SpaceWire Handbook

DRAFT - SVN version 60 - April 10, 2013

4Links Limited
Suite E U 2
Bletchley Park
Milton Keynes, MK3 6EB
United Kingdom

Foreword

This Draft Handbook is 4Links Limited's initial contribution to the forthcoming ECSS SpaceWire Handbook.

Change Log

Date	Version
29th June 2012	First draft issue
24th September 2012	Second draft issue
10th April 2013	Third draft issue

Table of Contents

1. Introduction

1.1. Purpose

This ECSS handbook is intended to help implementers and users of data handling systems who are basing designs on the ECSS E-50 series of standards. The handbook provides an overview of the E-50 standards and related CCSDS Recommended Standards and describes how the individual standards may be used together to form a coherent set of communications protocols. It also evaluates issues which could not be discussed in the Standards documents themselves, and provides guidance on option selection and implementation choices.

1.2. Scope

This handbook provides guidance to the ECSS E-50 series of standards including related CCSDS Recommendations. The information provided is informative and is a guide to best practice; it is not binding on implementers. The information contained in this handbook is not part of the ECSS Standards.

1.3. Acknowledgements

This Handbook has been authored and agreed upon by the following persons:

A.N. Person, Organisation (physical layer)

A.N. Other, Second Organisation (routing switches)

This Handbook has been prepared based on volunteer contributions of the authors.

1.4. Document structure

This document is divided into sections and annexes as follows:

- Section 1, “Introduction”, (this section) provides intentional and administrative information.
- Section 2, “Terms, definitions and abbreviations”, provides the definition and abbreviations of the terms used in the present document.
- Section 3, “References”, provides a list of the E-50 series of ESA SpaceWire standards.
- Section 4, “Introduction to SpaceWire”, provides an introduction to SpaceWire itself.
- Section 5, “ECSS-E-ST-50-12C and related standards”, provides detailed information on each of the individual ECSS and CCSDS standards covered by the handbook.
- Section 6, “Specific Implementation Topics”, addresses individual technical topics related to the ECSS E-50 standards.
- Section 7, “Supporting Components”, provides a summary of supporting components and products.
- Section 8, “Bibliography”, lists the other references included in this Handbook.

2. Terms, definitions and abbreviations

2.1. Terms and definitions from other documents

For the purpose of this document, the terms and definitions from ECSS-E-ST-50-12C apply, and are reproduced here.

2.1.1. acknowledge

indication that a message has been received successfully by its intended destination

2.1.2. binder

layer of tape wrapped around one or more cables to keep them together in a fixed position.

NOTE: The tape is usually PTFE and is wrapped in an overlapping spiral along the length of the cables to bind.

2.1.3. bit error rate

ratio of the number of bits received in error to the total number of bits sent across a link

2.1.4. byte

eight bits

2.1.5. cargo

data to encapsulate in packets and transfer from a source to a destination

2.1.6. character

control character or data character

2.1.7. character level

protocol level that deals with the encoding of data and control characters into a bit-stream

2.1.8. coding

translation from one set of bits to another new set of bits

2.1.9. content addressable memory

memory array which is accessed by searching for a match between an input data value in the contents of the memory array, where the output from the memory array is the index of the location that holds the searched for value

2.1.10. control character

character that is used to pass control information across a link. NOTE: Control characters include the L-Chars (ESC and FCT) and the end of packet markers (EOP and EEP).

2.1.11. control code

sequence of two control characters: NULL (ESC + FCT) which is used to keep a link active, and Time-Code (ESC + data character) which is used to distribute system time information over a SpaceWire network

2.1.12. data character

data byte encoded ready for transfer across a link

2.1.13. data rate

rate at which the application data is transferred across a link

2.1.14. data signalling rate

rate at which the bits constituting control and data characters are transferred across a link

2.1.15. data-strobe

encoding scheme in which a sequence of data bits and clock is encoded as the original data bit sequence, together with another bit sequence (strobe) which changes state whenever the data bit sequence does not

2.1.16. decoding

act of translating an encoded set of bits to the original set of bits prior to coding

2.1.17. de-serialization

transformation of a serial bit stream into a sequence of control or data characters

2.1.18. destination

node or unit that a packet is being sent to

2.1.19. destination address

route to be taken by a packet in moving from source to destination (path address) or an identifier specifying the destination (logical address)

2.1.20. destination list

list of destination identifiers which forms the destination address of a packet

2.1.21. destination identifier

address, or partial address, of the packet destination

2.1.22. driver

electronic circuit design to transmit signals across a particular transmission medium

2.1.23. end of packet marker

control character which indicates the end of a packet

2.1.24. error recovery scheme

method for handling errors detected within a SpaceWire link

2.1.25. exchange level

protocol level that defines the mechanisms for link initialization, link flow control, link error detection and link error recovery

2.1.26. filler

cylindrical piece of PTFE used to fill the gap between insulated wires or cables being grouped together and formed into a larger cable, which enhances the structure of the cable helping to keep the constituent wires in a fixed position relative to one another

2.1.27. flow control token (FCT)

control character used to manage the flow of data across a link, indicating that there is space for 8 more normal-characters in the receiver buffer

2.1.28. host receive buffer

buffer within a host system for receiving data from a link interface

2.1.29. host system

system that a link interface is connected to. NOTE: It can be, for example, a computer, sensor or memory unit and need not contain a computer or processor.

2.1.30. host transmit buffer

buffer within a host system for holding data prior to transmission through a link interface

2.1.31. input port

receive side of a link interface on a routing switch

2.1.32. jitter

random errors in the timing of a signal

2.1.33. lay-length

number of twists per foot expressed as the length between one complete turn of a single end in the cable

2.1.34. link

bidirectional connection of one unit to another unit for passing data and control information

2.1.35. link-character

control character used to manage the flow of data across a link.

NOTE: In this Standard, only ESC and FCT are used as link characters. NULL is formed from a pair of link-characters (ESC followed by FCT).

2.1.36. link destination

end of the link that is receiving a particular set of data or control information

2.1.37. link interface

SpaceWire interface comprising a transmitter which takes data from a host system and transmits it across a SpaceWire link, and a receiver which accepts data from a SpaceWire link and passes it to the host system

2.1.38. link receiver

receiver at one end of a link

2.1.39. link source

end of the link that is sending a particular set of data or control information

2.1.40. link transmitter

transmitter at one end of a link

2.1.41. logical address

data character at the start of a packet, which identifies the destination for the packet

2.1.42. low voltage differential signalling

particular form of differential signalling using low voltage swing signals

2.1.43. Mb/s

1 000 000 bits per second

2.1.44. network

set of units connected together via links and routing switches

2.1.45. network level

protocol level that defines the SpaceWire network routers and defines how packets of data are transferred across the network from source node to destination node

2.1.46. node

source or destination of a packet, which can be a processor, memory unit, sensor, EGSE or some other unit connected to a SpaceWire network

2.1.47. normal-character

data character or control character (EOP or EEP) that is passed from the exchange level to the packet level

2.1.48. NULL

token sent to keep the data link active when there are no data or control characters to send

2.1.49. output port

transmit side of a link interface on a routing switch

2.1.50. packet

sequence of normal-characters comprising a destination address, packet cargo and an end of packet marker

2.1.51. packet level

protocol level that defines how data is organized in packets ready for transfer across a link or network

2.1.52. packet cargo

data to transfer from a source to a destination

2.1.53. path address

series of one or more data characters at the start of a packet which define the route to be taken across a SpaceWire network

2.1.54. physical level

protocol level that specifies the physical interconnection medium, e.g. cables and connectors

2.1.55. pseudo-ECL (PECL)

emitter-coupled logic (ECL) referenced to +5 V

2.1.56. receiver

electronic circuit designed to receive signals sent across a particular transmission medium

2.1.57. router

routing switch

2.1.58. routing switch

switch connecting several links that routes packets from one link to another where the destination address of each packet by the switch is used to determine which link a packet is sent out on

2.1.59. serialization

transformation of a sequence of control or data characters into a serial bit stream

2.1.60. signal

measurable quantity that varies with time to transfer information by propagating along a transmission medium

2.1.61. signal level

protocol level which defines the electrical signals used for SpaceWire together with the data-strobe encoding and signal timing

2.1.62. skew

difference in time between the edges of two signals which should ideally be concurrent

2.1.63. source

node or unit sending a packet

2.1.64. Time-Code

code used to distribute system time over a SpaceWire network, which comprises ESC followed by a single data character holding six bits of the system time and two reserved bits

2.1.65. transmission medium

medium over which data is transferred e.g. screened twisted pair cables

2.1.66. transmitter

electronic circuit designed to transmit signals across a particular transmission medium

2.1.67. unit

box, board or subsystem, that can have one or more SpaceWire interfaces

2.2. Terms specific to the present handbook

2.2.1. SpaceWire End-Point

A SpaceWire End-Point is a hardware or software process that connects to one or more SpaceWire packet-level processes; a SpaceWire packet producer or consumer.

2.2.2. SpaceWire Node

A SpaceWire Node is the source or destination of a SpaceWire packet stream (i.e. a stream of bytes or time-codes), which comprises one or more end-points. A node can therefore be a processor, a memory unit, a sensor, an EGSE, or some other unit connected to a SpaceWire network.

2.2.3. SpaceWire Unit

A SpaceWire Unit is a box, board or subsystem, that can contain one or more SpaceWire nodes.

2.3. Abbreviated terms

The following abbreviated terms are used within this document:

Abbreviation	Meaning
AFRL	Air Force Research Laboratory
ANSI	American National Standards Institute
ASIC	application-specific integrated circuit
AWG	American wire gauge
CAN	controller-area network
CCSDS	Consultative Committee for Space Data Systems

Abbreviation	Meaning
CERN	European Organization for Nuclear Research
CNES	Centre national d'Ã©tudes spatiales
CODEC	coder / decoder
CRC	cyclic redundancy code
CTL	Computer Technology Limited
DC	direct current
DS	data-strobe
DMA	direct memory access
ECSS	European Cooperation for Space Standardization
EDAC	error-detecting and correcting [code]
EEP	error-end-of-packet
EGSE	electrical ground support equipment
EIA	Electronic Industries Alliance
EMC	electromagnetic compatibility
EMI	electromagnetic interference
EOP	end-of-packet
ESA	European Space Agency
ESCC	European Space Components Coordination
FCT	flow-control token
FDIR	failure detection, isolation and recovery
FIFO	first-in-first-out [buffer]
GOES-R	Geostationary Operational Environmental Satellite - R Series
HS	High-Speed [IEEE 1355 encoding]
IEEE	Institution of Electrical and Electronic Engineers
IP	Internet Protocol
ISO	International Standards Organisation
JAXA	Japan Aerospace Exploration Agency
LVDS	low voltage differential signalling
MDM	Micro-D [connector]
MTU	maximum transmission unit
NASA	National Aeronautics and Space Administration
PCB	printed circuit board
PECL	pseudo-ECL
PHY	physical layer [interface]
PTFE	Polytetrafluoroethylene
RAM	random-access memory
RMAP	Remote Memory Access Protocol
RMW	read / modify / write
DSS	Dornier Satellitensysteme GmbH
SE	single-ended
SSTL	Surrey Sattelite Technology Limited
TCP	Transmission Control Protocol
TDR	Time-domain reflectometry
TIA	Telecommunications Industry Association
TRAM	Transputer Module

Abbreviation	Meaning
TS	Three-of-six [IEEE 1355 encoding]
UART	Universal Asynchronous Receiver / Transmitter
USB	Universal Serial Bus
XOR	exclusive-or

3. References

This document is the handbook corresponding to the SpaceWire standard ECSS-E-ST-50-12C and the subsidiary SpaceWire standards listed below.

The following normative documents are referenced in this text or provide additional information useful for the reader.

ECSS-E-ST-50-12C	SpaceWire - Links, nodes, routers and networks
ECSS-E-ST-50-51C	SpaceWire protocol identification
ECSS-E-ST-50-52C	SpaceWire - Remote memory access protocol
ECSS-E-ST-50-53C	SpaceWire - CCSDS packet transfer protocol

These standards documents are available at **`http://www.spacewire.esa.int`**.

Also, the following normative non-ESA documents are referenced:

ANSI / TIA / EIA-644-A-2001	“Electrical Characteristics of Low Voltage Differential Signaling (LVDS) Interface Circuits”, Telecommunications Industry Association, February 2001
IEEE Standard for Heterogeneous Interconnect	“Low-Cost, Low-Latency Scalable Serial Interconnect for Parallel System Construction”, IEEE Standard 1355-1995, IEEE Computer Society, June 1996.

4. Introduction to SpaceWire

SpaceWire is a high-performance communications network that is particularly suited for use within a spacecraft. The network utilises point-to-point serial links, running at 2 Mbps to 200 Mbps or faster, and an arbitrary topology of wormhole routing switches to minimise latency. LVDS signal levels are employed. Timing information is transmitted with the data on each link, so the local oscillators in each end-point can have different frequencies and phases. Data is transferred in packets, although the packet length is unspecified to provide flexibility to the system designer. Several addressing schemes provide flexibility for nominal operation, and low-level link error detection supports failure detection, isolation and recovery.

SpaceWire provides compatible interfaces amongst spacecraft subsystems, allowing for easier re-use, faster development timescales and simpler EGSE arrangements.

The SpaceWire family of protocols has been developed by a worldwide community, under the overview of the European Space Agency's SpaceWire Working Group, specifically for use in a space environment. The open nature of this design process is one reason why SpaceWire has been adopted so widely - on over 100 space missions to date.

This section of the Handbook outlines much of the background to SpaceWire; most of the technical topics are then covered in much greater depth later in the document. Much of this background material has been drawn from [Walker 2003].

4.1. Background and History

SpaceWire is a data communication technology that was standardised by ECSS (the European Cooperation for Space Standardization) in January 2003 and re-issued as ECSS-E-ST-50-12C on 31 July 2008 [see section 3, "References", for information about the ESA SpaceWire standards documents]. Early versions of SpaceWire are flying on several missions, and it is planned for use on many missions worldwide. As a simple interface that can be used for a wide variety of different purposes, SpaceWire appears to offer an enabling technology for a "Building Block Architecture" such as described in NASA's Vision for Space Exploration [NASA 2004], for rapid deployment, such as DoD's PnPSat [McNutt 2009], and for cost reduction as a result of architecture and subsystem re-use as practised by ESA. While standardised in the 21st century, SpaceWire has evolved over many years, following a few key principles and concepts that are the foundation of its wide application and use.

The following sections describe the evolution of SpaceWire and draw from it several key principles and concepts.

4.1.1. 1960 - A Modular Computer

In the 1960s, a computer would be built from several different boxes, such as processor, memory, disc controller and communications controller. One way to connect the boxes together was to use a simple standard interface between any of these boxes, so that they could each access the others independently of each other.

An example of this was the interface of the CTL Modular One computer. The key concepts of this standard interface were:

- Keep the bus inside each box, so that the whole system did not share a single bus;
- Use an asynchronous interface, so that each box could run at its optimum speed and there was no need for global synchronisation;
- Use a symmetrical interface, so that any box could be connected to any box;
- Have flow-control across the interface so that data was not lost even if buffers were full (but this may have resulted in reduced performance if a communication was blocked).

These key principles resulted in a number of benefits:

- The system was scalable, so that systems could be built with any number of processors, memories, and peripherals;
- There were few constraints on the topology of the system, so that systems could be built with any shape as well as any size;
- Multiple units could be configured for redundancy and fault-tolerance;
- The system was truly modular, in that a huge variety of systems could be built from comparatively small number of building blocks.

While the Modular One computer systems built with these interfaces were never used in space, they were used by the European Space Agency for Ground Support and Operations.

Interestingly, this concept of modularity was pre-dated by the Ferranti Pegasus in the mid 1950s [Ross 2012].

4.1.2. 1980 - System on Chip, Serial Interfaces

During the 1980s, it became clear that it would be possible to put a complete computer on a single silicon chip, including processor, memory, and interfaces. One of the first examples of this was the INMOS transputer [INMOS 1985]. This had a conventional external memory bus, similar to other microprocessors, but it also had four serial interfaces or “links” that inherited the key principles of the Modular One interfaces.

Overall, the four links, including the physical layer interface, all the serialising and de-serialising (SERDES) and DMA logic for each direction for each link, took up about the same space as the fixed-point processor. By comparison the on-chip RAM, the floating-point processor and the memory interface (including all its pins) each took up significantly more chip area.

At the time the transputer was introduced, a 10 Mb/s Ethernet interface needed a chip-set of three chips, whereas a serial link needed around 2% of a single chip on the transputer and its DMA engine another 2%.

Performance of the early transputer links was modest, but at 20 Mbits/s in each direction (full-duplex) a single link was well over twice the performance of an Ethernet connection. With the four links per transputer running full-duplex at 20 Mbits/s, total serial throughput was 160 Mbits/s per transputer.

As well as keeping the key principles of the Modular One interfaces, the transputer links added the following:

- They were serial interfaces, to reduce pin count and to simplify connections between chips;
- They used DMA to access the transputer's memory, with very low processor overhead per packet.

4.1.3. Transputer serial links in space

The space industry recognised the potential of the transputer and its links for building fault-tolerant networks on-board spacecraft. Missions included the Cluster group from ESA, many satellites from SSTL and from CNES, and the SOHO collaboration between ESA and NASA. In fact, the transputers used in these missions were not specifically designed as Rad-Hard, but they were from batches selected for radiation tolerance and designed into fault-tolerant networks.

The SOHO satellite continues to send back images of solar corona discharges.

4.1.4. Modularity

In the early days of the development of the transputer, it was found that a useful way to explain the ideas was to compare the transputer with toy building blocks such as Lego™ and K'Nex™. These use a very simple standard interface that can be used to connect a wide variety of different building blocks, in order to build an even wider variety of constructions. The serial links of the transputer were such a simple and easily usable interface, and they encourage modularity.

The opportunity was taken to propose a standard Transputer Module, or TRAM, which used the serial links as their interface. These were printed circuit boards about half the size of a credit card, with just sixteen pins. In effect they were 16-pin Dual-Inline-Packages (DIPs) with 3.3 inches between the pins instead of the conventional 0.3 inches between pins. These modules were very popular and were made by INMOS and by other companies in Europe and the USA.

4.1.5. 1990+ Transputer links to IEEE-1355

Towards the end of the 1980s, a new generation of the transputer was planned, taking the links to 100 Mbits/s and adding some important new principles:

- Adding a minimalist packet protocol, consistent with the general move towards packet communication and switching;
- Adding a network protocol so that the packets could be routed through a network of routing switches;
- Adding virtual channels, so that a variety of different communications can share the same physical links.

The TRAM standard had been popular as a way to construct systems inside a box. The new 200 Mb/s links provided the opportunity to create a standard for connections between boxes, and an internal standard was proposed in the late 1980s. Colleagues at INMOS, together with

other contributors in Europe, took this forward to create the IEEE-1355 standard. To keep the standard simple, the network and virtual channel protocols were left out, but all the previous principles that have been outlined were included in IEEE-1355.

Notable among the contributors was CERN, who built a large test system with 1024 links, over which they ran a soak test for three months, logging 10^{17} bits transferred without a data error on a link [At one point during the test, a thunderstorm upset the computer and network that were controlling the test, but there was no failure on the links] [Haas 1998].

Also among the contributors, even in the early 1990s, were Dornier SatellitenSysteme (DSS, subsequently EADS-Astrium, in Munich).

The IEEE-1355 standard was confirmed in 1995, after which the European Space Agency and a number of other organisations in the space industry joined the activity.

For what at the time were probably correct commercial and political decisions, the new transputer and the IEEE-1355 standard were abandoned by the company that had taken over INMOS. The standard was used by Canon, who needed to adapt some aspects of the standard for a networking application. A number of small adaptations were also required for Space and so a new standardisation activity was launched by the European Space Agency. This activity became SpaceWire.

4.1.6. IEEE-1355 and early SpaceWire in Space

During the development of the SpaceWire standard, there was clearly an interest in using the IEEE-1355 standard and in drafts of the SpaceWire standard for space applications. EADS-Astrium Munich commissioned chips that were available in a rad-hard version, and these chips are flying on Rosetta, on Mars Express and on Venus Express from Europe, on Solar Dynamics Observatory and STEREO for NASA, and on the commercial Broadband Global Area Network satellites for Inmarsat. As well as the Data-Strobe (DS) version of IEEE-1355 that has carried through to SpaceWire, Rosetta is also carrying the “Three of Six” (TS) version of IEEE-1355, which has the benefit of galvanic isolation at a slight penalty in available bandwidth. Early versions of SpaceWire are also flying on SWIFT and on other missions classified for commercial or other reasons.

4.2. The current SpaceWire Standard

Compared with IEEE-1355, the SpaceWire standard:

- Evolves the DS (Data/Strobe) Physical alternative of IEEE-1355
- Corrects an initialisation bug in IEEE-1355
- Removes some ambiguities in IEEE-1355
- Removes the End of Message token (used in IEEE-1355 to implement virtual channels), using it instead as an Error End of Packet token
- Removes the TS (galvanically isolated, flying in Rosetta) physical alternative

- Removes the HS (Gbit/s) physical alternative
- Removes the SE (single-ended, chip-to-chip) physical alternative
- Uses LVDS rather than PECL, for lower power
- Uses space qualified connectors and cable
- Includes a simple Network Layer protocol
- Adds Time Code distribution

Apart from these changes, the SpaceWire standard embodies the key principles that have been outlined:

- Bus kept inside each unit, not over entire system;
- Serial interface;
- Asynchronous interface;
- Symmetrical interface;
- Flow-control across the interface;
- Minimalist packet protocol;

These qualities, as before, bring the benefits of scalability, topological flexibility, fault-tolerance and modularity

The standard is cleanly layered, with minimal overlap or interaction between the levels.

The levels that are defined are:

- the Physical level: two signal pairs in each direction, PCB traces, connector and cable;
- the Signal level: LVDS including failsafe, terminations, Data-Strobe signal encoding on the two pairs, signalling rate, skew and jitter;
- the Character level: Data characters, Control characters, Time Codes, parity, character(s) to be sent at initialisation or after error, host interface encoding;
- the Exchange level: Normal Characters (that are passed through the network) and Link Characters (that are local to a single physical connection), flow control, clock recovery, initialisation state machine, errors and error recovery, Time Code distribution;
- the Packet level: destination address, cargo, end-of-packet markers; and
- the Network level: Wormhole routing, path addressing, logical addressing, header deletion, group adaptive routing, how to do broadcast or multicast, network errors and recovery

It is useful to summarise a few of the main characteristics, particularly those that are different from some other networking standards:

- Data-Strobe encoding

- Low-level flow-control
- Packets
- Packet routing
- Time Codes and their distribution

The following sections highlight some of the main features of SpaceWire. Clause 5, “ECSS-E-ST-50-12C and related standards”, discusses each of these topics in greater depth.

4.2.1. Data-Strobe encoding

There is a need in any communication system for a means of recovering the clock from the received signals. In long-distance communication, this tends to be with a phase-locked loop per channel, which would be possible for space but which needs analog circuitry that is undesirable in space electronics. An alternative is to send a clock signal on a separate wire, but this has tight demands on skew between the signals. SpaceWire uses a Strobe signal on a separate wire, which is Gray-coded with the signal wire so that for each bit transmitted, there is a transition on either the Data or the Strobe signal. This still needs the skew to be controlled, but is more relaxed in this respect than separate clock and data. The technique was originated in the INMOS transputer’s DS-Links, then standardised in IEEE-1355, and was subsequently adopted by IEEE 1394/FireWire.

4.2.2. Low-level flow-control

Flow-control is often seen as a high-level protocol, and indeed for long-distance communication needs to be so. The lack of flow-control at a low level, however, requires buffers large enough that they (almost) never overflow. SpaceWire permits low-cost circuits with small buffers, and the flow-control ensures that data is preserved and that the buffers never overflow. Having larger buffers than the minimum permitted improves overall network performance, but the flow-control allows implementations of SpaceWire that can have little more logic and buffering than conventional RS232/422 UARTs, even though SpaceWire runs orders of magnitude faster than these UARTs.

4.2.3. Packets

SpaceWire uses a minimalist packet format, with data bytes and a packet terminator. Often, the first data byte(s) are interpreted as a routing header. For a point-to-point connection (i.e. not via a routing switch), the header can be zero length; for a routed packet, the header is a destination address that can be as long as necessary. The cargo can similarly be as long as necessary, and no limit is defined in the standard. In practice, most systems will benefit from imposing a form of Maximum Transfer Unit (MTU) to prevent a long packet blocking other traffic in the network. The packet termination is a single control character, either End-of-Packet (EoP) or Error- End-of-Packet (EEP).

After the standard was issued, it was agreed to include a protocol identifier (PID) as part of the header, between the destination address and the cargo. As in other standards such as Ethernet and Internet Protocol, the PID allows a variety of different higher-level protocols to inter-operate on the SpaceWire network without interfering with each other.

The minimalist packet protocol of SpaceWire provides what is absolutely necessary and no more. If extra information is required in a header, such as the source of the packet, a checksum, or a protocol to be encapsulated on SpaceWire, these can all be added at a higher level. All that is added, however, needs to be generated and checked for each packet, which can impose substantial delays in processing each packet. The simple raw SpaceWire packets provide a very efficient communication system with very low processing overheads as well as low overheads on packets. The packet overhead can be as low as four extra bits for each end-of-packet marker on a point-to-point SpaceWire link with no routing header bytes.

4.2.4. Packet Routing

SpaceWire can be used with or without routing switches, and satellites can include point-to-point connections as well as a network (or networks) with routing-switches.

When using routing switches, SpaceWire packet switching uses “Wormhole Routing” so that the front of a packet can have left the routing switch before the end of the packet has arrived.

The SpaceWire standard *requires* that routing switches to provide what the standard calls Path Addressing, and *permits* them to provide what it calls Logical Addressing. In each case, the first data character of a packet seen by the routing switch is used as a routing header to determine which output port of the routing switch the packet is routed to.

In Path Addressing, values of the first data character from 1 to 31 result in the packet being output to port 1 to 31 respectively. The special value of zero results in the packet being used internally by the configuration/management port of the routing switch. After the character has been used to address a particular output port, the character is no longer required and so is deleted.

In Logical Addressing, values of the first data character of a packet are used to index a look-up table to determine the output port. In this case the character is not normally deleted, as the same character can be used in several routing switches to steer a route through the network. For small networks such as tend to be used on satellites, logical addressing can provide an exceptionally low overhead for routing the packets.

The use of routing tables makes the router design considerably more complex and adds a critical requirement to protect the table contents against corruption by radiation effects.

4.2.5. Time Codes and their distribution

It is useful for all the subsystems on a satellite to have a reasonably consistent view of time, and SpaceWire provides a means of distributing such a consistent view. Time Codes are special sequences of characters which take priority over the normal data in a packet and are distributed to all nodes in the SpaceWire network. A small amount of jitter is normally introduced, both in the generation and distribution of Time Codes, resulting in a few microseconds variation in the view of time from different nodes in the network. A scheme has been proposed that is completely compatible and interoperable with the standard, where the jitter in Time Code generation and distribution can be reduced to a few tens of nanoseconds [Cook 2003].

4.2.6. SpaceWire in comparison with other communication techniques

SpaceWire is a recent newcomer to the world of computer networks and avionics data bus systems. This section of the Handbook introduces several competitors of SpaceWire.

The Onboard Computer and Data Handling Section at ESA provides a good comparison of computer and interconnect techniques on its website at
`http://www.esa.int/TEC/OBCDH/index.html`.

4.2.6.1. Asynchronous and Synchronous Serial Communications

Asynchronous UART connections require the clocks at each end to be controlled to within a few percent. Typically, oversampling of the received signal is performed at sixteen times the expected bit rate. After detecting a start bit transition, the remaining eight data bits are sampled using the predetermined clock frequency. Thus 8.5 bit-times elapse from the initial transition until the middle of the last bit period, and thus the receive clock must be accurate to just under 5% of the clock rate, even before allowing for jitter on the received signal edges. RS-422 differential signalling can provide data rates of 115 200 baud.

Using RS-422 with synchronous clocking, speeds of up to 10 Mbps are possible. In contrast, each bit of a SpaceWire data stream is self-clocked, and a SpaceWire receiver does not directly rely on a clock in the receiver.

4.2.6.2. MIL-STD-1553B

The MIL-STD-1553B standard was published in 1978. It provides a serial bus for avionics applications, and it has also found favour for spacecraft onboard data handling.

MIL-STD-1553B specifies a shared bus that comprises a number of redundant twin-ax coaxial cables. One Bus Controller and 31 Remote Terminals may be connected to each of these cables using transformer coupling. If the bus master fails, another device can take over its role. The signals on each cable are Manchester-encoded, half-duplex and clocked at a rate of 1 Mbps. The voltage swings at the Remote Terminals are in the range 1.4 to 20.0 V.

Data words on the MIL-STD-1553B bus are 16-bits in length, and groups of words may be transmitted without gaps to make messages. Between messages, there is a minimum gap of 4 µs. There is a rich command word structure for use by the Bus Controller. The Bus Controller initiates each transmission by a Remote Terminal, so it manages all of the scheduling on the bus.

Store-and-forward repeaters can be used between separate MIL-STD-1553B buses to build larger hierarchies of nodes.

Compared with SpaceWire, the data rate of MIL-STD-1553B is very slow. Also, the common bus nature of the architecture forces all of the Remote Terminals to share the same bus bandwidth, rather than using separate point-to-point links in parallel. The use of store-and-forward repeaters to increase the number of active notes beyond 31 adds considerable extra latency.

Despite MIL-STD-1553B having a shared bus topology, its transformer-coupling and resistive current limiting limits the effects of faults. SpaceWire, on the other hand, provides point-to-point paths between nodes and routers that cannot pick up interference from other nodes. Providing redundant routers, redundant links and the ability to re-route messages through the

network makes fault isolation in SpaceWire more effective.

For further details on MIL-STD-1553B, see [MIL-STD-1553B], [Klar 2008] and [Larsen 2011].

4.2.6.3. CAN bus

The CAN bus was originally proposed for the automotive industry, although it has since spread into lower-cost space applications. The ISO 11898 CAN standard provides a specification for 125 kbps and 1 Mbps physical and link-layer implementations (ISO 11898-3 and 11898-2, respectively). There is also a time-triggered CAN specification (ISO 11898-4). There is no Network Layer or Transport Layer in the CAN protocol stack. An application-layer protocol ("CANopen") and several specialist higher-level protocols have been proposed by the ECSS for space use.

Like MIL-STD-1553B, CAN uses differential signalling on a bus. Unlike MIL-STD-1553B, CAN solely uses resistive interconnections to the bus, and a differential swing of just 2 V. Onto this bus, the CAN protocol introduces a small number of fixed-format data frames. A substantial part of these data frames is set aside for the Arbitration & Control Field which allows nodes on the bus to compete for the right to transmit. Unlike the collision-based backoff mechanism of Ethernet, the CAN scheme is non-destructive, with the higher-priority frame continuing to be sent while the lower-priority one gives way.

The CAN standard is even less rich than SpaceWire, having no Network layer in its architecture. Its physical layer is very simple, but one node's non-adherence to the arbitration scheme could be very damaging to the whole bus, so reliability is not its strong point. Galvanic isolation is not built-in to the CAN standard, unlike MIL-STD-1553B, but isolation at each node is possible with active silicon devices. Redundancy management has also been proposed for CAN by the ECSS-E-ST-50-15 Working Group.

For details of the use of CAN within ESA, see [Furano 2011].

4.2.6.4. Ethernet

Ethernet is only specified for four common speeds - 10 Mbps, 100 Mbps, 1 Gbps and 10 Gbps, each with a different cabling requirement and signal coding arrangement. Each has a data extraction scheme that requires considerably more signal processing effort than SpaceWire, consuming more logic and consequently more power in operation.

The Ethernet data link layer has a minimum packet size of 64 octets, including the 32-bit CRC but not the 8-octet start-of-frame preamble or the 12-octet inter-frame gap, and is thus less suitable for rapid-fire small command and response activities than SpaceWire. In addition, its 48-bit hardware addresses are overkill for use on a spacecraft. Ethernet does provide a packet-wide cyclic redundancy check (CRC) that can signal errors once the whole of a packet has been received. The maximum packet size is 1518 octets, which requires data streams longer than this to be segmented.

4.2.6.5. TCP/IP Networking

A switched network, using Ethernet and protocols such as TCP and IP, has similarities to a SpaceWire network that contains routers. However, TCP/IP only supports logical addressing, and there is only a source route-setting scheme at the IP transport layer. Ethernet routers are often store-and-forward in nature, which cause higher latencies to transmitted packets than SpaceWire, which uses wormhole routing.

By allowing broadcast communications, Ethernet will circulate a packet forever in a network that contains cycles, (subject to the time-to-live controls in higher-level protocols, of course). SpaceWire networks, on the other hand, may contain cyclic connections, and avoid the need to reason about the whole network topology when the routing structure is being changed during failure recovery operations. In Ethernet, re-analysing the network using a spanning-tree algorithm typically causes the whole of the network to become idle during re-configuration.

Conventional TCP/IP networks allow multiple links between two routers to be used in parallel - 'link aggregation'. This is equivalent to the group-adaptive routing facility of SpaceWire, although the latter provides more control over non-nominal paths which will aid fault investigations.

Finally, the underlying IP protocol only provides a 'best effort' packet delivery mechanism, which can destroy packets if there is congestion in the network. In contrast, SpaceWire has a built-in flow control mechanism that blocks paths through the network if traffic is not consumed, which is easier to reason about, but can impact performance on other links.

4.2.6.6. Optical Fibre

ECSS-E-ST-50-12C SpaceWire is not specified for optical fibre connections. There are many reasons why this was not possible:

- flow-control uses up to 56 bytes of credit - insufficient for a high-speed line.
- the bit-error rate of optical fibre is likely to be of a similar magnitude to what the single-bit parity scheme of conventional SpaceWire can handle, so is probably acceptable for applications to deal with by themselves.
- SpaceWire signalling is not DC-balanced, so a different coding scheme is required.

The forthcoming SpaceFibre standard addresses many of these issues, such as providing an error-detecting and retry layer.

4.2.7. Where SpaceWire is being used

SpaceWire is being used on a wide variety of different missions, throughout the world. The European Space Agency plans to use SpaceWire for most, if not all, of its future missions [Parkes 2011]. A number of national missions, such as Taiwan's Argo 5 satellite, are using SpaceWire. Key US missions are the James Webb Space Telescope, the Lunar Reconnaissance Orbiter, and GOES-R.

The information given here is given in good faith to the best of 4Links' knowledge and belief. Many, but not all, of the missions are mentioned in [Parkes 2011]. In addition, estimates are included here of the types of mission and the extent to which SpaceWire is used in each

mission, and these are necessarily rather subjective. Some of the information is unpublished and has been gained from informal discussions at public meetings such as the SpaceWire Working Group and at conferences. 4Links welcomes additions, comments and corrections to this list.

The following table lists missions that have used IEEE 1355 or SpaceWire.

The “Mission Type” and “Extent of SpaceWire” columns represent

4.3. The Future of SpaceWire

4.3.1. How the SpaceWire Standards will Develop

The SpaceWire Working Group has already defined the Protocol Identifier so that multiple protocols can inter-operate on a SpaceWire network, and has defined a Remote Memory Access Protocol (RMAP). A number of other protocols are being defined, particularly to encapsulate CCSDS and IP packets in SpaceWire, and more such encapsulation can be expected. A new protocol for SpaceWire has been developed in the US that is similar to a cut-down TCP.

There are several examples of SpaceWire running at 400 Mbits/s or faster, whereas most current uses are between 10 Mbits/s and 200 Mbits/s. The current rad-hard silicon imposes limits on the speeds that can be used, but new ASIC chips and PHY chips which handle just the high-speed front end will make it easier to use SpaceWire at higher speeds.

A current ESA project is SpaceFibre, which aims to take the SpaceWire protocols up to between 1 Gb/s and 10 Gb/s, using a different physical layer that might include versions for both fibre and copper.

In 2002, a Plug and Play system over SpaceWire [Cook 2007b], was demonstrated that extended to modularity to system configuration. The demonstration was shown many times around the world and undoubtedly contributed to the wide adoption of SpaceWire. It was argued at the time that satellites are fixed configurations with no need for Plug and Play. Once such a plug-and-play capability is used, however, it can be used for the unexpected changes in system configuration and hence can assist Fault Detection Isolation and Recovery (FDIR). There may also be benefits from plug-and-play for the manned space program, where configurations are expected to change over time. And for Responsive Space, launching a satellite in a few days from mission definition means there is no time for system configuration or software development. The system must just plug together and work, so plug and play is necessary and DoD’s Air Force Research Laboratory (AFRL) have designed a plug-and-play system for SpaceWire [McNutt 2009].

In 2012, an ECSS SpaceWire Working Group is developing changes to the SpaceWire Standard to refine the specification of time-code distribution, to introduce distributed interrupts, and to separate the normative definitions from the descriptive text.

There are also community efforts to develop several physical layer variations, such as lower-mass cabling, short-distance single-ended cabling, and galvanically-isolated cabling.

4.3.2. How the use of SpaceWire will evolve

Most of the early uses of SpaceWire have been as medium- to high-speed replacements of point-to-point links such as RS422. A typical configuration would be to connect an imaging instrument to a DSP processor, or to cross-connect a pair of instruments to a pair of processors.

To some extent, this use of point-to-point links without routing switches took place because there were no Rad-Hard routing switches available. Such switches have now been developed, however, by ESA, by NASA, and by a number of companies, and they are being used to construct increasingly-complex networks.

NASA's James Webb Space Telescope is using routing switches to build a large but simple network. The SpaceWire network on JAXA's ASTRO-H spacecraft is extensive.

Routing switches can be used to build in the appropriate level of fault-tolerance, allowing different parts of the system to tolerate different numbers of faults. For example a daisy-chain (without the ends joined together or any cross-connections) does not tolerate some single faults. Connecting the two ends of the daisy-chain so that there is a ring is a simple way to provide tolerance of a single failure, whether the failure is in a node or a link. With three links per node, networks can be constructed which tolerate two failures, and in general, for n links per node, networks can be constructed to tolerate $n - 1$ failures.

Many of the SpaceWire systems being built are modelling earlier systems based on a bus and a global memory access model. Hence the first protocol to be defined is the remote memory access protocol, RMAP. For many applications. this model is appropriate and provides a minimal cost. For other applications, a network model such as Ethernet or the Internet is appropriate. These different models and their protocols can happily co-exist over a SpaceWire network, just as private Microsoft® and other protocols co-exist with TCP/IP over Ethernet.

There is a growing consensus that SpaceWire is the one interface standard that comes closest to meeting the widest variety of application needs for the space industry, and so it must be seen as a prime candidate as the interface of choice for modular systems and responsive lead-times.

4.4. Summary

SpaceWire has been an outstanding success in international collaboration, which has resulted in its use worldwide.

While apparently new technology, SpaceWire has a legacy going back to the CTL Modular One unit interface of 1965, and even the Ferranti Pegasus, and a significant element of that legacy has proved itself in space missions that have been flying for many years.

The legacy of a simple interface that can be used for almost anything has been retained by preserving a number of key principles. These key principles provide modularity, scalability, and reconfigurability, and are far more important than the implementation details.

Early uses of SpaceWire have been evolutionary and have not therefore exploited the full benefits that might be available from using SpaceWire. As more experience and confidence is gained, more of the benefits will be realised.

Benefits should also be realised from the evolution of SpaceWire itself. However, if the key principles and concepts which underpin the SpaceWire standard are lost in that evolution, then many of the benefits of SpaceWire will also be lost.

5. ECSS-E-ST-50-12C and related standards

This section describes how SpaceWire functions, as it routes packets between nodes and broadcasts time synchronisation packets.

5.1. ECSS-E-ST-50-12C : SpaceWire - Links, Nodes, Routers and Networks

ECSS-E-ST-50-12C is the primary standards document for SpaceWire. It provides a full specification of SpaceWire systems and all implementations must comply with its requirements.

The ECSS-E-ST-50-12C Standard covers:

- The Physical Layer of the protocol - cabling and connectors.
- The Signal Layer - the use of LVDS voltage levels, differential data-strobe encoding, clocking, skew and jitter; and signalling rates.
- The Character Level - an outline of the characters and control tokens used on the SpaceWire bus.
- The Exchange Level - a definition of the interconnection of the two ends of the link, in terms of the state machine used at start-up, the flow control mechanism, error detection, exception conditions and low-level error recovery.
- The Packet Level - an specification of the SpaceWire packet structure.
- The Network Level - an introduction to the concepts of packet addressing schemes, wormhole routing switches, and error detection and recovery at the router level.
- The Standard defines conformance criteria for SpaceWire itself, as well as for a number of subsets that might be used separately.

The SpaceWire Standard does not provide much guidance on high-level system design, or on how many of the concepts - such as error recovery - might be incorporated in an implementation. One of the aims of this SpaceWire Handbook is to provide extra guidance on the application of the Standard.

5.1.1. Commentary on the SpaceWire Standard

This section looks at each of the protocol levels of the SpaceWire Standard.

5.1.1.1. The SpaceWire Physical Layer

The physical level specification in the SpaceWire Standard is concerned with the construction at the hardware level of components with SpaceWire interfaces.

The physical layer comprises:

- the SpaceWire cabling;

- the SpaceWire connectors; and
- how SpaceWire is transmitted across printed circuit boards and back-planes.

The standard SpaceWire cable is a relatively heavy and inflexible cable made up of four braided inner conductor pairs, together with fillers, an outer braid and outer coatings. The Standard specifies the dimensional characteristics, the shielding characteristics, and the electrical impedance and skew characteristics of the cable.

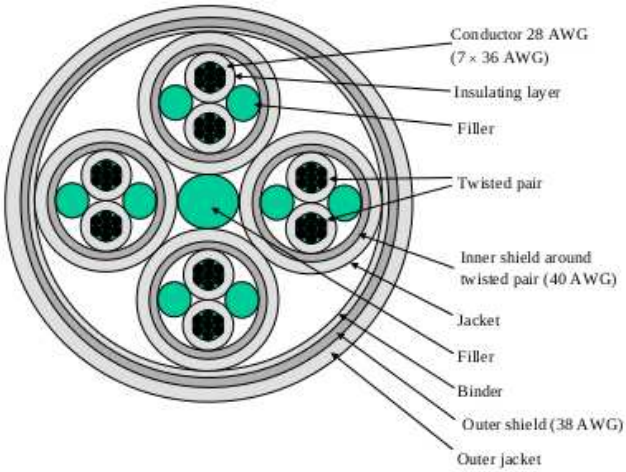

Figure 5.1 - SpaceWire cable construction [from ECSS-E-ST-50-12C figure 5-1]

The standard connectors for SpaceWire are relatively fragile 9-pin Micro-D connectors. Plugs are used on all SpaceWire cables, and sockets (or receptacles) are used on boards and other units. The pin-out groups the transmit and receive D and S signal pairs together, which is convenient. The SpaceWire connectors do not provide impedance-matched connections and care must be taken to equalise the lengths of the wires to the positive and negative conductors in each differential pair.

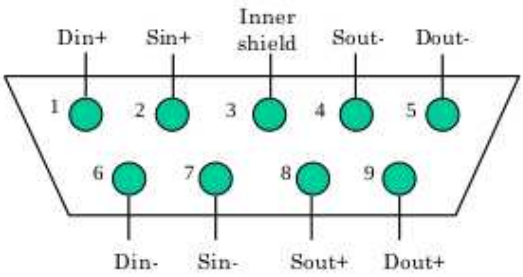

Figure 5.2 - SpaceWire connector [from ECSS-E-ST-50-12C figure 5-2]
(the contacts are viewed from the rear of a receptacle or the front of a plug)

The Micro D connector headshells are used as connections to the outer braid of the SpaceWire cable. The ground pin on a connector is joined to the braids of the two conductor pairs that are used for transmissions from that connector. As a consequence, the grounded braids for these inner connector pairs are isolated from one another for the whole length of the SpaceWire cable and there is no termination of these inner braids at the receiver end.

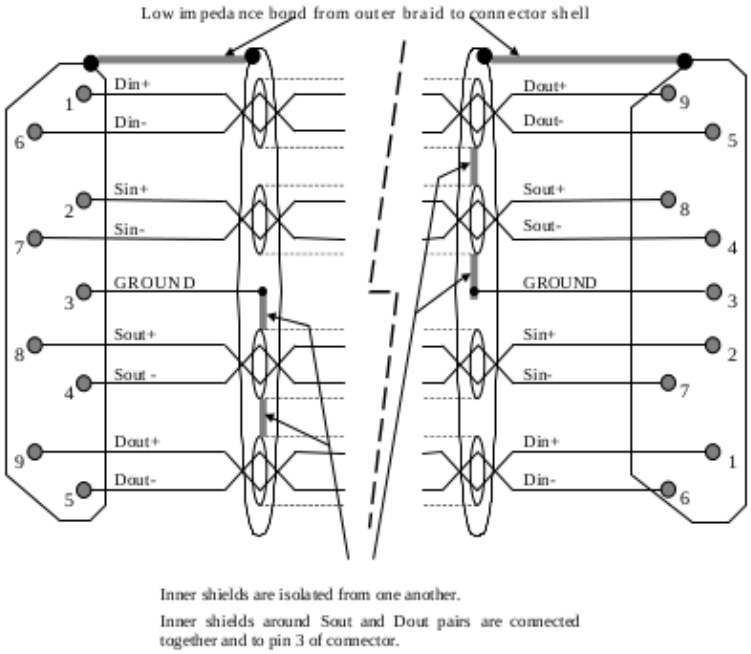

Figure 5.3 - SpaceWire cable assembly [from ECSS-E-ST-50-12C figure 5-3]

See section 6.1.5, “SpaceWire Signal Grounding”, for a discussion on grounding issues in SpaceWire systems.

As would be expected for a point-to-point communications standard, all SpaceWire links on a printed circuit board are point to point connections. Where SpaceWire signal cables connect to a printed circuit board, those signals should be wired as differential pairs, with care taken to meet the skew distance requirements of the Standard. Where SpaceWire signals are run differentially across a printed circuit board, the Standard specifies that the difference in their length should be less than 5 % of the track length and also no more than 5 mm in total. Similarly, the difference in track length for the data and strobe (D and S) pairs of differential signals should also be less than 5 % of the track length and no more than 5 mm in total. These requirements are consistent with the recommendations for high-speed differential data connections in textbooks such as [Johnson 1993].

Section 6.1.4, “SpaceWire Cabling”, provides information on cable length and signal integrity issues.

5.1.1.2. The SpaceWire Signal Layer

The signal level specification in the SpaceWire Standard states that SpaceWire shall use low voltage differential signalling (LVDS) that accords to ANSI/TIA/EIA-644.

A SpaceWire link comprises two pairs of differential signals data and strobe (D and S) in one direction and a corresponding data and strobe (D and S) in the opposite direction.

The encoding used is the data/strobe (DS) encoding scheme. This is defined by the IEEE standard 1355-1995. For any given transmit data rate, the data signal follows the data bitstream and the strobe signal changes state whenever the data does not change from one bit to the next.

The D and S signals can be exclusive-or-ed together to extract the data and the associated clock signal. As a consequence, the signalling rate can be changed on-the-fly at any time. The Standard specifies that the D and S signals will not change at the same time, but the receiver should be tolerant of such simultaneous transitions and not hang up.

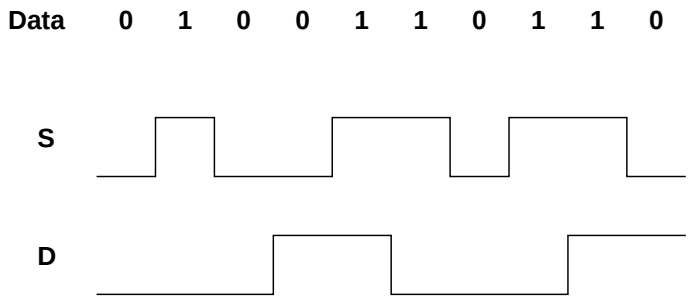

Figure 5.4 - SpaceWire data characters [from ECSS-E-ST-50-12C figure 4-3]

The minimum data signalling rate is specified in the Standard as 2 Mb/s second which is comfortably above the reciprocal of the disconnect timeout of 850 ns. The maximum data signalling rate is determined by the signal skew and jitter specifications which determine how close to each other the data (D) and strobe (S) signal transitions can occur.

For instance, in figure 5.5, the top two traces show the timing of the D and S signals as they are transmitted. The middle two traces show how the D and S signals are relatively delayed by skew and jitter by the time that they are received. In order for the clock to be successfully extracted, the received D and S signals must remain stable for the ‘set-up’ and ‘hold’ times of the extraction logic. As the data rate of the link is increased (i.e. T_{ui} decreases) and as the length of the cable increases (i.e. the t_{jitter} jitter increases), so the timing margin reduces towards zero. Eventually, there is not enough time to extract the clock reliably, and the link stops working.

Some example jitter and skew budgets are given in the SpaceWire specification.

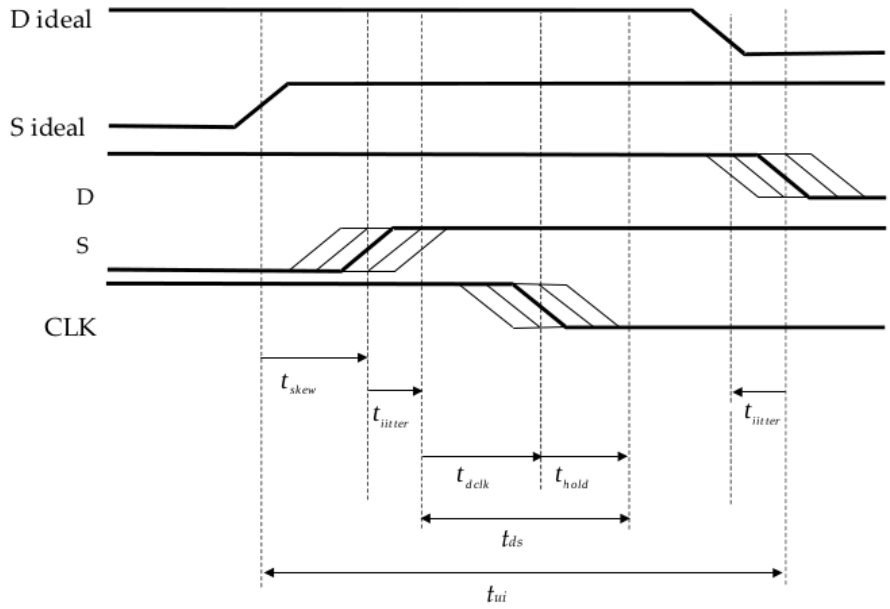

Figure 5.5 - Skew and jitter [from ECSS-E-ST-50-12C figure 6-2]

To provide a common start-up mechanism, each SpaceWire link transmitter initially attempts to connect at a data rate of 10 Mb/s ± 10%. Once connected, the link transmit speed can be varied between the minimum and maximum data signalling rate for that particular connection. A SpaceWire link receiver will be able to receive data at any allowable speed [up to the implementation limits], and track the transmit speed of the attached transmitter at all times.

Note, however, that the Standard permits SpaceWire to be clocked no more slowly than 2 Mb/s. Section 6.6.1 explains that the nominal 850 ns (±10%) timeout on link signal inactivity may provoke a link timeout for speeds lower than 1.3 Mb/s, even if the nominal threshold is 1.18 Mb/s. Also, sections 4.5 and 8.4.7 of the Standard specify how a failed link is re-synchronized and restarted using an “exchange of silence” protocol with timeouts of 6.4 µs and 12.8 µs, and very low data bit rates could cause data to still be flowing at the end of these silence periods. This is why 2 Mb/s is specified as the minimum bit rate in the standard, rather than 1.3 Mb/s as indicated by 1/(850ns-10%) b/s.

The Standard specifies that a receiver should not generate a stream of arbitrary data if it becomes disconnected. This is particularly important in a space application where redundant units are likely to be powered off.

The Standard recommends that a receiver should report a high level logic signal when it is disconnected from a transmitter, or if its inputs are shorted together. Similarly it specifies that a SpaceWire line driver should present a high impedance when it is not powered.

5.1.1.3. The SpaceWire Character Level

In SpaceWire characters, data bits are transmitted together with the parity bit of the previous character and a data / control flag bit. When the data / control flag bit is set to 0, this indicates that the character is a 10-bit data character, which thus contains eight data bits.

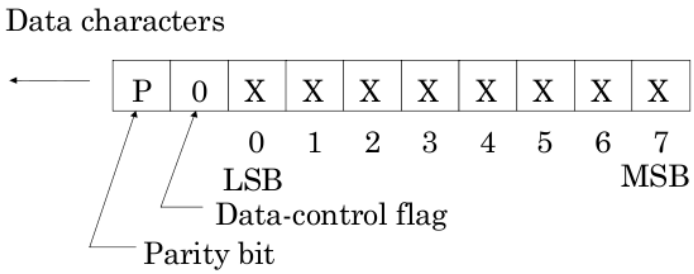

Figure 5.6 - SpaceWire data characters [from ECSS-E-ST-50-12C figure 7-1]

When the data / control flag bit is set to 1, the character is a 4-bit control character, which contains two data bits and can represent four distinct values. These control characters are either used individually, in the case of the flow control token (FCT), the normal End of Packet (EOP) and the Error End of Packet (EEP) control characters or in conjunction with other information in the case of the ESC escape character.

Control codes are built up from one or more control characters, possibly together with data characters. For instance, the NULL control code is made up from an ESC control character followed by a FCT control code. A timecode is made up from an ESC code followed by a data character that contain an eight-bit timecode field [see below].

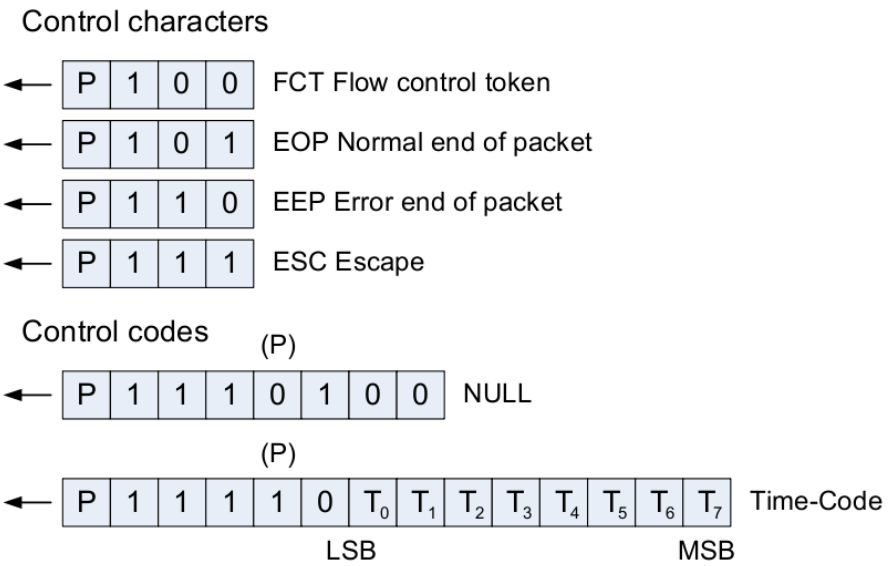

Figure 5.7 - SpaceWire control characters and control codes [from ECSS-E-ST-50-12C figure 7-2]

Each parity bit covers the two or eight bits of the previous character, plus itself and the data / control flag of the current character, and is defined to provide odd parity so that the total number of ones in the field just specified is an odd number.

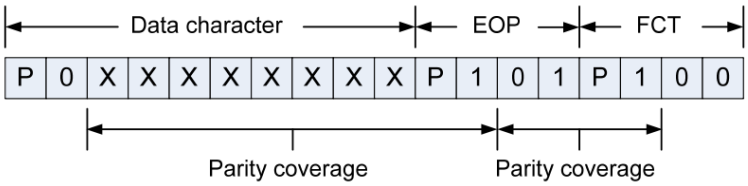

Figure 5.8 - SpaceWire parity coverage [from ECSS-E-ST-50-12C figure 7-3]

Each 8-bit SpaceWire data character therefore carries a 2-bit overhead, which translates to 20% of the link bandwidth being consumed by the character overhead. This is similar to, or better than, the parity bit, start bit and one/two stop bit overhead of an asynchronous UART stream.

One node in a SpaceWire network is responsible for generating timecodes, which are then propagated throughout the rest of the network by all of the routers within that network. Time codes are six bit values, typically incremented and transmitted once per second, and the remaining two bits of the time code character should be set to 0. Upon reception, only the least-significant six bits of the received time code values should be taken into account, although the Standard does not specify what to do if the most-significant two bits of the eight-bit received value are non-zero - and indeed whether the latter case provides a valid time code.

5.1.1.4. The SpaceWire Exchange Level

The exchange level is concerned with the control of the flow of characters on each point-to-point link in a SpaceWire network, and its aim is to ensure that characters are only transmitted when there is buffer space at the receiver to accommodate them.

A SpaceWire link carries a stream of data and control characters in each direction. Only the 8-bit contents of the data characters and the End-of-Packet and Error-End-of-Packet indicators are passed to and from the higher-levels of the protocol stack. These are called N-chars. All of the other control characters and multi-character control codes are only used at the exchange level or below. The logic at the exchange level manages the transmission of N-chars across the link, to guarantee that the buffers at the receivers never become full.

Data characters from the higher packet level may only be transmitted when the receiver has advertised that there is buffer space for those characters by the transmission of a Flow Control Token (FCT). Each FCT sent by a node advertises that it has space for eight more N-chars in its receive buffer. A receiver may transmit up to seven FCTs when a link starts, and will send another FCT each time the receive buffer has been emptied sufficiently to allow another eight data characters to be received.

As well as data characters and FCTs, the transmitter will transmit time-code control codes whenever one has been requested by the time-code logic. Time-codes have the highest priority on a link, followed by FCTs, and then data characters. When a transmitter has nothing else to send, it sends NULL control codes.

In order to ensure that a link starts up properly, and issues the correct number of FCTs before transferring data, the SpaceWire Standard specifies a state machine (see figure 5.9) that defines how a transmitter and a receiver progress from the **ErrorReset** state, through a number of intermediate states, into the **Run** state. Only when a link is in the **Run** state may

time codes and data characters be exchanged. This state machine relies on timeouts, incoming data values in either direction, and the absence of error conditions to arrive at (and then maintain) the **Run** state. After the detection of an error condition or the receipt of a link disable command from the controlling logic or processor, the link state returns to the **ErrorReset** state and when re-enabled the link reconnection handshake may be performed again.

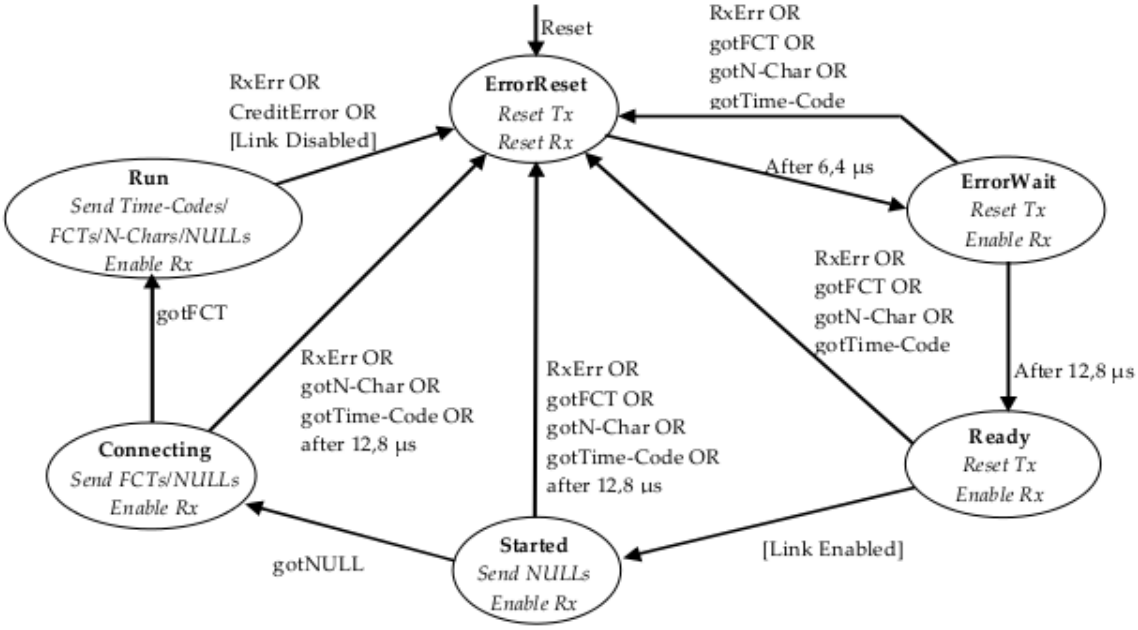

Figure 5.9 - the SpaceWire link state machine [from ECSS-E-ST-50-12C figure 8-2]

There are three signals that control the state machine:

- **LinkDisabled** is a static signal that is set within a node to disable the link. This signal is used directly to signal a transition from the **Run** state to the **ErrorReset** state. Indirectly, it contributes to the **LinkEnabled** signal that is required for the state machine to move from the **Ready** state to the **Started** state.

  AutoStart is a static signal supplied by a node that causes the link to start when it receives a NULL token while the state machine is in the **Ready** state. If **AutoStart** is not set, the link will remain in the **Ready** state until the **LinkStart** signal is asserted.

  LinkStart is a signal that should be asserted by the node logic when the node wishes to start the link, and then de-asserted once the link is running. Its purpose is to cause a one-off transition from the **Ready** state to the **Started** state.

In addition to these three control signals, there are two signals that are set by the state machine logic itself:

- **gotNULL** is a signal that is set when a NULL is received on a link, and reset when the link passes through the **ErrorReset** state. Its purpose is to remember whether an initial NULL has been received, which is the first part of the start-up handshake sequence.

- **LinkEnabled** is a signal that is set when the **LinkDisabled** signal is reset, and either **LinkStart** iis temporarily asserted, or **AutoStart** is asserted and **gotNULL** indicates that a NULL has recently been received.

The purpose of **AutoStart**, therefore, is to allow a link to be started when it receives a NULL from the other ends; in this case it does not send a NULL until it receives one. This contrasts with the situation where **AutoStart** is not set, when a pulse on **LinkStart** is required to start the link. **AutoStart** is therefore useful in slave nodes which do not attempt to send a first-NULL to start the far end of a link, and instead only respond to an active node which starts them. Notice that if the nodes at each end of the link have their **AutoStart** signals set, neither end sends an initial NULL and the link never starts.

Following an error that resets the state machine to the **ErrorReset** state, the state machine imposes a 6.4 µs delay before it enters the **ErrorWait** state, and then another 12.8 µs delay before it enters the **Ready** state. From then, if both ends of the link assert **LinkStart**, the link will be started after each end sends a NULL and then an FCT - just 2 µs being needed to transition into the **Run** state. The total delay of 19.2 µs ensures that both ends of the link see an idle line, and have cleared their **gotNULL** signal, before being prepared to receive an initial NULL.

Errors that cause link disconnection and/or timeout are a disconnect command, parity errors, escape errors, flow control credit errors and character sequence errors. When a receive error is detected, this information should be passed to the controlling logic or processor. If one node detects an error and disconnects its end of a link, the loss of signal transitions results in the other end of the link detecting the fault within a timeout period of 850 ns and also becoming disconnected. Thereafter, if the nodes hold **LinkStart** high, or one holds **LinkStart** high while the other has **AutoStart** asserted, the reconnection sequence will immediately begin to start again.

Outline the problem with legacy IEEE 1355 devices at link-start?

If one observes a link repeatedly disconnecting and attempting to reconnect, the period of attempted restarts will be approximately 32 µs, made up from 6.4 µs in state **ErrorReset**, 12.8 µs in state **ErrorWait**, a very brief period in state **Ready**, and 12.8 µs in state **Started**.

See section 5.1.2.1, “Overheads of the SpaceWire Exchange Level”, for a discussion on the performance and throughput of the SpaceWire Exchange Level.

5.1.1.5. The SpaceWire Packet Level

SpaceWire packets comprise a destination address, a cargo made up of one or more data bytes, and an end of packet marker. Destination addresses are not needed on point-to-point links, but are required in a network which contains router nodes. As much data as is required can be transmitted before the end of packet marker. See section 6.2, “Packet Length Choices”, for a discussion on very long packet lengths.

Whilst packets with zero bytes of cargo should not be transmitted by an end-point node, there are some exceptional circumstances in which an EOP or EEP might be received immediately after another EOP or EEP, and receiver logic (and software) should be designed to tolerate this circumstance. See section 8.9.3 of the Standard for further information on this.

Usually the normal end-of-packet marker EOP is used to terminate a packet, but it is possible to signal an error condition - typically that a data packet has been prematurely truncated in a router - by transmitting an error-end-of-packet EEP.

The packet level, therefore, is able to send packets of any length either to a single destination or to a router that can determine how to relay that packet to its eventual destination.

The overhead at the packet level is a consequence of the address byte(s) and the end-of-packet marker.

- For a point-to-point link, without any routing, there are no address bytes and the overhead is just 4 bits (for the EOP) on top of 10 bits per character in the packet.
- For packets routed using logical addressing, a single overhead byte is required for each packet, in addition to the end of packet marker.
- For packets routed using path addressing, an overhead of several address bytes might be needed per packet, plus the 4 bits of the end of packet marker.

5.1.1.6. The SpaceWire Network Level

The SpaceWire network level interconnects all of the end-point nodes of the SpaceWire network, using a number of router devices. Packets sent by each node will be prefixed by one (or more) address bytes, and the network layer uses this addressing information to route the packets to their destinations.

A SpaceWire network is made up of nodes which typically perform processing, or data storage, or are some form of instrumentation, as well as routers or routing switches which have many input ports and therefore allow a number of nodes to be joined together. Routers use the destination address at the beginning of a SpaceWire packet to determine how to relay that packet.

A router only accepts data for any particular link connection when there is buffer space available at the receiving node. It uses the flow control credit advertised by the receiver of a message to ensure that its output links are never flooded with data that cannot be successfully received. This flow control blocking propagates right across a SpaceWire network and therefore ensures that no data can be lost in transmission. Within each router, data bytes propagate from input port to output port just as soon as they are received; this is wormhole routing rather than store-and-forward routing. Wormhole routing considerably reduces the latency of transmissions, especially when multiple routers are cascaded together in a large network.

Several forms of addressing are implemented by SpaceWire router devices. All of them use the initial byte in a packet as the routing address, and routers may delete this byte after use to expose the next byte in the packet as the address for the next router in a chain. The initial address byte received at a router may be:

- **Zero**, in which case the packet is sent to the internal configuration port within this router. The header byte is deleted in this case.
- **1 to 31**, in which case the packet is sent to the output port with this number (i.e. path addressing). Routers might not implement all possible 31 ports. The header byte is deleted in this case.

- **32 to 254**, in which case the address is treated as a logical address, and the destination port on this router is looked-up in the router's logical address table. The logical address table indicates whether the header byte should be deleted in this case.
- **255**, in which case this value is treated as a logical address, as above. However, address 255 is reserved for future use, and should not be transmitted.

When using path addressing, each router receives the header address bytes at the start of a packet and uses the first one to determine the output port to which that message should be transmitted. Upon re-transmission, that header byte is deleted. Therefore a message with a number of header address bytes will have them successively removed as the message propagates through a network of routers that use path addressing, and when it arrives at its destination there will be no header address bytes remaining.

The second form of addressing within a SpaceWire network is logical addressing. If a router uses logical addressing, it employs a routing table to translate the logical address number in the first byte of a packet to the physical output port that it will use to relay that packet. Each router, therefore, needs to store a routing table that specifies how to relay a message to any possible final destination within the network. It also stores a flag to indicate whether the leading address byte should be removed when the packet is delivered to each output port.

Group adaptive routing allows a routing switch, which is joined to another routing switch with two or more SpaceWire links, to route incoming packets across any of those links, depending upon which ones are available at any instant. This provides load balancing and ensures that the combined link bandwidth is used with maximum efficiency. There is no requirement (but probably should be) that the grouped links go to the same device, they are (currently) allowed to separate into distinct paths in the network.

Alternatively, logical addressing may be used by itself, or together with path addressing, in a scheme called regional logical addressing. In this case, it is possible to operate a group of routers using logical addressing within a wider environment that uses path addressing. Each addressing byte is used to route the packet through groups of one or more routers before it is deleted, exposing the next byte for further steps.

5.1.2. SpaceWire Performance More needed here

- Maximum throughput - unidirectional and bidirectional

  With equal link speeds in each direction, the unidirectional data throughput is 80% of the raw bit rate, and bidirectional data throughput is 76% of the raw bit rate.
- Link / network latency and effect on throughput
- Time code accuracy (limited due to NULL/Data skew)

5.1.2.1. Overheads of the SpaceWire Exchange Level

For unidirectional traffic on a SpaceWire link, there is no additional overhead imposed by the exchange level of the protocol. The idle reverse channel of the link will carry a 4-bit FCT for every eight 10-bit N-chars carried in the forward direction, and these will be carried concurrently with the primary traffic. This is illustrated in figure 5.10.

In figures 5.10 and 5.11, the top trace shows characters passing one way along a SpaceWire link, and the bottom trace shows characters travelling in the opposite direction. The blue rectangles are data bytes and the orange ones are FCT characters. Gaps in the traces are where NULLs are transmitted; these are not shown. The initial FCTs are those sent when the link is initialised.

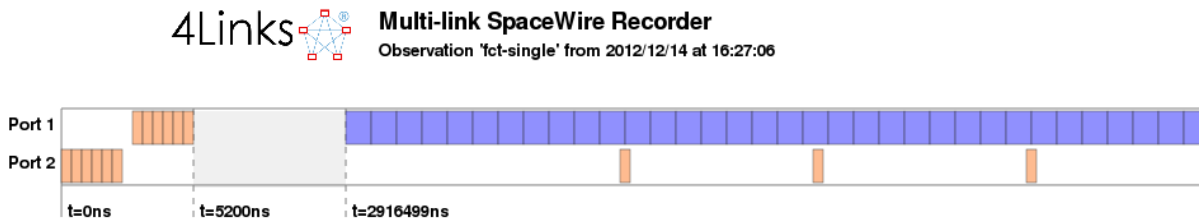

Figure 5.10 - FCTs for a unidirectional data flow

Figure 5.11 shows continuous bidirectional traffic on a SpaceWire link. If we assume that the link is saturates with long data packets, running at the same bit rate in each direction, then every eight 10-bit N-chars carried in each direction will cause one FCT to be sent in the other direction. Since the links are fully occupied in both directions with N-chars, the additional FCTs will slow the primary traffic; 80 bits of traffic will require an additional 4-bit FCT in each direction. This amounts to an extra 5% of overhead.

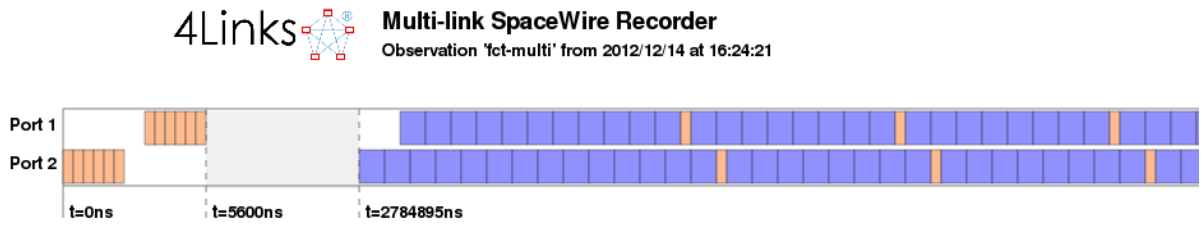

Figure 5.11 - FCTs for a bidirectional data flow

One consequence of the interleaving of FCTs in the data streams is that there is a limit to the difference that can be tolerated between the transmit and receive speeds on a link. If the data characters are transmitted at a high bit rate, and the FCTs are returned at a much slower bit rate, then the data transmission will be throttled due to a lack of flow control credit when the transmission time of eight N-chars is shorter than the time needed to send the FCT which enables their transmission.

We can explore the issue of widely-different transmit and receive speeds by looking at the extreme circumstance of a unidirectional flow of the shortest-permissible [1-byte] packets, each terminated with an EOP. Each four bytes and four EOPs will be balanced by an FCT flowing in the opposite direction, so (in steady-state) 56 bits must take longer to transmit than the 4 bits of FCT in the opposite direction. Therefore, the speed of the data packet transmission may be no faster than 14 times the speed of the FCT path if delays in the transmitted stream are to be avoided. This ratio raises to a speed ratio of 20 times for the continuous flow of data characters shown in figure 5.10.

Need another diagram here.

When the FCTs on the slow side of a link are mixed with data characters, we must ensure that flow control credit is always available for the primary data traffic on the fast side of the link. Whilst each 4-bit FCT might be queued up behind a 10-bit data character, there will be little impact overall because FCTs are transmitted with higher priority than data characters, and so short bursts of FCTs will be generated if need be.

5.2. ECSS-E-ST-50-51C : SpaceWire Protocol Identification

ECSS-E-ST-50-51C specifies a SpaceWire packet format that provides:

- a one-byte logical address, as the packet arrives at the receiving node. Note that this logical address might have been prefixed by other logical or path address bytes that might have been deleted as they passed through the routers in the SpaceWire network;
- a one-byte protocol identification value.
- further packet bytes corresponding to the packet format identified by the protocol identification value byte.

As an alternative to the one-byte protocol identification value, a three-byte extended protocol identifier may be used; the first byte of this is zero and the two that follow provide a 16-bit protocol identification value in big-endian byte order.

Four protocol identifier values have currently been standardised. The Remote Memory Access Protocol and the CCSDS Packet Transfer Protocol are specified in ECSS-E-ST-50-52 and ECSS-E-ST-50-53 respectively and these are discussed in the next two sections of this Handbook. The GOES-R Reliable Data Delivery Protocol [NASA 2005] and Serial Transfer Universal Protocol [EADS 2009] are not specified by the ECSS.

There is a reserved range of project-specific protocol identifier values that may be used within individual programmes.

5.3. ECSS-E-ST-50-52C : the Remote Memory Access Protocol

The SpaceWire Remote Memory Access Protocol (RMAP) defined in ECSS-E-ST-50-52C is a client-server protocol that provides a mechanism to read and write memory at a node in a SpaceWire network. It has gained popularity because it models the way that people are used to using a physical bus and thus helps them to map a model of that bus onto their SpaceWire network.

The RMAP protocol achieves these goals in a general fashion, which provides flexibility, but with high overheads for small transfers. All of the SpaceWire addressing schemes are accommodated - limited path addressing, logical addressing and regional addressing.

5.3.1. Commentary on the SpaceWire RMAP Standard

The Remote Memory Access Protocol (RMAP) provides a general register-setting and memory access service. Particular applications include:

- Controlling SpaceWire nodes by setting and reading their memory-mapped registers;
- Configuring routing switches, where a special configuration port is provided for the purpose;
- Transferring data to and from nodes, either in addressed blocks from memory or as a continuous stream at a non-incrementing port address.

The protocol provides commands for read, write, and read-modify-write. The command header is quite large (16 bytes plus a return path address, if needed for the SpaceWire addressing scheme in use). As a consequence, reading or writing a single 32-bit word requires more than sixteen extra bytes of overhead. This overhead ratio decreases as the cargo data size increases.

A CRC is provided to protect the RMAP protocol header, since it is important not to access the wrong memory address, to issue the wrong command, or to return the acknowledgement packet to the wrong SpaceWire node address. The CRC is only eight bits in length, which is not particularly secure, but this is in addition to the parity bits within each SpaceWire character, and so overall this mechanism is fairly robust. A CRC is also provided for the data that is written by an RMAP command, and for that which is read. This CRC is also only eight bits in length. The CRC algorithm used for both checks returns a value of zero if all of the bytes checked are themselves zero, so be aware that the CRC check of a buffer containing all zeroes will be successful.

Monitoring RMAP commands and responses for correct checksums is a good test of the whole SpaceWire network, especially if incrementing Transaction IDs can be used to identify any packet loss.

The ECSS-E-ST-50-52C RMAP protocol specification contains many configurable and optional features. Care must be taken to check the implementation details to determine whether two nodes will interoperate successfully. The basic RMAP read/write operations may be assumed to exist.

The RMAP Standard does not provide much guidance on:

- How the protocol (and higher-level applications in the affected nodes) operate in the presence of link errors;
- How to handle concurrency issues safely - such as synchronising exchanged data and managing semaphores.

5.4. ECSS-E-ST-50-53C : the CCSDS Packet Transfer Protocol

The ECSS-E-ST-50-53C CCSDS Packet Transfer Protocol transfers CCSDS Space Packets across a SpaceWire network. It encapsulates each CCSDS packet in a SpaceWire packet, routes it to its destination, and then removes the encapsulation to reveal the original CCSDS packet. In addition to the CCSDS packet itself, the protocol carries an ECSS-E-ST-50-51C Protocol Identifier field, a packet length value, a Target SpaceWire Address (of arbitrary length) and a one-byte Target Logical Address, a single-byte User Application Value, and a reserved 8-bit field that is always set to 0x00. The length of the CCSDS Space Packet is determined by the length of the encapsulating SpaceWire packet, i.e. by the receipt of the SpaceWire EOP character.

The CCSDS Packet Transfer Protocol provides unidirectional data transfers across point-to-point links with no guarantees for successful delivery. The protocol is asynchronous and any response message would have to be conveyed by a separate mechanism, such as a second CCSDS Packet Transfer Protocol channel.

6. Specific Implementation Topics

This section provides guidance on topics that are important for a successful SpaceWire implementation.

6.1. Physical Layer Interconnections

This section deals with the physical layer electrical interconnection issues such as LVDS signals, grounding and PCB layouts.

6.1.1. SpaceWire Binary Encoding

SpaceWire communication is pure digital, using Low Voltage Differential Signaling (LVDS). These two-level binary signals are recovered by a simple line-receiver with no signal processing requirement. The line code is a Gray-code which allows bit-boundaries to be recovered from the data stream - the transmit clock is recovered at the receiver.

As a result, a very wide range of speeds is automatically supported by SpaceWire (currently from <2 Mb/s to over 600 Mb/s). Bit-to-bit speed change capability is a useful by-product that allows spread-spectrum clocking to reduce electromagnetic interference and also allows reduced power consumption by throttling back the transmit speed when there is no active data to transmit.

6.1.2. LVDS Signals and Levels

It would be good to include the LVDS driver and receiver loop as a diagram. In addition, show the waveform diagram as well as typical noise margins.

The SpaceWire physical layer is required to use Low Voltage Differential Signalling (LVDS) as defined in the document ANSI/TIA/EIA-644 (A). There are other LVDS specifications, including M-LVDS, B-LVDS and the IEEE 1596 signal specification that are sometimes also discussed in the context of SpaceWire.

LVDS specifies a low differential output voltage, nominally 350 mV, sitting at a defined common mode level, nominally 1.25 V. The nominal output terminal voltages are thus between 1.075 V and 1.425 V. This allows an end-to-end voltage difference between ground wires (the common mode voltage range) of ±1 V whilst keeping the receiver input terminal voltages between 0.075 V and 2.425 V.

It is often believed (and implied in the ECSS-E-ST-50-12C SpaceWire Standard [section 4.3.2]) that LVDS drivers must be current sources. In fact, the circuit shown completely fails to achieve the common-mode voltage requirement and must be understood as a simplified illustration, not an actual implementation. True current sources cannot control the output common mode level and are always used within a feedback loop which has the effect of making them behave more as voltage sources, as the LVDS standard requires (the Standard specifies output voltages, not currents).

Devices such as the Actel RTAX use voltage outputs and a resistor network to produce correct LVDS levels. A lower supply voltage (2.5 V) is used than dedicated LVDS drivers (3.3 V or 5 V) and the off-chip series resistors limit fault currents. [Silicon commonly fails low

impedance (short-circuit or avalanche conduction) which can deliver large fault currents.] Actel specifies these LVDS circuits for operation up to 350 Mb/s (and Xilinx achieves >1 Gb/s). The suggested resistor network has the further benefit of providing a matched impedance at the source which absorbs any signal reflections or noise; perfect current mode (or perfect voltage mode) sources would totally reflect such interference to degrade data.

The reasons why the LVDS outputs must be controlled include:

- the control of signal rise-times to reduce the EMC spectrum;
- the control of link outputs when switching nominal and redundant modules;
- the control of over-voltage outputs under fault conditions;

It is difficult to find slow LVDS buffers. A typical rise/fall time of 1 ns is quite slow for such devices; 300 ps is more typical. The LVDS standard allow the rise/fall times to be as long as 30% of the bit period. For SpaceWire data rates of up to 50 Mb/s, the bit-period is 20 ns or more and typical LVDS buffers are an order of magnitude faster than is necessary. Such over-specification produces a greatly extended emission spectrum and requires much tighter matching of wire lengths and produces greater reflections from impedance mismatches such as occur in the micro-D connectors. If the rise and fall times were matched to the data rate, we would limit the highest frequencies produced and greatly ease construction.

More information about LVDS signalling for SpaceWire can be found in [Cook 2008a] and the effect of edge times on emissions in [Cook 2007c].

6.1.3. SpaceWire codec implementation

To be supplied. Should also add references to works on metastability. [Ginosar 2003, Ginosar 2011]

To be supplied - information on clock recovery circuits.

6.1.3.1. Asynchronous issues at the logic level

In most cases, the processors or I/O hardware at either end of a SpaceWire link will have independent hardware clocks. The worst case is when the clocks at the two ends are exactly the SAME frequency but phased due to transmission delays (particular cable lengths) so that metastability occurs at EVERY data transition!

The challenge for the hardware designer is how to transition from the transmit clock domain into the receive clock domain at each end of each link. The primary issue facing the CODEC designer, therefore, is that of metastability avoidance [Ginosar 2003, Ginosar 2011].

Add a few lines on recommended practices for clock domain crossings.

More information about asynchronous issues for SpaceWire can be found in [Cook 2008b].

6.1.4. SpaceWire Cabling

The SpaceWire Standard takes considerable trouble to specify suitable cable and its use to maximise signal integrity and to reduce electromagnetic emissions. Cables, by virtue of their length, can form effective electromagnetic radiators. Shielded cable is specified for SpaceWire to reduce radiation from the signal conductors which are, in turn, differential pairs to balance flow and return currents and limit emissions.

Unfortunately, the 9-pin Micro-D (MDM) connector specified in the SpaceWire standard is not designed for balanced signals and introduces a noticeable imbalance between the wires of a pair. This reduces the effectiveness of balanced transmission, getting progressively worse at higher frequencies, resulting in increased emissions.

There is little published material available on the subject of SpaceWire cable measurements. [Allen 2006] and [Mueller 2006] provide a good analysis of one NASA application and some Gore measurements, respectively. [Allen 2006] confirms that 200 Mbps SpaceWire is achievable on 20 m links, and [Mueller 2006] gives eye-pattern measurements of jitter for thicker-than-standard AWG26 wire and longer-than-standard 20 m connections.

Regarding cable assembly construction, it is important to 360°-terminate the outer shield of the cable to the backshell of the MDM connector, and to ensure a low impedance connection from the backshell, via the connector body, to chassis ground. This will maximise the shielding effectiveness. The choice of Micro-D connector, and its current pinout, prevents the inner cable shields from passing through bulkhead connectors, so the informal recommendations (e.g. in [Parkes 2012, section 3.1.3]) that terminating the inner shields at each end of the cable to the connector backshell together with the outer shield, leaving pin 3 of the connectors unconnected, might be more effective in some circumstances.

However, tests on Ethernet [ref?] have shown screened cable radiating MORE than unshielded cable - EMC is a complex business!

A recent ESA study into low-mass SpaceWire cables has explored various measures that can reduce the mass of a cable assembly, with the added benefit of increased flexibility [Rouchaud 2011]. The promising techniques include:

- removal of the overall shield;
- Replacing silver plated copper shields with silver plated aluminium;
- The use of non-twisted sub-miniature coaxial cables instead of shielded twisted pairs;
- Use of lighter-weight insulation and filler materials.
- Use of impedance-controlled twin-axial connectors rather than Micro-Ds.

The outcome of this research has been discussed at SpaceWire Working Group meetings but has not been standardised yet. The study has not yet published its measurements of acceptable SpaceWire bit rates for different cable lengths.

6.1.5. SpaceWire Signal Grounding

Whilst the differential levels of the SpaceWire signal layer provide a robust signalling scheme, the signals at the receiver are referenced to the local ground at the transmitter. If the ground potential between a SpaceWire transmitter and a SpaceWire receiver exceeds the receiver's common mode tolerance levels, communications will fail.

The design of a spacecraft's electrical grounding scheme is outside the scope of this Handbook, but recommendations such as [NASA 1998] contrast several alternatives:

- Single-point grounding, where all signals are referenced to a single location, and the mechanical assemblies are bonded together, but are electrically isolated from the signals.
- Multiple-point grounding, where the mechanical and electrical grounds are common. There are various arrangements which are better-suited to low-frequency and to RF systems, and other schemes that are inadvisable.

  In large spacecraft, travelling fast through the Earth's magnetic field, induced voltages in poorly-designed ground loops might eat up much of the noise immunity of a SpaceWire signal, so the grounding paths must be considered carefully.

In the context of SpaceWire, the Standard requires that the outer braid of each cable be used as the signal ground, and that the pin 3 shield connection should be isolated from the signal ground. The outer braid is required to be 360°-terminated to the connector headshell and therefore to the module assembly containing the SpaceWire circuit. This is difficult to accommodate in a single-point grounding arrangement. Also, it implies that the signal ground in the module should be connected to its mounting frame.

There are two important circumstances when the SpaceWire grounding scheme might cause problems:

- In a prototyping or test environment, the SpaceWire ground reference must be supplied independently of the nine pins of each SpaceWire connector. This requires that the headshells of the connectors on the units under test be wired with appropriate low-impedance connections and the jack-screws be used on the cables to ensure a good connection. Alternatively, a separate single-point ground must be made between all of the equipment under test. This connection might itself be a source of noise pick-up, so care must be taken in its layout and construction.

  If a suitable ground reference is not provided in a test environment, it is likely that the SpaceWire interfaces will operate most of the time, but that the immunity to electrical noise will be reduced. As a consequence, individual SpaceWire links might fail on an irregular basis. If failures are observed, checking the grounding design of the test environment is worthwhile.
- The requirements for electrical grounding of the SpaceWire connector headshells and cable shields to EGSE test equipment indirectly implies that a spacecraft under test must be connected to electrical ground for electrical safety reasons. If this is not acceptable, either the test equipment or the SpaceWire connections to the spacecraft must be electrically isolated. Isolating the test equipment is difficult if it incorporates USB connections to conventional PCs, although it might be easier for laptops with external power supplies. Ethernet is usually transformer-coupled and is therefore isolated.

Isolated differential buffers are available, which could be incorporated into special SpaceWire cables or into test equipment. These are often incapable of the full SpaceWire speed range, and may have common-mode voltage limitations that must be taken into account. Examples include the Analog Devices ADuM5400 family, and improved new products from Silanna and Telefunken Semiconductors that were previewed at the October 2012 SpaceWire Working Group meeting.

6.1.6. Printed Circuit Board layout

The design of printed circuit boards for SpaceWire units should follow best practice for the propagation of high-speed differential signals. These include:

- Ensure that the impedance of each SpaceWire differential track pair is 100 Ω by calculating the appropriate track widths and spacing between tracks using the relevant microstrip/stripline equations.
- Minimise skew within each differential track pair and between Din/Sin and Dout/Sout pairs by ensuring that their lengths are the same.
- Minimise discontinuities on the signal paths, i.e. reduce the number of vias, and use arcs or 45 degree bevels rather than 90-degree bends in the PCB tracks.
- Isolate the link signals, as far as possible, from any single ended neighboring signals. This should preferably be done by routing them on separate PCB layers, and including a screening layer (i.e. a ground-plane) between the logic and link signal layers.
- Stub lengths on the link tracks should be kept as short as possible.
- The LVDS drivers and receivers should be kept as close as possible to the relevant connector, to minimise opportunities for noise pick-up.
- Termination resistors must be at the end of the signal line (i.e. next to, or within, the receivers) - not in the middle (e.g. at the connector).

6.1.7. Other EMC Issues

To be supplied.

6.1.7.1. EMC Mitigation through Token Randomisation

The only frequent individual control token used in SpaceWire is the Flow Control Token (FCT). Data may fully occupy the stream resulting in a periodicity of 10-bits. When no data or flow control is sent the link sends NULL characters (so that a disconnection can be detected), each of which consists of a pair of control tokens, giving periodicities of 4- and 8-bits. NULLs, because they are regular and unchanging, concentrate energy into a small number of narrow-band signals.

This material affects the codec state machine, which then is not ECSS-E-ST-50-12C compliant. Leave out?

Some communication schemes randomise data, including idle sequences, in order to spread narrow concentrations of energy over a wider frequency range. Although data randomisation is possible with SpaceWire, it adds considerable complexity to an otherwise simple design -

and cannot include NULL tokens, thus seriously limiting the benefit.

6.1.7.2. EMC Mitigation through Bit Rate Randomisation

Although the SpaceWire Standard refers to jitter in the D and S signals, this is not actually a limiting factor. It is necessary to recover the transmit clock and to use this to latch (on both edges) D to recover the data stream. The real limiting factor is determined by the quality of the recovered clock and this is determined by the edge-to-edge spacing in the DS signals (D to D, D to S, etc.). The actual frequency of the clock is unimportant so long as it exceeds a minimum value, determined by the timeout detector (~2 Mb/s), and is below a maximum value determined by the receiver implementation.

It is perfectly allowable for the data rate to change from one extreme to the other (or any lesser range) from one bit to the next (incidentally, providing a potentially useful power saving facility). In fact, it is required to do so since a link must start at 10 Mb/s and then change, mid-bit-stream, to the working rate.

Thus it is possible to alter the edge-to-edge spacing (bit period) in a random sequence - a technique used elsewhere and known as spread-spectrum-clocking. This spreads the energy in single-frequency spikes over a range of nearby frequencies and reduces the peak levels significantly. It can easily be applied to SpaceWire, without rendering the implementation non-conformant to the Standard.

6.2. Packet Length Choices

The SpaceWire Standard allows packets to be of any length - including an unlimited length. The length of a packet is determined by the detection of an End-of-Packet (EOP) or an Error-End-of-Packet (EEP) marker, so an endless packet is the consequence of never receiving an EOP or EEP character. No guidance is given in the SpaceWire Standard on an appropriate packet length for any particular application.

Using an unlimited-length packet (i.e. a continuous data stream) might be considered appropriate if an endless sequence of characters needs to be carried across a SpaceWire network. In this case, once the stream has started, no further addressing decisions need be made and the channel path throught the network becomes fully-committed - a virtual circuit. It is therefore not possible to route any other information to the destination SpaceWire port, nor to any of the input ports that are being occupied by this stream on any intermediate routers. Thus, the capability of the SpaceWire network to send additional information to these nodes, perhaps for fault reporting or re-configuration controls, is removed.

Another concern when using an unlimited-length packet is that an error that is detected during its transmission through a router (for instance, a parity error) will result in an EEP being introduced. If the receiver is not expecting to handle end-of-packet markers, let alone EEPs, then handling this error marker is unlikely to be easy. Similarly, after sending an EEP, the router that detected the error should then spill the rest of the incoming packet until an EOP is seen [ECSS-E-ST-50-12C clause 11.3)]. If there is no EOP, that data stream will be spilled forever.

So if packets should have a finite length, what should it be?

- For an imaging application, an image might not be worth storing or analysing if it is not complete. In this case, using a single packet for the whole of an image might be appropriate. It might be useful to prefixing each image with a serial number, and possibly information about its capture.
- For images which are captured row-by-row, transmitting each row in a separate SpaceWire packet might be more beneficial. In this case, prefixing each row with its row number and the image serial number would help the application to defend against the loss of a packet. Providing a separate image header packet with the serial number of the image and the image capture information would also be helpful.
- Smaller fragments of command information or data should ideally be carried in identifiable packets.
- The SpaceWire Remote Memory Access Protocol (RMAP) limits the maximum packet data size to 16 Mbytes. RMAP packets also have a CRC, a Transaction ID field that could be used to identify the payload, and a response mechanism that can be used to account for lost transactions.

6.3. Timing Issues with Time-Code Distribution and Distributed Interrupts

Comment about circular paths around routers carrying time-codes and interrupts

6.4. Reducing Jitter in Time Codes / other places

More information about time-code jitter reduction for SpaceWire can be found in [Cook 2003].

6.5. SpaceWire Routing Issues

The SpaceWire Standard requires that routing switches should provide path addressing, and permits them also to provide logical addressing and its variant, regional addressing. See section 4.2.4, “Packet Routing”, for details of these schemes.

Specific issues regarding routing are those of router latency, and the potential for corruption of the logical addressing routing tables.

6.5.1. Wormhole Routing

SpaceWire routers are required to use wormhole routing, in which the head of a packet ‘pulls’ the rest of the packet through the router.

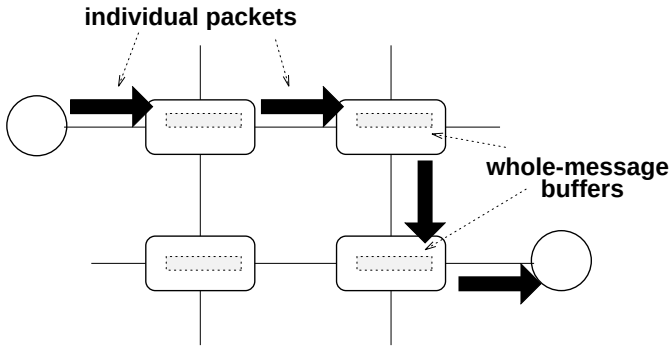

Figure 6.1 - store-and-forward routing in a router

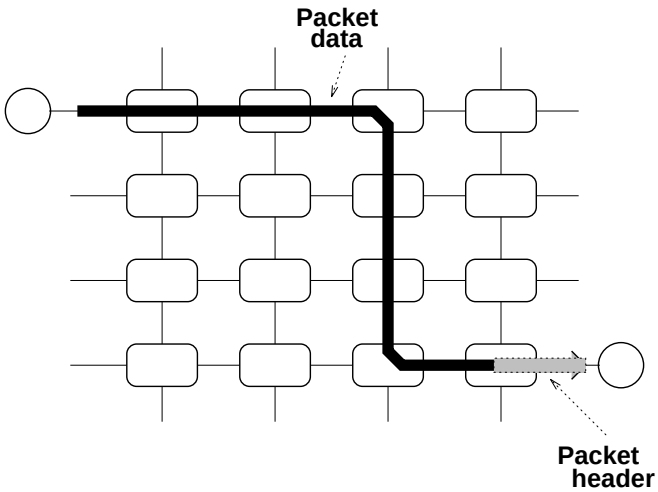

Figure 6.2 - wormhole routing in a router

Wormhole routing permits arbitrarily-long packets to be used. It also decreases routing latency, compared to store-and-forward routing, especially when cascaded routers are used.

6.5.2. Corruption of the Routing Tables

The logical addressing routing tables are stored in the routers. When a packet is received by a router, the first byte is tested to see if it is a path address or a logical address. If it is the latter, the address is looked up in a table (usually in RAM) to determine the correct output port to which this packet should be transmitted.

In a space radiation environment, the consequence of a charged particle event corrupting entries in this routing table could be severe. Therefore forms of automatic error detection and correction (EDAC) are likely to be required to protect this memory.

6.6. SpaceWire Network Architectures

Since SpaceWire is not a single bus structure, it provides many choices of interconnection architectures and topologies. This flexibility provides the ability to trade-off characteristics such as performance, complexity, cost, fault tolerance, power consumption and mass.

A mission's designers should determine their priorities from:

- The performance required: sometimes this will be different for different parts of the spacecraft, sometimes it is purely bandwidth, sometimes latency or delay through the network;
- The extent of fault-tolerance required, which may be different for different subsystems on the satellite.
- The mass of the cable harness;
- The power consumed by the network;
- The cost of the components to build the network;
- The availability of the communication components.

The configuration of a spacecraft's communication network will be determined by the bulk data flows, as well as the timely delivery of telecommand, telemetry and control information. It might be appropriate to separate some of these functions, or it might be best to combine them into one network to save resources. Resilience and redundancy should also be incorporated, which is a trade-off that requires extra resources.

Many aspects of the design of such an embedded computer network are similar to those of developing a parallel or distributed computer system.

The following sections illustrate some of the network building blocks that could be utilised in an on-board data handling environment.

We should perhaps try to focus on typical onboard architectures which uses relatively simple configurations but appears to cause much confusion during the design phase. e.g. how much buffering is needed in the nodes to account for latencies, which actual links speeds needs to be used etc.

6.6.1. Point-to-Point Serial Cable Replacement

A point-to-point RS-422 synchronous or asynchronous connection can beneficially be replaced by an individual SpaceWire connection, even though the topology remains the same.

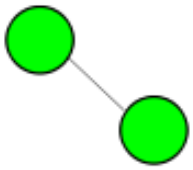

Figure 6.3 - point-to-point connection of two nodes

SpaceWire has many benefits even in this simple situation:

- The flow-controlled synchronous nature of SpaceWire is easier to reason about the traffic because data loss is impossible, instead being replaced by blocking issues that can be countered by buffering and by attention to timely processing responses.
- In almost all cases, point-to-point connections are usually supplemented by other information flows, such as telecommands to each of the nodes, and propagating these would often be easier if the commands were received at one node and then shared out along the intermediate path.
- Many point-to-point connections could be controlled using client-server programming techniques, perhaps using the SpaceWire RMAP protocol. This allows several logical streams of information to be multiplexed together, and a familiar software polling model to be used.
- Point-to-point connections provide no redundancy, nor alternative paths for data to flow if one becomes blocked. If the lack of redundant paths on a point-to-point link is unacceptable, two or more parallel SpaceWire connections could be provided instead. These could be used in a nominal / redundant configuration, or the traffic could be shared between the two links while they are working reliably using group-adaptive routing.

6.6.2. Chained and Cyclic Configurations

One of the simplest ways of allowing many nodes to communicate with each other is to join them in a chain, and for each to relay messages to their neighbours along that chain.

Figure 6.4 - chained connection of nodes

The drawbacks of this scheme are:

- The message flows aggregate across parts of the chain, potentially exceeding the bandwidths of those interconnections, or the routing capabilities of the nodes.
- Any failure in the chain - of nodes or communication links - causes the chain to be divided, and messages needing to pass the breakage will fail to arrive.
- The routing scheme must be designed to be deadlock-free in the presence of all possible flows of traffic between the nodes (in normal circumstances or when failures occur).

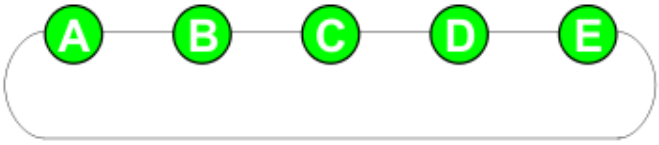

Figure 6.5 - a ring-connected set of nodes

Joining the chain into a ring avoids many of these issues and adds a little redundancy:

- Traffic will be spread more evenly amongst the nodes, rather than being unevenly distributed towards the centre of the chain;
- There will be two routes for a message to take from source to destination, providing a redundant path in every circumstances and allowing the shortest one to be chosen for performance reasons;
- The deadlock-free routing scheme becomes more complex to design and to enforce.

6.6.3. Mesh-Connected Networks

Mesh-connected networks are an elaboration of the chain or ring networks in the previous section. By joining nodes in a mesh or grid like configuration, many more nearest-neighbour communication channels are provided, and many more routes become available for messages to follow from their source note to their destination node. This reduces the number of nodes that any message has too traverse to reach its destination (its diameter).

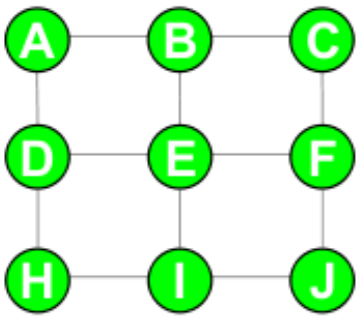

Figure 6.6 - a mesh network

If the outer nodes in the mesh are joined to those on the other side, forming a loop on each row and column, we have a toroidal structure with an even smaller diameter.

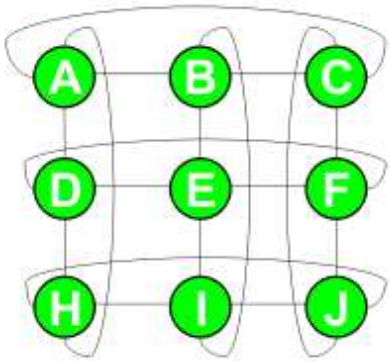

Figure 6.7 - a toroidal mesh network

A mesh configuration is particularly suited to data processing tasks where the algorithm can be copied across the nodes and each part of the data, held on one of the mesh nodes, is processed in parallel before the results are aggregated centrally.

As with the ring configurations above, the complexity of the routing algorithms in a mesh or toroidal network becomes greater, and even more care must be given to deadlock considerations.

6.6.4. Fully-Connected Networks

Adding yet more interconnections between nodes allows us to develop fully-connected (or star) networks in which every node is directly connected to every other node with which it wishes to communicate.

The advantage of a fully-connected network is that messages are delivered in just a single communication, and therefore the latency is lowest of any of these schemes. The disadvantage, however, is that there are more network cables which contribute weight and volume between the nodes, and also each node requires as many communication ports as there are nodes that they wish to communicate with. Eventually this valency makes a fully connected scheme impractical to implement.

Fully connected networks provide excellent resilience and the potential for redundant paths should one link fail.

6.6.5. Routed Networks

Rather than just joining nodes together and requiring them to route messages to each other, it is often simpler to separate the routing activities from the data-processing ones of the notes themselves. In this circumstance, we can introduce special purpose routers that solely relay traffic from the notes to other notes.

In the simplest circumstance there will just be a single router and all the nodes will connect directly to that router and all messages will take two hops to reach their destination. In more complex situations, there will be multiple routers, perhaps arranged in an hierarchical tree, and there is also the opportunity for redundant paths, redundant routers and full fault detection and recovery schemes.

6.6.5.1. Single Router Configurations

A typical application for a SpaceWire network is to connect a set of instruments to mass memory and processing resources, and a convenient way to do this is with a centralized routing switch, perhaps located on the mass memory or processor nodes.

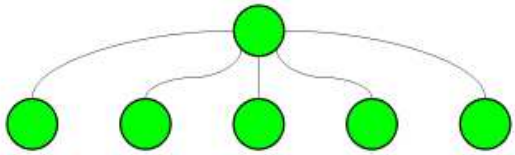

Figure 6.8 - connection of nodes using a single router

This configuration is equivalent to an Ethernet switch, since it allows all of the nodes to communicate directly with each other. There are some drawbacks to this simple scheme:

- There is no redundancy - the router is a single point of failure;
- The router has a limited number of ports, and this sets a maximum bound on the number of nodes that can be connected.
- The total length of cable (and hence harness mass) is likely to be considerably higher with all the cables coming together at the router, rather than just connecting nearest neighbours.
- Since SpaceWire routers employ wormhole routing, which ties up routing resources when other paths are in use, a more thorough blocking analysis will be required if routers are used in place of point-to-point links. This analysis will be of similar complexity to that needed for networks that relay packets through intermediate nodes.

6.6.5.2. Multiple Router Configurations - Making larger routers

To increase the number of router ports, and hence nodes in our system, we can provide multiple routers and cascade them together. Providing a hierarchical fan-out with a tree of routers is the simplest enhancement with the maximum gain in the number of nodes.

Notice in figure 6.9 below that all of the traffic from the left-most third of the network to the central and right-hand sections through just one link. Saturation of these inter-router links is likely to be a limiting factor in this design.

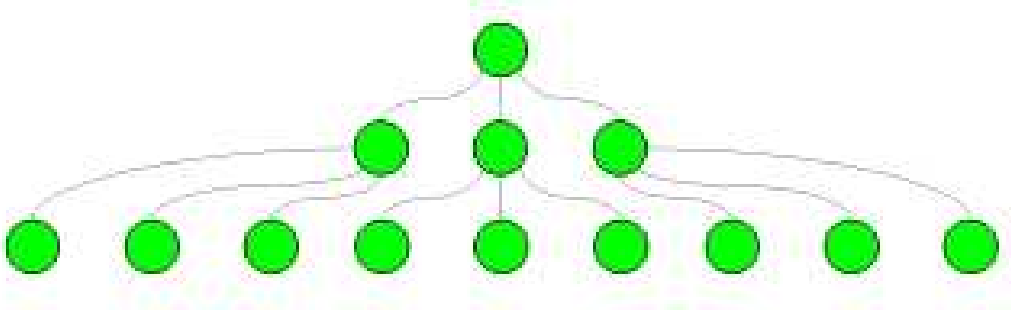

Figure 6.9 - connection of nodes using a tree of routers

6.6.5.3. Multiple Router Configurations - to increase routed bandwidth

The limited bandwidth in the hierarchic tree of routers, above, can be mitigated by using more links between the routers. Such a 'fat tree' can employ group adaptive routing in the doubled-up paths, or it can allow the logical routing tables to distribute the traffic in a programmed fashion. Alternatively, individual nodes can select the paths for their traffic individually.

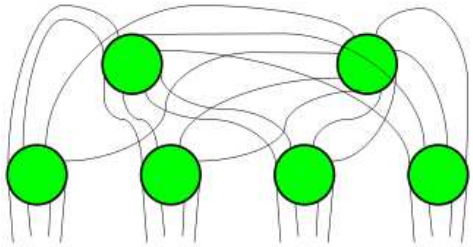

Figure 6.10 - connection of nodes using a 'fat tree' of routers

The extra paths between routers have introduced the possibility of routing cycles, leading to the issues of livelock or deadlock, although it would not be a good use of these parallel links if the routing algorithm traversed them more than once.

6.6.5.4. Multiple Router Configurations - to provide redundancy

A central SpaceWire routing switch has many attractions. It is simple, and each node is dependent only on the switch and does not have to handle traffic from other nodes. If one node fails, none of the other nodes is affected, so fault-tolerance with respect to the nodes is good. On the other hand, if the central router fails, it may cause the whole mission to fail.

Fault-tolerance can be improved by adding a cold-redundant switch, at the cost of approximately doubling harness mass. Alternatively, for the additional cost of the power to operate it, the extra switch could be used to provide additional routing bandwidth as well as its fault-tolerance properties.

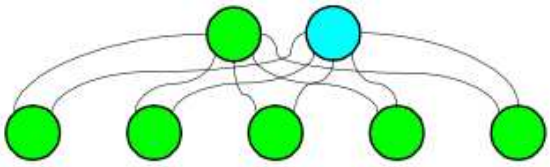

Figure 6.11 - connection of nodes using redundant switches

Many missions also need redundant instruments as well as redundant switches, and the harness mass increases again. Effectively this topology provides a 2x2 crossconnect between each node (both nominal and redundant) and each routing switch.

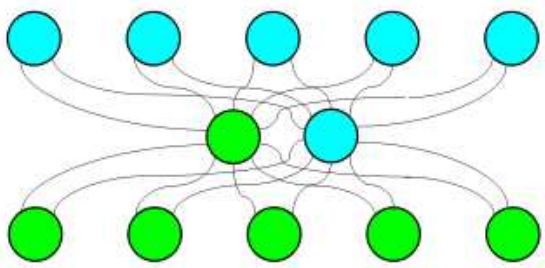

Figure 6.12 - connection of nodes and redundant copies

6.6.5.5. Multiple Router Configurations - Hybrid Networks

It has been seen that rings provide low harness mass at the expense of performance and centralised switches provide performance at a major cost in harness mass. Is it possible to mix the ring topology (where performance is less critical) with the centralised switch (where performance is needed)? SpaceWire does indeed provide the opportunity for such mixed or hybrid networks. In figure 6.13, the left nodes represent processors or mass memories or high-bandwidth instruments that need the dedicated connection to each routing switch. The remaining nodes do not need such high performance and so can be connected in a ring, with taps from the ring going to the switches. The number of SpaceWire links is not changed from the previous example but eight long connections to the central switches have been replaced by eight much shorter connections to make the rings.

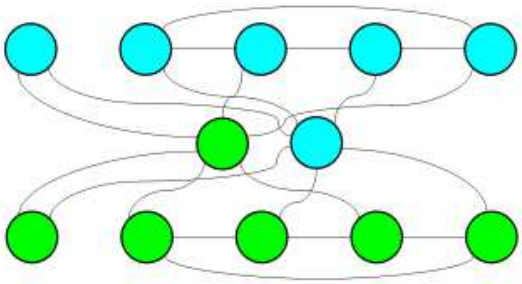

Figure 6.13 - hybrid connection of nodes and redundant copies

Many variations on this theme are possible and a trade-off study is likely to show for a particular mission that an alternative is preferable. In this example, we have chosen different connection techniques to balance between performance and harness mass, another balance might be with other parameters such as fault-tolerance, where some parts of the satellite may be more critical than others.

6.6.5.6. Cube-connected Cycles [Hypercubes]

One family of regular structures that can often be a useful basis for a design are the family of hypercubes, whose nodes all have a degree of n and where progressing from a hypercube of degree n to a hypercube of degree $n+1$ simply involves copying the original structure and joining each node to its copy. Hypercubes have a low diameter for a given value of n - although it can be bettered in some cases, for instance the Petersen graph. Routing algorithms are easy to develop for hypercubes. It is also often easier to split a hypercube across multiple boards on a back-plane than for many less-regular structures.

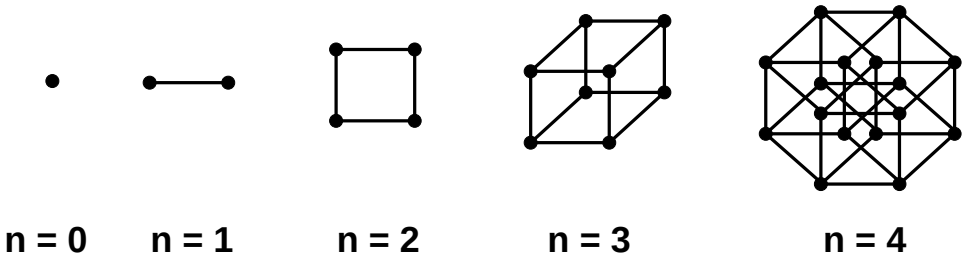

Figure 6.13 - hypercube connection of nodes

Two references that summarise results that are relevant for SpaceWire are [Thompson 1997] and [May 1993].

Further material on the network architectures outlined in this section may be found in [Cook 2007a].

6.7. Using the RMAP Protocol

The SpaceWire Remote Memory Access Protocol is frequently used for remote memory accesses, but is also used to stream data to and from FIFO-buffered instruments. It can also be used to send commands to instruments, typically by updating their registers, and to request responses in a similar fashion.

This section discusses how the RMAP protocol might be used in various circumstances:

- RMAP writes can be accompanied by a request for the target node to return an acknowledgement if the write data's CRC was successfully verified and subsequently that the write was successful. There is, of course, no guarantee that the acknowledgement will successfully be returned to the initiator node. In this case, the only way to determine that the write took place is to re-read the data using another RMAP transaction, and check it back at the initiating node.
- RMAP read-modify-write (RMW) commands are even more problemmatic. If they are implemented at all [since their implementation is optional], the maximum data size of a read-modify-write command is 4 bytes. Many errors in the RMW command are detected before execution and then an error indication will be returned. However, if no response is received for an RMW request, there is no way to tell whether the initial request was corrupted and lost, or whether the response itself was lost. Since RMW changes the memory in the target device, it cannot simply be sent again, and recourse must be made to the data read by the previous RMW command. In this case, it is likely to be easier to maintain the state of the memory in the target device back in the initiating node, and simply write updates to the target when appropriate - and these writes are retryable.

  For similar reasons, using RMAP RMW commands to implement crucial synchronisation primitives such as semaphores is likely to require extreme care in the presence of link errors. Relying on an implementation to perform read-modify-write in an atomic manner will also require careful research.

  Note that there is no definition in the Standard of the semantics of the RMW operation - not even that it will be read/modify/write! A buffer is only needed for a validated write (which is intended to be used only for small, single-digit-byte, messages). Non-validated

writes can be checked and an error message returned without a large buffer - although bad data will have been written by the time that this is reported - which may not be acceptable.

- Implementing the SpaceWire RMAP protocol in a server that is built into hardware is a very popular technique. The protocol requires a lot of checks to be performed when an RMAP command packet is received. The CRC validation of the write data requires a write buffer that is capable of storing the largest allowable packet.

  Another question that should be addressed during the design of an RMAP implementation for an instrument controller is the maximum time that the server should take before it returns its RMAP response. This should not be a problem for simple memory accesses. For commands that trigger data-capture actions in an instrument, the time taken for a complete response might be too long for an application to wait, or too long for the response to fit into a packet transmission schedule. In this case, it might be better to return an RMAP response of the initial instrument command, and allow a further command to poll for the result some time later.

- As suggested above, handling concurrency issues - such as guaranteeing the receipt of exchanged data, and managing semaphores - is really complicated. The implementation of operations such as these can be very difficult to verify, and faults can take a long time to discover and to debug.

  The RMAP protocol itself is client / server in nature, and there are simple design patterns which ensure that application-level uses of the client / server paradigm are provably deadlock-free when errors do not occur. Guaranteeing that such applications are deadlock-free in the presence of packet loss is more challenging.

- The RMAP command packet overhead is its primary performance issue.

6.8. FDIR (Failure Detection, Isolation and Recovery)

We have seen in section 6.6.5, “Routed Networks”, how static network configurations can be used to provide a communication structure for a spacecraft that is operating normally. We have also seen how static configurations rely solely on group adaptive routing for fault tolerance. This is a powerful technique, as fault detection and fail-over to an alternate path is very fast - of the order of micro-seconds.

However, such static configurations may not be able to deliver the necessary level of fault-tolerance. The underlying difficulty is that a fault changes the network, maybe destroying some of its regularity, and the network is only static until an event that it is intended to handle occurs. Faults make the network dynamic. Recovery from faults is also likely to be dynamic - powering-up a cold-redundant switch, for instance.

We therefore need to be able to manage changing network configurations. Three stages are involved:

- Discover that a communications failure has occurred;
- Probe the network to identify the currently-operational switches, connections and devices;

- Select routes and configure devices and switches.

These stages are repeated either at regular (not necessarily frequent) intervals or whenever a change is detected (by, for example, the unexpected behavior of a communication).

Needs a reference to the N-MaSS (FDIR) project.

6.8.1. Failure Detection

SpaceWire has many useful features that can indicate network faults:

- A link fails to connect if the initialisation state machine is unable to progress to the ‘connected’ state.
- Link timeouts occur whenever transitions in the DS signals are too infrequent and a bit is not received within 850 ns (±10%).
- A parity bit is contained in each transmitted character, including FCTs and NULLs, so corruption on the wire is likely to result in a parity error.
- Error conditions in a transmitting node can result in an Error End of Packet (EEP) being sent, which either leads to link disconnection or to the transmission of an EEP without a disconnect, depending on whether the error is observed at the failing link or on a downstream link in a network, respectively.
- Mis-routing (maybe caused by faulty physical addresses, or an incorrect logical address table), or traffic blockage in a router (maybe caused by a receiving node failing) can be detected and the packet truncated. This may introduce an EEP throughout the rest of the receive chain and cause that links to disconnect, as above.
- Higher-level protocols (such as SpaceWire RMAP - see section) contain header and data CRC fields. Should these CRCs be found to be faulty, this is an indication of a problem - either low-level character transmission or a higher-level packet assembly issue.

Detection of these faults can be used to trigger a cycle of FDIR activity.

Higher-level protocols can manage their own recovery in some circumstances - such as the TCP protocol’s retransmission mechanism - but after several retry attempts have failed, the TCP stack can report the issue to the FDIR infrastructure for lower-level repair.

6.8.2. Fault Isolation

When a fault is detected, FDIR logic must identify what has failed. At this time, it might be necessary to stop all transmissions, re-configure the network statically and then re-start everything, or it might be possible to make smaller changes to route around the failure without stopping most of the nodes.

Each active component in the network must be able to be probed and respond with information to assist the discovery process. Exactly what information is required is a matter of some judgment - a minimal set reduces initial traffic and is most easily implemented, but a more complete reply may reduce the total number of packets required.

An initial probe of a component should return:

- Whether the component is a switch or a device;
- How many ports are present on the component;
- Which port on the component received the request [the source port is already known, of course];
- The unique identifier of the component [to detect loops].

With this information, a complete map of discoverable components can be constructed.

6.8.3. Fault Recovery

Armed with a map of the current network, together with knowledge of what data flows each component wishes to maintain, a new set of routing tables (for logical addressing) or source routes (for path addressing) can be constructed. This might not be a simple task, because it has to take into account the bandwidths of the remaining links, the relay capabilities of the nodes, and so on.

There is also a possibility that the communication patterns of the original network can not be supported in the new network; deadlock cycles might be created unless mitigations are applied.

Dynamic routing algorithms, that change the routing structure on-the-fly, must also ensure that they do not make the network susceptible to deadlock while it is partially re-configured. Of course, such failures might be rectified by the FDIR scheme, but this is a more heavy-weight mechanism than is required here.

6.9. Plug-and-Play configuration

“Plug and Play” is a term with several interpretations.

One aspect of “Plug and Play” is a technique for dynamically managing the communications amongst modules that are brought together as pre-fabricated, maybe off-the-shelf components. Responsive space missions, for small satellites, or for the Space Shuttle / CEV / ISS, can integrate off-the-shelf and custom modules using SpaceWire for the interconnections.

The function of these modules can be known in advance of the integration, in which case the network discovery and route-creation of FDIR, as shown in section 6.8, “FDIR (Failure Detection, Isolation and Recovery)”, will provide the configuration mechanism.

An enhanced form of “Plug and Play” provides an electronic datasheet function to the modules, so that their precise specification can be determined by the integration tools. In this case, the software interfaces, module communications, calibration and QA functions could all be handled automatically.

Needs more - references, links to implementations, etc.

6.10. Backplanes

SpaceWire backplanes are typically used to support a number of modules, each containing one or more SpaceWire nodes. A backplane might be designed for a single spacecraft mission to couple a bespoke set of modules, or it might be general-purpose in nature and support a population of modules with similar interface layouts and capabilities.

Issues relating to backplane design include:

- The selection of the backplane connectors is key. What are the termination impedance and differential signal performance requirements, and the board mounting method [press-fit, surface-mount or through-hole]? Are the optimum choices available in space-qualified versions? Connectors will also be required for power distribution.
- What are the connection bandwidths available to each module? Should one use SpaceWire on the backplane, or something faster?
- Should one place the router components on the backplane, or distribute them across all of the modules?
  - On the backplane, finding the area for the router device(s) is difficult - this area is also needed for the tracks that carry the links.
  - Spread across the modules, the router must remain functional even if one or more of the modules are unplugged from the backplane. The internal bandwidth between the router components must be sufficient, and this might pose challenges for the connectors and for the tracking on the backplane itself.
- Configuration of the router must be co-ordinated at a single point, even if the router itself is distributed. Further failure detection, fault isolation and system recovery techniques rely on communications across the backplane, possibly using the SpaceWire RMAP protocol, and reconfiguration of that backplane under failure conditions.
- Redundant routers are likely to be required to support redundant modules. Redundant power supplies and power supply connections will be required too.

For further reading, see [Senior 2010a and Senior 2010b].

6.11. SpaceWire D vs Timed Ethernet

Both the draft SpaceWire D and Timed Ethernet are effectively shared bus technologies. SpaceWire can implement each point-to-point link in a network as a bus, but this hides the system-wide benefits of timed comms.

6.12. SpaceWire Verification and Validation

Bearing in mind the stringent quality requirements of the space industry, many users have requested validation equipment to certify their SpaceWire designs, to certify that these designs meet the detailed specifications of the SpaceWire Standard, and to state that their systems are “correct”.

There are several dangers in attempting to validate a SpaceWire design, to certify that the designs meets the detailed specifications of the SpaceWire Standard, or to state that a system is “correct”.

The first is that it is impossible to test most systems that contain internal state just from the exterior (black box testing), because it is impossible to step those designs through all of their internal states in order to check them completely. In addition, even if it were possible to explore every single state of such a system, the number of states might be so large as to make this task impossible. Even a system with an *n*-bit integer variable contains 2^n possible binary states when each of its values is considered separately. Any realistic system is therefore going to be impossible to test exhaustively.

Secondly, communication equipment such as SpaceWire inevitably contains some analog signals whose integrity is dependent upon the instantaneous values of electrical noise, of the signal loading from the test equipment, of errors induced by cosmic rays, and of other such imponderables. It is clearly impossible at the black-box level to test these. If one has access to the circuitry, fault injection could be used at the bit-level to test any triple modular redundancy (TMR) functions of the hardware.

The best that one can therefore achieve when testing a SpaceWire device or system is therefore :

- to explore all of the expected states of the system that can be reached through normal operation by exercising it using a range of values that are allowed in the SpaceWire specifications;
- to perform boundary testing by exploring ranges of situations that fall just outside of those allowed by the SpaceWire standards;
- to undertake stress testing by transmitting and receiving vast quantities of traffic and checking for errors;
- to explore the timing relationships between events, such as the near-simultaneous starting and ending of packets at each port of a SpaceWire router;
- to fuzz-test the system by subjecting it to a range of seemingly random values that ought not to cause it to fail;
- to run experiments that exercise the system under test within the electrical envelope of the SpaceWire standards, as well as some that use electrical conditions just outside of the SpaceWire Standard that one could hopefully expect the system to handle gracefully;
- ... but bear in mind that it is impossible to build an exhaustive set of these circumstances and that test coverage can therefore never reach 100%.

Testing therefore attempts to provide counterexamples of when a system does not successfully meet the standards. Whilst it is possible to do this very successfully, it is impossible to guarantee to find all embedded failures.

Obviously, in white box testing, when one has access to the internal design and state of a system, it is possible to perform much more exhaustive testing of those internal states than black-box testing can achieve.

Whilst it might be possible to build an excellent "Conformance Tester" unit, this could never be guaranteed to identify all the faults of a system-under-test and therefore should never claim that it has validated that system to the SpaceWire Standard - or to any other standard, for that matter.

A thorough design review, including input from an engineer with a thorough understanding of metastability issues, should be considered essential.

6.12.1. Testing the Physical and Signal Layers

Assuming that the SpaceWire network being tested has been constructed using cabling and connectors that meet the SpaceWire Standard, testing the physical layer mainly involves testing for continuity in all the conductors, testing that the SpaceWire I/O buffers present the correct impedances in each logic state, and ensuring that the network is complete.

It might also be advisable to use techniques such as time domain reflectometry (TDR) to ensure that all of the cabling and printed circuit tracking meets its impedance specifications, and eye diagrams to ensure that assumptions about cable skew and jitter are correct. For instance, the large mismatch introduced by the SpaceWire Micro-D connectors would show up clearly on a TDR plot. Skew and jitter can be estimated from eye diagrams of the received D and S signals on any link interface. Some good examples of both of these techniques are given in [Allen 2006] and [Mueller 2006].

Furthermore, we could imagine using a buffered break-out board to measure the quality of an LVDS signal less intrusively than if the signal were to be probed directly.

Many tests are possible to qualify the signal layer. These include:

- injecting signals of known quality and of known degradation into a SpaceWire node and checking that all of the signal layer properties behave as expected.
- undertaking a margins analysis, by imposing corrupting influences such as electrical and radiated noise, common mode voltages and out-of-specification wire impedances on top of the normal signals, to ensure that they can still be recovered reliably.
- performing bit-error rate soak testing to ensure that data demodulation is as reliable as the parameters measured above would suggest.

Sections 12.2.4 and 12.2.5 of the ECSS-E-ST-50-12C SpaceWire Standard provide a summary of all of the conformance items in the signal layer.

6.12.2. Testing the Higher SpaceWire Protocol Layers

The character and exchange levels of the SpaceWire protocol can be soak-tested by streaming synthetic traffic across each SpaceWire link. Whilst this can be rigorous for the main protocol paths, there are many other aspects of low-level SpaceWire transmission that can also be explored:

- Flow control token (FCT) generation, especially in unusual conditions such as when zero-length packets are received [EOP-terminated as well as EEP-terminated]. What happens if too many FCTs are received? Or too few?

- When time-codes are received - both in-sequence and out-of-sequence - are they handled properly?
- When parity errors occur, is the link disabled, and is the higher-level software properly informed?
- When link recovery is provoked after a condition that causes the link to be temporarily disabled. Do all of the states in the SpaceWire state machine trigger at the right times? Similarly, does a link always start reliably?
- When incorrect escape sequences [escape errors] are received.
- Are time-outs after a period of link inactivity detected properly?

Test equipment that can generate these circumstances, and record what a unit-under-test does if / when they are detected, will be necessary for all of the erroneous behaviours; most of these conditions cannot be generated using nominal hardware.

There are many aspects of the network layer that can be exercised:

- Addressing from each input port of the route to each output port
- Writing logical addresses to all locations in the logical addressing table and then propagating data through a route under control of all of these routing table entries
- Testing header deletion on the output of the router;
- Testing the optional header deletion in a router;
- Testing the buffering and blocking properties of the router when operating in the wormhole mode;
- Testing the behaviour when a router times out a packet because traffic has been blocked for too long on one of its output ports. This an optional 10X router function.
- Testing how the router spills packets that block or are otherwise incorrect;
- Checking the router arbitration mechanisms that ensure that particular operations are performed in preference to others.

6.12.3. Testing Higher-level SpaceWire Protocols

Passing RMAP protocol traffic across the SpaceWire network under test is particularly convenient because all RMAP headers contain a CRC error check. All of the RMAP data fields (in RMAP memory read responses and in RMAP memory write commands) are also CRC-protected. As a consequence, most low-level SpaceWire communication issues will be detected as wrongly checksummed RMAP packets or as incomplete RMAP client-server responses. Be aware that the RMAP CRC algorithm returns a value of zero for the CRC byte if all of the bytes included in the CRC are themselves zero - so a block of improperly zeroed memory included in an RMAP data field CRCs correctly.

If the RMAP target is being used as a remote memory device, it will be necessary to provide a good memory testing program to access this memory using the RMAP protocol, with the primary intention of testing the memory itself rather just than the SpaceWire RMAP protocol. Such a memory test should be designed to :

- test the integrity of all signals on the data bus, ensuring that each is capable of transmitting and receiving correctly with no shorts or crosstalk-coupled interactions between those lines. Crosstalk might be observed if any of the data wires are disconnected but just capacitatively coupled to their neighbours.
- test the address bus of the RMAP-controlled memory by writing to addresses that increase by powers of two to determine the size of the memory area before wrap-around occurs.
- run soak tests on the memory by writing blocks of data to the memory, filling up the area to be tested completely, and then reading back the values stored and checking them against the values that were initially written. This mechanism ensures that any words that are stored in the same location as other words due to addressing faults are detected. Repeated cycles of these tests could use fixed data patterns or pseudo-random sequences, depending on what the test is expected to reveal.

This memory test can be used throughout the integration, testing and flight phases of the mission and is also a good test of the RMAP protocol and of the remainder of the SpaceWire communications path. The memory being tested will be run at SpaceWire network speeds, rather than its native data rate, so further memory tests, running under control of the spacecraft computer, might be more challenging - but these will not exercise the SpaceWire link to any extent.

6.12.4. System-Level Testing

Most tests on the overall system will involve passing large amounts of traffic - actual or synthetic - through paths from one node to another through all of the intermediate routers. Since these tests are primarily tests of the communication infrastructure, it is not necessary to emulate the data traffic of the applications completely. Clearly, however, testing the system with traffic that closely emulates in-service traffic is likely to identify more faults or expose more weaknesses than testing with data of a very different structure.

Sometimes it is convenient to perform long term soak-tests of the SpaceWire network by using synthesised traffic made up from pseudorandom sequences or the output of linear feedback shift register sequences. Selection of synthetic traffic that can be generated identically at the transmitter and at the receiver nodes simplifies the checking for corruption.

Being able to record and subsequently reconstruct the link traffic across a number of SpaceWire links is an important diagnostic tool. This requires time-tags to be collected with each packet in order to correlate them together. Detailed time-tagging of traffic passing through a router is particularly effective at detecting a range of buffering, timeout and arbitration behaviours.

7. Supporting Components

Is this section just for ESA-sponsored products, or will the Working Group request all of the relevant companies to submit 'advertisements' for their products? The latter contributions will be far more bulky than the rest of the Handbook, will become out-of-date fast, and are probably best accommodated on ESA's SpaceWire website?

7.1. SpaceWire Circuits

7.1.1. Integrated Circuits

7.1.2. IP (Intellectual Property) Cores

7.1.3. Boards and spacecraft processors

7.2. SpaceWire Prototyping products

7.2.1. Bridges and Gateways

7.2.2. Routers

7.3. Test Equipment

7.3.1. Electrical conformance / physical-layer compliance testers

7.3.2. Low-level protocol analysers

7.3.3. Traffic Recorders

8. Bibliography

The normative documents relating to the SpaceWire protocols are listed in section 3, “References”.

Other documents referenced in this Handbook include:

[Allen 2006]
S. Allen, “SpaceWire Physical Layer Issues”, MAPLD International Conference, Sept 2006.

[Cook 2003]
B.M. Cook : “Reducing SpaceWire Time-code Jitter”, Presented at the International SpaceWire Symposium ESTEC, Noordwijk, November 2003.

[Cook 2007a]
B.M. Cook & C.P.H. Walker: “SpaceWire Network Topologies”, International SpaceWire Conference, 2007.

[Cook 2007b]
B.M. Cook & C.P.H. Walker: “SpaceWire Plug-and-Play: An Early Implementation and Lessons Learned”, Proceedings of the AIAA Infotech Conference, 7-9 May 2007, Rohnert Park, CA

[Cook 2007c]
B.M. Cook & C.P.H. Walker: “Reducing Electromagnetic Emissions from SpaceWire”, DASIA, 2007

[Cook 2008a]
B.M. Cook, W. Gasti & S. Landstroem : “SpaceWire Physical Layer Fault Isolation”, International SpaceWire Conference 2008.

[Cook 2008b]
B.M. Cook & C.P.H. Walker : “SpaceWire on FPGA - Challenges and Solutions”, Presented at DASIA conference, Palma, Majorca, Spain, 2008.

[Dean 2008]
B. Dean, R. Warren, and B. Boyes, “RMAP over SpaceWire on the ExoMars Rover for Direct Memory Access by Instruments to Mass Memory”, 2nd International SpaceWire Conference, Nara, Japan, Nov 2008.

[EADS 2009]
EADS Astrium GmbH: “STUP SpaceWire Protocol”, EADS Astrium GmbH, July 2009.

[Fronterhouse 2007]
D. Fronterhouse and J. Lyke, “Plug-and-Play Satellite (PnPSat) Demonstrating the Vision”, Proceedings of 1st International SpaceWire Conference, Dundee, Scotland, Sept 2007.

[Furano 2011]
G. Furano: “Command & Control Buses as enabler for modular, reconfigurable spacecrafts: present & future” Presented at NASA/ESA Conference on Adaptive Hardware and Systems, San Diego. Available at

http://www.see.ed.ac.uk/ahs2011/Tutorial_Presentations/tutorial_AHS11_Furano.pdf

[Ginosar 2003]
R. Ginosar: “Fourteen Ways to Fool Your Synchronizer”, in the Proceedings of the Ninth International Symposium on Asynchronous Circuits and Systems (ASYNC’03), IEEE, 2003.

[Ginosar 2011]
R. Ginosar: “Metastability and Synchronizers : A Tutorial”, IEEE Design & Test, Sept/Oct 2011.

[Haas 1998]
S. Haas, D.A Thornly, M. Zhu, R.W. Dobinson and B. Martin: “The Macramé 1024-Node Switching Network”, Microprocessors and Microsystems, IEEE 1355 Special Issue, V21, Nos 7,8, 30 March 1998.

[Ilstad 2007]
J. Ilstad, W. Gasti, P. Sinander, and S. Habinc, “SpaceWire Remote Terminal Controller”, Proceedings of International SpaceWire Conference, Dundee, Sept 2007.

[INMOS 1985]
INMOS Limited: “IMS T424 Transputer Reference Manual”, Bristol, England, 1985

[Jaff 2007]
P. Jaff, G. Clifford and J. Summers, “SpaceWire for Operationally Responsive Space as part of Tacsat-4”, Proceedings of 1st International SpaceWire Conference, Dundee, Scotland, Sept 2007.

[Johnson 1993]
H. Johnson & M. Graham : “High-Speed Digital Design”, Prentice Hall, ISBN 0-13-395724-1, 1993.

[Klar 2008]
R. Klar et al: “Design Considerations for Adapting Legacy System Architectures to SpaceWire”, International SpaceWire Conference, 2008.

[Larsen 2011]
J. Larsen: “A 1553 to SpaceWire Bridge”, International SpaceWire Conference, 2011.

[May 1993]
M.D. May, P.W. Thompson & P.H. Welch (eds) : “Networks, Routers & Transputers : Function, Performance & Application”, IOS Press, Amsterdam (ISBN 90-5199-129-0), 1993.

[McNutt 2009]
C.J. McNutt et al: “Modular Nanosatellites - Plug-and-Play (PnP) CubeSat”, 7th AIAA Responsive Space Conference, 2009.

[MIL-STD-1553B]
MIL-STD-1553b: “Military Standard Aircraft Internal Time Division Command/Response Multiplex Data Bus”, United States Department of Defense, 21 September 1978.

[Mills 2009]
S. Mills, S.M. Parkes and N. O'Gribin, "SpaceWire Data-handling with RMAP", Data Systems in Aerospace (DASIA), 2007, ISBN 92-9092-202-8.

[Mueller 2006]
J. Mueller, "Design Challenges of an Advanced SpaceWire Assembly for High Speed Inter-Unit Data Link", MAPLD International Conference, Sept 2006.

[NASA 1998]
NASA, "Electrical Grounding Architecture for Unmanned Spacecraft", NASA Technical Handbook NASA-HDBK-4001, Feb 1998.

[NASA 2004]
NASA: "The Vision for Space Exploration", 2004. Available at
http://www.nasa.gov/pdf/55583main_vision_space_exploration2.pdf

[NASA 2005]
NASA: "GOES-R Reliable Data Delivery Protocol (GRDDP)", July 2005. Available at
http://spacewire.esa.int/content/Standard/documents/417-R-RPT-0050_GRDDP_Rev2.1.pdf

[Parkes 2007]
S.M. Parkes, C. McClements, G. Kempf, S. Fischer, P. Fabry and A. Leon, "SpaceWire Router ASIC", Proceedings of 1st International SpaceWire Conference, Dundee, Scotland, Sept 2007.

[Parkes 2011]
S. Parkes, P. Armbruster & M. Suess: "The SpaceWire on-board data-handling network", ESA-Bulletin, v45, pp 35-45, February 2011.

[Parkes 2012]
S. Parkes, "SpaceWire User's Guide", STAR-Dundee Limited, 2012.

[Ross 2012]
H.M. Ross, "Pegasus the Seminal Early Computer", Bright Pen, ISBN 978-0-7552-1482-2, 2012.

[Rouchaud 2011]
G. Rouchaud, J. Ilstad and F. Mettendorff: "Low Mass SpaceWire", International SpaceWire Conference, 2011.

[Senior 2010a]
A. Senior, P. Worsfold & W. Gasti: "Evolution of the MARC SpaceWire and Power Distribution Architecture from Concept to Tested Hardware", International SpaceWire Conference 2010.

[Senior 2010b]
A. Senior, P. Worsfold & W. Gasti: "A SpaceWire Active Backplane Specification for Space Systems", International SpaceWire Conference 2010.

[Thompson 1997]
P. Thompson, A.M. Jones, N.J. Davies, M.A. Firth & C.J. Wright (Eds.) "The Network Designer's Handbook", Volume 51: Concurrent Systems Engineering Series, IOS Press, ISBN: 978-90-5199-380-6, 1997.

[Walker 2003]
C.P.H. Walker : “The Origins of SpaceWire”, Presented at the International SpaceWire Symposium, ESTEC, Noordwijk, November 2003.

Publications from the International SpaceWire Conferences and the SpaceWire Working Group meetings are available from the ESA SpaceWire website at **`http://www.spacewire.esa.int`**.

