Proposal to 1394 Trade Association Architectural WG and IEEE P1394a WG

Request and response error handling

Changes to IEEE1394-1995 standard

Draft v 1.0

Drafted by Philips

Contact:
Calto Wong (cxw@philabs.research.philips.com) or
Rudi Bloks (bloks@natlab.research.philips.com)

Date: March 13, 1997

1 Introduction

The IEEE 1394-1995 standard provides some guidelines (7.3.1.4) about how transaction errors should be handled. In the cases of channel and bandwidth allocation, the standard specifies procedures (8.4.3.1 and 8.4.3.2) which can lead to over- or under-allocation in some situations. Section 2 of this document discusses:

- typical error events in 1394-1995 split lock transactions;
- options for error handling;
- a proposal for a basic error recovery procedure which makes use of bus resets for the BANDWIDTH_AVAILABLE, CHANNELS_AVAILABLE registers and PCR's;
- a proposal for an extended error recovery procedure which makes use of an optional Error Status Register (ESR) for each of the above registers requiring protection. In case the ESR is not implemented by a node, the basic error recovery procedure can be attempted (bus reset).

Besides transaction request and response error handling, this document also proposes a number of changes to the current IEEE 1394-1995 standard which can be included in the P1394a draft. Section 3 of this document discusses our proposed changes to:

- determination of bus manager
- transaction data indication
- command reset effect on isochronous resource registers
- serial bus control confirmation
- gap-count value for 16 hops and 4.5 meter cables per hop
- reallocation time limit for isochronous resources after bus reset
- local self-ID packets to be communicated to link layer

2 Transaction request and response error handling

2.1 Overview of typical error events in 1394 split lock transaction

The major reason for coming up with a sound error handling strategy is because of the ill effects of request and response errors on resource allocation lock transactions. These lock transactions are invariably split or concatenated transactions. For simplicity, the case of split lock transactions will be used for analysis purpose. Figure 1 shows the events of a typical 1394 split lock transaction.

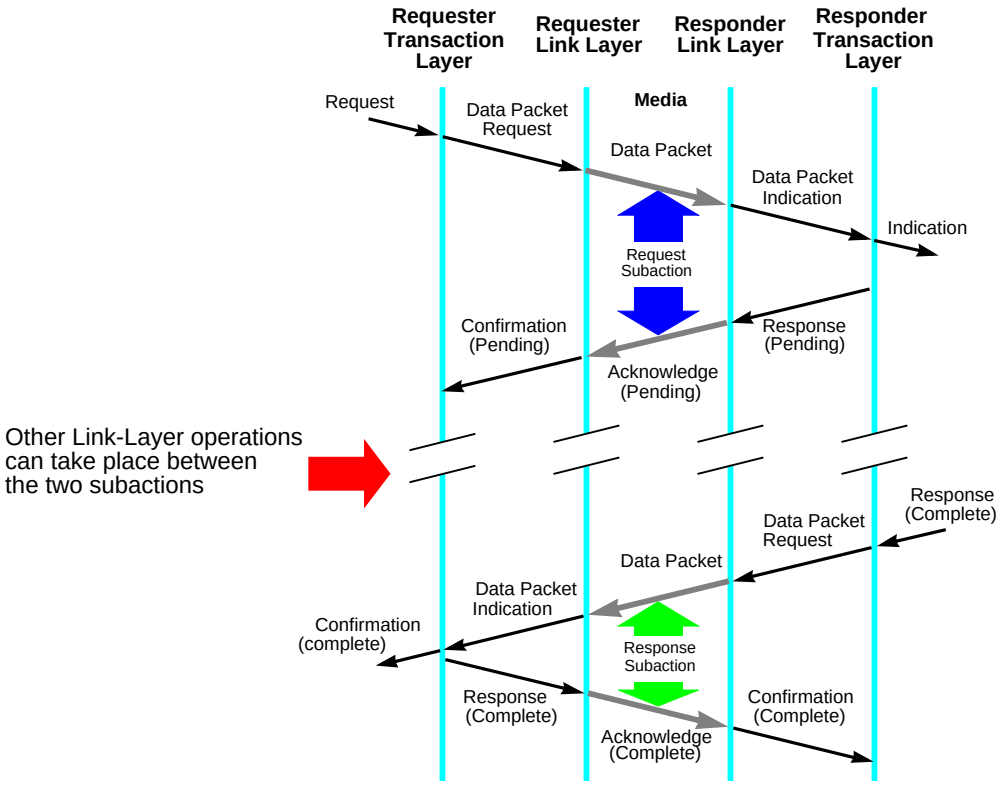


Figure 1:Events of a typical 1394 split lock transaction

Although the actual error probability of a typical IEEE 1394-1995 implementation is not known at this moment, there are all signs that an error event occurs only very infrequently (i.e. the 1394 bus is fairly reliable). Under this assumption, we can simplify the analysis by considering only isolated errors and assume that at most one error event will occur in the entire transaction. Table 1 lists these error events and the impact on the requester and responder.

Based on the 1394-1995 transaction layer service model, at the requester side, the Request Status and Response Code form part of the transaction data confirmation passed back to the requesting application, and can be used for error detection. The transaction event indication does not go to the requesting application, but only to the node controller. At the responder side, the transaction event indication goes only to the node controller, and the responding application does not get notified of any error in its response.

Error Source	Requester Request Status / Response Code Transaction Event Indication	Responder Response / Acknowledge Transaction Event Indication
E.1. Request packet data	COMPLETE / resp_data_error -	N / - -
E.2. Request packet header	ACKNOWLEDGE MISSING / - -	N / - -
E.3. Too many retrials for request packet	RETRY LIMIT / - -	N / - -
E.4. Response packet acknowledge	COMPLETE / resp_*[a] *	Y / *[b] ACKNOWLEDGE MISSING
E.5. Response packet data	DATA ERROR / resp_complete RESPONSE DATA ERROR	Y / ack_data_error RESPONSE DATA ERROR
E.6. Response packet header	TIMEOUT / - -	Y / *[c] ACKNOWLEDGE MISSING
E.7. Too many retrials for response packet	TIMEOUT / - -	Y / ack_busy_*[d] RESPONSE RETRY LIMIT
E.8. Request packet acknowledge	ACKNOWLEDGE MISSING / - UNSOLICITED RESPONSE	Y / ack_complete -
E.9. Response too late	TIMEOUT / - UNSOLICITED RESPONSE	Y / ack_complete -

Table 1: Typical error events in a 1394-1995 split lock transaction

a.The response code can be any allowed value reflecting the status of the response.
b.Request Status of link data confirmation is ACKNOWLEDGE MISSING
c.Request Status of link data confirmation is ACKNOWLEDGE MISSING
d.Acknowledge code is ack_busy_X, ack_busy_A or ack_busy_B

The requester can detect transaction errors by looking at the Request Status and Response Code. In case the Request Status is COMPLETE (E.1, E.4), the response is available if the Response Code is resp_complete. If the Response Code is any other value, the transaction can simply be retried, assuming that no side effects will be generated under such circumstances[1]. A simple retry is also applicable to the case when the Request Status is RETRY LIMIT (E.3).

If the Request Status is any other value (E.2, E.5, E.6, E.7, E.8, E.9), the responder may already have committed a response. In case the response has no side effects, the request can simply be retried. In

1. This needs to be confirmed.

general, the error recovery mechanism is application specific, since only the application is aware of the consequence of the error and the side effects. The need for error recovery is also application dependent. In the extreme case, the lack of error recovery can lead to application or even bus malfunctions. Two examples are given below.

2.1.1 Problem Example 1: Bandwidth (De)allocation

Node A (a camcorder) performs bandwidth (de)allocation by sending a compare-swap lock transaction request to the BANDWIDTH_AVAILABLE register of the IRM (Isochronous Resource Manager). The IRM modifies the BANDWIDTH_AVAILABLE register and returns a response packet to Node A.

Suppose the response packet gets corrupted. Node A will receive an erroneous response packet with data or header CRC error and conclude the transaction has failed. However, Node A cannot determine if the bandwidth (de)allocation has been performed. Re-reading BANDWIDTH_AVAILABLE register at IRM does not help, since other bandwidth (de)allocation transactions may have been processed.

Under this situation, the IEEE1394-1995 standard recommends Node A to retry the bandwidth (de)allocation (8.4.3.1). Retrying bandwidth allocation will mean that a portion of the isochronous bandwidth will no longer be available for isochronous data transfer. Retrying bandwidth deallocation can give the illusion that the bus has more isochronous bandwidth than available, leading to bandwidth overallocation and bus failure. The former causes isochronous communication to be unnecessarily inhibited. The latter can be detected by a manager or perhaps cycle master (if it monitors isochronous activity). Both situations are resolved after the next bus reset.

2.1.2 Problem Example 2: TV Channel Switching

Node A (a control panel) sends a write transaction to Node B (a TV) to switch to the next channel. Node B switches the channel, and returns a response packet to Node A. Suppose the response packet gets corrupted. Node A will receive the corrupted response. Since Node A cannot determine if the channel-up command has been executed, it may assume the transaction has failed, and send another channel-up command to retry the transaction. As a result, Node B will switch channel twice.

The same can occur if a response packet is repeatedly busy-ack-ed by the requester. After the retry-limit has been exceeded the transaction is over (according to IEEE1394-1995). This is probably a much more likely situation than corrupted packets.

2.2 Error handling strategy and options

The error handling procedure of global resources should be standardized. Global resources (bandwidth, channel numbers) should be protected against errors which can lead to bus malfunction. These include the BANDWIDTH_AVAILABLE and CHANNELS_AVAILABLE registers at the Isochronous Resource Manager, but also device-specific registers like the Plug Control Registers whose contents indicate how global resources may be consumed or released. The error handling procedure of application-specific resources is application dependent and remains within the scope of the application. However, it is desirable to have a common mechanism for passing error status.

2.2.1 Error handling options for resource allocation

As a first option, the requester could perform a bus reset whenever it cannot determine the status of the response, i.e. if the Request Status is neither COMPLETE (E.1, E.4) nor RETRY LIMIT (E.3). This trivial solution will restart all resource allocation. If the bus has a low error probability, this solution is attractive since it is compatible with current implementations. However, if the error probability becomes higher, a lot of bus resets may be generated, which can become a problem by itself.

It should be pointed out that this first option can only be applied for registers that revert back to some initial value upon bus reset, or whose side effects are undone at bus reset. For those registers that are not affected by bus resets, one of the following options can be used.

As a second option, the responder could re-send or undo a response when an error is detected, to enable the requester to directly associate transaction status with response status. This option requires an incompatible silicon implementation of link/transaction layer, and does not tackle all errors.

A third option is to specify that for these CSRs the responder must first successfully send a response packet before it actually executes the transaction (and invokes the side effect/CSR modification).

A fourth option is to make use of an Error Status Register (ESR), which is described in the next section.

A fifth option could be to introduce a complete error handling protocol.

2.3 Error handling using ESR (Error Status Register)

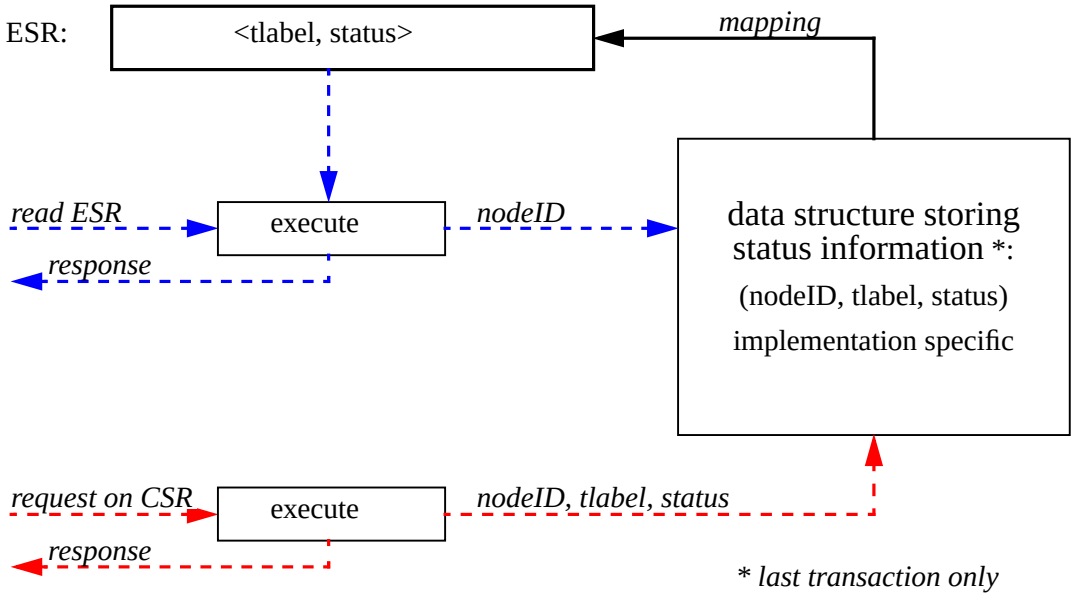
The idea behind the use of an Error Status Register (ESR) is to give the requester a chance to re-read the transaction status if an error during the transaction prevented it from acquiring the transaction status.

Consider node A making a transaction request to a CSR X at node B. To protect X against errors which prevent A from deducing the response status in the transaction data confirmation:

- B will store the response status of transactions which were received and executed from any external node A for later retrieval by that node.
- B will implement an Error Status Register X_ESR in its CSR space.
- B will provide any requesting node A with the status of (one of) its previously requested transaction on X when A performs a read request on X_ESR.
- The status of a node A can be cleared when A performs a write request (regardless of data written) on X_ESR. A node can only clear status information of its own transaction(s).

The ESR lends itself to a number of implementation options. In what follows two implementation examples are given. The implementation format should be standardized (not left to implementor).

2.3.1 ESR Example 1



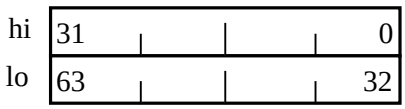
In this example, a responder will maintain a data structure which stores the transaction status information of the last transaction for each node requiring protection. For correct association with a particular transaction, the status information contains details about the requester nodeID and transaction label (tlabel) in addition to the actual transaction status.

After a request has been processed, the responder will update its data structure with the new status information. This can be done by using the requester nodeID field as the index to update the old (nodeID, tlabel, status) information of that node with the new status information.

When a requester wishes to examine the ESR, the requester nodeID will be used as an index to retrieve the status information of the previous transaction of that node. The extended (nodeID, tlabel, status) information of the ESR will allow the requester to determine if the previous request is executed or not, together with the transaction status.

2.3.2 ESR Example 2 (simplified implementation)

2 quadlet ESR:



ESR = bitmap for 63 nodes (only local bus error protection is supported)
Bit n indicates if last executed transaction for node n resulted in side effect / modification
Since no tlabel is stored, the bit must be reset to a safe value before attempting unsafe transactions.

Bit n is cleared when:

- transaction from node n is executed which did not result in side effect or CSR modification (e.g. READ transaction on standard CSR).
- WRITE to ESR is executed by node n (regardless of data).

A simplified implementation is to use a 2 quadlet ESR. This ESR only supports error protection of transactions arising from nodes in the local bus, and is particularly suited for the protection of resource allocation (e.g. BANDWIDTH_AVAILABLE) CSRs. It does so by providing a error status bitmap of all 63 nodes in the local bus.

A bit is set when the last executed transaction on the protected CSR from the corresponding node resulted in side effect or modification, such as in the case of a compare-swap lock transaction. Since no tlabel information is stored, the bit needs to be cleared before the requester performs such a transaction. The requester can then use this bit to determine if a transaction resulted in the expected side effect or modification.

For compare-swap lock transaction, a very convenient way to clear the ESR bit will be whenever the protected CSR is read. This is so because in resource allocation, the CSR will typically be read before a compare-swap lock transaction is performed.

2.4 Proposal: basic request and response error handling

This proposal uses bus resets to ratify errors which occur when a compare-swap lock transaction is made to the BANDWIDTH_AVAILABLE, CHANNELS_AVAILABLE, INPUT_MASTER_PLUG, INPUT_PLUG_CONTROL, OUTPUT_MASTER_PLUG or OUTPUT_PLUG_CONTROL registers. This simple solution is compatible with current silicon implementation and should work well in case the bus has a low error probability. If this is not the case, a more sophisticated error handling proposal based on ESR can be found in section 2.5.

2.4.1 Compare-swap lock transactions to BANDWIDTH_AVAILABLE register or CHANNELS_AVAILABLE register of the isochronous resource manager

When an application issues a compare-swap lock transaction to perform global resource allocation via the BANDWIDTH_AVAILABLE or CHANNELS_AVAILABLE register of the isochronous resource

manager and the transaction does not return with the Request Status of COMPLETE and Response Code of resp_complete, the application issuing such a transaction:

- shall issue a bus reset if the Request Status is neither RETRY LIMIT nor COMPLETE.
- may retry the transaction if the Request Status is COMPLETE[2] or RETRY LIMIT.

2.4.2 Compare-swap lock transactions to Plug Control Registers

When an application issues a compare-swap lock transaction to perform connection management via any of the INPUT_MASTER_PLUG, INPUT_PLUG_CONTROL, OUTPUT_MASTER_PLUG or OUTPUT_PLUG_CONTROL registers supported by the target node and the transaction does not return with the Request Status of COMPLETE and Response Code of resp_complete, the application issuing such a transaction:

- shall issue a bus reset if the Request Status is neither RETRY LIMIT nor COMPLETE.
- may retry the transaction if the Request Status is COMPLETE[3] or RETRY LIMIT.

2.4.3 All other transactions

When an application issues other types of transactions and a transaction does not return successfully to the judgement of the application, the error recovery procedure is application specific.

- If the response does not have any side effects, the application may retry the request to the extent allowed by the application requirements (e.g. timing).
- If the response has side effects, the error recovery mechanism is entirely application specific and beyond the scope of this paper.

2.5 Proposal: request and response error handling with ESR

This proposal applies to bandwidth and channel number resources maintained by the BANDWIDTH_AVAILABLE and CHANNELS_AVAILABLE registers at the IRM (Isochronous Resource Manager), together with device-specific Plug Control Registers whose contents indicate how global resources are consumed or released.

As an illustration, consider the BANDWIDTH_AVAILABLE register at IRM. For its protection, IRM can *optionally* implement an ESR (Error Status Register) BANDWIDTH_AVAILABLE_ESR, to be recognized by e.g. an entry in the CONFIGURATION ROM.

When an application wishes to perform bandwidth allocation, it issues a compare-swap lock transaction to the BANDWIDTH_AVAILABLE register at IRM. If the transaction returns with the Request Status of COMPLETE and Response Code of resp_complete, further actions will be according to section 8.4.3.1 of the IEEE 1394-1995 standard. Otherwise, if the Request Status is COMPLETE or RETRY LIMIT, bandwidth allocation has failed and the application may retry the transaction. Otherwise, if the application determines that BANDWIDTH_AVAILABLE_ESR is not implemented at IRM, it shall issue a bus reset. Otherwise, it shall issue a read transaction to BANDWIDTH_AVAILABLE_ESR at

2. Assuming the register will not be modified if the Response Code is not resp_complete.
3. As specified by section 7.9 of “Specifications of Digital Interface for Consumer Electronic Audio/Video Equipment Part 1”, the plug control registers shall store any value if and only if the compare/swap transaction returns resp_complete.

IRM until the Request Status is COMPLETE and Response Code is resp_complete. If ESR Status is ESR_COMPLETE[4], bandwidth allocation succeeds. Otherwise, bandwidth allocation has failed and the application may retry the transaction.

BANDWIDTH_AVAILABLE_ESR can be implemented as a 2-quadlet ESR containing the bitmap for 63 nodes. Bit n is set if the compare-swap lock transaction from node n modifies the BANDWIDTH_AVAILABLE CSR. Bit n is reset when node n reads the BANDWIDTH_AVAILABLE CSR before the lock transaction, or if the compare-swap lock does not modify the CSR e.g. because the argument value does not match the current value.

This simple scheme has one limitation. To ensure matching of the CSR status with the ESR, any node may issue a request to the CSR only after the result of any previous requests on the CSR has come to a conclusion (including reading ESR if necessary). This is especially problematic for devices running multiple applications on top of a single 1394 stack.

2.5.1 Error handling for application-specific resources

In general, the error handling procedure when the requesting application fails to obtain the response status of a target CSR is application-specific. If the request on the target CSR does not carry side effects, it can simply be retried to the extent allowed by the requesting application (e.g. timing). If the request on the target CSR carries side effects (e.g. interrupt), the use of an ESR (Error Status Register) is recommended to protect the target CSR. It provides a mechanism for the requesting application to determine the status of the response and its side effect. This can simplify the error handling procedures considerably.

The number of ESRs is the same as the number of CSRs requiring protection. This may be unacceptable if many CSRs need protection. It might be more elegant to define an error recovery protocol, based on a single universal ESR by means of which status for any other protected CSR can be obtained. The complicating factor for such a mechanism is that a simple read transaction of the ESR is not sufficient to get the desired status. To carry all the required information (CSR address for which to get status, and perhaps the tlabel of the transaction) a write request for quadlet or block is needed (requiring a write response which cannot carry useful information). The responder can then send back a write request with the requested information (again requiring a response packet by the original requester).

Although it has more overhead, it is more universal and technically superior and requires only one or two locations for the universal ESR (s).

Note: Since error recovery is optional, there should be an unambiguous way of finding out if a node implements any of the options presented here, and for which CSRs (only some may need protection). This could be part of a complete protocol, or it could be specified in a configuration ROM entry. Further study and feedback from others is needed to find the best (most acceptable) solution.

4. ESR_COMPLETE means that modification of the protected CSR has taken place.

3 Proposed changes to be included in P1394A

This section contains a list of proposed modifications of / additions to the text of the 1394 standard, including a reason why we think the change is necessary.

3.1 Determination of bus manager

Reference: IEEE1394-1995 standard, section 8.4.2.5.

It is currently stated that the incumbent bus manager has reestablished itself as the bus manager if the old_value returned in the lock response is 0x3F. This is true if the bus operates error-free.

In the case of transaction failures (bit errors, repeated busy-acks for responses) it is possible that the BUS_MANAGER_ID has been updated but the lucky bus manager did not get the response indicating this. A retry of the attempt to establish itself as bus manager will fail and result in a lock response with old_value equal to the actual phy_id of the node attempting to establish itself as bus manager.

We therefore propose to change the text so that the candidate bus manager has reestablished itself as the bus manager if the old_value returned in the lock response is either 0x3F or its own phy_id.

3.2 Transaction data indication

Reference: IEEE1394-1995 standard, sections 7.1.2.3 and 7.3.3.2.2.

The transaction layer communicates a data indication to the application, node controller or bus manager whenever it receives a transaction request (in the form of a link data indication from the link layer). The application, node controller or bus manager shall respond to this indication. However, according to the current standard, responses to incoming packets may not be generated if the incoming packet contained a data CRC error or format error.

Since this is not very clear in the text, we propose to explicitly mention in section 7.1.2.3 that a transaction data indication shall only be communicated to the application or node controller if the corresponding link data indication has a packet status other than DATA_CRC_ERROR or FORMAT_ERROR.

For consistency, in section 7.3.3.2.2, transition RX0:RX1 should exclude packets with DATA_CRC_ERROR or FORMAT_ERROR. Transition RX0:RX0b should acknowledge these errors with ACK_DATA_ERROR and ACK_TYPE_ERROR respectively. Consistent with (7.1.1.3), DATA_CRC_ERROR should generate a transaction event indication of REQUEST DATA ERROR.

3.3 Command Reset effect on isochronous resource registers

Reference: IEEE1394-1995 standard, sections 8.3.2.1, 8.3.2.3.7 and 8.3.2.3.8.

Currently, if a command reset is issued to the isochronous resource manager node all isochronous resources will be deallocated because, according to their definitions, BANDWIDTH_AVAILABLE and CHANNELS_AVAILABLE will both revert to their initial values.

The problem is that the previous owners of these resources are not aware of this. They cannot be told because there is no way to find out who owns what. Only a bus reset can repair this situation. Hence the whole concept of command resetting the isochronous resource manager node is useless.

We propose to extend the current definition of the 2 isochronous resource registers CHANNELS_AVAILABLE and BANDWIDTH_AVAILABLE so that these registers do **not** change value at command reset.

3.4 Serial bus control confirmation

Reference: IEEE1394-1995 standard, section 8.2.2 item (h).

It is stated that this primitive communicates several control parameter values back to the application. One of them is the Root ID (item h.).

It also states that "If no cycle master is active, this parameter shall have a value of $3F_{16}$". There is no reason not to inform the application of the Root ID if the root is not acting as cycle master. There is still a root and it's ID is known. We therefore propose to delete the quoted sentence.

3.5 Gap_count value for 16 hops and 4.5 meter cables per hop

Reference: IEEE1394-1995 standard, section 8.4.6.2.

The note below table 8-7 states that a gap_count value of 33 is sufficient for a worst case 16 hop 4.5 meter/hop bus. This should be 42, as the table itself suggests.

3.6 Reallocation time limit for isochronous resources after bus reset

Reference: IEEE1394-1995 standard, annex H (informative).

Reallocation of prior isochronous resources may be attempted as soon as possible after any bus reset. Allocation of new resources must wait until at least 1000 msec. have elapsed since the last bus reset. Although not specified anywhere in the normative part of the standard, it is suggested in annex H, figure H-1 and text below about point B, that at 625 msec. after completion of the self identify phase, all prior isochronous resources have been reallocated.

This raises the following question: Is reallocation of prior resources permitted between 625 msec. and 1000 msec. after a bus reset?

3.7 Local self-ID packets to be communicated to link layer

Reference: IEEE1394-1995 standard, section 4.4.2.3.2.

The physical layer is required to send all received self-ID packets to the attached link layer. The local self-ID packet(s) are only transmitted onto the cable, but not sent to the local link layer. The C code for SELF_ID_TRANSMIT_ACTIONS in table 4-46 lacks the proper statements.

This makes the self-ID analysis more complex because the attached manager has to read the local PHY registers to compile this info on its own before it can perform the mandatory consistency checks (see section 8.4.2.3). Therefore we would like to propose that the specification is changed such that physical layer implementations are required to communicate their own self-ID packets to the local link layer.