Serial Bus - IEEE 1394

Overview

automotive network
home cabling

Dipl. Ing. Matthias Rabel

Prof. Dr. H.P. Großmann
Department of Information Resource Management
University of Ulm, Germany
What is IEEE 1394

- 1394 is a peer-to-peer bus.
- Max. 63 nodes on bus.
- Automatic assignment of node addresses
- Variable speed data transmission from 25 and 50 Mbit/s for backplanes to 100, 200 and 400 Mbit/s for the cable medium
- The cable medium allows up to 16 physical connections (cable hops) each of up to 4.5 meters, giving a total cable distance of 72 meters between any two devices. Bus management recognizes smaller configurations to optimize performance.
- A fair bus access mechanism.
IEEE 1394 - Topology

Example
Serial Bus - Physical Topology
1394 Supports Isochronous data transfers

- Isochronous means just in time data
- A guaranteed bandwidth is provided by the bus.
- Isochronous data is NOT guaranteed to be either valid or arrive at the destination.

And Supports asynchronous data transfers

- No guaranteed bandwidth
- Link Layer guarantees arrival and validation.
1394 is hot plugable

- When a new node is connected the bus automatically re-configures itself. Each node is dynamically re-numbered.
- Isochronous data transmission survives a bus-reset.
- Dynamic node allocation requires software to “re-find” devices after a bus-reset. - Busmanagement and Transactionlayer.
Peripheral power

- 8-40 VDC carried by cable
- Total available power is system dependent
- Node power requirements must be declared in configuration ROM
- Cable system allows up to 1.5 A (60 watts) per link
- Nodes can either source or sink power
- Multiple power sources on one bus provide additional flexibility
IEEE 1394 - Overview

Extensions to the Specification

- 1394.a corrigenda and minor redefinition
- 1394.b extensions to Gigabit signaling and additional Physical Layer:
  - POF, CAT5 UTP, HPCF
- 1394.1 Serial Bus Bridges
- 1394.2 Serial Express - longer distances and faster
- 1394.OHCI Adaptations to OHCI Spec.
Bus Initialization - Timeline

- Tree Identify
- Self Identify
- Prior cycle master resumes...
- Reallocation of prior isochronous resources
- Incumbent bus manager active...
- Bus manager active...
- 125ms
- 625ms
- Isochronous resource manager active...
- Allocation of new iso resources.
- 1000 ms
Bus Reset Statemachine

R0: Reset Start
reset_start_actions

- bus_reset_signal_received()
  - All: R0a
    PH_STATE.ind(BUS_RESET_START);
    initiated_reset = FALSE;
  - All: R0b
    PH_STATE.ind(BUS_RESET_START);
    initiated_reset = TRUE;

R1: Reset Wait
reset_wait_actions

- arb_timer >= RESET_TIME
  - R0: R1
    reset_complete()
    - R1: T0
      R1: R0

T0: Tree-ID Start

LOMI Universität Ulm, 2000
Tree Identify Statemachine

T0: Tree-ID Start
   tree_ID_start_actions
   (child_count == NPORT) ||
   (((arb_timer >= (force_root ?
     FORCE_ROOT_TIMEOUT : 0)) &&
   (child_count == NPORT - 1))
   T0:T1
   R1:T0
   from R1: Reset
   Wait
   (arb_timer >= CONFIG_TIMEOUT)
   && (child_count < NPORT - 1)
   T0:T0
   signal PH_STATE.ind
   (CONFIG_TIMEOUT);

T1: Child Handshake
   child_handshake_actions
   child_handshake_complete();
   T1:T2
   T2:T3
   (portR(parent_port) ==
   RX_PARENT_NOTIFY) &&
   (arb_timer > contend_time)
   T3:T1
   root_contend_actions
   child[parent_port] = true;
   /* node is now root */
   portR(parent_port) ==
   RX_ROOTCONTENTION
   T2:S0
   to S0: Self-ID Start
   (portR(parent_port) ==
   RX_PARENT_HANDSHAKE))
   T2:T3
   portT(parent_port,
   TX_PARENT_NOTIFY);
Bus Identification

Example configuration after a bus initialization.

What happens?

• Topology information is cleared.
• Tree Identify
• Self Identify

Leaf nodes start with signal parent_notify
Response with signal child_notify and forward of parent_notify

In this example a collision occurred between node B and D.

Now a process starts called „root contention“:

- Withdraw of signal parent_notify
- Start of timers with random length
- After that check for signal from other node.
- If there is nothing set signal parent_notify else ... (See next page)

In case the signals collide again, process starts again
... Response with signal child_notify.

The node with only child ports is the root node.

Node D has now all ports labeled and drops its signals.

As node B doesn’t see the parent_notify signal any longer it drops also its child_notify signals.

Tree Identify is now completed
Self-Identify State Machine

S0: Self-ID Start
self_ID_start_actions();

S1: Self-ID Grant
self_ID_grant_actions();

S2: Self-ID Receive
self_ID_receive_actions();

S3: Send Speed Capabilities

S4: Self-ID Transmit
self_ID_transmit_actions();

portRspeed(parent_port) == S200
S4:S4a
max_peer_speed[parent_port] = S200;

portRspeed(parent_port) == S400
S4:S4b
max_peer_speed[parent_port] = S400;

self_ID_complete &
(root ll portR[parent_port] ==
RX_DATA_PREFIX)
S4:A0

S4: Self-ID Transmit
self_ID_transmit_actions();

portRspeed(parent_port) == S200
S4:S4a
max_peer_speed[parent_port] = S200;

portRspeed(parent_port) == S400
S4:S4b
max_peer_speed[parent_port] = S400;

S1: Self-ID Grant
self_ID_grant_actions();

all_child_ports_identified
S1:S4
max_peer_speed[parent_port] = S100;

S2: Self-ID Receive
self_ID_receive_actions();

portR(lowest_unidentified_child) ==
RX_DATA_PREFIX
S1:S2
receive_port = lowest_unidentified_child;

portR(parent_port) == RX_DATA_PREFIX
receive_port = parent_port;

(portR(receive_port) == IDLE) || (portR(receive_port) == RX_SELF_ID_GRANT)
S2:S0a
PHY_SPEED == S100 &
portR(receive_port) == RX_IDENT_DONE
S2:S0b
child_ID_complete[receive_port] = true;

S3: Send Speed Capabilities
arb_timer >= SPEED_SIGNAL_LENGTH
S3:S0
portTspeed(receive_port, S100);

portRspeed(receive_port) == S200
S3:S3a
max_peer_speed[receive_port] = S200;

portRspeed(receive_port) == S400
S3:S3b
max_peer_speed[receive_port] = S400;

portRspeed(receive_port) == S200
S3:S3a
max_peer_speed[receive_port] = S200;

portRspeed(receive_port) == S400
S3:S3b
max_peer_speed[receive_port] = S400;
Root node initiates the Self Identify process in a recursive manner.

After node A detects the signal grant it distributes its self id information across the bus.

And when it is finished it sends signal ident_done.

Node A is now finished and continues with normal operation. As long there is no other information on the bus the parent nodes to each ready node will send a data_prefix to keep them quiet.
The recursive Self identification is done when the last node, which is the root node has send its self id information.

Now nodes can arbitrate on the bus.
• Node A and E start arbitration at the same time

• Node D forwards the request to node B and denies access to its other children

• Node B does the same

• Since node B is root it doesn’t have to forward the request
Root grants access the first request while the other nodes acknowledge the denial by withdrawing their request and passing on the denial.

Node A sends the data prefix to warn all others that data is about to send.

At this point all physical connections point away from the node that won arbitration. This allows the second unused signal to be turned around and used as strobe to time the transmission of data.
Fair arbitration is based on a fairness interval

All nodes arbitrating send a request to the root which grants or denies access

Isochronous and asynchronous arbitration works the same way

Isochronous subactions are send immediately - that's why the channels are transmitted first
Urgent arbitration

3/4 of cycle time can be used for nodes using urgent arbitration

Note: Urgent arbitration is only used in backplane environment not on the cable media

This arbitration is used with small packets, because, these packets with a normal arbitration would increase enormously the LATENCY.
Cycle Structure

Cycle master maintains common clock source and generates cycle start packets at intervals set by a „cycle synch“ source (8 kHz ≈ 125 µs).

If a transfer is in progress on a cycle synch event, the cycle start packet is delayed.
Data Traffic - Timing implementation

nominal cycle 125 μs

packet # ; source data (1 source, 1 iso channel)

time stamp

TX

<

c 1
c 2 3

offset as content in cycle start packet

RX

<
c

c

c

LOMI Universität Ulm, 2000
Subactions and Gaps

Structure of isochronous subactions on the bus

Structure of asynchronous subactions on the bus
Nodes can concatenate subactions

- It can send its subaction immediately after an acknowledge packet
- If one node transmits more than one channel

This increases usage of timeframe

**Asynchronous concatenation**

**Isochronous Concatenation**
IEEE 1394 - Plug

Max. Length 4.5 m

smaller cable over longer distance
Serial Bus - Signaling

Data

1 0 1 1 0 0 0 1

Strb

Data xor Strb (delayed)

Data-strobe encoder example

Data
Clock
Data_Tx
Strb_Tx

data-strobe decoder example

Data_Rx
Clock = Data_Rx xor Strb_Rx (delayed)

Data_1 (rising edge of Clock)
Data_0 (falling edge of Clock)
• Clock = Data xor Strb

• Strb instead of Clock because of Quality.

• In one period of the clock two bits are sent
### Tree Identify - Statemachine - Kommunikation (Beispiel)

<table>
<thead>
<tr>
<th>Zustand</th>
<th>K1</th>
<th>K2</th>
<th>K3</th>
<th>K4</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td>ZZ ZZ ZZ</td>
<td>ZZ ZZ ZZ</td>
<td>T0</td>
<td>ZZ ZZ ZZ</td>
</tr>
<tr>
<td>T0:T1</td>
<td>T0</td>
<td>T0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T1</td>
<td>0Z 0Z 0Z</td>
<td>ZZ 0Z 0Z</td>
<td>T0</td>
<td>ZZ ZZ ZZ</td>
</tr>
<tr>
<td>T1:T2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td>0Z 0Z 0Z</td>
<td>0Z 0Z 0Z</td>
<td>T1</td>
<td>0Z 0Z 0Z</td>
</tr>
<tr>
<td>T2:T3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>ZZ ZZ ZZ</td>
<td>ZZ ZZ ZZ</td>
<td>ZZ ZZ ZZ</td>
<td>ZZ ZZ ZZ</td>
</tr>
<tr>
<td>T3:T4</td>
<td>T1</td>
<td>T1</td>
<td>T1</td>
<td>T1</td>
</tr>
<tr>
<td>T4</td>
<td>1Z 1Z 1Z</td>
<td>0Z 0Z 0Z</td>
<td>T2</td>
<td>0Z 0Z 0Z</td>
</tr>
<tr>
<td>T4:T5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T5</td>
<td>ZZ ZZ ZZ</td>
<td>ZZ ZZ ZZ</td>
<td>ZZ ZZ ZZ</td>
<td>ZZ ZZ ZZ</td>
</tr>
</tbody>
</table>

###树状结构 - 电气

<table>
<thead>
<tr>
<th>Zustand</th>
<th>K1</th>
<th>K2</th>
<th>K3</th>
<th>K4</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td>ZZ ZZ ZZ</td>
<td>ZZ ZZ ZZ</td>
<td>ZZ ZZ ZZ</td>
<td>ZZ ZZ ZZ</td>
</tr>
<tr>
<td>T0:T1</td>
<td>T0</td>
<td>T0</td>
<td>T0</td>
<td>T0</td>
</tr>
<tr>
<td>T1</td>
<td>ZZ ZZ ZZ</td>
<td>ZZ ZZ ZZ</td>
<td>ZZ ZZ ZZ</td>
<td>ZZ ZZ ZZ</td>
</tr>
<tr>
<td>T1:T2</td>
<td>T1</td>
<td>T1</td>
<td>T1</td>
<td>T1</td>
</tr>
<tr>
<td>T2</td>
<td>ZZ ZZ ZZ</td>
<td>ZZ ZZ ZZ</td>
<td>ZZ ZZ ZZ</td>
<td>ZZ ZZ ZZ</td>
</tr>
<tr>
<td>T2:T3</td>
<td>T2</td>
<td>T2</td>
<td>T2</td>
<td>T2</td>
</tr>
<tr>
<td>T3</td>
<td>ZZ ZZ ZZ</td>
<td>ZZ ZZ ZZ</td>
<td>ZZ ZZ ZZ</td>
<td>ZZ ZZ ZZ</td>
</tr>
<tr>
<td>T3:T4</td>
<td>T3</td>
<td>T3</td>
<td>T3</td>
<td>T3</td>
</tr>
<tr>
<td>T4</td>
<td>ZZ ZZ ZZ</td>
<td>ZZ ZZ ZZ</td>
<td>ZZ ZZ ZZ</td>
<td>ZZ ZZ ZZ</td>
</tr>
<tr>
<td>T4:T5</td>
<td>T4</td>
<td>T4</td>
<td>T4</td>
<td>T4</td>
</tr>
<tr>
<td>T5</td>
<td>ZZ ZZ ZZ</td>
<td>ZZ ZZ ZZ</td>
<td>ZZ ZZ ZZ</td>
<td>ZZ ZZ ZZ</td>
</tr>
</tbody>
</table>

###状态转换

- **TX_PARENT_NOTIFY**
- **RX_PARENT_NOTIFY**
- **TX_CHILD_NOTIFY**
- **RX_ROOT_CONTENTION**
- **RX_PARENT_HANDSHAKE**
- **IDLE**
Phy Packets

a) The self-ID packet  
b) The link on packet  
c) The PHY configuration packet

d) The extended PHY packets  
   phy ping packet  
   remote access packet  
   remote reply packet  
   remote command packet  
   remote confirmation packet  
   remote resume packet

IEEE 1394a:  
Asynchronous stream packet

phy ping packet  
remote access packet  
remote reply packet  
remote command packet  
remote confirmation packet  
remote resume packet
Example for a Phy packet

<table>
<thead>
<tr>
<th>transmitted first</th>
<th>phy_ID</th>
<th>0</th>
<th>L</th>
<th>gap_cnt</th>
<th>sp</th>
<th>del</th>
<th>c</th>
<th>pwr</th>
<th>p0</th>
<th>p1</th>
<th>p2</th>
<th>i</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>logical inverse of first quadlet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>self-ID packet #0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>transmitted last</td>
<td></td>
<td></td>
<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>
Link Layer - Primary Packet Format

Tcode - Transaction Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>write request for data quadlet</td>
</tr>
<tr>
<td>1</td>
<td>write request for data block</td>
</tr>
<tr>
<td>2</td>
<td>write response</td>
</tr>
<tr>
<td>3</td>
<td>reserved</td>
</tr>
<tr>
<td>4</td>
<td>read request for data quadlet</td>
</tr>
<tr>
<td>5</td>
<td>read request for data block</td>
</tr>
<tr>
<td>6</td>
<td>read response for data quadlet</td>
</tr>
<tr>
<td>7</td>
<td>read response for data block</td>
</tr>
<tr>
<td>8</td>
<td>cycle start request</td>
</tr>
<tr>
<td>9</td>
<td>lock request</td>
</tr>
<tr>
<td>A</td>
<td>isochronous data block</td>
</tr>
<tr>
<td>B</td>
<td>lock response</td>
</tr>
<tr>
<td>C-F</td>
<td>reserved</td>
</tr>
</tbody>
</table>
Link Layer - Asynchronous Packet Format

**tl** = Transaction Label **used in response subaction**

<table>
<thead>
<tr>
<th>rt_code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>retry_1: retry_A.retry_B supported and a new reservation is requested.</td>
</tr>
<tr>
<td>01</td>
<td>retry_X: retry_A.retry_B not supported or (if supported) reservation not requested.</td>
</tr>
<tr>
<td>10</td>
<td>retry_A: retry_A.retry_B supported and a reservation has been assigned (ack_busy_A previously returned)</td>
</tr>
<tr>
<td>11</td>
<td>retry_B: retry_A.retry_B supported and a reservation has been assigned (ack_busy_B previously returned)</td>
</tr>
</tbody>
</table>
Link Layer - Isochronous Packet Format

- Tag - Field describes format of contend
- Sy - Field can be used application specific

- Usually the data payload has an extra header called CIP containing timestamps and a exact content format description
### Link Layer - Acknowledge Packet

**Transmission Format:**
- **transmitted first:**
  - `ack_code`
  - `ack parity`
- **transmitted last:**

<table>
<thead>
<tr>
<th>ack_code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>reserved</td>
</tr>
<tr>
<td>1</td>
<td><code>ack_complete</code></td>
</tr>
<tr>
<td>2</td>
<td><code>ack_pending</code></td>
</tr>
<tr>
<td>3</td>
<td>reserved</td>
</tr>
<tr>
<td>4</td>
<td><code>ack_busy_X</code></td>
</tr>
<tr>
<td>5</td>
<td><code>ack_busy_A</code></td>
</tr>
<tr>
<td>6</td>
<td><code>ack_busy_B</code></td>
</tr>
<tr>
<td>7-C</td>
<td>reserved</td>
</tr>
<tr>
<td>D</td>
<td><code>ack_data_error</code></td>
</tr>
<tr>
<td>E</td>
<td><code>ack_type_error</code></td>
</tr>
<tr>
<td>F</td>
<td>reserved</td>
</tr>
</tbody>
</table>
Link-Layer Services

Requester Link Layer

Link Request

arbitration & packet transmission

Link Confirmation

Responder Link Layer

Link Indication

Link Response

not present for broadcast or isochronous packets
Transaction Layer - Unified Transaction
Transaction - Layer - Split Transactions

Other Link-Layer operations can take place between the two subactions.
Split Transaction using concatenated subactions
**Transaction Layer - Lock Model**

- Used for register manipulation

<table>
<thead>
<tr>
<th>Lock function</th>
<th>Update action</th>
</tr>
</thead>
<tbody>
<tr>
<td>mask_swap</td>
<td>new_value = data_value</td>
</tr>
<tr>
<td>compare_swap</td>
<td>if (old_value == arg_value) new_value = data_value;</td>
</tr>
<tr>
<td>fetch_add</td>
<td>new_value = old_value + data_value;</td>
</tr>
<tr>
<td>little_add</td>
<td>(little) new_value = (little) old_value + (little) data_value;</td>
</tr>
<tr>
<td>bounded_add</td>
<td>if (old_value != arg_value) new_value = old_value + data_value;</td>
</tr>
<tr>
<td>wrap_add</td>
<td>new_value = (old_value != arg_value) ? old_value + data_value : data_value;</td>
</tr>
</tbody>
</table>
differentiation of
- request
- response
packets

The “action”
states perform
the send actions,
including the
retry actions
Transaction Layer - Inbound Statemachine

Link Layer signals received packet

differentiation of incoming
  • requests
  and
  • responses

change to process state
Single phase

- signal busy to other incoming requests, as long a packet is not processed by the transaction layer (slow, due to its implementation in software)
- no queueing
Retry pending (unsuccessful sent) requests or responses up to maximum number (retry count)
Dual Phase Inbound and Outbound Retry Statemachines

Used to provide a kind of a fair access mechanism to nodes.

• While node is busy it sends retry code A,

• When the node becomes available it will accept transactions with retry code A, and sends retry code B to new transactions

• When all A transactions are completed it continues with the B transactions and sends retry code A
Busmanager

Functions:

a) Node Controller - CSR,

b) Isochronous resource manager
allocate and deallocate isochronous resources, channels and bandwidth
(1394.a for asynchronous streams only channel number)

c) Bus manager
activation of a cycle master, performance optimization,
power management, speed management and topology management
Repeater
   minimum capability, PHY may be powered from the bus
   a) have an active physical layer

Transaction capable
   node with an enabled link layer is transaction capable,
   b) active link layer,
   c) minimal configuration ROM

Isochronous capable
   Any node on a Serial Bus that can either send or receive isochronous packets is isochronous capable.
   d) free-running 24.576 MHz clock
   e) general ROM format.

Cycle master capable
   For any Serial Bus capable of isochronous operations, there shall be a cycle master.
   f) be able to generate cycle start events triggered by the 8 kHz
   g) broadcast cycle start packets,
Isochronous resource manager capable

For any Serial Bus capable of isochronous operations, there shall be an isochronous resource manager.

h) analyze received self-ID packets in order to correctly determine the physical ID of the isochronous resource manager node from all the candidates for this role

Bus manager capable

The most fully capable nodes on a Serial Bus are those that may become bus managers.

a) isochronous resource manager capable
b) provide advanced power management
c) provide facilities to optimize Serial Bus performance
d) describe the topology of the bus
e) cross-reference the maximum speed
Busmanager - Config ROM Format

Minimal ROM Format:

```
  01_16  
  _8     

  vendor_id  
  _24     
```

General ROM Format:

```
  info_length  
  _8        

  crc_length  
  _8        

  rom_crc_value  
  _16        

  bus_info_block

  root_directory

  unit_directories

  root & unit leaves

  vendor_dependent_information
```

Baseaddress at: xxxx FFFF F000 0400
Busmanager - Config ROM Format

Root directory
- Module_Vendor_Id entry
- Node_Capabilities entry
- Node_Unique_Id entry

Root leafs
- Node_Unique_Id leaf

Unit directory
- Unit_Power_Requirements entry
• Find exactly one cycle master – it has to be the root node
• If the actual root node is not cycle master capable
  • the force root flag of one cycle master capable not is set and
  • a bus reset will be done
• One bus manager or if not available one isochronous resource manager (irm) has to be found
• Every capable node checks self identify packets
  • the one with the lowest node Id will be the manager (nearest to root)
• The former (before the bus reset) active bus manager shall become again the bus manager
• The contention is done with lock requests to the bus manager register of the isochronous resource manager
What do we need:

- A common open standard.
- Something which is
  - Fast
  - Cheap
  - Easy to use
- Service integrating network
Project: Serial Bus as backbone in cars

Why not also in car?

Additional features required:

- Redundant connections
- Security features
- Special Gateways
IEEE 1394 - Chance

Today:
The PC evolves into a simpler easier to use appliance.

Future:
The PC evolves into a simpler easier to use appliance.
**Speed of data sources**

- **Uncompressed Video**
  - \((160\times120\text{ pixels})\times(24\text{ bits/pixel})\times(7\text{ fps})\) \(\approx 4\text{ Mbps}\)
  - \((640\times480\text{ pixels})\times(12\text{ bits/pixel})\times(30\text{ fps})\) \(\approx 120\text{ Mbps}\)

- **DV Compressed Video** \(\approx 30\text{ Mbps}\)

- **MPEG2** \(8\text{ Mbps (10.6 Mbps peak)}\)

- **ISDN 8 kHz x 8 bits** \(64\text{ kbps}\)

- **CD 44.1 kHz x 16 bits x 2 channels** \(1.4\text{ Mbps}\)

- **DAT 48 kHz x 16 bits x 2 channels** \(1.5\text{ Mbps}\)