Copyright

Copyright © 2000 KnowledgeTek, Inc.

All rights reserved. No part of this document may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from KnowledgeTek, Inc.
KnowledgeTek, Inc.

KnowledgeTek specializes in technical training on computer peripherals and related technologies. We offer classes in:

- Disk Drive Technology
- The Cutting Edge
- Disk Drive Servo
- PRML Read Channels
- PRML Lab
- Tape Storage Technology
- Media Noise
- Mag Recording Write Process
- DVD Technology
- The Head/Disk Interface: Advanced Tribology
- SCSI, The Nuts and Bolts
- High Speed SCSI
- IDE, The Nuts and Bolts
- ATAPI
- Fibre Channel
- 1394 Serial Bus
- PCMCIA (PC Card)
- Data Storage Interfaces
- Introduction to SCSI

We can be reached at:

KnowledgeTek, Inc.
7085 West 119th Place
Broomfield, CO 80020
Voice: (303) 465-1800
Fax: (303) 465-2600
email: seminars@KnowledgeTek.com
Web Site: www.knowledgetek.com
Disclaimer

This material is not a specification for any interface or system.

KnowledgeTek makes no warranty and assumes no liability arising out of the application or use of any product, hardware, software, system, circuit, or anything else described herein.

KnowledgeTek assumes no responsibility for errors appearing in this document.

KnowledgeTek assumes no responsibility for any claims that the concepts or details discussed in the seminar, or disclosed in the course materials, are proprietary to any person or company.

Company, brand, product, trade, and other names used in this document are trademarks of their respective holders.

Firewire is a registered trademark of Apple Computer, Inc.

Seminar participants are urged to clear any designs proposed for products with their patent and copyright council.
Trademarks

The 1394 symbol is a registered trademark of Apple Computer, Inc. and used with permission.
About The Instructor

Hugh Curley began working on mainframe computers in 1967 and expanded to personal computers in 1981. His background includes hands-on technical and managerial experience in field service, system-level test in manufacturing, and system-level test in engineering. In 1975 Hugh began teaching computers to engineers and discovered that he not only had good skills for the classroom process, but that he enjoyed teaching working engineers. Now, Hugh has accumulated extensive experience in developing and presenting highly technical courses to engineering specialists from different disciplines. He applies that experience and skill to every course he presents.

Hugh's content strengths are in data storage interfaces, which include IDE and ATAPI, but has particular skill and interest in SCSI. He has successfully presented many interface courses for us.

Current interests have led Hugh to represent KnowledgeTek as an active participant in the 1394A Committee and an interested attendee in other 1394 committees. He is also a member of the IEEE.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Volume 1</strong></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1394 Overview</td>
</tr>
<tr>
<td>2</td>
<td>1394 Asynchronous Operations</td>
</tr>
<tr>
<td>3</td>
<td>Control and Status Registers</td>
</tr>
<tr>
<td>4</td>
<td>Introduction To SCSI Over 1394</td>
</tr>
<tr>
<td>5</td>
<td>Serial Bus Protocol SBP-2</td>
</tr>
<tr>
<td>6</td>
<td>SCSI Over SBP-2</td>
</tr>
<tr>
<td>7</td>
<td>ATA Over SBP-2</td>
</tr>
<tr>
<td><strong>Volume 2</strong></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Physical</td>
</tr>
<tr>
<td>9</td>
<td>Arbitration</td>
</tr>
<tr>
<td>10</td>
<td>Isochronous Operations</td>
</tr>
<tr>
<td>11</td>
<td>Configuration</td>
</tr>
<tr>
<td>12</td>
<td>Bus Management</td>
</tr>
<tr>
<td>13</td>
<td>Implementation</td>
</tr>
<tr>
<td>14</td>
<td>Power Management</td>
</tr>
<tr>
<td>15</td>
<td>1394 Specifications</td>
</tr>
<tr>
<td>16</td>
<td>Audio/Video</td>
</tr>
<tr>
<td>17</td>
<td>1394b</td>
</tr>
</tbody>
</table>
### Table of Contents-Appendices

<table>
<thead>
<tr>
<th>Appendix</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>IEEE 1394-1995 Table of Contents</td>
</tr>
<tr>
<td>C</td>
<td>Isochronous Connection Management</td>
</tr>
<tr>
<td>D</td>
<td>Device Bay</td>
</tr>
<tr>
<td>R</td>
<td>Reduced Block Command Set</td>
</tr>
<tr>
<td>Z</td>
<td>Answers</td>
</tr>
</tbody>
</table>
Section 1

1394 Overview
Subjects Covered

Parallel vs Serial
Benefits of 1394
Packets
Isochronous
Other Serial Interfaces
Parallel Interfaces

Data

strobe

Termination
Timing Skew
Driver per Bit Wide
Many Signals Change Simultaneously (EMI & Power)
Expensive & Bulky Cables
Expensive & Bulky Connectors
Serial Interfaces

One **Very** Fast Driver rather than Many Fast Drivers

Point to Point Links
  Each Link Terminated
  Flexible Cabling

Great Out-of-Cabinet Solution
  Less RFI
  Cheaper Cables and Connectors

Can Be Made Self-Configuring
1394 Serial Bus

Targeted at Consumer Market

Low Cost
Unrestricted Cabling*
Supplies Power Over Cable
Self-Configuring
Multimedia (Scheduled Data Flow)
Device-Type Independent
High Speed Data Transfer
Enabling Protocol for Device Bay

*no loops
## 1394 Serial Bus Speeds

<table>
<thead>
<tr>
<th>Speed Code</th>
<th>Speed</th>
<th>Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>S100</td>
<td>100 Mbits/sec</td>
<td>(12.5 Mbytes/sec)</td>
</tr>
<tr>
<td>S200</td>
<td>200 Mbits/sec</td>
<td>(25 Mbytes/sec)</td>
</tr>
<tr>
<td>S400</td>
<td>400 Mbits/sec</td>
<td>(50 Mbytes/sec)</td>
</tr>
<tr>
<td>S800</td>
<td>800 Mbits/sec</td>
<td>(100 Mbytes/sec)</td>
</tr>
<tr>
<td>S1600</td>
<td>1600 Mbits/sec</td>
<td>(200 Mbytes/sec)</td>
</tr>
</tbody>
</table>
Using 1394 Serial Bus - Physical

1394 port

Stereo Interface

CPU

1394 cable
4.5 meters max

HDD

CD ROM

Digital Camera

Scanner

Printer

© KnowledgeTek, Inc.
rev 5.41b 15.H.01

Sect 1: Overview
Using 1394 Serial Bus - Logical
Serial Implementation of a Microprocessor Bus

- 1394 = 2 pair = 4
  - 1 Data line pair
  - 1 Clock line pair

ISA = 98 lines
- 16 Data lines
- 24 Address lines
- 45 Control lines

- 1394

Sect 1: Overview
Parallel Microprocessor Bus

ADDRESS
1FC0h
3C01h

CONTROL
Write
Write

DATA
3Eh
2Bh
Serial Microprocessor Bus

Communication performed with Packet

Each Packet contains its own control and address information
Isochronous

Same Time

Data delivered at a constant rate

Cycle Start packet every $125\mu$Sec triggers Isochronous Packets:

- 1200 Bytes from Camera to Disk = 10MB/sec
- 20 Bytes from CD to Speakers = 160KB/sec

Non-Isochronous (Asynchronous) Packets
Other Serial Interfaces - Fibre Channel

1 & 2 Gbps (200 MByte per second)
Much more expensive per node

Supported Topologies
- Arbitrated Loop
- Fabric (requires switch hardware)
- Point-to-Point

Can go long distances
Designed for High Performance, Cost is Secondary Applications
Other Serial Interfaces - USB

Designed for Lower Speed, Cost is Everything Applications
  Keyboard, Mouse, Phone, Printer, etc.
12 Mbps maximum
  Not fast enough for Mass Storage or Video
6 meters per segment maximum
  Everything controlled directly by PC

USB 2.0 (under development)
  will be 30X to 40X faster
# Reference - Number System Conversions

<table>
<thead>
<tr>
<th>Prefix</th>
<th>$10^m$</th>
<th>$2^n$</th>
<th>Decimal</th>
<th>Binary</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exa</td>
<td>18</td>
<td>60</td>
<td>0</td>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>Peta</td>
<td>15</td>
<td>50</td>
<td>1</td>
<td>0001</td>
<td>1</td>
</tr>
<tr>
<td>Tera</td>
<td>12</td>
<td>40</td>
<td>2</td>
<td>0010</td>
<td>2</td>
</tr>
<tr>
<td>Giga</td>
<td>9</td>
<td>30</td>
<td>3</td>
<td>0011</td>
<td>3</td>
</tr>
<tr>
<td>Mega</td>
<td>6</td>
<td>20</td>
<td>4</td>
<td>0100</td>
<td>4</td>
</tr>
<tr>
<td>Kilo</td>
<td>3</td>
<td>10</td>
<td>5</td>
<td>0101</td>
<td>5</td>
</tr>
<tr>
<td>Hecta</td>
<td>2</td>
<td></td>
<td>6</td>
<td>0110</td>
<td>6</td>
</tr>
<tr>
<td>Deca</td>
<td>1</td>
<td></td>
<td>7</td>
<td>0111</td>
<td>7</td>
</tr>
<tr>
<td>Unity</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>Decia</td>
<td>-1</td>
<td></td>
<td>9</td>
<td>1001</td>
<td>9</td>
</tr>
<tr>
<td>Centi</td>
<td>-2</td>
<td></td>
<td>10</td>
<td>1010</td>
<td>A</td>
</tr>
<tr>
<td>Milli</td>
<td>-3</td>
<td>-10</td>
<td>11</td>
<td>1011</td>
<td>B</td>
</tr>
<tr>
<td>Micro</td>
<td>-6</td>
<td>-20</td>
<td>12</td>
<td>1100</td>
<td>C</td>
</tr>
<tr>
<td>Nano</td>
<td>-9</td>
<td>-30</td>
<td>13</td>
<td>1101</td>
<td>D</td>
</tr>
<tr>
<td>Pico</td>
<td>-12</td>
<td>-40</td>
<td>14</td>
<td>1110</td>
<td>E</td>
</tr>
<tr>
<td>Femto</td>
<td>-15</td>
<td>-50</td>
<td>15</td>
<td>1111</td>
<td>F</td>
</tr>
<tr>
<td>Atto</td>
<td>-18</td>
<td>-60</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Review

1. What are the benefits or target market for 1394?
2. What speeds does 1394 operate?
3. What does a 1394 packet contain?
4. What is Isochronous and how does it operate?
Section 2

1394 Asynchronous Transactions
Subjects Covered

Asynchronous Transactions
  Read
  Write
  Lock
Acknowledges
Requests and Responses
Unified and Split Transactions
Addressing
Busy Retry
Asynchronous Transactions

Basic Read and Write Functions
Not synchronized to time
Normally does not include audio, video, multimedia
Accuracy of data delivery is more critical than timing
Write Transaction

How does node A know if the packet is received correctly?
Complete Write Transaction

1st Write Request Packet

Node A

Node B

Serial Bus

Write Req

Address

Data

Ack Comp

Acknowledge Complete Packet

40-50 ns to send ACK

Ack contiguous in time after write packet

Sect 2: Asynchronous Transactions
Acknowledges

Contiguous on the bus following the write data

Several Types:

- **Complete**: Operation completed satisfactorily
- **Busy (A/B/X)**: Operation not completed, receiver node busy
- **Error (data/type)**: Operation not completed, there was an error (address/conflict)
- **Pending**: Operation not completed, but is being processed
- **Tardy**: Node will take a while to respond (in low power state)

Each type will be described in detail later.

What about a read that requires a seek or search, and the data is delayed?
Read Operation

This is referred to as Split Response

all reads are

Sect 2: Asynchronous Transactions
Non-Unified Write Operation
Split Response

Node A

Node B

Serial Bus

Write Req
Address
Data

Ack Pend

Ack Comp

Write Resp

Sect 2: Asynchronous Transactions
Coherence Problem

A register on Node B indicates who is going to perform a given function. The register is initialized to 3Fh indicating no one has been assigned. Both A and C want to perform this function:

Node A: A reads register, 3Fh = No One Assigned

Node B: A writes its ID to the register

Node C: Both A and C think they are doing the job

Node C: C reads register, 3Fh = No One Assigned

C writes its ID to the register
Coherence Solution - Lock Transaction

Makes testing a flag and setting it one action

Required because of split response nature of 1394

Basic Functions:

**Compare and Swap**
- Mask and Swap
- Fetch and Add
- Little Add
- Bounded add
- Wrap Add

Not used by 1394

Used to communicate with some CSRs (section 3)
Lock Transaction

Asks Node B to:
- Check location Address
- Compare to Argument
- If the same, replace with Data
Lock Response

Node A

Lock Req
Comp & Swap
Address
Argument
Data

Node B

Ack Pend

Ack Comp

Lock Resp
Comp & Swap
Data

Data before Compare and Swap

Sect 2: Asynchronous Transactions
Solving Coherence Problems With Lock Transactions

Node A:
- Lock Req to B
  - Addr = Register Loc
  - Arg = 3F
  - Data = Node A ID
- Lock Resp to A
  - Data = 3F
  - Indicates previously unassigned
  - Node A performs the function

Node C:
- Lock Req to B
  - Addr = Register Loc
  - Arg = 3F
  - Data = Node C ID
- Lock Resp to C
  - Data = Node A ID
  - Indicates Node A performs the function
For More Information: Other Lock Functions

Mask & Swap  Set all bits that are ‘1’ in data
             Clear all bits that are ‘0’ in argument
Fetch & Add  Add data
Bounded Add  If memory ≠ argument, add data
Add Little   Fetch & Add but in little-endian order
Wrap & Add   If memory ≠ argument, add data
             otherwise set to data
For More Information: Lock Functions in C

<table>
<thead>
<tr>
<th>Function</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compare &amp; Swap</td>
<td>if (old_value == arg_value)</td>
</tr>
<tr>
<td></td>
<td>new_value = data_value;</td>
</tr>
<tr>
<td></td>
<td>else new_value = old_value;</td>
</tr>
<tr>
<td>Mask &amp; Swap</td>
<td>new_value = data_value</td>
</tr>
<tr>
<td>Fetch &amp; Add</td>
<td>new_value = old_value + data_value;</td>
</tr>
<tr>
<td>Bounded Add</td>
<td>if (old_value != arg_value)</td>
</tr>
<tr>
<td></td>
<td>new_value = old_value + data value;</td>
</tr>
<tr>
<td></td>
<td>else new_value = old_value;</td>
</tr>
<tr>
<td>Little Add</td>
<td>new_value = LittleEndAdd (old_value, data_value);</td>
</tr>
<tr>
<td>Wrap &amp; Add</td>
<td>if (old_value != arg_value)</td>
</tr>
<tr>
<td></td>
<td>new_value = old_value + data_value;</td>
</tr>
<tr>
<td></td>
<td>else new_value = data_value;</td>
</tr>
</tbody>
</table>
Addressing

64 bit Addresses

$2^{64} = 16$ ExaBytes Addressed

Each Node has $2^{48}$ Bytes = 256 TeraBytes of Address Range
Nodes And Busses

Bus #0
node 0  node 1  node 2  ...  node n_1  Bridge

Bus #1
node 0  node 1  node 2  ...  node n_2  Bridge

Bus #2
node 0  node 1  node 2  ...  node n_3  Bridge

Sect 2: Asynchronous Transactions
1023 Buses maximum (0 - 1022)
63 Nodes on a Bus maximum (0 - 62)

Bus 3FFh is local bus

Node 3Fh broadcasts to all nodes on indicated bus
 Addresses are shown from top - down throughout this course
Addressing - Different view

Node A

Node Address

48 bits

Node ID

16 bits

Memory Space

Node B

Node Address

48 bits

Node ID

16 bits
Node Memory Space

1394 does not use private space
# Size Notations

<table>
<thead>
<tr>
<th>Size in bits</th>
<th>16 bit word machine notation</th>
<th>32 bit word machine notation</th>
<th>IEEE standard notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Byte</td>
<td>Byte</td>
<td>Byte</td>
</tr>
<tr>
<td>16</td>
<td>Word</td>
<td>Half-word</td>
<td>Doublet</td>
</tr>
<tr>
<td>32</td>
<td>Long-word</td>
<td>Word</td>
<td>Quadlet</td>
</tr>
<tr>
<td>64</td>
<td>Quad-word</td>
<td>Double Word</td>
<td>Octlet</td>
</tr>
</tbody>
</table>

- Octet
- Used in this course
## Byte Ordering

<table>
<thead>
<tr>
<th>Quadlet 0</th>
<th>Quadlet 1</th>
<th>Quadlet 2</th>
<th>Quadlet m-1</th>
<th>Quadlet m</th>
</tr>
</thead>
<tbody>
<tr>
<td>msb</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Byte 0</td>
<td>Byte 4</td>
<td>Byte 8</td>
<td>Byte n-7</td>
<td>Byte n-3</td>
</tr>
<tr>
<td>Byte 1</td>
<td>Byte 5</td>
<td>Byte 9</td>
<td>Byte n-6</td>
<td>Byte n-2</td>
</tr>
<tr>
<td>Byte 2</td>
<td>Byte 6</td>
<td>Byte 10</td>
<td>Byte n-5</td>
<td>Byte n-1</td>
</tr>
<tr>
<td>Byte 3</td>
<td>Byte 7</td>
<td>Byte 11</td>
<td>Byte n-4</td>
<td>Byte n</td>
</tr>
</tbody>
</table>

Transmitted first

m = \( \frac{n-3}{4} \)

Transmitted last
Bit and Byte Ordering

Bit Ordering
msb

Bytes in a Quadlet
msB

Quadlets in an Octlet
msb

transmitted
first
last

1394
Generic Packet Format

transmitted first

- Destination ID
- Trans Specific
- tc ode
- Pri
- Source ID
- Transaction Specific
- Transaction Specific
- Header CRC
- Data
- Data CRC

transmitted last

Sect 2: Asynchronous Transactions
## Generic Packet Format Definitions

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination ID</td>
<td>High order 16 bits of address designating receiving node.</td>
</tr>
<tr>
<td>Source ID</td>
<td>High order 16 bits of sending node</td>
</tr>
<tr>
<td>tcode</td>
<td>Transaction code, identifies this as read, write, etc.</td>
</tr>
<tr>
<td>Pri</td>
<td>Priority, meaningful on backplane implementations only</td>
</tr>
<tr>
<td>Header CRC</td>
<td>32 bit Cyclic Redundancy Check for header quadlets.</td>
</tr>
<tr>
<td>Data CRC</td>
<td>32 bit Cyclic Redundancy Check for data quadlets.</td>
</tr>
</tbody>
</table>

V     CRC is the same CRC used by IEEE 802 and FDDI.
Transaction Codes (tcode)

<table>
<thead>
<tr>
<th>tcode</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0h</td>
</tr>
<tr>
<td>0001</td>
<td>1h</td>
</tr>
<tr>
<td>0010</td>
<td>2h</td>
</tr>
<tr>
<td>0011</td>
<td>3h</td>
</tr>
<tr>
<td>0100</td>
<td>4h</td>
</tr>
<tr>
<td>0101</td>
<td>5h</td>
</tr>
<tr>
<td>0110</td>
<td>6h</td>
</tr>
<tr>
<td>0111</td>
<td>7h</td>
</tr>
<tr>
<td>1000</td>
<td>8h</td>
</tr>
<tr>
<td>1001</td>
<td>9h</td>
</tr>
<tr>
<td>1010</td>
<td>Ah</td>
</tr>
<tr>
<td>1011</td>
<td>Bh</td>
</tr>
<tr>
<td>1100</td>
<td>Ch</td>
</tr>
<tr>
<td>1101</td>
<td>Dh</td>
</tr>
<tr>
<td>1110</td>
<td>Eh</td>
</tr>
<tr>
<td>1111</td>
<td>Fh</td>
</tr>
</tbody>
</table>

Sect 2: Asynchronous Transactions
this page is intentionally blank
Block Write Request Packet Format

- Transmitted first:
  - Destination ID
  - Source ID
  - Destination Memory Address (hi)
  - Destination Memory Address (lo)
  - Data Length in bytes
  - Header CRC
  - Data
  - Data CRC

- Transmitted last:
  - Pad to quadlet length

Pad zero bytes on end of data block if needed
Write Request Packet Format

Destination ID  16 bit ID of receiving node
Source ID  16 bit ID of sending node
tl  Transaction Label - defined later
rt  Retry Code - defined later
tcode = 1  Transaction code, identifies this packet as block write request
Pri  Priority, only meaningful on backplane
CRC  Check data for header or data (including pad bytes)
Data Length  Number of bytes in data field (does not include pad bytes)
## Block Read Request Packet Format

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination ID</td>
<td>ID of the device or module requesting the block</td>
</tr>
<tr>
<td>Source ID</td>
<td>ID of the source module that provides the block</td>
</tr>
<tr>
<td>Destination Memory Address (hi)</td>
<td>Address of the high-order byte of the destination memory address</td>
</tr>
<tr>
<td>Destination Memory Address (lo)</td>
<td>Address of the low-order byte of the destination memory address</td>
</tr>
<tr>
<td>Data Length</td>
<td>Number of bytes to be transferred</td>
</tr>
<tr>
<td>Header CRC</td>
<td>Cyclic Redundancy Check Code for the packet</td>
</tr>
</tbody>
</table>

**Transmit Order:**
- Destination ID
- Source ID
- Destination Memory Address (hi)
- Destination Memory Address (lo)
- Data Length
- Header CRC

**Notes:**
- The block read request packet is transmitted first.
- The header CRC is transmitted last.

---

© KnowledgeTek, Inc.

rev 5.41d 15.H.08

Sect 2: Asynchronous Transactions

---

1394
### Read Request for Data Block Packet Definitions

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination ID</td>
<td>Node ID of receiving node</td>
</tr>
<tr>
<td>Source ID</td>
<td>Node ID of requesting node</td>
</tr>
<tr>
<td>tl</td>
<td>Transaction label</td>
</tr>
<tr>
<td>rt</td>
<td>Retry code</td>
</tr>
<tr>
<td>tcode = 5</td>
<td>Transaction code, 5 = Read Block Request</td>
</tr>
<tr>
<td>Pri</td>
<td>Priority, for backplane environment</td>
</tr>
<tr>
<td>Memory address</td>
<td>48 bit address within the node. This</td>
</tr>
<tr>
<td></td>
<td>concatenated with the Destination ID is</td>
</tr>
<tr>
<td></td>
<td>the 64 bit system address.</td>
</tr>
<tr>
<td>Data Length</td>
<td>Length of expected data in bytes</td>
</tr>
</tbody>
</table>
**Block Read Response Packet Format**

Transmitted first:

<table>
<thead>
<tr>
<th>Destination ID</th>
<th>tl</th>
<th>rt</th>
<th>tcode=7</th>
<th>Pri</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source ID</td>
<td>rcode</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Length</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Header CRC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data CRC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Transmitted last:

Pad zero bytes on end of data block if needed

Sect 2: Asynchronous Transactions
## Read Response for Data Block Packet Definitions

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination ID</td>
<td>Node ID of receiving node</td>
</tr>
<tr>
<td>Source ID</td>
<td>Node ID of requesting node</td>
</tr>
<tr>
<td>tl</td>
<td>Transaction label</td>
</tr>
<tr>
<td>rt</td>
<td>Retry code</td>
</tr>
<tr>
<td>tcode = 7</td>
<td>Transaction code, 7 = Read Block Response</td>
</tr>
<tr>
<td>Pri</td>
<td>Priority, for backplane environment</td>
</tr>
<tr>
<td>rcode</td>
<td>Response Code - described later</td>
</tr>
<tr>
<td>Data Length</td>
<td>Length of data</td>
</tr>
<tr>
<td>Header CRC</td>
<td>32 bit CRC check for header information</td>
</tr>
<tr>
<td>Data field</td>
<td>Data payload</td>
</tr>
<tr>
<td>Data CRC</td>
<td>32 bit CRC check for data field</td>
</tr>
</tbody>
</table>
Write Response Packet Format

transmitted first

<table>
<thead>
<tr>
<th>Destination ID</th>
<th>tl</th>
<th>rt</th>
<th>tcode=2</th>
<th>Pri</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source ID</td>
<td>rcode</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Header CRC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

transmitted last
Write Response Packet Format Definitions

- **Destination ID**: 16 bit ID of receiving node
- **Source ID**: 16 bit ID of sending node
- **tl**: Transaction Label
- **rt**: Retry Code
- **tcode = 2**: Transaction code, 2 = write response
- **Pri**: Priority, only meaningful on backplane
- **rcode**: Response Code
- **Header CRC**: 32 bit Cyclic Redundancy Check for header quadlets.
### Response Codes (rcode)

<table>
<thead>
<tr>
<th>rcode</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0h Transaction completed successfully</td>
</tr>
<tr>
<td>0001</td>
<td>1h Reserved</td>
</tr>
<tr>
<td>0010</td>
<td>2h Reserved</td>
</tr>
<tr>
<td>0011</td>
<td>3h Reserved</td>
</tr>
<tr>
<td>0100</td>
<td>4h Resource conflict (retry)</td>
</tr>
<tr>
<td>0101</td>
<td>5h Hardware data error (data not available)</td>
</tr>
<tr>
<td>0110</td>
<td>6h Illegal request (invalid operation or unsupported value)</td>
</tr>
<tr>
<td>0111</td>
<td>7h Unavailable Address</td>
</tr>
<tr>
<td>1000</td>
<td>8h Reserved</td>
</tr>
<tr>
<td>1111</td>
<td>Fh Reserved</td>
</tr>
</tbody>
</table>
Exercise: What's Happening?

Did this work as expected?
Write 8 bytes to 000003C4D00h @ node 1
Read 5 bytes from 0000003C4D01h

Sect 2: Asynchronous Transactions
Which Data Is This Anyway?

To which request does the data belong?

1394
Transaction Labels (tl)

6 bit field
Unique for each outstanding operation between a pair of nodes
Sent in Request packet
Returned in corresponding Response packet
Requester uses it to match Response to Request
Transaction Labels

Node A

Read Req (tl=0)
Address x

Ack Pend

Node B

Read Req (tl=9)
Address y

Ack Pend

Read Resp (tl=9)
Data

Ack Comp
Single Data Quadlet Packets

Data being read/written is always 1 quadlet (32 bits)
No Data Length field
Single CRC for Header & Data
Required for certain register operations
Well suited to "Virtual Registers" implemented by microprocessor
Write Request for Single Data Quadlet

transmitted first

<table>
<thead>
<tr>
<th>Destination ID</th>
<th>tl</th>
<th>rt</th>
<th>tcode=0</th>
<th>Pri</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source ID</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Destination Memory Address (hi)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Destination Memory Address (lo)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quadlet Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data/Header CRC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

transmitted last
Read Request for Single Data Quadlet

transmitted first

- Destination ID
- Source ID
- Destination Memory Address (hi)
- Destination Memory Address (lo)
- Header CRC

transmitted last
Read Response for Single Data Quadlet

transmitted first

<table>
<thead>
<tr>
<th>Destination ID</th>
<th>tl</th>
<th>rt</th>
<th>tcode=6</th>
<th>Pri</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source ID</td>
<td>rcode</td>
<td>Reserved</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quadlet Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data/Header CRC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

transmitted last

© KnowledgeTek, Inc.
rev 5.41d 15.H.08

Sect 2: Asynchronous Transactions
Lock-Request Packet Format

<table>
<thead>
<tr>
<th>Field</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination ID</td>
<td>tl</td>
</tr>
<tr>
<td>Source ID</td>
<td>rt</td>
</tr>
<tr>
<td>tcode=9</td>
<td>Pri</td>
</tr>
<tr>
<td>Destination Memory Address (hi)</td>
<td></td>
</tr>
<tr>
<td>Destination Memory Address (lo)</td>
<td></td>
</tr>
<tr>
<td>Data Length</td>
<td>Extended tcode</td>
</tr>
<tr>
<td>Header CRC</td>
<td></td>
</tr>
<tr>
<td>Argument Value</td>
<td>0, 4, or 8 bytes</td>
</tr>
<tr>
<td>Data</td>
<td>0, 4, or 8 bytes</td>
</tr>
<tr>
<td>Data CRC</td>
<td></td>
</tr>
</tbody>
</table>

transmitted first

transmitted last
Lock Packet Definitions

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination ID</td>
<td>16 bit address of receiving node</td>
</tr>
<tr>
<td>Source ID</td>
<td>16 bit address of sending node</td>
</tr>
<tr>
<td>tl</td>
<td>Transaction label</td>
</tr>
<tr>
<td>rt</td>
<td>Retry Code</td>
</tr>
<tr>
<td>tcode = 9</td>
<td>Transaction Code, 9 = Lock request</td>
</tr>
<tr>
<td>Pri</td>
<td>Priority, valid only in backplane environment</td>
</tr>
<tr>
<td>Data Length</td>
<td>Quantity of bytes in Argument Value and Data Fields</td>
</tr>
<tr>
<td>Extended tcode</td>
<td>Identifies the lock subcommand, 2 = compare and swap</td>
</tr>
<tr>
<td>Argument value</td>
<td>Data to compare with memory data</td>
</tr>
<tr>
<td>Data</td>
<td>Contents to write to memory if compare is successful</td>
</tr>
</tbody>
</table>
## Extended tcode Function

<table>
<thead>
<tr>
<th>Extended tcode</th>
<th>Function</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1h</td>
<td>MASK_SWAP</td>
<td>new_value = data_value</td>
</tr>
<tr>
<td>2h</td>
<td>COMPARE_SWAP</td>
<td>if (old_value == arg_value)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>new_value = data_value;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>else new_value = old_value;</td>
</tr>
<tr>
<td>3h</td>
<td>FETCH_ADD</td>
<td>new_value = old_value + data_value;</td>
</tr>
<tr>
<td>4h</td>
<td>LITTLE_ADD</td>
<td>new_value = LittleEndAdd (old_value, data_value);</td>
</tr>
<tr>
<td></td>
<td>(little endian)</td>
<td></td>
</tr>
<tr>
<td>5h</td>
<td>BOUNDED_ADD</td>
<td>if (old_value != arg_value)</td>
</tr>
<tr>
<td></td>
<td>(unequal add)</td>
<td>new_value = old_value + data_value;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>else new_value = old_value;</td>
</tr>
<tr>
<td>6h</td>
<td>WRAP_ADD</td>
<td>if (old_value != arg_value)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>new_value = old_value + data_value;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>else new_value = data_value;</td>
</tr>
<tr>
<td>7h</td>
<td>Vendor specific</td>
<td></td>
</tr>
</tbody>
</table>
For More Information: **Lock Transaction Data Length Parameter**

Data Length = Argument Value Length + Data Value Length

Only Data Lengths of 4, 8, and 16 Bytes supported

Argument Value Length and Data Value Length Depend on Function

<table>
<thead>
<tr>
<th>Data Length (Bytes)</th>
<th>Function (extended tcode)</th>
<th>Data Value Length (Bytes)</th>
<th>Arg Value Length (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>FETCH_ADD, LITTLE_ADD</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>MASK_SWAP, COMPARE_SWAP, BOUNDED_ADD, WRAP_ADD</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>FETCH_ADD, LITTLE-ADD (64 bit)</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>MASK_SWAP, COMPARE_SWAP, BOUNDED_ADD, WRAP_ADD</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>
Lock-Response Packet Format

- Transmitted first:
  - Destination ID
  - Source ID
  - Reserved
  - Data Length
  - Extended tcode
  - Header CRC
  - Old Value
  - Data CRC

- Transmitted last:
  - tl
  - rt
  - tcode=B
  - Pri
Lock Response Pack Format Definitions

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination ID</td>
<td>High order 16 bits of address designating receiving node.</td>
</tr>
<tr>
<td>Source ID</td>
<td>High order 16 bits of sending node</td>
</tr>
<tr>
<td>tl</td>
<td>Transaction label</td>
</tr>
<tr>
<td>rt</td>
<td>Retry Code</td>
</tr>
<tr>
<td>tcode = B</td>
<td>Transaction code, B = LOCK RESPONSE</td>
</tr>
<tr>
<td>Pri</td>
<td>Priority, meaningful on backplane implementations only</td>
</tr>
<tr>
<td>rcode</td>
<td>Response Code</td>
</tr>
<tr>
<td>Data Length</td>
<td>Number of bytes in data field</td>
</tr>
<tr>
<td>Extended tcode</td>
<td>Specific lock function, 2 = compare and swap</td>
</tr>
<tr>
<td>Header CRC</td>
<td>32 bit Cyclic Redundancy Check for header quadlets.</td>
</tr>
<tr>
<td>Data CRC</td>
<td>32 bit Cyclic Redundancy Check for data quadlets.</td>
</tr>
<tr>
<td>Old Value</td>
<td>Data that was in referenced memory location of the selected node prior to</td>
</tr>
<tr>
<td></td>
<td>lock operation</td>
</tr>
</tbody>
</table>

Sect 2: Asynchronous Transactions
What If Packets Arrive Faster Than You Can Handle Them?

Can only process 1 packet at a time

Node A  Node B  Node C

Read Req
Address

Ack Pend

Working on Request from C

Read Req
Address

Ack Busy X

What do you do when bounced with a busy?

Sect 2: Asynchronous Transactions
Simple Retry

Node A

Node B

Working on Request for C

Node C

Read Req Address

ACK Busy X

Read Req Address

ACK Busy X

Read Resp Data

to C

Read Req Address

Ack Pend

Read Resp Data

Ack COMP

Sect 2: Asynchronous Transactions
Busy Options

Queue Packets
Still need Busy for when Queue is full

Single Phase Retry
When packet can’t be processed - return ACK BUSY X
Scheme on previous page
Simple to implement
High Priority devices hog the busy node

Dual Phase Retry
Fairness mode - make sure all bounced devices get a chance
Busy Retry Management

<table>
<thead>
<tr>
<th>Requester</th>
<th>Responder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retry Codes</td>
<td>ACK Busy Codes</td>
</tr>
<tr>
<td>Retry_1</td>
<td>ACK Busy_X</td>
</tr>
<tr>
<td>Retry_X</td>
<td>ACK Busy_A</td>
</tr>
<tr>
<td>Retry_A</td>
<td>ACK Busy_B</td>
</tr>
<tr>
<td>Retry_B</td>
<td></td>
</tr>
</tbody>
</table>

Single phase equipment uses only Retry X and Busy X
## Retry Code (rt)

<table>
<thead>
<tr>
<th>rt code</th>
<th>name</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 h</td>
<td>Retry_1</td>
<td>Reservation Requested</td>
</tr>
<tr>
<td>0 1 1 h</td>
<td>Retry_X</td>
<td>No Reservation Requested</td>
</tr>
<tr>
<td>1 0 2 h</td>
<td>Retry_A</td>
<td>Used on next retry after ACK Busy_A Only used in dual phase retry</td>
</tr>
<tr>
<td>1 1 3 h</td>
<td>Retry_B</td>
<td>Used on next retry after ACK Busy_B Only used in dual phase retry</td>
</tr>
</tbody>
</table>
Dual Phase Retry Overview

Triggered by “bouncing” a packet twice

Packet receiver goes into a special mode
   Divides all senders into two groups (A and B)
   Only works on one group (A or B) at a time
   Keeps servicing a group until it’s empty
   All new packets are always put in the other group (B or A)

Nobody keeps count or keeps track of the groups
   Accomplished by ACK and Retry codes
## Dual Phase Retry Management

<table>
<thead>
<tr>
<th>Outbound Device (Requester)</th>
<th>Inbound Device (Responder)</th>
</tr>
</thead>
<tbody>
<tr>
<td>All new packets coded Retry_X</td>
<td>If busy - respond ACK Busy_X</td>
</tr>
<tr>
<td>Re-send one packet with Retry_1</td>
<td>If still busy - respond ACK Busy_A</td>
</tr>
<tr>
<td>Go to service Retry_A only mode</td>
<td></td>
</tr>
</tbody>
</table>

Those in line stay in line

New packets get in other line

Repeat above but for B group

- Retry_A get ACK Busy_A
- Retry_B get ACK Busy_B
- Retry_X get ACK Busy_X
- Retry_1 get ACK Busy_B
- Service only Retry_A packets
- Stay in mode until no more Retry_A

© KnowledgeTek, Inc.
rev 5.41d 15.H.08

1394

Sect 2: Asynchronous Transactions
### Dual Phase Retry Codes

<table>
<thead>
<tr>
<th>Subaction Age</th>
<th>Prior ack code</th>
<th>Retry Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not Oldest</td>
<td></td>
<td>Single Phase</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Retry X</td>
</tr>
<tr>
<td></td>
<td>ack Busy X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ack Busy A</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ack Busy B</td>
<td></td>
</tr>
</tbody>
</table>

| Oldest        |                | Single Phase | Dual Phase |
|---------------|----------------|--------------|
|               |                | Retry X      | Retry 1    |
|               | ack Busy X     | Retry A      |            |
|               | ack Busy A     | Retry B      |            |
|               | ack Busy B     |              |            |

Only one request and one response per talker can be oldest. Selection of oldest is implementation dependent.
## Acknowledge Formats

<table>
<thead>
<tr>
<th>ACK Code</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0h Reserved</td>
</tr>
<tr>
<td>0001</td>
<td>1h ACK Complete</td>
</tr>
<tr>
<td>0010</td>
<td>2h ACK Pending</td>
</tr>
<tr>
<td>0011</td>
<td>3h Reserved</td>
</tr>
<tr>
<td>0100</td>
<td>4h ACK Busy_X</td>
</tr>
<tr>
<td>0101</td>
<td>5h ACK Busy_A</td>
</tr>
<tr>
<td>0110</td>
<td>6h ACK Busy_B</td>
</tr>
<tr>
<td>0111</td>
<td>7h Reserved</td>
</tr>
<tr>
<td>1000</td>
<td>8h Reserved</td>
</tr>
<tr>
<td>1001</td>
<td>9h Reserved</td>
</tr>
<tr>
<td>1010</td>
<td>Ah Reserved</td>
</tr>
<tr>
<td>1011</td>
<td>Bh ACK Tardy</td>
</tr>
<tr>
<td>1100</td>
<td>Ch ACK Conflict error</td>
</tr>
<tr>
<td>1101</td>
<td>Dh ACK Data error</td>
</tr>
<tr>
<td>1110</td>
<td>Eh ACK Type error</td>
</tr>
<tr>
<td>1111</td>
<td>Fh ACK Address Error</td>
</tr>
</tbody>
</table>

ACK code  1's complement

Sect 2: Asynchronous Transactions
Review

1. How does the “talking” node on “listening” node differentiate between Write Request, Write Response, Read Request, Read Response, Lock Request and Lock Response?

2. What does each of the above transactions do?

3. Why do the Acks not have an address?

4. How many address bits are required in 1394 addressing to identify a register location in one node of seven on a single bus? On a 1394 network with 62 busses?
Asynchronous Notes
Section 3

Control and Status Registers (CSRs)
Subjects Covered

1394 Register Space
Core Registers
Bus Dependent Registers
Configuration ROM
What is a Register?

A register is a place for storing information.

A Guest Register in a hotel or wedding is a book where guests write their name.

A Computer Register is normally a latch or group of latches.

A CSR is a fixed memory location in each node that keeps information describing that node.

Computer Registers and CSRs are generally volatile - they lose their contents on reset or power off.

ROM is non-volatile.
What are CSRs?

Control and Status Registers

A defined set of registers in a memory mapped address space intended to be used as part of an open interface.

Defines both a register set and a configuration ROM

Used by 1394, SCI (Scaleable Coherent Interface), NuBus (Texas Instruments), Multibus II (Intel)

Defined in ISO/IEC 13213 and ANSI/IEEE 1212

© KnowledgeTek, Inc.
rev 5.41E 15.H.31

1394

Sect 3: CSRs
CSR Registers

Registers are 4 bytes (32 bits) or 8 bytes (64 bits) wide

Registers are addressed by their offset from the initial register space or other base address

Most registers are optional and there is a large area for vendor specified registers or bus dependent information

Initial contents of each register is defined by spec

Results of a read or a write is defined by spec

Register locations are “Well Known Addresses” so other nodes can read or write to them.
1394 Addressing

Packets use 64 bit Addresses
Top 16 bits determine Node
Bottom 48 bits address location within Node

Node ID
(16 bits) → Address Location within Node
(48 bits)
Addresses are shown from top - down throughout this course.
Node Space Addressing

0000 0000 0000h

Initial Memory Space

FFFF DFFF FFFFh
FFFF E000 0000h

Private Space

FFFF EFFF FFFFh
FFFF F000 0000h

Register Space

FFFF FFFF FFFFh

256 TB

256 MB

256 MB
Register Space

Registers Base Address = FFFF F000 0000h.
Register Structure

Each register has defined bits, reserved bits, and vendor specific bits.

Some registers are read only (RO),
  some are read/write (RW) and
  some are write only (WO)

Some registers are limited to Quadlet read only or lock Transaction for access

In register space, there can be side effects from write Transactions
Addressing CSRs

Example: To address Bus Manager CSR (addr 021Ch) on node FFC8h:

<table>
<thead>
<tr>
<th>Register Space Offset</th>
<th>FFFF F000 0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus Mngr CSR Offset</td>
<td>021C</td>
</tr>
<tr>
<td>Node Address</td>
<td>FFC8</td>
</tr>
<tr>
<td>Register address</td>
<td>FFC8 FFFF F000 021C</td>
</tr>
</tbody>
</table>
## Core Registers (defined by 1212)

<table>
<thead>
<tr>
<th>Hex</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>*State Clear</td>
</tr>
<tr>
<td>0004</td>
<td>*State Set</td>
</tr>
<tr>
<td>0008</td>
<td>*16 bit ID of this node</td>
</tr>
<tr>
<td>000C</td>
<td>*Reset Start</td>
</tr>
<tr>
<td>0018</td>
<td>*Split timeout, Integers of second</td>
</tr>
<tr>
<td>001C</td>
<td>*Split timeout, fractions of second</td>
</tr>
<tr>
<td>0020</td>
<td>Node Self Test Argument, Hi</td>
</tr>
<tr>
<td>0024</td>
<td>Node Self Test Argument, Lo</td>
</tr>
<tr>
<td>0028</td>
<td>Node Self Test, Start</td>
</tr>
<tr>
<td>002C</td>
<td>Node Self Test, Status</td>
</tr>
<tr>
<td>0050</td>
<td>Interrupt Target</td>
</tr>
<tr>
<td>0054</td>
<td>Interrupt Mask</td>
</tr>
<tr>
<td>0058 to 007C</td>
<td>Assorted clock control, normally not implemented on Serial Bus</td>
</tr>
<tr>
<td>0080 to 00FC</td>
<td>Message Request/Response</td>
</tr>
</tbody>
</table>

* required in 1394-1995 or SBP-2
State Register

A write of a 1 bit to STATE CLEAR register clears the identified bit.

A write of a 1 bit to STATE SET register sets the identified bit.

A read of either address 0 or 4 gives the contents of the State Register.
State Registers

Unit depend  Unit Vendor Specific
Bus depend  Bus type specific (see next slide for 1394)
Lost  Set on reset, cleared by the software; indicates the unit is “lost”
Dreq  Disable request from unreliable nodes
Elog  An error has been detected and the error log has been updated
Attn  Attention; this node should be prepared for on-line replacement
Off  Prevent node access while a board is being replaced
State
0 - Running
1 - Initializing
2 - Testing
3 - Dead
State Register Bus Dependent Information

<table>
<thead>
<tr>
<th></th>
<th>Gone</th>
<th>r</th>
<th>r</th>
<th>r</th>
<th>r</th>
<th>Abdicate</th>
<th>Linkoff</th>
<th>CMSTR</th>
</tr>
</thead>
</table>

- **Gone**: Set to a 1 on any reset, cleared when reset is completed.
- **Linkoff**: Setting this bit powers off the Link layer (Defined in Implementation section).
- **CMSTR**: Node is Cycle Master Capable (defined in Isochronous section).
- **Abdicate**: After a bus reset, the incumbent Bus Manager will wait 125 mSec before doing a lock request to the Bus Manager ID CSR (1394a).
## Serial Bus Defined Registers

<table>
<thead>
<tr>
<th>Address(h)</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0200</td>
<td>CYCLE_TIME</td>
<td>For isochronous services, counts 24.576 MHz clocks</td>
</tr>
<tr>
<td>0204</td>
<td>BUS_TIME</td>
<td>For synchronized bus time</td>
</tr>
<tr>
<td>0208</td>
<td>POWER_FAIL_IMMINENT</td>
<td>Power fail warning</td>
</tr>
<tr>
<td>020C</td>
<td>POWER SOURCE</td>
<td>Power fail warning</td>
</tr>
<tr>
<td>0210</td>
<td>BUSY_TIMEOUT</td>
<td>For transaction capable nodes - limits</td>
</tr>
<tr>
<td></td>
<td>number of retries to a busy node</td>
<td></td>
</tr>
<tr>
<td>0218</td>
<td>PRIORITY BUDGET</td>
<td>For priority arbitration</td>
</tr>
<tr>
<td>021C</td>
<td>BUS_MANAGER_ID</td>
<td>For selecting or locating bus manager</td>
</tr>
<tr>
<td>0220</td>
<td>BANDWIDTH_AVAILABLE</td>
<td>Bandwidth allocation</td>
</tr>
<tr>
<td>0224-0228</td>
<td>CHANNELS_AVAILABLE</td>
<td>Channel allocation</td>
</tr>
<tr>
<td>022C</td>
<td>MAINT_CONTROL</td>
<td>Diagnostics, to generate specific errors</td>
</tr>
<tr>
<td>0230</td>
<td>MAINTUTILITY</td>
<td>Diagnostics</td>
</tr>
<tr>
<td>0234</td>
<td>BROADCAST CHANNEL</td>
<td>For broadcast via asynchronous streams</td>
</tr>
</tbody>
</table>
Serial Bus Defined Registers in Initial Units Space

<table>
<thead>
<tr>
<th>Offset (h)</th>
<th>Name</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>800 - 8FC</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>900 - 9FC</td>
<td>PLUG CONTROL REGISTERS</td>
<td>Logical connections of isochronous devices IEC 61883</td>
</tr>
<tr>
<td>A00 - AFC</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>B00 - CFC</td>
<td>FCP CMD Frame</td>
<td>IEC 61883</td>
</tr>
<tr>
<td>D00 - EFC</td>
<td>FCP RESP Frame</td>
<td>IEC 61883</td>
</tr>
<tr>
<td>F00 - FFC</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>1000 - 13FC</td>
<td>TOPOLOGY MAP</td>
<td>Bus Manager only</td>
</tr>
<tr>
<td>1400 - 1FFC</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>2000 - 2FFC</td>
<td>SPEED MAP</td>
<td>Bus Manager only (obsoleted)</td>
</tr>
<tr>
<td>3000 - FFFC</td>
<td>Reserved</td>
<td></td>
</tr>
</tbody>
</table>

TOPOLOGY MAP will be defined in the Bus Management section.
ROM Hierarchy

(Node address) FFFF F000 0400h (1K off Register Base address)

Chart from IEC 13213

* Present in Simple Devices
Info length = number of quadlets in bus_info_block (always 4)
CRC length = quadlets of this ROM protected; minimum = bus info block,
            maximum = 255
ROM CRC value = the 16 bit CRC check character for this ROM
## Bus Info Block

<table>
<thead>
<tr>
<th>Length = 4</th>
<th>CRC Length</th>
<th>CRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>31h &quot;1&quot;</td>
<td>33h &quot;3&quot;</td>
<td>39h &quot;9&quot;</td>
</tr>
<tr>
<td>34h &quot;4&quot;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IRC</th>
<th>I</th>
<th>B</th>
<th>PMC</th>
<th>Resv (3 bits)</th>
<th>cyc clk acc</th>
<th>max rec (4 bits)</th>
<th>r (2 bits)</th>
<th>Max ROM (2 bits)</th>
<th>G (4 bits)</th>
<th>Link Speed (3 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Node vendor ID**: (2 bits)
- **chip ID high**: (4 bits)
- **chip ID low**: (4 bits)

---

1394

Sect 3: CSRs
Bus Info Block

IR  Isochronous Resource Manager capable
C  Cycle master capable
I  Isochronous capable
B  Bus Manager capable
PMC Power Manager Capable
G  Generation number - Indicates information in configuration or any leaf or directory changed

Link Speed  Maximum speed of the node’s link layer

Cyc clk acc  Accuracy of cycle clock in parts per million (1-100)

Node vendor ID  24 bit globally unique Organizationally Unique Identifier (OUI) assigned by IEEE Registration Authority

Chip ID Hi/Lo  40 bit globally unique ID administered by node vendor. Node vendor ID concatenated with chip ID Hi and Lo yield a 64 bit Extended Unique ID (EUI-64).

Max ROM  Defines alignment for block read requests to configuration ROM
Maximum Record Length

max rec - the maximum of an asynchronous write addressed to this node

<table>
<thead>
<tr>
<th>max rec</th>
<th>Max size in bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0h</td>
<td>not specified</td>
</tr>
<tr>
<td>1h</td>
<td>4</td>
</tr>
<tr>
<td>2h</td>
<td>8</td>
</tr>
<tr>
<td>3h</td>
<td>16</td>
</tr>
<tr>
<td>4h</td>
<td>32</td>
</tr>
<tr>
<td>5h</td>
<td>64</td>
</tr>
<tr>
<td>6h</td>
<td>128</td>
</tr>
<tr>
<td>7h</td>
<td>256</td>
</tr>
<tr>
<td>8h</td>
<td>512</td>
</tr>
<tr>
<td>9h</td>
<td>1024</td>
</tr>
<tr>
<td>Ah</td>
<td>2048</td>
</tr>
<tr>
<td>Bh</td>
<td>4096</td>
</tr>
<tr>
<td>Ch</td>
<td>8192</td>
</tr>
<tr>
<td>Dh</td>
<td>16384</td>
</tr>
<tr>
<td>Eh - Fh</td>
<td>reserved</td>
</tr>
</tbody>
</table>
### Unit or Root Directory

<table>
<thead>
<tr>
<th>Key Type</th>
<th>Key Value</th>
<th>Information or address</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td></td>
<td>- immediate value</td>
</tr>
<tr>
<td>01</td>
<td></td>
<td>- initial-register-space offset for an immediate value</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>- indirect-space offset for a leaf</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>- indirect-space offset for a directory</td>
</tr>
</tbody>
</table>

**Key Type (2 bits)**
- 00 - immediate value
- 01 - initial-register-space offset for an immediate value
- 10 - indirect-space offset for a leaf
- 11 - indirect-space offset for a directory

**Key Value (6 bits)**
identifies the 24 bit directory entry
Root Directory Example

<table>
<thead>
<tr>
<th>LENGTH</th>
<th>0C</th>
<th>03</th>
<th>8D</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRC</td>
<td>00 83 D2</td>
<td>E4 71 04</td>
<td>00 00 02</td>
</tr>
</tbody>
</table>

Note: These three entries are required by 1394-1995. Many others are permissible.
Configuration ROM Example

What type of entry is the first one?  Using the reference material in the back of this section, what capabilities are supported?

What type of entry is the second one?  What is the significance of the 24 bit entry?

What type of entry is the third one?  What steps would you use to find the leaf?
Questions???

What is the Node address of that node?
   Check the NODE_ID CSR
Does that node support split timeout?
   Check the NODE_CAPABILITIES.spt bit
Is that node doing a reset?
   Check the STATE_BITS.lost bit
What protocol does that node support?
   Check the bus_info_block of the CONFIGURATION ROM
I have a problem I need to notify somebody
   Set the STATE_BITS.elog bit
Review - CSR

You need to discover the node capabilities. What is the full chain of pointers to find that information?

You need to discover the unit's power requirements. What is the full chain of pointers to find that information?
## Reference - Keyvalues

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Value Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Textual_Descriptor</td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>Bus_Dependent_Info</td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>Module_Vendor_ID</td>
<td></td>
</tr>
<tr>
<td>04</td>
<td>Module_Hardware_Version</td>
<td></td>
</tr>
<tr>
<td>05</td>
<td>Module_Spec_ID</td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>Module_Software_Version</td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>Module_Dependent_Info</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>Node_Vendor_ID</td>
<td></td>
</tr>
<tr>
<td>09</td>
<td>Node_Hardware_Version</td>
<td></td>
</tr>
<tr>
<td>0A</td>
<td>Node_Spec_ID</td>
<td></td>
</tr>
<tr>
<td>0B</td>
<td>Node_Software_Version</td>
<td></td>
</tr>
<tr>
<td>0C</td>
<td>Node_Capabilities</td>
<td></td>
</tr>
<tr>
<td>0D</td>
<td>Node_Unique_ID</td>
<td></td>
</tr>
<tr>
<td>0E</td>
<td>Node_Units_Extent</td>
<td></td>
</tr>
<tr>
<td>0F</td>
<td>Node_Memory_Extent</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Node_Dependent_Info</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Unit_Directory</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Unit_Spec_ID</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Unit_Software_Version</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Unit_Dependent_Info</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Unit_Location</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Unit_Poll_Mask</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>17h-2Fh</td>
</tr>
<tr>
<td></td>
<td></td>
<td>30h</td>
</tr>
<tr>
<td></td>
<td></td>
<td>31h-37h</td>
</tr>
<tr>
<td></td>
<td></td>
<td>38h</td>
</tr>
<tr>
<td></td>
<td></td>
<td>39h</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3Ah</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3Bh-3Fh</td>
</tr>
<tr>
<td></td>
<td></td>
<td>54h</td>
</tr>
</tbody>
</table>
### Reference - Node Capabilities ROM Entry

<table>
<thead>
<tr>
<th>0Ch</th>
<th>00h</th>
<th>spt</th>
<th>ms</th>
<th>int</th>
<th>ext</th>
<th>bas</th>
<th>prv</th>
<th>64</th>
<th>fix</th>
<th>lst</th>
<th>drq</th>
<th>r</th>
<th>elo</th>
<th>atn</th>
<th>off</th>
<th>ded</th>
<th>init</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

0Ch  Key type & key value  
*     Required by 1394-1995  
spt  Split timeout implemented  
ms  Message passing registers implemented  
int  Interrupt target and Interrupt mask registers implemented  
ext  Argument registers implemented  
bas  Test start and Test State registers implemented  
prv  Uses private space  
64  Uses 64 bit addressing (otherwise 32 bit addressing)  
fix  Uses fixed addressing (otherwise extended addressing)  
lst  State Bits.lost implemented  
drq  State Bits.dreq implemented (disable requests)  
elo  Error log implemented  
atn  State Bits.atn implemented  
off  State Bits.off implemented  
ded  Supports Dead state  
init  Supports initializing state
Sample Configuration ROM

- **Bus Info Block**
  - 4 0014h ROM CRC (calculated)
  - 3133 3934h (ASCII "1394")
  - node_options (00FF 2000h)
  - node_vendor_ID
  - chip_ID_hi
  - chip_ID_lo
  - LENGTH
    - Root directory CRC (calculated)
  - 03h module_vendor_ID
  - 0Ch node_capabilities (00 8380h)
  - 8Dh 2 (leaf pointer)
  - D1h 4 (unit directory pointer)
- **Root Dir**
- **Leaf**
  - 2 Leaf CRC (calculated)
  - node_vendor_ID
  - chip_ID_hi
  - chip_ID_lo
## Sample Configuration ROM (Continued)

<table>
<thead>
<tr>
<th>Unit Dir</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Unit directory CRC (calculated)</td>
</tr>
<tr>
<td>12h</td>
<td>unit_spec_ID (00 609Eh)</td>
</tr>
<tr>
<td>13h</td>
<td>unit_sw_version (01 0483h)</td>
</tr>
<tr>
<td>38h</td>
<td>command_set_spec_ID</td>
</tr>
<tr>
<td>39h</td>
<td>command_set_version</td>
</tr>
<tr>
<td>54h</td>
<td>Management Agent CSR Offset (00 4000h)</td>
</tr>
<tr>
<td>3Ah</td>
<td>Logical Unit Characteristics (01 0A08h)</td>
</tr>
<tr>
<td>14h</td>
<td>Logical Unit Number (00 0000h)</td>
</tr>
</tbody>
</table>
The 24 bit power requirements field specifies, in deci-watts, the power required by the unit in excess of the power requirements stated in the Self-ID packet. The Self-ID packet will be covered in configuration.

Self powered units will not have this entry in Configuration ROM.
## Command Set
### Unit Directory Entries

<table>
<thead>
<tr>
<th></th>
<th>12</th>
<th>13</th>
<th>38</th>
<th>39</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Bay</td>
<td>00805F</td>
<td>010000</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>SBP-2 SCSI</td>
<td>00609E</td>
<td>010483</td>
<td>00609E</td>
<td>0104D8</td>
</tr>
<tr>
<td>SBP-2 ATA</td>
<td>00609E</td>
<td>010483</td>
<td>00609E</td>
<td>040000</td>
</tr>
<tr>
<td>SBP-2 AV/C</td>
<td>00609E</td>
<td>010483</td>
<td>00A02D</td>
<td>010001</td>
</tr>
<tr>
<td>Camera 1.04</td>
<td>00A02D</td>
<td>000100</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Camera 1.20</td>
<td>00A02D</td>
<td>000101</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

12 00609E = NCITS
    00A02D = 1394TA

13 010483 = SBP-2

38 Same as 12

39 0104D8 = SCSI
    040000 = ATA
    010001 = AV/C
Logical Unit Characteristics

| 3Ah | Reserved | ORB Timeout | ORB Size |

ORB Timeout

ORB Size  In quadlets
### Logical Unit Number

<table>
<thead>
<tr>
<th>14h</th>
<th>r</th>
<th>o</th>
<th>Device Type</th>
<th>Logical Unit Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Device Type:**

- O: Ordered
- r: Reserved

**Device Type Values:**

- 0 = Block Device
- 1 = Sequential Device
- 2 = Printer
- 3 = Processor
- 4 = Worm
- 5 = CD-ROM
- 6 = Scanner
- 7 = Optical Memory
- 8 = Media Changer
- 9 = Communication
- A = Pre-Press
- B = Pre-Press
- C = Enclosure Services
- E = Reduced Block Command
Review

1. How many “state” registers are there in each node?
2. At what address is the “state” register?
3. Which node on a bus has the configuration ROM?
4. Which node has the Root Directory?
5. What is the value after bus reset of the register at offset 21Ch?
6. At what address (64 bits) will I find the beginning of config ROM?
Section 4

Introduction To SCSI Over 1394
Subjects Covered

Introduction to SCSI
CDB data structure
ORB data structure
SCSI

Small Computer System Interface

Specification under the control of the T10 Committee of NCITS
(\textit{NCITS} = \textit{National Committee for Information Technology Standardization}, formerly \textit{X3} committee of ANSI)

System Level Interface

- Drive appears as a 'stack' of logical blocks
- Each block has unique Logical Block Address (LBA)

Physical:
- 50 conductor cable (18 signals), usually ribbon cable (narrow)
- 68 conductor cable (27 signals), usually ribbon cable (wide)

Logical:
- Commands to write/read data to/from LBAs
- Many, many, many other esoteric commands
SCSI Devices

8 or 16 Devices Max
Initiator = Device Originating A Command
Target = Device Responding To A Command
Types of Bytes (Phases)
- Arbitration
- Selection
- Command Descriptor Bytes (CDB)
- Data (In/Out)
- Status
- Message (In/Out)

9 Signals To Move Data
(1 Byte + Parity)*

9 Control Signals
Transfer Bytes
Indicates Types of Bytes

*Wide option with more signals moves 2 Bytes at a time
Example SCSI Read Command

- Bus Free Arbitration
- Selection
- ID Message
- CDB
- CDB
- CDB
- Data Byte
- Data Byte
- Data Byte
- Status
- CC Message
- Bus Free
# Command Formats

<table>
<thead>
<tr>
<th>6 Byte Command</th>
<th>10 Byte Format</th>
<th>12 Byte Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>OP Code</td>
<td>OP Code</td>
<td>OP Code</td>
</tr>
<tr>
<td>LBA_{20-16}</td>
<td>LBA_{31-24}</td>
<td>LBA_{31-24}</td>
</tr>
<tr>
<td>LBA_{15-8}</td>
<td>LBA_{23-16}</td>
<td>LBA_{23-16}</td>
</tr>
<tr>
<td>LBA_{7-0}</td>
<td>LBA_{15-8}</td>
<td>LBA_{15-8}</td>
</tr>
<tr>
<td>Xfer Length</td>
<td>LBA_{7-0}</td>
<td>LBA_{7-0}</td>
</tr>
<tr>
<td>Control</td>
<td>Xfer Len_{15-8}</td>
<td>Xfer Len_{15-8}</td>
</tr>
<tr>
<td></td>
<td>Xfer Len_{7-0}</td>
<td>Xfer Len_{7-0}</td>
</tr>
<tr>
<td></td>
<td>Control</td>
<td>Control</td>
</tr>
</tbody>
</table>

Note: Most commands don't require LBA or Xfer Len and will use these fields differently.
Example SCSI Read CDB

- Read OpCode
- Logical Block Address 12345h
- Read 8 blocks

| 08h | 01h | 23h | 45h | 08h | 00h |

Sect 4: Introduction to SCSI over 1394
“Obvious” Way:

Read/Write Packets
Directly Address Data

1394

How it’s done:

Read/Write Packets
Move Commands

Data moved as
part of command

1394

Controller

1394

Memory Mapped Disk

© KnowledgeTek, Inc.
Transactions vs. Commands

Transactions
Read, Write, & Lock
Issued by any device
(if SCSI could be initiator or target)

Commands
SCSI Commands
Only Issued by Initiator
ORB - Operation Request Block

SCSI Commands are ‘wrapped” in an ORB:

<table>
<thead>
<tr>
<th>Next ORB</th>
<th>Pointer to Location of next ORB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Descriptor</td>
<td>Pointer to Data Location</td>
</tr>
<tr>
<td>Control Information</td>
<td>More on this later</td>
</tr>
<tr>
<td>CDBs (6, 10, 12, or 16 Bytes)</td>
<td>SCSI Command</td>
</tr>
</tbody>
</table>

Initiator builds ORB in its memory
Target transfers ORB into the controller
SCSI Over 1394

Initiator

Build ORB

ORB

Write ORB Address

Read Request for ORB

Read Response with ORB

Reads or Writes

To Move Data (command dependent)

Write Status

Target

Execute Command

Build Status

Status

Note: ACKs have been omitted to reduce clutter
SCSI Read Processing

Initiator

Build ORB

ORB

Write ORB Address

Read Request for ORB

Read Response with ORB

Write Data

Write Status

Target

Execute Command

Read Data

Build Status

1394

Sect 4: Introduction to SCSI over 1394
Protocol Layers

Upper Level Protocol

SCSI-3

CDBs, Messages, Status

SBP-2 ANNEX

ORBs

Transactions

Read, Write & Lock

Packets

1394 - 1995

Link

Bytes and Control

Physical

© KnowledgeTek, Inc.
rev 5.41d 15.H.08

Sect 4: Introduction to SCSI over 1394
Review

1. Differentiate the different levels referenced by the term SCSI
2. What is contained in a CDB?
3. What do the acronyms CDB and ORB mean?
4. What is contained in an ORB?
SCSI Over 1394 Notes
Section 5

Serial Bus Protocol
SBP-2
Subjects Covered

Command ORBs and fields
Management ORBs and fields
Status Blocks
Login and Resets
Serial Bus Protocol: SBP-2

Maps Upper Level Protocols (ULPs) onto 1394

ULPs = SCSI, ATA, IP, ??

SCSI, ATA, Other ULP

SCSI, ATA, Other ULP

1394

© KnowledgeTek, Inc.
rev 5.41d 15.H.08
SBP-2 Overview

Command set neutral

Current plans address SCSI, ATA, ATAPI, IP

ULP Commands are packaged in an ORB (Operation Request Block)

Devices must login before sending commands
   Exchange certain operational information
SBP - 2 Command Process

Initiator

Build ORB

ORB

Target

Write ORB Address

Read Request for ORB

Read Response with ORB

Read or Writes

To Move Data (command dependent)

Write Status

Execute Command

Build Status

Status

© KnowledgeTek, Inc.
rev 5.41d 15.H.08

1394
Sect 5: SBP-2
Operation Request Block - ORB

- **Next ORB**
  - Pointer to Next ORB
  - Allows chaining commands
  - Set to Null if no more ORBs in chain

- **Data Descriptor**
  - Pointer to Data Buffer

- **Control Information**
  - Info on how to execute command at the 1394 Bus Level

- **Command**
  - Upper Level Protocol (ULP) command
    - SCSI CDBs
    - ATA Command Block Registers
    - ATAPI Packet Command
    - TBI*

* To be invented
**ORB Control Information**

<table>
<thead>
<tr>
<th>N</th>
<th>Req Fmt</th>
<th>r</th>
<th>D</th>
<th>spd</th>
<th>Max Payload</th>
<th>P</th>
<th>PSize</th>
<th>Data Size</th>
</tr>
</thead>
</table>

- **N** Notify - Initiator Status has been posted
  - 1 = Post status at end of this ORB
  - 0 = Don’t post status unless there’s an error

- **Req Fmt** Request Format
  - 0 = SBP-2, 1 = Reserved, 2 = Vendor dependent, 3 = Dummy ORB

- **D** Direction: 0 = Data transfer into target memory
  - 1 = Data transfer into initiator memory

- **Speed**
  - 0 = S100, 1 = S200, 2 = S400, 3 = S800, 4 = S1600, 5 = S3200

- **Max Payload** Maximum number of bytes in a single read or write = $2^{(\text{max pay} + 2)}$

- **P** Page table present - Indicates Data Descriptor uses Indirect Mode

- **PSize** Determines size of the pages for Indirect Data Descriptors

- **Data Size** Size in bytes of the system memory of the Data Buffer (P=0)
  - Number of elements in Page table (P=1)
Using The Next ORB Pointer

Initiator builds Linked-List of ORBs
Writes address of first to Target
Target reads ORB #1
Executes ORB #1
Writes Status for ORB #1
Uses Next ORB Pointer to read ORB #2
Executes ORB #2
Writes Status for ORB #2
Uses Next ORB Pointer to read ORB #3
Executes ORB #3
Writes Status for ORB #3
Next ORB Pointer = Null Indicates done
Next ORB Pointer Format

N  Null Flag Bit
   0 = Offset hi and Offset lo are the address for the next ORB
   1 = This is the last ORB in the link list, ignore offset hi and offset lo

Reserved  Set to Zero

Next_ORB  Address of Next ORB in Initiators Node Memory Space
ORB must start on Quadlet boundary
(Bottom two bits of address must be zero)
Data Descriptor

Location of the Data Buffer
Read Commands - Data placed here
Write Commands - Data taken from here

Two Modes of Operation
Direct - Data Descriptor contains address
Indirect - Data Descriptor contains address of Page Table
P flag in Control Info indicates which
Using The Data Descriptor In Direct Mode
(P Flag = 0)

Note: Data Buffer can be located **anywhere**
not just in Initiator’s Node!
Using The Data Descriptor In Indirect Mode
(P Flag = 1)

ORB in Initiator Memory

Data Length = Page Size * Number of Segments

Note: Segments must be in same node as Page table

Node Memory

Page Table
Seg A Addr
Seg B Addr
Seg C Addr

Seg A
Seg C
Seg B

Page Size
(Set by Control Info)
Control Info For Indirect Data Descriptors

Page Table Present
Determines Page Size (segment)
Number of elements in table

Page Size = $2^{(P\text{Size} + 8)}$

<table>
<thead>
<tr>
<th>PSize</th>
<th>Page Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>256 Bytes</td>
</tr>
<tr>
<td>0 0 1</td>
<td>512 Bytes</td>
</tr>
<tr>
<td>0 1 0</td>
<td>1K Bytes</td>
</tr>
<tr>
<td>0 1 1</td>
<td>2K Bytes</td>
</tr>
<tr>
<td>1 0 0</td>
<td>4K Bytes</td>
</tr>
<tr>
<td>1 0 1</td>
<td>8K Bytes</td>
</tr>
<tr>
<td>1 1 0</td>
<td>16K Bytes</td>
</tr>
<tr>
<td>1 1 1</td>
<td>32K Bytes</td>
</tr>
</tbody>
</table>
Page Table Format

Page Table composed of Page Table Elements

Segment Length  Number of Bytes used in this segment
                Normally equal to Page Size

Segment Base    Address of 1st Byte of Segment
                Node ID same as for Page Table
                Append (PSize + 8) zero bits

Segment Offset  Used in Offset Transfers
                Must be zero in all but 1st Page Table Element
Normalized Page Tables

Segment Length = Page Size
Segment Length = Page Size
Segment Length = Remaining Data
Normalized Page Tables

Start of page for 2A

Start of Seg 2A including offset

Offset + Segment Length equals page size

Segment Length

Offset hi

Offset Lo

for page alignment or offset for 1st seg
Status Block

<table>
<thead>
<tr>
<th>SRC</th>
<th>Resp</th>
<th>D</th>
<th>Len</th>
<th>SBP Status</th>
<th>ORB Offset Hi</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ORB Offset Lo

Command Set Dependent

Note: If there is no error, the target need only post the first two quadlets of status
Status Block Definitions

ORB Offset  Identifies ORB for this status

Resp  Response
  0 = Request complete. The request completed without transport protocol error.
  1 = Transport failure. Target detected nonrecoverable transport error.
  2 = Illegal request. Unsupported bit or field in ORB.
  3 = Vendor dependent.

D  1 = Target transitioned to dead state

Len  Length. Number of valid quadlets -1 stored as status
     (Value of 7 means 8 quadlets were stored)

SRC  00b Solicited Status, not end of list
     01b Solicited Status, next ORB = Null
     10b Unsolicited Status
     11b Reserved
SBP-2 Status Block Definitions

The following are valid only if Resp = 0, Request Complete

0 = No additional sense to report
1 = Invalid request type
2 = Speed not supported
3 = Page size not supported
4 = Access denied
5 = Logical unit not supported
6 = Maximum payload too small
7 = Too many channels
8 = Resources unavailable
9 = Function rejected
A = Login ID not recognized
B = Dummy ORB completed
C = Request aborted
FF = unspecified error
Notify Bit

Notify Initiator that status has been posted.

1 - Post status at the completion of this ORB

0 - Only post status if it terminated in an error
Review

Covered

Initiator built Command ORB and notified target
SCSI commands are covered in the section 6
ATA commands are covered in the section 7
Target fetched ORB with a read transaction
Target executed command
Target used write transaction to send status to Initiator

Yet to cover

How does Initiator know where to write ORB address?
How does Target know where to write status?
How does the Target control who is sending it commands?
Can the Initiator send several commands at once (Stream)?
Management vs. Command ORBs
Agents

Management Agent - Manages Node
Login, Logout, Reset, etc.

Command Agent - Performs the Command
(Device Controller) stuff w/ ULP
Management Vs. Command ORBs

Management ORBs
- Sent to the Management Agent
- Execute a single function only (can’t be linked)
- Functions aimed at the node
  - Login, Logout, Logical Unit Reset, Target Reset, etc.

Command ORBs
- Sent to the Command Agent
- May be connected in Link Lists
- Functions aimed at the device (ULP Commands)
Command ORB Format

- Next ORB
- Data Descriptor
- Control Information
- Command
Management ORB Format

Note: All currently defined Management ORBs adhere to this format. However, the SBP-2 standard does not specify that future Management ORBs will necessarily follow this format. The standard specifies the format on a function by function basis.
Management ORB Fields

- **Response Address**: Location in system memory to write the response to this ORB.
- **N**: Notify Status Flag
  - 1 = Always report Status
  - 0 = Only report Status on Errors
- **Req Fmt**: Set to 00
- **Response Length**: Space reserved for Response at Response Address
- **Status FIFO**: Location in system memory to write the status block for this ORB.
- **Identifier**: Identifies who the ORB is for
  - LUN on Logins
  - Login ID on the other Management ORBs
## Management Functions

<table>
<thead>
<tr>
<th>Value</th>
<th>Management Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Login</td>
</tr>
<tr>
<td>1</td>
<td>Query Logins</td>
</tr>
<tr>
<td>2</td>
<td>Reserved</td>
</tr>
<tr>
<td>3</td>
<td>Reconnect</td>
</tr>
<tr>
<td>4</td>
<td>Set Password</td>
</tr>
<tr>
<td>5-6</td>
<td>Reserved</td>
</tr>
<tr>
<td>7</td>
<td>Logout</td>
</tr>
<tr>
<td>8-A</td>
<td>Reserved</td>
</tr>
<tr>
<td>B</td>
<td>Abort Task</td>
</tr>
<tr>
<td>C</td>
<td>Abort Task Set</td>
</tr>
<tr>
<td>D</td>
<td>Reserved</td>
</tr>
<tr>
<td>E</td>
<td>Logical Unit Reset</td>
</tr>
<tr>
<td>F</td>
<td>Target Reset</td>
</tr>
</tbody>
</table>
The SBP - 2 Command Process

Initiator

Build ORB

ORB

Target

Write ORB Address

Read Request for ORB

Read Response with ORB

Reads or Writes

To Move Data (command dependent)

Write Status

Who does the Read Req for the ORB?

Execute Command

Build Status

Status

How does the initiator know where to write the ORB address?

How does the target know where to write the status?
Login

Management ORB to Management Agent

Performed before any Command ORBs are sent by Initiator

Tells Target where to return Status

Response informs Initiator location of Command Agent

Where to write Command ORB addresses

Exclusive use provisions

Only one Initiator logged in at a time
Login Command Process

Initiator

- Build Login ORB
- Write Address of Login ORB
- Read Request for Login ORB
- Read Response with Login ORB
- Write Request with Login Response
- Write Request with Status

Target

- Execute Login
- Build Login Response
- Build Status

© KnowledgeTek, Inc.
Login ORB Format

Password Address

Login Response Address

N  Req  X  Reserved  Function  = 0  LUN
Fmt             

Password Length  Login response length

Status FIFO Address
Login ORB Fields

Login Response Address:
Address in system memory of where to write the Login Response Data

N:
Notify Status Flag

Req Fmt:
= 00

X:
Exclusive Flag
1 = No other Logins to this LUN
0 = Other Logins allowed

Password Length:
Length of Password in Bytes
If zero, no Password

Password Address:
Address in system memory of where to read Password from

Status FIFO Address:
Address in system memory of where to write status block
### Login

**Login Response Packet**

<table>
<thead>
<tr>
<th>Login Response Length = 12 or 16</th>
<th>Login ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command Block Agent Address</td>
<td></td>
</tr>
<tr>
<td>(address where Target needs to fetch ORBs from)</td>
<td></td>
</tr>
<tr>
<td>Reserved</td>
<td>Reconnect Hold</td>
</tr>
</tbody>
</table>

**Login ID**

Supplied by Target

Used by Initiator in Management ORBs to identify login connection

**Reconnect Hold**

Specified time target will hold resources waiting for a reconnect following a bus reset

Value of 5 means hold resources for 6 seconds
Login Command Process

How does Initiator know where Management Agent is?
Find Target’s Management Agent Register

FFFF F000 0400

Bus Info Block

ROM Root

Unit Directory

Management Agent Registers

FFFF F000 0000h
+
0001 0000h

FFFF F001 0000h

In Quadlets
Streaming

Initiator creates string of ORBs
Writes Address of 1st ORB to Command Agent
Target executes ORBs
    In Order
    Out of Order  Target Dependent
Writes Status Block when each ORB Complete

How do you know if the Target executes in order?
Adding To The Stream

Initiator creates string of ORBs
Writes Address of 1st ORB to Command Agent
Target executing ORBs

Initiator receives two more requests for this Target
Create additional ORBs
Point end of list to next ORB

What if the Command Agent has already read it?
# Command Block Agents

<table>
<thead>
<tr>
<th>Relative offset</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>Agent State</td>
<td>Reports fetch agent state</td>
</tr>
<tr>
<td>04h</td>
<td>Agent Reset</td>
<td>Resets fetch agent</td>
</tr>
<tr>
<td>08h</td>
<td>ORB Pointer</td>
<td>Address of request block</td>
</tr>
<tr>
<td>10h</td>
<td>Doorbell</td>
<td>Signals fetch agent to refetch an address pointer</td>
</tr>
<tr>
<td>14h</td>
<td>Status Acknowledge</td>
<td>Acknowledges receipt of unsolicited status</td>
</tr>
<tr>
<td>18h - 1Ch</td>
<td></td>
<td>Reserved for future standardization</td>
</tr>
</tbody>
</table>

**Agent States:**

- 0 = Reset
- 1 = Active
- 2 = Suspended
- 3 = Dead
Multiple ORB Streams

Initiator creates string of ORBs
Writes Address of 1st ORB to Command Agent
Target executing ORBs

Initiator receives two more requests for this Target
Create additional ORBs
Write address to Command Agent
Target can now execute commands from both strings

What if the Command Agent only supports a single ORB pointer?
Tell Us What’s Happening - Trace Format

Destination
Source
Transaction
Length
Address

3 5 RReq 14
5 3 RRsp

Data Bytes

Reason For Request
Key Info Returned

All Numbers In Hex
Trace doesn’t show Ack Packets
Tell Us What's Happening - Part 1

3 5 RRreq 14 FFFF F000 0400
5 3 RRsp 04 XX XX XX
   31 33 39 34
   E0 FF 80 02
   12 34 56 78
   9A BC DE F0

3 5 RRreq 4 FFFF F000 0414
5 3 RRsp 00 04 XX XX

3 5 RRreq 10 FFFF F000 0418
5 3 RRsp 03 12 34 56
   0C 00 83 80
   8D 00 00 02
   D1 00 00 04

3 5 RRreq 4 FFFF F000 0434
5 3 RRsp 00 07 XX XX

Read Config Room

Bus Info Block

Length of Pro File

Module Vendor Id
Node Capabilities
Module Unique Id

Indirect Offset
Unit Directory

Offset From Where we are now (FFFF F000 0434)

Length of Unit Directory

© KnowledgeTek, Inc.
rev 5.41d 15.H.08

1394
Sect 5: SBP-2

5 - 42
Tell Us What's Happening - Part 2

3 5 RReq 1C
FFFF F000 0438
3 5 RRsp
12 00 60 9E
13 01 04 83
38 00 60 9E
39 01 04 D8
14 0E 00 00
3A 00 0A 08
54 00 40 00

5 3 RRsp
12 00 60 9E
13 01 04 83
38 00 60 9E
39 01 04 D8
14 0E 00 00
3A 00 0A 08
54 00 40 00

Read Unit Directory
SCSI Device 3-35

Logical Unit Number - Reduced Block Cmd p.3-37

management agent CSR = 4000h quad offset
management

Write add to login ORB to management agent
add to login ORB

Read Request for login ORB

© KnowledgeTek, Inc.
rev 5.41d 15.H.08

1394
Sect 5: SBP-2
5 - 43
Tell Us What’s Happening - Part 3

5 3 WReq 0C 0000 1010 0000
00 0C 12 34 FF C5 FF FF
F0 10 01 00

5 3 WReq 08 0000 1020 0000
41 00 00 00 addr 5-18
10 00 00 00 5-19

3 5 WReq 08 FFFF F010 0108
FF C3 00 00
10 00 00 00

5 3 RReq 20 0000 1000 0000
00 00 00 00
10 00 00 20

3 5 RRsp 00 00 00 00
10 00 00 20
FF C3 00 00
20 00 00 00
82 90 00 20
12 00 00 00
08 00 00 00
00 00 00 00

Login response

Status

Initiator writes ORB pointer (address)

ORB address

Target requests read ORB

ORB

1394

Sect 5: SBP-2
### Tell Us What’s Happening - Part 4

<table>
<thead>
<tr>
<th></th>
<th>RReq</th>
<th>20</th>
<th>0000 1000 0020</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>00</td>
<td>80 00 00 00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00</td>
<td>00 00 00 00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FF</td>
<td>C3 00 00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
<td>00 00 20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>82</td>
<td>90 00 20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00</td>
<td>00 00 00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00</td>
<td>00 00 00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00</td>
<td>00 00 00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>RRsp</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>WReq</th>
<th>08</th>
<th>0000 2000 0000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>00</td>
<td>00 00 03 03</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00</td>
<td>00 00 00 00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>WReq</th>
<th>08</th>
<th>0000 1020 0000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>01</td>
<td>00 00 00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>00 00 00</td>
</tr>
</tbody>
</table>

### What Condition Is The Target In ?

- `has one outstanding WReq`  
- `wqie status for test unit ready cmd`
Review

1. What does the next ORB pointer point to?
2. What are the limitations on the location of each ORB?
3. What is addressed by the data descriptor field in the ORB?
4. Explain direct addressing
5. Explain indirect addressing
6. Where are the function codes?
7. Explain the login process
8. What is the main information passed in each transaction?
Section 6

SCSI Over SBP-2
Subjects Covered

Relationship of SCSI CDB, ORB and 1394 packet
SCSI status block
Messages
RBC
SCSI on 1394

Defined by SAM (SCSI Architectural Model)

SCSI-3 Transport via SBP-2

Serial Bus Protocol (The last two hours)

1394

Sect 6: SCSI over SBP-2
Relationship between 1394, SBP-2 and SCSI

SCSI CDB
- Op Code
- Op Code
- Dependent Bytes
- Control Byte

SBP-2 ORB
- Next ORB
- Data Descriptor
- Control Information
- Command Set
- Dependent Information

1394 Packet
- Dest ID
- Source ID
- Tcode
- Rcode
- Header CRC
- Data
- Data CRC

1394
Sect 6: SCSI over SBP-2
Using SCSI On 1394

Use Config ROM to find Management Agent Address

Login In with Management Agent
   Get a Login ID
   Locate Command Agent

Build ORB List

Write ORB List Address to Command Agent

(Watch Status FIFO for completion)

Add to List

Ring Door Bell
SCSI Status

Request Sense Command not needed
  Status returned for each ORB
  No contingent allegiance!

SBP-2  Adopted SCSI Status Format
  Sense Key, ASC, ASC-Q in Status Block

Can use Notify bit to reduce Status Traffic
this page is intentionally blank
Status Block

Note: If there is no error, the target need only post the first two quadlets of status.
Status Block Definitions

SRC

00b = Solicited Status, not end of list
01b = Solicited Status, next ORB = Null
10b = Unsolicited Status
11b = Unsolicited Status, ISOCH Error

Resp  Response.

0 = Request complete. The request completed without transport protocol error.
1 = Transport failure. Target detected nonrecoverable transport error.
2 = Illegal request. Unsupported bit or field in ORB
3 = Vendor dependent.

Len  Length. Number of valid quadlets -1 stored as status

SBP status

0 = No additional sense to report
1 = Invalid request type
2 = Speed not supported
3 = Page size not supported
4 = Access denied
5 = Logical unit not supported
6 = Maximum payload too small
7 = Too many channels
8 = Resources unavailable
9 = Function rejected
A = Login ID not recognized
FF = unspecified error
Status Block Definitions (continued)

sfmt  Status format
      0 = Current error (SCSI error code 70)
      1 = Deferred error (SCSI error code 71)
      2 = Reserved
      3 = Vendor dependent format

Status  This is the command set status (SCSI/ATA/ATAPI)
        0 = Good
        2 = Check condition
        4 = Condition met
        8 = Busy
        10h = Not supported by SBP-2 devices
        14h = Not supported by SBP-2 devices
        18h = Reservation conflict
        22h = Command terminated
        28h = Not supported by SBP-2 devices
        30h = Not supported by SBP-2 devices
        All other values are reserved for future standardization

v  The information stored in the Information quadlet is valid

m, e, l  File Mark, end of medium, incorrect length indicator are defined in the applicable command set standards
Status Block Definitions (continued)

<table>
<thead>
<tr>
<th>Sense Key</th>
<th>Sense key;</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No sense</td>
</tr>
<tr>
<td>1</td>
<td>Not ready</td>
</tr>
<tr>
<td>2</td>
<td>Recovered error</td>
</tr>
<tr>
<td>3</td>
<td>Medium error</td>
</tr>
<tr>
<td>4</td>
<td>Hardware error</td>
</tr>
<tr>
<td>5</td>
<td>Illegal request</td>
</tr>
<tr>
<td>6</td>
<td>Unit attention</td>
</tr>
<tr>
<td>7</td>
<td>Data protection</td>
</tr>
<tr>
<td>8</td>
<td>Blank check</td>
</tr>
<tr>
<td>9</td>
<td>Vendor dependent</td>
</tr>
</tbody>
</table>

Ah = Not supported by SBP-2 devices
Bh = Aborted command
Ch = Not supported by SBP-2 devices
Dh = Volume overflow
Eh = Miscompare
Fh = Reserved for future standardization

All other fields Defined by command set standards
No More Messages

Identify Message
- Performed with LUN on Login
- Each LUN has separate Login ID
  (Possible separate Command Agent)

Tagged Queuing
- Each ORB tagged with ORB Address
- Device can be Ordered or Unordered
- No mechanism for Ordered Subsequence

Disconnect/Reconnect
- Packetized Protocol handles

Address Pointers
- Overwrite or Re-Read
SAM Features Not Supported

Asynchronous Event Notification
(SBP-2 does support unsolicited status)

Soft Reset

Untagged Tasks

Linked Commands (or Flag)

NACA  BIT
1394 Reduced Block Commands (RBC)

SCSI Device Type = 0E

Subset of 18 SCSI commands for magnetic recording block devices
Both fixed and removable devices

Based on SCSI Block Commands (SBC) & SCSI Primary Commands (SPC)
Restricts options and parameters

Initial transport = 1394 with SBP-2 mapping

Proposal to ANSI committee October 1997
# Reduced Block Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>OP Code</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format Unit</td>
<td>04h</td>
<td>RBC</td>
</tr>
<tr>
<td>Inquiry</td>
<td>12h</td>
<td>SPC-2</td>
</tr>
<tr>
<td>Mode Select</td>
<td>15h</td>
<td>SPC-2</td>
</tr>
<tr>
<td>Mode Sense</td>
<td>1Ah</td>
<td>SPC-2</td>
</tr>
<tr>
<td>Persistent Reserve In</td>
<td>5Eh</td>
<td>SPC-2</td>
</tr>
<tr>
<td>Persistent Reserve Out</td>
<td>5Fh</td>
<td>SPC-2</td>
</tr>
<tr>
<td>Prevent/Allow Media Removal</td>
<td>1Eh</td>
<td>SPC-2</td>
</tr>
<tr>
<td>Read (10)</td>
<td>28h</td>
<td>RBC</td>
</tr>
<tr>
<td>Read Capacity</td>
<td>25h</td>
<td>RBC</td>
</tr>
<tr>
<td>Release</td>
<td>17h</td>
<td>SPC-2</td>
</tr>
<tr>
<td>Request Sense</td>
<td>03h</td>
<td>SPC-2</td>
</tr>
<tr>
<td>Reserve</td>
<td>16h</td>
<td>SPC-2</td>
</tr>
<tr>
<td>Start/Stop Unit</td>
<td>1Bh</td>
<td>RBC</td>
</tr>
<tr>
<td>Synchronize Cache</td>
<td>35h</td>
<td>RBC</td>
</tr>
<tr>
<td>Test Unit Ready</td>
<td>00h</td>
<td>SPC-2</td>
</tr>
<tr>
<td>Verify</td>
<td>2Fh</td>
<td>RBC</td>
</tr>
<tr>
<td>Write (10)</td>
<td>2Ah</td>
<td>RBC</td>
</tr>
<tr>
<td>Write Buffer</td>
<td>3Bh</td>
<td>SPC-2</td>
</tr>
</tbody>
</table>

**Notes:**
- Read (6) and Write (6) are not included
- Request Sense optional because of Auto Sense
- Details of commands provided in Appendix B
Asynchronous Event Notification
SCSI-2 (AEN)

Asynchronous Event Reporting
SCSI-3 (AER)

Unsolicited Status Sense
RBC

Device returns a Status Block without an ORB request
U (Unsolicited Status) bit in Status Block = 1

This Reports:
Unsolicited Status Sense
Power Management Class Event
Media Class Event
Device Busy Class Event
Unsolicited Status - Determining What Happened

<table>
<thead>
<tr>
<th>SRC</th>
<th>Resp</th>
<th>R</th>
<th>Len</th>
<th>SBP Status</th>
<th>ORB Offset Hi</th>
<th>ORB Offset Lo</th>
<th>sfmt</th>
<th>Status = 02h</th>
<th>v</th>
<th>m</th>
<th>e</th>
<th>i</th>
<th>sense key</th>
<th>ASC</th>
<th>ASCQ</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Sense Keys</th>
<th>ASC</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2h</td>
<td>04h</td>
<td>Device Not Ready</td>
</tr>
<tr>
<td>6h</td>
<td>28h</td>
<td>Not ready to Ready Transition</td>
</tr>
<tr>
<td>6h</td>
<td>29h</td>
<td>Power on Reset, bus reset, etc.</td>
</tr>
<tr>
<td>6h</td>
<td>7Eh</td>
<td>Notification of an Event</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASCQ = 02hPower Management Class Event</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASCQ = 04hMedia Class Event</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASCQ = 06hDevice Busy Class Event</td>
</tr>
</tbody>
</table>
# RBC Power Management Information

<table>
<thead>
<tr>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td>Status</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

## Event
- **00h** - No power state change
- **01h** - Device successfully change to the specified power state
- **02h** - Device failed to enter the last requested requested power state
- **03** - FFh - Reserved

## Status
- **00h** - Reserved
- **01h** - Action State
- **02h** - Idle State
- **03h** - Standby State
- **04** - FFh - Reserved
RBC
Media Event

<table>
<thead>
<tr>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td>Status</td>
<td>Start slot</td>
<td>End slot</td>
</tr>
</tbody>
</table>

**Event**

- 00h - Media status is unchanged
- 01h - Eject request
- 02h - Specified slot has new media
- 03h - Media has been removed from specified slot - requires user intervention
- 04 - FFh - Reserved

**Status**

Bit 1 - Media present
- 0 - Door or Tray open
- 2-7 - Reserved
RBC
Device Busy Event

<table>
<thead>
<tr>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td>Status</td>
<td>Time (MSB)</td>
<td>Time (LSB)</td>
</tr>
</tbody>
</table>

**Event**
- 00h - No event is available
- 01h - Timeout occurred
- 02h - FFh - Reserved

**Status**
- 00h - No event, Device ready to accept commands
- 01h - Device waking up
- 02h - Device completing an earlier command
- 03h - Device is completing a deferred operation
- 04h - FFh - Reserved
Review

1. Define how SCSI sense data in mapped into the status block.
2. What is the benefit of RBC?
3. How is the SCSI CDB mapped into the 1394 packet?
4. Which t code will be used to move the SCSI CDB?
SCSI Over SBP-2 Notes
Section 7

ATA Over SBP-2
Subjects Covered

IDE/ATA/ATAPI registers
Tailgate
Bridge
Byte ordering
ATA Or IDE

ATA = AT Bus Attachment
 Name of the ANSI Standard (X3T13 committee)

IDE = Integrated or Intelligent Drive Electronics
 Popular name in the industry

Physical: 40 pin ribbon cable
 Supports 2 devices max per cable
 18 inches maximum

Logical: Micro processor has direct access to control registers
 PIO = Programmed Input Output
 DMA = Direct Memory Access
ATA Task File

ATA Drive

Task File Registers

IF0: DATA*
IF1: FEATURES
IF2: SECT CNT
IF3: SECT NUM
IF4: CYL LOW
IF5: CYL HIGH
IF6: DEV/HEAD
IF7: CMND/STAT

*Access 16 bits wide

To IDE Bus

To Host
Example ATA Read Command

Device Reads Task File
Device Determines Physical Location
Device Seeks To Location
Device Reads Data
Device Checks ECC
Device Sets Status in 1F7h
DeviceInterrupts BIOS

BIOS writes 23h into 1F4h
BIOS writes 01h into 1F5h
BIOS writes 04h into 1F6h
BIOS writes 05h into 1F3h
BIOS writes 01h into 1F2h
BIOS writes 20h into 1F7h

BIOS checks status in 1F7h
BIOS reads data from 1F0h
BIOS reads data from 1F0h

BIOS reads data from 1F0h
ATA Packet Interface

SCSI Command Packets sent over ATA

Popular method for interfacing to CD-ROMs

New ATA Command: Packet Command (A0h)

"Here, execute this SCSI Command" command

Packet = 12 or 16 Byte SCSI Command
ATAPI Command Process

Step 1
Packet Command

Step 2
Send Packet

12 or 16 Bytes

Task File

INT.REASON
BYTE CNT
BYTE CNT
CMND A0

Task File

DATA

1394

Sect 7: ATA Over SBP-2
Using Native ATA or ATAPI Devices

Host CR uses normal ATA or ATAPI commands

1394 SBP-2

Tailgate

ATA Interface

Native ATA or ATAPI Device

Translates between 1394 SBP-2 and ATA
Separate board, chip on device, embedded in device controller
Tailgate Characteristics

- Low cost
- Does not support isochronous
- Allows only a single login to each logical unit
- Supports either 1 or 2 logical units
- PIO block commands (Read/Write Multiple) not supported
- Read and Write Long not supported
ATA Command ORB

Next ORB

Data Descriptor

device head  features  sector count  sector number

command  reserved

cylinder low  cylinder high

reserved  block count  reserved  reserved  n, p, a, t

SBP-2

ATA

© KnowledgeTek, Inc.
rev 5.41c 15.H.09
# ATA ORB Definitions

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Next ORB</td>
<td>Address of next command ORB in the chain</td>
</tr>
<tr>
<td>Data Descriptor</td>
<td>Serial Bus address of data source/destination (quadlet aligned)</td>
</tr>
<tr>
<td>n</td>
<td>Notify</td>
</tr>
<tr>
<td>rq fmt</td>
<td>0 = SBP-2 command format (1 &amp; 2 not defined)</td>
</tr>
<tr>
<td></td>
<td>3 = Dummy or ABORT command</td>
</tr>
<tr>
<td>d</td>
<td>Direction: 0 = use SPB-2 read; 1 = use SBP-2 write</td>
</tr>
<tr>
<td>spd</td>
<td>0=S100, 1=S200, 2=S400</td>
</tr>
<tr>
<td>max payload</td>
<td>Maximum data length per packet</td>
</tr>
<tr>
<td>p</td>
<td>1 = Use page tables</td>
</tr>
<tr>
<td>page size</td>
<td>Page table size</td>
</tr>
<tr>
<td>data size</td>
<td>Data length, quadlet multiple</td>
</tr>
<tr>
<td>rt</td>
<td>0 = Execute command and return status</td>
</tr>
<tr>
<td></td>
<td>1 = Return status only (don’t write task file except to select device)</td>
</tr>
<tr>
<td>pd</td>
<td>0 = PIO; 1 = DMA</td>
</tr>
<tr>
<td>at</td>
<td>1 = ATA command</td>
</tr>
</tbody>
</table>

© KnowledgeTek, Inc.
rev 5.41c 15.H.09

1394
Sect 7: ATA Over SBP-2
ATAPI Command ORB

SBP-2

ATAPI

Next ORB

Data Descriptor

<table>
<thead>
<tr>
<th>n</th>
<th>rq</th>
<th>fmt</th>
<th>r</th>
<th>d</th>
<th>spd</th>
<th>max payload</th>
<th>p</th>
<th>page size</th>
<th>data size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CDB-0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CDB-1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CDB-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CDB-3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CDB-4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CDB-5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CDB-6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CDB-7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CDB-8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CDB-9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CDB-10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>reserved</td>
</tr>
</tbody>
</table>

SCSI CDBs
ATAPI ORB Definitions

Next ORB: Address of next command ORB in the chain
Data Descriptor: Serial Bus address of data serve/destination (quadlet aligned)
n: Notify
rq fmt: 0 = SBP-2 command format (1 & 2 not defined)
3 = Dummy or ABORT command
d: Direction: 0 = use SPB-2 read; 1 = use SBP-2 write
spd: 0=S100, 1=S200, 2=S400
max payload: Maximum data length per packet
p: 1 = Use page tables
page size: Page table size
data size: Data length, quadlet multiple
rt: 0 = Execute command and return status
1 = Return status only (don’t write task file except to select device)
pd: 0 = PIO; 1 = DMA
at: 0 = ATAPI command
Tailgate Status Block

![Diagram of Tailgate Status Block]

If a failure occurred before command completed  
SBP-2 Info contains the relevant information  
If the failure was at the Tailgate or Device level  
Tailgate Info contains the relevant information
### Status Block Definitions

<table>
<thead>
<tr>
<th>SRC</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00b</td>
<td>Solicited Status, not end of list</td>
<td></td>
</tr>
<tr>
<td>01b</td>
<td>Solicited Status, next ORB = Null</td>
<td></td>
</tr>
<tr>
<td>10b</td>
<td>Unsolicited Status</td>
<td></td>
</tr>
<tr>
<td>11b</td>
<td>Reserved</td>
<td></td>
</tr>
</tbody>
</table>

#### ORB Offset
Identifies ORB for this status

#### Resp
Response

- **0** = Request complete. The request completed without transport protocol error.
- **1** = Transport failure. Target detected nonrecoverable transport error.
- **2** = Illegal request. Unsupported bit or field in ORB.
- **3** = Vendor dependent.

#### r
Reserved (set to 0)

#### v
Length. Number of valid quadlets -1 stored as status
SBP Status

Indicates status from the transport level:

0 = No additional sense to report
1 = Invalid request type
2 = Speed not supported
3 = Page size not supported
4 = Access denied
5 = Logical unit not supported
6 = Maximum payload too small
7 = Too many channels
8 = Resources unavailable
9 = Function rejected
A = Login ID not recognized
FF = unspecified error

If anything other than 0, Tailgate Info will be 0
# Tailgate Status

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0h</td>
<td>No error</td>
</tr>
<tr>
<td>1h</td>
<td>Data size not exact (informative)</td>
</tr>
<tr>
<td>2h</td>
<td>No ATAPI command phase</td>
</tr>
<tr>
<td>3h</td>
<td>Busy at start of command</td>
</tr>
<tr>
<td>4h</td>
<td>Task aborted</td>
</tr>
<tr>
<td>5h</td>
<td>Task set aborted</td>
</tr>
<tr>
<td>6h</td>
<td>Tailgate reset has completed</td>
</tr>
<tr>
<td>7h - FEh</td>
<td>Reserved</td>
</tr>
<tr>
<td>FFh</td>
<td>Other protocol errors</td>
</tr>
</tbody>
</table>
ATA Map 02

New mapping to replace Tailgate

Uses SCSI Host Driver

Bridge Device translates SCSI commands to ATA commands for ATA devices, and passes SCSI commands for ATAPI devices.
## Command Mapping

<table>
<thead>
<tr>
<th>SCSI Command</th>
<th>ATA Command</th>
<th>Op Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode Select (10)</td>
<td>Idle</td>
<td>E3</td>
</tr>
<tr>
<td>Mode Sense (10)</td>
<td>Identify Device</td>
<td>N/A</td>
</tr>
<tr>
<td>Read (10)</td>
<td>Read DMA or</td>
<td>C8 or</td>
</tr>
<tr>
<td></td>
<td>Read Sectors</td>
<td>20</td>
</tr>
<tr>
<td>Start/Stop Unit</td>
<td>Seek &amp; Standby</td>
<td>70 &amp;</td>
</tr>
<tr>
<td></td>
<td>Immediate</td>
<td>E0</td>
</tr>
<tr>
<td>Synchronize Cache</td>
<td>Flush Cache</td>
<td>E7</td>
</tr>
<tr>
<td>Test Unit Ready</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Write &amp; Verify (10)</td>
<td>Write Verify</td>
<td>3C</td>
</tr>
<tr>
<td>Write Buffer</td>
<td>Download Microcode</td>
<td>92</td>
</tr>
<tr>
<td>Write (10)</td>
<td>Write DMA or</td>
<td>CA or</td>
</tr>
<tr>
<td></td>
<td>Write Sectors</td>
<td>30</td>
</tr>
</tbody>
</table>
### Status

<table>
<thead>
<tr>
<th>Sfmt</th>
<th>Status</th>
<th>V</th>
<th>M</th>
<th>E</th>
<th>I</th>
<th>Sense Key</th>
<th>ASC</th>
<th>ASCQ</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>00 - only 1st 2 quadlets sent</td>
</tr>
<tr>
<td></td>
<td>02</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>02 - check condition</td>
</tr>
<tr>
<td></td>
<td>08</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>08 - busy</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>V</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>M &amp; E</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ATAPI = 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ATA = attempted to move more than 256 blocks</td>
</tr>
<tr>
<td></td>
<td>Sense key</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ATAPI from request sense</td>
</tr>
<tr>
<td></td>
<td>ASC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ATAPI from request sense</td>
</tr>
<tr>
<td></td>
<td>ASCQ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ATAPI from request sense</td>
</tr>
<tr>
<td></td>
<td>Information</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

- **Sfmt**: Indicates the format of the request.
- **Status**: Represents the status of the request.
- **V**: Controls the verification of the request.
- **M & E**: Enables or disables the error checking.
- **I**: Indicates whether the ATAPI is active.
- **Sense Key**: Specifies the type of sense data.
- **ASC**: Specifies additional sense code.
- **ASCQ**: Specifies additional sense quality code.
- **Information**: Provides additional information about the request.
### Status Sense Key And Code Meanings

<table>
<thead>
<tr>
<th>Sense Key</th>
<th>ASC</th>
<th>ASCQ</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>04</td>
<td>00</td>
<td>Device not powered</td>
</tr>
<tr>
<td>6</td>
<td>29</td>
<td>00</td>
<td>Unit attention, reset</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td>00</td>
<td>LUN not 0</td>
</tr>
<tr>
<td>5</td>
<td>24</td>
<td>00</td>
<td>LBA or Transfer Length out of range or Rel Addr or Byte clk bits set</td>
</tr>
<tr>
<td>2</td>
<td>04</td>
<td>01</td>
<td>DRDY bit set before issuing cmd</td>
</tr>
<tr>
<td>2</td>
<td>04</td>
<td>01</td>
<td>BSY bit set before issuing cmd</td>
</tr>
<tr>
<td>B</td>
<td>00</td>
<td>00</td>
<td>Transport failure during cmd execution</td>
</tr>
<tr>
<td>4</td>
<td>00</td>
<td>00</td>
<td>ERR &amp; ICRC set at the completion of cmd</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>00</td>
<td>ERR &amp; UNC set at the completion of cmd</td>
</tr>
<tr>
<td>3</td>
<td>21</td>
<td>00</td>
<td>ERR &amp; IDNF set at the completion of cmd</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>00</td>
<td>ERR &amp; AMNF set at the completion of cmd</td>
</tr>
<tr>
<td>3</td>
<td>00</td>
<td>00</td>
<td>DF was set at the completion of the cmd</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>00</td>
<td>ABRT was set at the completion of the command</td>
</tr>
</tbody>
</table>
Review

1. Define what a tailgate is in reference to 1394
2. Contrast the tailgate and the bridge