SpaceWire – Improvements in Support of Mission Requirements

Barry M Cook [1], Paul Walker [2]

[1,2] 4Links Limited
The Mansion,
Bletchley Park,
Milton Keynes,
MK3 6ZP,
UK
Email:[1] barry@4Links.co.uk
Email:[2] paul@4Links.co.uk

ABSTRACT

SpaceWire is a data transmission standard that defines communication links and networks. Fault-tolerant, redundant systems of high performance can be built. Link speeds currently available in products exceed 400Mb/s and system speeds are a large multiple of this. N+1 or N+M redundancy can be provided, as can point-to-point bandwidth multiplication by grouping links.

Time signal propagation is defined by the current standard across a SpaceWire link and globally throughout a network. A short (6-bit) value is transferred to all nodes in the network. It is intended that this value represents a tick that increments at regular intervals and is enhanced by transmission of courser interval data by means of supplementary packets transmitted normally. Jitter is of the order of micro-seconds. 4Links has demonstrated a technique, fully compliant with the standard, which allows zero-jitter time codes from the time-code generator and jitters of the order of nano-seconds across a network. This technique will be described and results graphically shown.

Routing switches, as defined in the current standard, are able to recognize packets as having one of two priorities. Whenever more than one packet is waiting for transmission, the router will select the higher priority packet. Packets, whether low or high priority, are forced to wait while any packet currently being transferred completes its passage through the link. Such delays occur at each routing switch and their duration is a function of the sizes of packets in the system. The delays have high variance and do not permit fast real-time performance. We propose introducing a pre-emptive priority mechanism by which transmission of lower priority packets may be temporarily suspended so that urgent packets may delivered to tight time deadlines. We show how this can be implemented.

Errors on a link cause that link to re-start, resetting values and recovering to a known good state. Unfortunately, an error in one direction of data transfer causes loss of data in the opposite direction while recovering. There is a loss of communication for many micro-seconds as this process completes. Such a long gap exists for historical compatibility reasons but need not be a limitation on new designs. We show how it is possible to significantly reduce recovery time in the direction in which the error occurs and to eliminate it entirely in the opposite direction.

.

KEYWORDS
SpaceWire, Real-time, FDIR, Determinacy

1. INTRODUCTION

Standards ISO/IEC 14575[1] and IEEE 1355[2] define an inter-processor communication link suitable for multi-processor applications. Data transfer is high speed and, more importantly, low latency. This standard was used to create links integrated with high performance processors and a low-latency routing switch, using *cut-through* (also known as *wormhole*) routing and outstanding aggregate performance demonstrated.

Some aspects of IEEE 1355, such as the signal levels (PECL for wired connections) and connectors used are not suited to Space applications and led to a variant being defined which has become known as *SpaceWire* (ECSS-E-50-12A[3]).

SpaceWire currently defines only twisted-pair wired connections using LVDS signaling and specifies connectors that are available in Space qualified versions (IEEE 1355 also contains co-axial cable and optical transmission specifications). One extension made to IEEE 1355 is the inclusion of a mechanism for global distribution of values for use as time-codes.

Although SpaceWire offers major benefits, for example multi-hundred megabit-per-second transmission without using phase-locked loops and the ability to build fault-tolerant networks, there are some limitations. One such is that it is difficult to determine how long it takes to deliver data and such times have high variance. Improvements are possible, often completely compatible with existing components. We consider here time-code jitter, real-time performance and time to recover from an upset, showing that significant improvements in determinacy are possible leading to increased applicability of SpaceWire in mission critical areas.

2. SPACEWIRE – OVERVIEW

SpaceWire defines a point-to-point link protocol between end-nodes and/or routing switches. Link implementations have shown that per-link transmission rates of several-hundred megabits-per-second are achievable. Systems built with routing switches run links concurrently – bandwidth accumulates as the system grows (unlike bussed systems where bandwidth is shared). Multiple links between routing switches are permitted which allows redundancy and/or bandwidth multiplication (device to device bandwidth may be increased by sharing 2 or more links). Low-level features in the routing switches make such multiple links transparent either as bandwidth multipliers or alternate connections in the event of a link failing (temporarily or permanently) – failover times are of the order of micro-seconds.

SpaceWire transmits basic tokens of either 4- or 10-bits. Each token contains a flag bit indicating its type and a parity bit for detecting transmission errors. Short, 4-bit, tokens are used for control – encoding end-of-packet marks, flow control credit and one *escape* token. Long, 10-bit, tokens are used for data. Compounds are formed with the escape control token: escape with flow control for a null token, and escape with data for a time code – see Table 1. Escape with end-of-packet or another escape is not currently used and their detection results in an error condition which, like parity errors, causes the link to be reset.

In order to confirm connection and to be able to detect loss of connection, the link is always sending tokens. Priority is given to time-code and flow control tokens, then data and flow control tokens, and finally, if there is nothing of more importance then a Null token is sent in order to maintain continuous transitions and indicate that the link is still connected.

Data transmission is subject to flow control, no data can be sent unless the receiver has space to receive it. Data transmission in one direction is matched by flow-control information in the reverse direction. Although flow control is per link, there is effectively end-to-end flow control with some en-route buffering. Such flow control ensures that data is not lost due to hold-ups in the network or receiving node, but can result in (temporary) blocking of paths in the network.

Data is transmitted in *packets*, the first byte(s) of which are interpreted as routing information and the tail is marked by an *end-of-packet*, EOP, token (or *error-end-of-packet*, EEP, if an error occurred in the transfer of that packet). Once a link has started to transfer a packet it is committed to transferring the whole of the packet – with consequences for the real-time behaviour of the system.

Table 1. SpaceWire tokens

	Use	Length
EOP	End-of-packet marker	4-bits
EEP	End-of-packet marker	4-bits
FCT	Flow control	4-bits
Data	Byte-wide data	10-bits
ESC-FCT	Null	8-bits
ESC-Data	Time code	14-bits

Link errors not only corrupt data but also desynchronize the flow control mechanism and token boundaries. In order to re-synchronize token boundaries and flow control, an error causes a link to stop sending for a period (several micro-seconds) so that a timeout is detected and both ends agree to restart from a known (idle) state. A gap in transmission of several micro-seconds at several hundred mega-bits per second causes a considerable delay in data transmission. In the case of a transient error affecting only the flow-control direction of a link, the loss in the data transfer direction may not be necessary if an alternative error-recovery technique is used.

3. SPACEWIRE TIME CODES

Time codes are globally transmitted – broadcast – from a single source to all nodes in the network. SpaceWire networks are permitted to have multiple connections in order to provide redundancy and fault tolerance. These alternate paths form loops that can lead to livelock with a simple broadcast mechanism – SpaceWire routing switches may receive a time code already sent and must only selectively transmit new values. A time code, therefore, contains a sequence number in support of this requirement.

Six-bits are currently used to define the sequence number and, together with two reserved bits, are contained in a normal data token. An escape token precedes this data token in order to identify it as a time code.

In normal usage, time codes are transmitted from the time source at regular intervals. Each node receives both the sequence number (and reserved bits) and a *tick* indicating its arrival time.

The time-code source and each routing switch does, however introduce some jitter on the received time code, which can amount to several microseconds. It is possible, completely within the standard, to send, forward and receive time-codes in such a way that the jitter is reduced to the order of tens of nanoseconds. Furthermore, it is possible to align received time-code indications so that all nodes are synchronized to within a few tens of nanoseconds.

We analyze the source of time uncertainty – jitter – and show how it may be reduced to zero on the time-code source, and close to zero in a network

3.1 JITTER PERFORMANCE

Jitter, uncertainty, in the time-code indication – the time code “tick” – has two sources. The SpaceWire transmitter being the major contributor, with a small addition from the receiver. It is assumed that a well-designed routing switch has a fixed delay and adds no further uncertainty.

3.1.1 Transmitter Contribution

Requests to send a time code will not occur at a regular time relative to the transmitted token stream. We may expect time ticks to be generated at regular intervals but the tokens, being of variable length, may not stop and start at the same times.

It may be possible to send a time-code at the instant a tick request is made, or it may be necessary for the currently being sent token to finish first – a delay of up to 10-bit times (we assume that a time tick will not be so soon after a previous time tick that it must wait 14 bit times). Thus, we see an introduced uncertainty, jitter, of 10 bit-times.

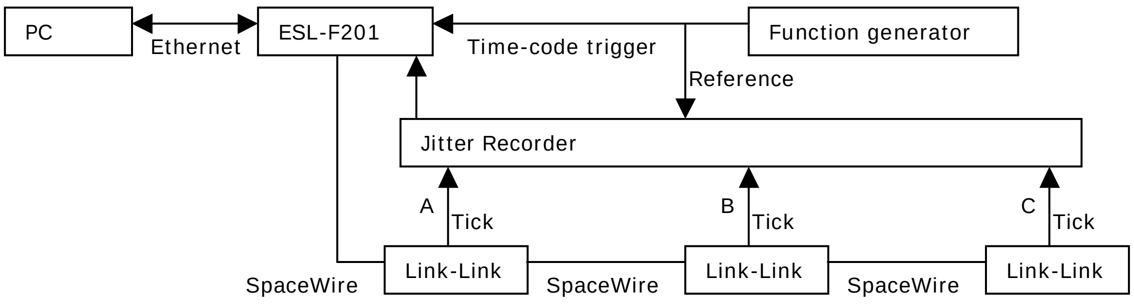


Fig. 1. Experimental test harness

The approved standard [3] considers only this contribution in its example jitter calculation. It takes the default transmit rate of 10Mb/s to give a transmit uncertainty of 1µs at each link. Across system having ten such links we see an accumulated jitter expectation of some 10µs.

3.1.2 Receiver Contribution

Link ends are not required to share a common clock signal, and in general will not do so, with the result that the SpaceWire transmitter and receiver are not synchronized. The receiver must resynchronize what is received with its own clock and an uncertainty of one sampling period is inevitable. The receive sampler may operate at a frequency corresponding to the link bit rate but need not do so. For example, the XOR-recovered clock may be used to group received bits into tokens and these tokens passed to the local clock domain, which may therefore be at a lower frequency than the bit-rate might suggest. The sampling uncertainty occurs at this last transition and can thus be greater than one bit time.

3.1.3 Experimental Verification

In order to test this theoretical prediction, and to provide a reference for comparison with an improved scheme, a test harness was constructed, as shown in Fig. 1. A PC connected, via Ethernet, to an EtherSpaceLink ESL-F201 able to generate time-codes at intervals set by an external function generator in turn connected by SpaceWire link to a unit

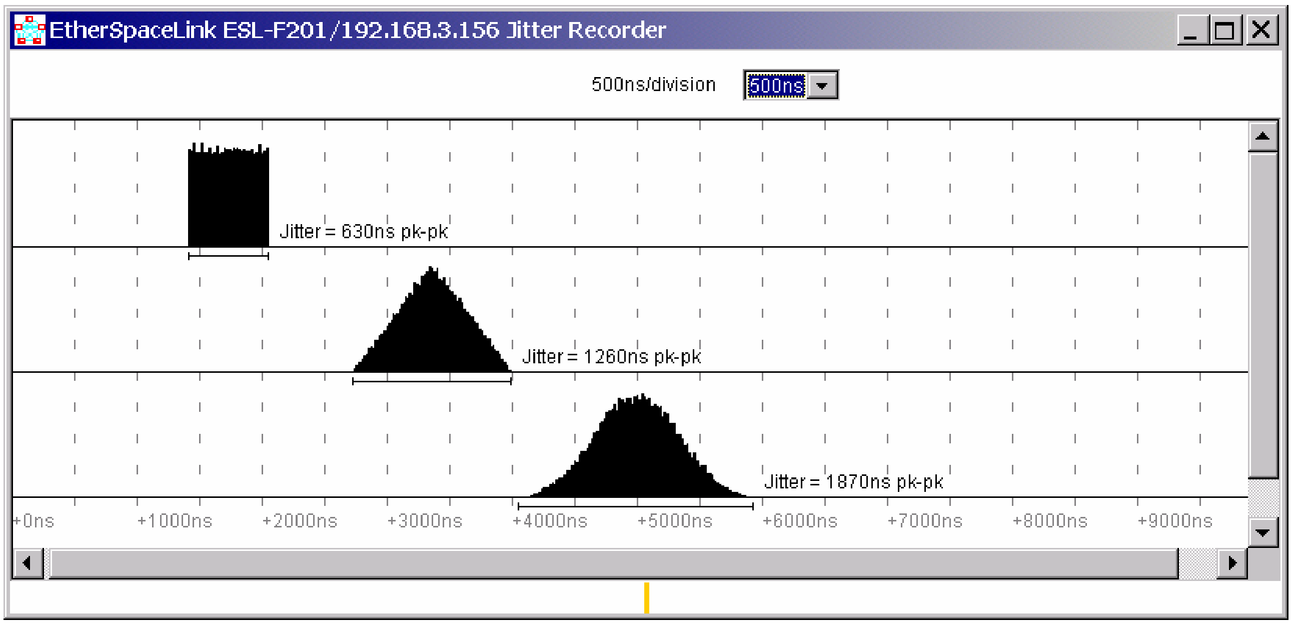


Fig. 2. Time code behaviour with links running normally (Idling at 12.5Mb/s)

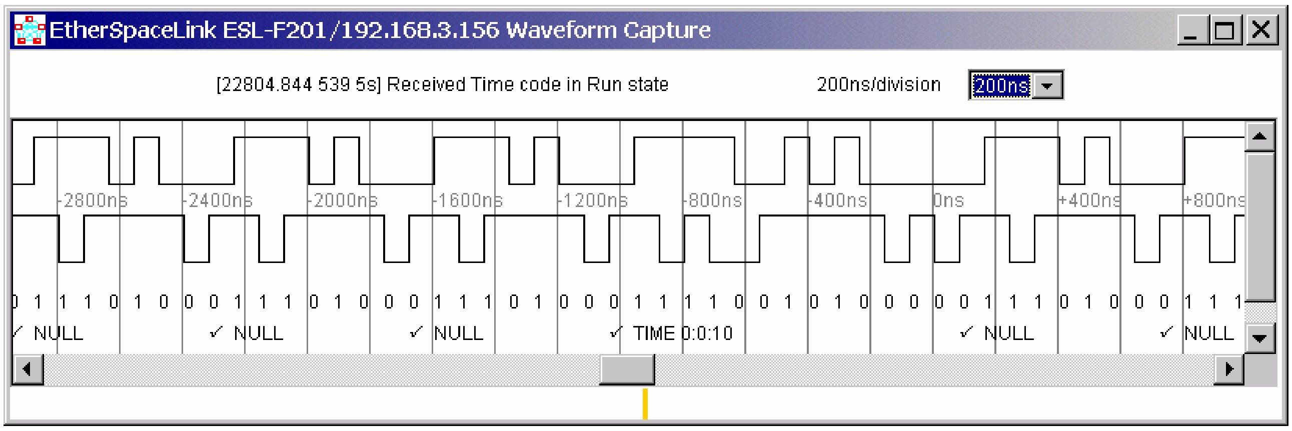
containing two link engines to copy data to another SpaceWire link and again on to a third link. Reference signals and received time ticks were collected in a jitter recorder that accumulated the number of ticks received at each of 1023 intervals of 10ns after the reference – a probability density function – the ESL-F201 was used to pass the recorded data back to the PC. An external function generator was used in preference to the internal time-code generator in the ESL-F201 in order to de-synchronize the tick requests from the link clock.

Links were set to transmit at 12.5Mb/s, for reasons explained later, and a large number of time-ticks were generated and recorded.

The resulting distributions are shown in Fig. 2. The top trace shows the distribution after one SpaceWire link (point A in Fig. 1.), the middle trace after two links (point B in Fig. 1.) and the bottom trace after three links (point C in Fig. 1.). Due to the need to scale the number of ticks received at each time point, it is hard to see points at which only a relatively few ticks were received – a bar appears under each group of received ticks indicating the range from earliest to latest, together with text giving the difference between these times – the peak-to-peak jitter.

All the SpaceWire links are idling, sending NULLs but no data. The transmit jitter introduced is thus only 8-bits, rather then up to 10-bits as would be found in an active link. Measured jitters should be scaled by 10 / 8 to get true values.

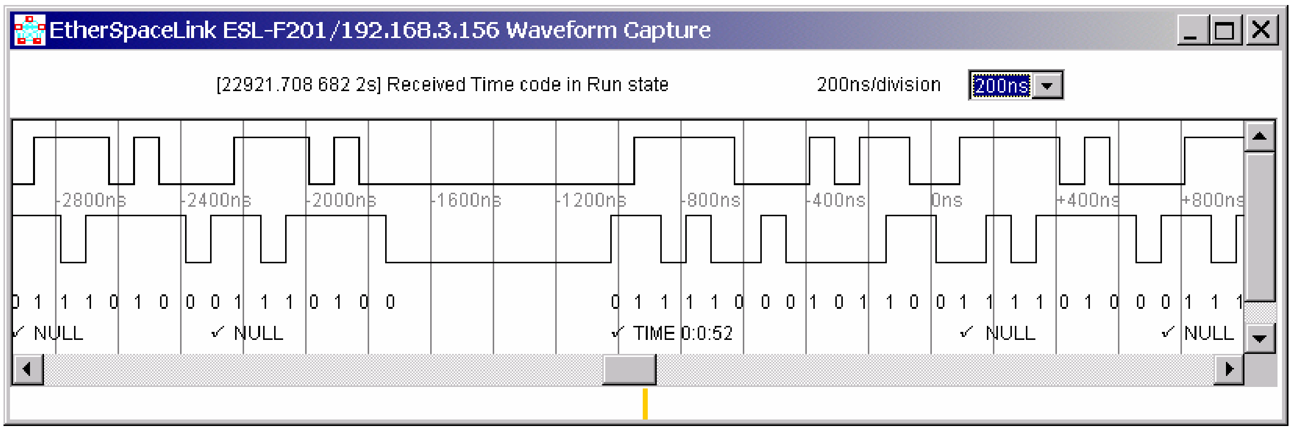
At 12.5Mb/s we have a bit time of 80ns and would expect to see a jitter of 8 * 80ns = 640ns. As seen in the top line of Fig. 2., our experiment confirms this. Traversing the second link doubles the jitter and the third link triples it. It is interesting to note that the distribution is rectangular at the first stage, triangular at the second and reaches a fair approximation of a normal distribution after three stages.


EtherSpaceLink ESL-F201/192.168.3.156 Waveform Capture

[22804.844 539 5s] Received Time code in Run state

200ns/division 200ns

Fig. 3. Physical layer waveforms at 12.5Mb/s, showing an inserted time-code.


EtherSpaceLink ESL-F201/192.168.3.156 Waveform Capture

[22921.708 682 2s] Received Time code in Run state

200ns/division 200ns

Fig. 4. Physical layer waveforms at 12.5Mb/s, showing a delayed inserted time-code.

3.2 LOW-LEVEL CODING – AN OPPORTUNITY TO DO BETTER

SpaceWire uses a Gray-code on two pairs of wires to transfer both data and their clock signal. Fig. 3. shows a sample of the signals on the “Data” (top line) and “Strobe” (bottom line) lines. As can be seen the bits are sent at regular intervals – every 80ns in this instance, corresponding to a data rate of 12.5Mb/s. Time codes are inserted into the stream at the next available regular interval.

Such regular bit times are not, however, obligatory. A benefit of carrying the timing data in the signals is that bits can be conveyed at irregular intervals, so long as each is interval is long enough to be identified by the receiver and not so long as to trigger a disconnect timeout. We can use irregular intervals to remove the major source of time-code jitter.
The interval between a time-code request and its being sent varies depending on how long the transmitter is occupied sending its current token. This variable interval causes the large jitter observed.

It is not necessary to send a time-code as soon as it is possible to do so. We may delay sending the code, until some time after the transmitter is able to send it. By doing so we may arrange for the interval between the request for time-code transmission and its being sent to be constant – and jitter-free. See Fig. 4.
The delay introduced must be as long as the longest token that may be being transmitted when the time-code request arrives – 10-bits – in order to be sure that it has been completely sent before the time-code must be sent. The period for which the data and strobe lines is held will be between zero (when the time-code request arrives exactly 10-bit times before the last token is completely sent) and ten bit times (when the last token finishes just as the time-code request arrives). In order to avoid a disconnect detection this delay must not exceed the timeout period of 850ns nominal. Taking a practical limit of 800ns over 10-bit times requires the link to be running at 80ns / bit or less – 12.5Mb/s or faster.

Time-code jitter is dramatically reduced by this technique, as can be seen in Fig. 5.

3.3 ACHIEVABLE PERFORMANCE

Delayed time-code transmission totally removes the transmitter contribution to jitter leaving only the receiver to transmitter synchronization uncertainty. A jitter performance of only 10ns per link is easily achievable.

The cost of this approach is the introduction of up to 10 bit-periods of inactivity, average 5 bit-periods, on the data and strobe lines. It is almost certain that time-codes constitute only a small fraction of the traffic on a link and their extension from 12- to 17-bits (average) will have minimal affect on link throughput.

Although the effect is most dramatic at low speeds, it is still useful at higher speeds. A 200Mb/s link has a bit-period of 5ns and 10-bits of jitter amount to 50ns. Reducing this to 10ns (or less, depending on the link system clock speed) is still a worthwhile gain – especially when multiplied by the number of hops in a network.

Links are required to start at 10Mb/s [3] but time-codes are not permitted to be sent until the link is running, at which time the link speed can be increased to at least 12.5Mb/s.

Lower speed links can also benefit from a modified form of this technique, as described in the next section.

3.4 REDUCING THE LOWER SPEED LIMIT

We assume, in the above calculation of the lower speed limit, that the delay is introduced only in one place – before the start of the time code. It would also be possible to add a little bit of delay to each bit of the time code itself to achieve a total of the required delay.

Before we can do this, however, we must be careful to consider when a time tick is generated from a received time code. All the added delay must occur before that point in time. It is not possible to know that a time-code has been correctly received until all its data bits and parity bit are seen and checked. At least ten bits are available for extension to give the required delay, allowing a link speed of down to 2.5Mb/s.

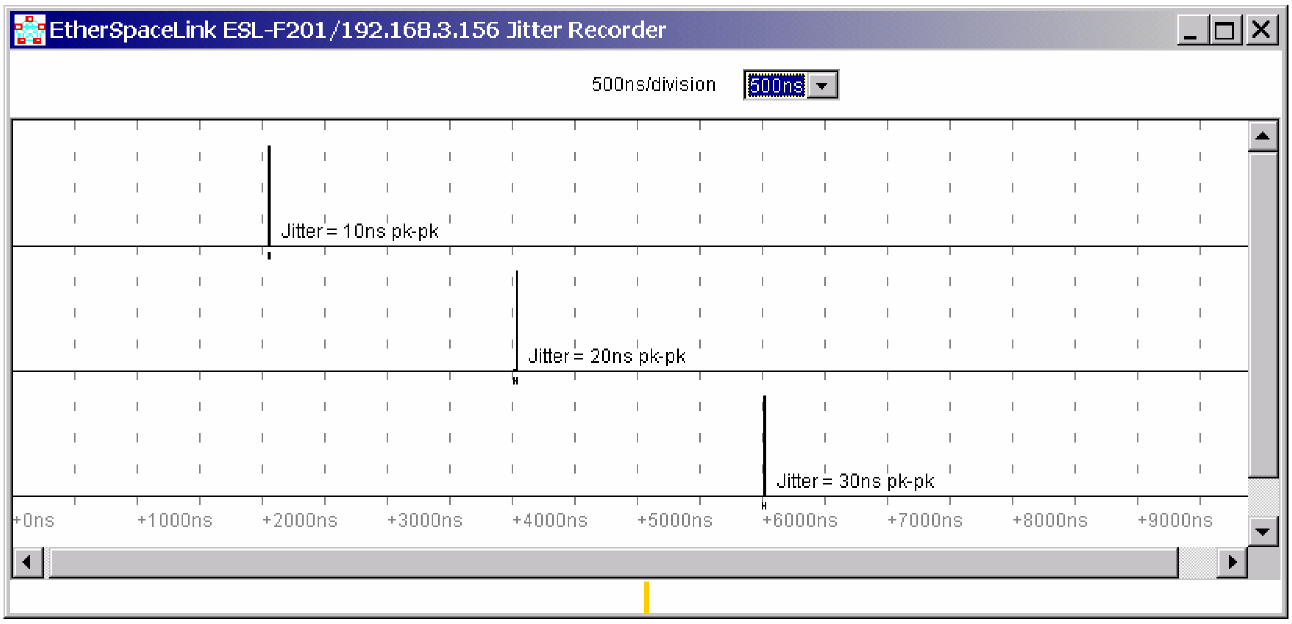


Fig. 5. Time code behaviour with delayed time-codes (Idling at 12.5Mb/s)

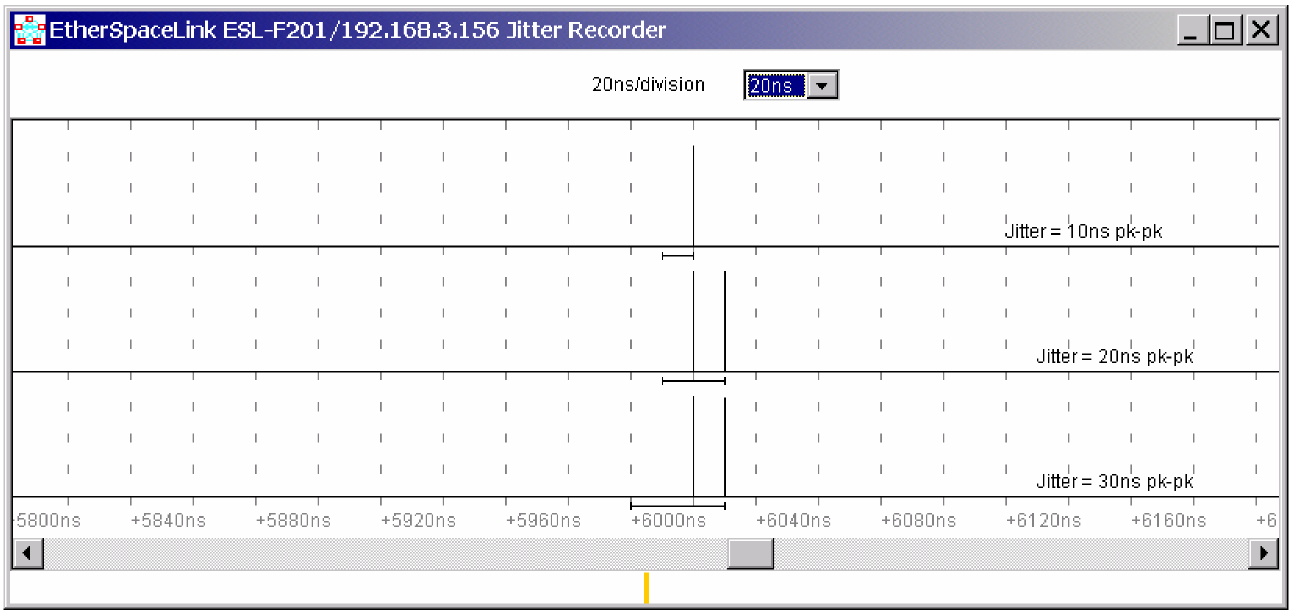


Fig. 6. Time code behaviour with output delays added

3.5 GLOBAL SYNCHRONISATION

Given low-jitter times it is possible to consider synchronizing all nodes in the system. Some nodes receive their time-codes before others but the order of receipt will be deterministic – for a given time-code source.

It is thus possible to delay early ticks at the time-code output of the links so that all are produced at the same time. Fig. 6. shows the test system with ticks at point A delayed by 3970ns and those at point B delayed by 1990ns while those at point C are not delayed at all. The result of the delays is that all ticks are received at the same time, to within the jitter interval – note the much expanded time scale of this figure relative to those above. The delays added compensate for the latency of the routing switches and also of the connecting cables (at about 5ns/m these cannot be ignored).

If time-codes are sourced at a different point in the network (e.g. if the primary source fails and its function is taken by an alternate source) it will be necessary to change the output delays applied – inclusion of a writable register between the link receiver and tick output pin is suggested.

It might be preferable to fix the delay from time source to tick at *all* nodes, including the longest path, to a value longer than that of the longest path, *in all circumstances*. For example, we might set the tick arrival time to be, say, 25µs after it is sent and then to maintain that interval even if the time code source changes. This would avoid time jumps in the event of failure of, and recovery from, time master problems.

4. REAL-TIME PERFORMANCE

Packet delivery determinacy is, currently, poor as packets must compete for access to common links in a network. Although there is a mechanism to prioritize some packets it still does not provide very good determinacy.

We require a system that supports real-time performance to be able to guarantee delivery of messages within a known time interval, and that the time interval is short relative to the control-loop requirements; a low variance in delivery times is desirable. SpaceWire systems constructed according to standard [3] can have predictable maximum delivery times but those times may not be short and the variance may be high.

Pre-emptive routing of (higher priority) real-time packets can achieve low-delay delivery with low variance – a high determinacy is achievable.

4.1 AS CURRENTLY DEFINED

Figure x. shows a minimal network to illustrate the behaviour of a system. Nodes A, B and C are sources of data, sending through routers J and K to destinations P, Q and R. Routers J and K are connected by one link (solid line) or two links (adding the dotted line).

Taking the situation when Node A is sending a packet to Node P and Node C wants to send an urgent (real-time) packet to Node R.

- Node A's packet occupies the link from A to J, the link from J to K and the link from K to P (routing is cut-through/wormhole);
- Node C's packet occupies the link from C to J but then stalls waiting for A's packet to complete.

We can place an upper bound on the waiting time as that time taken for A's packet to be transferred. Assuming that node P takes the packet as quickly as it is delivered, this time is determined by the transmission rate of the links and the size of A's packet. We cannot guarantee this time if we cannot guarantee P's behaviour (we assume that it will not fail and that it will not run slowly).

If we consider the situation when node B also has data to send over the link from J to K we see that if it presents a request while A's packet is in transit then router J has a choice of B's packet and C's packet when A's packet is complete. A simple priority mechanism is given in the standard [3] for this situation. C's packet may be assigned a higher priority than B's so that B's will be rejected in favour of C's. By this mechanism we need only consider the need to wait for one (low priority) message before our high priority message can be sent.

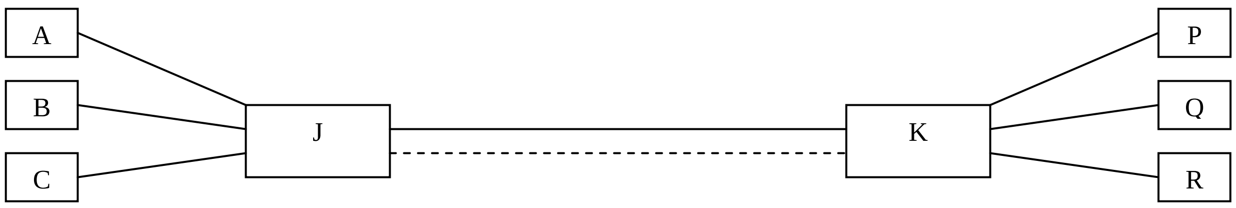


Fig 7 Minimal example network

Adding a second link (shown dotted in Fig x), dedicated to high priority packets, will allow such packets to be sent at the same time as low priority packets on the original link. Reserving a link for high priority messages does not allow its use as backup or bandwidth enhancement for low priority messages.

Since the delay is determined by the largest packet that may block the path it would be possible to require all packets in the system to have a maximum size. Large packets, such as mega-byte images, would have to be fragmented and sent in a number of packets. Some higher level protocols in use and proposed are intended to support fragmentation but they require more processing and protocol headers (typically 10-20 bytes per fragment) reduce available bandwidth.

Link contention may occur at each link in a multi-hop route and delays and variance accumulate across a network.

Packets of 100 bytes occupy 1000 bits (each data token requires 10-bits) if there are no flow control tokens mixed in, or 1050 bits if flow control is included – giving a worst-case wait, at each hop, of 10µs on a 100Mb/s link. One mega-byte packets introduce delays of 105ms per hop.

4.2 PRE-EMPTIVE PRIORITY

Delay and variance can be reduced to small values by allowing a high priority packet to pre-empt a low priority packet and be routed quickly to its destination. Pre-emption need only take a few bit times at each link and would bypass long or even stalled low priority packets. Multiple priorities could be implemented with higher priority packets pre-empting lower priority packets.

It must be understood that such pre-emption would have the effect of extending the transmission time of low priority packets. The delay would be no more than would occur if a low-priority packet was blocked by a higher priority packet in a non-pre-emptive network. If high priority traffic is relatively low volume we are likely to find that there are no significant additional considerations for low-priority data.

4.3 IMPLEMENTATION

Pre-emption is not currently supported in the SpaceWire standard [3] but it could easily be extended to do so.

A control code is required to indicate that a packet has been suspended and a new (higher priority) packet inserted in the link token stream – ESC-EOP is a logical choice. If multiple priorities are supported, the interrupting packet may itself be interrupted – but since strict nesting is implied, no additional control codes are required.

Example – assume that a low-priority packet is being sent when a higher priority packet arrives:
Non-pre-emptive transfer – the low priority packet (LLL) must complete before a high priority packet (HHH) is sent

LLL	EOP	HHH	EOP

Pre-emptive transfer –a high priority packet (HHH) is inserted into the middle of the low-priority packet (LLL)

LLL	ESC-EOP	HHH	EOP	LLL	EOP

Pre-emptive transfer – a very high priority packet (VVV) interrupts a high priority packet (HHH) that interrupts a low priority packet (LLL)

LLL	ESC-EOP	HHH	ESC-EOP	VVV	EOP	HHH	EOP	LLL	EOP

We minimize the delay of the highest priority packet, at the expense of delaying completion of lower priority packets.

Delay for the highest priority now becomes a small constant value at each routing switch and an upper bound on delivery time can be set that does not depend on the size of lower-priority packets, nor on the response time of the receivers of lower-priority packets. In a multi-priority scheme delays may only be introduced by higher priority packets.

4.3.1 PRIORITY PROPAGATION

Within a network, the packet header is used to index a routing table in a routing switch and the content of this table can contain the priority assigned to the packet. End nodes do not need to explicitly handle priority, but they must use appropriate header values. This puts all the implementation detail into the routing switches, and provides compatibility with non-pre-emptive networks (except in terms of performance).

4.3.2 FLOW CONTROL

A stalled low priority process that is waiting for more flow control credit must not be allowed to hinder the progress of a higher priority packet. Separate flow control credits must be maintained at each level of nesting.

4.4 BENEFITS

Predictability is highly valued in determining the behaviour of a system but not well supported by SpaceWire as currently defined. It is possible to introduce determinacy and permit performance predictions to be made with more accuracy than previously thought possible.

5. RECOVERY AFTER UPSET

Link start-up as originally defined for IEEE1355 was designed to support hot-plug and power down/up situations in which a link was broken both directions at once. Space applications add the possibly that a radiation induced event may interfere with only one direction of transfer. As SpaceWire is currently defined it turns a uni-directional upset into a bi-directional break in transmission that not only requires 20µs to correct but also corrupts data flowing in the direction that did not suffer the upset.

Turning a short event is into a significant break seriously reduces determinacy. We propose a fast-restart mechanism that improves determinacy and also reduces data loss due to upsets by not totally resetting transfers the direction not affected by the upset.

5.1 AS CURRENTLY DEFINED

Figure 8 shows simultaneous transfers between two nodes, A and B, and indicates a possible corruption, due to an event, of the transfer from A to B.

The upset causes corruption of data (or idles) in the direction A to B which results in the following error recovery procedure:

- B signals that it has seen an error by halting transmission of the transmission from B to A (it forcibly corrupts this transmission, even though the upset did not affect it directly);
- A detects that loss of transmission from B and halts its own transmission;
- Both ends wait nearly 20µs before starting to transmit a start-up sequence;
- An exchange of idle characters and flow-control tokens re-establishes normal operation.

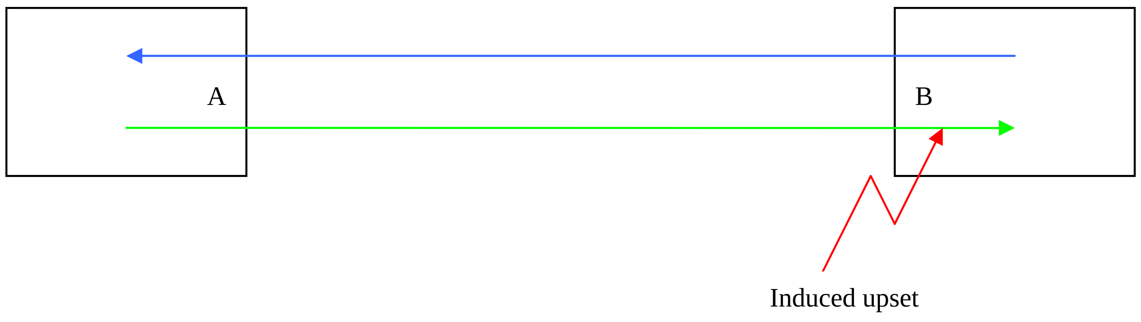


Fig 8. Data Transfer with Upset

Two aspects of the link have to be cleanly established after an error:

- Token boundaries in the data stream;
    - There are no explicit boundaries and tokens are not all of the same length. The link relies on staying synchronized after link-start. Link start establishes boundaries by starting from a situation in which no transitions are sent and detecting the first occurrence of a recognizable bit sequence (an idle character, the NULL token).
- Flow control credit has to be reset so that its value is the same at both ends of the link.
    - The period of no-transitions sets the flow control credit to zero and part of the link-start sequence is the transfer of flow control tokens.

5.2 ALTERNATIVE

Stopping the link, re-synchronizing and restarting it can be achieved on a much shorter time scale than required by the current standard. We present a technique that is similar in principle to the currently defined mechanism but without long periods of silence. Assuming that the link has merely been upset, rather than disconnected, we still have communication paths between A and B. This allows recovery without completely stopping transmission, instead using control tokens and otherwise illegal sequences to reset the upset link.

Upon seeing the upset, as an error in the data stream (probably a parity error, possibly an invalid token sequence), B takes the following actions:

- Ignores anything from A until the link has re-synchronized;
- Reports the error to A by inserting a suitable error-report token in the (not upset) data stream from B to A – using ESC-EEP, for example;
- Resets its flow control credit to zero and hence temporarily stops sending data, pending receipt of flow control credit after the link has re-synchronized (the upset may have introduced or removed a flow control token and we are uncertain how much credit is available until it is reset);
- Waits for a re-synchronizing sequence to appear from A to B, followed by flow-control tokens to restore B’s flow control credit (if no such re-synchronizing sequence is seen within a reasonable time we assume a more serious error and revert to the currently defined period-of-silence recovery mechanism);
- Sends flow control tokens to A to reset its flow control credit.

Meanwhile A continues normally until it sees the error-report token when it:

- Discards the rest of the current packet whose delivery is uncertain;
- Inserts a re-synchronization sequence (“111111111110” would be a suitable , short, sequence, not being a valid sequence in normal operation of the link);
- Resets its flow control credit to zero to await new flow control tokens from B.

If either end sees a timeout error it should use the currently defined error recovery procedure.

It is not necessary to wait on a period of silence, nor to reset the link speed to 10Mb/s for the restart. Recovery can take place at the link’s running speed and in just a few bits. Recovery time will be in the order of 30 bit times plus latency in the control flows at each end, one to three orders of magnitude faster than the normal mechanism. Data in the non-upset direction will be delayed by a few bit times but there is no need to discard the packet in transit at the time of the loss – only the packet in the direction of the upset will be lost.

This process is interoperable with the currently defined recovery mechanism, defaulting to that mechanism in a mixed implementation.

6. CONCLUSION

SpaceWire has many features to offer and has seen acceptance for payload data transfer. Command and control presents additional requirements that SpaceWire has not been able to meet satisfactorily. Merging the command and control network with the payload network would result in significant mass reduction and is a worthwhile goal.

We have shown three improvements that may be implemented in a SpaceWire network that improve its behaviour by increasing determinacy and allowing command and control to take priority over payload data. All the changes described can be implemented in the network infrastructure and do not require re-design of any existing end nodes.

Time-code jitter can be very significantly reduced with only a small change in the time-code transmitter. The receiver need not be altered, so long as it behaves deterministically. Jitter performance of 10ns per link is easily achieved and significantly smaller jitters are possible. It is possible to extend this benefit to produce accurately synchronized time signals at all nodes in a system.

Pre-emptive priority routing between switches allows highly predictable, low latency, delivery of critical data in support of real-time requirements. A fault tolerant SpaceWire network can be made to deliver a highly reliable service with little delay or variance for critical command and control functions, and also to provide “background” delivery of lower priority payload data. Multiple priority levels can be provided so that, for example, payload data can be prioritized but remain secondary to command and control. This can be achieved purely in the routing switches, no changes are needed in instruments or interfaces.

System synchronization with (low-jitter) time codes combined with pre-emptive priority routing can be used to implement a very accurate isochronous transfer mechanism. It has the advantage over other isochronous implementations that spare capacity produced when isochronous packets do not fully fill the time slot allocated is automatically used for other (lower-priority) traffic.

Data loss and variance can be reduced in the case of transient upsets by shortening link recovery time – improvements of one to three orders of magnitude can be achieved.

These improvements are compatible and inter-operable with existing SpaceWire interfaces and do not require re-design of anything beyond the routing switches.

The improved systems that can be built with these techniques have implications for the whole mission, not just payload but also command and control. The benefits of a more widely applicable data network include mass reduction and cost reduction from increased re-use of common SpaceWire components.

7. REFERENCES

[1] ISO/IEC 14575 Heterogeneous InterConnect

[2] IEEE Std 1355-1995 IEEE Standard for Heterogeneous InterConnect (HIC) (Low-Cost, Low-Latency Scalable Serial Interconnect for Parallel System Construction)

[3] ECSS-E-50-12A(24January2003) SpaceWire – Links, nodes, routers and networks