# Streaming Fabric Interface (SFI)

## Specification

*July 2022*

**Revision 1.0**

Reference Number: 644200

Intel Corporation and its subsidiaries (collectively, “Intel”) would like to receive input, comments, suggestions, and other feedback (collectively, “Feedback”) on this specification. To be considered for incorporation into the specification, Feedback must be submitted by e-mail to: fabricspecification@intel.com. To the extent that You provide Intel with Feedback, You grant to Intel a worldwide, non exclusive, perpetual, irrevocable, royalty-free, fully paid, transferable license, with the right to sub-license, under Your Intellectual Property Rights, to make, use, sell, offer for sale, import, disclose, reproduce, make derivative works, distribute, or otherwise exploit Your Feedback without any accounting. As used in this paragraph, “Intellectual Property Rights” means, all worldwide copyrights, patents, trade secrets, and any other intellectual or industrial property rights, but excluding any trademarks or similar rights. By submitting Feedback, you represent that you are authorized to submit Feedback on behalf of your employer, if any, and that the Feedback is not confidential .Intel does not control or audit third-party data. You should review this content, consult other sources, and confirm whether referenced data are accurate.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software, or service activation. Learn more at intel.com, or from the OEM or retailer.

No computer system can be absolutely secure. Intel does not assume any liability for lost or stolen data or systems or any damages resulting from such losses.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document. The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/performance.

Cost reduction scenarios described are intended as examples of how a given Intel-based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.

Results have been estimated or simulated using internal Intel analysis or architecture simulation or modeling and provided to you for informational purposes. Any differences in your system hardware, software or configuration may affect your actual performance.

Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting www.intel.com/design/literature.htm.

# Contents

## Figures

## Tables

# Revision History

| Revision Number | Description | Date |
|---|---|---|
| 1.0 | • Examples and clarifications on packet formats, data interleaving and shared credits<br>• Addition of data parity, viral and fatal signals<br>• Updates and additions to Parameters | July 2022 |
| 0.9 | • Update data_ecrc to data_trailer and allow the different PCIe* trailer sizes<br>• Add data_suffix to support future PCIe* placeholders of suffix<br>• End Bad (EDB) support<br>• Implementation notes and clarifications on parameters | August 2021 |
| 0.7 | • Initial Release | March 2020 |

# 1 Introduction

## 1.1 Terminology

### Table 1-1. Terms and Definitions

| Title | Definition |
|---|---|
| Ack | Acknowledge |
| Agent | Agent refers to the System on Chip (SoC) IP that connects to the fabric |
| ECRC | End-to-End Cyclic Redundancy Check. Defined in the *PCI Express* (PCIe*) Base Specification* |
| endpoint | As defined in the *PCI Express* (PCIe*) Base Specification* |
| FC | Flow Control |
| FC ID | Flow Control Identifier |
| HDR | Header |
| packet header | Protocol packet that is sent on the HDR channel. Used interchangeably with "protocol header" |
| IDE | Integrity and Data Encryption |
| LSB | Least Significant Bit |
| MAC | Message Authentication Code |
| MSB | Most Significant Bit |
| NAck | Negatively Acknowledge |
| NP | Non-Posted |
| PCIe* | Peripheral Component Interconnect Express* |
| REQ | Request |
| Root Complex | As defined in the *PCI Express* (PCIe*) Base Specification* |
| Root Port | As defined in the *PCI Express* (PCIe*) Base Specification* |
| RSP | Response |
| RTL | Register-Transfer Level. A design abstraction which models synchronous digital circuits |
| RX | Receiver |
| SFI | Streaming Fabric Interface |
| TLP | Transaction Layer Packet. As defined in the *PCI Express (PCIe*) Base Specification* |
| TX | Transmitter |
| VC | Virtual Channel |
| VC ID | Virtual Channel Identifier |
| VN | Virtual Network |

# 1.2 Reference Documents

### Table 1-2. Reference Documents

| Document | Document Location |
|---|---|
| *PCI Express* (PCIe*) Base Specification* | https://pcisig.com |

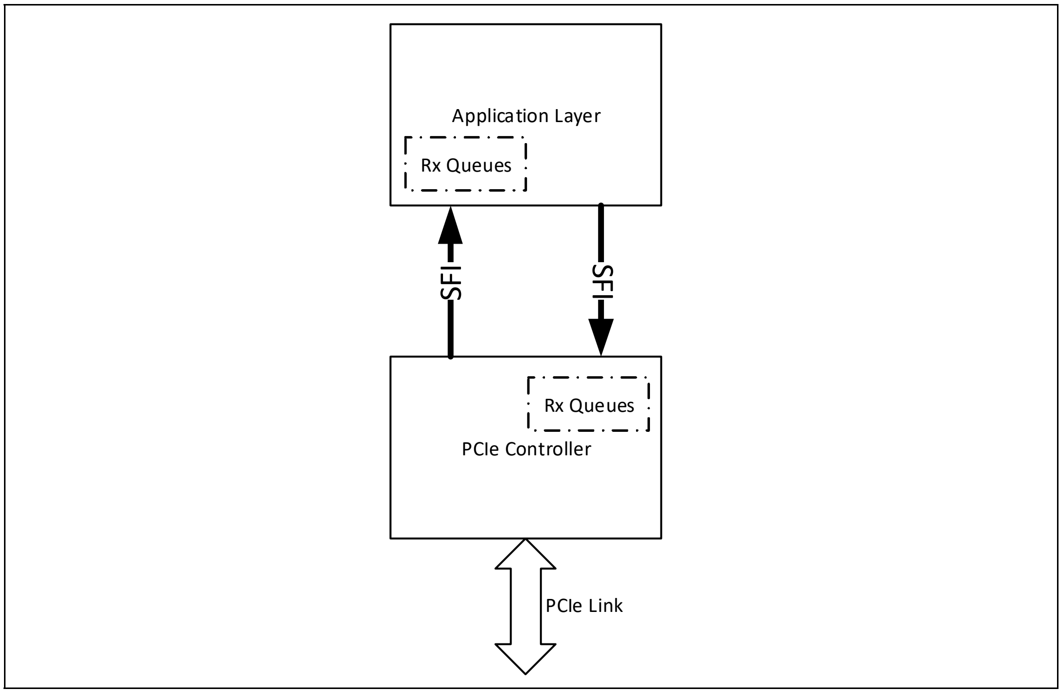
# 2 Overview

This specification describes the Streaming Fabric Interface (SFI) specification, which has been developed to map Load and Store protocols (such as PCIe*) between an agent and a fabric. The primary motivation is to provide a scalable streaming interface that can sustain the high bandwidth requirements of Load and Store protocols, such as PCIe*. The goal is to enable ease of implementation on both the transmit and receive side when transmitting such high data rates. Certain rules, when enforced, greatly simplify storage overhead in the context of read or write ports on the RX.

The SFI is applicable both in the context of a Host CPU (root complex) or in the context of a device endpoint. In both cases, the SFI serves to carry protocol layer (transaction layer) specific information between different processing entities. As an example on the device side, the SFI can be used to interface between the PCIe* controller and the application layer. Similarly, on the host side, the SFI can be used to interface between the PCIe* root port and the CPU fabric. The SFI can be parametrized to be wide enough and carry multiple packets in a single transfer. Data transfer is unidirectional, so the expectation is to have a pair of SFI instances between communicating blocks. Most of the description in this specification only references a TX and RX pair.

Different configurations can be enabled using the SFI as the intermediate interface. The SFI makes no assumptions around protocol and application specific responsibilities between the TX and the RX. It simply provides a mechanism and rules for high bandwidth packet transfer. Figure 2-1 shows an example interface instantiation in a PCIe* device.

**Figure 2-1. Example Topology and Interface Instantiations**



The SFI does not contain any new protocol definition. In fact, SFI semantics can be used to support different protocols as long as they can be mapped to the Flow Control (FC) and Virtual Channel (VC) semantics that SFI provides. For PCIe* protocol, the Header (HDR) formats follow the description outlined in the *PCI Express* (PCIe*) Base Specification 6.0* for FLIT mode headers.

The SFI supports advertisement of zero or more shared credit pools for the RX queues. The specification describes two different ways in which shared credits can be advertised or implemented.

# 3 SFI Interface

## 3.1 SFI Physical Channels

For each instance, besides certain global signals, the SFI defines two physical channels from TX to RX. The HDR and DATA channels carry packet transfers from TX to RX. Credit returns for each of the physical channels are from RX to TX and are considered part of that physical channel description. Figure 3-1 shows the different physical channels of the SFI.

Global Layer carries signals that apply across all physical channels. It carries the signals that are also used for initialization and shutdown of the interface. See Section 5 for initialization and shutdown flow.

Each of the HDR and DATA channels can carry multiple packets on the same cycle of transfer. Since most Load and Store protocols rely on ordering semantics, SFI assumes implicit ordering when multiple packets are sent on the same cycle. Packets are ordered from the least significant position to the most significant position. For example, if the Transaction Layer Packet (TLP) 0 begins from byte 0 of the header and TLP 1 begins from byte 16 of the header, then the RX must consider TLP 1 ordered behind TLP 0 whenever ordering rules apply. For transfers across different clock cycles, the ordering rules of the relevant protocol are followed (as an example, the SFI carries over all of the PCIe* ordering rules when used for PCIe*).

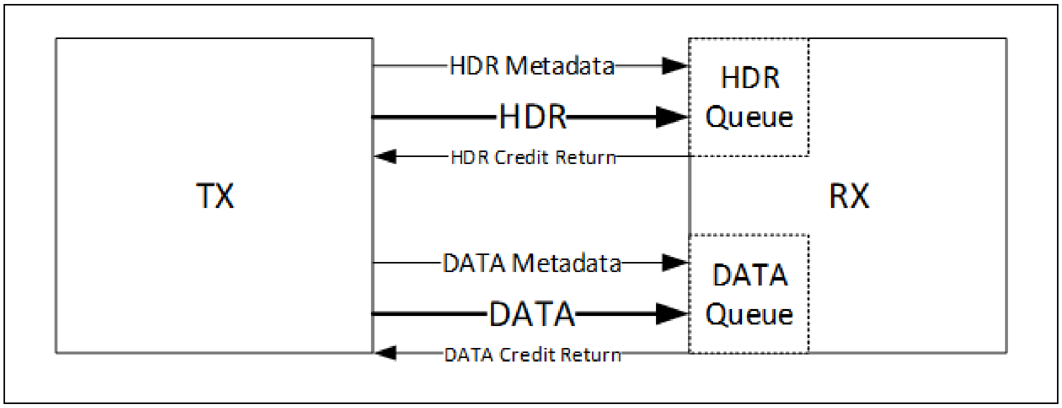
The VC assignment and decoding is implementation-specific but must be consistent between TX and RX to satisfy all the system level and protocol requirements. For example, in cases of link subdivision, it is permitted for the different ports from the controller perspective to map to different VCs on the SFI.

Multiple SFI interface instances could be needed to service a protocol (for example, a request going on one instance and the corresponding response coming back on a different instance).

Depending on the parameters used to configure the interface, each of the physical channels carries metadata to convey information about the position of different packets within a single transfer. This will be covered in more detail in a subsequent section.

Packet headers with data associated with them send the packet header on the HDR channel and the data on the DATA channel. It is assumed that the RX keeps track of the associated data length for each received header and only processing the relevant data size for that header. The data size is sent with the packet header information. As an example in PCIe*, the length field in the TLP header indicates how many four-byte (DWORD) chunks of data are associated with that header.

**Figure 3-1. SFI Physical Channels Showing the HDR and DATA Channels**



## 3.2 SFI Protocols Supported

The protocols are not defined as part of the SFI, but are defined in other documents. The present specification uses an example of PCIe* to illustrate recommended parameters and applications, but other streaming protocols can use similar semantics.

# 4 SFI Physical Channel Description

## 4.1 Global Layer

Global Layer carries signals that apply across all physical channels. It carries the signals that are also used for initialization and shutdown of the interface. See Section 5 for initialization and shutdown flows.

**Table 4-1. Signals of the Global Layer**

| Signal Class | Signal Name | Width | Direction | Description |
|---|---|---|---|---|
| `Init` | `txcon_req` | 1 bit | TX → RX | Connection request from TX<br>(0 → 1 connection request,<br>1 → 0 disconnection request) |
| | `rxcon_ack` | 1 bit | RX → TX | Connection acknowledge from RX<br>(0 → 1 connection acknowledge,<br>1 → 0 disconnection acknowledge) |
| | `rxdiscon_nack` | 1 bit | RX → TX | Disconnection Negatively Acknowledge (NAck) from RX |
| | `rx_empty` | 1 bit | RX → TX | RX queues are empty for all channels, and all credits have been returned. |
| RAS | `tx_viral` | 1 bit | TX → RX | Optional signal to indicate viral status from TX to RX. It is a level signal, once asserted it remains asserted until error handling is complete. |
| | `tx_fatal` | 1 bit | TX → RX | Optional signal to indicate fatal status from TX to RX. It is a level signal, once asserted it remains asserted until error handling is complete. |
| `Vendor Defined` | `tx_vendor_field` | VT bits | TX → RX | Optional signal to send static or global vendor-defined information from TX to RX.<br>For PCIe* protocol, it is permitted to carry additional global information for power or performance optimizations (for example, Link speed, width), but the SFI interface interoperability must not depend on these signals. |
| | `rx_vendor_field` | VR bits | RX → TX | Optional signal to send static or global vendor-defined information from RX to TX.<br>For PCIe* protocol, it is permitted to carry additional global information for power or performance optimizations (for example, Link speed, width), but the SFI interface interoperability must not depend on these signals. |

## 4.2 HDR Layer

The HDR Layer carries requests from TX to RX. The address and the protocol level command information are encapsulated in the **header** field.

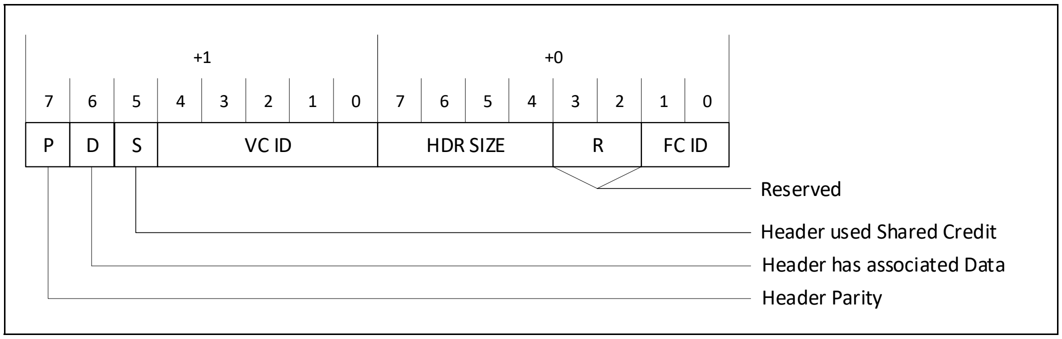
### Table 4-2. Fields of the HDR Layer

| Signal Class | Signal Name | Width | Direction | Description |
|---|---|---|---|---|
| **VALID** | **hdr_valid** | M bits | TX → RX | Indicates the start of a new packet on the corresponding header byte. It is required to have a fixed association between the individual bits of **hdr_valid** and the bytes of header. |
| | **hdr_block** | 1 bit | RX → TX | This is an optional signal. It is an indication from RX to TX for a temporary stall to header transfers.<br>See Section 4.4.1 for rules related to this signal.<br>Applications of temporary stalling include dynamic clock gating and/or temporary stalls due to clock crossing FIFOs. |
| | **hdr_early_valid** | 1 bit | TX → RX | This is an optional signal. It is an early valid indication from TX to RX indicating that TX has packets to send on the HDR channel.<br>See Section 4.4.1 for rules related to this signal.<br>In conjunction with the **hdr_block** signal, it is used to enable dynamic clock gating for RX. |
| **HDR** | **header** | H bytes | TX → RX | Header information. It can contain multiple packets transferred in the same cycle. FC and VC information is additionally embedded in **hdr_info_bytes**. H must be a multiple of M, and **hdr_valid[i]** corresponds to **header** byte (**i**×H/M). |
| | **hdr_info_bytes** | 2×M bytes | TX → RX | Indicates packet headersize, flow control, VC, and parity information. Information is valid whenever the corresponding **hdr_valid** is asserted. |
| **CREDIT** | **hdr_crd_rtn_valid** | 1 bit | RX → TX | Credit return valid. |
| | **hdr_crd_rtn_ded** | 1 bit | RX → TX | Indicates the credit returns are to the dedicated pool of credits. If this is 0 on a valid credit return, the credits should be returned to the corresponding shared pool of credits. It must be set to 0 if **hdr_crd_rtn_vc_id** is 5’d30 or 5’d31. Refer to Section 4.5.1 for details. |
| | **hdr_crd_rtn_fc_id** | 2 bits | RX → TX | Identifies the flow control class for this credit return. |
| | **hdr_crd_rtn_vc_id** | 5 bits | RX → TX | Identifies the VC for this credit return. |
| | **hdr_crd_rtn_value** | NHCRD bits | RX → TX | Indicates how many credits are returned in this cycle |
| | **hdr_crd_rtn_block** | 1 bit | TX → RX | This is an optional signal. It is an indication from TX to RX for a temporary stall to credit returns.<br>See Section 4.4.1 for rules related to this signal.<br>Applications of temporary stalls include dynamic clock gating and/or stalls due to clock crossing FIFOs. |

The width of **header** field "H" is a predetermined parameter based on the peak sustained bandwidth requirements. Rules for the HDR channel are outlined as follows:

1. A packet header must begin and end on the same cycle of transfer. Multiple packet headers can be sent on the same cycle.
2. The first packet header on a valid **header** transfer must start on byte 0 of the header field. Subsequent packets, if available for transfer in the same clock cycle, must begin at the next available header start location (i.e. if the prior packet header was at byte location "i×H/M", and had less than or equal to H/M bytes, the next packet header must start at "(i+1)×H/M" if being sent on the same clock cycle. If prior packet header took more than H/M bytes, then the next packet header must begin at the next available multiple of H/M byte, if being sent on the same clock cycle).
3. When **hdr_valid** corresponding to a **header** byte is asserted, the **hdr_info_bytes** field describes key attributes that can be used by the RX to decode the packet header. These are shown in Figure 4-1 (least significant byte is shown to the right).
    a. P is the Parity bit. Support for parity bit is optional. When supported, it is the XOR of all bits of a packet header and the non-parity bits of the corresponding **hdr_info_bytes**. Parity errors on the HDR channel are fatal to interface operation and must be reported as fatal errors. When not supported, it must be reserved; TX must drive 0, and RX must not check parity.
    b. D indicates the packet header has corresponding data associated with it.
    c. S indicates that the corresponding packet header used as a shared credit. If this packet header had data associated with it, then the data must also consume credits from the shared pool.
    d. Virtual Channel Identifier (VC ID) is the virtual channel identifier for the packet header - it can be in the range of 0 to 29 (VC ID 30 and 31 are reserved for advertising shared credits and not for use as VCs).
    e. Flow Control Identifier (FC ID) is the flow control identifier for the packet header. For PCIe*, three FC IDs are possible - 2'b00 for Posted, 2'b01 for Non-Posted, and 2'b10 for Completions.
    f. All Reserved bits must be driven to 0 by TX and ignored by RX. Switches and fabric routers must propagate the reserved bits as-is without any modifications.
4. The HDR SIZE is specified in *HGRAN* granularity (in other words, the packet header formats must be defined in the granularity determined by the design parameter HGRAN. Typically this parameter is set to four bytes in the context of PCIe* protocol, and hence, HDR SIZE would indicate the packet header size in the number of DWORDs). The **hdr_info_bytes** bytes are not included in the HDR SIZE computation.

**Figure 4-1. Format for hdr_info_byte**

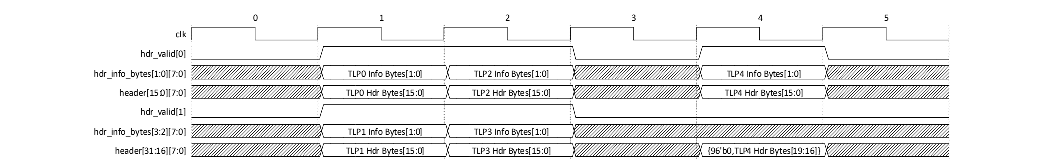


5. There must be a predetermined number of maximum packet headers that can be transmitted in one cycle, which are determined by the **header**field width (*H*) and the maximum packet header size. The **header** field width (*H*) must be chosen so as to allow the common case usage to sustain maximum throughput. Assume the common case application packet header size is 16 bytes (maps to four DWORD headers in PCIe*), and that we want to sustain two packet headers per cycle. Hence, *H* = *2*×(16) = 32 bytes. *M* must be 2 with **hdr_valid**[0] corresponding to **header** byte 0, and **hdr_valid**[1] corresponding to byte 16. If occasionally the packet header size is more than 16 bytes (as an example, if PCIe* TLP Prefix was used), then only one packet header can be transferred in that cycle, and **hdr_valid**[1] must not assert. For interoperability reasons, all TXs must support a configuration mode where they transfer only one packet header per cycle.
6. There must be a predetermined number of maximum FC and VC ID combinations allowed at a given transfer cycle. Both TX and RX must be aware of this at design compile time. A maximum of two implies that all the packet headers transferred on a given cycle can belong to two different FC within the same VC or the same FC for two different VC or all packet headers belong to the same FC and VC. This rule helps minimize the write ports in the storage of RX when buffers are shared between FC and VC. For interoperability reasons, all TXs must support a configuration mode where they transfer only one FC and VC combination per cycle.
7. If a packet header has data associated with it, the packet header is sent on the HDR channel and the associated data is sent on the DATA channel.
    a. TX must check for available credits for both header and DATA before scheduling either of the header or data transfers.
8. The credit granularity for packet headers is the maximum supported packet header size as well as any potential **data_trailers** or **data_suffix**. For example, if the maximum packet header size supported is 20 bytes, then 1 credit corresponds to 20 bytes worth of storage at the RX. Even if only a 16-byte packet header is sent, 1 credit is consumed corresponding to the full 20 bytes. For PCIe*, **data_trailer** bit is asserted to indicate the presence of PCIe* defined TLP trailer DW following the data payload. A couple of examples of this are mentioned next. For PCIe*, **data_suffix** bit is asserted to indicate the presence of TLP suffix following the data payload or trailer DWs.The space for trailer and suffix is accounted for as part of packet header credits.
    a. When End-to-End Cyclic Redundancy Check (ECRC) is supported, even though the ECRC bytes are sent on the DATA channel, the storage space for ECRC is included as part of the packet header credit consumed (similar to TLP digest in PCIe* protocol).
    b. If Integrity and Data Encryption (IDE) TLPs need to be sent across the interface, IDE prefixes are sent on the HDR channel, and the Message Authentication Code (MAC) bytes are sent on the DATA channel. The storage space for MAC and prefixes are included as part of the packet header credit (similar to PCIe* protocol rules).

### 4.2.1 Example Header Transfer

The waveform shown in Figure 4-2 demonstrates packet header transfers. Clock cycles 1 and 2 transfer two packet headers of size 4 (**HDR_SIZE** = 4) each, while clock cycle 4 transfers one packet header of size 5. In this example, **hdr_valid**[0] is always associated with header byte [0] and **hdr_valid**[1] is always associated with HDR byte[16].

**Figure 4-2. Example of Header Transfers**

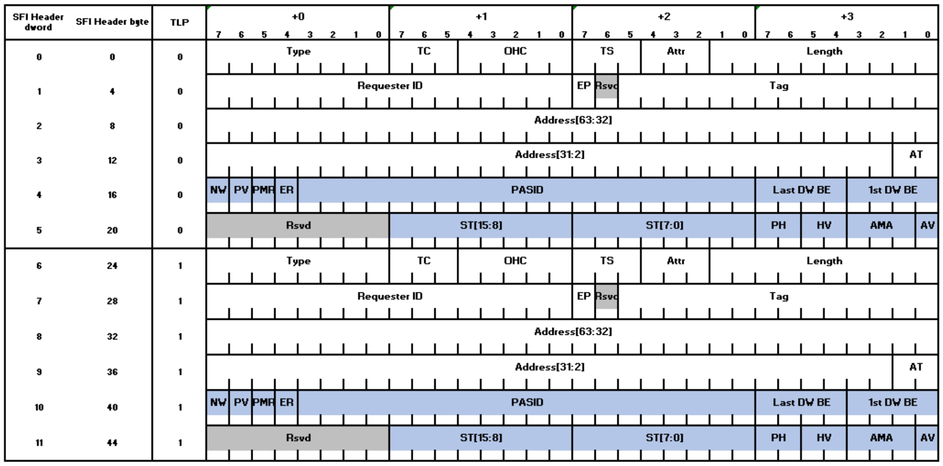


## 4.2.2 PCIe* and CXL.io* Packet Format Examples

As mentioned previously, PCIe* Flit Mode packet formats and byte assignments are followed for PCIe* protocol. Similarly, CXL.io* 256B Flit Mode packet formats and byte assignments are followed for CXL.io* protocol. This section presents examples and any additional rules associated with these formats when transporting over SFI. See the *PCI Express* (PCIe*) Base Specification* and *CXL* Base Specification* for the definitions of the individual fields within the headers.

The SFI parameter setting example in Table 6-2 is optimized for memory transactions using 6 DWORD PCIe* protocol headers (4 DWORDs base, 1 DW OHC-A, 1 DW of OHC-B). Figure 4-3 shows an example of header transfers using these parameter settings. In this example, the SFI transmitter can fit two 64-bit memory write PCIe* protocol headers with no prefixes and with OHC-A1 and OHC-B into a single SFI header. Because the protocol headers are the optimal size for this configuration, both TLP0 and TLP1 naturally align. **`hdr_valid[1:0]`** = 11b for Example 1.

**Figure 4-3. Example 1: Two 64-bit Memory Write Headers in one Clock Cycle**



However, it is possible that not every protocol header will be sized the same as the ideal utilization scenario in the context of the chosen parameter settings. In the example in Figure 4-4, the SFI transmitter is sending two 32-bit memory read PCIe* protocol headers with no prefixes and no OHC. Because the size of each protocol header is smaller than 6 DWORDs, the SFI transmitter must ensure that any subsequent protocol headers be placed at the interface on increments of 6 DWORDs. Any leftover SFI header space from the previous protocol headers ending location must be treated as reserved (driven to 0 by the transmitter and ignored by the receiver). **`hdr_valid[1:0]`** = 11b for Example 2.

### Figure 4-4. Example 2: Two 32-bit Memory Read Headers in One Clock Cycle

| SFI Header dword | SFI Header byte | TLP | +0 (7 6 5 4 3 2 1 0) | +1 (7 6 5 4 3 2 1 0) | +2 (7 6 5 4 3 2 1 0) | +3 (7 6 5 4 3 2 1 0) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | Type | TC OHC | TS Attr Length | |
| 1 | 4 | 0 | Requester ID | | EP Rsvd Tag | |
| 2 | 8 | 0 | Address[31:2] | | | AT |
| 3 | 12 | - | Rsvd | | | |
| 4 | 16 | - | Rsvd | | | |
| 5 | 20 | - | Rsvd | | | |
| 6 | 24 | 1 | Type | TC OHC | TS Attr Length | |
| 7 | 28 | 1 | Requester ID | | EP Rsvd Tag | |
| 8 | 32 | 1 | Address[31:2] | | | AT |
| 9 | 36 | - | Rsvd | | | |
| 10 | 40 | - | Rsvd | | | |
| 11 | 44 | - | Rsvd | | | |

On the other hand, if a protocol header size exceeds 6 DWORDs for this set of parameters, the SFI transmitter must ensure that it does not insert another protocol header in the same clock cycle. In Figure 4-5, the transmitter is sending a 64-bit memory write with a local vendor defined prefix and OHC-A1 and OHC-B for a total size of 7 DWORDs. Since this exceeds the 6 DWORD boundary, the SFI transmitter must only send one protocol header at that clock cycle. **`hdr_valid[1:0]`** = 01b for Example 3.

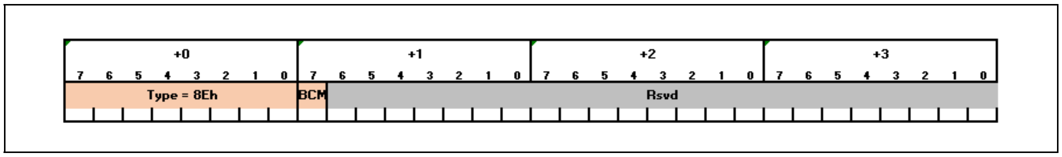
### Figure 4-5. Example 3: Header Transfer when Header Size Exceeds 6 DWORDs

| SFI Header dword | SFI Header byte | TLP | +0 (7 6 5 4 3 2 1 0) | +1 (7 6 5 4 3 2 1 0) | +2 (7 6 5 4 3 2 1 0) | +3 (7 6 5 4 3 2 1 0) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | Vendor Defined Local TLP Prefix | Vendor Defined | | |
| 1 | 4 | 0 | Type | TC OHC | TS Attr Length | |
| 2 | 8 | 0 | Requester ID | | EP Rsvd Tag | |
| 3 | 12 | 0 | Address[63:32] | | | |
| 4 | 16 | 0 | Address[31:2] | | | AT |
| 5 | 20 | 0 | NW PV PMR ER PASID | | | Last DW BE 1st DW BE |
| 6 | 24 | 0 | Rsvd | ST[15:8] | ST[7:0] | PH HV AMA AV |
| 7 | 28 | - | Rsvd | | | |
| 8 | 32 | - | Rsvd | | | |
| 9 | 36 | - | Rsvd | | | |
| 10 | 40 | - | Rsvd | | | |
| 11 | 44 | - | Rsvd | | | |

## 4.2.3 Additional Considerations

- The Byte Count Modified field does not have an allocated space in the PCIe* Flit Mode header formats. If this field is required, it must be sent over SFI using a Vendor defined local TLP prefix as shown in Figure 4-6. If this field is required, the parameter *BCM_EN* must be set to 1b, and all completion headers must carry this prefix. Additionally, this prefix must be the first DWORD for completion headers (i.e., other vendor defined prefixes and base header should come after this prefix).

Figure 4-6. Vendor Defined Prefix Used to Send BCM Field



- The "TLP Uses Dedicated Credits" field in Flit Mode local TLP Prefix from *PCI Express* (PCIe*) Base Specification* is not relevant for SFI, since shared vs dedicated credits is indicated via the "S" bit in **`hdr_info_bytes`**. Receivers must ignore this field if Flit Mode local TLP Prefix is present in the header. If *FLIT_MODE_PREFIX_EN* is 0, Transmitters must never send Flit Mode TLP prefix over SFI.

***Note:*** Currently, Flit Mode TLP prefix is only used to indicate dedicated credits over PCIe*/CXL.io* links and it is strongly recommended to set FLIT_MODE_PREFIX_EN=0. If PCIe* or CXL.io* add additional fields to Flit Mode TLP prefix, then it is permitted to set FLIT_MODE_PREFIX_EN to 1, but receivers must still ignore the "TLP Uses Dedicated Credits" field in that prefix and use the "S" bit in **`hdr_info_bytes`** to determine if the header used shared vs dedicated credits.

- Even when the PCIe* or CXL.io* Link itself trains to non-Flit mode configuration, the header formats on SFI follow the Flit Mode formats. It is the responsibility of the Transaction Layer to perform the necessary conversions before transmitting TLPs over the PCIe* or CXL.io* Link. See the *PCI Express* (PCIe*) Base Specification* for information related to how the field mappings change between Flit mode and non-Flit mode formats for PCIe*.

## 4.3 Data Layer

The DATA physical channel carries data for all requests that have data associated with it. The timing requirement between the HDR channel and the associated data coming on the DATA channel is determined by the parameters described in Section 6. It is required for TX to check both HDR channel and DATA channel credits before scheduling either the HDR or DATA on their respective channels.

Table 4-3. Fields of DATA Layer (Sheet 1 of 3)

| Signal Class | Signal Name | Width | Direction | Description |
|---|---|---|---|---|
| **VALID** | **`data_valid`** | 1 bit | TX → RX | Indicates that data corresponding to at least one packet is sent on the DATA signal class fields when asserted. If data associated with multiple packets are sent, they are disambiguated by the using the fields in the class |
| | **`data_block`** | 1 bit | RX → TX | This is an optional signal. It is an indication from RX to TX for a temporary stall to data transfers.<br>See Section 4.4.1 for rules related to this signal.<br>Applications include dynamic clock gating and/or temporary stalls due to clock crossing FIFOs in RX. |
| | **`data_early_valid`** | 1 bit | TX → RX | This is an optional signal. It is an early valid indication from TX to RX indicating that TX has packets to send on the DATA channel.<br>See Section 4.4.1 for rules related to this signal.<br>In conjunction with the **`data_block`** signal, it is used to enable dynamic clock gating for RX. |

**Table 4-3. Fields of DATA Layer (Sheet 2 of 3)**

| Signal Class | Signal Name | Width | Direction | Description |
|---|---|---|---|---|
| **DATA** | **data** | D bytes | TX → RX | Carries the raw data for associated packets. The raw data must be in 4-byte granularity. When DATA layer is present, D must be greater than or equal to 4. |
| | **data_parity** | D/8 bits | TX → RX | If *DATA_PARITY_EN* = 1, this signal is used to transfer data parity over the interface. Data parity for data transferred over a given clock cycle. It is computed for even parity of 1 parity bit over every 8B of **data**.<br>If **data_valid** = 1b, **data_parity**[n] is the XOR of all bits of data bytes [8×(n+1)-1: 8×n]; where “n = 0,1...D/8”. If 8×(n+1) > D, then data is 0 padded in the MSB for the purpose of parity computation. **data_parity** is computed regardless of **data_start** or **data_end** values.<br>At least one bit of **data_parity** must always exist if *DATA_PARITY_EN* = 1. If D <= 8, only one bit of **data_parity** is present.<br>Parity error handling/escalation is implementation/SoC specific. For example, it could be mapped to a fatal error, or RX could poison the protocol packets for the corresponding data which had a parity error. When not supported, it must be reserved; i.e., TX must drive 0, and RX must not check parity. |
| | **data_start** | DS bits | TX → RX | Indicates the start of a new data packet on the corresponding data byte. It is required to have a fixed association between the individual bits of **data_start** and the bytes of data. The parameter DS is determined by the maximum number of separate packets that need to be sustained per cycle. D must be a multiple of DS, and **data_start[i]** corresponds to **data** byte (**i**×D/DS). |
| | **data_info_byte** | DS×8 bits | TX → RX | For every **data_start** bit, there is a **data_info_byte**, which indicates the FC ID and the VC ID of the associated data packet. RX must not assume **data_info_byte** is valid if the corresponding **data_start** is not asserted. |
| | **data_end** | D/4 bits | TX → RX | Indicates the end of a new data packet on the corresponding data 4-byte chunk. It is required to have a fixed association between the individual bits of **data_end** and the bytes of data. Thus, **data_end[i]** corresponds to **data** bytes [(4×(**i+1)-1**):4×**i**]. |
| | **data_poison** | D/4 bits | TX → RX | Indicates one or more bytes of the associated data packet is poisoned. RX must treat the entire protocol packet payload corresponding to this data as poisoned. |
| | **data_edb** | D/4 bits | TX → RX | Indicates the packet associated with this data is bad and must be dumped by the RX. TX must have consumed the corresponding credits (both header and data credits), and hence, RX must return the corresponding credits after dumping the packet. This signal can be used to enable cut-through routing in switches and fabric as long as the final RX can guarantee dumping the packet before the commit point. |
| | **data_trailer** | D/4 bits | TX → RX | This is optional, only required if TLP trailers are supported. It should be asserted when the corresponding **data** bytes carry the TLP trailer DWORDs (for example ECRC or IDE MAC, and so on.). See the *PCI Express* (PCIe*) Base Specification* for details on TLP trailer definitions and support. |
| | **data_suffix** | D/4 bits | TX → RX | This is optional, only required if TLP suffixes are supported. When asserted, it indicates that the corresponding DWORD of data is a TLP suffix. See the *PCI Express* (PCIe*) Base Specification* for details on TLP suffix definitions and support. |
| | **data_aux_parity** | 1 bit | TX → RX | Optional signal. Whenever **data_valid** is 1b, **data_aux_parity** is the XOR of all the signals in DATA signal class, with the exception of **data** and **data_parity** signals. Parity errors on the **data_aux_parity** are fatal to interface operation and must be reported as fatal errors. When not supported, it must be reserved; i.e., TX must drive 0, and RX must not check parity. |

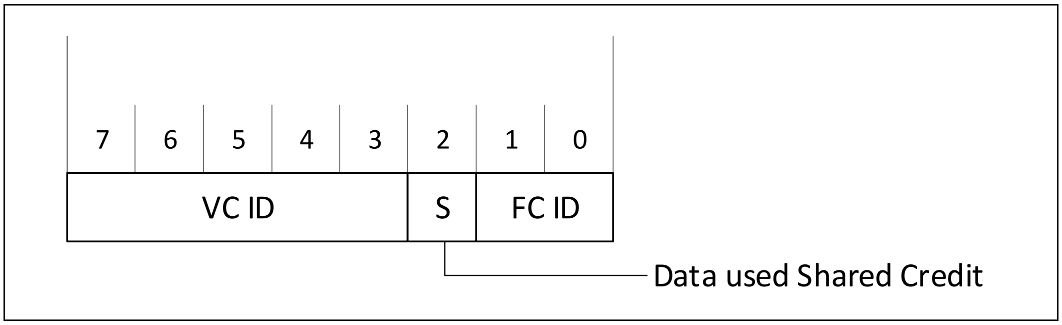
### Table 4-3. Fields of DATA Layer (Sheet 3 of 3)

| Signal Class | Signal Name | Width | Direction | Description |
|---|---|---|---|---|
| CREDIT | data_crd_rtn_valid | 1 bit | RX → TX | Credit return valid. |
| | data_crd_rtn_ded | 1 bit | RX → TX | Indicates the credit returns are to the dedicated pool of credits. If this is 0 on a valid credit return, the credits should be returned to the corresponding shared pool of credits. It must be set to 0 if **data_crd_rtn_vc_id** is 5'd30 or 5'd31. |
| | data_crd_rtn_fc_id | 2 bits | RX → TX | Identifies the flow control class for which data credits are being returned. For shared pool credit returns, it indicates the flow control of the transaction deallocated from RX buffers. |
| | data_crd_rtn_vc_id | 5 bits | RX → TX | Identifies the VC for which data credits are being returned. For shared pool credit returns, it indicates the VC of the transaction deallocated from RX buffers. |
| | data_crd_rtn_value | NDCRD bits | RX → TX | Indicates how many credits are being returned. Each credit corresponds to the chosen data granularity of storage in RX. |
| | data_crd_rtn_block | 1 bit | TX → RX | This is an optional signal. It is an indication from TX to RX for a temporary stall to credit returns.<br>See Section 4.4.1 for rules related to this signal.<br>Applications include dynamic clock gating and/or stalls due to clock crossing FIFOs. |

Rules for the DATA channel are outlined as follows:

1. The Data is always transferred in 4-byte granularity. The end positions of data are hence determined in 4-byte chunks. Thus, if *D = 64* (raw data is 64B), *D/4 = 16*, with **data_end**[0] corresponding to **data** bytes[3:0], **data_end**[1] corresponding to **data** bytes[7:4], and so on. **data_end** is used by RX to determine when a payload for a packet has finished transfer across SFI.
2. It is not required to have the same 4-byte granularity for **data_start**. An implementation limits the maximum number of starts in a cycle. For example, consider a 64-byte raw data bus (*D = 64),* and implementations want to limit the maximum number of starts in a cycle to 2. This means that *DS = 2*, and **data_start**[0] corresponds to **data** byte[0] and **data_start**[1] corresponds to **data** byte[32]. For interoperability reasons, all transmitters must have a configuration modewhere they only send bytes from at the most one data packet per cycle. When DS > 1, it is not required for transmitter to always start the first data packet at **data_start[0]** (since the receiver has to deal with data byte shifting to write into its storage buffers anyway, this allows the TX to streamline data relative to its storage buffer organization when applicable). RX must not consume unused data bytes at a given clock edge (with the exception of **data_parity** computation).
3. Every **data_start** bit has an associated **data_info_byte** with it that indicates the FC ID and VC ID of the corresponding packet. The format of **data_info_byte** is shown in Figure 4-7. FC ID and VC ID encodings are the same as those on the HDR channel.

**Figure 4-7. Format for data_info_byte**

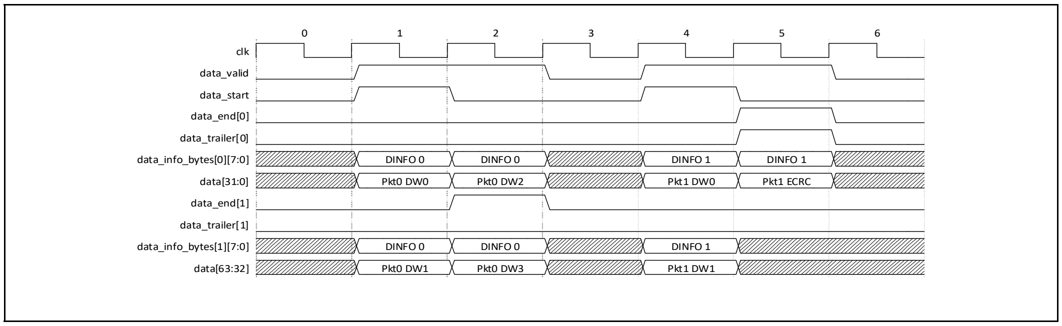


4. Unlike the HDR channel, data chunks from the same packet can be transferred over multiple cycles. For example, the raw data bus could be 64B, and a 128B data packet could be transferred over two clock cycles.
5. It is required for TX to maintain the same relative ordering of data packets as the associated headers for a given FC and VC combination.
6. RXs can optionally support data interleaving across different FC and VC combinations over different clock cycles. **`data_start`** must assert every time whenever the data stream interleaves different FC or VC; RX will use the **`data_info_byte`** along with **`data_start`** to assign the data stream to the appropriate VC/FC.This allows for low overhead time division multiplexed data path switching between two TXs that want to send packets to a common RX (with mutually exclusive VCs). This is the only time when multiple **`data_starts`** are permitted before a **`data_end`**. However, interleaving or bubbles in between a data transfer are not allowed at a given clock edge (i.e., it is not permitted to have multiple **`data_starts`** before a **`data_end`** on the same clock cycle for a given FC/VC).
7. The data credit granularity must be predetermined at design compile time between the TX and RX. The granularity must be a multiple of 4 bytes (DWORD). For example, if the credit granularity is chosen to be 16 bytes, then even a 4-byte data packet transferred uses one 16-byte worth of credit.

## 4.3.1 Example Data Transfer

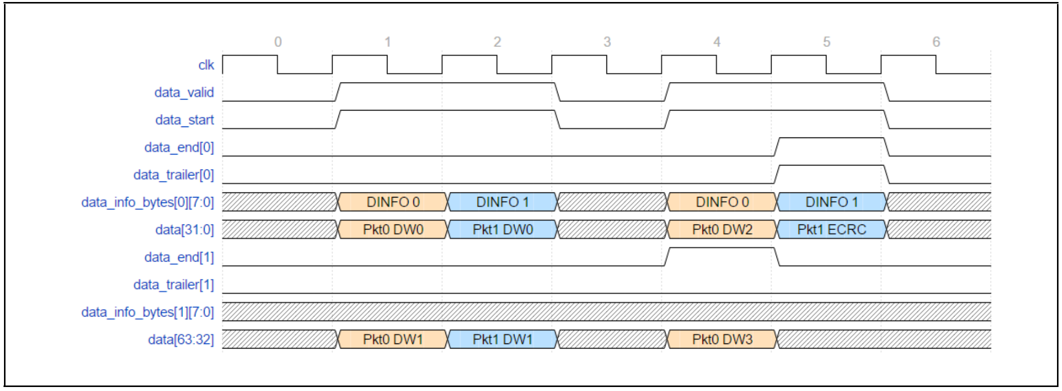
Figure 4-8 shows two examples of data transfer for a 64-bit data bus, and DS = 1. For Pkt0, it has four DWORDs of payload with no ECRC. For Packet 1, it has two DWORDs of payload and one DWORD of ECRC. In this example, TX chooses to drive DINFO along with valid data always.

**Figure 4-8. Example Data Transfer**



shows an example of data interleaving. The different colors belong to different FC, VC which are interleaved on the same data bus. In this example, TX chooses to only drive DINFO with corresponding **`data_start`**.

**Figure 4-9. Example of Data Interleaving**



# 4.4 Clocks and Resets

It is not required for TX or RX to coordinate reset. The initialization flow defines a separate handshake to ensure TX and RX exchange information about interface reset and flow control before traffic can begin. Power gating optimizations can be implemented by moving to a disconnected state.

SFI is a synchronous interface—both sides of the interface must run on the same clock. RXs of signals are permitted to instantiate clock crossing FIFOs internally, but must do so in a way that the rules of interface operation are not violated. SFI provisions for temporary stalling of packet transfers to enable dynamic clock gating and/or clock crossing related stalls.

## 4.4.1 Block and Early_valid rules

It is strongly recommended that **`*_block`** and **`*_early_valid`** signals be supported by implementations to ensure wider interoperability. The following rules apply when the **`early_valid`** and block signals are implemented.

1. Once **`hdr_block`** is asserted, the compile time parameter *TBN* determines the number of clock cycles after which TX must guarantee a stall (in other words, no more assertion of **`hdr_valid`**).

2. Once **hdr_block** deasserts, TX is permitted to begin header transfer from the next clock cycle.
3. Once **data_block** is asserted, the compile-time parameter *TBN* determines the number of clock cycles after which TX must guarantee a stall (in other words, no more assertion of **data_valid**).
4. Once **data_block** deasserts, TX is permitted to begin data transfer from the next clock cycle.
5. Once **hdr_crd_rtn_block** is asserted, the compile time parameter *RBN* determines the number of clock cycles after which RX must guarantee a stall (in other words, no more assertion of **hdr_crd_rtn_valid**).
6. Once **hdr_crd_rtn_block** deasserts, RX is permitted to begin header credit returns from the next clock cycle.
7. Once **data_crd_rtn_block** is asserted, the compile-time parameter *RBN* determines the number of clock cycles after which RX must guarantee a stall (in other words, no more assertion of **data_crd_rtn_valid**).
8. Once **data_crd_rtn_block** deasserts, RX is permitted to begin data credit returns from the next clock cycle.
9. TX must assert **hdr_early_valid** at least one cycle before **hdr_valid**. TX is permitted to keep **hdr_early_valid** asserted even if it has no immediate packets to transfer, if it wants to keep RX clocks running to achieve the best latency for imminent packet transfers.
10. TX must assert **hdr_early_valid** if it has packets to send, even if **hdr_block** signal is asserted. If **hdr_block** is asserted, TX must keep **hdr_early_valid** asserted until **hdr_block** deasserts and TX has transferred the packets. It is possible that **hdr_block** asserts and deasserts several times while packets are being transferred (for example, in cases of clock crossing related rate mismatches); TX must follow the rules related to **hdr_block** at all times.
11. TX must assert **data_early_valid** at least once cycle before **data_valid**. TX is permitted to keep **data_early_valid** asserted even if it has no immediate packets to transfer if it wants to keep RX clocks running to achieve the best latency for imminent packet transfers.
12. TX must assert **data_early_valid** if it has packets to send, even if **data_block** signal is asserted. If **data_block** is asserted, TX must keep **data_early_valid** asserted until **data_block** deasserts and TX has transferred the packets. It is possible that **data_block** asserts and deasserts several times while packets are being transferred (for example, in cases of clock crossing related rate mismatches); TX must follow the rules related to **data_block** at all times.

## 4.4.2 Dynamic Clock Gating

If dynamic clock gating is desired, implementations must support all of the *__block__ and *__early_valid__ signals.

From the SFI perspective, RX is permitted to transition to a clock gated state if the corresponding interface is in Disconnect state or all of the following conditions are true:

- The interface is in a Connected state, with no pending request from TX to Disconnect that RX has not responded to.
- **hdr_early_valid** and **data_early_valid** are deasserted from TX.
- No packet transfers are in flight (**hdr_valid** and **data_valid** are deasserted).

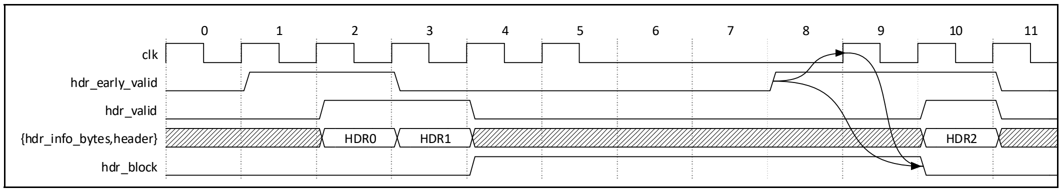
- RX has returned the minimum header and data credits required for a packet transfer on any of the supported VC and FC. The minimum credits are defined by the underlying protocol. For example, for PCIe* data credits on posteds, RX should have returned MPS worth data credits on all supported VCs.

RX must take into account any internal requirements for clock gating in addition to the SFI requirements before transitioning to a clock gated state. These internal requirements are implementation-specific and outside the scope of this specification.

Once the requirements are met, RX transitions to a clock gated state *TBN* cycles after asserting the **hdr_block** and **data_block** signals (if no packet transfers are received during that time, if packet transfers are received during this time, RX must abort the transition to a clock gated state and it is permitted to re-attempt at a later point if all the requirements for entry are met.)

Once in a clock gated state, RX must treat any transition on **hdr_early_valid**, **data_early_valid**, or **txcon_req** as an asynchronous trigger to exit the clock gated state. Once it has exited the clock gated state, it must revert to the regular protocol of accepting packets from TX. Figure 4-10 shows an example of entry and exit flow for the HDR channel. Similar handshakes would occur on the DATA channel as well.

### Figure 4-10. Example Signal Transitions on HDR Channel for Clock Gating Entry and Exit



From SFI perspective, TX is permitted to transition to a clock gated state if the corresponding interface is in Disconnect state or all of the following conditions are true:

- The interface is in a Connected state, with no pending request from TX to Disconnect that RX has not responded to.
- TX has no packet transfers pending.
- TX has accumulated the minimum header and data credits required for a packet transfer on any supported VC and FC. The underlying protocol defines the minimum credits.

TX must take into account any internal requirements for clock gating in addition to the SFI requirements before transitioning to clock gated state. These internal requirements are implementation specific and outside the scope of this specification.

Once the requirements are met, TX transitions to a clock gated state *RBN* cycles after asserting the **hdr_crd_rtn_block** and **data_crd_rtn_block** signals.

Exiting from a clock gated state for a TX is implementation-specific and outside the scope of this specification.

***Note:*** Implementation modules can have multiple SFI interfaces connected to them, where they may be RX or TX for different interfaces. Coarse level clock gating (shutting off global clock for example) for a module, should consider the corresponding RX or TX requirements of all SFI interfaces instantiated on that module.

## 4.5 Channel Flow Control

Each VC and FC must use credits on TX for sending any message and collect credit returns from the RX. The source should consume the full credits required for a message to complete. As mentioned earlier, TX must check for both HDR channel and DATA channel credits before sending either of them to the RX.

Each physical channel has dedicated credit return wires. During operation, the RX returns credits during initialization and whenever it has processed the message (in other words, guaranteed a buffer position for the next transaction).

To allow for batch processing at the RX, it is recommended to set the shared credit block size corresponding to the batch processing size at the RX. If credit sharing is enabled, shared credits must be allocated to an FC and VC in block granularity. For example, if *SH_DATA_CRD_BLK_SZ = 4*, if acquiring shared credits for a given FC and VC, it must be done in blocks of 4.

RX can advertise infinite credits by setting the **`*crd_rtn_value = 0`** for the corresponding VC and FC credit return. Typically, this is done after initialization if the RX can guarantee a spot to sink traffic from a given VC and FC (for example, if completions were preallocated). Once infinite credits are advertised for a VC and FC, there should be no credit returns for that VC and FC until the next disconnect or connect flow.

### 4.5.1 Sharing Credits

The SFI allows two schemes for supporting sharing of buffers between different FC and VC IDs. In both schemes, it is required for the RX to advertise the minimum number of dedicated resources needed for a forward progress guarantee. For large packet transfers, this means that the maximum payload size must be considered for dedicated credit advertisement. If *SHARED_CRD_EN=1*, it implies Type 1 scheme is used. For Type 2, from TX perspective, it looks the same as if shared crediting is disabled since the RX internally manages the shared resources. *SHARED_CRD_EN=0* is used for both the dedicated and Type 2 shared crediting schemes.

#### 4.5.1.1 Type 1: TX-Managed

In this scheme, the TX is responsible for managing shared buffers in the RX. RX advertises shared credits using SFI VC ID 30 and/or 31. Since up to 4 FC ID encodings are possible for each SFI VC, a total maximum of 8 shared pools can be supported as shown in Table 4-4.

**Table 4-4. Shared Pool to SFI VC and FC Mapping**

| Shared Pool | SFI FC ID | SFI VC ID |
|---|---|---|
| 0 | 0 | 30 |
| 1 | 1 | |
| 2 | 2 | |
| 3 | 3 | |
| 4 | 0 | 31 |
| 5 | 1 | |
| 6 | 2 | |
| 7 | 3 | |

**Note:** RX is permitted to only advertise credits for a subset of these. See Table 6-1 for parameters that determine the mapping.

It is permitted for RX to support fewer shared credit pools. Transactions on a given VC and FC can only use credits from one shared pool. When shared crediting using the Type 1 scheme is enabled, every SFI instance (a TX and RX pair) must define a mapping from each SFI VC and FC sharing credits to a shared pool of credits at design time. It is also permitted for an SFI VC and FC not to be mapped to any shared pool, in which case it is only going to use dedicated credits for that VC and FC.

Rules for TX:

- It must maintain a credit counter for dedicated credits for every VC and FC supported (*VCi_FCj_DED_CNTR_HDR, VCi_FCj_DED_CNTR_DAT* for VC "i" and FC "j"). Separate counters are maintained for header and data. The width of these counters must be sufficient to hold all the dedicated credits that can be advertised by the RX for the corresponding SFI VC and FC.
- It must maintain a credit counter for shared credits corresponding to the number of shared pools determined at design time (*SHRD_POOLx_CNTR_HDR, SHRD_POOLx_CNTR_DAT* for shared pool "x"). Separate counters are maintained for header and data. The width of these counters must be sufficient to hold all the shared credits that can be advertised by the RX for the corresponding shared pool. These counters track the shared pool credits available, and different VC and FC that share these credits have to arbitrate for these credits.
- It must maintain a counter for the number of **consumed** shared credits for every VC and FC sharing credits (*VCi_FCj_SHRD_USED_HDR, VCi_FCj_SHRD_USED_DAT* for VC "i" and FC "j"). Separate counters are maintained for header and data. The width of these counters must be sufficient to hold the maximum shared credits that can be used by a given SFI VC and FC. These set of counters effectively track the occupancy in the RX shared buffer for the corresponding VC and FC.
- It must maintain a counter for the number of shared credits **acquired** for every VC and FC sharing credits (*VCi_FCj_SHRD_ALLOC_HDR, VCi_FCj_SHRD_ALLOC_DAT* for VC "i" and FC "j"). Separate counters are maintained for header and data. The width of these counters must be sufficient to hold the maximum shared credits that can be acquired by a given SFI VC and FC. These are only incremented or decremented in block sizes and track how many shared credits have been allocated for a particular VC and FC, but not necessarily used to send traffic yet.

**Note:** By requiring both the Transmitter and Receiver to explicitly recognize credit blocks, the Receiver's buffer management logic is considerably simplified, while maintaining the efficient use of Receiver resources. Shared resource tracking at the receiver typically requires linked-list structures, and by organizing them in blocks, it gives a much easier way to scale bandwidth to process multiple TLPs per cycle per VC/FC. In addition, the linked-list management logic can be implemented at a block level - reducing the storage requirements of the linked-list (for example, for 1024 entries with a block size of 4, only 256 block pointers are needed from the point of view of linked-list tracking, a 4x reduction).

- It must instantiate a configuration register (*VCi_FCj_MAX_HDR, VCi_FCj_MAX_DAT*) for every SFI VC and FC mapped to a shared pool of credits. Separate registers are needed for header and data. The value programmed in this register determines the maximum outstanding shared credits acquired by the corresponding SFI VC and FC. Software can use this mechanism to limit the maximum occupancy of the shared pool by a given SFI VC/FC for Quality of Service (QoS) guarantees.

- If an SFI VC and FC are mapped to a shared pool, TX must first check for shared credit availability. Any given transaction will need one header credit and "y" data credits (where the number of data DWORDs associated with the transaction is greater than (y-1)*DATA_CRED_GRAN, but less than y*DATA_CRED_GRAN). "y" will be 0 for transactions that do not have data associated with them. Shared credits are available and used in one of the following two scenarios:

a. **Case 1: Credits acquired, but not used yet.**

*(VCi_FCj_SHRD_ALLOC_HDR - VCi_FCj_SHRD_USED_HDR) >= 1, AND*

*(VCi_FCj_SHRD_ALLOC_DAT - VCi_FCj_SHRD_USED_DAT) >= y, AND*

*VCi_FCj_SHRD_USED_HDR + 1 <= VCi_FCj_MAX_HDR, AND*

*VCi_FCj_SHRD_USED_DAT + y <= VCi_FCj_MAX_DAT.*

TX can schedule the transaction after incrementing *VCi_FCj_SHRD_USED_DAT* by "y" and *VCi_FCj_SHRD_USED_HDR* by 1.

b. **Case 2: Acquiring credits from Shared Pool "x" for VC "i" and FC "j"**
In this case, the transmitter is arbitrating for acquiring a block of shared credits. If *(VCi_FCj_SHRD_ALLOC_DAT - VCi_FCj_SHRD_USED_DAT)* is less than “y”, TX would need to acquire data shared credits. If (*VCi_FCj_SHRD_ALLOC_HDR - VCi_FCj_SHRD_USED_HDR)* is less than “1” TX would need to acquire header shared credits. Thus, TX might attempt to acquire data shared credits, or header shared credits, or both header and data shared credits.

For acquiring header shared credits:

*(VCi_FCj_SHRD_ALLOC_HDR - VCi_FCj_SHRD_USED_HDR) < 1, AND*

*VCi_FCj_SHRD_USED_HDR + 1 <= VCi_FCj_MAX_HDR, AND*

*SHRD_POOLx_CNTR_HDR >= SH_HDR_CRD_BLK_SZ, AND*

VC "i" and FC "j" can arbitrate for shared credits, once it wins arbitration, TX can schedule transaction after the following actions:

a. Increment *VCi_FCj_SHRD_USED_HDR* by 1
b. Increment *VCi_FCj_SHRD_ALLOC_HDR* by *SH_HDR_CRD_BLK_SZ*
c. Decrement *SHRD_POOLx_CNTR_HDR* by *SH_HDR_CRD_BLK_SZ*

For acquiring data shared credits:

*(VCi_FCj_SHRD_ALLOC_DAT - VCi_FCj_SHRD_USED_DAT) < y, AND*

*VCi_FCj_SHRD_USED_DAT + y <= VCi_FCj_MAX_DAT, AND*

*SHRD_POOLx_CNTR_DAT >= (ceiling(y/SH_DATA_CRD_BLK_SZ)× SH_DATA_CRD_BLK_SZ).*

VC "i" and FC "j" can arbitrate for shared credits, once it wins arbitration, TX can schedule transaction after the following actions:

a. increment *VCi_FCj_SHRD_USED_DAT* by "y"
b. increment *VCi_FCj_SHRD_ALLOC_DAT* by (*ceiling(y/SH_DATA_CRD_BLK_SZ )×SH_DATA_CRD_BLK_SZ*)
c. decrement *SHRD_POOLx_CNTR_DAT* by (*ceiling(y/ SH_DATA_CRD_BLK_SZ)×SH_DATA_CRD_BLK_SZ*)

Arbitration scheme for shared credits is implementation-specific, but TX must ensure fairness.

- When the TX consumes the shared pool credit for a transaction, it must do so for both header and data credits, and it indicates this on the "S" bit in the metadata for both header and data channels (by setting it to 1b)
- If TX is not able to use shared credits for a sustained time period (this mechanism is implementation-specific), it must use dedicated credits under the following conditions:

*VCi_FCj_DED_CNTR_HDR >= 1, AND*

*VCi_FCj_DED_CNTR_DAT >= y*

TX can schedule the transaction after decrementing *VCi_FCj_DED_CNTR_HDR* by 1 and *VCi_FCj_DED_CNTR_DAT* by "y".

- When TX consumes a dedicated pool credit, it must do so for both header and data credits, and it indicates this to the RX by setting the "S" bit to 0 in the metadata for both header and data channels.
- The following rules apply for credit returns after a successful connection:
  a. If **hdr_crd_rtn_valid = 1** and **hdr_crd_rtn_vc_id = 5'd30 or 5'd31**, TX must increment the corresponding *SHRD_POOLx_CNTR_HDR* ("x" is determined from the mapping given in Table 4-4) by **hdr_crd_rtn_value**.
  b. else, if **hdr_crd_rtn_valid = 1, hdr_crd_rtn_ded = 1, hdr_crd_rtn_vc_id** = i, and **hdr_crd_rtn_fc_id** = j, TX must increment the corresponding *VCi_FCj_DED_CNTR_HDR* counter by **hdr_crd_rtn_value**.
  c. else, if **hdr_crd_rtn_valid = 1**, **hdr_crd_rtn_ded = 0**, **hdr_crd_rtn_vc_id** = i, and **hdr_crd_rtn_fc_id** = j, TX must increment the corresponding *SHRD_POOLx_CNTR_HDR* ("x" is determined based on which shared pool VC "i" and FC "j" are mapped to) by **hdr_crd_rtn_value**. It must also decrement *VCi_FCj_SHRD_USED_HDR* and *VCi_FCj_SHRD_ALLOC_HDR* by **hdr_crd_rtn_value**.
  d. If **data_crd_rtn_valid = 1** and **data_crd_rtn_vc_id = 5'd30 or 5'd31**, TX must increment the corresponding *SHRD_POOLx_CNTR_DAT* ("x" is determined from the mapping given in Table 4-4) by **data_crd_rtn_value**.
  e. else, if **data_crd_rtn_valid = 1**, **data_crd_rtn_ded = 1**, **data_crd_rtn_vc_id** = i, and **data_crd_rtn_fc_id** = j, TX must increment the corresponding *VCi_FCj_DED_CNTR_DAT* counter by **data_crd_rtn_value**.
  f. else, if **data_crd_rtn_valid = 1**, **data_crd_rtn_ded = 0**, **data_crd_rtn_vc_id** = i, and **data_crd_rtn_fc_id** = j, TX must increment the corresponding *SHRD_POOLx_CNTR_DAT* ("x" is determined based on which shared pool VC "i" and FC "j" are mapped to) by **data_crd_rtn_value**. It must also decrement *VCi_FCj_SHRD_USED_DAT* and *VCi_FCj_SHRD_ALLOC_DAT* by **data_crd_rtn_value**.
- After disconnect/reset, TX must assign all the counters (*CNTR*, *ALLOC*, *USED*) to 0 before initiating a connect by asserting **txcon_req**.

Rules for the RX:

- As part of any connection flow, RX must advertise appropriately dedicated and shared pool credits. Credits are advertised as credit returns after **rxcon_ack** is asserted.
  a. Dedicated credits for a VC/FC are advertised by setting ***_crd_rtn_ded** = 1 for corresponding header or data credit returns. At least one dedicated header credit must be advertised for every supported SFI VC/FC (with the exception of

all FCs for VC 5’d30 and 5’d31, since these are only used to advertise shared credits). At least maximum payload size worth of data credits must be advertised for every supported SFI VC or FC. The maximum payload size per SFI VC/FC is a design-time determination and must adhere to the underlying protocol requirements.

    b. Shared credits for a pool are advertised by returning credits on the corresponding shared pool VC (in other words, VC ID 5’d30 or 5’d31) for both header and data. **`*_crd_rtn_ded`** `= 0` for these credit returns. This is required at every connect flow since there are no transactions in the RX buffer. RX must advertise sufficient shared credits to meet a given SFI instance's performance goals for the worst-case credit loop delays and QoS requirements across different SFI VCs.

- During a connected state, while traffic is flowing from TX to RX, credits are returned as transactions are deallocated from the RX buffers of RX. RX tracks the "S" bit from the metadata of a transaction to know whether to return a shared credit or dedicated credit for that transaction. Shared credits must only be returned when the corresponding block of entries has de-allocated (i.e., TX must have used the credits from a block, and the entire block must have de-allocated at RX). Header credits can be returned when the corresponding header block deallocates, it does not have to wait for the corresponding data to deallocate. Data credits can be returned when the corresponding data block bytes deallocate, it does not have to wait for the corresponding header to deallocate. RX must take DATA_CRED_GRAN into account for returning data credits. In other words, it must return one credit for every 4*DATA_CRED_GRAN bytes of data. RX is allowed to accumulate multiple credit returns to use the credit return bus efficiently.

    a. Dedicated credits are returned for a VC/FC by setting **`*_crd_rtn_ded`** `= 1` for the credit returns corresponding to that VC/FC.

    b. Shared credits are returned by setting **`*_crd_rtn_ded`** `= 0` for the credit returns. The VC ID and FC ID is set corresponding to the VC ID and FC ID of the block that deallocated from the RX buffers. (Besides the initial advertisement of shared credits, shared credit returns do not carry the VC ID of 5’d30 or 5’d31, but rather they carry the VC/FC ID of the transaction being deallocated from the RX buffers, this allows TX to manage the occupancy of shared buffers per VC/FC sharing the buffer and guarantee bandwidth QoS).

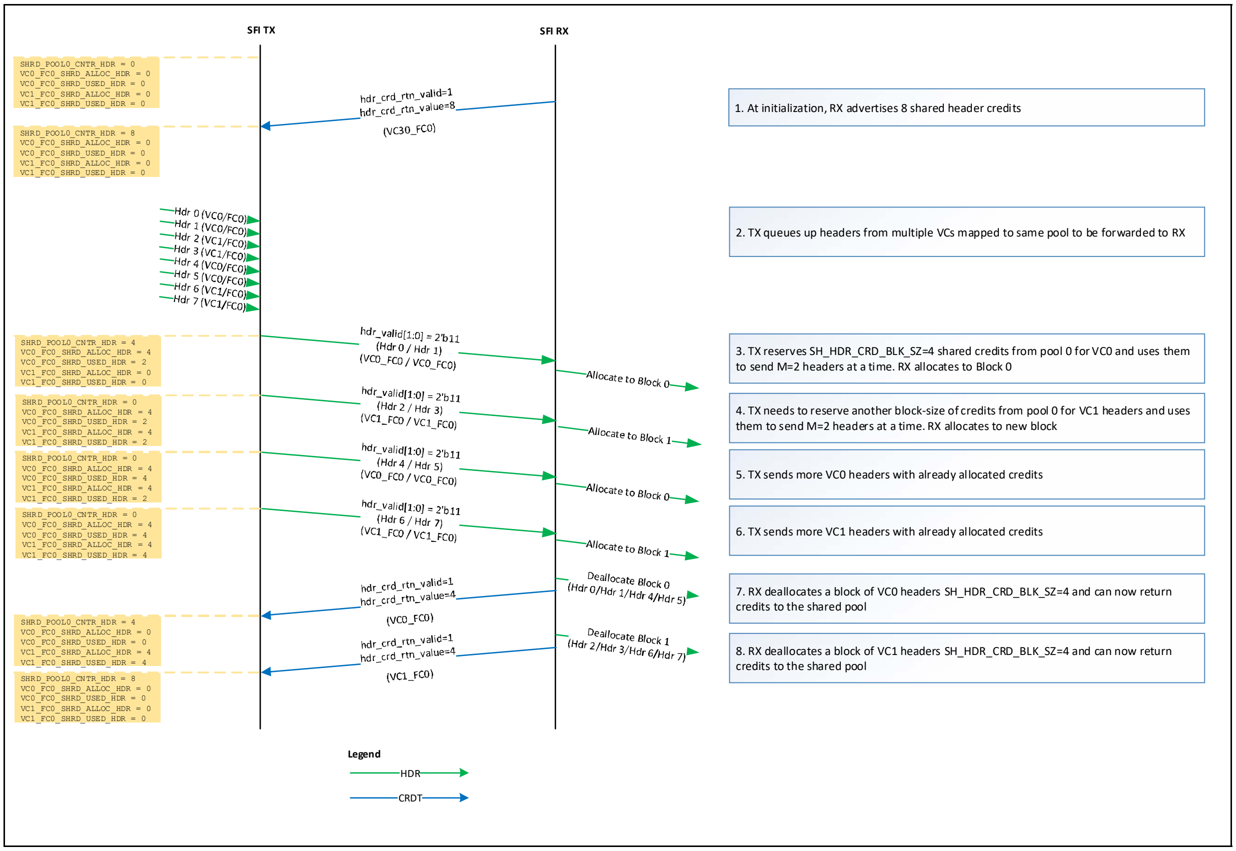
This is an example mapping of two shared credit pools advertised by an RX for PCIe* usage that support two VCs on the link:

- SFI VC encoding 5'd0: maps to PCIe* advertised VC0 on the link
- SFI VC encoding 5'd1: maps to PCIe* advertised VC1 on the link
- FC encoding 5'd0: Posted (P)
- FC encoding 5'd1: Non-Posted (NP)
- FC encoding 5'd2: Completions (C)
- VC encoding 5'd30, FC encoding 4'd0: Shared credit pool 1, all VCs, P or C can use this
- VC encoding 5'd30, FC encoding 4'd1: Shared credit pool 2, all VCs, NP can use this

### 4.5.1.1.1 Type 1 Shared Credit Example

Figure 4-11 gives a simple example of how shared credits are used by the transmitter, and released by the receiver for header transfer across SFI. The transmitter gating function around **MAX_HDR* is not shown for this example.

### Figure 4-11. Example Flow for Shared Credit Allocation and De-Allocation


SFI TX
SFI RX
SHRD_POOL0_CNTR_HDR = 0
VC0_FC0_SHRD_ALLOC_HDR = 0
VC0_FC0_SHRD_USED_HDR = 0
VC1_FC0_SHRD_ALLOC_HDR = 0
VC1_FC0_SHRD_USED_HDR = 0
hdr_crd_rtn_valid=1
hdr_crd_rtn_value=8
(VC30_FC0)
SHRD_POOL0_CNTR_HDR = 8
VC0_FC0_SHRD_ALLOC_HDR = 0
VC0_FC0_SHRD_USED_HDR = 0
VC1_FC0_SHRD_ALLOC_HDR = 0
VC1_FC0_SHRD_USED_HDR = 0
Hdr 0 (VC0/FC0)
Hdr 1 (VC0/FC0)
Hdr 2 (VC1/FC0)
Hdr 3 (VC1/FC0)
Hdr 4 (VC0/FC0)
Hdr 5 (VC0/FC0)
Hdr 6 (VC1/FC0)
Hdr 7 (VC1/FC0)
SHRD_POOL0_CNTR_HDR = 4
VC0_FC0_SHRD_ALLOC_HDR = 4
VC0_FC0_SHRD_USED_HDR = 2
VC1_FC0_SHRD_ALLOC_HDR = 0
VC1_FC0_SHRD_USED_HDR = 0
hdr_valid[1:0] = 2'b11
(Hdr 0 / Hdr 1)
(VC0_FC0 / VC0_FC0)
Allocate to Block 0
SHRD_POOL0_CNTR_HDR = 0
VC0_FC0_SHRD_ALLOC_HDR = 4
VC0_FC0_SHRD_USED_HDR = 2
VC1_FC0_SHRD_ALLOC_HDR = 4
VC1_FC0_SHRD_USED_HDR = 2
hdr_valid[1:0] = 2'b11
(Hdr 2 / Hdr 3)
(VC1_FC0 / VC1_FC0)
Allocate to Block 1
SHRD_POOL0_CNTR_HDR = 0
VC0_FC0_SHRD_ALLOC_HDR = 4
VC0_FC0_SHRD_USED_HDR = 4
VC1_FC0_SHRD_ALLOC_HDR = 4
VC1_FC0_SHRD_USED_HDR = 2
hdr_valid[1:0] = 2'b11
(Hdr 4 / Hdr 5)
(VC0_FC0 / VC0_FC0)
Allocate to Block 0
SHRD_POOL0_CNTR_HDR = 0
VC0_FC0_SHRD_ALLOC_HDR = 4
VC0_FC0_SHRD_USED_HDR = 4
VC1_FC0_SHRD_ALLOC_HDR = 4
VC1_FC0_SHRD_USED_HDR = 4
hdr_valid[1:0] = 2'b11
(Hdr 6 / Hdr 7)
(VC1_FC0 / VC1_FC0)
Allocate to Block 1
Deallocate Block 0
(Hdr 0/Hdr 1/Hdr 4/Hdr 5)
hdr_crd_rtn_valid=1
hdr_crd_rtn_value=4
(VC0_FC0)
SHRD_POOL0_CNTR_HDR = 4
VC0_FC0_SHRD_ALLOC_HDR = 0
VC0_FC0_SHRD_USED_HDR = 0
VC1_FC0_SHRD_ALLOC_HDR = 4
VC1_FC0_SHRD_USED_HDR = 4
Deallocate Block 1
(Hdr 2/Hdr 3/Hdr 6/Hdr 7)
hdr_crd_rtn_valid=1
hdr_crd_rtn_value=4
(VC1_FC0)
SHRD_POOL0_CNTR_HDR = 8
VC0_FC0_SHRD_ALLOC_HDR = 0
VC0_FC0_SHRD_USED_HDR = 0
VC1_FC0_SHRD_ALLOC_HDR = 0
VC1_FC0_SHRD_USED_HDR = 0
Legend
HDR
CRDT

1. At initialization, RX advertises 8 shared header credits

2. TX queues up headers from multiple VCs mapped to same pool to be forwarded to RX

3. TX reserves SH_HDR_CRD_BLK_SZ=4 shared credits from pool 0 for VC0 and uses them to send M=2 headers at a time. RX allocates to Block 0

4. TX needs to reserve another block-size of credits from pool 0 for VC1 headers and uses them to send M=2 headers at a time. RX allocates to new block

5. TX sends more VC0 headers with already allocated credits

6. TX sends more VC1 headers with already allocated credits

7. RX deallocates a block of VC0 headers SH_HDR_CRD_BLK_SZ=4 and can now return credits to the shared pool

8. RX deallocates a block of VC1 headers SH_HDR_CRD_BLK_SZ=4 and can now return credits to the shared pool

### 4.5.1.2 Type 2: RX-Managed

In this scheme, the RX is responsible for managing shared buffers. Only the dedicated credits are advertised to the TX. Typically, the advertised dedicated credits cover the point-to-point credit loop across the SFI, and the shared credits are used to cover the larger credit loops (for example the CPU fabric or Application Layer latencies). After a particular FC/VC ID transaction is received and shared credits are available, a credit can be returned for that FC/VC ID (without waiting for the transaction to deallocate from the RX queue). This implicitly gives a shared buffer spot for that FC/VC ID. Internally, the RX must track the credits returned to TX per FC/VC and the credits currently consumed by TX. With this tracking, RX can ensure the maximum number of buffers used per FC/VC. RX must guarantee the required dedicated resources for forward progress guarantee.

## 4.5.2 Flow Control Error Handling

Error handling for illegal flow control cases results in undefined behavior. It is recommended that agents and fabric check for illegal cases to trigger assertions in Register-Transfer Level (RTL) and also log or signal fatal errors to allow for post-silicon debug. It is expected that HDR and DATA streams are always consistent with each other, meaning that if TX must send the DATA in the same order, it is sending the corresponding headers and vice versa. It is strongly recommended for RX to flag fatal errors for violations.

# 5 Connect and Disconnect

This section describes the connect, and the disconnect flows for SFI. Flows are invoked during boot or reset and when going into a low power mode.

**Connect Flow**

SFI defines an initialization phase where information about credit availability in the RX is communicated to the TX after a connection is established. It is permitted for Reset to independently de-assert between the TX and RX sides of SFI. For independent reset, the initialization signals are driven to the disconnected condition when in reset, and no traffic is sent until initialization reaches the connected state.

**Disconnect Flow**

Agents optionally support the disconnect flow with the following primary usage models: reconfiguring credits and power saving. Without this flow, all SFI credits must be configured to a final value before the first connection can proceed. It is recommended that the Disconnect flow is co-ordinated through higher-level SoC or firmware involvement to facilitate optimal usage and avoid race conditions. As an example, one implementation could involve an SoC aggregator collect sideband idle indications from TX and RX before triggering TX to initiate the disconnect flow for power gating.

One end of SFI (after coming out of reset) does not have implicit requirements for when the other end should come out of reset. An explicit handshake mechanism during initialization ensures that both endpoints (and all pipeline stages between them) are out of reset before any credits or transactions being sent on SFI.

After connection, RX sends credits for dedicated VC buffers and shared buffers.

## 5.1 Initialization States

Initialization states are defined based on the following three wires:

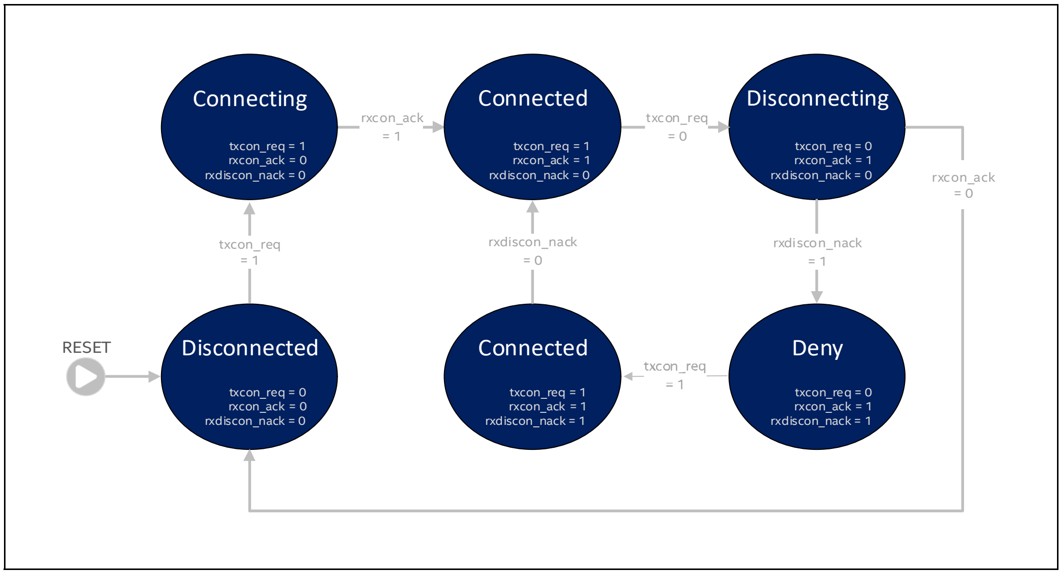
- **`txcon_req`**
- **`rxcon_ack`**
- **`rxdiscon_nack`**

The state is used to determine actions necessary on the RX and TX of the SFI as defined in Table 5-1. The basic state transitions are shown in Figure 5-1 and detailed rules for connect and disconnect flows are captured in subsequent sections.

### Table 5-1. Initialization State Actions

| txcon_req | rxcon_ack | rxdiscon_nack | State | TX Actions | RX Actions |
|---|---|---|---|---|---|
| 1 | 0 | 0 | Connection request (connecting) | Sink credits<br>Do not send packets | Do not send credits<br>Do not sink packets |
| 1 | 1 | 0/1 | Connected | Sink credits<br>Send packets | Send credits<br>Sink packets |
| 0 | 1 | 0 | Disconnection request (disconnecting) | Sink credits<br>Do not send packets | |
| 0 | 1 | 1 | Deny (disconnect rejected) and must go back to connected | | |
| 0 | 0 | 0 | Disconnected | Drop credits<br>Do not send packets | Do not send credits<br>Do not sink packets |
| 1/0 | 0 | 1 | Illegal states | n/a | n/a |

### Figure 5-1. Initialization State Machine



## 5.2 Signaling Rules

1. **txcon_req** signal use:
   0 to 1 → Connection request
   1 to 0 → Disconnection request
2. Credit return behaves similarly during the first initialization of credits as it does during the runtime return of credits.
3. The **rx_empty** signal indicates all channel credits returned from the RX and all RX queues are empty. This does not account for messages in flight or in intermediate buffers such as clock crossing queues.

a. During initialization, if Disconnect is not supported, TX is permitted to send messages as soon as any credits are available and not depend on **rx_empty** assertion. if Disconnect is supported and *TX_CRD_REG=0*, TX must stall the sending of any packets after connect until **rx_empty** is asserted. It must use the credits received as an indication of the total credits a RX has advertised.

***Note:*** 3 (a) implies a timing relationship between **rx_empty** and the credit return signals. Specifically, the assumption is that implementations guarantee **rx_empty** cannot race ahead of credit return signals. This is trivial in designs that do not support clock crossings or dynamic clock gating. When clock crossings or dynamic clock gating is supported (i.e., scenarios where ***crd_rtn_block** can assert), it is required that TX implementations expose configuration registers that can be programmed and/or reset to indicate the maximum credits RX can advertise (this avoids several race conditions related to in flight credit returns, idle determination and disconnect flow). *TX_CRD_REG* is the parameter that indicates support for this. These configuration registers (also referred to as credit registers) are programmed through implementation-specific sideband mechanisms. Suppose credit reconfiguration post reset is not required. In that case, it is permitted to set the reset defaults of these registers to be the maximum credits RX will advertise, and no further programming is required. However, if relying on software or firmware programming, TX must expose a ProgrammingDone bit with default value 0, and wait for software or firmware to write 1 to it before consuming the values in the credit registers.

4. TX is permitted to send packets when it receives enough credits from the RX. It must make sure there are sufficient HDR and Data credits for a given packet before it transmits on either channel.
5. Connection Ack always follows connection Req.
    a. Req is signaled by **txcon_req** transitioning from 0 → 1. This transition is indication the TX is ready to receive credits and is in normal operation.
    b. Ack is signaled with **rxcon_ack** transitioning from 0 → 1. Ack might be stalled until RX is ready to complete.
6. Disconnect Ack or NAck follows Disconnect Req.
    a. Disconnect Req is signaled by a **txcon_req** transition from 1 → 0.
    b. Disconnect Ack is signaled by a **rxcon_ack** transition from 1 → 0.
    c. Disconnect NAck is signaled by a **rxdiscon_nack** transition from 0 → 1.
    d. RX must select Ack versus NAck for each **disconnect** request. The response time is implementation-specific, but must not cause system level timeouts, and so on.

## 5.3 Reset and Connect Flow

Figure 5-2 shows the initialization flow, and the accompanying text focuses only on the initialization description. An interface is considered IDLE if either of the following conditions is true:

1. The Interface is in a Disconnect state with no outstanding connection request from TX. Credit avail counters of TX are 0. This is the same as Reset state.
2. The Interface is in a Connected state, and there are no outstanding transactions from TX. Credit avail counters would have the maximum advertised credits from RX.

**Steps for connection:**

1. After TX is out of reset, it asserts **txcon_req** to RX.
2. After RX is out of reset, it waits for **txcon_req** from TX.

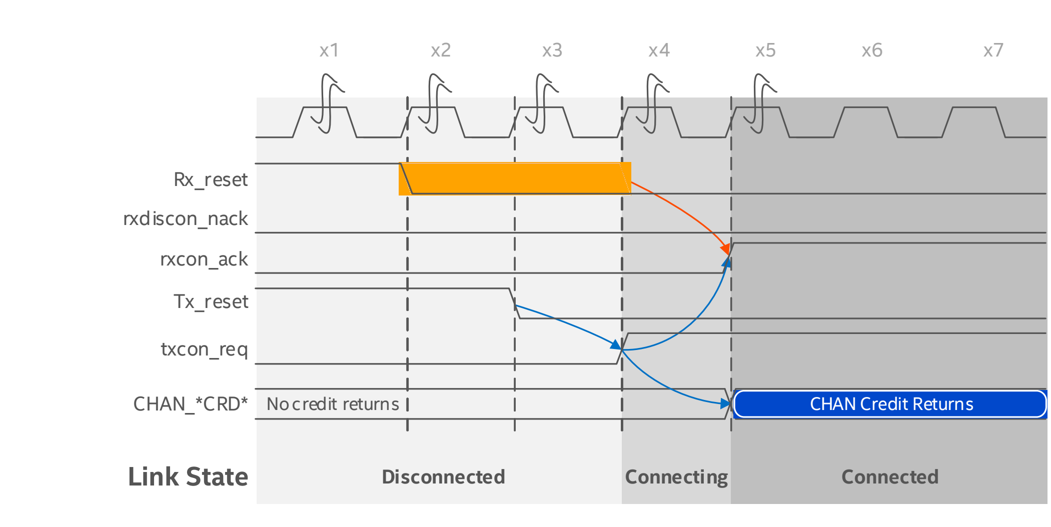
3. After RX receives **txcon_req**, it asserts **rxcon_ack** to TX. **rxcon_ack** must assert at least one cycle after **txcon_req** asserts.
4. After RX asserts **rxcon_ack** to TX, it can immediately start returning credits.
5. After the minimum credits are received to send packets, TX can start sending packets (provided conditions outlined in rule 3 [a] of Section 5.2 are met).

Table 5-2 lists each major time marker from Figure 5-2.

### Table 5-2. Reset Time Marker Description

| Time Marker | Description |
|---|---|
| X1 | TX and RX are in reset. |
| X2 | RX reset might de-assert. |
| X3 | TX reset de-asserts, allowing it to start initialization. |
| X4 | TX asserts **txcon_req**. This can be an arbitrary number of cycles after reset. Until the connection is complete, this signal must remain asserted and can only de-assert as part of the disconnect flow.<br>After asserting **txcon_req**, the TX must be able to accept credit returns because credit returns might be observed before observation of **rxcon_ack** due to intermediate pipeline stage differences. |
| X5 | **rxcon_ack** asserts after both RX reset de-asserts and **txcon_req** asserts. A fixed number of cycles is not required between X4 and X5. **rxcon_ack** can only de-assert as part of the disconnect flow. |
| X5 | RX can start returning credits simultaneously with the assertion of **rxcon_ack**. There are no strict expectations on flight time or observation of when TX observes the Ack relative to the credits. |

### Figure 5-2. Reset to Connected Timing Diagram



## 5.4 Disconnect and Reconnect Flow

As mentioned previously, it is recommended that implementations coordinate idle conditions through higher level SoC or firmware assistance for triggering a disconnect flow. This helps simplify hardware implementations. In addition, TX must ensure it has received all the expected credits from RX and have no transactions in flight before initiating a Disconnect flow. If *TX_CRD_REG* is 1, TX knows the maximum expected credit returns from RX via the credit registers. If *TX_CRD_REG* is 0, implementations guarantee the timing relationships between **rx_empty** and credit return signals as outlined in Section 5.2 rule 3 (a).

The following steps refer to Figure 5-3 and reference the timestamps in that diagram.

1. TX de-asserts **txcon_req** to disconnect at time x3.
   a. **rxdiscon_nack** must be de-asserted before **txcon_req** de-assertion.
   b. TX must not be sending messages on any channel.
2. RX must decide to acknowledge (Ack) or negatively acknowledge (NAck or reject) the disconnect.
   a. To Ack, RX de-asserts **rxcon_ack** after ensuring all pipelines are empty at time x4 in Figure 5-3, which marks the entry into a disconnected state. Optionally, it can also ensure that all credits have been returned.
   b. To NAck, the RX asserts the **rxdiscon_nack** at time x4 in Figure 5-4. It might choose to do this if it is unable to drain its pipelines without risking deadlock.
      - After NAck, the **txcon_req** must re-assert as shown at time x5 in Figure 5-4.
      - After that is observed, the **rxdiscon_nack** can de-assert at time x6 in Figure 5-4.
3. The Reconnect flow is the same as the Reset flow with the following exceptions:
   a. To start a new credit initialization, RX must reset its credit counters to reset values.
   b. TX resets its **credit_avail** counters to zero before asserting **txcon_req**. If *TX_CRD_REG = 1*, TX must reset the credit registers and if applicable, the ProgrammingDone bit must be reset to 0 as well.

***Note:*** The connect and disconnect flows are expected to complete within a few microseconds after initiation, but no timeout is explicitly defined. To meet this expectation for disconnect, the RX should reply Ack or NAck within this time window. The Agent, Fabric, or SoC can define a timeout requirement to ensure this.

**Figure 5-3. Connect to Disconnect to Connected Flow**

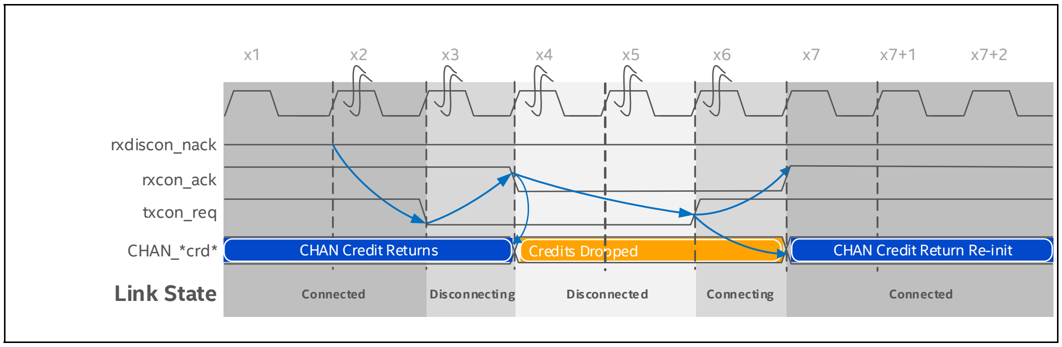
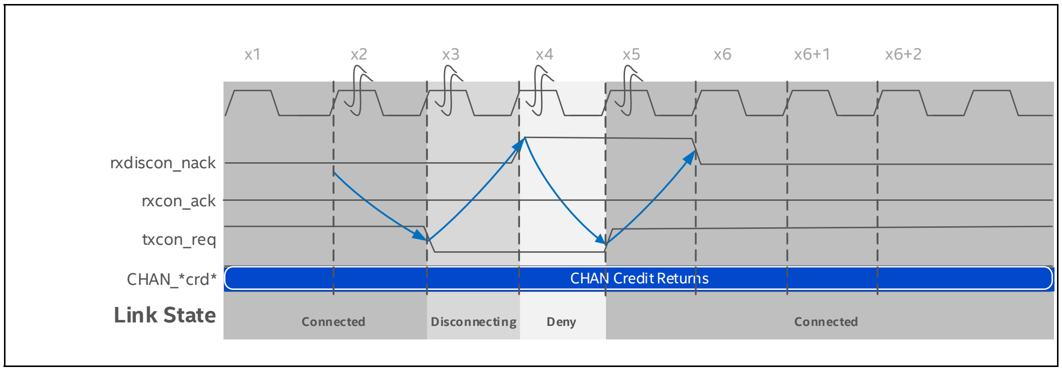


**Figure 5-4. Deny for Disconnect (Disconnect NAck)**



## 5.5 Surprise Reset

A surprise reset occurs when the TX or RX resets while SFI is connected. The recommended flow is to use disconnect before reset. This section captures the expected behavior if a surprise reset occurs.

There are two cases to consider for a surprise reset:

1. **`rxcon_ack`** 1 → 0 occurs because of a surprise reset on the RX side of the link while TX **`txcon_req`** is 1:

   TX forces itself to a disconnected state and re-starts initialization.

   a. If this happens when TX is in an idle state, it can recover without loss of messages.

2. **`txcon_req`** 1 → 0 occurs because of a surprise reset on the TX side of the link while **`rxcon_ack`** is 1:

   Follow the regular disconnect flow.

   a. If this happens when RX is in an idle state, disconnect should receive Ack and cleanly reach a disconnected state-provided TX stays in reset.

   b. If the disconnect is Denied (NAck) by RX, it results in a fatal or illegal link state that does not recover.

For both cases, if traffic is active (in other words, not idle), a loss of protocol messages can result and will likely be fatal to continue regular operation.

# 6 Mode Configuration

For a given SoC use case, SFI expects the configuration of an interface to be static. The exact solution for defining the configuration can be done in different ways and is not mandated by SFI. One way to configure the interface is through static RTL parameters.

### Table 6-1. Parameters Supported

| Parameter | Description |
|---|---|
| *VT* | Width of `tx_vendor_field`. |
| *VR* | Width of `rx_vendor_field`. |
| *VIRAL_EN* | This is either 0 or 1. Indicates the viral signal is supported on the global channel. |
| *FATAL_EN* | This is either 0 or 1. Indicates the fatal signal is supported on the global channel. |
| *BLOCK_EARLY_VLD_EN* | This is either 0 or 1. Indicates all the `*_block` and `*_early_valid` signals are supported. |
| *M* | Maximum number of headers that can be transmitted on a given cycle. |
| *MAX_HDR_WIDTH* | Maximum size of 1 header in bytes. One header credit corresponds to *MAX_HDR_WIDTH* bytes of storage. A minimum and maximum of one credit is used per header transfer. |
| *H* | Total width of `header` in bytes. If the interface is optimizing throughput for a header size of *HX* (can be less than *MAX_HDR_WIDTH*), this is *HX*M.* |
| *NHCRD* | Width (number of bits) of credit return value signal on HDR channel. |
| *HGRAN* | Design parameter for granularity of HDR SIZE field in `hdr_info_bytes`. Typically, this is set to four bytes (DWORD). |
| *HPARITY* | This is either 0 or 1. When set to 1, it indicates header parity is supported. When set to 0, it indicates header parity is not supported. |
| *HDR_MAX_FC_VC* | Maximum number of FC and VC ID combinations that can be received in one cycle on the HDR Layer. It is recommended that this should be 1 or 2.<br>All TX must support setting this to 1 for interoperability reasons. |
| *MAX_CRD_CNT_WIDTH* | Determines the width in number of bits for all the credit counters. |
| *SHARED_CRD_EN* | This is either 0 or 1. When set to 1, it implies Type 1 shared crediting is used. |
| *NUM_SHARED_POOLS* | Number of shared pools for Type 1 shared crediting, must be a number greater than or equal to 0 and less than or equal to 8. If SHARED_CRD_EN = 1, this must be greater than or equal to 1. If PCIE_SHARED_SELECT is 0, IP data sheet must provide shared pool to FC/VC mapping. |
| *PCIE_SHARED_SELECT* | This is either 0 or 1. If it is 1, PCIe* shared pool allocation is used; i.e.<br>SHARED_CRD_EN = 1,<br>NUM_SHARED_POOLS = 2 if PCIE_MERGED_SELECT = 1; shared pool 0 is for Posteds and Completions, shared pool 1 is for Non-Posteds.<br>else NUM_SHARED_POOLS=3 if PCIE_MERGED_SELECT = 0; shared pool 0 is for Posteds, shared pool 1 is for Non-Posteds, shared pool 2 is for Completions. |
| *PCIE_MERGED_SELECT* | This is either 0 or 1. If 1, Posteds and Completions share the same shared pool. If 0, Posteds and completions have separate shared pools of credits. |
| *SH_HDR_CRD_BLK_SZ* | Indicates the block size for shared header credit returns from the RX. It must be set to 1 or more.<br>It is strongly recommended for TX to provide a configuration register for this parameter. |
| *DATA_MAX_FC_VC* | Maximum number of FC and VC ID combinations that can be received in one cycle on the DATA Layer. It is recommended that this be 1 or 2.<br>All TX must support setting this to 1 for interoperability reasons. |
| *D* | Total width of the `data` signal in bytes. This must be a multiple of 4. |
| *DS* | Maximum number of independent data packets that can be sent in one cycle.<br>All TX must support setting this to 1 for interoperability reasons. |
| *NDCRD* | Width (number of bits) of credit return value signal on DATA channel. |
| *DATA_PARITY_EN* | This is either 0 or 1. If set to 1, `data_parity` signal is instantiated, and used to transfer data parity across the interface. |

| Parameter | Description |
|---|---|
| *DATA_AUX_PARITY_EN* | This is either 0 or 1. If set to 1, `data_aux_parity` signal is instantiated, and used to transfer data parity across the interface. |
| *DATA_PASS_HDR* | This is either 0 or 1. If set to 1, TX can allow Data to race ahead of the corresponding header. If set to 0, TX will always send header before the corresponding Data.<br>All TX must support setting this to 0 for interoperability reasons.<br>It is strongly recommended that this parameter be set to 0, so that the RX data tracking is simplified. |
| *HDR_DATA_SEP* | This is applicable when *DATA_PASS_HDR = 0.* If set to 1, data for a corresponding header is 1 cycle behind the header. If set to 0, data can be 1 or more cycles behind the corresponding header.<br>All TX must support setting this to 1 for interoperability reasons. Setting this greater than 1 is permitted, and it indicates a fixed delay of those many clock cycles. |
| *DATA_INTERLEAVE* | This is either 0 or 1. When advertised as 1 by a RX, it indicates that the RX is capable of dealing with data interleaving across different FC and VC over different clock cycles. |
| *ECRC_SUPPORT* | This is either 0 or 1. When set to 1, it indicates ECRC is supported. When set to 0, it indicates ECRC is not supported. |
| *IDE_SUPPORT* | This is either 0 or 1. When set to 1, it indicates that IDE TLPs are permitted to be sent across the interface. The formats of the prefixes and MAC follow the flit mode definitions given in *PCI Express* (PCIe*) Base Specification 6.0*. Implementations must also follow all the routing or ordering rules outlined in the specification. |
| *DATA_CRD_GRAN* | Credit granularity for data credits in DWORDs. Typically set to 4 DWORD (or 16 Bytes) to match PCIe*, in other words, each credit accounts for 16 bytes of storage in the RX. |
| *SH_DATA_CRD_BLK_SZ* | Indicates the block size for shared data credit returns from the RX. It must be set to 1 or more.<br>It is strongly recommended for TX to provide a configuration register for this parameter. |
| *TBN* | See Section 4.4.1.<br>0x0 – No Blocking<br>0x1 – Blocking is enabled with a response time of 1 cycle<br>0x2 – Blocking is enabled with a response time of 2 cycles<br>0x3 – Blocking is enabled with a response time of 3 cycles |
| *RBN* | See Section 4.4.1.<br>0x0 – No Blocking<br>0x1 – Blocking is enabled with a response time of 1 cycle<br>0x2 – Blocking is enabled with a response time of 2 cycles<br>0x3 – Blocking is enabled with a response time of 3 cycles |
| *TX_CRD_REG* | If 1, Tx exposes configuration registers that can be programmed or reset to values that indicate the maximum credits advertised by Rx for each credit pool. |
| *BCM_EN* | If 1, the vendor defined prefix shown in Figure 4-6 is the first DWORD for all completion headers sent over SFI. |
| *FLIT_MODE_PREFIX_EN* | If 0, transmitters must never send Flit Mode TLP prefix over SFI. See Section 4.2.3. |

## 6.1 Implementation Example

Table 6-2 gives an example set of key parameter values that can be used for PCIe* 6.0 implementation that is optimized for 16B payload memory transactions. The values are for an operating frequency of 2GHz for the interface. See the *PCI Express* (PCIe*) Base Specification 6.0* for details of TLP formats.

### Table 6-2. Example Parameter Assignments for PCIe* 6.0

| Parameter | Description |
|---|---|
| *M = 2* | At 2 GHz, a throughput of two headers per cycle would be optimized for 16B payloads. |
| *H = 48* | Optimized for 6DW header- 4DW of base, 1DW of OHC-A (that carries PASID and BE), 1DW of OHC-B (TLP hints PH/ST). If ECRC support is desired, then this should be 8DW per header, or *H=64*. |

| Parameter | Description |
|---|---|
| *HGRAN=4* | Design parameter for granularity of HDR SIZE field in **`hdr_info_bytes`**. Typically this is set to four bytes (DWORD). |
| *HPARITY=1* | When set to 1, it indicates HDR parity is supported. |
| *HDR_MAX_FC_VC=1* | Maximum number of FC and VC ID combinations that can be received in one cycle on the HDR layer. |
| *SH_HDR_CRD_BLK_SZ* | Implementation-specific, based on what block size optimizes RX storage structures. |
| *DATA_MAX_FC_VC=1* | Maximum number of FC and VC ID combinations that can be received in one cycle on the DATA layer. |
| *D=64* | Total width of the **`data`** signal in bytes. 64 B can hit 128 GB/s for larger payloads at 2 GHz. |
| *DS=2* | Maximum number of independent data packets that can be sent in one cycle. Allow sending two independent smaller payload packets, for example, two 16B payloads in one cycle. |
| *DATA_PASS_HDR=0* | If set to 0, TX will always send HDR before the corresponding Data.<br>It is strongly recommended that this parameter be set to 0, so that the RX data tracking is simplified. |
| *HDR_DATA_SEP=0* | If set to 0, data can be 1 or more cycles behind the corresponding header. |
| *DATA_INTERLEAVE=1* | When advertised as 1 by a RX, it indicates that the RX is capable of dealing with data interleaving across different FC and VC over different clock cycles. |
| *DATA_CRD_GRAN=4* | Credit granularity for data credits is set to 4DW. |
| *SH_DATA_CRD_BLK_SZ* | Implementation-specific, based on what block size optimizes RX storage structures. |