ETSI TS 101 493-3 V1.1.1 (2000-09)

Technical Specification

**Broadband Radio Access Networks (BRAN);
HIPERLAN Type 2;
Packet based convergence layer;
Part 3: IEEE 1394 Service Specific Convergence Sublayer (SSCS)**

Reference

DTS/BRAN-0024004-3

Keywords

access, broadband, radio, HIPERLAN

Contents

Intellectual Property Rights

Foreword

This Technical Specification (TS) has been produced by ETSI Project Broadband Radio Access Networks (BRAN).

The present document defines the functionality required for interworking HIgh PErformance Radio Local Area Network Type 2 (HIPERLAN/2) with IEEE 1394 [6]. Separate ETSI documents provide details on the system overview, data link control layer, radio link control sublayer, other convergence sublayers and conformance testing requirements for HIPERLAN/2.

The present document is Part 3 of a multi-part deliverable covering the Packet based Convergence Layer of HIPERLAN/2, as identified below:

Part 1: "Common Part";

Part 2: "Ethernet Service Specific Convergence Sublayer (SSCS)";

Part 3: "IEEE 1394 Service Specific Convergence Sublayer";

Part 4: "IEEE 1394 bridge layer".

Part 1, Common Part, describes the functionality for adapting variable length packets/frames to the fixed size used in the Data Link Control (DLC) layer. Further parts, each defining a Service Specific Convergence Sublayer (SSCS), describe the functionality required to support a certain protocol, e.g. IEEE 1394 [6] or Ethernet protocols. The SSCSs all use the services of the Common Part and the DLC [2]. It is envisioned that several, independent, service specific parts will be defined in the future as market requirements develop. As a result, further parts may be added in the future.

1 Scope

The present document is applicable to HIPERLAN/2 equipment supporting interworking with 1394 buses.

The present document only addresses the functionality required to transfer IEEE 1394 [6] traffic over the radio interface between HIPERLAN/2 devices. It does not address the requirements and technical characteristics for wired network interfaces at a HIPERLAN/2 device.

The present document uses the services provided by the Common Part of the Packet based Convergence Layer and by the Data Link Control layers of HIPERLAN/2.

The present document does not address the requirements and technical characteristics for type approval and conformance testing. These are covered in separate Technical Specifications.

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies.
- A non-specific reference to an ETS shall also be taken to refer to later versions published as an EN with the same number.

[1] ETSI TS 101 493-1: "Broadband Radio Access Networks (BRAN); HIPERLAN 2; Packet based Convergence Layer; Part 1: Common Part".

[2] ETSI TS 101 761-2: "Broadband Radio Access Networks (BRAN); HIPERLAN Type 2; Data Link Control (DLC) Layer; Part 2: Radio Link Control (RLC) sublayer".

[3] ETSI TS 101 761-1: "Broadband Radio Access Networks (BRAN); HIPERLAN Type 2; Data Link Control (DLC) Layer; Part 1: Basic Data Transport Functions".

[4] ETSI TS 101 764-4: "Broadband Radio Access Networks (BRAN); HIPERLAN Type 2; Data Link Control (DLC) Layer; Part 4: Extension for Home Environment".

[5] IEC 61883 (1998): "Consumer audio/video equipment - Digital interface".

[6] IEEE Std 1394 (1995): "Standard for a High Performance Serial Bus".

[7] IEEE Std 1394a (2000): "IEEE Standard for a High Performance Serial Bus - Amendment 1".

3 Definitions, symbols and abbreviations

3.1 Definitions

For the purposes of the present document, the following terms and definitions apply:

1394 controller: 1394 application that establishes an isochronous connection on a Serial Bus.

1394 SSCS: abbreviation for IEEE 1394 SSCS, the contents of the present document.

HL2 1394 bridge layeraccess point: term access point used in the basic specifications is replaced by the term central controller throughout this document to reflect that in home environment multiple HL2 devices can act as the access point to a fixed network, whereas the whole HL2 network is still controlled by a single entity, the central controller.

bridge: pair of two closely coupled serial bus nodes, called portals, capable of connecting two buses in a Serial Bus net. The bridge selectively forwards asynchronous and isochronous packets according to route information contained within the portals.

bus_ID: 10-bit number uniquely specifying a particular bus within a Serial Bus net.

central controller (CC): provides control functionality for the DLC layer equivalent to that of an access point as defined by HIPERLAN/2, but is not necessarily attached to a fixed network. The central controller functionality may be embedded in a wireless device.

extended unique identifier (EUI-64): 64-bit unique number identifiers derived from the IEEE/RAC-provided *company_id* value These EUI-64 values are, by definition, unique among themselves.

HIPERLAN/2: HIgh PErformance Radio Local Area Network Type 2, a short-range wireless LAN providing broadband local access. Standardized by ETSI Project BRAN.

HL2 1394 bridge layer: refers to the HIPERLAN/2 IEEE 1394 bridge layer (see Bibliography).

HL2 Bus: protocols that provide 1394 asynchronous and isochronous services on a HL2 wireless network.

information element: message used to transfer convergence layer specific information and that is transferred transparently by RLC messages.

listener: device that can sink isochronous streams is called a listener.

mac_ID: each WT is assigned a MAC ID that is unique for the AP by the AP RLC entity during association. The MAC ID is coded with 8 bits.

mobile terminal: term mobile terminal used in the basic specifications is replaced by the term wireless terminal throughout this document to reflect that in home environment not all HL2 terminals need to be mobile.

node_ID: 16-bit address which distinguishes the node from other nodes in the system. The 10 most significant bits are the same for all nodes on the same bus; this is called the bus_ID. The 6 least-significant bits are unique for each node on the same bus; this is called the physical_ID.

octet: eight bits of data.

overlaid connection: IEC 61883 [5] specification defines this operation that allows adding additional listeners to an existing connection.

physical_ID: 6-bit number uniquely specifying a particular node on a 1394 bus.

portal: node that connects a bridge to a Serial Bus.

protocol data unit (PDU): data unit exchanged between entities at the same ISO layer.

quadlet: four bytes, or 32 bits, of data.

reserved codes: set of values for a defined field that are not used within the present document, but are reserved for future revisions of the present document. A reserved code shall not be generated or, upon development of a future standard, may be used as specified by such a standard. The recipient of a *reserved* code shall check its value and shall reject code values that were not defined at the time of its inception.

reserved fields: set of bits not defined by the present document, but reserved for future revisions of the present document. A reserved field shall be zeroed or, upon development of a future standard, set to a value specified by such a standard. The recipient of a *reserved* field shall not check its value.

service data unit (SDU): data unit exchanged between adjacent ISO layers.

talker: device that can source isochronous streams is called a talker.

wired 1394: refers to a wired bus as specified in IEEE Std 1394-1995 as amended by IEEE Std 1394a-2000, and supplemented with IEC 61883 [5].

wireless 1394: refers to a HL2 Bus as specified in the present document.

wireless bridge: bridge which a least one of its portals is wireless.

wireless cycle master: refers to the cycle master on the HL2 Bus.

wireless node: device that belongs to the HL2 Bus and that implements the 1394 SSCS without any bridging functions on top of it. A wireless node is identical to a wireless terminal. The term *wireless node* is used when describing HL2 Bus behaviour.

wireless portal: wireless node that implements the HL2 1394 bridge layer, in addition of the 1394 SSCS.

wireless terminal (WT): HL2 home device, which is not acting as a central controller for a subnet. A wireless terminal is identical to a wireless node which is not acting as a central controller. The term *wireless terminal* is used when describing WT behaviour compared to CC behaviour.

3.2 Symbols

For the purposes of the present document, the following symbols apply:

05_{16}	Hexadecimal notation
$0000\ 0101_2$	Binary notation

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

AP	Access Point
BRAN	Broadband Radio Access Networks (Project)
CC	Central Controller
CL	Convergence Layer
CPCS	Common Part Convergence Sublayer
C-SAP	Control Service Access Point
CSR	Control and Status Register
DLC	Data Link Control
DUC	DLC User Connection
ETSI	European Telecommunications Standards Institute
EUI-64	Extended Unique Identifier
HARP	HIPERLAN/2 Address Resolution Protocol
HL2	HIPERLAN/2
IE	Information Element
IEEE	Institute of Electrical and Electronics Engineers
iPCR	input Plug Control Register
IRM	Isochronous Resource Manager
LAN	Local Area Network
LSB	Least Significant Bit
MAC	Medium Access Control
MSB	Most Significant Bit
oPCR	output Plug Control Register
PCR	Plug Control Register
PDU	Protocol Data Unit
RLC	Radio Link Control
SAP	Service Access Point
SDU	Service Data Unit
Self_ID	Self IDentify
SSCS	Service Specific Convergence Sublayer
WCM	Wireless Cycle Master
WCS	Wireless Cycle Slave
WT	Wireless Terminal

4 Overview

4.1 The 1394 SSCS

The IEEE 1394 Service Specific Convergence Sublayer (SSCS) is a part of the Packet based Convergence Layer and it resides on top of the Common Part and the DLC, as shown in Figure 1. The support of the DLC extensions for the home environment, as described in [4], is mandated by the 1394 SSCS.

The aim of this sublayer is to mimic the functions provided by IEEE 1394 [6] link layer. Bridge functions are out of the scope of this document. They are defined in the HIPERLAN/2 1394 bridge layer (see Bibliography). The HIPERLAN/2 protocol stack defined in the present document replaces the wired 1394 physical and link layer. The services and primitives provided by the 1394 SSCS are a subset of those provided by the wired 1394 physical and link stacks. Basic wired 1394 services and primitives are provided by the 1394 SSCS. Some of the wired 1394 services and primitives are related to the cable environment and the corresponding physical layer. They are not relevant for the HIPERLAN/2 environment, and thus not provided by the 1394 SSCS.

The 1394 SSCS allows the transportation of wired-1394 asynchronous and isochronous packets over HIPERLAN/2. It also provides 1394 clock propagation over HIPERLAN/2.

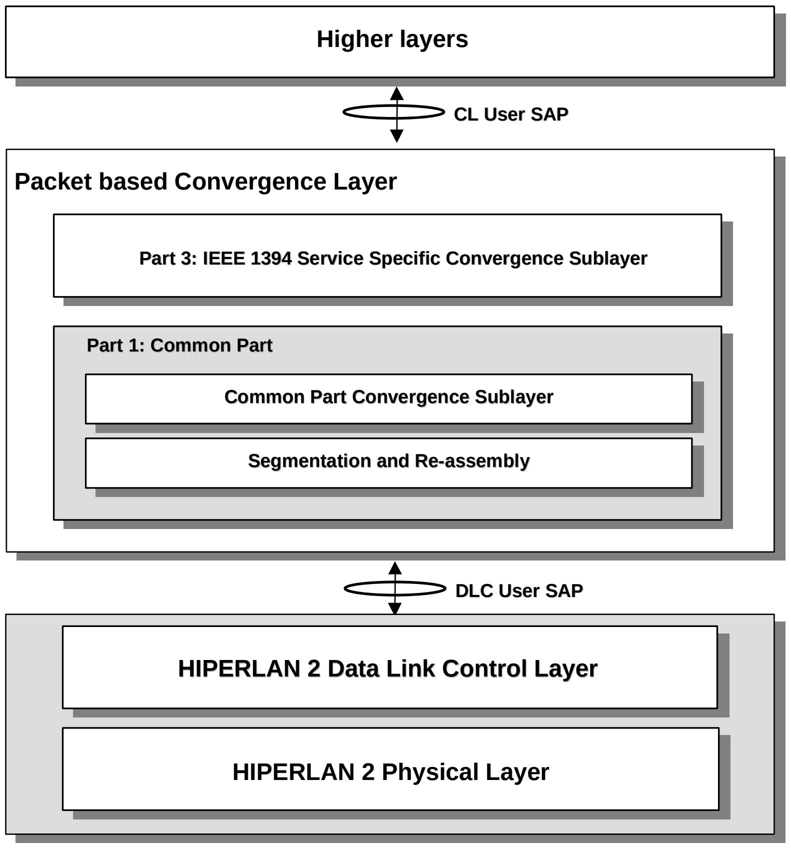
Higher layers
CL User SAP
Packet based Convergence Layer
Part 3: IEEE 1394 Service Specific Convergence Sublayer
Part 1: Common Part
Common Part Convergence Sublayer
Segmentation and Re-assembly
DLC User SAP
HIPERLAN 2 Data Link Control Layer
HIPERLAN 2 Physical Layer

Figure 1: IEEE 1394 [6] packet based convergence layer

The 1394 SSCS consists of a user plane and a control plane.

The user plane, described in clause 5, provides services to the higher layer via the CL-Service Access Point (SAP) and uses the services of the Common Part of the Packet based Convergence Layer [1].

The control plane, described in clause 6, interacts with the DLC control plane, i.e. the RLC sublayer, via the DLC Control SAP. It also uses the services of the Common Part of the Packet based Convergence Layer [1]. The DLC Control SAP is defined in [2] and [4].

The 1394 SSCS may be present in two types of devices:

- Wireless 1394 bridge devices: these devices contain bridge functions to allow the connection of a HIPERLAN/2 network to a wired-1394 bus or to another wireless network. A wireless bridge device has at least one of its portals that belong to the HIPERLAN/2 network. The wireless portal shall implement both the 1394 SSCS and the HIPERLAN/2 1394 bridge layer.
- Wireless 1394 devices: these devices implement only the 1394 SSCS, and thus they do not provide any bridge functions (see in Figure 2).

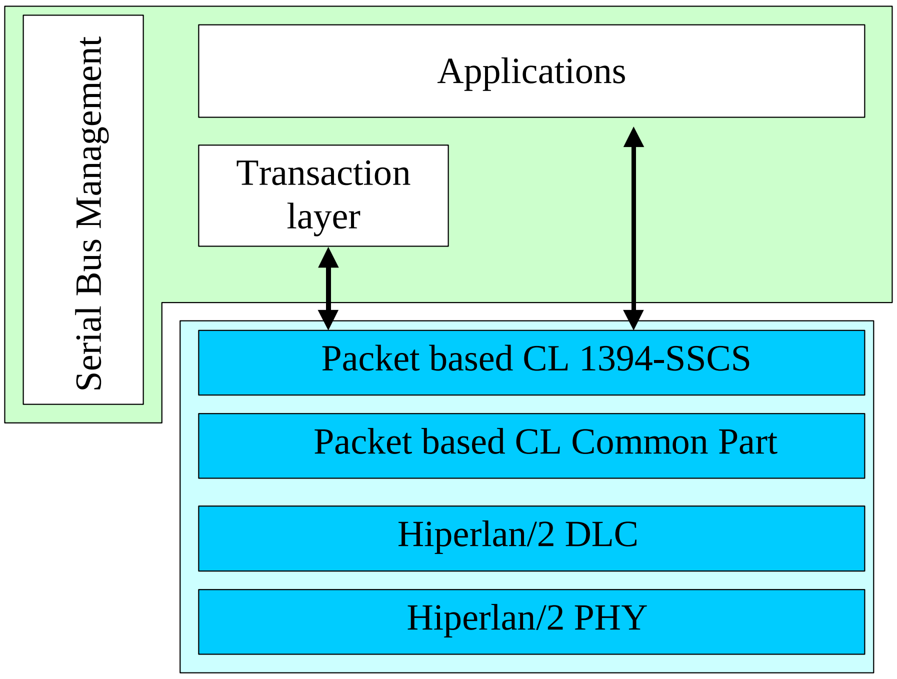

Figure 2: Protocol stack of a non-bridge wireless 1394 device

4.2 HIPERLAN/2 Bus

The net topology when applied to HIPERLAN/2 is shown in Figure 3. The HIPERLAN/2 network appears as a wired-1394 bus for all the attached nodes. This bus is called a HIPERLAN/2 Bus, which is often abbreviated as HL2 Bus. Regarding the HL2 Bus, all wireless nodes are peers (there is no particular base station). Regarding the HIPERLAN/2 network there is however one device in the cell which acts as the CC.

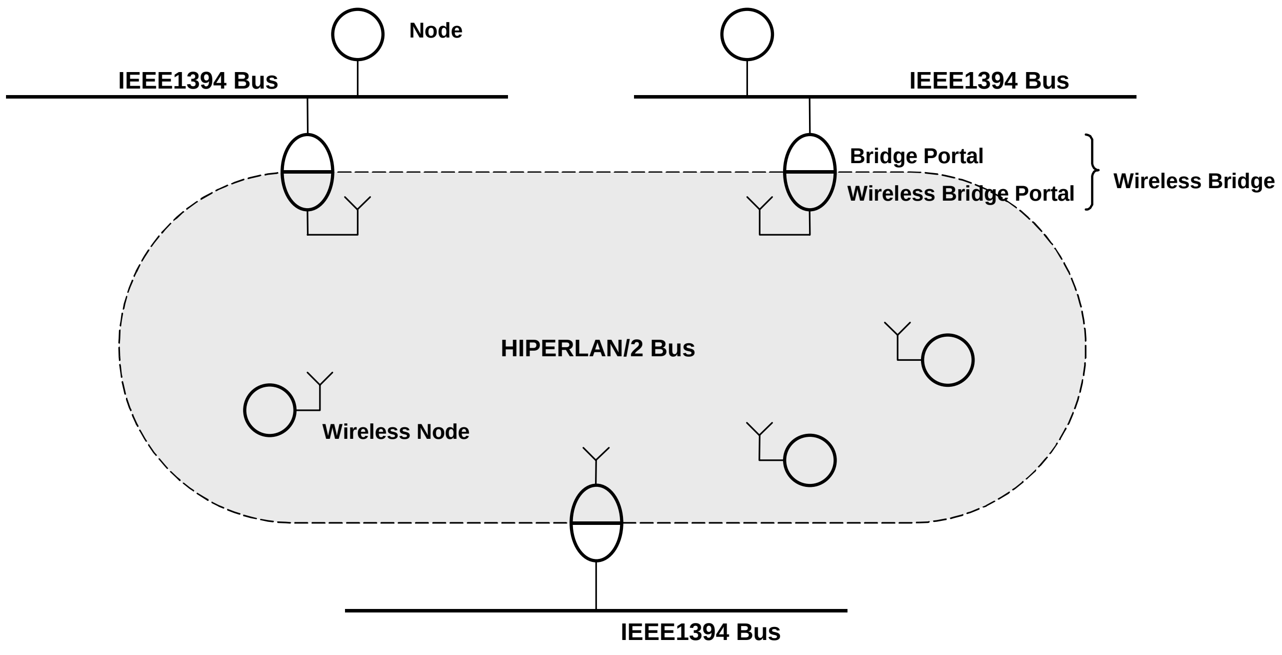

Figure 3: HL2 Bus connecting bridge and non-bridge 1394 devices

4.3 Numbering and coding conventions

Information elements and PDU are transferred between the 1394 SSCS and the underlying protocol layers in units of octets, in ascending numerical octet order (i.e. octet 1, 2, …, n-1, n), see Figure 4.

	MSB			Bits				LSB
	8	**7**	**6**	**5**	**4**	**3**	**2**	**1**
Octet 1								
Octet 2								
…								
Octet N-1								
Octet N								

Figure 4: Format convention

When a bit is contained within a single octet, the highest bit number (i.e. labelled 8) represents the high order or most significant bit (MSB).

In a multi-octet field, the order (i.e. the significance) of bit values within each octet decreases as the octet number increases. For example, a 16-bit field is coded in Figure 5.

	MSB			Bits				LSB
	8	**7**	**6**	**5**	**4**	**3**	**2**	**1**
Octet 1	2^{15} (MSB)	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}
Octet 2	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0} (LSB)

Figure 5: Corresponding coding in an IE or PDU

5 User plane services

5.1 General

The user plane procedures of the 1394 SSCS provide the capabilities to transfer 1394 SSCS SDUs between two wireless 1394 devices over the HIPERLAN/2 network.

The behaviour of the user plane procedures for CC and WT is identical.

User plane services cover:

a) Transport of 1394 SSCS SDU for both asynchronous and isochronous data;

b) Clock synchronization service: this function provides a mechanism that allows any isochronous capable node to have its own cycle clock synchronized to the cycle clock of a single node.

5.2 Primitives (informative)

The higher layers (node or bridge applications) access to the 1394 SSCS is defined by the service primitives defined in this subclause. Normative definitions are therefore contained in [7] and HL2 1394 bridge layer (see Bibliography), for node and bridge applications respectively.

NOTE: The primitives are defined only for the purpose of describing layer-to-layer and sublayer-to-sublayer interactions. These primitives are defined as an abstract list of parameters, and their concrete realization may vary between implementations. No formal testing of primitives is intended. The following primitive definitions have no normative significance.

5.2.1 Primitive types

Interface between layers

Four primitive types may be used between different layers:

- request, for a higher layer to request service from a lower layer;
- confirm, for the layer providing the service to confirm the activity has been completed;
- indication, for a layer providing service to notify the next higher layer of any specific service related activity;
- response, for a layer to acknowledge receipt of an indication primitive from the next lower layer.

Interface between sublayers

Two different primitives may be used between different sublayers. To indicate the absence of a Service Access Point these primitives differ to the primitives between different layers:

- invoke, for a higher layer to request service from a lower layer;
- signal, for a layer providing service to notify the next higher layer of any specific service related activity.

The defined types for each category of primitive are shown as a list in curly brackets.

EXAMPLE: CL_UNITDATA {request, indication}

In this example, the defined types are request and indication but not confirm or response.

5.2.2 Parameter definitions

Message unit: each piece of higher layer information that is included in the primitive is called a message unit. A series of one or more message units may be associated with each primitive where each separate unit is related to one information element in the corresponding DLC layer message. The list of message units is derived from the message definitions by reference to the information elements that may contain information from or (to) the CL.

5.2.3 Interface to the upper layer

5.2.3.1 Primitives related to isochronous traffic

These primitives are used either by a HL2 1394 bridge layer entity (see Bibliography) or by a streaming application.

5.2.3.1.1 CL_ISOCH_STREAM {request, indication}

This primitive is used to transmit isochronous packets on the bus, using parameters specified in Table 1. This primitive is not confirmed.

Table 1: CL_ISOCH_STREAM parameters

parameter	request	confirm	indication	response
Message units (possible elements)				
time_stamp	always	—	always	—
isoch_header	always	—	always	—
interface_data (SSCS SDU)	always	—	always	—

The ***time_stamp*** parameter specifies a 32 bits time stamp that shall be inserted in the 1394 SSCS PDU.

The ***isoch_header*** parameter includes *tag*, *channel*, *tcode*, and *sy* field, as defined by wired 1394.

The ***interface_data*** parameter consists of the *data* and *pad* bytes, as well as *data_CRC* fields, as defined by wired 1394.

5.2.3.1.2 CL_CYCLE_SYNCH {indication}

The 1394 SSCS entity uses this primitive to indicate to the upper layer that an isochronous period has passed (i.e. that CYCLE_TIME.*offset_count* has overflowed), using parameters specified in Table 2. This primitive is not confirmed.

Table 2: CL_CYCLE_SYNCH parameters

parameter	request	confirm	indication	response
Message units (possible elements)				
current_seconds_count	—	—	always	—
current_cycle_count	—	—	always	—

The ***current_seconds_count*** parameter contains the current value of the BUS_TIME register.

The ***current_cycle_count*** parameter contains the current CYCLE_TIME.*cycle_count* value.

5.2.3.2 Primitives related to asynchronous traffic

These primitives are used by a 1394 Transaction Layer.

5.2.3.2.1 CL_ASYNC_ACTION {request, indication, confirm, response}

This primitive is used to transmit one primary asynchronous packet on the HIPERLAN/2 bus, using parameters specified in Table 3.

Table 3: CL_ASYNC_ACTION parameters

parameter	request	confirm	indication	response
Message units (possible elements)				
time_of_life	always	—	always	—
response_flag	always	—	always	—
interface_data (SSCS SDU)	always	—	always	—
crc_result	—	—	always	always
reply	—	always	—	always

The ***time_of_life*** parameter is used to indicate how long this packet is allowed to live in the HL2 Bus. It may be negative, in that case an upper layer receiving it in an indication could take appropriate actions (such as discard the packet).

The ***response_flag*** parameter differentiates between request and response packets, for the purposes of determining which DUC shall be used.

The ***interface_data*** parameter consists of an asynchronous subaction (request or response) packet, as defined by wired 1394.

The ***reply*** code has one of the following values:

reply_accepted	the interface data was accepted, but has not necessarily been transferred over the HL2 Bus.
reply_busy	the interface data was rejected because buffer space was temporarily unavailable.
reply_missing	the interface data could not be delivered because the *destination_ID* is invalid.

The ***crc_result*** parameter indicates when the *data_CRC* was found to be invalid.

5.2.3.2.2 CL_ASYNC_STREAM {request, indication}

This primitive is used to transmit asynchronous stream packets on the bus, using parameters specified in Table 4. This primitive is not confirmed.

Table 4: CL_ASYNC_STREAM parameters

parameter	request	confirm	indication	response
Message units (possible elements)				
time_of_life	always	—	always	—
async_header	always	—	always	—
interface_data (SSCS SDU)	always	—	always	—
crc_result	—	—	always	—

The ***time_of_life*** is used to indicate how long this packet is allowed to live in the HL2 Bus. It may be negative, in that case an upper layer receiving it in an indication could take appropriate actions (such as discard the packet).

The ***async_header*** parameter consists of the asynchronous stream packet header as defined by wired 1394.

The ***interface_data*** parameter consists of the header, header_CRC, data, pad, and data_CRC fields, as defined by wired 1394.

The ***crc_result*** parameter indicates when the data_CRC was found to be invalid.

5.2.4 Interface to the common part

The primitive CPCS_UNITDATA {invoke, signal}, defined in [1], is used by the user plane procedures of the 1394 SSCS.

5.3 Clock synchronization service

5.3.1 General

Clock synchronization service aims to provide the means to synchronize clock registers of wireless nodes to their corresponding registers at the wireless cycle master (WCM) node. Clock registers are the following:

- BUS_TIME,
- CYCLE_TIME,
- LOCAL_SECONDS,
- LOCAL_CYCLES.

BUS_TIME and CYCLE_TIME registers are related to isochronous traffic, and shall be implemented by all isochronous capable wireless nodes. These registers are synchronized to the BUS_TIME and CYCLE_TIME registers of the WCM.

LOCAL_SECONDS and LOCAL_CYCLES maintain a local time within the HL2 Bus and shall be implemented by all wireless nodes. These registers are synchronized to the LOCAL_SECONDS and LOCAL_CYCLES registers of the WCM. The characteristic of these two registers is that they have to ensure that the time passes continuously. LOCAL_SECONDS and LOCAL_CYCLES registers of the WCM shall not be modified by the network cycle master in order to prevent any time discontinuities (see also in 6.5.2.2). LOCAL_SECONDS and LOCAL_CYCLES registers have only local significance. Time maintained by these registers is used by wireless nodes to timestamp asynchronous transactions (see subclause 5.4).

Detailed description of all these registers is given in Annex A; clauses A.1.4, A.1.5, A.1.6 and A.1.7.

Synchronization principles for BUS_TIME, CYCLE_TIME, LOCAL_SECONDS and LOCAL_CYCLES with their corresponding registers at the WCM are similar. The following principle is mainly focused on the synchronization of the CYCLE_TIME register.

Synchronization principle of the CYCLE_TIME register between the cycle master and a cycle slave is depicted in Figure 6. Referring to this illustration, when a frame start is observed by the cycle master the CYCLE_TIME register is latched in the *a* register, and when the frame start is observed by a cycle slave the CYCLE_TIME register is latched in the *b* register. The contents of the *a* register is transmitted to the cycle slave *a*' register. The time difference between *b* and *a*' is used to adjust the CYCLE_TIME register of the cycle slave.

NOTE: The offset of the clock PDU in the frame is not constant; it may change as depicted in Figure 6.

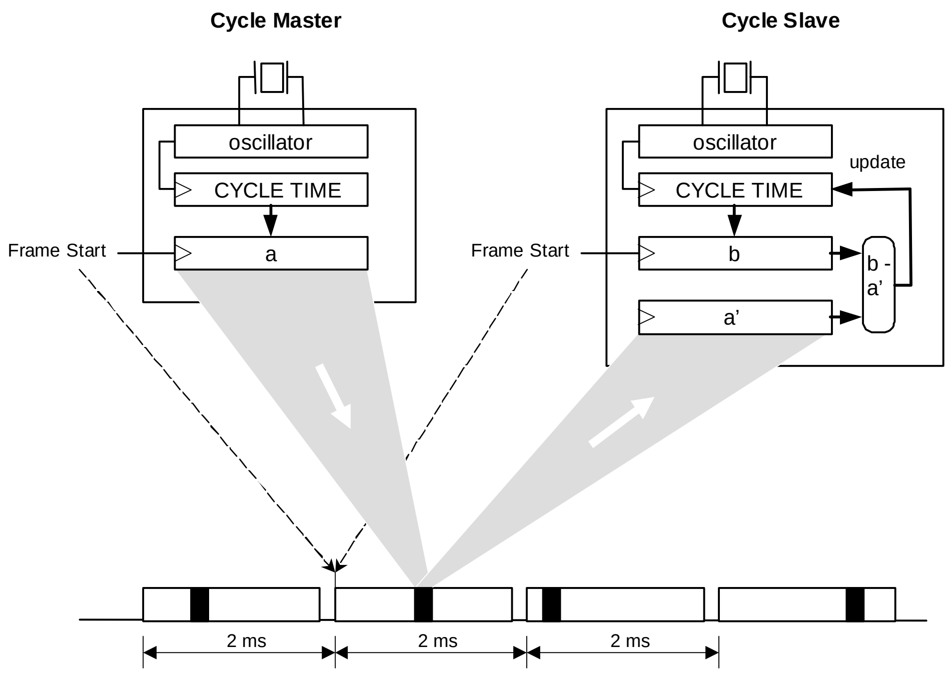

Figure 6: Synchronization principle of the CYCLE_TIME register

5.3.2 SSCS PDU format

The WCM shall broadcast clock information into clock PDU, which contains synchronized and local time values, as illustrated in Figure 7.

	8	7	6	5	4	3	2	1
Octet 1	isoch_seconds							
Octet 2								
Octet 3								
Octet 4								
Octet 5	isoch_cycles							
Octet 6						isoch_offset		
Octet 7								
Octet 8		reserved			frame_counter			
Octet 9	local_seconds							
Octet 10								
Octet 11								
Octet 12								
Octet 13	local_cycles							
Octet 14						local_offset		
Octet 15								
Octet 16		reserved						

Figure 7: Clock SSCS PDU format

Clock PDU fields are defined as follows:

- The 32-bit ***isoch_seconds*** field shall contain the value of the BUS_TIME register at the time when the DLC_MAC_FRAME_START primitive was received by the 1394 SSCS entity of the WCM.
- The 13-bit ***isoch_cycle*** field shall contain the value of CYCLE_TIME. *cycle_count* at the time when the DLC_MAC_FRAME_START primitive was received by the 1394 SSCS entity of the WCM.
- The 12-bit ***isoch_offset*** field shall contain the value of the CYCLE_TIME. *cycle_offset* at the time when the DLC_MAC_FRAME_START primitive was received by the 1394 SSCS entity of the WCM.
- The 32-bit ***local_seconds,*** 13-bit ***local_cycles*** and 12-bit ***local_offset*** fields are copies of like named fields from the LOCAL_SECONDS and LOCAL_CYCLES registers at the time when the DLC_MAC_FRAME_START primitive was received by the 1394 SSCS entity of the WCM.

- The 4-bit ***frame_ counter*** identifies the number of the MAC frame which preamble was taken as a reference point for the initialization of the all other fields of the SSCS PDU. DLC_MAC_FRAME_NUMBER primitive furnishes the MAC frame number.

5.3.3 Procedures

5.3.3.1 Procedure in the wireless cycle master (WCM)

Each time the MAC frame start preamble is detected by means of the DLC_MAC_FRAME_START indication primitive, the WCM shall take a snap-shot of its BUS_TIME, CYCLE_TIME, LOCAL_SECONDS, LOCAL_CYCLES registers, and shall construct a SSCS PDU as follows:

- *isoch_seconds* contains the BUS_TIME.*seconds* value,
- *isoch_cycles* contains the CYCLE_TIME.*cycle_count* part,
- *isoch_offset* contains the CYCLE_TIME.*cycle_offset* part,
- *local_seconds* contains the LOCAL_SECONDS.seconds register value,
- *local_cycles* contains the LOCAL_CYCLES.*cycle_count* part,
- *frame_counter* contains the number of the frame which preamble was taken as reference point to all wireless nodes.

The WCM sends this SSCS PDU using the CPCS_UNITDATA invoke primitive to the CPCS entity.

5.3.3.2 Procedure in a wireless cycle slave (WCS)

Each time the MAC frame start preamble is detected by means of the DLC_MAC_FRAME_START indication primitive, the WCS shall take a snap-shot of its BUS_TIME (if any), CYCLE_TIME (if any), LOCAL_SECONDS, LOCAL_CYCLES registers, and shall store the result in a temporary register along with the *frame_counter* of the MAC_FRAME_START primitive.

When the WCS receives a CPCS_UNITDATA signal primitive for the clock DLC Connection, it shall check whether the received SSCS PDU corresponds to a measured snap-shot (by checking the *frame_counter field*). If the frame_counter matches, the WCS is able to update its clock registers. The way registers are updated is implementation dependent.

In the case the 1394 SSCS entity receives SSCS PDUs from the forwarding channel (see in 6.5), then snap-shots for the two last frames shall be stored in a temporary register.

5.4 Asynchronous transaction data transport service

5.4.1 General

The purpose of the asynchronous transaction service is to reliably communicate asynchronous transaction components, i.e. request or response subaction packets, between two 1394 SSCS peer entities. A busy-acknowledge based flow control limits the flow rate of the higher level transmitter to that which can be supported by the lower layers.

The upper layer is responsible for creation and checking of 1394 specified packets, including their *header_CRC* and *data_CRC* values. Higher level CRC coverage is a necessity because the undetected error rate of the lower level transport (approximately 10^{-11}) is less than that expected by 1394.

At the higher level, losses of asynchronous packets are detected through timeouts. In support of such timeout protocols, higher level transmitters send each packet to the 1394 SSCS entity with a time_of_life parameter, thereby limiting its effective lifetime within the HL2 Bus. At the higher level, receivers are required to check the time_of_life parameter, discarding these packets if their allocated time-to-live has expired.

When passing through the 1394 SSCS entity, the packet's time_of_life parameter shall be converted to a time_of_death label, by adding the current time to the time_of_life parameter. The time_of_death label uses the clock synchronization service (at least only the seconds and cycle counts of it). When passed to the higher level, the packet's time_of_death parameter shall be returned to a time_of_life parameter, thereby insulating the higher level protocols from the lowest level timing parameters.

5.4.2 Asynchronous SSCS PDU coding

The 1394 SSCS asynchronous subaction PDU consists of a format, a time-stamp and a wired-1394 asynchronous packet, as illustrated in Figure 8.

	8	7	6	5	4	3	2	1
Octet 1	format							
Octet 2	seconds							
Octet 3								
Octet 34	cycles							
Octet 45						reserved	type	
Octet 56	destination_ID							
Octet 67								
Octet 78	tlabel						rt	
Octet 89	tcode				pri			
...	header quadlets							
Octet 4n+1	header_CRC							
Octet 4n+2								
Octet 4n+3								
Octet 4n+4								
	(data quadlets)							
Octet m+1	(data_CRC)							
Octet m+2								
Octet m+3								
Octet m+4								

Figure 8: Asynchronous subaction SSCS PDU format

Each PDU starts with a ***format*** field that describes which encapsulation scheme is applied for this asynchronous subaction packet PDU. ***Format*** values are described in Table 5.

Table 5: *format* values

Value	Name	Description
0	iso_stream_bus_packet	Encapsulation of wired-1394 isochronous stream packets
1	async_stream_bus_packet	Encapsulation of wired-1394 asynchronous stream packets
2	async_subaction	Encapsulation of asynchronous subaction packets
3-254	-	Reserved

The 1394 SSCS PDU also contains a time_of_death label. This time_of_death label has 16-bit ***seconds*** and 13-bit ***cycles*** field components, representing time in seconds and multiples of 125 µs units respectively.

The 2-bit ***type*** field specifies the packet type, as specified in Table 6.

Table 6: Asynchronous subaction *type* values

Value	Name	Description
0	ASYNC_REQUEST	Asynchronous request and response packet with time_of_death header
1	—	Reserved
2	CLOSE_REQUEST	Clean-closing request (header quadlet only), see 6.8.2
3	CLOSE_RESPONSE	Clean closing response (header quadlet only), see 6.8.2

The remainder of the packet consists of a subaction as defined by wired 1394. The ***tlabel***, ***rt***, ***pri*** fields, the remaining header quadlets, and the data quadlets are processed as user data and (except for their inclusion in the CRC calculations) are ignored by the 1394 SSCS entity. Other fields are described in the remainder of this subclause and can effect the 1394 SSCS entity processing.

The ***destination_ID*** that consists of a 10-bit *bus_ID* and 6-bit *physical_ID* components specifies which node is intended to receive the packet.

The ***header_CRC*** value is located at a *tcode*-dependent offset (as defined in [6]) and covers the fourth through preceding bytes. The initial four bytes are not covered by the *header_CRC* value.

Some packets contain data after the header_CRC value. For such packets, the ***data_CRC*** value covers these data bytes and the zero-valued pad bytes defined in [6].

5.4.3 Asynchronous packets management

To transfer asynchronous transaction between two nodes A and B, a single duplex DUC shall be opened, as defined in 6.8. One DUC is used to send requests from A to B and responses from B to A, which is sufficient if A is the requester and B is the responder. A different DUC (if necessary) would be used to send requests from B to A and responses from A to B. This prevents deadlocks that may appear if requests and responses are using a single DLC connection.

Thus, the opening of a DUC necessarily involves the passing of an information element, that specifies whether the opening node is the requester or responder. This is further described in 6.8.1.

When a 1394 SSCS receives a CL_CONTROL.*asynchronous_path_reset*, all open DUCs shall be closed as described in 6.8.2.

5.4.4 Procedures

5.4.4.1 Procedure in the sender

This service is started on the upper layer request, when using the CL_ASYNC_ACTION request primitive. The service involves a sequence of steps, listed below.

1) If the *destination_ID.busID* and request/response property (see 5.4.3) are associated with an open DUC, then skip to step 5.

2) The *destination_ID* field is processed to generate physicalAddr, the HL2-bus destination's physical_ID address:

   a) If *destination_ID.bus_ID* equals NODE_IDS.*bus_ID* (NODE_IDS is a CSR described in Annex A), then physicalAddr equals *destination_ID.physical_ID*.

   b) Otherwise, the *destination_ID.bus_ID* value is used as a parameter of a HARP operation (see 6.7) that returns a physicalAddr address. If the HARP operation fails to return a valid physicalAddr address, then a CL_ASYNC_ACTION confirm, with a *reply_missing* code, is returned.

3) The physicalAddr value is processed to retrieve the mac_ID address of the destination node by checking self-ID information (see 5.4.3 and 6.4). If no mac_ID address is found, then a CL_ASYNC_ACTION confirm, with a *reply_missing* code, is returned.

4) If no currently open DUC supports communications with the mac_ID address, then a DUC is opened for this purpose as described in 6.8. For efficiency, the *destination_ID.bus_ID* value may remain associated with the open DUC until the DUC is closed.

5) If the *time_of_life* value is negative, the packet shall be discarded. Otherwise, the *time_of_life* is added to the time values contained in LOCAL_SECONDS/LOCAL_CYCLES (these are CSRs described in Annex A) to generate a *time_of_death* value. The *time_of_death* value shall then be placed in the *seconds*/*cycles* fields of the packet.

6) If buffer space is not available in the lower layer, then a CL_ASYNC_ACTION confirm, with a *reply_busy* code, is returned. Otherwise, the 1394 SSCS PDU is posted to the CPCS in a CPCS_UNITDATA invoke primitive and a CL_ASYNC_ACTION confirm, with a *reply_accepted* code is returned.

5.4.4.2 Procedure in receiver

The *header_CRC* is checked. If the header is found to be corrupted, the packet is discarded.

The time values contained in LOCAL_SECONDS/LOCAL_CYCLES (these are CSRs described in Annex A) are subtracted from the *time_of_death* value to generate the time_of_life value that is included in the CL_ASYNC_ACTION indication primitive.

The *data_CRC* is checked and the result of this check is included in the CL_ASYNC_ACTION indication primitive.

NOTE: For efficiency, a *source_ID.bus_ID* of a packet received from the requester (see 6.8.1) may be associated (as described in 5.4.4.1) with the DUC. This has the effect of eliminating the need to do an additional HARP when the expected response is returned.

5.5 Stream data transport service

5.5.1 Isochronous stream data transport service

5.5.1.1 General

The purpose of the isochronous stream data transport service is to reliably transmit isochronous streams between two 1394 SSCS peer entities.

The upper layer is responsible for the generation and the reception of SDUs, representing individual 125 µs isochronous packets. Each SDU is received from the upper layer with a time stamp. This time represents the time when the packet was generated. It is encoded in the PDU, as described below, and thus transmitted by the 1394 SSCS entity to its peer so that the SDU is delivered to the destination upper layer with the time stamp. The time stamp may then be used by the upper layer to provide a constant delay service.

Each SDU is protected by a 32 bits CRC. Corrupted data are discarded by the 1394 SSCS entity.

The sending 1394 SSCS entity is responsible for concatenating the product of multiple SDUs into a single larger 2ms PDU. The receiving 1394 SSCS entity is responsible for regenerating multiple SDUs, representing individual 125 µs isochronous packets from a 2 ms PDU.

5.5.1.2 Isochronous stream SSCS PDU coding

For efficiency, several SDUs are collected into a single PDU, as illustrated in Figure 9 and described in the remainder of this Part.

	8	7	6	5	4	3	2	1
Octet 1	format							
Octet 2	seconds							
Octet 3								
Octet 4	cycles							
Octet 5						Reserved		
—	tagged_SDU[1]							
—	tagged_SDU[2]							
—	....							
—	tagged_SDU[N]							

Figure 9: Isochronous SSCS PDU format

A SDU is defined here as a 1394 isochronous packet as defined by wired 1394. The associated 1394 isochronous packet header is not part of the SDU, but is one of the parameters of the associated primitive (as described in 5.2.3.1.1).

Each PDU starts with a ***format*** field that describes which encapsulation scheme is applied for this stream PDU. ***Format*** values are described in Table 5.

A quadlet time-stamp label indicates then the isochronous cycle when the first SDU within the PDU was created. The time stamp label has 16-bit ***seconds*** and 13-bit ***cycles*** field components, representing BUS_TIME/CYCLE_TIME specified time in seconds and cycles (multiples of 125 μs units) respectively.

The remainder of the packet consists of the concatenation of tagged SDUs. For each SDU, the tag consists of its corresponding wired-1394 header, except that the channel number is replaced by 6-bits of the primitive supplied time-stamp value.

	8	7	6	5	4	3	2	1	
Octet 1	data_length								Tag
Octet 2									
Octet 3	tag		cycle_low						
Octet 4	tcode				sy				
...	data_quadlets								SDU
Octet 4n+1	data_CRC								
Octet 4n+2									
Octet 4n+3									
Octet 4n+4									

Figure 10: Isochronous tagged SDU format

data_length field represents the length of the following tagged SDU. It contains the length of the data field of the 1394 packet (without comprising the 1394 packet padding bytes). The *data_length* field is the same as the *data_length* field of the corresponding wired-1394 packet header (retrieved from or sent in the primitive).

tag field contains the *tag* field value of the corresponding wired-1394 packet header (retrieved from or sent in the primitive).

cycle_low field contains the 6 least significant bits of the *cycle_count* field from the time stamp that came to the 1394 SSCS entity along with the SDU.

tcode field contains the *tcode* field value of the corresponding wired-1394 packet header (retrieved from or sent in the primitive).

sy field contains the *sy* field value of the corresponding wired-1394 packet header (retrieved from or sent in the primitive).

data_quadlets represent data bytes as they were received in the SDU. They may contain padding bytes (the total number of data_quadlet bytes represents an integer number of quadlets).

data_CRC is the data CRC as defined by wired 1394.

NOTE: data_length field does not necessarily give the SDU size. Some padding bytes may be necessary to add to the data_length in order to get the total SDU size.

5.5.1.3 Procedure

5.5.1.3.1 Procedure in the sender

Upon the reception of a CL_ISOCH_STREAM request primitive, the 1394 SSCS entity shall check whether a multicast DUC is open for this SDU: the 1394 SSCS entity shall extract the 1394 *channel* field from the isoch_header primitive parameter. It shall then check whether there is an active DLC User Connection for a multicast channel corresponding to this 1394 channel (see 6.9). If there is no such open DUC, then the SDU shall be discarded. Otherwise, the 1394 SSCS entity collects several SDUs, received in several CL_ISOCH_STREAM request primitives into a single PDU.

The 1394 SSCS entity shall encode the *seconds* and *cycles* fields of the PDU with the time_stamp value of the first SDU (that was received in the CL_ISOCH_STREAM request primitive).

For each SDU, all the 1394 header fields, except the *channel* and the *header CRC* (that were received in the CL_ISOCH_STREAM request primitive) are copied into the corresponding fields tagging the SDU (*tag*, *tcode*, *sy*).

For each SDU, the *cycle_low* field is encoded with the 6 less significant bits of the time_stamp.*cycle_offset* primitive parameter.

data_CRC field is also encoded with the data_CRC from the CL_ISOCH_STREAM request primitive. This corresponds to a wired-1394 data CRC.

The 1394 SSCS entity shall build and post a PDU to the CPCS every HL2 frame. A DLC primitive (DLC_MAC_FRAME_START), described in [4] is available to determine when a HL2 frame starts.

Each PDU nominally contains 16 SDUs, although the number of SDUs can be less than 16 (if an isochronous packet is not sent in every cycle), 15 (if an isochronous packet is sent in every cycle, but the isochronous cycle duration is larger than 1/16 of the HL2 frame duration), or 17 (if an isochronous packet is sent in every cycle, but the isochronous cycle duration is shorter than 1/16 of the HL2 frame duration).

SDUs shall not be fragmented among different PDUs.

If the upper layer does not generate CL_ISOCH_STREAM request primitives within some 2 ms periods, empty PDUs may also be generated (a PDU with no SDU). In this case the empty PDU shall contain at least an empty SDU (i.e. data_length field encoded with 0, no data_quadlet field and a data_CRC encoded with 0). An empty PDU shall have 0 in the *seconds* and *cycles* fields as well as in the *tag*, *tcode*, *sy* and *cycle_low* fields.

5.5.1.3.2 Procedure in the receiver

The 1394 SSCS entity shall regenerate individual SDUs. It shall parse the PDU, and the individual *data_length* fields to recover individual SDUs. Since every *data_quadlets* field is quadlet aligned (i.e. it corresponds to an integer number of quadlets), the *data_length* field does not necessarily point to the end of the *data_quadlets* field.

For each recovered SDU:

- The 1394 SSCS entity shall compute the *data_CRC* field, and discard corrupted SDUs.
- Based on the *cycle_low* field of every SDU tag, and on the *seconds* and *cycles* fields of the PDU, the 1394 SSCS entity shall generate a time_stamp parameter.
- Based on the SDU tag, the 1394 SSCS entity shall generate the isoch_header primitive parameter corresponding to the SDU: *tag*, *tcode* and *sy* are extracted from the received SDU tag and copied as they are in the corresponding isoch_header fields.
  The PDU was received on a multicast channel for which there is a correspondence to a 1394 channel, which is used to fill the *channel* field of the isoch_header primitive parameter.
- The 1394 SSCS entity shall send the SDU along with the time_stamp and isoch_header parameters to the upper layer in a CL_ISOCH_STREAM indication primitive.

The receiver of isochronous streams shall be prepared for these streams to experience a maximum transmission delay of 8 ms as described in C.4.2.

5.5.2 Asynchronous stream data transport service

5.5.2.1 General

The purpose of the asynchronous stream data transport service is to reliably transmit asynchronous streams between two 1394 SSCS peer entities.

The upper layer is responsible for the generation and the reception of SDUs, representing individual asynchronous stream packets. Each SDU is received from the upper layer with a time_of_life.

When passing through the 1394 SSCS entity, the packet's time_of_life parameter shall be converted to a time_of_death label, by adding the current time to the time_of_life parameter. The time_of_death label uses the clock synchronization service (at least only the seconds and cycle counts of it). When passed to the higher level, the packet's time_of_death parameter shall be returned to a time_of_life parameter, thereby insulating the higher level protocols from the lowest level timing parameters.

Each SDU is also protected by a 32 bits CRC. Corrupted data are detected by the 1394 SSCS entity, and are signalled to the upper layer as corrupted.

5.5.2.2 Asynchronous stream SSCS PDU coding

An SDU is defined here as a 1394 asynchronous stream packet as defined by wired 1394. The associated 1394 asynchronous stream packet header is not part of the SDU, but is one of the parameters of the associated primitive (as described in 5.2.3.2.2)

	8	7	6	5	4	3	2	1
Octet 1	format							
Octet 2	seconds							
Octet 3								
Octet 4	cycles							
Octet 5						reserved		
Octet 6	data_length							
Octet 7								
Octet 8	tag		reserved					
Octet 9	tcode				sy			
...	data_quadlets							
Octet 4n-3	data_CRC							
Octet 4n-2								
Octet 4n-1								
Octet 4n								

Figure 11: Asynchronous stream PDU format

Each PDU starts with a ***format*** field that describes which encapsulation scheme is applied for this stream PDU. ***Format*** values are described in Table 5.

A quadlet time_of_death label having a 16-bit ***seconds*** and 13-bit ***cycles*** field components represents time in seconds and multiples of 125 µs units respectively.

data_length field represents the length of the following SDU. It contains the length of the data field of the 1394 packet (without comprising the 1394 packet padding bytes). The *data_length* field is the same as the *data_length* field of the corresponding wired-1394 packet header (retrieved from or sent in the primitive).

tag field contains the *tag* field value of the corresponding wired-1394 packet header (retrieved from or sent in the primitive).

reserved field corresponds to the *channel* field of the wired-1394 packet header, and shall be filled with 0.

tcode field contains the *tcode* field value of the corresponding wired-1394 packet header (retrieved from or sent in the primitive).

sy field contains the *sy* field value of the corresponding wired-1394 packet header (retrieved from or sent in the primitive).

data_quadlets represent data bytes as they were received in the SDU. They may contain padding bytes (the total number of data_quadlet bytes represents an integer number of quadlets).

data_CRC is the data CRC as defined by wired 1394.

5.5.2.3 Procedure

5.5.2.3.1 Procedure in the sender

Upon the reception of a CL_ASYNC_STREAM request primitive, the 1394 SSCS entity shall check whether a DLC multicast group is existing for this SDU: the 1394 SSCS entity shall extract the 1394 *channel* field from the *async_header* primitive parameter. It shall then check whether there is a DLC multicast group corresponding to this 1394 channel (at least whether the 1394 SSCS entity has joined such a DLC multicast group as described in 6.10). If the 1394 SSCS entity did not join such a DLC multicast group, then the SDU shall be discarded. Otherwise, the 1394 SSCS entity shall build a PDU according to 5.5.2.2.

If the *time_of_life* value is negative, the packet shall be discarded. Otherwise, the *time_of_life* is added to the time values contained in LOCAL_SECONDS/LOCAL_CYCLES (these are CSRs described in Annex A) to generate a *time_of_death* value. The *time_of_death* value shall then be placed in the *seconds*/*cycles* fields of the packet.

All the 1394 header fields, except the *channel* and the *header CRC* (that were received in the CL_ISOCH_STREAM request primitive) are copied into the corresponding fields of the PDU (*data_length tag*, *tcode*, *sy*).

Reserved bits are set to 0.

If the *data_quadlets* field is present *data_CRC* field is also encoded with the data_CRC from the CL_ISOCH_STREAM request primitive. This corresponds to a wired-1394 data CRC. If the *data_quadlets* field is absent (as can occur, when a channel is active but no data is available to be transmitted), then the *data_CRC* value is not included on the wired-1394 packet and is therefore not included in the 1394 SSCS PDUs.

The 1394 SSCS entity shall build and post a PDU to the CPCS in a CPCS_UNITDATA invoke primitive.

5.5.2.3.2 Procedure in the receiver

Upon the reception of a CPCS_UNITDATA signal primitive from the CPCS:

- The 1394 SSCS entity shall compute the *data_CRC* field, if present. The result of this operation is included in the CL_ASYNC_STREAM indication primitive. Based on the *cycle_low* field of every SDU tag, and on the *seconds* and *cycles* fields of the PDU, the 1394 SSCS entity shall generate a time_stamp parameter.
- The 1394 SSCS entity shall generate the *async_header* primitive parameter. It shall extract the *tag*, *tcode* and *sy* PDU fields and copy as they are in the corresponding isoch_header fields.
  The PDU was received on a multicast channel for which there is a correspondence to a 1394 channel that is used to fill the *channel* field of the isoch_header primitive parameter.
- *time of life* primitive parameter is generated by subtracting the time values contained in LOCAL_SECONDS/LOCAL_CYCLES (these are CSRs described in Annex A) from the *time_of_death* value.
- The 1394 SSCS entity shall send the SDU along with the *time_of_life* and *isoch_header* parameters to the upper layer in a CL_ISOCH_STREAM indication primitive.

6 Control plane services

6.1 General

The control plane of the 1394 SSCS interacts with the DLC control plane (i.e. the RLC sublayer) via the DLC Control Service Access Point defined in [2]. It also uses the services of the Common Part of the Packet based Convergence Layer [1].

The following functions are performed by the control plane procedures of the 1394 SSCS:

a) Triggering of DLC connection setup. This includes:

- Clock information DLC Connections (done at association time, and when connectivity to the wireless cycle master changes);
- Isochronous and asynchronous streams. This is done on upper layer request using isochronous resource manager (IRM) services (as described in 6.8);
- Asynchronous transaction data. This is done when the address resolution mechanism requests it (as described in 6.5) .

b) Triggering of DLC multicast and broadcast join procedures. This includes:

- Clock information DLC Connections (done at association time, and when connectivity to the wireless cycle master changes);
- Isochronous and asynchronous streams. This is done on upper layer request using plug control register (PCR) services (as described in 6.8).

c) Triggering of a bus reset process. This includes:

- 1394 physical_ID allocation;

- Distribution of all the 1394 physical_IDs of the HL2 Bus.

The following DLC C-SAP primitives defined in [4] are used by the control plane procedures of the 1394 SSCS:

- DLC_SETUP {request, indication};
- DLC_CONNECT {request, confirm, indication, response};
- DLC_RELEASE { request, confirm, indication, response };
- DLC_MULTICAST_JOIN {request, confirm, indication, response};
- DLC_MULTICAST_LEAVE {request, indication};
- DLC_MULTICAST_CONNECT {request, confirm, indication, response};
- DLC_MULTICAST_RELEASE {request, confirm, indication, response};
- DLC_MULTICAST_MODIFY {request, confirm, indication, response};
- DLC_INFO_TRANSFER {request, confirm, indication, response};
- DLC_MAC_FRAME_START {indication, response};
- DLC_MAC_FRAME_NUMBER {indication, response};
- DLC_START_CL_HO {indication, response};
- DLC_START_CC {indication, response}.

The DLC_INFO_TRANSFER primitive corresponds to the RLC_INFO primitive in the RLC layer. The RLC_INFO function, which is optional in RLC layer (as described in [2]), shall be mandatory for a RLC that supports the 1394 SSCS services.

6.2 Primitives (informative)

These primitives are used by the upper layer in a wireless 1394 device.

6.2.1 CL_CONTROL {request, confirm}

The upper layer uses this service to request the convergence layer to perform specific actions. This service is confirmed.

Table 7: CL_CONTROL parameters

parameter	request	confirm	indication	response
Message units (possible elements)				
bus_reset	always	always	—	—
enable_cycle_master	always	always	—	—
asynchronous_path_reset	always	always	—	—
node_type	always	always	—	—

The ***bus_reset*** parameter is used to initiate a bus reset.

The ***enable_cycle_master*** parameter permits to enable / disable wireless cycle master as defined in [6].

The ***asynchronous_path_reset*** parameter is used to force a reset of the asynchronous data transport service without loss of data. This means that all the DUCs being used for the asynchronous data transport service shall be closed.

The ***node_type*** parameter indicates the type of the node, as defined in Table 8.

Table 8: node_type values

Value	Definition
0	Non-bridge device
1 - 3	Bridge, as defined by HL2 1394 bridge layer

This information is used in the self identify process as described in 6.4.

6.2.2 CL_SELF_ID {indication}

The bus reset service is further described in 6.4.

It reports to an application layer the self_ID information on the HL2 Bus (number of connected nodes, as well as their physical_ID).

Table 9: CL_SELF_ID parameters

parameter	request	confirm	indication	response
Message units (possible elements)				
current_physical_ID	—	—	always	—
node_number	—	—	always	—
toggle_bit	—	—	always	—
For (i=0;i<Node_number; i++) {				
physical_ID	—	—	always	—
node_type	—	—	always	—
IRM_flag	—	—	always	—
}				

The ***current_physical_ID*** parameter indicates the physical_ID of the current node on the HL2 Bus.

The ***node_number*** field indicates the number of nodes on the HL2 Bus.

The ***toggle_bit*** field indicates some physical_IDs are going to be reallocated (portals can react accordingly).

The ***physical_ID*** parameter indicates the physical_ID of one node on the HL2 Bus.

The ***node_type*** parameter indicates the type of the node, identified by physical_ID on the HL2 Bus. The coding of this parameter is as defined in Table 8.

The ***IRM_flag*** parameter indicates if the node identified by the physical_ID on the HL2 Bus is the isochronous resource manager.

6.3 Association - Initialization

The CL capabilities (at least the CL version as defined in 7.2) should be exchanged between WT and CC convergence layers during the HIPERLAN/2 association process. The WT shall complete the association by sending one DLC_INFO_TRANSFER request primitive containing the EUI_64 information element defined below to the CC, using acknowledged unicast connections.

Information element	Reference	Status	Total length (octets)	Description
EUI_64	7.4.5	mandatory	11	Contains the WT's EUI_64

The EUI_64 information element shall be transmitted in the <<CL-ATTRIBUTE>> field of the DLC_INFO_TRANSFER request/response primitive.

Once the association procedure (as defined in [2]) is complete), a 1394 SSCS entity that is a wireless cycle master shall setup the clock channels (as described in 6.5). A 1394 SSCS entity that is a wireless cycle slave shall join the multicast clock group as described in 6.5).

6.4 Bus reset service

6.4.1 General

A bus reset can be initiated by any WT in the HL2 Bus, including the CC. The HL2 Bus reset is always controlled by the CC. The CC shall initiate a bus reset when:

- a WT associates to the CC; i.e. a wireless node joins the HL2 Bus,
- a WT dissociates from the CC; i.e. a wireless node leaves the HL2 Bus,
- the CC receives a request from a WT to initiate a bus reset,
- the CC receives a request from the upper layer to initiate a bus reset.

6.4.2 Procedures

A HL2 Bus reset is not an atomic action and the CC may consume time and resources in order to reliably propagate bus reset information. The HL2 Bus reset should be as less disruptive as possible regarding ongoing connections.

For the CC 1394 SSCS entity a specific state (bus_reset_state) is defined. The CC 1394 SSCS entity enters the bus_reset_state as soon as it decides to initiate a bus reset. It leaves the bus_reset_state when the bus reset is completed (see in 6.4.2.3).

Two types of information element are used by the CC to perform the bus reset: BUS_SUSPEND and BUS_RESUME. A third information element, BUS_RESET, is used by WTs to request the CC to initiate a bus reset.

6.4.2.1 CC or WT initiated bus reset

A bus reset is CC initiated when an event (such as WT association or WT dissociation) appears in the lower layers. As soon as such an event occurs, the CC enters into the bus_reset_state.

A bus reset may also be WT initiated. In this case, the WT sends DLC_INFO_TRANSFER request primitive containing the BUS_RESET information element defined below.

Information element	Reference	Status	Total length (octets)	Description
BUS_RESET	7.4.5	mandatory	4	Initiates a bus reset

When the CC receives such an information element, it shall enter in bus_reset_state, and start a bus reset phase. It shall also acknowledge the WT by sending a DLC_INFO_TRANSFER response primitive containing the same BUS_RESET information element.

NOTE: When a bus reset is initiated by the upper layer of the device where the CC runs, it can still be considered as a WT initiated bus reset, where the BUS_RESET information element flows internally to the device.

6.4.2.2 Bus reset worst case duration

The bus reset worst case duration is the maximum time the CC is allowed to spend in bus_reset_state. It starts when the CC decides to enter this state, and stops when the BUS_RESUME information element has been received from the last WT. This ΔT time is fixed to 1 second.

6.4.2.3 Procedure in the CC side

When entering into the bus_reset_state, the CC shall start the ΔT timer to monitor the bus reset duration. The CC 1394 SSCS entity shall then first send a DLC_INFO_TRANSFER request primitive containing the BUS_SUSPEND information element defined below to all associated WTs, using acknowledged unicast connections.

Information element	Reference	Status	Total length (octets)	Description
BUS_SUSPEND	7.4.7	mandatory	2xN+2	Indicates the start of the bus reset phase

The BUS_SUSPEND information element shall be transmitted in the <<CL-ATTRIBUTE>> field of the DLC_INFO_TRANSFER request/response primitive. The BUS_SUSPEND information element contains the physical_id allocation corresponding to this new bus reset (see in 6.4.3).

The CC 1394 SSCS entity shall manage isochronous resources as described in 6.9.2.4.

When the CC 1394 SSCS entity has received the DLC_INFO_TRANSFER response primitive containing the BUS_SUSPEND information element from all associated WTs, it shall invoke the CL_SELF_ID indication primitive and send a DLC_INFO_TRANSFER request primitive containing the BUS_RESUME information element defined below to all associated WTs, using acknowledged unicast connections.

Information element	Reference	Status	Total length (octets)	Description
BUS_RESUME	7.4.8	mandatory	2xN+2	Indicates the end of the bus reset phase

The BUS_RESUME information element shall be transmitted in the <<CL-ATTRIBUTE>> field of the DLC_INFO_TRANSFER request/response primitive.

The bus reset is complete when the BUS_SUSPEND information element have been received from all associated WTs. Then the CC leaves the bus_reset_state.

As said in 6.4.2.2, the bus reset duration shall be less than the ΔT timer. If a bus reset cannot be completed within the ΔT timer, another bus reset shall be started.

The CC shall take appropriate actions (DFS, WT dissociation, …) so that it can succeed in one bus reset. The detail of these actions is out of the scope of the present document.

If the CC 1394 SSCS entity fails to send the DLC_INFO_TRANSFER request primitive containing the BUS_SUSPEND information element to all associated WTs, it shall retransmit this primitive to only those WTs that did not acknowledge it, provided that the ΔT timer did not expire.

If the CC 1394 SSCS entity fails to send the DLC_INFO_TRANSFER request primitive containing the BUS_RESUME information element to all associated WTs, it shall retransmit this primitive to only those WTs that did not acknowledge it, provided that the ΔT timer did not expire.

6.4.2.4 Procedure in the WT side

When a WT 1394 SSCS entity receives a DLC_INFO_TRANSFER request primitive containing the BUS_SUSPEND information element, it shall acknowledge it by sending a DLC_INFO_TRANSFER response primitive containing the same BUS_SUSPEND information element.

The behaviour of a WT 1394 SSCS entity after the reception of a BUS_SUSPEND information element shall be as follows:

- all established isochronous connections shall be maintained;
- for the asynchronous transaction data transport service:
    - the 1394 SSCS entity shall reject any CL_ASYNC_ACTION request from its upper layer (i.e. respond with an ack_busy to any transaction layer packet) – as a result the upper layer will not be able to initiate new resource reservation from this time,
    - the 1394 SSCS entity shall process packets coming from the lower layer (the CPCS). This includes internal CSR access as well as transaction packets to deliver to the upper layer. The 1394 SSCS entity shall process its PCRs, if any, (reset the PCRs, start a timer and wait for a 1394 controller to configure the plus again as described in 6.9.2.2 and 6.9.2.3).

When a WT 1394 SSCS entity receives a DLC_INFO_TRANSFER request primitive containing the BUS_RESUME information element, it shall acknowledge it by sending a DLC_INFO_TRANSFER response primitive containing the same BUS_RESUME information element.

The behaviour of a WT 1394 SSCS entity after the reception of a BUS_RESUME information element shall be as follows:

- the WT resumes normal processing of asynchronous transactions;
- the WT shall invoke the CL_SELF_ID indication primitive.

When the upper layer receives a CL_SELF_ID indication primitive, it shall reserve resources that were reserved before the bus reset, as defined in 6.9, Annex C and [5], Part 1.

NOTE: Since a device running a CC 1394 SSCS also contains a WT 1394 SSCS, WT 1394 SSCS behaviour when a DLC_INFO_TRANSFER request primitive containing the BUS_RESUME information element is received also applies for the device containing the CC.

If the WT 1394 SSCS receives more than one BUS_SUSPEND information elements before the reception of a BUS_RESUME information element, only the first one shall be considered (others shall be ignored). If the WT 1394 SSCS receives more than one BUS_RESUME information elements with no reception of a BUS_SUSPEND information element, only the first one shall be considered (others shall be ignored).

6.4.3 Assignment of physical_IDs

The CC shall assign a new physical_ID each time a WT associates. If the CC runs out new physical_IDs, the CC can recycle old physical_IDs, but shall inform about that by setting the *toggle* bit in the CL_SELF_ID primitive.

As physical_IDs assignment is centrally managed, the CC has to keep as much as possible stable physical_IDs through bus resets. The CC shall assign the same physical_IDs to still associated WTs. However if the bus reset is WT-initiated, the CC shall assign a new physical_ID to that WT which initiates the bus reset. If a WT leaves (disassociates) and joins (associates) again the HL2 Bus it shall be assigned a new physical_ID.

6.5 Clock information connection control

6.5.1 General

A 1394 channel is reserved and is dedicated to the multicast clock distribution. The 1394 clock channel is 00_{16}. Two multicast groups may correspond to this 1394 clock channel (a main and a forwarding group). The main group corresponds to clock information directly sent by the WCM. The forwarding group corresponds to clock information repeated by the 1394 SSCS entity of the CC. Selection between main and forwarding group is done by using the *relay* bit of the 1394 CHANNEL IE.

Every isochronous capable HL2 1394 device shall join the main multicast group of the clock channel just after the association phase. The CC side of 1394 SSCS entity allocates a multicast mac_ID for this main multicast group.

Cycle master function is enabled on a 1394 SSCS entity by invoking the CL_CONTROL request primitive on the corresponding entity with *enable_cycle_master* as primitive parameter.
A 1394 SSCS entity that contains the IRM function can also enable itself its cycle master function (see 6.5.2.1).

Once the cycle master function is enabled on a device, the 1394 SSCS entity shall setup a multicast DLC connection for the clock multicast mac_ID. According to the DLC multicast procedures (as defined in [4]), every 1394 SSCS entity belonging to the clock multicast group (i.e. all the wireless cycle slaves) will get the DLC multicast connection setup.

If one of the 1394 SSCS entity does not correctly receive the clock information (because no radio contact to the WCM), it shall ask to join the forwarding multicast group.

When at least one WT asks to join the forwarding multicast group, the corresponding multicast DLC connection shall be setup. The 1394 SSCS entity of the CC shall also repeat clock information as described in 5.3.

6.5.2 Wireless cycle master procedure

6.5.2.1 Wireless cycle master election

The wireless cycle master (WCM) is the cycle master on the HL2 Bus.

The 1394 SSCS entity that contains the IRM function (as described in 6.9) shall enable its cycle master function if no other WCM is operating on the HL2 Bus (no multicast DLC connection is enabled for the multicast group of the clock channel).

If the 1394 SSCS entity receives a CL_CONTROL request primitive with the enable_cycle_master action from its upper HL2 1394 bridge layer entity, it shall enable its cycle master function. An upper layer that is not a HL2 1394 bridge layer entity shall not generate such a request as described in Annex C.

If the 1394 SSCS entity receives a CL_CONTROL request primitive with the disable_cycle_master action from its upper HL2 1394 bridge layer entity, it shall disable its cycle master function. An upper layer that is not a HL2 1394 bridge layer entity shall not generate such a request as described in Annex C.

6.5.2.2 Wireless cycle master operation

As any isochronous capable device the 1394 SSCS entity shall join the main multicast group of the clock channel during the initialization phase. It shall send a DLC_MULTICAST_JOIN request primitive containing the CHANNEL information element defined below to the CC, using acknowledged unicast connections. (if the WCM and the CC functions are hosted by the same device, this message flow is internal).

Information element	Reference	Status	Total length (octets)	Description
CHANNEL	7.4.6	mandatory	3	Contains the channel number

The CHANNEL information element shall be transmitted in the <<CL-ATTRIBUTE>> field of the DLC_MULTICAST_JOIN request/response primitive.

The *relay* bit, which is defined in 6.10.2, is set to 0.

When the cycle master function is enabled on a device B, the cycle master function may already operate on another device A. In that case device B, the new WCM, shall first update its LOCAL_TIME register value based on observed values provided by device A, the old WCM, and continues as described below. The LOCAL_TIME register is defined in Annex A and in 5.3.

When the cycle master function is enabled, the 1394 SSCS entity shall setup the multicast DLC connection for the clock multicast mac_ID. It shall use the fixed slot allocation mode of the DLC (as described in [4]), with one LCH slot every frame. It shall use the non acknowledged transport mode of the DLC.
When the cycle master function is disabled, the 1394 SSCS entity shall release the multicast DLC connection for the clock multicast mac_ID.

When a WCM detects that another WCM is operating on the HL2 Bus, the 1394 SSCS entity that contains the IRM and is also WCM shall disable its cycle master function.

6.5.3 Wireless cycle slave procedure

The 1394 SSCS entity shall join the main multicast group of the 1394 clock channel during the initialization phase (i.e. just after the association phase). It shall send a DLC_MULTICAST_JOIN request primitive containing the CHANNEL information element defined below to the CC, (if the WCM and the CC functions are hosted by the same device, this message flow is internal).

Information element	Reference	Status	Total length (octets)	Description
CHANNEL	7.4.6	mandatory	3	Contains the channel number - The *relay* bit, which is defined in 6.10.2, is set to 0.

The CHANNEL information element shall be transmitted in the <<CL-ATTRIBUTE>> field of the DLC_MULTICAST_JOIN request/response primitive.

When the multicast DLC connection for the main multicast group of the clock channel is setup, the 1394 SSCS entity shall check that the clock information is correctly received. If the clock information is not well received (no or loose radio contact with the WCM), the 1394 SSCS entity shall join the forwarding multicast group of the clock channel. It shall send a DLC_MULTICAST_JOIN request primitive containing the CHANNEL information element defined below to the CC, (if the WCM and the CC functions are hosted by the same device, this message flow is internal).

Information element	Reference	Status	Total length (octets)	Description
CHANNEL	7.4.6	mandatory	3	Contains the channel number - The *relay* bit, which is defined in 6.9.3.2, is set to 1.

The CHANNEL information element shall be transmitted in the <<CL-ATTRIBUTE>> field of the DLC_MULTICAST_JOIN request/response primitive.

A 1394 SSCS entity shall check it correctly receives the clock information at the initialization, and also during the normal operation: the radio contact may change, and the 1394 SSCS entity may have to join the forwarding channel if it looses direct contact to the WCM.

In the same way, a device that joined the forwarding multicast group, shall periodically check whether it gets clock information from the main channel. If it does, it shall leave the forwarding multicast group by using a DLC_MULTICAST_LEAVE request primitive with the CHANNEL information element.

6.5.4 CC specific procedure

When a WT joins the forwarding multicast group of the clock channel, the 1394 SSCS entity of the CC shall set up a forwarding multicast DLC connection for this group, as defined in 6.9.3.2. Parameters for the forwarding DLC connection are the same as the main DLC connection (fixed slot allocation, one LCH slot every frame, non-acknowledged transport mode).

The 1394 SSCS entity of the CC shall release the forwarding multicast DLC connection when it detects that the last WT left the forwarding group.

6.6 CL responsibility handover

When the 1394 SSCS of the CC receives the DLC_START_CL_HO indication primitive, it means a CC handover is on going, and another CC capable device accepted to become the CC (called the new CC hereafter). CC responsibility handover at the lower layer is described in [4].

Then the old CC shall perform the CL responsibility handover as follow:

It shall perform a bus reset procedure to inform other devices about the ongoing CC handover.

It shall send a BUS_SUSPEND message to all the devices (WTs and new CC), as described in 6.4 with the information that the new CC is the IRM. The description of the IRM in the BUS_SUSPEND IE is depicted in 7.4.7.

Eventually it sends a BUS_RESUME message to all devices (WTs and CC), as described in 6.4.

The new CC may become the wireless cycle master as defined in 6.5.

Once the bus reset phase is finished, the 1394 SSCS of the old CC indicates the completion of the procedure to the DLC layer by a DLC_START_CL_HO response.

When the new CC receives a DLC_START_CC indication primitive, it shall respond with a DLC_START_CC response primitive. The DLC_START_CC primitive means that the CC handover is completed.

NOTE: There is no convergence layer information to be handed over during a CC responsibility handover. After the CC responsibility handover bus reset, isochronous resources will be reclaimed as described in 6.9. The *handle* field of the RECLAIM_THIS request contains the mac_ID to be associated with the 1394 channel. So that, after the resource reclaim period, the new IRM gets the (1394 channel – multicast mac_IDs) pairs that go on. If some resources are not reclaimed after the resource reclaim period, it has to be released by the new IRM. The exact interface between the IRM and the RLC to retrieve these un-reclaimed resources is out of the scope of the present document.

6.7 HL2 Address Resolution service (HARP)

6.7.1 General

This service is provided as a future-proof feature. This service is attached to the asynchronous transaction data transport from the user plane. If the bus_ID part of the *destination_ID* of the 1394 SSCS SDU is neither set to the HL2 Bus bus_ID, neither to $3F_{16}$ (local bus), this service is used in order to resolve the destination address.

When a 1394 SSCS entity receives SDU of which the destination bus_ID is unknown, it has to initiate the address resolution service to get the physical_ID of the destination wireless device. In order to guarantee the accuracy of the request, the 1394 SSCS entity uses an acknowledged protocol by polling every node of the HL2 Bus.

The HARP service uses the DLC_INFO_TRANSFER primitive of the DLC-C-SAP for direct link.

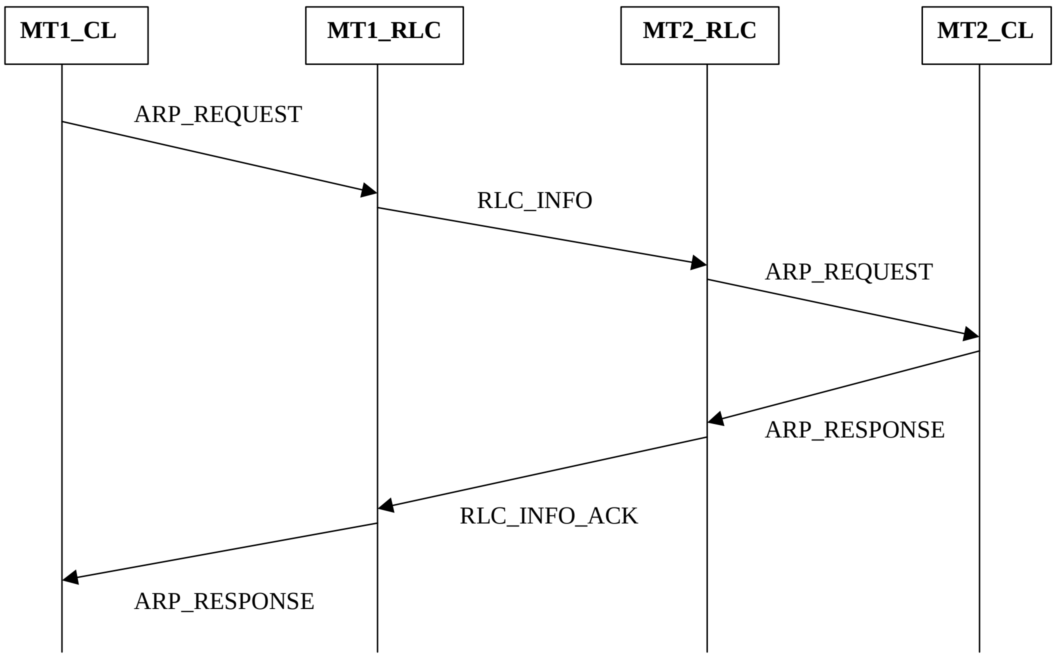

Figure 12: HARP request

The initiating 1394 SSCS entity sends a HARP_REQUEST to all other WTs, that answers back a HARP_RESPONSE indicating whether they are the destination 1394 SSCS entity or not.

6.7.2 Procedures

6.7.2.1 Procedure in the sender

When a 1394 SSCS entity wants to initiate a HARP, it shall send a DLC_INFO_TRANSFER request primitive containing the HARP_REQUEST information element below to all the wireless nodes, using acknowledged unicast connections.

Information element	Reference	Status	Total length (octets)	Description
HARP_REQUEST	7.4.3	mandatory	4	Contains the unknown bus_ID

The HARP_REQUEST information element shall be transmitted in the <<CL-ATTRIBUTE>> field of the DLC_INFO_TRANSFER request/response primitive.

When a WT gets an HARP_RESPONSE information element with acceptance of forwarding from another WT, it knows whether that WT is the destination wireless device or not.

HARP protocol shall stop at the first positive response got. If no positive response comes (invalid bus_ID), then the 1394 SSCS entity generates a response packet (wired-1394 response packet with resp_address_error response code (as defined in [6]) to the upper layer.

6.7.2.2 Procedure in the receiver

Upon receiving a DLC_INFO_TRANSFER indication the 1394 SSCS entity generates a DLC_INFO_TRANSFER response primitive containing the HARP_REQUEST information element. Among all the 1394 SSCS entities that have received an HARP_REQUEST, only the one that is in charge of forwarding the asynchronous packets to the target bus_ID shall set Forwarding bit indication to 1 in the HARP_RESPONSE. It shall also put its own physical_ID in the IE. The others shall set this bit to 0.

Information element	Reference	Status	Total length (octets)	Description
HARP_RESPONSE	7.4.4	Mandatory	5	Contains the unknown bus_ID and physical_ID

The HARP_RESPONSE information element shall be transmitted in the <<CL-ATTRIBUTE>> field of the DLC_INFO_TRANSFER response primitive.

6.8 Asynchronous transaction connection control service

6.8.1 Asynchronous transaction mapping to a DLC connection

6.8.1.1 General

Wired-1394 transactions are mapped on DLC connections as follow: one 1394 SSCS entity uses one DLC duplex connection to send transaction requests and receive transaction responses. If the 1394 SSCS entity is intended to receive transaction requests from the peer entity, another DLC duplex connection is needed.

Therefore, when setting up a DLC duplex connection, it is necessary to indicate whether the initiating side is a transaction requester or a responder.

6.8.1.2 Procedures

The 1394 SSCS initiating the connection shall send a DLC_SETUP request primitive containing the TRANSACTION_INDICATOR information element defined below to the other WT.

Information element	Reference	Status	Total length (octets)	Description
TRANSACTION_INDICATOR	7.4.10	mandatory	3	Indicates whether the initiating WT is the requester or the responder

The TRANSACTION_INDICATOR information element shall be transmitted in the <<CL-ATTRIBUTE>> field of the DLC_SETUP request/response primitive.

A 1394 SSCS receiving a DLC_SETUP indication primitive with the TRANSACTION_INDICATOR information element, and accepting the connection shall send a DLC_CONNECT request primitive with no specific information element. This TRANSACTION_INDICATOR IE indicates whether the WT that initiated the connection is a transaction requester or a responder and respectively.

6.8.2 Clean closing service

6.8.2.1 General

This service is used to clean close DLC user connections. A WT going to close a DLC user connection is thus able to purge DLC buffers before closing. This feature allows some devices with limited ARQ resources to cleanly close some DLC user connections and swap on others without loosing data. The way a device shares a limited amount of ARQ resources between a larger amount of DLC user connections is however out of the scope of the present document.

6.8.2.2 Clean closing SSCS PDU format

Clean close request and clean close response PDU are based on an asynchronous subaction SSCS PDU (see 5.4.2) with following characteristics:

	8	**7**	**6**	**5**	**4**	**3**	**2**	**1**
Octet 1	seconds							
Octet 2								
Octet 3	cycles							
Octet 4						reserved	type	
...	not used							

Figure 13: clean closing SSCS PDU format

Seconds and *cycles* fields shall be set to 0.

type field shall be set to either CLOSE_REQUEST or CLOSE_RESPONSE (see in Table 6).

Other asynchronous subaction SSCS PDU fields (*destination_ID, tlabel, rt, tcode, pri, header_quadlets, header_CRC, data_quadlets, data_CRC)* shall not be included in the remaining part of the PDU.

6.8.2.3 Procedures

6.8.2.3.1 Procedure at the initiating side

When one 1394 SSCS entity wants to close a DLC user connection, it shall send a clean closing request PDU, defined in 6.8.2.2, to the 1394 SSCS entity of the receiver and start a split timeout, as far a standard 1394 transaction.

It shall then stop sending user data to this DLC user connection.

It shall then wait for the reception of a clean closing response SSCS PDU. When either the clean closing response SSCS PDU comes back or the split timeout expires, then the 1394 SSCS entity sends a DLC_RELEASE request primitive.

6.8.2.3.2 Procedure at the other side

When a 1394 SSCS entity gets a clean closing request, it has to stop transmitting packets after sending the current one.

It shall then respond with the clean closing response PDU to the sender.

The DLC user connection will be released by the clean closing initiator. But as soon as the 1394 SSCS entity sent a clean closing response PDU, it shall consider the DUC is not open anymore.

6.9 Isochronous stream connection control service

6.9.1 General

The isochronous stream connection control service is used to allocate resources to the stream data transport service.

The management of the isochronous resources in a Serial Bus is covered by the isochronous resource manager (IRM) function described in wired 1394. It is located in the 1394 SSCS of the CC. The IRM provides channels and bandwidth reservation facilities, so that other device applications can make some resource reservation (lock request / lock response). The IRM does not provide any BANDWIDTH_AVAILABLE and CHANNEL_AVAILABLE registers. In fact due to wireless transmission constraints (link radio quality), there are different physical modes possible when a talker sends data to the listener. The selection of the physical mode depends from the radio link quality. Moreover from the selected physical mode depends also the duration of the allocated time slots in the MAC frame (this is further detailed in [3] and [4]. Therefore, the bandwidth to be reserved (in terms of time duration in the MAC frame) depends from the selected physical mode, and thus depends from who is the talker and who is the listener. Therefore a specific command based lock request lock response has been defined for the purpose of the IRM resources (see Annex A) in order to put the bandwidth, the talker, and at least one listener in a single command. The mechanism for isochronous data flow management uses the functions described in IEC 61883 [5],clause 7 of Part 1, with the exception of the oPCR definition, which is described in A.2.1 of the present document. It also uses the connection management procedures (CMP), as described in IEC 61883 [5],clause 8 of Part 1.

The 1394 controller reserves bandwidth and channel in the IRM and configures the iPCR and the oPCR. It can also modify or release the bandwidth in the IRM.

An application can connect a device on a IEEE 1394 [6] channel by writing a channel number into its plug control register (input plug for devices that receive data from the channel, output plug for devices that send data on the channel).

The oPCR register (or set of registers) shall be implemented in a 1394 SSCS of a device that can source isochronous streams. It is defined in A.2.1. This device is called a talker.

The iPCR register (or set of registers) shall be implemented in a 1394 SSCS of a device that can sink isochronous streams. It is defined in A.2.2. This device is called a listener.

The oPCR defined here is not totally the same as the one described in the IEC 61883 [5] standard. It is adapted to HL2.

Isochronous streams shall use fixed slot allocation, which is a mode of capacity allocation negotiated during DUC setup, as defined in [4].

Relaying of isochronous streams is out of the scope of the present document. Relaying of multicast data is not defined in [4]. In the case there is no radio contact between the talker and the listener, or in the case the radio contact gets lost between the talker and the listener, the listener will fail in receiving isochronous streams.

There are some features (link adaptation) in the lower layers (see [3] and [4]) in order to adapt the reception when the link radio quality is changed. Therefore, when the link radio quality decreases, another physical mode may be selected. If there is sufficient bandwidth, this mechanism is transparent to the 1394 SSCS entity and only requires peer to peer DLC protocols. In a similar way, when other listeners join the multicast group, the link adaptation for direct link multicast as described in [4] shall be used, so that the correct physical mode is selected. If there is sufficient bandwidth, this is also transparent to the 1394 SSCS entity.
If the link radio quality decreases and there is not enough bandwidth to sustain the connection, then the multicast DLC connection is released. When a 1394 SSCS entity observes a DLC multicast connection release, it shall update the corresponding PCRs and report to the 1394 controllers that a connection has been released.

6.9.2 SSCS procedure

6.9.2.1 General

The procedures defined below are used in three types of operations: allocating, reclaiming and releasing resources.

The wireless devices allocate resources when they want to send or receive streams.

The action of claiming new resources is split in two scenarios:

- Non overlaid connection consists in a wireless 1394 channel setup where both the source and the destination are connected at the same time.
- Overlaid connection allows adding listeners to an existing connection.

The wireless devices shall reclaim resource after a bus reset operation.

6.9.2.1.1 Resources allocation

6.9.2.1.2 Non overlaid connection

One isochronous channel is mapped to one multicast DUC.

First, the 1394 controller shall a command based lock request (as defined in Annex A) towards the IRM (containing the bandwidth, the channel, the talker identifier and one listener identifier. to allocate the channel and the bandwidth. If there are available resources, the lock command succeeds.

The 1394 controller shall then send a lock request to the talker and listener at iPCR and oPCR addresses to set the channel of the previous step.

Both talker and listener shall perform the multicast join procedure on this channel.

After the multicast join procedure is complete, the CC shall start a multicast setup procedure into the multicast group. The stream is opened.

Then, both the talker and the listener shall send a lock response to the 1394 controller.

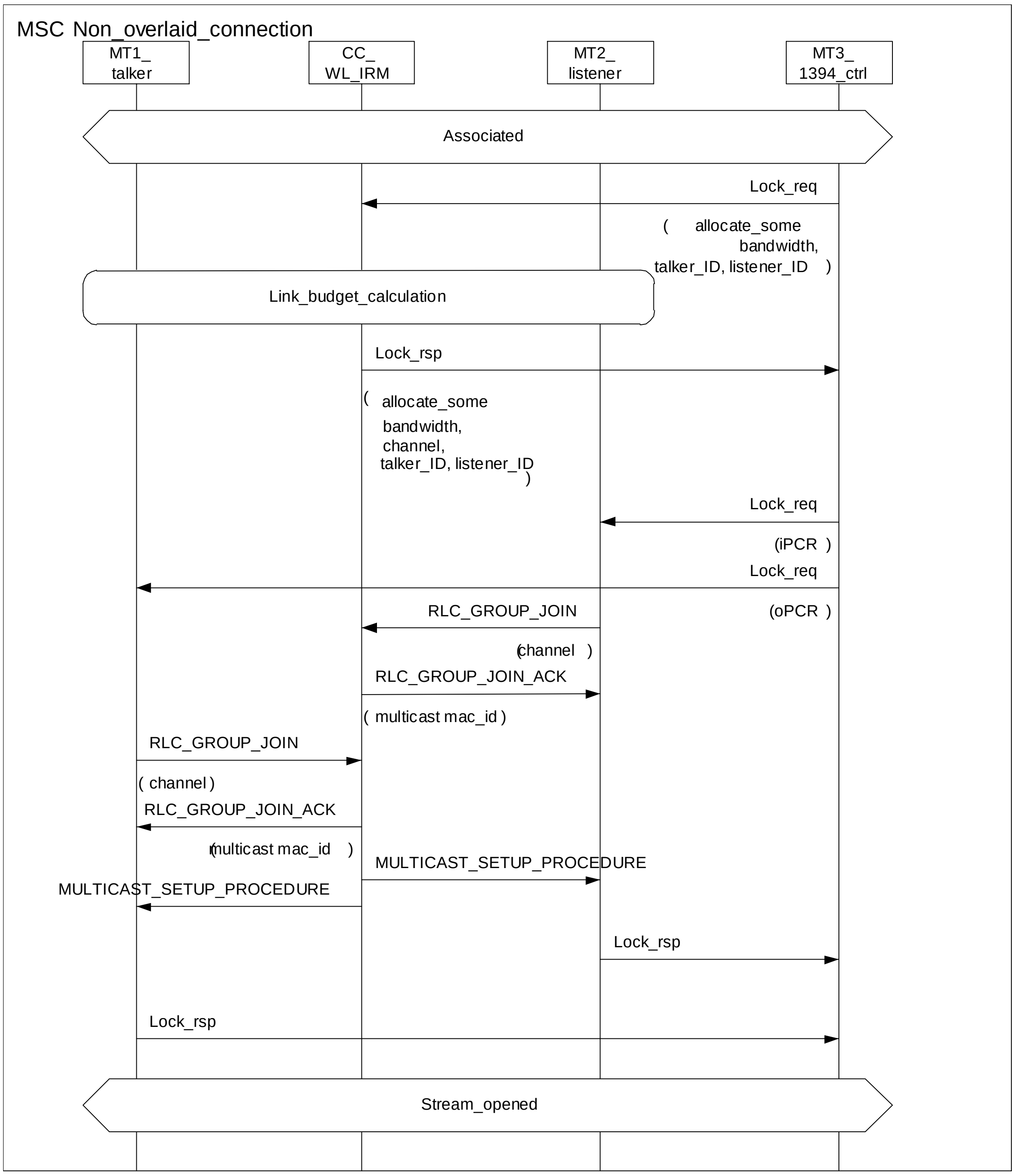

Figure 14: Non overlaid connection mechanism

6.9.2.1.3 Overlaid connection

A HL2 multicast connection for a channel already exists.

The 1394 controller shall set the oPCR of the talker to increment the point to point connection counter.

It shall then set the iPCR of the new listener with the channel by sending a lock request. The new listener shall perform a multicast join procedure.

The CC shall start a multicast setup procedure with the new listener. The stream is opened.

Then, the listener shall send a lock response to the 1394 controller.

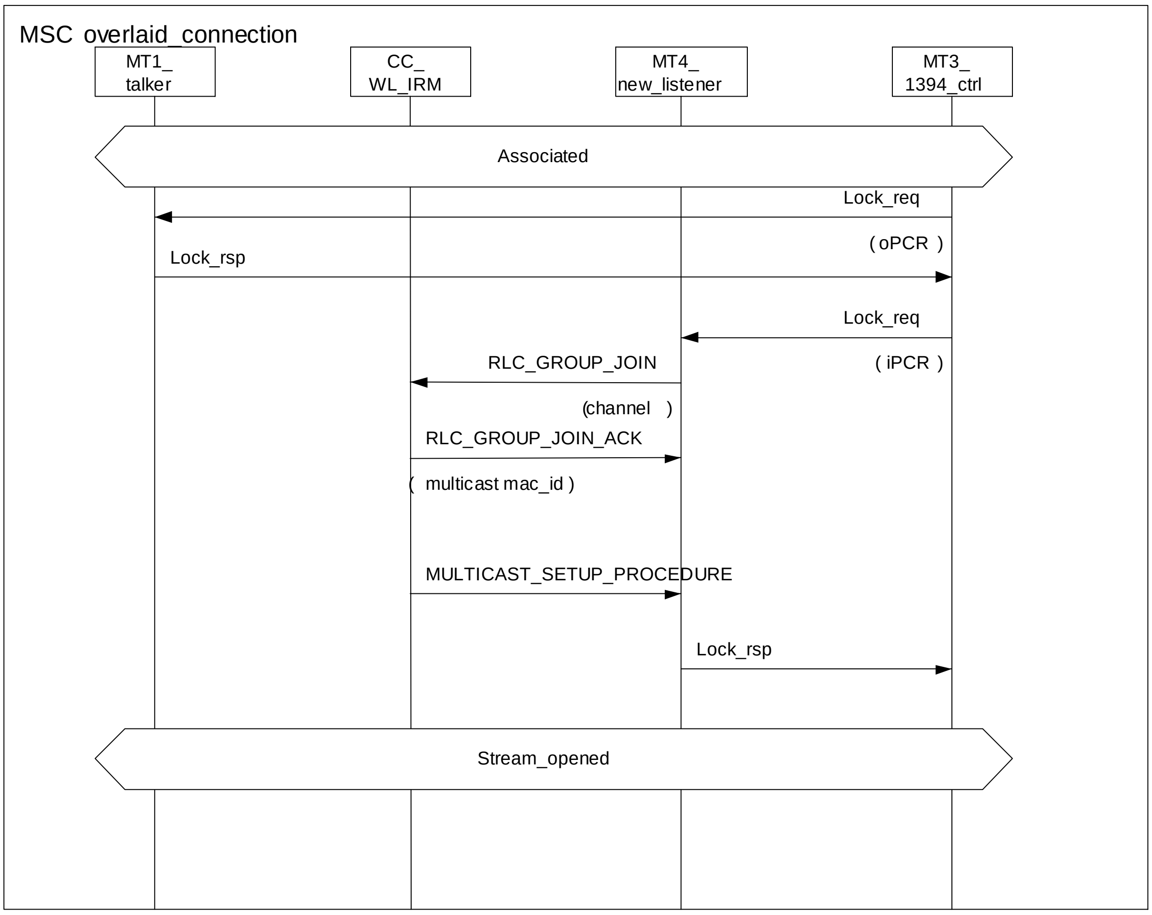

Figure 15: Overlaid connection mechanism

6.9.2.1.4 Resources reclaim

When a 1394 controller is indicated a bus reset, it shall re-allocate all its connections within one second (i.e. send lock request to the IRM as described in 6.9.2.4, and to the PCRs as described in 6.9.2.2 and in 6.9.2.3). After this time it is not allowed to send either re-allocation or normal allocation messages during the worst case bus reset duration time (ΔT), see in 6.4.2.2. But a device can receive and accept a request during this period, as defined in 6.4.

The 1394 controller shall wait 1+ΔT s to allocate new isochronous resources.

6.9.2.1.5 Resources release

To release the resources allocated to an isochronous channel, the 1394 controller shall release the bandwidth and channels reserved in the IRM, by sending the appropriate command to the IRM (see 6.9.2.4 and A.4).

The talker and the listener can autonomously disconnect from the isochronous stream (if the DLC connection has been released) and report that disconnection to the 1394 controller.

6.9.2.2 Procedure at the talker side

The talker shall behave as described in IEC 61883 [5] Part 1, with some specific HL2 behaviours that are described hereafter:

Resource allocation:

When the talker receives a lock request from the 1394 controller on its oPCR, and that oPCR's *channel* field is nonzero, it shall process the compare and swap transaction as specified in wired 1394 (compare the *arg_value* and *data_value*). If the *arg_value* and *data_value* are different, then a negative lock response is generated. Otherwise, the talker shall initiate a multicast join procedure to the CC on this channel.

It shall send a DLC_MULTICAST_JOIN request primitive containing the CHANNEL information element defined below to the CC.

Information element	Reference	Status	Total length (octets)	Description
CHANNEL	7.4.6	mandatory	3	Contains the channel number

The CHANNEL information element shall be transmitted in the <<CL-ATTRIBUTE>> field of the DLC_MULTICAST_JOIN request/response primitive. The *relay* bit, which is defined in 6.9.3.2, is set to 0. The talker gets the multicast mac_ID in the DLC_MULTICAST_JOIN confirmation primitive.

When the talker receives a DLC_MULTICAST_CONNECT indication primitive containing the multicast mac_ID, it shall send a positive lock response to the 1394 controller.

If the multicast join procedure or the multicast setup procedure fails, the talker shall return a lock response with an error code in the *channel* field to the 1394 controller, see A.2.3. If the multicast setup procedure failed due to transient transmission error (timeouts, for instance), a TRANSIENT_ERROR code shall be returned in the lock response and the 1394 controller is expected to retry the resource allocation. If the error is more severe (such as bandwidth limitations), a PERMANENT_ERROR code shall be returned in the lock response and the 1394 controller is not be expected to retry the resource allocation.

Resource reclaim:

When the talker receives a BUS_RESUME information element contained in the <<CL-ATTRIBUTE>> field of the DLC_INFO_TRANSFER request primitive, as defined in the bus reset procedure in 6.4, it shall reset the oPCR and start a T timer (T=1s + ΔT).

If channel bits are not written before the T timeout, the 1394 SSCS shall generate a multicast leave procedure towards the CC to leave the multicast group.

The talker shall leave the multicast group by sending a DLC_MULTICAST_LEAVE request primitive to the CC, for the corresponding multicast mac_ID.

Resource release:

When the *point-to-point connection counter* field or the *broadcast connection counter* field of the oPCR is set to 0 by a 1394 controller, (the talker is disconnected from a channel) then the talker shall release the multicast DLC connection corresponding to that channel, by sending a DLC_MULTICAST_RELEASE request primitive. It shall leave the multicast group (by sending a DLC_MULTICAST_LEAVE request primitive).

If the multicast DLC connection is released by the lower layer, then the *point-to-point connection counter* field and the *broadcast connection counter* field of the oPCR shall be set to 0. The 1394 SSCS entity shall also leave the corresponding multicast group (by sending a DLC_MULTICAST_LEAVE request primitive). It shall also set the *disconnected_pcr* bit in a write request to the INTERRUPT_TARGET CSR of at least the 1394 controllers that modified the corresponding oPCR.

6.9.2.3 Procedure at the listener side

The listener shall behave as described in IEC 61883 [5], Part 1, with some specific HL2 behaviours that are described hereafter:

Resource allocation:

When the listener receives a lock request from the 1394 controller on its iPCR, and that iPCR's *channel* field is nonzero, it shall process the compare and swap transaction as specified in wired 1394 (compare the *arg_value* and *data_value*). If the *arg_value* and *data_value* are different, then a negative lock response is generated. Otherwise, the listener shall initiate a multicast join procedure to the CC on this channel:

It shall send a DLC_MULTICAST_JOIN request primitive containing the CHANNEL information element defined below to the CC.

Information element	Reference	Status	Total length (octets)	Description
CHANNEL	7.4.6	mandatory	3	Contains the channel number

The CHANNEL information element shall be transmitted in the <<CL-ATTRIBUTE>> field of the DLC_MULTICAST_JOIN request/response primitive. The *relay* bit, which is defined in 6.9.3.2, is set to 0.The listener gets the multicast mac_ID in DLC_MULTICAST_JOIN confirmation primitive.

When the listener receives a DLC_MULTICAST_CONNECT indication primitive containing the multicast mac_ID, it shall send a positive lock response to the 1394 controller.

If the multicast join procedure or the multicast setup procedure fails, the listener shall return a lock response with an error code in the *channel* field to the 1394 controller, see A.2.3. If the multicast setup procedure failed due to transient transmission error (timeouts, for instance), a TRANSIENT_ERROR code shall be returned in the lock response and the 1394 controller is expected to retry the resource allocation. If the error is more severe (such as bandwidth limitations), a PERMANENT_ERROR code shall be returned in the lock response and the 1394 controller is not be expected to retry the resource allocation.

Resource reclaim:

When the listener receives a BUS_RESUME information element contained in the <<CL-ATTRIBUTE>> field of the DLC_INFO_TRANSFER request primitive, as defined in the bus reset procedure in 6.4, it shall reset the iPCR and start a T timer ($T=1s + \Delta T$).

If channel bits are not written before the T timeout, the 1394 SSCS shall generate a multicast leave procedure towards the CC to leave the multicast group.

The listener shall leave the multicast group by sending a DLC_MULTICAST_LEAVE request primitive to the CC, for the corresponding multicast mac_ID.

Resource release:

When *point-to-point connection counter* field or the *broadcast connection counter* field of the iPCR is set to 0 (the listener disconnected from a channel) then the listener shall release the multicast DLC connection corresponding to that channel, by sending a DLC_MULTICAST_RELEASE request primitive. It shall leave the multicast group (by sending a DLC_MULTICAST_LEAVE request primitive).

If the multicast DLC connection is released by the lower layer, then the *point-to-point connection counter* field and the *broadcast connection counter* field of the iPCR shall be set to 0. The 1394 SSCS entity shall also leave the corresponding multicast group (by sending a DLC_MULTICAST_LEAVE request primitive). It shall also set the *disconnected_pcr* bit in a write request to the INTERRUPT_TARGET CSR of at least the 1394 controllers that modified the corresponding oPCR.

6.9.2.4 Procedure at the IRM side

The IRM can receive different lock request commands, which are defined in A.4.
When the IRM receives a lock request from the 1394 controller, it shall perform the requested action. If the action is done, the IRM shall send a lock response to the 1394 controller with the *status* field set to DONE code. If the action is not correctly done, it shall send a lock response to the 1394 controller with the *status* field set to the appropriate failure code.

The text below describes the different lock request commands.

Resource allocation on a new channel (ALLOCATE_SOME):

If the IRM receives an ALLOCATE_SOME request while being processing a bus reset (i.e. the CC started a bus reset and did not complete it yet, see 6.4), then a ALLOCATE_SOME response shall be answered (with the *status* field set to RESET code). Neither bandwidth nor channel shall be reserved.

Else, if the IRM receives an ALLOCATE_SOME request (new resource reservation on a not yet existing channel), containing the *talker_ID*, the *listener_ID*, the isochronous *channel* and the *bandwidth* parameters, it shall check if it is possible to allocate such bandwidth on the requested channel. The 1394 SSCS of the CC gets the information about the link radio quality from the DLC layer. The exact procedure is out of the scope of the standard.

If there are enough available resources, the IRM shall send an ALLOCATE_SOME response with the *status* field set to DONE code and the *handle* field containing the allocated multicast mac_ID. It shall also update the bandwidth and channel in the IRM.

If there are not available resources (not enough bandwidth, or channel already reserved), the IRM shall send an ALLOCATE_SOME response, with the *status* field set to an appropriate failure code (indicating which from the bandwidth or the channel was not available). Whatever the failure is (channel or bandwidth failure), neither bandwidth nor channel shall be reserved.

When the IRM receives a DLC_MULTICAST_JOIN request primitive, it shall perform a multicast join procedure. It shall then respond with a DLC_MULTICAST_JOIN response primitive.

When the IRM has completed the multicast join procedure for both the talker and the listener, it shall start a multicast setup procedure between the talker and the listener, as described in [4]. The IRM shall send a DLC_MULTICAST_CONNECT request primitive to the DLC.

If the multicast setup fails, the IRM shall not free resources by itself, but shall wait for the 1394 controller to free allocated resources via appropriate lock command.

Resource allocation on an existing channel (MODIFY_BANDWIDTH)

If the IRM receives a MODIFY_BANDWIDTH request while being processing a bus reset (i.e. the 1394 SSCS entity started a bus reset and did not complete it yet, see 6.4), then an MODIFY_BANDWIDTH response shall be answered (with the *status* field set to RESET code).

Else, if the IRM receives a MODIFY_BANDWIDTH request (asking for a modified bandwidth in a specified channel), it shall check if it is possible to modify such bandwidth on the requested channel. This exact procedure is out of the scope of the standard.

If the modification of the bandwidth is correctly done, the IRM shall send a MODIFY_BANDWIDTH response, with the *status* field set to DONE code and the *handle* field containing the allocated multicast MAC_ID. It shall also update the bandwidth in the IRM (the exact procedure is out of the scope of the present document). It shall then send the DLC_MULTICAST_MODIFY request primitive to the DLC layer to modify the multicast connection.

If the modification of the bandwidth is not correctly done, the IRM shall send an MODIFY_BANDWIDTH response, with the *status* field set to the appropriate failure code.

Note: a new listener may also be connected to an existing channel, without claiming for new resources on the IRM. This will appear to the IRM by a DLC_MULTICAST_JOIN indication primitive. The IRM shall then setup the multicast DLC connection corresponding to the 1394 channel towards this new listener.

Resource reclaim (RECLAIM_THIS):

When the CC side of the 1394 SSCS entity receives a BUS_RESUME information element contained in the DLC_INFO_TRANSFER request primitive, as defined in the bus reset procedure in 6.4, it shall release the bandwidth and channel registers and start a T timer ($T=1s + \Delta T$).

When the IRM receives a RECLAIM_THIS request (asking for a resource reclaim), it shall update the channel and the bandwidth and set the corresponding resources as reclaimed.

As already said any command different than a RECLAIM_THIS shall be rejected by sending the corresponding command in a lock response (with the *status* field set to RESET code).

If the re-allocation of the resources is done, it shall send a RECLAIM_THIS response, with the *status* field set to DONE code.

If the re-allocation of the resources is not correctly done, it shall send a RECLAIM_THIS response, with the *status* field set to the appropriate failure code.

When the timer T expires, unreclaimed resources are released (Group mac_IDs are released in the CL). The IRM shall then initiate a multicast release procedure toward the talker and each listener of these Group mac_IDs.

Resource release (RELEASE_THIS):

If the CC receives a RELEASE_THIS request, while being processing a bus reset (i.e. the CC started a bus reset and did not complete it yet, see 6.4), then a RELEASE_THIS response shall be answered (with the *status* field set to RESET code).

When the IRM receives a RELEASE_THIS request, (asking for a release of an isochronous connection), it shall free the reserved channel and bandwidth. It shall also send a DLC_MULTICAST_RELEASE request primitive for the corresponding multicast mac_ID.

If the DLC_MULTICAST_RELEASE succeeds, it shall send a RELEASE_THIS response, with the *status* field set to DONE code.

Otherwise, it shall send a RELEASE_THIS response, with the *status* field set to a failure code.

6.10 Asynchronous streams connection control service

6.10.1 IRM and PCR functions

IRM and PCR functions are used as described in 6.9 with some specificity:

- The 1394 controller reserves a channel with no bandwidth. The channel 31 is reserved for the global broadcast.
- No bandwidth is written in the PCR.
- After WTs have joined, the IRM does not perform any RLC multicast DLC connection setup. WTs are allowed to use the asynchronous stream as soon as they have joined the channel.

6.10.2 Asynchronous stream relaying

6.10.2.1 General

In the case some WTs do not have any radio contact between them, correctness of higher layer rely on a relaying mechanism for asynchronous stream data. This shall be done as described below:

There is a distinction between the main channel and the forwarding channel.

The main channel is defined by an IEEE 1394 [6] channel number. It is identified in the CHANNEL information element with the *relay* bit set to 0. A multicast mac_ID is defined and associated to the main channel. Any WT is allowed to send data in the main channel. Any WT, belonging to the main multicast group is allowed to listen and receive data from the main channel.

The forwarding channel is defined by the same IEEE 1394 [6] channel as the main channel. It is identified in the CHANNEL information element with the *relay* bit set to 1. When used, a specific multicast mac_ID is defined and associated to it (different from the main channel multicast mac_ID). The 1394 SSCS entity of the CC has to repeat on the forwarding channel any data sent on the main channel.

6.10.2.2 Procedures

6.10.2.2.1 Procedure in the WT side

When a WT fails in receiving data from one sender on the main multicast channel carrying an asynchronous stream (a IEEE 1394 [6] channel with no bandwidth), it shall ask to join the forwarding channel.

It shall send a DLC_MULTICAST_JOIN request primitive containing the CHANNEL information element defined below to the CC.

Information element	Reference	Status	Total length (octets)	Description
CHANNEL	7.4.6	mandatory	3	Channel set to forwarding channel number; *relay* bit set to 1

The CHANNEL information element shall be transmitted in the <<CL-ATTRIBUTE>> field of the DLC_MULTICAST_JOIN request/response primitive.

6.10.2.2.2 Procedure in the CC side

When at least one WT has asked to join the forwarding channel, the 1394 SSCS entity of the CC has to allocate a multicast mac_ID for the forwarding channel.

It shall send a DLC_MULTICAST_JOIN response primitive, with the allocated mac_ID, containing the CHANNEL information element defined below to the WT.

Information element	Reference	Status	Total length (octets)	Description
CHANNEL	7.4.6	mandatory	3	Channel set to forwarding channel number; *relay* bit set to 1

The CHANNEL information element shall be transmitted in the <<CL-ATTRIBUTE>> field of the DLC_MULTICAST_JOIN request/response primitive.

It shall then repeat in the forwarding channel any data received in the main channel.

The forwarding multicast group is released when the last WT left the forwarding multicast group.

7 Convergence layer specific parameters

7.1 Convergence layer identifier

The convergence layer ID field of the convergence layer IE, defined in [2], shall be set according to Table 10 to signal the support of 1394 SSCS in the WT and AP. The first 3 bits identifies the convergence layer. In case of the Packet based CL the next 5 bits identifies the Service Specific Convergence Sublayer.

Table 10: convergence layer identifier

Bits 8 7 6 5 4 3 2 1	Meaning
0 0 1 0 0 0 0 1	1394 SSCS supported

All other values reserved

7.2 Convergence layer version

The convergence layer version number is an 8-bit field used in the RLC [2]. This field is split into two 4-bit subfields. The four most significant bits (bits 5 to 8) identify the version of the Common Part and the four least significant bits (bits 1 to 4) identify the version of the SSCS. The convergence layer version number shall be set according to Table 11 to indicate the support of this edition (version 1) of the 1394 SSCS.

Table 11: convergence layer version

Bits 8 7 6 5 4 3 2 1	Meaning
x x x x 0 0 0 1	1394 SSCS version 1 supported

7.3 Maximum transmission unit

The maximum transmission unit (MTU) used in the Common Part of the Packet based Convergence Layer [1] shall be set to 12 432 octets.

7.4 Information elements for 1394 SSCS

7.4.1 Information element format

In order to transfer convergence layer specific information between CC and WT, as well as between two WTs, a number of CL information elements are defined. These convergence layer specific information elements are transferred transparently by RLC messages within the RLC specific field <<CL-ATTRIBUTES>>, as defined in [2].

Each convergence layer information element consists of a *type* field, a *length* field and an *information* field, as defined in Figure 16.

The purpose of the information element ***type*** is to identify the function of the information element being sent. The IE *type* field is 8 bits, allowing for up to 256 information elements to be defined for a certain SSCS.

The ***reserved*** field is for future use and shall be coded as zero.

The ***length*** field indicates the length of the information field in bytes. It does not include the length of the IE *type* field, the *reserved* field, or the length of the *length* field itself.

	8	7	6	5	4	3	2	1
Octet 1	type							
Octet 2	reserved		length = L					
Octet 3	information							
...								
Octet L+2								

Figure 16: convergence layer information element structure

7.4.2 Information element type coding

The IE *type* codes of Table 12 are used in the 1394 SSCS:

Table 12: information element *type* coding

IE Type code	IE Name
0000 0001$_2$	HARP_REQUEST (see 7.4.3)
0000 0010$_2$	HARP_RESPONSE (see 7.4.4)
0000 0011$_2$	EUI_64 (see 7.4.5)
0000 0100$_2$	CHANNEL (see 7.4.6)
0000 0101$_2$	BUS_SUSPEND (see 7.4.7)
0000 0110$_2$	BUS_RESUME (see 7.4.8)
0000 0111$_2$	BUS_RESET (see 7.4.9)
0000 1000$_2$	TRANSACTION_INDICATOR (see 7.4.10)

All other values reserved for future IEs.

7.4.3 HARP_REQUEST information element

The <<HARP_REQUEST>> information element is used in the HARP.

	8	7	6	5	4	3	2	1
Octet 1	<<HARP_REQUEST>>							
Octet 2	reserved		length = 2					
Octet 3	bus_ID[9..2]							
Octet 4	bus_ID[1..0]		reserved					

Figure 17: HARP_REQUEST information element

7.4.4 HARP_RESPONSE information element

The <<HARP_RESPONSE>> information element is used in the HARP.

	8	7	6	5	4	3	2	1
Octet 1	<<HARP_RESPONSE>>							
Octet 2	reserved		length = 3					
Octet 3	bus_ID[9..2]							
Octet 4	bus_ID[1..0]		physical_ID					
Octet 5	reserved							fwd

Figure 18: HARP_RESPONSE information element

7.4.5 EUI_64 information element

The <<EUI_64 >> information element is used to inform the CC of the WT's EUI-64 during association.

The node type informs the CC if the node is a bridge or not. The value 0 indicates a non-bridge node. Other values indicate a bridge node.

	8	7	6	5	4	3	2	1
Octet 1	<<EUI_64>>							
Octet 2	reserved		length = 9					
Octet 3	GUID							
Octet 4								
Octet 5								
Octet 6								
Octet 7								
Octet 8								
Octet 9								
Octet 10								
Octet 11	node_type							

Figure 19: EUI_64 information element

7.4.6 CHANNEL information element

The <<CHANNEL>> information element is used in the multicast join procedure. It is used by the WT to inform the CC about the channel number. It is also used in the asynchronous streams connection control service in the stream relaying procedure.

	8	7	6	5	4	3	2	1
Octet 1	<<CHANNEL>>							
Octet 2	reserved		length = 1					
Octet 3	channel						relay	reserved

Figure 20: CHANNEL information element

The *relay* field makes the distinction between the main channel and the forwarding channel, as defined in 6.5.3.2.

The values of the *channel* are defined in Table 13. The **c[0]** to **c[31]** bits specify the isochronous channel that is concerned.

Table 13: Values of the channel

Value	Status
C[0] to C[29]	Asynchronous & isochronous stream
C[30]	Clock channel
C[31]	Global broadcast
C[32] to C[47]	Reserved
C[48] to C[63]	Error, see Annex A

7.4.7 BUS_SUSPEND information element

The <<BUS_SUSPEND>> information element is used in the self-identity process. It is used by the CC to inform WTs about the start of the bus reset phase.

	8	7	6	5	4	3	2	1
Octet 1	<<BUS_SUSPEND>>							
Octet 2	Reserved		length = 2xN					
Octet 3	mac_ID (of the IRM)							
Octet 4	node_type		physical_id (of the IRM)					
Octet 5	mac_ID							
Octet 6	node_type		physical_id					
...	...							
Octet 2xN-1	mac_ID							
Octet 2xN	node_type		physical_id					

Figure 21: BUS_SUSPEND information element

Fields shall be as follows:

- *N* is the number of wireless nodes in the HL2 Bus. N shall not be greater than 20.
- *node-type* is as delivered by the upper layer in the CL_SELF_ID primitive (see 6.2.2).
- The first two bytes of the list shall contain information related to the IRM.

7.4.8 BUS_RESUME information element

The <<BUS_RESUME>> information element is used in the self-identity process. It is used by the CC to inform WTs about the completion of the bus reset phase.

	8	7	6	5	4	3	2	1
Octet 1	<<BUS_RESUME>>							
Octet 2	reserved		length = 2xN					
Octet 3	mac_ID (of the IRM)							
Octet 4	node_type		physical_id (of the IRM)					
Octet 5	mac_ID							
Octet 6	node_type		physical_id					
...	...							
Octet 2xN-1	mac_ID							
Octet 2xN	node_type		physical_id					

Figure 22: BUS_RESUME information element

<<BUS_RESUME>> IE fields shall be identical to those of the <<BUS_SUSPEND>> IE.

7.4.9 BUS_RESET information element

The <<BUS_RESET>> information element is used in the self-identity process. It is used by the WT to initiate the bus reset phase.

	8	7	6	5	4	3	2	1
Octet 1	<<BUS_RESET>>							
Octet 2	reserved		length = 2					
Octet 3	mac_ID							
Octet 4	node_type		physical_ID					

Figure 23: BUS_RESET information element

mac_ID, *node_type* and *physical_id* fields are those of the WN that requests the bus reset.

7.4.10 TRANSACTION_INDICATOR information element

The <<TRANSACTION_INDICATOR>> information element is used in the DLC user connection control process, when a DLC connection is setup for a 1394 transaction data service.

	8	7	6	5	4	3	2	1
Octet 1	<<TRANSACTION_INDICATOR>>							
Octet 2	reserved		length = 1					
Octet 3	reserved							request

Figure 24: TRANSACTION_INDICATOR information element

The values of the *request* bit are specified in Table 14:

Table 14: *request* values

request	Name	Cause
0	REQUEST	Initiating transaction is a requester
1	RESPONSE	Initiating transaction is a responder

Annex A (normative): HIPERLAN/2 CSRs

The intent of the present document is to isolate the higher level 1394 applications from the physical nature of the HL2 DLC and PHY characteristics, so that only minor changes are required when porting an application from wired 1394 to wireless 1394. However, there are still several differences, due to 1394 physical related CSRs that are meaningless for a HL2 bus, as well as some specificity from the wireless medium that have been minimized (new lock format for IRM registers).

A.1 CSR registers

A.1.1 Standard CSR registers

For HL2 Bus, the standard CSRs of Table A.1 shall be implemented.
NODE_IDS, MESSAGE_REQUEST and MESSAGE_RESPONSE registers are the same as defined in wired 1394.
SPLIT_TIMEOUT, CYCLE_TIME and BUS_TIME register format is the same as in wired 1394, but their exact usage is defined hereafter.
INTERRUPT_TARGET, LOCAL_SECONDS and LOCAL_CYCLES registers (shaded in Table A.1) are additional registers that are not specified by wired 1394.
Other wired 1394 defined CSRs are not relevant for HL2 (as described in A.1.2) and shall not be implemented.

Table A.1: Standard CSR registers

offset	register	reference	description
8_{16}	NODE_IDS	[6]	Address selection
18_{16}	SPLIT_TIMEOUT	A.1.8	Local-bus split-response timeout
50_{16}	INTERRUPT_TARGET	A.1.3	Controller event register
$5C_{16}$	LOCAL_CYCLES	A.1.6	Local-bus free-running timer (LSBs)
58_{16}	LOCAL_SECONDS	A.1.7	Local-bus free-running timer (MSBs)
80_{16}	MESSAGE_REQUEST	[6]	Message request register
$C0_{16}$	MESSAGE_RESPONSE	[6]	Message-response register
200_{16}	CYCLE_TIME	A.1.4	Net-synchronous isochronous timer (MSBs)
204_{16}	BUS_TIME	A.1.5	Net-synchronous isochronous timer (LSBs)

Except for the INTERRUPT_TARGET, LOCAL_CYCLES and LOCAL_SECONDS registers, the locations and formats of these registers are specified by the CSR Architecture. These offset addresses are also listed here, for the convenience of the reader; in the case of a definition conflict, the CSR Architecture shall have precedence.

A.1.2 Unimplemented wired-1394 registers

The 1394-1995 and 1394a-2000 bus dependent registers, defined in [6] and [7] respectively , which are listed below shall not be implemented:

1394-1995 registers:

POWER_FAIL_IMMINENT	— Only applicable to backplane shared power-supply designs
POWER_SOURCE	— "
BUSY_TIMEOUT	— Managed by end-to-end timeout
BUS_MANAGER_ID	— "
BANDWIDTH_AVAILABLE	— Replaced by message-based IRM
CHANNELS_AVAILABLE	— "
MAINT_CONTROL	— Unnecessary and not applicable
MAINT_UTILITY	— Unnecessary and not applicable
TOPOLOGY_MAP	— Unnecessary and not applicable
SPEED_MAP	— Unnecessary and not applicable

1394a-2000 registers:

PRIORITY_BUDGET	— Not used
BROADCAST_CHANNEL	— Unnecessary

A.1.3 INTERRUPT_TARGET register

All isochronous controllers shall implement this register.

This register provides a target address for sending of interrupt-event indication to the controller portion of a node. A quadlet write transaction data corresponds to 32 events that can be signalled. Only one of these events is specified (this corresponding to the least-significant bit of this register), as illustrated in Figure A.1.

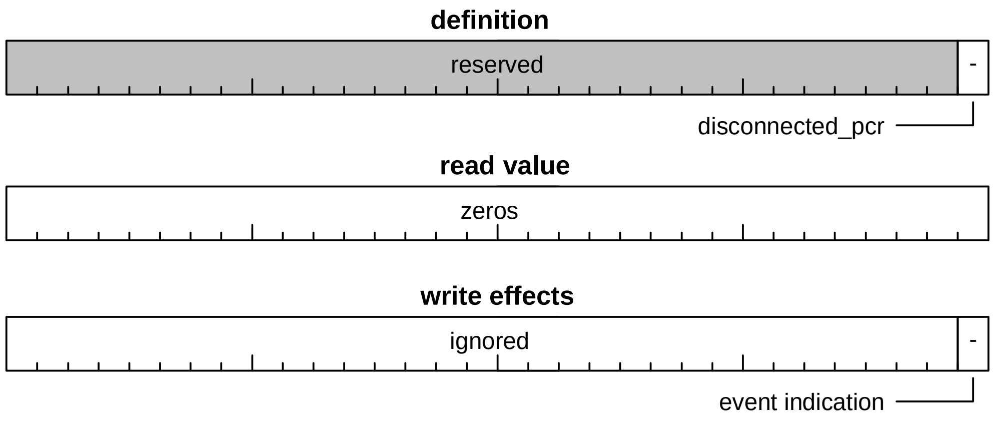

Figure A.1: INTERRUPT_TARGET register

A quadlet write with *disconnected_pcr* bit set to one indicates that the requesting node has been disconnected from an isochronous stream and that PCR has been updated. That write shall be ignored if the *disconnected_pcr* bit is zero.

A.1.4 CYCLE_TIME register

Distinction: This register is read-only.
Distinction: This register is updated once each 2ms frame, rather than once every 125 µs cycle.
Distinction: The synchronizing clock-master node is not necessarily the IRM.
All nodes capable of providing isochronous services shall implement this register. The implementation is optional for nodes that do not provide isochronous services.

Nodes that are isochronous capable shall implement a clock that shall run freely and shall update the contents of the CYCLE_TIME register. The CYCLE_TIME register shall also be updated, once each 2ms frame, to the clock-reference value implied by values transmitted within frames. The fields within the CYCLE_TIME register specify the current time value, in a 1394 compatible fashion, as illustrated in Figure A.2.

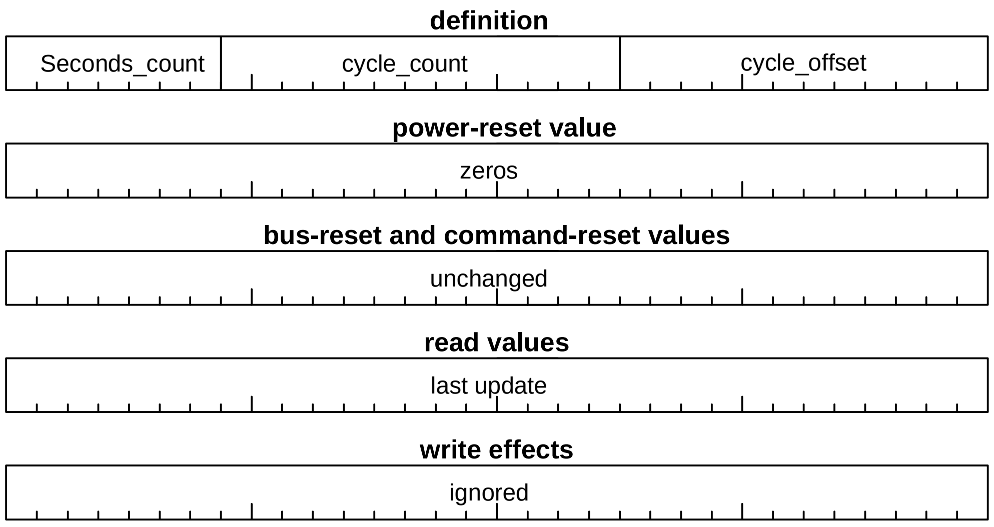

Figure A.2: CYCLE_TIME register

The 7-bit ***second_count*** field is an exact copy of the 7 least significant bits of the BUS_TIME register.

The 13-bit ***cycle_count*** field specifies the time offset from the last integer second value, in units of 125µs.

The 12-bit ***cycle_offset*** field specifies the time offset from the last integer *cycle_count* value, in units of 125µs/(2 048+1 024). This prevision corresponds to one tick of a 24,576 MHz clock.

A.1.5 BUS_TIME register

Distinction: This register is read-only and is updated by the clock-master node once each 2ms frame
All nodes capable of providing isochronous services, and then implementing the CYCLE_MASTER register, shall implement this register. The implementation is optional for nodes that do not provide isochronous services.

Nodes that are isochronous capable shall implement a clock that shall run freely and shall update the contents of the BUS_TIME register. The BUS_TIME register shall also be updated, once each 2ms frame, to the clock-reference value implied by values transmitted within frames. The BUS_TIME register specifies the current time value, in seconds, as illustrated in Figure A.3.

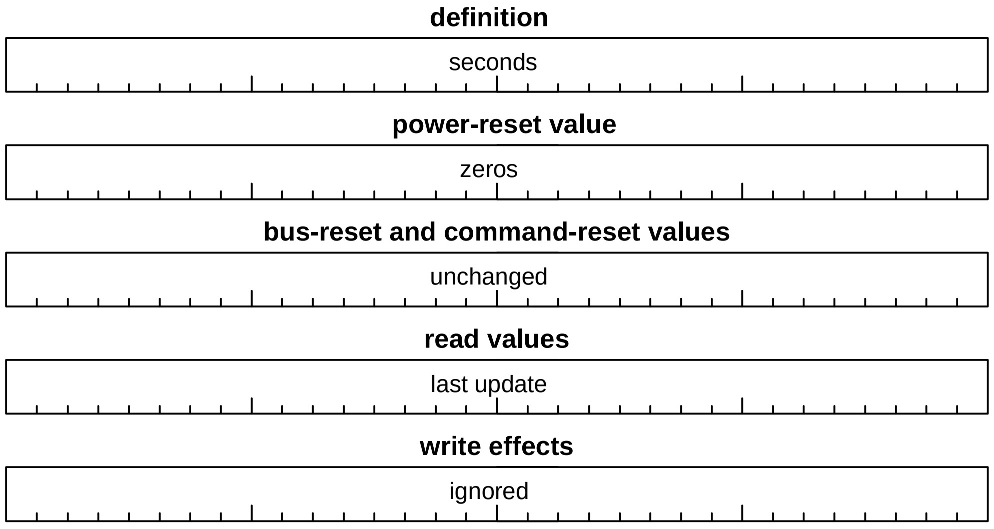

Figure A.3: BUS_TIME register

The 32-bit ***seconds*** field provides a measure of the current time in seconds and has a range of 2^{32} seconds, or approximately 136 years.

A.1.6 LOCAL_CYCLES register

All nodes shall implement the LOCAL_CYCLES register. This register provides access to a local-synchronized clock value. This clock is provided to support asynchronous split-response timeouts.

The fields within this register specify the current time value, in a CYCLE_TIME compatible fashion, as illustrated in Figure A.4.

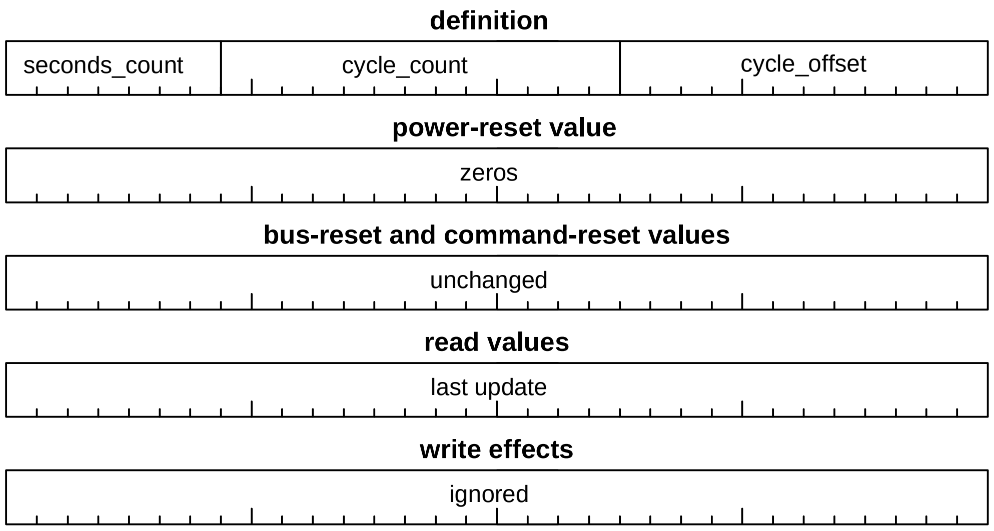

Figure A.4: LOCAL_CYCLES register

The 7-bit ***second_count*** field is an exact copy of the 7 least significant bits of the LOCAL_SECONDS register.

The 13-bit ***cycle_count*** field specifies the time offset from the last integer second value, in units of 125 µs.

The 12-bit ***cycle_offset*** field specifies the time offset from the last integer *cycle_count* value, in units of 125 µs/(2 048+1 024). This prevision corresponds to one tick of a 24,576 MHz clock.

The behaviour of this register depends on a node's clock-master capabilities, as follows:

a) Clock-master. The clock-master node, based on its free running clock, asserts its time value every 2 ms frame, for the purpose of synchronizing non clock-master nodes.

b) Non clock-master. This register shall be updated, once each 2 ms frame, based on the clock-reference value transmitted by the clock-master node within frames. Between these updates this register is autonomously increased by the node.

A.1.7 LOCAL_SECONDS register

All nodes shall implement the LOCAL_SECONDS register. The fields within this register specify the current time value, in a BUS_TIME compatible fashion. The register specifies the current time value, in seconds, as illustrated in Figure A.5.

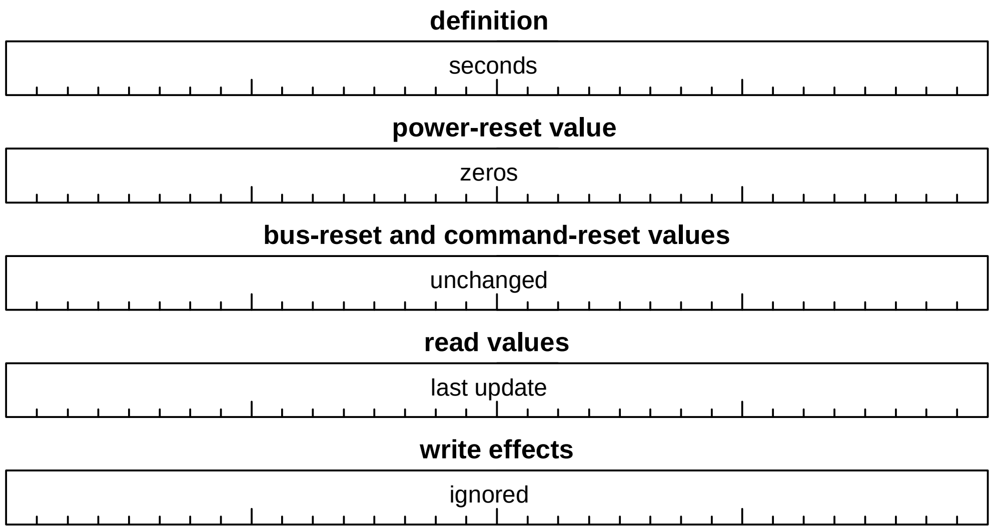

Figure A.5: LOCAL_SECONDS register

The 32-bit ***seconds*** field provides a measure of the current time in seconds and has a range of 2^{32} s, or approximately 136 years.

The behaviour of this register depends on a node's clock-master capabilities, as described in A.1.6.

A.1.8 SPLIT_TIMEOUT register

Distinction: the initial value of this register corresponds to 200 ms, as opposed to the initial wired-1394 100 ms value.
Distinction: the measurement points and the duration of timeouts differ from 1394, see C.2.
All nodes shall implement this register.

Split-transaction error detection expects all HL2 Bus nodes to share the same time-out value and that requester and responder behave in complementary fashion, as specified in C.2. The SPLIT_TIMEOUT register establishes the time-out value for the detection of split-transaction errors, as illustrated in Figure A.6.

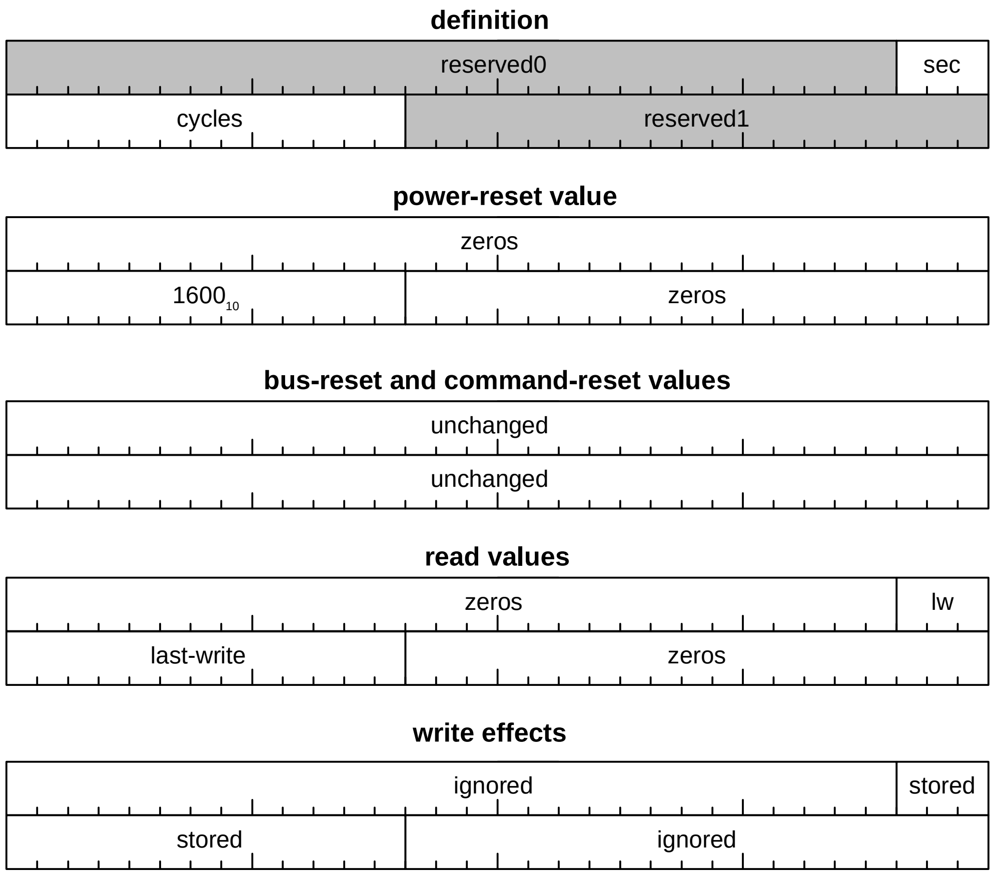

Figure A.6: SPLIT_TIMEOUT register

The 3-bit ***sec*** field, in units of seconds, and the 13-bit ***cycles*** field, in units of 125 µs, together specify the time-out value. The value of *cycles* shall be less than 8 000.

The minimum time-out value is 1 600 cycles (0,2 s). If a value smaller than this is written to the SPLIT_TIMEOUT register the node shall not revise the stored value but shall behave as if a value of 1 600 had been stored.

NOTE: The Serial Bus definition of the SPLIT_TIMEOUT register differs from that of the CSR Architecture. Serial Bus interprets the most significant 13 bits of the SPLIT_TIMEOUT_LO register as units of 1/8 000 s, rather than a true binary fraction of a second with units of 1/8 192 s. Since precise time-outs are not necessary, the bus manager may ignore this difference when calculating values for use within the SPLIT_TIMEOUT_LO register.

A.2 PCRs

A.2.1 oPCR format

Distinction: The oPCR payload contents are specified differently and include bus-dependent overhead.

The oPCR is specified in clause 7.7 of the IEC 61883 [5], Part 1.

The usage of the oPCR is the same as described in IEC 61883 [5], Part 1, with the addition that some HL2 specific error codes are specified in the present document. These HL2 specific error codes (see A.2.3) are used in the PCR lock response to indicate an error to the initiating controller.

The format of the HL2 oPCR is also slightly different from the IEC 61883 [5] oPCR (as described below).

The oPCR is depicted in Figure A.7.

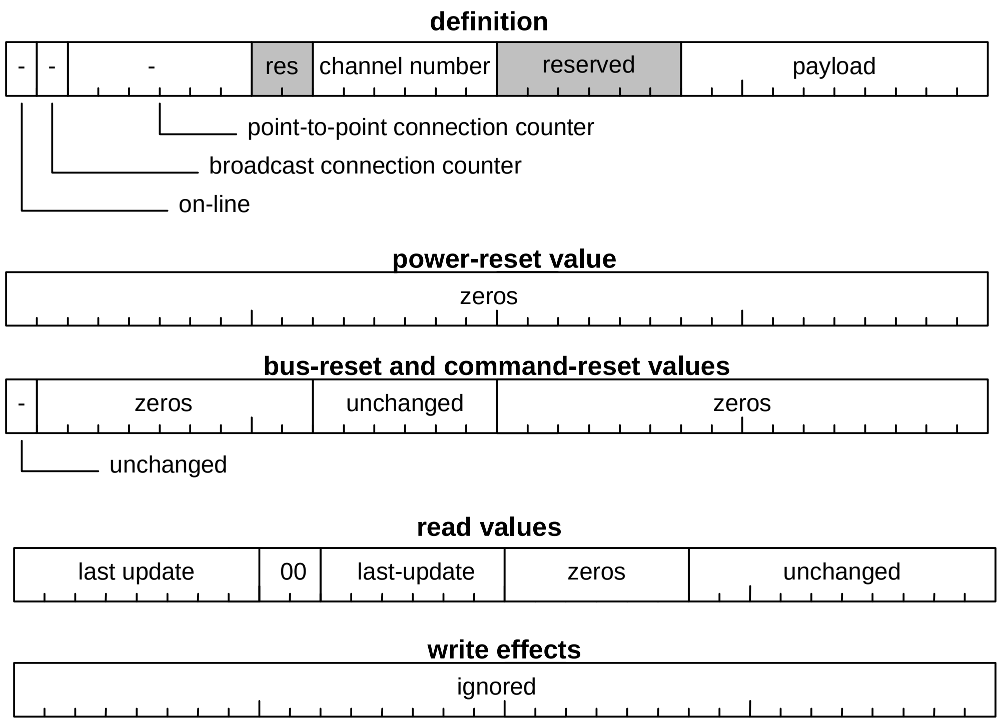

Figure A.7: oPCR format

The ***on-line*** bit always indicates whether the corresponding output plug is *on-line* (value one) or *off-line* (value zero).

The ***broadcast connection counter*** bit indicates whether a broadcast-out connection to the output plug exists (value one) or not (value zero).

The 6-bit ***point-to-point connection counter*** indicates the number of point of point connections to the output plug.

The 6-bit ***channel number*** indicates the actual channel number used by the output plug.

The 10-bit ***payload*** reflects the maximum number of bytes to be transmitted within a 2 ms HL2 frame (including the SSCS and CPCS overheads) divided by 16.

The HL2 Bus 500 Hz (± 20 ppm) physical layer framing clock is not synchronized to the wired-1394 8 kHz (± 100 ppm) cycle clock. To account for the clock deviations, the application shall add a 1-byte margin to the oPCR.*payload* value.

A.2.2 iPCR format

The iPCR is specified in clause 7.8 of the IEC 61883 [5], Part 1. The format of the iPCR is depicted in Figure A.8:

The usage of the iPCR is the same as described in IEC 61883 [5] Part 1, with the addition that some HL2 specific error codes are specified in the present document. These HL2 specific error codes (see A.2.3) are used in the PCR lock response to indicate an error to the initiating controller.

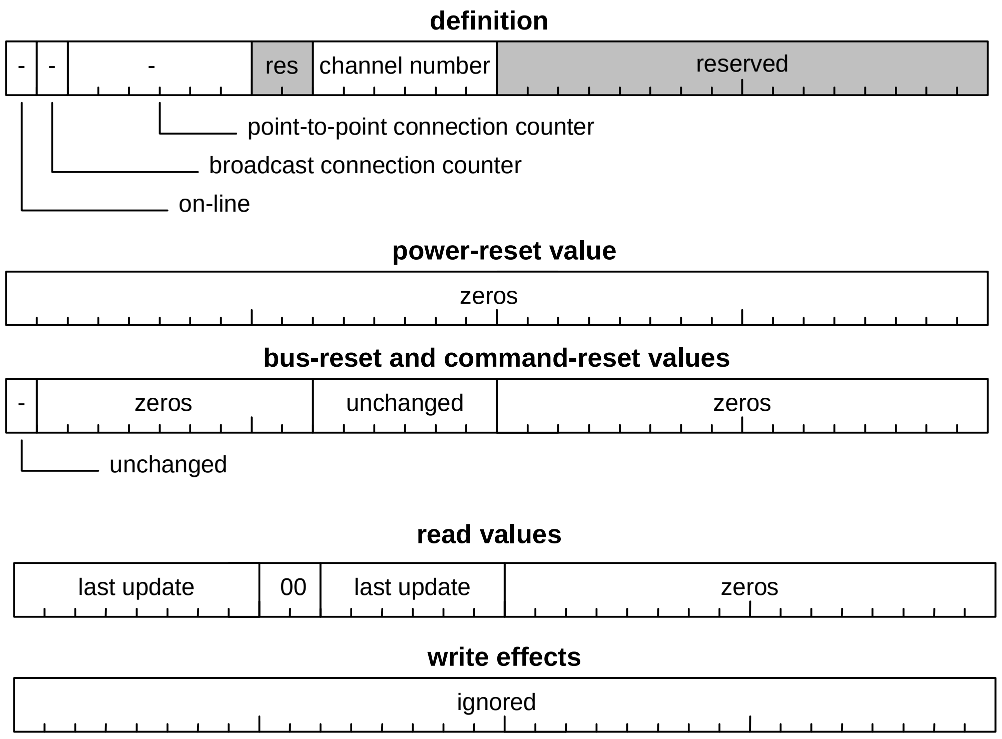

Figure A.8: iPCR format

The definition of the fields is the same as in the oPCR.

The ***on-line*** bit, the ***broadcast connection counter*** bit, the 6-bit ***point-to-point connection counter*** field, and the 6-bit ***channel number*** field are described in IEC 61883 [5], subclause 7.8 of Part 1.

A.2.3 PCR error codes

Lock transactions on PCRs can fail due to HL2 specific reasons, as described in 6.9.2.2 and in 6.9.2.3. Therefore some specific channel numbers are reserved for that purpose (see 7.4.6). When an error occurs, these reserved channel numbers shall be used in the lock response, as defined in Table A.2.

Table A.2: PCR error codes

Channel value	Name	Description
62	TRANSIENT_ERROR	A non fatal error occurred
63	PERSISTENT_ERROR	A severe error occurred

When a TRANSIENT_ERROR is reported, the 1394 controller is expected to retry the lock request on the PCR.

When a PERSISTENT_ERROR is reported, the 1394 controller is not expected to retry the lock request on the PCR until the error condition has been resolved.

A.3 ROM information

The HL2 Bus provides 1394 services over an alternate (RF) physical layer. The *bus_info_block* has a distinctive "0000" label (to avoid possible confusion with wired-1394), but is otherwise as specified by wired 1394. Other bus-directory entries are used to better identify the HL2 physical layer and its bus-dependent CSR behaviours, as illustrated in Figure A.9.

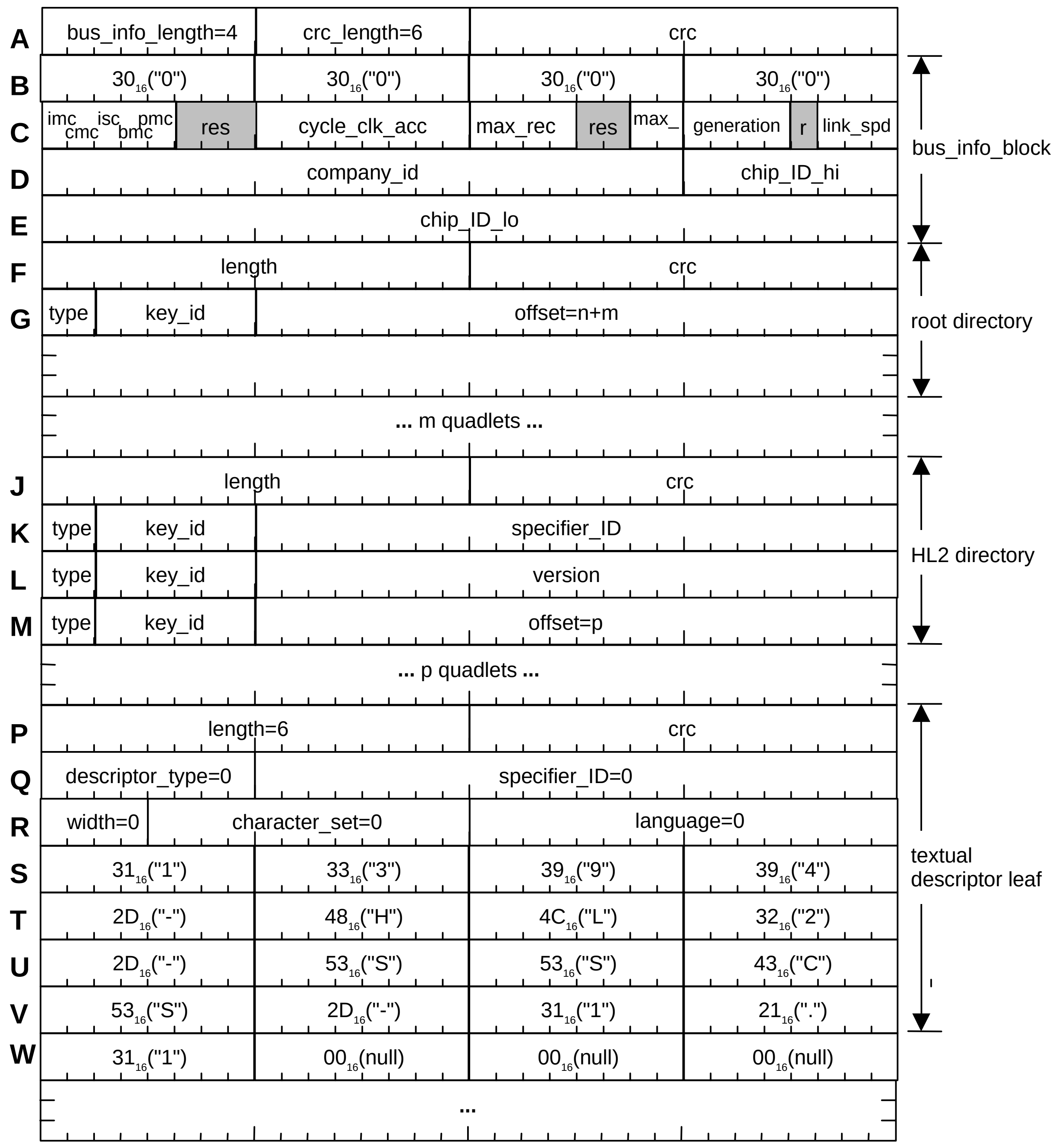
A bus_info_length=4 | crc_length=6 | crc
B 30₁₆("0") | 30₁₆("0") | 30₁₆("0") | 30₁₆("0")
C imc cmc isc bmc pmc | res | cycle_clk_acc | max_rec | res | max_ | generation | r | link_spd
D company_id | chip_ID_hi
E chip_ID_lo
F length | crc
G type | key_id | offset=n+m
... m quadlets ...
J length | crc
K type | key_id | specifier_ID
L type | key_id | version
M type | key_id | offset=p
... p quadlets ...
P length=6 | crc
Q descriptor_type=0 | specifier_ID=0
R width=0 | character_set=0 | language=0
S 31₁₆("1") | 33₁₆("3") | 39₁₆("9") | 39₁₆("4")
T 2D₁₆("-") | 48₁₆("H") | 4C₁₆("L") | 32₁₆("2")
U 2D₁₆("-") | 53₁₆("S") | 53₁₆("S") | 43₁₆("C")
V 53₁₆("S") | 2D₁₆("-") | 31₁₆("1") | 21₁₆(".")
W 31₁₆("1") | 00₁₆(null) | 00₁₆(null) | 00₁₆(null)
...
bus_info_block
root directory
HL2 directory
textual descriptor leaf

Figure A.9: HL2-Bus ROM format

The 8-bit ***bus_info_length*** field, the ***imc*** (isochronous resource manager capable) bit, ***cmc*** (cycle master capable) bit, ***isc*** (isochronous capable) bit, 8-bit ***cycle_clk_acc*** (cycle clock accuracy) field, 4-bit ***max_rec*** (maximum receive buffer size) field, and 2-bit ***max_ROM*** (maximum ROM access) fields are defined by wireless 1394. The ***bmc*** (bus manager capable) bit, ***pmc*** (power manager capable) bit, 4-bit ***generation*** field, and the 3-bit ***link_spd*** fields shall be zero. The *quadlet[A]*.***crc_length*** field shall have a value of 6 and the *quadlet[A]*.***crc*** field shall cover the following 6 quadlets.

A node that is CC capable (as defined in [4] shall be isochronous resource manager capable, and thus shall set the *imc* bit to 1. A node that is not CC capable (WT only) shall clear the *imc* bit to 0.

The 24-bit ***company_id*** identifies the vendor that administers the following ***chip_ID_hi*** and ***chip_ID_lo*** field values. The 8-bit ***chip_ID_hi*** and 32-bit ***chip_ID_lo*** values shall be concatenated to produce a 40-bit *chip_ID* value. The concatenation of the *company_id* and *chip_ID* values shall be the globally unique EUI-64 value for this node.

The initial root-directory entry is a quadlet in size and points to a Bus_Dependent_Info directory; the 2-bit *quadlet[G[*.***type*** field has a Directory=3 value; the 6-bit *quadlet[G]*.***key_id*** (key identifier) has a Bus_Dependent_Info=2 value, and the 24-bit *quadlet[G]*.***offset*** field specifies the number of quadlets between here and the start of the bus-directory directory.

The first quadlet of the bus-directory entry has 16-bit *quadlet[J]*.***length*** and *quadlet[J]*.***crc*** fields, whose meanings are defined by like named fields in the CSR Architecture. The 2-bit *quadlet[K]*.***type*** field has an Immediate=0 value; the 6-bit *quadlet[K]*.***key_id*** (key identifier) has a Specifier_ID=12_{16} value; the 2-bit *quadlet[L]*.***type*** field has an Immediate=0 value; the 6-bit *quadlet[K]*.***key_id*** has a Version=13_{16} value.

The concatenation of the 24-bit *quadlet[K]*.***specifier_ID*** and 24-bit *quadlet[L]*.***version*** fields forms the EUI-48 identifier that uniquely specifies this 1394 SSCS "bus" standard. The specifier_ID is an IEEE/RAC supplied *company_id* value; the version field is a value administered by the *company_id*-specified organ**iza**tion. For this technical specification, these shall have the values listed below:

company_id	xxxxxx_{16}
version	xxxxpq_{16}

The *p* and *q* values are hexadecimal digits representing the phase and the revision of the HL2 1394 SSCS TS. For this particular TS, *p* and *q* values shall both be 1.

NOTE: No fields are allocated to represent editorial changes.

The 2-bit *quadlet[M]*.***type*** field has a Leaf=2 value and the 6-bit *quadlet[M]*.***key_id*** field (key identifier) has a Textual_Descriptor=1 value. The 24-bit *quadlet[M]*.***offset*** field equals the number of following quadlets before the textual-descriptor leaf.

The leading quadlet of the textual-descriptor leaf provides *quadlet[P]*.***length*** and *quadlet[P]*.***crc*** fields, as specified by the CSR Architecture. The 8-bit *quadlet[Q]*.***descriptor_type***, 24-bit *quadlet[Q]*.***specifier_ID8***, 4-bit *quadlet[R]*.***width***, 12-bit *quadlet[R]*.***character_set***, and 16-bit *quadlet[R]*.***language_id*** fields shall be zero (this identifies the remaining text to be in minimal ASCII English). The following text shall specify the "1394-HL2-SSCS-1" string, which corresponds to the present document.

A.4 Locked messages

The HL2 Bus IRM interface differs from that defined by wired 1394, because isochronous bandwidth allocations depend on the *talker_ID* and *listener_ID* identifiers of the participating nodes. Since a different IRM interface is mandated, an efficient (as opposed to legacy-compatible) locked-message transport is provided.

A locked-message exchange has two phases: call and return, where the call and return phases correspond to the request and response subactions of a distinctive lock transaction. The request provides a command byte and multiple command-dependent arguments; the response returns a status byte and multiple status-dependent extended-status bytes, as described in the remainder of this subclause.

The locked-message transaction is distinguished by its address-offset and lock-subcommand values. The address offset shall be the start of the MESSAGE_REQUEST register (see A.1.1). The lock subcommand is the bus-dependent (value 7) value, as specified by the CSR architecture.

A.4.1 Locked message formats

The locked-message components include a 16-byte request and 8-byte response packet payloads, as illustrated in Figure A.10.

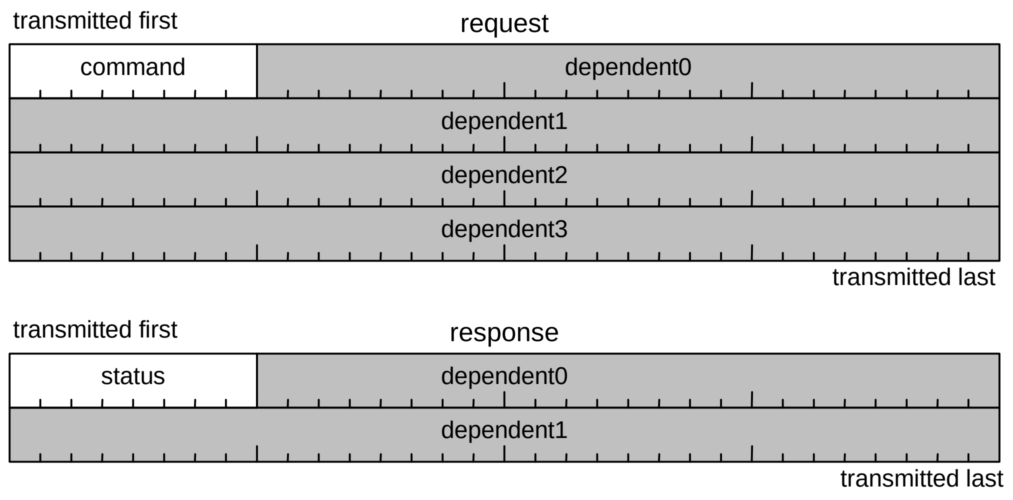

Figure A.10: Locked message formats

The 8-bit ***command*** values specify the locked-message operation to be performed, as specified in Table A.3). The definitions of status-field values is *command*-value dependent.

Table A.3: Locked-message *command* values

Value	Name	Subclause	Description
0	ALLOCATE_SOME	A.4.2	Available channel(s) and bandwidth request
1	MODIFY_BANDWIDTH	A.4.3	Modify isochronous bandwidth allocation
2	RECLAIM_THIS	A.4.4	Isochronous reallocation, channel and bandwidth
3	RELEASE_THIS	A.4.5	Isochronous release, channel and bandwidth
4-255	—		Reserved

A.4.2 ALLOCATE_SOME

An available channel and isochronous bandwidth are allocated for communication between physical_ID-addressed talker_ID and listener_ID nodes, using the parameters illustrated in Figure A.11.

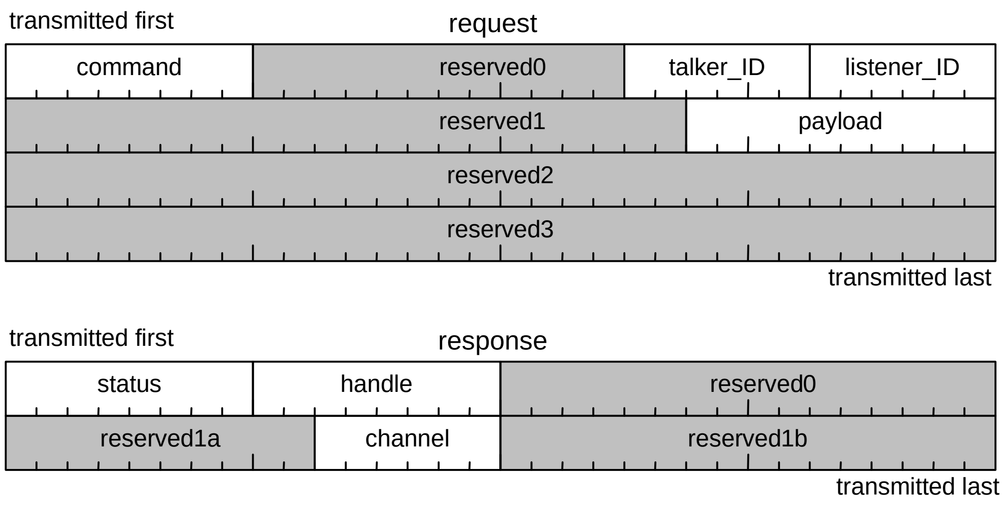

Figure A.11: ALLOCATE_SOME register format

A.4.2.1 ALLOCATE_SOME request

The 8-bit ***command*** distinctively identifies this command (see Table A.3).

The 6-bit ***talker_ID*** and 6-bit ***listener_ID*** values specify the physical_ID addresses of the local talker and listener nodes respectively.

The 10-bit ***payload*** value specifies the necessary isochronous bandwidth, as specified in A.4.2.1.

A.4.2.2 ALLOCATE_SOME response

The returned 8-bit ***status*** field indicates the success or failure of the command, as specified in Table A.4.

Table A.4: ALLOCATE_SOME response format

Value	Name	Description
0	DONE	Successful completion
1	CHAN	Channel allocation failure
2	RESET	Bus reset in process
3	BW	Isochronous bandwidth allocation failure
4	BOTH	Channel and bandwidth allocation failure
5-254	—	Reserved
255	BAD_COMMAND	Unsupported command

The 8-bit ***handle*** field contains the allocated multicast mac_ID, which is needed on following same-channel IRM commands.

The returned 6-bit ***channel*** value identifies the allocated isochronous channel number.

A.4.3 MODIFY_BANDWIDTH

The isochronous bandwidth associated with a specific channel is modified (increased or decreased), using the parameters illustrated in Figure A.12.

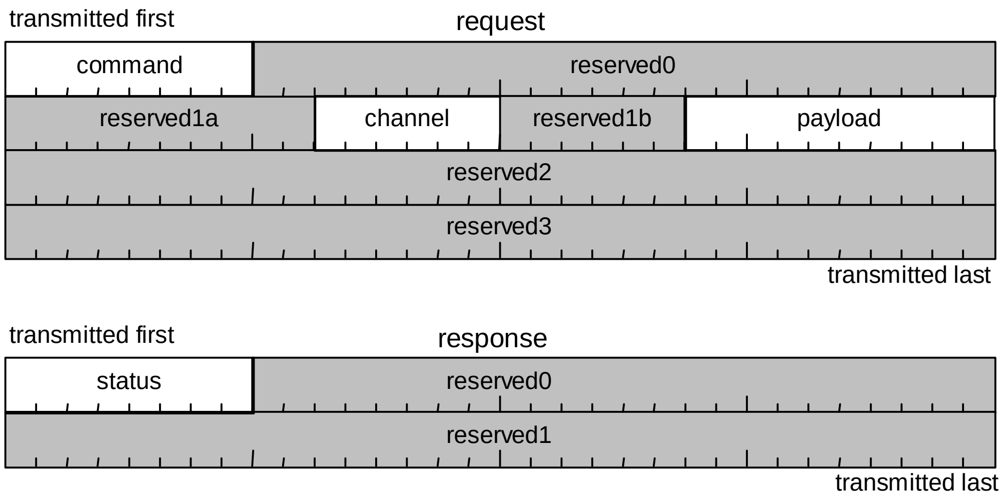

Figure A.12: MODIFY_BANDWIDTH register format

A.4.3.1 MODIFY_BANDWIDTH request

The 8-bit ***command*** distinctively identifies this command (see Table A.3).

The 6-bit ***channel*** value identifies the isochronous channel number for which bandwidth is modified.

The 10-bit ***payload*** value specifies the necessary isochronous bandwidth, as specified in A.2.1.

A.4.3.2 MODIFY_BANDWIDTH response

The returned 8-bit ***status*** field indicates the success or failure of the command, as specified in Table A.5.

Table A.5: MODIFY_BANDWIDTH response-*status* values

Value	**Name**	Description
0	DONE	Successful completion
1	CHAN	Channel failure
2	RESET	Bus reset in process
3	BW	Bandwidth modify failure
4-254	—	Reserved
255	BAD_COMMAND	Unsupported command

A.4.4 RECLAIM_THIS

A specific channel and isochronous bandwidth are reclaimed, using the parameters illustrated in Figure A.13.

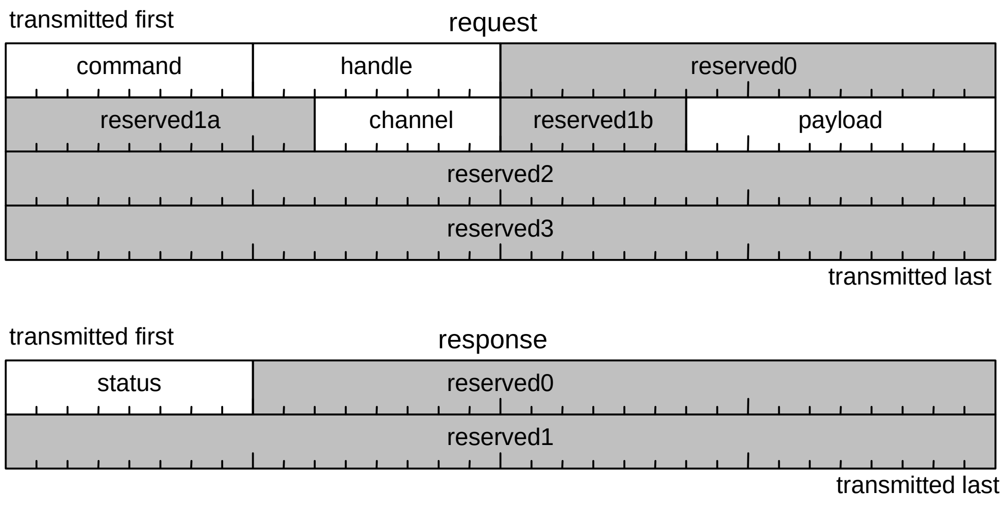

Figure A.13: RECLAIM_THIS register format

A.4.4.1 RECLAIM_THIS request

The 8-bit ***command*** distinctively identifies this command (see Table A.3).

The 8-bit ***handle*** value shall equal that returned by the initial ALLOCATE_SOME locked-message.

The 6-bit ***channel*** value specifies the requested isochronous channel number.

The 10-bit ***payload*** value specifies the necessary isochronous bandwidth, as specified in A.2.1.

A.4.4.2 RECLAIM_THIS response

The returned 8-bit ***status*** field indicates the success or failure of the command, as specified in Table A.6.

Table A.6: RECLAIM_THIS response-*status* values

Value	Name	Description
0	DONE	Successful completion
1	CHAN	Channel reclaim failure
2	RESET	Bus reset in process
3	BW	Bandwidth reclaim failure
4	BOTH	Channel and bandwidth reclaim failure
5-254	—	Reserved
255	BAD_COMMAND	Unsupported command

A.4.5 RELEASE_THIS

A specific channel and isochronous bandwidth are released, using the parameters illustrated in Figure A.14.

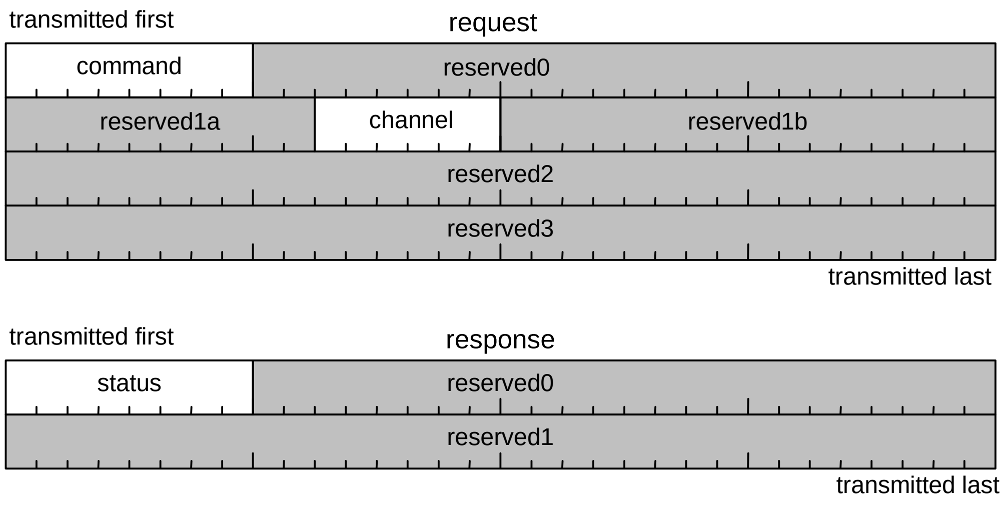

Figure A.14: RELEASE_THIS register format

A.4.5.1 RELEASE_THIS request

The 8-bit ***command*** distinctively identifies this command (see Table A.3).

The 6-bit ***channel*** value specifies the released isochronous channel number.

A.4.5.2 RELEASE_THIS response

The returned 8-bit ***status*** field indicates the success or failure of the command, as specified in Table A.7.

Table A.7: RELEASE_THIS response-*status* values

Value	**Name**	Description
0	DONE	Successful completion
1	CHAN	Channel release failure
2	RESET	Bus reset in process
3-254	—	Reserved
255	BAD_COMMAND	Unsupported command

Annex B (informative): Quadlet-structured PDU formats

Within HL2 standards, the PDU formats are normally illustrated in byte-wide Figures. For the convenience of wired-1394 readers, this Annex provides quadlet-wide equivalents of these Figures.

Fields of quadlet-wide PDUs are the same as those of the byte wide PDUs. See 5.4.2, 5.5.1.2 and 5.5.2.2 for fields description.

B.1 Asynchronous subaction

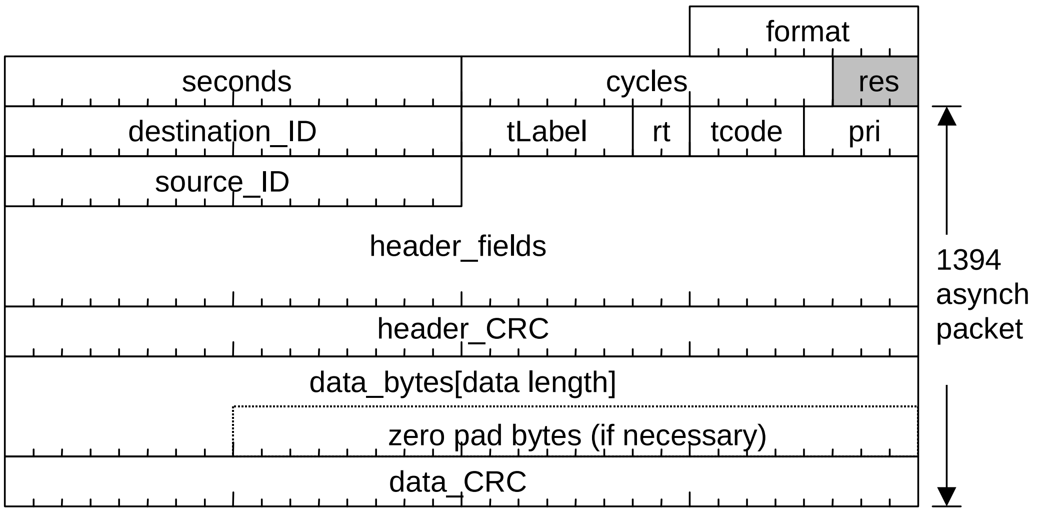

Figure B.1: Asynchronous subaction PDU format

B.2 Isochronous streams

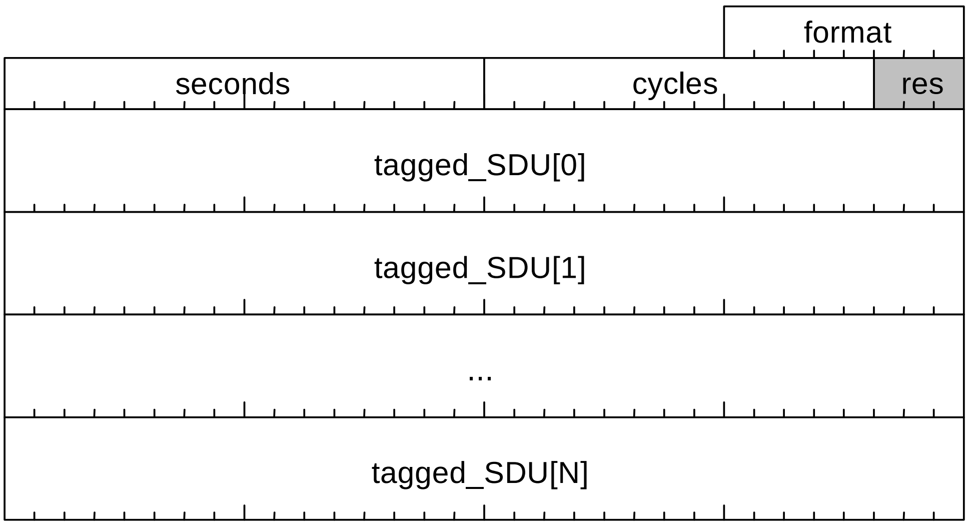

Figure B.2: Isochronous stream PDU format

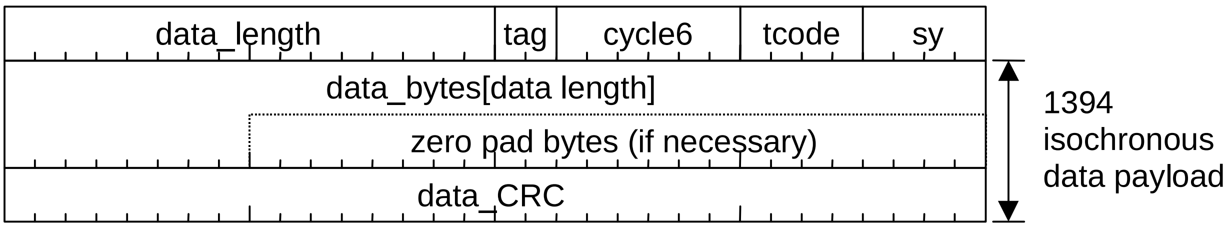

Figure B.3: Isochronous tagged SDU

B.3 Asynchronous streams

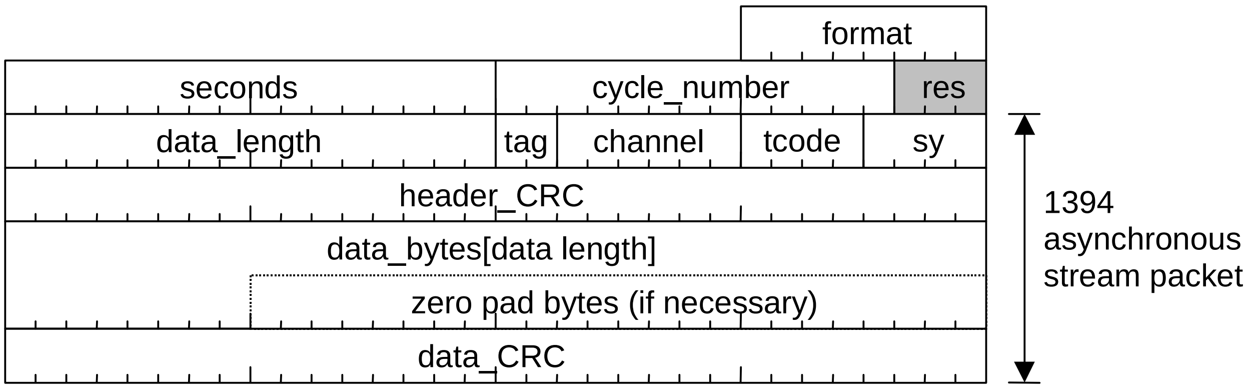

Figure B.4: Asynchronous stream PDU format

Annex C (normative): Nonbridge upper layer

A wireless node that does not gather bridge functions may implement the 1394 SSCS only. In that case, the upper layer that is not a HL2 1394 bridge layer entity shall react as described in this Annex.

This Annex contains informative as well as normative parts.

C.1 Primitives (informative)

The upper layer uses the primitives defined in 5.2.1 and 6.2. They are adapted to the 1394 SSCS, and are slightly different from those provided by the 1394 link layer.

C.2 Time stamps for split-response timeouts (normative)

Split-responses buses, like wired 1394 or HL2 Bus, detect transmission failures through split-response timeouts. On the HL2 Bus, time stamps are used to limit the request-transmission time, the responder transaction processing time, and the response-transmission time. The requester normally abandons the transaction after the sum of these worst-case times, but avoids reuse of the transaction's *tlabel* identifier until an even later time, to cope with possible inaccuracies in transmission-time limits.

The timeout requirements for wired 1394 and HL2 Bus are similar, but differ in where timing checks are performed, as illustrated in Figure C.1. Although implementation details differ, both buses provide for a SPLIT_TIMEOUT responder-processing delay and a 2×SPLIT_TIMEOUT timeout recovery delay in the requester.

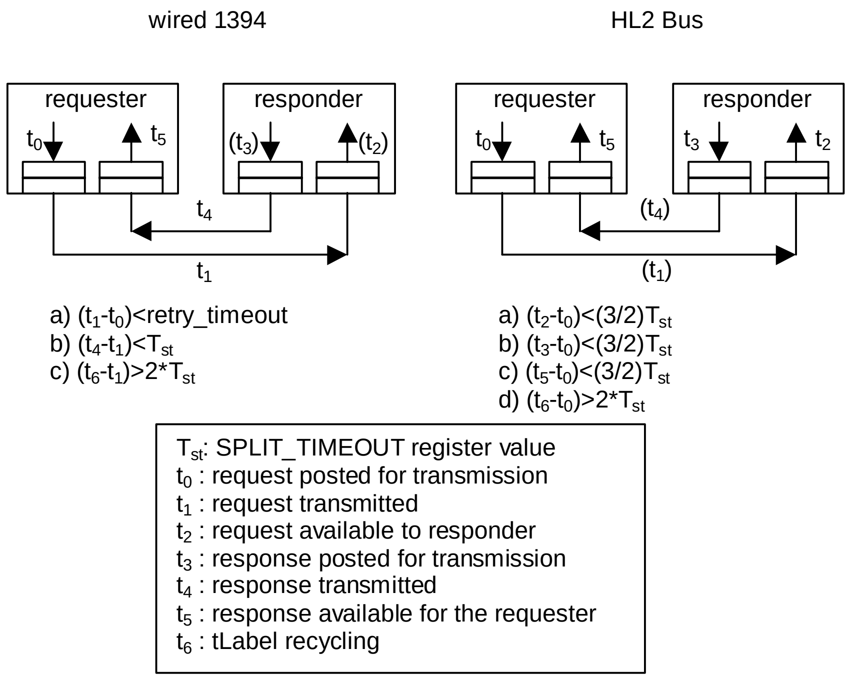
wired 1394
HL2 Bus
requester
responder
t_0
t_5
(t_3)
(t_2)
t_4
t_1
requester
responder
t_0
t_5
t_3
t_2
(t_4)
(t_1)
a) (t_1-t_0)<retry_timeout
b) $(t_4-t_1)<T_{st}$
c) $(t_6-t_1)>2*T_{st}$
a) $(t_2-t_0)<(3/2)T_{st}$
b) $(t_3-t_0)<(3/2)T_{st}$
c) $(t_5-t_0)<(3/2)T_{st}$
d) $(t_6-t_0)>2*T_{st}$
T_{st}: SPLIT_TIMEOUT register value
t_0 : request posted for transmission
t_1 : request transmitted
t_2 : request available to responder
t_3 : response posted for transmission
t_4 : response transmitted
t_5 : response available for the requester
t_6 : tLabel recycling

Figure C.1: Split-response timeout times

A wired-1394 request-transmission begins at time t_0 and a busy-retry timeout limits the arbitration delay (t_1- t_0) to a software-specifiable value. The t_1 transmission time of a request and the t_4 transmission time of the response are visible to the requester and responder. The responder limits its processing time to ensure that (t_4- t_1)<T_{st} and the requester terminates the transaction if (t_4- t_1)> T_{st}. To account for timing differences and, the requester avoids reusing the transaction *tlabel* identifier until time t_6, where (t_6- t_1)>2×T_{st}.

HL2 Bus transmission times are not visible to the 1394 SSCS entity, so timeouts are defined in a different (although similar) fashion. A HL2 Bus request-transmission begins and ends at times t_0 and t_2 respectively, and a delay of $½T_{st}$ is allocated for this request-packet transmission.

The responder is expected to limit its processing time to ensure that $(t_3- t_2)<½T_{st}$. A response-transmission begins and ends at times t_3 and t_5 respectively, and a delay of $½T_{st}$ is allocated for this response-packet transmission.

The requester terminates the transaction when $(t_5- t_0)>(3/2)T_{st}$ but (to account for timing differences and irregularities) avoids reuse of a timed-out transaction's *tlabel* identifier until time t_6, where $(t_6- t_0)>2T_{st}$.

Time stamps on HL2 Bus packets effectively limit the lifetime of request and response packets. These time-stamp values are unknown to the lower-layer protocols, so actual lifetimes can exceed their allocated limits. The receivers are therefore responsible for checking received time-stamp values, so that stale packets can be discarded.

To realize the desired behaviours, the processing of packet time-stamp values (see 5.2.3.2.1 and 5.4.4) is specified as follows:

a) Requester inclusion: the requester shall prepend its request packet with a time_of_life=$(3/2)T_{st}$.

b) Responder rejection: when the request is received at time t_2, a responder shall behave as follows:

1) The request may be rejected if time_of_life<T_{st} (because the request transmission time has exceeded $½T_{st}$)

2) The request should be rejected if time_of_life<$½T_{st}$ (to allow for a response transmission time of $½T_{st}$)

c) Responder discard: the responder shall discard an in-progress request or response, if time_of_life≤0.

d) Responder processing: when generating a response, the response time_of_life shall be set to the request time_of_life minus the response processing time $(t_3- t_2)$.

NOTE 1: This time_of_life calculation results in equal time_of_death values in both request and response asynchronous subaction SSCS PDUs (see 5.4.2).

e) Requester discard: when received at time t_5, the requester shall discard a response if time_of_life≤0.

f) Requester recycle: a requester may release a *tlabel* when the first of the following occurs:

1) Completion: matching response is received

2) Termination: $(t_6-t_0)>2T_{st}$

NOTE 2: Some of the data can still be in the lower layer when the current bus time exceeds the PDU specified time_of_death. Then the discarding facilities of the DLC (see [3]) may be used to avoid transmitting useless data. The details of this mechanism are implementation dependent and outside the scope of the present document.

C.3 User plane

C.3.1 Isochronous stream transport service

C.3.1.1 HL2 Bus maximum transmission delay (informative)

The receiver of isochronous streams shall be prepared for these streams to experience a maximum transmission delay of 6,1 ms.

The example on Figure C.2 shows how consecutive channel traffic on wired 1394 are concatenated over a duration of approximately 2 ms. The MAC frame top signals are generated by the appropriate primitive.

It is assumed that the 1394 SSCS collects all data of each cycle until it receives this frame top. But the wired 1394 8 kHz cycle clock is not synchronized on the 500 Hz HL2 Bus clock. Thus the concatenation can reach 17 cycles of 125 µs.

In Figure C.2, a fixed number of slots is reserved in each frame starting at position #Q (fixed slot allocation).

The 16 (15 or 17) wired 1394 cycle data acquired are dealt by the 1394 SSCS, CPCS and SAR and provide N packets of 48 bytes. These bytes are encapsulated by the DLC and transmitted in the slots reserved on the air interface.

The maximum transmission jitter is 6,1 ms, coming from:

- 2,1 ms for the worst case acquisition period over one MAC frame: in the normal case it is 2 ms, but due to the 1394 clock / HL2 clock difference, a 1394 SSCS PDU may sometimes contain 17 SDUs (see 5.5.1.2), and thus the acquisition period may contain an additional 1394 cycle (0,1 ms);
- 2 ms because the granted slot may have just occurred when the corresponding data come to the DLC, so that it shall wait for the next frame (worst case 2 ms);
- 2 ms because the scheduler may have rescheduled the frame at that point, so that the slot is now at the other side of the frame (worst case 2 ms).

NOTE: As already mentioned, a 1394 SSCS entity may generate 16-SDU PDUs in average, and sometimes a 17-SDU PDU. As mentioned in A.2.1, a secure margin shall be reserved by 1394 controller to take the 1394/HL2 clock difference. Nevertheless, when a 17-SDU PDU is transmitted, the last SDU of the 17-SDU PDU (or at least part of it) may have to wait for the next frame (because the other 16 SDUs have already used all granted slots). The 17th SDU may thus experience a 2 ms more transmission delay. But, since it did not experience any acquisition delay (the PDU has been completed with it), no specific additional delay is taken into account here.

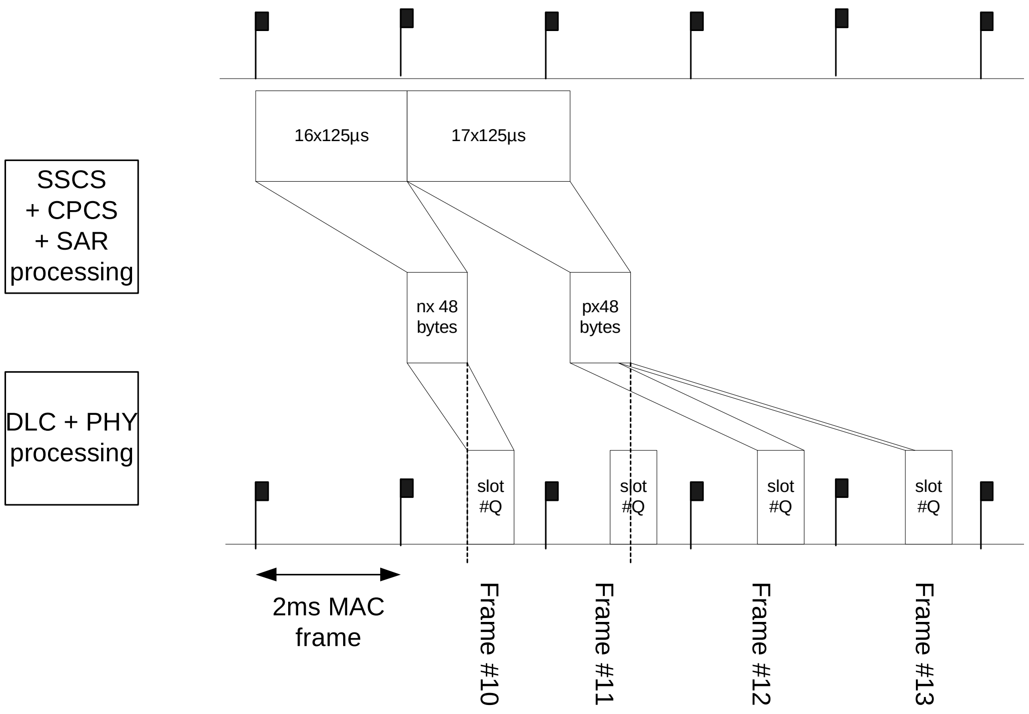

Figure C.2: maximum transmission delay

C.3.1.2 Relative CIP time stamps (normative)

On wired 1394, CIP headers transport an absolute time stamp. The time stamp location in the CIP and meaning is as described in IEC 61883 [5] . Depending on the CIP data format, the time stamp may be located in the header or the following data, as illustrated in Figure C.3 and Figure C.4. These Figures illustrate tagged SDUs as specified in 5.5.1.2. On the HL2 Bus, the format of these isochronous payloads is the same, but a relative time stamp shall be provided.

A 4-bit *cCount* time-stamp field can be located in the CIP header, as illustrated in Figure C.3.

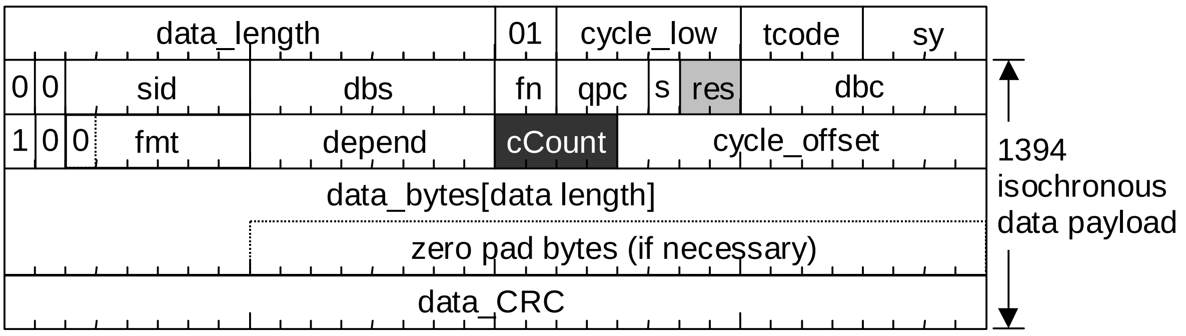

Figure C.3: CIP header time stamp

The value of cCount in an HL2 Bus packet (hl2Bus.cCount) can be readily derived from the value defined by IEC 61883 [5] for the CIP header (cipStd.cCount), and shall be as specified below:

hl2Bus.cCount= (cipStd.cCount-time_stamp.cycles)&0xF;
where:
time_stamp is a parameter of the CL_ISOCH_STREAM primitive, which specifies the isochronous cycle when the isochronous packet is presented to the 1394 SSCS.

Alternatively, a 13-bit *cycle_count* time-stamp field can be located at header-dependent positions within transmitted data blocks, as illustrated in Figure C.4.

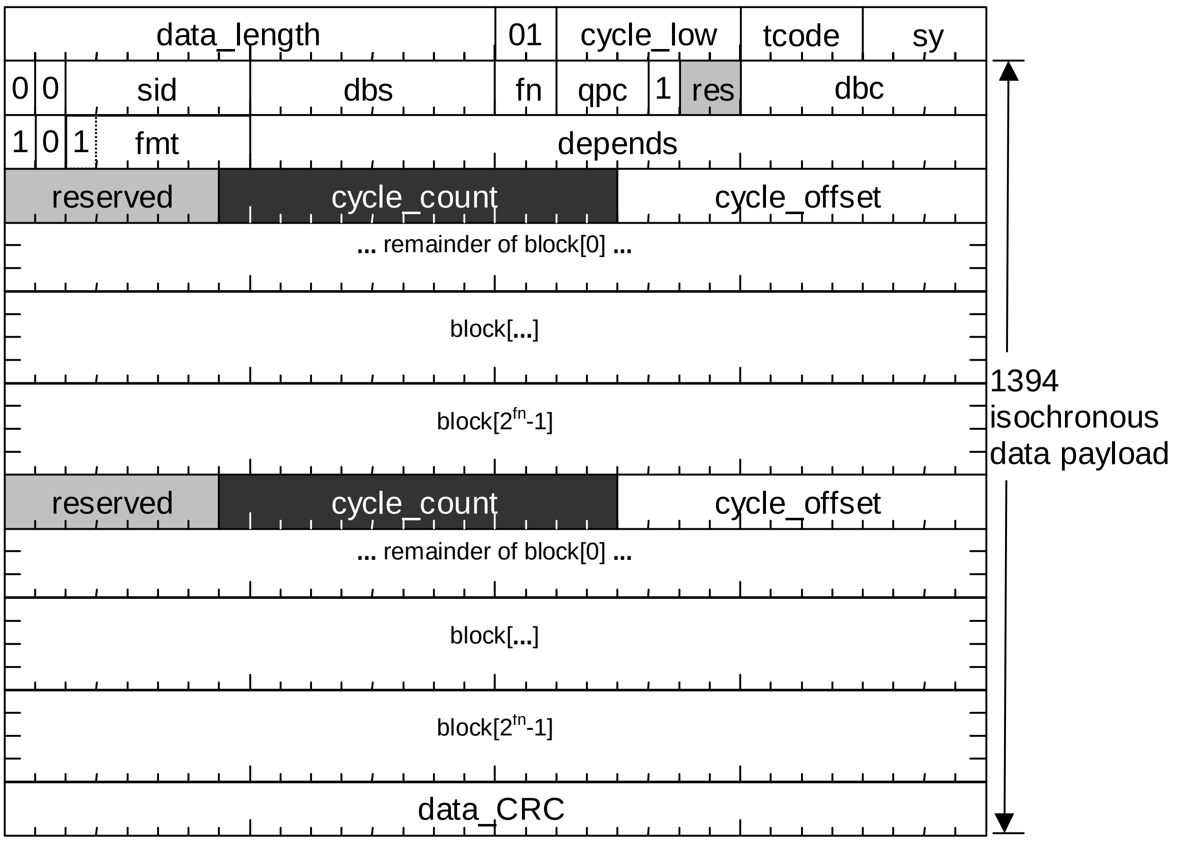

Figure C.4: Source packet header time stamps

The value of cycle_count in an HL2 Bus packet (*hl2Bus.cycle_count*) can be readily derived from the value defined by IEC 61883 [5] for the CIP header (*cipStd.cycle_count*), and shall be as specified below:

hl2Bus.cycle_count= (8000+cipStd.cycle_count-time_stamp.cycles)%8000;
where:
time_stamp is a parameter of the CL_ISOCH_STREAM primitive, which specifies the isochronous cycle when the isochronous packet is presented to the 1394 SSCS.

C.3.2 Asynchronous transactions

C.3.2.1 Subaction rt field (normative)

If the packet originated on wired 1394 and passed through a bridge, the *rt* field and the *header_CRC* values shall be the values that were observed on the bus when the packet was received. Except for its inclusion in the *header_CRC* calculation, the value of the *rt* field shall be ignored. If the request packet passes through wireless 1394 and returns to a wired 1394 environment, the wired-1394 specified *rt* field value shall be used. Since the transmitted *rt* value may change, wired-1394 transmitters are required to recompute the *header_CRC* value while the packet is being transmitted.

If the packet is originated from a HL2 Bus node, the *rt* field shall be zero.

All asynchronous packets are prepended with the same seconds/cycles quadlet and transmitted as received on 1394. For the sake of brevity, these other packet formats are not illustrated.

C.3.2.2 Available self-ID information (informative)

The HL2 Bus self-ID information includes a 2-bit *node_type* field, a 6-bit *physical_ID* field, and an 8-bit *mac_ID* field for each of the attached nodes. The IRM information block is always the first of these, allowing the IRM node to be readily identified. See 7.4.7 for details.

C.4 Control plane (normative)

C.4.1 Clock connection control

The upper layer shall not generate a CL_CONTROL request primitive that enables / disables wireless cycle master, as defined in 6.5.

C.4.2 Isochronous stream connection control

The upper layer shall use the mechanism for isochronous data flow management, as defined in 6.9, i.e. the connection management procedures, described in IEC 61883 [5], clause 8 of Part 1, with the following adaptations:

- IRM register access is as described in 6.9 (based on command based lock messages).
- PCR lock transactions response can sometimes report error codes as specified in A.2.3.
- Listener and talker nodes can autonomously disconnect themselves from an isochronous stream, update their PCRs, signal the 1394 controllers by writing to the INTERRUPT_TARGET register (see 6.9.2.2, 6.9.2.3 and A.1.3).

Bibliography

The following material, though not specifically referenced in the body of the present document (or not publicly available), gives supporting information.

- ETSI TS 101 493-4: "Broadband Radio Access Networks (BRAN); HIPERLAN Type 2; Packet based Convergence Layer; Part 4: IEEE 1394 bridge layer".

History

Document history		
V1.1.1	September 2000	Publication