HYPERTRANSPORT™
SYSTEM
ARCHITECTURE

MindShare, Inc.
Jay Trodden
Don Anderson

“The ‘must-have’ PC architecture reference set.”
—PC Magazine’s “Read Only” column

PC SYSTEM ARCHITECTURE SERIES
Are your company’s technical training needs being addressed in the most effective manner?

MindShare has over 25 years experience in conducting technical training on cutting-edge technologies. We understand the challenges companies have when searching for quality, effective training which reduces the students’ time away from work and provides cost-effective alternatives. MindShare offers many flexible solutions to meet those needs. Our courses are taught by highly-skilled, enthusiastic, knowledgeable and experienced instructors. We bring life to knowledge through a wide variety of learning methods and delivery options.

training that fits your needs

MindShare recognizes and addresses your company’s technical training issues with:

- Scalable cost training
- Just-in-time training
- Training in a classroom, at your cubicle or home office
- Customizable training options
- Overview and advanced topic courses
- Reducing time away from work
- Training delivered effectively globally
- Concurrently delivered multiple-site training

MindShare training courses expand your technical skillset

- PCI Express 2.0®
- Intel Core 2 Processor Architecture
- AMD Opteron Processor Architecture
- Intel 64 and IA-32 Software Architecture
- Intel PC and Chipset Architecture
- PC Virtualization
- USB 2.0
- Wireless USB
- Serial ATA (SATA)
- Serial Attached SCSI (SAS)
- DDR2/DDR3 DRAM Technology
- PC BIOS Firmware
- High-Speed Design
- Windows Internals and Drivers
- Linux Fundamentals
- ... and many more.

All courses can be customized to meet your group’s needs. Detailed course outlines can be found at www.mindshare.com

bringing life to knowledge.

real-world tech training put into practice worldwide

*PCI Express ® is a registered trademark of the PCISIG*
Engage MindShare

Have knowledge that you want to bring to life? MindShare will work with you to “Bring Your Knowledge to Life.” Engage us to transform your knowledge and design courses that can be delivered in classroom or virtual classroom settings, create online eLearning modules, or publish a book that you author.

We are proud to be the preferred training provider at an extensive list of clients that include:

- ADAPTEC
- AMD
- AGILENT TECHNOLOGIES
- APPLE
- BROADCOM
- CADENCE
- CRAY
- CISCO
- DELL
- FREESCALE
- GENERAL DYNAMICS
- HP
- IBM
- KODAK
- LSI LOGIC
- MOTOROLA
- MICROSOFT
- NASA
- NATIONAL SEMICONDUCTOR
- NETAPP
- NOKIA
- NVIDIA
- PLX TECHNOLOGY
- QLOGIC
- SIEMENS
- SUN MICROSYSTEMS
- SYNOPSYS
- TI
- UNISYS

MindShare Learning Options

- **MindShare Classroom**
- **MindShare Virtual Classroom**
- **MindShare eLearning**
- **MindShare Press**

**Classroom Training**

Invite MindShare to train you in-house, or sign-up to attend one of our many public classes held throughout the year and around the world. No more boring classes, the ‘MindShare Experience’ is sure to keep you engaged.

**Virtual Classroom Training**

The majority of our courses live over the web in an interactive environment with WebEx and a phone bridge. We deliver training cost-effectively across multiple sites and time zones. Imagine being trained in your cubicle or home office and avoiding the hassle of travel. Contact us to attend one of our public virtual classes.

**eLearning Module Training**

MindShare is also an eLearning company. Our growing list of interactive eLearning modules include:

- Intro to Virtualization Technology
- Intro to IO Virtualization
- Intro to PCI Express 2.0 Updates
- PCI Express 2.0
- USB 2.0
- AMD Opteron Processor Architecture
- Virtualization Technology
- ...and more

**MindShare Press**

Purchase our books and eBooks or publish your own content through us. MindShare has authored over 25 books and the list is growing. Let us help make your book project a successful one.
Part One: Overview of HyperTransport

Chapter 1: Introduction to HyperTransport

Background: I/O Subsystem Bottlenecks

Server Or Desktop Computer: Three Subsystems
CPU Speed Makes Other Subsystems Appear Slow
Multiple CPUs Aggravate The Problem
DRAM Memory Keeps Up Fairly Well
I/O Bandwidth Has Not Kept Pace
This Slows Down The Processor
It Also Hurts Fast Peripherals
Reducing I/O Bottlenecks

The Shared Bus Approach
A Shared Bus Runs At Limited Clock Speeds
A Shared Bus May Be Host To Many Device Types
Backward Compatibility Prevents Upgrading Performance
Special Problems If The Shared Bus Is PCI
A Note About PCI-X
The Point-to-Point Interconnect Approach
A Note About Connectors

What HT Brings
Key Features Of HyperTransport Protocol
The Cost Factor
Networking Support
## Chapter 2: HT Architectural Overview

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transfer Types Supported</td>
<td>21</td>
</tr>
<tr>
<td>Address-Based Semantics</td>
<td>22</td>
</tr>
<tr>
<td>Data Transfer Type and Transaction Flow</td>
<td>23</td>
</tr>
<tr>
<td>Programmed I/O Transfers</td>
<td>24</td>
</tr>
<tr>
<td>DMA Transfers</td>
<td>25</td>
</tr>
<tr>
<td>Peer-to-Peer Transfers</td>
<td>26</td>
</tr>
<tr>
<td>HT Signals</td>
<td>27</td>
</tr>
<tr>
<td>Link Packet Transfer Signals</td>
<td>28</td>
</tr>
<tr>
<td>Link Support Signals</td>
<td>29</td>
</tr>
<tr>
<td>Scalable Performance</td>
<td>30</td>
</tr>
<tr>
<td>Data Widths</td>
<td>31</td>
</tr>
<tr>
<td>Clock Speeds</td>
<td>32</td>
</tr>
<tr>
<td>Extending the Topology</td>
<td>33</td>
</tr>
<tr>
<td>Packetized Transfers</td>
<td>35</td>
</tr>
<tr>
<td>Control Packets</td>
<td>36</td>
</tr>
<tr>
<td>Information packet (4 bytes)</td>
<td>36</td>
</tr>
<tr>
<td>Request packet (4 or 8 bytes)</td>
<td>36</td>
</tr>
<tr>
<td>Response packet (4 bytes)</td>
<td>36</td>
</tr>
<tr>
<td>Data Packets</td>
<td>37</td>
</tr>
<tr>
<td>HyperTransport Protocol Concepts</td>
<td>37</td>
</tr>
<tr>
<td>Channels and Streams</td>
<td>37</td>
</tr>
<tr>
<td>Virtual Channels</td>
<td>37</td>
</tr>
<tr>
<td>I/O Streams</td>
<td>39</td>
</tr>
<tr>
<td>Transactions (Requests, Responses, and Data)</td>
<td>39</td>
</tr>
<tr>
<td>Transaction Requests</td>
<td>40</td>
</tr>
<tr>
<td>Transaction Responses</td>
<td>40</td>
</tr>
<tr>
<td>Transaction Types</td>
<td>41</td>
</tr>
<tr>
<td>Sized Read Transactions</td>
<td>41</td>
</tr>
<tr>
<td>Sized Write Transactions</td>
<td>42</td>
</tr>
<tr>
<td>Non-Posted Sized Writes</td>
<td>42</td>
</tr>
<tr>
<td>Posted Sized Writes</td>
<td>43</td>
</tr>
<tr>
<td>Flush</td>
<td>44</td>
</tr>
<tr>
<td>Fence</td>
<td>45</td>
</tr>
<tr>
<td>Atomic</td>
<td>46</td>
</tr>
<tr>
<td>Broadcast</td>
<td>46</td>
</tr>
<tr>
<td>Managing the Links</td>
<td>47</td>
</tr>
<tr>
<td>Flow Control</td>
<td>47</td>
</tr>
<tr>
<td>Initialization and Reset</td>
<td>49</td>
</tr>
<tr>
<td>Configuration</td>
<td>50</td>
</tr>
<tr>
<td>Error Detection and Handling</td>
<td>50</td>
</tr>
</tbody>
</table>
Part Two: HyperTransport Core Topics

Chapter 3: Signal Groups

Introduction ............................................................................................................................... 53
The Signal Groups ................................................................................................................... 54
The High Speed Signals (One Set In Each Direction) ...................................................... 55
  The CAD Signal Group .......................................................................................... 55
  Control Signal (CTL) .......................................................................................... 55
  Clock Signal(s) (CLK) .......................................................................................... 55
Scaling Hazards: Burden Is On The Transmitter............................................................... 56
The Low Speed Signals ........................................................................................................... 56
  Power OK (PWROK) And Reset (RESET#) ............................................................ 56
  LDTSTOP# ............................................................................................................. 57
  LDTREQ# .............................................................................................................. 57
Where Are The Interrupt, Error, And Wait State Signals?............................................... 57
  Interrupt Signaling ............................................................................................ 58
  Error Signaling ................................................................................................... 58
  Wait State Signaling ............................................................................................ 58
No Arbitration Signals Either ............................................................................................ 58

Chapter 4: Packet Protocol

The Packet-Based Protocol ..................................................................................................... 60
  8 Bit Interfaces .................................................................................................... 60
  Interfaces Narrower Than 8 Bits ........................................................................... 62
  Interfaces Wider Than 8 Bits ................................................................................. 63
    16 Bit Interfaces .................................................................................................. 63
    32 Bit Interfaces .................................................................................................. 64
The Two Packet Types: Control And Data .......................................................................... 65
  Control Packet Purpose ...................................................................................... 65
    Information packets ........................................................................................... 65
    Request packets ................................................................................................ 65
    Response packets ............................................................................................... 66
  Data Packets .......................................................................................................... 66
The Need To Interleave Control And Data Packets .......................................................... 67
  The CTL Signal Indicates Packet Type ............................................................... 67
Packet Format: Control Packets ......................................................................................... 69
  Control Packets: Information .............................................................................. 71
    NOP Packet ....................................................................................................... 71
    Sync/Error Packet ............................................................................................ 74
  Control Packets: Requests .................................................................................... 75
Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sized Read And Sized Write Requests</td>
<td>75</td>
</tr>
<tr>
<td>Generic RdSized And WrSized Request Packet Format</td>
<td>76</td>
</tr>
<tr>
<td>RdSized And WrSized Requests: Transaction Limits</td>
<td>78</td>
</tr>
<tr>
<td>RdSized And WrSized (Dword) Transactions</td>
<td>79</td>
</tr>
<tr>
<td>RdSized (Byte) Transactions</td>
<td>79</td>
</tr>
<tr>
<td>WrSized (Byte) Transactions</td>
<td>79</td>
</tr>
<tr>
<td>RdSized And WrSized Requests: Other Notes</td>
<td>80</td>
</tr>
<tr>
<td>Coherency</td>
<td>80</td>
</tr>
<tr>
<td>WrSized Requests And The Posted Bit</td>
<td>80</td>
</tr>
<tr>
<td>Errors During RdSized Transactions</td>
<td>81</td>
</tr>
<tr>
<td>PassPW and Response May Pass Posted Requests bits</td>
<td>81</td>
</tr>
<tr>
<td>Compatibility Bit</td>
<td>82</td>
</tr>
<tr>
<td>Broadcast Message Requests</td>
<td>82</td>
</tr>
<tr>
<td>Flush Requests</td>
<td>84</td>
</tr>
<tr>
<td>Flush Requests: Transaction Limits</td>
<td>86</td>
</tr>
<tr>
<td>Fence Requests</td>
<td>87</td>
</tr>
<tr>
<td>Fence Requests: Transaction Limits</td>
<td>88</td>
</tr>
<tr>
<td>Atomic Read-Modify-Write Requests</td>
<td>89</td>
</tr>
<tr>
<td>Two Problems In Shared Memory Schemes</td>
<td>89</td>
</tr>
<tr>
<td>Atomic RMW Variants</td>
<td>90</td>
</tr>
<tr>
<td>Compare And Swap</td>
<td>90</td>
</tr>
<tr>
<td>Fetch And Add</td>
<td>90</td>
</tr>
<tr>
<td>Atomic RMW Requests: Transaction Limits</td>
<td>93</td>
</tr>
<tr>
<td>Control Packets: Responses</td>
<td>94</td>
</tr>
<tr>
<td>Read Responses</td>
<td>94</td>
</tr>
<tr>
<td>Target Done Responses</td>
<td>97</td>
</tr>
</tbody>
</table>

Chapter 5: Flow Control

The Problem                                                      99

How PCI Handles Flow Control                                     101

PCI Target Flow Control Problems                                  101

PCI Target Not Ready To Start                                     101

PCI Target Starts Data Transfer, But Can’t Continue               101

PCI Target Starts, Can Continue, But Needs More Time              102

PCI Initiator Flow Control Problems                               102

PCI Initiator Starts, But Can’t Continue                          102

PCI Initiator Starts, Can Continue, But Needs Wait-States         102

All PCI Flow Control Problems Hurt Performance                   103
Contents

HyperTransport Flow Control: Overview ................................................................. 103
  Packets Never Start Unless Completion Assured .............................................. 103
  Transfer Length Is Always Known .................................................................. 103
  Split Transactions Used When Response Is Required ...................................... 104
  Flow Control Pins Are Eliminated ................................................................... 104
  Flow Control Buffers Mean No Bus Wait States ............................................ 104
  Flow Control Buffers For Each Virtual Channel .......................................... 105
Flow Control, A System View ............................................................................. 105
Flow Control Buffer Arrangement ....................................................................... 106
  Details Associated With Figure 5-3 ................................................................. 107
    Flow Control Buffer Pairs (Item 1) ............................................................... 108
    Posted Request Buffer (Command) ............................................................... 108
    Posted Request Buffer (Data) ....................................................................... 108
    Non-Posted Request Buffer (Command) ....................................................... 108
    Non-Posted Request Buffer (Data) ............................................................... 108
    Response Buffer (Command) ....................................................................... 108
    Response Buffer (Data) ................................................................................ 108
    Receiver Flow Control Counters (Item 2) ................................................. 108
    Transmitter Flow Control Counters (Item 3) ............................................. 109
    NOP Packet Update Information (Item 4) .................................................. 109
    Control Logic (Item 5) ................................................................................ 109
    Transmit And Receive FIFO (Item 6) .......................................................... 110
Example: Initialization And Use Of The Counters ............................................... 110
  Basic Steps In Counter Initialization And Use ................................................. 110
    Initializing The Flow Control Counter ....................................................... 111
    Device 1 Sends Two Posted Request Packets .......................................... 112
    New Entries Available: Update Flow Control Information .......................... 114
A Few Implementation Notes ............................................................................... 115
  Information Packets Not Flow-Controlled ..................................................... 115
  Transmitter Must Be Able To Track 15 Buffer Entries .................................... 115
  Sometimes Two Counters Must Be Checked ............................................... 116
  NOP Packets Cannot Be Completely Blocked ............................................. 116
  The Isochronous Flow Control Option .......................................................... 116
    How About NOP Updates For Isochronous Buffers? ............................... 116
    Isochronous Traffic/Non-Isochronous Flow Control ............................... 117
    Isochronous Traffic Disabled At Initialization ....................................... 117
Chapter 6: I/O Ordering

The Purpose Of Ordering Rules ................................................................. 120
Maintain Data Coherency ................................................................. 120
Avoid Deadlocks ......................................................................... 120
Support Legacy buses ................................................................. 120
Maximize Performance ................................................................. 120

Introduction: Three Types Of Traffic Flow ......................................... 121

The Ordering Rules ........................................................................... 122
General I/O Ordering Limits .......................................................... 122
Ordering Covers Targets At Same Hierarchy Level ....................... 122
Read And Non-Posted Write Completion At Target ...................... 123
What If A Device Requires Response Ordering? .......................... 124
Support For The Producer-Consumer Ordering Model ............... 124
Producer-Consumer Model Simpler If Flag/Data In Same Place .... 124

Upstream Ordering Rules ................................................................. 125
Reordering Packets In Different Transaction Streams ................. 125
No Reordering Packets In A Strongly Ordered Sequence ............ 126
Packets With PassPW Bit Clear Are Restricted In Passing .......... 127
Packets With PassPW Bit Set May Or May Not Pass ................. 128
Non-Posted Requests May Pass Each Other ............................... 129
Posted Requests And Responses Must Be Able To Pass ............. 130
Posted Request Must Be Able To Pass A Response ................. 131
Non-Posted Requests Or Response May Pass A Response .......... 132

Host Ordering Requirements .......................................................... 132
Host Ordering Requirements: General Features ......................... 133
Two Ordering Points Are Defined .................................................. 134
Globally Ordered (GO) ................................................................. 134
Globally Visible (GV) ................................................................. 134
Ordering Rule Summary ............................................................... 134
Host Responses To Non-Posted Requests .................................. 135
An Example (Refer to Table 6-1 and Figure 6-13) ..................... 136
Double-Hosted Chain Ordering .................................................... 136

Chapter 7: Transaction Examples

Packets As Transaction Building Blocks .......................................... 140

Transaction Examples: Introduction ............................................... 144
Packet Format And Optional Features .......................................... 144
General Sequence Of Events ......................................................... 144
Example 8: Atomic Read-Modify-Write ............................................................................. 182
  Example 8: Atomic RMW Request Packet Setup........................................................ 183
    Command[5:0] Field (Byte 0, Bit 5:0) ..................................................................... 183
    SeqID[3:0] Field (Byte 0, Bit 7:6) and (Byte 1, Bit 6:5) .......................................... 183
    UnitID[4:0] Field (Byte 1, Bits 4:0) ..................................................................... 183
    PassPW Bit Field (Byte 1, Bit 7) .......................................................................... 183
    SrcTag[4:0] Field (Byte 2, Bits 4:0) ..................................................................... 183
    Compat Bit Field (Byte 2, Bit 5) .......................................................................... 183
    Mask/Count[3:0] Field (Byte 2, Bits 7:6) and (Byte 3, Bits 1:0) .......................... 184
    Start Address Field (Bytes 4-7, Bit 7:0) and (Byte 3, Bit 7:2) .......................... 184
  Example 8: Atomic RMW Request Data Packet ......................................................... 184
  Example 8: Atomic RMW Request, Sequence Of Events ........................................... 184
  Example 8: Atomic RMW Response Packet Setup.................................................... 185
    Command[5:0] Field (Byte 0, Bit 5:0) ..................................................................... 185
    Isoc Bit Field (Byte 0, Bit 7) .................................................................................. 185
    UnitID[4:0] Field (Byte 1, Bits 4:0) ..................................................................... 185
    Bridge Bit Field (Byte 1, Bit 6) .......................................................................... 185
    PassPW Bit Field (Byte 1, Bit 7) .......................................................................... 185
    SrcTag[4:0] Field (Byte 2, Bits 4:0) ..................................................................... 185
    Error Bit Field (Byte 2, Bit 5) ............................................................................. 185
    Count[3:0] Field (Byte 2, Bits 7:6) and (Byte 3, Bits 1:0) ..................................... 186
    Nxa Bit Field (Byte 3, Bit 5) ............................................................................... 186
  Example 8: Atomic RMW Response Data Packet...................................................... 187
  Example 8: Atomic RMW Response, Sequence Of Events ......................................... 187
  Some Notes About Atomic RMW Operations ............................................................. 188
Example 9: WrSized Request Crosses A Bridge............................................................... 189
  Example 9: Request Packet On Bus 1 ........................................................................ 190
    Command[5:0] Field (Byte 0, Bit 5:0) ..................................................................... 190
    SeqID[3:0] Field (Byte 0, Bit 7:6) and (Byte 1, Bit 6:5) .......................................... 190
    UnitID[4:0] Field (Byte 1, Bits 4:0) ..................................................................... 190
    PassPW Bit Field (Byte 1, Bit 7) .......................................................................... 190
    SrcTag[4:0] Field (Byte 2, Bits 4:0) ..................................................................... 190
    Compat Bit Field (Byte 2, Bit 5) .......................................................................... 190
    Mask/Count[3:0] Field (Byte 2, Bits 7:6) & (Byte 3, Bits 1:0) ............................ 191
    Start Address Field (Bytes 4-7, Bit 7:0) & (Byte 3, Bit 7:2) ............................ 191
  Example 9: Sized (Dword) Write Data Packet: Bus 1 ............................................... 191
  Example 9: Request/Data Sequence Of Events On Bus 1 .......................................... 191
  Example 9: Bridge Reissues Request Packet: Bus 0 ................................................... 192
    Command[5:0] Field (Byte 0, Bit 5:0) ..................................................................... 192
    SeqID[3:0] Field (Byte 0, Bit 7:6) and (Byte 1, Bit 6:5) .......................................... 192
    UnitID[4:0] Field (Byte 1, Bits 4:0) ..................................................................... 192
    PassPW Bit Field (Byte 1, Bit 7) .......................................................................... 192
## Contents

<table>
<thead>
<tr>
<th>Field/Bit Field Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SrcTag[4:0] Field (Byte 2, Bits 4:0)</td>
<td>192</td>
</tr>
<tr>
<td>Compat Bit Field (Byte 2, Bit 5)</td>
<td>193</td>
</tr>
<tr>
<td>Mask/Count[3:0] Field (Byte 2, Bits 7:6) &amp; (Byte 3, Bits 1:0)</td>
<td>193</td>
</tr>
<tr>
<td>Start Address Field (Bytes 4-7, Bit 7:0) &amp; (Byte 3, Bit 7:2)</td>
<td>193</td>
</tr>
<tr>
<td>Example 9: Sized (Dword) Write Data Packet: Bus 0</td>
<td>193</td>
</tr>
<tr>
<td>Example 9: Request/Data Sequence Of Events: Bus 0</td>
<td>193</td>
</tr>
<tr>
<td>Example 9: Response Packet On Bus 0</td>
<td>194</td>
</tr>
<tr>
<td>Command[5:0] Field (Byte 0, Bit 5:0)</td>
<td>194</td>
</tr>
<tr>
<td>Isoc Bit Field (Byte 0, Bit 7)</td>
<td>194</td>
</tr>
<tr>
<td>UnitID[4:0] Field (Byte 1, Bits 4:0)</td>
<td>194</td>
</tr>
<tr>
<td>Bridge Bit Field (Byte 1, Bit 6)</td>
<td>194</td>
</tr>
<tr>
<td>PassPW Bit Field (Byte 1, Bit 7)</td>
<td>194</td>
</tr>
<tr>
<td>SrcTag[4:0] Field (Byte 2, Bits 4:0)</td>
<td>194</td>
</tr>
<tr>
<td>Error Bit Field (Byte 2, Bit 5)</td>
<td>194</td>
</tr>
<tr>
<td>NXA Bit Field (Byte 3, Bit 5)</td>
<td>195</td>
</tr>
<tr>
<td>Example 9: Response, Sequence Of Events On Bus 0</td>
<td>195</td>
</tr>
<tr>
<td>Example 9: Response Packet On Bus 1</td>
<td>195</td>
</tr>
<tr>
<td>Command[5:0] Field (Byte 0, Bit 5:0)</td>
<td>195</td>
</tr>
<tr>
<td>Isoc Bit Field (Byte 0, Bit 7)</td>
<td>196</td>
</tr>
<tr>
<td>UnitID[4:0] Field (Byte 1, Bits 4:0)</td>
<td>196</td>
</tr>
<tr>
<td>Bridge Bit Field (Byte 1, Bit 6)</td>
<td>196</td>
</tr>
<tr>
<td>PassPW Bit Field (Byte 1, Bit 7)</td>
<td>196</td>
</tr>
<tr>
<td>SrcTag[4:0] Field (Byte 2, Bits 4:0)</td>
<td>196</td>
</tr>
<tr>
<td>Error Bit Field (Byte 2, Bit 5)</td>
<td>196</td>
</tr>
<tr>
<td>NXA Bit Field (Byte 3, Bit 5)</td>
<td>196</td>
</tr>
<tr>
<td>Example 9: Response, Sequence Of Events On Bus 1</td>
<td>196</td>
</tr>
</tbody>
</table>

### Chapter 8: HT Interrupts

- **Introduction** ........................................................................................................................................ 199
- **Discovering a Device’s Interrupt Requirements** .................................................................................. 200
- **The Interrupt Message Address Range** .................................................................................................. 201
- **Interrupt Requests** ............................................................................................................................... 204
  - Interrupt Request Packet ...................................................................................................................... 204
  - Interrupt Request Data Packet .......................................................................................................... 206
  - The End of Interrupt (EOI) Message ..................................................................................................... 207
  - EOI Packet Format .............................................................................................................................. 208
- **Interrupt Discovery and Configuration Capability Block** ..................................................................... 209
  - Interrupt Capability Block Format ...................................................................................................... 209
  - Last Interrupt Supported ...................................................................................................................... 211
  - Interrupt Definition Registers ......................................................................................................... 211
Contents

Chapter 9: System Management

System Management Transactions ..................................................................................... 216
  Sources of SM Request ................................................................................................. 216
  System Management Address Range .......................................................................... 217
  The SMC & Upstream Request Packets ...................................................................... 218
    Upstream Request Packet Format ............................................................................... 218
    System Management Commands — Upstream ........................................................ 219
  The Host Bridge & Downstream Request Packets .................................................... 220
    Downstream Request Packet Format ......................................................................... 220
    System Management Commands ............................................................................. 221
  HT Link Disconnect/Reconnect Sequence .................................................................... 223
    Reference Information: LDTSTOP# Procedures .................................................... 223
  Example SM Sequence: Link Initialization Disconnect ............................................. 225
    Background ................................................................................................................ 225
    Setup and Assumptions ............................................................................................. 225
    The Link Initialization Disconnect Sequence ........................................................ 227

Chapter 10: Error Detection And Handling

Introduction .................................................................................................................... 230
  Types Of Errors ............................................................................................................. 230
  Reporting Methods ....................................................................................................... 230
  The Role Of PCI Configuration Space ......................................................................... 230
  Most Types Of Error Checking Are Optional ............................................................ 231
  System Handling Of HyperTransport Errors Varies .................................................. 231
  The Error Types ............................................................................................................. 231
    CRC Errors ................................................................................................................ 231
      CRC On 8, 16, or 32 bit Interfaces .......................................................................... 232
      CRC Generation/Checking: 8/16/32 bit links ...................................................... 232
      CRC Generation/Checking: 2/4 bit links ............................................................. 233
        4 Bit CAD Width ................................................................................................. 233
        2 Bit CAD Width ............................................................................................... 233
      Logging CRC Errors ............................................................................................ 233
      Programming The CRC Error Reporting Policy .................................................. 234
        CRC Interrupts .................................................................................................... 234
        CRC Sync Flood ............................................................................................... 235
        CRC Test Mode .................................................................................................. 237
    Protocol Errors ........................................................................................................... 237
      CTL Signal Four-Byte Boundary Violation ......................................................... 237
      CTL Deassertion Violation .................................................................................. 238
      CTL/Data Interleaving Violation ......................................................................... 238
Contents

Bad Command Code In Control Packet .............................................................. 238
CTL Deassertion Timeout Violation ................................................................. 238
CTL Deasserted During CRC Transmission ....................................................... 238
Logging Protocol Errors .................................................................................. 238
Programming The Protocol Error Reporting Policy ........................................ 239
Receive Buffer Overflow Errors ...................................................................... 241
Logging Receive Buffer Overflow Errors ........................................................ 241
Programming The Buffer Overflow Error Reporting Policy ............................ 242
End-Of-Chain Errors ....................................................................................... 243
  How A Device Knows It Is At The End Of A Chain ........................................ 243
  Logging End-Of-Chain Errors ....................................................................... 245
  Programming The EOC Error Reporting Policy ............................................. 246
Chain Down Errors ........................................................................................... 247
Response Errors .............................................................................................. 248
  Response Error Logging And Reporting Policy ............................................... 249
Error Reporting ............................................................................................... 250
  Error Responses (Non-Posted Requests Only) ................................................ 250
    Error Response Returned By The Target ...................................................... 251
    Error Response Returned By An End-Of-Chain Device ................................ 251
Fatal And Non-Fatal Interrupts ......................................................................... 252
Sync Flood: When All Else Fails ....................................................................... 253
  Device Initiating The Sync Flood ................................................................. 253
  Devices Detecting Sync Flood ...................................................................... 253
  Sync Flooding And HyperTransport Bridges ............................................... 254
Miscellaneous Notes ......................................................................................... 254
  Flooding Continues Until Reset .................................................................... 254
  CRC Not Checked During Sync Flood ........................................................... 254
  Sync Flood Example ...................................................................................... 254
  Sequence of events: (Figure 10-15 on page 255) .......................................... 254

Chapter 11: Routing Packets
Packet Routing: Shared Bus vs. Point-Point Topology .................................... 258
  Shared Bus Routing ....................................................................................... 258
  HyperTransport Point-Point Routing ............................................................. 259
Review Of Packet Types And Formats ............................................................. 259
  Control Packets ........................................................................................... 259
    Information Packets: No Routing Required ............................................... 259
    Request Packet Routing Information ......................................................... 260
      Six Request Types ..................................................................................... 261
    Response Packet Routing Information ...................................................... 262
    Data Packet Routing Depends On Control Packets .................................... 263
Directed vs. Broadcast Requests ....................................................................... 263
## Contents

### Accepting Packets .......................................................... 264
   Rules For Acceptance .......................................................... 264
   A Note About The Subtractive Decoder ................................. 264

### Forwarding Packets .......................................................... 265
   Rules For Forwarding ............................................................ 265
   Other Notes On Forwarding ................................................... 265
   Forwarding Into The End Of Chain ........................................ 265
   Forwarding If Initialization Is Not Complete ......................... 266

### Rejecting Packets .............................................................. 266
   Rules For Rejection ............................................................. 266

### Host Bridge Behavior .......................................................... 267
   Directed Request With UnitID = 0 ........................................ 267
   Accepted ................................................................................ 267
   Rejected ................................................................................ 267
   Response UnitID And Bridge Fields ...................................... 268
   Broadcast Request ............................................................... 268
   Always Accepted ................................................................. 268
   Directed Request With Non-Zero UnitID ............................... 268
   Accepted Requests .............................................................. 268
   Internal Target ...................................................................... 268
   Peer-to-Peer Target .............................................................. 269
   Compatibility Chain Requests .............................................. 269
   Rejected Requests ............................................................... 270
   Responses Received By The Host Bridge ................................. 270
   Response With Bridge Bit = 1 .............................................. 270
   Response With Bridge Bit = 0 .............................................. 270

### HyperTransport Bridges: Additional Routing Rules ............... 271

### Tunnel Fairness And Forward Progress .................................. 271
   Fairness Is Critical In A Point-Point Topology ..................... 271
   HyperTransport Imposes A Fairness Algorithm ..................... 272
   The Basic Policy .................................................................... 272
   The Algorithm ........................................................................ 272
   First, Calculate The Insertion Rate ....................................... 272
   Insertion Rate Calculation Example .................................... 273

### Chapter 12: Reset & Initialization ......................................... 276
   General ................................................................................. 276
   Cold Reset ............................................................................ 276
   Sources of Cold Reset .......................................................... 277
   Resetting the Primary HT Bus .............................................. 277
   Resetting Secondary Side Of HT-to-HT Bridge ....................... 278
## Contents

Signalling and Detecting Cold Reset ................................................................. 280
Effects of Cold Reset ......................................................................................... 281

**Link Initialization** .......................................................................................... 282
- Low-Level Link Width Initialization ................................................................. 282
- Determining Low-Level Link Width ................................................................. 283
  - Example 1: 4-bit device connected to 8-bit Device ....................................... 284
  - Example 2: 8-Bit Device Connected to 4/8-Bit Device ................................. 285
  - Example 3: 32-bit Upstream and 16-bit Downstream ................................... 286
- Negotiated Link Width Stored in Link Config Registers .................................... 288
- Low-Level Clock Initialization ......................................................................... 289
- The Default Clock Frequency .......................................................................... 291
- Control and CAD Sequence after Reset is Removed ....................................... 291
  - Clock Synchronization (CTL=0 & CAD=0) ................................................... 292
  - Duration of CTL & CAD Driven Low ............................................................ 293
- Packet Framing and Initializing the CRC Window .......................................... 294
- Tuning the Link Width (Firmware Initialization) ............................................... 295
  - Tuning Example 1: 4-bit device connected to 8-bit Device ......................... 297
  - Tuning Example 2: 8-bit device connected to 4/8-bit Device ...................... 297
  - Tuning Example 3: 32-bit Upstream and 16-bit Downstream .................... 298
- Tuning the Clock Frequency ............................................................................ 299

**Warm Reset** ................................................................................................ 302
- Warm Reset Generated by Software ............................................................... 303

**LDTSTOP# Disconnect Sequence** ................................................................. 304

### Chapter 13: Device Configuration

HyperTransport Uses PCI Configuration ......................................................... 306
What PCI Configuration Accomplishes ........................................................... 306
HyperTransport System Limits .......................................................................... 307
- 256 Buses In A System ..................................................................................... 307
- 32 UnitIDs Per Bus ........................................................................................... 307
- One To Eight Functions Per Device ................................................................. 307
- 256 Bytes Of Configuration Space .................................................................... 308
Configuration Accesses: Reaching All Devices ............................................... 308
Review: How PCI Handles Configuration Accesses ......................................... 309
- Two Configuration Cycle Types ....................................................................... 309
- Type 1 Cycle Until Target Bus Is Reached ...................................................... 309
- Target Bus Bridge: Convert To Type 0; Assert IDSEL ...................................... 310
- An Example: A PCI Configuration Space Access ............................................. 310
  - Events In PCI Configuration Space Example (see Figure 13-1) .................... 311
How HyperTransport Handles Configuration Accesses ................................................ 311
  Configuration Cycles Are Memory Mapped .......................................................... 311
How The 32MB Configuration Area Is Used .......................................................... 312
  Upper 16 Address Bits Indicate Type 0 And Type 1 Cycle ................................ 313
HyperTransport Type 1 Configuration Cycle (See Figure 13-3) .............................. 314
HyperTransport Type 0 Configuration Cycle (See Figure 13-3) .............................. 314
No IDSEL Signal Needed In HyperTransport ....................................................... 314
Example: HT Configuration Space Access ............................................................ 315
  Events In HT Configuration Example (see Figure 13-4) ..................................... 316
Initializing Bus Numbers And Unit IDs ............................................................... 316
  Case 1: A Single Chain With One Host Bridge ................................................. 316
  Case 2: A HyperTransport Bridge Is Discovered ............................................ 318
    A Note About Bus Numbering In HyperTransport ..................................... 318
  Case 3: Initializing A Double Hosted Chain ..................................................... 320
    Only One Master Host Bridge ..................................................................... 320
    Master Bridge Initialization/Configuration Sequence ................................. 320
HyperTransport Configuration Space Format ...................................................... 321
  Two Header Formats Are Used .......................................................................... 321
  The Type 0 Header Format ................................................................................ 322
PCI Advanced Capability Registers ................................................................. 323
  Many Advanced Capabilities Are Defined ......................................................... 324
  Discovering The Advanced Capability Blocks .................................................. 324
HyperTransport Configuration Type 0 Header Fields ........................................... 325
  Header Command Register .............................................................................. 325
  Header Status Register ..................................................................................... 326
Other Fields In The Header .................................................................................. 329
  Cache Line Size Register. (Offset 0Ch) ............................................................. 329
  Latency Timer Register. (Offset 0Dh) ............................................................... 329
  Base Address Registers. (Offset 10h-24h) ....................................................... 329
    I/O BAR ........................................................................................................... 329
    Memory BAR ................................................................................................. 329
  CardBus CIS Pointer. (Offset 28h) .................................................................. 330
  Capabilities Pointer. (Offset 34h) .................................................................... 330
  Interrupt Line Register. (Offset 3Ch) ............................................................... 330
  Interrupt Pin Register. (Offset 3Dh) ................................................................. 330
  Min_Gnt and Max_Latency Registers. (Offsets 3Eh and 3Fh) ........................... 330
HyperTransport Uses Advanced Capability Blocks .............................................. 330
  HyperTransport Block Types Currently Defined ............................................. 331
    Block Formats Vary With Capability And Device Type .................................. 332
The Slave/Primary Interface Block ..................................................................... 332
Contents

Description Of Slave/Primary Interface Fields ........................................................... 333
  Capability ID Register. (Offset 00h) ................................................................. 334
  Capabilities Pointer Register. (Offset 01h) ...................................................... 334
  Slave Command Register. (Offset 02h-03h) .................................................... 335
  Link Control Registers. (Offset 04h and 08h) .................................................. 336
  Link Configuration Registers. (Offset 06h and 0Ah) ....................................... 339
  Revision ID Register. (Offset 0Ch) ................................................................. 343
  Link Frequency Registers. (Offset 0Dh and 11h) .......................................... 344
  Link Error Registers. (Offset 0Dh and 11h) .................................................... 345
  Link Frequency Capability Registers. (Offset 0Eh and 12h) ......................... 346
  Feature Capability Register. (Offset 10h) ....................................................... 347
  Enumeration Scratch Pad Register. (Offset 14h) ............................................ 349
  Error Handling Register. (Offset 16h) ............................................................ 350
  Memory Base Upper Register. (Offset 18h) .................................................... 352
  Memory Limit Upper Register. (Offset 19h) .................................................... 352
The Host/Secondary Interface Block .......................................................................... 354
Description Of Host/Secondary Interface Fields ....................................................... 355
  Capability ID Register. (Offset 00h) ................................................................. 355
  Capabilities Pointer Register. (Offset 01h) ...................................................... 355
  Host Command Register. (Offset 02h-03h) .................................................... 356
  Link Control Register. (Offset 04h) ................................................................. 358
  Link Configuration Register. (Offset 06h) ....................................................... 358
  Revision ID. (Offset 08h) .................................................................................. 358
  Revision ID. (Offset 08h) .................................................................................. 358
  Link Frequency And Link Error Registers. (Offset 09h) .............................. 359
  Link Frequency Capability. (Offset 0Ah) ......................................................... 359
  Feature Capability Register. (Offset 0Ch) ....................................................... 359
  Enumeration Scratchpad Register. (Offset 10h) ............................................ 360
  Error Handling Register. (Offset 12h) ............................................................ 360
  Memory Base/Limits Upper Registers. (Offset 14h, 15h) ............................ 361
  Memory Base/Limits Upper Registers. (Offset 14h, 15h) ............................ 361
  Revision ID Capability Block ............................................................................. 361

Chapter 14: Electrical
Background and Introduction ...................................................................................... 364
Power Requirements ..................................................................................................... 366
  Power Supply Voltage ...................................................................................... 366
  Differential Pair Power Consumption ............................................................. 366
Differential Signaling Characteristics .......................................................................... 367
  Differential DC Characteristics ....................................................................... 367
    Differential DC Impedance ........................................................................... 368
    Differential Output Voltage - DC ............................................................... 369
    Differential Input Voltage - DC ................................................................. 371
## Contents

- Differential AC Characteristics ................................................................. 372
- Differential AC Impedance ................................................................. 372
- Differential Output Voltage - AC ................................................................. 374
- Differential Input Voltage - AC ................................................................. 376
- Input Rising and Falling Edge Rates ................................................................. 377
- Single-Ended Signaling Characteristics ................................................................. 378
- Differential Timing Characteristics ................................................................. 379
  - Differential Signal Skew ................................................................. 379
  - Source Synchronous Clock Skew ................................................................. 381
    - Source Synchronous Clock Skew at the Transmitter ................................................ 381
    - Source Synchronous Clock Skew at the Receiver ................................................................. 383
  - Setup and Hold Timing ................................................................. 384
- Testing ................................................................................................................. 385

---

## Chapter 15: Clocking

- Introduction ........................................................................................................... 388
- Clock Initialization ........................................................................................................... 388
- Synchronous Clock Mode ................................................................................................. 389
  - A Conceptual Example ................................................................................................. 389
  - Sources of Transmit and Receive Clock Variance ................................................................. 390
    - Invariant Sources ........................................................................................................... 391
      - Cross-byte skew in multi-byte link implementations ......................................................... 391
      - Sampling Error ........................................................................................................... 392
    - Variant Sources ........................................................................................................... 392
      - Reference Clock Distribution Skew ................................................................................. 392
      - PLL Variation in Transmitter and Receiver ................................................................. 392
      - Transmitter and Link Transfer Variation ........................................................................... 393
      - Receiver Transfer Variation ........................................................................................ 393
      - Dynamic Cross Byte-Lane Variation ................................................................................. 393
  - An Example Timing Budget ................................................................................................. 393
  - Clock Variance, FIFO Size, and the Read Pointer ................................................................. 394
    - Minimum FIFO Size ........................................................................................................... 395
    - Write-to-Read and Read-to-Write Separation ........................................................................... 395
      - Scenario 1: Tx Out Clock and Rx Clock are in Sync ......................................................... 396
      - Scenario 2: Tx Clock Out Lags Rx Clock ................................................................................. 397
      - Scenario 3: Rx Clock Lags Tx Clock Out ................................................................................. 398
  - Buffering Width and Speed Differences ........................................................................... 399
    - CAD/CTL synchronization time: ................................................................................. 399
Contents

Pseudo-Synchronous Clock Mode ................................................................. 399
Why Use Pseudo-Synchronous Clock Mode? .............................................. 400
Implementation Issues .................................................................................. 401
Methods and Procedures ............................................................................. 401
FIFO Management .................................................................................... 401
Is Support for Pseudo-Sync Mode Required? ........................................... 402
Asynchronous Clock Mode ............................................................................ 402
Transmit Clock Slower Than Receive Clock ............................................... 402
Transmit Clock Faster Than Receive Clock ................................................. 402

Part Three: HyperTransport Optional Topics

Chapter 16: HyperTransport Bridges

HyperTransport Bridges Uses PCI Configuration ......................................... 408
Basic Jobs Of A HyperTransport Bridge ....................................................... 408
How Does The Bridge Manage It All? ......................................................... 409
Same Slave/Primary And Host/Secondary Blocks ....................................... 409
HyperTransport Bridge Header Fields ......................................................... 409
Bridge Header Command Register .............................................................. 410
Bridge Header Status Register ................................................................. 413
Secondary Status Register ......................................................................... 415
Memory And Prefetchable Base And Limit Registers .................................. 416
Memory And Prefetchable Memory Base/Limit Notes ............................... 419
Bus-To-Bus Forwarding Rules .................................................................... 419
64 Bit Addressing And The 40 Bit HyperTransport Space ......................... 419
To Disable Memory or Prefetchable Memory Decoding ................................ 419
The Optional Address Remapping Registers .............................................. 419
I/O Base And Limit Registers ...................................................................... 420
Base/Limit Notes ....................................................................................... 421
Bus-To-Bus Forwarding Rules .................................................................... 421
To Disable I/O Decoding ............................................................................ 422
Bridge Control Register ............................................................................. 422
Other Fields In The Header ........................................................................ 424
Primary Latency Timer Register ................................................................. 424
Base Address Registers ............................................................................. 424
I/O BAR ..................................................................................................... 424
Memory BAR ............................................................................................ 424
Capabilities Pointer ................................................................................... 425
Interrupt Line Register ............................................................................... 425
Interrupt Pin Register ................................................................................ 425
Cache Line Size Register .......................................................................... 425
Chapter 17: Double-Hosted Chains

Introduction ..................................................................................................................... 428
  Reasons For Implementing A Double-Hosted Chain ......................................................... 428
  PCI Configuration Plays Key Role In Chain Setup .............................................................. 429
    Slave Command CSR .................................................................................................. 429
    Host Command CSR .................................................................................................. 430
Two Types Of Double-Hosted Chains ............................................................................... 431
  Sharing Double-Hosted Chain .................................................................................... 431
    If Possible, Assign All Devices To Master Host Bridge ................................................. 432
    If Slave Must Access Devices, It Uses Peer-to-Peer Transfers ....................................... 432
  Non-Sharing Double-Hosted Chain ............................................................................... 432
    Software May Break The Chain .................................................................................. 432
  Additional Notes About Double-Hosted Chains ............................................................. 433
    Initialization In A Double-Hosted Chain .................................................................... 433
    Type 0 Configuration Cycles In A Double-Hosted Chain ............................................. 434

Chapter 18: HT Power Management

Background ...................................................................................................................... 436
  Reporting Power Management Events to the Host Bridge ................................................. 436
  Reporting Host Power Management Events to SMC ......................................................... 437
    Processor VID/FID ...................................................................................................... 437
  Reporting Power Management Events to HT Devices ..................................................... 437
  Signaling Wakeup ......................................................................................................... 439
  X86 Power Management Support ................................................................................... 441
    Stop Clock Signal ....................................................................................................... 441
      HT Method of STPCLK# Signaling ........................................................................... 442

Chapter 19: Networking Extensions Overview

An Important Note ............................................................................................................ 444
  Server And Desktop Topologies Are Host-Centric ........................................................ 444
    Upstream And Downstream Traffic ............................................................................. 446
    Storage Semantics In Servers And Desktops ................................................................ 446
      Targets Are Assigned An Address Range In Memory Map ...................................... 446
      Each Byte Transferred Has A Unique Target Address ............................................. 446
      The Requester Manages Target Addresses ................................................................ 446
      Storage Semantics Work Fine In Servers And Desktops ........................................... 447
    1.04 Protocol Optimized For Host-Centric Systems .................................................... 447

Some Systems Are Not Host-Centric ............................................................................. 448
Contents

The Need For Networking Extensions ................................................................. 448
Communications Processing Is Often Less Vertical ...................................... 448
Communications Processing Example ........................................................... 449
Summary Of Anticipated Networking Extension Features .......................... 450
Network Extensions Adds Message Semantics ............................................... 450
16 New Posted Write Virtual Channels ......................................................... 451
Direct Peer-to-Peer Transfers Added ............................................................ 451
Link-Level Error Detection And Handling ................................................... 451
64 Bit Addressing Option ............................................................................. 452
Increased Number Of Host Transactions .................................................... 452
End-To-End Flow Control ............................................................................. 452
Switch Devices Formally Defined ................................................................. 453

Part Four: HyperTransport Legacy Support

Chapter 20: I/O Compatibility

Introduction .................................................................................................. 457
PCI Bus Issues ............................................................................................. 458
PCI Ordering Requirements .......................................................................... 458
Avoiding Deadlocks ..................................................................................... 460
Subtractive Decode ....................................................................................... 461
Subtractive Decode: The PCI Method ......................................................... 461
Subtractive Decode: The Simple HT Method .............................................. 461
Subtractive Decode: HT Systems Requiring Extra Support ....................... 462
  The Problem ......................................................................................... 462
  The Solution ....................................................................................... 462
Subtractive Decode: Behind PCI Bridge ..................................................... 463
Subtractive Decode: Legacy System Considerations .................................. 464
Subtractive Decode: Without Software Initialization ................................. 464
HT-to-PCI Address Remapping ................................................................. 465
Transaction Translation ............................................................................... 465
PCI Burst Transactions ............................................................................... 466
PCI-X Bus Issues ....................................................................................... 467
PCI-X Ordering Requirements ................................................................. 467
Transaction Translation ............................................................................... 468
AGP Bus Issues ......................................................................................... 470
AGP Configuration Space Requirements ................................................... 470
AGP Ordering Requirements ...................................................................... 471
  PCI-Based Ordering ........................................................................... 472
  Low Priority Ordering ....................................................................... 472
  High Priority Ordering ....................................................................... 472
Transaction Translation ............................................................................... 473
Contents

ISA/LPC Buses ................................................................. 473
  Deadlocks ........................................................................ 473
  Deadlock Scenario 1 ......................................................... 474
  Deadlock Scenario 2 ......................................................... 475

Chapter 21: Address Remapping
  Introduction ..................................................................... 477
  The Address Remapping Capability Block ..................... 478
  I/O Address ReMapping ................................................ 481
    X86 Processor and PCI I/O Remapping Example .......... 481
    PowerPC and PCI I/O Remapping Example .................. 483
  DownStream HT to Expansion Bus Memory Mapping ...... 485
    Downstream Memory Access Without Remapping .......... 485
    Memory Accesses with Remapping .............................. 486
    SBNPctl and SBPreCtl .................................................. 488
  DMA Mapping ............................................................... 489
    Number of DMA Mappings ............................................. 489
    DMA Secondary Base N and DMA Secondary Limit N .... 489
    DMA Primary Base N .................................................... 489
    DMA Control Field ...................................................... 490

Chapter 22: X86 CPU Compatibility
  Background ...................................................................... 491
    Legacy Signals ............................................................ 492
    Legacy Special Cycles .................................................. 493
    System Management Messages .................................... 493
  X86 Interrupt Support .................................................. 494
    APIC Interrupt Support ............................................... 495
      Legacy Method of Handling APIC Interrupts ............. 495
      HT Method of Handling APIC Interrupts .................... 498
      HT I/O Device Delivery of X86 Interrupts ................. 498
      EOI (End of Interrupt) Message ............................... 502
    Legacy Interrupts (8259 Interrupt Controllers) ........... 503
      The Legacy Method of Handling 8259 Interrupts ........ 503
      HT Method of Handling 8259 Interrupts .................... 505
      Signaling INTR ...................................................... 506
      Signaling Interrupt Acknowledge ............................. 507
      8259 EOI Command .................................................. 507
      Legacy NMI Signaling ............................................... 508
The A20 Mask ................................................................. 508
  The Legacy Method of Signaling A20M# .................. 509
  The HT Method of Signaling A20M# ......................... 510
System Management Mode (SMI# & SMIACT#) .......... 511
  SMM Applications ..................................................... 511
  Legacy SMM Signals ................................................... 512
  The HT Method of Signaling SMI# & SMIACT# .......... 513
Numeric Error Handling (FERR# and IGNNE#) .... 515
  Numeric Error Handling (The Original Method) ....... 516
  DOS-Compatible Error Handling ............................ 517
  HT Method of Signaling FERR# and IGNNE# .......... 519
    HT Method of FERR# Signaling ......................... 520
    HT Method of IGNNE# Signaling ....................... 520
X86 Instructions and Special Cycles ......................... 521
Appendix
Glossary of Terms ..................................................... 525
Index ................................................................. 541
Introduction to HyperTransport

This Chapter

This chapter discusses some of the motivations leading to the development of HyperTransport. It reviews some of the attributes that limit the ability of older generation I/O buses to keep pace with the increasing demands of new applications and advances in processor and memory technologies. The chapter then summarizes the key features behind the improved performance of HT over earlier buses.

The Next Chapter

The next chapter provides an overview of HT architecture, including the primary elements of HT technology and the relationship between them. The chapter describes the general features, capabilities, and limitations of HT and introduces the terminology and concepts necessary for in-depth discussions of the various HT topics in subsequent chapters.

Background: I/O Subsystem Bottlenecks

New I/O buses are typically developed in response to changing system requirements and to promote lower cost implementations. Current-generation I/O buses such as PCI are rapidly falling behind the capabilities of other system components such as processors and memory. Some of the reasons why the I/O bottlenecks are becoming more apparent are described below.
HyperTransport System Architecture

Server Or Desktop Computer: Three Subsystems

A server or desktop computer system is comprised of three major subsystems:

1. Processor (in servers, there may be more than one)
2. Main DRAM Memory. There are a number of different synchronous DRAM types, including SDRAM, DDR, and Rambus.
3. I/O (Input/Output devices). Generally, all components which are not processors or DRAM are lumped together in this subsystem group. This would include such things as graphics, mass storage, legacy hardware, and the buses required to support them: PCI, PCI-X, AGP, USB, IDE, etc.

CPU Speed Makes Other Subsystems Appear Slow

Because of improvements in CPU internal execution speed, processors are more demanding than ever when they access external resources such as memory and I/O. Each external read or write by the processor represents a huge performance hit compared to internal execution.

Multiple CPUs Aggravate The Problem

In systems with multiple CPUs, such as servers, the problem of accessing external devices becomes worse because of competition for access to system DRAM and the single set of I/O resources.

DRAM Memory Keeps Up Fairly Well

Although it is external to the processor(s), system DRAM memory keeps up fairly well with the increasing demands of CPUs for a couple of reasons. First, the performance penalty for accessing external memory is mitigated by the use of internal processor caches. Modern processors generally implement multiple levels of internal caches that run at the full CPU clock rate and are tuned for high "hit rates". Each fetch from an internal cache eliminates the need for an external bus cycle to memory.

In addition, in cases where an external memory fetch is required, DRAM technology and the use of synchronous bus interfaces to it (e.g. DDR, Rambus, etc.) have allowed it to maintain bandwidths comparable with the processor external bus rates.
I/O Bandwidth Has Not Kept Pace

While the processor internal speed has raced forward, and memory access speed has managed to follow along reasonably well with the help of caches, I/O subsystem evolution has not kept up.

This Slows Down The Processor

Although external DRAM accesses by processors can be minimized through the use of internal caches, there is no way to avoid external bus operations when accessing I/O devices. The processor must perform small, inefficient external transactions which then must find their way through the I/O subsystem to the bus hosting the device.

It Also Hurts Fast Peripherals

Similarly, bus master I/O devices using PCI or other subsystem buses to reach main memory are also hindered by the lack of bandwidth. Some modern peripheral devices (e.g. SCSI and IDE hard drives) are capable of running much faster than the busses they live on. This represents another system bottleneck. This is a particular problem in cases where applications are running that emphasize time-critical movement of data through the I/O subsystem over CPU processing.

Reducing I/O Bottlenecks

Two important schemes have been used to connect I/O devices to main memory. The first is the shared bus approach, as used in PCI and PCI-X. The second involves point-to-point component interconnects, and includes some proprietary busses as well as open architectures such as HyperTransport. These are described here, along with the advantages and disadvantages of each.

The Shared Bus Approach

Figure 1-1 on page 12 depicts the common “North-South” bridge PCI implementation. Note that the PCI bus acts as both an “add-in” bus for user peripheral cards and as an interconnect bus to memory for all devices residing on or below it. Even traffic to and from the USB and IDE controllers integrated in the South Bridge must cross the PCI bus to reach main memory.
HyperTransport System Architecture

Figure 1-1: Typical PCI North-South Bridge System
Chapter 1: Introduction to HyperTransport

Until recently, the topology shown in Figure 1-1 on page 12 has been very popular in desktop systems for a number of reasons, including:

1. A shared bus reduces the number of traces on the motherboard to a single set.
2. All of the devices located on the PCI bus are only one bridge interface away from the principal target of their transactions — main DRAM memory.
3. A single, very popular protocol (PCI) can be used for all embedded devices, add-in cards, and chipset components attached to the bus.

Unfortunately, some of the things that made this topology so popular also have made it difficult to fix the I/O bandwidth problems which have become more obvious as processors and memory have become faster.

A Shared Bus Runs At Limited Clock Speeds. The fact that multiple devices (including PCB connectors) attach to a shared bus means that trace lengths and electrical complexity will limit the maximum usable clock speed. For example, a generic PCI bus has a maximum clock speed of 33MHz; the PCI Specification permits increasing the clock speed to 66MHz, but the number of devices/connectors on the bus is very limited.

A Shared Bus May Be Host To Many Device Types. The requirements of devices on a shared bus may vary widely in terms of bandwidth needed, tolerance for bus access latency, typical data transfer size, etc. All of this complicates arbitration on the bus when multiple masters wish to initiate transactions.

Backward Compatibility Prevents Upgrading Performance. If a critical shared bus is based on an open architecture, especially one that defines user “add-in” connectors, then another problem in upgrading bus bandwidth is the need to maintain backward compatibility with all of the devices and cards already in existence. If the bus protocol is enhanced and a user installs an “older generation card”, then the bus must either revert back to the earlier protocol or lose its compatibility.

Special Problems If The Shared Bus Is PCI. As popular as it has been, PCI presents additional problems that contribute to performance limits:

1. PCI doesn’t support split transactions, resulting in inefficient retries.
2. Transaction size (there is no limit) isn’t known, which makes it difficult to size buffers and causes frequent disconnects by targets. Devices are also allowed to insert numerous wait states during each data phase.
3. All PCI transactions by I/O devices targeting main memory generally
2

HT Architectural Overview

The Previous Chapter

To understand why HT was developed, it is helpful to review the previous generation of I/O buses and interconnects. This chapter review the factors that limit the ability of older generation buses to keep pace with the increasing demands of new applications. Finally, this chapter discusses the key factors of the HT technology that provides its improved capability.

This Chapter

This chapter provides an overview of the HT architecture that defines the primary elements of HT technology and the relationship between these elements. This chapter summarizes the features, capabilities, and limitation of HT and provides the background information necessary for in-depth discussions of the various HT topics in later chapters.

The Next Chapter

The next chapter describes the function of each signal in the high- and low-speed HyperTransport signal groups.

General

HyperTransport provides a point-to-point interconnect that can be extended to support a wide range of devices. Figure 2-1 on page 21 illustrates a sample HT system with four internal links. HyperTransport provides a high-speed, high-performance, point-to-point dual simplex link for interconnecting IC components on a PCB. Data is transmitted from one device to another across the link.
HyperTransport System Architecture

The width of the link along with the clock frequency at which data is transferred are scalable:

- Link width ranges from 2 bits to 32-bits
- Clock Frequency ranges from 200MHz to 800MHz (and 1GHz in the future)

This scalability allows for a wide range of link performance and potential applications with bandwidths ranging from 200MB/s to 12.8GB/s.

At the current revision of the spec, 1.04, there is no support for connectors implying that all HyperTransport (HT) devices are soldered onto the motherboard. HyperTransport is technically an “inside-the-box” bus. In reality, connectors have been designed for systems that require board to board connections, and where analyzer interfaces are desired for debug.

Once again referring to Figure 2-1, the HT bus has been extended in the sample system via a series of devices known as tunnels. A tunnel is merely an HT device that performs some function, but in addition it contains a second HT interface that permits the connection of another HT device. In Figure 2-1, the tunnel devices provide connections to other I/O buses:

- Infiniband
- PCI-X
- Ethernet

The end device is termed a cave, which always represents the termination of a chain of devices that all reside on the same HT bus. Cave devices include a function, but no additional HT connection. The series of devices that comprise an HT bus is sometimes simply referred to as an HT chain.

Additional HT buses (i.e. chains) may be implemented in a given system by using a HT-to-HT bridge. In this way, a fabric of HT devices may be implemented. Refer to section entitled, “Extending the Topology” on page 33 for additional detail.
Chapter 2: HT Architectural Overview

Figure 2-1: Example HyperTransport System

Transfer Types Supported

HT supports two types of addressing semantics:

1. legacy PC, address-based semantics
2. messaging semantics common to networking environments
The first part of this book discusses the address-based semantics common to compatible PC implementations. Message-passing semantics are discussed in Chapter 19, entitled “Networking Extensions Overview,” on page 443.

**Address-Based Semantics**

The HT bus was initially implemented as a PC compatible solution that by definition uses Address-based semantics. This includes a 40-bit, or 1 Terabyte (TB) address space. Transactions specify locations within this address space that are to be read from or written to. The address space is divided into blocks that are allocated for particular functions, listed in Figure 2-2 on page 23.

HyperTransport does not contain dedicated I/O address space. Instead, CPU I/O space is mapped to high memory address range (FD_FC00_0000h—FD_FDFF_FFFFh). Each HyperTransport device is configured at initialization time by the boot ROM configuration software to respond to a range of memory address spaces. The devices are assigned addresses via the base address registers contained in the configuration register header. Note that these registers are based on the PCI Configuration registers, and are also mapped to memory space (FD_FE00_0000h—FD_FFFF_FFFFh. Unlike the PCI bus, there is no dedicated configuration address space.

Read and write request command packets contain a 40-bit address Addr[39:2]. Additional memory address ranges are used for interrupt signaling and system management messages. Details regarding the use of each range of address space is discussed in subsequent chapters that cover the related topic. For example, a detailed discussion of the configuration address space can be found in Chapter 13, entitled "Device Configuration," on page 305.
Data Transfer Type and Transaction Flow

The HT architecture supports several methods of data transfer between devices, including:

- Programmed I/O
- DMA
- Peer-to-peer

Each method is illustrated and described below. An overview of packet types and transactions is discussed later in this chapter.
3 Signal Groups

The Previous Chapter

The previous chapter provided an overview of the HT architecture that defines the primary elements of HT technology and the relationship between these elements. The chapter summarized the features, capabilities, and limitation of HT and provided the background information necessary for in-depth discussions of the various HT topics in later chapters.

This Chapter

This chapter describes the function of each signal in the high and low speed HyperTransport signal groups. The CAD, CTL, and CLK high speed signals are routed point-to-point as low-voltage differential pairs between two devices (or between a device and a connector in some cases). The RESET#, PWROK, LDTREQ#, and LDTSTOP# low speed signals are single-ended low voltage CMOS and may be bused to multiple devices. In addition, each device requires power supply and ground pins. Because the CAD bus width is scalable, the actual number of CAD and CLK signal pairs varies, as does the number of power and ground pins to the device.

The Next Chapter

The next chapter describes the use of HyperTransport control and data packets to construct HyperTransport link transactions. Control packet types include Information, Request, and Response variants; data packets contain a payload of 0-64 valid bytes. The transmission, structure, and use of each packet type is presented.

Introduction

Signals on each HyperTransport link fall into two groups: high speed signals associated with the sending and receiving of control and data packets, and miscellaneous low-speed signals required for such things as reset and power management. Whereas the low speed signals are not scalable and employ conventional low voltage CMOS signalling, the high speed signal group is scal-
HyperTransport System Architecture

able in terms of both bus width and clock rate, and each signal is actually a low-voltage differential signal pair.

While device pin count varies with scaling, signal group functions remain the same; the only real difference in signaling over a 32-bit link vs. a 2-bit link is the number of bit times required to shift information onto the bus.

The Signal Groups

As illustrated in Figure 3-1 on page 54, the high-speed HyperTransport signals on each link consist of an outbound (transmit) set of signals and an inbound (receive) set of signals for each device; these are routed point-to-point. Having two sets of uni-directional signals allows concurrent traffic. In addition, there is one set of low speed signals that may be bused to multiple devices.

*Figure 3-1: HyperTransport Signal Groups*
Chapter 3: Signal Groups

The High Speed Signals (One Set In Each Direction)

Each high-speed signal is actually a differential signal pair. CAD (Command/Address/Data) information consists of the two basic types of HyperTransport packets: control and data. When a link transmitter sends packets on the CAD bus, the receive side of the interface uses the CLK and CTL signals, also supplied by the transmitter, to latch in packet information during each bit time. CTL distinguishes control packets from data packets.

The CAD Signal Group

The CAD bus is always driven by the transmitter side of a link, and is comprised of signal pairs that carry HyperTransport requests, responses, and data. Each CAD bus may consist of between 2 bits (two differential signal pairs) and 32 bits (thirty-two differential signal pairs). The HyperTransport specification permits the CAD bus width to be different (asymmetrical) for the two directions. To enable the corresponding receiver to make a distinction as to the type of information currently being sent over the CAD bus, the transmitter also drives the CTL signal (see the following description).

Control Signal (CTL)

This signal pair is driven by the transmitter to qualify the information being sent concurrently over the CAD signals. If this signal is asserted (high), the transmitter is indicating that it is sending a control packet; if deasserted, the transmitter is sending a data packet. The receiver uses this information when routing incoming CAD information to appropriate request queues, data buffers, etc. There is one (and only one) CTL signal for each link direction, regardless of the width of the CAD bus.

Clock Signal(s) (CLK)

As a source-synchronous connection, each HyperTransport transmitter sends a differential clock signal along with CAD and CTL signals to the receiver at the other end of the link. There is one CLK signal pair for each byte of CAD width. While the timing on each clock pair is the same, replicating clocks help in routing of CAD signal pairs with respect to their clock signals. The current Hyper-Transport specification allows clock speeds from 200MHz (default) to 800MHz.
HyperTransport System Architecture

Scaling Hazards: Burden Is On The Transmitter

It is a requirement in HyperTransport that the transmitter side of each link must be aware of the capabilities of its corresponding receiver and avoid the double hazard of a scalable bus: running at a faster clock rate than the receiver can handle or using a wider data path than the receiver supports. Because the link is not a shared bus, the transmitter side of each device is concerned with the capabilities of only one target. Refer to “Link Initialization” on page 282 for a description of how HyperTransport links are initialized and configured to avoid these problems.

The Low SpeedSignals

Power OK (PWROK) And Reset (RESET#)

PWROK used with RESET# indicates to HyperTransport devices whether a Cold or Warm Reset is in progress. Which system logic component is responsible for managing the PWROK and RESET# signals is beyond the scope of the HyperTransport specification, but timing and use of the signals are defined. The basic use of the signals includes:

- At power up, PWROK is asserted by system logic when it can be guaranteed that system power and clocks related to HyperTransport are within proper limits.
- RESET# is asserted by system logic to indicate that a reset is required. The state of PWROK when RESET# is seen asserted indicates the type of reset to be performed. PWROK and RESET# both asserted is a warm reset; PWROK deasserted and RESET# asserted indicates cold reset.
- After initial system power up, reset, and initialization, a cold or warm reset may also be generated under software control writing configuration registers in the host bridge.

The HyperTransport specification describes the actions to be taken by devices during either type of reset event. Refer to Chapter 12, entitled "Reset & Initialization,” on page 275 for a thorough discussion of how PWROK and RESET are used during system power-up and initialization.
Chapter 3: Signal Groups

LDTSTOP#

(Note: the signal names LDTSTOP# and LDTREQ# were carried forward from the earlier name AMD assigned to HyperTransport technology — Lightning Data Transfer).

LDTSTOP# is an input to HyperTransport devices which is asserted by system logic to enable and disable link activity during power management state transitions. Support for this signal is optional for HyperTransport devices.

A transmitter which detects LDTSTOP# asserted finishes sending any control packet in progress, then commences a disconnect NOP sequence followed by disabling its output drivers (if so enabled in the transmitter’s Configuration Space Tri-State Enable Bit). Upon receipt of the disconnect NOP sequence, the target also turns off its input receivers (if similarly enabled in it’s Configuration Space Tri-State Enable Bit).

Later, when the transmitter detects LDTSTOP# deasserted, it re-enables its drivers and begins the initialization sequence. A receiver that responds to LDTSTOP# deasserted turns its input receivers on.

LDTREQ#

LDTREQ# is a wire-or’d output from HyperTransport devices that is used to request system logic to re-enable links previously disabled using the LDTSTOP# mechanism. Upon receipt of the LDTREQ# signal from one or more HyperTransport devices, system logic (typically the South Bridge) deasserts LDTSTOP# which triggers the sequence described previously. Specifically, the LDTREQ# signal indicates that a HyperTransport transaction is required somewhere in a system that is currently in the ACPI C3 state; the system is required to transition to the C0 state. Support for this signal is optional for HyperTransport devices.

Where Are The Interrupt, Error, And Wait State Signals?

The HyperTransport specification eliminates a number of control signals that are commonly found on other buses. While devices are not prohibited from implementing signals beyond those defined in the specification, HyperTransport is a generic, simple interface and handles interrupts, errors, and data wait states in the following general way:
4 Packet Protocol

The Previous Chapter

The previous chapter described the function of each signal in the high and low speed HyperTransport signal groups. The CAD, CTL, and CLK high speed signals are routed point-to-point as low-voltage differential pairs between two devices (or between a device and a connector in some cases). The RESET#, PWOK, LDTREQ#, and LDTSTOP# low speed signals are single-ended low voltage CMOS and may be bused to multiple devices. In addition, each device requires power supply and ground pins. Because the CAD bus width is scalable, the actual number of CAD and CLK signal pairs varies, as does the number of power and ground pins to the device.

This Chapter

This chapter describes the use of HyperTransport control and data packets to construct HyperTransport link transactions. Control packet types include Information, Request, and Response variants; data packets contain a payload of 0-64 valid bytes. The transmission, structure, and use of each packet type is presented.

The Next Chapter

The next chapter describes HyperTransport flow control, used to throttle the movement of packets across each link interface. On a high-performance connection such as HyperTransport, efficient management of transaction flow is nearly as important as the raw bandwidth made possible by clock speed and data bus width. Topics covered here include background information on bus flow control and the initialization and use of the HyperTransport virtual channel flow control buffer mechanism defined for each transmitter-receiver pair.
The Packet-Based Protocol

HyperTransport employs a packet-based protocol in which all information — address, commands, and data — travel in packets which are multiples of four bytes each. Packets are used in link management (e.g. flow control and error reporting) and as building blocks in constructing more complex transactions such as read and write data transfers.

It should be noted that, while packet descriptions in this chapter are in terms of bytes, the link’s bidirectional interface width (2, 4, 8, 16, or 32 bits) ultimately determines the amount of packet information sent during each bit time on HyperTransport links. There are two bit times per clock period.

Before looking at packet function and use, the following sections describe the mechanics of packet delivery over 2, 4, 8, 16, and 32 bit scalable link interfaces.

8 Bit Interfaces

For 8-bit interfaces, one byte of packet information may be sent in each bit time. For example, a 4-byte request packet would be sent by the transmitter during four adjacent bit times, least significant byte first as shown in Figure 4-1 on page 61. Total time to complete a four-byte packet is two clock periods.
Example:
4 Byte Packet On An 8-Bit Interface

Byte 0 - 3

Device A

CAD0-7

Device B

D, C, B, A
Interfaces Narrower Than 8 Bits

For link interfaces which are narrower than 8 bits, the first byte of packet information is shifted out over multiple bit times, least significant bits first. Referring to Figure 4-2 on page 62, a 2-bit interface would require four bit times to transmit each byte of information. After the first byte is sent, subsequent bytes in the packet are shifted out in the same manner. Total time to complete four byte packet: eight clock periods.

Figure 4-2: Four Byte Packet On A 2-Bit Interface
Interfaces Wider Than 8 Bits

For 16 or 32 bit interfaces, packet delivery is accelerated by sending multiple bytes of packet information in parallel with each other.

16 Bit Interfaces

On 16-bit interfaces, two bytes of information may be sent in each bit time. Referring to Figure 4-3 on page 63, note that even numbered bytes travel on the lower portion of the 16 bit interface, odd numbered bytes on the upper portion.

Figure 4-3: Four Byte Packet On A 16-Bit Interface
Flow Control

The Previous Chapter

The previous chapter described the use of HyperTransport control and data packets to construct HyperTransport link transactions. Control packet types include Information, Request, and Response variants; data packets contain a payload of 0-64 valid bytes. The transmission, structure, and use of each packet type is presented.

This Chapter

This chapter describes HyperTransport flow control, used to throttle the movement of packets across each link interface. On a high-performance connection such as HyperTransport, efficient management of transaction flow is nearly as important as the raw bandwidth made possible by clock speed and data bus width. Topics covered here include background information on bus flow control and the initialization and use of the HyperTransport virtual channel flow control buffer mechanism defined for each transmitter-receiver pair.

The Next Chapter

The next chapter describes the rules governing acceptance, forwarding, and rejection of packets seen by HyperTransport devices. Several factors come into play in routing, including the packet type, the direction it is moving, and the device type which sees it. A related topic also covered in this chapter is the fairness algorithm used by a tunnel device as it inserts its own packets into the traffic it forwards upstream on behalf of devices below it. The HyperTransport specification provides a fairness algorithm and a hardware method for tunnel management packet insertion.

The Problem

On any bus where an agent initiates the exchange of information (commands, data, status, etc.) with a target, a number of things can cause a delay (or even end) the normal completion of the intended transfer. The throttling of information delivery on a bus is referred to as flow control. PCI is a good example of a
bus protocol which has reasonably high burst bandwidth, but is subject to performance hits caused by an unsophisticated flow control mechanism. Before looking at the HyperTransport approach to flow control, some of the general problems in bus flow control are described in the following section in terms of the PCI protocol. Refer to Figure 5-1 on page 100.

*Figure 5-1: PCI Interface Handshake Signals*
How PCI Handles Flow Control

While the PCI specification permits 64-bit data bus and 66MHz clock options, a generic PCI bus carries only 32 bits (4 bytes) of data and runs at a 33MHz clock speed. This means that the burst bandwidth for this bus is 132MB/s (4 bytes x 33MHz = 132MB/s). In many systems the PCI bus is populated by all sorts of high- and low-performance peripherals such as hard drives, graphics adapters, and serial port adapters. All PCI bus master devices must take turns accessing the shared bus and performing their transfers. The priority of a bus master in accessing the bus and the amount of time it is allowed to retain control of the bus is a function of PCI arbitration. In a typical computer system, the PCI arbiter logic resides in the system chipset.

Once a PCI bus master has won arbitration and verifies the bus is idle, it commences its transaction. After decoding the address and command sent by the master, one target claims the cycle by asserting a signal called DEVSEL#. At this point, if both devices are prepared, either write data will be sent by the initiator or read data will be returned by the target. For cases where either the master or target are not prepared for full-speed transfer of some or all of the data, flow control comes into play. In PCI there are a number of cases that must be dealt with.

PCI Target Flow Control Problems

PCI Target Not Ready To Start. In some cases, a PCI device being targeted for transmission is not prepared to transfer any data at all. This could happen if the target is off-line, does not have buffer space for write data being sent to it, or does not have requested read data available. It may also occur if the transaction must cross a bridge device to a different bus. Many bus protocols, including PCI, place a limit on how long the bus may be stalled before completing a transaction; in cases where a target can’t meet the requirement for even the first data, a mechanism is required to indicate the transaction should be abandoned and re-attempted later. PCI calls the target cancellation of a transaction (without transferring any data) a Retry; a Retry is indicated when a target asserts the STOP# signal (instead if TRDY#) in the first data phase.

PCI Target Starts Data Transfer, But Can’t Continue. Another possibility is that a transaction started properly, some data has transferred, but at some point before completion the target “realizes” it can’t continue the transfer within the time allowed by the protocol. The target must indicate to the master that the transaction must be suspended (and resumed later at the point where it
HyperTransport System Architecture

left off). PCI calls this target suspension of a transaction (with a partial transfer of data) a Disconnect. A Disconnect is signalled when the target asserts the STOP# signal in a data phase after the first one.

**PCI Target Starts, Can Continue, But Needs More Time.** Sometimes a transaction is underway and the target requires additional time to complete transmission of a particular data item; in this case, it does not need to suspend the transaction altogether, but simply stretch one or more data phases. The generic name for this is wait-state insertion. Wait states are a reasonable alternative to Retry and Disconnect if there are not too many of them; when there are excessive wait states, bus performance would be better served by the devices giving up the bus and allowing it to be used by other devices while they prepare for the resumption of the suspended transaction. PCI targets de-assert the TRDY# signal during any data phase to indicate wait states. A target must be prepared to complete each data phase within 8 PCI clocks (maximum of seven wait states), except for the first data phase which it must complete within 16 clocks. If a target cannot meet the “16 and 8 tick” rules for completing a data phase, it must signal Retry or Disconnect instead.

**PCI Initiator Flow Control Problems**

While many flow control problems are associated with the target of a transaction, there are a couple which may occur on the initiator side. Again, the cases are described in terms of PCI protocol.

**PCI Initiator Starts, But Can’t Continue.** Some bus protocols also allow an initiator to break off a transaction early in the event it can’t accept the next read data or source the next write data within the time allowed by the protocol — even with wait states. PCI initiators suspend transactions simply by de-asserting the FRAME# signal early. As a rule, the master will re-arbitrate later for the PCI bus and perform a new transaction which picks up from where it left off previously.

**PCI Initiator Starts, Can Continue, But Needs Wait-States.** Some bus protocols allow an initiator to insert wait states in a transfer, just as the target may. Other bus protocols (e.g. PCI-X) only allow targets to insert wait states — based on the assumption that a device which starts a transaction should be ready to complete it before requesting the bus. In any case, PCI initiators de-assert the IRDY# signal to indicate wait states. An initiator must be prepared to complete each data phase within 8 clocks (maximum of seven wait states); if it can’t meet this rule for any data phase, it must instead suspend the transaction by de-asserting FRAME#.
All PCI Flow Control Problems Hurt Performance

Each of the initiator and target flow control problems just described impact PCI bus performance for both the devices involved in the transfer, and for devices waiting to access the bus. While not every transaction is afflicted with target retries and disconnects, or early de-assertion of FRAME# by initiators, they happen enough to make effective bandwidth considerably less than 132MB/s on the PCI bus. In addition, arbitration and flow control uncertainties make system performance difficult to estimate.

HyperTransport Flow Control: Overview

All of the flow control problems described previously for PCI severely hurt bus performance and would be even less acceptable on a very high-performance connection. The flow control scheme used in HyperTransport applies independently to each transmitter-receiver pair on each link. The basic features include the following.

Packets Never Start Unless Completion Assured

All transfers across HyperTransport links are packet based. No link transmitter ever starts a packet transfer unless it is known the packet can be accepted by the receiver. This is accomplished with the “coupon based” flow control scheme described in this section, and eliminates the need for the Retry and Disconnect mechanisms used in PCI.

Transfer Length Is Always Known

Hypertransport control packets have a fixed size (four or eight bytes) and data packets have a known and maximum transfer length, unlike PCI data transfers. This makes buffer sizing and flow control much more straightforward as both transmitter and receiver are aware of their actual transfer commitments. It also makes the interleaving of control packets with data packets much simpler.
The Previous Chapter

The previous chapter described the rules governing acceptance, forwarding, and rejection of packets seen by HyperTransport devices. Several factors come into play in routing, including the packet type, the direction it is moving, and the device type which sees it. A related topic also covered in this chapter is the fairness algorithm used by a tunnel device as it inserts its own packets into the traffic it forwards upstream on behalf of devices below it. The HyperTransport specification provides a fairness algorithm and a hardware method for tunnel management packet insertion.

This Chapter

This chapter describes the ordering rules which apply to packets associated with the three types of HyperTransport I/O traffic: PIO, DMA, and Peer-to-Peer. Depending on whether compatibility with the full producer-consumer ordering model used in PCI is required or relaxed ordering is permissible, attribute bits in request and response packets may be set or cleared. These bits are defined by the requester and are used by devices in the path to the target, and within the target, to enforce proper ordering. HyperTransport applies dedicated sets of ordering rules for upstream I/O traffic, downstream I/O traffic, and the special ordering required of host bridges and in double-hosted chains. Refer to Chapter 20, entitled "I/O Compatibility," on page 457 for a description of the additional ordering requirements when interfacing HyperTransport to other compatible protocols (e.g. PCI, PCI-X, and AGP).

The Next Chapter

In the next chapter, examples are presented which apply the packet principles described in the preceding chapter. The examples also entail more complex system transactions than discussed previously, including reads, posted and non-posted writes, and atomic read-modify-write operations.
The Purpose Of Ordering Rules

Some of the important reasons for enforcing ordering rules on packets moving through HyperTransport include the following:

Maintain Data Coherency

If transactions are in some way dependent on each other, a method is required to assure that they complete in a deterministic way. For example, if Device A performs a write transaction targeting main memory and then follows it with a read request targeting the same location, what data will the read transaction return? HyperTransport ordering seeks to make such events predictable (deterministic) and to match the intent of the programmer. Note that, compared to a shared bus such as PCI, HyperTransport transaction ordering is complicated somewhat by point-to-point connections which result in target devices on the same chain (logical bus) being at different levels of fabric hierarchy.

Avoid Deadlocks

Another reason for ordering rules is to handle cases where the completion of two separate transactions are each dependent on the other completing first. HyperTransport ordering includes a number of rules for deadlock avoidance. Some of the rules are in the specification because of known deadlock hazards associated with other buses to which HyperTransport may interface (e.g. PCI).

Support Legacy buses

One of the principal roles of HyperTransport is to serve as a backbone bus which is bridged to other peripheral buses. HyperTransport explicitly supports PCI, PCI-X, and AGP and the ordering requirements of those buses.

Maximize Performance

Finally, HyperTransport permits devices in the path to the target, and the target itself, some flexibility in reordering packets around each other to enhance performance. When acceptable, relaxed ordering may be enabled by the requester on a per-transaction basis using attribute bits in request and response packets.
Chapter 6: I/O Ordering

Introduction: Three Types Of Traffic Flow

Hypertransport defines three types of traffic: Programmed I/O (PIO), Direct Memory Access (DMA), and Peer-to-Peer. Figure 6-1 on page 121 depicts the three types of traffic.

1. Programmed I/O traffic originates at the host bridge on behalf of the CPU and targets I/O or Memory Mapped I/O in one of the peripherals. These types of transactions often are generated by CPU to set up peripherals for bus master activity, check status, program configuration space, etc.

2. DMA traffic originates at a bus master peripheral and typically targets main memory. This traffic is used so that the CPU may be off-loaded from the burden of moving large amounts of data to and from the I/O subsystem. Generally, the CPU uses a few PIO instructions to program the peripheral device with information about a required DMA transfer (transfer size, target address in memory, read or write, etc.), then performs some other task while the DMA transfer is carried out. When the transfer is complete, the DMA device may generate an interrupt message to inform the CPU.

3. Peer-to-Peer traffic is generated by an interior node and targets another interior node. In HyperTransport, direct peer-to-peer traffic is not allowed. As indicated in Figure 6-1 on page 121, the request is issued upstream and must travel to the host bridge. The host bridge examines the address and determines whether the request should be reflected downstream. If the request is non-posted, the response will similarly travel from the target back up to the host bridge and then be reissued to the original requester.

Figure 6-1: PIO, DMA, And Peer-to-Peer Traffic
HyperTransport System Architecture

The Ordering Rules

HyperTransport packet ordering rules are divided into groups: general rules, rules for upstream I/O ordering, and rules for downstream ordering. Even the peer-to-peer example in Figure 6-1 on page 121 can be broken into two parts: the request moving to the bridge (covered by upstream ordering rules) and the reflection of the request downstream to the peer-to-peer target (covered by downstream I/O ordering rules). Refer to Chapter 20, entitled “I/O Compatibility,” on page 457 for a discussion of ordering when packets move between HyperTransport and another protocol (PCI, PCI-X, or AGP).

General I/O Ordering Limits

Ordering Covers Targets At Same Hierarchy Level

Ordering rules only apply to the order in which operations are detected by targets at the same level in the HyperTransport fabric hierarchy. Referring to Figure 6-2 on page 122, assume that two peer-to-peer writes targeting devices on two different chains have been performed by the end device in chain 0.

Figure 6-2: Targets At Different Levels In Hierarchy And In Different Chains
Chapter 6: I/O Ordering

In the illustration Figure 6-2 on page 122, assume that Request A, a write transaction, is sent first. This is immediately followed by Request B, another write request. HyperTransport general ordering rules are then applied:

1. Upstream ordering rules assure that the two writes (Request A and Request B) arrive at the host bridge in the order they were generated.
2. When the host bridge then reflects the two write transactions downstream onto the separate chains (Chain 1 and Chain 2), downstream ordering rules guarantee that they will leave the host bridge in the order they arrived.
3. Once the two writes reach their respective chains, there is no way to guarantee that they will arrive at their respective targets in the order the requester intended because the ultimate targets are at different levels in the hierarchy.
4. The HyperTransport specification indicates that if the requester must be certain of the completion order at the targets, it should either poll the target of Request A for completion before issuing Request B or use a non-posted write for Request A and wait for the response to return before sending Request B.

Read And Non-Posted Write Completion At Target

Non-posted transactions issued by one requester to the same target are required to complete at the target in the order they were issued by the requester. This means that any combination of reads and non-posted writes must complete at the target in the original order they were issued. However, there is no ordering guarantee on the responses which are returned for each.

Figure 6-3: Non-Posted Requests And Responses At Target
The previous chapter described the ordering rules which apply to packets associated with the three types of HyperTransport I/O traffic: PIO, DMA, and Peer-to-Peer. Depending on whether compatibility with the full producer-consumer ordering model used in PCI is required or relaxed ordering is permissible, attribute bits in request and response packets may be set or cleared. These bits are defined by the requester and are used by devices in the path to the target, and within the target, to enforce proper ordering. HyperTransport applies dedicated sets of ordering rules for upstream I/O traffic, downstream I/O traffic, and the special ordering required of host bridges and in double-hosted chains. Refer to Chapter 20, entitled "I/O Compatibility," on page 457 for a description of the additional ordering requirements when interfacing HyperTransport to other compatible protocols (e.g. PCI, PCI-X, and AGP).

This Chapter

In this chapter, examples are presented which apply the packet principles in the preceding chapters and includes more complex system transactions, not previously discussed. The examples include reads, posted and non-posted writes, and atomic read-modify-write.

The Next Chapter

HT uses an interrupt signaling scheme very similar to PCI’s Message Signaled Interrupts. The next chapter defines how HT delivers interrupts to the Host Bridge via posted memory writes. This chapter also defines an End of Interrupt message and details the mechanism that HT uses for configuring and setting up interrupt transactions (which is different from thePCI-defined mechanisms).
HypterTransport System Architecture

Packets As Transaction Building Blocks

HyperTransport control packet types — information, request, and response are used in various combinations to accomplish transactions. In many transactions, data packets are also used with the control packets to carry a data payload ranging from 0-64 valid bytes. Transactions start when the transmit interface of a device sends an information or request control packet. Any bridges or tunnels in the path between a requester and the ultimate target have responsibilities for forwarding any request, response, and data packets associated with the transfer in the proper direction. Note: This chapter highlights key packet fields used in the construction of HyperTransport transactions; refer to Chapter 4, entitled "Packet Protocol," on page 59 for a more complete description of HyperTransport packets and the bit fields associated with them.

Table 7-1 on page 140 summarizes the interaction between a request agent and the ultimate target of a HyperTransport transaction following the sending of various types of information and request control packets.

Table 7-1: Implications Of Sending Information And Request Control Packets

<table>
<thead>
<tr>
<th>Packet Type</th>
<th>Command Name</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information</td>
<td>NOP</td>
<td>Used by each node transmitter to indicate the idle condition, report receiver flow control updates, and send other miscellaneous information to its corresponding receiver. These packets are not forwarded by the receiver and no response or data is associated with them.</td>
</tr>
<tr>
<td>Information</td>
<td>Sync/Error</td>
<td>Sent by each node transmitter during link synchronization or by a device enabled to report errors using the Sync flood mechanism to indicate the need for link reset and re-synchronization. During a Sync flood, each recipient re-issues the Sync packets onto all outgoing links on the chain until reset is detected. There are no response or data packets associated with a Sync packet.</td>
</tr>
</tbody>
</table>
# Chapter 7: Transaction Examples

Table 7-1: Implications Of Sending Information And Request Control Packets

<table>
<thead>
<tr>
<th>Packet Type</th>
<th>Command Name</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td><strong>Sized Write</strong></td>
<td>A posted sized write is used to initiate a write transfer of dwords or bytes of data to a target. For dword writes, the 1-16 dword data packet immediately follows the write request. For byte writes, a single dword “byte mask” precedes a data packet of 1-8 dwords (containing up to 32 valid bytes). No response is ever returned to a posted write and devices in the target path may deallocate buffers as soon as the request and data are forwarded.</td>
</tr>
<tr>
<td></td>
<td><strong>(Non-Posted)</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>dword or byte transfers OK</strong></td>
<td></td>
</tr>
<tr>
<td>Request</td>
<td><strong>Sized Write</strong></td>
<td>A non-posted sized write is also used to initiate a write transfer of dwords or bytes of data to a target. For dword writes, the 1-16 dword data packet immediately follows the write request. For byte writes, a single dword “byte mask” precedes a data packet of 1-8 dwords (containing up to 32 valid bytes). The <strong>Target Done</strong> response will be sent when the write completes (either by the target or by an EOC device). Bridges in the target path must track outstanding non-posted write requests until the target done response is returned.</td>
</tr>
<tr>
<td></td>
<td><strong>(Non-Posted)</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>dword or byte transfers OK</strong></td>
<td></td>
</tr>
<tr>
<td>Request</td>
<td><strong>Broadcast</strong></td>
<td>Broadcast messages originate at the host bridge, and are accepted and propagated downstream on all links by each device which sees them. As they are posted requests, there is no response and devices in the target path may deallocate buffers as soon as the broadcast message request is forwarded.</td>
</tr>
<tr>
<td></td>
<td><strong>Message</strong></td>
<td></td>
</tr>
</tbody>
</table>
# HypterTransport System Architecture

## Table 7-1: Implications Of Sending Information And Request Control Packets

<table>
<thead>
<tr>
<th>Packet Type</th>
<th>Command Name</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>Sized Read dword or byte transfers OK</td>
<td>A sized read is used to initiate a read transfer of dwords or bytes from a target. For byte reads, a single dword of data is returned immediately after the read response. For dword reads, 1-16 dwords are returned immediately after the read response. Bridges in the target path must track all outstanding read requests until the read response and data are returned.</td>
</tr>
<tr>
<td>Request</td>
<td>Flush</td>
<td>Issued by a requester to force its preceding posted writes in the same transaction stream to the Host Bridge. This is a non-posted request and the Host Bridge returns a Target Done Response when the flush of all previous posted writes for this source is completed to memory (or to the destination chain in a peer-to-peer transaction). Bridges in the target path must track outstanding Flush requests until the Target Done Response is returned. There is no data packet associated with the Flush.</td>
</tr>
<tr>
<td>Request</td>
<td>Fence</td>
<td>Issued to force the host bridge to place a barrier between previous and subsequent posted writes in all transaction streams. The Host Bridge will push previous writes for all streams to memory before allowing any subsequent posted writes with (PassPW clear) to be processed. Unlike Flush, this command is posted. There will be no response; devices in the target path may deallocate buffers as soon as Fence request is forwarded. There is no data packet associated with the Fence.</td>
</tr>
</tbody>
</table>
Chapter 7: Transaction Examples

Table 7-1: Implications Of Sending Information And Request Control Packets

<table>
<thead>
<tr>
<th>Packet Type</th>
<th>Command Name</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>Atomic RMW</td>
<td>Issued by a requester seeking to perform a read-modify-write of a memory location in a single transaction. This hybrid request causes a transaction made up of a <em>non-posted write</em> operation followed by a <em>read response with data</em>. There are two variants of Atomic RMW (Fetch &amp; Add and Compare &amp; Swap.) Because Atomic RMW requests are always non-posted, bridges in the path must track outstanding Atomic RMW requests until the response/data are returned.</td>
</tr>
<tr>
<td>Response</td>
<td>Read Response</td>
<td>Issued by a target when it is ready to either return previously requested read data (see Sized Read and Atomic RMW requests) or an error indication that the request did not complete properly. The read response immediately precedes read data being returned by the target. If an error occurred, the requested amount of data is returned anyway; response error bits will indicate that it is not valid and whether the response was sourced by the original target or an end-of-chain device.</td>
</tr>
<tr>
<td>Response</td>
<td>Target Done</td>
<td>Issued by a target to confirm the completion of an earlier non-posted write or Flush request. There is no data packet associated with a Target Done response. If an error occurred in completing the original request, the target done response error bits indicate the failure and whether it occurred at the original target or the request was inadvertently sent to an end-of-chain device.</td>
</tr>
</tbody>
</table>
8 HT Interrupts

The Previous Chapter
To review the principles of HT transactions and to provide a more comprehensive understanding, the previous chapter presented examples of complex system transactions, including reads, posted and non-posted writes, and atomic read-modify-write.

This Chapter
HT uses an interrupt signaling scheme very similar to PCI’s Message Signaled Interrupts. This chapter defines how HT delivers interrupts to the Host Bridge via posted memory writes. This chapter also defines an End of Interrupt message and details the mechanism that HT uses for configuring and setting up interrupt transactions (which is different from the PCI-defined mechanisms).

The Next Chapter
Rather than requiring HT devices to incorporate additional pins for signaling system-related items such as power management events, System Management messages are defined that permit signaling via transactions, thereby creating virtual wires. The next chapter defines the mechanism used to send these messages, by detailing the System Management packets and protocol. Some System Management functions such as changing the operating frequency of an HT link require that transactions be stopped. The chapter also introduces the ability to temporarily disconnect the links to allow these types of state changes, and to save power.

Introduction
HT, unlike most legacy I/O bus implementations, does not define the use of interrupt pins, nor an interrupt controller. Instead, interrupt delivery is distributed to the HT devices themselves. Each device delivers interrupts by performing memory writes to memory address locations reserved for that purpose. The data written to these locations provides information that historically comes
from or is handled by an interrupt controller (such as interrupt priority and vector information that specifies the location of the interrupt service routine). This method of interrupt delivery is commonly referred to as Message Signaled Interrupts.

HT supports message signaled interrupts via two message types:

- **Interrupt Request message** — Interrupt requests are forwarded upstream as sized write transactions that target a reserved interrupt request address range. The host bridge receives these packets and based on the target address recognizes the transaction as an interrupt request. The specific actions taken by the bridge to process the interrupt request is platform-specific and not specified.

- **End of Interrupt message** — HT also supports an End Of Interrupt (EOI) message that may be used by devices that require confirmation that their interrupt service routine has completed. These messages originate at the host and are forwarded downstream as a broadcast. Like the interrupt request message, the EOI request packet address must also fall within the reserved address range.

**Discovering a Device’s Interrupt Requirements**

HT defines an interrupt capability block as illustrated in Figure 8-1 on page 201. The presence of the capability block in configuration space indicates that the HT device uses interrupts. This block, named the Interrupt Discovery and Configuration Capability block, defines the number of interrupt sources each HyperTransport technology function can generate. This block also allows software to configure each interrupt independently.
Chapter 8: HT Interrupts

The Interrupt Message Address Range

The reserved address range used by the Interrupt Request and EOI messages is pictured in Figure 8-2 on page 202. This 3984MB address range is mapped from location FD_0000_0000h to FD_F8FF_FFFFh.

The specified range reserved for the interrupt request packets seems straightforward until one looks at the interrupt request packet definition (Figure 8-2 on page 202). Note that the address field (byte 7) defined by the specification includes only Addr[39:32]. These upper eight address bits identify a 4GB
address range starting at FD_0000_0000h. When reviewing the address map in Figure 8-2, it can be seen that several reserved address blocks fall within this 4GB address range — from Legacy PCI Ack through Configuration.

Figure 8-2: Interrupt Request and EOI Message Reserved Address Range

Figure 8-3 on page 203 depicts the format of the Interrupt Request packet. Note that the specification defines only Addr[39-32] to identify the interrupt packet address. If the interrupt request and EOI packets were limited to Addr[39:32], the host could not differentiate the interrupt packets from packets associated with other address ranges and functions.

The address associated with the interrupt packets must include additional address bits to distinguish between the different address ranges. If the Host Bridge is to resolve the address to within the specified interrupt address range (FD_0000_0000h to FD_F8FF_FFFFh), then Addr[31:24] must be included within the interrupt packets. Verification of the specification’s intent can be found in the x86 compatibility definitions, which specify Addr[31:24] be delivered in the IntrInfo[31:24] field of the interrupt packet. To maintain compatibility with earlier versions of HT implementations, the specification sets a default value of F8h for IntrInfo[31:24].
For system platform implementations other than x86, the specification leaves open the possibility of the interrupt range being extended, but does not explicitly state that the interrupt range can be extended in the absence of the PIC IACK, System Management, and IO mappings. For example, some platforms may only need support for the interrupt and configuration packets. This would require the use of Addr[39:26], thereby permitting the Host Bridge to distinguish between the interrupt and configuration requests.
9 System Management

The Previous Chapter

HT uses an interrupt signaling scheme very similar to PCI’s Message Signaled Interrupts. The previous chapter defines how HT delivers interrupts to the Host Bridge via posted memory writes. The chapter also defined an End of Interrupt Message and details the mechanism that HT uses for configuring and setting up interrupt transactions (which is different from the PCI-defined mechanisms).

This Chapter

Rather than requiring HT devices to incorporate additional pins for signaling system-related items such as power management events, System Management messages are defined that permit signaling via transactions, thereby creating virtual wires. This chapter defines the mechanism used to send these messages, by detailing the System Management packets and protocol. Some System Management functions such as changing the operating frequency of an HT link require that transactions be stopped. This chapter also introduces the ability to temporarily disconnect the links to allow these types of state change and to save power.

The Next Chapter

There are two aspects in dealing with link or internal errors in HyperTransport: detection and handling. The next chapter describes the error types defined by the specification and what devices may do about them. Some devices may choose to detect and handle some errors but not others. The PCI configuration space registers, used to program the error strategy and log errors, are described here — as are the reporting mechanisms: error response, fatal and non-fatal interrupts, and Sync flood.
HyperTransport System Architecture

System Management Transactions

HT provides a message passing mechanism between the Host Bridge and the System Management Controller (SMC). One of the primary purposes of HT messages is to eliminate dedicated pins and traces that would otherwise be required to signal various events, reducing pin count and cost. These System Management (SM) messages are delivered via packets that support a wide variety of functions including:

- HT Power Management
- X86 Power Management
- X86 Legacy CPU Signalling (e.g. A20M, FERR#, and IGNNE#)

HT System Management messages in conjunction with LDTSTOP# may be used to support operations such as changes in operating frequency and link width, or to disable the links to save power. It is also through System Management (SM) requests that many of the x86 compatibility mechanisms are accomplished as indicated above. Further, x86 platforms are required to support SM and LDTSTOP# for power management. Power Management support for HT devices is optional in non-x86 platforms; however, many non-x86 systems do support power management. Note also that the specification requires all HT devices to forward SM packets in both directions.

Sources of SM Request

System Management requests may be either sent in the upstream or downstream direction, as illustrated in Figure 9-1 on page 217. All SM requests moving upstream originate at the System Management Controller (SMC) and downstream requests originate at the Host Bridge. Note that the SMC typically resides in the south bridge (or I/O Controller Hub) where the legacy signals typically originate and where power management registers reside.
System Management transactions are recognized by their assigned address range. The HT specification reserves a 1MB address range for system management transactions from FD_F910_0000h to FD_F91F_FFFFh. In reality, only the upper address bits are needed to identify that the transaction falls within the assigned 1MB range. SM request packets include only the upper 20 bits (A39:A20) of the HT address for identifying the SM range (FD_F91h). Note that the lower 5 nibbles (or 20 bits) of the address are not defined and could theoretically be any value between 0_0000h and F_FFFFh. The 1MB block of SM address space serves only to identify SM transactions and does not actually target any memory locations.

Figure 9-1: SM Request Sources
The System Management Controller (SMC) generates SM requests in response to both software initiated events (i.e., writes to registers within the south bridge) and hardware events (e.g., inactivity timeouts).

**Upstream Request Packet Format**

SMC-originated messages are delivered as posted Sized Write transactions, consisting of a SM request packet followed by a 4 byte data packet. An SM transaction is identified by both the Sized Write command and the assigned SM address range. The format of the upstream moving SM request packet is illustrated in Figure 9-2. The specification requires the following field values:

- **Byte 0** — `Cmd` defined as posted, sized, byte write = 101000d
- **Bytes 2 & 3** — `Count [3:0] = 0000b` (specifies 4 byte data packet)
- **Byte 4** — Defines types of upstream SM Request. (See “System Management Commands — Upstream” on page 219 for details.)
- **Bytes 5 & 6** — `Address[39:20] = FDF9h`

An interesting aspect of the upstream SM transactions is that the Host Bridge reflects them all back downstream across all links as broadcast SM messages.

---

**Figure 9-2: Format of SM Request Packet Issued by the System Management Controller**

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Bits</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></td>
<td>SeqD[3:2]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>SeqD[1:0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>Count[1:0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Count[3:2]</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SysMgmtCmd[7:0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Addr[23:20] = 1h</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Addr[31:24] = F9h</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Addr[39:32] = FDh</td>
</tr>
</tbody>
</table>
Chapter 9: System Management

System Management Commands — Upstream. The System Management Command field (SysMgtCmd) defines the type of SM request being issued. Table 9-1 on page 219 summarizes the SM command options for the upstream direction. All of the upstream messages represent state changes of various signals as indicated in Table 9-1. (Chapter 22, entitled "X86 CPU Compatibility," on page 491 details each signal type.)

Three distinct pieces of information may be included in the upstream SysMgtCmd:

- The Base Command Type — The upper nibble defines the primary command type (processor input signals and STPCLK) and is always present.
- Signal State Bit Map — Defines new state of the signal (1 = signal asserted, 0 = signal deasserted). For example, STPCLK may be asserted by the SMC (bit 0 of SysCmd = 1) to indicate a power management request. Subsequently, the SMC would need to deassert STPCLK by sending another message, with bit 0 of SysCmd = 0. In addition, when an SM message is sent to the Host Bridge, the bridge will send the packet back downstream using its SM request packet.
- System Management Action Field (SMAF) — In the upstream direction this field is only defined only for the STPCLK message. SMAF qualifies the nature of the power management event being signalled. Note that the definition of the SMAF is platform specific.

Table 9-1: Summary of Upstream SysMgtCmd Encodings

<table>
<thead>
<tr>
<th>SysMgtCmd 7:4 3:0</th>
<th>Command Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 xxxx</td>
<td>Reserved</td>
</tr>
<tr>
<td>0001 xsss</td>
<td>x86 legacy inputs to the processor. Bits [3:0], labeled “s,” are bit maps that correspond to each input and the new logic state of the specified signal. (1 = signal asserted and 0 = signal deasserted) Bit 0 = IGNNE state Bit 1 = A20M state Bits 2 and 3 = Reserved</td>
</tr>
</tbody>
</table>
10 Error Detection And Handling

The Previous Chapter

Rather than requiring HT devices to incorporate additional pins for signaling system-related items such as power management events, System Management messages are defined that permit signaling via transactions, thereby creating virtual wires. The previous chapter defined the mechanism used to send these messages, by detailing the System Management packets and protocol. Some System Management functions such as changing the operating frequency of an HT link require that transactions be stopped. This chapter also introduced the ability to temporarily disconnect the links to allow these types of state change and to save power.

This Chapter

As indicated in the chapter title, there are two aspects in dealing with link or internal errors in HyperTransport: detection and handling. This chapter describes the error types defined by the specification and what devices may do about them. Some devices may choose to detect and handle some errors and not others. The PCI configuration space registers used to program the error strategy and log errors are described here, as are the reporting mechanisms: error response, fatal and non-fatal interrupts, and Sync flood.

The Next Chapter

Reset signalling and timing along with actions taken by the system and devices during reset are the primary topics discussed in the next chapter. It also discusses the software initiated reset and why it’s required. The process of determining the default speed and link width and the subsequent software tuning of bus speed and link width are also detailed.
Introduction

HyperTransport defines six types of errors, and three basic ways they may be reported to the system.

Types Of Errors

The error types which may be detected, logged, and reported are:

1. CRC (Cycle Redundancy Code) Errors
2. Protocol Errors
3. Receive Buffer Overflow Errors
4. End Of Chain Errors
5. Chain Down Errors
6. Response Errors

Reporting Methods

Once an error is detected, it can be conveyed to other devices in the system in the following ways:

1. Error Responses
2. Error Interrupts (fatal and non-fatal)
3. Sync Flooding

The Role Of PCI Configuration Space

The PCI Configuration Space required of each HyperTransport device performs several roles in error handling. The Command and Status registers in the header and the Link Error and Error Handling registers in the HyperTransport Advanced Capability Register block are used to report error handling capabilities, program the error reporting mechanism to be used if an error occurs, and to log the errors which occur so that software can later assess the error events seen by each device.

Once the error capabilities of a device have been determined and the error reporting strategy is programmed in configuration space, any errors which occur will be handled accordingly. For example, a HyperTransport device
Chapter 10: Error Detection And Handling

which detects a protocol error may be programmed to set the corresponding log bit in the configuration space Error Handling register and generate a fatal interrupt message.

Most Types Of Error Checking Are Optional

To accommodate differences in how devices and applications may view certain types of errors, the specification only requires CRC generation/checking on each link; other aspects of error detection and handling are optional. If a particular error is not checked, the corresponding enable and logging bits in configuration space must be hardwired to 0.

System Handling Of HyperTransport Errors Varies

As in many other bus protocols, HyperTransport bus behavior during error events is well specified but the action taken by the system in response to reported errors is implementation specific. However, if Sync flood is used as a reporting mechanism, a reset is required on the affected chain(s) to restore proper protocol.

The Error Types

The following section summarizes the required CRC generation/checking as well as the optional protocol, receive buffer overflow, end of chain, chain down, and response error handling.

CRC Errors

The Cycle Redundancy Code (CRC) is used to detect transmission errors on all enabled byte lanes on each link. The 32 bit CRC value is calculated and sent at prescribed intervals by each transmitter, then checked against the CRC value calculated by the corresponding receiver as packets arrive. CRC is calculated by finding the remainder when the sum of packet data (CAD bits plus CTL signal during each bit time) is divided by the CRC polynomial. The polynomial used is:

\[ X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1 \]
CRC On 8, 16, or 32 bit Interfaces

For interfaces which are 8-, 16-, or 32-bits wide, CRC is independently generated and checked for each byte of CAD width. Figure 10-1 on page 232 illustrates CRC “stuffing” into the CAD packet stream on each 8-bit CAD interface.

![Figure 10-1: 8/16/32 Bit Interfaces: CRC Inserted Into CAD Stream Every 512 Bit Times](image)

**CRC Generation/Checking: 8/16/32 bit links**

(Refer to Figure 10-1 on page 232)

1. After link initialization, each transmitter begins sending packets (NOP, etc.). CRC calculation is based on “raw” CAD/CTL bit patterns on each CAD byte without regard to the packet types being sent.
2. 512 bit times after initialization, the first 32-bit CRC value has been calculated for each byte lane. The window for “stuffing” the 32-bit CRC value into its CAD stream is 64 bit times into the next “window”. Note: because of this delay, there is no CRC sent during the first window.
3. Although each window for CRC calculation is 512 bit times, in reality all windows (after the first one) are actually 516 bit times because CRC for each window is inserted into the following one for four additional bit times. Note that the CRC value stuffed into each window is not included in the subsequent CRC calculation for that window.
4. There is no special signalling associated with CRC transmission; both devices simply count the bit times starting with link initialization and “know” where the CRC payload falls in each window.
5. CRC is calculated and sent independently for each 8 bits of CAD width. The
Chapter 10: Error Detection And Handling

CTL signal itself is included in the CRC calculation for the lowest byte of CAD (bits 0-7). On a bus wider than 8 bits, the CTL signal is also factored into the CRC calculation for each of the upper CAD bytes, but is assumed to be 0 during all bit times.

6. During the driving of the CRC value itself, the CTL signal is driven = 1 (Control) by the transmitter. The CRC bits are inverted before being transmitted onto the link.

CRC Generation/Checking: 2/4 bit links

On links narrower than 8 bits, the CRC value is generated in the same way as for 8-bit links carrying the same value. It simply takes longer to move the packets and CRC value across the link — causing the calculation window and stuffing point for the CRC value to be stretched accordingly. The extra assertions of the CTL signal (after the first bit time in each byte) are not used by the transmitter or receiver in the CRC calculation.

4 Bit CAD Width. A CAD width of four bits requires twice as many bit times as an 8 bit bus for moving information across the link. Therefore:

- The CRC window size is 1024 bit times.
- The CRC stuffing point starts 128 bit times after the start of a window.
- It takes 8 bit times to transfer the 32-bit CRC value.

2 Bit CAD Width. A CAD width of two bits requires four times as many bit times as an eight bit bus for moving information across the link. Therefore:

- The CRC window size is 2048 bit times.
- The CRC stuffing point starts 256 bit times after the start of a window.
- It takes 16 bit times to transfer the 32-bit CRC value.

Logging CRC Errors

CRC errors impact both control and data information; if these errors occur on any CAD byte lane, the corresponding error bit(s) will be set in the HyperTransport Advanced Capability block Link Control CSR. The four bits (one for each byte lane) are illustrated in Figure 10-2 on page 234 below.
Routing Packets

The Previous Chapter

The previous chapter described HyperTransport flow control, used to throttle the movement of packets across each link interface. On a high-performance connection such as HyperTransport, efficient management of transaction flow is nearly as important as the raw bandwidth made possible by clock speed and data bus width. Topics covered here include background information on bus flow control and the initialization and use of the HyperTransport virtual channel flow control buffer mechanism defined for each transmitter-receiver pair.

This Chapter

This chapter describes the rules governing acceptance, forwarding, and rejection of packets seen by HyperTransport devices. Several factors come into play in routing, including the packet type, the direction it is moving, and the device type which sees it. A related topic also covered in this chapter is the fairness algorithm used by a tunnel device as it inserts its own packets into the traffic it forwards upstream on behalf of devices below it. The HyperTransport specification provides a fairness algorithm and a hardware method for tunnel management packet insertion.

The Next Chapter

The next chapter describes the ordering rules which apply to packets associated with the three types of HyperTransport I/O traffic: PIO, DMA, and Peer-to-Peer. Depending on the whether compatibility with the full producer-consumer ordering model used in PCI is required or relaxed ordering is permissible, attribute bits in request and response packets may be set or cleared. These bits are defined by the requester and are used by devices in the path to the target, and within the target, to enforce proper ordering. HyperTransport applies dedicated sets of ordering rules for upstream I/O traffic, downstream I/O traffic, and the special ordering required of host bridges and in double-hosted chains. Refer to Chapter 20, entitled "I/O Compatibility," on page 457 for a description of the additional ordering requirements when interfacing HyperTransport to other compatible protocols (e.g. PCI, PCI-X, and AGP).
Packet Routing: Shared Bus vs. Point-Point Topology

Routing information in a shared bus topology such as PCI or PCI-X is somewhat simpler than in a point-point topology such as HyperTransport.

Shared Bus Routing

Referring to the PCI/PCI-X shared bus example illustrated in Figure 11-1 on page 258, it should be clear that if a transaction appears on the shared bus, all devices “see it” and have an opportunity to decode the address and command and claim the cycle. Devices other than bridges have no responsibilities for routing information to their neighbors. Also note that arbitration on a shared bus is simple because a single arbiter can manage the entire bus. In PCI/PCI-X, the arbiter is typically in the bus Host Bridge; the arbiter considers requests from each master, then grants the bus to each in turn, hopefully applying a reasonable fairness algorithm.

Figure 11-1: Routing: Shared Bus vs. HyperTransport Point-Point

PCI/PCI-X System

- CPU
- System Memory
- Host Bridge
- Dev
- Dev
- Dev

HyperTransport System

- CPU
- System Memory
- Host Bridge
- Tunnel
- End

Each device is attached directly to the host bridge and “sees” all bus transactions.

Downstream devices depend on “forwarding” by tunnels for their attachment to host bridge.
Chapter 11: Routing Packets

HyperTransport Point-Point Routing

In contrast to the shared bus approach, the HyperTransport topology distributes responsibility for routing and forwarding packets among all devices, with the exception of single-link end (cave) devices. For example, the tunnel peripheral device in Figure 11-1 on page 258 must observe a set of rules governing acceptance, forwarding, and rejection of packets moving both upstream and downstream. The end device in Figure 11-1 on page 258 is dependent on the tunnel to do this. Note that a benefit of a point-point bus is the elimination of shared bus arbitration. Packet transfer is subject only to flow control on each link.

Review Of Packet Types And Formats

How a packet is routed depends in large part on the type of packet it is. Each packet in HyperTransport is a multiple of four bytes in size, and the specification divides packets into two types: control and data. All control packets contain a Command Type field in the first byte which identifies which type of control packet it is and the format of the remaining packet fields to follow. It also indicates whether data packets follow immediately (writes), will return later (reads), or are not required.

Control Packets

Control packets are sent across a link to initiate specific tasks; they contain information fields used for several purposes: address decoding, virtual channel and transaction stream management, error reporting, and routing. Devices perform routing functions by extracting information from key fields in control packets. Control packets are further divided into three groups: information, requests, and responses.

Information Packets: No Routing Required

Information packets include NOP and Sync/Error. These four-byte packets are used for communication between two ends of a link interface. When issued by a transmitter, they are always accepted by the corresponding receiver; they are never forwarded to another link. This means that are no routing issues associated with them. These two packet types will not be discussed further in this chapter.
Request Packet Routing Information

Request packets are used to initiate various transactions and control operations. Packet format depends on the request type; four byte request packets are sent when no address field is needed; eight byte requests are sent otherwise. Figure 11-2 on page 260 depicts a generic eight byte RdSized or WrSized request packet and the key fields used in request packet routing.

Figure 11-2: Generic WrSized Or RdSized Request Packet: Key Routing Fields

Table 11-1: Definitions Of Request Packet Fields Used In Routing

<table>
<thead>
<tr>
<th>Byte</th>
<th>Bit</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5:0</td>
<td><strong>Command Type Code.</strong> This code indicates the request type.</td>
</tr>
<tr>
<td>1</td>
<td>4:0</td>
<td><strong>UnitID.</strong> This is the UnitID (0-31d) of the requester</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td><strong>Compat.</strong> This bit is set by bridges in downstream request packets tar-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>geting the system subtractive decode device (e.g. compatibility bridge)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>residing on the system “compatibility chain.”</td>
</tr>
<tr>
<td>3</td>
<td>7:2</td>
<td><strong>Start Address[39:2]</strong> The dword-aligned, 40 bit target start address.</td>
</tr>
<tr>
<td>4-7</td>
<td>7:0</td>
<td>Refer to the HyperTransport address map for address use.</td>
</tr>
</tbody>
</table>
Six Request Types. HyperTransport supports six request command types. Most have a number of variants. The following table summarizes each request, the number of bytes in the packet, the Command Code (Byte 0, Bits 5:0 of the request packet), and notes about its use.

Table 11-2: Request Packet Command Code Summary

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Packet Size</th>
<th>CMD Code</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broadcast Message</td>
<td>4 Bytes</td>
<td>111010b</td>
<td>Originate at host bridges and travel downstream. All devices accept them and propagate them downstream onto all links. EOC device accepts message and drops it.</td>
</tr>
<tr>
<td>Sized Read (RdSized)</td>
<td>8 Bytes</td>
<td>01xxxxb</td>
<td>May be issued by any device. Read response is returned. Use of “xxxx” option bits: [3] = response may pass (if 1) [2] = dword/byte (1 = dword) [1] = Isoc channel (1 = Isoc) [0] = Coherency (1 = Req’d)</td>
</tr>
<tr>
<td>Sized Write (WrSized)</td>
<td>8 Bytes</td>
<td>x01xxxb</td>
<td>May be issued by any device. Use of “xxxx” option bits: [5] = posted req (1 = posted) [2] = dword/byte (1 = dword) [1] = Isoc channel (1 = Isoc) [0] = Coherency (1 = Req’d)</td>
</tr>
<tr>
<td>Flush</td>
<td>4 Bytes</td>
<td>000010b</td>
<td>Forces all preceding posted writes in same transaction stream to host bridge.</td>
</tr>
<tr>
<td>Fence</td>
<td>4 Bytes</td>
<td>111100b</td>
<td>Barrier to subsequent posted writes from all streams (except Isoc) until all previous posted writes complete.</td>
</tr>
<tr>
<td>Atomic Read-Modify-Write</td>
<td>8 Bytes</td>
<td>111101b</td>
<td>Hybrid read and write command for modifying a memory location atomically.</td>
</tr>
</tbody>
</table>
The Previous Chapter

There are two aspects in dealing with link or internal errors in HyperTransport: detection and handling. The previous chapter described all error types and discussed what devices may do about them. Some devices may choose to detect and handle some errors but not others. The PCI configuration space registers used to program the error strategy and log errors were described, as well as the reporting mechanisms: error response, fatal and non-fatal interrupts, and Sync flood.

This Chapter

Reset signalling and timing along with actions taken by the system and devices during reset are the primary topics discussed in the chapter. This chapter also discusses the software initiated reset and why it’s required. The process of determining the default speed and link width and the subsequent software tuning of bus speed and link width are also detailed.

The Next Chapter

HyperTransport uses PCI configuration. The next chapter describes HyperTransport configuration for devices other than HyperTransport-to-HyperTransport bridges: host bridges, tunnels, and end (I/O hub) devices. HyperTransport-to-HyperTransport bridges have a different PCI header format than these devices, and are described separately in the chapter on HT Bridges. Many attributes of HyperTransport configuration are exactly the same as for generic PCI devices, although some PCI configuration space header fields are used differently in HyperTransport, and some not at all. Devices require at least one HyperTransport-specific advanced capability register block in addition to the basic PCI configuration space header fields.
HyperTransport System Architecture

General

HyperTransport defines cold reset, warm reset, and a link initialization process. Link initialization involves determining the widest link and highest clock frequency that two devices attached to a link may use. Link initialization involves a multi-step process that is initiated during cold reset.

A hardware-based handshake process determines the link width to be used immediately following cold reset. This process is performed by devices at both ends of every link. The handshake determines the smaller of the devices attached to each link with respect to maximum receiver width (up to 8 bits). For example, one receiver may have a maximum width of 4 and the other a maximum width of 8, thus the smaller CAD width (4) is used. A detailed description of this process is described in the section entitled, “Low-Level Link Width Initialization” on page 282. Also, following cold reset all devices use the default clock frequency of 200 MHz (required by all devices). The initial width negotiated and the default speed may be less than the actual capability of the devices and thus require further tuning.

Next, firmware (e.g. BIOS) optimizes all links so they use the widest path and the fastest clock that the attached devices support. This is done by reading link capability registers within the devices attached to each link. These registers report the maximum link width and maximum frequency at which each device is designed to operate. Software determines the highest common actual CAD width and clock frequency to be used.

Finally, firmware initiates a Warm RESET (or assertion of LDTSTOP#) to cause the new link width and clock values to take effect. Warm RESET or LDTSTOP# assertion is typically triggered by writing to an implementation-specific register in the I/O Controller Hub.

Details regarding the process described above are discussed in the remainder of this chapter. Note however, that portions of the RESET initialization process that involve clock and buffer initialization are detailed in the section called “Link Initialization” on page 282.

Cold Reset

Cold Reset is signaled during the power-up sequence under hardware control. This section details the sources, effects, and characteristics of a HyperTransport cold reset.
Chapter 12: Reset & Initialization

Sources of Cold Reset

In addition to the hardware generation of cold reset during the powerup sequence, platform developers may also provide hooks for generating cold reset under software control. An optional method of generating a HyperTransport cold reset is defined by the specification for the secondary bus of a HT-to-HT bridge (discussed on page 278). However, software generation of cold reset for the secondary side of the Host-to-HT bridge can be implementation specific.

Resetting the Primary HT Bus

Some implementation-specific mechanism must be defined to initiate a cold reset at powerup. The HT specification does not precisely define the source of HT cold reset for the system. It may be generated by system board logic or could be incorporated into the Host to HT bridge or other HT device residing on the system board. Figure 12-1 illustrates an example of HT RESET# generation and distribution.

Figure 12-1: Example of Reset Distribution in an HT System
Further, the specification does not require a software controlled method of cold reset generation. However, a host bridge could optionally implement a mechanism similar to that provided by the bridge control register of an HT-to-HT bridge. (See next section.)

Once reset is signalled, any HT device has the option of extending it (via open drain signaling) to ensure the amount of time it needs to complete its internal initialization. In this way, reset remains asserted until the last HT device in the chain completes its initialization. All HT devices that signal cold reset must correctly sequence RESET# and PWROK as discussed in “Signalling and Detecting Cold Reset” on page 280.

**Resetting Secondary Side of HT-to-HT Bridge**

An HT Bridge is required to propagate cold reset from its primary to its secondary side, but is not allowed to propagate any form of reset from its secondary to primary side. Thus, when the HT-to-HT Bridge initiates an HT cold reset to its secondary side, it will be distributed to all devices in the downstream chain as depicted in Figure 12-2.

HT defines a optional method for HT-to-HT Bridges to generate a cold reset on the secondary bus under software control. This is done via two HT-to-HT Bridge configuration registers, the *bridge control register* in the configuration header and the *command register* located in the HT capability registers. These registers are depicted in Figure 12-3 on page 280. Each of these registers has a bit that in combination permits the generation of cold reset as follows:

- HT Command Register, Bit 0 (optional) — selects cold reset when cleared (0)
- Bridge Control Register, Bit 6 — forces a secondary bus reset when set (1)

When a cold reset is selected, the bridge will deassert PWROK as part of the reset sequence, thereby causing a cold reset. It is the responsibility of hardware to sequence PWROK and RESET# correctly, as described in “Signalling and Detecting Cold Reset” on page 280.
Chapter 12: Reset & Initialization

Figure 12-2: Example HT-to-HT Bridge Forwarding Cold Reset
Device Configuration

The Previous Chapter

Reset signalling and timing, along with actions taken by the system and devices during reset, are the primary topics discussed in the previous chapter. The chapter also discussed software initiated reset and why it’s required. The process of determining the default speed and link width and the subsequent software tuning of bus speed and link width are also detailed.

This Chapter

HyperTransport uses PCI configuration. This chapter describes HyperTransport technology configuration for host bridges, tunnels, and end (cave) devices. These devices use the type 0 configuration header format, while HyperTransport-to-HyperTransport bridges and bridges between HyperTransport and other PCI compatible protocols (e.g. PCI and PCI-X) use the type 1 header format and are described separately in Chapter 16, entitled ”HyperTransport Bridges,” on page 407. Many aspects of HyperTransport device configuration are exactly the same as for generic PCI devices, although some header fields are used differently in HyperTransport, and some not at all. Devices also require at least one HyperTransport-specific advanced capability register block in addition to the basic PCI configuration space header fields.

The Next Chapter

The high speed signaling performed by HT devices is based on point-to-point differential signaling and source synchronous clocking. Details associated with link power requirements and the driver and receiver characteristics are discussed in the next chapter. Also, the characteristics of the system-related signals, including RESET#, PWROK, LDTSTOP#, and LDTREQ# are discussed.
HyperTransport Uses PCI Configuration

Many current generation computers use the PCI configuration method and the 256 byte PCI configuration space memory required of all PCI-compliant devices to help set up and manage system chipsets and I/O peripherals. Using PCI configuration for a bus protocol such as HyperTransport goes a long way toward promoting software compatibility with the millions of systems already supporting buses employing PCI-based configuration, including PCI, AGP, PCI-X, USB, etc. HyperTransport is designed for PCI plug-and-play configuration and to minimize impact on existing BIOS and driver software.

What PCI Configuration Accomplishes

During system initialization, low level BIOS or other system software uses configuration transaction cycles to “walk” each PCI-compatible bus (PCI, PCI-X, HyperTransport, AGP, etc.) and read the PCI configuration space of each device function it finds. Once discovered, basic and advanced capability features of each device are set up as appropriate. Collectively, PCI configuration cycles may be used for many aspects of device management, including:

- **Assignment of system resources.** Unlike earlier bus protocols, including the Industry Standard Architecture (ISA), PCI compatible plug-and-play devices are not allowed to establish their own base addresses and interrupt levels using fixed schemes or through user manipulation of jumpers and switches. Instead, the designer of a PCI compatible device “hard codes” information in selected PCI Configuration Space fields describing the fixed requirements of the device with respect to memory and I/O addresses needed, whether system interrupt support is required, arbitration needs, etc. Once the system address maps and interrupt routing are determined, software then returns to programmable fields in the PCI Configuration Space of each device and programs address ranges, interrupt routing, etc.

- **Enabling of device capabilities and options.** In addition to assignment of system resources to PCI compatible devices, software also uses the PCI Configuration Space to select device options, enable bus mastering and target decoding of memory and I/O transactions, program error response strategy, and set up other basic PCI and advanced capability protocol features.

- **Checking of dynamic (error) status.** Finally, the PCI configuration space is used to log errors resulting from attempted transactions. These logged errors, if checked by software, provide a picture of the nature of the error,
which device(s) detected it, etc. The Status register in the configuration space header is used for generic PCI-type error logging; in addition, advanced capability register blocks also contain logging fields for errors related to a specific capability (e.g. HyperTransport CRC errors, buffer overflow errors, etc.).

HyperTransport System Limits

HyperTransport shares PCI terminology in describing a system in terms of the number of buses, devices, functions, and configuration space.

256 Buses In A System

PCI permits 256 buses in a system and each PCI host bridge or PCI-to-PCI bridge secondary interface is host to a new bus with a unique bus number. Unlike PCI, a HyperTransport bus may not end with a single electrical connection. Tunnel devices enable the construction of device chains which are still viewed as a single logical bus. The 256 bus limit in HyperTransport, then, is actually 256 chains.

32 UnitIDs Per Bus

PCI permits a maximum of 32 physical devices per bus. In HyperTransport, each functional device can request multiple device numbers, called UnitIDs. The reason for this is because HyperTransport ordering rules consider the transactions from each UnitID to be a unique transaction stream; owning multiple UnitIDs enables a device to source more than one transaction stream (e.g. a standard transaction stream and an isochronous transaction stream for its high priority traffic). The 32 device per bus limit in PCI is a 32 UnitID per bus limit in HyperTransport.

One To Eight Functions Per Device

As in PCI, HyperTransport allows 1-8 logical functions in a physical device package. Each function has its own 256 byte configuration space, and will be assigned unique UnitID(s).
256 Bytes Of Configuration Space

Just as in other PCI devices, each function of a HyperTransport device must implement a 256 byte configuration space memory. The first one-fourth of the configuration space is the header. In addition to the header, devices also must implement at least one set of HyperTransport advanced capability registers.

Configuration Accesses: Reaching All Devices

The process of HyperTransport device configuration depends on software being able to access the 256 byte configuration space of each function in each device on each bus in the system. Configuration cycles originate at the CPU that executes the configuration software; the cycles then move in the direction of the target. This section compares the PCI and HyperTransport methods used to reach the configuration space of a device which may reside on a bus many levels deep in the topology.

Implied in plug-and-play address assignment on buses such as PCI and HyperTransport is the fact that until it is discovered and assigned an address range by low-level software, a device can’t claim normal memory or I/O transactions. Furthermore, whenever a bus reset occurs, each device “forgets” its address ranges and other information programmed in configuration space and can no longer be targeted with transactions which depend on assigned addresses. So, how can a device’s configuration space be set up if it doesn’t know its target address?

In addition to the problem of simple devices recognizing their own configuration cycles in an uninitialized system, the complex topologies permitted in PCI, PCI-X, and HyperTransport require that bridges be programmed to forward configuration transactions to the proper bus before a device can even consider claiming it.

Before looking at how HyperTransport differs from PCI in its handling of system-wide configuration accesses, here is a quick review of how PCI handles them.
Review: How PCI Handles Configuration Accesses

With the exception of chipsets, PCI devices generally power up (or come out of reset) disabled with respect to either generating transactions as bus master or decoding memory or I/O transactions as targets. This is because they are not aware of either their own plug-and-play addresses or those of other devices. The Configuration Read and Configuration Write transactions are the only ones a PCI device may decode following reset. Configuration cycles originate at the CPU, and instead of carrying conventional address information (which would be useless), these cycles start downstream carrying the following attributes about the target in the 32-bit address of the configuration read or write transaction:

- Bus number the target resides on (0-255 decimal)
- Device number of the target (0-31 decimal)
- Function number inside the target (0-7 decimal)
- Double Word Offset in target’s configuration space (0-63 decimal)

Note that while addresses are not known after reset, bus number and device number are functions of the board layout and ARE known.

Two Configuration Cycle Types

As PCI configuration cycles travel downstream, there are two variants: type 0 and type 1. The type is indicated in the lowest two bits of the 32-bit PCI address. Having two types is necessary because PCI devices don’t know their bus number or device numbers and must depend on upstream bridges to help select them.

Type 1 Cycle Until Target Bus Is Reached

Starting at the host bridge, a type 1 configuration cycle is propagated downstream until it reaches the bridge with a secondary bus number equal to that of the configuration cycle bus number field. Type 1 configuration cycles are ignored by all devices except bridges which will claim them and pass them on to the next downstream bus if the bus number field of the configuration cycle is between the values programmed in the bridge’s secondary and subordinate bus number registers.
14 Electrical

The Previous Chapter

HyperTransport uses PCI configuration. The previous chapter described Hyper-Transport technology configuration for host bridges, tunnels, and end (cave) devices. These devices use the type 0 configuration header format, while Hyper-Transport-to-HyperTransport bridges and bridges between HyperTransport and other PCI compatible protocols (e.g., PCI and PCI-X) use the type 1 header format and are described separately in Chapter 16, entitled “HyperTransport Bridges,” on page 407. Many aspects of HyperTransport device configuration are exactly the same as for generic PCI devices, although some header fields are used differently in HyperTransport, and some not at all. Devices also require at least one HyperTransport-specific advanced capability register block in addition to the basic PCI configuration space header fields.

This Chapter

The high speed signaling performed by HT devices is based on point-to-point differential signaling and source synchronous clocking. Details associated with link power requirements and the driver and receiver characteristics are discussed in this chapter. Also, the characteristics of the system-related signals, including RESET#, PWROK, LDTSTOP#, and LDTRQ# are discussed.

The Next Chapter

The next chapter focuses on the source synchronous clocking environment within HT. This involves the use of the source synchronous transmit clock to load data into a receive FIFO and the transfer of data into the receiver time domain with a receive clock that unloads data from the FIFO. Additionally, the specification defines three clocking modes that require different levels of support for passing packets between these two clock domains.
HyperTransport System Architecture

Background and Introduction

First, a brief review of the essential elements of the high-speed link is provided including an introduction to the primary aspects of the electrical signaling environment.

Each link consists of two sets of uni-directional signals (see Figure 14-1) that support concurrent data transfers in each direction. HT achieves high performance by transferring data at a maximum clock frequency of 800MHz, coupled with the use source synchronous double data rate (DDR) clocking techniques. DDR clocking permits data transfer on both the rising and falling edges of each clock. HT also relies on low voltage swing differential signaling with on-die differential termination to facilitate the high-speed data rates and to improve noise immunity.

Figure 14-1: Link Signals

\[
V_{ref} = 1.2 \text{ Volts}
\]

HyperTransport Device A  
HyperTransport Device B

- CLK[m:0] pair
- CTL pair
- CAD[n:0] pairs
- PWROK
- RESET#
- LDTSTOP#
- LDTREQ#

System Logic

\[n = 1,3,7,15 \text{ or } 31\]
\[m = 0,0,0,1 \text{ or } 3 \text{ respectively}\]
Chapter 14: Electrical

The following list reviews each of the differential high speed signals that is used when transferring data across the link.

- **CAD (Command, Address and Data)** — Carries HyperTransport requests, responses, data packets, and other information across the link. CAD can be different widths in each direction depending on performance needs.
- **CTL (Control)** — When asserted, CTL indicates that the CAD signals are carrying a control packet. When deasserted, CTL indicates that the CAD signals are carrying a data packet. There is one CTL signal for each data direction.
- **CLK (Clock)** — This is the source synchronous clock used when transmitting CAD and CTL signals. Each byte of CAD has its own clock. Note that the CTL signal is clocked by the same clock used for CAD[7:0].

Because the link width is scalable, the number of source clocks used also varies. Table 14-1 on page 365 below lists the transmit signals that share the same source synchronous clock.

<table>
<thead>
<tr>
<th>Signal Group</th>
<th>Source Synchronous Clock</th>
</tr>
</thead>
<tbody>
<tr>
<td>CADOUT [7:0], CTLOUT</td>
<td>CLKOUT (0)</td>
</tr>
<tr>
<td>CADOUT [15:8]</td>
<td>CLKOUT (1)</td>
</tr>
<tr>
<td>CADOUT [23:16]</td>
<td>CLKOUT (2)</td>
</tr>
<tr>
<td>CADOUT [31:24]</td>
<td>CLKOUT (3)</td>
</tr>
</tbody>
</table>

The four system-related signals associated with each link are implemented as single-ended LVCMOS signals and as open drain wired-OR outputs to allow multiple sources to drive them. These signals include:

- **PWROK (Power OK)** — Driven by system logic, this signal is a required input to each device. It may also be driven by HT devices in conjunction with RESET# to extend the reset time needed for their internal initialization.
- **RESET#** — This signal is driven by system logic and is a required input to each HyperTransport device. It may also be driven by HT devices in conjunction with PWROK# to extend the reset time needed for their internal initialization.
- **LDTSTOP# (Lighting Data Transfer Stop)** — Supports power management and other features that require a change of state on the links (e.g., it disables the link during power state transitions).
HyperTransport System Architecture

- LDTREQ# (Lightning Data Transfer Request) — An output from HT devices that permits a device to request the links be re-enabled for normal operation.

The following sections describe the power requirements, electrical, and timing characteristics of both the differential and single-ended signals, as well as an overview of the testing environment.

Power Requirements

The HT specification defines the link supply voltage requirements and the power consumption allowed by HT transmitters and receivers.

Power Supply Voltage

Single fixed power supply provides power to all the transmitter and receiver circuits. The HT link supply voltage (VLDT, or Voltage Lightning Data Transport) is rated at:

1.2 volts ± 5%

It is possible to have tight tolerance on VLDT (+-5%) on the power supply voltage output because most HyperTransport signals are differential with minimal current transients on simultaneous switching signals. The tight tolerance requirement on the supply voltage ensures that less power supply feedback noise will propagate through the system.

Differential Pair Power Consumption

The power consumption associated with the transmitter and receiver of each differential signaling pair is defined by the specification. Table 14-2 on page 367 lists these parameters. Note that the values are based on specified values of $R_{ON}$ and $R_{TT}$ as described in the next section.
Differential Signaling Characteristics

This section describes the DC and AC characteristic of the differential transmitters and receivers. This includes impedance and voltage levels required for compliant operation of the differential transmitters and receivers.

Differential DC Characteristics

Figure 14-2 illustrates a differential link connection between a transmitter and receiver. The specification defines the impedance and subsequent voltages that will be developed under steady-state conditions (i.e., when transmitter is driving a differential 0 or 1).

Table 14-2: Differential Pair Power Consumption

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Min (mW)</th>
<th>Typical (mW)</th>
<th>Max (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC power per output signal pair ($P_{DC}$)</td>
<td>5.9</td>
<td>7.2</td>
<td>9.0</td>
</tr>
<tr>
<td>AC power per differential pair, transmitter ($P_{TAC}$)</td>
<td></td>
<td></td>
<td>53.0</td>
</tr>
<tr>
<td>AC power per differential pair, receiver ($P_{RAC}$)</td>
<td></td>
<td></td>
<td>13.0</td>
</tr>
<tr>
<td>AC power per differential signal pair, total ($P_{AC}$)</td>
<td></td>
<td></td>
<td>66.0</td>
</tr>
</tbody>
</table>
The high speed signaling performed by HT devices is based on point-to-point differential signaling and source synchronous clocking. Details associated with link power requirements and the driver and receiver characteristics are discussed in this chapter. Also, the characteristics of the system-related signals, including RESET#, PWROK, LDTSTOP#, and LDTREQ# are discussed.

This chapter focuses on the source synchronous clocking environment within HT. This involves the use of the source synchronous transmit clock to load data into a receive FIFO and the transfer of data into the receiver time domain with a receive clock that unloads data from the FIFO. Additionally, the specification defines three clocking modes that require different levels of support for passing packets between these two clock domains.

The next chapter describes the configuration of devices which use the Hyper-Transport technology type 1 configuration header for bridges. Such devices include HyperTransport-to-HyperTransport bridges and bridges to other PCI compatible protocols (e.g. HyperTransport-to-PCI or PCI-X). In this chapter, the basic architecture of a HyperTransport-to-HyperTransport bridge is reviewed and the configuration header fields are described. Differences in usage of bit fields by HyperTransport bridge interfaces vs. PCI bridge interfaces are emphasized. The format of PCI compatible bridge headers is formally defined in the PCI-to-PCI Bridge Architecture Specification, Revision 1.1.
Introduction

The point-to-point high-speed transmission of data from transmitter to receiver across a HT link relies on source synchronous clocking. The specification identifies three modes of clocking:

- Synchronous
- Pseudo-Synchronous
- Asynchronous

The 1.04 version of the specification states that: “Only the synchronous clocking mode is fully specified in this revision of this specification.” The specification further states that the other modes will be completely specified in later versions of the specification. All HT compliant devices are required to support the synchronous clocking mode. This introduction defines all three modes, and later sections in this chapter detail each mode. As you may have guessed, the section on the synchronous mode contains considerably more detail than the other modes.

Clock Initialization

The receive FIFO in each device must be able to absorb timing differences between the transmit and receive clocks. Data is written into the FIFO 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 (Tx Clock Out) and the receive clock domain (Rx Clock). 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.

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 write (load) pointer. The same deassertion CTL and CAD signals is read from the FIFO synchronous to the receive clock domain and used to initialize the read (unload) pointer. The separation between the write and read pointers is calculated based on worst-case variation between the transmit and receive clocks.
Chapter 15: Clocking

Note also that CTL cannot be used to initialize the pointers for byte lanes other than 0 in a multi-byte link, because CTL only exists within the byte 0 transmit clock domain.

Synchronous Clock Mode

The specification requires that all HT devices support the synchronous clock mode. This mode is the least complicated method of transferring data from transmitter to receiver. Synchronous clock mode requires that the transmit clock and receive clock have the same source, and operate at the same frequency. If we were to assume that the transmit clock and the receive clock always remained synchronized, then a simple clocking interface could be used as described in the following example.

A Conceptual Example

In this synchronous example, the transmit clock (Tx Clock) and receive clock (Rx Clock) are presumed to be in synchronization. Note, however, that source synchronous clocking requires that Transmit Clock Out (Tx Clk Out) be 90° phase shifted from Tx Clock. In this example all other sources of transmit to receive clock variation are ignored, including the expected clock drift associated with PLLs.

Refer to Figure 15-1 on page 390 during the following discussion. (Note that only one link direction is illustrated.) The transmitter delivers data synchronously across the link using the transmit clock. Tx Clock Out is sourced later and lags the data by 90° (or one-half bit time), thereby centering the clock edge in the middle of the valid data interval. When the data arrives at the receiver it is clocked into the FIFO using Tx Clock Out. Note that the clocked FIFO has two entries, which provides a separation of 1 between Tx Clock Out and Rx Clock. Data written into the FIFO during clock 1 would not be read from the FIFO using Rx Clock until clock 2. This one entry separation (called write-to-read separation) permits time for the sample to be stored prior to being read (i.e. the FIFO entry is not being written to and read from in the same clock cycle). In short, two FIFO entries are sufficient to provide the separation needed to ensure that data is safely stored and transferred into the receive clock domain.
However, in the real world many factors contribute to timing differences between the transmit and receive clock that are potentially significant, even though the clocks originate from the same source. These real world perturbations result in somewhat more complicated implementations that must account for and manage the worst case variation between the transmit and receive clocks. Specifically, the specification describes the receive FIFO implementation for handling the variation between the transmit and receive clocks.

Sources of Transmit and Receive Clock Variance

The specification defines and details the sources of transmit and receive clock variation that can exist. These clock differences can create FIFO overflow or underflow if not identified and taken into account. The clock differences can be attributed to two different categories or sources:

- **Invariant sources** — components that represent a constant phase shift between the transmit and receive clock domain.
- **Variant sources** — dynamic variations in the transmit and receive time domain (these phase variations can occur even though both transmit and receive clock are running at the same frequency).
Chapter 15: Clocking

The sources of clock variation in some cases can accumulate over time, causing clock variation to increase over time. However, all of the sources of clock variation are naturally limited in terms of the maximum amount of change that can occur. For example, a PLL is designed to produce an output clock that is synchronized with the input source clock, but with certain limitations. That is, variation of output frequency is specified not to change beyond a certain phase shift. The time over which the clock phase may change can be relatively short or perhaps much longer depending upon conditions. The consideration and assessment of the sources of clock variance is done to determine a FIFO size that can absorb the worst-case clock variation. This would occur if all sources of clock variation simultaneously reach their extremes, a very unlikely circumstance.

This chapter discusses the variant and invariant sources of transmit clock to receive clock variance. It also provides an example timing budget for each source.

**Invariant Sources**

The time-invariant factors contribute a small proportion of the overall clock variance. The invariant factors include:

1. Cross-byte skew in multi-byte link implementations
2. Sampling Error

**Cross-byte skew in multi-byte link implementations.** Differences in the arrival of Tx Clock Out at the receiver (CLKin) between each byte lane is caused by path length mismatch. This constant skew is termed $T_{bytelane\text{const}}$ in the specification. The specification allows up to 1000ps for this skew. Consequently, when multiple bytes are clocked into the FIFO the maximum skew could result in one of the bytes being clocked into the FIFO 1000ps later than the associated bytes. Thus, when the associated bytes are clocked out of the FIFO by Rx Clock, one byte having arrived late may be left behind. This problem is solved by adding additional entries in the FIFOs to handle the maximum lane-to-lane skew, ensuring that all associated bytes are clocked out at the same time. Note that lane-to-lane skew may change due to the effects of temperature, voltage change, etc. This parameter called $T_{bytelane\text{var}}$ is included in the variant source list.
The Previous Chapter

The previous chapter focused on the source synchronous clocking environment within HT. This involves the use of the source synchronous transmit clock to load data into a receive FIFO and the transfer of data into the receiver time domain with a receive clock that unloads data from the FIFO. Additionally, the specification defines three clocking modes that require different levels of support for passing packets between these two clock domains.

This Chapter

This chapter describes the configuration of devices which use the HyperTransport technology type 1 configuration header for bridges. Such devices include HyperTransport-to-HyperTransport bridges and bridges to other PCI compatible protocols (e.g. HyperTransport-to-PCI or PCI-X). In this chapter, the basic architecture of a HyperTransport-to-HyperTransport bridge is reviewed and the configuration header fields are described. Differences in usage of bit fields by HyperTransport bridge interfaces vs. PCI bridge interfaces are emphasized. The format of PCI compatible bridge headers is formally defined in the PCI-to-PCI Bridge Architecture Specification, Revision 1.1.

The Next Chapter

The next chapter describes the features of the optional HyperTransport double-hosted chain topologies. Topics include the reasons behind sharing and non-sharing chains, PCI configuration space registers used to initialize the fabric for multiple hosts, and tunnel support for upstream and downstream packets moving in both directions.
HyperTransport System Architecture

HyperTransport Bridges Uses PCI Configuration

HyperTransport bridges use the PCI configuration method and the 256 byte configuration space to set up and manage one primary bus and one or more secondary buses. The primary and secondary interfaces of a HyperTransport bridge may both be HyperTransport, or either one could bridge to another PCI-compatible protocol such as PCI or PCI-X. Because the configuration header contains bits to manage two different interfaces, there are two things to keep in mind when describing bridges between HyperTransport and another protocol:

1. For the HyperTransport interface, the meaning of some bits in the configuration header depends upon the particular interface (primary or secondary) that is implemented as HyperTransport.
2. For the interface that is not HyperTransport, bit field definitions revert back to the bus protocol being supported (e.g. PCI or PCI-X).

Basic Jobs Of A HyperTransport Bridge

As in the case of PCI bridges, a HyperTransport bridge has a number of responsibilities:

1. It extends the topology through the addition of one or more secondary buses. Each HyperTransport chain (bus) can support up to 32 UnitIDs. Because a device is permitted to consume multiple UnitID’s, implementing a bridge is a reasonable way to add a new chain that can support 32 additional UnitIDs (the bridge secondary interface consumes at least one of the new UnitIDs).
2. It acts as host for each of its secondary chains. There are many aspects to this, including ordering responsibilities, error handling, maintaining a queue for outstanding transactions routed to other buses, reflecting peer-to-peer transactions originating below it, decoding memory addresses so it may claim and forward transactions moving between the primary and secondary bus, forwarding/converting configuration cycles based on target bus number, etc.
3. In cases where it bridges between HyperTransport and PCI/PCI-X, the bridge also must translate protocols for transactions going in either direction. It may also have to remap address ranges between the 40-bit HyperTransport address range and the 32/64-bit PCI or PCI-X range.
How Does The Bridge Manage It All?

HyperTransport bridges make use of the same HyperTransport Host/Primary and Slave/Secondary advanced capability blocks already defined for non-bridge devices. Non-bridge HyperTransport device configuration is described in Chapter 13, entitled "Device Configuration," on page 305. Second, they implement the type 1 configuration space header common to all PCI-compatible bridges (with the redefinition of certain bits described shortly). This is fortunate because:

1. HyperTransport bridges maintain software compatibility with PCI bridges.
2. Bridge headers already contain two sets of control/status registers for managing two independent interfaces, making support for mixed HyperTransport and PCI/PCI-X bridges a reasonable extension. Each set of primary/secondary bus registers is programmed (and interpreted) according to whether the specific interface is HyperTransport, PCI, or PCI-X.

Same Slave/Primary And Host/Secondary Blocks

The Slave/Primary and Host/Secondary capability blocks used in non-bridge devices such as tunnels and end (cave) devices are also used for the HyperTransport interfaces of bridges. Figure 16-1 on page 410 depicts a simple HyperTransport-to-HyperTransport bridge with a single secondary chain. Note that a bridge must implement a separate Host/Secondary interface block for each secondary bus it supports.

HyperTransport Bridge Header Fields

In this section, the configuration space type 1 header format for HyperTransport bridge devices is described. For the most part, HyperTransport bridges use these fields in the same way as PCI bridges; the differences are described here. Header fields not mentioned are used in the same way as in PCI bridges. Refer to “The Type 0 Header Format” on page 322 for a description of HyperTransport non-bridge type 0 headers.
Bridge Header Command Register

Lower 16 bits at dword 01. The bridge header Command register is used by software to enable basic capabilities of the bridge on its primary bus, including bus mastering, target address decoding, error responses, etc. Bits marked “0” in Figure 16-2 are not used (hardwired = 0); refer to Table 16-1 for bit definitions.
Figure 16-2: HyperTransport Bridge Header Command Register

Table 16-1: HyperTransport Bridge Header Command Register Bit Fields

<table>
<thead>
<tr>
<th>Bit</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>I/O Space. When this bit is set to a one, the primary bus interface of the bridge may act as a target of requests in the I/O portion of the HyperTransport memory map. If this bridge is also a subtractive decoder, the setting of this bit does not affect the device’s ability to claim requests withCompat bit set. Warm Reset to 0.</td>
</tr>
</tbody>
</table>
17 Double-Hosted Chains

The Previous Chapter

The previous chapter described the configuration of devices that use the Hyper-Transport technology type 1 configuration header for bridges. Such devices include HyperTransport-to-HyperTransport bridges and bridges to other PCI compatible protocols (e.g. HyperTransport-to-PCI or PCI-X). The basic architecture of a HyperTransport-to-HyperTransport bridge is reviewed and the configuration header fields are described. Differences in usage of bit fields by HyperTransport bridge interfaces vs. PCI bridge interfaces are emphasized. The format of PCI compatible bridge headers is formally defined in the PCI-to-PCI Bridge Architecture Specification, Revision 1.1.

This Chapter

This chapter describes the features of the optional HyperTransport double-hosted chain topologies. Topics include the reasons behind sharing and non-sharing chains, PCI configuration space registers used to initialize the fabric for multiple hosts, and tunnel support for upstream and downstream packets moving in both directions.

The Next Chapter

HT provides a variety of mechanisms to support power management. These mechanisms include LDTSTOP#, LDTREQ#, STPCLK messages, and STOP_GRANT messages. While these mechanism are optional for HT devices, the specification requires this support for x86-based platforms. Note also that functions other than power management may make use of these signals and messages. This chapter discusses the strategy employed by HT for implementing power management and how a given platform can use these mechanisms to support power management.
HyperTransport System Architecture

Introduction

A HyperTransport chain consists of a host bridge at one end and some collection of devices connected to it in a daisy-chain arrangement. At the end of the chain, there is a device with a single-link connection. This could either be an end (I/O hub) device, or a multi-link device (e.g. tunnel) which has its downstream link disabled.

By contrast, a double-hosted chain has a host bridge at either end and some collection of multi-link devices between them. Figure 17-1 on page 428 illustrates a double-hosted chain. Note that there are no end (I/O hub) devices in a double-hosted chain.

Reasons For Implementing A Double-Hosted Chain

A double hosted-chain can be useful in fault-tolerant applications where a backup host interface takes over in the event of a failure of the primary interface. It also permits the sharing of a single set of resources and inter-processor communications by two CPUs in a clustering arrangement. Note: the HyperTransport I/O Link Specification Network Extensions allow extending the multiple-host concept to broader topologies using switch and router components. Refer to Chapter 19, entitled "Networking Extensions Overview," on page 443.
Chapter 17: Double-Hosted Chains

PCI Configuration Plays Key Role In Chain Setup

PCI configuration cycles are used to program bridges, tunnels, and end devices in each HyperTransport all chain. Two key registers used in setting up double-hosted chain (DHC) parameters are the HyperTransport Host Command CSR for host bridges and the HyperTransport Slave Command CSR for interior devices such as tunnels. These two registers and the key fields pertaining to double-hosted chains are described below. Refer to Chapter 13, entitled “Device Configuration,” on page 305 for a more complete description of PCI device configuration.

Slave Command CSR. Figure 17-2 and Table 17-1 on page 429 show the format of the Slave Command Register used by all non-host interfaces; key fields used in double-hosted chain configuration are highlighted.

Figure 17-2: Slave Command CSR: Key Fields In DHC Configuration

Table 17-1: Slave Command CSR: Definitions Of Key Fields In DHC Configuration

<table>
<thead>
<tr>
<th>Bit</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Master Host. This read-write bit is set by hardware automatically to indicate which link is attached to the host. Any write to the Command register will cause this bit to be set to indicate which link the write arrived on. Warm Reset = 0.</td>
</tr>
<tr>
<td>11</td>
<td>Default Direction. This read-write bit determines the default direction requests should be sent when originating with this device. A “0” in this register indicates requests should be sent in the direction of the master host (see previous bit). A “1” indicates requests should be sent in the other direction. Bit has no meaning for a single-link device. Warm Reset = 0.</td>
</tr>
</tbody>
</table>
HyperTransport System Architecture

**Host Command CSR.** Figure 17-3 and Table 17-2 show the format of the Host Command Register used by all host interfaces; key fields used in double-hosted chain configuration are highlighted.

*Figure 17-3: Host Command CSR: Key Fields In DHC Configuration*

<table>
<thead>
<tr>
<th>Bit</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>Double Ended.</strong> If this read-write bit is set = 1, there is another bridge at the other end of the chain (double-hosted chain). This bit does not affect hardware and can be used by software during initialization. If not implemented, hardwire this bit = 0. Cold reset = 0.</td>
</tr>
<tr>
<td>7</td>
<td><strong>Chain Side.</strong> This bit is set to indicate which side of the host bridge is being accessed. A “0” in this field indicates the read is coming from within; a “1” indicates the read is coming from the chain attached to the host interface. If double-hosted chains are not supported, this bit is hardwired = 0.</td>
</tr>
<tr>
<td>8</td>
<td><strong>Host Hide.</strong> This read-write bit is used to hide a bridge’s configuration space from accesses coming from the chain side. If the bit is set = 1, the host will behave as an end-of-chain device during configuration cycles; if it is clear, configuration accesses from the chain are allowed by the device. For hosts which support DHCs, this bit is cleared on warm reset. If a host does not support double-hosted chains, this bit is hardwired = 1.</td>
</tr>
<tr>
<td>10</td>
<td><strong>Act As Slave.</strong> This bit, when set, causes host to act as a slave, using the device number programmed in bits 6:2 as the base UnitID for requests and responses it sources. In this mode, the host also won’t set the Bridge bit in its responses. If this bit is clear, interface behaves as a host — using UnitID 0, setting the Bridge bit on responses it sends, etc. If host doesn’t support double-hosted chains, this bit is hardwired = 0. Cold reset = 0.</td>
</tr>
</tbody>
</table>
Chapter 17: Double-Hosted Chains

Two Types Of Double-Hosted Chains

There are two basic arrangements for double-hosted chains: sharing and non-sharing.

Sharing Double-Hosted Chain

In a sharing double-hosted chain, traffic is allowed to flow from end to end. Either host may target any of the devices in the chain, including the other host. In this arrangement, one host is the master host bridge and the other is the slave host bridge. The determination about which host is master or slave is not defined in the specification, but must be defined before reset occurs. Most likely, the system board layout will determine master/slave host bridges — possibly through a strapping option on the motherboard. Figure 17-4 on page 431 depicts a sharing double-hosted chain with master and slave host bridges.

Figure 17-4: Sharing Double-Hosted Chain With Master/Slave Host Bridges
The Previous Chapter

The previous chapter described the features of the optional HyperTransport double-hosted chain topologies. Topics include the reasons behind sharing and non-sharing chains, PCI configuration space registers used to initialize the fabric for multiple hosts, and tunnel support for upstream and downstream packets moving in both directions.

This Chapter

HT provides a variety of mechanisms to support power management. These mechanisms include LDTSTOP#, LDTREQ#, STPCLK messages, and STOP_GRANT messages. While these mechanisms are optional for HT devices, the specification requires this support for x86-based platforms. Note also that functions other than power management may make use of these signals and messages. This chapter discusses the strategy employed by HT for implementing power management and how a given platform can use these mechanisms to support power management.

The Next Chapter

The next chapter summarizes some of the major additions to the HyperTransport protocol which will be forthcoming in Release 1.05 and Release 1.1 of the specification. Collectively, these additions are referred the HyperTransport Networking Extensions, and target some of the special requirements of communications processing. Key features include a message passing protocol for larger packets, a formal definition for switch devices, a link-level error recovery method, sixteen optional additional posted write virtual channels with defined arbitration and bandwidth allocation, direct peer-peer transfers, an increase in the number of outstanding transactions for host bridges, and a 64-bit addressing option.
HyperTransport System Architecture

Background

Over the years, power management control has migrated to the operating system in many platforms. System and I/O devices designers provide registers for the OS to control power at the function, device, bus, and system levels.

- Hardware registers for power management reside in chipsets and IO devices (e.g. PCI).
- Transitions in power state typically occur under software control as chipset hardware detects inactivity time-outs and interrupts the processor, which in turn executes the power management routines.

Some buses, such as PCI, define power management registers for each function that can be programmed to cause changes in power states and to enable wakeup, if supported (e.g. modem wake-up). Other buses like the ISA bus appeared before power management was widely implemented, making it difficult to implement power management. That is, lacking a standard set of configuration registers, an ISA device doesn’t play well in either power management or plug-and-play schemes. The HT specification defines the signals LDTSTOP# and LDTREQ# (LDT Request) to support power management, but does not define a standard set of registers for meeting low power requirements. Instead, to meet platform power management requirements, devices can gate clocks, stop PLLs, and power down portions of the device after the LDTSTOP# signal is asserted. Remote wakeup can be implemented using LDTREQ#.

In addition to LDTSTOP# and LDTREQ#, the STPCLK and STOP_GRANT messages can also be used to support power management activities.

Reporting Power Management Events to the Host Bridge

HT provides two System Management messages that can be used to report power management events to the host.

- STPCLK message — uses the SMAF field to define the type of event being reported and the action to be taken. The SMAF field encoding is not defined by the HT specification.
- SMI message — used to notify (interrupt) the processor when a system management event occurs. The SMI message can support power management as well as other features. (Note: SMI is an x86 legacy signal and is discussed in more detail in “X86 CPU Compatibility” on page 491.)
Chapter 18: HT Power Management

Message transmission may be stimulated by either hardware or software. A system that supports power management will very likely include a set of registers within the chipset (e.g., the South bridge or ICH) to support power management. For example, a chipset may give system software the ability to select which hardware events will cause the chipset to send a power management notification message to the Host Bridge. Similarly, software (e.g. ACPI software) may access a register, causing a power management to be sent. The actual entity responsible for sending these message to the Host Bridge is the System Management Controller (SMC).

Reporting Host Power Management Events to SMC

The Host Bridge may also need to report power management messages to the System Management Controller residing on the HT bus (e.g. within the South bridge). The mechanism for reporting such events is the STOP_GRANT message. Like the STPCLK message, the SMAF field defines the type of event being reported and the action required.

Processor VID/FID

The specification defines a STOP_GRANT message for Voltage ID and Frequency ID changes associated with the processor. The Athlon processors support changing the frequency and voltage used by the processor as a means of power conservation. That is, if the processor clock frequency is slowed, the voltage may also be lowered for greater power conservation. When the host initiates a VID/FID change, this indicates that the system is entering a low-power state. Consequently, the north bridge must generate a STOP_GRANT message with an SMAF code specifying the VID/FID change. This message results in the assertion of LDTSTOP#.

Reporting Power Management Events to HT Devices

In response to a STOP_GRANT message, the SMC controller may assert LDTSTOP to all HT device interfaces as illustrated in Figure 18-1 on page 438. For example, when the Host Bridge detects that the Processor is entering a VID/FID change, it will send a STOP_GRANT message indicating the change and causing the SMC to assert LDTSTOP#. Details regarding the timing relationships between the reception of STOP_GRANT and the assertion of LDTSTOP# is discussed in “The Link Initialization Disconnect Sequence” on page 227.
HyperTransport System Architecture

As described earlier, devices can gate clocks, stop PLLs, and power down portions of the device after the LDTSTOP# signal is asserted. However, LDTSTOP# can be asserted for a variety of reasons besides power management. In such cases, it may be inappropriate for the device to enter a low power state. If a device needs to differentiate between the causes of LDTSTOP# assertion, it must monitor STOP_GRANT cycles and decode the SMAF code so that it knows the reason for LDTSTOP# being driven by the SMC.

Figure 18-1: LDTSTOP# is an Input to All HT Devices Except the SMC.
Chapter 18: HT Power Management

Signaling Wakeup

Most power management schemes provide a mechanism that allows a device to initiate a sequence that returns the system from a power conservation state to a fully operational state. HT provides this ability via the LDTREQ# signal. Figure 18-2 shows that the LDTREQ# signal is an output from each HT device interface and an input to the SMC.

Figure 18-2: LDTREQ# is an Output from All HT Devices and an Input to the SMC.
Networking Extensions Overview

The Previous Chapter

HT provides a variety of mechanisms to support power management. These mechanisms include LDTSTOP#, LDTREQ#, STPCLK messages, and STOP_GRANT messages. While these mechanisms are optional for HT devices, the specification requires this support for x86-based platforms. Note also that functions other than power management may make use of these signals and messages. The previous chapter discusses the strategy employed by HT for implementing power management and how a given platform can use these mechanisms to support power management.

This Chapter

This chapter summarizes some of the major additions to the HyperTransport protocol which will be forthcoming in Release 1.05 and Release 1.1 of the specification. Collectively, these additions are referred the HyperTransport Networking Extensions, and target some of the special requirements of communications processing. Key features include a message passing protocol for larger packets, a formal definition for switch devices, a link-level error recovery method, sixteen optional additional posted write virtual channels with defined arbitration and bandwidth allocation, direct peer-peer transfers, an increase in the number of outstanding transactions for host bridges, and a 64-bit addressing option.

The Next Chapter

HT is designed to support a variety of I/O and processor buses via bridges. The specification defines specific requirements for supporting PCI, PIC-X, AGP, and processor buses. The next chapter discusses these support requirements.
An Important Note

At the time of this writing, the HyperTransport I/O Link Specification Revision 1.04 is released. The 1.04 revision of the specification does not deal with the networking extensions but much of the work on the HyperTransport 1.05 and 1.1 revisions has been done by the HyperTransport Technical Working Group, and quite a bit of preliminary information on this important addition to the protocol has been released.

The networking extensions are backward compatible with the 1.04 and earlier revisions. Compatibility extends to any mix of devices which may or may not support the extensions.

This chapter presents material based on information currently available. Check http://www.hypertransport.org for updated information on all revisions to the specification.

Server And Desktop Topologies Are Host-Centric

As illustrated in Figure 19-1 on page 445, a typical desktop or server platform is somewhat vertical. It has one or more processors at the top of the topology, the I/O subsystem at the bottom, and main system DRAM memory in the middle acting as a holding area for processor code and data as well as the source and destination for I/O DMA transactions performed on behalf of the host processor(s). The host processor plays the central role in both device control and in processing data; this is sometimes referred to as managing both the control plane and the data plane.

HyperTransport works well in this dual role because of its bandwidth and the fact that the protocol permits control information including configuration cycles, error handling events, interrupt messages, flow control, etc. to travel over the same bus as data — eliminating the need for a separate control bus or additional sideband signals.
Figure 19-1: Host-Centric HyperTransport System

- **CPU**
- **HyperTransport Host Bridge**
- **DRAM Memory**
- **AGP**
- **System**
- **Infiniband**
- **PCI-X**
- **GB Ethernet**
- **SCSI**
- **PCI-X Audio**
- **PCI-X ISA**
- **Super IO**

**HyperTransport Devices**

**Host-Centric HyperTransport System Attributes:**
1. Centralized processing. CPU(s) handle both control and data processing.
2. Most IO Bus master transfers target main memory on behalf of the CPU(s).
3. Virtual channels limited to three: posted request, non-posted request, response.
4. Very little peer-to-peer traffic in IO subsystem.
5. Address-based storage semantics:
   5a. Source manages all target addresses; sources treat targets as source or sink for data.
   5b. Fixed-length transfers of 1 byte to 64 bytes.
Upstream And Downstream Traffic

There is a strong sense of upstream and downstream data flow in server and desktop systems because very little occurs in the system that is not under the direct control of the processor, acting through the host bridge. Nearly all I/O initiated requests move upstream and target main memory; peer-peer transactions between I/O devices are the infrequent exception.

Storage Semantics In Servers And Desktops

Without the addition of networking extensions, HyperTransport protocol follows the conventional model used in desktop and server busses (CPU host bus, PCI, PCI-X, etc.) in which all data transfers are associated with memory addresses. A write transaction is used to store a data value at an address location, and a read transaction is used to later retrieve it. This is referred to as associating storage semantics with memory addresses. The basic features of the storage semantics model include:

**Targets Are Assigned An Address Range In Memory Map**

At boot time, the amount of DRAM in the system is determined and a region at the beginning of the system address map is reserved for it. In addition, each I/O device conveys its resource requirements to configuration software, including the amount of prefetchable or non-prefetchable memory-mapped I/O address space it needs in the system address map. Once the requirements of all target devices are known, configuration software assigns the appropriate starting address to each device; the target device then “owns” the address range between the start address and the start address plus the request size.

**Each Byte Transferred Has A Unique Target Address**

In storage semantics, each data packet byte is associated with a unique target address. The first byte in the data packet payload maps to the start address and successive data packet bytes are assumed to be in sequential addresses following the start address.

**The Requester Manages Target Addresses**

An important aspect of storage semantics is the fact that the requester is completely responsible for managing transaction addresses within the intended tar-
Chapter 19: Networking Extensions Overview

get device. The target has no influence over where the data is placed during write operations or retrieved in read operations.

In HyperTransport, the requester generates request packets containing the target start address, then exchanges packets with the target device. The maximum packet data payload is 64 bytes (16 dwords). Transfers larger than 64 bytes are comprised of multiple discrete transactions, each to an adjusted start address. Using HyperTransport’s storage semantics, an ordered sequence of transactions may be initiated using posted writes or including a non-zero SeqID field in the non-posted requests, but there is no concept of streaming data, per se.

Storage Semantics Work Fine In Servers And Desktops

As long as each requester is programmed to know the addresses it must target, managing address locations from the initiator side works well for general purpose data PIO, DMA, and peer-peer exchanges involving CPU(s), memory and I/O devices. When the target is prefetchable memory, storage semantics also help support performance enhancements such as write-posting, read pre-fetching, and caching — all of which depend on a requester having full control of target addresses.

1.04 Protocol Optimized For Host-Centric Systems

Because the HyperTransport I/O Link Protocol was initially developed as an alternative to earlier server and desktop bus protocols that use storage semantics (e.g. PCI), the 1.04 revision of the protocol is optimized to improve performance while maintaining backwards compatibility in host-centric systems:

1. The strongly ordered producer-consumer model used in PCI transactions which guarantees flag and data coherence regardless of the location of the producer, consumer, flag location, or data storage location is available in the HyperTransport protocol.
2. Virtual channel ordering may optionally be relaxed in transfers where the full producer-consumer model is not required.
3. The strong sense of upstream and downstream traffic on busses such as PCI is also preserved in HyperTransport. Programmed I/O (PIO) transactions move downstream from CPU to I/O device via the host bridge. I/O bus master transactions move upstream towards main memory.
4. Direct peer-peer transfers are not supported in the 1.04 revision of the HyperTransport I/O Link Specification; requests targeting interior devices must travel up to the host bridge, then be reissued (reflected) back downstream towards the target.
20 I/O Compatibility

The Previous Chapter

The previous chapter summarized some of the major additions to the Hyper-Transport protocol which will be forthcoming in Release 1.05 and Release 1.1 of the specification. Collectively, these additions are referred the HyperTransport Networking Extensions, and target some of the special requirements of communications processing. Key features include a message passing protocol for larger packets, a formal definition for switch devices, a link-level error recovery method, sixteen optional additional posted write virtual channels with defined arbitration and bandwidth allocation, direct peer-peer transfers, an increase in the number of outstanding transactions for host bridges, and a 64-bit addressing option.

This Chapter

HT is designed to support a variety of I/O and processor buses via bridges. The specification defines specific requirements for supporting PCI, PIC-X, AGP, and processor buses. This chapter discusses these support requirements.

The Next Chapter

The large 1 Terabyte HT address space may be outside the limits of a given processor or expansion bus. When address locations are mapped into the HT space that exceed the processor or expansion bus address space then the addresses must be remapped to/from HT space. The next chapter discusses the HT solution for remapping prefetchable memory, MMIO, and I/O addresses.

Introduction

PC compatibility remains important in many operating environments and may include the use of several legacy buses. HT is intended to support connections to a variety of I/O buses, and the specification gives special attention to the PC legacy buses to ensure compatible support.
HyperTransport System Architecture

Several areas must be considered to ensure compatibility with I/O buses, including:

- Protocol differences
- Ordering requirements
- Command translation
- Address space and range differences
- Potential deadlocks

The buses currently supported and discussed by the HT specification include PCI, PCI-X, AGP, and host processor buses, with special attention given to issues associated with PC legacy support via the ISA/LPC buses.

PCI Bus Issues

Several features of the PCI bus must be handled in the correct fashion when interfacing with the HT bus. For background information and details regarding PCI ordering, refer to MindShare’s PCI System Architecture book, 4th edition.

PCI Ordering Requirements

Transaction ordering on the PCI bus is based on the Producer/Consumer programming model. This model involves 5 elements:

1. Producer — PCI master that sources data to a memory target
2. Target — main memory or any PCI device containing memory
3. Consumer — PCI master that reads and processes the Producer data from the target
4. Flag element — a memory or I/O location updated by the producer to indicate that all data has been delivered to the target, and checked by the Consumer to determine when it can begin to read and process the data.
5. Status element — a memory or I/O location updated by the Consumer to indicate that it has processed all of the Producer data, and checked by the Producer to determine when the next batch of data can be sent.

This model works flawlessly in PCI when all elements reside on the same shared PCI bus. When these elements reside on different PCI buses (i.e. across PCI to PCI bridges, the model can fail without adherence to the PCI ordering rules.
Chapter 20: I/O Compatibility

The PCI specification, versions 2.2 and 2.3, defines the required transaction ordering rules. These ordering rules are included in this section as review and to identify rules that have may have no purpose in some HT designs. Table 20-1 on page 459 defines the ordering rules for PCI bridges. When reading the table, please note the following:

- PMW stands for posted memory write.
- DRR and DRC stand for Delayed Read Request and Delayed Read Completion, respectively.
- DWR and DWC stand for Delayed Write Request and Delayed Write Completion, respectively.
- “Yes” specifies that the transaction just latched must be ordered ahead of the previously latched transaction indicated in the column heading.
- “No” specifies that the transaction just latched must never be ordered ahead of the previously latched transaction indicated in the column heading.
- “Yes/No” entries mean that the transaction just latched is allowed to be ordered ahead of the previously-latched operation indicated in the column heading, but such reordering is not required. The Producer/Consumer Model works correctly either way.

<table>
<thead>
<tr>
<th>Transaction just latched</th>
<th>Posted Memory Write</th>
<th>Delayed Request</th>
<th>Delayed Completion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PMW (row 1)</td>
<td>DRR (row 2)</td>
<td>DWR (row 3)</td>
</tr>
<tr>
<td>PMW (row 1)</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DRR (row 2)</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DWR (row 3)</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRC (row 4)</td>
<td>No</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>DWC (row 5)</td>
<td>Yes/No</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note that all of the transaction types listed under the heading, “transaction just latched” (except Delayed Write Completions, because the write has already completed) must never be reordered ahead of a previously posted memory write transaction (column 1). These rules are present to enforce proper opera-
HyperTransport System Architecture

Avoiding Deadlocks

PCI ordering rules require that Posted Memory Writes (PMWs) in Row 1, be ordered ahead of the delayed requests and delayed completions listed in columns 2-5. This requirement is based on avoiding potential deadlocks. Each of the deadlocks involve scenarios arising from the use PCI bridges based on earlier versions of the specification. If all PCI bridge designs used in HT platforms are based on 2.1 and later versions of the PCI specification, the PCI ordering rules with “Yes” entries in row 1 can be treated as “Yes/No.”

Table 20-1 also specifies that Delayed Read Completions and Delayed Write Completions in rows 4 and 5, must be ordered ahead of the Delayed Requests in Columns 2 and 3. These ordering rules arise from potential deadlocks that can occur when two hierarchical bridges are implemented as illustrated in Figure 20-1 on page 460. Refer to MindShare’s PCI System Architecture book for a detailed explanation of this deadlock. If a given platform avoids this topology, then the “Yes” entries in rows 4 and 5 can be treated as “Yes/No.”

*Figure 20-1: Topology Causing Deadlock Scenario for Rows 4 and 5*
Subtractive Decode

PCI employs a technique referred to as subtractive decode to handle devices that are mapped into memory or I/O address space by user selection of switches and jumpers (e.g. ISA devices). Consequently, configuration software has no knowledge of the resources assigned to these devices. Fortunately, these PC legacy devices are mapped into relatively small ranges of address space that can be reserved by platform configuration software.

**Subtractive Decode: The PCI Method**

Subtractive decode is a process of elimination. Since configuration software allocates and assigns address space for PCI, HT, AGP and other devices, any access to address locations not assigned can be presumed to target a legacy device, or may be an errant address.

All PCI devices must perform a positive decode to determine if they are being targeted by the current request. This decode must be performed as a fast, medium, or slow decode. The device targeted must indicate that it will respond to the request by signaling device select (DEVSEL#) across the shared bus. When device driver software issues a request with an address that has not been assigned by configuration software, no PCI device is targeted (i.e. no DEVSEL# is asserted within the time allowed) By process of elimination, the subtractive decode agent recognizes that no PCI device has responded and therefore it asserts DEVSEL# and forwards the transaction to the ISA bus, where the request is completed.

**Subtractive Decode: The Simple HT Method**

An HT system with a single chain can possibly implement subtractive decode without extra host support required of more complex HT systems. Figure 20-2 on page 462 illustrates a simple system with a single-hosted chain. Note that the subtractive decode agent is at the end of the chain. If a request initiated at the host reaches the South Bridge, then the bridge knows that no other HT agents have claimed the transaction based on positive decode; therefore, a subtractive decode is safe.
21 Address Remapping

The Previous Chapter
HT is designed to support a variety of I/O and processor buses via bridges. The specification defines specific requirements for supporting PCI, PIC-X, AGP, and processor buses. The previous chapter discussed these support requirements.

This Chapter
The large 1 Terabyte HT address space may be outside the limits of a given processor or expansion bus. When address locations are mapped into the HT space that exceed the processor or expansion bus address space, then the addresses must be remapped to and from HT space. This chapter discusses the HT solution for remapping memory, MMIO, and I/O addresses.

The Next Chapter
Many HT platforms may be based on x86 processors. Compatibility support for these processors is defined by the HT specification. This chapter discusses the x86 features that require specific support by HT technology and details the HT methods of signaling.

Introduction
An HT-based system may include processors or expansion buses that have smaller address ranges than the large 1 TeraByte space used by HT. In such cases address translation may be required. For example, the x86 CPU might have a maximum memory address range of 64GB and a PCI bus implementation may be limited to 4GB of memory address space. If addresses are allocated in HT address space beyond the range of the CPU and Expansion bus, then address translation is required.
HyperTransport System Architecture

To illustrate the possible remapping requirement, Figure 21-1 on page 479 depicts an implementation similar to that described in the previous paragraph. This example does not depict a typical implementation, but rather illustrates an extreme case where address remapping is required by both the CPU to HT Bridge and the HT-to-PCI Bridge. The system allocates a 4GB range of processor memory address space for PCI devices that is near the top of the 64GB range. This processor address space is mapped into the HT space above the 64GB range of the processor and beyond the address range of the PCI bus. When software executing on the CPU accesses a memory location within a PCI device two address translations must occur:

1. The CPU to HT Bridge must translate the CPU address within the 60 - 64GB address range to the 1,008 to 1,012GB range.
2. The HT-to-PCI Bridge must translate the HT address down to the 0-4GB range of the PCI bus.

A similar address translation must take place to support I/O address space. HT maps I/O space very high in its address space and outside the range of both the CPU and PCI bus, thereby creating the need for address remapping.

HT provides an Address Remapping Capability Block that support remapping HT addresses to expansion buses. However, the specification does not define a mechanism for remapping CPU to HT address space.

The Address Remapping Capability Block

This capability block allows software to control the HT to expansion bus bridge address for:

- remapping HT I/O address space down to a lower range within an expansion bus.
- remapping HT MMIO space to expansion bus address space.
- remapping HT Memory space to expansion bus address space.
- remapping Bus Master memory transactions from expansion bus address space to HT address space.

Figure 21-2 on page 479 illustrates the format of the Address Remapping capability block. The following sections describe each register and its remapping function.
Chapter 21: Address Remapping

Figure 21-1: HT Address Space May Exceed that of the Processor and Expansion Bus

Figure 21-2: Format of the Address Remapping Capability Block

<table>
<thead>
<tr>
<th>Cap Type</th>
<th>Map Type</th>
<th>I/O Size</th>
<th># of DMA Mappings</th>
<th>Capabilities Pointer</th>
<th>Capability ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>En</td>
<td>Reserved</td>
<td></td>
<td></td>
<td>Secondary Bus Non-Prefetchable Window Base</td>
<td>00h</td>
</tr>
<tr>
<td>En</td>
<td>Reserved</td>
<td></td>
<td></td>
<td>Secondary Bus Prefetchable Window Base</td>
<td>04h</td>
</tr>
<tr>
<td>En</td>
<td>Reserved</td>
<td></td>
<td></td>
<td>DMA Primary Base 1</td>
<td>08h</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DMA Secondary Base 1</td>
<td>DMA Secondary Limit 1</td>
<td>0Ch</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DMA Primary Base N</td>
<td>+8N=0Ch</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DMA Secondary Limit N</td>
<td>+8N=10h</td>
</tr>
</tbody>
</table>
The registers within the remapping capability block are briefly described below. Detailed discussion of their implementation is discussed in the following sections.

- **Number of DMA Mappings** — This read-only field indicates how many DMA Primary/Secondary register sets (if any) are defined by the remapping capability block.
- **I/O Size** — This register is intended to permit configuration software to limit the I/O address range supported by the expansion bus to a range smaller than the 32MBs supported by HT.
- **Mapping Type** — The HT specification does not currently define a mapping type. This read-only register must be zero (0) and is reserved for future extensions.
- **Capability Type** — A read-only value of 01000b indicates an address remapping capability block.
- **Secondary Bus Non-Prefetchable Window Base** — This register is written by configuration software to define the HT address range that is mapped to MMIO space on the secondary (expansion) bus. Contents of this register are address bits 39:20, providing a minimum HT range of 1MB for expansion bus MMIO addresses.
- **SBNPctl** — This register permits software to enable/disable the Secondary Bus Non-Prefetchable Window Base register. When the window register is enabled, the **SBNPctl** register also specifies which HT attributes are supported by the expansion bus. (i.e. HT requests passing through the window may have these attributes active). The attributes include NonCoherent, Isochronous, and Compatibility.
- **Secondary Bus Prefetchable Window Base** — This register is written by configuration software to define the HT address range that is mapped to prefetchable memory address space on the secondary (expansion) bus. Contents of this register are address bits 39:20, providing a minimum HT range of 1MB for expansion bus prefetchable memory addresses.
- **SBPreCtl** — This register permits software to enable/disable the Secondary Bus Prefetchable Window Base register. When the window register is enabled, the **SBPreCtl** register also specifies which HT attributes are supported by the expansion bus. The attributes include NonCoherent, Isochronous, and Compatibility.
- **DMA Primary Base 1-N** — The **DMA Primary Base** register permits configuration software to define the HT base address (bits 39:24) for each DMA mapping. These mappings are used by bus masters residing on the expansion bus when accessing HT address space that is outside the range of the expansion bus.
- **DMACtl 1-N** — This register permits software to enable/disable the corresponding DMA mapping. When the mapping is enabled, the **DMACtl** regis-
ter also specifies whether NonCoherent and Isochronous transactions are supported on the expansion bus.

- **DMA Secondary Base 1-N and DMA Secondary Limit 1-N** — These registers create a DMA memory window within the secondary (expansion) bus address range for DMA transfers to HT address space. The starting address of the HT range is specified by the DMA Primary Base register.

### I/O Address ReMapping

Because HT does not directly support I/O address space, it reserves a 32MB of memory address space for transporting I/O addresses across the bus. When the processor initiates an I/O transaction, the address must be translated by the CPU to HT bridge to an address within the reserved HT I/O address range (FD_FC00_0000h- FD_FDFF_FFFFh). HT transports the transaction across the HT chain to the HT-to-PCI bridge which must translate the HT I/O address so that is falls within the expansion bus I/O address range.

The **I/O Size** register within the Address Remapping Capability block supports the I/O address space remapping. This 5-bit register defines the size of the I/O address space on the expansion bus, up to 32MBs. Note that the upper 15 bits of the HT I/O address range (FD_FC00_0000h- FD_FDFF_FFFFh) identifies the address as I/O, while the lower 25 address bits define any location within the 32MB I/O address range. The I/O Size register specifies the number of upper address bits (Addr 24:0) that will not be used when generating an I/O address on the expansion bus. For example, an I/O size value of 01101b (13d) results in an expansion bus I/O range of 4KB (Addr 11:0). The default value of zero causes all 25 bits of a HyperTransport I/O request to be passed to the expansion bus, thereby supporting the maximum 32MB I/O range. The next two sections give example implementation of I/O address remapping.

### X86 Processor and PCI I/O Remapping Example

This example assumes that the x86 processor and the PCI bus support a maximum I/O address space of 64KB as illustrated in Figure 21-3 on page 483. The Host to HT bridge translates each processor initiated I/O address up to the reserved HT I/O address range and the HT-to-PCI bridge translates the address back down to the PCI I/O address range. HT-to-PCI bridges use the PCI to PCI bridge’s I/O Base and Limit configuration registers that are defined by the PCI 2.3 and earlier specifications, along with the I/O size field of the Address Remapping capability block.
The Previous Chapter

The large 1 Terabyte HT address space may be outside the limits of a given processor or expansion bus. When address locations are mapped into the HT space that exceed the processor or expansion bus address space, then the addresses must be remapped to and from HT space. The previous chapter discussed the HT solution for remapping memory, MMIO, and I/O addresses.

This Chapter

Many HT platforms may be based on x86 processors, and the HT specification defines the necessary compatibility support. This chapter discusses the x86 features that require specific support by HT and details how HT provides the capability.

Background

Several x86 processor-specific features require support for both hardware and software compatibility, including:

- A method for supporting x86 CPU legacy signals
- A method for supporting x86 Special Cycles
- x86 legacy interrupt support — PCI (8259) and APIC
- Power Management

Other legacy issues related to Industry Standard Architecture (ISA) platforms deal with I/O bus compatibility, rather than processor-related issues. These topics are discussed in the chapter entitled, “I/O Compatibility.”
Legacy Signals

Support for the features mentioned above involves a number of X86 signals. These signals include those preserved throughout the x86 processor evolution to maintain compatibility:

- INTR (Interrupt Request from the 8259 Interrupt Controllers)
- APIC bus (Advanced Programmable Interrupt Controller bus)
- A20M (Processor Address Line A20 Mask)
- SMI (System Management Interrupt)
- SMIACT (System Management Interrupt Active)
- STPCLK (Stop Clock)
- FERR (Floating-Point Error)
- IGNNE (Ignore Numeric Error)

Figure 22-1 on page 492 illustrates the typical implementation of these signals in a non-HT system. Note that these signals are typically routed directly between the CPU and South bridge in a legacy platform with PCI.
Chapter 22: X86 CPU Compatibility

Legacy Special Cycles

Some x86 CPU events are signaled via special cycles. The CPU uses its system interface (e.g. front-side bus) to signal special cycles to the Host Bridge. The events signaled by an x86 CPU include:

- SHUTDOWN
- HALT
- INTA (Interrupt Acknowledge)
- STOP_GRANT
- Voltage ID/Frequency ID Change (Athlon processor)
- WBINVD (write-back and invalidate)
- INVD (invalidate)

Special cycles have various jobs related to x86 functions such as Interrupts, System Management Mode (SMM), power management, and cache coherency.

Note that each of the x86 signals and special cycles are discussed within the section that describes the x86 function to which the signal or special cycle relates.

System Management Messages

HyperTransport eliminates the direct signal routing between the x86 CPU and the compatibility bridge (South Bridge, ICH, etc.) used in legacy platforms. Instead, HT defines System Management (SM) requests that serve to convey information that otherwise would be conveyed via signals. These messages act as virtual wires that signal INTR, FERR#, IGNNE#, A20M#, STPCLK#, SMI# and SMIACT#. Note that the default state of virtual wires is deasserted. Figure 22-2 on page 494 illustrates that messages may move in either direction.

Delivery of special cycle messages is also done via SM messages. When the Host Bridge receives a special cycle from the CPU, it sends an SM message that delivers the special cycle message to interested parties residing on the HT bus. Refer to Chapter 22, entitled "X86 CPU Compatibility," on page 491 for a detailed explanation of the SM messages.
X86 Interrupt Support

The HT specification defines the mechanism necessary to support x86 compatible interrupt handling. This mechanism supports both single and multiprocessor interrupt handling:

- Cascaded PIC (8259) Interrupt controllers for legacy support with single processor systems.
- APIC support for x86 multiprocessor systems.

The APIC solution includes support for delivering 8259 interrupt requests to a single processor, and HT takes advantage of this support for delivering interrupt requests in single processor platforms. Therefore, the HT method of supporting APIC interrupt controllers is discussed first, followed by a discussion of 8259 support.
Chapter 22: X86 CPU Compatibility

APIC Interrupt Support

The APIC subsystem was designed to support multi-processing implementations. The subsystem permits an interrupt to be directed to a particular processor for handling or may deliver an interrupt to a group of processors and allow them to arbitrate to determine which one will accept and service the interrupt. In addition to standard interrupt delivery, the APIC bus supports delivery of other interrupts including 8259 interrupts, NMI (Non Maskable Interrupts), and SMI# (x86 System Management Interrupts).

Legacy Method of Handling APIC Interrupts

Figure 22-3 on page 495 illustrates a standard APIC subsystem implemented in a legacy-based system with a PCI-X expansion bus. The APIC subsystem is split between the IO APIC located in the ICH and local APIC modules associated with each CPU. A synchronous APIC bus connects the IO APIC with the local APIC modules of all processors.
Appendix

Glossary of Terms
Address Map. The 1.04 revision of the HT specification defines a 40-bit memory map. All resources which can be targeted with directed packets are mapped into this space. In addition, reserved portions of the memory map are used for special purposes, including IO accesses, broadcast messages, interrupts, configuration cycles, etc.

AGP (Accelerated Graphics Port). A high-performance, point-to-point interface which connects a graphics adapter to the main memory controller in the Memory Controller Hub (MCH) or North Bridge (Host/PCI Bridge).

Atomic Read-Modify-Write Transaction. A hybrid read and write operation issued from one source, targeting main memory. An Atomic RMW is guaranteed to complete without intervening accesses of the same location by any other device. This command is useful when a memory semaphore is being updated by one of the devices sharing it. HT supports two Atomic RMW variants: Fetch and Add and Compare and Swap.

Base Address Registers (BARs). Device configuration registers that define the start address, length, and type of address space required and owned by a device. The type of space implemented will be either memory or I/O. The value written to this register during device configuration will program its address decoders to detect and accept accesses within the indicated range. Because HT memory maps IO accesses, an IO request in the Base Address Register will result in the assignment of a starting address in the memory map range reserved for IO.

Bit Time. One half of a link clock period. As a double data rate (DDR) interface, HT CAD and CTL information is sent during each bit time, resulting in two bits transferred, per signal, per clock.

Bridge. A device that provides a logical and electrical interface between two independent buses. Examples would be the bridge between the host processor bus and HyperTransport, a bridge between PCI/PCI-X and HT, or a bridge between two HT buses. Each secondary interface hosts a new bus (chain).

Broadcast Message. A special case of a posted write request which is used to deliver a message to all devices that see it. There is no specific target; each device accepts it (based on the command type) and forwards it downstream. The end-of-chain device accepts the message, and drops it.

Broadcast Request. See Broadcast Message.
HyperTransport System Architecture

**Bus Concurrency.** Separate transfers occurring simultaneously on two or more buses. Because HT is implemented with independent point-to-point connections instead of a shared bus, HT concurrency can occur within the same chain.

**Byte.** 8 bits of digital information.

**Byte Mask.** In HT, there are no separate byte enable signals; data bus usage is implied in the command type and the mask/count field which accompanies request and response packets. For WrSized (byte) requests, a 32 bit “byte mask” precedes the 1-16 dword data packet and indicates valid bytes being sent — much like byte enables in PCI.

**Byte Read.** Sized read requests (RdSized) in HT carry a bit indicating whether the data to be transferred is in bytes or dwords. For byte reads, the maximum transfer size is one dword (four bytes); the mask field in the request indicates the valid bytes being transferred. Any byte combination is valid.

**Byte Write.** Sized write requests (WrSized) in HT also carry a bit indicating whether the data to be transferred is in bytes or dwords. For byte writes, any combination of bytes within a 32-byte, address aligned group may be transferred. The count field in the request indicates the total dwords sent; valid bytes within those dwords are indicated in the 32-bit byte mask which precedes the data.

**Cache.** A relatively small amount of high-speed Static RAM (SRAM) that is used by CPUs to keep copies of code/data information recently read from system DRAM memory. Data from internal CPU caches may be accessed at full internal clock speed, avoiding a bus cycle to main memory.

**Cache Line.** When data is moved into or out of a cache, the transfer occurs in fixed amounts called cache lines. The size of a cache line is cache design dependent, but typically is 32, 64, or 128 bytes.

**CAD Bus.** The CAD (Command, Address and Data) bus carries all information, control, and data packets between two devices on a link. There is one CAD bus in each link direction, and bus width ranges from 2 bits to 32 bits wide.

**Capability Registers.** HT devices implement one or more sets of advanced capability registers to support features beyond basic PCI compliance. Capability register defined include: Host/Secondary interfaces, Slave/Primary interfaces, Interrupt Discovery and Configuration, Address Remapping, Revision ID, etc.

**Cave Device.** A single-link HT device. These always reside at the end of a chain and are also referred to as end devices.
Chain. In HT, a logical bus may be comprised of multiple devices daisy-chained together. At the top of the chain is a host bridge, in the middle there may be dual-link tunnel devices. At the end of a chain is a device with a single link connection to the chain. This may be a cave device, a tunnel device using only one interface, or the primary interface of a bridge to a new chain (bus).

Chain Down Error. In HT, each device is required to track outstanding requests until a response is returned. A chain down error is said to have occurred if a link goes down between the time a non-posted request is issued and its response returns. A reset will flush any pending requests after a chain down error.

Chain Fail. A chain fail occurs when a Sync flood or an event that can cause one is detected. Each device which detects the event sets the chain fail bit in its Error Handling register. The bit is cleared on reset.

Clocking Modes. HT supports three clocking modes of a link receiver interface with respect to the corresponding transmitter: synchronous clocking, pseudo-synchronous clocking, and asynchronous clocking.

Coherency. If the information resident in a cache accurately reflects the original information in DRAM memory, the cache is said to be coherent or consistent. In HyperTransport, transactions targeting DRAM main memory may either require action to guarantee coherency, or not. The coherent bit in WrSized and RdSized request packets indicates whether or not the host bridge must take coherency actions (cause a snoop of the CPU caches, etc.).

Coherent Bit. See Coherency.

Command Code. Each HyperTransport control packet contains a 6-bit command code in the first byte. This information informs other devices of the intended operation and the format of the remainder of the packet. Some of the 6-bit command codes include options bits which may be used to indicate whether the packet is to travel in the isochronous channel, is posted or not, etc.

Compatibility Bit. The compatibility bit (Compat) in an HT request is set by the host bridge to indicate the packet must be forwarded down the compatibility chain in the direction of the compatibility bridge where it will be accepted, regardless of the address it carries. The use of this bit provides compatibility with the South Bridge subtractive decoder in PCI systems where transactions which are otherwise unmapped in the system may be claimed by the subtractive decoder and forwarded to the compatibility bus (e.g. ISA).
Configuration Cycle. A link transaction to read or write the contents of a device's configuration registers is called a configuration cycle. In HT, configuration accesses are performed using RdSized and WrSized requests targeting addresses reserved for configuration type 0 and type 1 cycles.

Configuration Space. Each HT device is required to implement the 256 byte configuration space required of all PCI-compliant devices. Because PCI permits 256 busses in a system, 32 logical devices per bus, and 8 functions per device, the total configuration address space to reach all possible devices and functions is 16MB (256 busses x 32 devices x 8 functions x 256 bytes = 16MB). HT memory maps the entire configuration address space in a 32 MB reserved address range; the lower 16MB of the reserved range is for type 0 configuration cycles and the upper 16MB of the reserved range indicates type 1 configuration cycles.

Configuration Header Region. The first one-fourth of the configuration space (64 bytes) has a well-defined format and is referred to as the configuration header region. The two key header formats are type 0 (non-bridge) and type 1 (HT bridges or bridges between HT and other compatible protocols (PCI, PCI-X, or AGP).

Consistency. See Coherency.

Control Packet. Control packets include information, request, and response types. Information packets are used for local communication between the transmitter-receiver pairs on each link. Request packets may be 4 or 8 bytes, and are used to initiate transactions. Response packets are 4 bytes and are returned by the target of each non-posted request.

CRC. In HT, a Cycle Redundancy Code (CRC) is used to assure the integrity of transmitted data. Starting after reset, each transmitter on each link interface calculates a 32-bit CRC value and periodically sends it to the corresponding receiver where it is checked against the value calculated as CAD packets arrive. CRC is calculated independently for each 8 bits of CAD width. The error handling strategy for handling CRC errors is programmable.

CRC Testing Mode. HT provides a method for stress-testing CRC checking at the receiver. If both devices on a link support the CRC test mode, a transmitter can enter the CRC test mode under software control and generate dummy packets which are checked for CRC validity by the receiver, then dropped.

CTL Signal. The link control (CTL) signal is driven by each transmitter to indicate to the receiver that control packet information is in transit over the CAD bus; when CTL is deasserted by the transmitter, a data packet is in transit. The receiver uses the CTL signal to demultiplex control and data packets.
Index

**Numerics**

Count[3:0] 76, 91
Mask[3:0] 76, 78
Addr[39:2] 22
Command 69, 74, 146, 151, 154, 156, 162, 164, 168, 170, 175, 176, 180, 183, 185, 190, 192, 194, 195
8259 EOI Command 507
8259 Interrupt Compatibility 505
A
A20 Mask Support 508
Accepting Packets 264
Address Map 23, 202, 312, 446
Address Remapping 477
Address Remapping Capability Block 419, 478, 479
Address Remapping Capability Block Format 479
Address-Based Semantics 21, 22
Advanced Capability Registers 323
AGP Compatibility 470
AGP Ordering 471
APIC Interrupt Support 495
Asynchronous Clock Mode 402
Atomic Operations 46
Atomic Read-Modify-Write 46, 66, 75, 89, 91, 182, 250, 261
B
Bandwidth per Pin 17
Base Address Register 329, 424
Bit-Time 294, 381
Broadcast 263
Broadcast Message 46, 70, 75, 82, 83, 207, 261, 266, 521
Broadcast Request 220, 263, 268
Bus Numbering 318
Byte Mask 141, 159, 160, 252, 466
Byte Read 37, 78, 79, 96, 167
Byte Write 42, 78, 79, 156
C
Cache Line Size Register 329, 425
CAD Signal Group 55
CAD Signals 28, 30, 35, 55, 291, 292, 294, 365, 388
Capabilities Pointer 209, 328, 330, 334, 355, 414, 425
Capability Command Register. 266
Capability ID 209, 324, 331, 334, 355
Capability Registers 50, 225, 245, 276, 299, 300, 308, 323, 470
Capability Type 209, 332, 336
Cave 20, 33
Cave Device 243, 245, 259, 321, 333
Chain 20, 33, 82
Chain Down Error 230, 247
Chain Fail 247, 350, 351
Chain Initialization 316, 318
CLK Signal 28, 35, 55, 383
Clock Distribution Skew 392
Clock Frequency Tuning 299
Clock Initialization 388
Clock Signals 55
Clock Variance Timing Budget 393
Clocking Mode 388, 394, 401
Coherent Bit 488
Cold Reset 276
Command Register 325, 410
Compare and Swap 70, 90, 184, 188
Compatibility Bit 82, 152, 260, 264, 269, 326, 463, 488
Configuration 434
Configuration Cycle Types 309
Index

Configuration Cycles 75, 82, 267, 306, 308, 309, 310, 311, 313, 315, 316, 408, 429, 434, 444
Configuration Space 117, 200, 230, 306, 308, 310, 311, 408, 434, 470
Configuration Space Mapping 312
Consortium URL 16, 17
Control Packet 28, 35, 55, 65, 140, 238
Control Packet Format 69
Control Packet Types 36
CRC 50, 58, 231, 232, 234, 235, 242, 254, 282
CRC Test Mode 237, 337
CRC Window 223, 282, 294
Cross-Byte Skew 391
CTL Signal 35, 55, 67, 68, 104, 231, 237, 294, 349, 365
Data Coherency 120
Data Packet 28, 35, 66, 67, 140, 263, 506
Data Packet Types 37
Direct Memory Access 25, 121
Directed Requests 263, 266, 267, 268
DMA Primary Base Register 480
DMA Remapping 480
DMA Secondary Base and Limit Registers 481
DMA Transactions 121
DMA Transfers 25
DMA Window Control Register 480
Double-Host Chain Initialization 320
Double-Hosted Chain 137, 320, 428, 431, 432, 433, 434
Double-Hosted Chain Ordering 137
Downstream Ordering Rules 123, 136
Dword Read 37, 79, 161, 164
Dword Write 67
End of Chain 230, 243, 265, 432
End of Interrupt 200, 207, 502, 507
Enumeration Scratchpad Register 360
EOI Packet Format 208, 209
EOI Request Message Format 503
Error Bit 42, 49, 241, 243, 250, 251
Error Conditions 230
Error Handling Register 230, 234, 247, 350, 360
Error Interrupts 230
Error Logging 234, 239, 241, 243, 245, 249, 307
Error Reporting 60, 234, 239, 242, 246, 250, 259, 517
Error Response 230, 250, 251
Error Signaling 58
Fairness Algorithm 48, 258, 272
Fairness and Forward Progress 271, 272
Fatal Interrupt 231, 234, 252
Feature Capability Register 347, 348, 359
Fence Request 45, 87, 88, 179, 180
Fence Transactions 45
Fetch and Add 90, 184
Index

FIFO Size Calculation 394
flow 101
Flow Control 36, 42, 47, 58, 65, 68, 71, 99, 101, 103
Flow Control Buffers 71, 75, 104, 105, 106, 107, 145, 241, 452
Flow Control Counters 107
Flush Request 39, 44, 66, 84, 86, 97, 173, 175, 176
Flush Transactions 44
Forwarding Packets 259, 265, 271
Frequency Capability Register 299
G
Globally Ordered 134, 136
Globally Visible 84, 124, 134, 135
H
High-Speed Signals 28, 55
Host Bridge 24, 34, 39, 82, 121, 132, 137, 200, 216, 220, 226, 258, 267, 270, 307, 316, 320, 428, 432, 436, 446, 493
Host Ordering 132
Host/Secondary Interface 354, 355, 409
HyperTransport Consortium 16
HyperTransport Protocol Characteristics 16
HyperTransport-to-HyperTransport Bridge 318, 408, 409
I
I/O Address ReMapping 481
I/O Base and Limit Registers 420, 482
I/O Stream 37, 39, 137
Information Packet 36, 65, 71, 115, 145, 259
Initialization Complete 266, 295, 317
Interleaving Control and Data Packets 67
Interrupt Capability Block Format 209
Interrupt Definition Register 209, 210, 211, 500
Interrupt Discovery and Configuration Capability Block 200, 209, 210
Interrupt Information Fields 205
Interrupt Line Register 330, 425
Interrupt Pin Register 330, 425
Interrupt Request Data Packet 206
Interrupt Request Packet 204
Interrupt Request Packet Format 205
Interrupt Requests 204
Interrupt Signaling 58
Interrupt Type Field 205
Isoc Bit 116, 360
Isochronous Channel 37, 48, 73
Isochronous Flow Control 75, 116, 348
Isochronous Transactions 48
L
Latency Timer Register 329, 424
LDTREQ# 29, 30, 57, 366, 436, 439, 440, 441
LDTSTOP# 29, 30, 57, 72, 216, 223, 225, 226, 282, 300, 304, 365, 436, 437
LDTSTOP# Signaling 438
Legacy Numeric Error Handling 516
Legacy Signal Support 492
Index

Link Configuration Register 282, 288, 295, 337, 339, 358
Link Control Register 73, 225, 227, 234, 244, 295, 317, 336, 358, 432
Link Error Register 238, 239, 241, 245, 345, 359
Link Frequency Capability Register 282, 299, 346
Link Frequency Register 300, 301, 344, 399
Link Initialization 282, 288
Link Initialization Disconnect 225
Link Management Overview 47
Link Support Signals 29
Link Width Initialization 282, 283
Link Width Tuning 295
Low-Level Link Initialization 282, 291, 433
Low-Speed Signals 56

M
Master Host Bridge 320, 431, 432
Max_Latency Register 330
Memory Base and Limit Register 419, 485
Memory Base Upper Register 352
Memory Map 23, 312, 446
Message Semantics 21, 450
Min Grant Register 330
Minimum FIFO Size 395

N
Networking Support 17
Non-Fatal Interrupt 234, 249
Non-NXA Error 247
Non-Posted Request 66, 94, 105, 108, 125, 132, 135, 250
Non-Posted Sized Writes 42

Non-Sharing Double-Hosted Chain 432
NXA Error 251

O
Ordering Rules, HT 122

P
Packet Framing 294
Packet Rejection 265, 266
Packetized Transfers 35
PassPW Bit 45, 124, 126, 127, 128
PCI Ordering 458, 459, 460
PCI-X Ordering 467
Pin Count 31
Point-to-Point Interconnect 14
Posted Request 37, 44, 45, 58, 70, 77, 80, 81, 105, 108, 111, 125, 127, 130, 131, 136, 241
Posted Sized Writes 43
Posted Write 37, 43, 44, 45, 70, 84, 125
Prefetchable Memory Base and Limit Registers 419, 486, 487
Primary Latency Timer Register 424
Processor VID/FID 437
Producer/Consumer Model 458, 459, 472
Programmed I/O 24, 41
Programmed I/O Transactions 121
Protocol Errors 230, 237, 238, 239, 345, 350
Pseudo-Synchronous Clock Mode 399

544
Index

Pseudo-Synchronous Clocking Mode 401
PWROK 29, 49, 56, 223, 280, 281, 302, 356, 365, 378

R
Read Request 66
Read Response 36, 37, 66, 70, 94, 262
Receive Buffer Overflow 50, 230, 241
Receive FIFO 110, 282, 290, 388, 401
Receiver Flow Control Counters 108
Request EOI Field 206
Request packet 65
Request Packet Types 36
Reset 27, 29, 49, 56, 74, 110, 223, 276, 278
RESET# 29, 49, 56, 223, 247, 277, 278, 302, 304, 365, 378
Response Errors 50, 230
Response packet 66
Response Packet Types 36
Revision ID Capability Block 361
Revision ID Register 343
Routing Packets 258

S
Scalability 15, 20, 30
Scalable Bandwidth 20
Scalable Clock Speed 32
Scalable Data Width 31
Secondary Bus Non-Prefetchable Window Base 480
Secondary Bus Non-Prefetchable Window Control Register 480
Secondary Bus Prefetchable Window Control Register 480
Secondary Bus Window Base register 488
Secondary Status Register 254, 415
Sequence ID 39, 75, 124, 125
Signal Groups 54
Signaling Interrupt Acknowledge 507
Signaling INTR 506
Sized Byte Read 37, 79, 507
Sized Byte Write 37, 79
Sized Dword Read 37, 79
Sized Read 70, 75
Sized Read Transactions 41
Sized Write 75
Sized Write Transactions 42
Slave Host Bridge 320, 431
Slave/Primary Interface 332, 333
SM Request Packet Format, Downstream 220
SM Request Packet Format, Upstream 218
SM Request, Sources of 216
Source Tag 39, 94
Special Cycles 493, 521
Specification 1.04 20
Status Register 230, 307, 326, 413, 415
Stop Clock Signal 441
STOP_GRANT Cycles 438
Storage Semantics 446, 447
STPCLK# 441
STPCLK# Signaling 442
Strongly Ordered Sequence 126
## Index

<table>
<thead>
<tr>
<th></th>
<th>Page References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subtractive Decode</td>
<td>77, 82, 264, 412, 461</td>
</tr>
<tr>
<td>Sync Flood</td>
<td>230, 235, 253</td>
</tr>
<tr>
<td>Sync Pattern</td>
<td>74</td>
</tr>
<tr>
<td>Sync/Error Packet</td>
<td>74</td>
</tr>
<tr>
<td>Synchronous Clock Mode</td>
<td>389</td>
</tr>
<tr>
<td>Synchronous Clocking</td>
<td>390</td>
</tr>
<tr>
<td>System Block Diagram</td>
<td>21</td>
</tr>
<tr>
<td>System Management</td>
<td>216, 436, 493</td>
</tr>
<tr>
<td>System Management Address</td>
<td>217</td>
</tr>
<tr>
<td>System Management</td>
<td>493, 511</td>
</tr>
<tr>
<td>Transactions</td>
<td></td>
</tr>
<tr>
<td>Target Done</td>
<td>36, 37, 39, 66, 94, 97</td>
</tr>
<tr>
<td>Transaction Requests</td>
<td>40</td>
</tr>
<tr>
<td>Transaction Responses</td>
<td>40</td>
</tr>
<tr>
<td>Transaction Stream</td>
<td>39, 125, 137</td>
</tr>
<tr>
<td>Transaction Types</td>
<td>41</td>
</tr>
<tr>
<td>Transmit and Receive</td>
<td>110</td>
</tr>
<tr>
<td>Transmitter Flow Control</td>
<td>109</td>
</tr>
<tr>
<td>Tunnel</td>
<td>20, 33, 105, 125, 244, 271, 300, 321, 333</td>
</tr>
<tr>
<td>Type 0 Configuration Cycle</td>
<td>267, 310, 314, 315, 434</td>
</tr>
<tr>
<td>Type 1 Configuration Cycle</td>
<td>309, 314, 432</td>
</tr>
<tr>
<td>Virtual Channel Buffers</td>
<td>38</td>
</tr>
<tr>
<td>Virtual Channel</td>
<td>37, 38, 69, 105, 125, 447, 451</td>
</tr>
<tr>
<td>Virtual Channel Buffers</td>
<td>38</td>
</tr>
<tr>
<td>Wakeup Signaling</td>
<td>439, 440</td>
</tr>
<tr>
<td>Warm Reset</td>
<td>302</td>
</tr>
<tr>
<td>Write Request</td>
<td>42, 43, 66, 75</td>
</tr>
<tr>
<td>X86 Power Management</td>
<td>441</td>
</tr>
<tr>
<td>X86-Based Interrupt Request Message Format</td>
<td>502</td>
</tr>
<tr>
<td>X86-Based Interrupt Request Message Format</td>
<td>502</td>
</tr>
</tbody>
</table>

## T

<table>
<thead>
<tr>
<th></th>
<th>Page References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target Done</td>
<td>36, 37, 39, 66, 94, 97</td>
</tr>
<tr>
<td>Transaction Requests</td>
<td>40</td>
</tr>
<tr>
<td>Transaction Responses</td>
<td>40</td>
</tr>
<tr>
<td>Transaction Stream</td>
<td>39, 125, 137</td>
</tr>
<tr>
<td>Transaction Types</td>
<td>41</td>
</tr>
<tr>
<td>Transmit and Receive</td>
<td>110</td>
</tr>
<tr>
<td>Transmitter Flow Control</td>
<td>109</td>
</tr>
<tr>
<td>Tunnel</td>
<td>20, 33, 105, 125, 244, 271, 300, 321, 333</td>
</tr>
<tr>
<td>Type 0 Configuration Cycle</td>
<td>267, 310, 314, 315, 434</td>
</tr>
<tr>
<td>Type 1 Configuration Cycle</td>
<td>309, 314, 432</td>
</tr>
</tbody>
</table>

## U

<table>
<thead>
<tr>
<th></th>
<th>Page References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit ID</td>
<td>77, 94, 125, 154, 179, 243, 268, 316</td>
</tr>
<tr>
<td>Upstream Ordering Rules</td>
<td>122, 123, 125</td>
</tr>
</tbody>
</table>