1394c (S800T) symbol encoding for tolerance to single byte errors

Jerry Hauck and Colin Whitby-Strevens
September 26th, 2003

1 Summary

The current proposal for encoding 1394 data, arbitration states and control states onto 10 bit symbols for use in 1394c (aka T_mode) is not robust to byte errors which may occur when the 802.3 Clause 40 physical layer encoding is used to transport 1394 protocols. In particular a single byte error in the 802.3 Clause 40 encoding may result in two errored 10 bit symbols at the 1394 level. This is particularly of concern should the two symbols be a pair of 1394 control symbols (used to separate data from arbitration requests).

This paper proposes an alternative encoding which has the property that a single 802.3 Clause 40 byte error will never result in two consecutive 1394 control symbols being errored. It also adds features to assist robustness in the case of errored arbitration requests, and robustness in the case of bursts of byte errors.

2 Background

IEEE 1394c aims to extend 1394b to use the IEEE 802.3 Clause 40 physical layer encoding to transport FireWire protocols at the S800 effective data rate over unshielded twisted pair cabling at distances of up to 100m.

The FireWire arbitration protocols defined in 1394b are retained unchanged. In effect, the 1394c specification allows one or more ports on a PHY to operate using the new encoding, while the remaining ports operate using the existing encodings defined in IEEE 1394a-2000 or IEEE 1394b-2002. 1394c uses the same packet formats and arbitration signaling as are defined in 1394b for use in “Beta mode” operation.

The 1394b format alternates arbitration request signaling and packet transmission. Special control symbols are used at the start and end of packets. All symbols are encoded as 10 bit values. The 1394b format is illustrated as follows:-

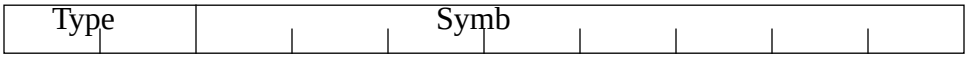
where arb is a data symbol representing an arbitration request, SP is a SPEED control symbol which introduces a packet and indicating its “speed”, DP is a DATA_PREFIX control symbol which introduces a packet, d is a data symbol representing packet payload, and DE is a DATA_END control symbol which terminates a packet. For robustness against single bit errors, control symbols are typically sent twice, or at least a packet is introduced by a pair of control symbols, both of which set the “in packet” context.

In 1394b Beta mode, all symbols are encoded as 10-bit symbols using an extension of the IBM "8B10B" code. Both arbitration signals and data signals use 8B10B "data values". The 1394b encoding requires that the "in-packet" context be maintained in order to distinguish between a data symbol used as packet payload and a data symbol used as an arbitration request symbol. The "in packet" context is set by the SPEED and DATA_PREFIX control symbols, and reset by control symbols such as DATA_END, GRANT, ARB_CONTEXT.

In 1394b Beta mode, signaling uses NRZ encoding, which is electrically specified to have a bit error rate of less than 1 error in 10^{12} bits. The 8B10B encoding will always detect a single bit error in a control symbol, and will detect a single bit error in a data symbol either immediately or shortly afterwards due to a running disparity failure. If a bit error occurs in an arbitration symbol, this is relatively harmless as the error will be corrected in the succeeding arbitration symbol. The 1394b arbitration mechanisms are also robust against dropped or erroneous arbitration symbols - in general the result will be no worse than "unfairness" or a dropped isochronous packet, and usually totally benign. If a bit error occurs in the packet data payload, then this will be caught by packet checksums, and a retry made (where appropriate).

The format uses double control symbols at all times, in particular at the start and end of packets, to provide a measure of robustness against a bit error occuring within a control symbol. The chance of two errors occurring in two consecutive control symbols in an otherwise relatively reliable connection is considered remote.

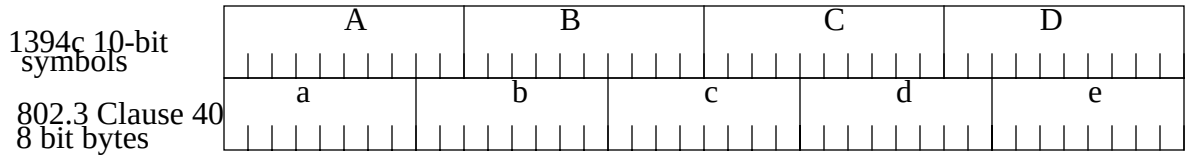
In 1394c T mode, the symbols are (currently) encoded as follows:-



0	0	Data value
0	1	Arbitration Request (encoded)
1	0	Control Symbol (encoded)

The Arbitration Requests and Control Symbols are encoded identically to 1394b before 8B10B encoding. This encoding lacks any form of robustness. The "IBM 8B10B" style of encoding is not considered useful, as the failure mode of the 802.3 Clause 40 signaling is not to generate single bit errors.

The resulting 10 bit symbol stream is then transmitted as a stream of 802.3 Clause 40 8 bit bytes, on a "4 to 5" basis - i.e. five 8 bit bytes transmitted for every four 10 bit 1394c symbols, as illustrated thus:-



3 Problem statement

The electrical signaling used for transmission of 802.3 Clause 40 bytes may result in an errored byte. In an errored byte, all the 8 bits can be considered as scrambled (unreliable). The immediate problem that this causes is that a single 802.3 Clause 40 error will result, in some cases, in two consecutive 1394c symbols being in error.

For example, in the above figure, an error in byte a will result in symbol A being errored, and an error in byte e will result in symbol D being errored. But an error in byte b will result in both symbols A and B being errored, c will impact B and C, and d will impact C and D. Thus there is a 60% chance of a single 802.3 Clause 40 error affecting two consecutive symbols.

Depending on where in the data stream the original error occurs, there is a significant possibility that the two errored 1394c symbols will be the two consecutive control symbols which either introduce or terminate a packet. This is sufficiently serious that an alternative, more robust encoding is needed.

4 Proposed Encoding, overview

4.1 Robustness assistance

One benefit of the 802.3 Clause 40 signaling scheme is that errored bytes are flagged as such. Thus one element of robustness can take advantage of this - the receiving port can take special action on receipt of an errored byte.

4.2 Main properties

The new encoding seeks first to protect the symbol type, by:-

1) using only two symbol types - DATA or NON_DATA
2) replicating the symbol type bit
3) placing the two replicated bits at either end of the symbol

Thus a single byte error may affect a type bit in two consecutive symbols, and will affect one or the other of the type bits in a single symbol, but will never affect both type bits in a single symbol.

Having protected the symbol type, the new encoding seeks to protect at least one of a pair of control symbols by using alternate encodings for symbols in positions A and B and for symbols in position C and D above.

This exploits the fact that there are only 16 control symbols values required (actually, only 14). Thus a control symbol can be distinguished from a request symbol, and a full decode provided, in five bits.

In the case of A and B, the control encoding can be distinguished from request encoding and be completely determined from the more significant 5 bits after the most significant type bit, and in the case of C and D, from the less significant 5 bits before the less significant type bit.

Similarly, the new encoding further seeks to protect the distinction between control symbols and arbitration requests by

1) replicating the bit which distinguishes between control symbols and arbitration request symbols
2) placing the two replicated bits at either end of the sub-symbol

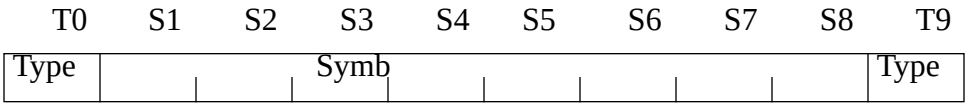
This allows further robustness from knowledge of whether the symbol is a control symbol or an arbitration request, even though the particular control or request information may have been lost.

Thus in the case of errored byte a, symbol A is lost.
In the case of errored byte b, symbol B is lost, but symbol A is still decoded correctly if a control symbol.
In the case of errored byte c, both smbols B and C are still decoded correctly if they are control symbols.
In the case of errored byte d, symbol C is lost but symbol D is decoded correctly if a control symbol.
In the case of errored byte e, symbol D is lost.

In all the above cases, some robustness can be provided on the basis of still being able to distinguish whether the lost symbol is a data symbol, a control symbol or an arbitration request.

The new encoding also permits arbitration requests to be received correctly despite the symbol being affected by an errored byte in some situations.

5 Proposed encoding - details



0	Data value	0
1	Control Symbol or Arbitration Request (encoded)	1

5.1 Symb encoding for Arbitration Requests

`0aaaiii0` - Arbitration request, `aaa` set as per bits ABC in 1394b Tables 13-2, 13-4 and 13-5, `iii` set as per bits DEH in 1394b Tables 13-3, 13-4 and 13-5.

5.2 Symb encoding for control symbols when T0/T9 = 1 - symbol positions AB

`1cccc001` - Control symbol, `cccc` set as per bits PQRS in 1394b Table 13-1.

5.3 Symb encoding for control symbols when T0/T9 = 1 - symbol positions CD

`100cccc1` - Control symbol, `cccc` set as per bits PQRS in 1394b Table 13-1.

6 Further robustness measures

If a data symbol is encountered when not in packet context, then it and all succeeding data symbols are replaced by DATA_NULL. This includes the packet ending symbol if DATA_END, but not if GRANT or DATA_PREFIX. The packet remnant is turned into a NULL packet as the speed of the packet is unknown, and also the speed at which to send the DATA_END is unknown.

It should be noted that if an unexpected end of packet occurs (for example, on the reception of a sudden IDLE or an arbitration request) when in packet context, then the arbitration state machine terminates the packet with DATA_END, if sufficient time is available, otherwise it terminates the packet with an ARB_CONTEXT control symbol. This already provides robustness against dropped end of packet control symbols.

7 Symbol decode rules

(Note, in the tport state machine C code, the distinction is made between pkt_prefix and pkt. Both of these are represented here as “in packet”).

The symbol decode rules are divided into two groups. The first group deals with symbols where the bytes contributing to the symbol have been received correctly, but some previous error may have led to some form of inconsistency.

The second group deals with symbols where one or both of the bytes contributing to the symbol has or have been received marked as erroneous by the 802 compatible PMD.

If a received control symbol or arbitration request can be fully decoded according to the tables below, and is specified in the table as “Accepted”, but does not correspond to a valid symbol or arbitration request, then it is ignored and the invalid count is incremented (possibly as part of being incremented due to receipt of an errored byte or burst of bytes). Other actions noted in the table are still taken.

Table 1: Symbol decode rules for non-errored symbols

#	T0	T9	S1	S8	In packet?	Result
1	0	0	x	x	N	Replace with DATA_NULL
2	0	0	x	x	Y	Accept as data
3	opposite values		x	x	x	Ignore, increment invalid count
4	1	1	0	0	N	Accept arbitration request
5	1	1	0	0	Y	Accept arbitration request if valid (missed packet ending) and clear “in packet”, otherwise ignore and increment invalid count
6	1	1	opposite values		x	Errored arbitration or control request, ignore and increment invalid count
7	1	1	1	1	N	If invalid control symbol, ignore and increment invald count. Set “in packet” if DATA_PREFIX, SPEEDa or SPEEDb. Replace DATA_END by DATA_NULL.
8	1	1	1	1	Y	If invalid control symbol, ignore and increment invald count. Clear “in packet” unless DATA_PREFIX, SPEEDa, SPEEDb, SPEEDc or DATA_END.

Table 2: symbol decode rules for errored symbols

#	Symbol position	Errored byte(s)	T0	T9	S1	S8	In packet?	Result
9	A	a	x	0	x	x	Y	Accept as data.
10	A	a	x	0	x	x	N	Replace with DATA_NULL
11	A	a	x	1	x	0	N	Ignore
12	A	a	x	1	x	0	Y	Ignore, clear “in packet”
13	A	a	x	1	x	1	x	Ignore
14	A	ab	x	x	x	x	x	Ignore
15	A	b	0	x	x	x	N	Replace with DATA_NULL
16	A	b	0	x	x	x	Y	Accept as data

Table 2: symbol decode rules for errored symbols

#	Symbol position	Errored byte(s)	T0	T9	S1	S8	In packet?	Result
17	A	b	1	x	0	x	N	Accept as arbitration request
18	A	b	1	x	0	x	Y	Accept as arbitration request, clear “in packet”
19	A	b	1	x	1	x	N	Accept (as per row 7 above)
20	A	b	1	x	1	x	Y	Accept (as per row 8 above)
21	B	b	x	0	x	x	N	Replace with DATA_NULL
22	B	b	x	0	x	x	Y	Accept as data
23	B	b	x	1	x	0	N	Ignore
24	B	b	x	1	x	0	Y	Ignore, clear “in packet”
25	B	b	x	1	x	1	x	Ignore
26	B	bc	x	x	x	x	x	Ignore
27	B	c	0	x	x	x	N	Replace with DATA_NULL
28	B	c	0	x	x	x	Y	Accept as data
29	B	c	1	x	0	x	N	Ignore
30	B	c	1	x	0	x	Y	Ignore, clear “in packet”
31	B	c	1	x	1	x	N	Accept (as per row 7 above, but interpreting the control value purely on S2-S5)
32	B	c	1	x	1	x	Y	Accept (as per row 8 above, but interpreting the control value purely on S2-S5)
33	C	c	x	0	x	x	N	Replace with DATA_NULL
34	C	c	x	0	x	x	Y	Accept as data
35	C	c	x	1	x	0	N	Ignore
36	C	c	x	1	x	0	Y	Ignore, clear “in packet”
37	C	c	x	1	x	1	N	Accept (as per row 7 above, but interpreting the control value purely on S4-S7)

Table 2: symbol decode rules for errored symbols

#	Symbol position	Errored byte(s)	T0	T9	S1	S8	In packet?	Result
38	C	c	x	1	x	1	Y	Accept (as perrow 8 above, but interpreting the control value purely on S4-S7)
39	C	cd	x	x	x	x	x	Ignore
40	C	d	0	x	x	x	N	Replace with DATA_NULL
41	C	d	0	x	x	x	Y	Accept as data
42	C	d	1	x	0	x	N	Ignore
43	C	d	1	x	0	x	Y	Ignore, clear “in packet”
44	C	d	1	x	1	x	x	Ignore
45	D	d	x	0	x	x	N	Replace with DATA_NULL
46	D	d	x	0	x	x	Y	Accept as data
47	D	d	x	1	x	0	N	Accept as arbitration request
48	D	d	x	1	x	0	Y	Accept as arbitration request, clear “in packet”
49	D	d	x	1	x	1	N	Accept (as per row 7 above)
50	D	d	x	1	x	1	Y	Accept (as per row 8 above)
51	D	de	x	x	x	x	x	Ignore
52	D	e	0	x	x	x	N	Replace with DATA_NULL
53	D	e	0	x	x	x	Y	Accept as data
54	D	e	1	x	0	x	N	Ignore
55	D	e	1	x	0	x	Y	Ignore, clear “in packet”
56	D	e	1	x	1	x	x	Ignore

As 802.3 Clause 40 errors may well occur in bursts of multiple bytes, the invalid count is inremented only on the first error in a burst of 64 bytes or less. The invalid count is incremented for every 64 bytes errored in a burst.

The port will also maintain a register of the maximum errored byte burst length.