HyperTransport™ Consortium

HyperTransport I/O Technology
DirectPacket™ Specification

Efficient User Packet Handling
Supports Streaming Communications

White Paper
HTC_WP03

June 2004

The HyperTransport Consortium

www.hypertransport.org
The HyperTransport Technology Consortium disclaims all warranties and liability for the use of this document and the information contained herein and assumes no responsibility for any errors that may appear in this document, nor does the HyperTransport Technology Consortium make a commitment to update the information contained herein.

DISCLAIMER
This document is provided “AS IS” with no warranties whatsoever, including any warranty of merchantability, non-infringement, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification or sample. The HyperTransport Technology Consortium disclaims all liability for infringement of property rights relating to the use of information in this document. No license, express, implied, by estoppels, or otherwise, to any intellectual property rights is granted herein.

TRADEMARKS
HyperTransport is a licensed trademark of the HyperTransport Technology Consortium.

Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.
I - HyperTransport Overview – A Powerful Board-level Architecture

HyperTransport technology is a high-bandwidth chip-to-chip interconnect technology that provides an integrated framework linking all core board-level functional units including processor, memory and I/O devices. This makes it a powerful, high-performance board-level architecture for variety of product application and market segments. While HyperTransport technology features protocol compatibility with traditional PCI (Peripheral Component Interconnect) buses, its 22.4 Gigabyte/second aggregate bandwidth provides the highest effective throughput available in a board-level interconnect technology and its DirectPacket™ protocols support the intermixing of traditional load/store traffic with communications-oriented user-packet based traffic.

As an optimized board-level architecture, HyperTransport’s advantages are many: the highest throughput of any standard interconnect solution, the lowest possible latency, scalable performance, standardized interfaces, simplified implementation, minimized software overhead, and the efficient intermixing of load/store traffic with packet-bus traffic. These advantages have resulted in the widespread adoption of HyperTransport across a wide spectrum of high performance products ranging from consumer devices, personal computers, servers, network equipment, and all the way to supercomputers.

In many cases, HyperTransport’s integration extends into the processor, such as in AMD’s Opteron and Athlon64 64-bit x86 processors, Transmeta’s Efficeon x86 processor, Broadcom’s BCM1250 64-bit MIPS processor, and PMC-Sierra’s RM9000 64-bit MIPS processor family. In these instances, HyperTransport operates as a fully integrated front-side bus and the traditional NorthBridge-SouthBridge structure is eliminated. In other instances, such as in Apple’s G5 PowerMac, HyperTransport is used as an integrated, high performance I/O bus that pipes PCI, PCI-X, USB, Firewire and audio/video links through the system. In all cases, HyperTransport replaces the overlapping processor and local I/O buses of earlier generation systems with a unified, high bandwidth, low latency, and low-cost architecture that is scalable, low-cost and extensible to future product generations.
HyperTransport technology is defined by channel topology, signal electrical characteristics, and command/address/data packet protocols. Topology defines the structure of the HyperTransport link. Electrical protocols define the physical characteristics of the HyperTransport signal interface. The packet protocols define how data is organized and transferred across the HyperTransport link.

**HyperTransport Link Topology**

HyperTransport employs dual, point-to-point unidirectional data links – one for input and one for output – with a concise signal set using 1.2V low-voltage differential signaling (LVDS) and carrying both standard computer-based load/store data and communications-oriented packet data in HyperTransport packets and streaming channels.

Figure 1 - The HyperTransport Link consists of one host and at least one tunnel/endpoint and optional tunnel and bridge devices. A tunnel enables the HyperTransport link to be passed from one HyperTransport enabled device to another. A bridge enables a HyperTransport link to interface to another interconnect technology such as PCI, PCI-X, PCI Express or AGP.
The HyperTransport daisy-chain topology includes a required “host” device, at least one end-point or “cave,” optional “tunnel” devices that connect the link to other HyperTransport devices and optional “bridge” devices that interface with non-HyperTransport interconnect technologies. There can be a maximum of 32 HyperTransport-enabled devices in a single daisy chain, although tunnels and bridges can be employed to create tree structures where additional, but separate HyperTransport daisy chains can be linked to the first chain. HyperTransport switch devices, defined in Specification 1.05 can be deployed to create switched topology systems.

HyperTransport I/O Device Configurations

Figure 2 – HyperTransport-enabled devices are configured to be one of four types. The host, often a CPU, is always considered the top of the link and traffic from the host is downstream while traffic to the host is upstream. A cave serves as an endpoint to a HyperTransport link. A bridge can be configured as an endpoint with a secondary interface to other interconnect technologies, or as a tunnel with an interface to other technologies.

HyperTransport Signals

The HyperTransport link includes a variable width data path (2, 4, 8, 16 or 32 bits), one or more clock lines (one per each 8-bit data path) and a single control line. Commands, addresses, and data are carried in packets over the data path, eliminating many sideband control signals needed in traditional multiplexed, multi-drop bus standards such as PCI and PCI-X. System level control lines RESET# and PWROK complete the required set of signal lines.
Figure 3 - The HyperTransport Link (16-bit wide link shown) consists of a set of command/address/data (CAD) lines (ranging from 2, 4, 8, 16, or 32 bits wide), one control line per link, and one clock line per 8 bits of CAD. Command and data information carried on the CAD lines is assembled into HyperTransport control and data packets.

With this concise set of data and control lines, the HyperTransport specification provides a powerful, high bandwidth, easily implemented chip-to-chip communications channel capable of delivering up to 22.4 Gigabytes/second aggregate bandwidth for both standard computing-oriented applications (all HyperTransport specification levels) and communications-oriented, packet stream applications (HyperTransport specification levels DirectPacket 1.1 and above).

For more information on HyperTransport technology, please refer to the other white papers available at www.hypertransport.org, in particular, the “HyperTransport™ I/O Technology Overview, An Optimized, Low-latency Board-level Architecture” white paper published in 2004.

**HyperTransport Base Packet Protocols**

HyperTransport is an efficient data transport mechanism with the least overhead of any modern I/O interconnect architecture. Command information is carried as a control packet of four or eight bytes. HyperTransport data traffic is carried as a data packet that consists of an 8- or 12-byte header (one 8-byte control packet for writes or two control packets, one 4-byte and one 8-byte, for reads) followed
by a 4-64 byte data payload. All HyperTransport information is carried in multiples of four bytes (32-bits).

![HyperTransport Packet Format](image)

Figure 4 – HyperTransport control packets consist typically of 4 to 8 bytes of command information. With optional 64-bit extended addressing, control packets can be 12 bytes. Data packets consist of 4- to 64-byte data payloads (in increments of 4 bytes) and directly follow either 1) an 8-byte read request followed by a 4-byte read response or 2) an 8-byte write request control packet.

HyperTransport uses a single control line to determine when the link is carrying a control packet (the control signal is asserted) or a data packet (the control signal is de-asserted). Deterministic control of packet type is a significant feature of the link because the control signal can be used to insert control packets in the middle of a long data packet. A special HyperTransport Priority Request Interleaving™ feature contributes to the very low latency characteristics of the HyperTransport link by enabling concurrent data streams to be initiated in the middle of a longer data stream.

HyperTransport commands and data are separated into one of three types of virtual channels: non-posted requests, posted requests and responses. Non-posted requests require a response from the receiver. All read requests and some write requests are non-posted requests. Posted requests do not require a response from the receiver. Write requests are posted requests. Responses are replies to non-posted requests. Read responses or target done responses to non-posted writes are types of response messages.

**HyperTransport Control Packets**

Base control packets are 4 or 8 bytes long (unless using extended 64-bit addressing, in which case they are 12 bytes long) and are divided into three general types: request packets, response packets and info packets. As shown below, the Cmd[5:0] field determines the type of the control packet.
### Common HyperTransport Command Types

<table>
<thead>
<tr>
<th>Command Field</th>
<th>Command Type</th>
<th>Packet Type/ (size in bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Cmd[5:0]</code></td>
<td>Write Request</td>
<td>Req/Addr/Data (8)</td>
</tr>
<tr>
<td></td>
<td>- Nonposted/posted</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- data length = byte/doubleword</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- normal/isochronous</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- noncoherent/coherent</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Write Request with Extended Address</td>
<td>Req/Addr/Data (12)</td>
</tr>
<tr>
<td></td>
<td>Read Request</td>
<td>Req/Address (8)</td>
</tr>
<tr>
<td></td>
<td>- Ordering - pass/no pass</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- data length = byte/doubleword</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- normal/isochronous</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- noncoherent/coherent</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read Response</td>
<td>Resp/Data (4)</td>
</tr>
<tr>
<td></td>
<td>Target Done</td>
<td>Response (4)</td>
</tr>
<tr>
<td></td>
<td>Broadcast Message</td>
<td>Req/Address (8)</td>
</tr>
<tr>
<td></td>
<td>Atomic Read-Modify-Write</td>
<td>Req/Addr/Data (8)</td>
</tr>
<tr>
<td></td>
<td>Address Extension</td>
<td>Address (12)</td>
</tr>
<tr>
<td></td>
<td>Flush posted writes</td>
<td>Request (4)</td>
</tr>
<tr>
<td></td>
<td>Fence for posted requests</td>
<td>Request (4)</td>
</tr>
<tr>
<td></td>
<td>Extended Flow Control for VCSets 0-7</td>
<td>Info (4)</td>
</tr>
<tr>
<td></td>
<td>Link Synchronization and Error Packet</td>
<td>Info (4)</td>
</tr>
</tbody>
</table>

Figure 5 – HyperTransport control packets use a 6-bit command field to indicate the packet type – write and read requests, special requests, responses, and information packets.

Shown below is a HyperTransport request control packet with some of the types of requests that can be specified. The last 4 bytes of this 8-byte control packet contain the higher order 32-bits of the 40-bit address.
Figure 6 – HyperTransport request control packets showing the Command field and the packet type – write request, write request/extended address, read request, broadcast message, and atomic read-modify-write.

Shown below is a HyperTransport Read Response Packet with the important fields highlighted. Immediately following a Read Response Packet (assuming normal completion) will be a 4-to 64-byte data packet. As previously noted, the Command field specifies the command (Read Response). The Byte Count is contained in the two fields, Count[1:0] and Count[3:2] and the Isoc bit is set depending on the type of bandwidth required for the data payload. Isochronous traffic such as multimedia data streams (information such as video frames or audio information that must arrive in sequence and without delay) are supported
in HyperTransport using the Isoc bit. When set, it means that the accompanying data payload or data packet contains information that must be passed along with a higher priority than normal or non-isochronous traffic. The Error0 and Error1 bits give the status of the transaction. Normal completion is the rule, as the HyperTransport base link at speeds in excess of 1.0 GHz has been proven to an extremely reliable link. Field tests have shown expected error rates to be 1 error per 100 years. This is on the order of SRAM interface error rates. (Note that memory errors commonly occur in the storage cell, not in transit on the interface signals. These are extremely reliable).

<table>
<thead>
<tr>
<th>Bit-Time</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Isoc</td>
<td>Rsv</td>
<td>Cmd[5:0]: 110000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Pass_PW</td>
<td>Bridge</td>
<td>Rsv</td>
<td>UnitID[4:0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Count[1:0]</td>
<td>Error0</td>
<td>SrcTag[4:0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Rsv/RqUID</td>
<td>Error1</td>
<td>Rsv/RspVCS</td>
<td>Count[3:2]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**HyperTransport Read Response Packet Format**

**Important Read Response Packet Fields**

<table>
<thead>
<tr>
<th>Command Type</th>
<th>Error Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 7 – HyperTransport Read Response control packets showing the Command field and other important fields highlighted. Read Response packets will be followed by a 4- to 64-byte data packet.

**II - DirectPacket™ Advanced Communications-oriented Protocols**

HyperTransport Specification 1.03 defines the base HyperTransport control and data packet types. HyperTransport DirectPacket Specification 1.1 and higher define extended communications features that include peer-to-peer routing, messaging semantics for user packet handling, 16 new virtual channels and an error retry protocol. By using unused bits in the base HyperTransport packet protocol to define these communications-oriented features, HyperTransport
packets that carry user packet information are enabled without adding any additional packet header overhead.

**HyperTransport Peer-to-peer and Host Reflected Routing**

To maintain PCI compatibility, the base HyperTransport specification supports host-reflected routing. In this scheme, all traffic passes through the host device in the daisy chain. In the PCI bus structure, since it is a parallel physical bus, it is easy to understand the order in which data transactions take place. In typical processor-based computing systems, the order of transactions is often very important. Since HyperTransport is a distributed system, ordering is more difficult to discern. Thus, in order to replicate the ordering properties of PCI, all base HyperTransport transactions are reflected through the host.

![Host Reflected Routing](image)

Figure 8 – Host reflected routing requires that all traffic pass through the host in order to maintain PCI compatibility.

For communication systems, however, this host reflected transfer is more of an overhead burden than an advantage. Often, processing elements are daisy chained and devices need to communicate only directly to the next device in the chain. Thus, for those systems that do not require PCI conformity,
HyperTransport specifications 1.1 (DirectPacket) and up support peer-to-peer routing.

![Peer-to-Peer Routing](image)

Figure 9 – HyperTransport DirectPacket™ protocols support peer-to-peer routing that allows two devices to communicate directly, greatly reducing link traffic and off-loading the host of the burden of reflecting the device-to-device traffic. This direct device-to-device transaction is typical in complex communications systems that deploy a chain of special function and packet processing engines.

HyperTransport DirectPacket allows each device to specify an address range over which peer-to-peer transactions are allowed. Transactions out of this address range continue to employ host reflected routing. This allows the intermixing of peer-to-peer and host reflected devices and maintains seamless compatibility with those transactions that require PCI-type global ordering.

**High-Efficiency User Packet Handling**

HyperTransport DirectPacket™ (specification level 1.1) introduced powerful communications protocols that enable HyperTransport links to carry user packet data efficiently. It does so by defining message-based packet bus protocols that support message semantics. These co-exist with the read/write or load/store memory semantics of the base HyperTransport packet protocol.

Compute-oriented data transfers use a load/store metaphor and require the interconnect link to instruct each attached device precisely where to store or retrieve the data in system memory. This is called memory semantics and is characteristic of a read/write bus such as PCI or basic processor buses. Memory
semantics requires the source to know and understand the memory and buffer limitations of the destination in the same way that a processor is intimately familiar with the memory constraints of its system.

Communications technologies on the other hand, use a message passing channel metaphor, where messages are passed through channels. Messages contain the channel number and data in the form of user packets containing control and data payload information. It is the receiver’s responsibility to determine where data in the message streams are to be stored. This message semantics-based approach simplifies the work of the link. The link is only responsible for providing the source/destination, control information and data payload, and does not have to specify exact memory locations or be concerned at all with memory storage management.

Figure 10 – Message semantics-based channel-oriented links do not specify where the receiver is to place received data packets. The channel specifies only the source and destination, delivers the packetized data and leaves it up to the receiver to handle the specifics of data storage and buffer management.
Using memory-based read/write buses such as PCI, PCI Express, and base HyperTransport, the host or peripheral reads from a specific memory address or writes to a specific memory address. This works fine when dealing with compute-oriented data transactions. However, when the need arises to support channelized user packet information that is being streamed in from a communications oriented protocol device or subsystem through an interface such as SPI-4, PL-3, GMII, or XAUI, this load/store bus mechanism must be beefed up to handle the flow of packetized data.

The typical mechanism deployed in load/store buses to support packet handling is the DMA loop. With the data being sent in one direction into memory locations, the DMA circuitry manages absorbing the data from the packet stream, storing it into available memory locations, and reporting back to the transmitter what memory space is available. When the receiver is finished processing the data packets, it sends back the freed up locations in memory to the transmitter. The transmitter sends more data and the DMA circuitry places the new data into its memory. A constant DMA loop has the actual data traveling in one direction to the destination and free memory addresses being sent in the opposite direction to the source, a mechanism that is a poor use of available bandwidth.

DMA loops also introduce system design issues, including how to share complex data structures (the data payload of user packets with the encapsulated header information) in a way that both transmitter and receiver can efficiently reference the descriptors in local memory storage. Use of multiple channels sharing the same transmitter or receiver further complicates matters. The receiving device will need an adequate supply of complete data structures, or it will waste cycles and bandwidth through channel starvation. Therefore the receiver must manage the flow and acquisition of the descriptors that define the data structure. This task often requires a pre-fetch architecture to maintain an adequate flow. DMA loop mechanism complexities make guaranteeing high throughput of packet data problematic. As a result, it is difficult to support the robust throughput requirements of the telecom equipment market using load/store-based architectures.
On the other hand, a packet bus using message semantics encapsulates the data into user packets and sends them to their destinations on channels. It simplifies design issues, because it allows the receiver to deal with the packet data structure instead of each byte of the underlying data structure and eliminates the backward flow of available memory addresses. In a packet bus structure, each channel has a backpressure flag that allows the receiver to manage the flow of the packet data stream to avoid channel starvation and congestion. The transmitter sends packets to one of the available channels, i.e., one that doesn’t exhibit any backpressure, and arbitrates between eligible traffic according to its own needs. The receiver accepts the packet stream from the I/O channel, classifies the packets in some way and assigns them to one of its internal queues with its own internal DMA controller. In this system, the receiver is able to manage it own memory resources according to its own needs instead of trying to coordinate each buffer or data byte with the transmitter. In essence, it lets the transmitter and receiver handle a complete packet at one time, rather than juggling each data transfer one byte at a time.

This approach yields a decidedly more robust and cleaner mechanism for handling user data packets. In addition, the HyperTransport DirectPacket protocol enables the intermixing of both standard load/store base packets and message passing communications protocol packets. This gives HyperTransport-enabled systems the best of both worlds – high bandwidth compute-oriented transactions seamlessly mixed with high efficiency native user packets.

HyperTransport DirectPacket user packet handling was implemented by redefining some bits and using some previously unused bits of the base HyperTransport control packet format. This is in stark contrast to other native packet handling technologies that add a completely new layer in the protocols. This latter approach adds 4 to 8 bytes of additional overhead to every user packet and makes devices that deploy the new layer software incompatible with previous devices. HyperTransport in contrast, adds no additional protocol overhead and enables prior specification compatible devices to interoperate cleanly with packet carrying capable devices.
Figure 11 – By redefining some of the command field bits and accessing some Reserved field bits, HyperTransport DirectPacket specification enables the carrying of user packet data in virtual channels within a standard-sized HyperTransport control packet (followed of course by a HyperTransport data packet).

Using a redefined Posted Write Command control packet, HyperTransport DirectPacket creates a new Virtual Channel control packet format that enables the selection of 1 of 16 virtual channels. With a Start of Message bit, an End of Message bit, and a Byte Count field, the new control packet allows the insertion of variable sized user packets into the standard 4- to 64-byte HyperTransport Data packet with no additional overhead.

If the user packet is equal to or less than 64 bytes, it can be placed in a single HyperTransport data packet. If it is more than 64 bytes, it can be placed in a
sequence of data packets by breaking it up into 64-byte segments. If the last segment is less than 64 bytes, the Byte Count value is used to set the size of the last HyperTransport data packet.

![HyperTransport User Packet Handling](image)

**Figure 12** – HyperTransport DirectPacket carries user packet data in virtual channels specified by one or more virtual channel packets (redefined posted write control packets) and one or more data packets.

### III - 16 Streaming Virtual Channels

In addition to the 6 virtual channels with flow control supported by the base HyperTransport specification, DirectPacket and Specification 2.0 support an additional 3 standard virtual channels and 16 streaming virtual channels with flow control.

When combined with the message semantics of DirectPacket, the new streaming virtual channels make it easy to interface HyperTransport links to packet bus technologies such as SPI-4.2. A bridge device between HyperTransport and SPI-4.2 can map 16 SPI-4.2 channels into HyperTransport channels and vice versa. This is setup using a HyperTransport capability block that allows a device to
declares whether they support virtual channels and sets the configuration fields to control them.

For bridging to devices that require thousands of channels, an optional end-to-end flow control protocol is available. This lets many thousands of channels to be layered flexibly on top of the 16 streaming channels or on top of the other base HyperTransport channels.

Finally, a two-priority arbitration algorithm has been implemented to allow streaming channels to interact with each other and with other traffic cleanly. The algorithm has configurable bandwidth limits for high priority traffic and weighted arbitration for low priority traffic. HyperTransport also includes channel starvation prevention mechanisms.

**Error Retry Protocol**

While the base HyperTransport electrical link is inherently reliable, and its predicted error rate falls in the once per century category, to future-proof the protocol, an optional error retry protocol was introduced in DirectPacket. This protocol defines a 32-bit CRC appended to every packet, a history structure on the transmit side of each link, packet counters and acknowledge bits in the NOP packet. Each NOP packet carries a packet count to acknowledge the last packet received without an error. If the receiver detects a CRC error, the packet is discarded and a retry handshake is initiated. The transmitter then retransmits the packets from the last acknowledged good packet. Each link implements log bits and a retry counter that enable higher-level software to monitor the health of a link, since link errors are corrected in hardware without high-level intervention.

**IV - Conclusion**

HyperTransport's DirectPacket protocol is the leanest, most efficient protocol to process user packet data and to support native packet handling. Since HyperTransport DirectPacket utilizes unused bits in the base HyperTransport packet format, there is no extra overhead for supporting user packets. This makes it possible to support user packets with far less overhead than other approaches. The encoding requires less overhead bytes (command and control bytes) using typical, 64-bytes packet sizes and HyperTransport's link electrical characteristics avoid the delays caused by serializer/deserializer functions used...
by other technologies. The result is that HyperTransport’s DirectPacket™ moves a user packet from point A to point B more efficiently than any other combined load-store plus packet handling protocol.

DirectPacket makes no assumptions about how the system architecture must handle packet data. The assumption of other technologies supporting native packet handling is that the basic specification defines both board-level packet transfers and the entire system architecture. Consequently, there are several additional layers of protocol management that over-burden even the most basic packet data transfers. HyperTransport takes the opposite approach, defining just the level of protocol required to move user packet data from point A to point B and leaving the rest of the system architecture to the OEM to implement as needed.

HyperTransport DirectPacket protocols give the designer a powerful, flexible and comprehensive tool for integrating computing and communications functionality in board-level systems.

End of White Paper HTC_WP03
About the HyperTransport Consortium

The HyperTransport Technology Consortium is a membership-based non-profit organization in charge of managing and promoting HyperTransport Technology. It consists of over 40 member companies including major industry players in the personal computer, server, network equipment, silicon IP, software and supercomputing markets. Founding members include Advanced Micro Devices, Alliance Semiconductor, Apple Computer, Broadcom Corporation, Cisco Systems, NVIDIA, PMC-Sierra, Sun Microsystems, and Transmeta. Membership is open to any company interested in leveraging the HyperTransport technology and is based on a minimal yearly fee that includes the right to royalty-free use of HyperTransport technology and Intellectual Property. For more information, please visit: http://www.hypertransport.org/org_join.html.

The HyperTransport-enabled product portfolio includes tunnel, bridge, and graphic chips; programmable-logic devices; security processors; IP cores; BIOS software; verification and test tools; and training courses and an architecture reference manual. A full product listing can be found at: http://www.hypertransport.org/featuredproducts/products.html.