HyperTransport™ I/O Link Specification

Revision 1.04

5/30/2002

HyperTransport™ Technology Consortium

About HyperTransport™ Technology

HyperTransport technology is a high-speed, high-performance, point-to-point link for integrated circuits, and is designed to meet the bandwidth needs of tomorrow's computing and communications platforms. HyperTransport technology helps reduce the number of buses while providing a high-performance link for PCs, workstations, and servers, as well as numerous embedded applications and highly scalable multiprocessing systems. It is designed to allow chips inside of PCs, networking and communications devices to communicate with each other up to 48 times faster than with some existing bus technologies.

About the HyperTransport Technology Consortium

The HyperTransport Technology Consortium is a nonprofit corporation managed by its members. The consortium promotes the common business interests of providers to the networking, telecommunications, computer and high-performance embedded application through the conduct of a forum for the future development and adoption of the HyperTransport specification.

AMD, Apple Computers, Broadcom, Cisco Systems, NVIDIA, PMC-Sierra, SGI, SiPackets, Sun Microsystems, and Transmeta are the charter members that comprise the Executive Committee of the HyperTransport Technology Consortium.

Companies interested in the HyperTransport specification are invited to join the consortium. Members of the consortium pay annual dues and receive a royalty-free license to HyperTransport IP, gain access to technical documentation and may attend consortium meetings and events. To become a member, visit the consortium Web site at www.hypertransport.org. Please review the Bylaws of the HyperTransport Technology Consortium, and complete the online membership application.

REVISION	CHANGE	SECTION	DATE
1.03	Initial release	All	10/10/01
1.04	Fixed errata and made clarifications	All	5/30/02
	Fully defined open-drain signal behavior	2, 13, 17.5	
	Document Isochronous VC limitations	4.4.3, 4.4.4, 7.5.4.9, D.1	
	Allow RMW with count other than 1 or 3	4.4.5	
	Relaxed response UnitID for rejected packets	4.9.3	
	Updated upstream configuration requirements	4.9.4, F	
	Swapped Interrupt and Address chapters (5 and 9)	5, 9	
	Added new x86 reserved address range	5	
	Tighten configuration and I/O space access requirements	5, B	
	Highlighted required registers	7.3-7.5	
	Expanded description of Address registers	7.3.5, 7.4.6	
	ISA and VGA enable bits made required and documented	7.4.9.3	
	Require a revision ID in every function	7.5	
	Extend HT Capability Type field to 5 bits	7.5.3.1, 7.7.1	
	Documented behavior of multiple nonprefetchable memory range register extensions	7.5.13	
	Added Isochronous, NonCoherent, and Compat bits to Address Remapping Block	7.7	
	Created Revision ID Capability	7.8	
	Documented reset data corruption case	10.1	
	Documented behavior when all UnitIDs are consumed	12.3	
	Combined Appendices B and C	B	
	Relaxed PCI ordering, added HT to PCI command mapping	B.2	
	Added PCI-X ordering rules and command mapping	B.4	
	Added Deadlock Appendix C	C	
	Document legacy interrupt boot requirements	F.1	
	Document legacy PIC multiple ExtInt requirements	F.1.3	
	Document delay between STOP_GRANT and LDTSTOP#	F.2	
	Document A20M ordering requirements	F.2.1.1	
	Updated LDTREQ# requirements	F.2.4	
	Updated differential signal input edge rate requirements	17.10	

Contents

List of Figures

List of Tables

Preface

This Document

The *HyperTransport™ I/O Link Specification*~~, Version 1.03,~~ defines and describes the input/output link protocol and electrical interface for the HyperTransport™ technology link. The document is divided into two principal parts: Protocol and Electrical. The protocol part includes information on HyperTransport technology signals, packets, commands, interrupts, configuration accesses, address map, error handling, clocking, and initialization. The electrical part It includes information on I/O power supply, AC and DC characteristics, transfer timing, and phase recovery timing.

It is intended for system designers, circuit designers, sales and marketing engineers, and other technology professionals. This document serves as the primary reference for the HyperTransport protocol.

Organization

The document is ~~divded~~divided into two sections, each with ~~appendicies~~appendices.

Section 1 – Protocol

Section 2 – Electrical Interface.

~~About HyperTransport™ Technology~~

~~HyperTransport technology, formerly codenamed Lightning Data Transport (LDT), was developed at AMD with the help of industry partners to provide a high-speed, high-performance, point-to-point link for interconnecting integrated circuits on a board. Designed to operate with a top signaling rate of 1.6 GHz on each wire pair, a HyperTransport technology bus is designed to support a peak aggregate bandwidth of 12.8 Gbytes/s.~~

~~HyperTransport technology is a packet-based link implemented on two independent unidirectional sets of wires. The link is nominally point-to-point and connects two devices. Chains of HyperTransport links can also be used as an I/O channel, connecting I/O devices and bridges to a host system.~~

~~The HyperTransport link is designed to deliver a high-performance and scalable interconnect between CPU, memory, and I/O devices. The HyperTransport link uses low swing differential~~

~~signaling with on-die differential termination to achieve very high data rates. The HyperTransport link uses scalable frequency and data width to achieve scalable bandwidth.~~

~~HyperTransport technology provides significantly more bandwidth than current technologies, uses low-latency responses and low pin counts, and supports legacy PC buses. It is also designed to be extensible to new systems network architecture (SNA) buses, transparent to operating systems, and has little or no impact on peripheral drivers.~~

~~HyperTransport technology is designed for use in networking, telecommunications, computer, and high performance embedded applications, and any other application in which high speed, low latency, and scalability are necessary.~~

~~HyperTransport™ Technology Consortium~~

~~On July 23, 2001, the formation of the HyperTransport Technology Consortium was announced. The charter is to manage and control the development and evolution of the HyperTransport technology specifications. To obtain the specifications and information about joining the HyperTransport Technology Consortium, please visit the Sonsortium's web site at: http://www.hypertransport.org.~~

Section 1 – Protocol

1 Overview

This document describes the HyperTransport™ technology I/O link. HyperTransport technology, formerly code-named Lightning Data Transport (LDT), is a packet-based link implemented on two unidirectional sets of signals. The link is packet-based, nominally point-to-point, and connects exactly two devices. Devices can have multiple HyperTransport links, allowing the construction of larger HyperTransport fabrics.

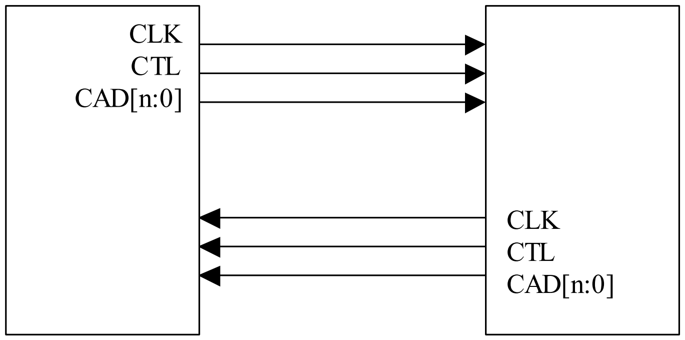


Figure 1. HyperTransport™ I/O Link

HyperTransport technology is used as an I/O channel, connecting chains of HyperTransport I/O devices and bridges to a host system. The interface from the host to the HyperTransport chain(s) is called the host bridge.

1.1 Terminology

For reference, the following terms are used in the HyperTransport protocol:

- *Bit-time*—Half of a clock period in duration. Two data bits are transmitted on each signal per cycle.
- *Byte*—Eight bits.
- *Doubleword*—Four bytes.
- *Quadword*—Eight bytes.
- *Packet*—A series of bit-times that forms the basis of communication between two nodes.
- *Transaction*—A sequence of packets that are exchanged between two or more nodes in the system and that result in a transfer of information.
- *Source*—The node that initiates a transaction.
- *Target*—The node that ultimately services the transaction on behalf of the source. Note that there may be intermediary nodes between the source and the target.
- *I/O stream*—A collection of transactions that can be treated independently in the fabric with respect to ordering rules. A given I/O stream always originates from the same node and terminates at the same destination.
- *Unit* or *function*—A logical entity within a node that can act as a source or destination of transactions. Functions are useful for describing the transaction ordering rules of the HyperTransport protocol.
- *Node*—A physical entity that connects to one end of a HyperTransport link.
- *Chain*—A set of HyperTransport technology devices connected in a straight-line daisy-chain topology with each device connecting to at most two others. Devices with two link interfaces have no logical bridging function between the two interfaces; the entire chain operates as a single logical bus.~~A single HyperTransport *chain* contains no HyperTransport-to-HyperTransport bridge devices. It may contain native HyperTransport technology peripheral devices (like an Ethernet controller) and may also contain bridges to other interconnects (like PCI). A HyperTransport chain is terminated at one or both ends by a bridge. In the simplest topology, a HyperTransport chain connects to the host bridge at one end and has no connection at the other end.~~
- *Fabric*—A HyperTransport I/O *fabric* is implemented as one or more daisy chains of HyperTransport technology devices, with a bridge to the host system at one end.
- *Bridge*—A device that bridges between a logical primary bus (connecting it to the host) and one or more logical secondary buses. It contains a bridge register set to control mapping of transactions between the two buses.~~A HyperTransport technology *bridge* device has a primary link, that being the upstream link in the direction of the host, and one or more secondary links.~~

- *Tunnel*—A device that implements two link interfaces and is capable of forwarding traffic between other devices within a chain.~~dual-link device that is not a bridge. Tunnel devices act as I/O building blocks.~~
- *Cave*—A device that implements a single primary link interface.
- *Slave*—A tunnel or cave, implementing HyperTransport link(s) as its primary interface, not implementing the Host bridge functionality.
- *Host*—A host can contain multiple bridges, each supporting either a single HyperTransport chain or a tree of HyperTransport chains.
- *Host bridge*—The interface from the host to the HyperTransport chain.
- *Undefined*—Operations or behaviors that are described as undefined in this specification may result in any outcome from no change in the state of the system to creating an environment in which the system no longer continues to operate. Note that a cold reset of the HyperTransport fabric may not be sufficient to restore deterministic operation since the effects of an undefined action may propagate beyond the HyperTransport fabric.

1.2 HyperTransport™ Technology in x86 Platforms

This specification is written as a generic reference suitable for implementation with all CPU architectures. However, because of the legacy infrastructure associated with x86 platforms, some additional features must be supported in those platforms. These additional requirements are specified in Appendix F.

2 Signaling

The HyperTransport™ technology signals listed in Table 1 constitute a single unidirectional connection between two nodes. A full link requires a connection in each direction. However, the connections need not be the same width in each direction.

Table 1. Link Signals

Signal	Width	Description
CAD	2, 4, 8, 16, or 32	Command, Addresses, and Data (CAD). Carries HyperTransport™ technology requests, responses, addresses and data. CAD width can be different in each direction.
CTL	1	When asserted, the CAD signals carry a control packet. When deasserted, the CAD signals carry data.
CLK	1, 2, or 4	Clocks for the CAD and CTL signals. Each byte of CAD has a separate clock signal. Regardless of the width of the link, CTL is clocked by the same CLK as CAD[0].

HyperTransport links wider than 8 bits are built by ganging multiple 8-bit links in parallel to form either 16- or 32-bit links. Links wider than 8 bits have one clock per byte, but still only one CTL bit for the whole link. The forwarded clock for the CTL signal is the clock for the least-significant byte.

In addition to the link signals, all HyperTransport technology devices require the reset/initialization input pins listed in Table 2.

Table 2. Reset/Initialization Signals

Signal	Width	Description
PWROK	1	Power and clocks are stable.
RESET#	1	Reset the HyperTransport™ chain.

All devices in a given HyperTransport chain receive the same PWROK and RESET# signals. ~~In some implementations, these signals may be open drain wired-OR signals, thus allowing multiple sources (possibly including host bridges) to initiate a reset sequence.~~ HyperTransport technology devices must sample PWROK and RESET# as inputs, and may optionally drive these signals low as open-drain outputs. Pullup resistors must be provided by the system. These signals control the power-up and reset sequence for their HyperTransport links, and may or may not also control the power-up and reset sequence for other logic within the device—this is device-specific. See Section 4.1.2 for a description of these signals in the context of systems with multiple HyperTransport chains. See Chapter 12 for information on reset sequencing.

HyperTransport technology devices deployed in x86 or other systems requiring power management include the signals listed in Table 3, which are used during the sequencing of system activities such as power-savings state transitions. These signals are also open-drain wired-OR, allowing multiple sources to request link disconnection and reconnection, and requiring pullup resistors.

Table 3. Power Management Signals

Signal	**Width**	**Description**
LDTSTOP#	1	Enables and disables links during system state transitions.
LDTREQ#	1	~~Requests re-enabling links for normal operation.~~Indicates link is active or requested by a device.

~~1~~PWROK, RESET#, LDTSTOP#, and LDTREQ# will likely have long transition times, and therefore will require hysteresis and/or debounce logic in the input path for correct operation.

3 Packet Definition

This chapter describes the packet definition for the HyperTransport™ link. HyperTransport technology supports link widths of 2, 4, 8, 16, and 32 bits. All tables later in this chapter assume an 8-bit wide link.

The packet structure for 16- and 32-bit links can be derived from the 8-bit link packet structure by combining the fields within adjacent bit-times. Some examples include:

$BT1_{16}[15:0] = BT2_{8}[7:0] \, || \, BT1_{8}[7:0]$

$BT1_{32}[31:0] = BT4_{8}[7:0] \, || \, BT3_{8}[7:0] \, || \, BT2_{8}[7:0] \, || \, BT1_{8}[7:0]$

where BTN_m represents the Nth bit-time within a packet for a link of width m and "||" represents concatenation.

Since all packets are multiples of four bytes long, packet boundaries always fall on bit-time boundaries.

The packet structure for 2- and 4-bit links can be derived from the 8-bit link packet structure by splitting the 8-bit link bit-times into adjacent bit-times. Some examples:

$BT1_{2}[1:0] = BT1_{8}[1:0]$
$BT2_{2}[1:0] = BT1_{8}[3:2]$
$BT3_{2}[1:0] = BT1_{8}[5:4]$
$BT4_{2}[1:0] = BT1_{8}[7:6]$

$BT1_{4}[3:0] = BT1_{8}[3:0]$
$BT2_{4}[3:0] = BT1_{8}[7:4]$

3.1 Use of the CTL Signal

HyperTransport technology links carry control packets and data packets, distinguished by the use of the CTL signal. Link transmitters assert CTL during all bit-times of control packets, and deassert it during data packets. The purpose of the CTL signal is to allow control packets to be inserted in the middle of long data packets.

The following rules govern packet transmission.

1. CTL is asserted through all bit-times of a control packet.

2. Control packets larger than four bytes must be transmitted contiguously, without deassertion of CTL.
3. CTL is deasserted through all bit-times of a data packet.
4. CTL may be asserted on four-byte boundaries within a data packet to insert a control packet. Control packets inserted into data packets must not themselves have an associated data packet. When CTL is deasserted at the conclusion of a control packet, data transfer continues from the point where it left off.
5. Write request and read response packets always have an associated data packet. The data packet might not immediately follow the last bit-time of its associated control packet, because other control packets may be inserted before the data packet. However, because inserted control packets cannot have associated data, there can only be one data transfer outstanding.
6. The order of operations on the link is determined by the order of the control packets. The fact that data transfer for a control packet may be delayed does not affect how it is ordered.
7. The bit-time immediately following the last bit time of a data packet is always the start of a control packet (CTL must be asserted).
8. CTL may only be asserted or deasserted on a four-byte boundary, starting at the point in the initialization sequence (Section 12.2) where CAD and CTL both transition from 1 to 0. This alignment must be maintained until RESET# is asserted or a disconnect sequence (Section 8.3) completes.
9. CTL may only be deasserted when data transfer due to a previously transmitted control packet is being sent.

3.2 Packet Structure

This section defines the basic control and data packet types and shows the position of the fields that are common to all the control packet types. All packets are multiples of four bytes long.

3.2.1 Control Packets

Control packets consist of four or eight bytes. This section shows the basic structure of each of these control packet forms.

In the tables that follow, the unlabelled packet fields are command-specific. Some common control packet fields are as follows:

- *Cmd[5:0]* is the command field that defines the packet type.
- *Isoc* indicates that this packet may have different priority and ordering requirements from other packets, as described in Appendix D.
- *UnitID[4:0]* serves to identify one of the participants in a transaction. Since all packets are transferred either to or from the host bridge at the end of the ~~fabric~~chain, either the source or destination node is implied. The value 0 is reserved for the UnitID of the host bridge. See

Section 4.2 for more details on the use of UnitID. Nodes with multiple logical I/O streams can own multiple UnitID values.

- *Bridge* indicates that this response packet was placed onto the link by the host bridge, and it is used to distinguish responses traveling upstream from responses traveling downstream. In the case of two host bridges sending packets to each other on a double-ended chain, the target host bridge appears to the requesting host bridge as a HyperTransport technology slave device. Therefore, the bridge bit will be clear on responses to requests issued from the far host bridge.
- *SeqID[3:0]* is used to tag groups of requests that were issued as part of an ordered sequence by a device and must be strongly ordered within a virtual channel. All requests between the same source and destination and within the same I/O stream and virtual channel that have matching nonzero SeqID fields must have their ordering maintained. The SeqID value of 0x0 is reserved to mean that a transaction is not part of a sequence. Transactions with this value have no sequence-ordering restrictions, although they may be ordered for other reasons as described in Chapter 6. Tunnels are required to keep requests in the same I/O stream with matching nonzero SeqID fields in order when forwarding them.
- *PassPW* indicates that that this packet is allowed to pass packets in the posted request channel of the same I/O stream. Otherwise, this packet must stay ordered behind them. This bit should be cleared to maintain the ~~PCI-like~~full producer/consumer ordering model of HyperTransport technology. Systems that do not require ~~PCI-like~~this ordering may set PassPW for higher performance.
- *SrcTag[4:0]* is a transaction tag that is used to uniquely identify all transactions in progress initiated by a single requester. Each UnitID can have up to 32 transactions in progress at one time. The concatenation of source UnitID and SrcTag serves to uniquely identify nonposted requests. The SrcTag field is not relevant for posted requests and is reserved. SrcTag is used to match responses with their requests.
- *Addr[39:2]* represents the doubleword address accessed by the request. Not all address bits are included in all request types. Where finer granularity is required, byte masks are used.

Reserved fields in command packets must always be driven to 0 by transmitters when originating packets and must be assumed to be undefined by receivers. Reserved fields should be preserved when forwarding packets through a tunnel or ~~when a host reissues a peer-to-peer cycle~~HyperTransport-to-HyperTransport bridge.

3.2.1.1 Info Packet

Info packets (defined in Table 4) are always four bytes long. They are used for nearest neighbor communication between nodes, and so exist at the lowest level of the protocol. They are not routed within the fabric, and they require no buffering in the nodes. They are not flow-controlled, and they can always be accepted by their destination.

Table 4. Info Packet Format

Bit-Time	7	6	5	4	3	2	1	0
0	*Command-Specific*		Cmd[5:0]					
1	*Command-Specific*							
2	*Command-Specific*							
3	*Command-Specific*							

3.2.1.2 Request Packet

Request packets are either four or eight bytes long, depending upon whether the request has an associated address. Table 5 shows a request packet with an address. Four-byte request packets do not contain the address field.

Table 5. Request Packet Format with Address

Bit-Time	7	6	5	4	3	2	1	0
0	SeqID[3:2]		Cmd[5:0]					
1	PassPW	SeqID[1:0]		UnitID[4:0]				
2	*Command-Specific*							
3	*Command-Specific*							
4	Addr[15:8]							
5	Addr[23:16]							
6	Addr[31:24]							
7	Addr[39:32]							

3.2.1.3 Response Packet

Response packets (defined in Table 6) are always four bytes long.

Table 6. Response Packet Format

Bit-Time	7	6	5	4	3	2	1	0
0	*Command-Specific*		Cmd[5:0]					
1	PassPW	Bridge	*Cmd-Specific*	UnitID[4:0]				
2	Rsv		Error	*Command-Specific*				
3	Rsv		NXA	Rsv			*Command-Specific*	

The Error bit is present in all responses and is used to indicate that the issued request could not be completed. If the Error bit is set, the Non-Existent Address (NXA) bit is valid. If the NXA bit is set, that means that the request could not be completed because no agent on the chain accepted the request. If the NXA bit is clear, it means that the request packet reached its addressed target, but could not be completed by the device.

3.2.1.4 Command Field Encoding

The command field, shown in Table 7, is valid for all control packets.

Table 7. Command Field Encoding for All Control Packets

Code	VChan	Command	Comments/Options	Packet Type
000000	-	NOP	Null packet. Contains flow control information.	Info
000001		Reserved-HOST		
000010	NPC	Flush	Flush posted writes	Request
000011 0001xx		Reserved-HOST		
001xxx 101xxx	NPC ~~or~~ PC ~~(bit 5)~~	Wr (sized)	Write Request [5] Defines whether request is posted: 0: Nonposted 1: Posted [2] Defines the data length: 0: Byte 1: Doubleword [1] Defines bandwidth/latency requirements: 0: Normal 1: Isochronous [0] Indicates whether access requires host cache coherence (ignored if access is not to host memory): 0: Noncoherent 1: Coherent	Req/Addr/Data

Code	VChan	Command	Comments/Options	Packet Type
01xxxx	NPC	Rd (sized)	Read Requests [3] Defines ordering requirements for response: 0: Response may not pass posted requests 1: Response may pass posted requests [2] Defines the data length: 0: Byte 1: Doubleword [1] Defines bandwidth/latency requirements: 0: Normal 1: Isochronous [0] Indicates whether access requires host cache coherence (ignored if access is not to host memory): 0: Noncoherent 1: Coherent	Req/Address
100xxx		Reserved-I/O		
110000	R	RdResponse	Read Response	Resp/Data
110001 110010		Reserved-HOST		
110011	R	TgtDone	Tell source of request that target is done.	Response
11010x		Reserved-HOST		
11011x		Reserved-I/O		
11100x		Reserved-HOST		
111010	PC	Broadcast	Broadcast Message	Req/Address
111011		Reserved-HOST		
111100	PC	Fence	Fence for posted requests	Request
111101	NPC	Atomic-RMW	Atomic Read-Modify-Write	Req/Addr/Data
111110		Reserved-I/O		
111111	-	Sync/Error	Link Synchronization and Error Packet	Info

Notes:

The fields in Table 7 are defined as follows:

***Code** is the 6-bit command encoding in each packet.*

***VChan** indicates the virtual channel that the packet travels in. Info packets are only used for single-link communication and do not use buffer space, and thus are not in a virtual channel. See Section 4.7 and Section 4.8 for more information.*
***PC**—Posted Command (Request)*
***NPC**—Nonposted Command (Request)*
***R**—Response*

Code	VChan	Command	Comments/Options	Packet Type
Command *is the mnemonic used to represent the command.* ***Comments/Options*** *gives a short description of the command and enumerates any option bits within the Code field.* ***Packet Type*** *indicates the type of packet(s) used by the command.* ***Reserved-I/O*** *identifies code points that are reserved for future use.* ***Reserved-HOST*** *identifies code points that may be used in a host-specific protocol and will not be used to implement future features in the HyperTransport™ I/O Link Protocol Specification.*				

Receiving a packet with a reserved command code is a protocol error (see Section 10.1.3) and may result in undefined operation of devices that do not implement recovery from protocol errors.

3.2.2 Data Packet

Data packets contain the data payload for transactions. Data packets follow write request and read response packets. Data packets range in length from four to 64 bytes, in multiples of four bytes (one doubleword), as indicated by the Count field of the most recent payload-bearing command. Within a doubleword, data bytes appear in their natural byte lanes. For transfers of less than a full doubleword, the data is padded with undefined bytes to achieve this byte-lane alignment.

Table 8 shows an example of an eight-byte data packet.

Table 8. Eight-Byte Data Packet Format

Bit-Time	7	6	5	4	3	2	1	0
0	Data[7:0]							
1	Data[15:8]							
2	Data[23:16]							
3	Data[31:24]							
4	Data[39:32]							
5	Data[47:40]							
6	Data[55:48]							
7	Data[63:56]							

The data packet for a sized read response is arranged with the lowest addressed doubleword returned first, and the remainder of the addressed data is returned in ascending address order by doubleword. Sized read responses can contain any number of contiguous doublewords within a 64-byte aligned block. Although, for sized byte reads, not all bytes are guaranteed to be valid. The data cannot wrap from the most significant doubleword in the aligned 64-byte block to the least significant doubleword in the block.

Sized doubleword writes work in the same way as sized doubleword read responses and can contain anywhere from one to 16 doublewords in ascending address order.

Sized byte writes, defined in Section 4.4.1~~, as shown in Table 9~~, transmit one doubleword worth of masks first, followed by from one to eight doublewords of data in ascending address order, as shown in Table 9. Mask[0] corresponds to Data[7:0], Mask[1] to Data[15:8], and so on. Thirty-two mask bits are always transmitted, regardless of the amount of data. All-zero byte masks are permitted. Interrupt and system management messages, which are composed of byte write packets to predefined address ranges, are the only byte write packets that do not require at least one doubleword of data.

Table 9. Sized Byte Write Data Packet Format

Bit-Time	7	6	5	4	3	2	1	0
0	Mask[7:0]							
1	Mask[15:8]							
2	Mask[23:16]							
3	Mask[31:24]							
4	Data[7:0]							
5	Data[15:8]							
6	Data[23:16]							
7	Data[31:24]							
8+	Packet may contain up to eight doublewords of data.							

4 Fabric Operation

The HyperTransport™ link is a pipelined, split-transaction interconnect where transactions are tagged by the source and responses can return to the source out of order. This chapter outlines the basic operation of the link.

4.1 Topology

HyperTransport I/O fabrics are implemented as one or more *daisy chains* of HyperTransport technology-enabled devices, with a bridge to the host system at one end. Devices can implement either one or two links.

- A dual-link device that is not a bridge is called a *tunnel*.
- Single-link devices must always sit on the end of the chain, so only one single-link device is possible in a chain.

Direct peer-to-peer communication between devices in the chain is not allowed. All packets (except for info packets) travel between one device and the host bridge. This means that at a high level, the fabric appears as a group of devices directly connected to the host bridge, but not to each other. Packets flowing into the fabric from a host bridge are said to be flowing *downstream*. Packets flowing to a host bridge from a HyperTransport technology device are said to be flowing *upstream*.

A single HyperTransport I/O *chain* contains no HyperTransport-to-HyperTransport bridge devices. It can contain native HyperTransport technology peripheral devices (like an Ethernet controller) and can also contain bridges to other interconnects (like PCI). A chain is terminated at one or both ends by a bridge. In the simplest topology, a chain connects to the host bridge at one end and has no connection at the other end.

A HyperTransport *tree* contains one or more HyperTransport-to-HyperTransport bridge devices. A HyperTransport technology *bridge device* has a primary link, being the upstream link in the direction of the host and one or more secondary links. Each HyperTransport chain that connects to a bridge's secondary link is assigned a unique bus number (see Section 7.2.3 for details). The HyperTransport-to-HyperTransport bridge device operates as a host bridge for devices on its secondary chain. In addition to its secondary links, a HyperTransport technology bridge device may have a downstream link that is associated with the same bus number as the bridge's primary link. The root of the HyperTransport tree connects to the host.

The *host* can contain multiple bridges, each supporting either a single HyperTransport I/O chain or a tree of HyperTransport I/O chains. Some example configurations and topologies are shown in Figure 2 and Figure 3. In these figures, "P" indicates a primary interface capability block and "S" indicates a secondary interface capability block. See Section 7.5 for details of these capability blocks.

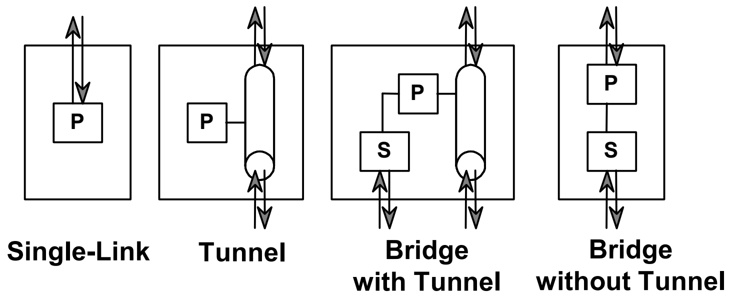


Figure 2. Example Device Configurations

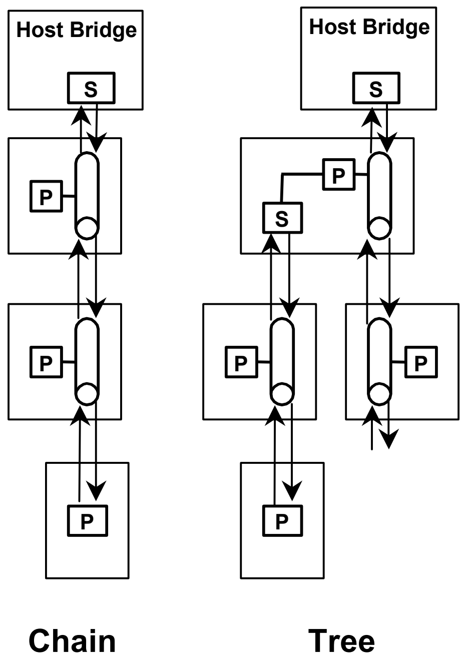


Figure 3. Example Topologies

For convenience in integrating multiple functions onto a single chip, or to allow parallelism between independent request streams, individual HyperTransport technology devices can use multiple UnitID values. There is no specific limit on the number of physical devices. However, there are only 31 UnitIDs available to ~~each chain~~each chain. No combination of devices that exceeds 31 UnitIDs may be connected to a single chain.

4.1.1 Double-Hosted Chains

Physically, a chain can be connected to a host bridge at each end, as long as the chain contains no single-link devices. This may be useful to provide another path to I/O devices in the event of a host bridge or link failure, or to allow sharing of I/O devices between independent hosts to implement clustering. One bridge is designated the *master bridge* for the shared chain, while the other will be the *slave bridge*. This designation must be made before the ~~fabric~~ chain is reset. (The method of doing so is beyond the scope of this specification.)

There are two types of double-hosted chains supported by HyperTransport technology: sharing and non-sharing.

- In a *sharing double-hosted chain*, traffic is allowed to flow end to end, and both hosts are able to issue requests to each other and to any device. Generally, all devices in the chain should belong to the master host to avoid a peer-to-peer transaction deadlock as described in Section 4.7. A device belongs to a host when the Master Host and Default Direction bits (defined in Section 7.5.3.2) point to that host. If devices need to be accessed from either host, the slave host may have its Act as Slave bit (defined in Section 7.5.3.3.6) set so that all requests pass through the master host to maintain ordering, as defined in Section 6.1.
- A *non-sharing double-ended chain* appears logically as two distinct daisy chains, each attached to only one host bridge. Software will select a point to break the chain in two and reconfigure the devices to divide them between the bridges in order to balance traffic. Once the chain is broken, the hosts will not be able to issue requests to each other until a reset.

The initialization sequence described in Section 12.3 will ensure that all devices are assigned unique device numbers. In the event of a node or link failure, the sequence will cause the devices on each side of the break in the chain to belong to the host bridge on that end, forcing a non-sharing chain.

Because devices accept requests from both directions in either double-hosted chain type, they must keep track of which link incoming request packets were received on and send any responses back on the same link. An interior node may see the same SrcTag active from the host bridges at both ends of the link. The node must recognize the two host bridges as having disjoint SrcTag spaces.

Double-hosted chain support for hosts is optional, but recommended. In order to support double-hosted chains, a host must implement the Double Ended and Chain Side fields of the HyperTransport technology Command register, specified in Section 7.5.3.3, and the host must properly accept cycles targeted to it, as described in Section 4.9.4. If a host does not support double-hosted chains, it cannot be connected to the secondary port of a bridge (for clustering).

To support a sharing double-hosted chain, the host must also implement the Device Number and Host Hide fields of the Command register, specified in Section 7.5.3.3, and deal with the ordering described in Section 6.4.

4.1.2 HyperTransport™ Technology Signals PWROK and RESET#

This section describes the HyperTransport technology signals PWROK and RESET# in the context of various system topologies. Section 4.1.2.1 describes requirements that all HyperTransport technology-enabled devices and systems must meet. Section 4.1.2.2 describes some host implementations.

4.1.2.1 Requirements

All devices on a HyperTransport I/O chain are expected to share a single logical PWROK/RESET# signal pair. Due to the potential for devices sampling these signals on different clocks, copies of the signals coming from different drivers, slow edges being sensed at different times, or receivers having different thresholds, these signals will not necessarily be observed to transition simultaneously at all devices. The system must guarantee that all devices see PWROK and RESET# pulses overlap and that the duration of the overlap meets the minimum requirements given in Section 12.2.~~,~~ These signals are inputs to each device on the chain and may be driven by one or more devices on the chain. These signals control the powerup and reset sequence for each link interface in the chain and may optionally control the powerup and reset sequence for other logic inside any device along the chain.

A HyperTransport-to-HyperTransport bridge device must have dedicated PWROK/RESET# pin pairs for its primary chain and for each secondary chain. The bridge must be able to drive PWROK/RESET# on its secondary chain. In addition, the bridge must pass the assertion and deassertion of PWROK/RESET# from its primary chain to its secondary chain. A bridge does not pass the assertion and deassertion of PWROK/RESET# from its secondary chain to its primary chain. A bridge is required to provide appropriate error responses to any outstanding non-posted requests when the secondary bus reset is asserted, as described in Section 10.1.6.

4.1.2.2 Host Implementations

In the case of a host with a single HyperTransport chain, the host's reset signal can be independent of the HyperTransport link's PWROK/RESET# signals. This allows software running on the host to reset the HyperTransport chain without requiring the host to be reset (see Section 7.4.9.6). In such an implementation, the host bridge must be able to both drive and sample PWROK and RESET#. In addition, the host bridge must pass the assertion and deassertion of host reset (or PWROK) to RESET# (or PWROK). Other implementations are possible—for example, host reset and HyperTransport technology reset functions may be tied to a single host reset pin.

In the case of a host with multiple host bridges, there can be independent PWROK/RESET# signal pairs for each chain connected to the host. As in the previous case, each of these PWROK/RESET# signals can be independent of host reset. Other implementations are possible—for example, host reset and all HyperTransport technology reset functions may be tied to a single host reset pin.

Proper sequencing of the PWROK and RESET#, as described in Section 12.2, must be assured, even if the host's own PWROK and reset signals do not follow these sequencing rules.

Devices used in x86 systems have specific mandatory PWROK and RESET# requirements, described in Appendix F.

4.2 Transactions and UnitID

Since all HyperTransport technology transactions consist of a series of packet transfers between a device and the host bridge, the use of the UnitID field can be simply summarized in Table 10.

Table 10. UnitID Field Usage

	Upstream	**Downstream**
Request	UnitID is source of request: Device's UnitID	UnitID is source of request: Host's UnitID: always 0
Response	UnitID is target of request: Device's UnitID~~.~~ Bridge bit clear~~.~~	UnitID is source of request: Device's UnitID~~.~~ Bridge bit set~~.~~
Note: *The Bridge bit in a downstream response must be set and the UnitID must be that of the request, even when the responder is a slave, not a host, as described in Section 4.9.3.*		

Peer to peer communication is implemented as a pair of HyperTransport technology transactions—a transaction generated by the source device and targeted at the host, and a transaction generated by the host and directed to the target device. The UnitIDs in the request and response packets associated with these two transactions follow the rules in Table 10.

4.3 Link Synchronization

The Sync pattern is used to indicate that a resynchronization event has occurred in the system, such as a reset or a ~~fabric~~ chain error, which requires all links to be resynchronized. The Sync pattern is defined in Table 11.

Table 11. Sync Pattern Format

Bit-Time	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	11		Cmd[5:0]: 111111					
1	11111111							
2	11111111							
3	11111111							

CRC checking on a link is shut down when a Sync packet is received. See Section 10.1 for a description of CRC.

All fields in a Sync pattern (including the command) are all 1s. Receivers on 8-, 16-, or 32-bit links may detect a Sync pattern by observing at least 16 bit-times of all 1s on byte lane 0 of the link (starting with the rising edge of CLK in 8- or 16-bit links), or by decoding the Sync command via its normal command decode logic. Sync patterns on 4- and 2-bit links require two times and four times the number of all-1-bit-times, respectively, as 8-bit, 16-bit, and 32-bit links.

Once a transmitter places a Sync pattern onto an active link, it keeps that pattern on the link until after the link is reset and synchronized. This allows a receiver to detect Sync via either method.

4.4 Requests

4.4.1 Sized Reads and Writes

The Sized Read or Write request is defined in Table 12. Sources use the Sized Read and Write requests (byte or doubleword) to initiate transactions to either memory or I/O. The data returned for Sized Reads cannot be coherently cached, as HyperTransport I/O provides no coherence primitives. Sized requests contain the starting doubleword address of the data and a set of data elements to be transferred. Bit 2 of the command field indicates whether the data elements to be transferred are bytes or doublewords, as defined in Table 7. Table 7 also defines the Coherence, Isochronous, Posted, and Response May Pass Posted Request bits in the command field.

Table 12. Sized Read or Write Request Format

Bit-Time	7	6	5	4	3	2	1	0
0	SeqID[3:2]		Cmd[5:0]					
1	PassPW	SeqID[1:0]		UnitID[4:0]				
2	Mask/Count[1:0]		Compat	SrcTag[4:0]/Rsv				
3	Addr[7:2]						Mask/Count[3:2]	
4	Addr[15:8]							
5	Addr[23:16]							
6	Addr[31:24]							
7	Addr[39:32]							

Doubleword operations can transfer any number of contiguous complete doublewords within a 64-byte aligned block. The Count field encodes the number of doubleword data elements that should be transferred, beginning at the specified address, and going in ascending order. Count codes of 0 through 15 represent 1 through 16 data elements to be transferred, respectively. Requests that cross a 64-byte boundary must be broken into multiple transactions, issued in ascending address order.

Byte reads can transfer any combination of bytes within an aligned doubleword. The Mask field is used to indicate which bytes within the doubleword are being read. Mask[0] corresponds to the lowest addressed byte, and Mask[3] corresponds to the highest addressed byte. Byte-maskable reads crossing an aligned doubleword boundary must be broken into multiple requests, each within a single doubleword. The mask bits can be ignored for reads to regions where reads are guaranteed not to have side effects. A read where all mask bits are 0 still causes host coherence action (if to memory and Cmd[0] is asserted) and still returns a RdResponse packet with one doubleword of (invalid) data.

Byte writes can transfer any combination of bytes within a naturally aligned 32-byte address region. Transfers that cross an aligned 32-byte boundary must be broken into multiple HyperTransport technology transactions, issued in ascending address order. Address bits [4:2] identify the first doubleword of data sent in the data packet within the 32-byte region defined by address bits [39:5]. The data packet for a byte-write operation contains byte mask information in the first doubleword of the data packet. The Count field is used to indicate the total size of the data packet in doublewords, including the byte masks, so it will range from one to eight to indicate two through nine doublewords to be transferred. In general, it is illegal for a byte-write packet to contain byte masks and no data, meaning the Count field must contain a nonzero value. The exceptions to this are interrupt and system management messages—they take the form of byte writes to predefined address regions and do not require data to be transferred. See Section 3.2.2 for the format of the data packet. The Count field specifies the length of the data packet independent of the value of the byte masks. Nonzero byte masks for doublewords that are not sent result in undefined behavior. Byte masks may be 0 for doublewords that are sent. The entire byte mask doubleword may be 0, in which case the system performs all activities usually associated with the request. However, no data is written.

The sized command field contains a bit that indicates whether the access requires coherence action to be taken by the system for host memory accesses. If this bit is set, the host must take whatever action is appropriate to ensure that any caching agent remains coherent with system memory. Writes must cause caches to be updated or invalidated. Reads must return the latest modified copy of the data, even if main memory is stale. If the bit is clear, reads and writes can happen directly to and from main memory without polling or modifying cache states. Most devices require host hardware to maintain coherence between processor caches within the host and host memory. Some devices may not require coherence to be maintained, or may have alternative application-specific means of ensuring memory coherence, and may clear the coherent bit to indicate this to the host.

Transactions also have an Isochronous bit in the command encoding associated with them that must be maintained by tunnels even when Isochronous flow control mode is disabled. Host bridges should maintain the bit when forwarding peer-to-peer requests if possible. See Appendix D for details of how this is used.

Sized Writes have a Posted bit. Besides serving as a virtual channel identifier, a set Posted bit indicates that the write request will receive no response in the fabric. The requester's buffer can be deallocated as soon as the write is transmitted. As such, the SrcTag field is reserved for posted requests. No assumptions can be made about the uniqueness or meaning of SrcTags for posted requests, either relative to other posted requests, or to other traffic. All nonposted requests (such as Reads, Nonposted Writes, Flushes, and Atomic RMW) for a given UnitID must have a unique SrcTag value for each outstanding request.

Reads have a Response May Pass Posted ~~Writes~~ Requests bit in the command field. This bit should be cleared to maintain the ~~PCI-like~~full producer/consumer ordering model of HyperTransport technology. Systems that do not require ~~PCI-like~~this ordering may set PassPW for higher performance. This bit is carried with the request but does not serve any purpose until the response is generated. At that time, it becomes the PassPW bit in the response.

Unlike read requests, write requests do not contain a Response May Pass Posted ~~Writes~~ Requests bit. Therefore, the PassPW bit in the TgtDone packet will generally be set. However, this is not strictly required—responders can choose to clear the PassPW bit in the TgtDone packet based on implementation-specific considerations. See ~~Appendix~~ Sections F.2.1.1 and F.2.5 for ~~one~~ some examples.

The Compat bit is used to implement the subtractive decode necessary for boot firmware devices. When set, it indicates that address decode in the host has found no mapping for the given access, and therefore the access should be routed to the bus segment containing the subtractive decode device. As part of the initialization sequence, all HyperTransport technology devices determine whether they own (or are) the subtractive decode device. Accesses with the Compat bit set are always accepted by devices that own it and ignored by all other devices, regardless of address. The

Compat bit may only be asserted for downstream requests. It is reserved for upstream requests and configuration space requests.

4.4.2 Broadcast Message

Broadcast messages (defined in Table 13) are used by the host to communicate information to all HyperTransport technology-enabled devices.

Table 13. Broadcast Message Format

Bit-Time	7	6	5	4	3	2	1	0
0	SeqID[3:2]		Cmd[5:0]					
1	PassPW	SeqID[1:0]		UnitID[4:0]				
2	Reserved							
3	Addr[7:2]						Rsv	
4	Addr[15:8]							
5	Addr[23:16]							
6	Addr[31:24]							
7	Addr[39:32]							

Broadcast messages can only be issued by the host bridge, and they travel in the downstream direction for the entire length of the chain, being both accepted and forwarded by all devices. Features that are implemented using Broadcast messages have reserved address ranges associated with them that are recognized by all devices. All information (including potential write data) necessary to the specific type of operation being performed is carried in the address field.

Broadcasts travel in the posted channel, and the SrcTag field is reserved. No assumptions can be made about the uniqueness of SrcTags, either relative to other Broadcast messages or other traffic.

4.4.3 Flush

Flush is designed to make sure that posted writes have been observed at host memory. It applies only to requests in the same I/O stream as the flush. The Flush command is defined in Table 14.

Table 14. Flush Format

Bit-Time	7	6	5	4	3	2	1	0
0	SeqID[3:2]		Cmd[5:0]					
1	PassPW	SeqID[1:0]		UnitID[4:0]				
2	Rsv			SrcTag[4:0]				
3	Rsv							

Flush functions very similarly to a Read operation, except that it returns no data. Like Reads, Flush goes in the nonposted request virtual channel. For a Flush to perform its intended function, the PassPW bit must be clear, so that the Flush pushes all requests in the posted channel ahead of it. It is expected that Flushes will never be issued as part of an ordered sequence, so their SeqID will always be 0. Flush requests with PassPW set or with a nonzero SeqID are legal, but their effect is unpredictable. Flushes do not affect traffic in the Isochronous virtual channels, so the Isochronous bit does not exist in this command and is assumed to be 0. When Isochronous Flow Control is disabled, Isochronous traffic flows in the normal virtual channels and is affected by Flushes.

All nonposted requests (such as Reads, Nonposted Writes, Flushes, and Atomic RMW) for a given UnitID must have a unique SrcTag value for each outstanding request.

Note that Flush only guarantees that posted requests have been flushed to their destination within the host. If the requests were peer-to-peer, this only means that they reached their destination host bridge, not the final device.

The Flush response is returned from the host bridge when the requests have become globally visible in the host. Since there is no data, a TgtDone response with PassPW set is used.

Flush is only issued from a device to a host bridge or from one host bridge to another. Devices are never the target of a Flush so they do not need to perform the intended function. If a device at the end of the chain receives a Flush, it must decode it properly to maintain proper operation of the flow control buffers and should return a TgtDone with the Error and NXA bits set.

4.4.4 Fence

Fence is designed to provide a barrier between posted writes, which applies across all UnitIDs and therefore across all I/O streams and all virtual channels. The Fence command is defined in Table 15.

Table 15. Fence Format

Bit-Time	7	6	5	4	3	2	1	0
0	SeqID[3:2]		Cmd[5:0]					
1	PassPW	SeqID[1:0]		UnitID[4:0]				
2	Rsv							
3	Rsv							

Fence goes in the posted request virtual channel and has no response. There is therefore no SrcTag field in the request packet. A Fence with PassPW clear will not pass anything in the posted channel regardless of UnitID. Packets with their PassPW bit clear will not pass a Fence regardless of UnitID. Packets with their PassPW bit set may pass a Fence. ~~Note that while a nonposted request with PassPW clear will not pass a Fence as it is forwarded through the chain, it may do so after it reaches a host bridge. See Section 6.2.~~

For a Fence to perform its intended function, the PassPW bit must be clear so that the Fence pushes all requests in the posted channel ahead of it. Fence requests are never issued as part of an ordered sequence, so their SeqID will always be 0. Fence requests with PassPW set, or with a nonzero SeqID, are legal, but may have an unpredictable effect. Fences do not affect traffic in the Isochronous virtual channels, so the Isochronous bit does not exist in this command and is assumed to be 0. When Isochronous Flow Control is disabled, Isochronous traffic flows in the normal virtual channels and is affected by Fences.

Fence is only issued from a device to a host bridge or from one host bridge to another. Devices are never the target of a fence so they do not need to perform the intended function. If a device at the end of the chain receives a fence, it must decode it properly to maintain proper operation of the flow control buffers. The device should then drop it. The node can choose to log this as an end-of-chain error, as described in Section 10.1.5.

4.4.5 Atomic Read-Modify-Write

The Atomic Read-Modify-Write (RMW) request is defined in Table 16. The Atomic RMW request supports two forms of atomic RMW operation on a naturally aligned quadword location:

- Fetch and Add
- Compare and Swap

Table 16. Atomic Read-Modify-Write (RMW) Request Format

Bit-Time	7	6	5	4	3	2	1	0
0	SeqID[3:2]		Cmd[5:0]					
1	PassPW	SeqID[1:0]		UnitID[4:0]				
2	Count[1:0]		Compat	SrcTag[4:0]				
3	Addr[7:3]					Rsv	Count[3:2]	
4	Addr[15:8]							
5	Addr[23:16]							
6	Addr[31:24]							
7	Addr[39:32]							

The Fetch and Add operation is:

```
FetchAdd(Out, Addr, In) {
Out = Mem[Addr];
Mem[Addr] = Mem[Addr] + In; // Unsigned add without saturation or carry
}
```

The Compare and Swap operation is:

```
CompareSwap(Out, Addr, Compare, In) {
      Out = Mem[Addr];
      If (Mem[Addr] == Compare) Mem[Addr] = In;
}
```

These operations must be performed atomically by the target of the request, meaning that no other agent in the system may access the addressed location between the time that it is read and written on behalf of the atomic request.

A Fetch and Add request must be accompanied by one quadword of data (the input value) and have a Count field value of 1. A Compare and Swap request must be accompanied by two quadwords of data (the compare and input values) and have a Count field value of 3. The Compare value is first, followed by the input value. The value of the Count field is used to distinguish between the two request types. While the action taken when the Count field is not 1 or 3 is undefined, all devices must use the value of Count to determine the size of the data payload accompanying the request in order to forward it properly, regardless of its value.

From a transaction perspective, an Atomic RMW request is a nonposted write that generates a read response. The read response packet contains a single quadword—that being the original value at the addressed location. Note that for Compare and Swap, the value of the Count field in the response packet is different from that in the associated request packet.

It is expected that Atomic RMW requests will be generated by HyperTransport I/O devices or bridges and directed to system memory (DRAM) that is controlled by the host. Therefore, the Compat bit will normally be cleared. No target other than the host is required to support atomic operations, and hosts are not required to support atomic operations to address ranges outside of system memory. If a target receives an unsupported atomic operation, it may either return a one-quadword read response~~,~~ with the Error bit set, or it may perform the RMW in a non-atomic way. If a host receives an atomic operation that does not target system memory, it may either abort it or reflect it as a peer-to-peer cycle. Peer-to-peer reflection of atomic operations will be required in a future revision of this specification.

Unlike the RdSized request packet, the Atomic RMW request packet does not contain the RespPassPW, Isoc, or Coherent bits in the command field of the packet, as defined in Table 7. The implied values of these bits are as follows:

- Coherent: 1—The addressed data may be cached.
- Isochronous: 0—Isochronous Atomic RMW requests are not supported.
- RespPassPW: 0—The response to the Atomic RMW request may not pass posted writes.

All nonposted requests (such as Reads, Nonposted Writes, Flushes, and Atomic RMW) for a given UnitID must have a unique SrcTag value for each outstanding request.

4.5 Responses

4.5.1 Read Response (RdResponse)

A node that is the target of a request for data (such as Sized Read or Atomic ~~Read-Modify-Write~~RMW request) returns a read response packet to the source followed by a data packet that contains the requested data. The format of the read response packet is shown in Table 17.

Table 17. Read Response (RdResponse) Packet Format

Bit-Time	7	6	5	4	3	2	1	0
0	Isoc	Rsv	Cmd[5:0]					
1	PassPW	Bridge	Rsv	UnitID[4:0]				
2	Count[1:0]		Error	SrcTag[4:0]				
3	Rsv		NXA	Rsv			Count[3:2]	

The Count field encodes the size minus 1 (in doublewords) of the data packet, so that intermediate nodes forwarding the response know how much data to expect. For doubleword read requests, the Count is just taken from the request packet. For byte read requests, the data field always fits within a single doubleword, so the Count field is always 0 (one doubleword). For Read-Modify-Write requests, the Count field is always 1 (one quadword).

The Error bit is used to indicate that an error occurred during the read. This can be due to the accessed address being non-existent, an ECC/Parity error in DRAM or a cache, or other problems. The requested number of data elements are always driven to the ~~bus~~chain, whether they are valid or not, but the Error bit indicates that the data cannot be used. A data packet with 1s in all data bit positions must follow a read response packet with the Error and NXA bits set. The NXA bit is reserved if the Error bit is not set.

The Isoc bit is set to indicate that this response ~~should flow in the Isochronous response channel~~has special bandwidth and latency requirements, and must be set if the Isoc bit was set in the request. See Appendix D for details. The Isoc bit is required to be maintained even when passing through a tunnel with Isochronous flow control mode disabled. Host bridges should return the value of Isoc from the request when forwarding peer-to-peer responses.

The SrcTag field is copied from the request that caused the response.

4.5.2 Target Done (TgtDone)

Target Done (defined in Table 18) signals that a transaction not requiring returned data (such as Sized Write or Flush) has completed at its target.

Table 18. Target Done (TgtDone) Format

Bit-Time	7	6	5	4	3	2	1	0
0	Isoc	Rsv	Cmd[5:0]					
1	PassPW	Bridge	Rsv	UnitID[4:0]				
2	Rsv		Error	SrcTag[4:0]				
3	Rsv		NXA	Rsv				

The target can release its command buffer as soon as it issues TgtDone. A nonposted request will result in either a RdResponse or a TgtDone, but not both. TgtDone also has the Error bit, similar to the one in RdResponse, which is used to indicate an error at the target in those cases where the error is detected prior to sending the response.

The Isoc bit is set to indicate that this response has special bandwidth and latency requirements,~~should flow in the isochronous response channel~~ and must be set if the Isoc bit was set in the request. See Appendix D for details. The Isoc bit is required to be maintained even when passing through a tunnel with Isochronous flow control mode disabled. Host bridges should return the value of Isoc from the request when forwarding peer-to-peer responses.

The SrcTag field is copied from the request that caused the response.

4.6 I/O Streams

HyperTransport technology has the concept of I/O streams, which are groupings of traffic that can be treated independently by the fabric.

Because no peer-to-peer communication exists within the fabric, and all packets travel either to or from the host bridge, the traffic to or from each node in the fabric could, in theory, be treated independently by the fabric, leaving the host bridge to manage interactions between streams.

Upstream requests contain the ID of the source node, and upstream responses contain the ID of the node that generated the response. Therefore, UnitID may be used to identify I/O streams for upstream packets. Note that Fence requests occupy all UnitIDs (see Section 4.4.4 for details).

Downstream responses contain the ID of the node to which the response is being sent. However, downstream requests contain the ID of 0 (the encoding reserved for the host bridge), and not the ID of the node that is targeted by the request. Therefore, it is impossible to determine independent I/O streams in downstream request traffic, and it must be assumed that all downstream traffic (both requests and responses) is in the same stream.

The host bridge is responsible for managing interactions between streams. No stream information is propagated through the host bridge. The host bridge is responsible for maintaining ordering within the host domain in whatever fashion is appropriate.

A single physical node can be allocated multiple UnitIDs if the node generates multiple independent streams of traffic. This allows more concurrency among the traffic to and from that device. If allocating multiple UnitIDs is not done, all traffic to and from that device will be ordered as a single stream, and knowledge of the possible concurrency will be lost.

4.7 Virtual Channels

HyperTransport technology supports three virtual channels of information:

- Posted Requests
- Nonposted Requests (reads, flushes, nonposted writes)
- Responses

Requests may cause responses to be issued by receiving nodes. Requests received by a host bridge may also cause downstream requests to be issued (peer-to-peer reflection). Other nodes may not make accepting a request either dependent on the ability of that node to issue an outgoing request or dependent upon the receipt of a response due to a request previously issued by that node. Further, they may not make issuing a response dependent upon the ability to issue a request or dependent upon receipt of a response due to a previous request.

All devices must guarantee that the three virtual channels are not capable of blocking each other due to buffer management and routing issues, which is why each channel has command and data buffer space separate from the other two. However, in order to properly maintain I/O ordering, some rules are added which create dependencies between packets (in the same I/O stream) in different virtual channels. See Chapter 6.

Note that in a shared double-hosted chain, if the chain is not partitioned between the two host bridges, there is the possibility of a deadlock. A deadlocking loop can be formed if peer-to-peer requests are issued in opposite directions by two different intermediate nodes. Each reflected peer-to-peer request coming out of a host bridge can be blocked behind a stack of requests targeting the other host bridge. The host bridge will only be able to queue a finite number of peer-to-peer requests in from the link without issuing one. Similarly, for a host bridge connected to two chains, each of which is terminated by another host, a deadlocking loop can be formed if a device on each chain is attempting to send peer-to-peer requests to each other in the direction away from the common host bridge. See Section 4.1.1 for more information on double-hosted chains.

HyperTransport technology includes support for an optional operating mode in which the number of virtual channels is doubled to support Isochronous operation. See Appendix D for more information.

4.8 Flow Control

HyperTransport technology receivers contain the following types of buffers:

- Non~~-P~~posted Requests
- Posted Requests
- Responses

- Nonposted Request Data
- Posted Request Data
- Response Data

Request and response buffers contain enough storage to store the largest control packet of that type. All data buffers can hold 64 bytes.

Table 7 defines the virtual channels and the buffers used for each of the control packets.

These buffers are flow-controlled at the link level using a coupon-based scheme in which the transmitter contains a counter for each type of buffer at the receiver. At link reset, the transmitter clears its counters, and when reset deasserts, the receiver sends NOP packets to indicate how many buffers of each type it has available. When the transmitter sends a non-info packet, it decrements the associated counter, and when a particular counter contains a 0, the transmitter stops sending packets to the associated buffer. When the receiver frees a buffer, it sends a NOP packet to the transmitter, and the transmitter increments the associated counter.

A transmitter cannot issue a control packet that has an associated data packet unless the receiver has both the appropriate control and data buffers available. If this rule is violated, one virtual channel can block another and lead to deadlock, because commands with associated data packets cannot be interleaved on the link.

HyperTransport technology supports an optional ~~operation~~ operating mode in which the number of virtual channels and associated flow control buffer types is doubled in order to support high-priority Isochronous communication. See Appendix D for details.

It is the responsibility of nodes generating requests to be able to accept the resulting responses without other dependencies. The common way to do this is to pre-allocate enough space for all responses (including response data). Otherwise, the response and/or response data flow control buffers may get filled with responses that are not yet ready to be accepted by the internal node logic. Due to peer-to-peer requests, host bridges are exempt from this rule.

It is also required for deadlock avoidance that devices always be able to accept posted requests without any other dependencies (such as issuing cycles back to the same chain or receiving responses from the chain). Due to peer-to-peer requests, host bridges are exempt from this rule.

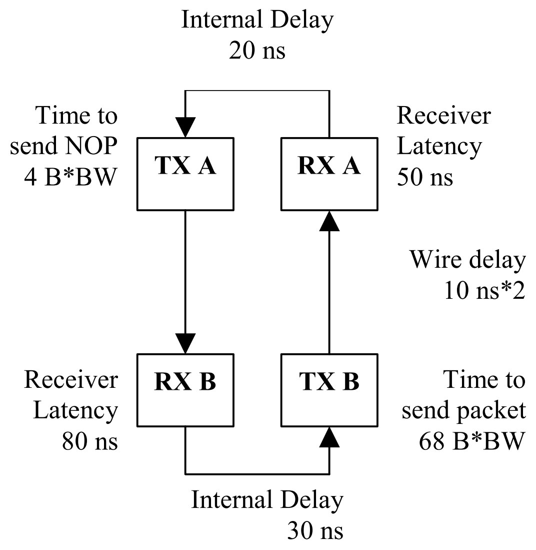
The format of the NOP packet is shown in Table 19. Bit 7 of bit-time 2 within the NOP packet is used to allow link interface hardware to differentiate a HyperTransport I/O device from a host device that implements a superset of the HyperTransport I/O protocol. Such a protocol could be used for the purpose of communication between devices inside the host. The link transmitter of a HyperTransport I/O device must always place a 0 in this bit position. The link receiver of a HyperTransport I/O device may ignore ~~it~~ the bit completely.

Table 19. NOP Packet Format

Bit-Time	7	6	5	4	3	2	1	0
0	Rsv	DisCon	Cmd[5:0]					
1	ResponseData[1:0]		Response[1:0]		PostData[1:0]		PostCmd[1:0]	
2	0	Diag	Isoc	Rsv	NonPostData[1:0]		NonPostCmd[1:0]	
3	Rsv							

Each 2-bit field in the packet indicates how many buffers of each type have become available. Hence each 2-bit field can free zero, one, two, or three buffers. Receivers are not limited to having three buffers of a particular type, and they can free up additional buffers by sending additional NOP packets.

While the goal is to size each buffer at the receiver to bury the round-trip latency from the transmitted packet to the returning NOP packet, this is not strictly required by this specification. It is the responsibility of each device to guarantee that NOP packets cannot be prevented from being issued due to transmission of other traffic, to avoid starvation of the far transmitter.



Notes:
For a minimal 2-bit 400-Mbit/s link,
Max Bandwidth: 100 Mbytes/s
Total loop delay: 200 + 720 ns
*Data in loop = 100 Mbytes/s * 920 ns, or 92 bytes*
So, in this case, at least two buffers are needed.

For a 32-bit, 2-Gbit/s link,
Max bandwidth: 8 Gbytes/s
Total loop delay: 200 + 9 ns
Data in loop = 1672 bytes
So, at least 27 buffers would be needed to bury latency in this case.

Figure 4. Example Data Buffer Sizing Calculation

Figure 4 illustrates how to calculate the size of data buffers that bury latency. For example:

If a transmitter receives more increments than it can keep track of, it must not allow its counter to wrap, but must discard the extras (saturate). This has the effect that the link will use the maximum number of buffers that both the transmitter and receiver can support. All transmitter counters must be a minimum of four bits wide, allowing up to 15 buffers to be tracked without loss.

The Diag bit is used to indicate the beginning of a CRC testing phase, as described in Appendix G. Everything following the NOP packet, until the conclusion of the current CRC test interval, is ignored. This test feature is optional—receivers are not required to implement support for this test mode. Support for this mode is indicated in bit 2 of the Feature Capability register, described in Section 7.5.10.3.

The DisCon bit is set to indicate that the link transmitter is beginning an LDTSTOP# disconnect sequence. When this bit is set, all the buffer-release fields in the packet must be 0. See Section 8.3 for details. While support for this bit is optional in non-x86 systems because the LDTSTOP# feature is optional, it is recommended in all systems for greater interoperability.

The Isoc bit is set to indicate that the flow-control information in the associated packet pertains to the Isochronous virtual channels. Isochronous flow-control information ~~should~~ must only be sent and utilized when the link has Isochronous flow control mode enabled, as described in Section 7.5.4.9.

4.9 Routing

HyperTransport technology has both directed and broadcast requests. Directed requests may travel in either the posted or nonposted channel; broadcast requests travel only in the posted channel. Directed packets are relayed down the ~~fabric~~ chain until they reach their destination, where they are absorbed. Broadcast packets are relayed down the entire length of the ~~fabric~~chain, but they are also accepted at each node they pass through, and they are terminated by the node at the far end of the ~~fabric~~chain. Broadcast packets can only be initiated by a host bridge.

An attempt to issue or forward a new packet into the end of the chain will result in one of the rejection outcomes described in Section 4.9.3.

4.9.1 Acceptance

A node will accept an incoming packet if any of the following are true:

- The packet is a Broadcast request.

~~The packet is a configuration space request targeted at a device or bus owned by this node.~~

- The packet is a directed request with a UnitID of 0 (indicating it is from a host bridge) and (for packets with a Compat bit) the Compat bit clear, to an address owned by this node.
- The packet is a directed request with a UnitID of 0 and a set Compat bit, and this node is either the subtractive decode device or a bridge to it.
- The packet is a response with the Bridge bit set (indicating it is from a host bridge) and a UnitID owned by this node.
- Tunnels must be able to accept downstream packets from either link in a double-hosted chain.

4.9.2 Forwarding

Whenever a node forwards a packet, it always sends the packet along the direction it was previously traveling.

A node will forward an incoming packet to its outgoing link if any of the following are true:

- The packet is a Broadcast request.

~~□The packet is a configuration space request targeted at a device or bus not owned by this node.~~

- The packet is a directed request with a UnitID of 0 (indicating it is from a host bridge) and (for packets with a Compat bit) the Compat bit clear, to an address not owned by this node.
- The packet is a directed request with a UnitID of 0 and a set Compat bit, and this node is neither the subtractive decode device nor a bridge to it.
- The packet is a directed request with a non-zero UnitID (indicating that it is from an interior node).
- The packet is a response with the Bridge bit set (indicating it is from a host bridge) and a UnitID that does not match this node.
- The packet is a response with the Bridge bit clear (indicating it is from an interior node).

A HyperTransport technology device may receive a request from one link that should be forwarded to the other link while its End of Chain and Initialization Complete Configuration Space Register (CSR) bits are still clear. In this case, the Drop on Uninitialized Link bit defined in Section 7.5.3.2.5 determines if the device will pend the request until the End of Chain or Initialization Complete CSR bit becomes set (indicating that the initialization attempt has completed), or reject the packet. See Section 12.3 for an example of an initialization sequence that makes use of this requirement.

A packet being forwarded to a link interface that has its End of Chain bit set is rejected. See sections 7.5.4.5 and 7.5.4.6 for the definitions of the End of Chain and Initialization Complete CSR bits and more details on how they can affect forwarding.

4.9.3 Rejection

A device at the end of the ~~fabric~~ chain is indicated by the End of Chain CSR bit (or by the Initialization Complete bit clear when the Drop on Uninitialized Link bit is set). In that case, the device is unable to forward packets or issue them in the direction of the unusable link. If a packet is rejected, one of the following actions is taken instead, depending on the type of packet:

- Broadcast requests are silently dropped—they have successfully traversed the whole ~~fabric~~chain.

- Nonposted downstream-directed requests (those with a UnitID of 0) are responded to with a TgtDone (for Writes) or Read Response (for Reads) packet with the Error bit set, the NXA bit set, and the Bridge bit clear~~, and a UnitID of 0~~. The UnitID of the response may be either 0 or the UnitID of the responding device. Read responses return the requested number of doublewords with a data value of all 1 bits. See Section 10.1.5 for more details.
- Nonposted upstream-directed requests (those with a nonzero UnitID) are responded to with a TgtDone (for Writes) or Read Response (for Reads) packet with the Error bit set, the NXA bit set, the Bridge bit set, and a UnitID matching that of the request, even when the responder is not the host of the chain. Read responses return the requested number of doublewords with a data value of all 1 bits. See Section 10.1.5 for more details.
- Response and posted request packets are dropped. See Section 10.1.5 for more details.

4.9.4 Host Bridges

Host bridges are always at the ends of the ~~fabric~~chain, and therefore never forward packets. However, the acceptance of a packet by a host bridge will likely result in action within the host.

Host bridges take the following action upon receiving a packet:

1. Directed requests with a UnitID of 0 must be coming from another host bridge on the far end of a double-hosted chain. Type 0 configuration accesses to the device number specified in the Device Number register (see Section 7.5.3.3.3) are directed to the bridge CSRs if the host supports use in a double-hosted chain and the Host Hide bit is clear (see Section 7.5.3.3.5). Optionally, the host bridge can also implement a memory or I/O space region addressable from the far host bridge to be used for messaging in clustered systems. (A description of how this would be used and what it would look like is beyond the scope of this specification.) In that case, the bridge would respond to accesses to this area as if it were an interior node. The responses would have the Bridge bit clear and a UnitID of 0 unless the responding host has its Act as Slave bit set (see Section 7.5.3.3.6), in which case responses will carry the value of the host's Device Number register, as defined in Section 7.5.3.3.3. All requests to addresses not included above are considered accesses to nonexistent addresses. If nonposted, they are responded to with a Target Done (for Writes) or Read Response (for Reads) packet, with the Error bit set, the NXA bit set, the Bridge bit clear, and a UnitID of 0. Read responses return the requested number of doublewords, with a data value of all 1 bits. Posted requests to nonexistent addresses generate no response and are silently dropped. This can be reported as an end-of-chain error, as described in Section 10.1.5.
2. Broadcast requests must be coming from another host bridge on the far end of a double-hosted chain. They have successfully traversed the whole ~~fabric~~ chain and may be silently dropped. Optionally, the host bridge could also implement a region addressable by broadcasts from the far host bridge. (A description of how this would be used and what it would look like is beyond the scope of this specification.) In that case, the bridge would handle accesses to this area as if it were an interior node, and route the broadcast to the appropriate internal target.

3. Directed requests with a nonzero UnitID are from interior nodes, and they are accepted and handled by the node logic. Address decode within the host determines the proper destination for the request. This may be an internal destination, the same HyperTransport chain from which the request was received, or a different HyperTransport chain. When the request maps to a HyperTransport chain, it is issued on that chain with a UnitID of 0. A SrcTag (if nonposted) and SeqID are issued from the pool of tags and SeqIDs available for downstream cycles on that chain. Because the cycle is now a downstream cycle, the Compat bit may become set and the coherence bit cleared. All other fields are passed through unchanged.
   - A sequence of peer-to-peer requests in one I/O stream sent upstream with matching sequence IDs must be reissued downstream with matching sequence IDs.
   - Hosts may implement a compatibility chain, to which requests that map to no other target may be sent. If no compatibility chain exists and the request maps to no internal target, then the request has reached the end of chain, and is treated like a rejected cycle, as described in Section 4.9.3. Note that HyperTransport host bridges that implement bridge headers always have a target from the HyperTransport chain's point of view—the primary bus of the bridge. If no target is found on this internal bus, that error occurs internally to the bridge, not on the HyperTransport chain. In this case, an error response must be handled according to the rules of Section 10.2.1.
   - HyperTransport technology hosts must maintain information about nonposted peer-to-peer requests that were forwarded so that when responses are returned from the target chain, responses with the correct attributes for the original request can be issued on the source chain. (The implementation details of the structures used to maintain this information are beyond the scope of this specification.)
   - Hosts that support upstream configuration cycles must convert upstream Type 1 requests to the current bus number to downstream Type 0 requests.
4. Responses with the Bridge bit set are silently dropped. This means that a host bridge tried to respond to an interior node that did not pick up the response. The node can choose to log this error and report it as a response error, described in Section 10.1.7.
5. Response packets with the Bridge bit clear are responses to requests issued by this bridge. The host bridge will match this to one of its outstanding requests. If no match exists, the node can choose to log this error and report it as a response error, described in Section 10.1.7.

4.9.5 Fairness and Forward Progress

In order to issue packets, a node must insert them into the stream of traffic that it is forwarding. A node must guarantee that forward progress is always made by not allowing forwarded and injected traffic to starve each another. Tunnels are required to implement the method described in this section of assuring fair access to the ~~fabric~~ chain for all units, approximating the round-robin behavior of a fair bus. Some HyperTransport technology devices may be used in applications where the fairness consideration is not relevant. One such example is a simple Southbridge that is always placed at the end of a HyperTransport chain.

4.9.5.1 Policy

Each unit is allowed to insert packets into a busy link at a rate matching that of the heaviest unit inserting traffic through it. In addition, the unit can freely use any idle time on the link. This property must be met over a window in time small enough to be responsive to the dynamic traffic patterns, yet large enough to be statistically convergent. In order for a system of units to behave consistently, each unit must implement this policy using the same algorithm as described below.

Generation of Info packets (NOP and Sync) is not restricted by this algorithm, since they exist on a per-link basis and are not forwarded.

4.9.5.2 Algorithm

The algorithm consists of two parts.

- The first is the method used to calculate the insertion rate the unit can use.
- The second governs how the unit achieves that insertion rate.

This algorithm must be implemented independently for both the upstream and downstream direction to support double-hosted chain configurations. The algorithm requires no dedicated control or status registers and has no configurable parameters.

To calculate the insertion rate, the maximum packet-forwarding rate must be deduced for the heaviest downstream unit. This is done by implementing 32 3-bit counters, one for each potential downstream unit as well as a single 8-bit counter. At reset, all counters are reset to 0. When a packet is forwarded the 3-bit counter corresponding to the unit that generated the packet is incremented. The 8-bit counter is incremented once for every forwarded packet. When one of the 3-bit counters overflows, the value of the 8-bit counter (post increment) is captured (hereafter referred to as the denominator). All counters are then cleared. The packet rate of the worst-case downstream unit has now been calculated and is equal to 8/denominator. On average, the unit can insert eight packets for every ‘denominator’ packets forwarded. This insertion should be paced and not inserted as bursts. Packets can always be inserted when there are no packets waiting to be forwarded. The denominator register is set to 1 on reset.

To insert, the unit has an 8-bit counter referred to as Window, which at reset is set to 1. It also has a 1-bit register, referred to as Priority, that is cleared to 0 at reset. When a unit has packets ready to be sent on the outbound links, it decides which to send based on the following cases:

- Forward packet to send and no local packet to send—The forward packet is sent and the Window register is decremented.
- No forward packet to send and a local packet to send—The local packet is sent and the Priority register is cleared.

- Both forward packet and local packet to send—If the bit in the Priority register is set, the local packet is sent and the Priority bit is cleared. Otherwise, the forward packet is sent and the Window register is decremented.

Whenever the Window register is decremented to 0, its next value is recalculated and the Priority bit is set. In order to achieve non-integral insertion rates, the new value of the Window register must be loaded probabilistically. Each unit will implement a 9-bit linear feedback shift register using the polynomial x^9+x^4+1. It is advanced once every time the Window register value is recalculated. The Window register is loaded with (denominator+LFSR[2:0]) >> 3.

4.9.5.3 Implementation Note

Care must be taken in implementing the packet insertion logic in order to avoid a potential starvation problem. The packet inserter is basically a two-input arbiter between issued packets and forwarded packets. The packets to this arbiter are generated when there is a packet ready to go from one of these sources, and there are free buffers (as indicated by buffer release messages) at the other end of the link to receive the packet. It is possible that there is one packet to be issued and one to be forwarded, both in the same virtual channel and therefore requiring the same buffer type(s). If the forwarded packet is chosen and there is only one buffer of the needed type free, the issued packet cannot be transmitted. When the fairness logic next allows a packet to be inserted, a packet from a different virtual channel can be chosen, allowing the priority of the packet inserter to swing back to forwarding. Upon arrival of the buffer release message that would allow the blocked packet to go, the packet no longer has priority in the inserter, and therefore cannot go. If another packet in the same channel is forwarded before priority changes back to inserting, this situation can persist, starving packet insertion in a particular virtual channel.

5 Addressing

The HyperTransport™ technology address map is shown in Table 20~~Table 47Table 47~~. The bulk of the address space can be used for either memory or memory-mapped I/O. The partitioning of this space into regions for each is implementation-specific. Unlike PCI, configuration and I/O accesses are performed with the same read and write commands used to access memory, with the upper address bits identifying the accesses instead of special command codes.

Table 20. HyperTransport™ Technology Address Map

Base Address	Top Address	Size	Use
00_0000_0000h	FC_FFFF_FFFFh	1012 Gbytes	~~DRAM~~System Memory/ Memory-Mapped I/O
FD_0000_0000h	FD_F8FF_FFFFh	3984 Mbytes	Interrupt/EOI
FD_F900_0000h	FD_F90F_FFFFh	1 Mbyte	Legacy PIC IACK
FD_F910_0000h	FD_F91F_FFFFh	1 Mbyte	System Management
FD_F920_0000h	FD_F92F_FFFFh	1 Mbyte~~s~~	Reserved – x86
FD_F930_0000h	FD_FBFF_FFFFh	45 Mbytes	Reserved
FD_FC00_0000h	FD_FDFF_FFFFh	32 Mbytes	I/O
FD_FE00_0000h	FD_FFFF_FFFFh	32 Mbytes	Configuration
FE_0000_0000h	FF_FFFF_FFFFh	8 Gbytes	Reserved

While the interrupt, IACK, and system management spaces may not be used by all devices, they must not be used for any other function, and all devices must forward them properly.

Some hosts only recognize interrupts with Address[31:24]=F8h.

Writes to configuration and I/O space must be nonposted and cannot cross a doubleword boundary. Posted writes or broadcasts to configuration or I/O space may result in undefined operation. Accesses of more than one doubleword to configuration or I/O space may be aborted by the target.

Similarly, writes to interrupt and system management spaces must be posted. Nonposted writes to interrupt or system management space may result in undefined operation.

Writes or broadcasts to IACK space may result in undefined operation. Upstream accesses to IACK space may result in undefined operation.

HyperTransport technology devices must have address windows aligned on 64-byte boundaries to guarantee that a maximum-size request will not cross a device boundary.

6 I/O Ordering

This chapter explains the ordering rules for all three types of I/O traffic—peer-to-peer, DMA, and PIO. *Peer-to-peer traffic* is traffic that has both its requester and target on the HyperTransport™ I/O link. Since HyperTransport technology does not allow peer-to-peer traffic directly, all peer-to-peer traffic (whether to the same ~~fabric~~ chain or not) goes upstream into the host and then back downstream. For purposes of ordering, the upstream and downstream legs are considered independently.

These ordering rules only apply to the order in which operations are seen by targets at the same level of the fabric hierarchy. Consider two ordered peer-to-peer Write requests issued by a HyperTransport I/O device to two different targets on different HyperTransport I/O ~~fabrics~~chains. The ordering rules on the originating HyperTransport chain ensure that the two Writes reach the host bridge in the appropriate order. The host is responsible for ensuring that the two Writes reach their target host bridges in the correct order. However, beyond that point, the Writes are in independent ~~fabrics~~chains, and there is no assurance about the order in which they will reach their final target. If an I/O device requires assurance of final completion, it must have a way of polling the target device to determine that the first Write has been observed before issuing the second Write, or it must use nonposted Writes.

Ordered operations that return responses (Reads or nonposted Writes) are required to complete at the target in the correct order, but no assurance is made about the order in which the returning responses will be received. All HyperTransport I/O devices must be able to accept responses out of order or restrict themselves to one outstanding nonposted request. A bridge that is between a HyperTransport technology device and an I/O protocol that requires responses to be returned in order must provide sufficient buffering to be able to reorder as many responses as it may have outstanding requests. Devices must issue responses to nonposted requests only after the results of those requests are globally visible.

HyperTransport supports the same producer-consumer ordering model as PCI when the PassPW and sequence ID bits are 0. In this model, a producer device anywhere in the system can generate data and modify a flag to indicate data availability to a consumer of the data anywhere in the system. The flag and data may each be located at the producer device, consumer device, or host memory. They are not required to be located in the same device as long as the consumer waits for the flag read response before issuing the data read. In the case where the consumer issues two ordered reads with non-zero sequence Ids (not waiting for the flag read response before sending the data read) producer-consumer ordering is only supported when the flag and data are co-located in the same device. The ordering rules described in this section ensure that if the flag is modified after the data has been made available, a read of the flag by the consumer will ensure that all data can be read.

When the PassPW bit is 1, these ordering rules can be relaxed for applications where the flag and data are restricted in their locations. With the use of nonzero sequence IDs, HyperTransport links can maintain ordered sequences within otherwise unordered virtual channels.

HyperTransport ordering semantics ensure that a configuration access followed by a second ordered transaction are delivered to a device in order. HyperTransport devices should ensure that a subsequent ordered read to the same configuration location as a previous configuration access results in the latest written data. However, devices are not required to ensure that all side effects of the first configuration access are visible before receiving the second. Software can ensure the side effects of the first configuration access have become visible by waiting for the response to the first non-posted configuration access before issuing the second transaction.

6.1 Upstream I/O Ordering

HyperTransport technology recognizes three types of traffic—posted requests, nonposted requests, and responses—each in a separate virtual channel. These three types of traffic can be distinguished by their command encoding. Requests have a sequence ID (SeqID) tag. Requests in the same I/O stream and virtual channel with matching non-zero SeqIDs are considered part of a strongly ordered sequence. Sequences are designed to support groups of HyperTransport technology transactions generated by a single request on the source I/O bus. Requests and responses both have a May Pass Posted Writes (PassPW) bit.

For definitions of I/O streams and virtual channels, see sections 4.6 and 4.7, respectively.

HyperTransport technology has the following upstream ordering rules:

1. Packets from different sources are in independent I/O streams and with the exception of the Fence requests, have no ordering guarantees. Devices receiving packets in different I/O streams may reorder them freely.
2. Packets in the same I/O stream and virtual channel that are part of a sequence (having matching nonzero SeqIDs) are strongly ordered (regardless of PassPW) and may not pass each other. Devices receiving them must keep them strongly ordered.
3. Packets in the same I/O stream, but not in the same virtual channel or not part of the same ordered sequence, use the passing rules listed in Table 21.

Table 21. Packet Ordering Rules

Row Pass Column?	**Posted Request**	**Nonposted Request**	**Response**
Posted Request	PassPW: Yes/No !PassPW: No	Yes	Yes
Nonposted Request	PassPW: Yes/No !PassPW: No	Yes/No	Yes/No
Response	PassPW: Yes/No* !PassPW: No	Yes	Yes/No

Notes:

**—HyperTransport technology implementations are strongly encouraged to allow responses with PassPW set to pass posted requests. However, they cannot rely upon this behavior system-wide to ensure deadlock-free operation. Allowing responses with PassPW set to pass posted writes creates more deterministic latency on behalf of Isochronous read traffic. See Appendix D for more details.*

PassPW*—Relates to the packet represented by the row, not the column.*

No*—Indicates the subsequently issued transaction is not allowed to complete before the previous transaction to preserve ordering in the system. This implies an interaction between the otherwise independent virtual channels within HyperTransport technology.*

Yes*—Indicates the subsequently issued transaction must be able to pass the previous transaction, or deadlock may occur. This means that the packet type given in the column cannot be permitted to block the packet type given in the row at any point in the HyperTransport fabric or host.*

Yes/No*—Indicates the subsequently issued transaction may optionally be allowed to complete before the previous transaction if there is any advantage to doing so. There are no ordering requirements between the two transactions. However, support for reordering is not required—failure to reorder the packets will not lead to deadlock.*

6.2 Host Ordering Requirements

The host bridge and host system are required to preserve the ordering of transactions in the virtual channels provided in the HyperTransport I/O fabric as defined in Section 6.1, and to guarantee that transactions that are ordered within the HyperTransport fabric are ordered within the host. This means that, for an ordered pair of transactions, the second transaction cannot take effect in the host fabric (capturing data for a read to an internal target,~~or~~ exposing new data for writes and reads with side effects to internal targets, or queuing peer-to-peer I/O for transmission to an external target) until the first transaction has reached its ordering point. The definition of this ordering point depends on the type of transactions in the ordered pair and the relationship of their targets. ~~However, i~~In the case of ~~posted~~ peer-to-peer I/O operations, the host ~~can~~ only guarantees that the first operation has been queued for issue~~d~~ on its target link, and ordered with respect to requests from all other sources (Globally Ordered); it ~~has no way of knowing~~does not indicate whether the operation has reached its final target device.

Read or Write accesses from HyperTransport technology are treated differently depending on the target space within the host to which they are aimed. Accesses to cacheable ~~DRAM~~ system memory within the host have the strongest set of ordering requirements. Accesses to noncacheable regions (uncacheable ~~host DRAM~~system memory, I/O space, or memory-mapped space on an I/O device) have weaker requirements. Accesses to the reserved interrupt or system management ranges have their own special ordering requirements. The rules governing the host's processing of ordered HyperTransport I/O transactions are expressed in Table 22.

There are two defined ordering points, Globally Ordered (GO), and Globally Visible (GV). Table 22 defines what ordering point the first request in an ordered pair must reach before the second request can take effect.

Table 22. Host Ordering Rules

First Command	**Second Command**	**Second Command Waits for the First Command to Be:**
Cacheable Write	Cacheable Write	GV
Cacheable Write	Cacheable Rd	GO
Cacheable Read	Cacheable Read or Write	GO
Non-Cacheable	Non-Cacheable	GO
Cacheable Write	Non-Cacheable	GV
Cacheable Read	Non-Cacheable	GO
Non-Cacheable	Cacheable	GO
Cacheable Write~~/Fence~~	Flush/Interrupt/ System Management/Response	GV
Cacheable Read	Flush/Interrupt/ System Management/Response	No wait requirements
Non-Cacheable	Flush/Interrupt/ System Management/Response	GO
Flush/Response	Any	No wait requirements
Interrupt/System Management	Fence or Response	GV
Interrupt/System Management	Any but Fence or Response	No wait requirements
Posted Cacheable	Fence	GV
Posted Non-Cacheable	Fence	GO
Any Nonposted	Fence	No wait requirements

First Command	**Second Command**	**Second Command Waits for the First Command to Be:**
Fence	Any	GV
Notes: ***Globally Ordered (GO)****—The first transaction has reached a point where it is assured to be observed in the correct order (relative to the second transaction) from any observer in the HyperTransport fabric. The two transactions are assured to complete in order, but have not necessarily completed yet, so sideband access mechanisms such as cache agents will not necessarily receive the correct results.* ***Globally Visible (GV)****—The first transaction is visible to all ~~processors~~observers. That is, any ~~processor read~~access mechanism will return the new data. This means that in addition to being globally ordered, all side effects (such as cache state transitions) initiated by the first transaction have completed.* *Globally Visible implies Globally Ordered, so a host may use a more restrictive rule in some cases to simplify the implementation. In the absence of sideband access mechanisms such as caching agents in the host, they are equivalent. Both ordering points apply only to destinations in the chain host. They do not include effects caused by the transactions outside the host fabric.*		

6.2.1 Host Responses to Nonposted Requests

The host cannot generate a response to a nonposted request until all side effects of the request are globally visible. For a memory request this means that all cache state transitions initiated by the request have been completed. For I/O requests, this means that data writes or read side effects have occurred. A response to a nonposted request implies that all previous ordered requests to memory are globally visible. It also implies that all previous ordered requests to I/O have been globally ordered, but it cannot be assumed that they are globally visible.

6.3 Downstream I/O Ordering

The rules for downstream ordering are the same as those for upstream ordering, with the exception that I/O streams are identified by the target of the transaction, rather than the source. The same three virtual channels exist. However, UnitID may not be used to identify unique I/O request streams in the downstream direction, so it must be assumed that all downstream traffic is in the same stream. This asymmetry in the definition of I/O streams for upstream and downstream traffic is why it is important for a node to be able to differentiate upstream responses from downstream responses. The Bridge bit is used in the response packet for this purpose.

Once a node has accepted a packet, it has been separated from forwarded traffic and no longer must be ordered with forwarded traffic.

The host must also guarantee that peer-to-peer traffic that was part of an ordered sequence when received is also emitted downstream as an ordered sequence.

6.4 Ordering in Sharing Double-Hosted Chains

In general, upstream traffic and downstream traffic moving in the same direction along a HyperTransport chain have no ordering dependencies with respect to each other, as they will be in different I/O streams. The exception is the case of communication directly between host bridges at opposite ends of a double-hosted chain, as defined in Section 4.1.1. In this case, requests from one host bridge to the other are always traveling downstream, and responses from that host bridge are traveling upstream.

In the event that one host bridge (bridge A) issues a posted write to the other (bridge B), and bridge B issues a read request to A, the read response will be traveling in the same direction as the posted write. Despite the fact that the request is moving downstream and the response is moving upstream, both must be treated as being in the same I/O stream (the response must push the request if PassPW is clear) in order to support producer/consumer communication between the hosts.

In this case, both the request and response will contain a UnitID of 0. Therefore, this requirement can be supported simply by doing ordering checks based solely on UnitID for upstream responses, and excluding information about whether the request they are checking against is moving upstream or downstream. If a host has its Act as Slave bit set (defined in Section 7.5.3.3.6), the UnitID of requests and responses from it will not be 0. However, the requirement to maintain ordering still exists, and the use of UnitID to achieve this can be maintained.

7 Configuration Accesses

HyperTransport™ technology implements configuration space similarly to PCI, as defined in the *PCI Local Bus Specification, Revision 2.2*. HyperTransport technology devices and bridges (including host bridges) must implement appropriate PCI configuration headers. Buses and devices are numbered in a fashion that maps into PCI bus and device numbers. Configuration software should be able to accomplish configuration of HyperTransport chains in a way that is indistinguishable from an equivalent PCI bus hierarchy. Configuration space is mapped to a predefined region of the HyperTransport technology system address space. See Chapter 5~~9~~ for details.

7.1 Configuration Cycle Types

PCI uses two types of configuration cycles, type 0 and type 1. The two types are needed because it must be possible to access the configuration space of devices on a bus without the devices knowing on which bus they are located.

Type 0 cycles are used to access devices on the current bus. They contain a function number and register number. The bus number is implicitly the current bus. The device number is indicated by the IDSEL# pins, which are asserted as appropriate by the bridge. Therefore, PCI devices do not need to know their bus number or device number in order to respond to configuration accesses.

Type 1 cycles are used to transmit configuration cycles over intermediate buses. They contain bus number, device number, function number, and register number fields. Bridges forward type 1 cycles through the bus hierarchy and translate them to type 0 cycles when driving them onto their final destination bus. Host bridges can optionally implement the capability to transmit PCI special cycles to remote buses using Device 31, Function 7, Register 0, Type 1 configuration cycles.

HyperTransport technology also requires two types of configuration cycles, for the same reasons as PCI.

A HyperTransport technology Type 0 access is performed by issuing a RdSized or nonposted WrSized request with an address of the form shown in Table 23. They are only issued by host bridges and therefore always travel downstream. Unlike PCI, HyperTransport technology Type 0 accesses contain the device number, because all HyperTransport technology devices know what their device numbers are. HyperTransport technology has no analog of the IDSEL# signals. Host bridges that support double-ended links will respond to Type 0 accesses on their secondary interfaces at the Device Number specified in Host Interface Command register. See Section 7.5.3.3.3. Note that, in a double-hosted link, this implies that both bridges could be responding to the same address—which one you are talking to is determined by which direction the packets are traveling. This function is only intended to be used by system-sizing firmware.

Table 23. HyperTransport™ Technology Type 0 Access Format

39 24	23 16	15 11	10 8	7 2
FDFEh	Reserved	Device Number	Function Number	Register Number

A HyperTransport technology Type 1 access is performed by issuing a RdSized or nonposted WrSized request with an address of the form shown in Table 24. In general, Type 1 accesses are issued by host bridges. Unlike PCI, all HyperTransport technology Type 1 accesses can flow upstream and must be forwarded by HyperTransport-to-HyperTransport bridges, as well as hosts that support sharing double-hosted chains. Furthermore, these devices must support Type-1-to-Type-0 conversion upstream in addition to downstream and when re-issuing peer-to-peer cycles downstream.

Table 24. HyperTransport™ Technology Type 1 Access Format

39 24	23 16	15 11	10 8	7 2
FDFFh	Bus Number	Device Number	Function Number	Register Number

RdSized and WrSized configuration accesses of greater than one doubleword are not supported.

Posted configuration writes are not allowed and their effect is undefined.

7.2 Configuration Space Mapping

7.2.1 Function and Register Numbering

The numbering of functions and registers within a device is device-specific, except that every implemented device number must have a function 0 containing a standard configuration header that identifies the device. Certain other standard Configuration Space Registers (CSRs) are required by the HyperTransport technology specification.

7.2.2 Device Numbering

HyperTransport technology devices are identified by UnitIDs, which range from 00h to 1Fh. A single physical device can own multiple UnitID values. Every HyperTransport technology device owns the device numbers that correspond to its UnitIDs, and it must implement a configuration space (with configuration header) at the Device Number equal to its Base UnitID value. It may choose to implement configuration spaces corresponding to any number of its remaining UnitIDs, including none. Each implemented space must contain an appropriate configuration header. Unimplemented spaces must not be responded to by the device. Accesses to these device numbers will not be accepted by any device on the HyperTransport chain and therefore will receive a response with the error and NXA bits set.

As described in Appendix E.3, some systems require a compatibility chain that is enumerated as Bus 0. In such a system, any configuration space registers implemented in the Bus 0 space by the host must appear in the uppermost device numbers on that bus. In such a case, the number of devices (and therefore UnitIDs) available to the HyperTransport chain implementing Bus 0 is reduced accordingly. If the host were to occupy Device 0, then the HyperTransport chain could not be enumerated.

As described in Appendix F.4, some legacy operating systems may require AGP configuration registers to be implemented in the Bus 0, Device 0 range, in which case the host's configuration space registers must appear somewhere other than that range. Even though the AGP CSRs appear in Device 0, UnitID 0 is still reserved for host use so that devices can distinguish upstream cycles from downstream cycles, as described in Section 4.9. This requires the AGP bridge to consume at least one extra UnitID, as it cannot use UnitID 0.

For HyperTransport technology hosts that also implement AGP configuration space and require legacy operating system compatibility, the host may either:

- Make configuration space relocatable so that HyperTransport bus enumeration may occur.
- Place configuration space in the uppermost device numbers and provide a mechanism for hiding Device 0 registers to allow HyperTransport bus enumeration.

The means to accomplish either of these two actions, or perhaps other solutions to this problem, are implementation-specific and beyond the scope of this specification.

7.2.3 Bus Numbering

Each HyperTransport chain in the system is assigned a single bus number. For double-hosted physical chains, which are logically partitioned between two host bridges, each logical chain has a separate bus number.

Bus numbers are assigned by system initialization software at reset and follow the conventions used for PCI bus numbering, which require that the bus tree be numbered in a depth-first fashion. No distinction is made between PCI buses and HyperTransport chains in the numbering.

7.3 HyperTransport™ Technology Device Header

Devices that sit on a HyperTransport technology chain and perform non-bridging functions implement device headers, as shown in Table 25. The fact that this is a device header is contained in the header type register. Connecting to two links within a HyperTransport chain does not constitute a bridging function. Fields in a HyperTransport technology device header are the same as defined in the *PCI Local Bus Specification, Revision 2.2*, with the exceptions listed in the following sections.

Table 25. HyperTransport™ Technology Device Header Format

31 24	23 16	15 8	7 0	
Device ID		Vendor ID		**00h**
Status		Command		**04h**
Class Code			Revision ID	**08h**
BIST	Header Type	Latency Timer	Cache Line Size	**0Ch**
Base Address Registers				**10h**
				14h
				18h
				1Ch
				20h
				24h
Cardbus CIS Pointer				**28h**
Subsystem ID		Subsystem Vendor ID		**2Ch**
Expansion ROM Base Address				**30h**
Reserved			Capabilities Pointer	**34h**
Reserved				**38h**
Max_Lat	Min_Gnt	Interrupt Pin	Interrupt Line	**3Ch**
Note: *Shaded registers contain minimum-required read-write bits. Other registers are read-only or contain only device-dependent bits.*				

Each field is defined as readable and writeable by software (R/W), readable only (R/O), or readable and cleared by writing a 1 (R/C). Additionally, each field is affected by cold reset only or by both cold and warm reset.

7.3.1 Command Register: Offset 04h

The following bits are implemented in the Command register of a HyperTransport technology device. All other bits are not applicable and must be hardwired to 0.

7.3.1.1 I/O Space Enable (Bit 0): R/W: Warm Reset to 0

This bit must be set for the device to accept any non-compatibility accesses (requests with the Compat bit clear) to the I/O address space, as given in Chapter 5~~9~~. If this device is a subtractive-decode device, requests with the Compat bit set will still be accepted, regardless of the state of this bit.

7.3.1.2 Memory Space Enable (Bit 1): R/W: Warm Reset to 0

This bit must be set for the device to accept any non-compatibility accesses to the memory address space, as given in Chapter 5~~9~~. If this device is a subtractive-decode device, requests with the Compat bit set will still be accepted, regardless of the state of this bit.

7.3.1.3 Bus Master Enable (Bit 2): R/W: Warm Reset to 0

This bit must be set to allow the device to issue memory or I/O requests onto the HyperTransport chain. Requests from other devices on the chain may still be forwarded, independent of the state of this bit.

7.3.1.4 SERR# Enable (Bit 8): R/W: Warm Reset to 0

If this bit is set, the device will flood all its outgoing links with sync packets when it detects an error that causes a sync flood (see Section 10.2.3). If this bit is clear, the device may not generate sync packets except as part of initial link synchronization, although it can still propagate them from one link to the other within a chain.

7.3.2 Status Register: Offset 06h

The following bits are implemented in a HyperTransport technology device's Status register. All other bits are not applicable to HyperTransport technology and must be hardwired to 0.

7.3.2.1 Capabilities List (Bit 4): R/O

This read-only bit will always be set to 1, to indicate that the device has a capabilities list containing (at least) configuration information specific to HyperTransport technology.

7.3.2.2 Signaled Target Abort (Bit 11): R/C: Cold Reset to 0

This bit is set by a HyperTransport technology device that returns a Target Abort for a transaction addressed to it. (Target Aborts in HyperTransport technology are signaled by a non-NXA error response, as described in Section 10.2.1.)

7.3.2.3 Received Target Abort (Bit 12): R/C: Cold Reset to 0

This bit is set by a HyperTransport technology device that receives a Target Abort for a request it issued. (Target Aborts in HyperTransport technology are signaled by a non-NXA error response, as described in Section 10.2.1.)

7.3.2.4 Received Master Abort (Bit 13): R/C: Cold Reset to 0

This bit is set by a HyperTransport technology device that receives a Master Abort for a request it issued. (~~Target~~ Master Aborts in HyperTransport technology are signaled by an ~~non-~~NXA error response, as described in Section 10.2.1.)

7.3.2.5 Signaled System Error (Bit 14): R/C: Cold Reset to 0

This bit is set by a HyperTransport technology device that has flooded the link with Sync packets to signal a system error. See Section 10.2.3. A device that is only forwarding Sync packets from another device on the same chain should not set this bit, so that the device initiating the sync flood can be localized. Software will not be able to access the device (including this bit) via the flooded link until a reset has occurred.

7.3.3 Cache Line Size Register: Offset 0Ch: R/O

This register is not implemented by HyperTransport technology devices, and returns 0s if read.

7.3.4 Latency Timer Register: Offset 0Dh: R/O

This register is not implemented by HyperTransport technology devices, and returns 0s if read.

7.3.5 Base Address Registers (BARs): Offsets 10-24h: R/W: Warm Reset

Base Address registers (BARs) for HyperTransport technology devices are implemented as described in the PCI specification. Bit 0 is read-only and indicates if a BAR is for memory or I/O space.

Table 26. Memory Space BAR Format

N+1	N 4	3	2	1	0
Base Address	Size Bits	Prefetchable	64b	Rsv	0

Devices hardwire bits 4 through N of a memory BAR to 0 to indicate how much address space they require. For example, to allocate 4Kbytes of memory space, bits 4 through 11 would be read-only 0. Enumeration software will write 1's into the register and read it back to determine how much space to allocate. Bit 3 indicates if the memory space allocated to the BAR is prefetchable (can be read without side effects). Bit 2 indicates if a BAR is 32 or 64 bits wide. Incoming addresses in the memory-mapped I/O space address range (as described in Chapter 5~~9~~) are compared directly to the ~~values~~ writeable bits in memory space BARs. If the memory space BARs are programmed to support 64-bit addressing, then the incoming addresses from the HyperTransport technology link must be 0 extended to 64 bits before being compared to the BAR value. If the memory space BARs are programmed to support 32-bit addressing, the BAR value is 0 extended to 40 bits before being compared to the address. HyperTransport technology devices must have memory windows aligned on 64-byte boundaries to guarantee that a maximum-size request will not cross a device boundary and should therefore always indicate a size of at least 64 bytes. The reset value of these registers is implementation-specific, so software must initialize them before setting the Memory Space ~~or I/O Space~~ Enable bit~~s~~ in the Command register.

Table 27. I/O Space BAR Format

N+1	N 2	1	0
Base Address	Size Bits	Rsv	1

Like memory space BARs, I/O space BARs have bits 2 to N hardwired to 0 to indicate the amount of address space required. Incoming addresses in the I/O space address range (which is only a 25-bit space) have only their bottom 25 bits compared to the writeable bits of the I/O space BARs. Bits 31:26 of ~~the~~ an I/O space BAR must be 0 for a match to occur. ~~HyperTransport technology devices must have windows aligned on 64-byte boundaries to guarantee that a maximum-size request will not cross a device boundary.~~ The reset value of these registers is implementation-specific, so ~~firmware~~ software must initialize them before setting the ~~Memory Space or~~ I/O Space Enable bit~~s~~ in the Command register.

7.3.6 CardBus CIS Pointer: Offset 28h: R/O

This register is not implemented by HyperTransport technology devices and returns 0s if read.

7.3.7 Capabilities Pointer: Offset 34h: R/O

Every HyperTransport technology device has a capabilities pointer to a linked capabilities list that contains (at least) the capability registers specific to HyperTransport technology.

7.3.8 Interrupt Line Register: Offset 3C: R/W: Warm Reset

The Interrupt Line register should be readable and writeable and may be used by software as a scratchpad to track interrupt routing. The reset value is implementation-specific.

7.3.9 Interrupt Pin Register: Offset 3Dh: R/O

The Interrupt Pin register is reserved, since HyperTransport technology delivers interrupts via messages and does not define any interrupt pins, although devices that generate interrupts may need to provide this register for compatibility with existing software. (Some operating systems will not correctly utilize interrupts for a function without a non-zero value in its Interrupt Pin configuration space register.)

7.3.10 Min_Gnt, and Max_Lat Registers: Offsets 3E and 3Fh: R/O

These registers are not implemented by HyperTransport technology devices, and they return 0s if read.

7.4 HyperTransport™ Technology Bridge Headers

Devices that bridge between HyperTransport chains and other bus protocols that implement configuration mechanisms and bridge headers as described in the *PCI-to-PCI Bridge Architecture Specification Revision 1.1* (including subsidiary HyperTransport chains) implement such a bridge header, with the exceptions listed below. Note that HyperTransport technology can be the primary or secondary bus of the device, or both. Some register bits have meanings specific to HyperTransport technology only when a HyperTransport technology chain is connected to a specific port of the bridge. If that interface is to a non-HyperTransport bus, the requirements of that bus protocol determine the meaning of the bit.

Unless otherwise noted, the registers described in this section are reset by the reset mechanism of the primary bus and not that of the secondary bus. For a HyperTransport-to-HyperTransport bridge, this is HyperTransport technology RESET# of the primary chain. For a Host-to-HyperTransport bridge, this is a host-specific reset signal. Implementations that do not require the ability to reset the HyperTransport chain independently of the host may choose to combine the two.

Each field is defined as readable and writeable by software (R/W), readable only (R/O), or readable and cleared by writing a 1 (R/C). Additionally, each field is affected by only cold reset or both cold and warm reset.

Table 28. HyperTransport™ Technology Bridge Header Format

31 24	23 16	15 8	7 0	
Device ID		Vendor ID		**00h**
Status		Command		**04h**
Class Code			Revision ID	**08h**
BIST	Header Type	Primary Latency Timer	Cache Line Size	**0Ch**
Base Address Register 0				**10h**
Base Address Register 1				**14h**
Secondary Latency Timer	Subordinate Bus Number	Secondary Bus Number	Primary Bus Number	**18h**
Secondary Status		I/O Limit	I/O Base	**1Ch**
Memory Limit		Memory Base		**20h**
Prefetchable Memory Limit		Prefetchable Memory Base		**24h**
Prefetchable Base Upper 32 Bits				**28h**
Prefetchable Limit Upper 32 Bits				**2Ch**
I/O Limit Upper 16 Bits		I/O Base Upper 16 Bits		**30h**
Reserved			Capabilities Pointer	**34h**
Expansion ROM Base Address				**38h**
Bridge Control		Interrupt Pin	Interrupt Line	**3Ch**
Note: *Shaded registers contain minimum-required read-write bits. Other registers are read-only or contain only device-dependent bits.*				

7.4.1 Command Register: Offset 04h

All of the Command register bits implemented in the Device Header Command register (Section 7.3.1), are implemented in bridges that have a HyperTransport link on their primary bus, and they affect operation only on the primary bus. If the Bus Master Enable bit is cleared in a bridge device, preventing forwarding of transactions to the primary bus, transactions that would be forwarded must be master-aborted on the secondary bus. If the secondary bus is a HyperTransport link, this is indicated by returning a response with the NXA bit set.

In addition, bridge devices have the option of implementing the VGA Palette Snoop Enable bit (bit 5). If not implemented, it is hardwired to 0. This bit functions similarly to the way it is described in *PCI-to-PCI Bridge Architecture Specification Revision 1.1*. If the bit is set, WrSized operations to addresses in the first 64 Kbytes of the I/O address range (as defined in Chapter 5~~9~~) have address bits 9:0 compared to addresses 3C6h, 3C8h, and 3C9h. (Address bits 1:0 are not included in WrSized packets, and so must be determined from the address of the least significant enabled byte in the packet.) Accesses on a primary HyperTransport link that match are forwarded to the secondary bus of the bridge. Accesses on a secondary HyperTransport link that match are not accepted, regardless of the state of other address decode registers.

If this bit is set when the address mapping extensions described in Appendix A are in use, the address decode behavior of the device may be undefined.~~The VGA Palette Snoop Enable bit cannot be used with the address mapping extensions described in Appendix 0.~~

7.4.2 Status, ~~Cache Line Size,~~ Primary Latency Timer, Base Address, Interrupt Pin, and Interrupt Line Registers

For a bridge with a HyperTransport I/O link as its primary bus, these registers are implemented the same way as the corresponding registers in a HyperTransport technology device header. They are not related to the secondary bus.

7.4.3 Cache Line Size: Offset 0Ch: R/O

When both the primary and secondary buses are HyperTransport links, this register is reserved.

~~7.4.3~~7.4.4 Secondary Latency Timer Register: Offset 1Bh: R/O

When the secondary bus is a HyperTransport link, this register is reserved.

~~7.4.4~~7.4.5 Secondary Status Register: Offset 1Eh

When the secondary bus is a HyperTransport link, most of the bits defined in the ~~in~~ *PCI-to-PCI Bridge Architecture Specification Revision 1.1* are not relevant and are reserved, being hardwired to 0. The exceptions are listed in the following sections.

~~7.4.4.1~~7.4.5.1 Signaled Target-Abort (Bit 11): R/C: Cold Reset to 0

When set, this bit indicates that the bridge has issued a Target Abort on the secondary bus. (Target Aborts in HyperTransport technology are signaled by a non-NXA error response, as described in section 10.2.1.)

~~7.4.4.2~~7.4.5.2 Received Target-Abort (Bit 12): R/C: Cold Reset to 0

If set, this bit indicates that the bridge has received a Target Abort from the secondary bus. (Target Aborts in HyperTransport technology are signaled by a non-NXA error response, as described in Section 10.2.1.)

~~7.4.4.3~~7.4.5.3 Received Master-Abort (Bit 13): R/C: Cold Reset to 0

When set, this bit indicates that the bridge has received a Master Abort from the secondary bus. (Master Aborts in HyperTransport technology are signaled by an ~~non-~~NXA error response, as described in Section 10.2.1.).

~~7.4.4.4~~7.4.5.4 Detected System Error (Bit 14): R/C: Cold Reset to 0

When set, this bit indicates that the bridge detected sync packet flooding on its secondary bus. See Section 10.2.3.

~~7.4.5~~7.4.6 Memory and Prefetchable Memory Base and Limit Registers: Offsets 20-2Ch: R/W: Warm Reset

Table 29. Memory Base and Limit Register Format

31 … 20	19 … 1	0
Address[31:20]	Reserved	Size

The Size bit exists only in the Prefetchable Memory Base and Limit registers, and is read-only. If 1, it indicates that the Prefetchable Memory Upper registers are implemented. The value of the bit in the base and limit registers must be the same.

Bits 19:0 of the Base addresses are assumed to be 0. Bits 19:0 of the Limit addresses are assumed to be all 1's. To disable decode of a memory space (when there are no devices configured in that space on the secondary bus, for example), the Base register should be programmed to a higher value than the Limit register.

Table 30. Prefetchable Memory Upper Register Format

15 … 0
Address[63:32]

It is strongly recommended that these registers be implemented to allow maximum compatibility with PCI devices.

For accesses coming in on a HyperTransport link, these registers are compared to addresses in the memory-mapped I/O range only, as defined in Chapter 5~~9~~. If the memory space ~~BARs~~ ranges are programmed to support 64-bit addressing, then the incoming addresses from HyperTransport technology must be 0 extended to 64 bits before being compared to the ~~BAR value~~range. Matching addresses are forwarded from the primary to the secondary bus, and ignored on the secondary bus. Non-matching addresses are forwarded from the secondary to the primary bus, and ignored on the primary bus. ~~Because HyperTransport technology supports 40-bit memory space addressing only, a~~Accesses outside the memory-mapped I/O~~40-bit~~ space (FD_0000_0000 and above) on the secondary bus are ignored. HyperTransport technology devices must have windows aligned on 64-byte boundaries so that a maximum-size request will not cross a device boundary. The reset value of these registers is implementation-specific, so firmware must initialize them before setting the Memory Space or Bus Master Enable bits in the Command register.

~~7.4.6~~7.4.7 I/O Base and Limit Registers: Offsets 1C, 1D, 30, and 32h: R/W: Warm Reset

Table 31. I/O Base and Limit Register Format

7 4	3 1	0
Address[15:12]	Reserved	Size

The Size bit of the I/O Base and Limit Registers are read-only and if 1, it indicates the I/O Base and Limit Upper Registers are implemented. The value of the bit in both registers must be the same.

Bits 11:0 of the Base address are assumed to be 0. Bits 11:0 of the Limit address are assumed to be all 1's. To disable decode of the I/O space (when there are no devices configured in I/O space on the secondary bus, for example), the Base register should be programmed to a higher value than the Limit register.

Table 32. I/O Base and Limit Upper Register Format

15 0
Address[31:16]

It is strongly recommended that these registers be implemented to allow maximum compatibility with PCI devices.

For accesses coming in on the primary (HyperTransport) link, these registers are only compared to addresses in the 32-Mbyte I/O range, as defined in Chapter 5~~9~~. Only the low 25 bits of the incoming byte address are used. All bits above bit 24 are forced to 0 before the comparison. Matching addresses are forwarded from the primary to the secondary bus and ignored on the secondary bus. Non-matching addresses are forwarded from the secondary to the primary bus and ignored on the primary bus. Because HyperTransport technology supports only 25-bit I/O space addressing, accesses outside the 25-bit space on the secondary bus are ignored. HyperTransport technology devices must have windows aligned on 64-byte boundaries so that a maximum-size request will not cross a device boundary. The reset value of these registers is implementation-specific, so firmware must initialize them before setting the I/O Space or Bus Master Enable bits in the Command register.

~~7.4.7~~7.4.8 Capabilities Pointer Register: Offset 34h: R/O

Every bridge with a HyperTransport link on at least one port has a capabilities pointer to a linked capabilities list that contains (at least) the capability registers specific to HyperTransport technology.

~~7.4.8~~7.4.9 Bridge Control Register: Offset 3Eh

All unspecified bits are reserved and return 0 if read.

~~7.4.8.1~~7.4.9.1 Parity Error Response Enable (Bit 0)

When the secondary bus is a HyperTransport link, this bit is reserved.

~~7.4.8.2~~7.4.9.2 SERR# Enable (Bit 1): R/W: Warm Reset to 0

This bit controls the mapping of system errors from the secondary to the primary bus of the bridge. If set, and the SERR# Enable in the Command register is set, system errors will propagate. System errors in HyperTransport technology are indicated by flooding the chain with sync packets, as described in Section 10.2.3.

~~7.4.8.3~~7.4.9.3 ISA Enable (Bit 2): R/W: Warm Reset to 0

This bit is implemented similarly to the way it is described in the *PCI-to-PCI Bridge Architecture Specification Revision 1.1*. For HyperTransport technology requests in the bottom 64 Kbytes of I/O space (see Chapter 5~~9~~), this modifies the response to accesses that hit in the range defined by the I/O Base and Limit registers. If this bit is set, transacations addressing bytes 256 to 1023 of each 1Kbyte block on the primary bus are not passed to the secondary bus. Conversely, transactions addressing bytes 256 to 1023 in each 1Kbyte block on the secondary bus will be passed to the primary bus.

Implementation of this bit is ~~optional in non-x86 systems, but recommended in all systems for greater interoperability. If not implemented, it should read as 0~~required for PCI compatibility.

~~It cannot be used with~~If this bit is set when the address mapping extensions described in Appendix A are in use, the address decode behavior of the device may be undefined.

~~7.4.8.4~~7.4.9.4 VGA Enable (Bit 3): R/W: Warm Reset to 0

This bit functions similarly to the way it is described in the *PCI-to-PCI Bridge Architecture Specification Revision 1.1*. If enabled, RdSized and WrSized operations within the address range 0_000A_0000–0_000B_FFFFh (inclusive) or within the first 64 Kbytes of the I/O range (as defined in Chapter 5~~9~~), with address bits 9:0 in the (inclusive) range 3B0–3BBh or 3C0–3DFh, are forwarded from the primary to the secondary interface and are ignored on the secondary interface, overriding the values in the Memory Base and Limit and I/O Base and Limit registers.

Implementation of this bit is ~~optional in non-x86 systems, but recommended in all systems for greater interoperability. If not implemented, it should read as 0~~required for PCI compatibility.

~~It cannot be used with~~If this bit is set when the address mapping extensions described in Appendix A are in use, the address decode behavior of the device may be undefined.

~~7.4.8.5~~7.4.9.5 Master-Abort Mode (Bit 5): R/W: Warm Reset to 0

This bit controls the behavior on the source bus when a transaction forwarded through the bridge receives a Master Abort on the target bus. (Master Aborts in HyperTransport technology are signaled by an NXA error response. See Section 10.2.1.)

When the Master-Abort Mode bit is set, and a nonposted request forwarded from a HyperTransport chain receives a Master Abort on the target bus, the source bus request will receive a Target Abort (an error response with the NXA bit clear). (That is, it will be signaled as an internal error on the originating chain.)

When the Master-Abort Mode bit is clear, the request will appear to complete normally on the source bus. The response will return with the Error bit clear. Writes will receive a TgtDone. Reads will receive a RdResponse, with the appropriate amount of data, which will be all hexadecimal Fs.

~~7.4.8.6~~7.4.9.6 Secondary Bus Reset (Bit 6): R/W: Warm Reset to 0

This bit allows software to reset the secondary bus of the bridge. Writing a 1 to this bit causes the bridge to force the secondary bus into reset. Writing a 0 to this bit cause the bridge to stop forcing the secondary bus into reset. The remaining details are dependent on the type of secondary bus.

This bit is required for bridges that have a HyperTransport link on their primary interface. This bit is also required for host bridges in which HyperTransport technology RESET# is independent of host reset.

If the secondary bus of the bridge is a HyperTransport I/O chain, writing a 1 to this bit will cause the RESET# signal for that chain to be asserted. If the Warm Reset bit in the Host/Secondary Interface Command register (Section 7.5.3.3.1) is clear, the PWROK signal for that chain will also be deasserted. When, after being initially set, the Secondary Bus Reset bit is cleared, the chain will come out of reset. If the Warm Reset bit is set, this simply results in the deassertion of RESET#. It is the responsibility of software to delay deasserting the reset long enough to satisfy the RESET# pulse width requirement. If the Warm Reset bit is clear, clearing the Secondary Bus Reset will cause PWROK to assert. Hardware will then wait for the appropriate amount of time and deassert the RESET# pin. If the programmer wants to be able to determine that the bus has come out of reset, software can poll the Initialization Complete bit of the Link Control register (Section 7.5.4.5).

~~7.4.8.7~~7.4.9.7 Fast Back-to-Back Enable, Primary Discard Timer, Secondary Discard Timer, Discard Timer Status, Discard Timer SERR# Enable (Bits 11:7): R/O

All of these bits are controls for the secondary interface of the bridge. If the secondary interface is a HyperTransport link, these bits are reserved and hardwired to 0.

7.5 Capability Registers

Configuration and status information specific to HyperTransport technology is mapped into configuration space using the capabilities list methodology described in the *PCI Local Bus Specification, Revision 2.2*.

- A device with multiple HyperTransport technology interfaces (e.g., a HyperTransport-to-HyperTransport bridge with HyperTransport links on both the primary and secondary interface) must implement one capability block for each interface. Therefore, a single-link device would have a single primary interface block containing one active Link Control register, and a tunnel device would have a single primary interface block containing two active Link Control registers.
- A bridge would have one primary interface block containing one or two active Link Control registers (depending on whether or not the device provides a tunnel to allow a chain to continue on the primary bus) in addition to one secondary interface block for each HyperTransport technology bridge.
- A HyperTransport technology bridge header and secondary interface capability block are required for each chain.
- Only one primary interface capability block is required for a device that uses HyperTransport technology as its primary interface.
- Primary interface capability blocks always have two Link Control registers. If a link is not implemented, (such as in a single-link device), the Link Control register for that link will be read-only and indicate Link Failure and End Of Chain, as described below.
- If a device is a host bridge or has a different bus for its primary interface, only secondary interface block(s) are required and bridge headers are optional.
- Every function of every device implemented in a HyperTransport node must place a HyperTransport capability list item in its configuration space, indicating the version of the specification to which it is compliant. This can be the primary or secondary interface capability if already present, or the HyperTransport Revision ID capability otherwise.

The layout of the capabilities block is determined by the value in the Capability Type field in the Command register, but the Capability ID register, Capabilities Pointer register, and Capability Type field are always the same. The offset at which the block begins is implementation-specific.

The layout of a Slave/Primary Interface block is shown in Table 33.

Table 33. Slave/Primary Interface Block Format

31 24 23 16	15	8	7 0	
Command	Capabilities Pointer		Capability ID	**+00h**
Link Config 0	Link Control 0			**+04h**
Link Config 1	Link Control 1			**+08h**
LinkFreqCap0	Link Error 0	Link Freq 0	Revision ID	**+0Ch**
LinkFreqCap1	Link Error 1	Link Freq 1	Feature	**+10h**
Error Handling	Enumeration Scratchpad			**+14h**
Reserved	Mem Limit Upper		Mem Base Upper	**+18h**

***Note:** Shaded registers contain minimum-required read-write bits. Other registers are read-only or contain only device-dependent bits.*

The layout of a Host/Secondary Interface block is shown in Table 34.

Table 34. Host/Secondary Interface Block Format

31 24 23 16	15	8	7 0	
Command	Capabilities Pointer		Capability ID	**+00h**
Link Config	Link Control			**+04h**
LinkFreqCap	Link Error	Link Freq	Revision ID	**+08h**
Reserved	Feature			**+0Ch**
Error Handling	Enumeration Scratchpad			**+10h**
Reserved	Mem Limit Upper		Mem Base Upper	**+14h**

***Note:** Shaded registers contain minimum-required read-write bits. Other registers are read-only or contain only device-dependent bits.*

Unless otherwise noted, the registers described in this section are reset by the reset mechanism of the primary bus and not that of the secondary bus. For a HyperTransport-to-HyperTransport bridge, the reset mechanism is the HyperTransport technology RESET# signal of the primary chain. For a Host-to-HyperTransport bridge, the reset mechanism is a host-specific reset signal. Implementations that do not require the ability to reset the HyperTransport chain independently of the host may choose to combine the two.

Registers marked "Chain Reset" are reset with which the chain they are associated. For host interface blocks, this is HyperTransport technology RESET#, not host reset. For device primary interface blocks, this is HyperTransport technology RESET# on the primary chain. For device secondary interface blocks, this is HyperTransport technology RESET# on the bridge's secondary chain.

Each field is defined as readable and writeable by software (R/W), readable only (R/O), readable and settable by writing a 1 (R/S), or readable and cleared by writing a 1 (R/C). Additionally each field is affected by only cold reset or by both cold and warm reset.

7.5.1 Capability ID: Offset 00h: R/O

The capability ID for HyperTransport technology is 08h.

7.5.2 Capabilities Pointer: Offset 01h: R/O

This register contains a pointer to the next capability in the list, or a value of 00h if this is the last one.

7.5.3 Command Register: Offset 02h

The Command register contains bits used to configure the HyperTransport interface, as shown in Table 35. All unspecified bits are reserved and are hardwired to 0.

Table 35. Command Register Format

Type	15 13	12	11	10	9	8	7	6	5	4	3	2	1	0
Slave/Pri	000	Drop on Uninit	Default Direction	Master Host	Unit Count					Base UnitID				
Host/Sec	001	Drop on Uninit	Inbound EOC Err	Act as Slave	Rsv	Host Hide	Chain Side	Device Number					Double Ended	Warm Reset

Note: Any write to the slave command register may affect the Master Host bit, as described below.

7.5.3.1 Capability Type (Bits 15:13): R/O

This field indicates the type of information present in this capability block. For the primary and secondary interface capability blocks, there are 3 bits used in the encoding. For all other HyperTransport capability blocks, 5 bits are used. Currently, these encodings are defined as shown in Table 36.

Table 36. Capability Type Encoding

Encoding	Capability Type
000xx	Slave or Primary Interface
001xx	Host or Secondary Interface
01000	Reserved (Switch)
01001-01111	Reserved
10000	Interrupt Discovery and Configuration
10001	Revision ID
10010-10011	Reserved
10100	Address Mapping
10101-10111	Reserved
11000	Reserved (Retry Mode)
11001-11111	Reserved

Primary and secondary interface encodings indicate that this block is used to configure the primary (including the interface of a HyperTransport technology slave device) or secondary (including the interface of a host bridge) interface of a device, respectively.

The layout of the rest of the bits of the Command register depends on the Capability Type field, as described in the following sections.

7.5.3.2 Slave/Primary Interface Command Bits

7.5.3.2.1 Base UnitID (Bits 4:0): R/W: Warm Reset to 0

This field contains the lowest numbered UnitID belonging to this device. If the device owns multiple UnitIDs, the additional ones occupy the next consecutive UnitID values above the base. The contents of this field are used to generate the UnitID field in request and response packets issued by this device, to identify responses returning to this device, and to identify configuration requests directed to this device.

7.5.3.2.2 Unit Count (Bits 9:5): R/O

This field contains the number of UnitIDs that this device requires. Therefore, the highest UnitID used by this device is given by (BaseUnitID + UnitCount – 1). If the highest UnitID used exceeds 1Fh, the behavior of the device is undefined.

7.5.3.2.3 Master Host (Bit 10): R/W: Warm Reset to 0

This bit indicates which link is the path to the master (or only) host bridge of the HyperTransport chain. It is readable from software, but not directly writeable. Any time the Command register is written, this bit is loaded with the link number from which the write came. For a device with only one link interface, this bit may be hardwired to point to the implemented link.

7.5.3.2.4 Default Direction (Bit 11): R/W: Warm Reset to 0

This bit determines the default direction for a HyperTransport technology device to send requests. A 0 indicates requests should be sent toward the master host bridge, as indicated by the Master Host bit. A 1 indicates requests should be sent in the opposite direction. For a device with only one link interface, this bit has no meaning and should be hardwired to 0. An intelligent device can choose to ignore this if it bases the direction of DMA upon some other criteria.

7.5.3.2.5 Drop on Uninitialized Link (Bit 12): R/W: Cold Reset to 0

This bit determines what will happen to packets issued by a device or forwarded from a receiving link interface to a transmitting interface whose Initialization Complete and End of Chain bits are clear. If both are deasserted for a given link, packets to be transmitted on that link will be stalled until either Initialization Complete sets (in which case they will be transmitted) or End Of Chain sets (in which case they will be treated as End Of Chain packets; see Section 10.1.5). In the case where hardware is broken, it is possible that neither of these events occurs, in which case the packet can hang. If Drop on Uninitialized Link is set, a transmitter with its Initialization Complete bit clear will always act as if the End of Chain bit were set. Hosts that use the initialization sequence described in Section 12.3 are encouraged to implement a timeout counter to prevent a system-wide initialization error due to link-level initialization problems on a non-default chain. Packet forwarding behavior is described in Table 37.

Table 37. Packet Forwarding Behavior

End of Chain	**Initialization Complete**	**Drop on Uninitialized Link**	***Action***
1	-	-	*Reject*
0	1	-	*Forward*
0	0	1	*Reject*
0	0	0	*Stall*

7.5.3.3 Host/Secondary Interface Command Bits

7.5.3.3.1 Warm Reset (Bit 0): R/W: Warm Reset to 1

This optional bit allows a reset sequence initiated by the Secondary Bus Reset bit of the Bridge Control register (see Section 7.4.9.6) to be either warm or cold. The contents of this bit have an effect only when software initiates a reset sequence. If it is 0, PWROK will be driven low as part of the sequence, causing a cold reset. It is the responsibility of the hardware to sequence PWROK and RESET# correctly. If not implemented, this bit is read-only and hardwired to 1. Changing the state of the Warm Reset bit while the Secondary Bus Reset bit is asserted results in undefined behavior.

7.5.3.3.2 Double-Ended (Bit 1): R/W: Warm Reset to 0

This bit indicates that there is another bridge at the far end of the HyperTransport chain. For bridges that do not support double-ended chains, this bit must be hardwired to 0. This bit controls no hardware. It exists as a scratchpad for software during link configuration.

7.5.3.3.3 Device Number (Bits 6:2): R/W: Cold Reset to 0

This optional register contains the device number of configuration accesses that the host bridge responds to when accessed from the chain attached to the host interface. While this will typically be 0, in some cases there may be legacy software (Section 7.2.2) or ordering (Section 4.1.1) considerations that require host configuration space registers to be located somewhere other than Device 0. If not implemented, this register is read-only and hardwired to 0. When accessed from the host side, the bridge may locate its configuration space registers at a different location, so that when enumerated in a double-hosted chain, one host will not relocate the register space of another This value in this register is used as the UnitID of hosts with the Act as Slave bit set (defined in Section 7.5.3.3.6). If the Act as Slave bit is implemented, this register must be implemented.

7.5.3.3.4 Chain Side (Bit 7): R/O

This bit indicates which side of the host bridge is being accessed. A 0 indicates that the read is coming from within the host. A 1 indicates that the read is coming from the chain attached to the host interface. In a host that does not support double-hosted chains, this bit is always 0, because there cannot be an access from the chain.

7.5.3.3.5 Host Hide (Bit 8): R/W: Warm Reset to 0

This bit, when set, causes the host's configuration space to be inaccessible from the chain attached to the host interface (any accesses to configuration space are treated as if they have reached the end of chain). When clear, the host should respond to configuration cycles from the chain. The determination of the register set to be presented to the chain is implementation-specific, although a configuration header including a host capability block must be presented for chain enumeration purposes. In a host that does not support double-hosted chains, this bit is read-only 1, because the host is never accessible from the chain.

7.5.3.3.6 Act as Slave (Bit 10): R/~~C~~W: Cold Reset to 0

This optional bit, when set, causes a host to act as a slave, using the device number defined in Section 7.5.3.3.3 as the UnitID for requests and responses that it originates, ~~and~~ it does not set the Bridge bit on responses it generates, and follows the routing rules for ordinary nodes defined in Section 4.9. When clear, a host acts normally (uses UnitID 0 and sets the Bridge bit on responses). If a host does not support sharing double-hosted chains, this bit may be ~~reserved~~read-only 0. This bit takes effect only after a warm reset.

7.5.3.3.7 Host Inbound End of Chain Error (Bit 11): R/C: Cold Reset to 0

This bit indicates that a packet received from a far host has taken an end-of-chain error. (See Section 10.1.5.) This bit is hardwired to 0 if the device does not check for this error condition.

7.5.3.3.8 Drop on Uninitialized Link (Bit 12): R/W: Cold Reset to 0

This bit is defined in the same way as bit 12 of the Slave/Primary Interface Command register.

7.5.4 Link Control Register: Offsets 04h and 08h

Host/secondary interface blocks implement one copy of the Link Control register (defined in Table 38). Slave/primary interface blocks implement two copies of this register, one for each link. For devices that implement only one link in the chain, all bits of the second control register are reserved and hardwired to 0, except for the Link Failure, End of Chain, and Transmitter Off fields, which are hardwired to 1.

Table 38. Link Control Register

15	14	13	12	11 8	7	6	5	4	3	2	1	0
Rsv	ExtCTL	LSEn	IsocEn	CRC Error	TXO	EOC	Init	LkFail	CFE	CST	CFIE	Rsv

7.5.4.1 CRC Flood Enable (Bit 1): R/W: Warm Reset to 0

~~When clear, this bit prevents CRC errors from generating sync packets and causing the system to come down, and from setting the LinkFail bit. However, CRC checking logic still runs on all lanes enabled by LinkWidthIn, and detected errors still set the CRC Error bits.~~ When asserted, sync flooding will be initiated and the LinkFail bit will be set whenever any of the CRC Error bits for this link are asserted. CRC checking logic runs on all lanes enabled by LinkWidthIn, and detected errors still set the CRC Error bits, regardless of the state of this bit. Note that this bit is not reset by secondary chain reset, so it will not be reset in a bridge when that bridge takes a sync flooded link through warm reset, and may cause sync flooding to be immediately restarted after the reset sequence is complete. In that case, software should clear this bit before taking the chain out of reset.

7.5.4.2 CRC Start Test (Bit 2): R/W: Warm Reset to 0

When this optional bit is written to a 1 by software, the hardware initiates a CRC test sequence on the link, as described in Appendix G. When the test sequence is complete, and CRC has been checked on all CRC intervals containing test pattern data, hardware clears the bit. Software can determine that the test has completed by reading the bit and checking the status of the CRC Error bits. Implementation of CRC test pattern generation is optional. If not implemented, this bit must be hardwired to 0. Software should not set this bit unless it has checked the CRC Test Mode Capability bit, as defined in Section 7.5.10.3, of the device on the other side of the link.

7.5.4.3 CRC Force Error (Bit 3): R/W: Warm Reset to 0

When this bit is set, bad CRC is generated on all transmitting lanes, as enabled by LinkWidthOut. The covered data is not affected.

7.5.4.4 Link Failure (Bit 4): R/C: Evaluated at Cold Reset~~ to 0~~

The LinkFail bit is set if a failure has been detected on the link. Devices with only one HyperTransport link will hardwire LinkFail for the nonexistent link to 1. Devices that contain an interface that is not used by the system will also set LinkFail for the unused link to 1. As described in Section 12.2, devices can identify unused links because their CAD[0] input is tied to a logical 0.

The LinkFail bit is set by hardware in the event of a link error that results in a sync flood, such as a CRC, protocol, or overflow error (See Section 10.2.4). It is not set by a device that is merely forwarding sync packets, only by a device that originates them.

7.5.4.5 Initialization Complete (Bit 5): R/O: Warm Chain Reset to 0

This read-only bit is reset to 0 and set by hardware when the low-level link initialization sequence (see Section 12.2) is successfully complete in both the receiver and transmitter. If there is no device on the other end of the link, or if that device is unable to properly perform the low-level link initialization protocol, the bit never gets set. A device may receive a request before initialization of all the attached links is complete. If the request needs to be forwarded to an uninitialized link, the disposition of the request is determined by the Drop On Uninitialized Link bit (see Section 7.5.3.2.5).

7.5.4.6 End of Chain (Bit 6): R/S: Evaluated at Warm Chain Reset~~ to 0~~

The End of Chain bit is set to indicate that the given link is not part of the logical HyperTransport chain. Packets that are issued or forwarded to this link are either dropped or result in an NXA error response, as appropriate (see Section 4.9.2). Packets received from this link are ignored, CRC is not checked, and sync flooding from this link is ignored and not propagated. If the transmitter is still enabled (Transmitter Off CSR bit is clear) when the End of Chain bit is set, the transmitter must drive NOP packets (all CAD bits 0, with CTL asserted) with good CRC. This is required to prevent the far receiver from seeing garbage when we are no longer sending to it. It is the responsibility of software to make sure that no traffic is going across the link when End of Chain is set, so that the switch to NOPs does not occur in the midst of a packet.

Slave devices with only one HyperTransport link will hardwire End of Chain for the nonexistent link to 1. Devices that contain an interface that is not used by the system will also set End of Chain for the unused link to 1 at the rising edge of reset. (As described in Section 12.2, devices can identify unused links because their CAD[0] input is tied to a logical 0.) End of Chain can be set by software by writing a 1 to indicate the logical end of the chain, or by partitioning a double-hosted chain into two independent logical chains. This bit cannot be cleared by software. A write of 0 to this bit position has no effect.

7.5.4.7 Transmitter Off (Bit 7): R/S: Warm Chain Reset to 0

This bit provides a mechanism to shut off a link transmitter for power savings or EMI reduction. When set, no output signals on the link toggle and are driven to electrical levels that satisfy the DC specification. This bit resets to 0 and can be set by software writing a 1 to the bit. This bit cannot be cleared by software. A write of 0 to this bit position has no effect. If End of Chain is set on an active link, the Transmitter Off bit should not be set until the transmitter has driven enough NOPs to fill the receiver's receive FIFOs.

When this bit is set, the link receiver should also be disabled if necessary to prevent DC current paths as a result of the inputs that may be invalid or floating.

7.5.4.8 CRC Error (Bits 11:8): R/C: Cold Reset to 0

These bits are set by hardware when a CRC error is detected on an incoming link. Errors are detected and reported on a per byte-lane basis where bit 8 corresponds to the least-significant byte lane. Four bits are required to cover the maximum HyperTransport technology width of 32 bits. Error bits for unimplemented (as specified by Max Link Width In, Section 7.5.5.1) or unused (as specified by Link Width In, Section 7.5.5.5) byte lanes return 0 when read.

7.5.4.9 Isochronous Flow Control Enable (Bit 12): R/W: Cold Chain Reset to 0

This optional bit controls whether Isochronous flow control, as described in Appendix D.1, is enabled for this link. The bit is set to enable Isochronous flow control mode and cleared to disable Isochronous flow control mode. Note that the Isoc bit in requests and responses is used regardless of this setting. Only Isoc flow control packets are prevented by clearing it. This bit only takes effect after a warm reset. This bit is reserved if the Isochronous flow control mode capability bit is cleared. (See Section 7.5.10.1) It is the responsibility of system-sizing software to ensure that this bit is set to the same value on both sides of the link and only set if both sides of the link have Isochronous flow control mode capability.

When a device forwards Isochronous traffic from a link that has Isochronous Flow Control enabled to a link that does not, the packet is unmodified but must be forwarded in the normal virtual channel set. Conversely, when traffic is forwarded from a link that does not have Isochronous Flow Control enabled to one that does, the packets are unmodified but are forwarded in the Isochronous virtual channel set.

7.5.4.10 LDTSTOP# Tristate Enable (Bit 13): R/W: Cold Reset to 0

This bit controls whether the transmitter tristates the link during the disconnected state of an LDTSTOP# sequence, as described in Section 8.3. When the bit is set, the transmitter tristates the link. When the bit is clear, the transmitter continues to drive the link. This bit is reserved if the

LDTSTOP# capability bit is cleared. (See Section 7.5.10.2) The behavior of the link transmitter and receiver in both the tristate and driven cases is described in Table 39.

Table 39. LDTSTOP# Tristate Enable Bit Encoding

LDTSTOP# Tristate Enable	Link State in LDTSTOP# Disconnect State	Transmitter Behavior	Receiver Behavior
0	Driven	CAD and CTL logically undefined, but driven to electrical levels that satisfy DC specification. CLK running.	Ignores CAD and CTL logical values.
1	Tristate	CAD, CTL and CLK placed in high impedance state.	Disables DC current paths that could be created as a result of CAD, CTL and CLK inputs being tristated and ignores logical values.

7.5.4.11 Extended CTL Time (Bit 14): R/W: Cold Reset to 0

If this bit is set, during the link initialization sequence in Section 12.2 following an LDTSTOP# disconnect sequence, CTL will be asserted for 50 us after the point where both the transmitting device has asserted CTL and it has sampled CTL asserted from the other side of the link. If this bit is clear, CTL need only be asserted at least 16 bit-times after both sides assert CTL in 8-bit or larger links. (32 bit-times for 4-bit links, 64 bit-times for 2-bit links) Software must set this bit if the device on the other side of the link has its Extended CTL Time Required bit (Section 7.5.10.4) set. The extension allows devices using DLLs in their receivers enough time to lock to the transmit clock. This is necessary after LDTSTOP# because transmit clocks are only required to be stable at the time when CTL is asserted.

7.5.5 Link Configuration Register: Offsets 06h and 0Ah

As with the Link Control register, there may be either one or two copies of the Link Configuration register, one for each link. If only one link is implemented by the device, the second register is reserved. All unspecified bits are reserved. The Link Configuration register is defined in Table 40.

As described in the following subsections, software updates to the upper half of this register take effect after a warm reset sequence and, depending on the field, also after an LDTSTOP# disconnect sequence.

Table 40. Link Configuration Register Definition

15	14 12	11	10 8	7	6 4	3	2 0
Dw Fc Out En	LinkWidthOut	Dw Fc In En	LinkWidthIn	Dw Dc Out	MaxLinkWidthOut	Dw Fc In	MaxLinkWidthIn

7.5.5.1 Max Link Width In (Bits 2:0): R/O

This field contains three bits that indicate the physical width of the incoming side of the HyperTransport link implemented by this device.

The encodings are as shown in Table 41

Table 41. Max Link Width In Bit Field Encoding

LinkWidth[2:0]	Width
000	8 bits
001	16 bits
010	Reserved
011	32 bits
100	2 bits
101	4 bits
110	Reserved
111	Link physically not connected

7.5.5.2 Doubleword Flow Control In (DwFcIn, Bit 3): R/O

This bit is set to indicate that this receiver is capable of doubleword-based data buffer flow control.

7.5.5.3 Max Link Width Out (Bits 6:4): R/O

This field contains three bits that indicate the physical width of the outgoing side of the HyperTransport link implemented by this device. It uses the same encodings as the MaxLinkWidthIn field.

7.5.5.4 Doubleword Flow Control Out (DwFcOut, Bit 7): R/O

This bit is set to indicate that this transmitter is capable of doubleword-based data buffer flow control.

7.5.5.5 Link Width In (Bits 10:8): R/W: Cold Chain Reset

This field controls the utilized width (which may not exceed the physical width) of the incoming side of the links of the HyperTransport link implemented by this device. It uses the same encoding as the MaxLinkWidthIn field. After cold reset, this field is initialized by hardware based on the results of the link-width negotiation sequence described in Section 12.2. This sequence also identifies physically unconnected links. Based on sizing the devices at both ends of the link, software can then write a different value into the register. The chain must pass through warm reset or an LDTSTOP# disconnect sequence for the new width values to be reflected on the link.

The LinkWidthIn CSR in the link receiver must match the LinkWidthOut CSR in the link transmitter of the device on the other side of the link. The LinkWidthIn and LinkWidthOut registers within the same device are not required to have matching values. If two sides of a link are programmed to different widths when a RESET# or LDTSTOP# assertion occurs, the link will not be able to complete the initialization sequence. The system design must ensure that RESET# or LDTSTOP# will not be asserted while software is writing new link width values. The means to ensure this is system-specific and beyond the scope of this specification.

7.5.5.6 Doubleword Flow Control In Enable (DwFcInEn, Bit 11): R/W: Cold Chain Reset to 0

This optional bit may be set to program the receiver into doubleword-based flow control mode. After checking that devices on both sides of a link support this mode (by reading the bits defined in sections 7.5.5.2 and 7.5.5.4), software may set this bit and/or the Doubleword Flow Control Out Enable bit. The chain must pass through warm reset for the new flow control method to be used on the link. See appendix H for more details about this mode.

7.5.5.7 Link Width Out (Bits 14:12): R/W: Cold Chain Reset

This field is similar to the LinkWidthIn field, except that it controls the utilized width of the outgoing side of the links implemented by this device. Like LinkWidthIn, this field is initialized after cold reset by hardware based on the results of the link width negotiation sequence described in Section 12.2. Byte lanes that are disabled due to the LinkWidthOut value being set narrower than the physically implemented width of the link will have their transmitters shut down in the same way as if Transmitter Off was set.

7.5.5.8 Doubleword Flow Control Out Enable (DwFcOutEn, Bit 15): R/W: Cold Chain Reset to 0

This optional bit is similar to DwFcInEn, except that it puts the transmitter into doubleword-based flow control mode.

7.5.6 Revision ID Register: Offset 08h or 0Ch: R/O

The Revision ID register is defined as shown in Table 42.

Table 42. Revision ID Register Definition

7 5	4 0
MajorRev	MinorRev

7.5.6.1 Minor Revision (Bits 4:0)

This field contains the minor revision of the *HyperTransport™ I/O Link Protocol Specification* to which the particular implementation conforms.

7.5.6.2 MajorRevision (Bits 7:5)

This field contains the major revision of the *HyperTransport™ I/O Link Protocol Specification* to which the particular implementation conforms.

7.5.7 Link Frequency Register: Offsets 09h or 0Dh and 11h (Bits 3:0): R/W: Cold Reset to 0

As with the Link Control and Link Configuration registers, there may be either one or two copies of the Link Frequency register, one for each link. If the device only implements one link, the second register is reserved.

The Link Frequency register specifies the operating frequency of the link's transmitter clock—the data rate is twice this value. The encoding of this field is shown in Table 43.

Table 43. Link Frequency Bit Field Encoding

Link Frequency Encoding	Transmitter Clock Frequency (MHz)
0000	200 (default)
0001	300
0010	400
0011	500
0100	600
0101	800
0110	1000*
0111 to 1110	Reserved
1111	Vendor-Specific
Note: *Electrical requirements of the link above 800MHz have not been fully specified at this time.*	

Software can write a nonzero value to this register, and that value will take effect as a result of either a warm reset or LDTSTOP# disconnect sequence on the associated chain. For host interface blocks, the change is effected by HyperTransport technology RESET#, not host reset. For secondary interface blocks, the change is effected by HyperTransport technology RESET# on the bridge's secondary chain. If two sides of a link are programmed to different frequencies when a RESET# or LDTSTOP# assertion occurs, the link may not be able to complete the initialization sequence. The system design must ensure that RESET# or LDTSTOP# will not be asserted while software is writing new link frequency values. The means to ensure this is system-specific and beyond the scope of this specification.

See Section 11.1 for a definition of the HyperTransport technology clocking modes, and for how the Link Frequency register controls the HyperTransport technology transmitter frequency in each mode. In asynchronous and pseudo-synchronous modes, the Link Frequency register specifies the maximum operating frequency. In synchronous mode, both the receiver and transmitter operate at the programmed frequency.

HyperTransport technology devices are not required to support all the transmitter clock frequencies in Table 43. All HyperTransport technology devices must support a 200-MHz synchronous link.

7.5.8 Link Error Register: Offsets 09h or 0Dh and 11h (Bits 7:4)

The Link Error register (defined in Table 44) occupies bits 7:4 of the byte containing the Link Frequency register. It contains error control and log bits for the link. Devices that do not check for one or more error conditions should hardwire the log bits for those conditions to 0.

Table 44. Link Error Register Definition

7	6	5	4
CTL Timeout	End of Chain Error	Overflow Error	Protocol Error

7.5.8.1 Protocol Error (Bit 4): R/C: Cold Reset to 0

This bit indicates a protocol error has been detected on the link. See Section 10.1.3.

7.5.8.2 Overflow Error (Bit 5): R/C: Cold Reset to 0

This bit indicates a receive buffer overflow error has been detected on the link. See Section 10.1.4.

7.5.8.3 End Of Chain Error (Bit 6): R/C: Cold Reset to 0

This bit indicates that a posted request or response packet has been given to this transmitter to be issued when this link is the end of chain. See Section 10.1.5.

7.5.8.4 CTL Timeout (Bit 7): R/W: Warm Reset to 0

This bit indicates how long CTL may be low before a device indicates a protocol error. A 0 in this bit indicates 1 millisecond; a 1 indicates 1 full second. See Section 10.1.3. This bit is optional for devices that do not detect protocol errors.

7.5.9 Link Frequency Capability Register: Offsets 0Ah or 0Eh and 12h: R/O

The Link Frequency Capability register (LinkFreqCap) is a 16-bit read only register that indicates the clock frequency capabilities of the associated link. Each bit in LinkFreqCap corresponds to one of the 16 possible encodings of the Link Frequency register as defined in Section 7.5.7. Bit N of LinkFreqCap corresponds to encoding N of the LinkFreq field. A 1 in LinkFreqCap means that the link supports the corresponding link frequency, and a 0 means the link does not support that frequency. Bit [0] of LinkFreqCap must be 1, since all links are required to support 200-MHz operation. A 1 in bit [15] indicates that vendor-specific frequencies are available, the use and support of which are beyond the scope of this specification.

The read-only value in LinkFreqCap specifies the frequency capabilities of the link independent of other practical constraints. For example, a specific device with multiple HyperTransport links may require all the links to run at the same frequency, or the system's electrical parameters may impose frequency restrictions on a specific link's operation that are not reflected in the Link FreqCap value. System firmware must deal with these system-specific requirements.

7.5.10 Feature Capability Register: Offset 0Ch or 10h: R/O

This register contains bits to indicate which optional features are supported by this device. All unspecified bits are reserved.

7.5.10.1 Isochronous Flow Control Mode (Bit 0)

This bit is set to indicate that the device is capable of supporting Isochronous ~~operation~~ flow control as defined in Appendix D.1, and clear to indicate that the device is not. Isochronous ~~operation~~ flow control is enabled by bit 12 of the Link Control register~~, bit 12~~.

7.5.10.2 LDTSTOP# (Bit 1)

This bit is set to indicate that the associated interface supports the LDTSTOP# protocol, as described in Section 8.3, and clear to indicate that it does not.

7.5.10.3 CRC Test Mode (Bit 2)

This bit is set to indicate that the associated interface supports the CRC Testing Mode, as described in Appendix G, and clear to indicate that it does not.

7.5.10.4 Extended CTL Time Required (Bit 3)

This bit is set to indicate that this device requires CTL to be asserted for 50 us during the initialization sequence specified in Section 12.2 after an LDTSTOP# disconnect.

7.5.10.5 Extended Register Set (Bit 8)

This bit is set to indicate that the associated interface includes the Enumeration Scratchpad, Error Handling, and Memory Base/Limit Upper registers. If this bit is 0, software should not attempt to access these registers, since they may have alternative functions.

This bit exists in host/secondary interface blocks only, because the extended register set represented by this bit is required in all non-host devices, bridges, and switches~~, and~~. The extended register set is strongly recommended for hosts as well, but optional due to the configuration space constraints of some host implementations.

~~This bit exists in host/secondary interface blocks only.~~

7.5.11 Enumeration Scratchpad Register: Offset 10h or 14h: R/W: Cold Reset to 0

This register provides a scratchpad for enumeration software.

7.5.12 Error Handling Register: Offset 12h or 16h

The Error Handling register (defined in Table 45) contains routing enables from the various error log bits to the various error reporting mechanisms, as well as the Chain Fail and Response Error status bits. For definitions of the reporting mechanisms, see Section 10.2. Devices that do not check for one or more error conditions should hardwire the log and enable bits for those conditions to 0.

Table 45. Error Handling Register Definition

Byte	7	6	5	4	3	2	1	0
0	SERR Fatal Enable	CRC Fatal Enable	Resp Fatal Enable	EOC Fatal Enable	Overflow Fatal Enable	Prot Fatal Enable	Overflow Flood Enable	Prot Flood Enable
1	SERR NonFatal Enable	CRC NonFatal Enable	Resp NonFatal Enable	EOC NonFatal Enable	Overflow NonFatal Enable	Prot NonFatal Enable	Response Error	Chain Fail

7.5.12.1 Protocol Error Flood Enable (Bit 0): R/W: Warm Reset to 0

When asserted, this bit will cause the link to be flooded with Sync packets whenever the Protocol Error bit is asserted in (one of) the Link Error register(s). As with the CRC Flood Enable, this bit is not cleared by secondary chain reset in a bridge, so it may need to be cleared during a secondary bus reset or sync flooding could recur after reset.

7.5.12.2 Overflow Error Flood Enable (Bit 1): R/W: Warm Reset to 0

When asserted, this bit will cause the link to be flooded with Sync packets whenever the Overflow Error bit is asserted in (one of) the Link Error register(s). As with the CRC Flood Enable, this bit is not cleared by secondary chain reset in a bridge, so it may need to be cleared during a secondary bus reset or sync flooding could recur after reset.

7.5.12.3 Protocol Error Fatal Enable (Bit 2): R/W: Warm Reset to 0

When asserted, this bit will cause the fatal error interrupt to be asserted whenever the Protocol Error bit is asserted in (one of) the Link Error register(s). If the fatal error interrupt is not implemented, this bit is hardwired to 0.

7.5.12.4 Overflow Error Fatal Enable (Bit 3): R/W: Warm Reset to 0

When asserted, this bit will cause the fatal error interrupt to be asserted whenever the Overflow Error bit is asserted in (one of) the Link Error register(s). If the fatal error interrupt is not implemented, this bit is hardwired to 0.

7.5.12.5 End of Chain Error Fatal Enable (Bit 4): R/W: Warm Reset to 0

When asserted, this bit will cause the fatal error interrupt to be asserted whenever the End of Chain Error bit is asserted in (one of) the Link Error register(s), or the Inbound End of Chain Error bit is set in the Host Command register. If the fatal error interrupt is not implemented, this bit is hardwired to 0.

7.5.12.6 Response Error Fatal Enable (Bit 5): R/W: Warm Reset to 0

When asserted, this bit will cause the fatal error interrupt whenever the Response Error bit (9) is asserted. If the fatal error interrupt is not implemented, this bit is hardwired to 0.

7.5.12.7 CRC Error Fatal Enable (Bit 6): R/W: Warm Reset to 0

When asserted, this bit will cause the fatal error interrupt whenever any of the CRC Error bits are asserted in (either of) the Link Control register(s). If the fatal error interrupt is not implemented, this bit is hardwired to 0.

7.5.12.8 System Error Fatal Enable (Bit 7): R/W: Warm Reset to 0

This bit is implemented for host interfaces only. For slave interfaces, it is hardwired to 0.

When asserted in a host, this bit will cause the fatal error interrupt whenever the System Error Detected bit is asserted in the Secondary Status register. If the fatal error interrupt is not implemented, this bit is hardwired to 0.

7.5.12.9 Chain Fail (Bit 8): R/O: Warm Chain Reset to 0

This bit indicates that the chain has gone down. It is set whenever a device detects sync flooding or a sync-flood generating error. It is cleared by reset of the failed chain.

7.5.12.10 Response Error (Bit 9): R/C: Cold Reset to 0

This bit indicates that the given interface has received a response error. (See Section 10.1.7.)

7.5.12.11 Protocol Error Nonfatal Enable (Bit 10): R/W: Warm Reset to 0

When asserted, this bit will cause the nonfatal error interrupt to be asserted whenever the Protocol Error bit is asserted in (one of) the Link Error register(s). If the nonfatal error interrupt is not implemented, this bit is hardwired to 0.

7.5.12.12 Overflow Error Nonfatal Enable (Bit 11): R/W: Warm Reset to 0

When asserted, this bit will cause the nonfatal error interrupt to be asserted whenever the Overflow Error bit is asserted in (one of) the Link Error register(s). If the nonfatal error interrupt is not implemented, this bit is hardwired to 0.

7.5.12.13 End of Chain Error Nonfatal Enable (Bit 12): R/W: Warm Reset to 0

When asserted, this bit will cause the nonfatal error interrupt to be asserted whenever the End of Chain Error bit is asserted in (one of) the Link Error register(s), or the Inbound End of Chain Error bit is set in the Host Command register. If the nonfatal error interrupt is not implemented, this bit is hardwired to 0.

7.5.12.14 Response Error Nonfatal Enable (Bit 13): R/W: Warm Reset to 0

If asserted, this bit will cause the nonfatal error interrupt whenever the Response Error bit (9) is asserted. If the nonfatal error interrupt is not implemented, this bit is hardwired to 0.

7.5.12.15 CRC Error Nonfatal Enable (Bit 14): R/W: Warm Reset to 0

When asserted, this bit will cause the nonfatal error interrupt whenever any of the CRC Error bits are asserted in (either of) the Link Control register(s). If the nonfatal error interrupt is not implemented, this bit is hardwired to 0.

7.5.12.16 System Error Nonfatal Enable (Bit 15): R/W: Warm Reset to 0

This bit is implemented for host interfaces only. For slave interfaces, it is hardwired to 0.

When asserted in a host, this bit will cause the nonfatal error interrupt whenever the System Error Detected bit is asserted in the Secondary Status register. If the nonfatal error interrupt is not implemented, this bit is hardwired to 0.

7.5.13 Memory Base Upper 8 Bits: Offset 14h or 18h: R/W: Warm Reset to 0

This register extends the Nonprefetchable Memory Base register, defined for bridges in Section 7.4.6, to 40 bits. Nodes that implement multiple bridge headers sharing a single HyperTransport primary interface use the base and limit extension values in the primary capability block for all bridges. Bridges containing HyperTransport secondary capability blocks use the values in that block, unless their primary interface is also a HyperTransport link, in which case they use the values from the appropriate primary capability block. In that case, the values from the secondary capability block are unused. Software can determine which devices are affected by this register by reading the BaseUnitID and UnitCount fields of the HyperTransport Command register in the primary interface.

7.5.14 Memory Limit Upper 8 Bits: Offset 15h or 19h: R/W: Warm Reset to 0

This register extends the Nonprefetchable Memory Limit register, defined for bridges in Section 7.4.6, to 40 bits and has the same requirements as the Base extension register defined in Section 7.5.13.

7.6 Interrupt Discovery and Configuration Capability Block

As shown in Table 46, the Interrupt Discovery and Configuration Capability block defines the mechanism to declare how many interrupt sources each HyperTransport technology function can generate and allows software to configure each interrupt independently. Each function can have its own capability block, facilitating a mapping of interrupts to functions. Existing software may not be able to use this mechanism, so an additional, alternative means of configuring interrupts (described in Appendix F.1.4) may be required for compatibility. Interrupts are described in Chapter 9~~5~~.

Table 46. Interrupt Discovery and Configuration Capability Block Definition

31 24	23 16	15 8	7 0	
Capability Type	Index	Capabilities Pointer	Capability ID	+00h
Dataport				+04h

- *Capability ID* is read-only 08h to indicate that this is a HyperTransport technology capability list item.
- *Capabilities Pointer* is a read-only register pointing to the next item in the capability list, or 00h if this is the last item.
- *Interrupt Register Index* selects the interrupt definition register to be accessed through the dataport defined below.
- *Capability Type* is read-only 80h to indicate that this is an interrupt discovery and configuration block.

- *Interrupt Register Dataport* provides read or write access to the interrupt definition register selected by the index defined above. This port should only be accessed as a whole doubleword. All indexes not listed are reserved and should read 0s.

7.6.1 Last Interrupt: Index 01h: R/O

Bits 23:16 contain the last interrupt message defined by this device. Interrupt definitions are numbered beginning with 0, so a device that defines four interrupts would have the value 3 in this field.

7.6.2 Interrupt Definition Registers: Index 10h and Higher: Warm Reset

Each interrupt message defined by a device has a 64-bit definition register, consuming two indexes. Interrupt 0 would occupy indexes 10 and 11h, interrupt 1 uses 12 and 13h, etc. Bits 31:0 are accessed through the lower (even) index and bits 63:32 are accessed through the high (odd) index. These bits are defined in Table 47.

Table 47. Interrupt Definition Registers

Bit	R/W	Reset	Description
63	R/C	0	Waiting for EOI: If RQEOI is 1, then this bit is set by hardware when an interrupt request is sent and cleared by hardware when the EOI is returned. Software may write a 1 to this bit to clear it without an EOI.
62	R/W	0	PassPW: When 1, interrupt messages will be sent with the PassPW bit set and no ordering of the message with other upstream cycles is guaranteed. When 0, interrupt messages will be sent with PassPW clear, and the device must guarantee that the interrupt message will not pass upstream posted cycles within its queues. If a device supports only one of these behaviors, this bit is read-only and indicates which behavior is supported.
61:56	R/O	0	Reserved
55:32	R/W	0	IntrInfo[55:32]
31:24	R/W	F8h	IntrInfo[31:24]: Must default to F8h for compatibility with HyperTransport technology 1.01 and earlier devices. Values of F9 or above must not be used or conflicts with non-interrupt address spaces (defined in Chapter 5~~9~~) will result. Some hosts only recognize interrupts with this field set to F8h.
23: 6	R/W	0	IntrInfo[23:6]
5	R/W	0	IntrInfo[5]: Request EOI: When set, after each interrupt request is sent, the device waits for the Waiting for EOI bit to be cleared before sending

Bit	R/W	Reset	Description
			another interrupt.
4: 2	R/W	0	IntrInfo[4:2]: Message Type. Some devices may allow only certain application-specific combinations of message type with other bits. See Appendix F.1 for one example.
1	R/W	0	Polarity: For ~~devices with~~ external interrupt ~~inputs~~sources, when this bit is set, the interrupt signal is active-low. When clear, the interrupt signal is active-high. For ~~devices without external~~ internal interrupt sources, this bit is reserved.
0	R/W	1	Mask: When this bit is set, interrupt messages will not be sent from this source.

7.7 Address Remapping Capability Block

This configuration space capability block defines the location of the downstream memory windows on the secondary bus and defines 0 to 15 upstream memory windows on the secondary bus that will be mapped to different addresses on the primary bus. See Appendix A for use of this capability. All registers assume their default values upon warm reset. This capability is recommended for all bridges that have a HyperTransport link as their primary interface. The Address Remapping Capability Block is defined in Table 48.

Table 48. Address Remapping Capability Block Definition

31 28 27	26-25	24 20	19 16	15 8	7 0	
Cap. Type	Map Type	I/O Size	# of DMA Mappings	Capabilities Pointer	Capability ID	**+00h**
SBNPCtrl	Reserved		Secondary Bus Non-Prefetchable Window Base			**+04h**
SBPreCtrl	Reserved		Secondary Bus Prefetchable Window Base			**+08h**
DMACtrl 1	Reserved			DMA Primary Base 1		**+0Ch**
DMA Secondary Base 1				DMA Secondary Limit 1		**+10h**
…						**…**
DMACtrl n	Reserved			DMA Primary Base n		**+8N+Ch**
DMA Secondary Base n				DMA Secondary Limit n		**+8N+10h**

7.7.1 Capability Header

7.7.1.1 Capability ID: R/O

The value 08h indicates that this is a HyperTransport technology capability list item.

7.7.1.2 Capabilities Pointer: R/O

This field points to the next item in the capability list, or 00h if this is the last item.

7.7.1.3 Number of DMA Mappings: R/O

This field indicates how many (if any) DMA Primary/Secondary register sets are defined by this register block. In a HyperTransport-to-PCI bridge, at least one per REQ/GNT pair is suggested.

7.7.1.4 I/O Size: R/W: Warm Reset to 0

This field defines how many bits of downstream I/O addresses are discarded. The default is 0 to pass all 25 bits of a HyperTransport technology I/O cycle. All discarded address bits are 0s on the secondary bus. There may be a limited number of valid settings of this field for some devices.

7.7.1.5 Mapping Type: R/O

This field is currently 0 for this address mapping definition. Other values are reserved for future extensions.

7.7.1.6 Capability Type: R/O

The value 01000b indicates this is an address mapping extension block.

7.7.2 Secondary Bus Window Control Registers: R/W: Warm Reset to 0

(SBNPCtrl and **SBPreCtrl** above)

Table 49. Secondary Bus Window Control Register Definition

3	2	1	0
Enable	Isochronous	NonCoherent	Compat

- *Compat* indicates if downstream requests that pass through this memory window will have the compat bit set.
- *NonCoherent* indicates if downstream requests that pass through this memory window will have the coherent bit cleared, allowing hosts to relax memory ordering.
- *Isochronous* indicates if downstream requests that pass through this memory window will have the Isoc bit set. If this device has Isochronous Flow Control enabled, the requests will be issued in one of the Isoc virtual channels.
- *Enable* controls if downstream requests will be modified.

~~7.7.2~~7.7.3 Secondary Bus Window Base Registers: R/W: Warm Reset to 0

The secondary bus prefetchable and non-prefetchable window base fields define base address bits 39:20 of their respective memory windows on the secondary bus for downstream cycles. Each window has an Enable bit in the control register, which when clear (the default) disables any address mapping of cycles in the primary bus windows of that type.

7.7.4 DMA Window Control Register: R/W: Warm Reset to 0

Table 50. DMA Window Control Register Definition

3	2	1	0
Enable	Isochronous	NonCoherent	Reserved

- *NonCoherent* indicates if upstream requests that pass through this DMA window will have the coherent bit cleared, allowing hosts to relax memory ordering.
- *Isochronous* indicates if upstream requests that pass through this DMA window will have the Isoc bit set. If this device has Isochronous Flow Control enabled, the requests will be issued in one of the Isoc virtual channels.
- *Enable* controls if upstream requests will be compared against this DMA window.

~~7.7.3~~7.7.5 DMA Primary Base Register: R/W: Warm Reset to 0

When the Enable bit of the control register is set, this register defines base address bits 39:24 on the primary bus for a DMA mapping.

~~7.7.4~~7.7.6 DMA Secondary Base and Limit Registers: R/W: Warm Reset to 0

When enabled, each register pair defines bits 39:24 of the base and limit of an upstream memory window on the secondary bus, which will be mapped to the memory window starting at the corresponding primary base on the primary bus. (As with a PCI bridge, the unspecified least significant bits of base are assumed to be 0s and the LSBs of limit are assumed to be 1s.) Upstream cycles on the secondary bus outside the DMA windows and downstream memory windows will be passed to the primary bus unmodified.

7.8 Revision ID Capability

As shown in Table 51, the Revision ID Capability block indicates the revision of the HyperTransport specification to which this node is compliant. Every function of every device implemented in a HyperTransport node must place a HyperTransport capability list item in its configuration space, indicating the version of the specification to which it is compliant. This can be the primary or secondary interface capability if already present, or the HyperTransport Revision ID capability otherwise.

Table 51. Revision ID Capability Block Definition

31 24	23 16	15 8	7 0
Capability Type	Revision ID	Capabilities Pointer	Capability ID

- *Capability ID* is read-only 08h to indicate that this is a HyperTransport technology capability list item.
- *Capabilities Pointer* is a read-only register pointing to the next item in the capability list, or 00h if this is the last item.
- *Revision ID* indicates the revision of this specification to which this node is compliant. It follows the format of the Revision ID register in Section 7.5.6
- *Capability Type* is read-only 88h to indicate that this is a Revision ID capability block.

8 System Management

HyperTransport™ technology includes features that can be deployed in x86 systems to implement legacy behaviors or to implement system-level behaviors such as low-power state transitions. These features are also useful for non-x86 systems that require power management, and LDTSTOP# provides a faster method to change link frequency and width than warm reset. From the perspective of this specification, support of power management by devices other than the system management controller (typically part of the Southbridge) is optional. However, all devices must be capable of forwarding system management packets upstream and downstream. LDTSTOP# support is required in x86 systems.

HyperTransport technology system management supports several system-level functions. This chapter lists each of the functions and the means by which they are implemented using HyperTransport technology system management messages. The term system management controller (SMC) is used in this chapter to denote the HyperTransport technology device that controls system management state transition and legacy x86 pin sequencing.

8.1 Command Mapping

The system management controller (SMC) generates upstream system management requests by directing a posted byte WrSized packet to the system management address range defined in Chapter 5~~9~~. The count field is always 0, which indicates that only a single doubleword data packet follows the write, and it contains byte masks, not data. The byte masks are not used by the system management request and must always be all 0 bits. The format of these packets is as shown in Table 52.

Table 52. System Management Request WrSized Packet Format

Bit-Time	7	6	5	4	3	2	1	0
0	SeqID[3:2]		Cmd[5:0]					
1	PassPW	SeqID[1:0]		UnitID[4:0]				
2	Count[1:0]		Reserved					
3	Rsv						Count[3:2]	
4	SysMgtCmd[7:0]							
5	Addr[23:20]				Rsv			
6	Addr[31:24]							
7	Addr[39:32]							

The host generates downstream system management requests by sending a broadcast packet down all the HyperTransport I/O chains in the system. The address range in the broadcast packet identifies it as a system management request. The format of this packet is shown in Table 53.

Table 53. System Management Request Broadcast Packet Format

Bit-Time	7	6	5	4	3	2	1	0
0	SeqID[3:2]		Cmd[5:0]					
1	PassPW	SeqID[1:0]		UnitID[4:0]				
2	Reserved							
3	Reserved							
4	SysMgtCmd[7:0]							
5	Addr[23:20]				Rsv			
6	Addr[31:24]							
7	Addr[39:32]							

For both upstream and downstream cases, the type of system management request (SysMgtCmd[7:0]) is encoded as shown in Table 54.

Table 54. System Management Request Type Encoding

SysMgtCmd	Command Type
0000 xxxx	Reserved
00x1 xxxx	x86 Encoding, see Appendix F.2.1
0100 xxxx	SHUTDOWN Bits [3:0]: Implementation-specific ~~reserved for host-specific use.~~
0101 xxxx	HALT Bits [3:0]: Implementation-specific ~~reserved for host-specific use.~~
011x xxxx	x86 Encoding, see Appendix F.2.1
10xx xxxx	x86 Encoding, see Appendix F.2.1
11xx xxxx	Reserved

8.2 Special Cycles

The special cycles carried by system management packets are as follows:

- HALT—Generated by processor in response to execution of a HALT instruction.
- SHUTDOWN—Generated by processor in response to a catastrophic error.

These packets originate from the host and are broadcast downstream to all HyperTransport I/O devices in the system.

8.3 Disconnecting and Reconnecting HyperTransport™ Links

The *HyperTransport™ I/O Link Protocol Specification* comprehends the need for system states in which the HyperTransport links are disabled to save power, and therefore includes two features to facilitate this behavior. While these features have been described elsewhere in the specification, this section defines their use. These features are:

- Disconnect form of the NOP packet
- LDTSTOP# signal

The system-level conditions that control the assertion and deassertion of LDTSTOP# are outside the scope of this specification. However, the following rules govern the use of LDTSTOP# and the disconnection and reconnection of the HyperTransport link.

1. Once LDTSTOP# is asserted, it must remain asserted for at least 1 us. LDTSTOP# assertion must not occur while new link frequency and width values are being assigned by link-sizing software, or undefined operation may occur. (This is because both sides of a link must have link width and frequency programmed, and if one side has been programmed with new values and the other has not yet been programmed, the width and/or frequency of the two sides will not match.)
2. PWROK and RESET# assertions have priority over LDTSTOP# assertion, and LDTSTOP# must be deasserted before RESET# is deasserted. See Section 12.2 for more details.
3. A transmitter that perceives the assertion of LDTSTOP# finishes sending any control packet that is in progress and then sends a disconnect NOP packet (bit 6 in the first bit-time set). After sending this packet, the transmitter continues to send disconnect NOP packets through the end of the current CRC window (if the window is incomplete) and continuing through the transmission of the CRC bits for the current window. After sending the CRC bits for the current window, the transmitter continues to drive disconnect NOP packets on the link for no less than 64 bit-times, after which point the transmitter waits for the corresponding receiver on the same device to complete its disconnect sequence, and then disables its drivers (if enabled by the LDTSTOP# Tristate Enable bit described in Section 7.5.4.10). No CRC bits are

transmitted for the last (partial) CRC window, which only contains disconnect NOP packets. Since the HyperTransport protocol allows control packets to be inserted in the middle of data packets, and since transmitters react to the assertion of LDTSTOP# on control packet boundaries, a given data packet could be distributed amongst two or more devices after the disconnect sequence is complete.

4. A receiver that receives the disconnect NOP packet continues to operate through the end of the current CRC window and into the next CRC window until it receives the CRC bits for the current window. After sampling the CRC bits for the current window, the receiver disables its input receivers to the extent required by the LDTSTOP# Tristate Enable bit described in Section 7.5.4.10.
5. Note that LDTSTOP# can deassert either before or after the link disconnection sequence is complete. A link transmitter is not sensitive to the deassertion of LDTSTOP# until both its disconnect sequence as described in step 3 is complete, and the disconnect sequence for the associated receiver on the same device is complete.
   A link receiver is not sensitive to the deassertion of LDTSTOP# until both its disconnect sequence is complete and the disconnect sequence for the associated transmitter on the same device is complete.
6. A transmitter that perceives and is sensitive to the deassertion of LDTSTOP# enables its drivers as soon as the implementation allows, begins toggling the CLK with a minimum frequency of 2MHz and places the link in the state associated with the beginning of the initialization sequence (CTL = 0, CAD = 1s, CLK toggling). The transmitter is required to have CLK running within 1 us (to assure that the receive logic has a clock source). The clock frequency does not have to match the currently programmed frequency before CTL is asserted.
   A receiver that perceives and is sensitive to the deassertion of LDTSTOP# waits at least 1 us before enabling its inputs. This 1-us delay is required to prevent a device from enabling its input receivers while the signals are invalid before the transmitter on the other side of the link has perceived and reacted to the deassertion of LDTSTOP#.
   When a transmitter's corresponding receiver on the same device has been enabled, it is free to begin the initialization sequence described in Section 12.2.
7. After reconnecting to the link, the first transmitted packet after the initialization sequence must be a control packet, as implied by the state transitions of the CTL signal during link initialization. This is true even if the link was disconnected in the middle of a data packet transmission.
8. The CRC logic on either side of the link should be re-initialized after a disconnect sequence in exactly the same way as for a reset sequence.
9. Link disconnect and reconnect sequences do not cause flow control buffers to be flushed, nor do they cause flow control buffer counts to be reset.
10. LDTSTOP# should not be reasserted until all links have reconnected to avoid invalid link states. The means to ensure this is beyond the scope of this specification, although it is expected that this will be under software control.

The electrical state of the HyperTransport link during the disconnect state is controlled by a configuration bit, as described in Section 7.5.4.10.

9 Interrupts

HyperTransport™ technology provides a generic message-based interrupt system. Usage of the information carried in the messages is implementation-specific. See Appendix F.1 for x86-specific usage. The required programming model for discovery and configuration of interrupts in a HyperTransport technology device is described in Section 7.6.

9.1 Interrupt Requests

All interrupt requests, regardless of interrupt class, are sent from the interrupting device to the host bridge using posted byte WrSized packets to the reserved range defined in Chapter 5~~9~~. The Count field is always 0, which indicates that only a single doubleword data packet follows the write. The doubleword data packet is not used to carry byte masks; instead, it is used to carry interrupt information, as described below. Some systems limit the use of IntrInfo (see Appendix F.1 for one example). In general, software is required to configure devices to send only interrupt requests that are valid for the host. The format of interrupt-request packets is shown in Table 55~~Table 20Table 20~~.

Table 55. Interrupt Request Packet Format

Bit-Time	CTL	7	6	5	4	3	2	1	0
0	1	SeqID[3:2]		Cmd[5:0]					
1	1	PassPW	SeqID[1:0]		UnitID[4:0]				
2	1	Count[1:0]		Rsv	Reserved				
3	1	IntrInfo[7:2]						Count[3:2]	
4	1	IntrInfo[15:8]							
5	1	IntrInfo[23:16]							
6	1	IntrInfo[31:24]							
7	1	Addr[39:32]							
8	0	IntrInfo[39:32]							
9	0	IntrInfo[47:40]							
10	0	IntrInfo[55:48]							
11	0	Reserved							

The host bridge is then responsible for delivery to the correct internal target or targets.

Application Note
Because interrupt request packets travel in the posted channel, they push posted writes with the same UnitID as the interrupt request if the PassPW bit in the interrupt request packet is clear. Therefore, all preceding posted writes with the same UnitID, source, and target as the interrupt request will be visible at their targets within the host before the interrupt is delivered. If a flush is used to push posted writes before the interrupt request is sent, the PassPW bit in the interrupt request may be set. A fence may be issued ahead of the flush (or interrupt request with PassPW clear) to push posted writes in all UnitIDs upstream of the device sending the interrupt request.

The type of interrupt is identified by IntrInfo[4:2], and the meaning is implementation-specific, with the exception that Type 111b is reserved for the End of Interrupt (EOI) message.

Interrupts may require an EOI indication to acknowledge the servicing of the interrupt, controlled by IntrInfo[5] (RQEOI). A subsequent interrupt from that source shall not be sent until the Waiting for EOI bit is cleared. IntrInfo[31:8] will be returned in the EOI message, although some hosts may not support use of all bits. IntrInfo[7:6] may have special meanings in some systems, and therefore their use may be restricted. (See Appendix F.1 for one example of when IntrInfo is restricted.) Host bridges must be able to accept multiple interrupt requests without blocking the posted channel.

9.2 End of Interrupt (EOI)

EOI messages are sent in Broadcast message packets to all nodes across the HyperTransport I/O fabric. Each device is responsible both for accepting the EOI and clearing outstanding interrupts associated with the specified IntrInfo, and for passing the EOI down the ~~fabric~~chain. Some systems limit the use of IntrInfo, see Appendix F.1 for an example. The format of an EOI packet is shown in Table 56~~Table 21Table 21~~.

Table 56. EOI Packet Format

Bit-Time	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	SeqID[3:2]		Cmd[5:0]					
1	PassPW	SeqID[1:0]		UnitID[4:0]				
2	Reserved							
3	Reserved			MT[2:0]=111b			Rsv	
4	IntrInfo[15:8]							
5	IntrInfo[23:16]							
6	IntrInfo[31:24]							
7	Addr[39:32]							

EOIs use the same reserved address range as interrupt requests. The IntrInfo[4:2] field must always contain the value 111 (EOI). IntrInfo[31:8] duplicates IntrInfo[31:8] from the interrupt being acknowledged. The exception is that IntrInfo[15:8] in EOI may be 00h to match any value of IntrInfo[15:8] in the interrupt definition as a “wildcard” value for system hosts that do not return it in EOI. EOI may only be sent downstream.

As an alternative to sending EOI if the function that sent the interrupt can be uniquely identified, a configuration space write to the Interrupt Definition register (defined in Section 7.6.2) can be performed to clear the Waiting for EOI bit. This will complete service of the interrupt, and enable the interrupting function to send more interrupts.

10 Error Handling

10.1 Error Conditions

This specification defines a variety of error conditions. Each detectable error condition has one or more log bits associated with it, which are set when that condition is detected. The log bits are persistent through warm reset and are cleared by software writes of 1s. Implementation of all error checking logic beyond CRC checking is optional. Devices that do not check for one or more error conditions should hardwire the log bits for those conditions to 0. For reporting methods, see Section 10.2.

Depending on signaling conditions and synchronization between devices, one device may detect a reset before another and start driving reset signaling (CTL=0, CAD=1s). To prevent logging false errors, if CTL is deasserted when an error is detected, a device should drop the affected packet but otherwise delay responding to that error until CTL is asserted, or if RESET# is asserted first, the error is dropped. If CTL is deasserted for longer than the time specified by the CTL Timeout bit, a protocol error is logged immediately.

If RESET# is detected by a transmitting device before the receiving device during a data packet, some or all of the data may be received as 0's instead of the original values. While some devices are immune to the corruption that results due to higher-level protocol requirements before committing data to action, other devices will need to specifically eliminate this corruption. In order to achieve this, devices that are vulnerable to this corruption must not commit data to action until a subsequent control packet has been received.

10.1.1 Transmission Errors: 8-Bit, 16-Bit, and 32-Bit Links

A 32-bit cyclic redundancy code (CRC) covers all HyperTransport links. The CRC is calculated on each 8-bit lane independently and covers the link as a whole, not individual packets. CTL is included in the CRC calculation. In each bit-time, CAD is operated on first, beginning with bit 0, followed by CTL. For 16- and 32-bit links, where the upper byte lanes do not have a CTL bit associated with them, a CTL value of 0 (Data) is used.

The CRC is computed over 512 bit-times. Each new CRC value is stuffed onto the CAD bits of the link 64 bit-times after the end of the 512-bit-time window and occupies the link for 4 bit-times. Therefore, bit-times 64–67 (the first bit-time being 0) of each CRC window after the first contain the CRC value for the previous window. There is no CRC transmission during the first 512-bit-time window after the link is initialized, and the value of the transmitted CRC bits is not included in the CRC calculation for the current window. Therefore, each CRC window after the first is 516 bit-times in length—512 of which are included in the calculation of the CRC that will be transmitted in the next window. There is no indication on the bus that CRC information is being transmitted. It is the responsibility of the parties on both ends of the link to count bit-times from the beginning of the first valid packet after link synchronization to determine the boundaries of the CRC windows. During transmission of the CRC, the CTL bit will be driven to a value of 1 (Control).

For example, the contents of 8-,16- and 32-bit links during the first three CRC windows after link synchronization are shown in Table 57.

Table 57. CRC Window Contents After Link Synchronization

CRC Window After Sync	Number of Bit Times	Link Contains
1st	512	Payload for first window
2nd	64	Payload for second window
	4	CRC of first window
	448	Payload for second window
3rd	64	Payload for third window
	4	CRC for second window
	448	Payload for third window

The polynomial used to generate the CRC is:

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1$$

The CRC is calculated by computing the remainder resulting from the division of the data by the CRC polynomial. The register used to perform the calculation is seeded with all 1 bits at the beginning of each CRC window. Note that, in the classical CRC definition, thirty-two 0 bits are appended to the end of the data word before performing the division. This is not done in HyperTransport links. The CRC bits are inverted before being transmitted on the link to catch a wider range of bit errors.

The code below shows the calculation performed on the CRC accumulation register across a single bit-time (9 bits) of data.

```
static uint poly = 0x04C11DB7;   /* the polynomial */
uint compute_crc(uint data, uint crc)
{
 int i;
 for (i=0; i<9; ++i) {
   uint tmp = crc >> 31;   /* store highest bit */
   crc = (crc << 1) | ((data >> i) & 1);   /* shift message in */
   crc = (tmp) ? crc ^ poly : crc;   /* subtract poly if greater */
   };
   return crc;
 };
```

Detection of a link error on any byte lane will cause the appropriate CRC error bit to be asserted in the Link Control CSR. A CRC error must be assumed to have corrupted both data and control information, allowing the link interface to reach an indeterminate state. Corrupted information may have been passed to other interfaces of the device. CRC errors may be mapped to cause sync flooding, fatal, or nonfatal error interrupts.

It is possible that a sync flood may begin just before or at the bit-time where the CRC is transmitted, resulting in some or all bits of the CRC being transmitted as 1s. In this situation, a device should recognize the sync flood instead of detecting a CRC error. In order to guarantee this, a device should delay setting the CRC error bit(s) of the Link Control CSR until a sync flood has been ruled out (typically 16 bit-times).

10.1.2 Transmission Errors: 2-Bit and 4-Bit Links

For the purpose of CRC coverage, 2- and 4-bit links are analogous to 8-bit links running at quarter and half speed, respectively. That is, the CRC value generated for 2- or 4-bit links is identical to that generated for an 8-bit link carrying the same values. The extra CTL values are not used by the receiver and are not included in the CRC calculation.

Table 58 summarizes the CRC differences between 2-bit, 4-bit, and 8-bit (or wider) links.

Table 58. CRC Values for Different Link Widths

What	8-Bit	4-Bit	2-Bit
CRC calculation (LSB first) ("‖" means concatenation)	CTL ‖ CAD[7:0]	CTL_0 ‖ CAD_1[3:0] ‖ CAD_0[3:0]	CTL_0 ‖ CAD_3[1:0] ‖ CAD_2[1:0] ‖ CAD_1[1:0] ‖ CAD_0[1:0]
CRC window size	512 bit-times	1024 bit-times	2048 bit-times
CRC value stuffed onto CAD:	64 bit-times after start of window	128 bit-times after start of window	256 bit-times after start of window
CRC transmission length	4 bit-times	8 bit-times	16 bit-times
CRC test mode duration	512 bit-times	1024 bit-times	2048 bit-times

10.1.3 Protocol Errors

Protocol errors represent basic failings of the low-level packet protocol. Detectable protocol errors include the following:

- CTL transition on other than a 4-byte boundary, except during CRC diagnostic mode.
- CTL deassertion when data transfer is not pending, other than during CRC diagnostic mode.
- Command with associated data packet inserted when data transfer due to a previous command is pending.
- Bad command encoding detected in a control packet.
- CTL deasserted for more than the CTL Timeout bit (Section 7.5.8.4) allows.
- CTL deasserted during CRC transmission.

Detection of a protocol error results in the setting of the Protocol Error bit of the Link Error CSR. At this point, framing on the link must be assumed to have been lost, and the link interface may go to an indeterminate state. Protocol errors may be mapped to sync flood, fatal or nonfatal error interrupts.

10.1.4 Receive Buffer Overflow Errors

The HyperTransport technology flow control mechanism is supposed to ensure that received packets always have buffer space awaiting them. In the event that a packet is received that has no buffer available to receive it, the Overflow Error bit will be set in the Link Error CSR. Since this will only happen if packet-tracking state has been corrupted, the link interface must be assumed to be in an indeterminate state. Overflow errors may be mapped to sync flood, fatal, or nonfatal error interrupts.

10.1.5 End of Chain Errors

Directed packets (request or response) hit the end of a HyperTransport chain when they are forwarded the length of the chain without being accepted by any device along the way and reach a transmitter with its End of Chain bit set in the Link Control CSR. A packet may also be said to hit the end of chain if it is issued by a device to a link interface that has the End of Chain bit set. (A transmitter may also have the Initialization Complete bit in the Link Control CSR clear with the Drop On Uninitialized Link bit in the Command register set, which is equivalent to having End of Chain set.) Once the end of chain is reached, there is nowhere to forward the packet, so it is dropped.

For nonposted requests, hitting the end of chain is indicated by generating an appropriate response (RdResponse or TgtDone) for the request, with the Error and NXA bits set, and all read data as Fs. No logging is required, and no further action is taken.

For posted requests and responses, returning an error response is not possible. Accordingly, the fact that the packet was dropped is indicated by setting the End of Chain Error bit in the Link Error CSR of the link interface containing the disabled transmitter.

In double-hosted systems, a second type of End of Chain error is possible, where a posted request or response packet is received from one host by the other host and is not of a type that the receiving host accepts. In this case, there is no associated transmitter. The dropped packet is logged by asserting the Host Inbound End of Chain Error bit in the Error Status register.

End of Chain errors may be mapped to fatal or nonfatal error interrupts.

10.1.6 Chain Down Errors

Host interfaces are required to store state for nonposted requests that they issue to a HyperTransport chain, in order to match the SrcTag of the response with the original request. It is possible for the HyperTransport chain to come down after a nonposted request has been issued to it, but before the response is received. This can occur because of sync flooding on the chain or the assertion of HyperTransport technology RESET#. If this occurs, the host must flush the state of all outstanding nonposted requests and return (non-NXA) error responses for them. No logging occurs on the host interface. Signaled Target Abort may be asserted on the primary interface of the bridge, depending on what that interface connects to.

Slave devices are assumed to have all internal state reset on HyperTransport technology RESET#, and all subsidiary buses reset. Therefore, there is normally no need to flush nonposted request state in a slave device due to a chain going down. However, if some type of intelligent slave were implemented that maintained state through a HyperTransport technology RESET#, this device would need to flush its nonposted request state as well, and log appropriate error state.

10.1.7 Response Errors

Several types of errors are possible in which a response is received by a device that does not properly match a request:

- Response received by a device that does not have a request outstanding with that SrcTag.
- RdResponse received in response to a WrSized or Flush request.
- TgtDone received in response to a RdSized or Atomic Read-Modify-Write request.
- RdResponse received in response to a RdSized request with a Count field not matching the original request.
- RdResponse received in response to an Atomic Read-Modify-Write request with a Count field not equal to 1.

All of these errors are logged by setting the Response Error bit of the Error Status CSR. Response errors may be mapped to fatal or nonfatal error interrupts.

10.2 Error Reporting

HyperTransport technology devices detecting errors have several ways to report those errors to the system. They are listed here in order of increasing severity.

10.2.1 Error Responses

For nonposted requests that encounter an error condition, a HyperTransport technology error response may be issued. This is the preferred means of error signaling, where possible, because the error is localized to a particular transaction and indicated to the requester of that transaction, which may then take appropriate action. Error responses are indicated by the presence of an asserted Error bit in the response packet. There are two subtypes of error responses, indicated by the state of the NXA bit in the packet.

If the NXA bit is clear, this is a (non-NXA) error response. This indicates that the device receiving the request took an error. It is equivalent to a PCI Target Abort. If the transaction was a read, the returning data cannot be used. If the transaction was a write, the target location must be assumed to have gone to an undefined state. Devices receiving a (non-NXA) error response set the Received Target Abort bit in their Status CSR or (for bridges receiving the response on their secondary bus) their Secondary Status CSR. Devices driving a (non-NXA) error response set the

Signaled Target Abort bit in their Status CSR or Secondary Status CSR, as appropriate. (Non-NXA) error responses pass through bridges as (non-NXA) error responses.

If the NXA bit is set, this is a nonexistent address (NXA) error response. It indicates that a directed request failed to find a device on the chain that would accept it. It is roughly equivalent to a PCI Master Abort. Devices receiving an NXA error response set the Received Master Abort bit in their Status CSR or (for bridges receiving the response on their secondary bus) their Secondary Status CSR. NXA error responses propagate through HyperTransport technology bridges in the same manner as Master Aborts through PCI bridges – they are either converted to normal (non-error) responses or to (non-NXA) error responses, depending on the state of the Master Abort Mode bit of the Bridge Control CSR. All Fs are returned as data for read responses.

10.2.2 Error Interrupts

HyperTransport technology optionally defines two severity levels of error interrupt, Fatal Error and NonFatal Error. These may be used to report error conditions to the host which cannot be reported via an error response, but which do not prevent the HyperTransport chain from transmitting packets. The Fatal and NonFatal Error interrupts are implemented by providing the capability to generate two types of interrupt packets and additionally may be provided by external interrupt pins on a HyperTransport technology device. Devices that do not implement either should hardwire the enables for that interrupt to 0.

10.2.3 Sync Flooding

Sync packet flooding is used to report errors to the host that cannot be signaled by any other in-band means, due to the chain reaching a state where it can no longer be trusted to transmit packets. Sync flooding also has the effect of putting the entire chain into an inactive state after an error has been taken, with the intent of shutting down transmission before potentially corrupted data reaches its final destination. Devices detecting a sync flood must assume any data that they have recently received may be corrupted, and they should stop transferring data to other interfaces as quickly as possible. Once sync flooding has occurred, a warm reset of the chain is required to re-enable normal functioning of the chain. Because error status bits are persistent through warm reset, they can be polled to determine the cause of the sync flood event.

When an error that causes sync flooding is detected, the detecting device drives sync packets (CAD and CTL to all 1s) on its transmitter(s). This is recognized by devices at the other ends of the links as a sync packet, even if the nodes are out of sync or the clock has been corrupted. Any device detecting sync flooding on one of its receivers after link initialization has completed drives sync packets on its transmitter(s), including the one back to the device it received the sync flood from. All transmitters, once flooding, continue to drive sync packets until reset. In this way, the sync flood propagates the entire length of the chain in both directions. CRC is not generated on links transmitting sync packets, nor is it checked on incoming links on which a sync packet has been detected. All devices participating in the flooding set the Chain Fail bit in the Error Status register, whether they initiated it or are just propagating it.

Sync flood initiation by a HyperTransport technology device is analogous to SERR# assertion by a PCI device. Devices must have the SERR# Enable bit of the Command register set in order to initiate sync flooding. Devices initiating sync flooding set the Signaled System Error bit in the Status register. Sync flood propagation from the secondary to the primary interface of a HyperTransport-to-HyperTransport bridge is analogous to SERR# propagation through a PCI-PCI bridge. The bridge sets the Detected System Error bit in its Secondary Status register when it detects the sync flooding, as long as it is not the initiator of the sync flood. The SERR# Enable bit of the Bridge Control register must be set in order for the secondary interface to propagate the sync flood information to the primary interface. The primary interface in turn uses the SERR# Enable bit of the Command register to determine whether to propagate the sync flood to the primary bus. If propagating the sync flood to the next bus up (or to the host) is not desired, the sync flood may at this point be converted to a Fatal interrupt or a NonFatal Error interrupt. Even if a device has the SERR# Enable bit cleared, if an error occurs that would cause it to initiate a sync flood, the LinkFail bit will be set.

Sync flood propagation from device to device along a chain is analogous to SERR# assertion propagating along the bus in PCI. No enables are required for sync flood propagation within a chain.

Sync flooding always propagates from the primary to the secondary interface of HyperTransport-to-HyperTransport bridges. Bridges can also initiate sync flooding on their secondary bus due to internal errors. This has no analog in PCI. It conveys no error information and is merely used to disable links on subsidiary chains and to stop traffic as quickly as possible. No enables are required.

10.2.4 Error Routing CSRs

Table 59 shows the CSR fields used to log each error type and route error assertion to the appropriate reporting method. Entries are of the form Register/Subfield.

All checked-for error conditions have a log bit associated with them. The “enable” columns in Table 59 give the CSR bit that routes that error condition to the appropriate response. If the log bit is ever asserted with that enable, the error notification will occur.

Table 59. Error Routing Registers

Error Type	**Log Bit**	**Flood Enable**	**Fatal Error Enable**	**Nonfatal Error Enable**
Protocol	LinkErr/ProtErr	ErrHnd/ProtFloodEn	ErrHnd/ProtFatalEn	ErrHnd/ProtNonFatalEn
Overflow	LinkErr/OvfErr	ErrHnd/OvfFloodEn	ErrHnd/OvfFatalEn	ErrHnd/OvfNonFatalEn
EOC	LinkErr/EocErr	▨	ErrHnd/EocFatalEn	ErrHnd/EocNonFatalEn
Inbound EOC*	HstCmd/InbEocErr*	▨		
Response	ErrHnd/RespErr	▨	ErrHnd/RespFatalEn	ErrHnd/RespNonFatalEn
CRC	LinkCtrl/CrcErr[3:0]	LinkCtrl/CrcFloodEn	ErrHnd/CrcFatalEn	ErrHnd/CrcNonFatalEn
SERR*	SecStatus/SerrDet*	BridgeCtrl/SerrEn	ErrHnd/SerrFatalEn*	ErrHnd/SerrNonFatalEn*

Notes:

** --Indicates host-only bits. These error conditions are not checked in slaves, and the CSR bits are reserved in slaves.*

Shaded enable boxes indicate that the given error type may not be mapped to the given reporting method.

Sync flooding can only be initiated on the primary link by a device if the SERR# Enable bit in the Command register (7.3.1.4) is set.

11 Clocking

HyperTransport™ technology systems consist of devices connected by HyperTransport links. Devices within a HyperTransport fabric may or may not be clocked by clocks derived from the same frequency source. Section 11.1 describes the clock source requirements for HyperTransport technology devices.

11.1 Clocking Mode Definitions

Each HyperTransport technology device has a transmit clock, which is used to generate its CLK outputs, and a receive clock, to which incoming packets are synchronized in the receiver.

Three operating modes of HyperTransport technology devices are defined.

- In *Synchronous (Sync)* mode, each transmit clock must be derived from the same time base as the receive clock in the device to which it is connected. In addition, the transmit clocks from each side of the link must operate at the frequency programmed by their respective Link Frequency registers. Both the receiver and transmitter of a given side of the link will operate at the same frequency, because there is only one Link Frequency register for each side of the link.
- In P*seudo-synchronous (Pseudo-sync)* mode, each transmit clock must be derived from the same time base as the receive clock in the device to which it is connected. The transmit clock frequency for either device may be arbitrarily lower than the frequency programmed into its Link Frequency register and must not exceed the maximum allowed receive clock frequency in the other device. The maximum allowed receive clock frequency of a link is the highest frequency indicated in the Frequency Capability register.
- In *Asynchronous (Async)* mode, each transmit clock need not be derived from the same time base as the receive clock in the device to which it is connected. In order to cope with frequency error due to running nominally matched transmitter/receiver pairs from different time bases, the maximum transmit clock frequency for one device can exceed the maximum receive clock frequency in the other device by no more than 2000 parts per million (2000 ppm). Further, the transmit output clocks can exceed the frequency programmed into the Link Frequency registers by no more than 1000 ppm. As in Pseudo-sync mode, the transmit clock frequency for one device may be arbitrarily lower than the frequency programmed into its Link Frequency register and must not exceed the maximum allowed receive clock frequency in the other device. See Section 11.3 for a description of one scheme for handling this situation.

See Section 7.5.7 for a description of HyperTransport link frequency selection.

All HyperTransport technology devices must support Sync mode operation. Devices may also implement Pseudo-sync and Async modes based on their unique requirements. The means by which the operating mode is selected for a device that can support multiple modes is outside the scope of this specification.

11.2 Receive FIFO

Each HyperTransport technology receiver contains a FIFO that is clocked by the transmitter. See the *HyperTransport™ Technology Electrical Specification* for the details of this structure and the guidelines that govern its design. This section introduces the receive FIFO to motivate the discussion in sections 11.3 and 12.2.

The FIFO (shown in Figure 5) is sized to absorb the *dynamic variatio*n between the transmitter's clock and the receiver's clock. Some sources of this dynamic variation are:

- Temperature
- Voltage (either of the transmitter or the receiver)
- Accumulated phase error in a PLL
- Noise that affects the clock and data in the same way. (If the noise affected clock and data differently then this would affect the maximum bit rate, not the buffer depth.)

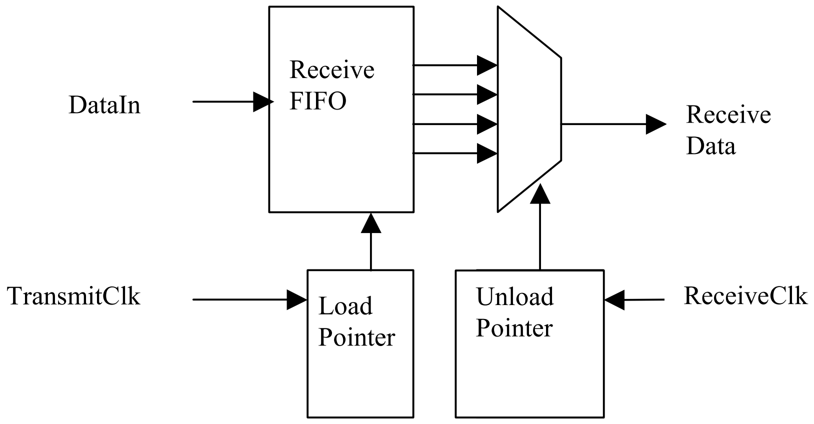


Figure 5. Receive FIFO

For links that are wider than 8 bits, the FIFO absorbs the difference in delay between 8-bit segments of the link. For links in which the two connected devices are clocked by different sources, the FIFO absorbs the frequency variation. See Section 11.3. Also, the FIFO can be used to provide buffering between a narrow high-speed link and a wider slower data path inside a receiver.

Clocking successive bit-times into different FIFO entry flops serves to increase the valid time of each flop. Before operating the link, the load and unload pointers must be initialized relative to each other. The sequence that supports this requirement is described in Section 12.2.

11.3 Async Mode Implementation Example

In Async mode, the transmit clock in one device and the receive clock in the other device are not derived from the same time base. The relationship between the FIFO load pointer and unload pointer may have to be adjusted dynamically.

If the receive clock is faster than the transmit clock, then the unload pointer occasionally has to be frozen. This situation can occur in both Pseudo-sync and Async modes.

If the receive clock is slower than the transmit clock, one method for ensuring that the transmitter does not overrun the receiver is described below. Note that this situation can only occur in Async mode. The transmitter frequency can exceed its programmed frequency by no more than 1000 ppm and can exceed the receiver's frequency by no more than 2000 ppm. This is less than the rate at which CRC bits are inserted onto the link (4 parts in 1092 for the first CRC window after initialization). CRC bits are sent by the transmitter and recomputed by the receiver. The receiver recomputes the CRC bits from the packet stream that is registered into the receiver's clock domain from the receive FIFO. The CRC bits are not placed directly into the receive FIFO, however. Instead, the CRC bits are placed into dedicated flops that are clocked by the transmit clock, and the bits are evaluated by receive-clock logic only after sufficient time has passed to ensure that these flops can be reliably sampled. Since the CRC bits only appear every 512 bit-times there is a sufficiently large sample window for these flops. By not placing CRC bits into the receive FIFO and therefore not incrementing the unload pointer, the receiver can always keep up with the transmitter.

11.4 Link Frequency Initialization and Selection

Cold reset initializes HyperTransport I/O link transmitters to a link clock frequency of 200 MHz. All HyperTransport I/O link receivers must support a HyperTransport technology clock of at least 200 MHz. Initialization firmware can reprogram the link transmitter frequencies and initiate a warm reset or LDTSTOP# disconnect sequence to invoke the change to the link clock frequencies. See Section 7.5.7 for details.

12 Reset and Initialization

12.1 Definition of Reset

Two types of reset are defined at the fabric level as follows:

- *Cold reset*—Node logic is reset. All links are reset. UnitIDs are assigned. All CSRs are reset. Cold reset is caused by the deassertion of PWROK together with the assertion of RESET#. Note that this sequence may be initiated under software control.
- *Warm reset*—Same as cold reset, except that CSRs defined to be persistent (expected to be mostly error state) are not reset. Warm reset is caused by asserting RESET# and keeping PWROK asserted. It may be initiated under software control. The system must ensure that warm reset does not occur during changes in link frequency or width settings, or else the link may not complete initialization.

The means by which PWROK and RESET# are generated within a specific system are outside of the scope of this specification.

12.2 System Powerup, Reset, and Low-Level Link Initialization

For a cold reset sequence, PWROK is asserted at least 1 ms after the power and clock sources for all HyperTransport™ technology devices have become stable. RESET# must be asserted 1 ms before PWROK is asserted, and RESET# must remain asserted for at least 1 ms beyond the assertion of PWROK. Since the state of RESET# is undefined during some of the time before PWROK is asserted, PWROK's deassertion should be combined with RESET# to generate internal resets.

RESET# must remain asserted until the CLK signal from all transmitters is stable. If a device requires more than 1 ms after PWROK assertion to stabilize its transmit clocks, it may drive RESET# to extend it until transmit clocks are stable.

For a warm reset sequence, RESET# must be asserted for at least 1 ms.

LDTSTOP# must be deasserted at least 1 us before RESET# is deasserted, and it must remain deasserted until the link has completed the synchronization sequence described below.

A cold reset enables transactions to flow across the link using the minimum CAD widths that are common to the transmitters and receivers on each side of the link. This is accomplished as follows:

A HyperTransport technology device whose receiver is connected to a narrower transmitter on another device must have its unused CAD inputs connected to a logical 0. A device whose HyperTransport link is not used in the system must have its CLK, CTL and CAD inputs connected to a logical 0.

While RESET# is asserted during a cold reset, each device drives its CTL signal to a logical 0, and drives its output CAD signals to a value that is based on the width of its receiver, according to Table 60. This value must be held until after the device has asserted its own CTL signal and sampled the assertion of the CTL signal driven from the other device. (This assures that each device can sample CAD safely, even if the device takes considerable time after reset to stabilize clocks and sample CAD.) If the transmitter is narrower than the receiver, all the output CAD signals are driven to a logical 1.

Table 60. CAD Value Driven Based on Receiver Width

Receiver Width (Bits)	CAD[31:0] Value Driven	Notes
2	0000 0003	
4	0000 000F	
8	FFFF FFFF	Required for backward compatibility
16	FFFF FFFF	Required for backward compatibility
32	FFFF FFFF	

At the deasserting edge of cold RESET#, each device samples its input CAD signals and uses this sampled value to determine its transmitter and receiver widths, according to Table 61. The result of this process is reflected in the cold reset values of the LinkWidthIn and LinkWidthOut registers. If all CAD inputs are logical 0, the link is unused, and the End of Chain bit in the Link Control register will be set.

Table 61. CAD Value Sampled for Transmitter and Receiver Width

CAD[31:0] Value Sampled (Hex)	Transmitter and Receiver Widths (Bits)	Notes
0000 0000	N/A	Unused link
0000 0003	2	
0000 000F	MIN(4, Receiver width, Transmitter width)	
0000 00FF	MIN(8, Receiver width, Transmitter width)	
0000 FFFF	MIN(8, Receiver width, Transmitter width)	Required for backward compatibility
FFFF FFFF	MIN(8, Receiver width, Transmitter width)	Required for backward compatibility

Warm reset preserves the transmitter and receiver widths programmed by software, so it is slightly different than cold reset.

While RESET# is asserted during a warm reset, each HyperTransport technology device drives its outbound link(s) to the state listed in Table 62.

Table 62. Signal States During Reset

Signal	State During Reset
CLK	Toggling
CTL	Logic 0
CAD[n-1:0] (programmed width)	Logic 1
CAD[31:n] (if present)	Logically undefined but within DC electrical specification. Logic 0 recommended for easier debug.

Note that this state does not correspond to any particular HyperTransport technology packet type.

The link initialization sequence from this point forward in time involves transmitting and receiving values on the CTL and CAD signals, and it is the same for cold and warm reset. The timing of the sequence described below, in terms of bit-times, is the same for 8-, 16-, and 32-bit links. The bit-time counts for 4- and 2-bit links should be doubled and quadrupled, respectively.

The discussion below refers to the CLK edges (rising or falling) across which the transmitter places new values on the CTL and CAD signals. Implied in this text is that the receiver registers the new data (using the transmitted clock) using that the same clock edge (rising or falling). The transmitter's physical interface to the link delays the clock relative to the data in order to position the clock in the middle of the data window.

A device-specific time after the deassertion of RESET#, each device asserts its CTL signal across a rising CLK edge, initiating a sync sequence. The timing of this transition as observed at the receiver is shown in Figure 6.

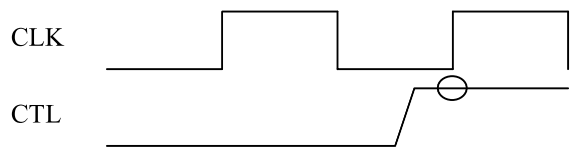


Figure 6. Sync Sequence Timing for Link Initialization

The assertion of the CTL signal serves to indicate to the device at the other side of the link that this device is ready to initialize the link. Devices perform whatever device-specific functions they may require between the time RESET# is deasserted and the time they assert CTL. This may include ramping their internal clocks to full frequency, initializing the receivers, and reading configuration state from off-chip.

When a device has asserted its own CTL signal and sampled the assertion of the CTL signal driven from the other device, it continues to drive the Sync packet for 16 bit-times (or 50 us after an LDTSTOP# disconnect, as specified by the Extended CTL Time bit in Section 7.5.4.11) and then inverts both CAD and CTL across a rising clock edge.

From this point until the initialization sequence is complete, unused bits of CAD are logically undefined, but the transmitter must drive to electrical levels that satisfy the DC specification. Logic 0s are recommended for easier debug.

The deassertion of the incoming CTL/CAD signals across a rising CLK edge is used in the transmit clock domain within each receiver to initialize the load pointer. The deassertion of the incoming CTL and CAD signals is synchronized to the core clock domain and used to initialize the unload pointer within each receiver. The length and uncertainty of this synchronizer must be included to determine the proper relationship between the load pointer and the unload pointer. Note that CTL cannot be used to initialize the pointers for byte lanes other than 0 in a multi-byte link, as CTL only exists within the byte 0 transmit clock domain. After this point, all transitions of CTL must be on a 4-byte boundary.

Each device continues to drive this state on its outbound links for the number of bit-times shown in the table below.

Each device then drives the CAD signals to logic 1 on a 4-byte boundary across a rising CLK edge, while leaving the CTL signal deasserted, for exactly four bit-times. The transition from all CAD signals deasserted to all CAD signals asserted serves to frame incoming packets. The first bit-time after these four must have CTL asserted, and is both the first bit-time of a new command packet and the first bit-time of the first CRC window. It also occurs across a rising CLK edge.

Once the initialization sequence is complete, the transmitter should always drive unused bits of CAD to logic 0 to reduce noise and power.

The entire sequence is shown in Table 63.

Table 63. Values of CTL and CAD During Link Initialization Sequence

CTL	CAD	Duration: 8-, 16-, and 32-Bit Links (Bit Times)	Duration: 2- and 4-Bit Links (Bit-Times)	Notes
0	1	N/A	N/A	Value held during reset
1	1	16 (minimum)	64/32 (minimum)	CTL asserts device-specific time after RESET# deasserts Pattern held at least 16/32/64 bit-times after both devices sample assertion of CTL (50 us following LDTSTOP# if CTL extended).
0	0	512+4**N**	2048+16**N**/1024+8**N**	1->0 transition on incoming CTL/CAD initializes load pointer in transmit clock time domain 1->0 transition on incoming CTL/CAD synchronized to core clock and used to initialize unload pointer in receive clock time domain
0	1	4	16/8	0->1 transition on CAD serves to frame incoming packets
1	??	N/A	N/A	0->1 transition on CTL defines start of first control packet and represents first bit-time of first CRC window

Note:

N can be any integer, starting with 0.

Using the initialization sequence as defined in the above section, synchronous, pseudo-synchronous, and asynchronous devices can inter-operate as long as they share a common input clock.

12.3 I/O ~~Fabric~~ Chain Initialization

I/O ~~fabric~~ chain initialization is largely software driven. A sample initialization sequence consists of the following steps:

1. One or more reset sequence initiators assert RESET# (and possibly deassert PWROK). Each initiator must sequence the PWROK and RESET# signals according to the rules defined in Section 12.2. Multiple initiators may or may not release (deassert) RESET# at the same time. In any event, the last initiator to release RESET# determines when the initialization sequence begins. Note this means that each initiator must sample as well as drive RESET#.

   PWROK and RESET# are propagated from the primary side of a bridge to the secondary side, but not from secondary to primary.

2. The low-level link initialization sequence described in Section 12.2 takes place between each node in the ~~fabric~~chain.
3. Each node sends buffer-release packets to inform the transmitter(s) to which it connects how many buffers it contains.

In a double hosted chain, the host bridge at one end of the chain is designated the master host bridge, and the other the slave. How a host bridge determines whether it is a master or slave is outside the scope of this specification.

4. The slave host bridge, if any, goes to sleep and waits for the master host to initialize the chain, so that only one host will be accessing the HyperTransport technology command registers (causing the Master Host bit to change) of any one device at a time. The method of ensuring this (timers, sideband signals, etc.) is implementation-specific. The master host bridge proceeds with the initialization sequence. At the beginning of the sequence, nextFreeID = 01h.
5. The master host bridge checks the Initialization Complete bit for the outgoing link of the last initialized device on the chain (initially its own) to determine if a device has been detected at the other end of the link. It also checks the ~~CRC E~~error bits to see if the link has taken any errors since reset. If there is no device, or the link is taking errors, chain sizing is complete—proceed to step 9.
6. ~~The bridge~~Software issues CSR accesses to Device Number 00h, which is the Base UnitID that all devices assume at reset, and which is also the default responded to by host bridges. The reads are responded to by the first uninitialized device on the chain. By reading the Class Code, Vendor ID, and Device ID, software can determine the type of device with which it is talking. Performing a write to the Command register (without changing any fields) will cause the Master Host bit to get set, which indicates which link on that device is pointing toward the host bridge. By polling the CRC, Protocol, Overflow, and End of Chain Error bits for that link (see sections 7.5.4.8 and 7.5.10), software can determine that the device is not seeing errors on the link between it and the host. Software can then set the Flood, Fatal, and/or Nonfatal Enable bits for the link from both ends. If the link is taking errors, chain sizing is complete, proceed to step 9.

~~6.~~7. Software reads the Unit Count field. If nextFreeID+UnitCount exceeds the largest available UnitID value for the chain (typically 31), the device cannot be initialized and the device before it in the chain must have the End Of Chain bit set. Initialization is done.

~~7.~~8. ~~By reading the Class Code, Vendor ID, and Device ID, software can determine the type of device with which it is talking. If it has reached a HyperTransport technology device, software~~ Software writes the Base Unit ID with nextFreeID, and increments nextFreeID by the Unit Count value of the device. Now that the device will no longer respond to accesses to 00h, the process can be repeated for the next link, starting back at step 5. If a slave host bridge has been reached, software sets the Double-Ended bit on both that bridge and the master host bridge, and proceeds to step 10 for link partitioning.

~~8.~~9. At this point, an end to the chain has been found without reaching another host bridge. Software sets the End of Chain and Transmit Off bits for the last link in the chain, and ~~fabric~~ chain initialization is complete. If there is a bridge at the other end and the sizing algorithm has not reached it due to a break in the chain, it will wake up after the master host has completed initialization, find its Double-Ended bit clear, and size the chain from the other end to the break, starting at step 5. The result will be two single-ended chains, each with a master host bridge. Initialization is done.

~~9.~~10. At this point, the entire chain has been sized and found to have host bridges at both ends. When the slave host bridge wakes up, it will find its Double-Ended bit set and know that no sizing is required on its part. All intermediate devices will have their Master Host bit pointing towards the master host bridge.

In a non-sharing double-hosted chain, software must select the location at which it wishes to break the chain and then access the nodes on either side of the break from the host bridge on that side. First, the End of Chain bit for the link to be broken is set from each side while the link is idle. When both devices are ignoring the link, the Transmit Off bit for each side can be set. At this point, the slave host bridge should write the HyperTransport technology command registers of all devices on its side of the break so that Master Host and Default Direction point towards it.

In a sharing double-hosted chain, the peer-to-peer deadlock loop described in Section 4.7 may make load balancing impossible, since all devices must have their Master Host bits pointing towards the master host bridge.

If peer-to-peer transactions are not used in a sharing double-hosted chain, load balancing can be achieved by changing the Master Host and/or Default Direction bits on some of the devices in the chain, resulting in a "soft" partitioning.

In any chain configuration, partitioning must be done before setting the bus master enable bits in the devices, as described in Section 7.3.1.3.

The initialization process can be made more robust by providing a facility to time out the CSR accesses used for sizing, in the event that a device fails to respond. This possibility is beyond the scope of this specification.

12.3.1 Finding the Firmware ROM

System implementations can be built in which software initialization code is stored in a firmware ROM that resides behind a default bridge on one of several HyperTransport I/O chains connected to the host. Further, system implementations can be built that do not require the host to be hardware-configured to identify the I/O chain that contains the default bridge. One possible method of initializing a system with these two characteristics involves a host that after reset sends a firmware code-fetch down each I/O chain connected to it. The I/O chain that contains the default bridge will respond without error, while all the other I/O chains will respond with an NXA error. This allows the host to identify the compatibility I/O chain, and subsequent firmware fetches can be directed down that chain only. To guarantee that this method will work even when devices on the compatibility chain are slow to initialize, the Drop on Uninitialized Link bit, described in Section 7.5.3.2.5, is inactive by default. Once the compatibility chain has been successfully accessed, the Drop on Uninitialized Link bit should be set to prevent hangs if a link becomes inoperable.

12.4 Link Width Initialization

Note that the hardware-sequenced link-width negotiation sequence described in Section 12.2 does not result in the links operating at their maximum width potential. 16-bit, 32-bit, and asymmetrically-sized operation must be enabled by a software initialization step. Each link controller contains two pairs of control register fields relating to the width of the link, as follows:

- A pair of fields that are hardwired to indicate the maximum supported widths of the inbound and outbound links.
- A pair of fields that are initialized after a cold reset to a particular value based on the result of the link-width negotiation sequence, as described in Section 12.2. This pair of fields controls the actual link width and is persistent across a warm reset.

At cold reset, all links power-up and synchronize as described in Section 12.2. Firmware (or BIOS) interrogates all the links in the system, reprograms all the links to the desired width, and then takes the system through a warm reset to change the link widths. See Section 7.5.5 for details on the Link Configuration register, which contains the Link Width fields.

After a HyperTransport technology disconnect-reconnect sequence, devices that implement the LDTSTOP# protocol described in Section 8.3 are required to update their link widths in exactly the same way as they do after a warm reset sequence. This allows initialization software for systems built from such devices to use the LDTSTOP# protocol rather than warm reset to invoke link width changes.

12.5 Link Frequency Initialization

At cold reset, all links power-up with 200-MHz clocks. For each link, firmware reads the Frequency Capability register, described in Section 7.5.9, of each device to determine the supported clock frequencies. The reported frequency capability of each device, combined with system-specific knowledge of the board layout and power requirements is used to determine the frequency to be used for each link. Firmware will write the Frequency register, described in Section 7.5.7, for both devices of each link to set the frequency to be used. Once all devices have been configured, firmware will initiate an LDTSTOP# or RESET# of the affected buses to cause the new frequency to take effect.

Protocol Appendices

A Address Remapping Capability

HyperTransport™ technology is meant to provide a high-bandwidth backbone for I/O systems, which are likely to contain a variety of other buses with varying addressing capabilities. If multiple buses with limited address space are to be combined, then it is helpful to be able to map the smaller address spaces of individual buses into different locations within the HyperTransport technology address map. The capability block defined in Section 7.7, combined with a HyperTransport technology bridge header defined in Section 7.4 provides the programming interface for controlling three mechanisms. When these mechanisms are in use, setting the bits in the bridge header that affect address decoding (VGA Enable, VGA Palette Snoop, and ISA Enable) ~~cannot be used~~may result in undefined address decode behavior.

A.1 I/O Space Aliasing

The I/O Size register indicates the number of upper bits of I/O address space that are not used when forwarding downstream I/O space cycles to the secondary bus. This allows I/O addresses to be translated down into the address range that is available on the secondary bus.

In Figure 7, the I/O Window for the first PCI bridge (defined by the I/O Base and Limit registers) is 10_0000–17_FFFFh, with an I/O size of 6 to create a 19-bit PCI I/O space. The second bridge has an I/O Window of 40_0000–4F_FFFFh, with an I/O size of 5 for a 20-bit PCI I/O space. Because some PCI devices only support 16-bit I/O decoding, this allows more devices than would be possible on a single PCI bus.

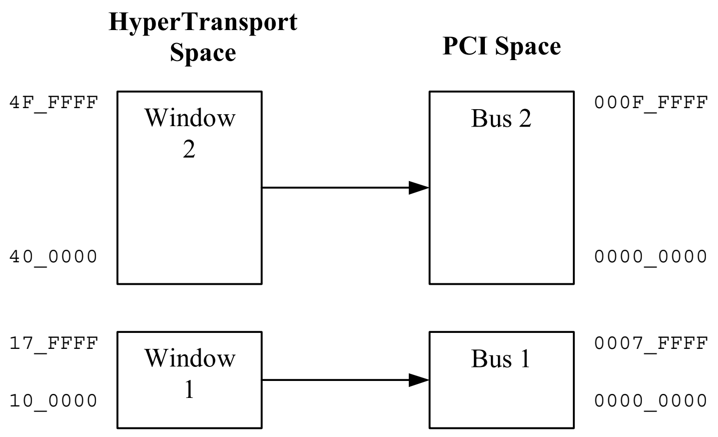


Figure 7. I/O Space Aliasing

A.2 Memory Space Mapping

The Secondary Bus Prefetchable Window Base and Secondary Bus Non-Prefetchable Window Base registers allow downstream accesses to be mapped to arbitrary positions in secondary bus memory space. While the Memory Base and Limit registers always define the range of addresses to be claimed on the primary bus and forwarded to the secondary bus, cycles that are claimed have their addresses modified because of the difference in the base addresses of the windows on the two buses, as these equations describe:

```
PriSecNPDiff = PriNPBase - SecNPBase
SecNPAddr = PriNPAddr - PriSecNPDiff
```

```
PriSecPFDiff = PriPFBase - SecPFBase
SecPFAddr = PriPFAddr - PriSecPFDiff
```

Because the addresses of the downstream memory windows on the secondary bus have been shifted from their locations on the primary bus, the address range of cycles that a bridge will not claim on the secondary bus must also be shifted. Therefore, memory cycles with addresses from SecNPBase to SecNPLimit or from SecFPBase to SecFPLimit will not be claimed by the bridge on the secondary bus.

```
SecNPLimit = PriNPLimit - PriSecNPDiff
SecPFLimit = PriPFLimit - PriSecPFDiff
```

Once claimed, a memory cycle forwarded from the secondary bus to the primary bus has its address modified according to the DMA Windows in the following section.

A.3 DMA Window Remapping

The DMA Secondary Base, DMA Primary Base, and DMA Secondary Limit registers define memory windows in the secondary bus memory space that are mapped to arbitrary positions on the primary bus. The resulting location of the DMA window on the primary bus is defined by these equations:

```
PriSecDMADiff = PriDMABase - SecDMABase
PriDMALimit = SecDMALimit + PriSecDMADiff
```

A cycle whose address falls within a DMA window on the secondary bus will have its address on the primary bus modified by this equation:

```
PriDMAAddr = SecDMAAddr + PriSecDMADiff
```

Accesses outside both the secondary bus DMA windows and the secondary bus memory windows defined above are passed upstream with unmodified addresses.

Software should ensure that the locations of the DMA windows on the secondary bus are outside of the secondary bus memory windows and that the DMA windows on the primary bus are outside of the primary bus memory windows, or undefined operation may result.

Figure 8 illustrates this example usage in a HyperTransport-to-PCI bridge:

```
Sec Bus Non-Prefetchable Base = 00_000h, Prefetchable Base = 00_C00h
DMA Sec Base1 = 00_40h, DMA Sec Limit1 = 00_7F, DMA Pri Base1 = 0C_C0
DMA Sec Base2 = 00_80h, DMA Sec Limit2 = 00_BF, DMA Pri Base2 = 0C_80
Bridge Header has NP base and limit of 08_8000_0000 and 08_BFFF_FFFF
Bridge Header has P base and limit of 09_C000_0000 and 0A_FFFF_FFFF
```

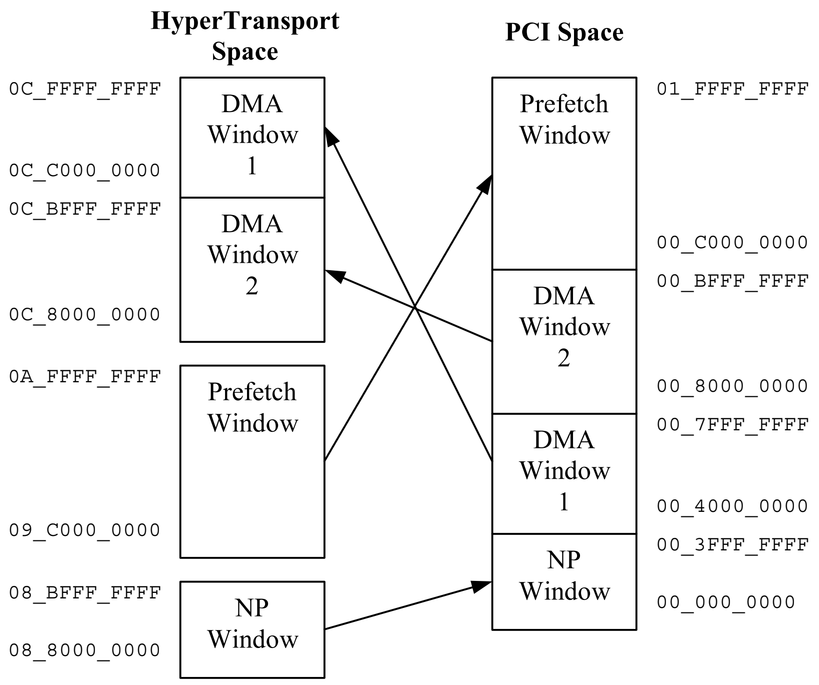


Figure 8. ~~DMA~~ Memory Window Remapping

B Ordering Rules and Mapping of ~~Supported~~ Other I/O Protocols

Ordering requirements for request packets are determined by their requester, and they follow the request all the way to the destination. Ordering rules for responses are also determined by the original requester and are taken from the request packet.

This appendix provides the mapping from the traffic types of each of the supported protocols to HyperTransport™ technology packet fields.

HyperTransport™ technology is intended to support connections to I/O bridges that use a variety of I/O protocols. At this time, there are ~~two~~ four supported I/O bus protocols identified, PCI, PCI-X, ~~and~~ AGP, and host processors, each of which has different ordering requirements, as described in this appendix.

~~A.1~~B.1 Processors

Processors should generate nonposted writes for I/O and configuration space. It is implementation-specific as to whether processors generate posted or nonposted writes for memory-mapped I/O.

In order to safely implement the producer-consumer model in all configurations, processor requests should follow PCI ordering rules, with the PassPW bit always clear on ~~both~~ requests ~~and responses~~.

Because I/O space cycles cannot cross doubleword boundaries, unaligned accesses by a processor that cross a doubleword boundary must be broken into successive HyperTransport cycles that will guarantee correct order of operation for the particular processor architecture.

~~B.1~~B.2 PCI

B.2.1 Ordering

The PCI ordering rules listed in Table 64 are taken from the *PCI Local Bus Specification, Revision 2.2* ~~*(6/8/98 draft)*~~, Appendix E. See that specification for more information.

Table 64. PCI Bus Transaction Ordering Rules

Row Pass Column?	**Posted Memory Write (PMW)**	**Delayed Read Request (DRR)**	**Delayed Write Request (DWR)**	**Delayed Read Completion (DRC)**	**Delayed Write Completion (DWC)**
PMW	No	Yes	Yes	Yes	Yes
DRR	No	Yes/No	Yes/No	Yes/No	Yes/No
DWR	No	Yes/No	Yes/No	Yes/No	Yes/No
DRC	No	Yes	Yes	Yes/No	Yes/No
DWC	Yes/No	Yes	Yes	Yes/No	Yes/No

Notes:

No*—Indicates that the subsequently issued transaction is not allowed to complete before the previous transaction to preserve ordering in the system.*

Yes*—Indicates that the subsequently issued transaction must be allowed to complete before the previous transaction or deadlock may occur. Reasons for all eight Yes entries are given in the PCI specification. However, it indicates that the four Yes entries in the first row are only required for backward compatibility with earlier revisions of the specification. For the PMW/DRR and PMW/DWR cases, there is an additional reason—they are required to be Yes because the fourth row requires DRCs to be able to pass DRRs and DWRs. In the case where a DRR or DWR occurs followed by a PMW and followed by a DRC, the PMW must pass the DRR/DWR in order to allow the DRC to do so, because the DRC may not pass the PMW.*

Yes/No*—Indicates the subsequently issued transaction may be allowed to complete before the previous transaction. There are no ordering requirements between the transactions.*

~~In addition, PCI supports the concept of performing multiple system reads due to a single PCI bus read in anticipation of that data being needed soon—this is called PCI *prefetching*. Within a prefetch, reads are required to be strongly ordered in ascending order by address.~~

~~Moved to command mapping section: PCI is also capable of generating operations with discontiguous byte masks. If this occurs for read requests that cross aligned doubleword boundaries, they must be broken on doubleword boundaries into multiple transactions in the HyperTransport protocol. Similarly, write requests with discontiguous byte masks must be broken at 32-byte boundaries. In this case, the requests must be strongly ordered in ascending order by address.~~

~~Write requests with contiguous byte masks must be broken at 64-byte boundaries and strongly ordered in ascending order by address.~~

B.2.2 Command Mapping

Table 65~~Table 58~~ shows the mapping of PCI transaction types to HyperTransport ~~technology~~ packet types and Table 66 shows the mapping in the opposite direction.

Table 65. PCI Transaction Mapping to HyperTransport~~™ Technology~~ Packets

PCI Transaction Type	**HyperTransport~~™ Technology~~ Packet Type**
Posted Memory Write (PMW)	WrSized, Posted, PassPW = 0
Delayed Read Request (DRR)	RdSized, PassPW = 0, RespPassPW = 0
Delayed Write Request (DWR)	WrSized, Nonposted, PassPW = 0
Delayed Read Completion (DRC)	RdResponse, PassPW = 0 (from request packet)
Delayed Write Completion (DWC)	TgtDone, PassPW = 1 *

Note:

**—In some cases, the PassPW bit must be clear in a TgtDone. See Sections F.2.1.1 and F.2.5 for ~~one~~ some examples.*

~~PCI Read Line and Read Multiple requests that cause prefetches across a 64-byte boundary use sequence tagging, with each PCI Read being assigned a different non-zero sequence ID. Nonposted PCI operations with discontiguous byte masks that get broken into multiple operations in the HyperTransport protocol must also be tagged as part of a single sequence.~~ All ~~other~~ PCI requests use a sequence ID of 0.

PCI is ~~also~~ capable of generating operations with discontiguous byte masks. If this occurs for read requests that cross aligned doubleword boundaries, they must be broken on doubleword boundaries into multiple transactions in the HyperTransport protocol. Similarly, write requests with discontiguous byte masks must be broken at 32-byte boundaries. ~~In this case, the requests must be strongly ordered in ascending order by address.~~

Configuration and I/O requests must be broken on doubleword boundaries.

Write requests with contiguous byte masks must be broken at 64-byte boundaries~~ and strongly ordered~~. Posted writes that are broken into multiple HyperTransport requests must be issued in ascending order by address.

Table 66. HyperTransport Packet Mapping to PCI Transactions

HyperTransport Packet Type	**PCI Transaction Type**
WrSized to Memory Space[1]	Posted Memory Write (PMW)
WrSized to Configuration or I/O Space	Delayed Write Request (DWR)[2]
RdSized of Memory, Configuration, or I/O Space	Delayed Read Request (DRR)[2]
RdSized of Legacy PIC IACK Space	Interrupt Acknowledge
RdResponse	Delayed Read Completion (DRC)
TgtDone	Delayed Write Completion (DWC)

Notes:

1. A nonposted write to memory space will still result in a posted write on the PCI bus. The HyperTransport-to-PCI bridge must respond with a TgtDone after all data phases for the write have completed.

2. While the decision to delay a request is actually made by the target, these cycles cannot be posted and must result in a RdResponse or TgtDone on the HyperTransport Link after their completion.

~~B.2~~B.3 AGP

B.3.1 Ordering

These ordering rules are taken from the *Accelerated Graphics Port (AGP) Interface Specification, Revision 2.0*, Section 3.4. See that specification for more information.

AGP essentially consists of three separate channels, each with its own distinct ordering rules. No ordering is maintained between the three channels—traffic is completely independent. First, AGP contains a modified PCI channel, which maintains PCI ordering. The other two channels are called the high-priority (HP) and low-priority (LP) AGP channels.

The ordering rules presented here for reads are somewhat different from what appears in the AGP specification. That document defines ordering between reads in terms of the order that data is returned to the requesting device. We are concerned here with the order in which the reads are seen at the target—the I/O bridges can reorder returning read data if necessary. This leads to a slightly relaxed set of rules.

~~B.2.1~~B.3.1.1 HP AGP Ordering Rules

1. Writes may not pass writes.

~~B.2.2~~B.3.1.2 LP AGP Ordering Rules

1. Reads (including flushes) may not pass writes.
2. Writes may not pass writes.
3. Fences may not pass other transactions or be passed by other transactions.

AGP may also generate requests with discontiguous byte masks, with the same rules as PCI.

B.3.2 Command Mapping

The three channels of AGP are all completely independent as far as ordering is concerned, so (for optimal performance) a HyperTransport Link-to-AGP I/O bridge should assign each of these I/O streams to a separate UnitID.

The PCI channel of AGP uses the PCI mapping listed in Table 65~~Table 58~~.

The LP and HP channels never accept requests, so there is no need to specify the ordering of returning responses with respect to requests.

Table 67~~Table 59~~ shows the mapping of HP AGP transaction types to HyperTransport ~~technology~~ packet types.

Table 67. HP AGP Transaction Mapping to HyperTransport~~™ Technology~~ Packets

HP AGP Transaction Type	HyperTransport~~™ Technology~~ Packet Type
HP Write	WrSized, Posted, PassPW = 1
HP Read	RdSized, PassPW = 1, RespPassPW = 1

HP writes are placed in the posted request channel, while reads are placed in the nonposted request channel. Within each of these virtual channels, a single sequence ID is used to force the traffic to remain strongly ordered.

The PassPW and RespPassPW bits are set for reads because they are independent of the write traffic in the channel. The PassPW bit for writes does not matter in a pure HP AGP channel, because all the posted writes in the channel are strongly ordered due to the sequence ID anyway. But, if traffic from this channel were ever mixed with another I/O stream, having it set would minimize the interaction between the two.

There are two possible mappings of LP AGP traffic into the HyperTransport protocol. The first puts all traffic in the HyperTransport protocol nonposted channel as shown in Table 68~~Table 60~~.

Table 68. Simple LP AGP Transaction Mapping to HyperTransport~~™ Technology Packets~~

LP AGP Transaction Type	HyperTransport~~™ Technology~~ Packet Type
LP Write	WrSized, Nonposted, PassPW = 1
LP Read	RdSized, PassPW = 1, RespPassPW = 1
LP Flush	None (wait for all outstanding writes to complete)
LP Fence	None

All transfers in the low-priority channel are in the same virtual channel (nonposted requests), and they are all assigned to the same non-zero sequence ID, which keeps them strongly ordered. While this is a stronger ordering rule than required by AGP, it is sufficient. Since all transactions are strongly ordered, there is no need to do anything with a fence request.

Even though LP Writes are not posted in this mapping into the HyperTransport protocol, they can still be posted from the AGP point of view. The transaction can complete on the AGP bus without waiting for TgtDone in the HyperTransport protocol. However, the I/O bridge must remember that TgtDone is outstanding and not retire the buffer or SrcTag until it is received. Since the writes are not posted, there is also no need to issue an explicit HyperTransport technology flush packet. The I/O bridge can simply wait for TgtDone to be received for all outstanding writes and then complete the flush operation on the AGP bus.

The values of the PassPW and RespPassPW bits do not matter in this mapping of a pure LP AGP channel, because there are no posted writes in this channel in either direction in the HyperTransport protocol. However, if the traffic in this channel were ever to be combined with another I/O stream, setting them both would minimize the interactions with that stream.

The second mapping of LP AGP onto the HyperTransport protocol as shown in Table 69~~Table 61~~ puts LP writes in the posted channel:

Table 69. Alternate LP AGP Transaction Mapping to HyperTransport™

LP AGP Transaction Type	HyperTransport~~™ Technology~~ Packet Type
LP Write	WrSized, Posted, PassPW = 0
LP Read	RdSized, PassPW = 0, RespPassPW = 1
LP Flush	Flush, PassPW = 0
LP Fence	None (wait for all outstanding read responses)

No use of nonzero sequence IDs is required. Ordering between LP writes is maintained by the fact that they are in the posted channel with their PassPW bits clear. LP reads are prevented from passing LP writes for the same reason. Flush operations use the HyperTransport technology flush packet. Fences still do not result in HyperTransport technology packets being sent, but they do require action in this mapping. Because no operation can pass a write, fences only need to be concerned with preventing other operations from passing reads. Therefore, they can be implemented by stalling all subsequent requests until responses have been received to all outstanding read requests.

As above, the value of RespPassPW on reads is not important in a pure LP AGP channel, but setting it may ease some interaction problems in a mixed channel.

B.4 PCI-X

B.4.1 Ordering

These ordering rules are taken from the PCI-X specification, rev 1.0a. See that specification for more information.

Table 70. PCI-X Transaction Ordering Rules

Row Pass Column?	**Posted Memory Write (PMW)**	**Split Read Request (SRR)**	**Split Write Request (SWR)**	**Split Read Completion (SRC)**	**Split Write Completion (DWC)**
PMW	RO = 0: No RO = 1: Yes/No	Yes	Yes	Yes	Yes
SRR	No	Yes/No	Yes/No	Yes/No	Yes/No
SWR	No	Yes/No	Yes/No	Yes/No	Yes/No
SRC	RO = 0: No RO = 1: Yes/No	Yes	Yes	Yes/No	Yes/No
SWC	Yes/No	Yes	Yes	Yes/No	Yes/No

Notes:

*1. The definitions of **Yes**, **No**, and **Yes/No** are the same as in the PCI Ordering Rules table. The rows and columns here map exactly onto the corresponding ones in the PCI table, with Split Reads and Writes being similar to Delayed Reads and Writes.*

***RO** is the PCI-X Relaxed Ordering bit.*

PCI and PCI-X ordering rules are very similar, with the exception of the Relaxed Ordering bit (RO in the table), which when set allows posted writes to be passed by other posted writes or read completions.

B.4.2 Command Mapping

Table 71 shows the mapping of PCI-X transaction types to HyperTransport packet types and Table 72 shows the mapping in the opposite direction.

Table 71. PCI-X Transaction Mappings to HyperTransport Packets

PCI-X Transaction Type	**HyperTransport Packet Type**
Posted Memory Write (PMW)	WrSized, Posted, PassPW = **RO**, Coherent = !No Snoop
Split Write Request (SWR)	WrSized, Nonposted, PassPW = 0, Coherent = 1
Split Read Request (SRR)	RdSized, PassPW = 0, RespPassPW = **RO**, Coherent = !No Snoop
Split Write Completion (SWC)	TgtDone, PassPW = 1[2]
Split Read Completion (SRC)	RdResponse, PassPW = **RO**

Notes:

*1. **RO** is the PCI-X Relaxed Ordering bit.*

2. In some cases, the PassPW bit must be clear in a TgtDone. See Sections F.2.1.1 and F.2.5 for some examples.

3. The '!' character indicates logical inversion.

All PCI-X transactions use a sequence ID of 0. Similarly to PCI, PCI-X may also have requests with discontiguous byte enables, which may need to be broken in multiple transactions following the same rules as for PCI. The only exception is that Posted writes with the Relaxed Ordering bit set do not need to be issued in ascending address order. As in PCI to HyperTransport packet mapping, PCI-X Configuration and I/O requests must be broken at doubleword boundaries.

Table 72. HyperTransport Packet Mappings to PCI-X Transactions

HyperTransport Packet Type	**PCI-X Transaction Type**
WrSized to Memory Space[2]	Posted Memory Write (PMW), **RO** = PassPW[1]
WrSized to Configuration or I/O Space	Split Write Request (SWR)[3]
RdSized	Split Read Request (SRR)[3], **RO** = RespPassPW[1]
RdResponse	Split Read Completion (SRC), **RO** = PassPW
TgtDone	Split Write Completion (SWC)

Notes:

*1. **RO** is the PCI-X Relaxed Ordering bit. If the Enable Relaxed Ordering bit in the PCI-X Command register is cleared, the RO bit must not be set in requests, regardless of the value of PassPW or RespPassPW.*

2. A nonposted write to memory space will still result in a posted write on the PCI-X bus. The HyperTransport to PCI-X bridge must respond with a TgtDone after all data phases for the write have completed.

3. While the decision to split a request is actually made by the target, these cycles cannot be posted and must result in a RdResponse or TgtDone on the HyperTransport Link after their completion.

4. The "No Snoop" bit in PCI-X requests from a HyperTransport bridge is always 0.

C Summary of Deadlock Scenarios

Several scenarios have been mentioned throughout this specification that could cause deadlocks in HyperTransport systems. This appendix summarizes them and provides more information about their possible causes and how they are avoided.

C.1 Reflection/Forwarding Loops

C.1.1 Problem

In order to provide the same producer/consumer ordering behavior as PCI, peer-to-peer requests must be reflected through the host of a HyperTransport chain. This results in the host creating dependencies in upstream traffic upon downstream traffic when the host reissues upstream requests as downstream requests. If any other device in the chain also reflects traffic in the same virtual channel, a loop forms and if the number of packets in that channel exceed the amount of buffering available, the loop deadlocks because traffic can no longer be drained from the loop to make room for new traffic. An example, illustrated in Figure 9, is a device relying on host memory to sink traffic. In this example, downstream traffic targeting the reflecting device is reissued upstream to the host, making progress of downstream traffic through the reflecting device dependent upon the upstream flow of traffic out from the reflecting device. The loop forms when a source device at the bottom of the chain has its upstream traffic reflected downstream by the host for peer-to-peer access. If the relecting device is unable to sink all the traffic from the host, the peer-to-peer traffic will get back up the downstream channel, into the host, and then backup the upstream channel. At this point, the peer-to-peer traffic and reflected traffic are blocking each other and the system deadlocks. An even worse case is when the peer-to-peer traffic is targeting the reflecting device, in which case the reflecting device must sink all the traffic from the host in addition to all the peer-to-peer traffic targeting it to avoid a deadlock.

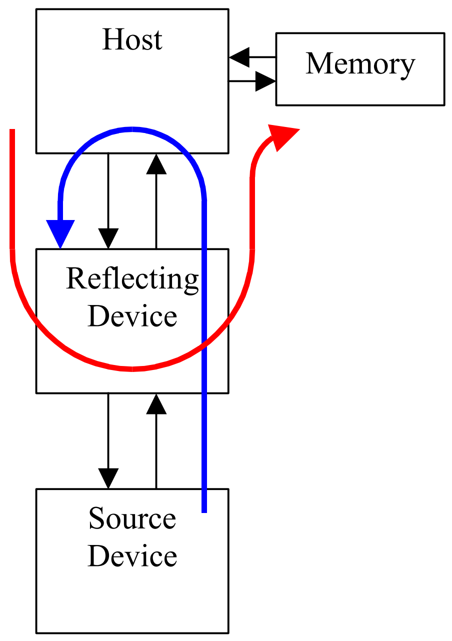


Figure 9. Reflection Example

Double-hosted chains can have the same problems, unless all requests are issued in the direction of the master host. This is because requests directed down to the slave host may be reflected upstream if the slave host does not accept them. Combining this reflected traffic with peer-to-peer traffic reflected off the master host creates a loop than can result in deadlock, as mentioned in Section 4.1.1.

C.1.2 Solution 1: Avoidance

Section 4.7 avoids these deadlocks by placing these requirements on devices:

- Do not make acceptance of a request dependent upon the ability to issue another request.
- Do not make acceptance of a request dependent upon receipt of a response.
- Do not make issuance of a response dependent upon the ability to issue a request.
- Do not make issuance of a response dependent upon receipt of a response.

In order to comply with these requirements, whatever operations the reflecting device would have performed before sending data upstream to host memory must instead be performed by the host. Some example operations are address translation, such as the Graphics Aperture Remapping Table for Advanced Graphics Port support, and data modification. This solution may also result in different ordering behavior than the actual reflection by the device would have exhibited.

The above rules do not apply to the host bridge of a chain. In a double-hosted chain, only one of the hosts may violate the rules to provide peer-to-peer reflection as the host bridge. The other host must have its ActAsSlave bit set and follow the rules.

C.1.3 Solution 2: Switching Channels

If the host, reflecting device, and all devices (if any) between them support Isochronous flow control (see Section D.1), and the traffic to be reflected and peer-to-peer traffic flowing within the chain is not Isochronous, then reflected normal traffic can be sent in the Isochronous channels.

C.1.4 Solution 3: Modifying Requests

A less costly solution that is possible in some systems is to modify requests flowing through the host and reflecting device to avoid the deadlock.

1. Define a "Reflection Region" known to the host and reflecting device, likely an address base/limit register pair.

2. The host forces all downstream requests to the reflection region to be nonposted, allowing all outstanding requests to be accounted for. This is only deadlock-free for requests originating from the host, not from other chains.

3. The reflecting device only responds to the downstream request after the upstream reflection has completed. Writes can be reflected as posted requests, allowing the response to the downstream request to be issued as soon as the upstream cycle is issued.

4. The host is programmed with the buffer depth of the reflecting device and will not issue more requests to the reflection region than the reflecting device can handle.

5. The reflecting device monitors upstream traffic and if the address of an upstream request is within the reflection region, the reflecting device modifies the request before forwarding it, such that the resulting request no longer targets the reflection region and instead targets host memory, removing the request from the deadlock loop. The ordering of the modified packets must not be affected by the modification. There cannot be any devices between the host and the reflecting device initiating peer-to-peer requests to the reflection region.

   An alternative solution for reflecting devices in systems where there is no peer-to-peer traffic other than to the reflection region is to convert upstream posted writes to the reflection region to nonposted writes. This allows them to be counted and throttled, such that the reflecting device can ensure that it has enough resources to sink all requests from the host and all peer-to-peer accesses to the reflection region without blocking the non-posted buffers in the loop between the host and the reflecting device. Note that the converted upstream writes will need to be issued with a UnitID of the reflecting device, and with unique SrcTags issued from the pool of tags available to the reflecting device.

6. Only accesses to memory or I/O space may be reflected. The reflections must target memory space.

C.2 Packet Issue and Acceptance

Section 4.8 contains two requirements for deadlock avoidance: control/data buffer dependency and response buffer dependency.

C.2.1 Control/Data Buffer Dependency

A control packet that has an associated data packet cannot be issued unless both control and data buffers exist for accepting the packet. Because data packets for different command packets cannot be distinguished on the link, only one command with associated data is allowed to be issued at one time. If a command having an associated data packet were issued when no data buffer was free to accept it, no other commands with associated data could issue, either. This creates a new dependency of the later command with data upon the first. If both commands are in the same virtual channel, this is not a problem, however, if the subsequent command is required to pass the first according to the ordering rules of Section 6.1, this can result in a deadlock.

For example, a nonposted command buffer, a posted command buffer, and a posted data buffer could be free. If a device issued a nonposted write command, and then a posted write needed to be sent, the posted write would be blocked until the nonposted data buffer became free. Because the posted write is now stuck behind the nonposted write, the deadlock described in Section C.5.1 could occur.

C.2.2 Response Buffer Dependency

A device cannot issue a nonposted request unless the device can accept the response that will result. If a device issued more nonposted requests upstream than it had resources available to sink the resulting downstream responses, then the downstream response buffers throughout the chain above the device would fill, preventing the flow of both host and peer-to-peer responses, which in turn would result in a deadlock.

C.2.3 Posted Request Acceptance

A device must be able to sink posted requests targeting it without other dependencies. Posted requests must be able to make forward progress in the system or they will block all traffic, due to ordering rules that do not allow other packets to pass posted writes.

C.3 Legacy Buses

As described in Section E.1, ISA and LPC both present the problem that once a transaction begins, no other transactions can begin until the prior transaction completes. This results in two possible deadlock scenarios:

C.3.1 Host/DMA Deadlock

If an ISA or LPC bus master (or the legacy DMA controller itself) initiates a read of host memory (or any nonposted upstream request) and the host issues a posted write (or any posted cycle) to the ISA or LPC bus before the read response returns to the ISA/LPC device, the read response can become stuck behind the posted write. The preferred solution is to require all requests sent to the ISA/LPC bus to be nonposted. Because responses are required to be able to pass nonposted requests, the deadlock cannot occur. Requests to the ISA/LPC bus are identified through both positively decoded spaces known to be allocated to the legacy bus, and any requests that have a subtractively decoded destination (compat bit set).

C.3.2 Peer-to-Peer Deadlock

Because the ISA or LPC bus is unable to sink any requests while it waits for a response to its own DMA requests, it is possible for the downstream nonposted request channel to fill, which in turn will cause upstream nonposted peer requests to the host to block, which will prevent the ISA/LPC bridge from making forward progress on its own nonposted requests. The solution to this issue is for the host to limit the number of requests it makes to the ISA/LPC bus to a known number of requests (typically one) that the bridge can sink. Because the host cannot limit peer requests without eventually blocking the upstream nonposted channel (and causing another deadlock), no peer requests to the ISA/LPC bus are allowed.

Peer requests to devices below the ISA/LPC bridge on the chain (including other devices in the same node as the ISA/LPC bridge) cannot be performed without deadlock unless the ISA/LPC bridge sinks the abovementioned known number of requests without blocking requests forwarded down the chain (or to other devices within the same node). This can be implemented with a buffer (or set of buffers) in the bridge node reserved for requests targeting the bridge, separate from the buffering for other requests.

C.4 System Management

The system management controller (typically located in the South Bridge or I/O Hub device) must be able to sink system management messages regardless of other traffic (such as accesses to legacy buses), as required in Section F.2. If the SMC did not sink SM messages without dependencies, then the downstream posted channel would fill, blocking all other traffic and could cause many of the deadlock scenarios described throughout this appendix. This is a specific case of the above rule that posted request must be able to make forward progress in the system.

Section F.2.3 describes a power management/throttling deadlock between the host and the SMC. Normally, after sending a STPCLK assertion message, the SMC is required to wait for a STOP_GRANT message from the host and send a STPCLK deassertion message before sending and any subsequent STPCLK assertion messages. If the SMC sends an unsolicited STPCLK assertion message for throttling, and receives a nonposted request to initiate a system state transition requiring a subsequent STPCLK assertion, the SMC must issue the subsequent STPCLK assertion without waiting for a STOP_GRANT. This is because the host will not respond to throttling STPCLK assertions while waiting for a state transition to occur.

C.5 PCI Requirements

Because HyperTransport ordering rules share many of the requirements of PCI ordering rules, there are certain requirements inherited from PCI for deadlock avoidance.

C.5.1 Posted Requests Must Pass Nonposted Requests

As described in the *PCI Local Bus Specification*, Revision 2.2, Appendix E, rule 5, posted requests must be allowed to pass stalled nonposted requests to avoid deadlocks. Referring to Figure 10, one example deadlock scenario involves two bridges (X and Z) that do not support delayed (split) transactions on either side of a bridge that does (Y).

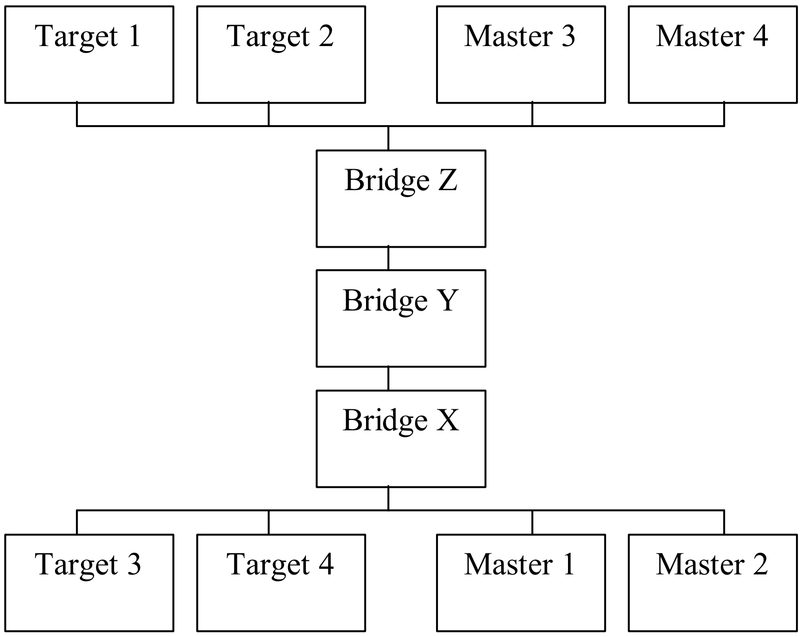


Figure 10. Example PCI System

Assume that reads have been issued from Master 1 to Target 1 and from Master 3 to Target 3, and both are delayed by bridge Y. Masters 2 and 4 issue a series of writes to Targets 2 and 4, respectively, which are posted in bridges X and Z. In order for either bridge X or Y to complete the reads and comply with the producer/consumer ordering model requirement that read responses cannot pass posted writes, they must flush their posted writes to bridge Y. Since bridge Y has limited resources for buffering the writes and the number of writes is unlimited, bridge Y must be allowed to reorder the posted writes past the reads or the system will be unable to make progress.

Substitute a HyperTransport chain for bridge Y, and this scenario becomes immediately relevant to all HyperTransport devices. Additionally, following this rule throughout a HyperTransport chain allows the posted channel to carry requests that need to be assured of forward progress, such as interrupts and system management messages.

C.5.2 Responses Must Pass Nonposted Requests

Rule 6 of PCI Appendix E requires responses (delayed completions) to be able to pass nonposted (delayed) requests to avoid a deadlock when two devices capable of split (delayed) transactions send nonposted requests to each other. For example, if Bridge A sends nonposted request 1 down to the secondary bus and bridge B sends nonposted request 2 up to the primary bus, the response (completion) of transaction 1 cannot make upstream progress if it is stuck behind request 2, which cannot make progress because response 2 is stuck behind request 1.

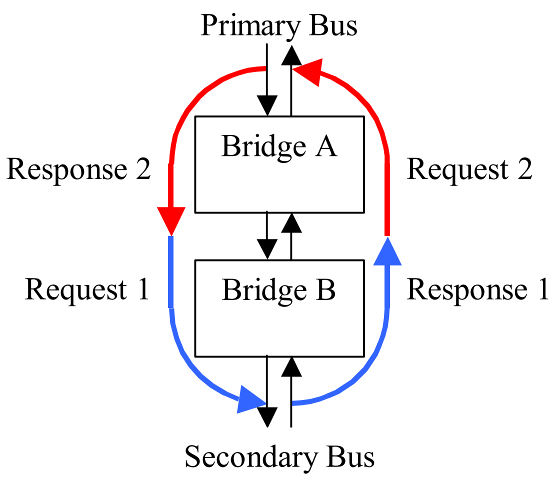


Figure 11. Request/Response Deadlock Loop

C.5.3 Posted Requests Must Pass Responses

PCI Rule 7 requires posted requests to be able to pass responses (delayed completions) and again refers to Figure 10 for an example. Assume that bridge Y has responses at the head of its queues in each direction, bridge X is full of downstream posted write data, and bridge Z is full of upstream posted write data. Bridges X and Z cannot complete their posted writes and retry their reads to accept the responses once bridge Y fills with posted write data in both directions (again we assume that bridge Y has limited resources for storing requests and the masters providing X and Z with data are not limited), and bridge Y cannot accept more posted write data until it completes its reads and delivers the responses, unless it is allowed to reorder the posted writes past the responses.

D Considerations for Isochronous Traffic

A problem relative to bounding latency for Isochronous traffic is the presence in the system of a very slow I/O device that is the target of posted requests. This situation can cause other requesters in the system to experience large and unpredictable latencies. Since HyperTransport™ technology devices are not required to reorder responses, downstream responses with PassPW set can get stuck behind responses with PassPW clear, which are in turn stuck behind the posted requests. In addition, peer-to-peer traffic directed at the slow device can back up the upstream posted channel as well.

The most complete ~~(and costly)~~ solution to this problem would be to implement Isochronous flow control mode as defined in Section D.1 However, a simpler solution may be possible for most systems and is defined in Section D.2.

D.1 Isochronous Flow Control Mode (Optional)

In Isochronous (ISOC) flow control mode, there are two classes of service defined. The high-priority service class is intended to support Isochronous traffic, and the low-priority service class is intended for all other traffic. The following HyperTransport technology features support the high-priority service class:

- Dedicated posted command, nonposted command and response virtual channels—the ISOC virtual channels.
- Dedicated flow control buffers in support of the ISOC virtual channels.
- The Isoc bits in the read and write command fields identify commands that should travel in the ISOC channels.
- The Isoc bit in the read response and target done packets identify responses that should travel in the ISOC response channels.
- The Isoc bit in the NOP packet identifies buffer release packets for the ISOC virtual channels.
- Broadcast, Fence, ~~and~~ Flush, and Atomic RMW packets do not travel in or affect the ISOC virtual channels.

The following rules govern device operation in ISOC flow control mode:

1. All devices ~~enter normal (non-~~disable ISOC~~)~~ flow control mode after a cold reset. Software may enable ISOC flow control ~~mode operation~~ and sequence the chain through a warm reset to ~~initiate the transition to~~enable ISOC flow control mode. See Section 7.5.4.9 for details.
2. There are no ordering constraints between transactions in the ISOC and non-ISOC channels. Furthermore, ISOC traffic is invisible to the fairness algorithm implemented for non-ISOC traffic required by Section 4.9.5.
3. The ordering constraints for transactions within in the ISOC channels are identical to those for transactions within the non-ISOC channel, as defined in Chapter 6.

4. High-priority traffic must always be serviced before low-priority traffic, and there is no guarantee against high-priority traffic starving low-priority traffic, although it is expected that the total ISOC bandwidth would never exceed the overall available bandwidth. This eliminates the need for a fairness algorithm (like that in Section 4.9.5) to regulate the insertion of ISOC traffic.
5. ISOC flow control is enabled on a per-link basis to allow ISOC requests and responses to "tunnel" through non-ISOC devices on a chain.

It is intended that Isochronous sources generate requests with the Isoc bit set in order to get service from the system with deterministic worst-case latency. The actual latency and bandwidth guarantee for ISOC requests is system-dependent and outside the scope of this specification.

D.2 Normal Flow Control Mode

There are ~~two observations that together may make ISOC mode unnecessary, and that can be leveraged to build simple bounded systems with more deterministic latencies:~~

~~No known devices use the programming paradigm that is enabled by the PCI ordering rule in which downstream responses are not allowed to pass downstream posted requests.~~

~~Devices that generate peer-to-peer traffic are extremely uncommon.~~

~~In light of these observations, for~~ systems in which Isochronous traffic is an important consideration, but ~~in which~~ the cost of ISOC flow control mode is prohibitive. If these systems do not require the ordering rule in which downstream responses are not allowed to pass downstream posted requests and do not require peer-to-peer traffic on the same chain as Isochronous traffic, the following steps can be taken.

1. Build an operating mode in which the host sets the PassPW bit in all downstream responses. ~~This violates the PCI's downstream ordering rule, but is not believed to be a material issue.~~
2. Build HyperTransport technology implementations in which responses with PassPW set will pass stalled posted requests.
3. Do not populate Isochronous devices on the same HyperTransport chain as devices that generate peer-to-peer traffic.

Normal ~~(non-ISOC)~~flow control mode is characterized as follows:

- There are no dedicated ISOC virtual channels.
- The Isoc bit in the NOP packet must be 0.

The Isoc bit in the read, write, RdResponse, and TgtDone commands may be either set or cleared. ISOC commands may be used in normal mode in simple, bounded systems in order to get lower latency service for Isochronous sources. The rules which govern the system's behavior on behalf of Isoc requests in ~~non-ISOC~~normal mode and the latency assurances provided to those requests are platform-specific and outside the scope of this specification. In ~~non-ISOC~~normal mode, HyperTransport technology devices may ignore the Isoc bit, but must preserve it so that ISOC requests and responses may be handled properly by ISOC devices on either side of non-ISOC devices. If any device (including non-ISOC devices) receives a nonposted request with the Isoc bit set, the Isoc bit must be set in the response.

E Southbridges and Compatibility Buses

This appendix provides some considerations for including compatibility buses (such as ISA or LPC) and Southbridges in systems based on HyperTransport™ technology.

E.1 ISA/LPC Deadlock Case

A system that contains an ISA or Low Pin Count (LPC) DMA controller or supports ISA/LPC bus masters has a particular deadlock scenario that needs to be addressed. Since ISA and LPC do not support retry, a downstream posted write could block a response to a nonposted request from the ISA or LPC bus, causing deadlock.

One solution to this problem is for the host to decode programmed I/O requests to the ISA/LPC memory range and emit all such requests in the nonposted channel. Alternatively, the host could avoid the implementation of a positive decode for the ISA/LPC memory range and emit all default requests (those with the Compat bit set) in the nonposted channel. This solution does not permit peer-to-peer requests to be issued from or to ISA/LPC devices, since such requests may result in deadlock.

There is an additional source of deadlocks involving nonposted peer-to-peer cycles in the same chain as the ISA/LPC bridge. If the host issues multiple outstanding nonposted requests to the ISA/LPC bus, and the ISA/LPC bridge is not able to buffer them all, the nonposted flow-control buffers could fill up downstream. Nonposted peer-to-peer cycles going upstream would not be able to complete, filling upstream nonposted buffers. At this point, nonposted requests issued upstream on behalf of the ISA/LPC bus could not complete, and because ISA and LPC do not support retry, the host-issued nonposted requests will never complete, either, causing a deadlock.

The solution is to either not allow nonposted peer-to-peer activity on the same chain as an ISA/LPC bridge, or only allow the host to issue a single outstanding request to ISA/LPC or multiple outstanding requests to other spaces, but not a mix of the two. Furthermore, the host must continue to service the peer-to-peer requests in either case, or the upstream nonposted buffers could fill and create a deadlock. Because the host will only issue one request to ISA/LPC at a time, and they must not interfere with peer-to-peer cycles, the SeqID for requests to ISA/LPC will always be 0.

E.2 ISA/LPC Write Post Flushing

Some ISA or LPC bridges require the ability to know when all posted writes they have issued are guaranteed to be globally visible. This requirement is typically handled by having a WSC# (Write

Snoop Complete) pin on the Southbridge, which is asserted whenever all previously posted writes are guaranteed to be visible to all processors.

No direct HyperTransport protocol support is required for a HyperTransport-to-PCI bridge to implement the WSC# bit. The HyperTransport-to-PCI bridge can follow each posted write from the ISA/LPC bridge with a HyperTransport technology flush request. The response to the flush guarantees that the posted write is globally visible. The HyperTransport-to-PCI bridge can then keep its WSC# pin asserted whenever it has no flushes outstanding.

E.3 Subtractive Decoding

This section provides some considerations for including a subtractive decoding device or bridge in a system based on HyperTransport technology.

E.3.1 Subtractive Decode in the General Case

Since HyperTransport technology devices in a chain do not sit on the same bus, they cannot normally perform subtractive decode by waiting to see which requests are not responded to by other devices on the chain. Subtractive decoding devices and bridges are supported in HyperTransport technology using the Compat bit. All hosts connecting to HyperTransport I/O chains are required to have registers that specify positive decode ranges for all HyperTransport technology I/O devices and bridges. One of these I/O chains may also include a subtractive bridge (potentially to a PCI, ISA, or LPC bus). Requests that do not match any of the positive ranges are routed to the HyperTransport I/O chain containing the subtractive bridge (the *compatibility chain*) with the Compat bit set. The Compat bit indicates to the subtractive bridge that it should claim the request, regardless of address. Requests that are within the positively decoded ranges of the compatibility HyperTransport I/O chain do not have the Compat bit set and are passed down the chain to be detected by positively decoded devices and bridges, like any other HyperTransport I/O chain.

It is worth noting that a system which sets up the subtractive decode path in hardware can access memory and I/O spaces owned by the subtractive device without requiring software initialization of the link. This is true even though the devices on the HyperTransport chain have not had their UnitIDs programmed to nonzero values—the Compat bit will cause accesses to reach the subtractive device.

E.3.2 Subtractive Decode in x86 Legacy Systems

Some x86 systems may have legacy software considerations (such as Cardbus legacy compatibility and power-management requirements) that require the compatibility chain to be numbered as Configuration Bus Number 0. In such a system the host bridge that controls the compatibility chain must be identified with a device configuration header rather than a bridge header. This bridge still requires a positive decode range so that it knows whether to set the Compat bit for transactions that do not fall in this (or any other) range. Therefore implementation-specific range registers need to be defined for this bridge.

E.3.3 Subtractive Decode in the Simplest Case

Another way to support subtractive decode in small systems based on HyperTransport technology is to place the subtractive decode device on the end of a single-hosted HyperTransport chain. In that case, the subtractive device can safely assume that all requests that reach it are destined for it.

E.3.4 Subtractive Decode Behind a PCI Bridge

If the subtractive target on the HyperTransport chain is a bridge to PCI, there are several additional issues.

The most straightforward approach is to build a HyperTransport-to-PCI bridge that performs subtractive decode and implements a standard PCI bridge header. See the *PCI to PCI Bridge Architecture Specification, Revision 1.1*, for a description of subtractive decoding PCI to PCI bridges.

- The bridge performs subtractive decode using the Compat bit, as described above, for transactions that originate on the primary bus.
- The bridge performs positive decode for transactions that originate on the secondary bus. It forwards to its primary bus any transaction that originates on the secondary bus and does not fall inside the address ranges programmed into the bridge header. This implies that peer-to-peer transactions targeted at a subtractive decoding device on the secondary bus and sourced on either the secondary or subordinate buses are not supported.
- The secondary bus segment is by definition not Bus 0, because configuration software will encounter a bridge header and number the bus accordingly. This may not be compatible with some legacy software requirements.

Some legacy systems may require that the compatibility bus be Bus 0, which is not allowed to be behind another bridge. Therefore, another approach can be used that allows the PCI bus that contains the subtractive decode device to be configured as Bus 0. In this approach, the HyperTransport-to-PCI bridge implements a function header rather than a bridge header.

- The bridge performs subtractive decode for transactions that originate on the primary bus, using the Compat bit.
- The bridge claims all transactions that originate on the secondary bus and forwards them to the primary bus, but it does not do this subtractively. This implies that peer-to-peer transactions

that are targeted at devices on the secondary bus and sourced on either the secondary or subordinate buses are not supported.

- The primary interface of the bridge must be a device on Bus 0 so that configuration cycles will reach it.~~.~~

E.4 VGA Palette Snooping

The *PCI Local Bus* and *Bridge Architecture* s~~S~~pecifications define VGA palette snooping. This allows a device on the same bus as the device owning the VGA palette range or a bridge that forwards the VGA palette range to pick up write data as the access goes by.

No direct support for VGA palette snooping is provided in the HyperTransport protocol. It can be supported at a level above the HyperTransport protocol by designating an address range as an alias of the VGA palette range and by having the host bridge generate a posted write to the alias as well as the write to the original address. The snooping device must recognize the aliased write and translate it back to the VGA palette range before forwarding or operating on it. The details of this mechanism are implementation-specific.

F Required Behavior in x86 Platforms

This appendix specifies mandatory behavior of HyperTransport™ technology devices designed for x86 platforms.

While optional for some HyperTransport technology-enabled devices, the following features are required for devices in x86 platforms:

- LDTSTOP# pin
- LDTSTOP# Tristate Enable bit
- Support for LDTSTOP# disconnect sequences and the Discon bit of NOP packets
- LDTSTOP# is input-only to all devices except the system management controller (Southbridge)
- VGAEN and ISAEN bits in all bridges
- Upstream accesses to configuration space are illegal, therefore hosts will have the Host Hide bit (see Section 7.5.3.3.5) read-only as 1.

F.1 Interrupts

x86 HyperTransport technology systems can use interrupt requests (see Section 9.1~~5.1~~) instead of an APIC bus or discrete pins. Table 73 shows the format of x86 interrupt-request packets. Some Hypertransport devices (such as disk controllers or network adaptors) may still need discrete pins for use with legacy operating systems that do not support APIC or HyperTransport interrupt discovery and configuration for boot.

F.1.1 Interrupt Request

Table 73. x86 Interrupt Request Packet Format

Bit-Time	CTL	7	6	5	4	3	2	1	0
0	1	SeqID[3:2]		Cmd[5:0]					
1	1	PassPW	SeqID[1:0]		UnitID[4:0]				
2	1	Count[1:0]		Reserved					
3	1	MT[3]	DM	RQEOI	MT[2:0]			Count[3:2]	
4	1	IntrDest[7:0]							
5	1	Vector[7:0]							
6	1	Address[31:24]							
7	1	Address[39:32]							
8	0	IntrDest[15:8]							
9	0	IntrDest[23:16]							
10	0	IntrDest[31:24]							
11	0	Reserved							

Note:

Address [39:24] must be FDF8h in x86 systems.

There are three classes of interrupts supported in x86 HyperTransport technology systems:

- Arbitrated (Low Priority)
- Fixed
- Non-vectored

Arbitrated interrupts are only delivered to one of the addressed destinations within the host targeted by the interrupt. The ultimate target is either the lowest priority destination or a destination that is already servicing the same interrupt source (the focus processor). Arbitrated interrupts have 256 possible sources. Each interrupt source is identified by an 8-bit vector ID.

Fixed interrupts are delivered to all destinations addressed by the interrupt message. They can be used to send single, multicast, or broadcast interrupts. Fixed interrupts also have 256 possible sources, identified by vector ID.

Nonvectored interrupts do not carry source information. Therefore, the vector must be 00h. They must be delivered to all addressed destinations (DM=0, IntrDest=0xFF). They consist of the following types:

- SMI
- NMI
- INIT
- ExtInt (Legacy PIC)

The set of potential destinations is determined by the IntrDest and Destination Mode (DM) fields. The DM field determines if IntrDest represents a physical identifier or a logical identifier, as shown in Table 74.

Table 74. Destination Mode Bit Field Encoding

DM	Destination Mode
0	Physical
1	Logical

In Physical mode, IntrDest[31:8] must be 0, and each interrupt destination (processor) within the host is assigned a unique 8-bit physical ID. The physical ID 0xFF is reserved and is used to indicate that the interrupt should be broadcast to all possible destinations. A destination is considered a target for a physical mode interrupt if its ID matches IntrDest[7:0] or if IntrDest[7:0] equals 0xFF.

In Logical mode, each interrupt destination is assigned a 32-bit logical ID. The determination of what constitutes a valid logical ID is system-specific, and the method of comparison of logical ID to IntrDest[31:0] is programmable. For example, a system can choose a one-hot address representation, assigning one bit to each processor (limited to 32 processors), or it can define a portion of the logical address to be fully decoded and the rest of the bits to be one-hot encoded.

Not all x86 platforms support IntrDest[31:8]. See platform-specific documentation to determine if this feature is available.

Arbitrated and fixed interrupts can be edge-triggered or level-sensitive, as identified by the Trigger Mode (TM) field, carried in the RQEOI bit of the HyperTransport technology interrupt request. Edge-sensitive and level-sensitive interrupts cannot be mapped to the same vector.

Trigger Mode is encoded as shown in Table 75.

Table 75. Trigger Mode Bit Field Encoding

RQEOI	Trigger Mode
0	Edge
1	Level

Level-sensitive interrupts require an End of Interrupt (EOI) message (described below) to be transmitted to acknowledge the servicing of the interrupt. A subsequent level-sensitive interrupt using the same vector will not be sent until an EOI message has been received. Edge-triggered interrupts do not signal the servicing of the interrupt. Only the vector will be returned in the EOI.

Non-vectored interrupts are always edge-triggered and therefore no HyperTransport technology EOI is used.

The type of interrupt is identified by the Message Type (MT) field.

Table 76 summarizes the allowed combinations of these fields (all combinations not listed are reserved).

Table 76. Interrupt Request Bit Field Encoding Summary

MT[3:0]	Message Type	RQEOI	Vector	DM	Dest
0_000	Fixed	0 or 1	0–FFh	0	0–FFh
				1	0–FFFF_FFFFh
0_001	Arbitrated			0	0–FFh
				1	0–FFFF_FFFFh
0_010	SMI	0	0	0	FFh
0_011	NMI				
0_100	INIT				
0_101	Startup (Host Only)				
0_110	ExtInt	0	0	0	FFh
1_011	Legacy PIC NMI (LINT1)				
1_110	Legacy PIC ExtInt (LINT0)				
x_111	Reserved(EOI)				

Startup messages are used in interprocessor communication only. They are similar to fixed interrupts in that they carry a vector, but they have their own Message Type.

HyperTransport technology I/O host bridges must not combine multiple interrupt transactions into a single transaction within the host.

~~F.1.1~~F.1.2 Standard EOI

Table 77 shows the format of the EOI returned by the host to indicate that an interrupt request with RQEOI=1 has been serviced.

Table 77. Standard End-of-Interrupt (EOI) Format

Bit-Time	7	6	5	4	3	2	1	0
0	SeqID[3:2]		Cmd[5:0]					
1	PassPW	SeqID[1:0]		UnitID[4:0]				
2	Reserved							
3	Rsv			MT[2:0]=111b			Rsv	
4	Reserved							
5	Vector[7:0]							
6	Addr[31:24]							
7	Addr[39:32]							

~~F.1.2~~F.1.3 Legacy PIC (8259) Interrupt Request, Acknowledge, and EOI

The PIC is assumed to reside in the Southbridge. Interrupts are requested using the ExtInt message type as described above. The processor that services the interrupt request issues an interrupt acknowledge cycle to the Southbridge. An interrupt acknowledge transaction can be directed to the interrupt controller by performing a byte read within the reserved IACK range defined in Chapter 5~~9~~. Any read within this address range generates a RdSized request with the Compat bit set. This request packet is routed directly to the Southbridge if the Southbridge is a native HyperTransport technology device. If the Southbridge is implemented as a PCI device, then the request packet is routed to the intervening HyperTransport-to-PCI bridge. The bridge generates an interrupt acknowledge cycle on the PCI.

In both cases, the interrupt vector is returned in the eight least-significant bits of the RdResponse, independent of the byte masks in the RdSized request. The 24 most-significant bits are 0.

Even when legacy PIC interrupts are configured as level-sensitive, the HyperTransport technology interrupt request is sent as edge mode to indicate that a HyperTransport technology EOI is not used. EOI to the legacy PIC is performed as an I/O access to the PIC address, not a HyperTransport technology EOI.

In normal operation, the Legacy PIC will issue a single ExtInt request when an interrupt is pending and the host will issue an IACK request to determine the source of the interrupt and enable further interrupt requests. Once the PIC receives an IACK request, it will respond with the vector for the active interrupt. After servicing the interrupt, if this interrupt or another one still requires service, the PIC will issue another ExtInt request. The host must be able to accept a new ExtInt immediately after issuing an IACK request. Under certain conditions (such as a noisy or unusually frequent interrupt source), the PIC may issue multiple ExtInt requests between IACK requests. Proper host behavior in this case is to treat the multiple requests as a single interrupt service request, but must ensure that once an IACK request has been issued, further ExtInt requests will result in another IACK request.

~~F.1.3~~F.1.4 Alternate Interrupt Discovery and Configuration Mechanism

For compatibility with existing software, HyperTransport technology devices that generate interrupts may need to provide a memory-mapped version of the interrupt discovery and configuration register set in addition to the one described in Section 7.6. The memory-mapped register set is comparable to a standard IOAPIC register set, and the redirection table entries would have the layout shown in Table 78. This register set only provides legacy software an alternate way to access the relevant subset of the registers described in Section 7.6. It does not provide any new state or registers, only a second location in address space with fields rearranged for legacy compatibility. Bits that are not accessed through this mechanism are unaffected by it, and retain the values written to them by the configuration-space mechanism.

Table 78. Redirection Table Format

Bit	R/W	Reset	Description
63:56	R/W	0	IntrInfo[15:8] Destination
55:32	R/W	0	IntrInfo[55:32] Extended Destination: If a device does not support 32-bit destinations, this field is read-only 0.
31:17	R/O	0	Reserved. Note that IntrInfo[31:24] (Extended Address) and IntrInfo[7] (MT[3]) can only be accessed through the configuration mechanism detailed in Section 7.6.
16	R/W	1	Mask: When this bit is set, the interrupt is masked.
15	R/W	0	IntrInfo[5] Request EOI: If set, after each interrupt request is sent the device waits for the Waiting for EOI bit to be cleared before sending another interrupt.
14	R/O	0	Waiting for EOI: If RQEOI is 1, then this bit is set by hardware when an interrupt request is sent and cleared by hardware when the EOI is

Bit	R/W	Reset	Description		
			returned.		
13	R/W	0	Polarity: For ~~devices with~~ external interrupt ~~inputs~~sources, when this bit is set, the interrupt signal is active-low. If clear, the interrupt signal is active-high. For ~~devices without external pins~~internal interrupt sources, this bit is reserved.		
12	R/O	0	Reserved		
11	R/W	0	IntrInfo[6] Destination Mode: 0=Physical, 1=Logical		
10: 8	R/W	0	IntrInfo[4:2] Message Type[2:0]	**Encoding**	**Message Type**
			* Note that the register encoding for some messages is different from the encoding sent in the interrupt request.	000	Fixed
			The *Startup* message is only used for inter-processor communication, not for I/O devices.	001	Arbitrated
				010	SMI
				011	Reserved
				*100	NMI
				*101	INIT
				*110	Reserved *(Startup)*
				*111	ExtInt
7: 0	R/W	0	IntrInfo[23:16] Vector		

F.2 System Management

The system Southbridge is defined, in part, to include the platform system management logic that controls ACPI-defined and platform-specific system state transitions. In order to power-manage the system properly, the Southbridge is expected to reside on Bus 0. The Southbridge and host use HyperTransport technology system management messages to facilitate system state transitions.

Devices with both an ISA/LPC interface and system management logic (i.e., a Southbridge) must be able to accept downstream SM messages even when a ISA/LPC master is in control of the bus in order to maintain the correct virtual wire behavior and prevent deadlocks. This is because SM messages travel in the posted channel. If the Southbridge allowed the posted channel to back up, responses to reads of system memory executed on behalf of an ISA/LPC master would not be able to pass the SM messages.

x86-platform Southbridges are required to include BIOS-programmable configuration registers called system management action fields (SMAF). These specify the value for bits 3:1 of the STPCLK assertion system management message sent from the Southbridge to the host, based on the system state transition being executed. The Southbridge is required to provide separate BIOS-programmable SMAF registers for (1) each ACPI-defined state (as well as throttling) supported by the Southbridge, and (2) host-initiated Voltage ID/Frequency ID (VID/FID) changes. These registers are to be programmed by BIOS after boot, prior to any system state transitions from the fully operational state.

x86-platform HyperTransport technology devices monitor the SMAF value broadcast with the STOP_GRANT special cycle and take the appropriate power management actions based upon the SMAF value.

The Southbridge is required to control LDTSTOP# in support of VID/FID change. It may optionally be asserted during other system state transitions and HyperTransport link width or frequency changes as well. No other devices are allowed to control LDTSTOP#, and LDTSTOP# must not be asserted without a prior STOP_GRANT message, as described in the sequence below.

In the ACPI-defined S3, S4, and S5 states, RESET# is asserted and PWROK is deasserted.

All system state transitions and HyperTransport link width or frequency changes forced by LDTSTOP# follow this sequence:

1. The sequence starts with one of the following three methods: (1) the host accesses a Southbridge register (as is the case with ACPI-defined system and CPU sleep state transitions), (2) the host sends a VID/FID change system management cycle to the Southbridge, or (3) the Southbridge logic initiates the sequence without a HyperTransport technology transaction (as is the case with throttling).
2. The Southbridge responds by sending a STPCLK assertion system management message to the host with UnitID matching the UnitID of the response to the host access from step 1, if a response is required. Bits 3:1 of this message contain the SMAF value associated with the system state transition being executed. The host will broadcast the STPCLK assertion message down all chains.
3. After the STPCLK assertion message is sent to the host, the Southbridge may send the response to the initiating transaction from step 1, if a response is required, with the PassPW bit cleared. Such responses are required to follow the STPCLK assertion system management message to guarantee that the host does not execute any additional instructions after the initiating command of step 1, as is required by some operating systems.
4. The host is required to respond to the STPCLK assertion system management message by broadcasting a STOP_GRANT system management message down all chains. This is intended to indicate that the host is ready for the next step in the state transition.

Note: There may be an arbitrarily large delay from the STPCLK assertion message to the STOP_GRANT message. The Southbridge is required to wait for the STOP_GRANT system management message prior to sending a STPCLK deassertion system management message.

The following steps assume that RESET# is not asserted as part of the system power state transition; if RESET# is asserted, it must be asserted after the STOP_GRANT system management message is received by the Southbridge; the resume that occurs after the reset is as specified in Chapter 12. Note: There are platform level exceptions to the previously stated rules. For example, in response to an ACPI-defined Power Button Override event, a Thermal Protection event, or other mechanisms beyond the scope of this specification, there is a direct transition to S5 that skips the STPCLK/STOP_GRANT protocol.

5. The Southbridge may assert LDTSTOP# a system-specific time after the STOP_GRANT system management message is received, based upon SMAF value. The SMAF code mapping is beyond the scope of this specification. The delay between reception of STOP_GRANT and LDTSTOP# assertion should allow enough time for STOP_GRANT to have reached all other devices in the system. The Southbridge is required to assert LDTSTOP# if any of the following occurs:
   - A VID/FID transition is being executed.
   - The HyperTransport link width or frequency is being transitioned without RESET# assertion.
   - The ACPI defined C3 processor state is being entered.
   - S1 state is being entered.
   - The S3 state is being entered.
6. If LDTSTOP# was asserted, then it may be deasserted, as required by the system management logic.
7. After LDTSTOP# is deasserted, the Southbridge is required to send the STPCLK deassertion system management message to the host in order for the host to resume to the fully operational state. The host will broadcast that STPCLK message down all chains. The STPCLK deassertion message is not sent upon resume from any state in which PWROK was deasserted or RESET# is asserted. This covers S3, S4, S5, and G3 (mechanical off). The STPCLK deassertion message is sent to exit any STOP_GRANT state in which PWROK is not deasserted and RESET# is not asserted.

To meet platform power consumption requirements, devices in the system may need to gate clocks, stop PLLs, or power down portions of the design after LDTSTOP# assertion. A device is enabled to take these steps when it receives a STOP_GRANT cycle with a specific SMAF value prior to LDTSTOP# assertion.

In the event that a STOP_GRANT does not reach the device before LDTSTOP# assertion (perhaps due to unusual delays or a large system), the device will not take the additional power management actions. When a STPCLK deassertion is received, devices should purge the previous SMAF code that was not acted on to prevent a device from reacting to a "stale" STOP_GRANT SMAF code that is no longer valid. Reacting to a stale STOP_GRANT could result in a device taking an in-appropriate power management action. System-level mechanisms for ensuring that the STOP_GRANT SMAF is always recognized before LDTSTOP# is asserted are beyond the scope of this specification.

F.2.1 Command Encoding

For both upstream and downstream cases, the type of system management request (SysMgtCmd[7:0]) is encoded as shown in Table 79.

Table 79. System Management Request Command Encoding

SysMgtCmd	Command Type
0000 xxxx	Reserved
0001 xxxx	x86 legacy inputs to the processor. New state of signal: [0]: IGNNE [1]: A20M [2]: Reserved [3]: Reserved
0010 xxxx	x86 legacy output from the processor. New state of signal: [0]: FERR [3:1]: Reserved
0011 xxxx	[0]: STPCLK~~.~~ [3:1]: SMAF
0100 xxxx	SHUTDOWN [3:0]: Implementation-specific~~Reserved.~~
0101 xxxx	HALT [3:0]: Implementation-specific ~~Reserved.~~
0110 xxxx	STOP_GRANT [0]: Reserved [3:1]: SMAF~~.~~
0111 xxxx	VID/FID Change [3:0]: Implementation-specific ~~Reserved.~~
1000 xxxx	WBINVD [3:0]: Implementation-specific ~~Reserved.~~
1001 xxxx	INVD [3:0]: Implementation-specific ~~Reserved.~~
1010 xxxx	[0]: SMIACK [3:1]: Implementation-specific ~~Reserved.~~
1011 xxxx	x86 platform-specific functions.
11xx xxxx	Reserved

F.2.1.1 x86 Legacy Signals: Inputs to the Processor

The information associated with the x86 legacy signals is transported using system management packets in HyperTransport technology systems. The legacy signals that are inputs to processors are as follows:

- IGNNE
- A20M
- STPCLK

These packets originate from the SMC and are sent upstream to the host as a posted write. They will then be reflected down all HyperTransport I/O chains as a broadcast packet. For each bit, a 1 represents an assertion of the associated legacy pin, and a 0 represents a deassertion of that pin.

When the A20M or STPCLK virtual wire is changed as a result of a nonposted host request, the message signaling the change in state of the virtual wire must be perceived by the host before the response to the host request. In order to achieve this behavior, the following conditions must be met:

- The UnitID of the virtual wire message must match the UnitID of the response.
- The PassPW bit in the response must be clear, even if it is a TgtDone.
- The host must not execute any instructions beyond the one that requested the change in state until the side effects of the virtual wire message have occurred.

F.2.1.2 x86 Legacy Signals: Outputs from the Processor

The legacy signals that are outputs from processors are as follows:

- FERR
- SMIACK

These packets originate from the host and are broadcast downstream to all HyperTransport technology I/O devices in the system. For each bit, a 1 represents an assertion of the associated legacy pin, and a 0 represents a deassertion of that pin.

The legacy pin represented by SMIACK is asserted when the processor enters system management mode (SMM) and is deasserted when the processor exits SMM.

F.2.1.3 x86 Special Cycles

The special cycles carried by system management packets are as follows:

- HALT—Generated by processor in response to execution of a HALT instruction
- SHUTDOWN—Generated by processor in response to a catastrophic error
- STOP_GRANT—Generated by processor in response to a STPCLK assertion
- VID/FID Change—Generated by processor in response to a software controlled voltage (VID) or frequency (FID) change
- WBINVD—Generated by processor in response to execution of a WBINVD instruction
- INVD—Generated by processor in response to execution of an INVD instruction

These packets originate from the host and are broadcast downstream to all HyperTransport technology I/O devices in the system.

F.2.2 VID/FID Changes

The Southbridge is required to support VID/FID changes as follows:

- Execute the system state transition specified in Section F.2 in response to the VID/FID message from the host.
- Assert LDTSTOP# as described in the above sequence.
- Include a BIOS-programmable configuration register that specifies the LDTSTOP# assertion time associated with VID/FID change system state transitions. Values ranging from 1 microsecond to 100 microseconds are recommended.

F.2.3 Throttling

Throttling differs from most system state transitions in that the Southbridge sends STPCLK assertion messages to the host without direct initiating messages. Because of this, the possibility of a deadlock exists when the host initiates a system state transition simultaneously with a STPCLK assertion message from the Southbridge. Therefore, to avoid this possibility, the following Southbridge requirements exist:

- If a STPCLK assertion message for throttling is sent from the Southbridge and then a system state transition is initiated via a nonposted access from the host to the Southbridge prior to the STOP_GRANT message for throttling, then the Southbridge is required to send another STPCLK assertion message to the host with the SMAF field programmed for the host-initiated system state transition. The response to the host access must then follow.
- If a STPCLK assertion message for throttling is sent from the Southbridge and then a system state transition is initiated via a posted access from the host to the Southbridge (such as the VID/FID system management cycle), then the Southbridge is required to (1) wait for the

STOP_GRANT system management message from the host, (2) send a STPCLK deassertion message, and (3) send the STPCLK assertion message to the host with the SMAF field programmed for the host-initiated system state transition.

There is no deadlock possibility when roughly coincident throttling STPCLK assertion messages occur with interrupt requests. They are naturally resolved as follows:

- If a STPCLK assertion message for throttling is sent from the Southbridge simultaneously with a host-initiated nonposted command that results in an interrupt request (e.g., SMI), then the Southbridge sends the interrupt request to the host followed by the response to the nonposted command. The host is required to send the STOP_GRANT system management message after it receives the response.
- If an asynchronous interrupt request (not initiated by a host nonposted request) is received by the host after the STPCLK assertion message, then the interrupt request is accepted by the host, regardless of whether the STOP_GRANT system management message has been sent. However, the host might not act on the interrupt request until the STPCLK deassertion message is received by the host.

F.2.4 C3 System State Transitions and LDTREQ#

It is possible that LDTSTOP# will be asserted during system state transitions to ACPI-defined C3 (this is only expected on battery-powered platforms). A Southbridge on such a platform is required to deassert LDTSTOP# when any devices require use of the chain. It is recommended that this be accomplished through a signal specified here called LDTREQ#.

LDTREQ# is an open-drain signal connected to all HyperTransport technology devices on the platform that are capable of generating bus master activity while the system is in the C3 state. For proper C3 operation, transactions originating at the host processor must not cause LDTREQ# assertion.~~LDTREQ# can be asserted by any HyperTransport technology device while LDTSTOP# is asserted to indicate to the Southbridge that a HyperTransport technology transaction is required somewhere in the system. The Southbridge responds by deasserting LDTSTOP# and transitioning the system to the C0 state.~~

~~LDTREQ# assertion forces the ACPI-defined BM_STS bit in the Southbridge to be set to a 1. LDTREQ# is required to be asserted whenever a device has an outstanding transaction in the HyperTransport fabric or needs to inject a new transaction into the HyperTransport fabric, regardless of the host state or whether LDTSTOP# is asserted.~~

A device is required to assert LDTREQ# whenever it has an outstanding transaction in the HyperTransport fabric or needs to inject a new transaction into the HyperTransport fabric, regardless of the host processor state or whether LDTSTOP# is asserted. Devices only assert LDTREQ# for HyperTransport transactions they initiate, not for HyperTransport transactions they forward.

The Southbridge responds to LDTREQ# assertion by:

- Setting the ACPI-defined BM_STS bit to a 1.
- Deasserting LDTSTOP# if asserted.
- Transitioning the host processor to the C0 state if the Host is in the C3 state.

F.2.5 SMI and STPCLK

The system Southbridge is the only device that is allowed to generate SMI interrupt and STPCLK system management messages. Since both of these messages replace legacy signals, they have special ordering requirements to remain compatible with legacy behavior. In legacy systems, both of these signals have the following behavior:

1. The host causes the signal to be asserted with an instruction that (1) requires a response and (2) allows for no further instruction execution until the response is received.
2. The host detects the assertion of the signal prior to the response.
3. After the response, the host responds to the signal (by taking the SMI interrupt or initiating the STPCLK sequence) prior to executing any more instructions.

Thus, to replicate this behavior, the following requirements exist:

- The Southbridge may generate SMI or STPCLK messages in response to host-initiated transactions. If the host-initiated transaction requires a response, then the response is required to follow the SMI or STPCLK message upstream.
- The UnitID of the SMI or STPCLK message must match the UnitID of the response, or the upstream ordering between the two is not ensured.
- In order to guarantee that the response to the host does not pass the SMI or STPCLK message, the PassPW bit in the response must be clear, even if it is a TgtDone.
- As long as the SMI or STPCLK message is received prior to the response to the initiating instruction, the host is required to guarantee that it execute no more instructions beyond the initiating instruction, before it responds to the SMI or STPCLK message.

The host bridge responds to SMI with the SMIACK assertion system management message down all HyperTransport chains. The SMIACK assertion message is required to represent the system state of all the processing elements behind the host bridge. Therefore, the host responds with a single SMIACK assertion message after the SMI interrupt is received. However, as processing elements behind the host bridge exit the SMIACK state, multiple SMIACK deassertion messages may be sent downstream from the host bridge.

The Southbridge is required to send no more than one SMI interrupt request message before receiving the SMIACK assert system management message. After the SMIACK assert system management message is received by the Southbridge, it may send another SMI to the host. In some systems, the host bridge may send more than one SMIACK assertion for an SMI. The Southbridge should tolerate this and is allowed to act on the first SMIACK assertion received.

F.2.6 Default State of Virtual Wires

It is required that the state of the virtual wires in the Southbridge and the host match after reset. The default state for all virtual wires, including all interrupts, IGNNE, A20M, FERR, STPCLK, and SMIACK, is deasserted.

~~A.3~~F.3 Initialization Issues

Hosts on x86 platforms may not be capable of accepting upstream requests until initialized by software. Therefore, it is required that after deassertion of RESET# or transmission of an INIT interrupt message, no upstream system management messages, interrupt requests, fences or flushes be generated until enabled by the host. The method used to meet this requirement is outside the scope of this specification. Note that upstream sized read and write requests to memory and I/O space are also disabled after RESET# by the Bus Master Enable configuration bit, as described in Section 7.3.1.3.

~~A.4~~F.4 AGP Bridge Issues

Some legacy operating systems require that the location of AGP-specific configuration registers must be hardwired as follows:

- The AGP-defined capabilities header must be in Bus 0, Device 0, Function 0.
- The AGP aperture base address register must be at Bus 0, Device 0, Function 0, Offset 10h.

Therefore, to meet these requirements, it is recommended that AGP devices be designed as follows:

- The AGP bridge resides on the HyperTransport chain specified to be Bus 0.
- The AGP device uses multiple UnitIDs.
- The base UnitID register is programmed to 0 after the conclusion of I/O ~~fabric~~ chain initialization. A different UnitID value must be used during the initialization sequence (See Section 12.3).
- The device number that matches the base UnitID register contains the capabilities header and the AGP aperture base address register (at Offset 10h).
- The device number that is one greater than base UnitID is used for the PCI-to-PCI bridge header that corresponds to the AGP bridge.
- The UnitID that matches the base (0) is not used for any AGP-initiated I/O streams or responses so that there is no conflict with host-initiated I/O streams or responses. Only the UnitIDs greater than the base are used for I/O streams.
- It is expected that the AGP-defined graphics address remapping table (GART) is located in the host. Therefore, the AGP aperture base address register and any other registers that are located in the AGP device but required by the host are copied via software into implementation-specific host registers.

In the situation described above, the host's configuration registers should be placed somewhere other than Device 0, in order to avoid conflicting with the predefined AGP registers. In a sharing double-hosted chain, this requires the hosts to implement the Device Number field (defined in Section 7.5.3.3.3) so that the hosts may address each other after the AGP bridge has assumed Device 0.

Note that if legacy OS support is not required, the AGP device's base UnitID register may be programmed to any value compliant with the HyperTransport protocol.

~~A.5~~F.5 Configuration Space Access Mechanism

All x86 HyperTransport host bridges must implement the configuration transaction mechanism described in Section 3.2.2.3.2 of the *PCI Local Bus Specification*, Revision 2.2, for generating configuration space accesses. This mechanism entails a 32-bit address register at I/O space CF8h and a 32-bit data register at I/O space CFCh. x86 processor accesses to these I/O space registers result in the appropriate HyperTransport technology configuration transaction, as defined in Section 7.1.

G CRC Testing Mode

Writing a 1 to the CRC Start Test bit of the Link Control register (see Section 7.5.4.2) causes the transmitter to enter CRC diagnostic mode.

- The transmitter begins by issuing a NOP packet with its Diag bit set, which instructs the receiver to ignore the CAD and CTL signals for the following 512 bit-times in each byte lane, not counting the bit-times allotted to CRC stuffing.
- The transmitter can then drive any pattern it wants on the CAD and CTL signals (other than during CRC stuffing), even to the extent of allowing CTL to change state between arbitrary bit-times, with one exception. The test pattern may not contain four consecutive bit-times of all 1 bits on any byte lane (CAD and CTL signals), as that ~~would~~ could be interpreted by the receiver as a sync packet. How the transmitter decides what to transmit as a test pattern is beyond the scope of this specification.
- CRC is still generated and checked for the interval, and CRC stuffing occurs normally, but the received data is ignored, and packet generating and tracking state machines are suspended in the state they were in when the diagnostic NOP was received.
- CRC errors detected during this time will be logged by setting the CRC Error bits, and will be treated as fatal if the CRC Flood Enable bit is set.
- If the CRC Force Error bit (Section 7.5.4.3) is set when the CRC Start Test bit is set, the test pattern will contain at least one CRC error in each active byte lane.
- When the test interval has completed, and the last CRC covering any part of the test interval has been stuffed, hardware clears the CRC Start Test bit in the transmitter.
- Packet transmission resumes from the suspended state, which may be in the middle of a data packet.

This test mode should not be used unless both sides of the link indicate support for it in bit 2 of the Feature Capability register as defined in Section 7.5.10.3.

H Doubleword-Based Data Buffer Flow Control

HyperTransport™ technology provides an operating mode in which posted request data, nonposted request data, and response data buffers are flow controlled with doubleword granularity. In this mode, the command packets (posted requests, nonposted requests and responses) are still flow-controlled using packet size granularity. All HyperTransport technology devices must support the 64-byte granular flow-control mode as previously described. Further, 64-byte granular flow control is the default operation for all devices after cold reset (see Section 12.1 for the definition of cold reset). Initialization firmware can determine the capability of devices on either side of a link, and if they support doubleword-based flow control, program them to operate in that mode. Switching between flow-control modes requires cycling through a software-initiated warm reset. An LDTSTOP# disconnect sequence cannot be used to switch between flow-control modes because flow-control buffer state must be kept consistent across LDTSTOP# disconnects.

Table 80 shows the NOP packet format for doubleword-based flow control. In this mode, the 2-bit data buffer flow-control fields previously described are interpreted as the upper two bits of a 5-bit flow-control field. There are three 5-bit fields in total, each corresponding to one of the three virtual data channels. Each 5-bit field indicates to the transmitter that the receiver is freeing from 0 to 31 doublewords of data buffer within a channel. The byte mask doubleword for sized byte writes is included as data in the doubleword-based flow-control calculation by both the transmitter and the receiver.

Table 80. NOP Packet Format for Doubleword-Based Flow Control

Bit-Time	7	6	5	4	3	2	1	0
0	Rsv	DisCon	Cmd[5:0]					
1	ResponseData[4:3]		Response[1:0]		PostData[4:3]		PostCmd[1:0]	
2	0	Diag	Isoc	RespData[2]	NonPostData[4:3]		NonPostCmd[1:0]	
3	ResponseData[1:0]		PostData[2:0]			NonPostData[2:0]		

As in packet-based data buffer flow control mode, if a transmitter receives more increments than it can keep track of, it must not allow its counter to wrap, but must discard the extras. This has the effect that the link will use the maximum amount of buffer storage that both the transmitter and receiver can support. All transmitter counters must be a minimum of six bits wide, allowing up to 63 doublewords of buffer storage to be tracked without loss.

Doubleword-based flow control is expected to be deployed only in special circumstances where large block-sized, high-performance transfers are not important to the operation of the HyperTransport technology device within the system. All HyperTransport technology devices must support 64-byte flow-control mode and are encouraged to implement a large enough 64-byte buffer pool to fully utilize the HyperTransport links in the system applications envisioned for that device.

I Quick Reference for x86 Systems

Code	Hex	VChan	Command	Comments/Options	Packet Type
000000	00	-	NOP	Null packet. Contains flow control information.	Info
000010	02	NPC	Flush	Flush Posted Writes within one I/O stream.	Request
001xxx 101xxx	08 28	NPC PC	Wr (sized)	Write Request [5] Defines whether request is posted: 0: Nonposted 1: Posted [2] Defines the data length: 0: Byte 1: Doubleword [1] Defines bandwidth/latency requirements: 0: Normal 1: Isochronous [0] Indicates whether access requires host cache coherence (ignored if access is not to host memory): 0: Noncoherent 1: Coherent	Req/Addr/Data
01xxxx	1x	NPC	Rd (sized)	Read Request [3] Defines ordering requirements for response: 0: Response may not pass posted requests 1: Response may pass posted requests [2] Defines the data length: 0: Byte 1: Doubleword [1] Defines bandwidth/latency requirements: 0: Normal 1: Isochronous [0] Indicates whether access requires host cache coherence (ignored if access is not to host memory): 0: Noncoherent 1: Coherent	Req/Address
110000	30	R	RdResponse	Read Response	Resp/Data
110011	33	R	TgtDone	Tell source of request that target is done.	Response
111010	3A	PC	Broadcast	Broadcast Message	Req/Address
111100	3C	PC	Fence	Fence Posted Writes within all I/O streams.	Request
111101	3D	NPC	Atomic-RMW	Atomic Read-Modify-Write	Req/Addr/Data
111111	3F	-	Sync/Error	Link Synchronization and Error Packet	Info

PCI Command Encodings

Code	Hex	Command	Posted
0000	0	Interrupt Acknowledge	
0001	1	Special Cycle	●
0010	2	I/O Read	
0011	3	I/O Write	
0110	6	Memory Read	
0111	7	Memory Write	●
1010	A	Configuration Read	
1011	B	Configuration Write	
1100	C	Memory Read Multiple	
1101	D	Dual Address Cycle	
1110	E	Memory Read Line	
1111	F	Mem Write and Invalidate	●

HyperTransport^{TM} Technology Address Map

Base Address	Top Address	Size	Use
00_0000_0000	FC_FFFF_FFFF	1012 GB	System Memory/ Memory-Mapped I/O
FD_0000_0000	FD_F8FF_FFFF	3984 MB	Interrupt/EOI
FD_F900_0000	FD_F90F_FFFF	1 MB	Legacy PIC IACK
FD_F910_0000	FD_F91F_FFFF	1 MB	System Management
FD_F920_0000	FD_F92F_FFFF	1 MB	Reserved – x86
FD_F930_0000	FD_FBFF_FFFF	45 MB	Reserved
FD_FC00_0000	FD_FDFF_FFFF	32 MB	I/O
FD_FE00_0000	FD_FFFF_FFFF	32 MB	Configuration
FE_0000_0000	FF_FFFF_FFFF	8 GB	Reserved

Read/Write/Broadcast (Downstream Only) / Atomic **Read-Modify-Write** ~~(Upstream Only)~~

Bit-Time	7	6	5	4	3	2	1	0
0	SeqID[3:2]		Cmd[5:0]					
1	PassPW	SeqID[1:0]		UnitID[4:0]				
2	Mask/Count[1:0]		Compat	SrcTag[4:0]				
3	Addr[7:2]						Mask/Count[3:2]	
4	Addr[15:8]							
5	Addr[23:16]							
6	Addr[31:24]							
7	Addr[39:32]							

There are two types of Read-Modify-Write Request, carrying either one or two quadwords (QW) of data:
Type 0: Fetch and Add has Count = 1 and adds the QW of data to the QW addressed.
Type 1: Compare and Swap has Count = 3 and if the QW addressed matches the first QW of data it is replaced by the second QW of data.

Flush/Fence (Upstream Only)

Bit-Time	7	6	5	4	3	2	1	0
0	SeqID[3:2]		Cmd[5:0]					
1	PassPW	SeqID[1:0]		UnitID[4:0]				
2	Rsv			SrcTag[4:0]				
3	Rsv							

Read Response/Target Done

Bit-Time	7	6	5	4	3	2	1	0
0	Isoc	Rsv	Cmd[5:0]					
1	PassPW	Bridge	Rsv	UnitID[4:0]				
2	Count[1:0]		Error	SrcTag[4:0]				
3	Rsv		NXA	Rsv			Count[3:2]	

NOP

Bit-Time	7	6	5	4	3	2	1	0
0	Rsv	Discon	Cmd[5:0]					
1	ResponseData[1:0]		Response[1:0]		PostData[1:0]		PostCmd[1:0]	
2	0	Diag	Isoc	Rsv	NonPostData[1:0]		NonPostCmd[1:0]	
3	Rsv							

Configuration Cycle Addressing (Nonposted) in HyperTransport^{TM} Technology (PCI carries type in bit 0)

39 24	23 16	15 11	10 8	7 2	1:0
FDFE/FDFF	Rsv/Bus Number	Device Number	Function Number	Register Number	00

FDFE is used for Type 0 cycles, which do not carry a bus number since they are targeted at the current bus.
FDFF is used for Type 1 cycles, which do carry the number for the destination bus.
Type 1 cycles are also used to send special cycles to other buses ([15:0]=FF00).

Interrupt Request Addressing (Byte Posted Write) / **End-of-Interrupt Addressing** (Broadcast)

39 24	23 16	15 8	7	6	5	4 2	1:0
FDF8	Vector	Destination	MT[3]	DM	TM	Message type	00

DM: Destination Mode. 0 = Physical/EOI, 1 = Logical TM: Trigger Mode. 0 = Edge/EOI, 1 = Level
Message Types: 0 = Fixed, 1 = Lowest Priority Mode, 2 = SMI, 3 = NMI, 4 = INIT, 5 = Startup,
6 = External (Legacy PIC) INTR ([23:0]=00FF18), 7 = APIC EOI (Destination=0)

System Management (Special Cycle) **Address Encoding** (Byte PW Upstream, Broadcast Downstream)

39 20	19 16	15 12	11 8	7 2	1:0
FDF91	Reserved	Command Type	Payload	Reserved	00

Command Types: To CPU 1: Payload[1] = A20M Virtual Wire; Payload[0] = IGNNE V.W.
3: Payload[0] = STPCLK Virtual Wire
From CPU 2: Payload[0] = FERR Virtual Wire, 4: Shutdown, 5: Halt, 6: Stop-Grant,
7: VID/FID Change, 8:WBINVD, 9:INVD, 10:Payload[0]=SMIACK Virtual Wire
Payload[3:1] = System Management Action Field (STPCLK and Stop-Grant)

Section 2 – Electrical Interface

13 HyperTransport™ Link Overview

The HyperTransport™ link is designed to deliver a scalable and high performance interconnect between CPU, memory, and IO devices. The HyperTransport link uses low-swing differential signaling with on-die differential termination to achieve high data rates: 400 million transfers per second (MT/s), 600 MT/s, 800 MT/s, 1.0 GT/s, 1.2 GT/s, and 1.6 GT/s. The HyperTransport link uses scalable frequency and data width to achieve scalable bandwidth.

The HyperTransport link consists of two independent source synchronous clocked unidirectional sets of wires. Each set of wires includes CADOUT [n:0], CLKOUT[m:0], and CTL, where n=1, 3, 7, 15, or 31, and m=0, 0, 0, 1, or 3, respectively. HyperTransport link packets are carried on the high speed CADOUT and CTLOUT wires and timed to CLKOUT, which is nominally 90° delayed from CADOUT and CTLOUT. In the receiver, the packets are received on high speed CADIN and CTLIN wires and captured by simple sampling with CLKIN. The transfer timing requirements for data capture at the receiver is defined in this specification. Once captured from the interconnect, the packet must be passed into the receive clock domain which may or may not be derived from the same frequency source as the transmit clock domain. The ability to pass the packets between these two clock domains depends upon the clocking mode and the accumulated phase error between them. The accumulated phase error between the transmit and receiver clock domains is defined as phase recovery timing in this specification.

CADOUT, CTLOUT, and CLKOUT signals use differential drivers and have a point-to-point topology from the transmitter to the receiver. The receiver provides on-die differential termination as defined in this specification. The AC and DC device output and input voltage requirements are defined in this specification.

In addition to the low-swing differential signals, the HyperTransport technology defines four single-ended LVCMOS signals used for link reset and power down initiation and cessation. PWROK is a required input to each HyperTransport device to indicate that all required system power supplies are within specification and that the reference clock is within specification. PWROK is driven by system reset logic. RESET# is a required input to each HyperTransport device to indicate the system reset state. RESET# is driven by system reset logic. LDTSTOP# and LDTREQ# are used in systems requiring power management to signal requests for power related system activities. The AC and DC device output and input requirements for these signals are defined in this specification.

Figure 12 on page 184 shows the basic HyperTransport link interconnect for up to 32-bit links. Table 81 on page 185 describes the link signal types.

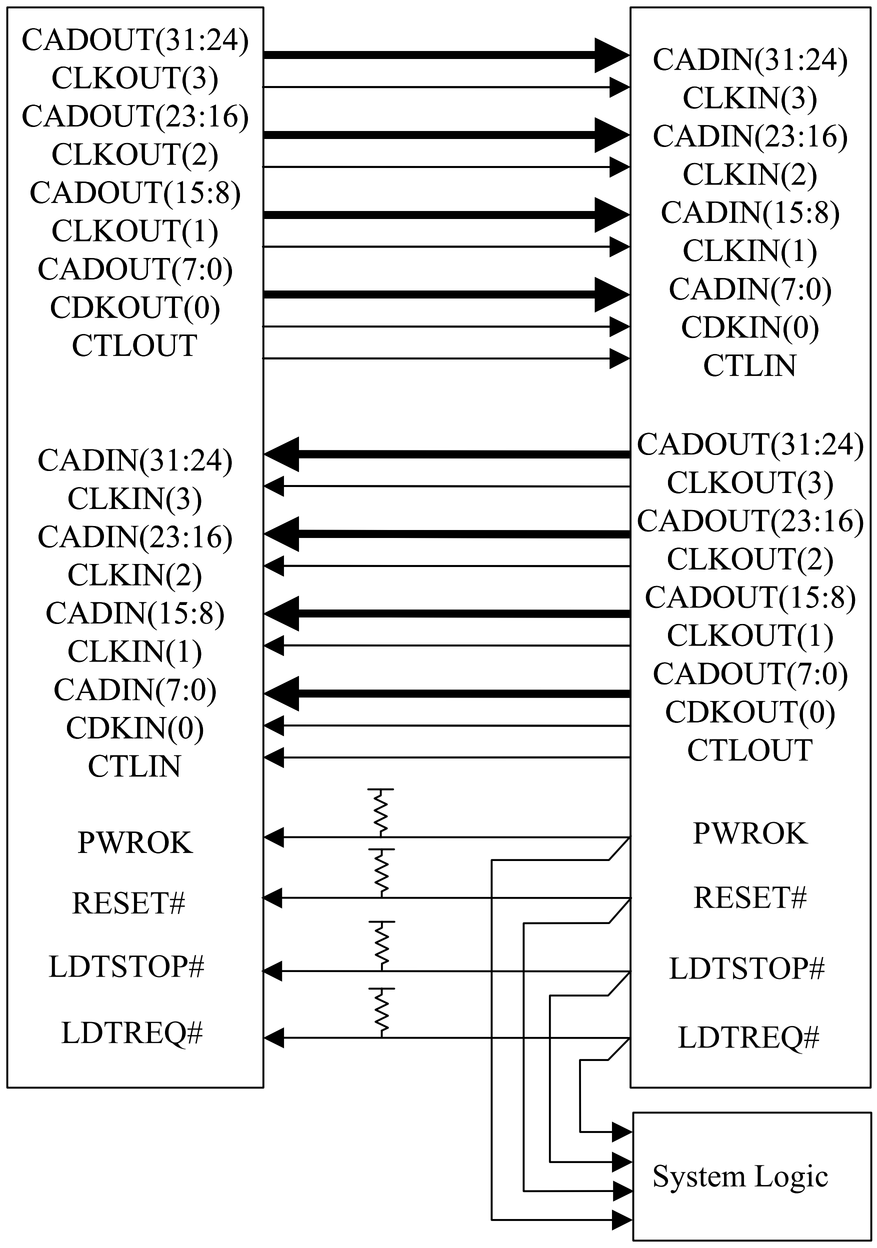


Figure 12. HyperTransport™ Link Interconnect

Table 81. HyperTransport™ Link Signal Types

Name	Driver Type	Receiver Type
CADOUT / CADIN	Differential	Differential, terminated
CLKOUT / CLKIN	Differential	Differential, terminated
CTLOUT / CTLIN	Differential	Differential, terminated
LDTSTOP# [1]	Open Drain LVCMOS	Single-ended 2.5V tolerant LVCMOS
LDTREQ# [1]	Open Drain LVCMOS	Single-ended 2.5V tolerant LVCMOS
PWROK [1,2]	Open Drain ~~or PP~~ LVCMOS	Single-ended 2.5V tolerant LVCMOS
RESET# [1,2]	Open Drain ~~or PP~~ LVCMOS	Single-ended 2.5V tolerant LVCMOS

Notes:

1. *~~OD output implementations will~~These signals require a single pullup resistor on the system board for required functionality. ~~Recommended~~ The value of this ~~pullup~~ resistor is required to be >=1 KΩ. Routing on PCB of these signals must be done in a daisy-chain fashion, with any stubs being less than 1” in length.*
2. *Some devices may use these signals as both input and output.*

14 Electrical Overview

The HyperTransport link electrical specification provides for very high speed date rates by taking advantage of the inherent common-mode noise rejection and low skew properties of low-swing differential signals. On-die differential termination is included to increase the signal-to-noise ratio seen at the receiver while allowing for very simple system interconnect designs. The electrical requirements support multiple driver implementations and simple receiver data recovery methods that can be implemented in multiple logic process generations.

15 Supply Characteristics

The supply for HyperTransport link drivers and receivers is a single fixed supply. The differential nature of HyperTransport link switching minimizes the current transients required of the VLDT supply when compared to single-ended systems, however the requirements and the design of the VLDT regulation and distribution system must be considered carefully. Voltage mode drives implemented completely in the VLDT domain can cause significant noise on VLDT. The AC impedance of the VLDT distribution system must be considered along with the transient requirements of the link in order to maintain the specified VLDT tolerance. The VLDT supply needs only to source current.

Table 82. HyperTransport™ Link Power Supply Characteristics

Parameter	Description	Min	Typical	Max	Units
VLDT	HyperTransport Link Supply Voltage[1]	1.14	1.2	1.26	V
VLDT tolerance	VLDT supply tolerance[1]	–5		+5	%

Notes:

1. *Measured at the external connection to the HyperTransport device package. The VLDT as measured on the die should maintain a 1.1V to 1.3V range under all conditions. This ±100mV variation at the die is considered when defining the DC output characteristics in this specification.*

16 Power Requirements

HyperTransport link power consumption per differential pair under DC conditions is calculated from specified R_{ON} and R_{TT} values.

Table 83. Power Requirements

Parameter	**Description**	**Min**	**Typical**	**Max**	**Units**
P_{DC}	DC power per output bit [1,2]	5.9	7.2	9.0	mW
P_{AC}	AC power per output bit [1,2]			66	mW
P_{TAC}	Transmitter AC power per bit [1,3]			53	mW
P_{RAC}	Receiver AC power per bit [1]			13	mW

Notes:

1. *Includes both true and complement drivers or receivers.*
2. *Includes both differential transmitters and receivers operating at 1.6GT/s and VLDT at min, typical, or maximum respectively.*
3. *Implementations that supply much of the pre-driver from a supply other than VLDT can consume much less than this specified maximum.*

17 Input/Output DC Voltage Characteristics

The DC characteristics are valid and should be measured only when the circuitry has assumed steady-state conditions. Steady-state is attained when there are no transient effects present in the driver, receiver, interconnect, supply circuitry, or distribution paths. The use of switching waveforms to illustrate $\Delta V_{OD,}$ $\Delta V_{ID,}$ $\Delta V_{OCM,}$ and ΔV_{ICM} definitions does not imply that these measurements are taken under switching conditions, only that the values of two different logic states be compared. These DC specifications are to be used for circuit verification and characterization, system validation, and production test.

17.1 Impedance Requirements

R_{TT} is the value of the differential input impedance of the receiver under DC conditions implemented with an on-die differential terminating resistor. This specification must be supported by any compensation technique used within the receiver across all device specific process, voltage, and temperature operating points. The R_{TT} value is defined to match the Z_{OD} of the coupled transmission lines and to provide a slightly overdamped single-ended termination.

R_{ON} is the driver output impedance under DC conditions. This range must be maintained over the valid V_{OD} range. This specification must be supported by any compensation technique used within the output driver across all device specific process, voltage, and temperature operating points. The R_{ON} value is defined to match one-half of the Z_{OD} of the coupled transmission lines.

ΔR_{ON} (pullup) is the allowable difference in the driver output impedance between the true and complement when driving a logic 0 and when driving a logic 1 (additionally defined as when true is driven high and when complement is driven high). ΔR_{ON} (pullup) is defined to limit differences in both output rising edge slew rate and the resulting differential skew and crossing point shift.

ΔR_{ON} (pulldown) is the allowable difference in the driver output impedance between the true and complement when driving a logic 1 and when driving a logic 0 (additionally defined as when true is driven low and when complement is driven low). ΔR_{ON} (pulldown) is defined to limit the differences in both output falling-edge slew rate and the resulting differential skew and crossing point shift.

Table 84 on page 189 gives the DC specifications for these parameters.

Table 84. R_{TT} and R_{ON} DC Specifications

Parameter	Description	Min	Typical	Max	Units
R_{TT}	Differential Termination	90	100	110	Ω
R_{ON}	Driver Output Impedance	45	50	55	Ω
ΔR_{ON} (pullup)	High Drive Impedance Magnitude Change	0		5	%
ΔR_{ON} (pulldown)	Low Drive Impedance Magnitude Change	0		5	%

17.2 DC Output Voltage Requirements

These specifications place requirements on the driver and are derived from the specified R_{ON} and R_{TT} tolerance or ΔR_{ON} (pulldown) or ΔR_{ON} (pullup) tolerances.

17.2.1 ATE Test Environment

The specified values are valid and should be tested directly at the transmitter output pins DO+ and DO–. Automated test equipment power supplies, supply distribution, and signal interconnect should be designed to provide best case operating conditions such that the ATE equipment can then effectively apply guard band to production test points as necessary. For output signals specifically, this requirement means driving an ideal 100-Ω Z_{OD} environment.

17.2.2 Reference System Load

The following reference system load is provided for simulation or system test environments where the more realistic system load is desired.

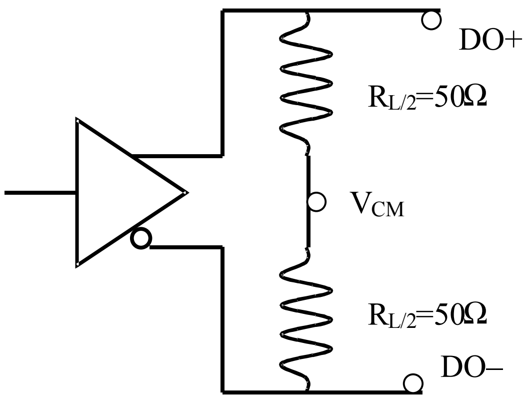


Figure 13. DC Output Reference System Load

17.2.3 Output Voltage Parameter Descriptions

V_{OD} is the differential output voltage or the voltage difference between true and complement under DC conditions. V_{OD} is equal to $|DO+ - DO-|$ in the following figure.

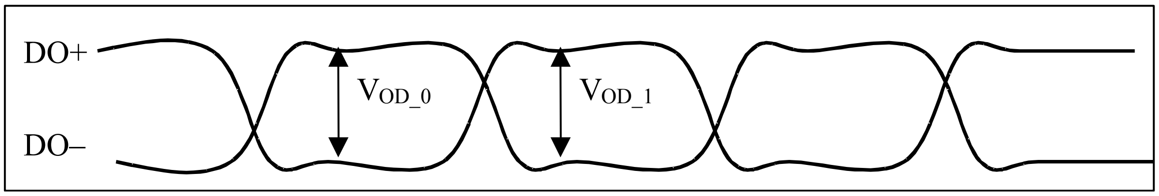


Figure 14. V_{OD} DC

ΔV_{OD} is the change in magnitude between the differential output voltage while driving a logic 0 and while driving a logic 1. ΔV_{OD} is equal to $V_{OD\_0} - V_{OD\_1}$.

V_{OCM} is the output common-mode voltage defined as the average of the true voltage magnitude and the complement voltage magnitude relative to ground under DC conditions. V_{OCM} is not directly measurable under operation unless the output load circuit is used and the V_{OCM} measured at the point marked V_{cm}. In operational systems this value will be derived using the following equation. V_{OCM} is equal to (DO+ + DO–) / 2 in the following figure.

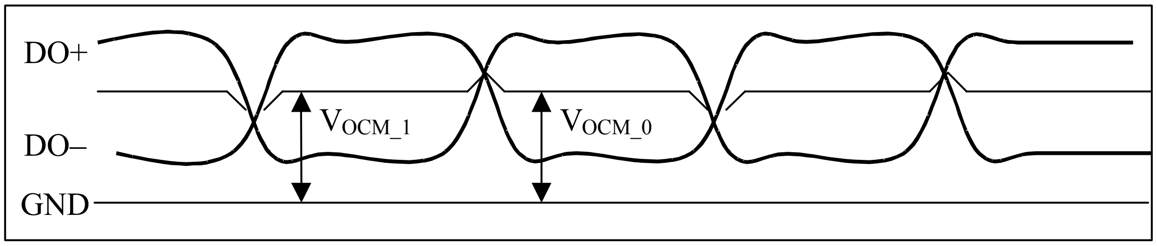


Figure 15. V_{OCM} DC

Δ V_{OCM} is the change in magnitude between the output common-mode voltage while driving a logic 0 and while driving a logic 1 under DC conditions. Δ V_{OCM} is equal to $V_{OCM\_1} - V_{OCM\_0}$.

17.3 DC Input Requirements

These parameters place requirements on the receiver and are derived from the output parameters and interconnect effects.

17.3.1 ATE Test Environment

The specified values are valid and should be tested directly at the receiver input pins RI+ and RI–. Automated test equipment power supplies, supply distribution, and signal line losses should be calibrated such that the parameters are tested directly at the receiver inputs. Detailed requirements are ATE equipment specific and beyond the scope of this document.

17.3.2 Input Parameter Descriptions

V_{ID} is the input differential voltage or the voltage difference between the true and complement under DC conditions. V_{ID} is equal to $|RI+ - RI-|$ on the following figure.

Figure 16. V_{ID} DC

ΔV_{ID} is the change in magnitude between the input differential voltage while receiving a logic 0 and while receiving a logic 1. ΔV_{ID} is equal to $V_{ID\_0} - V_{ID\_1}$.

V_{ICM} is the input common-mode voltage defined as the average of the true voltage magnitude and the complement voltage magnitude relative to ground under DC conditions. V_{ICM} is equal to $(RI+ + RI-) / 2$ in the following figure.

Figure 17. V_{ICM} DC

ΔV_{ICM} is the change in magnitude between the input common-mode voltage while driving a logic 0 and while driving a logic 1. ΔV_{ICM} is equal to $V_{ICM\_1} - V_{ICM\_0}$.

17.4 Differential Signal DC Specifications

Table 85 defines the allowed values for each of the DC characteristics. More detail regarding the derivation of these values is included in Appendix J on page 217.

Table 85. HyperTransport™ Link Differential Signal DC Specifications

Parameter	**Description**	**Min**[1]	**Typical**[2]	**Max**[3]	**Units**
V_{OD}	Differential Output Voltage [4]	495	600	715	mV
ΔV_{OD}	Change in V_{OD} Magnitude	–15		15	mV
V_{OCM}	Output common-mode voltage [4]	495	600	715	mV
Δ V_{OCM}	Change in V_{OCM} magnitude	–15		15	mV
V_{ID}	Input differential voltage	200	600	1000	mV
ΔV_{ID}	Change in V_{ID} magnitude	–15		15	mV
V_{ICM}	Input common-mode voltage	440	600	780	mV
ΔV_{ICM}	Change in V_{ICM} magnitude	–15		15	mV

Notes:

1. *Minimum values assume $V_{LDT} = V_{LDT}$_min as a measurement condition.*
2. *Typical values assume $V_{LDT} = V_{LDT}$_typ as a measurement condition.*
3. *Maximum values assume $V_{LDT} = V_{LDT}$_max as a measurement condition.*
4. *See Appendix J on page 217 for derivation of V_{OD} and V_{OCM}.*

17.5 Single-Ended Signal AC/DC Specifications

Table 86 defines the allowed values for the single-ended signals defined by the HyperTransport technology.

Table 86. HyperTransport™ Link Single-Ended Signal AC/DC Specifications

Symbol	Parameter	Test Conditions		Min	Typical	Max	Unit
V_{DD}	DC Supply Voltage			2.37	2.5	2.63	V
V_{IH}	High Level Input Voltage	$V_{OUT} \geq V_{VOH}$(min)		1.7		V_{DD} + 0.3	V
V_{IL}	Low Level Input Voltage	$V_{OUT} \leq V_{VOL}$(max)		–0.3		0.7	V
Tr	Input rising rate	Vil < Vin < Vih, monotonic[1]		0.01			V / ns
Tf	Input falling-edge rate	Vih > Vin > Vil, non-monotonic		0.01			V / ns
V_{OL}	Low Level Output Voltage	V_{DD} = min, $V_I = V_{IH}$ or V_{IL}	~~I_{OH}~~-I_{OL} = 2 mA			0.7	V
I_I	Input Current	V_{DD} = max, V_I = V_{DD} or GND				±500	µA

Notes:

1. *This rising edge is only guaranteed to be monotonic if the pull-up resistor is >=1Kohm and routing of these signals is done in a daisy chain fashion*

17.6 Input/Output AC Voltage Characteristics

The AC characteristics are valid and should be measured only when the circuitry has not yet reached steady-state conditions. This is the normal operating state of the link and considers that signals will be switching and contain noise induced by crosstalk, reflections, inter-symbol interference, and other effects. Power supplies will contain noise induced from simultaneous switching, resonance, and other effects. These AC characteristics are to be used to circuit verification and system characterization and validation. Testing many of these parameters will not be possible in a production test flow. These specifications must be guaranteed by design or characterized across process, voltage, and temperature for a given product. Some electrical parameters are specified differently for links designed to operate up to 800 MT/s and links designed to operate above 800 MT/s. Additionally, the required interconnect and test load circuits vary for links designed to operate in these two frequency ranges.

17.7 Impedance Requirements

R_{TT} is the value of the differential input impedance of the receiver under AC conditions implemented with an on-die differential terminating resistor. Techniques used to compensate R_{TT} for changes due to P, V, or T fluctuations can result in R_{TT} having a non-linear I-V curve; therefore R_{TT} is specified under AC conditions and should be characterized or guaranteed over all process, voltage, and temperature operating points.

R_{ON} (pullup) is the driver output impedance while driving high under AC conditions. This value and tolerance must be maintained from 0.5 * VLDT_nom to VLDT_nom. R_{ON}(pulldown) is the driver output impedance while driving low under AC conditions. This value and tolerance must be maintained from 0V to 0.5 * VLDT. Techniques used to compensate the output driver for changes due to P, V, or T variations can result in the driver having a non-linear I-V curve; therefore R_{ON} is specified under AC conditions and should be characterized or guaranteed over all process, voltage, and temperature operating points.

C_{OUT} is the driver output pad capacitance and is limited to act, along with the recommended transmitter package trace single-ended impedance of 35–65 Ω and maximum length of less than 850 mils, to create a matched impedance between the driver R_{ON} and the characteristic impedance of the package trace.

C_{IN} is the receiver input pad capacitance and is limited to act, along with the recommended receiver package trace single-ended impedance of 35–65 Ω and maximum length of less than 850 mils, to create a matched impedance between the interconnect transmission line and the characteristic impedance of the receiver package and input pad.

Table 87 on page 195 gives the AC impedance specifications for these parameters.

Table 87. AC Impedance Specifications

Parameter	Description	Min	Typ	Max	Units
R_{TT}[1]	Differential Termination	90	100	110	Ω
R_{ON}(pullup)[2]	Driver Output Impedance driving high	45	50	55	Ω
R_{ON}(pulldown)[3]	Driver Output Impedance driving low	45	50	55	Ω
Cout	Output pad capacitance for devices rated above 800 MT/s.			3	pF
	Output pad capacitance for devices rated up to 800 MT/s.			5	pF
Cin	Input pad capacitance for devices rated above 800 MT/s.			2	pF
	Input pad capacitance for devices rated up to 800 MT/s			5	pF

Notes:

1. *R_{TT} range is valid for input V_{ID} of 0.25 * VLDT and 0.75 * VLDT or between 0.285V and 0.945V*
2. *R_{ON}(pullup) range is valid for outputs between 0.5 * VLDT and VLDT*
3. *R_{ON}(pulldown) range is valid for outputs between 0V and 0.5 * VLDT*

17.8 AC Output Requirements

These parameters place requirements on the driver and add, to the DC signal characteristics, both supply and signal noise caused by signal transitions under AC conditions.

17.8.1 ATE Test Environment

In a dedicated ATE test environment, the device under test should drive an ideal load under ideal conditions. This implies that automated test equipment power supplies, supply distribution, and signal interconnect be designed as to provide best case operating conditions and not mimic a reference system load. This design allows the test engineer to accurately characterize the device performance and to define the production test point and guard band such that devices meet the specified characteristics in system or reference system environments.

17.8.2 Reference System Load

The following reference system load is provided for simulation or system test environments where the more realistic system load is desired.

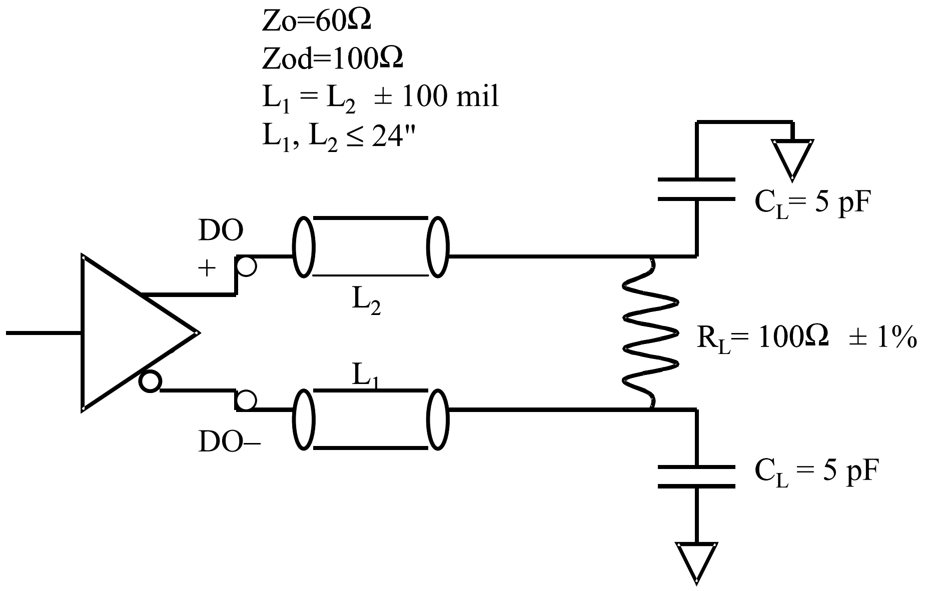


Figure 18. AC Reference System Load for Links up to 800 MT/s

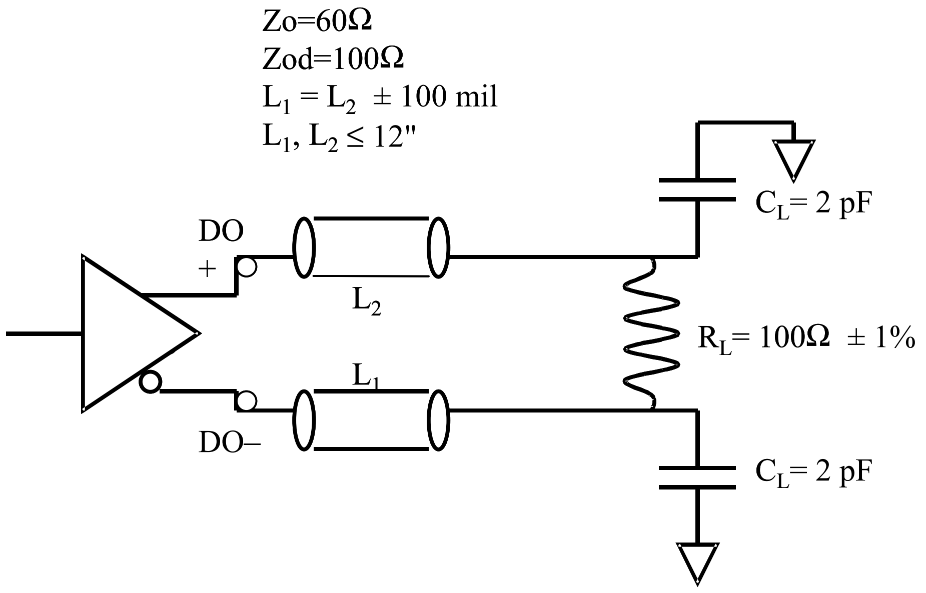


Figure 19. AC Reference System Load for Links Above 800 MT/s

Additionally, the specified values are valid and should be tested under the following AC conditions:

> Output edge rate for both rising and falling edges of both true and complement signals must be between 2 V/ns and 4 V/ns (measured in a single-ended fashion) for all links. Output edge rate should be measured at nodes DO+ and DO– between ± 200 mV from the crossing point.

These output test load circuits are defined to roughly represent a test load of the specified $C_{IN.}$

17.8.3 Output Parameter Descriptions

V_{OD} is the differential output voltage or the voltage difference between true and complement under AC conditions. V_{OD} is equal to $|DO+ – DO–|$ on the following figure.

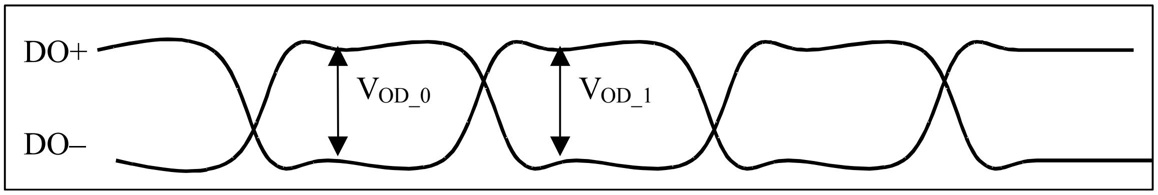


Figure 20. V_{OD} **AC**

ΔV_{OD} is the change in magnitude between the differential output voltage while driving a logic 0 and while driving a logic 1. ΔV_{OD} is equal to $V_{OD\_0} – V_{OD\_1}$.

V_{OCM} is the output common-mode voltage defined as the average of the true voltage magnitude and the complement voltage magnitude relative to ground under AC conditions. V_{OCM} is equal to (DO+ + DO–) / 2 in the following figure. V_{OCM} can be measured at any point in time, including but not limited to the crossing point, and has no periodicity requirements.

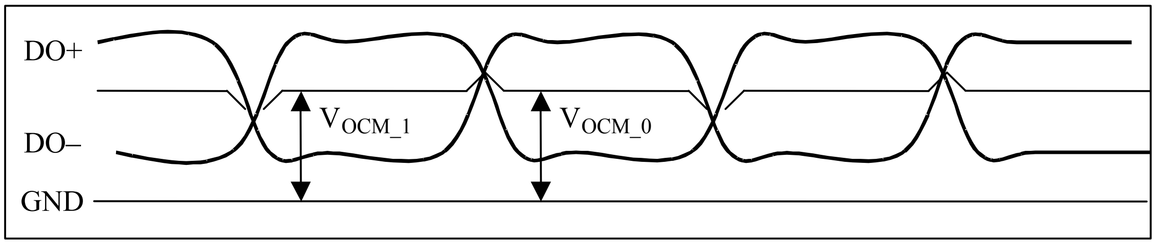


Figure 21. V_{OCM} **AC**

ΔV_{OCM} is the peak change in magnitude between the output common-mode voltage while driving a logic 0 and while driving a logic 1 under AC conditions. ΔV_{OCM} is equal to $V_{OCM\_1} – V_{OCM\_0}$.

17.9 AC Input Requirements

These parameters place requirements on the receiver and are derived from the output parameters.

17.9.1 ATE Test Environment

In a dedicated ATE test environment, the device under test should be driven by an ideal driver through ideal interconnect under ideal conditions. This implies that automated test equipment power supplies, supply distribution, and signal interconnect be designed as to provide best case operating conditions and not mimic a reference system load. This design allows the test engineer to accurately characterize the device performance and to define the production test point and guard band such that devices meet the specified characteristics in system or reference system environments.

17.9.2 Input Parameter Descriptions

V_{ID} is the input differential voltage or the voltage difference between the true and complement under AC conditions. V_{ID} is equal to $|RI+ – RI–|$ on the following figure.

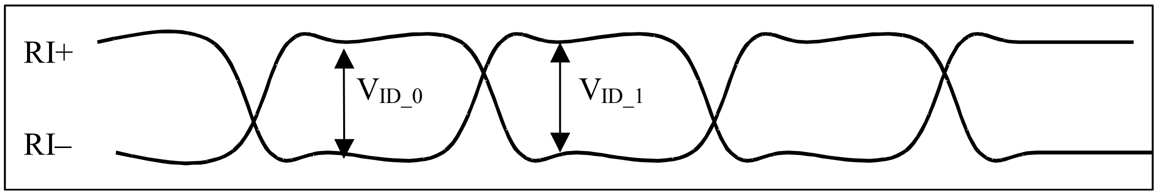


Figure 22. V_{ID} AC

ΔV_{ID} is the change in magnitude between the input differential voltage while receiving a logic 0 and while receiving a logic 1. ΔV_{ID} is equal to $V_{ID\_0} – V_{ID\_1}$.

V_{ICM} is the input common-mode voltage defined as the average of the true voltage magnitude and the complement voltage magnitude relative to ground under AC conditions. V_{ICM} can be measured at any point in time, including but not limited to the crossing point, and has no periodicity requirements. V_{ICM} is equal to (RI+ + RI–) / 2 in Figure 23 on page 199.

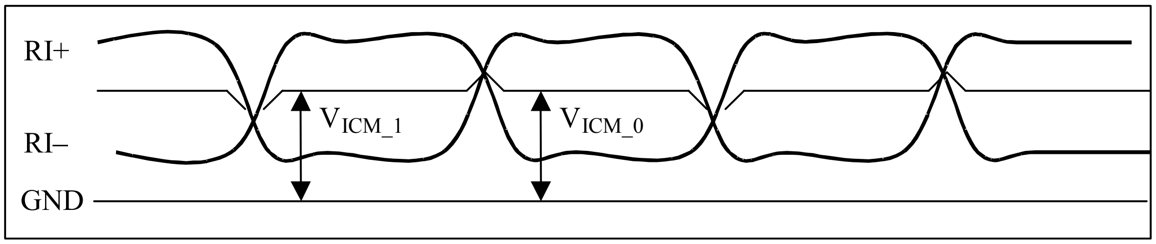


Figure 23. V_{ICM} AC

ΔV_{ICM} is the peak change in magnitude between the input common-mode voltage while driving a logic 0 and while driving a logic 1. ΔV_{ICM} is equal to $V_{ICM\_1} - V_{ICM\_0}$.

T_R is the input rising differential edge (logic 0 => logic 1) rate. T_R is measured differentially between – 150 mV and + 150 mV.

T_F is the input falling differential edge (logic 1 => logic 0) rate. T_F is measured differentially between + 150 mV and – 150 mV.

17.10 Differential Signal AC Specifications

Table 88 defines the allowed values for each of the AC characteristics. More detail regarding the derivation of these values is included in Appendix J on page 217.

Table 88. HyperTransport™ Link Differential Signal AC Specifications

Parameter	Description	Min	Typ	Max	Units
V_{OD}	Differential Output Voltage	400	600	820	mV
ΔV_{OD}	Change in V_{OD} Magnitude	–75		75	mV
V_{OCM}	Output common-mode voltage	440	600	780	mV
ΔV_{OCM}	Change in V_{OCM} magnitude	–50		50	mV
V_{ID}	Input differential voltage	300	600	900	mV
ΔV_{ID}	Change in V_{ID} magnitude	–125		125	mV
V_{ICM}	Input common-mode voltage	385	600	845	mV
ΔV_{ICM}	Change in V_{ICM} magnitude	–100		100	mV
T_R[1]	Input rising edge rate	2.0		8.0	V / ns
T_F[1]	Input falling-edge rate	2.0		8.0	V / ns

Notes:

1. *Input edge rates are measured in a ~~single-ended~~differential fashion~~±150 mV from the crossing point~~.*

18 Link Transfer Timing Characteristics

The HyperTransport link uses a source synchronous clocked transfers to transmit and receive packets across the interconnect. Transfer timing is dependent upon the devices outputs, the interconnect, and the receiver inputs to minimize skew induced between signal edges. The amount of skew directly corresponds to the link frequency that can be attained.

The HyperTransport technology defines the required output skew, the interconnect skew, and the receiver input skew required to close timing for each of the specified link frequencies. The HyperTransport link uses a simple timing methodology that accounts for simultaneous worst case combinations of uncertainties. This timing methodology is a pessimistic approach that attempts to cover all cases that could occur in operational systems. Timing is defined to provide zero additional margin, which places the requirement on transmitter, interconnect, and receiver designers to meet these specifications over all process, voltage, and temperature corners.

18.1 Signal Groups

HyperTransport link transfer timing generally describes the timing required between the CAD/CTL signal group and the associated CLK signal. The definition of signals included in these groups varies by link width.

Table 89. Signal Groups for Transfer Timing

Link Width (TX or RX)	**Group Names**	**Signals**	**Associated clock**
2-Bit (TX)	CAD/CTLOUT	CADOUT[1:0], CTLOUT	CLKOUT
4-Bit (TX)	CAD/CTLOUT	CADOUT[3:0], CTLOUT	CLKOUT
8-Bit (TX)	CAD/CTLOUT	CADOUT[7:0], CTLOUT	CLKOUT
16-Bit (TX)	CAD/CTLOUT_0	CADOUT[7:0], CTLOUT	CLKOUT[0]
	CADOUT_1	CADOUT[15:8]	CLKOUT[1]
32-Bit (TX)	CAD/CTLOUT_0	CADOUT[7:0], CTLOUT	CLKOUT[0]
	CADOUT_1	CADOUT[15:8]	CLKOUT[1]
	CADOUT_2	CADOUT[23:16]	CLKOUT[2]
	CADOUT_3	CADOUT[31:24]	CLKOUT[3]

Link Width (TX or RX)	**Group Names**	**Signals**	**Associated clock**
2-Bit (RX)	CAD/CTLIN	CADIN[1:0], CTLIN	CLKIN
4-Bit (RX)	CAD/CTLIN	CADIN[3:0], CTLIN	CLKIN
8-Bit (RX)	CAD/CTLIN	CADIN[7:0], CTLIN	CLKIN
16-Bit (RX)	CAD/CTLIN_0	CADIN[7:0], CTLIN	CLKIN[0]
	CADIN_1	CADIN[15:8]	CLKIN[1]
32-Bit (RX)	CAD/CTLIN_0	CADIN[7:0], CTLIN	CLKIN[0]
	CADIN_1	CADIN[15:8]	CLKIN[1]
	CADIN_2	CADIN[23:16]	CLKIN[2]
	CADIN_3	CADIN[31:24]	CLKIN[3]

18.2 Device Output Timing Characteristics

18.2.1 Differential Output Skew

T_{ODIFF} defines the allowable output differential skew as defined by the time difference measured in a single-ended fashion at the midpoint of the transition of the true signal and the midpoint of the transition of the complement signal.

Differential output skew is limited primarily by ΔV_{OCM} such that at the given minimum output edge rate differential skew would cause a violation of ΔV_{OCM} before violating the output differential skew specification.

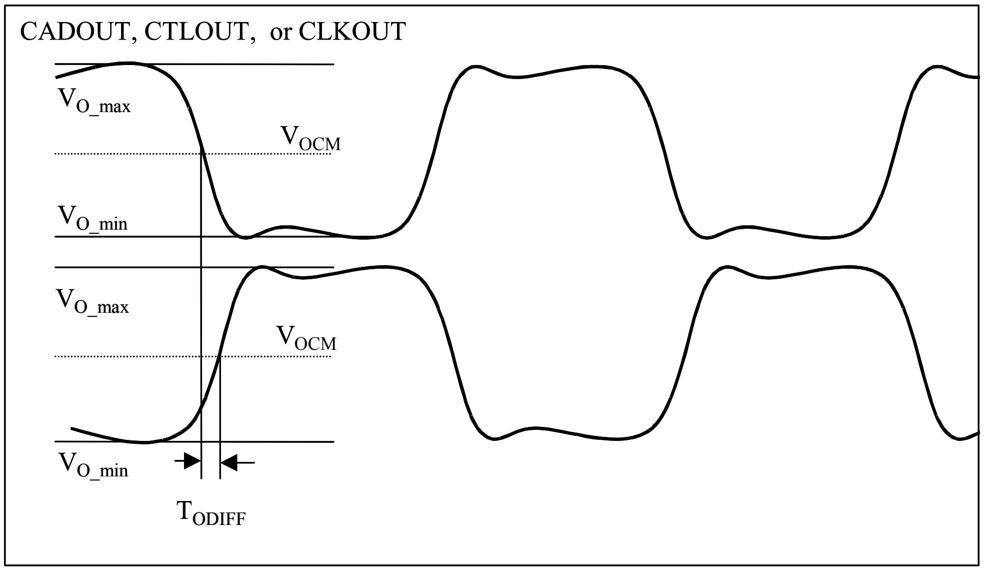


Figure 24. T_{ODIFF}

18.2.2 T_{CADV} ($T_{CADValid}$)

T_{CADV} defines the CAD/CTLOUT valid time from CAD/CTLOUT to CLKOUT or from CLKOUT to CAD/CTLOUT and is simultaneously an aggregate measurement of the accuracy of the transmitter to place the CAD/CTLOUT edges relative to CLKOUT edge, the minimum CLKOUT bit time and, the CAD/CTLOUT group skew.

Nominally, CLKOUT is driven delayed by one-half of a bit time from the CAD/CTLOUT transitions. This delay provides required setup and hold time to and from the CLKOUT edge at the receiver and therefore allows for simple data recovery. T_{CADV_min} is measured at the device pins from the crossing point of either the latest CAD/CTLOUT transition to the crossing point of the CLKOUT transition or the CLKOUT transition to the earliest CAD/CTLOUT transition. T_{CADV_max} is measured at the device pins from either the crossing point of the earliest

CAD/CTLOUT transition to the crossing point of the CLKOUT transition or the CLKOUT transition to the latest CAD/CTLOUT transition.

Because T_{CADV} is an aggregate measure of different uncertainties, it must be measured over a large number of samples and under conditions defined to maximize CADOUT/CTLOUT group skew, CLKOUT edge placement error, and CLKOUT phase compression.

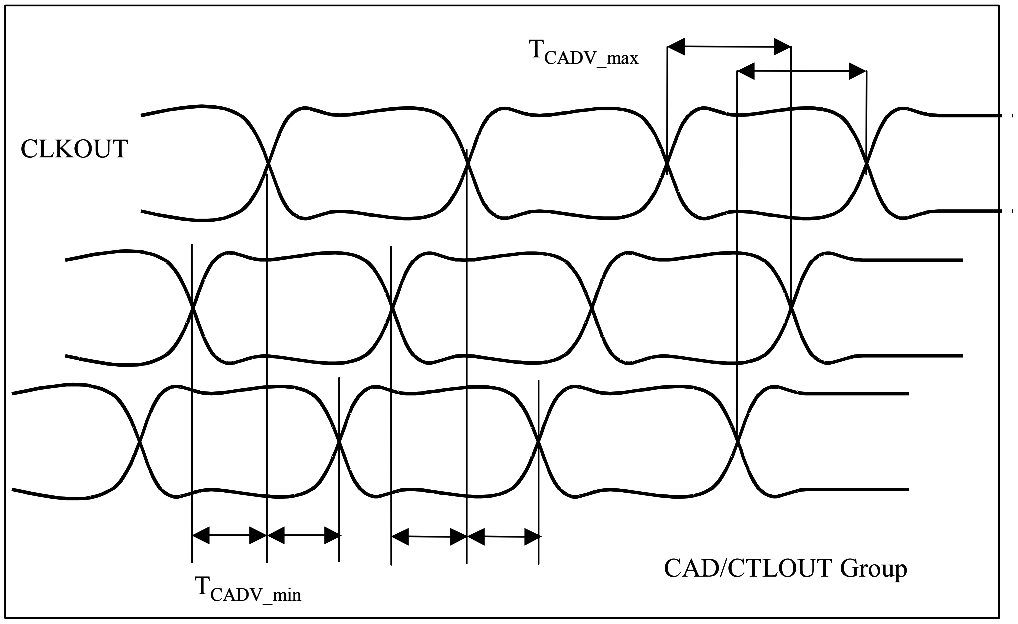


Figure 25. T_{CADV}

18.3 Device Input Timing Characteristics

18.3.1 Input Differential Skew

T_{IDIFF} defines the allowable input differential skew as defined by the time difference measured in a single-ended fashion at the midpoint of the transition of the true signal and the midpoint of the transition of the complement signal.

Differential input skew is limited primarily by ΔV_{ICM} such that at the given minimum output edge rate differential skew would cause a violation of ΔV_{ICM} before violating the output differential skew specification.

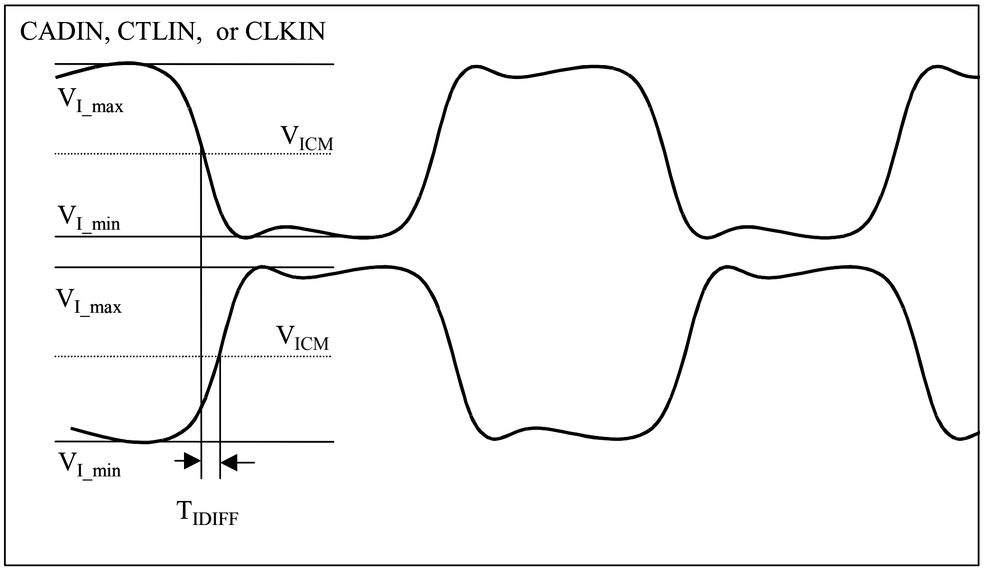


Figure 26. T_{IDIFF}

18.3.2 T_{SU} and T_{HD}

T_{SU} defines the receiver's required input setup time. T_{SU} is measured from the crossing point of the last CADIN transition to the CLKIN transition crossing point. T_{SU} accounts for receiver package skew, distribution skew, and device input setup time. T_{HD} defines the receiver's required input hold time. T_{HD} is measured from the crossing point of the earliest CADIN transition to the CLKIN transition crossing point. T_{HD} accounts for receiver package skew, distribution skew, and device input hold time. T_{SU} and T_{HD} do not necessarily cover the required time to attain V_{ID_min} (AC) at the specified minimum input edge rates. This is addressed further in Appendix M on page 232.

In the following figure, T_{SU}_max represents the maximum setup time that the device can require. This corresponds to the minimum setup time that the system can provide to the device input.

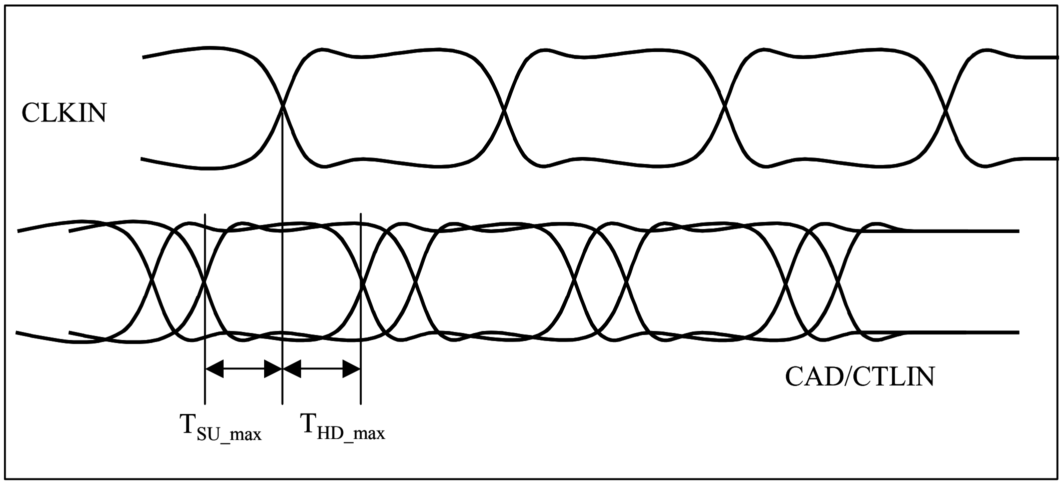


Figure 27. T_{SU} and T_{HD}

18.4 Interconnect Timing Characteristics

18.4.1 $T_{CADVRS/RH}$

$T_{CADVRS/RH}$ defines the remaining CADIN valid times to CLKIN (T_{CADVRS}) and from CLKIN to CADIN (T_{CADVRH}) measured at the receiver inputs. $T_{CADVRS/RH}$ are used as an aggregate and accumulative measure of the timing uncertainty composed of device output skew, clock edge placement error, and interconnect skew at the device inputs. As such, $T_{CADVRS/RH}$ must be measured over a large number of samples and conditions which will maximize device output skew, interconnect skew, and clock edge placement error. T_{CADVRS} is measured from the crossing point of the last transitioning CADIN signal to the crossing point of the CLKIN transitioning signal at the receiver. T_{CADVRH} is measured from the CLKIN transitioning signal to the first CADIN signal at the receiver.

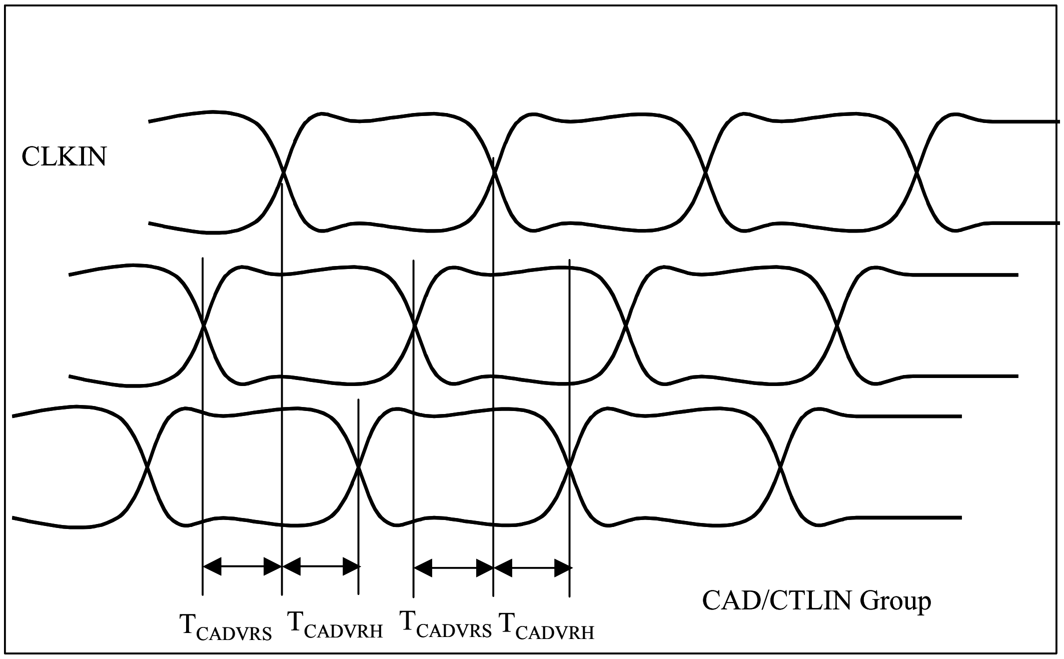


Figure 28. T_{CADVRS} / T_{CADVRH}

18.5 Transfer Timing Characteristics

Table 90 defines the allowed values for the transfer timing characteristics. More detail regarding the derivation of these values is included in Appendix K on page 219.

Table 90. HyperTransport™ Link Transfer Timing Specifications

Parameter	Description	Link Speed	Min	Max	Units
T_{ODIFF}	Output differential skew	400 MT/s 600 MT/s 800 MT/s 1000 MT/s 1200 MT/s 1600 MT/s		70 70 70 60 60 60	ps ps ps ps ps ps
T_{IDIFF}	Input differential skew	400 MT/s 600 MT/s 800 MT/s 1000 MT/s 1200 MT/s 1600 MT/s		90 90 90 65 65 65	ps ps ps ps ps ps
T_{CADV}	Transmitter output CAD/CTLOUT valid relative to CLKOUT	400 MT/s 600 MT/s 800 MT/s 1000 MT/s 1200 MT/s 1600 MT/s	695 467 345 280 234 166	1805 1200 905 720 600 459	ps ps ps ps ps ps
T_{CADVRS}	Receiver input CADIN valid time to CLKIN	400 MT/s 600 MT/s 800 MT/s 1000 MT/s 1200 MT/s 1600 MT/s	460 312 225 194 166 116		ps ps ps ps ps ps
T_{CADVRH}	Receiver input CADIN valid time from CLKIN	400 MT/s 600 MT/s 800 MT/s 1000 MT/s 1200 MT/s 1600 MT/s	460 312 225 194 166 116		ps ps ps ps ps ps

Parameter	Description	Link Speed	Min	Max	Units
T_{SU}	Receiver input setup time	400 MT/s 600 MT/s 800 MT/s 1000 MT/s 1200 MT/s 1600 MT/s	0 0 0 0 0 0	250 215 175 153 138 110	ps ps ps ps ps ps
T_{HD}	Receiver input hold time	400 MT/s 600 MT/s 800 MT/s 1000 MT/s 1200 MT/s 1600 MT/s	0 0 0 0 0 0	250 215 175 153 138 110	ps ps ps ps ps ps

18.6 Reconciling Transfer Timing and Link Frequency

The transfer timing characteristics are defined to divide the available bit time for each link frequency between the transmitter, interconnect, and receiver uncertainty. In order to reconcile these numbers into an operating link bit rate, the following relationships are used to simultaneously ensure both input setup and input hold time for data recovery under worst case conditions.

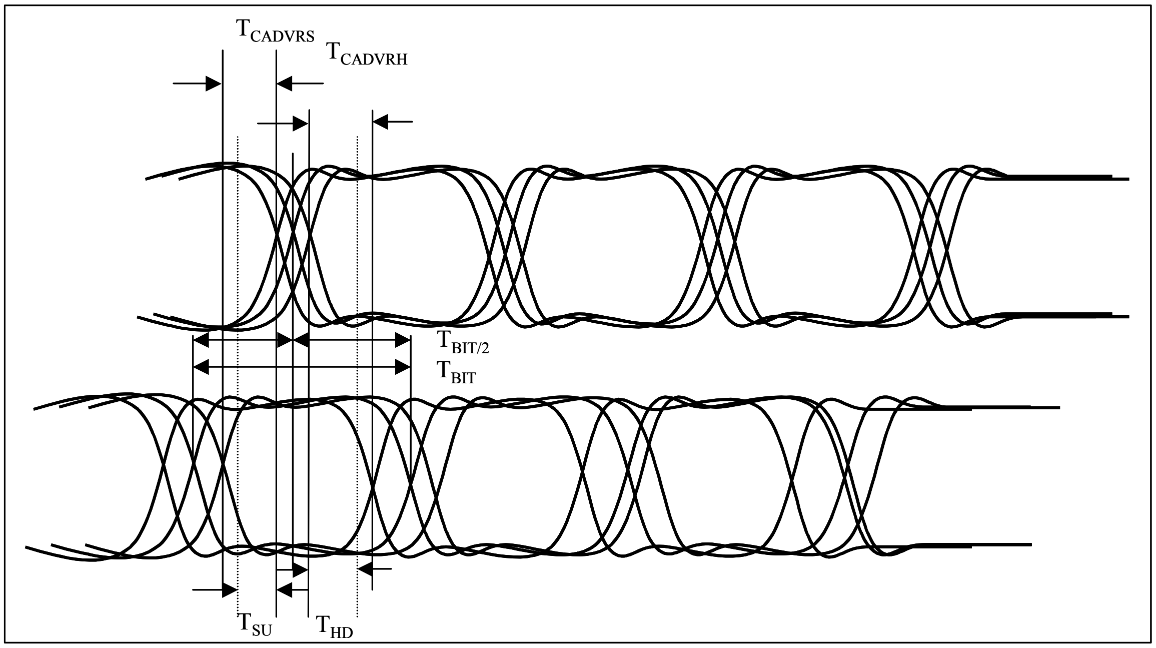


Figure 29. Reconciling Link Transfer Timing

From Figure 29, the achievable maximum link frequency (minimum T_{BITMIN}) occurs when $T_{SU} <= T_{CADVRS}$ and $T_{HD} <= T_{CADVRH}$ simultaneously such that

$$T_{BITMIN} >= T_{BIT}/2 - T_{CADVRS} + T_{SU} + T_{BIT}/2 - T_{CADVRH} + T_{HD}$$

where T_{BIT} is the bit rate at which T_{CADVRS} and T_{CADVRH} are measured or calculated.

Since all systems under certain conditions will exhibit asymmetries in T_{CADVRS} and T_{CADVRH}, while the requirements for T_{SU} and T_{HD} remain symmetrical, either half of the equation could limit the operating link frequency.

$T_{BITMIN}/2 >= T_{BIT}/2 - T_{CADVRS} + T_{SU}$

$T_{BITMIN}/2 >= T_{BIT}/2 - T_{CADVRH} + T_{HD}$

Or combined in a different fashion.

$T_{BITMIN} >= MAX ((T_{BIT}/2 - T_{CADVRS} + T_{SU}) * 2, (T_{BIT}/2 - T_{CADVRH} + T_{HD}) * 2)$

Table 91 presents results calculated using this relationship. This relationship is further defined in Appendix K on page 219.

Table 91. Calculated Minimum Link Bit Times

Parameter	Description	Link Speed	Min	Max	Units
T_{BIT_min}	Calculated minimum required BIT time using the above relationship	400 MT/s 600 MT/s 800 MT/s 1000 MT/s 1200 MT/s 1600 MT/s	2080 1473 1150 919 777 613		ps ps ps ps ps ps

19 Phase Recovery Timing Characteristics

In addition to recovering the data from the interconnect, the receiver is responsible for passing this data from the link transmit clock domain to the device specific receive clock domain.

In general, clock forwarding data recovery methods require a FIFO in the receiver that is written in the transmit clock domain and read in the receive clock domain. The design and operation of this FIFO must account for the dynamic variations in phase between the transmit clock domain (TCLK) and the receive clock domain (RCLK). The FIFO depth must be large enough to store all transmitted data until it has been safely read into the receive clock domain. The separation from the write pointer to which the FIFO data is written and the read pointer from which the FIFO location is read (write-to-read separation) must be large enough to ensure the FIFO location can be read into the receive clock domain. Additionally, the separation from the read pointer from which the FIFO location is read to the write pointer location at which the FIFO location is overwritten (read-to-overwrite separation) must be large enough to avoid the FIFO location being overwritten prior to being read into the receive clock domain. The pointer initialization occurs at link initialization and consists of initializing the write pointer and setting the read pointer to a location that simultaneously satisfies both conditions stated above. Whether the read pointer location remains static or is periodically updated depends upon the clocking mode of the link.

19.1 Receiver Modes of Operation

~~The *HyperTransport I/O Link Protocol Specification*, order# 23888,~~Section 11.1 defines three different clocking modes of the receiver: synchronous, pseudo-synchronous, and asynchronous. Only the synchronous clocking mode is fully specified in this revision of this specification.

19.1.1 Synchronous Operation

In synchronous mode, each transmit clock must be derived from the same time base as the receive clock in the device to which it is connected. This eliminates any frequency difference between the transmit and receiver clock domains. A receive FIFO implemented to support synchronous clocking mode needs only to initialize the read pointer at link initialization. No additional updates to the read pointer are necessary.

19.1.2 Pseudo Synchronous Operation

In pseudo-synchronous mode, each transmit clock must be derived from the same time base as the receive clock in the device to which it is connected. The HyperTransport link output clock frequency for either device may be arbitrarily lower than the frequency programmed into its

LinkFreq register, and must not exceed the maximum allowed receive clock frequency in the other device. The maximum allowed receive clock frequency of a link is the highest frequency indicated in the frequency capability register. A receive FIFO implemented to support pseudo-synchronous mode must both initialize the read pointer at link initialization and must periodically be kept from incrementing in order to maintain the required read-to-write pointer separation. This clocking mode will be fully specified in a future revision of this specification.

19.1.3 Asynchronous Operation

In asynchronous mode, each transmit clock need not be derived from the same time base as the receive clock in the device to which it is connected. In order to cope with the frequency error due to running nominally matched transmitter/receiver pairs from different time bases, the maximum CLKOUT frequency for one device can exceed the maximum receive clock frequency in the other device by no more than 2000 parts per million. An example of how this might be implemented is included in ~~the *HyperTransport I/O Link Protocol Specification*, order# 23888~~Section 11.3. This clocking mode will be fully specified in a future revision of this specification.

19.2 Phase Recovery Timing Variations

The required FIFO depth and write-to-read pointer separation are dependent upon the following long term timing uncertainties.

Temperature variations of active circuitry along the clock generation and distribution paths:

Local and temporal temperature variations will affect the phase error, duty cycle, and phase compression of both the transmitter and receiver PLLs. Temperature variations will affect the delay with which the various clocks are distributed.

Voltage variations of active circuitry along the clock generation and distribution paths:

Local and temporal supply voltage variations (within the specified limits) will affect the phase error, duty cycle, and phase compression (jitter) of both the transmitter and receiver PLLs. Voltage variations also affect the distribution path delays.

Accumulated phase error in any of the clock generating phase lock loops:

The receiver and transmitter PLLs will accumulate phase error relative to the reference clock due to inherent error in generating and comparing the voltages nodes to generate the desired output frequency.

Uncorrelated noise between TCLK and RCLK:

The transmit clock and the receive clock will contain uncorrelated noise induced by various means (crosstalk, simultaneous switching outputs, etc) that will affect their relative phase error.

Reference clock spread spectrum clocking phase error induced by distribution path variations:

Spread spectrum clocking techniques used to lessen a system's peak electromagnetic emissions will induce phase error between the transmit and receive clock by the modulation frequency and the difference in delay through the distribution paths of each clock domain.

19.2.1 Uncertainty When Initializing the Pointers

TCLK to RCLK phase error during initialization:

~~The *HyperTransport I/O Link Protocol Specification*, order# 23888,~~Section 12.2 states that the read pointer is initialized after the CTL/CADOUT signals are sampled low in the core clock domain. It cannot be assumed, however, that the initial transition of the CTL/CADOUT signal was driven into the FIFO with a CLKOUT edge that had minimum or maximum skew relative to any RCLK edge. Therefore, the receive FIFO must be sized and the read pointer initialized to cover both of the following cases:

- The initial CLKOUT is driven at the earliest possible time with respect to RCLK, and subsequent edges are driven at the latest possible time with respect to RCLK (and therefore write data into the FIFO later) and still require the minimum write-to-read pointer separation.
- The initial CLKOUT is driven at the latest possible time with respect to RCLK, and subsequent edges are driven at the earliest possible time with respect to RCLK (and therefore write data into the FIFO earlier) and still require the minimum read-to-overwrite pointer separation.

Accounting for both of these cases in the FIFO design requires that the FIFO depth account for two times the dynamic timing variations due to temperature, voltage, and noise changes since the read pointer initialization methods could be in error by, at most, the sum of these variations.

Inherent sampling error in detecting the initial CTLIN transition in the receive clock domain:

Sampling the CADIN/CTLIN deassertion in the receive clock domain will have a synchronization error of up to 1 receive clock bit time for most implementations. This sampling error will result in the pointers being initialized up to one receive clock early or late from the ideal timing standard.

19.2.2 Other Factors Affecting FIFO Size and Read Pointer Separation

Frequency and/or width translation using the receive FIFO:

The FIFO in some implementations is used to translate TCLK to RCLK frequency and link to core width. Other implementations will translate from link speed and width to some slower and wider intermediary operation prior to the FIFO. For implementations that use the FIFO to perform this translation, the FIFO must be made large enough to absorb and store a full receive line until that full line can be read into the receive clock domain. For example, a transmit date rate of 1600 MT/s at one byte wide writing a FIFO that is read with a core clock of 200 MT/s at 8 bytes wide would need to allow 7 additional FIFO locations (at transmit rate) to store the additional 7 bytes until read into the receive clock domain without being overwritten. Additionally, the FIFO needs to contain 7 locations (at transmit rate) as to ensure that all 8 bytes of data had been successfully written prior to reading.

Cross byte skew between CLKIN signals for multibyte link implementations:

The receive FIFO size must also account for variations in CLKIN signals for multibyte link implementations. The uncertainty between any two CLKIN signals must be added to the overall TCLK uncertainty in order to ensure that data written into FIFO with the CLKIN having the largest accumulated phase error to the receive clock can be read properly. This skew contains both a constant (path length mismatch) and a time variant portion (voltage, temperature, and noise dependent).

CADIN/CTLIN synchronization time:

Since sampling the initial CTL/CADOUT signal in the RCLK domain will have some synchronization delay, this device specific synchronization delay should be removed from the initial read pointer.

19.3 Phase Recovery Timing Characteristics

In Table 92 on page 215:

Trefclk defines maximum reference clock phase error allowed between the transmitter and receiver.

TxmtPLL defines the maximum phase error of the transmit clock due to PLL temperature variations, voltage variations, and accumulated phase error.

Txmttransfer defines the maximum phase error of the transmit clock due to noise.

Tbytelanevar defines the maximum time variant phase error between CLKIN signals to the receiver and therefore the maximum additional phase error between TCLK and RCLK.

Tbytelaneconst defines the maximum constant phase error between CLKIN signals to the receiver due to distribution path length mismatch.

TrcvPLL defines the maximum phase error of the receive clock due to PLL temperature variations, voltage variations, and accumulated phase error.

Trcvtransfer defines the maximum phase error in the transmitter clock due to uncertainty on the receiver package, receiver pad, and receiver clock distribution.

Table 92. HyperTransport™ Link Phase Recovery Timing Characteristics

Parameter	Description	Link Speed	Min	Max	Units
Trefclk	Uncertainty in CLKIN relative to RCLK due to reference clock variations between transmitter and receiver	Any		733	ps
TxmtPLL	Uncertainty in CLKIN relative to RCLK due to accumulated phase error due to PLL run-out and low frequency supply variations	Any		3500	ps
Txmttransfer	Uncertainty in CLKIN relative to RCLK due to transmitter and interconnect transfer effects	400 MT/s		918	ps
		600 MT/s		592	ps
		800 MT/s		469	ps
		1000 MT/s		358	ps
		1200 MT/s		295	ps
		1600 MT/s		228	ps
Tbytelanevar	Variable uncertainty in CLKIN relative to RCLK due to multiple versions of CLKIN	Any		250	ps
Tbytelaneconst	Constant uncertainty in CLKIN relative to RCLK due to CLKIN distribution path length mismatch	Any		1000	ps
TrcvPLL	Uncertainty in RCLK relative to CLKIN due to accumulated phase error due to PLL run-out and low frequency supply variations	Any		3500	ps
Trcvtransfer	Uncertainty in CLKIN relative to RCLK due to receiver package and receiver transfer effects	400 MT/s		425	ps
		600 MT/s		250	ps
		800 MT/s		188	ps
		1000 MT/s		130	ps

Parameter	Description	Link Speed	Min	Max	Units
		1200 MT/s		109	ps
		1600 MT/s		81	ps

19.4 Reconciling Phase Recovery Timing to Receiver FIFO Depth and Read Pointer Initialization

19.4.1 Read Pointer Initialization

The initial read-to-write pointer separation must account for all of the factors outlined above. While many of these factors are implementation specific, a nominal implementation (TCLK and RCLK of equal frequency) would initialize the read pointer according to the following relationship:

Maximum time variant phase error = Trefclk + TxmtPLL + Txmttransfer + Tbytelanevar + TrcvPLL + Trcvtransfer

Maximum constant phase error = Tbytelaneconst

Maximum CADIN/CTLIN sampling error = 1 RCLK bit time

Minimum read-to-write pointer separation > Maximum time variant phase error + Maximum CADIN/CTLIN sampling error) + ½ Maximum additional constant phase error

19.4.2 Minimum FIFO Depth

The minimum FIFO depth chosen for any implementation must be sized to accommodate both the read-to-write pointer separation and the read-to-overwrite pointer separation:

FIFO phase error > 2 * (Minimum read-to-write pointer) + Tbit

Minimum FIFO lines (TCLK) = FIFO phase error / Tbit (rounded up to whole integer)

Electrical Interface Appendices

J DC and AC Characteristics and Relationships

J.1 DC Parameters

The DC characteristics of the HyperTransport™ link are derived from the allowed variations in V_{LDT}, R_{ON}, and $R_{TT.}$ The relationships used for V_{OD} and V_{OCM} are shown below.

Note: *VLDT_min and VLDT_max are assumed to be 1.1V and 1.3V respectively to account for the minimum and maximum supply levels at the driver or receiver.*

V_{OD} DC values are calculated from the following relationships:

$V_{OD\_min} > V_{LDT\_min} * R_{TT\_min} / (R_{ON\_max} + R_{TT\_min} + R_{ON\_max})$

$V_{OD\_max} < V_{LDT\_max} * R_{TT\_max} / (R_{ON\_min} + R_{TT\_max} + R_{ON\_min})$

V_{OCM} DC values are calculated from the following relationships:

$V_{OCM\_min} > V_{LDT\_min} * (((R_{TT\_min} + R_{ON\_min})/(R_{TT\_min} + R_{ON\_min} + R_{ON\_max})) + (R_{ON\_min} / (R_{ON\_min} + R_{TT\_min} + R_{ON\_max}))) / 2$

$V_{OCM\_max} < V_{LDT\_max} * (((R_{TT\_min} + R_{ON\_max})/(R_{TT\_min} + R_{ON\_max} + R_{ON\_min}))+(R_{ON\_min})/(R_{ON\_min} + R_{TT\_max} + R_{ON\_min})) / 2$

J.2 Relationships Between AC and DC Parameters

The relationships between AC and DC parameters allow for the existence of AC noise on the signals in addition to the maximum V_{LDT} noise allowed. The following table shows the considered AC power supply noise and the remaining signal noise margin. Note that the minimum specifications of V_{OD}, V_{OCM}, V_{ID}, and V_{ICM} already account for –100mV of supply noise from the nominal.

Table 93. Relationships Between AC and DC Parameters

Parameter	Min (DC)	Min (AC)	Signal Noise
V_{OD}	495 mV	400 mV	95 mV
V_{OCM}	495 mV	440 mV	55 mV
V_{ID}	200 mV	300 mV	100 mV
V_{ICM}	440 mV	385 mV	55 mV

J.3 Relationships Between Output and Input Parameters

The relationships between output and input parameters comprehends the inclusion of noise and attenuation on the interconnect. The following table shows the allow degradation in each of the output parameters from transmitter to receiver.

Table 94. Relationships Between Output and Input Parameters

Parameter	Output	Input	Loss
VOD (DC)	495 mV	200 mV	295 mV
VOD (AC)	400 mV	300 mV	100 mV
VOCM (DC)	495 mV	440 mV	55 mV
VOCM (AC)	440 mV	385 mV	55 mV

K Detailed Transfer Timing Budget

The following diagram provides a conceptual view of the clock generation and distribution scheme for the HyperTransport link along with the CADOUT/CTLOUT transmission path from transmitter to receiver. From this diagram the different types of system level uncertainties, transmitter uncertainties, interconnect uncertainties, and receiver uncertainties can be identified.

K.1 HyperTransport™ Link Transmitter

The HyperTransport link transmitter is responsible for driving the CTL/CADOUT wires within specified uncertainties. Additionally, the transmitter is responsible for shifting CLKOUT 90° from CADOUT/CTLOUT. This 90° phase shift provides the required setup and hold time at the receiver for simple data recovery.

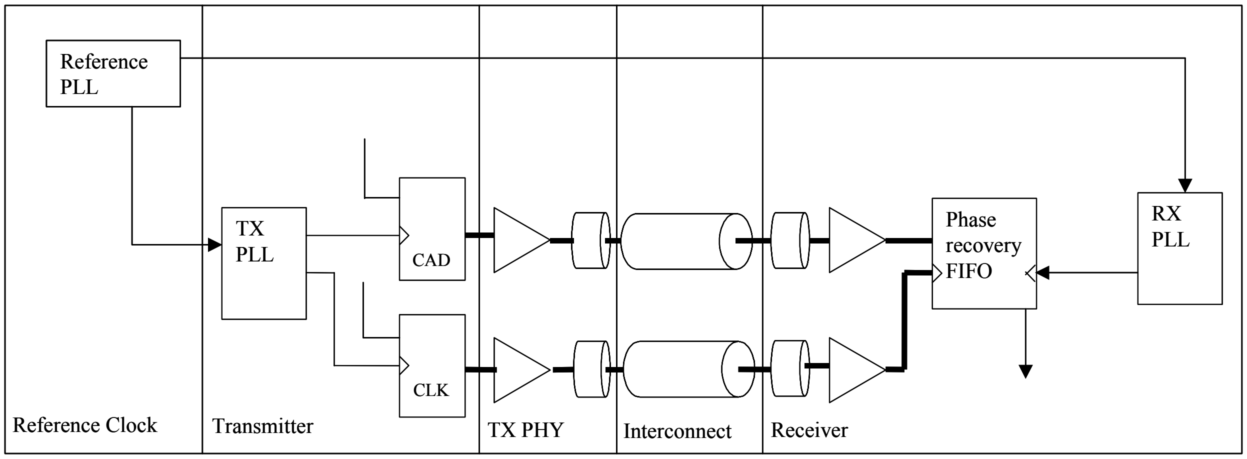


Figure 30. Representative Transmitter and Receiver

K.2 Differential Skew

Differential skew (skew between _H and _L signals within a pair) is caused by mismatch in distribution paths and crosstalk, ISI, SSO and other electrical disturbances which affect _H differently than _L and visa versa.

Table 95. Differential Skew

Symbol	**Description**	**400 Mb/s**	**600 Mb/s**	**800 Mb/s**	**1000 Mb/s**	**1200 Mb/s**	**1600 Mb/s**	**Unit**
DiffTXskew	True to complement output skew	50	50	50	50	50	50	ps
Diffpkgskew	True to complement package route mismatch	20	20	20	10	10	10	ps
DiffPCBskew	True to complement PCB route mismatch	20	20	20	5	5	5	ps

K.3 Transmitter Clock Uncertainties

The transmit clock is normally generated on the transmitter by a PLL that uses a reference clock generated in a manner consistent with the intended clocking mode of the link. This PLL will have the following uncertainties associated with generating and distributing the required transmit clocks for the transmitter.

Table 96. Transmitter Clock Uncertainties

Symbol	Description	400 Mb/s	600 Mb/s	800 Mb/s	1000 Mb/s	1200 Mb/s	1600 Mb/s	Unit
TPLLdc	2% duty cycle variation between opposing edges over 1 bit time	100	67	50	40	33	25	ps
TPLLjtr	Uncertainty in subsequent internal transmit clocks due to PLL variation between any 2 edges including that contributed by reference clock SSC techniques.	150	67	50	20	17	13	ps
TPLLerror	Uncertainty in subsequent CADOUT edges due PLL accumulated phase error (≤ 20 ps/ns over 1 bit time) in the internal transmit clock	50	33	25	20	17	13	ps
TPLLsup	Uncertainty in subsequent internal transmit clocks due to temporal PLL power supply modulation (50 ps/ns)	125	83	63	50	42	31	ps
Tclkskew	Uncertainty in the CLKOUT relative to CADOUT caused by load variations between the 90 degree phase shifted clock relative to the 0 degree clock	20	20	20	10	10	10	ps

K.4 Transmitter PHY Uncertainties

The transmitter will have uncertainties associated with distributing the clock and driving both CADOUT/CTLOUT and the phase delayed CLKOUT. The following skews associated with the PHY are expected.

Table 97. Transmitter PHY Uncertainties

Symbol	Description	400 Mb/s	600 Mb/s	800 Mb/s	1000 Mb/s	1200 Mb/s	1600 Mb/s	Unit
TPHYskew	Uncertainty in CADOUT or CLKOUT relative to other CADOUTs due to internal transmit clock distribution variation between any 2 clock loads	60	50	40	35	30	25	ps
TPHYtmp	Uncertainty in CADOUT or CLKOUT relative to CADOUT caused by driver skew variation due to process and local temperature differences	60	50	40	35	30	25	ps
TPHYssn	Uncertainty in CADOUT or CLKOUT relative to CADOUT caused by driver skew variation due to simultaneous switching outputs	100	50	40	35	30	25	ps
TPHYsup	Uncertainty in CADOUT or CLKOUT relative to CADOUT caused by driver variation due to temporal power supply modulation (50 ps/ns)	63	42	31	25	21	16	ps

K.5 Transmitter Package Skew

The transmitter package will induce additional skew between signals associated with the variation in signal trace lengths. Note that there is a requirement to match the CLKOUT package trace length to the median length of the CADOUT/CTLOUT trace lengths to ensure an amount of symmetry between the uncertainties seen in relation between these two types of signals. Both the allowed CADOUT/CTLOUT skew induced by the overall package trace length mismatch and the relative CLKOUT to CADOUT/CTLOUT skew are listed.

Table 98. Transmitter Package Skew

Symbol	**Description**	**400 Mb/s**	**600 Mb/s**	**800 Mb/s**	**1000 Mb/s**	**1200 Mb/s**	**1600 Mb/s**	**Unit**
TPKGskew	Uncertainty in CAD/CTLOUT relative to CLKOUT due to transmitter PKG trace length mismatch	50	40	25	20	13	10	ps
TPKGskewcad	Uncertainty in CAD/CTLOUT relative to other CAD/CTLOUT due to transmitter PKG trace length mismatch	100	80	50	40	26	20	ps

K.6 Receiver Package Skew

Receiver package skew is expected to be identical to the transmitter package skew.

Table 99. Receiver Package Skew

Symbol	**Description**	**400 Mb/s**	**600 Mb/s**	**800 Mb/s**	**1000 Mb/s**	**1200 Mb/s**	**1600 Mb/s**	**Unit**
RPKGskew	Uncertainty in CADIN relative to CLKIN due to receiver PKG trace length mismatch	50	40	25	20	13	10	ps
RPKGskewcad	Uncertainty in CADIN relative to other CADIN due to transmitter PKG trace length mismatch	100	80	50	40	26	20	ps

K.7 PCB Skew

The PCB will induce skew due to both transmission line effects and route length mismatch. Note again the requirement to match the PCB trace route length of the CLKOUT transmission line to the median of the CADOUT/CTLOUT lengths to which it is associated.

Table 100. PCB Skew

Symbol	**Description**	**400 Mb/s**	**600 Mb/s**	**800 Mb/s**	**1000 Mb/s**	**1200 Mb/s**	**1600 Mb/s**	**Unit**
TPCBskew	Uncertainty in CADIN relative to CLKIN due to PCB trace length mismatch	50	50	30	20	15	10	ps
TPCBskewcad	Uncertainty in CADIN relative to other CADIN due to PCB trace length mismatch	100	100	60	40	30	20	ps
TSKTskew	Uncertainty in CADIN relative to other CADIN due to socket/connector skew	150	60	40	20	15	10	ps
Txtalk	Uncertainty in CADIN relative to other CADIN due to push out or pull in caused by interconnect cross talk	100	50	40	33	25	20	ps
Tisi	Uncertainty in CADIN relative to other CADIN due to push out or pull in caused by inter-symbol interference	170	150	130	100	80	60	ps

K.8 Receiver Setup and Hold times

The receiver setup and hold times are considered to contain all uncertainties on the die. This may include receiver pad skew, clock distribution skew, and target flip-flop setup and hold time requirements across all process corners, and operating voltage and temperature conditions.

Table 101. Receiver Setup and Hold Time

Symbol	**Description**	**400 Mb/s**	**600 Mb/s**	**800 Mb/s**	**1000 Mb/s**	**1200 Mb/s**	**1600 Mb/s**	**Unit**
Rsetup or Rhold	Receiver setup or hold time. [1]	200	175	150	133	125	100	ps
Notes: *1. Receiver setup or hold time includes all receiver skew, distribution skew, and flip-flop setup or hold time.*								

K.9 Transfer Timing Example

The following table gives a summary of the expected uncertainties for the specified link frequencies. These numbers are simple sums of the listed parameters.

Table 102. Transfer Timing Overview

Transfer Timing	**400 Mb/s**	**600 Mb/s**	**800 Mb/s**	**1000 Mb/s**	**1200 Mb/s**	**1600 Mb/s**	**Unit**
Transmitter clock uncertainty	435	260	198	140	118	91	ps
Transmitter PHY Uncertainties	283	192	151	130	111	91	ps
Transmitter PKG Skew	50	40	25	20	13	10	ps
Motherboard PCB Skew (time variant)	420	260	210	153	120	90	ps
Motherboard PCB Skew (route mismatch)	50	50	30	20	15	10	ps
Receiver PKG Skew	50	40	25	20	13	10	ps
Receiver Setup and Hold Time	200	175	150	133	125	100	ps

In order to derive the specified timing transfer parameters, the following relationships must be used:

$T_{CADV\_min} = T_{BIT}/2$ – Transmitter PHY uncertainties - Transmitter PKG Skew – Transmitter clock uncertainty /2

$T_{CADV\_max} = T_{BIT}/2$ + Transmitter PHY uncertainties - Transmitter PKG Skew – Transmitter clock uncertainty /2

$T_{CADVRS} = T_{CADV\_min}$ – Motherboard PCB skew / 2 (both time variant and route mismatch)

$T_{CADVRH} = T_{CADV\_min}$ – Motherboard PCB skew / 2 (both time variant and route mismatch)

$T_{BIT\_min} = \text{MAX}((T_{BIT}/2 - T_{CADVRS} + T_{SU}) * 2, (T_{BIT}/s - T_{CADVRH} + T_{HD}) * 2)$

L Detailed Phase Recovery Timing Budget

L.1 System Reference Clock Uncertainties

The reference clock distributed to the transmitter will vary from that distributed to the receiver by the amount of output skew in the clock generator and any time variant skew induced by the distribution scheme of the system.

The input reference clocks to the transmitter and receiver must be derived from the same time base to uphold the requirements of the synchronous clocking mode. Additionally, the time variable skew between the input reference clock edges must be specified and included in the overall CLKIN to RCLK uncertainty.

Table 103. TX to RX Input Reference Clock Distribution Uncertainty

Parameter	**Type**	**Description**	**400 Mb/s**	**600 Mb/s**	**800 Mb/s**	**1000 Mb/s**	**1200 Mb/s**	**1600 Mb/s**	**Unit**
Tclk2Rclkskew	Time variant	Reference clock distribution variation due to voltage, temperature, or data dependencies	400	400	400	400	400	400	ps
Tclk2Rclkssc	Time variant	Accumulated phase error due to SSC reference clock and 20 ns mismatch distribution path. Accumulates at 125ns/7.5uS	333	333	333	333	333	333	ps

L.2 Transmitter CLKOUT Uncertainties

L.2.1 Transmitter PLL Variations

The transmitter PLL contributes phase error to the link transfer clock through long term accumulated phase error between the reference clock and CLKOUT due to low frequency power supply drift and PLL long term accumulated error.

The long-term variations in CLKIN relative to RCLK must be considered in the design of the receive FIFO and are characterized as follows.

Table 104. Transmitter PLL Variations

Parameter	**Type**	**Description**	**400 Mb/s**	**600 Mb/s**	**800 Mb/s**	**1000 Mb/s**	**1200 Mb/s**	**1600 Mb/s**	**Unit**
TPSltdrift	Time variant	Uncertainty in TCLK due to long term power supply drift	1000	1000	1000	1000	1000	1000	ps
TPLLlterror	Time variant	Uncertainty in TCLK due to transmitter PLL long term (Accumulated) phase error	2000	2000	2000	2000	2000	2000	ps
TPLLtmp	Time variant	Uncertainty in TCLK due to temperature variations	500	500	500	500	500	500	ps

L.2.2 Transmitter and Link Transfer Variations

The transmitter clock error (accumulated over a single bit time), the transmitter PHY, and the interconnect contribute small amounts of phase error into the link transfer clock domain through all of the parameters included in the link transfer timing.

Table 105. Transmitter and Link Transfer Variations

Symbol	**Type**	**Description**	**400 Mb/s**	**600 Mb/s**	**800 Mb/s**	**1000 Mb/s**	**1200 Mb/s**	**1600 Mb/s**	**Unit**
TPLLdc	Time variant	2% duty cycle variation between opposite edges	100	67	50	40	33	25	ps
TPLLjtr	Time variant	Uncertainty in subsequent internal transmit clocks due to PLL variation between any 2 edges including that contributed by reference clock SSC techniques.	150	67	50	20	17	13	ps
TPLLerror	Time variant	Uncertainty in subsequent CADOUT edges due PLL accumulated phase error (≤ 20 ps/ns over 1 bit time) in the internal transmit clock	50	33	25	20	17	13	ps
TPLLsup	Time variant	Uncertainty in subsequent internal transmit clocks due to temporal PLL power supply modulation (50 ps/ns)	125	83	63	50	42	31	ps
TPHYtmp	Time variant	Uncertainty in CADOUT or CLKOUT relative to CADOUT caused by driver skew variation due to process and local temperature differences	60	50	40	35	30	25	ps
TPHYssn	Time variant	Uncertainty in CADOUT or CLKOUT relative to CADOUT caused by driver skew variation due to simultaneous switching outputs	100	50	40	35	30	25	ps

Symbol	**Type**	**Description**	**400 Mb/s**	**600 Mb/s**	**800 Mb/s**	**1000 Mb/s**	**1200 Mb/s**	**1600 Mb/s**	**Unit**
TPHYsup	Time variant	Uncertainty in CADOUT or CLKOUT relative to CADOUT caused by driver variation due to temporal power supply modulation (50 ps/ns)	63	42	31	25	21	16	ps
Txtalk	Time variant	Uncertainty in CADIN relative to other CADIN due to push out or pull in caused by interconnect cross talk	100	50	40	33	25	20	ps
Tisi	Time variant	Uncertainty in CADIN relative to other CADIN due to push out or pull in caused by inter-symbol interference	170	150	130	100	80	60	ps

L.2.3 Transmitter Cross Byte Lane Variations

In systems implementing multibyte HyperTransport links the cross byte lane skew must be considered in the design of the receive FIFO and is characterized as follows.

Table 106. CLKOUT Byte Lane Uncertainty

Parameter	**Type**	**Description**	**400 Mb/s**	**600 Mb/s**	**800 Mb/s**	**1000 Mb/s**	**1200 Mb/s**	**1600 Mb/s**	**Unit**
Tclk2Tclkskew	Time variant	Transmit clock skew between TCLKs for each byte lane at the receiver	250	250	250	250	250	250	ps
Rclk2Rclkskew	Invariant	Absolute receiver clock skew between different CLKOUT for each byte lane	1000	1000	1000	1000	1000	1000	ps

L.3 Receiver CLKIN Uncertainties

L.3.1 Receiver PLL Variations

The receiver PLL contributes phase error into the core clock domain through long term accumulated phase error between the reference clock and RCLK due to low frequency power supply drift and PLL long term accumulated error.

Table 107. Receiver PLL Variations

Parameter	Type	Description	400 Mb/s	600 Mb/s	800 Mb/s	1000 Mb/s	1200 Mb/s	1600 Mb/s	Unit
RPSltdrift	Time variant	Uncertainty in RCLK due to long term power supply drift	1000	1000	1000	1000	1000	1000	ps
RPLLlterror	Time variant	Uncertainty in RCLK due to Receiver PLL long term (Accumulated) phase error	2000	2000	2000	2000	2000	2000	ps
RPLLtmp	Time variant	Uncertainty in RCLK due to temperature variation	500	500	500	500	500	500	ps

L.3.2 Receiver Transfer Variations

The receiver will contribute small amounts of phase error in the received CLKIN due to distribution effects.

Table 108. Receiver Transfer Variations

Parameter	Type	Description	400 Mb/s	600 Mb/s	800 Mb/s	1000 Mb/s	1200 Mb/s	1600 Mb/s	Unit
Rclksup	Time variant	Uncertainty in RCLK due to power supply modulation	125	83	63	50	42	31	ps
RPLLdc	Time variant	Uncertainty in RCLK due to receiver PLL duty cycle variation between opposite edges	100	67	50	40	33	25	ps
RPLLjtr	Time variant	Uncertainty in RCLK due to PLL variation between any 2 edges including that contributed by reference clock SSC techniques	150	67	50	20	17	13	ps
RPLLerror	Time variant	Uncertainty in RCLK due to PLL variation due to accumulated phase error (≤ 20 ps/ns)	50	33	25	20	17	13	ps

L.4 CADIN/CTLIN Sampling Error

The CADIN/CTLIN sampling error is device specific but amounts to one receive clock bit time. These numbers assume that TCLK is equal to RCLK which is neither required or an optimal implementation in many cases.

Table 109. CTL Sampling Error

Parameter	**Type**	**Description**	**400 Mb/s**	**600 Mb/s**	**800 Mb/s**	**1000 Mb/s**	**1200 Mb/s**	**1600 Mb/s**	**Unit**
Tsamplingerror	Invariant	Uncertainty in read pointer due to CTL sampling error in the receive clock domain (1 device specific RCLK bit time)	2500	1667	1250	1000	833	625	ps

Using the relationships given in Section 19.4 on page 216 results in the following phase recovery FIFO sizes and pointer separation design examples.

Table 110. Phase Recovery FIFO Examples

Phase Recovery Timing Uncertainties	**400 Mb/s**	**600 Mb/s**	**800 Mb/s**	**1000 Mb/s**	**1200 Mb/s**	**1600 Mb/s**	**Unit**
Trefclk	733	733	733	733	733	733	ps
TxmtPLL	3500	3500	3500	3500	3500	3500	ps
Txmttransfer	918	592	469	358	294	227	ps
Tbytelanevar (Variant)	250	250	250	250	250	250	ps
Tbytelaneconst (Invariant)	1000	1000	1000	1000	1000	1000	ps
TrcvPLL	3500	3500	3500	3500	3500	3500	ps
Trcvtransfer	425	250	188	130	108	81	ps
Tsampling (invariant)	2500	1667	1250	1000	833	625	ps
1 RCLK bit time (invariant)	2500	1667	1250	1000	833	625	ps
Total (2x variant + 1x invariant)	24651	21983	20779	19942	19437	18832	ps
Minimum FIFO Depth	10	14	18	20	24	32	entries
Safe Write-to-Read Pointer Separation	5	7	9	10	12	16	entries

M Combining Voltage and Transfer Characteristics

T_{SU} and T_{HD}, T_{CADVRS}, T_{CADVRH}, input edge rate, and V_{ID_min} AC:

While the link frequency must simultaneously satisfy the device T_{SU} and T_{HD}, the CADIN valid window is defined by the following relationship:

TCADVRS + TCADVRH + Transmit clock uncertainty / 2

This valid window definition is useful in understanding how the V_{ID_min} AC spec of 300mV can be attained at the highest link frequency at the specified minimum input edge rate. During this CADIN valid window the CADOUTs must transition from crossing to V_{ID_min} AC and back to crossing. As an example, this CADIN valid window for 1600 MT/s is worst case 116 + 116 + 91 = 323 ps. This 323 ps provides the required time to make these transitions with the specified input edge rate under worst case conditions. Note that the specified minimum input rising edge rate plus the minimum input falling-edge rate determines the minimum differential edge rate.