upon completion of a ChgToX, the completer indicates whether it will provide data by using RdRspStatus of OKAY (data must be provided) or RdRspStatus of OKAY_NODATA (data must not be provided). It is possible to receive data even if the cache still contains a copy of the line, and it is also possible that the completer does not return data for a ChgToX that has had the cache line probed out prior to the completion. The completer of a ChgToX must always provide data when it asserts RdRspStatus=OKAY. When the completer asserts RdRspStatus=OKAY_NODATA, it may not assert PassDirty and RdRspDataStatus of NODATA must be observed (aligned with the data if there is a read response data lag).

When the cache line is already present prior to a ChgToX* command, a race condition exists where another transaction may probe out the originator's copy of the data prior to the completion of the ChgToX*. In this case with ChgToX, the data may be provided by the read response completion but it is not guaranteed.

When ChgToX* completes without the completer returning data and without a copy of the line already or still present in the originator's cache, the line is then in a writeable state without any valid readable data (empty Mp). As not all caching agents support the Mp state, the originator also has the option of going to the I state, with or without a VicBlkCln. If the current contents of the line are desired, a RdBlkX must be issued to fetch the read data.

AckCancel on a ChgToX* command is asserted if and only if the cache line is invalid (I) just prior to receiving the response. Originators may not cancel a ValBlk request and must not assert AckCancel.

The cache line is specified by the ReqAddr. Bits [log2(cache line size)-1:0] of the address are ignored for addressing purposes. For ChgToX, these bits specify the critical byte that is requested to be returned first, if possible.

3.6.6 Cache Eviction Commands

The cache eviction commands include:

- VicBlkFull
- VicBlkFullZero
- VicBlkFullComp
- VicBlkPtl
- VicBlkCln
- VicBlkClnD

The cache eviction commands are used to indicate to the system the eviction or downgrading in cache state of a cache line in an originator cache. They never generate probes. VicBlkFull, VicBlkFullZero, VicBlkFullComp, VicBlkPtl and VicBlkClnD require response acknowledgements, a response acknowledgement is optional on VicBlkCln. All are responded to on the write response bus. The originator may or may not plan to retain the line, as indicated by the command's ReqAttr field. When it plans to retain the line, this is referred to as a downgrade. When it does not plan to retain the line, this is referred to as an eviction. The planned cache line state indicated in ReqAttr may differ from the final cache line state, but only as allowed by legal silent downgrades (see Internally-initiated Cache Line State Transitions) or intervening probes. It is not necessary to indicate another command if the planned cache state does not match the final cache state.

The VicBlkFull* and VicBlkPtl commands indicate that the originator is writing back an either fully (VicBlkFull*) or partially (VicBlkPtl) modified line to system memory. Until the transaction completion is received, the originator must retain the line and continue to respond to probes as if the eviction had not been issued. The originator must set AckCancel on the response acknowledge if and only if another probe has passed the dirty data to another originator (the requesting originator no longer has "write responsibility" for the dirty data). A VicBlkFull* requires that all OrigDataBytEn bits for the line are set, while a VicBlkPtl allows any combination of OrigDataBytEn bits to be set.

The VicBlkCln* commands are used to indicate eviction or downgrade of a clean line. They are only useful in certain systems, so originators may support configuration options to enable generating them as opposed to doing clean evictions silently. The VicBlkCln command indicates just the fact of the eviction, and would normally be used in systems with probe filters or memory directories that are tracking originator cache state. A response acknowledgement is required only for some probe filters use cases when the VicBlkCln indicates (using ReqAttr) that it is maintaining a copy of the line ("rinsing" VicBlkCln), for example going from F->S or E->S. If a response acknowledgement is used, AckCancel is set if and only if the originator no longer has the line due to intervening probe activity, and had indicated on the original request that it was retaining a copy. The VicBlkClnD command includes the data being evicted as well, even though it is clean, and would normally be used in systems with downstream victim caches. A response acknowledgement is required. AckCancel for VicBlkClnD indicates that the line originally was either exclusive (E) or forward (F) and is no longer either exclusive (E) or forward (F) due to intervening probes.

When a response acknowledgement is used, the eviction/downgrade is not performed until the response is received (or as allowed by legal silent downgrades). However, even when performing a downgrade to exclusive, the originator may not perform a write to the line between the time that the eviction data has been committed and the write response has been received. Any probes received between the request and the completion must override the planned retained cache state. When a fetch or share probe is received while a VicBlkFull* downgrade to exclusive is in process, the final cache state must be forward (F), shared (S) or invalid (I) since these transitions would normally take an originator to owned (O) if the downgrade was not in process.

When a response acknowledgement is not used (for VicBlkCln), the eviction/downgrade must be complete before the request is sent, such that any probes that are received prior to the completion will observe the cache state as indicated by the VicBlkCln (or as allowed by legal silent downgrades).

VicBlkFullZero and VicBlkFullComp differ from VicBlkFull only in the compression of data.

The evicted cache line is specified by the ReqAddr. Bits [log2(cache line size)-1:0] of the address are ignored.

See Signaling Evictions for more information on these commands.

3.6.7 Barrier Commands

The barrier commands include:

- Fence
- Flush

Barrier commands are used to indicate ordering requirements between otherwise unordered transactions. They never transfer data or require a response acknowledgment, and are responded to on the write response bus. Fence and Flush have no address associated with the command (ReqAddr field is ignored). ReqBlockLevel must be nonzero. Fence and Flush can be issued in any virtual channel.

See Barriers for more information on Fence and Flush ordering.

3.6.8 Cache Maintenance Commands

The cache maintenance commands include:

- ClnBlkAll
- WbInvBlkAll
- InvBlkAll

An originator issues cache maintenance commands to force downgrades to a cache line throughout the system. The cache line to be operated upon is specified by the ReqAddr. Bits [log2(cache line size)-1:0] of the address are ignored. These requests do not transfer data to or from the originator. Accordingly, they use the write response bus and do not require a response acknowledgment.

The ClnBlkAll command direct all caches with modified data to write a copy of the line back to memory and retain the line in a clean (E, F, or S) state. The *InvBlkAll commands invalidate the specified line in all caches. WbInvBlkAll causes any modified data to be written back to memory, while InvBlkAll causes modified data to be discarded.

3.6.9 Address Translation Services

This section covers the following commands:

- AddrXlateRdSz
- AddrXlateWrSz

These commands are used to provide a virtual address to physical address service in the SOC. An IP that uses address translation services is obligated to implement both an originator (to request address translations) and a completer (to receive address invalidations).

Each command has a subcommand encoded in ReqAttr[2:0]. All subcommand encodings not specified in this section are reserved.

Address Translation

Using AddrXlateRdSz and ReqAttr[2:0] = 000b, the originator provides a virtual address on ReqAddr (along with ReqVirtAddr=1) to be translated. ReqLen specifies the length of up to eight page table entries times the width of the read response data bus.

The completer indicates the success or failure of the translation in RdRspStatus. When the translation was successful (RdRspStatus=OKAY), the completer provides up to eight page tables, but it is not obligated to return all requested page tables. At any time, the completer may truncate the response to provide less than the requested length by asserting RdRspLast.

When the translation was not successful (RdRspStatus!=OKAY), the data returned should be ignored (except that parity may be checked as parity is always valid whenever data transfer is provided). The completer may also truncate the read response data isomg RdRspLast after providing at least one beat of data.

The format of the SOC page table entries, including the status to indicate that the virtual address could not be translated is outside the scope of this specification.

The originator may cache this virtual to physical address translation.

Address Invalidation

Using AddrXlateWrSz and ReqAttr[2:0] = 000b, the originator (which is the IP that provides the address translation services) informs the completer (which is an IP that requests address translations and caches them) that a prior virtual address to physical address mapping is no longer effective. The virtual address to be invalidated is provided in ReqAddr (along with ReqVirtAddr=1) and additional information is provided in the accompanying data. The ReqLen field specifies the length of the accompanying data. This command may be sent to multiple address translations, so it is possible to receive this command for addresses that have not previously been translated by this specific IP.

Apart from the ReqAddr, additional metadata on the invalidation request (as examples - a size, address space ID, or whether or not current transactions must be flushed prior to the write response) is provided in the data that accompanies the request. The length and encoding of the data is outside the scope of this specification.

A response from the address invalidation command is only provided after the address is removed from any cached translation lookaside buffers and any transactions that must be flushed have received a completion.

3.6.10 Other Commands

This section covers the following commands:

- Extension
- ExtensionD
- QosControl
- WrNoDataNC
- ErrEvent
- Cancel
- SpecDramRd/SpecPrefCmd

Extension and ExtensionD

These commands are reserved for future extensions to the command field or to carry extra request fields.

QosControl

Update the priority of an existing request, or all requests in a virtual channel, when there is no other request that can be sent. This command consumes no flow control credits and has no response.

WrNoDataNC

Used to initiate commands that only carry information encoded in the address. These are typically system management commands.

ErrEvent

The originator has detected an error that is uncorrectable that is unrelated to any transaction. Such an error is normally system fatal. Hardware should take steps to preserve any pertinent error capture data. Sending this command consumes no flow control credits and has no response. Originators should avoid "loops" in error reporting. An ErrEvent notification received from one SDP port must not generate an ErrEvent notification to another SDP port that is associated with the same IP or fabric, nor should an SYSFATALERR on a write response status cause an ErrEvent command.

The encoding of ReqAttr during ErrEvent signifies the severity and/or type of error observed. At present, only one event – signifying a system wide fatal error – is supported. This command is not intended for errors that can be isolated to a component and recovered using an IP specific reset or other recovery.

Cancel

A response to a read command sent to the completer is no longer needed. If possible, the completer is directed to cancel the read command (identified based on its ReqTag) and complete the transaction by returning a one-beat read response with RdRspDataStatus of NODATA (00b). The completer does not send a response for the cancel command itself. If the completer cannot cancel the identified read request, it drops the cancel command and returns the requested data. This command does not consume a flow control credit. The Cancel command can only be used on completer ports. The ReqCmd field encoding used for Cancel is *reserved* for originator ports.

SpecDramRd/SpecPrefCmd

The command encoding is known as SpecDramRd when ReqAttr=00h. When ReqAttr is not 00h, a more generic prefetch command is available. ReqIO and ReqSpecialAddr must be zero for this command.

An implementation may place restrictions on legal encodings of ReqLen and ReqAddr alignment, for example by requiring them to be cache line aligned and cache line in length.

SpecDramRd (ReqAttr=00h): An originator uses SpecDramRd to indicate that it is likely that a read will occur to the specified cache line address (although not necessarily with the same critical byte address). This command consumes a flow control credit and has no response. There is no method to cancel a SpecDramRd. The ReqUnitID and ReqTag of this command is not used since there is no response.

SpecPrefCmd (ReqAttr != 00h): An originator may use SpecPrefCmd to specify a prefetch, and may optionally place the data into a completer cache to be retrieved at a later time (ReqAttr[5:4]=10b). When the data is placed in the completer cache, the originator may request the data to be provided or have a response (OKAY_NODATA) when the prefetch is complete. The use of a response is specified in ReqAttr[7:6] as shown in Table 15 SpecPrefCmd ReqAttr Usage.

Once data is placed into the completer cache, the originator may request a retrieval using SpecPrefCmd and ReqAttr[5:4]=01b. It is illegal to retrieve data that has not been previously prefetched and allocated by a prior SpecPrefCmd, and that allocation command must have been completed before the retrieval command is specified. When performing a retrieval, it is required that ReqAttr[7:6] indicate a response with data (11b).

Use of the completer cache features requires the originator to know the organization and amount of cache reserved for this originator. If the data cannot be prefetched or retrieved and a response is provided, the completer may respond with a TRANSERR. One example would be overflowing the allocated space.

These commands are unordered with respect to prior writes.

Architecture Note: SpecPrefCmd with ReqAttr=C0h is effectively the same as a RdSized command (except for read-after-write ordering). This type of usage is not supported. The use of "read response with data" encoding is intended solely for cases when the data is being prefetched into or retrieved from the completer cache.

3.7 Ordering

The ordering of two outstanding requests only applies to the ordering that the requests are completed in the fabric with respect to other requests. No guarantee is made that the order in which the responses are observed in the SDP port correlates to the order that the requests were completed in the fabric, even when the two requests were guaranteed ordered. No guarantee is made of ordering by two outstanding requests from different SDP ports, even if the originators know that the requests were issued in a specific order.

Any request ordering requirements that are not guaranteed by SDP must be enforced either by the originator by waiting for a response to the prior requests before issuing an ordered request, or through the use of barrier commands. Once a completion has been returned, that request has always been ordered and, in the case of coherent operations and system management commands, any side effects have been observed at other ports. A second request that is submitted after a response has been received for a first request will observe the results of the first request. This holds even for requests submitted from different SDP ports.

Requests in the same virtual channel that are considered unordered may be reordered, but there is no guarantee of the ability to pass.

3.7.1 Request Ordering

Caching Requests

Caching read (e.g. RdBlkS), speculative DRAM read (SpecDramRd), speculative prefetch (SpecPrefCmd), cache upgrade (e.g. ChgToX), cache maintenance (e.g. WbInvBlkAll), and cache eviction (e.g. VicBlkFull) commands are unordered across SDP and the fabric, even for two requests from the same source and with the same address. Caching requests must be issued with ReqPassPW field set to 1 and ReqBlockLevel set to 00b (if either signals are present on the interface).

Due to the unordered nature of the caching requests, an originator is required to limit outstanding requests to the same cache line address from all units behind the port so that there is never more than one of the following commands: RdBlkC, RdBlkL, RdBlkX, ChgToX, ChgToXNR, ValBlk, VicBlkCln, VicBlkClnD, VicBlkFull VicBlkPtl, and WrSized* (WrSized, WrSizedFull, WrSizedFullZero, WrSizedFullComp) with ReqAttr[2:1] = 11b (shared) or Reqattr[2:1] = 10b (known invalid). For these commands, once an originator has initiated one such request, it may not submit another such request to the same cache line address until it has received the last beat of the final response and committed to send the response acknowledgement (if any). It is permissible for the second request to pass the first response acknowledgement provided that no dependencies are made between the two. For originators or commands that do not use the Response Acknowledge channel, a second request may not be initiated until after the response for the first request has been received.

As an exception to the above rules, an originator is allowed to have simultaneous WrSized* commands with either ReqAttr[2:1]=11b (shared) or ReqAttr[2:1] = 10b (known invalid) as these are guaranteed ordered (write-after-write). In addition, an originator that requires the Ordered Cacheable Command feature is also allowed to have one or more RdBlkS commands simultaneous with one or more WrSized* commands, as this feature guarantees read-after-write ordering.

Provided that the originator is not dependent on the order in which they complete, there is no limit (other than limits due to the number of outstanding tags) on the number of outstanding requests for commands not listed in the prior paragraph (including cache maintenance operations, RdBlkS, and SpecDramRd/SpecPrefCmd); even if the addresses overlap with other outstanding commands of either type.

Architecture Note: RdBlkS commands are not required to be serialized since they do not have the potential to receive exclusive or modified data.

Non-Caching Requests

The following section defines ordering of non-caching requests include RdSized, RdSizedNC, RdSizedDW, RdSizedNoWriter, WrSized, WrSizedFull, WrSizedFullZero, WrSizedFullComp, WrSizedNC, WrSizedFullNC, WrNoDataNC, and the Atomics before it's completion is returned.

SDP supports two flavors of ordering for non-caching requests. Same-address ordering only applies to requests to the same address and is enabled when ReqBlockLevel is 00b or not present. Write-ordering applies to requests regardless of the address and is enabled when ReqBlockLevel is not 00b. In addition, SDP supports two barrier commands to enforce ordering beyond the two mechanisms.

Same-Address Ordering

SDP supports same-address ordering between two (or more) outstanding requests provided that the requests are all issued with ReqBlockLevel set to 0, in the same virtual channel, same ReqPassPW value, and that the later requests do not have a different quality of service than the earlier requests. Same-address ordering is enabled by setting ReqBlockLevel to 0. If ReqBlockLevel is not-present, same-address ordering is always enabled.

SDP guarantees write-after-write ordering and read-after-write ordering. That is, a read will observe the value of a previous valid write to the same address and a second valid write will not pass a first valid write to the same address. Reads are ***not*** ordered with respect to other reads nor are they ordered with respect to writes that are submitted after the read (read-after-read ordering and write-after-read ordering is not guaranteed). Reads or writes that complete with an error response are not necessarily ordered with respect to other reads or writes that complete without an error response. The statement that read-after-read ordering and write-after-read ordering is not guaranteed does not mean that it is illegal for an originator to issue a second transaction prior to the response from the first transaction – the completer must support the possibility. However, the completer may perform the two operations in any order.

Requests that are submitted with a ReqBlockLevel set to zero are unordered with non-barrier requests submitted with non-zero ReqBlockLevel values, or requests in other virtual channels (except as noted for posted and non-posted channel relationships when ReqPassPW=0), or different ReqQosPriority values. An originator may use QoS priority escalation to maintain same-address ordering when elevating the priority of a request stream.

Same Destination Ordering in System Management Spaces

In some system management spaces, the ordering model guarantees "same-destination" regardless of the actual address or offset within the space. When this form of ordering model is guaranteed, SDP guarantees that write-ordering and read-after-write ordering is ensured such that a later request (read or write) may not pass a previous write as long as the two requests are in the same virtual channel, same ReqQosPriority, within the same system address space, and have the same final destination.

The same-destination ordering model is ensured for:

1) Command Processor Region (for CP communication)
2) Doorbell Region
3) All APIC Ranges
4) Distributed Virtual Memory Region
5) Bus Lock Region
6) DVM/Bus Lock Protocol Region

For APIC ranges, the ordering ensures that all APIC registers for the same processor unit are handled at the same destination. No ordering exists between APIC accesses made by separate processors (i.e. with different UnitIDs).

For the DVM/Bus Lock Protocol region, the ordering is across accesses within this region, as well as with accesses made to the Distributed Virtual Memory and Bus Lock regions. For example, a SysMgmtJoin is not allowed to pass a prior SysMgmLeave, a Distributed Virtual Memory. All accesses within these three regions have the "same destination", so the ordering is guaranteed across all addresses within the region.

Since this ordering model requires the final destination to be the same, command processor region and doorbell regions only guarantee two requests are in order if the request is made to the same CP or the same doorbell client.

Write-Ordering

SDP supports write ordering guarantees between outstanding requests from the same source in the same virtual channel and same ReqQoSPriority values. The term "write" (in the write-ordering section only) refers to the flavors of WrSized* and WrNoDataNC, and does not include Atomics.

Write-ordering guarantees that a later request must not pass a previous write. That is, a later read must push all previous writes and a later write is not made globally visible until all previous writes have been made globally visible regardless of address. The second request is said to be blocked behind the write.

For write-ordering to be guaranteed, both requests must be from the same source as defined by ReqBlockLevel, the second blocked request (but not necessarily the write) must be issued with ReqPassPW set to 0, and both requests must be in the same virtual channel. Requests issued in the non-posted virtual channel with ReqPassPW set to 0 are ordered behind requests in the posted virtual channel even though they are not the same virtual channel.

Within the fabric, ReqPassPW is meaningless for requests that are not in the posted or non-posted channel. However, it may be used to pass the value to the completer port.

SpecPrefCmd

The SpecPrefCmd is not ordered with respect to other SpecPrefCmd, nor is it ordered with respect to all other reads or writes.

3.7.2 Barriers

Barriers are used to create an ordering relationship between requests that were issued prior to the barrier (earlier) and/or requests that are issued after the barrier (later). There are two barriers – Flush and Fence.

Fence

Fence is used to indicate ordering between otherwise unordered requests within a virtual channel. Requests issued before the Fence are guaranteed to complete before any requests issued after the Fence.

ReqBlockLevel is used, as with other requests, to control whether Fence applies across UnitIDs or StreamIDs. A ReqBlockLevel of 00b is illegal. The address field is not valid for Fence and is ignored.

The response for Fence is returned on a Write Response Channel, since a Fence completion returns no data and indicates that all requests prior to the Fence have completed. Note that for posted requests, this may indicate only that the request has reached its target in the fabric, not its final destination.

Flush

Flush is used to push prior requests, including both reads and writes, to their completion point within the fabric.

Flush can be restricted to pushing requests of a particular UnitID or stream, based on the ReqBlockLevel field. A ReqBlockLevel of 00b is illegal. As with Fence, the address field associated with a Flush is ignored.

The response for Flush is returned on a Write Response Channel, since a Flush completion returns no data. The Flush completion is not returned until all earlier requests have received their responses. Note that for posted requests, this may indicate only that the request has reached its target in the fabric, not its final destination.

3.7.3 Response Ordering

A non-caching read or write request in the non-posted virtual channel with ReqRspPassPW field set to 0 specifies that the response must block behind posted writes that are flowing in the same direction as the response. When the response comes back across the port interface, the completer clears RdRspPassPW or WrRspPassPW to indicate that this response must block behind posted writes. There is no equivalent of ReqBlockLevel for responses; the blockage always applies across all UnitID and StreamID values. Since posted requests can only be received by completers and responses can only be received by originators, RdRspPassPW and WrRspPassPW is only meaningful for interfaces that have both originator and completer ports.

Write responses do not usually push posted requests, ReqRspPassPW on write requests should normally be 1, but it may be cleared for special cases (such as configuration or I/O port writes) where write responses are required to push the posted write channel. ReqRspPassPW is meaningless for any request in any channel other than the non-posted virtual channel. Whenever an interface either does not have both originator and completer ports, or does not require this blocking on the response, or issues a request that is not in the non-posted virtual channel, ReqRspPassPW must be set to 1. This indicates that the response to the request is not ordered with respect to write commands in the posted virtual channel. **Architecture Note:** Not setting ReqRspPassPW to 1 for posted requests may lead to deadlock.

No support is provided to keep responses in order with respect to other responses. The order in which responses are received does not indicate the order in which the corresponding requests were actually completed nor does it indicate the order in which the result were made visible to other ports.

3.8 Locks and Exclusive Access Requests

3.8.1 Bus Locks

Bus locks provide a mechanism for an originator unit to complete a series of transactions (usually a read and a write) atomically by preventing other originator units in the system from submitting any new requests for the duration of the lock.

An originator unit acquires a lock by issuing a lock request system management message on its SDP interface, which includes a global originator unit identifier (assigned through a means outside the scope of this document). This message is received and processed by the lock serialization point within the fabric. The lock serialization point broadcasts the lock request to this and all other originator units in the system through the Probe Request Channel. After receiving a lock request, originator units must stop initiating new transactions and wait for all outstanding transactions to complete. After this has occurred, each originator unit sends a lock grant system management message and may not initiate any more transactions until it has received a lock release message. When the lock serialization point has received acknowledgments from all originator units, it informs the original requester that the lock has been granted through a Probe Request Channel lock granted message. The global originator unit identifier from the lock request is repeated on this message to communicate which unit has been granted the lock.

The unit that has been granted the lock can then issue a series of operations to memory or I/O and know that they will be completed atomically. It is required that all reads issued by the lock owner must be RdSizedNoWriter. This command may be used by the completer to identify the atomic addresses. Furthermore, the use of this command ensures that no other caches in the system have write access to the line.

After these operations have completed, the originator unit then issues a request to unlock the fabric. The lock serialization point broadcasts the release to this and all other originators through a Probe Request Channel. Only after receiving a lock release system management message can any originator unit (including the unit that released the lock) resume normal unlocked operation.

To accommodate originators with a variable number of originator units participating in the bus lock protocol (for example, due to low-power states), each unit that participates in bus locks must specifically join the protocol prior to initiating other relevant requests and must leave the protocol before going into a low-power state. When an originator unit joins the protocol, the current value of the bus lock state is delivered in the join message.

It is outside the scope of this specification which originators (fabric ports) and commands are subject to the bus lock. For some originators, the fabric may block some or all commands during the period another originator has been granted the lock and for other originators or other command types (e.g. realtime clients), the fabric may continue to process commands even while the bus is locked by another port.

As a result, it is noted that coherency probes are legal during the bus-lock period and may be observed by originators that have sent all lock grant messages or even originators that have been granted the lock. An originator must continue to accept and process probes. Normally, it would require a software error for the agents that are not subject to bus lock to perform any transactions that may cause probes to the address(es) that are part of the atomic read-modify-write operation for which the bus lock was granted, but the originator is required to maintain forward progress and coherency under these exigent conditions.

For more information, refer to System Messages.

3.8.2 Exclusive Access Requests

When the Exclusive Lock Request Feature is implemented, SDP provides two commands – RdSizedDWX and WrSizedCndlDR. These commands allow the user to implement arbitrary locked operations to memory (ReqIO and ReqSpecialAddr must both be zero) using the following sequence:

1) The originator issues a RdSizedDWX with a ReqAddr and ReqLen covering the byte addresses to be locked.
2) The completer invalidates all cached copies of the address.
3) The completer, subject to space and availability, starts tracking the address.
4) Any atomic, write, or cacheable request for exclusive access will cause the completer to stop tracking the address and removing it from its tracker.
5) The originator performs the operation internally to the originator.
6) The originator issues a WrSizedCndlDR with the same ReqUnitID, same ReqAddr, and same ReqLen as specified on the RdSizedDWX.
7) The completer, upon receipt of the WrSizedCndlDR observes if the address range is being tracked and that the same originator and unit requested this line using a RdSizedDWX and no intervening stores may have occurred.
8) If the completer indicates that the line has not been modified since this RdSizedDWX, it will complete the store and return a data status of all zeros.
9) If the completer indicates that the line is not being tracked, indicating that either the originator did not issue a RdSizedDWX, the completer could not track the address or that another originator has possibly stored into the line after the RdSizedDWX, the store is not completed and the completer will return a data status of all ones.
10) Any error during RdSizedDWX or WrSizedCndlDR gives a RdRspStatus that is not OKAY. In these error scenarios, the tracking is not set up during a RdSizedDWX and the store is not completed. It is presumed that the originator will retry the operation, including reissuing another RdSizedDWX.

In addition to having resource restrictions on the number of addresses and originators that may be tracked at a single time, the completer is also not required to track the precise block address. Completers may treat any store to any byte in the cache line block regardless of which bytes are actually altered during the WrSizedCndlDR.

It is considered an error for an originator to issue a WrSizedCndlDR without having issued a prior RdSizedDWX with the same ReqAddr, ReqLen, and ReqUnitID. However, since the completer is not able to accurately determine if the RdSizedDWX was never issued, was issued without tracking space available in the completer, or an intervening store caused the completer to remove this RdSizedDWX from the exclusive request tracking structure, no error status is used to indicate this potential issue.

3.9 Coherency

This topic discusses the cache state model supported by the SDP coherency protocol.

A cache line is a fixed-size block of system memory that starts at an address aligned to its size (this is also referred to as natural alignment). The size of this block is fixed for a given system architecture. While RdSized* and WrSized* requests can span more than one cache line, all probes operate on a single cache line.

3.9.1 Coherency Control Point

A coherency control point (CCP) is an agent in the system that maintains system-level memory coherency by managing the cache line state of cached copies of system memory cache blocks. There are many different methods that a CCP can employ to maintain system memory coherency. These methods may be classified broadly as:

- Distributed Directory
- Central Directory
- Probe Filter

In the distributed directory method, each cache maintains a local directory that tracks the state of every cache block that it holds; there is no centralized directory. In this case the coherency control point maintains memory coherency by broadcasting probes to all the caches in the system whenever a request is issued by a processor that requires a change in state for a cache line that others may hold. The control point must wait for a response from each cache before sending data to the requester.

On the opposite extreme, memory coherency can be maintained by providing a central directory that tracks which caches have a copy of each memory block and the state of that cached block. In this case, when a processor issues a request, the CCP uses the directory to determine if any cache has a up-to-date copy of the requested block. If so, it can direct that cache (or any one of the plurality of caches that hold a copy) to send that line to the requester and then send a probe to each cache in the system that has a copy of the cache block, if that copy must be invalidated. The problem with this method is that it requires a very large directory and does not scale well as system memory size and number of caches grow.

Most systems use a combination of distributed and centralized directories. In addition to the directories maintained by each cache, the system maintains a non-comprehensive directory that is used to track the state of a significant number of the most recently requested cache lines and the locations of any copies. This method is called a probe filter because it is used to reduce (but not eliminate) the need to broadcast probes.

To work effectively, the probe filter benefits from receiving information about significant cache line state transitions in the caches. Some of this information can be obtained by observing the caching read requests that each originator emits. Cache line state transitions that are caused by internal actions, such as internal sharing of a line that was in a writeable state, are not observable unless the originator sends out a message that informs the CCP of these transitions.

SDP supports the implementation of a system probe filter by providing protocol commands that allow an originator to indicate certain cache line state transitions caused by internal actions that would not otherwise be seen by the CCP.

3.9.2 Cache Coherency States

The SDP architecture supports a cache coherency model that employs eight cache line states defined in Table 43 SDP Supported Cache Line States. A particular cache may not support all of these states.

Table 43 SDP Supported Cache Line States

Cache State	**Meaning**
Invalid (I)	The cache does not contain any valid data for this line.
Shared (S)	The cache line contains valid data for the full line. It may or may not match memory, but if it does not, one other cache must have the line in the O state. Other caches may have a read-only copy of the line in the S state.
Exclusive (E)	Cache has valid data for this line that matches the contents of memory and the right to modify the line. All other caches are in the I state for this line.
Fowarder (F)	Cache has valid data for the full line that matches the contents of memory. Other caches may have a copy in the S state. If a probe requesting data hits on a line in the F state, this cache should respond rather than other caches that might also have a read-only copy of the line.
Modified (M)	Cache has valid data for this line that is newer than memory and the right to modify the line. All other caches are in the I state for this line. For caches that implement the Dirty state, the Modified state is when the line has been locally modified.
Modified/ Partial (Mp)	Cache has valid data for 0 or more bytes in this line that are newer than memory, and the right to modify the line. All other caches are in the I state for this line.
Dirty (D)	Cache has valid data for the full line that is newer than memory; the data was received from another cache and not modified locally. The cache has the right to modify the line. All other caches are in the I state for this line.
Owned (O)	Cache has valid data for the full line that is newer than memory. Other caches may have a copy of the line in the S state.
Implementation Note: SOC-15 data fabric does not support the Modified/Partial (Mp) state.	

If multiple caches have copies of the same line, the data must match. The above states may be thought of in terms of 3 attributes: read access, write access, and responsibility to return data on a probe. A fourth attribute, locally written, distinguishes M from D, but is not generally visible to the cache coherency protocol. The attributes then map to the cache states as shown in Table 44.

Table 44 Cache Line State Attributes

	Cache State

Attribute	I	S	F	E	O	Mp	M or D
Read access (may read all or part of the line)	No	Yes	Yes	Yes	Yes	Yes/No[3]	Yes
Write access (may modify the line)	No	No	No	Yes	No	Yes	Yes
Responsible to return data on probes	No	No	If requested[2]	If requested[2]	Yes[1]	Yes/No[1,4]	Yes[1]
Notes: 1. A line in O, Mp, M or D must return data if probed unless PrbRD=00. 2. A line in F or E will normally return/forward the (clean) data if probed with PrbRD = 1x. 3. A line in Mp may read those bytes that have been written. 4. A line in Mp with no bytes written does not have any data to return.							

3.9.3 Cache Line State Transitions Due to Requests

When an originator issues a command, the completion of the transaction can cause a line state transition in its local cache. The fabric may not assume that the transition has occurred until it sees the Ack for the command. The final state may also depend on probes that were received between the command and the response, which may also result in the setting of the AckCancel bit.

Table 45 Originator Request Cache State Transitions and Table 46 show the legal combinations of commands, initial cache states from which they may be generated, and final states in the system and originator. This table assumes a non-error response to a command – errors such as SLVERR, DECERR, PROTVIOL, and TRANSERR as examples are not expected to alter the state of the cache at all. If a particular initial state is not listed for a command, it is illegal to generate that command from that state. Commands that receive read responses use the response to determine final state in the originator. RdRspStatus[State] uses the same encodings as PrbRspStatus[State] – see Probed Cache State Transitions and Responses for details.

Commands that use the write response bus do not receive system State or PassDirty in the response, so those fields are marked as — in the table. If a particular response status is not listed, it is an illegal status. In general, the read response status is expected to be generated by ORing together the probe response status from the various caching agents probed, (Either at most one state is non-I, in which case ORing them returns the value of that state, or one is O or F and some number of others S, in which case ORing them returns O or F.) but the fabric may also affect it, for instance by clearing PassDirty when writing dirty data back to memory. In some cases, multiple final states are possible, in which case the originator implementation may choose from among the listed options.

For most commands, the originator is not issued a probe and the consolidated system command state returned on the response does not include the cache(s) of the originator. When Table 45 indicates the initial state of the originator may be “any”, the originator may be probed unless the completer can predict the possible cache state using a directory of some sort. Since there may be other SDP originators in the system and because the ProbeTag does not have a correlation to the ReqTag, normally an originator cannot tell if a probe is issued on behalf of its request or another request. An originator may not depend on receiving a probe for a command.

Table 45 Originator Request Cache State Transitions for Commands Other Than ChgToX*

Command(s)	Initial State at Orig	Final State at Probed Caches	{Final State, PassDirty}	State at Orig After Response	Notes
RdSized, RdSizedDW	Any	Unchanged	xx0	Unchanged	A RdSized* command can be issued regardless of the state of the corresponding cache line. The originator may be probed.
RdSizedNC	Any	—	—	Unchanged	
RdSizedNoWriter	Any	I, S, F, O	xx0	Unchanged	The originator may be probed.
WrSizedNC, WrSizedFullNC	Any	—	—	Unchanged	
WrSized, WrSizedFull, WrSizedFullZero, WrSizedFullComp (ReqAttr[2] = 0b)	Any	I	—	I	The originator may be probed.
WrSized, WrSizedFull, WrSizedFullZero, WrSizedFullComp (ReqAttr[2:1] = 10b)	I	I	—	I	
WrSized, WrSizedFull, WrSizedFullZero, WrSizedFullComp (ReqAttr[2:1] = 11b)	Any except I	I	—	S	The originator is not probed.
WrSizedFull (ReqAttr[2:1] = 11b	I	I	—	S	Only a full cache line write may allocate in S.

Atomic, AtomicNR	I	I	000	I	
AtomicNC, AtomicNRNC	I	—	—	I	
RdBlkL	I, S	I	000	E	RdBlkL starting from S is only possible when one unit has the line in S and another requests it from I.
		I	001	M, D	Originators that do not support D use M.
		F, O	110	S	
		S	010	F	
		S	011	O	
RdBlkS	I, S	S, O	110 or 010	S	Fabric must force system state to O or S if all probe status is I. RdRspState will always be forced to S on SDP regardless of system state returned from all probed caches. RdBlkS starting from S is only possible when one unit has the line in S and another unit requests it from I.
RdBlkC	I, S	I	000	E	
		F, O	110	S	
		S	010	F	
RdBlkX	I, S, F	I	000	E	
		I	001	M, D	If the read was issued by a prefetch instead of a due to a store, the cache will install it in D.
	Mp	I	000	E, M, D	E is only permitted if the line was invalidated by a probe and ownership transferred out or if the Mp state had no valid byte enables set in the first place. If not, the line must go M (D for a prefetch) and the incoming data be merged with the written bytes in the cache.
		I	001	M, D	PassDirty may only be set if line was invalidated by a probe.
ValBlk	I, S, F, O	I	—	M	An originator using ValBlk must store into the full cache line.
VicBlkPtl	Mp	—	—	I	If line is passed dirty due to a probe before response, the originator must set AckCancel.
VicBlkFull* (ReqAttr[1:0]=00b)	O, M, D	—	—	I	
VicBlkFull* (ReqAttr[1:0]=01b)	O, M, D	—	—	S	
VicBlkFull* (ReqAttr[1:0]=10b)	M, D	—	—	E	
VicBlkFull* (ReqAttr[1:0]=11b)	O, M, D	—	—	F	
VicBlkCln, VicBlkClnD (ReqAttr[1:0]=00b)	S, F, E	—	—	I	If E/F is S or I by the completion, the originator must set AckCancel in some conditions.
VicBlkCln, VicBlkClnD (ReqAttr[1:0]=01b)	S, F, E	—	—	S	
InvBlkAll, WbInvBlkAll	Any	I	—	I	The originator may be probed. When WbInvBlkAll targets a non-coherent I/O address, the originator must invalidate the cache state as probes are not issued by non-coherent I/O devices.
ClnBlkAll	Any	I, S, F, E	—	I, S, F, E	The originator may be probed.
SpecPrefCmd	Any	—	—	Unchanged	

Table 46 Originator Request Cache State Transitions for ChgToX* Commands

Command	Initial State at Orig	State at Orig Prior to Response (Note 2)	Final State at Probed Caches	{Final State, PassDirty}	State at Orig After Response (Note 3)	Notes
ChgToX	I	I	I	000 without returned data (OKAY-NODATA)	Mp	See note 4. **AckCancel=1.**
					I	The originator may choose to remain in I. AckCancel=1.
				000 with returned data (OKAY)	E	The completer may optionally return clean data on a ChgToX. The originator is now E. **AckCancel=1**
				001 (OKAY)	$D^{\text{Note 1}}$	**AckCancel=1.**
	S, F	S, F		000 (OKAY or OKAY-NODATA)	E	AckCancel=0. The completer may or may not return clean data. If returned, the data matches the current copy of the line that the originator has.
				001 (OKAY)	$D^{\text{Note 1}}$	AckCancel=0.
		I		000 without returned data (OKAY-NODATA)	Mp	See note 4. **AckCancel=1**.
					I	The originator may choose to remain in I. AckCancel=1.
				000 with returned data (OKAY)	E	The completer may optionally return clean data on a ChgToX. The originator is now E. AckCancel=1.
				001 (OKAY)	$D^{\text{Note 1}}$	AckCancel=1.
	O	O		000 without returned data (OKAY-NODATA)	M	AckCancel=0.
				000 with returned data (OKAY)		AckCancel=0. The completer may optionally return clean data on a ChgToX. The originator must discard this data as the Owned copy in the originator is the most recent copy of the cache line.
		S, F		000 (OKAY or OKAY-NODATA)	E	AckCancel=0. The completer may or may not return clean data. If returned, the data matches the current copy of the line that the originator has.
		I		000 without returned data (OKAY-NODATA)	Mp	See note 4. **AckCancel=1**.
					I	The originator may choose to remain in I. AckCancel=1.
				000 with returned data (OKAY)	E	The completer may optionally return clean data on a ChgToX. The originator is now E. AckCancel=1.
				001 (OKAY)	$D^{\text{Note 1}}$	AckCancel=1.
ChgToXNR	I	I		000	Mp	See note 4. **AckCancel=0 (since the originator never had the line).**
					I	The originator may choose to remain in I. AckCancel=1.
	S, F	S, F			E	AckCancel=0.
		I			Mp	See note 4. **AckCancel=1**.
					I	The originator may choose to remain in I. AckCancel=1.
	O	O			M	AckCancel=0.
		S, F			E	AckCancel=0.
		I			Mp	See note 4. **AckCancel=1**.
					I	The originator may choose to remain in I. AckCancel=1.

Notes:
1) When the state at the originator is marked "D", any originator that does not support the Dirty state would use M.
2) The state at the originator should only change between the initial state (prior to the request) and the state just prior to the response due to an intervening probe.
3) The state at the originator after the response does not take into account any silent transitions that might not normally take place. In many cases, the originator might store into the line and transition to M (or Mp if supported). However, the originator is not required to complete the store.
4) In the rows where Mp is listed as a final state, this is a state where the originator has exclusive privileges for writing but does not have the data. This is the Mp state (without any byte enables set). In many cases, the originator might then write some of the bytes (staying in Mp but setting some by te enables), or all bytes of the cache line (a silent transition from Mp to M).
5) AckCancel settings that are in bold are under consideration for change in a future specification.

Originators that do not support the E/M states (write-through originators) must generate RdBlkC or RdBlkS in lieu of RdBlkL.

The allowed mapping of requests to probes is shown in Table 47 Probes Issued in Response to Commands. The fabric or probed cache may choose to take a more restrictive action or, by using a probe filter directory, may issue a less restrictive probe if at all. An originator may not rely on the expected PrbRD or PrbAction as long as the completer uses a set of encodings that provides a similarly proper coherency transition.

Table 47 Probes Issued in Response to Commands

Command	**Expected Probe**	**Expected PrbRD**
All non-coherent commands (RdSizedNC, WrSizedNC, WrSizedFullNC, AtomicNC)	—	—
RdSized, RdSizedDW	NOP	Any
RdSizedNoWriter	Share or Invalidate (note 1)	!00b
RdSizedDWX	Invalidate	01b
Atomic, AtomicNR	Invalidate	01b
WrSized* without ReqAttr[Steering]=1	Invalidate	01b
WrSizedFull* without ReqAttr[Steering]=1	Invalidate	00b
WrSized* with Steering Tag	Store	01b
WrSizedFull* with Steering Tag	Store	00b
RdBlkL	Migrate	!00b
RdBlkC	Fetch	!00b
RdBlkS	Share	!00b
RdBlkX	Invalidate	!00b
ChgToX, ChgToXNR	Invalidate	!00b
ValBlk	Invalidate	00b
VicBlkPtl, VicBlkFull, VicBlkFullZero, VicBlkFullComp	—	—
VicBlkCln, VicBlkClnD	—	—
InvBlkAll	Invalidate	00b
WbInvBlkAll	Invalidate	01b
ClnBlkAll	Clean	01b
SpecDramRd	—	—
—	Demote (note 2)	Must be 00b
DvmOp (WrSized to special address)	A pair of DVM probes chained together.	

Notes:
1) RdSizedNoWriter does not issue a NOP probe. This allows an originator to use this command to perform a read-modify-write while under a bus lock. It is implementation dependent on whether or not it uses a Share or Invalidate probe.
2) Demote probes are issued by internal fabric commands and not associated with a request command. Only a PrbRD of 00b (no return data) is allowed.

3.9.4 Probed Cache State Transitions and Responses

When a cache is probed, state changes occur based solely on the value of the PrbAction field and the current state of the cache. In some cases, the probed cache has multiple legal options. The probed cache indicates its final state to the system via the ProbeRspStatus[State] field. A probed cache may also return data as indicated by its state and the encoding in the PrbRD field (for some combinations of state and PrbRD, data transfer is optional). Any time dirty data is provided and writeback responsibility is given up, the PrbRspStatus[PassDirty] must be set.

The allowed combinations of states and responses for each type of probe are shown in Table 48 Probed Cache State Transitions. When more than one final state is listed, the bolded state is the "preferred" state transition.

Table 48 Probed Cache State Transitions

PrbAction	Initial State	Data Transfer	Final State PrbRspStatus[4:3]	PassDirty PrbRspStatus[2]
NOP	I	0	I (00)	0
	S	Allowed if RD=11	S (01)	0
	F	Allowed if RD=1x	**F (11)**, S (01)	If Final=F
	E	Allowed if RD=1x	**E (10)**, S (01)	0
	O	If RD!=00	O (11)	0
	M, D	If RD!=00	**M, D (10)**	0
	Mp	If RD!=00	Mp (10)	0
Share, Fetch, Migrate	I	0	I (00)	0
	S	Allowed if RD=11	S (01)	0
Share	F, E	Allowed if RD=1x	**F (11)**, S (01)	If Final=F
Fetch, Migrate	F, E	Allowed if RD=1x	S (01)	0
Share, Fetch	O, M, D	1	O (11)	0
Migrate	O	1	**S (01)**	1
	D	1	**S (01)**, I (00)	1
	M	1	S (01), **I (00)**	1
Share, Fetch, Migrate	Mp	1	I (00)	1
Invalidate, Store	I	0	I (00)	0
	S	Allowed if RD=11	Invalidated (10)	0
	F, E	Allowed if RD=1x	Invalidated (10)	0
	O, M, D, Mp	If RD!=00	Invalidated (10)	If RD!=00
Clean	I	0	I (00)	0
	S	Allowed if RD=11	S (01)	0
	F	Allowed if RD=1x	**F (11),** S (01)	If Final=F
	E	Allowed if RD=1x	**E (10)**, S (01)	0
	O	1	**F (11)**, S (01)	1
	M, D	1	**E (10)**, S (01)	1
	Mp	1	I (00)	1
Demote	I	None (PrbRD must be zero for Demote)	Unchanged, I (00)	0
	S		Unchanged, S (01)	0
	F		**F (11),** S (01)	If Final=F
	E		**F (11)**, S (01)	If Final=F
	O		Unchanged, O (11)	0
	M, D, Mp		Unchanged (10)	0
Rinse	I	0	Unchanged, I (00)	0
	S	Allowed if RD=11	Unchanged, S (01)	0
	F	Allowed if RD=1x	**F (11)**, S (01)	If Final=F
	E	Allowed if RD=1x	**F (11)**, S (01)	If Final=F

	O, M, D	1	**F (11)**, S (01)	If Final=F
	Mp	1	I	0
Note: PrbRD=00b is not used for all cases when the coherency state requires a downgrade of dirty data as this would constitute data loss. For the purposes of this table, the PrbRD!=0 is assumed when a 1 is in the "Data Transfer" column. The fabric/completer may use PrbRD=00b on Invalidate/Store probes only if the entire line is committed to be overwritten and the write is already ordered.				

Note that three different probe encodings (Share, Fetch, and Migrate) alias similar sets of allowed transitions. The multiple encodings are provided to indicate what the desired transitions are for the "F, E" and "O, M, D" rows. The ***desired*** transitions are shown in Table 49 Preferred Probed Cache State Transitions for Share, Fetch, Migrate (with data transfer and PassDirty rules as given above).

Table 49 Preferred Probed Cache State Transitions for Share, Fetch, Migrate

PrbAction	Initial State	Final State
Share	O, M, D	O
	F, E	F
Fetch	O, M, D	O
	F, E	S
Migrate	F, E, O, D	S
	M	I

Since silent transitions from the E, D, and Mp states to the M state are always possible, they are not distinguished in the returned PrbRspStatus[State] field. Accordingly, that field is encoded as shown in Table 50 Encoding of PrbRspStatus[State,PassDirty].

Table 50 Encoding of PrbRspStatus[State,PassDirty]

PrbRspStatus[State, PassDirty]	Probe Type	Final State
00x	All except Invalidate, Store	I
000	Invalidate, Store	I (prior state I)
01x	All	S
10x	All except Invalidate, Store	X = E, M, D, Mp
10x	Invalidate, Store	I (prior state not I)
110	All	O
111	All	F

3.9.5 Internally-initiated Cache Line State Transitions

In addition to cache line state transitions caused by completed SDP commands or probes, cache line state transitions may occur due to activity internal to an originator without any indication on the SDP interface. These cache line state transitions are called silent transitions.

Unless specifically disallowed by the following rules, any silent cache line state transition is allowed:

- An originator with an invalid cache state must first perform an SDP command to gain read or write access before any cache state transition is allowed.
- An originator without write access may not modify a line or move to a state that has write access without first performing an SDP command to gain write access.
- An originator without the responsibility to forward data may not silently move to a state where it may forward data.
- An originator may silently give up read access at any time or it may silently drop the responsibility to forward data. When configured to signal clean evictions, the originator may send a VicBlkCln* message for these state transitions, but it is not required to do so under all conditions.
- An originator with write access may silently modify all or part of a line, creating the responsibility to write back this modified data (or provide it in a probe with PassDirty) in the future. Since the originator already has write access, an SDP message is not required to signal the modification.
- An originator may silently give up write access at any time, provided that the contents of the line is not newer than the memory. This includes the possibility of Mp with no byte enables set. When doing so, it may or may not maintain read access and it may or may not drop the responsibility to forward data. When configured to signal clean evictions, the originator may send a VicBlkCln* message for these state transitions, but it is not required to do so under all scenarios.
- An originator may not silently move from any state to an owned state.
- An originator with a cache line that is newer than the memory may not silently transition from that cache state to any cache state which does not properly reflect the modified state including which bytes have been modified. A VicBlkFull* or VicBlkPtl message is always required to make these transitions.
- An originator with only partially modified bytes of a cache line (Mp) may not move to any state that requires the full cache line (O, E, F, or S) unless it first modifies the entire line.

- In general, an originator that has modified a line should not move to the D (dirty unwritten) state.

3.9.6 Signaling Evictions

An originator that signals an eviction (VicBlkFull* or VicBlkPtl) for a modified or dirty line and does not block probes based on these evictions architecturally performs the cache state transition to the new cache state (as specified by the ReqAttr field) only after receiving the write response message and before sending the response acknowledge message. Until the VicBlk* completes, it must provide the most recent copy of the cache line to all outstanding probes. However, this intervening probe activity and/or other legal silent cache state transitions may also cause the originator to have changed the cache line state so that it no longer matches what was specified in the original ReqAttr field. In these cases, the originator is not required to perform another SDP message, even if the final cache state does not match what was originally encoded in the ReqAttr field. It is required to set AckCancel on the response acknowledge if intervening probe activity passed the dirty line.

An originator that blocks probes that are received during an eviction architecturally performs the cache state transition immediately after the SDP message is committed to be sent on the non-blocking writeback virtual channel.

An originator that is configured to signal clean evictions (VicBlkCln or VicBlkClnD) for these state transitions, and is using a response acknowledge on these, may perform the cache state transition before the message is sent on the SDP interface or at any time before the response for this message is received. However, it must have completed the cache state transition before the response acknowledge message is sent. During the period that the VicBlkCln message is pending and/or in progress, the originator may have performed legal silent transitions and the final state of the cache line may no longer be the same as the cache state indicated in the message. In this case, it is not required to perform another SDP message. For VicBlkClnD, it is required to set AckCancel if the line was E or F before the request and is no longer E or F by the time the transaction completes due to intervening probe activity. For VicBlkCln, it is required to set AckCancel if it had stated that it was maintaining a copy of the line but then it does not have a copy due to intervening probe activity.

3.9.7 Probe and Response Interlocks for Single Cache Line Address

The fabric (completer-proxy) ensures that the originator never observes two outstanding probes for the same cache line address.

The fabric (completer-proxy) ensures that any response that will be acknowledged by the originator is not provided while there is a probe outstanding at the originator for the same cache line address as the response, nor will it send a probe for the same cache line address between the response (the first response if it is EARLY or RETRANSMIT) and the acknowledgement.

3.9.8 Storing Probes

Storing probes (PrbAction = 0111b) are used to invalidate a cache line and request the originator to install (refetch) the same line. After the line is invalidated, the originator may use this recently de-allocated cache line or it may allocate a new cache line. Depending on this allocation and other state internal to the originator, the originator ***may*** refetch the line using a new RdBlkL request. On this request, the originator should set ReqAttr[1:0] = 10b to indicate that the RdBlkL is refetching a storing probe. Apart from the mandatory invalidation, all storing probe information, including which cache receives the refetched line, may be treated as a hint and may be ignored. The completer may not depend on this refetch happening, however, if the originator does submit the request; the completer is able to match the RdBlkL address to the request that caused the storing probe.

In certain race conditions, the completer may not have the cache line buffered after a storing probe refetch reaches the memory probe. In these cases, the completer may respond to a RdBlkL with ReqAttr[1:0] with a RdRspStatus of OKAY_NODATA and then provide RdRspDataStatus of NODATA. Only RdBlkL commands that are issued after the originator has observed a storing probe should set the ReqAttr indication, and only these commands may be completed with OKAY_NODATA.

Storing probes are expected to only be used in systems with a cache line size of at least 64 bytes, so PrbAddr[5:2] are redefined to indicate storing information as shown in Table 51 Storing Probe PrbAddr Usage.

Table 51 Storing Probe PrbAddr Usage

Probe Address	Meaning
PrbAddr[5]	When 1b, the originator is directed to perform a RdBlkL if the cache state (prior to the storing probe) is F, E, M, Mp, D, or O ("focus"). When 0b, the originator may perform a RdBlkL independent of the cache state.
PrbAddr[4]	When 0b, PrbAddr[3:2] are reserved and the originator should attempt to direct the RdBlkL response to either the L3 cache or a core L2 that is selected by the originator.
PrbAddr[3:2]	When PrbAddr[4] is 1b, the originator is should attempt to direct the RdBlkL response data to an L2 that is identified using the physical core L2 identifier in PrbAddr[3:2]. When PrbAddr[4] is 0b, this field is *reserved*.

3.10 System Messages

System messages are used to perform a number of system-wide functions that require communication / coordination between originators and between an originator and a system device within the fabric such as an interrupt controller or a bus lock serialization point.

System messages are used for system messages I/O interrupts, inter-processor interrupts, bus locks, and distributed virtual memory operations (DVM operations).

System messages are routed by the fabric. System message requests can be initiated by originators by requests to special addresses. Messages from system devices can be delivered to coherent originators on a Probe Request Channel(s) or to completers on an Originator Request Channel(s) (with data possibly provided on the Originator Write Data Channels). System messages that are delivered to both coherent originators on a Probe Request Channel(s) and to completers on an Originator Request Channel(s) use a separate virtual channel on the request to avoid deadlock.

In addition to the use for system messages, the system address range is also used for access to save areas and address spaces such as the PCI I/O and PCI configuration space. This allows an originator to access these address ranges without needing to know an SOC-specific address. These address ranges are defined in this section but the accesses are not considered system messages. The accesses follow the same normal flow for all SDP commands, except there may be restrictions on the ReqLen and ReqCmd used for the access as indicated in each section.

3.10.1 Decoding of System Message Types

Prior to revision 1.3.1 of this specification, the system message space was decoded as the last 12GB of the system memory space. In addition to limiting the amount of address space available to system memory space, this also required the system memory space to have a 12GB region that was not available for use as DRAM or MMIO. A system management function was recognized by the completer as part of decoding the address.

To accommodate a larger system message area, the “System Management Access” feature was introduced to separate the system memory space (ReqSpecialAddr=0) from addresses that are used SOC specific purposes (ReqSpecialAddr=1) as described in this section. For backwards compatibility, SDP ports that are compliant with revision 1.3.2 and any later specifications still support decoding of the address to recognize a system management access when the System Management Access feature is not implemented. However, these ports can only access the legacy 12GB of this space. Use of the System Management Access feature is required to access the doorbell and command processor communication.

3.10.2 System Message Types

System messages follow the normal flow for SDP requests and/or probes except the address is a system unique range at the top of physical address space minus 12 GB. The specification assumes a 64-bit address size. When a port implements fewer address bits (“n”), the most significant bit is considered “sign-extended” only if bits [n:36] are all ones and bits 35:34 are 01b, 10b, or 11b to create a 64-bit address for system management address matching purposes. Use of the system address space requires use of limited commands, virtual channel (ReqVC), and/or data size (ReqLen) for predictable operation. In many cases, address bits are used as portions of the system message. All address bits not defined are reserved. For DVM operations, and when Probe Interrupt Delivery Feature version 2 is implemented, some system management messages in the Probe Request Channel(s) are delivered where the address is not in the system unique range at the top of physical address space minus 12 GB. Whenever the PrbAction is 1000b, the PrbAddr field is always interpreted as a DVM operation and not as a memory address. Likewise, whenever the PrbAction is 1001b or 1011b, the PrbAddr field is always interpreted as a system message and not as a memory address.

Command Processor Area

The system management area reserves a region for “command processor” communication across multiple disparate command processors. The encoding of this region is outside the scope of this specification.

Table 52 Command Processor Area

Address Range	0000_0001_0000_0000h – 0000_0001_00FF_FFFFh
Address Size	64 KB, used space is system and IP specific.
Usage	Command Processor communication
Originator/Completer	Originators and completers.
Commands	RdSizedNC, WrSizedNC, WrNoDataC
Probes	None.

Doorbell Region

SDP originators may send doorbells on the fabric across IPs. This doorbell region differs from the “interrupt region” by being a decoded address space based off of the MMIO region for graphics doorbells. Bits 33:28 of the address decodes the IP that is being accessed. Use of the space and meaning of the IPs and doorbell offset is system dependent. To access this feature, the System Management Access feature must be implemented and ReqSpecialAddr=1 is required to be asserted.

Table 53 Doorbell Area

Address Range	0000_0200_0000_0000h – 0000_0203_FFFF_FFFFh
Address Size	16 GB,
ReqAddr[33:28]	Doorbell/client type Value / Doorbell type: 0x0 – IH 0x3 – CP 0x4 – VCN0 0x6 – RLC 0x7 – VCN1 0xC – VCN2 0xD – VCN3 0xE – AIDi.SDMA0 0x8 – AIDi.SDMA1 0x9 – AIDi.SDMA2 0xA – AIDi.SDMA3
ReqAddr[27:12]	Physical ASID ID (PASID)
ReqAddr[11:2]	Doorbell offset
Usage	Communication of graphics doorbells across system.
Originator/Completer	Originators and completers.
Commands	WrSizedNC **Note**: This region is "write-only".
Probes	None.

System DMA (SDMA) Area

The system management area reserves a region for SDMA communication between the SDMA front-end and back-end. The encoding of this region is outside the scope of this specification.

Table 54 System DMA (SDMA) Area

Address Range	0000_0001_0100_0000h – 0000_0001_01FF_FFFFh
Address Size	16 MB, used space is system and IP specific.
Usage	SDMA communication between its front-end and back-end.
Originator/Completer	Originators and completers.
Commands	RdSizedNC, WrSizedNC, WrNoDataC
Probes	None.

IP Save Areas

The system address range provides access to three ranges of protected memory space for IP use – CPU, security processor (PSP), and system management controller (SMU) private areas. The originator accesses these ranges with reads and writes as if it was non-coherent system memory (subject to address and size alignment requirements).

Table 55 CPU Save State Area

Address Range 1	FFFF_FFFD_F700_0000h – FFFF_FFFD_F7FF_FFFFh
Address Size for range 1	16 MB, used space is system and IP specific. Each processor thread may use up to 64KB, allowing up to 256 threads behind a single SDP port.
Address Range 2	FFFF_FFFF_0000_0000h – FFFF_FFFF_7FFF_FFFFh
Address Size for range2	2 GB, used space is system and IP specific. Each processor thread may use up to 64KB, allowing up to 32K threads.
Usage	CPU save area, address layout is system and IP specific. Address range 1 is the address range expected to be used by originators. The second address range is reserved for internal data fabric remapping to allow multiple SDP ports that may independently use 16MB of addresses between FFFF_FFFD_F700_0000h – FFFF_FFFD_F7FF_FFFFh into a single consolidated memory range. The use of address range 2 is outside the scope of this specification.
Originator/Completer	Originators only.
Commands	Non-caching reads, writes, RdBlkS, RdBlkC, RdBlkL, RdBlkX, ChgToX*, ValBlk, WbInvBlkAll, SpecDramRd, VicBlkCln, and VicBlkFull*. **Note**: This region is ***not*** system cache coherent, even when accessed with RdBlk*. Each individual cache line is expected to be accessed and cached only by a single core.
Probes	None.

Table 56 Security Processor (PSP) Private Area

Address Range	FFFF_FFFD_FB00_0000h – FFFF_FFFD_FB7F_FFFFh.
Address Size	8 MB, used space is system and IP specific.
Usage	Security processor save area, address layout is system and IP specific.
Originator/Completer	Originators only.
Commands	Non-caching reads and writes.
Probes	None.

Table 57 System Management Controller (SMU) Private Area

Address Range	FFFF_FFFD_FB80_0000h – FFFF_FFFD_FBFF_FFFFh.
Address Size	8 MB, used space is system and IP specific.
Usage	System Management Controller private area, address layout is system and IP specific.
Originator/Completer	Originators only.
Commands	Non-caching reads and writes.
Probes	None.

Interrupt Region

SDP originators may send interrupts into the data fabric and receive “end-of-interrupt” (EOI) and “interrupt acknowledge” (IACK) messages using three interrupt regions. Except for EOI, a response is not generated until the interrupt has been delivered. The fabric may also send interrupts to an originator that is a core using a fourth interrupt region.

In addition, when Probe Interrupt Delivery Feature is implemented, SDP originators may also receive interrupts through one of two address spaces. Table 62 Interrupt Vector Region is used for Probe Interrupt Delivery Feature version 1 and Table 63 Interrupt Delivery Region for Probe Interrupt Delivery version 2 is used for Probe Interrupt Delivery Feature version 2. Note that delivery version 2 does not use an address range that is in the system address map (of 2^{PA}-12GB), but are unique from a memory address by having a PrbAction of 1001b or 1011b.

Implementation Note: Address and data payload definitions are different from non-coherent HyperTransport™.

Table 58 Interrupt Region

Address Range	FFFF_FFFD_F800_0000h - FFFF_FFFD_F8FF_FFF0h, except as overlapped with the EOI region.
Address Size	16 MB.
ReqAddr[23:16]	Vector[7:0].
ReqAddr[15:10]	Reserved.
ReqAddr [9]	Trigger Mode (0=edge, 1=level).
ReqAddr [8]	Destination Mode (0=physical, 1=logical).
ReqAddr [7:4]	Message Type. See Table 59.
Usage	Initiation of an interrupt by SDP originator. Global visibility of the interrupt is not guaranteed by write completion, however any write ordering rules include interrupt delivery.
Originator/Completer	Originators only.
Commands	WrSizedNC, quadword aligned access only.
Length	Two doublewords. All byte enables must be set.
Data[63:40]	Reserved (zeros).
Data[39:32]	Message Signaled Interrupt ExtData[7:0].
Data[31:0]	Interrupt destination. Encoding is system dependent.
Probes	None.
Architecture Note: SDP data movement always takes the selected data in the natural byte lane of OrigData and interrupt region is no exception to this. In this table, Data[63:0] is picked from OrigData[n+63:n], where n is the byte lane specified by ReqAddr[m:0], even though the ReqAddr field contains message type and other interrupt parameters.	

Table 59 Message Type Interrupt Encoding

Message Type	MT[3:0][2]	TM (edge=0/ level=1)	Vector	DM (physical=0 /logical=1)	Destination
Fixed Interrupt	0000b	0/1	0-FFh	0/1	0-maximum per SOC[1] (directed), FFFFFFFFh (broadcast)
Lowest Priority (Arbitrated) Interrupt	0001b	0/1	0-FFh	0/1	0-maximum per SOC[1] (directed), FFFFFFFFh (broadcast)
SMI	0010b	0	0	0	0-maximum per SOC[1] (directed), FFFFFFFFh (broadcast)
AVIC Doorbell	0010b	0	1	0	0-maximum per SOC[1]
NMI	0100b	0	0	0	0-maximum per SOC[1] (directed), FFFFFFFFh (broadcast)
INIT	0101b	0	0	0	0-maximum per SOC[1] (directed), FFFFFFFFh (broadcast)
LVT-Performance Monitor	0110b	0	0	0	Reserved, must be zero. The interrupt is directed back to the originating agent ("self-interrupt") as determined by ReqUnitID. Reserved unless Probe Interrupt Delivery version 2 is implemented.
LVT-IBS	0110b	0	1	0	Reserved, must be zero. The interrupt is directed back to the originating agent ("self-interrupt") as determined by ReqUnitID. Reserved unless Probe Interrupt Delivery version 2 is implemented.
External Interrupt	0111b	0	0	0	0-maximum per SOC[1] (directed), FFFFFFFFh (broadcast)
GIC SPI assert	1000b	0	0-FFh	0	FFFF_FFFFh
GIC SPI deassert	1001b	0	0-FFh	0	FFFF_FFFFh
LINT1 (Legacy PIC NMI)	1011b	0	0	0	FFFFFFFFh (broadcast)
LINT0 (Legacy PIC ExtInt)	1110b	0	0	0	FFFFFFFFh (broadcast)
EOI	1111b	N/A (0)	0-FFh	N/A (0)	N/A (0). See Table 60.

Notes:

1) The maximum value for destinations is SOC-dependent.

2) All other values for MT are reserved. All other combinations of TM, Vector, DM and/or destination are reserved.

Table 60 End of Interrupt (EOI) Range

Address Range	FFFF_FFFD_F800_00F0h - FFFF_FFFD_F8FF_00F0h.
Address Size	N/A (overlaps with Interrupt Region).
ReqAddr [23:16]	Vector[7:0].
ReqAddr [7:4]	Message Type = 1111b (EOI).
Usage	Signals an end of interrupt. Completion message may precede the delivery of the end of interrupt (posted).
Originator/Completer	Completers only.
Commands	WrNoDataNC.
Probes	None.

Table 61 Interrupt Acknowledge Region

Address Range	FFFF_FFFD_F900_0000h - FFFF_FFFD_F900_0003h.

Address Size	1 doubleword.
Usage	Interrupt Acknowledge, returns the vector of the last legacy external interrupt.
Originator/Completer	Originators and I/O completers.
Commands	RdSizedNC, doubleword aligned access only.
Length	1 doubleword.
Probes	None.

Table 62 Interrupt Vector Region for Probe Interrupt Delivery version 1

PrbAddr Range	FFFF_FFFE_6800_0000h - FFFF_FFFE_68FF_FF00h.
PrbAddr[23:8]	Physical thread identifier.
PrbAddr[7:0]	Reserved.
Address Size	16MB.
Usage	Notifying a core of an interrupt vector (when Probe Interrupt Delivery feature version 1 is implemented – reserved for all other variants).
Commands	None. Either the interrupt region (FFFF_FFFD_F8xx_xxxxh) or the APIC register space is used for sending interrupts from originators.
Probes	Yes, PrbAction=1001b

Table 63 Interrupt Delivery Region for Probe Interrupt Delivery version 2

PrbAddr Range	0000_0100_0000_0000h - 0000_01FF_FFFF_FFC0h. This address space is differentiated from DRAM addresses by the PrbAction field only. No requests are made to this address space.
PrbAddr[39:36]	When PrbAddr[32] = 0b, the task priority register (bits 7:4) at the time the interrupt was sent. Reserved for non-vector interrupts.
PrbAddr[35:33]	Reserved.
PrbAddr[32]	Vector/Non-Vector Indication. When 0b, the interrupt is a vectored interrupt (specified by PrbAddr[31:24]). When 1b, the interrupt is a non-vectored interrupt (specified by PrbAddr[31:24]). Reserved when PrbAddr[6]=1b.
PrbAddr[31:24]	When PrbAddr[32] = 0b, PrbAddr[31:24] contains an interrupt vector. When PrbAddr[32] = 1b, the non-vectored interrupt is specified as follows: **Value in PrbAddr[31:24]** — **Meaning** 03h — Asserting NMI 04h — Asserting INIT 05h — Asserting SMI 09h — Asserting STARTUP 0Fh — Asserting DBREQ2 10h — Asserting MCE 11h — Asserting ExtInt 12h — Asserting AVIC Reserved when PrbAddr[6]=1b. All other non-vectored encodings are reserved.
PrbAddr[23:8]	Physical thread identifier.
PrbAddr[7]	Reserved.
PrbAddr[6]	Message information. When 0b, this is an interrupt delivery. When 1b, this is a message that a previous delivered vector is superseded by a higher priority vector. When PrbAddr[6] is 1, bits 32:24 do not carry any meaningful information and are reserved.
Address Size	1TB.
Usage	Notifying a core of an interrupt (when Probe Interrupt Delivery feature version 2 is implemented – reserved for all other variants).
Commands	None. Either the interrupt region (FFFF_FFFD_F8xx_xxxxh) or the APIC register space is used for sending interrupts from originators.
Probes	Yes, PrbAction=1011b.

Table 64 X86 Legacy Wire Region

Address Range	FFFF_FFFD_F910_0000h - FFFF_FFFD_F910_FF00h.
Address Size	64KB.
Usage	Address dependent, see below. ReqAddr[7:0] are reserved and must be zero.
Originator/Completer	Originators and I/O completers.
Commands	WrNoDataNC.
Probes	IGNNE (PrbAction=1001b) and WBINVD (PrbAction=1011b) only.

Table 65 x86 Legacy Wire Address Definition

IGNNE wire change	FFFF_FFFD_F910_1x00h, bit 8 = new IGNNE value. From I/O originator, sent to all x86 CPU originators through the Probe Request Channel(s). The write response does not indicate that the wire change has been delivered to all CPUs.
FERR wire change	FFFF_FFFD_F910_2x00h, bit 8 = new FERR value. From CPU originator, sent to I/O completer(s).
SHUTDOWN	FFFF_FFFD_F910_4000h. From CPU originator, sent to I/O completer(s), signals a shutdown event.
WBNOINVD	FFFF_FFFD_F910_7000h. From CPU originator, signals a write back cache (without invalidate) instruction to the fabric.
WBINVD	FFFF_FFFD_F910_8000h. From CPU originator, signals a write back cache invalidate instruction to the fabric. SDP originators that are caching graphic blocks receive the write back invalidate message. The response on the Probe Response Channel(s) should only be provided when the cache invalidate is globally visible.
INVD	FFFF_FFFD_F910_9000h. From CPU originator, signals a cache invalidate instruction to the fabric.
SMIACK wire change	FFFF_FFFD_F910_Ax00h, bit 8 = new SMIACK value. From CPU originator, sent to I/O completer(s). The legacy pin SMIACK is asserted whenever a CPU unit (thread) enters system management mode (SMM) and deasserted when it exits SMM. Each CPU originator thread asserts and deasserts the signal separately.
INTx wire change	FFFF_FFFD_F910_Bx00h, bits 11-9 = INTx identifier (system dependent). When bit 8=0, signals a deassert message; when bit 8=1, signals an assert message. From I/O originator. The fabric counts assert and deassert messages and sends an assert/deassert message to I/O completer(s) when the count changes between zero and one. The behavior when the fabric overflows the count is unpredictable.
PCOMMIT	FFFF_FFFD_F910_Cx00h. • Bit 8, when zero, signifies the flush is to persistent external storage. • Bit 8, when one, signifies the flush is only past any memory encryption phase. • Bits 11:9 are *reserved*. From CPU or security processor originator, signals a persistent commit request. Signals the fabric to commit all previously committed writes (those writes that have had their write responses delivered) as indicated by bit 8. The completion to the PCOMMIT transaction comes only after this has occurred. Each CPU unit/security processor may have at most one outstanding PCOMMIT message at a time.

Special Register Regions

The system address range provides access to a protected register space for scalable machine check architecture (MCA) registers. The originator accesses these ranges with reads and writes as if it was non-coherent system memory (subject to address and size alignment requirements).

Architecture Note: The Scalable Machine Check Architecture Range is deprecated as of “Scalable MCA v3”. When Scalable MCA v3 or later is implemented, this range is reserved.

Table 66 Scalable Machine Check Architecture Range

Address Range	FFFF_FFFD_F940_0000h - FFFF_FFFD_F940_FFF8h.
Address Size	64KB.
ReqAddr [15:8]	MCA bank number.
ReqAddr [7:3]	MCA register number.
Usage	Accesses scalable MCA registers.
Originator/Completer	Originators only.
Commands	RdSizedNC and WrSizedNC, quadword aligned accesses only.
Length	Two doublewords. For writes, all byte enables must be set.
Probes	None.

The system address range provides access to a protected register space for security processor to a system management hub. The originator accesses these ranges with reads and writes as if it was non-coherent system memory (subject to address and size alignment requirements).

Table 67 Secure System Management Hub Registers

Address Range	FFFF_FFFD_F950_0000h - FFFF_FFFD_F95F_FFFFh and FFFF_FFFD_FA80_0000h – FFFF_FFFD_FABF_FFFFh (two separate regions)
Address Size	1MB and 4MB

ReqAddr [19:0]	System management hub register
Usage	Provides secure access to system management hub registers.
Originator/Completer	Originators only.
Commands	RdSizedNC and WrSizedNC.
Length	One doubleword.
Probes	None.

The system address range reserves a region for protected core registers. This range of addresses is not currently observable on any SDP and only the address range is defined in this specification.

Table 68 Secure Core and Cache Controller Registers

Address Range	FFFF_FFFD_F990_0000h - FFFF_FFFD_F99F_FFFFh
Address Size	1MB
ReqAddr [19:0]	Internal core to cache controller register
Usage	Provides secure access to core and cache controller registers.
Originator/Completer	N/A
Commands	N/A
Length	One doubleword.
Probes	None.

The system address range provides access to a protected register space for Advanced Programmable Interrupt Controller (APIC registers. The originator accesses these ranges with reads and writes as if it was non-coherent system memory (subject to address and size alignment requirements).

Table 69 APIC Register Range

Address Range	FFFF_FFFD_FE00_0000h - FFFF_FFFD_FE00_07FCh (software-visible registers) and FFFF_FFFD_FE01_0000h - FFFF_FFFD_FE01_07FCh (firmware-only registers).
Address Size	Two regions of 2KB.
ReqAddr [10:4]	APIC register number.
ReqAddr[3:2]	Ignored/reserved.
Usage	Accesses the APIC registers.
Originator/Completer	Originators only.
Commands	RdSized, RdSizedNoWriter, RdSizedDW, RdSizedNC, WrSized, and WrSizedNC. Predictable operation occurs only for defined APIC register addresses (all defined APIC register addresses are quadword aligned).
Length	Must not cross cache-line boundary boundary. Predictable operation occurs only when a doubleword boundary is not crossed. All byte enables in the doubleword must be set for predictable write operation.
Probes	None.

Table 70 APIC Task Priority Range

Address Range	FFFF_FFFD_FE02_0000h - FFFF_FFFD_FE02_FF00h
Address Size	2KB.
ReqAddr [15:12]	New task priority value.
ReqAddr[11:2]	Reserved
Usage	Updates the Task Priority Register in the APIC.
Originator/Completer	Originators only.
Commands	WrNoDataNC only.
Probes	None.

Table 71 In Service Vector Range

Address Range	FFFF_FFFD_FE04_0000h - FFFF_FFFD_FE04_FF00h
Address Size	64KB
ReqAddr [15:8]	Interrupt Vector
ReqAddr[7:2]	Reserved
Usage	When Probe Interrupt Delivery version 2 is implemented, transitions the APIC state to in-service for the specified interrupt vector. Reserved if the feature is not implemented.
Originator/Completer	Originators only.
Commands	WrNoDataNC only.
Probes	None, however refer to Table 63 Interrupt Delivery Region for Probe Interrupt Delivery version 2 for probes that are delivered in connection with this.

Responses to APIC register and interrupt service vector writes are generated only after all side effects of the operation have been observed including update of remote APIC state but does not necessarily guarantee delivery of an interrupt to the processor. Responses to Task Priority Writes are delivered only after the TPR value has been updated, including delivery of any newly enabled interrupts to the processor.

System Management APIC Save/Restore Region

The system address range provides a range for system management to save/restore the APIC register space for all CPUs in the system. The address is similar to the APIC range except the CPU unit identifier is embedded in the address instead of ReqUnitID and the APIC register number is shifted right by two bits.

Table 72 System Management APIC Save/Restore Region

Address Range	FFFF_FFFD_FE08_0000h – FFFF_FFFD_FE09_FFFFh.
Address Size	128KB (256 regions of 512 bytes)
ReqAddr[16:9]	CPU unit identifier (physical)
ReqAddr[8:2]	APIC register number.
Usage	Accesses the APIC registers.
Originator/Completer	Originators only.
Commands	RdSized, RdSizedDW, RdSizedNC, WrSized, and WrSizedNC, doubleword aligned accesses only. All byte enables must be set on writes.
Probes	None.

CPU Virtual Wire Region

The system address range provides a range for CPUs to send messages to all other CPUs in the system. The fabric delivers the message through the Probe Request Channel(s). In addition to the usage for Send Event (see below), the remainder of the address range from FFFF_FFFD_F970_0000h through FFFF_FFFD_F97F_FFFFh (1MB) is reserved for future expansion.

Table 73 Send Event Region

Address Range	FFFF_FFFD_F970_0000h.
Address Size	N/A.
Usage	Send Event. Write completion does not indicate that the event has been delivered to all CPUs.
Originator/Completer	Originators only.
Commands	WrNoDataNC.
Probes	Yes, PrbAction = 1001b

Trusted Platform Module Secure Kernel Initialization Region

The system address range provides a range for CPUs to participate in a Trusted Platform Module (TPM) secure kernel initialization sequence. This sequence is outside the scope of this document – refer to Security for more information.

Table 74 Trusted Platform Module Secure Kernel Initialization Region

Hash Start	FFFF_FFFD_F920_0000h. WrNoDataNC on CPU interfaces, or WrSizedNC of one doubleword on I/O interfaces. Sets hash received status.
Hash Stop	FFFF_FFFD_F928_0000h. WrNoDataNC on CPU interfaces, or WrSizedNC of one doubleword on I/O interfaces. Sets hash received status.
Hash Data	FFFF_FFFD_F928_0004h - FFFF_FFFD_F928_0007h (WrSizedNC of one doubleword to the aligned address. All byte masks must be set. The data payload contains one doubleword of the secure kernel image.
Hash Error	FFFF_FFFD_F928_0008h – FFFF_FFFD_F928_000Bh (RdSizedNC of one doubleword, bit 1 = hash received status, bit 0 = hash error signal). Only for CPU interfaces.
Init State Virtual Wire	FFFF_FFFD_F928_1000h – FFFF_FFFD_F928_1100h (WrNoDataNC with ReqAddr[8] = INIT state). Entering INIT state clears hash received status. Only for CPU interfaces.

GPU Virtual Wire Region

The system address range provides a range for GPUs to communicate the need for coherency probes. GPU caches have a reset state of disabled and do not receive memory coherency probes until a command to enable the cache is sent to the fabric. In addition, the GPU may disable hardware coherency at any time, for example, due to low-power states.

Table 75 GPU Virtual Wire Region

Address Range	FFFF_FFFD_F911_0x00h.
Address Size	N/A.
Usage	GPU cache enable/disable.
ReqAddr[8]	0 = GPU cache enabled

	1 = GPU cache disabled (reset state).
Enable Action	The originator must not send caching read requests until receiving the write response from an enable command. Before sending the request, the originator must be ready to receive probes.
Disable Action	The originator must quiesce all caching read requests and wait for their completion. After initiating the request, the originator must not send any further caching read requests (at least, until re-enabling). The originator must continue to respond to probes, which may be in flight at the time of the write response.
Originator/Completer	Originators only.
Commands	WrNoDataNC.
Probes	None.

PCI Address Spaces

The system address range provides access to PCI I/O ports (legacy IN/OUT instructions), PCI configuration and peer-to-peer device messages. The originator accesses these ranges with reads and writes as if it was non-coherent system memory (subject to address and size alignment requirements). The fabric is responsible for breaking large accesses to PCI configuration space.

Table 76 PCI Legacy I/O Port Region

Address Range	FFFF_FFFD_FC00_0000h – FFFF_FFFD_FC00_FFFFh.
Address Size	64KB.
Usage	Access to legacy PCI I/O ports.
Originator/Completer	Originators and I/O completers.
Commands	RdSizedNC, WrSizedNC.
Size	At originators: One doubleword only, must not cross quadword boundary. At I/O completers: One doubleword, must not cross doubleword boundary.
Probes	None.

Table 77 PCI Configuration Region

Address Range	FFFF_FFFE_0000_0000h – FFFF_FFFE_1FFF_FFFFh.
Address Size	512MB.
ReqAddr[28]	0=type 0 access, 1=type 1 access.
ReqAddr[27:20]	Bus number (within this PCIe segment).
ReqAddr[19:15]	Device number.
ReqAddr[14:12]	Function number.
ReqAddr[11:2]	Register number.
Usage	Access to PCI configuration registers.
Originator/Completer	Originators and I/O completers. Reserved at originators if multiple PCIe segments are supported.
Commands	RdSized, RdSizedNoWriter (not valid at I/O completers), RdSizedDW, RdSizedNC, WrSized, and WrSizedNC.
Size	At originators: Must not cross cache-line boundary. Predictable operation occurs only for requests that are either (a) one doubleword in size, or, (b) two doublewords without crossing a quadword boundary. At I/O completers: One doubleword, must not cross doubleword boundary. When multiple PCIe segments are supported, this address is not used at originators. Instead, the PCI configuration region is a programmable MMIO address BAR that is outside the scope of this specification. The fabric decodes the BAR and routes the configuration request to the correct SDP port associated with this segment. During this operation, the fabric converts the request to this address range. The I/O agents behind a single SDP port must be within the same PCIe bus segment.
Probes	None.

Table 78 Device Peer-to-Peer Message Region

Address Range	FFFF_FFFE_2000_0000h – FFFF_FFFE_2FFF_FFFFh.
Address Size	256MB.
ReqAddr[27:20]	Bus number within the same PCIe segment.
ReqAddr[19:2]	System dependent.
Usage	Peer to peer device messages within the same PCIe segment.
Originator/Completer	I/O originators and I/O completers.
Commands	RdSizedNC, WrSizedNC.
Size	Maximum size of one cache line., must not cross a cache line address boundary.
Probes	None.

Table 79 Device Peer-to-Peer Message Region with Segments

Address Range	FFFF_FFFE_3000_0000h – FFFF_FFFE_3FFF_FFFFh.

Address Size	256MB.
ReqAddr[27:20]	Bus number.
ReqAddr[19:12]	Segment Number.
ReqAddr[11:2]	System dependent.
Usage	Peer to peer device messages within the same PCIe segment.
Originator/Completer	I/O originators and I/O completers.
Commands	RdSizedNC, WrSizedNC.
Size	Maximum size of one cache line., must not cross a cache line address boundary.
Probes	None.

Distributed Virtual Memory Region

The system address range provides a range for DVM units (generally CPUs and IOMMUs) to participate in a distributed virtual memory sequence. Refer to DVM Operations for more information on the protocol.

Table 80 Distributed Virtual Memory Region

DVM Operation Request (DVMOpReq)	
ReqAddr	FFFF_FFFE_65xx_xxy0h.
ReqAddr[23:8]	Global originator unit identifier.
ReqAddr[7:6]	DVM target encoding: 00b – The DVM command is intended for both CPU and/or I/O memory management units, as subject to the SOC configuration (as programmed at the DVM controller). 01b – The DVM command is intended only for I/O memory management units. 10b – The DVM command is intended only for CPU memory management units. 11b - The DVM command is intended for both CPU and I/O memory management units.
ReqAddr[5:4]	Reserved.
Command	WrSizedNC.
Size	Four doublewords.
Data Payload	Refer to Table 83 DVM Field Assignments.
Action	The fabric delivers the DVM operation to all DVM units (including this originator).
Probes	N/A. A DVM Operation Message is delivered through PrbAction = 1000b but does not use a PrbAddr of FFFF_FFFD_F960_xx00h. See DVMOpMsg below.
DVM Operation Message (DVMOpMsg)	
PrbAddr	Refer to Table 83 DVM Field Assignments for PrbAddr usage on DVMOpMsg (PrbAddr carries the DVM operation payload).
PrbAction	1000b (signifies use of PrbAddr for DVM operation payload).
Action	Upon receipt of a DVMOpMsg, the unit processes the payload asynchronously. There is no response.
DVM Synchronization Request (DVMSyncReq)	
ReqAddr	FFFF_FFFE_66xx_xxy0h.
ReqAddr[23:8]	Global originator unit identifier.
ReqAddr[7:6]	DVM target encoding: 00b – The DVM command is intended for both CPU and/or I/O memory management units, as subject to the SOC configuration (as programmed at the DVM controller). 01b – The DVM command is intended only for I/O memory management units. 10b – The DVM command is intended only for CPU memory management units. 11b - The DVM command is intended for both CPU and I/O memory management units.
ReqAddr[5:4]	Reserved.
Command	WrNoDataNC.
Action	The fabric orders DVM synchronization requests and delivers a DVMSyncMsg to all DVM units (including this originator). A DVMCompMsg (with matching global originator unit identifier) is sent when this unit's DVM synchronization is completed. At most, each originator unit may have one outstanding DVMSyncReq at a time.
Probes	See DVMSyncMsg.
DVM Synchronization Message (DVMSyncMsg)	
PrbAddr	FFFF_FFFE_6600_0000h.
PrbAction	1001b.
Action	Upon receipt of a DVM synchronization message, each unit waits for all prior DVM operations to complete. Once these have completed, each unit then sends a DVM completion request (DVMCompReq).
DVM Completion Request (DVMCompReq)	
ReqAddr	FFFF_FFFE_67xx_xx00h.
ReqAddr[23:8]	Unpredictable (ignored).
Command	WrNoDataNC.
Action	The fabric counts DVM completion requests waiting for all DVM units to report that a synchronization

	has completed.
Probes	See DVMCompMsg.
DVM Completion Message (DVMCompMsg)	
PrbAddr	FFFF_FFFE_67xx_xx00h.
PrbAddr[23:8]	Global originator unit identifier whose DVMSyncReq has completed.
PrbAction	1001b.
Action	Signifies that a prior DVMSyncReq from the matching global originator unit identifier has completed. DVM units must ignore any DVMCompMsg received without a matching identifier. There is no response to this message.

Bus Lock Region

The system address range provides a range for CPU units to participate in a bus sequence. Refer to Bus Locks for more information on the protocol.

Table 81 Bus Lock Region

Bus Lock Request (BusLockReq)	
ReqAddr	FFFF_FFFE_60xx_xx00h.
ReqAddr[23:8]	Global originator unit identifier.
Command	WrNoDataNC.
Action	The fabric orders bus lock requests and locks the bus. At most, each originator unit may have one outstanding bus lock request at a time.
Probes	See BusLockReqMsg.
Bus Lock Request Message (BusLockReqMsg, aka "quiesce")	
PrbAddr	FFFF_FFFE_6000_0000h.
PrbAction	1001b
Action	Each originator unit waits for all prior operations to complete. Once complete, each unit sends a BusLockGrant. No further operations are initiated until the bus lock is released.
Bus Lock Grant (BusLockGrant)	
ReqAddr	FFFF_FFFE_61xx_xx00h.
ReqAddr[23:8]	Unpredictable (ignored).
Command	WrNoDataNC.
Action	The fabric counts bus lock grant requests until all units have entered the bus locked state. Once all grants have been received, the fabric sends a bus lock grant message.
Bus Lock Grant Message (BusLockGrantMsg)	
PrbAddr	FFFF_FFFE_61xx_xx00h.
PrbAddr[23:8]	Global originator unit identifier for the unit that has been granted the bus lock (from a prior BusLockReq).
PrbAction	1001b
Action	Signifies that a prior BusLockReq from the matching global originator unit identifier has completed and the identified unit may atomically access the bus. When the atomic accesses are complete, the unit must send a BusLockRelease. Units must ignore any BusLockGrantMsg received without a matching identifier.
Bus Lock Release (BusLockRelease)	
ReqAddr	FFFF_FFFE_62xx_xx00h.
ReqAddr[23:8]	Unpredictable (ignored).
Command	WrNoDataNC.
Action	Signifies the bus lock has been released. Sent after a granted unit has completed the atomic operations. The unit may ***not*** initiate other requests until a BusLockReleaseMsg has been received.
Bus Lock Release Message (BusLockReleaseMsg)	
PrbAddr	FFFF_FFFE_6200_0000h.
PrbAction	1001b
Action	Signifies that the bus lock has complete and all units may resume normal request operation.

DVM/Bus Lock Protocol Region

The system address range provides a range for DVM and bus lock units (generally CPUs) to join or leave the protocol in order to provide for an accurate count of responses in the case of units that enter or leave low-power state(s) that preclude them from participating in the protocol.

Table 82 DVM/Bus Lock Protocol Region

System Management Join Request (SysMgmtJoin)	
ReqAddr	FFFF_FFFE_63xx_xx00h.
ReqAddr[23:8]	Global unit identifier
Command	WrNoDataNC.

Action	The fabric delivers a SysMgmtJoinMsg after updating the count of expected responses. The originator unit does not act on any bus lock and DVM messages until this message is received.
Probes	See SysMgmtJoinMsg**.**
System Management Join Message (SysMgmtJoinMsg)	
ReqAddr	FFFF_FFFE_63xx_xx00h.
ReqAddr[23:8]	Global unit identifier
PrbAddr[7]	Current bus lock state (1 = bus locked)
PrbAction	1001b.
Action	Upon receipt of a SysMgmtJoinMsg, the identified unit must act on any future bus lock and DVM messages. PrbAddr[7] must be sampled for the current bus lock state.
System Management Leave Request (SysMgmtLeave)	
ReqAddr	FFFF_FFFE_64xx_xx00h.
ReqAddr[23:8]	Global unit identifier.
Command	WrNoDataNC.
Action	The fabric delivers a SysMgmtLeaveMsg after updating the count of expected responses. The originator unit continues to act on any bus lock and DVM messages until this message is received.
Probes	See SysMgmtLeaveMsg**.**
System Management Join Message (SysMgmtLeaveMsg)	
PrbAddr	FFFF_FFFE_64xx_xx00h.
PrbAddr[23:8]	Global unit identifier.
PrbAction	1001b.
Action	Upon receipt of a SysMgmtLeaveMsg, the identified unit no longer acts on any future bus lock and DVM messages.

3.11 DVM Operations

An originator unit initiates a distributed virtual memory (DVM) operation using a DVM Operation WrSized. This DVM operation uses the ReqAddr field and a two-doubleword data payload to encode the request. The request goes to a fabric DVM synchronization point generates a system management message using the data payload and broadcasts this to all DVM originators. This message is delivered through two chained probes (PrbAction=1000b) if the originator implements a Probe Request Channel, or through a WrSized command. When delivered through a Probe Request Channel, the DVM payload is contained in PrbAddr and the DVM operation system management message does not generate a probe response.

Since the PrbAddr bits carry the DVM operation payload, any SOC that supports DVM is required to have the most-significant bit of PrbAddr set to the maximum of the following values:

- The most-significant bit of ReqAddr.
- Bit 47 when 48-bit virtual address is supported.
- Bit 51 when 57-bit virtual address is supported.

IP Reuse Implementation Note: For compatibility purposes, when the SDP ReqAddr width is larger than the SOC ReqAddr width due to IP reuse, and the SOC ReqAddr width does not support 57-bit virtual addresses, fabric must sign-extend VA[46] into VA[56:47] of the probe address payload.

At some point after one or more DVM operations, an originator unit may request a DVM synchronization operation. The synchronization operand synchronizes the actual completion of all prior DVM operations by this unit.

An originator unit requests synchronization by issuing a DVM synchronization system management message on its SDP interface, which includes a global originator unit identifier (assigned through a means outside the scope of this document). This message is received and processed by the DVM synchronization point within the fabric which broadcasts the synchronization request to this and all other originators in the system through either the Probe Request Channel(s) or as a WrNoDataNC operation. After receiving a synchronization request, originator units must complete all prior DVM operations. Each originator unit sends a DVM completion system management message when this has occurred.

When the DVM synchronization point has received acknowledgments from all originator units, it informs the original requester that the synchronization has completed through a Probe Request Channel DVM completion message. The global originator unit identifier from the DVM synchronization request is repeated on this message to communicate which unit’s DvmSyncReq has been completed.

To accommodate originators with a variable number of originator units participating in the DVM operation protocol (for example, due to low-power states), each unit that generates or receives DVM operations must specifically join the protocol prior to initiating other relevant requests and must leave the protocol before going into a low-power state.

Whether a DVM operation affects CPU memory management units, I/O memory management units, or both may be specified by the initiator in ReqAddr[7:6]. It is necessary to follow up with DvmSyncRequest(s) that affect the same targets as prior DvmOpReq in order for these agents to signal that they are complete.

Table 83 DVM Field Assignments shows the mapping of the DVM operation fields to the data payload of the DVM operation command and the DVM operation probe requests generated by the fabric.

Table 83 DVM Field Assignments

Format (see Note)	**DVM TLBI Operation Field**	**Data Payload (OrigData bits 127:0)**	**DVMop Payload (PrbAddr[n:2])**	**Notes**
All		Data[1:0]	-	*Reserved* (PrbAddr[1:0] not implemented)
All		Data[3:2]	Beat 0 PrbAddr[3:2]	Reserved
All	VA Valid	Data[4]	Beat 0 PrbAddr[4]	
All	PCID Valid	Data[5]	Beat 0 PrbAddr[5]	Reserved when bit 9 specifies that the DVM operation packet carries an IOMMU domain ID.
All	ASID Valid	Data[6]	Beat 0 PrbAddr[6]	
All	Leaf	Data[7]	Beat 0 PrbAddr[7]	See Table 84 Leaf Encoding
All	Include Nested	Data[8]	Beat 0 PrbAddr[8]	When 1, nested page tables are included.
Cores	PCID Select	Data[9]	Beat 0 PrbAddr[9]	When 0, bits 27:16 contain PCID and bits 31:28 are reserved.
IOMMU	IOMMU Domain Select			When 1, bits 31:28 contain IOMMU domain ID.
All	Guest OS/HV	Data[11:10]	Beat 0 PrbAddr[11:10]	See Table 85 Guest OS/HV Encoding
All	MsgType	Data[14:12]	Beat 0 PrbAddr[14:12]	See Table 86 DVM Message Type Encoding
Cores	VA[56]	Data[15]	Beat 0 PrbAddr[15]	Virtual address bit 56.
IOMMU				*Reserved*.
Cores	PCID	Data[27:16]	Beat 0 PrbAddr[31:16]	PCID[11:0] (when bit 9 is 0, this field is PCID).
Cores	Reserved	Data[31:28]		*Reserved* (when bit 9 is 0, this field is reserved).
IOMMU	IOMMU Domain ID	Data[31:16]		IOMMU Domain ID[15:0] (when bit 9 is 1, this field is the IOMMU domain)
All	ASID	Data[47:32]	Beat 0 PrbAddr[47:32]	
Cores	VA[55:52]	Data[51:48]	Beat 0 PrbAddr[51:48]	Virtual address bits 55:52.
IOMMU				*Reserved*
All		Data[63:52]	-	*Reserved*
All		Data[65:64]		*Reserved* (PrbAddr[1:0] not implemented)
All	Include Global	Data[66]	Beat 1 PrbAddr[2]	When 1, global pages are included.
All	Count	Data[69:67]	Beat 1 PrbAddr[5:3]	Count is a zero-origin count of the number of pages to
All		Data[74:70]	Beat 1 PrbAddr[10:6]	*Reserved* (PrbAddr[63:49] not implemented)
All	Page Size	Data[75]	Beat 1 PrbAddr[11]	0b = 4KB or 16KB (dependent on SOC page-size configuration outside the scope of this document), 1b =
All	VA[47:12]	Data[111:76]	Beat 1 PrbAddr[47:12]	Virtual address bits 47:12. When count page size is 1, VA bits[20:12] must be zero.
Cores	0000b, VA[51:48]	Data[120:112]	Beat 1 PrbAddr[56:48]	Virtual address bits 51:48 with four zeros padded above.
IOMMU	VA[56:48]			Virtual address bits 56:48.
		Data[127:121]	-	*Reserved* (PrbAddr[63:n] not implemented)

Note:

The location of VA[56:52] (virtual address bits 56:52) differs between originators that receive the DVM operation message through a Probe Request Channel.

For originators that receive the DVM operation message through a Probe Request Channel (shown above with "Cores" in the Format column), VA[56] is placed in OrigData[15] and expected on beat 0 PrbAddr[15]. VA[55:51] is placed in OrigData[51:48] and expected on beat 0 PrbAddr[51:48]. OrigData[120:112] is reserved and must be zero.

For originators that receive the DVM operation message through a downstream Originator Request Channel (shown above with "IOMMU" in the Format column), VA[56:52] is placed in the "natural" position of OrigData[120:112]. OrigData[15] and OrigData[51:48] are reserved.

Originators are required to place zeros on reserved bits when it generates a message and ignore reserved bits on received messages.

Implementation Note: The data fabric follows the core format (carries VA[56:52] at PrbAddr[15, 51:48]). The DVM logic within the fabric swizzles OrigData[120:112] on WrSizedNC when the DVM operation is initiated from a IOMMU agent. Furthermore, the logic that converts the fabric probe message to a downstream WrSizedNC then reverses that swizzle and places PrbAddr0[15,51:48] onto OrigData[120:112].

Table 84 Leaf Encoding

Value	Description
0b	Operation applies to all levels
1b	Operation applies to leaf level translation only

Table 85 Guest OS/HV Encoding

Value	Description
00b	Both guest OS and hypervisor
01b	*Reserved*
10b	Guest OS
11b	Hypervisor

Table 86 DVM Message Type Encoding

Value	Description
000b	TLB invalidate
001b	*Reserved*
010b	*Reserved*
011b	*Reserved*
100b	*Reserved, see note 1*
101b	*Reserved*
110b	*Reserved*
111b	*Reserved*
Notes: 1. DVM sync (and DVM completion) uses a separate system memory address and is not encoded in MsgType.	

3.12 Error Handling

A general error handling guideline is outside the scope of this specification and is left to the responsibility of the originator, completer, or SOC architecture. The following section provides guidelines and some specific architectural requirements of SDP.

3.12.1 Parity Generation and Checking

Originators and completers must support data parity unless a higher level of protection, such as ECC, is used instead. Data parity must always be generated across the *Data fields on the Read Response and Originator Data channels, regardless of byte enables, the address and length of the original request, and regardless of the RdRspStatus field.

For parity bits that protect meta-data, any fields that are not implemented should be treated as if zeros for the purposes of parity generation and checking. This facilitates the ability to hook up a port that doesn't provide all the optional signals to a port that supports these signals by allowing them to be driven to zero at an interface level.

Parity may be checked against the covered signals in that channel on each cycle where OrigDataVld, RdRspVld (or RdRspDataVld when applicable) is asserted. Data parity is checked regardless of the value of the byte enables. The enabling of parity checking is implementation specific. Any port that implements the any versions of the Enhanced RAS Features must provide a means to disable the checking of the meta-data parity and it is recommended that this method resets to have the checking disabled unless there is a means to guarantee that firmware may program the intended enable/disable value before the port is initialized. Data parity checking may reset to disabled or enabled, depending on the error handling guidelines for the SOC.

Parity is always even-parity, that is the number of set bits in the protected group (including the parity bit itself) must always be even. Responding to the detection of an odd number of bits is implementation specific. An originator might respond to a data or meta-data parity error by initiating an ErrEvent request. When data parity error is observed, it is recommended that an originator (or completer) treat the data as if RdRspDataStatus was DATERR (or OrigDataError was 1). This behavior is not recommended for meta-data parity errors.

3.12.2 Error Responses and Unsupported Command/Action Encodings

Completers are recommended to respond to reserved ReqCmd encodings using a WrRspStatus of SLVERR and repeating the ReqTag, ReqUnitID, ReqVC, etc. Completers are recommended to respond to unsupported (but defined) ReqCmd encodings using the appropriate read response or write response bus for that encoding with a status of SLVERR; however, it is acceptable to treat these as reserved encodings and always use a write response. This allows for a defined response in the case of new command encodings defined in future SDP architecture enhancements.

As discussed in Read Response and Data Associated with Error Status, when a read response is completed, all data beats of a read response must be returned to the requester regardless of the RdRspStatus. Any dummy data that is necessary must be generated by the completer, provided that no trust level is broken by supplying private data. It is acceptable for this dummy data to be all ones. An originator should not cache this data.

Originators are recommended to respond to reserved or unsupported PrbAction encodings using PrbRspStatus = 00010b (transaction error), repeating the PrbTag. Originators which do not support system management messages are recommended to initiate an ErrEvent request if a probe is received with PrbAction of 1000b or 1001b.

3.12.3 Write Error Responses Prior to Data

Regardless of whether or not an error is detected on the request itself, the final write response may not be generated prior to the receipt of all expected beats of data.

3.12.4 Using Response Channels for Unsolicited Fatal Errors

When the Completer Fatal Error Notification Feature is implemented, the completer (or fabric acting as a proxy for the completer) may utilize a Write Response Channel (or a Read Response Channel if and only if the Write Response Channel is not implemented) to notify an originator of a system fatal error. This may allow the originator to gather and protect key debug information to aid in the resolution of this error. The behavior of an originator after receiving a write response that indicates a system fatal error is similar to the behavior of a completer after receiving a request with command ErrEvent with ReqAttr=01h.

A completer signals the fatal error by asserting WrRspVld and WrRspStatus=SYSFATALERR. WrRspTag=00h indicates that the reason is a system fatal error. Other encodings of WrRspTag are reserved for future enhancements. An originator should treat all reserved WrRspTag encodings as if the WrRspTag was zero.

All other fields on the write response channel are reserved and must be ignored by the originator. The completer must observe WrRspRdy asserted in order to complete the transfer of the fatal error notification but it is not necessary for the completer to have any write response credits.

When the Write Response Channel is not implemented, a Read Response Channel may be used in a similar manner (RdRspVld=1, with RdRspStatus=SYSFATALERR, and RdRspTag indicating the reason for the fatal error). In addition, data transfer does not occur (RdRspDataStatus=NODATA).

3.12.5 Uncorrected (Poisoned) Data

Both data channels have an indication that the data is uncorrected or “poisoned” when RdRspDataStatus is DATERR or OrigDataError is asserted. All SDP originators and completers are required to accept uncorrected data and respond appropriately according to the SOC-15 Reliability, Availability and Serviceability requirements (refer to http://twiki.amd.com/twiki/bin/view/SIG/RasPlanning#SOC15). The following are general but not necessarily all-encompassing examples of recommended behavior:

1) As an originator, receiving the uncorrected read response data and storing it in the cache with an indication that it is uncorrected/poisoned data. When the data is consumed (e.g. used as a result in an instruction), an appropriate machine check exception or interrupt may be logged and generated, or when the data is probed out or evicted out, asserting OrigDataError on the outgoing transaction.

2) As a completer, receiving the uncorrected write data and storing it in the memory with an indication that it is uncorrected/poisoned data and returning the uncorrected error status if the data is read.
3) As a port that acts as a transport to other SDP ports or other port definitions that include a similar uncorrected data status, passing the uncorrected status to the eventual recipient.
4) Using ErrEvent or other similar method of reporting a fatal error when a port receives uncorrected data that cannot be consumed by the client or passed on to another port.

An error is not generally expected to be logged or reported in the case of SDP ports or fabrics that pass uncorrected data from one client to another client when both sides have appropriate support for handling uncorrected/poisoned data.

When RdRspDataStatus is DATERR or OrigDataError is asserted, the actual contents of RdRspData and OrigData are unpredictable. The data does not necessarily bear any correlation to the actual contents of the memory and may be replaced with arbitrary data at any point in the transport. However, the parity is always expected to be driven appropriately for the data contents.

When a read transaction completes normally, except that the contents of the memory are uncorrected/poisoned, the appropriate RdRspStatus is OKAY. It is architecturally legal for a completer to provide uncorrected data and a simultaneous error such as DECERR, SLVERR, or TRANSERR, but this only occurs in uncommon situations where the two errors are simultaneous.

Since the command WrNoDataNC has no opportunity to present uncorrected data, it is recommended that any system management command that has an opportunity to report an error uses the ErrEvent command instead of, or shortly after, the WrNoDataNC command.

3.13 Security

For more information on the SOC-level security architecture, please consult the latest version of the specification posted on the SOC-15 Security Twiki page: http://twiki.amd.com/twiki/bin/view/NB/SOC15DVSecurity.