ECSS-E-50-12A
24 January 2003

EUROPEAN COOPERATION

ECSS

FOR SPACE STANDARDIZATION

Space engineering

SpaceWire - Links, nodes, routers and networks

ECSS Secretariat
ESA-ESTEC
Requirements & Standards Division
Noordwijk, The Netherlands

Published by:	ESA Publications Division ESTEC, P.O. Box 299, 2200 AG Noordwijk, The Netherlands
ISSN:	1028-396X
Price:	€ 30

Printed in The Netherlands

Foreword

This Standard is one of the series of ECSS Standards intended to be applied together for the management, engineering and product assurance in space projects and applications. ECSS is a cooperative effort of the European Space Agency, national space agencies and European industry associations for the purpose of developing and maintaining common standards.

Requirements in this Standard are defined in terms of what shall be accomplished, rather than in terms of how to organize and perform the necessary work. This allows existing organizational structures and methods to be applied where they are effective, and for the structures and methods to evolve as necessary without rewriting the standards.

The formulation of this Standard takes into account the existing ISO 9000 family of documents.

This Standard has been prepared by the ECSS E-50-12 Working Group, reviewed by the ECSS Technical Panel and approved by the ECSS Steering Board.

(This page is intentionally left blank)

Introduction

General

SpaceWire technology has grown organically from the needs of on-board processing applications. This Standard provides a formal basis for the exploitation of SpaceWire in a wide range of future on-board processing systems.

One of the principal aims of SpaceWire is the support of equipment compatibility and reuse at both the component and subsystem levels. In principle a data-handling system developed for an optical instrument, for example, can be used for a radar instrument by unplugging the optical sensor and plugging in the radar one. Processing units, mass-memory units and down-link telemetry systems developed for one mission can be readily used on another mission, reducing the cost of development, improving reliability and most importantly increasing the amount of scientific work that can be achieved within a limited budget.

Integration and test of complex on-board systems is also supported by SpaceWire with ground support equipment plugging directly into the on-board data-handling system. Monitoring and testing can be carried out with a seamless interface into the on-board system.

SpaceWire is the result of the efforts of many individuals within the European Space Agency, European Space Industry and Academia.

Purpose

This Standard addresses the handling of payload data and control information on board a spacecraft. It is a standard for a high speed data link, which is intended to meet the needs of future, high capability, remote sensing instruments and other space missions. SpaceWire provides a unified high speed data-handling infrastructure for connecting together sensors, processing elements, mass-memory units, downlink telemetry subsystems and EGSE equipment.

The purpose of this Standard is:

- to facilitate the construction of high-performance on-board data-handling systems;
- to help reduce system integration costs;
- to promote compatibility between data-handling equipment and subsystems;
- to encourage reuse of data-handling equipment across several different missions.

SpaceWire has taken into consideration two existing standards, IEEE 1355-1995 and ANSI/TIA/EIA-644. SpaceWire is specifically provided for use onboard a spacecraft.

Guide to this Standard

This Standard begins with clause 1 which introduces the scope of the Standard. clause 2 then gives a list of applicable documents. clause 3 provides the necessary definitions of terms and abbreviations and explains the notation used throughout the document. A brief overview of the Standard is given in clause 4 to familiarize the reader with the basic SpaceWire concepts, prior to the detailed specification of subsequent clauses.

The body of this Standard is presented in clauses 5 to 11, which ascend through the various normative levels of the Standard.

Clause 5 (Physical Level) covers cables, connectors, cable assemblies and printed circuit board tracks.

Clause 6 (Signal Level) deals principally with electrical characteristics, and coding and signal timing.

Clause 7 (Character Level) describes how data and control characters are encoded.

Clause 8 (Exchange Level) presents the way in which a SpaceWire link operates including link initialization, normal operation, error detection and error recovery.

Clause 9 (Packet Level) describes the way in which data is encapsulated in packets for transfer across a SpaceWire network.

Clause 10 (Network Level) deals with the structure and operation of a SpaceWire network.

The error recovery scheme is described as a whole in clause 11, which brings together the error detection, error recovery and error reporting mechanisms from all the protocol levels to aid comprehension.

This Standard concludes in clause 12 with a list of conformance statements, highlighting those parts of the Standard to conform to for SpaceWire compatibility of a system.

There are three annexes:

- Annex A: The differences between this Standard and IEEE Standard 1355-1995 [1].
- Annex B: State exit conditions for encoder-decoder state machine.
- Annex C: Availability of referenced documents including web addresses for electronic versions.

Finally, a list of informative references is included in the Bibliography.

Disclaimer – intellectual property

The implementation of this Standard can touch on intellectual property covered by patent rights. ECSS is not responsible for identifying all the patents involving a license to implement the SpaceWire Standard. Furthermore, ESA is not responsible for ensuring the existence or legal validity of any patent related to the SpaceWire Standard.

Contents

Figures

Tables

(This page is intentionally left blank)

1 Scope

This Standard specifies the physical interconnection media and data communication protocols to enable the reliable sending of data at high-speed (between 2 Mb/s and 400 Mb/s) from one unit to another. SpaceWire links are full-duplex, point-to-point, serial data communication links.

The scope of this Standard is the physical connectors and cables, electrical properties, and logical protocols that comprise the SpaceWire data link. SpaceWire provides a means of sending packets of information from a source node to a specified destination node. SpaceWire does not specify the contents of the packets of information.

This Standard covers the following protocol levels:

- Physical level: Defines connectors, cables, cable assemblies and printed circuit board tracks.
- Signal level: Defines signal encoding, voltage levels, noise margins, and data signalling rates.
- Character level: Defines the data and control characters used to manage the flow of data across a link.
- Exchange level: Defines the protocol for link initialization, flow control, link error detection and link error recovery.
- Packet level: Defines how data for transmission over a SpaceWire link is split up into packets.
- Network level: Defines the structure of a SpaceWire network and the way in which packets are transferred from a source node to a destination node across a network. It also defines how link errors and network level errors are handled.

(This page is intentionally left blank)

2 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this ECSS Standard. For dated references, subsequent amendments to, or revisions of any of these publications do not apply. However, parties to agreements based on this ECSS Standard are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references the latest edition of the publication referred to applies.

ECSS-P-00	Standardization policy
ECSS-P-001	Glossary of terms
ECSS-E-00	Space engineering — Policy and principles
ECSS-E-50 [1]	Space engineering — Communications
ECSS-Q-70-08	Space product assurance — The manual soldering of high-reliability electrical connections
ECSS-Q-70-26	Space product assurance — Crimping of high-reliability electrical connections
ANSI/TIA/EIA-644	1995 Telecommunications Industry Association, “Electrical Characteristics of Low Voltage Differential Signaling (LVDS) Interface Circuits”, March 1996
ESCC 3401/071 [1]	Connectors, Electrical, Rectangular, Microminiature, Solder Buckert Contacts with EMI Backshell, based on type MDM
ESCC 3902/003	Cable, “Spacewire”, Round, Quad using Symmetric Cables, Flexible, –200 to +180 °C

[1] To be published.

(This page is intentionally left blank)

3
Terms, definitions and abbreviated terms

3.1 Terms and definitions

The following terms and definitions are specific to this Standard in the sense that they are complementary or additional with respect to those contained in ECSS-P-001.

3.1.1
acknowledge
indication that a message has been received successfully by its intended destination

3.1.2
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.

3.1.3
bit error rate
ratio of the number of bits received in error to the total number of bits sent across a link

3.1.4
byte
eight bits

3.1.5
cargo
data to encapsulate in packets and transfer from a source to a destination

3.1.6
character
control character or data character

3.1.7
character level
protocol level that deals with the encoding of data and control characters into a bit-stream

3.1.8
coding
translation from one set of bits to another new set of bits

3.1.9
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

3.1.10
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).

3.1.11
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

3.1.12
data character
data byte encoded ready for transfer across a link

3.1.13
data rate
rate at which the application data is transferred across a link

3.1.14
data signalling rate
rate at which the bits constituting control and data characters are transferred across a link

3.1.15
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

3.1.16
decoding
act of translating an encoded set of bits to the original set of bits prior to coding

3.1.17
de-serialization
transformation of a serial bit stream into a sequence of control or data characters

3.1.18
destination
node or unit that a packet is being sent to

3.1.19
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)

3.1.20
destination list
list of destination identifiers which forms the destination address of a packet

3.1.21
destination identifier
address, or partial address, of the packet destination

3.1.22
driver
electronic circuit design to transmit signals across a particular transmission medium

3.1.23
end of packet marker
control character which indicates the end of a packet

3.1.24
error recovery scheme
method for handling errors detected within a SpaceWire link

3.1.25
exchange level
protocol level that defines the mechanisms for link initialization, link flow control, link error detection and link error recovery

3.1.26
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

3.1.27
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

3.1.28
host receive buffer
buffer within a host system for receiving data from a link interface

3.1.29
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.

3.1.30
host transmit buffer
buffer within a host system for holding data prior to transmission through a link interface

3.1.31
input port
receive side of a link interface on a routing switch

3.1.32
jitter
random errors in the timing of a signal

3.1.33
lay-length
number of twists per foot expressed as the length between one complete turn of a single end in the cable

3.1.34
link
bidirectional connection of one unit to another unit for passing data and control information

3.1.35
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).

3.1.36
link destination
end of the link that is receiving a particular set of data or control information

3.1.37
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

3.1.38
link receiver
receiver at one end of a link

3.1.39
link source
end of the link that is sending a particular set of data or control information

3.1.40
link transmitter
transmitter at one end of a link

3.1.41
logical address
data character at the start of a packet, which identifies the destination for the packet

3.1.42
low voltage differential signalling
particular form of differential signalling using low voltage swing signals

3.1.43
Mb/s
1 000 000 bits per second

3.1.44
network
set of units connected together via links and routing switches

3.1.45
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

3.1.46
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

3.1.47
normal-character
data character or control character (EOP or EEP) that is passed from the exchange level to the packet level

3.1.48
NULL
token sent to keep the data link active when there are no data or control characters to send

3.1.49
output port
transmit side of a link interface on a routing switch

3.1.50
packet
sequence of normal-characters comprising a destination address, packet cargo and an end of packet marker

3.1.51
packet level
protocol level that defines how data is organized in packets ready for transfer across a link or network

3.1.52
packet cargo
data to transfer from a source to a destination

3.1.53
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

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

3.1.54
pseudo-ECL (PECL)
emitter-coupled logic (ECL) referenced to +5 V

3.1.55
receiver
electronic circuit designed to receive signals sent across a particular transmission medium

3.1.56
router
routing switch

3.1.57
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

3.1.58
serialization
transformation of a sequence of control or data characters into a serial bit stream

3.1.59
signal
measurable quantity that varies with time to transfer information by propagating along a transmission medium

3.1.60
signal level
protocol level which defines the electrical signals used for SpaceWire together with the data-strobe encoding and signal timing

3.1.61
skew
difference in time between the edges of two signals which should ideally be concurrent

3.1.62
source
node or unit sending a packet

3.1.63
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

3.1.64
transmission medium
medium over which data is transferred e.g. screened twisted pair cables

3.1.65
transmitter
electronic circuit designed to transmit signals across a particular transmission medium

3.1.66
unit
box, board or subsystem, that can have one or more SpaceWire interfaces

3.2 Abbreviated terms

The following abbreviated terms are defined and used within this Standard.

Abbreviation	**Meaning**
ACK	acknowledge
API	application programming interface
AWG	American wire gauge

BER	bit error rate
CAM	content addressable memory
DC	direct current
DMA	direct memory access
DS	data-strobe
DS-DE	data-strobe, differentially ended NOTE Used in IEEE Standard 1355-1995 [1] to indicate a link with differentially encoded data and strobe signals.
DSP	digital signal processing
ECL	emitter-coupled logic
EEP	error end of packet NOTE Used to indicate that an error occurred in the current packet.
EGSE	electronic ground support equipment
EMC	electromagnetic compatibility
EMI	electromagnetic interference
EOP	end of packet marker
ESA	European Space Agency
ESC	escape character NOTE Escape character is defined in the Character Level.
ESD	electrostatic discharge
FCT	flow control token
FIFO	first in first out memory
L-Char	link-character
LSB	least significant bit
LVDS	low voltage differential signalling
Mb/s	Megabits per second
MSB	most significant bit
N-Char	normal-character
PCB	printed circuit board
PECL	pseudo-ECL
PFA	Perfluoral oxide Copolymer NOTE A type of plastic used to cover wires and cables.
PTFE	Polytetrafluroethylene NOTE A type of plastic used to cover wires and cables.
SCI	scaleable coherent interface

3.3 Conventions

3.3.1 Signal naming

All electrical signals are shown in uppercase letters.

The two signals making up a differential pair are given the suffixes + and – to indicate the positive and negative components of the differential signal, respectively.

The SpaceWire differential signals are referred to as D+,D- and S+,S- for Data and Strobe, respectively. When considering the driven end of a SpaceWire link these signals may be designated Dout+, Dout- and Sout+ and Sout- for Data and Strobe, respectively. Similarly the signals at the input end of a SpaceWire link are Din+, Din- and Sin+, Sin-.

3.3.2 Packet formats

Packet formats are represented in two ways in this Standard. The first way is graphical and is shown in Figure 1. The field at the top is the one that is transmitted first.

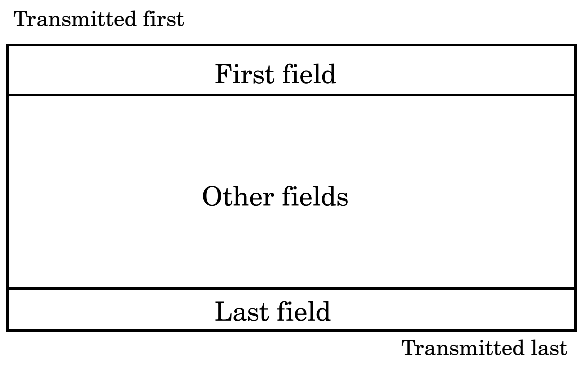


Figure 1: Graphical packet notation

The second packet representation is textual. Each field is enclosed in chevrons <>. The fields comprising a packet are written left to right in the order that they are transmitted. The example below is equivalent to that shown in Figure 1.

EXAMPLE :<First field><Other fields><Last field>

3.3.3 State diagram notation

All state diagrams in this Standard use the style shown in Figure 2. States are represented by ellipses with the state name written inside the ellipse in bold. Actions to take while in a particular state are written in italics inside the ellipse underneath the state name. Transitions from one state to another are indicated by arrows. The event that causes a transition is written alongside the arrow. Unconditional transitions are indicated by arrows without an event name written next to it. Reset conditions are indicated by transition arrows that start in empty space. Transitions can be enabled by a guarded condition so that the transition only takes place if the guard condition is true. Guard conditions are written in square brackets alongside the transition they affect.

State names referred to in the text of the Standard are in italics e.g. *FirstState*.

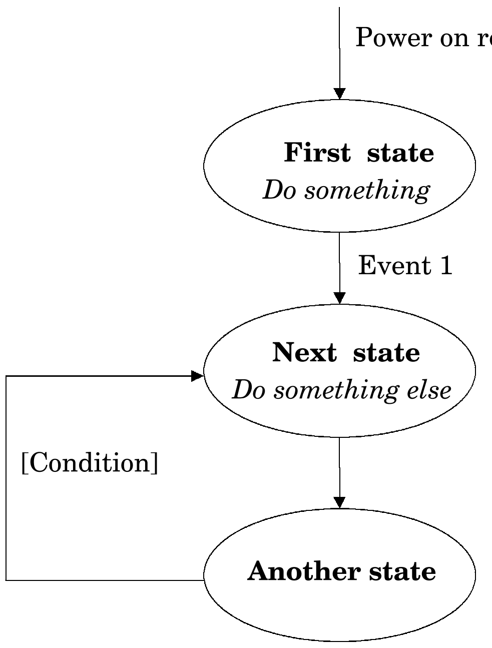


Figure 2: State diagram style

(This page is intentionally left blank)

4 Overall description

4.1 General

This clause provides an overview of the Standard giving the rationale behind key decisions made in the development of the Standard.

SpaceWire takes into consideration the "DS-DE" part of IEEE Standard 1355-1995 [1], as well as ANSI/TIA/EIA-644 and IEEE Standard 1596.3-1996 [2] Low Voltage Differential Signalling (LVDS). See annex A for details of the main differences between SpaceWire and IEEE Standard 1355-1995 and the reasons for those differences.

SpaceWire is a full-duplex, bidirectional, serial, point-to-point data link. It encodes data using two differential signal pairs in each direction. That is a total of eight signal wires, four in each direction.

4.2 Physical level

4.2.1 General

The physical level of the Standard covers cables, connectors, cable assemblies and printed circuit board (PCB) tracks. SpaceWire was developed to meet the EMC specifications of typical spacecraft.

4.2.2 Cables

The SpaceWire cable comprises four twisted pair wires with a separate shield around each twisted pair and an overall shield.

To achieve a high data signalling rate with SpaceWire over distances up to 10 m a cable with the following characteristics is used:

- characteristic impedance matched to the line termination impedance;
- low signal-signal skew between each signal in a differential pair and between Data and Strobe pairs;
- low signal attenuation;
- low crosstalk;
- good EMC performance.

4.2.3 Connectors

The SpaceWire connector has eight signal contacts plus a screen termination contact. A nine-pin micro-miniature D-type is specified as the SpaceWire connector. This type of connector is available qualified for space use.

4.2.4 Cable assemblies

SpaceWire cable assemblies are made from SpaceWire cable up to 10 m in length terminated at each end by nine-pin micro-miniature D-type plugs.

4.2.5 Printed circuit board tracks

SpaceWire includes specifications for running SpaceWire signals over printed circuit boards including backplanes using pairs of tracks with 100 Ω differential impedance.

4.2.6 Electromagnetic compatibility

SpaceWire was developed to meet the electromagnetic compatibility (EMC) specifications of typical spacecraft. EMC testing was performed by Patria Finavitec Oy with support from the University of Dundee following EMC specifications derived from the EMC specifications for the Rosetta [3] and other ESA missions. The testing covered:

- Radiated emission, electric and magnetic fields;
- Radiated susceptibility, electric and magnetic fields;
- Conducted susceptibility;
- Conducted emission;
- Electrostatic discharge;
- Signalling rate;
- Bit error rate;
- Fault isolation; and
- Power consumption.

The EMC test results are provided in [4].

4.3 Signal level

4.3.1 General

The signal level part of this Standard covers signal voltage levels, noise margins and signal encoding.

4.3.2 Signal level and noise margins

Low voltage differential signalling or LVDS (ANSI/TIA/EIA-644) is specified as the signalling technique to use in SpaceWire.

LVDS uses balanced signals to provide very high-speed interconnection using a low voltage swing (350 mV typical). The balanced or differential signalling provides adequate noise margin to enable the use of low voltages in practical systems. Low voltage swing means low power consumption at high speed. LVDS is appropriate for connections between boards in a unit, and unit to unit interconnections over distances of 10 m or more.

The signalling levels used by LVDS are illustrated in Figure 3.

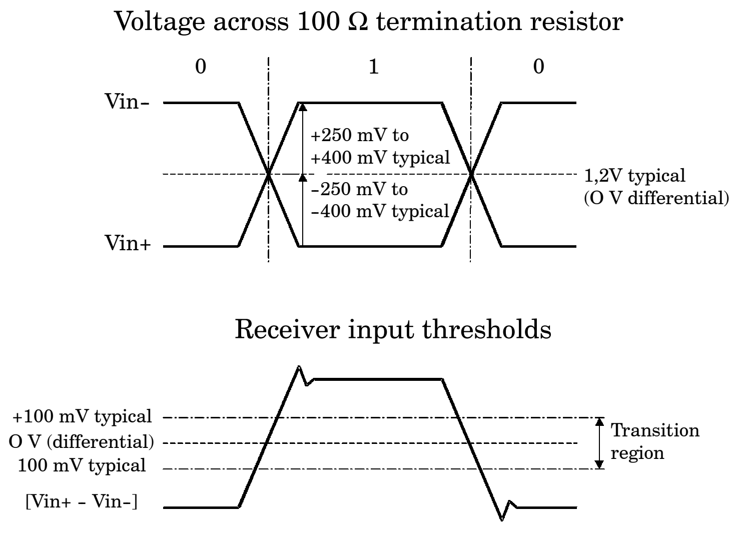


Figure 3: LVDS signalling levels

A typical LVDS driver and receiver are shown in Figure 4, connected by a media (cable or PCB traces) with 100 Ω differential impedance.

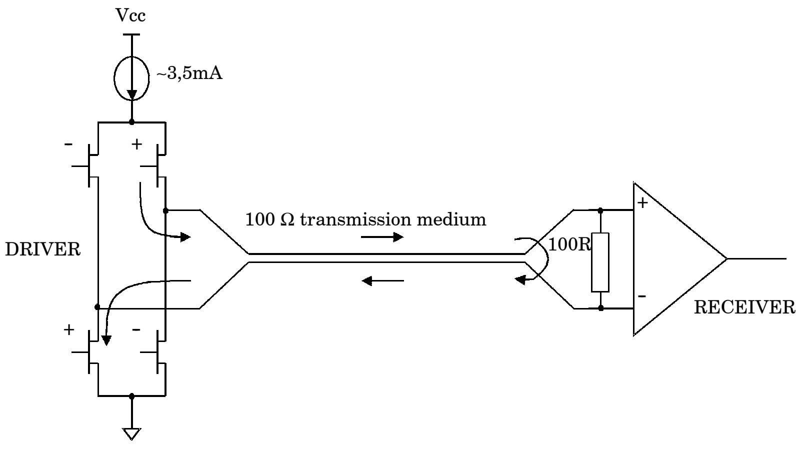


Figure 4: LVDS operation

The LVDS driver uses current mode logic. A constant current source of around 3,5 mA provides the current that flows out of the driver, along the transmission medium, through the 100 Ω termination resistance and back to the driver via the transmission medium. Two pairs of transistor switches in the driver control the direction of the current flow through the termination resistor. When the driver transistors marked "+" in Figure 4 are turned on and those marked "-" are turned off, current flows as indicated by the arrows on the diagram creating a positive voltage across the termination resistor. When the two driver transistors, marked "-", are turned on and those marked "+" are turned off, current flows in the opposite direction producing a negative voltage across the termination resistor. LVDS receivers are specified to have high input impedance so that most of the current flows through the termination resistor to generate around ±350 mV with the nominal 3,5 mA current source.

LVDS has several features that make it very attractive for data signalling [5]:

- Near constant total drive current (+3,5 mA for logic 1 and -3,5 mA for logic 0) which decreases switching noise on power supplies.
- High immunity to ground potential difference between driver and receiver - LVDS can tolerate at least ±1 V ground difference.
- High immunity to induced noise because of differential signalling normally using twisted-pair cable.
- Low EMI because small equal and opposite currents create small electromagnetic fields which tend to cancel one another out.
- Not dependent upon particular device supply voltages.
- Simple 100 Ω termination at receiver.
- Failsafe operation, i.e. the receiver output goes to the high state (inactive) whenever
  - the receiver is powered and the driver is not powered;
  - the inputs are short circuited;
  - input wires are disconnected.
- Power consumption is typically 50 mW per driver - receiver pair for LVDS compared to 120 mW for ECL or PECL.

The following two standards deal with LVDS

a. ANSI/TIA/EIA-644 that defines the driver output characteristics and the receiver input characteristics only.

b. IEEE Standard 1596.3 that defines the signalling levels used and the encoding for packet switching used in SCI data transfers [2].

The signal levels and noise margins for SpaceWire are defined taking into consideration the ANSI/TIA/EIA-644 since this deals with LVDS only whereas IEEE Standard 1596.3 [2] is concerned with the use of LVDS specifically for SCI.

4.3.3 Data encoding

SpaceWire uses Data-Strobe (DS) encoding. This is a coding scheme which encodes the transmission clock with the data into Data and Strobe so that the clock can be recovered by simply XORing the Data and Strobe lines together. The data values are transmitted directly and the strobe signal changes state whenever the data remains constant from one data bit interval to the next. This coding scheme is illustrated in Figure 5. The DS encoding scheme is also used in the IEEE Standard 1355-1995 [1] and IEEE 1394-1995 (Firewire) Standard [6].

The reason for using DS encoding is to improve the skew tolerance to almost 1-bit time, compared to 0,5 bit time for simple data and clock encoding.

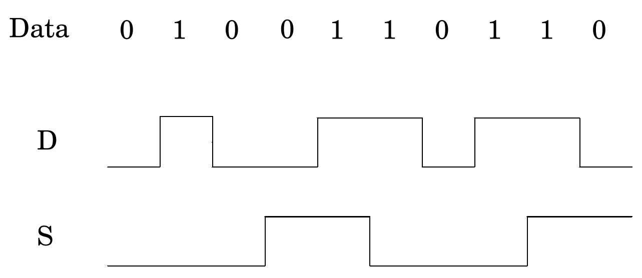


Figure 5: Data-Strobe (DS) encoding

A SpaceWire link comprises two pairs of differential signals, one pair transmitting the D and S signals in one direction and the other pair transmitting D and S in the opposite direction. That is a total of eight wires for each bidirectional link.

4.4 Character level

SpaceWire takes into consideration the character level protocol defined in IEEE Standard 1355-1995 [1], but it additionally includes Time-Codes to support the distribution of system time.

There are two types of characters:

- Data characters which hold an eight-bit data value, transmitted least significant bit first. Each data character contains a parity bit, a data-control flag and the eight bits of data. The parity bit covers the previous eight bits of a data character or two bits of a control character, the current parity bit and the current data-control flag. It is set to produce odd parity so that the total number of 1's in the field covered is an odd number. The data-control flag is set to zero to indicate that the current character is a data character.
- Control characters which hold two control bits. Each control character is formed from a parity bit, a data-control flag and two control bits. The data-control flag is set to one to indicate that the current character is a control character. Parity coverage is similar to that for a data character. One of the four possible control characters is the escape code (ESC). This can be used to form control codes. Two control codes are specified and valid which are the NULL code and the Time-Code.

NULL is formed from ESC followed by the flow control token (FCT). NULL is transmitted whenever a link is not sending data or control tokens, to keep the link active and to support link disconnect detection.

The Time-Code is used to support the distribution of system time across a network. A Time-Code is formed by ESC followed by a single data-character.

The data and control characters are illustrated in Figure 6.

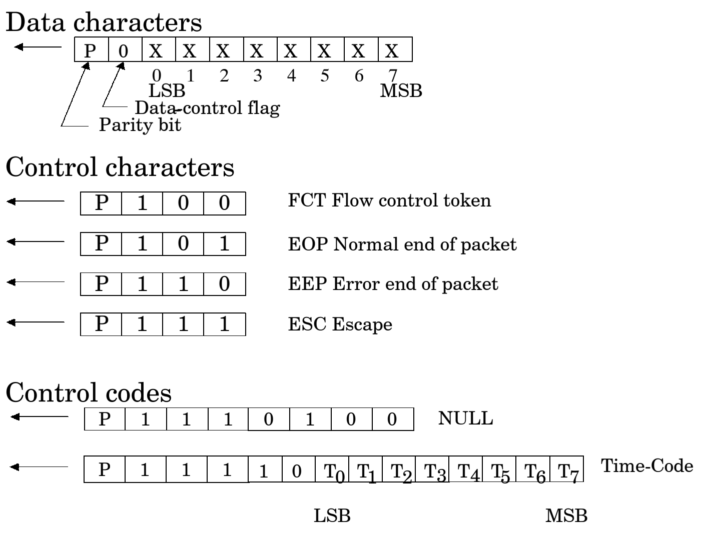


Figure 6: Data and control characters

4.5 Exchange level

The exchange level protocol is a significantly more capable version than that defined in IEEE Standard 1355-1995 [1] and provides the following services:

- **Initialization**: Following reset the link output is held in the reset state until it is instructed to start and attempts to make a connection with the link interface at the other end of the link. A connection is made following a handshake that ensures both ends of the link are able to send and receive characters successfully. Each end of the link sends NULLs, waits to receive a NULL, then sends FCTs and waits to receive an FCT. Since a link interface cannot send FCTs until it has received a NULL, receipt of one or more NULLs followed by receipt of an FCT means that the other end of the link has received NULLs successfully and that full connection is achieved.

- **Flow control**: A transmitter can only transmit N-Chars (normal characters, which are data characters, EOP or EEP) if there is space for them in the host system receive buffer at the other end of the link. The host system indicates that there is space for eight more N-Chars by requesting the link transmitter to send a flow control token (FCT). The FCT is received at the other end of the link (end B) enabling the transmitter at end B to send up to eight more N-Chars. If there is more room in the host receive buffer then multiple FCTs can be sent, one for every eight spaces in the receive buffer. Correspondingly, if multiple FCTs are received then it means that there is a corresponding amount of space available in the receiver buffer, e.g. four FCTs means that there is room for 32 N-Chars.
- **Detection of disconnect errors**: Link disconnection is detected when following reception of a data bit no new data bit is received within a link disconnect timeout window (850 ns). Once a disconnection error is detected, the link attempts to recover from the error (see Figure 7).
- **Detection of parity errors**: Parity errors occurring within a data or control character are detected when the next character is sent, since the parity bit for a data or control character is contained in the next character. Once a parity error is detected, the link attempts to recover from the error (see Figure 7).
- **Link error recovery**: Following an error or reset the link attempts to re-synchronize and restart using an “exchange of silence” protocol (see Figure 7). The end of the link that is either reset or that finds an error, ceases transmission. This is detected at the other end of the link as a link disconnect and that end stops transmitting too. The first link resets its input and output for 6,4 µs to ensure that the other end detects the disconnect. The other end also waits for 6,4 µs after ceasing transmission. Each link then waits a further 12,8 µs before starting to transmit. These periods of time are sufficient to ensure that the receivers at both ends of the link are ready to receive characters before either end starts transmission. The two ends of the link go through the NULL or FCT handshake to re-establish a connection and ensure proper character synchronization.

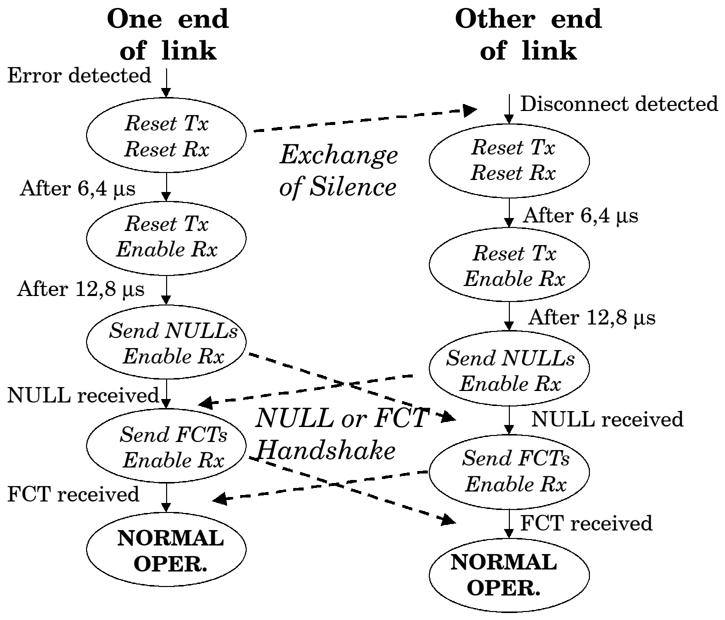


Figure 7: Link restart

4.6 Packet level

The packet level protocol follows the packet level protocol defined in IEEE Standard 1355-1995 [1]. It defines how data is encapsulated in packets for transfer from source to destination. The format of a packet is illustrated in Figure 8.

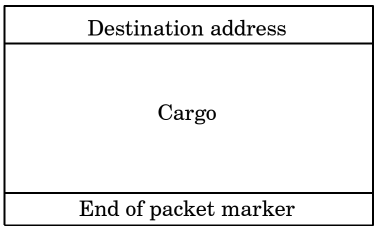


Figure 8: Packet format

The "destination address" is a list of zero or more data characters that represent the destination identity. This list of data characters represents either the identity code of the destination node or the path that the packet takes to get to the destination node.

The "cargo" is the data to transfer from source to destination.

The "end of packet marker" is used to indicate the end of a packet. Two end of packet markers are defined:

a. EOP normal end_of_packet marker - indicates end of packet;

b. EEP error end_of_packet marker - indicates that the packet is terminated prematurely due to a link error.

Since there is no start_of_packet marker, the first data character following an end_of_packet marker (either EOP or EEP) is regarded as the start of the next packet.

The packet level protocol provides support for packet routing via wormhole routing switches [7].

4.7 Network level

The network level defines what a SpaceWire network is, describes the components that make up , explains how packets are transferred across it, and details the manner in which it recovers from errors.

A SpaceWire network is made up of a number of SpaceWire nodes interconnected by SpaceWire routing switches. SpaceWire nodes are the sources and destinations of packets and provide the interface to the application systems. SpaceWire nodes can be directly connected together using SpaceWire links or they can be interconnected via SpaceWire routing switches using SpaceWire links to make the connection between node and routing switch. A SpaceWire routing switch has several link interfaces connected together by a switch matrix, which allows any link input to pass the packets that it receives on to any link output for retransmission.

4.8 Application programming interface

The application programming interface (API) is not defined in this Standard. However, a typical application interface comprises the following services:

- **Open link**: Starts a link interface and attempts to establish a connection with the link interface at the other end of the link.
- **Close link**: Stops a link and breaks the connection.
- **Write packet**: Sends a packet out of the link interface.

- **Read packet**: Reads a packet from the link interface.
- **Status and configuration**: Reads the current status of the link interface and sets the link configuration.

5 Physical level (normative)

5.1 General

The physical level provides the actual interface between nodes including both the mechanical and electrical interface. This clause covers:

- cable construction,
- connectors,
- cable assemblies, and
- PCB and backplane tracking.

5.2 Cables

5.2.1 Generic

a. The SpaceWire cable shall be constructed according to ESCC 3902/003 and the specific details given in the following subclauses.

b. The SpaceWire cable shall comprise four twisted pair wires with a separate shield around each twisted pair and an overall shield as illustrated in Figure 9.

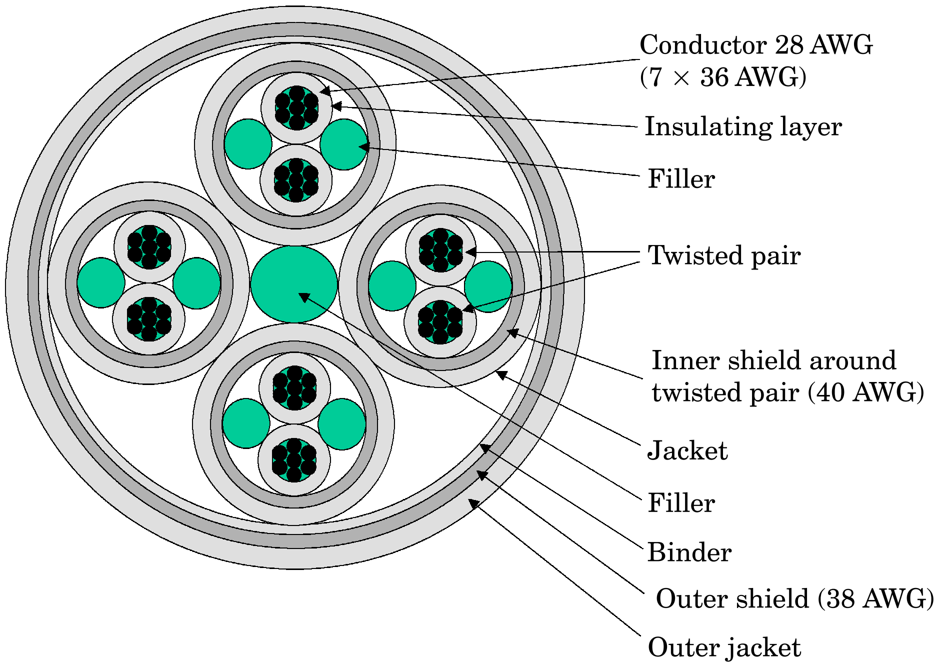


Figure 9: SpaceWire cable construction

5.2.2 Inner conductors

5.2.2.1 Conductor

a. Each signal wire shall be 28 AWG, constructed from seven strands of 36 AWG silver-coated, high-strength copper alloy.

b. The thickness of the silver coating shall be 2,0 μm minimum.

5.2.2.2 Tensile characteristics

a. The minimum elongation of each strand shall be 6,0 %.

b. The tensile strength of each strand shall be at least 350 N/mm^2.

5.2.2.3 Insulator

Each signal shall be insulated using expanded, microporous PTFE with only those additives for processing and pigmentation.

5.2.2.4 Insulator colour

The insulator around the signal wires shall be white.

5.2.2.5 Electrical characteristics

The maximum DC resistance of the inner conductor shall be 256 Ω/km.

5.2.3 Twisted pair

5.2.3.1 Lay-length

The lay-length of the two insulated conductors comprising a differential signal pair shall not be less than 12 times and not more than 16 times the outside diameter of the unshielded twisted pair.

5.2.3.2 Fillers

Fillers shall be used with the differential signal pairs so as to ensure a smooth and uniform diameter under the shielding in order to contribute to a uniform impedance over the cable.

5.2.3.3 Filler material

The filler material as used for the differential signal pairs shall be expanded microporous PTFE with only those additives for processing.

5.2.3.4 Construction of filler

The filler material shall be extruded or wrapped from tapes to a diameter of 1,0 mm.

5.2.3.5 Shield

a. Each differential signal pair shall be shielded by a braided shield.

b. The braided shield type shall be of push-back type and provide not less than 90 % coverage.

5.2.3.6 Shield wire size

The shield wire size shall be 40 AWG.

5.2.3.7 Shield material

a. All strands used in the manufacture of the braided shield shall be silver-coated, soft or annealed oxygen-free high conductivity copper.

b. The thickness of silver shall be 2,5 µm minimum.

c. Any strand shall show an elongation of 10 % minimum.

5.2.3.8 Protective sheath

The protective sheath for the shielded differential signal pairs shall be a layer of extruded fluorpolymer PFA with only those additives for processing and pigmentation.

5.2.3.9 Protective sheath wall thickness

The wall thickness of the protective sheath for the shielded differential signal pair shall be 0,15 mm nominal.

5.2.3.10 Protective sheath colour

The jacket colour of the differential signal pairs shall be white.

5.2.3.11 Characteristic impedance

The characteristic impedance of each differential signal pair shall be (100 ± 6) Ω differential impedance.

5.2.3.12 Skew

The skew between each signal in each differential signal pair shall be less than 0,1 ns/m.

5.2.4 Complete cable

5.2.4.1 Construction

Four sets of differential signal pairs shall be twisted together not less than 12 times and not more than 16 times the outside diameter of two shielded and jacketed differential signal pairs.

5.2.4.2 Filler

A filler shall be used in the centre of the four differential signal pairs so as to ensure a smooth and uniform diameter under the shielding in order to contribute to a uniform impedance over the cable.

5.2.4.3 Filler material

The filler material as used for the complete cable shall be microporous PTFE with only those additives for processing.

5.2.4.4 Construction of filler

The filler material shall be extruded or wrapped from tapes to a diameter of 1,0 mm.

5.2.4.5 Binder

A binder shall be applied over the four differential signal pairs and central filler to keep the signal pairs and filler together in a fixed position.

5.2.4.6 Binder material

The material shall be virgin, wrapped, expanded microporous PTFE with only those additives for processing.

5.2.4.7 Binder construction

The material shall be wrapped with an overlap of 50 % maximum.

5.2.4.8 Outer shield

a. The set of four jacketed and screened differential signal pairs shall be shielded by an outer braided shield.

b. The braided shield type shall be of push-back type and provide not less than 90 % coverage.

5.2.4.9 Outer shield wire size

The shield wire size shall be 38 AWG.

5.2.4.10 Outer shield material

a. All strands used in the manufacture of the braided shield shall be silver-coated, soft or annealed oxygen-free high conductivity copper.

b. The thickness of silver shall be 2,5 μm minimum.

c. Any strand shall show an elongation of 10 % minimum.

5.2.4.11 Shield isolation

The twisted pair shields shall not make contact with one another nor with the outer shield.

5.2.4.12 Outer jacket

The outermost jacket over the four twisted screened and jacketed differential signal pairs shall be a layer of extruded Fluoropolymer PFA with only those additives for processing and pigmentation.

5.2.4.13 Outer jacket wall thickness

The wall thickness of the jacket for the shielded differential signal pair shall be 0,25 mm nominal.

5.2.4.14 Jacket colour

a. The colour of the jacket shall be white.

b. There shall be no identifying marking on the cable jacket.

NOTE Applying pressure to the cable during the marking process can adversely affect the electrical properties of the cable.

5.2.4.15 Signal skew

a. The skew between the parts of the differential signal in one differential signal pair shall be 0,1 ns/m maximum.

b. The skew between one differential signal pair and each other differential signal pair within the cable shall be 0,15 ns/m maximum.

5.2.5 Cable physical parameters

5.2.5.1 Cable diameter

The outside diameter of the complete cable shall be 7 mm maximum.

5.2.5.2 Cable minimum bend radius

The minimum bend radius of complete cable shall be 45 mm.

5.2.5.3 Adhesion of inner conductor

The minimum stripping force shall be 1,0 N.

5.2.5.4 Cable weight

The maximum weight of the SpaceWire cable shall be 80 g/m.

5.2.5.5 Cable maximum ratings

a. The maximum ratings defined in Table 1 shall be met.

b. The total temperature of the wire (i.e. ambient plus rise) shall not exceed the maximum operating temperature of the wire.

Table 1: SpaceWire cable maximum ratings

No.	Characteristics	Symbol	Maximum ratings	Unit	Remarks
1	Operating voltage (continuous)	V_{op}	200	V_{rms}	
2	Current	I	1,5	A	
3	Operating rate	F_M	400	Mb/s	
4	Operating temperature range	T_{op}	-200 to +180	°C	T_{amb} [a]
5	Storage temperature range	T_{stg}	-200 to +180	°C	

[a] The specified current generates a temperature rise of approximately 50 °C above ambient temperature in a vacuum environment. See 5.2.5.5.b. for precautions to take on the total temperature of the wire.

5.3 Connectors

5.3.1 General

The SpaceWire connector shall be a nine contact micro-miniature D-type with solder contacts, as defined in ESCC 3401/071, or crimp contacts.

5.3.2 Receptacles

a. Receptacles shall be used on board and unit assemblies.

b. Receptacles shall be equipped with female contacts.

c. Receptacles with flying leads should be used for connection to a PCB rather than PCB mounting connectors to improve mechanical shock and vibration resistance of the unit.

d. Soldering shall conform to ECSS-Q-70-08.

e. Crimping shall conform to ECSS-Q-70-26.

5.3.3 Plugs

a. Plugs shall be used on cable assemblies.

b. Plugs shall be equipped with male contacts as follows:

 1. The SpaceWire conductors shall be directly soldered or crimped to the contacts as described in subclause 5.4.
 2. The overall shield of the SpaceWire shall be connected to the shell via an EMI backshell.

c. Soldering shall conform to ECSS-Q-70-08.

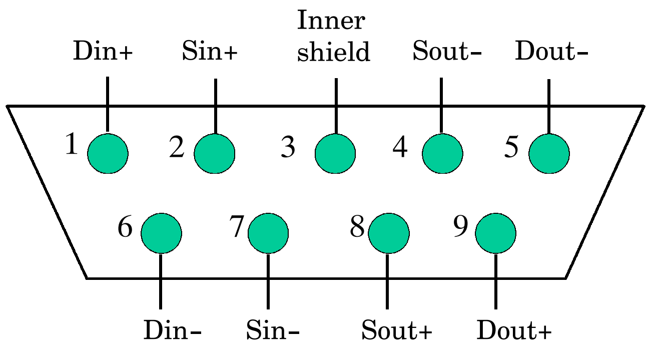
d. Crimping shall conform to ECSS-Q-70-26.

5.3.4 Connector contact identification

The connector contact identification given in Table 2 and Figure 10 shall be used.

Table 2: Connector contact identification

Contact number	Signal name
1	Din+
2	Sin+
3	Inner shield
4	Sout-
5	Dout-
6	Din-
7	Sin-
8	Sout+
9	Dout+



Viewed from rear of receptacle or front of plug.

Figure 10: SpaceWire connector contact identification

5.3.5 Inner shield connection

a. The inner shield connection shall be connected to the inner shield of the SpaceWire cable.

b. This inner shield of the SpaceWire cable should be connected to signal ground according to the EMC requirements of the mission.

c. The connection referred in b. should be performed via a parallel resistor and capacitor.

NOTE See subclause 5.4 for the cable connection to pin 3 of the connector.

5.3.6 Flying lead connectors

a. Flying lead connectors should be used for connection to a PCB.

b. Flying lead connectors used for connection to a PCB should have all the leads cropped to the same short length (less than 25 mm) and the wires comprising the differential signal pairs should be twisted together.

NOTE This helps to minimize the discontinuity in impedance caused by the connector.

5.3.7 PCB mounting connectors

a. PCB mounting right-angled connectors should not be used.

b. If a PCB mounting right-angled connector is used, signal path length compensation shall be performed by adjusting the length of tracks on the PCB, as follows. The topmost row of pins on the right-angled connect have longer leads than the bottom row. Signals connected to the top row shall be given correspondingly shorter PCB track lengths than tracks going to the bottom row. Track length compensation shall be performed at the connector end of the PCB tracks to maintain the differential signal across the PCB.

5.4 Cable assembly

5.4.1 General

Cable assemblies consist of two identical plug connectors joined by a length of cable.

5.4.2 Cable length

a. The maximum length of the cable assembly should be 10 m to ensure that the end to end skew and jitter introduced by the cable assembly does not exceed the maximum budget for the cable.

b. Longer length cables may be used at slow data signalling rates provided that the signal attenuation (see clause 6) and system jitter and skew limits are not violated at the operating data signalling rate (see subclause 6.6.4).

5.4.3 Cable connections

a. The connector contacts shall be terminated as shown in Figure 11 and Table 3.

NOTE The cable signal wires cross over to achieve a transmit to receive interconnection, e.g. Dout+ is connected to Din+ .

b. The individual shields of the differential signal pairs carrying the output signals Dout+, Dout- and Sout+ and Sout- shall be connected together and to pin 3 of the connector.

NOTE The shields are terminated at the end of the cable from which the signals are being driven, following good EMC practice. In this way two of the differential pairs are connected at one end of the cable and the remaining two at the other end. A symmetrical arrangement results, avoiding the problem of having to know which end of the cable is which during installation.

c. A metal shell shall be used for each connector to provide necessary shielding of the connector.

d. The outer shield of the cable shall be bonded to the connector shell via a low impedance connection (less than 1 Ω).

e. The metal shell shall be bonded to the main body of the connector via a low impedance connection (less than 1 Ω).

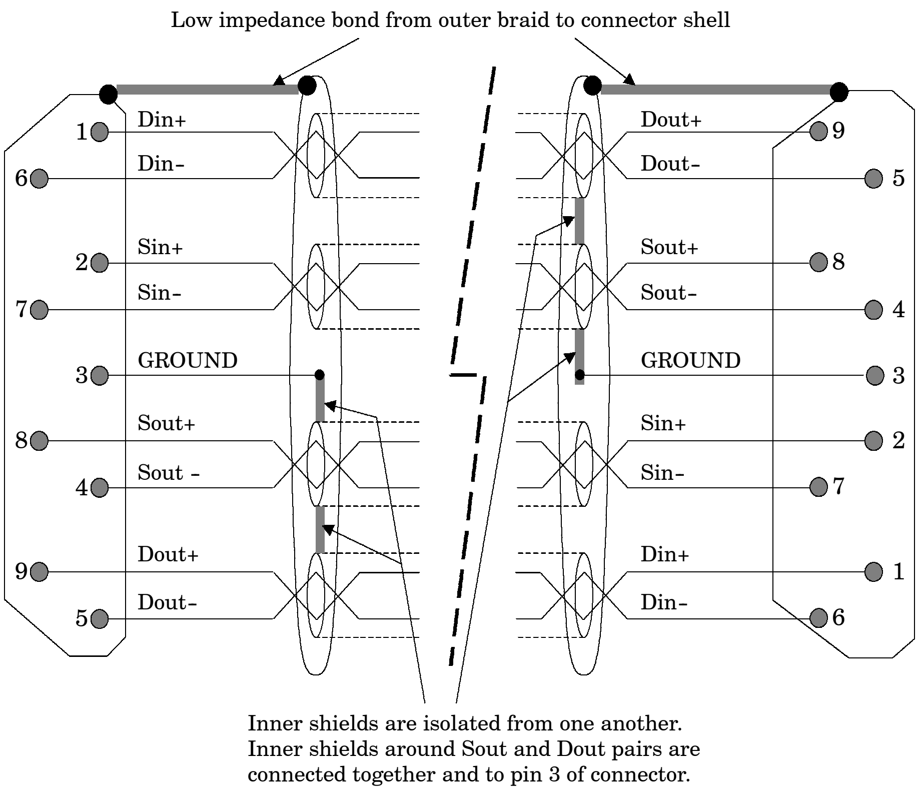

Low impedance bond from outer braid to connector shell
1
6
2
7
3
8
4
9
5
Din+
Din-
Sin+
Sin-
GROUND
Sout+
Sout -
Dout+
Dout-
Dout+
Dout-
Sout+
Sout-
GROUND
Sin+
Sin-
Din+
Din-
9
5
8
4
3
2
7
1
6
Inner shields are isolated from one another.
Inner shields around Sout and Dout pairs are connected together and to pin 3 of connector.

Figure 11: SpaceWire cable assembly

Table 3: Cable assembly signal wire connections

Signal at A end	Pin at A end		Pin at B end	Signal at B end
A-Din+	1	- Connection -	9	B-Dout+
A-Din-	6	- Connection -	5	B-Dout-
A-Sin+	2	- Connection -	8	B-Sout+
A-Sin-	7	- Connection -	4	B-Sout-
A- (Drains of pairs 5,9 and 4,8)	3	- No Connection -	3	B-(Drains of pairs 5,9 and 4,8)
A-Sout+	8	- Connection -	2	B-Sin+
A-Sout-	4	- Connection -	7	B-Sin-
A-Dout+	9	- Connection -	1	B-Din+
A-Dout-	5	- Connection -	6	B-Din-
A-Shield	Shell	- Connection -	Shell	B-Shield

5.5 PCB and backplane tracking

5.5.1 General

As well as routing SpaceWire signals through a cable, the signals can also be transmitted across a PCB or along a backplane.

NOTE Only point to point connections are supported on a PCB or backplane, not multi-drop bus structures. Bus type structures are built from point to point connections between nodes on the backplane.

5.5.2 Differential signal pairs

5.5.2.1 Differential impedance

Differential pair signals shall run on a pair of close, parallel PCB tracks with a differential impedance of (100 ± 6) Ω.

NOTE This differential impedance can be achieved by adjusting the track thickness, width, separation and height above the ground plane.

5.5.2.2 Difference in track length for a differential pair

To avoid skew between the two parts of the differential signal, the difference in track length between the two signals from a differential pair shall be less than 5 % of the track length and no more than 5 mm.

5.5.2.3 Difference in track length for Data and Strobe

The skew introduced between the Data and Strobe (D and S) signals shall be minimized as specified here. For PCB tracks, skew is controlled by making the tracks all close to the same length. The difference in track length between the Data and Strobe signals shall be less than 5 % of the track length and no more than 5 mm.

(This page is intentionally left blank)

6

Signal level (normative)

6.1 Low voltage differential signaling (LVDS)

SpaceWire shall use low voltage differential signalling (LVDS) with electrical characteristics as defined in ANSI/TIA/EIA-644.

6.2 Failsafe operation of LVDS

a. When any of the following fault conditions occur, the receiver outputs shall not oscillate and shall be locked to high logic level provided that a noise threshold of 10 mV is not exceeded at the receiver input.

1. Driver not powered.
2. Driver disabled.
3. Driver not connected to receiver.
4. Receiver inputs open circuit (i.e. cable or wire in cable disconnected).
5. Receiver inputs shorted together.

b. When the driver is not powered its output should be high impedance i.e. > 100 kΩ.

c. When the receiver is not powered its input should be high impedance i.e. > 100 kΩ.

6.3 Signal coding

6.3.1 Data-strobe (DS)

a. SpaceWire shall use Data-Strobe (DS) encoding.

NOTE DS encoding is defined in subclause 5.3.5 of IEEE Standard 1355-1995 [1] and also defined in IEEE Standard 1394-1995 [6]. See annex A for details of the differences between this Standard and IEEE Standard 1355-1995 and the reasons for those differences.

b. The data bit stream to transmit shall be encoded using two signals Data and Strobe as follows. The Data signal shall follow the data bit stream, i.e. be high when the data bit is 1 and low when the data bit is 0. The Strobe signal shall change state whenever the Data does not change from one bit to the next.

NOTE The DS encoding is illustrated in Figure 12.

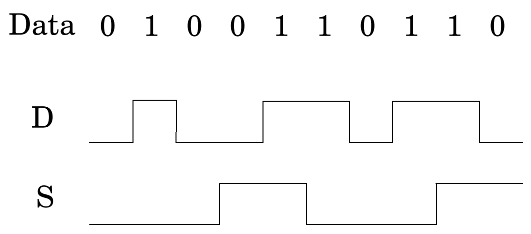


Figure 12: Data-Strobe (DS) encoding

6.3.2 Simultaneous transition on data and strobe signals

a. As data corruption following simultaneous transitions on the Data and Strobe lines is expected, the SpaceWire receiver shall be tolerant of simultaneous transitions on the Data and Strobe lines, i.e. the receiver shall not hang up.

b. When the SpaceWire transmitter is reset it shall be a controlled reset avoiding simultaneous transitions of Data and Strobe signals.

EXAMPLE After stopping transmission the Strobe signal can be reset first, followed by the Data signal.

NOTE Simultaneous transitions on the Data and Strobe lines are not part of the normal operation of SpaceWire. They can occur, however, either when a SpaceWire cable is plugged in while the transmitter is trying to make a connection, or when the LVDS driver or receiver circuits are enabled while the transmitter is trying to make a connection.

6.4 Differential DS

SpaceWire shall use low voltage differential signalling (LVDS) for the Data and Strobe signals.

6.5 SpaceWire link

A SpaceWire link shall comprise two pairs of differential signals, one pair transmitting the D and S signals in one direction and the other pair transmitting D and S in the opposite direction.

NOTE This is a total of eight wires for each bidirectional link.

6.6 Data signalling rate

6.6.1 Minimum data signalling rate

The minimum data signalling rate at which a SpaceWire link shall operate is 2 Mb/s.

NOTE 1 The minimum data signalling rate is the lowest data signalling rate at which a SpaceWire link can operate.

NOTE 2 The minimum data signalling rate is set by the disconnect timeout (subclause 8.9.2.1 and 8.11.2) to greater than 1,18 Mb/s, i.e. 1/850 ns.

6.6.2 Maximum data signalling rate

The maximum data signalling rate is the highest data signalling rate at which a SpaceWire link can operate and is defined by consideration of signal skew and jitter (see subclause 6.6.4).

6.6.3 Operational data signalling rate

A SpaceWire link can operate at any data signalling rate between the minimum data signalling rate and the maximum possible data signalling rate.

The link in one direction can operate at a different data signalling rate to the same link in the opposite direction. Links within a system can operate at different data signalling rates.

6.6.4 Effects of skew and jitter

6.6.4.1 Skew and jitter

The maximum data signalling rate that can be achieved is different from one system to another, depending on several factors such as cable length, driver-receiver technology, and encoder-decoder design, and is limited by skew and jitter. Figure 13 illustrates the effect of skew and jitter on the Data and Strobe signals, where the parameters are as follows:

- t_{skew} is the skew between the Data and Strobe signals.
- t_{jitter} is the jitter on the Data or Strobe signal. t_{jitter} data = t_{jitter} strobe since they follow identical signal paths (as close as possible).
- t_{dclk} is the delay in the receiver from the edge of the Data or Strobe signal, through the XOR operation which produces the clock signal, to the clocking in of the data in the input flip-flop. This may be regarded as the set-up time for the data input flip-flop from the edge of the Data or Strobe signal.
- t_{hold} is the hold time for the Data signal after the clocking of the data into the input flip-flop.
- t_{ui} is the unit interval or bit period. $t_{\text{ui}} = 1/F_{\text{op}}$, where F_{op} is the link operating data signalling rate.
- The t_{dclk} and t_{hold} parameters may be combined into a minimum specification for the separation of consecutive edges on the Data and Strobe signals at the input to the decoder, $t_{\text{ds}} = t_{\text{dclk}} + t_{\text{hold}}$.
- t_{margin} is the available margin. $t_{\text{margin}} = t_{\text{ui}} - (t_{\text{skew}} + 2{*}t_{\text{jitter}} + t_{\text{dclk}} + t_{\text{hold}})$.

NOTE 1 Figure 14 illustrates the contributors to skew and jitter in a typical system.

NOTE 2 Table 4, Table 5 and Table 6 provide the example jitter and skew budgets at three different operating frequencies (100 Mb/s, 200 Mb/s and 400 Mb/s).

NOTE 3 The example jitter and skew figures for 400 Mb/s operation (Table 6) assume that the LVDS driver or receiver are integrated in the same package as the encoder-decoder.

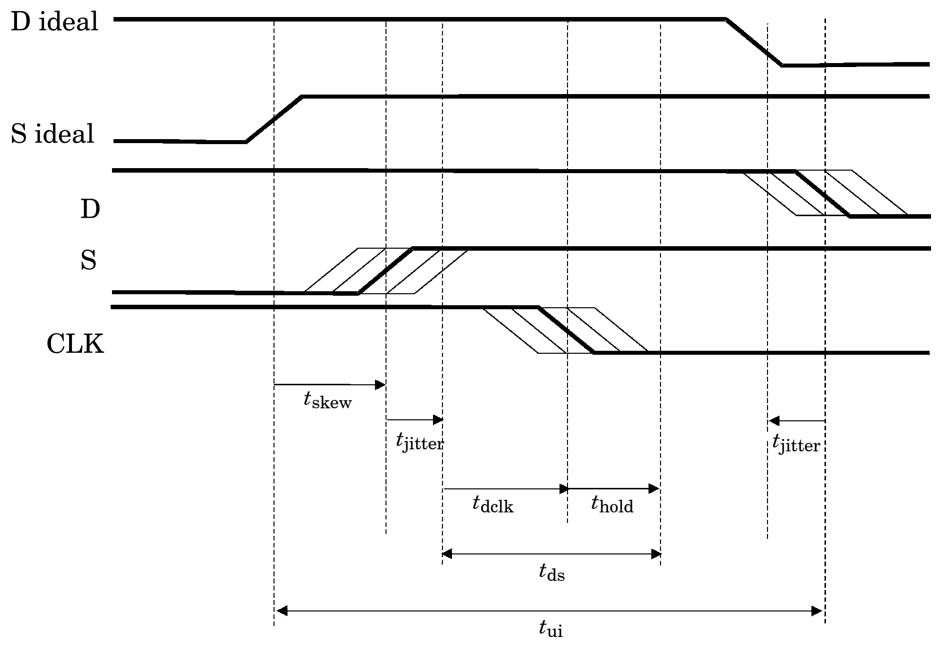


Figure 13: Skew and jitter

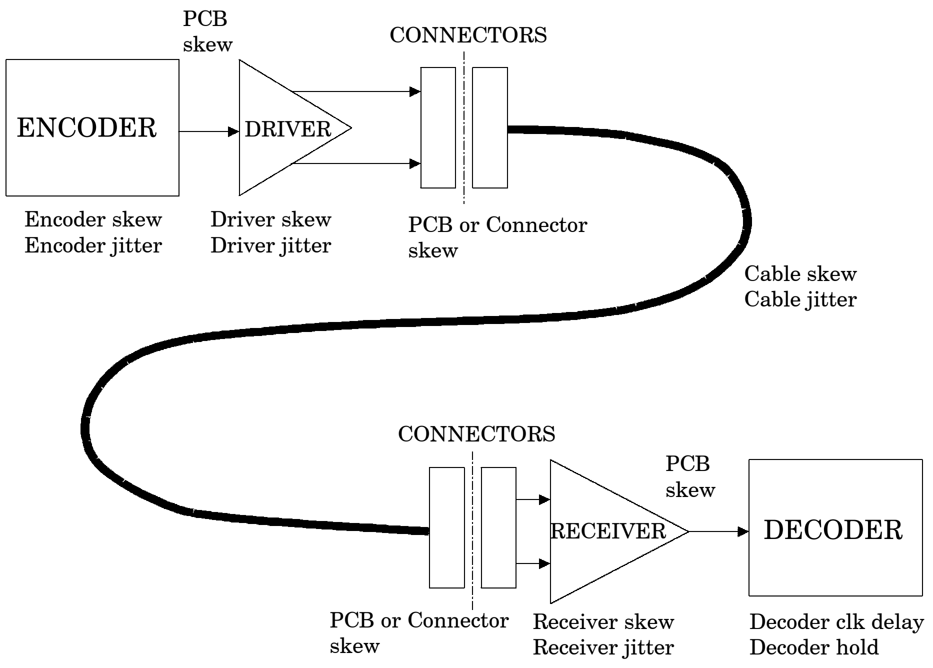


Figure 14: Contributors to skew and jitter

6.6.4.2 Timing margin

The maximum data signalling rate for a SpaceWire link shall be set so that the timing margin (t_{margin}) is greater than zero.

Table 4: Example jitter and skew budget at 100 Mb/s

	Data jitter t_{jitter} (ns)	**Strobe jitter** t_{jitter} (ns)	**Skew** t_{skew} (ns)	**Min edge separation** t_{ds} (ns)	**Total** (ns)
Encoder skew			0,50		
Encoder jitter	0,50	0,50			
PCB skew			0,05		
Driver skew			1,00		
Driver jitter	0,50	0,50			
PCB/connector skew			0,10		
Total transmitter	**1,00**	**1,00**	**1,65**		**3,65**
Cable jitter	0,50	0,50			
Cable skew			1,00		
Total cable	**0,50**	**0,50**	**1,00**		**2,00**
PCB/connector skew			0,10		
Receiver skew			1,50		
Receiver jitter	0,50	0,50			
PCB skew			0,05		
Decoder clock delay and hold				1,00	
Total receiver	**0,50**	**0,50**	**1,65**	**1,00**	**3,65**
Total system	**2,00**	**2,00**	**4,30**		**8,30**
Margin					1,70

Table 5: Example jitter and skew budget at 200 Mb/s

	Data jitter t_{jitter} (ns)	**Strobe jitter** t_{jitter} (ns)	**Skew** t_{skew} (ns)	**Min edge separation** t_{ds} (ns)	**Total** (ns)
Encoder skew			0,50		
Encoder jitter	0,10	0,10			
PCB skew			0,05		
Driver skew			0,07		
Driver jitter	0,20	0,20			
PCB/connector skew			0,10		
Total transmitter	**0,30**	**0,30**	**0,72**		**1,32**
Cable jitter	0,50	0,50			
Cable skew			1,00		
Total cable	**0,50**	**0,50**	**1,00**		**2,00**
PCB/connector skew			0,10		
Receiver skew			0,12		
Receiver jitter	0,20	0,20			
PCB skew			0,05		
Decoder clock delay and hold				1,00	
Total receiver	**0,20**	**0,20**	**0,27**	**1,00**	**1,67**
Total system	**1,00**	**1,00**	**1,99**	**1,00**	**4,99**
Margin					0,01

Table 6: Example jitter and skew budgets at 400 Mb/s

	Data jitter t_{jitter} (ns)	**Strobe jitter** t_{jitter} (ns)	**Skew** t_{skew} (ns)	**Min edge separation** t_{ds} (ns)	**Total** (ns)
Encoder skew			0,20		
Encoder jitter	0,10	0,10			
PCB/connector skew			0,05		
Total transmitter	**0,10**	**0,10**	**0,25**		**0,45**
Cable jitter	0,35	0,35			
Cable skew (5m max. length)			0,50		
Total cable	**0,35**	**0,35**	**0,50**		**1,20**
PCB/connector skew			0,05		
Receiver jitter	0,10	0,10			
Decoder clock delay and hold				0,50	
Total receiver	**0,10**	**0,10**	**0,05**	**0,50**	**0,75**
Total system	**0,55**	**0,55**	**0,80**	**0,50**	**2,40**
Margin					0,10

6.6.5 Initial operating data signalling rate

a. After a reset or disconnect (see subclause 8.9.2.1) the SpaceWire link transmitter shall initially commence operating at a data signalling rate of (10±1) Mb/s.

b. The SpaceWire link transmitter shall operate at (10±1) Mb/s until commanded to operate at a different data signalling rate.

AIM: To provide all systems with a common, slow, initial data signalling rate so that system operation can be validated before switching to higher and possibly widely different data signalling rates.

NOTE This initial slow data signalling rate is applicable to all SpaceWire systems, but they need not be capable of higher data signalling rates.

6.6.6 Altering data signalling rate

The transmitter operating rate shall not be changed before the link connection has been made fully (i.e. the exchange-level state machine is in the *Run* state; see clause 8).

NOTE Once in the Run state (see subclause 8.5) the transmitter operating rate can be set at any rate between the minimum (see subclause 6.6.1) and the maximum data signalling rate (see subclause 6.6.2).

7

Character level (normative)

7.1 General

The character level protocol defined in this clause takes into consideration the DS-SE and DS-DE character level encoding given in IEEE Standard 1355-1995 [1], but it additionally includes Time-Codes for sending system time information across a SpaceWire link. The host interface to the SpaceWire encoder-decoder is specified.

7.2 Data characters

As illustrated in Figure 15, a data character shall contain a parity bit, a data-control flag and eight bits of data as follows. The data-control flag shall be set to zero to indicate that the current character is a data character. The eight-bit data value shall be transmitted least significant bit first.

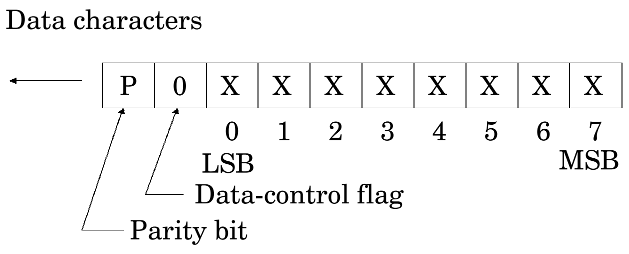


Figure 15: SpaceWire data characters

7.3 Control characters and control codes

a. A control character shall be formed from a parity bit, a data-control flag and a two-bit control code with the data-control flag set to one to indicate that the current character is a control character.

NOTE The different control characters are illustrated in Figure 16.

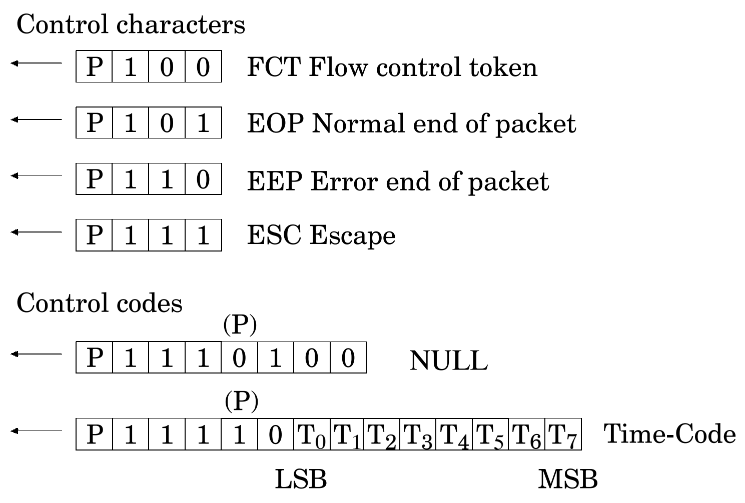

Control characters
P 1 0 0 FCT Flow control token
P 1 0 1 EOP Normal end of packet
P 1 1 0 EEP Error end of packet
P 1 1 1 ESC Escape
Control codes
(P)
P 1 1 1 0 1 0 0 NULL
(P)
P 1 1 1 1 0 T0 T1 T2 T3 T4 T5 T6 T7 Time-Code
LSB
MSB

Figure 16: SpaceWire control characters and control codes

b. The NULL control code shall be formed from ESC followed by the flow control token (FCT).

NOTE 1 The parity bit (P) in the middle of the control code is zero, in accordance with subclause 7.4 b.).

NOTE 2 NULL is transmitted whenever a link is not sending data or control tokens, to keep the link active and to support link disconnect detection (see clause 8).

c. The time control code (Time-Code) shall be formed from ESC followed by a single data character.

NOTE 1 The parity bit (P) in the middle of the Time-Code is one, in accordance with subclause 7.4 b.).

NOTE 2 The Time-Code is used to distribute system time information (see subclause 8.12) and control flags isochronous with the time-code distribution.

d. Six bits of time information shall be held in the least significant six bits of the Time-Code (T0-T5) and the two most significant bits (T6, T7) shall contain control flags that are distributed isochronously with the Time-Code.

e. An escape character (ESC) followed by ESC, EOP or EEP is an invalid sequence and shall be noted as an escape error (see subclause 8.9.2.3).

7.4 Parity for error detection

a. A parity bit shall be assigned to each data or control character to support the detection of transmission errors.

b. The parity bit shall cover the previous eight bits of a data character or two bits of a control character, the current parity bit, and the current data-control flag. The parity bit shall be set to produce odd parity so that the total number of 1's in the field covered is an odd number.

NOTE The coverage of the parity bit is illustrated in Figure 17.

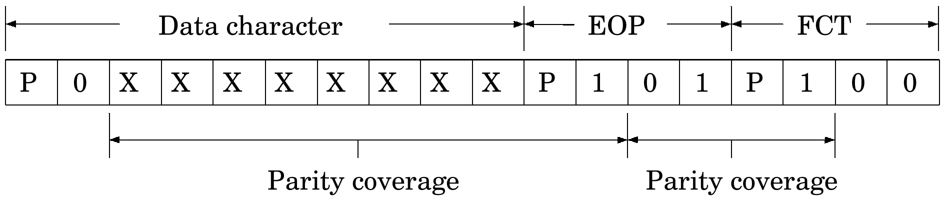


Figure 17: Parity coverage

7.5 Transmit bit pattern after reset or link error

a. After reset or link error (while in the ErrorReset state, see clause 8) the Data and Strobe signals shall be set to zero.

b. When the transmitter is enabled after reset the first bit that is sent shall be a parity bit, this bit shall be set to zero so that the first transition shall be on the Strobe line.

NOTE This results in the patterns shown in Figure 18 appearing when a link is started.

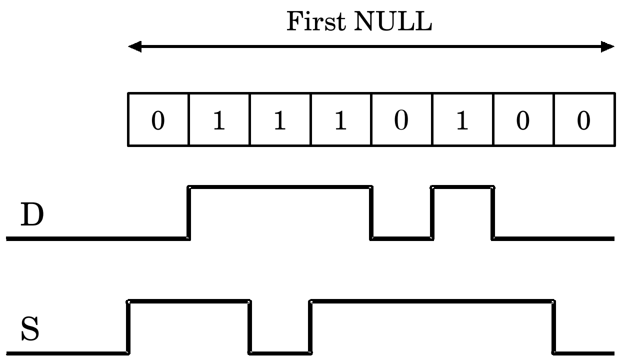


Figure 18: Data and strobe signals on link start

7.6 Host interface to transmitter and receiver

a. When the transmit and receive data interfaces to the host comprise eight data bits and one control flag, the coding given in Table 7 shall be used.

NOTE Thus, for example, any code with the control flag set to one and the least significant bit of the data set to zero represents an EOP. This prevents the transmitter being asked to send an invalid control code.

Table 7: Transmitter and receiver host data interface coding

Control flag	Data bits (MSB ... LSB)	Meaning
0	xxxxxxxx	8-bit data
1	xxxxxxx0 (use 00000000)	EOP
1	xxxxxxx1 (use 00000001)	EEP

b. EEP may be written into the transmitter interface

NOTE A SpaceWire node is the source of a packet and does not normally send packets that are in error (indicated by termination with an EEP) so normally there is no need to write EEP into the transmitter interface, unless, for example, some exception occurs in the node during the transmission of a packet. As specified in subclause 10.6.3, a SpaceWire Router forwards packets where an error has occurred (i.e. packets that are terminated by an EEP) so in the case of a router any EEP is written into the transmitter interface.

c. For the two control codes (EOP and EEP) only the least significant bit is decoded. When writing to the transmit interface the remaining data bits should be set to zero. The receiver should set the seven most significant data bits to zero when the control bit is set.

7.7 Time interface

a. The time interface to the host system shall comprise two signals, TICK_IN and TICK_OUT, a six-bit time output port, a six-bit time input port, a two-bit control flag input port and a two-bit control flag output port.

b. When TICK_IN is asserted and the link interface is in the *Run* state (see subclause 8.5) it shall cause the transmitter to send a Time-Code immediately after the current character has been transmitted.

c. TICK_OUT shall be asserted whenever the link interface is in the *Run* state and the receiver receives a valid Time-Code.

d. Only one node in a SpaceWire network should have an active TICK_IN signal.

e. All other nodes should keep the TICK_IN signal not asserted.

NOTE The node with the active TICK_IN signal provides the master time reference for the entire SpaceWire network.

f. A six-bit time output shall be provided from the link receiver to the local time counter.

NOTE The other two bits of the time output are the two control-flag outputs and are reserved for future use.

g. A six-bit time input shall be provided to the link transmitter from the local time counter.

NOTE The other two bits of the time input are the two control-flag inputs and are reserved for future use.

h. The two control flags are reserved for future use and shall both be set to zero.

8 Exchange level

8.1 General

The exchange level protocol takes into consideration the DS-SE and DS-DE exchange level protocol given in subclause 5.7 of the IEEE Standard 1355-1995 [1], but it also includes additional features in the state machine in order to eliminate problems with the *ResetLinkCommand* and several ambiguities within the IEEE Standard 1355-1995 have been resolved. See annex A for details of the differences between SpaceWire and IEEE Standard 1355-1995 and the reasons for those differences.

The exchange level is responsible for making a connection across a link and for managing the flow of data across the link.

8.2 Link-characters and normal-characters (normative)

8.2.1 Definitions

At the exchange level, data and control characters are separated into two types: link-characters (L-Char) and normal-characters (N-Char).

L-Chars are those that are used in the exchange level and which are not passed on to the packet level. The flow control token (FCT) character and escape (ESC) character are L-Chars. The NULL control code (ESC + FCT) and the Time-Code (ESC + data character) are escape sequences and may be regarded as L-Chars. They are not passed on to the packet level.

N-Chars are the characters that are passed on to the packet level: data characters and end-of-packet markers (EOP and EEP).

8.2.2 Actions

a. Only N-Chars shall be passed from the host interface to the link for transmission.

   NOTE The link interleaves L-Chars and N-Chars during transmission, but passes only N-Chars on to the host interface on the receiving side.

b. A received character shall not be acted upon until its parity has been checked.

8.3 Flow control (normative)

a. To avoid host receive buffer overflow and subsequent loss of data, data flow across a link shall be controlled using flow control tokens sent from one end of the link (end A) to the other end (end B) to signify that end A is ready to receive some more data.

NOTE The FCT (flow control token) is defined in subclause 7.3.

b. A FCT sent out by a link interface shall be used to indicate that there is space for eight more N-Chars in the host receive buffer.

c. For each FCT sent the host system shall reserve room in its receive buffer to accommodate eight N-Chars.

NOTE The receive buffer is typically a FIFO memory.

d. The transmitter shall not transmit any N-Chars until it has received one or more FCTs to indicate that the receiver is ready. The transmitter shall keep a credit count of the number of N-Chars that it has been authorized to send. Each time a link interface receives an FCT its transmitter shall increment the credit count by eight. Whenever the transmitter sends an N-Char it shall decrement the credit count by one. If the credit count reaches zero the transmitter shall cease sending N-Chars until it receives another FCT increasing the credit count again to eight. When the credit count is zero the transmitter shall continue to send L-Chars.

e. In state *ErrorReset* the credit count shall be set to zero.

f. If an FCT is received when the credit count is at or close to its maximum value (i.e. within eight of the maximum value) then the credit count shall not be incremented and a credit error shall be flagged (see subclause 8.5.3.10 and 8.9.2.4).

g. The credit counter shall hold a maximum credit count of 56 (i.e. seven FCTs).

h. On reset (or after a link disconnect) the initial number of FCTs to transmit shall be set according to the size of the receive buffer (one FCT for every eight N-Chars that can be held in the receive buffer) up to a maximum of seven.

NOTE If the receive buffer has a capacity of more than 56 (seven FCTs worth) then the number of FCTs to transmit initially shall be set to seven. A maximum of seven FCTs can be outstanding at any time.

i. A link interface shall keep a count of the number of outstanding N-Chars it expects to receive, i.e. the number it has asked for by sending FCTs. This outstanding count shall be incremented by eight each time an FCT is transmitted and shall be decremented by one each time an N-Char is received.

j. After a link reset or after a link disconnect, the initial value of the outstanding count shall be zero.

k. The outstanding counter shall contain a maximum count of 56, corresponding to seven FCTs.

l. An FCT shall not be transmitted unless there is room in the outstanding counter to record eight more outstanding N-Chars and there is room in the receive buffer to reserve space for those eight more N-Chars (see subclause 8.4.8).

m. When transmission of a Time-Code or an FCT is requested then it shall be sent immediately, as soon as the transmitter has finished sending the current character (or NULL or Time-Code). When no Time-Code or FCT is requested and N-Chars are available from the host interface and the flow control credit count is above zero, the transmitter shall send N-Chars. If no Time-Code, FCT or N-Char are sent, then the transmitter shall send NULL to indicate that the link is still active (and to prevent the disconnect detection mechanism from being triggered at the other end of the link).

n. The order of priority for transmission of characters shall be as follows:
   1. Time-Code, highest priority;
   2. FCTs;
   3. N-Chars;
   4. NULL, lowest priority.

8.4 Encoder-decoder block diagram

8.4.1 General

An example block diagram of a SpaceWire encoder-decoder is illustrated in Figure 19.

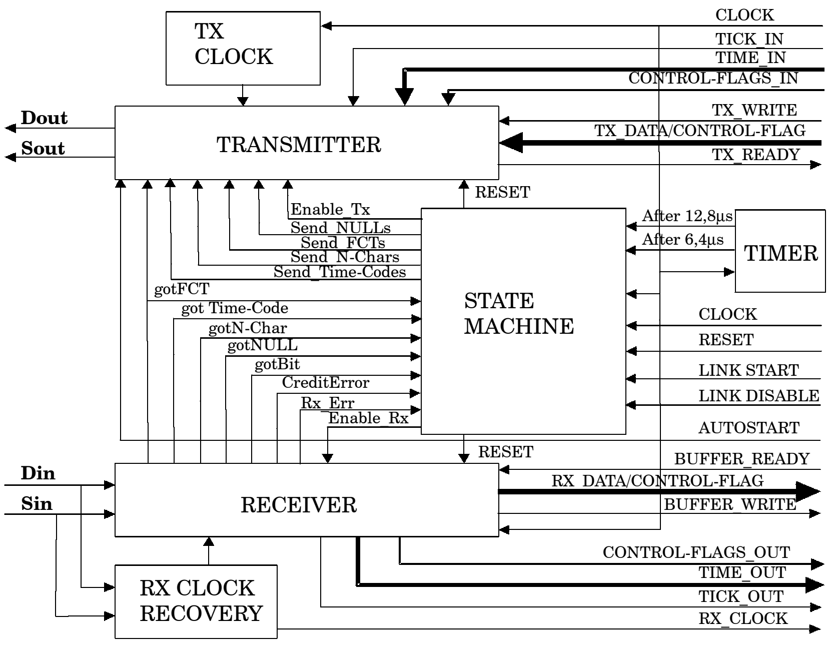


Figure 19: Example SpaceWire link interface block diagram

8.4.2 Transmitter

The transmitter is responsible for encoding data and transmitting it using the DS encoding technique. It receives N-Chars for transmission from the host system. If there is neither a Time-Code, FCT nor an N-Char (data, EOP or EEP) to transmit, the transmitter sends NULL. The transmitter sends N-Chars only if the host system at the other end of the link (end B) has room in its host receive buffer. This is indicated by the link interface at end B sending an FCT, showing that it is ready to accept another 8 N-Chars. The transmitter is responsible for keeping track of the FCTs received and the number of N-Chars sent to avoid input buffer overflow at the other end of the link. To do this the transmitter holds a credit count of the number of characters it has been given permission to send.

The transmitter can be in one of four possible states:

a. **Reset**: The transmitter does nothing.

b. **Send NULLs**: It can only send NULL on the link. It does not read N-Chars from the Transmit Host Interface. It does not accept an order to send FCT from the Host System. It does not send Time-Codes.

c. **Send FCTs or NULLs**: It sends FCT or NULL but still does not read N-Chars from the Transmit Host Interface. It does not send Time-Codes.

d. **Send Time-Codes, FCTs, N-Chars or NULLs**: Normal behaviour, sending NULLs, FCTs, Time-Codes and N-Chars.

The change of state is controlled by the State Machine (see subclause 8.4.6 and 8.5).

The transmitter is also responsible for sending FCTs whenever the local Host System has space to receive eight more N-Chars (see subclause 8.4.8). The host system requests the transmitter to send FCTs when it has room for at least eight more N-Chars that have not already been reserved for data by requesting a previous FCT. The transmitter can only send an FCT when it is in state “Send FCTs or NULLs” or “Send Time-Codes. FCTs. N-Chars or NULLs”.

The transmitter can only send Time-Codes in state d. When the TICK_IN signal is asserted the transmitter sends out a Time-Code as soon as the transmitter has finished sending the current character or control code. The value of the Time-Code is the value of the TIME_IN and CONTROL-FLAGS_IN signals at the point in time when TICK_IN is asserted.

A typical interface between the host system and the transmitter comprises TX_READY, TX_WRITE and TX_DATA (see Figure 19). When the transmitter is ready to receive another N-Char from the host system, it asserts the TX_READY signal. When the host system has an N-Char to transmit and the TX_READY signal is asserted it may put the N-Char onto the TX_DATA lines and assert the TX_WRITE signal. When the transmitter has registered the N-Char data it de-asserts the TX_READY signal.

8.4.3 Transmit clock

The transmitter can operate at any data signalling rate from the minimum to the maximum possible (see subclause 6.6). The transmit clock is responsible for producing the variable data signalling clock signals used by the transmitter. The transmit clock signals are typically derived by dividing down the local system clock or a phase locked loop multiple of the local system clock.

8.4.4 Receiver

The receiver is responsible for decoding the DS signals (Din and Sin) to produce a sequence of N-Chars (data, EOP, EEP) that are passed on to the host system. It also receives NULLs, FCTs and Time-Codes. NULLs represent an active link. They are flagged to the exchange-level state machine (see subclause 8.5) but are ignored otherwise. When an FCT is received the receiver informs the transmitter so that it can update its credit count accordingly. All other control characters received are flagged to the state machine. The receiver ignores any N-Chars, L-Chars, parity errors or escape errors until the first NULL is received. The disconnection detection mechanism within the receiver is enabled as soon as the first bit arrives (i.e. first transition detected on D or S inputs to receiver).

Time-Codes hold system time information. A valid Time-Code causes the assertion of the TICK_OUT signal from the receiver. The value of the Time-Code is placed on the TIME_OUT and CONTROL_FLAGS_OUT outputs when the TICK_OUT signal is asserted. These signals are used by the host system to update or regulate its system clock.

The receiver is also responsible for detection of disconnect, parity, escape and credit errors and it flags these errors to the state machine.

The receiver can be in one of four states:

a. **Reset**: The receiver does nothing.

b. **Enabled**: The receiver is enabled and is waiting for the first bit to arrive.

c. **GotBit**: The receiver has received the first bit (First Bit Received) and disconnect error detection is enabled. The receiver is enabled to listen for NULLs only.

d. **GotNULL**: The receiver has received a NULL and is enabled to receive NULLs, FCTs, Time-Codes and N-Chars. Disconnect, parity and escape error detection is enabled.

The change of state from *Reset* to *Enabled* is controlled by the state machine (see subclause 8.4.6 and 8.5).

The receiver is responsible for receiving FCTs from the other end of the link and for passing these FCTs on to the credit counter in the transmitter.

A typical interface between the receiver and the host system comprises BUFFER_READY, BUFFER_WRITE and RX_DATA. When the host system is ready to receive another N-Char from the receiver it asserts the BUFFER_READY signal. When the receiver has received an N-Char and the BUFFER_READY signal is asserted it puts the N-Char onto the RX_DATA lines and assert the BUFFER_WRITE signal. When the host system has registered the N-Char data it de-asserts the BUFFER_READY signal. If an N-Char is received and the BUFFER_READY signal is not asserted then a credit error has occurred (see subclause 8.5.3.10).

8.4.5 Receive clock recovery

The receive clock (RX_CLOCK) is recovered by simply XORing the received Data and Strobe signals together. The receive clock recovery circuit provides all the clock signals used by the receiver with the exception of the local clock signal use for disconnect timeout.

8.4.6 State machine

The state machine controls the overall operation of the link interface. It provides link initialization, normal operation and error recovery services. The operation of the state machine is described in the form of a state diagram in subclause 8.5.

8.4.7 Timer

The timer provides the After 6,4 μs and After 12,8 μs timeouts used in link initialization. See the state diagram of Figure 20. The timer is started when the state machine moves into particular states. When the state machine moves into the *ErrorReset* state the After 6,4 μs timer is started. When it moves into the *ErrorWait* state, *Started* state or the *Connecting* state the After 12,8 μs timer is started.

8.4.8 Receive buffer data management

The host system is responsible for data buffer management. This makes the SpaceWire interface more versatile and eases partitioning of the error recovery mechanism (see subclause 11.4) across the various levels of this Standard. Several different types of host receiver buffering may be implemented:

- FIFO buffering – where the size of the FIFO buffer depends upon the particular application.
- Memory buffering – where direct memory access (DMA) is used to transfer data to host system memory. As soon as the DMA channel has been set up, several FCTs can be requested immediately to allow the transfer of the data as fast as possible.
- No buffering – where the host system is able to accept data at the highest rate that the link interface can provide it. In this case several FCTs can be sent initially, followed by one more every time eight normal characters are received.

NOTE Due to the NULL or FCT handshake used during initialization (see subclause 8.5 and 8.7), it is expected that the host system flags that it is ready to receive eight more N-Chars before or during link initialization (see subclause 8.3.j.). If this is not the case, link initialization fails until the host system is ready to receive data – the link interface is not able to send an FCT. When the host systems at both ends of the link have indicated that they are ready to receive data, then the link connection is made.

8.4.9 Receive FIFO buffering

Most implementations of a SpaceWire interface are likely to include transmit and receive FIFOs. This subclause describes one way in which these FIFOs can be used.

After system reset the transmit and receive FIFOs are empty. When link connection is made any data written to the transmit FIFO can be transmitted if FCTs have been received to enable transmission. The receive FIFO is able to accept data while it still has space available. Data is read from the receive FIFO by the host system.

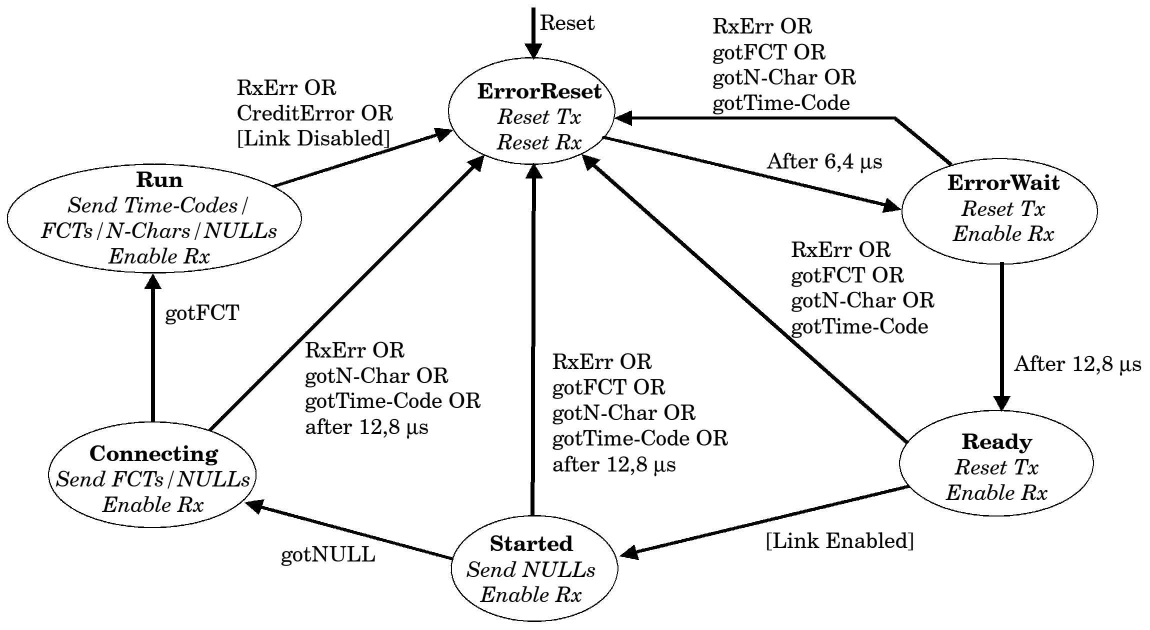
Using a FIFO simplifies the interface to the host system. FIFO half-full or half-empty flags can be used to trigger DMA or processor intervention to read from or write to the FIFO before it becomes full or empty. This helps smooth the flow of data across a link and maintain high data throughput.

8.5 State Diagram (normative)

8.5.1 General

The complete state transition diagram for the SpaceWire link interface is illustrated in Figure 20.

The state diagram notation is explained in subclause 3.3.3.



RxErr = Disconnect error OR Parity error OR Escape error (ESC followed by EOP or EEP or ESC).
NOTE Disconnect error only enabled after First Bit Received. Parity Error, Escape Error, gotFCT, gotN-Char, gotTime-Code only enabled after First NULL Received (i.e. gotNULL asserted). Thus RxErr OR gotFCT OR gotN-Char OR gotTime-Code is really RxErr OR (gotNULL AND (gotFCT OR gotN-Char OR gotTime-Code)).

Figure 20: State diagram for SpaceWire link interface

8.5.2 Definition of states

8.5.2.1 General

A table listing the exit conditions from each state is included in annex B for clarification purposes.

8.5.2.2 ErrorReset

a. The *ErrorReset* state shall be entered after a system reset, after link operation is terminated for any reason or if there is an error during link initialization.

b. In the *ErrorReset* state the Transmitter and Receiver shall all be reset.

c. When the reset signal is de-asserted the *ErrorReset* state shall be left unconditionally after a delay of 6,4 μs (nominal) and the state machine shall move to the *ErrorWait* state.

d. Whenever the reset signal is asserted the state machine shall move immediately to the *ErrorReset* state and remain there until the reset signal is de-asserted.

8.5.2.3 ErrorWait

a. The *ErrorWait* state shall be entered only from the *ErrorReset* state.

b. In the *ErrorWait* state the receiver shall be enabled and the transmitter shall be reset.

NOTE This allows the receiver to start the disconnection detection mechanism (after registering a transition on the D or S line) and to begin looking for the arrival of a NULL.

c. If a NULL is received then the gotNULL condition shall be set.

NOTE This condition is acted upon in the *Started* state.

d. The *ErrorWait* state shall be left unconditionally after a delay of 12,8 μs (nominal) and the state machine shall move to the *Ready* state.

e. If, while in the *ErrorWait* state, a disconnection error is detected, or if after the gotNULL condition is set, a parity error or escape error occurs, or any character other than a NULL is received, then the state machine shall move back to the *ErrorReset* state.

NOTE The *ErrorReset* and *ErrorWait* state with their 6,4 μs and 12,8 μs delays ensure that the receivers at both ends of a link are enabled before either end begins transmission, following an exchange of silence (see subclause 8.9.4).

8.5.2.4 Ready

a. The *Ready* state shall be entered only from the *ErrorWait* state.

b. In the *Ready* state the link interface is ready to initialize as soon as it is allowed to do so. The receiver shall be enabled and the transmitter shall be reset.

c. If a NULL is received then the gotNULL condition shall be set.

NOTE This condition is acted upon in the *Started* state.

d. The state machine shall wait in the *Ready* state until the [Link Enabled] guard becomes true (see subclause 8.6) and then it shall move on into the *Started* state.

e. If, while in the *Ready* state, a disconnection error is detected, or if after the gotNULL condition is set, a parity error or escape error occurs, or any character other than a NULL is received, then the state machine shall move to the *ErrorReset* state.

NOTE In the *Ready* state the two receivers are enabled and the state machine is waiting for the local host system to command the link to start.

8.5.2.5 Started

a. The *Started* state shall be entered from the *Ready* state when the link interface is enabled.

NOTE In the *Started* state the state machine begins making a connection with the link interface at the other end of the link by sending one or more NULLs.

b. When the *Started* state is entered a 12,8 µs (nominal) timeout timer shall be started.

c. In the *Started* state the receiver shall be enabled and the transmitter shall send NULLs.

d. If a NULL is received then the gotNULL condition shall be set.

e. The state machine shall move to the *Connecting* state if the gotNULL condition is set.

NOTE The NULL that set the gotNULL condition can have been received in the *ErrorWait*, *Ready* or *Started* states.

f. In the *Started* state the sending from the transmitter of at least one NULL shall be requested before moving to the *Connecting* state.

g. If, while in the *Started* state, a disconnection error is detected, or if after the gotNULL condition is set, a parity error or escape error occurs, or any character other than a NULL is received, then the state machine shall move to the *ErrorReset* state.

h. If the 12,8 µs timeout timer referred to in point b. above expires (i.e. no NULL received since leaving the *ErrorReset* state) then the state machine shall move to the *ErrorReset* state.

NOTE In the *Started* state the attempt to make a connection across the link is started. NULLs are transmitted and the receiver is waiting to receive a NULL.

8.5.2.6 Connecting

a. The *Connecting* state shall be entered from the *Started* state after a NULL is received (gotNULL condition set).

b. On entering the *Connecting* state a 12,8 µs timeout timer shall be started.

c. In the *Connecting* state the receiver shall be enabled and the transmitter shall be enabled to send FCTs and NULLs.

d. If an FCT is received (gotFCT condition true) the state machine shall move to the *Run* state.

e. If, while in the *Connecting* state, a disconnect error, parity error or escape error is detected, or if any character other than NULL or FCT is received, then the state machine shall move to the *ErrorReset* state.

f. If the 12,8 µs timeout referred to in point b. above occurs then the state machine shall move to the *ErrorReset* state.

NOTE The *Connecting* state is entered when the link interface (end A) receives a NULL, waiting then for the reception of an FCT indicating that the other end of the link (end B) has also received a NULL. When the link interface receives a NULL and an FCT it means that communication is established in both directions. If an FCT fails to arrive within 12,8 μs then something is wrong with the link connection and so the link interface is reset once more (*ErrorReset* state) and connection is attempted once again.

8.5.2.7 Run

a. The *Run* state shall be entered from the *Connecting* state.

NOTE In the *Run* state the receiver is enabled and the Transmitter is enabled to send Time-Codes, FCTs, N-Chars and NULLs.

b. If the link interface is disabled, or if a disconnect error, parity error, escape error or credit error is detected (see subclause 8.5.3), while in the *Run* state, then the state machine shall move to the *ErrorReset* state.

NOTE The *Run* state is the state for normal operation where link connection has been made and L-Chars and N-Chars can flow freely in both directions across the link. The link remains in the *Run* state until an error occurs or until the link is disabled.

8.5.3 Transitions types

8.5.3.1 Reset

Reset represents power on reset, other hardware reset or software commanded reset.

8.5.3.2 After *t* μs

After 6,4 μs or after 12,8 μs represents a delay of the specified time measured from when the current state is entered. The actual time intervals are nominal delays (see subclause 8.11).

8.5.3.3 [Link Enabled]

[Link Enabled] is a condition for the transition to occur (i.e. a guard). [Link Enabled] can be set true by software or hardware (see subclause 8.6).

8.5.3.4 gotNULL

a. A gotNULL condition shall be asserted when a NULL is received

NOTE GotNULL means that a NULL detection sequence (see Figure 21) has been received.

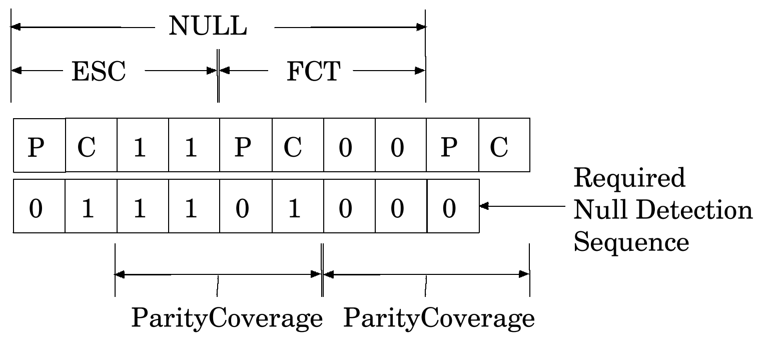
b. NULL detection shall be enabled whenever the receiver is enabled.

c. Any sequence of bits encountered after reset (*ErrorReset* state) prior to the first NULL being received shall be ignored.

d. NULL detection shall include all three parity bits related to the NULL, i.e. the parity bit that covers the data-control flag of the ESC character, the parity bit that covers the ESC character, and the parity bit that covers the FCT character. Hence the NULL shall be detected and gotNULL asserted, when the 011101000 sequence of bits is received as illustrated in Figure 21.

NOTE During initialization the character following a NULL is a control character (either another NULL or an FCT) so the last parity bit of the NULL is zero.

e. If a parity error occurs within the NULL detection sequence then the gotNULL condition shall not be asserted.



P = Parity Bit (Odd Parity)
C = Data-Control Flag (=1)

Figure 21: NULL detection sequence

8.5.3.5 gotFCT

FCTs shall be valid only when received in the *Connecting* and *Run* states.

NOTE 1 If received in any other state they represent an error.

NOTE 2 gotFCT means that an FCT has been received.

8.5.3.6 gotN-Char

An N-Char received when the exchange-level state machine is not in the *Run* state shall be interpreted as an error.

NOTE gotN-Char means that an N-Char has been received.

8.5.3.7 GotTime-Code

A time-code received when the exchange-level state machine is not in the *Run* state shall be interpreted as an error.

NOTE gotTime-Code means that a time-code has been received.

8.5.3.8 [Link Disabled]

[Link Disabled] is a condition set by external hardware or software in order to disable and stop the link interface (see subclause 8.6).

8.5.3.9 RxErr

8.5.3.9.1 General

RxErr or receiver error is shorthand for disconnect error, parity error or escape error.

8.5.3.9.2 Disconnect error

a. The disconnect error shall be detected by the receiver in the link interface.

NOTE Disconnect error is an error condition asserted when the length of time since the last transition on the D or S lines was longer than 850 ns nominal (see subclause 8.11).

b. The disconnect detection mechanism shall be activated after leaving the *ErrorReset* state as soon as the first edge is detected on the D or S line.

8.5.3.9.3 Parity error

a. The parity error shall be detected by the receiver in the link interface.

NOTE The parity error event occurs if a parity error is detected (see subclause 7.4).

b. Parity detection shall be enabled whenever the receiver is enabled after the first NULL is received.

8.5.3.9.4 Escape error

a. The Escape error shall be detected by the receiver in the link interface.

NOTE The escape error event occurs if an ESC character is followed by any control character other than an FCT (ESC followed by FCT is a NULL, see subclause 7.3). ESC followed by a data character is a Time-Code (subclause 7.3).

b. Escape error detection shall be enabled whenever the receiver is enabled after the first NULL is received.

8.5.3.10 Credit error

The credit error shall be detected by the receiver in the link interface.

NOTE Credit error occurs if data is received when the host system is not expecting any more data, i.e. when all the N-Chars expected, according to the requested eight more N-Chars and subsequent transmitted FCTs, have been received. A credit error also occurs if an FCT is received when the credit error is at or close to its maximum value (see subclause 8.3 f.). A credit error indicates that some undetected error has occurred on the link, which has caused the corruption of the credit count.

8.5.3.11 Character sequence error

a. The character sequence error shall be detected by the state machine in the link interface.

b. Any characters received before a NULL is received shall be ignored.

c. Once a NULL is received, an FCT received before a NULL is sent shall indicate an error (i.e. FCT received in *ErrorWait*, *Ready* or *Started* state).

d. An N-Char should only be expected after both a NULL and an FCT are received otherwise an error occurs (i.e. N-Char can only be received in the *Run* state).

NOTE In the state diagram of Figure 20, the invalid gotFCT or gotN-Char events are shown explicitly, rather than as a general character sequence error event.

8.6 AutoStart (normative)

a. A link interface should be able to be commanded to start automatically on receipt of a NULL. In this case the Link Enabled condition in the state machine should be set as follows:

[Link Enabled] = (NOT [Link Disabled]) AND ([LinkStart] OR ([AutoStart] AND gotNULL))

Where:

LinkDisabled is the flag set by software or hardware to indicate that the link is disabled. This corresponds to the Link Disabled condition in the state diagram.

LinkStart is a flag set by software or hardware to start a link, i.e. to cause the transition from the *Ready* state to the *Started* state.

AutoStart is a flag set by software or hardware to request the link to start automatically on receipt of a NULL.

gotNULL is the flag indicating that the link interface has received a NULL detection sequence as defined in subclause 8.5.3.4.

b. LinkStart and AutoStart should only be acted upon when the link interface is not disabled i.e. [LinkDisabled] = False.

NOTE The AutoStart facility enables the setting up of a system where one end (end A) of the link is held waiting for the other end (end B) to attempt connection. As soon as end B tries to connect end A responds immediately. This allows the control of the connection of a link from one end of the link only.

8.7 Link initialization

This subclause explains how the state diagram given in subclause 8.5 handles link initialization, going from the reset of one end of a link through to the link operating normally and sending data in both directions. The basic state diagram with the receiver error conditions removed is illustrated in Figure 22.

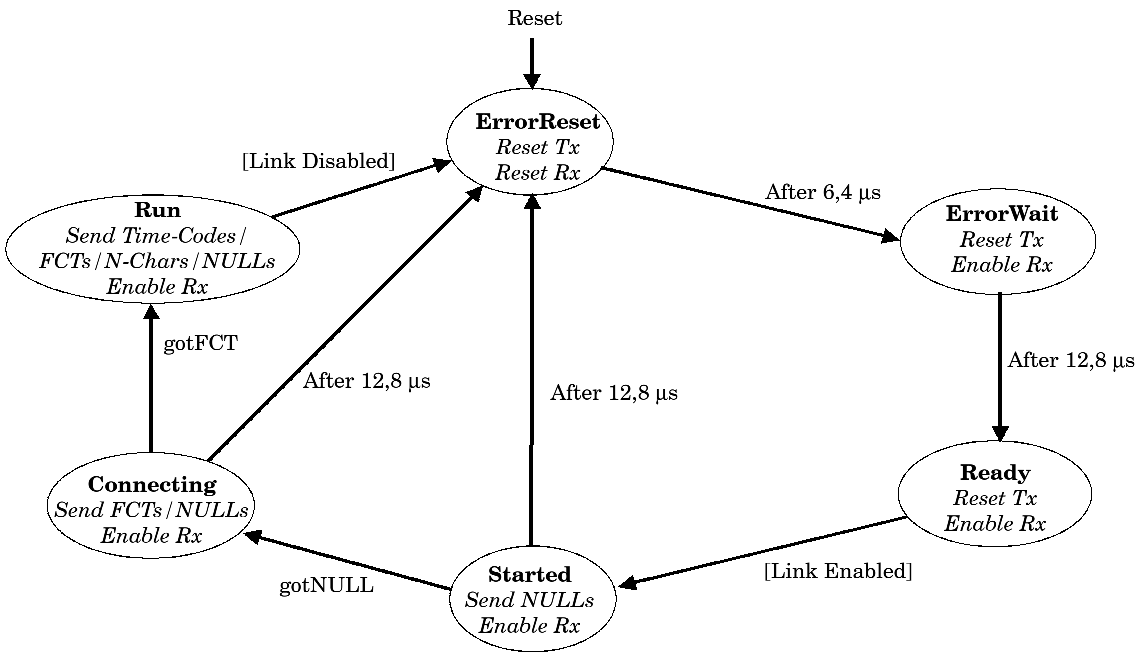


Figure 22: Basic state diagram for SpaceWire link interface

After a link interface (one end of a link) is reset, it enters the *ErrorReset* state where the transmitter and receiver are reset. The transmitter reset is a controlled reset, resulting first in the transmitter stopping transmission, followed by resetting of the Strobe signal and then the Data signal. This sequence avoids the simultaneous transition of both Data and Strobe signals.

The link interface remains in the *ErrorReset* state for approximately 6,4 µs and then moves to the *ErrorWait* state. In the *ErrorWait* state the transmitter remains disabled, but the receiver is enabled so that it can begin searching for NULLs.

The link interface remains in the *ErrorWait* state for 12,8 µs and then moves into the *Ready* state. The 6,4 µs delay from *ErrorReset* to *ErrorWait* and the 12,8 µs delay from *ErrorWait* to *Ready* make sure that the receivers at both ends of a link are ready to receive characters before either end starts transmission.

The link interface can be enabled in many possible ways, for example by software command, automatically when the receiver detects a NULL, or the link can be permanently enabled (see subclause 8.6). When a link interface is enabled the [LinkEnabled] condition becomes true. The link interface moves from the *Ready* state to the *Started* state as soon as the link is enabled.

In the *Started* state the link interface instructs the transmitter to start sending NULLs. It remains in this state until the receiver detects that a NULL is received over the link or until a connection timeout expires. The connection timeout is set to a nominal 12,8 µs since this period is generated for the *ErrorWait* state timeout. If a NULL is received then the link interface moves to the *Connecting* state. If no NULL is received within 12,8 µs it moves to the *ErrorReset* state. In the latter case the link interface goes through the reset sequence (*ErrorReset, ErrorWait, Ready*) and attempts to make a connection again a short time later.

In the *Connecting* state the link interface sends some FCTs (and NULLs) and waits for the reception of an FCT. If an FCT is received the link interface moves on to the *Run* state. If an FCT is not received within 12,8 µs then link connection was not made properly, so the link interface moves back to the *ErrorReset* state. The link interface then goes through the reset sequence (*ErrorReset, ErrorWait, Ready*) and attempts to make a connection again a short time later.

When the link enters the *Run* state it starts normal operation, sending and receiving data and control characters. It remains in the *Run* state until the link is disabled. The link interface then moves through the reset sequence (*ErrorReset, ErrorWait, Ready*) and stays in the ready state until the link is enabled once more.

Table 8 and Figure 23 illustrate an example of an initialization sequence. Link interface A is at one end of the link and link interface B is at the other end.

Table 8: Example of initialization sequence

Link interface end A	**Link interface end B**	**Event or condition causing transition**
ErrorReset	*ErrorReset*	End A times out after 6,4 µs and moves to the *ErrorWait* state.
ErrorWait	*ErrorReset*	End B times out after 6,4 µs and moves to the *ErrorWait* state.
ErrorWait	*ErrorWait*	End A times out after 12,8 µs and moves to the *Ready* state.
Ready	*ErrorWait*	End A is link enabled so moves to the *Started* state.
Started Sending NULLs	*ErrorWait*	End B detects NULL sent from end A. This is registered as gotNULL by end B. There is no state change.
Started Sending NULLs	*ErrorWait*	End B times out after 12,8 µs and moves to the *Ready* state.
Started Sending NULLs	*Ready*	End B is link enabled so moves straight to the *Started* state.
Started Sending NULLs	*Started Sending NULLs*	End B sends a NULL. It has already detected a NULL (gotNULL) so can now move to the *Connecting* state.
Started Sending NULLs	*Connecting*	End A detects NULL sent from end B (gotNULL) and can move to the *Connecting* state. End B sends out FCTs (and NULLs).

Table 8: Example of initialization sequence *(continued)*

Link interface end A	Link interface end B	Event or condition causing transition
Connecting	*Connecting*	End A sends out FCTs (and NULLs). End B sends out FCTs (and NULLs). End A receives an FCT and moves to the *Run* state.
Run	*Connecting*	End A sends out FCTs, N-Chars and NULLs. End B receives an FCT and moves to the *Run* state.
Run	*Run*	Both ends are in the *Run* state and begin normal operation sending and receiving N-Chars, FCTs and NULLs.

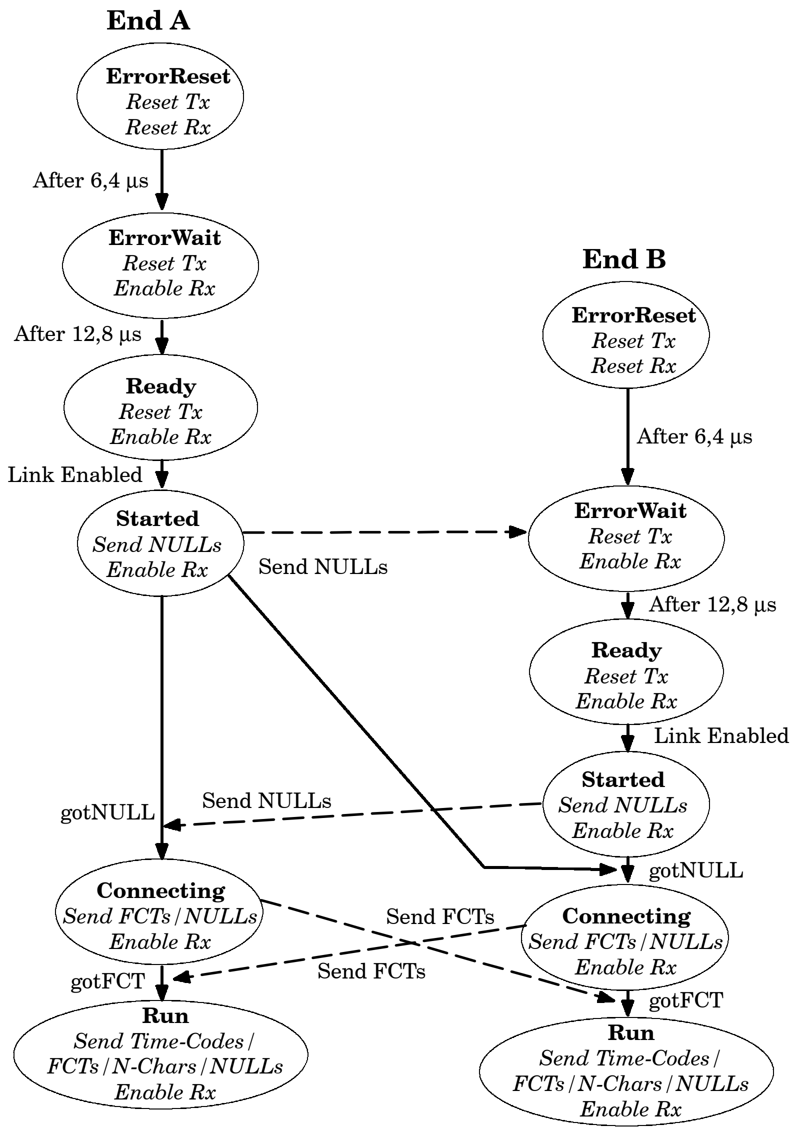


Figure 23: Example of typical initialization sequence

A link only sends FCTs once it receives a NULL. So, when a link receives an FCT it knows that the link is connected in both directions. A NULL correlation (or any other method of synchronization through NULL detection) in the *ErrorWait*, *Ready* and *Started* states ensures proper character synchronization. The NULL or FCT handshake sequence ensures that the link is connected in both directions before normal link operation begins.

The time taken from a link being enabled in the *Started* state to normal operation in the *Run* state can be as little as the time taken to transfer two NULLs and an FCT. End A is enabled and sends a NULL. End B is autostart enabled when it receives the NULL from end A and sends a NULL followed by an FCT. End A receives the NULL from end B and sends an FCT. Both ends receive FCTs and move to the *Run* state. At a link data signalling rate of 10 Mb/s this can take just 2 µs.

8.8 Normal operation

In normal operation both ends of the link are in the *Run* state and are sending and receiving Time-Codes, FCTs, N-Chars and NULLs.

An example is a host system with buffer space sufficient to hold 16 N-Chars. This host system at one end of a link (end A) indicates that it is ready to receive N-Chars by twice flagging that it has room for 8 more characters to the link interface. The link interface sends two FCTs to the other end of the link (end B), which increments its credit count accordingly (from zero to 16). The link interface at end B indicates to its host system that it is ready to transmit data (N-Chars). When the host system at end B has data to transfer, it passes it to the link interface, which sends it across the link to end A. As each character is transmitted by the link interface (end B), it decrements its credit count until it reaches zero, at which point the link interface (end B) indicates to its host system that it is not ready to transfer any more data. The data received at end A is passed on to its host system, which places it in its 16 character buffer. As the host system uses the data out of this buffer it makes space for the reception of more data. As soon as there is space for another 8 more characters it flags this to the link interface, which then sends out another FCT informing end B that 8 more normal characters may be sent.

8.9 Error detection (normative)

8.9.1 General

a. There are five forms of receiver error that can be detected and acted upon at the exchange level – disconnect errors, parity errors, escape errors, credit errors and character sequence errors. Whenever one of these errors occur both character synchronization and flow-control status shall cease to be valid.

b. The error detecting end shall be reset and re-initialized to recover character synchronization and flow control status.

   NOTE This forces the remote end to do the same.

c. Empty packets, which can be received in addition to these errors, shall be discarded.

8.9.2 Handling receiver errors

8.9.2.1 Disconnect error

8.9.2.1.1 General

An operational link interface sends Time-Codes, FCTs, N-Chars or NULLs continuously, and thus the Data, Strobe or both signals are always changing.

8.9.2.1.2 Handling

a. The receiver shall detect a disconnection when the time interval from the last transition on either the Data or Strobe signal exceeds the disconnect‐detection time.

b. The disconnect‐detection time shall be 850 ns nominal (see subclause 8.11.2 for the disconnect timing specification).

NOTE A disconnection cannot be detected unless the receiver has previously received at least one bit.

c. The detection of a disconnection shall provoke a disconnect error.

NOTE A disconnect error can either be caused when one end of the link is disabled or when the link is physically disconnected (intentionally or unintentionally).

d. If a physical disconnection is the cause of the disconnect error then both ends of the link shall try repeatedly to make a connection until the link is reconnected or until the link interfaces are disabled.

e. If a disconnect error is detected then the link interface shall follow the exchange of silence error recovery procedure described in subclause 8.9.4.

f. If the disconnect error occurs in the *Run* state then the disconnect error shall be flagged up to the network level as a link error (see subclause 8.9.5).

8.9.2.2 Parity error

a. When a parity bit is received it shall be checked (see subclause 7.4).

b. If a parity error occurs after the first NULL is received, then the link interface shall follow the error recovery procedure described in subclause 8.9.4.

c. If the parity error occurs in the *Run* state then the parity error shall be flagged up to the network level as a link error (see subclause 8.9.5).

8.9.2.3 Escape error

a. An ESC character shall only be used to form either the NULL i.e. ESC followed by FCT, or the Time‐Code, i.e. ESC followed by a single data character (see subclause 7.3).

b. If a ESC character is received followed by any control character other than an FCT then the link interface shall follow the error recovery procedure described in subclause 8.9.4.

c. If the escape error occurs in the *Run* state then the escape error shall be flagged up to the network level as a link error (see subclause 8.9.5).

8.9.2.4 Credit error

a. A credit error shall be identified when in the *Run* state, a normal character is received when the host system is not expecting any N‐Chars.

NOTE A credit error can be caused if an error occurs undetected by the parity bit (e.g. two bits in error) which results in one or more spurious FCTs.

b. In the event of a credit error the link interface shall follow the error recovery procedure described in subclause 8.9.4.

c. If the credit error occurs in the *Run* state then the credit error shall be flagged up to the network level as a link error (see subclause 8.9.5).

8.9.2.5 Character sequence error

8.9.2.5.1 General

During initialization it is possible for a link interface to receive FCTs or N-Chars when they are not expected. Any unexpected characters are caught by the exchange-level state machine resulting in the link being reset and re-initialized (see Figure 20).

8.9.2.5.2 Handling

As a character sequence error can only occur during link initialization, it shall not be flagged up to the network level as a link error (see subclause 8.9.5).

8.9.3 Handling empty packets

8.9.3.1 General

Empty packets, i.e. an EOP or EEP followed immediately by another EOP or EEP, are not expected, but they can occur if a packet terminated by an EEP comprises just the header and the EEP (see subclause 11.4) and the header characters are stripped off as the packet progresses through a network leaving just the EOP (see subclauses 10.2.7 and 10.3.3).

8.9.3.2 Handling

a. In the *Run* state, if the next N-Char received after an EOP or EEP is received is another EOP or EEP, then the link interface may discard the second EOP or EEP (see subclause 10.6.4.3).

b. This error shall not be flagged up to the network level as a link error (see subclause 8.9.5).

   NOTE In this way empty packets are discarded quietly.

8.9.4 Exchange of silence error recovery procedure

8.9.4.1 General

When one end of the link (end A) is disabled or detects an error, it ceases transmission. As described in subclause 8.9.2.1, this causes a disconnect error at the other end of the link (end B). End B then ceases transmission, resulting in a disconnect error at end A. This procedure is known as an "exchange of silence" (see Figure 7).

8.9.4.2 Handling

After an exchange of silence, both ends of the link shall cycle through the reset sequence (*ErrorReset, ErrorWait, Ready*) ending up in the *Ready* state ready to begin operation once enabled, proceeding then as follows.

a. If both ends are enabled then they shall move to the *Started* state and re-initialize.

b. If one end (end A) is disabled and the other end (end B) is enabled then end B shall move from the *Ready* state to the *Started* state and shall send NULLs for 12,8 μs. Since end A is disabled it cannot respond. End A shall, however, have started its disconnect timer and shall also have registered that a NULL was received. When end B completes the 12,8 μs timeout it shall move to the *ErrorReset* state and disconnect (stop its output). End A can detect the disconnection so shall also move to the *ErrorReset* state. Both ends shall once again move through the reset sequence. This series of events shall continue until either end A is enabled or end B is disabled.

8.9.5 Reporting link errors to network level

a. Receiver errors (i.e. disconnect error, parity error, escape sequence error, character sequence error and credit error) during link initialization (i.e. *ErrorReady*, *ErrorWait*, *Ready*, *Started* and *Connecting* states) shall not be reported to the network level.

b. Once a link connection is established (*Run* state), a receiver error represents a failure of the link connection and shall be reported to the network level so that appropriate action for error recovery or reporting, as described in sub-clause 10.6, can be taken.

c. A link error is reported to the network level whenever any of the following errors occur while a link interface is in the *Run* state: disconnect error, parity error, escape sequence error, or credit error.

NOTE The exclusion of character sequence error from this list is because a character sequence error can only occur during initialization.

8.10 Exception conditions

8.10.1 General

The following subclauses describe the exception conditions where events do not follow the expected sequence.

8.10.2 Disconnect error while waiting to start

"Waiting to start" means that a link interface is in either the *ErrorReset*, *Error-Wait*, *Ready* or *Started* state. A disconnect error cannot be detected while waiting to start, unless the other end of the link (end B say) has sent at least one bit, so that the disconnect detect mechanism at end A can be activated. End B then gives up waiting for end A to send a NULL and moves to the *ErrorReset* state and stops its transmitter, thus causing the disconnect. An alternative possibility is that the link becomes physically disconnected. Table 9, Table 10, Table 11 and Table 12 illustrate the various sequences of events starting from when end B has just moved to the *ErrorReset* state.

If a physical disconnection has occurred then both ends of the link continue to try to make a connection, cycling around the reset sequence, until they are disabled or until the connection is re-established.

Table 9: End A in ErrorReset state

Link Interface end A	Link interface end B	Event or condition causing transition
ErrorReset	*Started*	End B times out while waiting to receive a NULL from end A or end B detects a disconnect. End B moves to the *ErrorReset* state.
ErrorReset	*ErrorReset*	End A and end B are both in the *ErrorReset* state. They then step through the reset sequence and start again when both ends are enabled.

Table 10: End A in ErrorWait state

Link interface end A	Link interface end B	Event or condition causing transition
ErrorWait	*Started*	End B times out while waiting to receive a NULL from end A or end B detects a disconnect. End B moves to the *ErrorReset* state.
ErrorWait	*ErrorReset*	End B stops transmission. This is detected at end A as a disconnect error. End A moves to the *ErrorReset* state.
ErrorReset	*ErrorReset*	Both ends step through the reset sequence and start again when both ends are enabled.

Table 11: End A in Ready state

Link interface end A	Link interface end B	Event or condition causing transition
Ready	*Started*	End B times out while waiting to receive a NULL from end A or end B detects a disconnect. End B moves to the *ErrorReset* state.
Ready	*ErrorReset*	End B stops transmission. This is detected at end A as a disconnect error. End A moves to the *ErrorReset* state.
ErrorReset	*ErrorReset*	Both ends step through the reset sequence and start again when both ends are enabled.

Table 12: End A in Started state

Link interface end A	Link interface end B	Event or condition causing transition
Started	*Started*	End B times out while waiting to receive a NULL from end A or end B detects a disconnect. End B moves to the *ErrorReset* state.
Started	*ErrorReset*	End B stops transmission. This is detected at end A as a disconnect error. End A moves to the *ErrorReset* state.
ErrorReset	*ErrorReset*	Both ends step through the reset sequence and start again when both ends are enabled.

8.10.3 Link connected in one direction but not in the other

A link can be connected in one direction and not in the other while a link is in the process of being plugged in or if there is a break in the link cable.

In this case events follow the sequence listed in the Table 13. For convenience it is assumed that both links are in the *Started* state and that end A is connected to end B, but end B is not connected to end A.

Table 13: Link connected in one direction (A to B) but not in other

Link interface end A	**Link interface end B**	**Event or condition causing transition**
Started	*Started*	End A is sending NULLs to end B and these are received, starting the disconnect timer of end B and registering as GotNULL at end B. End B moves therefore to the *Connecting* state. End B is also sending NULLs to end A but these are not received at end A because the link is not connected in this direction.
Started	*Connecting*	End A is in the *Started* state waiting for NULLs to arrive. After waiting for 12,8 µs end A times out and moves to the *ErrorReset* state. End B sends NULLs and FCTs to end A but these are not received at end A. End B is able to receive FCTs but none are sent by end A because end A has not received a NULL
ErrorReset	*Connecting*	End B continues to send NULLs and FCTs and is able to accept FCTs. End A ceases transmission and this is detected at end B as a disconnect. End B moves to the *ErrorReset* state.
ErrorReset	*ErrorReset*	Both ends are in the *ErrorReset* state and they now cycle through the reset sequence (e.g. *ErrorReset, ErrorWait,* and *Ready*) until the link is properly connected or until one or both link interfaces are disabled.

8.10.4 Parity error while waiting to start

Parity errors are only recognized after a NULL is received. If a parity error occurs during link initialization the effect is identical to a disconnect error.

8.10.5 One end starts as other end disconnects

One end (end A) arrives at the *Start* state 12,8 µs before the other end (end B) arrives at the *Start* state. The sequence of event is illustrated Table 14.

Table 14: One end starts as other end disconnects

Link interface end A	Link interface end B	Event or condition causing transition
Started	*Ready*	The timeout timer at end A expires (after 12,8 µs) so end A moves to the *ErrorReset* state. End B, in the *Ready* state, has just been enabled and now moves to the *Started* state.
ErrorReset	*Started*	End B has already received a NULL from end A (gotNULL is TRUE) so moves straight on into the *Connecting* state.
ErrorReset	*Connecting*	End A stops transmitting and causes end B to detect a disconnect. End B then moves on into the *ErrorReset* state.
ErrorReset	*ErrorReset*	Both ends step through the reset sequence and start properly next time round.

8.10.6 D connected, S disconnected

If D is connected and S disconnected then the clock generated in the receiver follows the Data signal, i.e. there is a clock edge every time the Data signal changes. This results in a continuous sequence of 0101010101.

During initialization this sequence is ignored as it does not produce a NULL so initialization fails until the Strobe signal is properly connected. In this case the link interface cycles round *ErrorReset*, *ErrorWait*, *Ready*, *Started* until full link connection is achieved or until the link is disabled.

If S becomes disconnected after a NULL is received, this sequence produces a parity error for both control characters (4 bits) extracted from the 01010101010 sequence, but does not produce a parity error for data characters (10 bits).

If the disconnection occurs so that the 01010101010 sequence is regarded as control characters a parity error is detected and the link interface cycles through *ErrorReset*, *ErrorWait*, *Ready* and *Started* until S is connected or the link is disabled.

If the disconnection occurs so that the 010101010 sequence is regarded as data characters then no parity error is detected and the link interface receives a continuous series of data characters with the value AA hex (note that SpaceWire is least significant bit first).

8.10.7 S connected, D disconnected

If S is connected and D disconnected then the clock generated in the receiver follows the Strobe signal, i.e. there is a clock edge every time the Strobe signal changes. This results in a continuous sequence of 1111111111 since the Data signal input goes to 1 when the data line is disconnected. If the data line is shorted to ground then the continuous sequence 0000000000 is received.

During initialization either sequence is ignored as it does not produce a NULL so initialization fails until the Data signal is properly connected. In this case the link interface cycles round *ErrorReset*, *ErrorWait*, *Ready*, *Started* until full link connection is achieved or until the link is disabled.

If D becomes disconnected after a NULL is received the received data sequence quickly produces a parity error because the parity is even for both control characters (4 bits) and data characters (10 bits) extracted from the received sequence, whereas the expected parity is odd (see subclause 7.4). The parity error causes the link interface to cycle through *ErrorReset*, *ErrorWait*, *Ready* and *Started* until D is connected or the link is disabled.

8.10.8 One side of differential pair disconnected

The effect of disconnecting one side of the differential pair depends upon the values of the internal bias resistors at the interface, on the length of cable and on the grounding arrangements. Three cases are possible:

a. The Data and Strobe signals are still received correctly but with a much reduced noise margin. The link then continues operating with a significant increase in the number of detected parity errors.

b. The Strobe signal sits at logic 0 or logic 1. This is similar in effect to the D connected, S disconnected case of subclause 8.10.6.

c. The Data signal sits at logic 0 or logic 1. This is similar in effect to the S connected, D disconnected case of subclause 8.10.7 above.

8.11 Link timing (normative)

8.11.1 D and S reset timing

The delay between the reset of the Strobe signal and the Data signal shall be between 555 ns (i.e. the period of slowest permitted transmit clock, 2 MHz – 10 %) and the shortest clock cycle time for the transmitter (i.e. the period of maximum clock, dependent upon implementation).

8.11.2 Disconnect timing

The disconnect timeout of 850 ns nominal shall be from 727 ns (i.e. 8 cycles of 10 MHz + 10 % clock) to 1000 ns (i.e. 9 cycles of 10 MHz - 10 % clock).

8.11.3 Exchange timeout periods

a. The 6,4 μs (nominal) timeout period shall be from 5,82 μs (i.e. 64 cycles of 10 MHz + 10 % clock) to 7,22 μs (i.e. 65 cycles of 10 MHz - 10 % clock).

b. The 12,8 μs (nominal) timeout period shall be from 11,64 μs (i.e. 128 cycles of 10 MHz + 10 % clock) to 14,33 μs (i.e. 129 cycles of 10 MHz - 10 % clock).

8.12 System time distribution (normative)

8.12.1 General

As defined in subclause 7.3, a Time-Code comprises the ESC character followed by a single 8-bit data character. The data character contains two control flags and a six-bit time-count which may be the least significant six bits of system time.

8.12.2 Handling

a. Each node or router shall contain one six-bit time counter.

b. A single link interface shall manage the distribution of time.

c. The time-master interface shall have a TICK_IN input, which is asserted periodically (e.g. every millisecond) by its host system.

d. When the time master link interface receives a tick (TICK_IN asserted) it shall increment its time-counter and then send out a Time-Code with the six-bit time field of the data character set to the new value of the time-counter and the other two bits set to the value of the control flags.

NOTE The frequency at which the TICK_IN signal is driven is called the tick rate.

e. The Time-Code shall be sent out as soon as the current character or control code is transmitted.

f. Time-Codes shall not be sent out until a link interface is in the *Run* state (see subclause 8.5).

g. When the link interface at the other end of the link receives the Time-Code, it shall update its associated time-counter with the new time and assert its TICK_OUT output signal and copy the values of the two control flags in the Time-Code to the control-flag outputs.

h. The new time should be one more than the time-counter's previous time-value.

NOTE This fact can be used for checking on Time-Code validity.

i. If a link interface receives a Time-Code that is equal to its current six-bit time value then it shall not emit a TICK_OUT output signal.

NOTE This prevents repeated Time-Code propagation in a circular network.

j. When a link interface on a router or node receives a Time-Code it shall check that it is one more (modulo 64) than its current time setting.

k. The router or node shall then increment the router's time-count and emit a tick signal.

NOTE This tick signal propagates to all the output ports of the router so that they all emit the Time-Code. This Time-Code is the same value as that received by the router since the router time-counter has been incremented. If there is a circular connection then the router receives a Time-Code with the same time value as the router time-counter. The control-flags of the Time-Codes that are emitted are set to the control flag values of the received Time-Code that caused the subsequent emission of Time-Codes by the router.

l. If the router or node receives a Time-Code with the same time value as the router time-counter the Time-Code shall be ignored.

NOTE 1 In this way, time flows forwards through a network reaching all nodes, but is suppressed if it flows backwards due to a circular connection.

NOTE 2 With the provision of this basic time-distribution function, application level protocols can be used, for example, to distribute specific time values at full resolution (not just 6 bits) and to issue time dependent commands.

m. After reset or disconnect-reconnect (state machine in *ErrorReset* state) the time-counter shall be set to zero and any control-flag outputs shall be set to zero.

n. If a received Time-Code is not one more than (modulo 64) or equal to the current time-count at the receiving link interface, then either the Time-Code or the time-count shall be considered invalid.

NOTE This can happen if a Time-Code is lost, or if a link is reset or restarted after a disconnect.

o. If the Time-Code is invalid then the time-count shall be updated to the new value but the tick-output shall not be asserted.

NOTE 1 This prevents propagation of invalid Time-Codes across a network. When the next Time-Code is received it is expected that the time-counter matches the Time-Code and normal operation resumes.

NOTE 2 Nodes using the system time distribution function can either use the TICK_OUT signal as a periodic timing signal or use the value of the time-count as an indication of the least significant 6 bits of system time.

p. As a missing tick results in a timing discrepancy:

1. The TICK_OUT signal should not be used to increment a counter with the expectation that this counter corresponds to the system time.
2. Rather a time-lock or phase-lock technique should be used where a free running local time-counter is updated to be an exact multiple of the system tick rate every time the TICK_OUT signal is asserted.

NOTE 1 The reason for this is that when using the TICK_OUT signal as a periodic timing signal the Time-Code can be missed so that a TICK_OUT signal is missed.

NOTE 2 The accuracy with which system time can be distributed is dependant upon the number of links over which it is distributed and the operating rate of each of those links. A delay of at least 14 bit-periods (ESC + data character = (4 + 10) bits) is encountered for each link that the Time-Code traverses, due to the time taken for each link interface on the way to receive a Time-Code. This gives rise to a time-skew across a network of $ST_{\text{skew}} = 14N/R$, where N is the number of links traversed and R is the average link operating rate. Jitter is also introduced at each link interface due to the variation in time spent waiting for the transmitter to finish transmitting the current character or control code. At each link interface a delay of 0 bit periods to 10 bit periods can be encountered. Across a network, this gives rise to a total jitter of $ST_{\text{jitter}} = 10N/R$. For an average rate of 100 Mb/s and 10 links traversed, the time skew is 1,4 µs and the jitter 1,0 µs. The skew and jitter may be higher than indicated above depending on the implementation of the link interface. A time accuracy across a network of better than 10 µs may be difficult to achieve.

9
Packet level (normative)

9.1 General

The packet level protocol agrees with the principles of the DS-SE and DS-DE character level encoding given in clause 9 of the IEEE Standard 1355-1995 [1]. Some ambiguities in the IEEE Standard 1355-1995 are resolved in this Standard. See annex A for details of the differences between SpaceWire and IEEE Standard 1355-1995 and the reasons for the differences.

9.2 Packet composition

9.2.1 General

A packet shall comprise a destination address, a cargo and an end_of_packet (EOP or EEP) marker, i.e.

<destination address> <cargo> < end_of_packet >

9.2.2 Destination

a. The destination address shall consist of a list of zero or more destination identifiers (dest_id):

   <destination address> = <dest_id1> <dest_id2> 0 <dest_idN>

b. A destination identifier shall comprise one data character.

c. The destination identifier list shall not be delimited.

   NOTE 1 The case of zero destination identifiers in the destination list (i.e. the destination list is empty) is intended to support a network which is simply a single point-to-point link from source to destination.

   NOTE 2 The case of one or more destination identifiers in the destination list is intended to support routing of a packet across a network.

9.2.3 Cargo

a. The cargo shall comprise the data characters that are to transfer from the source to the destination.

b. The cargo shall contain one or more characters.

NOTE Subclause 8.9.3 describes what happens to empty packets.

c. Empty cargoes shall be allowed to move across the network.

NOTE A cargo can become empty if there is an error on the link.

9.2.4 End_of_Packet markers

a. There shall be two possible end_of_packet markers: EOP and EEP.

NOTE The EOP and EEP control character formats are described in subclause 7.3 of this Standard.

b. EOP (end_of_packet) shall be used as the normal end of packet marker indicating the end of a packet.

c. EEP (error end_of_packet) shall be used to indicate the end of a packet in which an error occurs. The data in this packet can be valid, but the packet shall be prematurely terminated at the point that the error occurred.

d. The first data character following either end_of_packet marker shall be taken as the first character of the next packet.

9.3 N-Char interleaving

N-Chars from one packet shall not be interleaved with N-Chars from another packet.

NOTE N-Chars can be interleaved with FCTs, NULLs and Time-Codes as explained in 8.3 n.

10

Network level

10.1 General

This clause describes the network level. First the basic concepts are described and then the SpaceWire routing switches, nodes and networks are defined. Network level errors are described and the network level error recovery protocol is defined.

10.2 Network and routing concepts

10.2.1 Purpose

In this subclause the basic concepts of packet routing networks that apply to SpaceWire networks are introduced.

10.2.2 Networks

A network is made up of a number of links, nodes and routing switches. The nodes are the sources and destinations of packets. For example, a processor is a type of network node.

Links provide the means for passing packets from one node to another.

Nodes can be either directly connected by links or connected via routing switches. Usually a node can only support a few links (e.g. six links) and so can only be directly connected to a limited number of other nodes (e.g. six other nodes).

Routing switches connect together many nodes and provide a means of routing packets from one node to one of many other possible nodes.

An example network comprising several nodes and routing switches is illustrated in Figure 24. This figure is for illustrative purposes and does not show a practical network. Packets can be transferred from one node to another through one or more routing switches, or directly from node to node where direct connections exist.

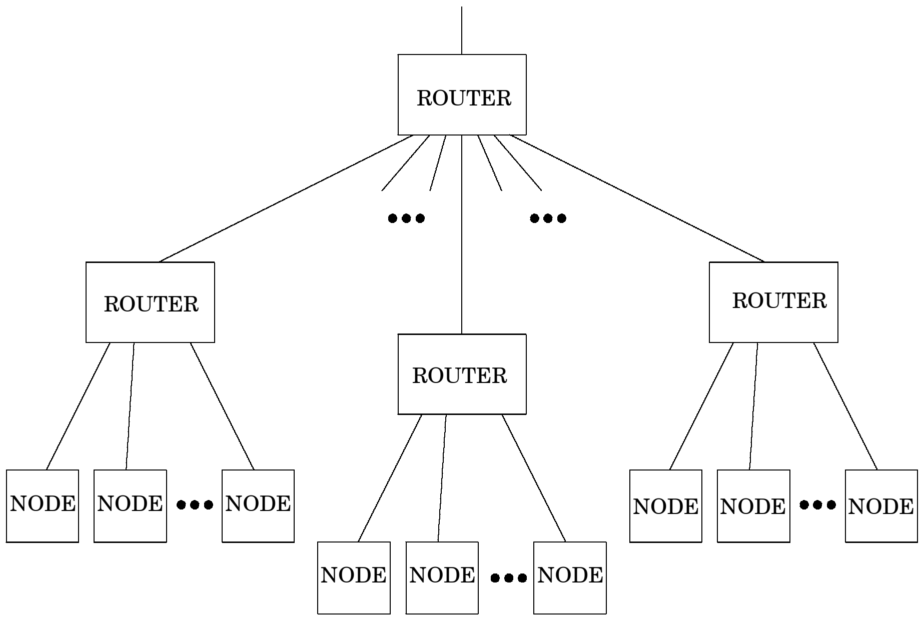


Figure 24: An example network

There are two types of routing switch: static and dynamic. A static routing switch sets up connections between nodes and does not change them very often. Dynamic switches change the routing frequently, usually on a packet by packet basis, and are consequently also known as packet routing switches. SpaceWire routing switches are dynamic, packet routing switches. Packets can be interleaved across data links and routing switches to provide many virtual communication channels across a few physical data links.

10.2.3 Packets

Data are split into packets so that it can be transported in manageable chunks across a network. Packets of data are the smallest elements of data that are handled at the network level. Packets are regarded as indivisible by the network level and are transported whole across a network.

SpaceWire packets have a simple packet structure (see clause 9):

- Destination: the address of the destination node or task.
- Cargo: the data to deliver.
- End_of_Packet marker: a special character which indicates the end of a packet.

10.2.4 Flow control

Flow control is used to manage the movement of packets across a link connecting a node or a router to another node or router. A node or router accepts data only when buffer space for that data is available in the receiving node or router. When the receive-buffer becomes full the receiver stops the transmitting node from sending any more data.

SpaceWire uses flow control tokens to manage the flow of data across a link connecting one node or router to the next node or router (see subclause 8.3).

10.2.5 Wormhole routing

Wormhole routing is a particular form of packet routing. Each packet contains a header which holds the destination node address either as the route through the network or as the identity of the destination node. As soon as the header for a packet is received the switch determines the output port to route the packet to by checking the destination address. If the requested output port is free then the packet is routed immediately to that output port. That output port is now marked as busy until the last character of the packet has passed through the switch – indicated by the end of packet marker being detected by the switch. Wormhole routing cuts down on the amount of buffering used within each switch, compared to a store and forward technique where an entire packet is first received and stored before it is sent out of the switch.

Wormhole routing is illustrated in Figure 25 which shows a packet being sent from one node to another through a routing switch (router). The header of the packet is marked as black and the rest of the packet as grey. As soon as the router receives the header it checks the requested output port. If the output port is free then the router makes a connection between the input port and the output port. The packet then flows through the router. When the end_of_packet (EOP or EEP) marker is received by the switch the router terminates the connection and frees the output port for its next packet which can come from any input port.

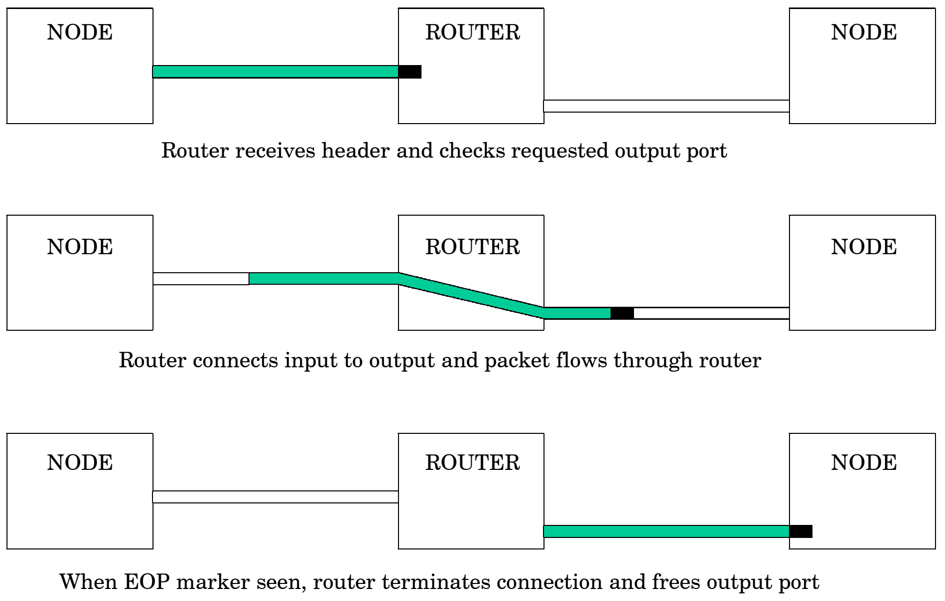

NODE
ROUTER
NODE
Router receives header and checks requested output port
NODE
ROUTER
NODE
Router connects input to output and packet flows through router
NODE
ROUTER
NODE
When EOP marker seen, router terminates connection and frees output port

Figure 25: Wormhole routing

If a requested output port is busy then the input port halts the incoming packet until the output port becomes free. This is achieved by the input port ceasing to send flow control tokens to the source node. The link connecting the source node to the routing switch is then blocked until the routing switch output finishes transferring its current packet and becomes free to transmit the new packet.

10.2.6 Cascading

Direct connection between nodes can be used to interconnect a limited number of nodes (e.g. six nodes). A single routing switch can usually connect many more nodes depending on the size of the switch (e.g. 32 nodes). When larger networks are used, several switches can be cascaded to form larger networks (see Figure 24). So, to arrive at its destination a packet can travel through several switches.

10.2.7 Header deletion

Header deletion is a simple and effective technique designed to manage the transfer of packets across an arbitrary sized network. Header deletion is illustrated in Figure 26. The first header data character (destination identifier) of a packet is used to specify the router output port address. When a packet is received at a routing switch its first destination identifier is checked to determine the output port to route the packet through. The first destination identifier of the header is then deleted and the packet passes through the switch without this first destination identifier. The second destination identifier of the original header (now the first destination identifier) is used for any subsequent routing.

Figure 27 shows a packet passing through two routing switches. The destination address of the packet comprises three destination identifiers. At the first router the first destination identifier is used to determine the requested output port. This destination identifier is then stripped off. At the second router the second destination identifier is used and stripped off. Finally the packet arrives at the destination node with the third destination identifier at the front of the packet. This can be used to determine where the packet is to go within the destination node (see subclause 10.2.8).

At each stage as the packet passes though the network, it can be regarded as a packet comprising a single destination identifier header, a cargo and an end of packet marker. When, at the first router this single destination identifier header is stripped off, the first data character of the cargo becomes the new destination identifier header. All stages treat the packet in the same way, using the first destination identifier to determine the destination (output port) and then deleting that first destination identifier before forwarding the packet.

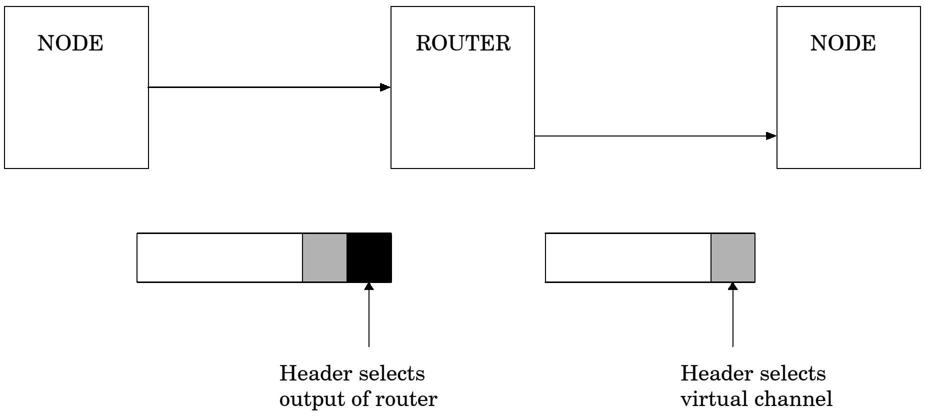

NODE
ROUTER
NODE
Header selects output of router
Header selects virtual channel

Figure 26: Header deletion

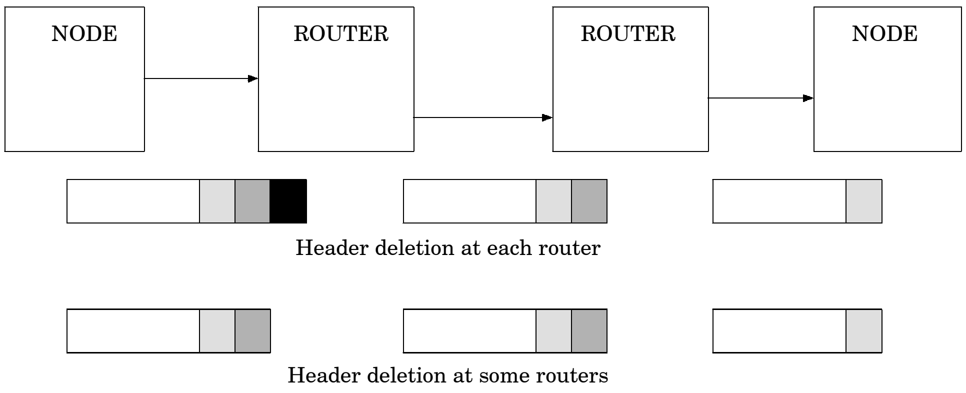

NODE
ROUTER
ROUTER
NODE
Header deletion at each router
Header deletion at some routers

Figure 27: Header deletion across multiple switches

Header deletion can be implemented in all routers in a network or at some selected routing devices only. The latter case cannot be performed appropriately by the router unless information is made available to it beforehand specifying the node addresses for which header deletion is applied (see 10.3.3 for requirements on this subject).

10.2.8 Virtual channels

Many packets from different sources and going to different destinations can be routed through a single data link. Each source-destination pair forms a virtual channel which is mapped on to the physical network comprising links and routing switches.

This concept can be extended within the source and destination nodes. An example is a processing device attached to a network. There can be several tasks running on the processor. These tasks can send or receive information to or from other tasks running on other processors within the network. When a packet arrives at a destination node its header (first data character) can be inspected to see what task it is intended for. The header is stripped off and the packet put in a buffer which can be accessed by the destination task.

10.2.9 Packet addressing

10.2.9.1 Purpose

In this subclause several different packet-addressing schemes are considered and compared.

10.2.9.2 Path addressing

With path addressing the destination address is specified as a sequence of router output port numbers used to guide the packet across the network.

Path addressing is simple and uses comparatively few gates for implementation. Its drawback is that the destination address can become relatively large if several routing switches are traversed. Also the length of the destination address can vary depending on where the destination is located on the network relative to the source. The complexity of packet addressing is handled by the source node and the routing switches are comparatively simple.

10.2.9.3 Logical addressing

In logical addressing each destination has a unique number or logical address associated with it. These numbers can be assigned arbitrarily to nodes provided that no two nodes have the same logical address. When a source node transfers a message to a destination node it simply addresses the packet with the logical address. To support logical addressing each routing switch is provided with a routing table. This tells the router the output port to transfer a packet to, for each possible logical address. Figure 28 illustrates an example of a routing table.

Routing table	
Logical destination	Physical output port
1	8
2	1
3	3
4	1
...	...

Figure 28: Example routing table

In this example, when a packet is received with a logical address of 1 it is routed to output port 8 of the router. A packet with logical address of either 2 or 4 is routed to output port 1 and a packet with logical address 3 is routed to output port 3.

For a reasonable sized network the routing table can become fairly large. Initialization of the routing table can be done in several ways, for example using separate control or configuration links. With logical addressing the complexity of packet addressing is handled by the routing switches rather than by the source node, as is the case with path addressing.

10.2.9.4 Regional logical addressing

Logical addressing can be used in conjunction with header deletion. In this case the routing table also holds information about the headers to delete or to keep, for each logical address. This facilitates multiple level logical addressing schemes. For local addresses a single logical address is used whereas to send a packet to more remote locations a double logical address is used (or more logical addresses as appropriate for the network). In the latter case the first logical address represents the route from one region to the destination region and the second logical address represents the local address within the destination region. When the packet arrives at the destination region the routing switch that transfers the packet to the destination region strips off the first logical address making the local address visible for subsequent local routing.

Regional logical addressing can reduce the size of the routing switches used for logical addressing at the expense of a slightly longer address (two or more data characters), when packets are for transmission to logical addresses in remote regions.

10.2.9.5 Interval labelling

Interval labelling is based on logical addressing. Destinations are grouped together in contiguous intervals, e.g. 1-3, 4-9, 10-32. Each interval is assigned to an output port of the routing switch so that, following the example, destinations 1-3 are all reached via one particular output port. Interval labelling was designed to reduce the size of the routing tables and to speed up the time taken to decode the destination address within a routing switch.

Interval labelling is more complex than logical addressing, but uses smaller routing tables.

10.2.9.6 Group adaptive routing

Group adaptive routing is a means of routing packets to a requested destination over different paths through a network. This is illustrated in Figure 29.

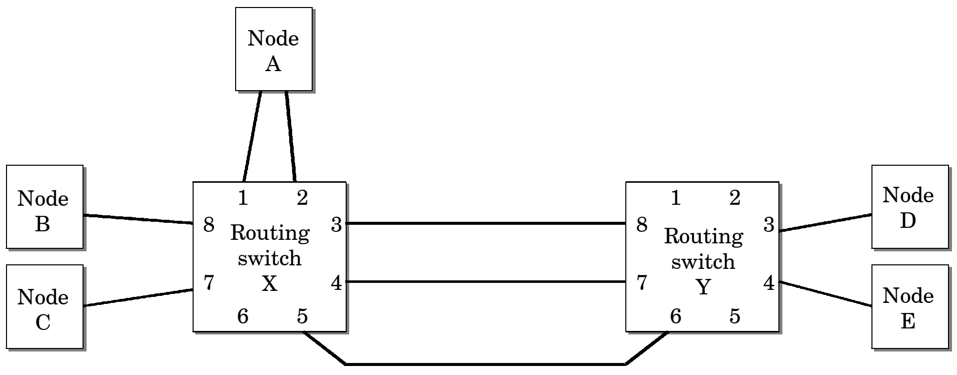


Figure 29: Group adaptive routing

Assume that node B wants to send a packet to node D and that logical addressing is being used. The packet is sent and subsequently arrives at routing switch X. The routing table inside routing switch X indicates output port 3 as output port to route the packet. Output port 3 is currently busy sending another packet, so the packet is stalled and remains waiting until output port 3 becomes free before it can move on.

There are three links that run from routing switch X to routing switch Y. Any of these three links that is free can be used to send the packet. Re-routing a packet out of one of several possible equivalent output ports is known as Group adaptive routing. Links that connect to the same destination (node or routing switch) are called a group. Any link in a group can be used to transfer data to their destination.

Group adaptive routing is an effective means of controlling allocation of link bandwidth, making sure that efficient use is made of the available network resources.

Group adaptive routing can be implemented relying on the configuration registers to hold information about equivalent output ports. When a packet is received it can be routed to any of the equivalent output ports that are currently free, or to whichever port become available first. Assignment of a packet to an output port involves also the consideration of any arbitration scheme that is implemented within the routing switch. In the event of several packets competing for a set of links, subclause 10.3.5 specifies the means of arbitration when an output port becomes available, giving access to the newly freed output port to the packet with the highest priority destination address.

10.3 SpaceWire routing switches (normative)

10.3.1 Purpose

This subclause defines SpaceWire routing switches.

10.3.2 Routing switch

a. A SpaceWire routing switch shall comprise a number of SpaceWire link interfaces (encoder-decoders) and a routing matrix.

NOTE The routing matrix enables the transfer of packets arriving at one link interface to another link interface on the routing switch, and the sending out from this link. Each link interface can be considered as comprising an input port (the link interface receiver) and an output port (the link interface transmitter).

b. A SpaceWire routing switch shall transfer packets from the input port of the switch where the packet arrives, to a particular output port determined by the packet destination address.

NOTE The destination address can comprise several data characters (see subclause 9.2.2).

10.3.3 Routing scheme

a. A routing switch shall use the leading data character of a packet to determine the output port of the routing switch to route the packet.

NOTE The leading data character is the very first data character sent over a link or the first data character following the EOP or EEP that terminated the previous packet.

b. A SpaceWire routing switch shall either implement only path addressing, or a combination of path addressing, logical addressing and regional addressing.

c. A routing table within the routing switch shall hold the logical-physical mapping and shall determine whether header deletion is applied (i.e. when a particular physical output-port represents a gateway between distinct regions).

d. The routing switch addresses shall be assigned as shown in Table 15.

NOTE 1 A leading data character with a value of 0 results in the packet being routed to the routing switch configuration logic.

NOTE 2 A leading data character with a value of 6 results in the packet being routed to output port 6.

NOTE 3 A leading data character with a value of 49 results in the packet being routed to the output port that is referred to in location 49 of the routing table within the routing switch.

Table 15: Routing switch addresses

Address range	Function	Header deletion
0	Internal configuration port	YES
1-31 (01-1F hex)	Physical output ports	YES
32–254 (20-FE hex)	Logical addresses, which are mapped on to the physical output ports.	Optional on each output port. Header deleted if the physical output port is a gateway between distinct regions. Header can also be deleted on final link to a node.
255 (FF hex)	Reserved logical address, which is mapped on to physical output port. Treated in the same way as any other logical address (refer to 10.3.3 o.), but is reserved for future use (refer to 10.3.3 n.).	Optional on each output port. Header deleted if the physical output port is a gateway between distinct regions. Header can also be deleted on final link to a node.

e. A NULL routing table entry shall mean that the routing table entry is undefined and if referenced shall lead to an invalid address error (see subclause 10.6.4).

f. Path addressing shall be limited to 32 ports maximum (including the configuration port).

NOTE Routing switches with less than this maximum number may be used.

g. Accessing an output port that does not exist shall result in an invalid address error (see subclause 10.6.4).

h. Logical addressing shall be limited to 224 logical addresses.

i. Regional addressing shall be used for larger networks with each cluster limited to a maximum of 224 logical addresses.

NOTE Regions using logical addressing can be connected to regions using path addressing.

j. Configuration ports shall be accessed using path addressing only.

NOTE A routing table cannot address the internal configuration port (address 0).

k. Header deletion shall always be applied to path addresses.

l. If header deletion applies then the leading data character (destination identifier) of each packet shall be deleted.

m. One and only one data character (destination identifier) shall be deleted by each routing switch with header deletion enabled, that is traversed by the packet.

   NOTE For very large switches the two leading destination identifiers can be used by a single routing switch to determine the destination address This enables switches with up to 961 (31 × 31) physical links and with over 50,000 (224 × 224) logical addresses.

n. Logical address 255 is reserved for future use and should not be used.

o. Logical address 255 shall be treated in the same way as any other logical address.

10.3.4 Wormhole routing

The implementation of wormhole routing within SpaceWire routing switches is as follows:

a. When a packet arrives at a routing switch the assigned output port shall be determined.

b. If the output port is free, i.e. not currently transmitting a packet, then the output port shall be allocated to transmit the newly arrived packet.

c. As each character of the packet arrives at the input port it shall be transferred immediately to the output port for transmission.

d. The output port shall not transmit any other packet until the packet that it is currently transmitting is sent or terminated following an error.

e. If the input port is waiting for packet characters to arrive then the output port shall also wait.

f. If the output port is waiting to transmit packet characters then the input port shall also wait.

g. If the assigned output port is busy then the newly arrived packet shall wait at the input port until the assigned output port is free to transmit the new packet.

h. When the output port finishes transmission of its current packet then it shall be available to accept a packet from another input port.

10.3.5 Arbitration

a. Two or more input ports can all be waiting to send data out of the same output port. SpaceWire routing switches shall provide a means of arbitrating between input ports requesting the same output port.

   NOTE Priority based, round-robin or random arbitration schemes can be implemented.

b. When the assigned output port becomes free the input port selected through arbitration shall transfer one packet to the output port.

c. An output port that has several input ports waiting to use it shall perform arbitration after the current packet is transmitted.

   NOTE Subclause 10.6.4 explains what happens if the destination address is invalid, i.e. not recognized by the routing switch.

10.3.6 Group adaptive routing

a. SpaceWire routing switches shall implement group adaptive routing for logical addresses and regional addresses.

NOTE SpaceWire routing switches can implement group adaptive routing for path addresses.

b. Group adaptive routing shall follow any arbitration scheme implemented within a routing switch.

c. The arbitration shall apply to all output ports within a group.

10.3.7 Packet distribution, broadcast and multicast

10.3.7.1 General

SpaceWire routing switches can implement packet distribution where data arriving at an input port is sent out of several output ports simultaneously.

Packet distribution is a restricted form of broadcast or multicast. The restriction is that a routing switch only distributes packets to output ports connected to nodes. This restriction is applied through use of appropriate network architectures and corresponding configuration of the routing switches in the network (see subclause 10.5).

General broadcast or multicast implemented by a routing switch can give rise to system level problems when broadcast or multicast packets cycle round a circularly connected network indefinitely, resulting in a blocked network.

Broadcast and multicast can be implemented at a higher level by sending a packet to all (broadcast) or several (multicast) nodes on a network, one after the other.

10.3.7.2 Handling

a. Configuration registers within routing switches that implement packet distribution shall be used to specify the output ports to send packets.

b. The reset state of routing switches that implement packet distribution shall disable all packet distribution.

10.4 SpaceWire nodes (normative)

a. A SpaceWire node shall comprise one or more SpaceWire link interfaces (encoder-decoders) and an interface to the host system.

NOTE A SpaceWire node represents an interface between a SpaceWire network and an application system using the network services.

b. A SpaceWire node shall accept a stream of packets from the host system for transmission or provide a stream of packets to the host system after reception from the SpaceWire link, or do both.

10.5 SpaceWire network (normative)

a. A SpaceWire network shall comprise two or more SpaceWire nodes and zero or more SpaceWire routing switches.

b. SpaceWire nodes and SpaceWire routing switches shall be interconnected with SpaceWire links.

c. Packets shall be transferred from one SpaceWire node to another across SpaceWire links and through SpaceWire routing switches.

d. When packet distribution is used (see subclause 10.3.7) only output ports connected to nodes shall be used to forward the packets.

10.6 Network level error recovery (normative)

10.6.1 Types of errors at packet level

The following types of error can occur at the packet level:

- Link error, i.e. error detected at exchange level (see subclause 8.9.5),
- EEP received,
- Invalid destination address.

10.6.2 Link error recovery

When a link error is detected within a link interface the following actions shall be taken at the network level to recover from the error.

a. If the error is detected at a source or destination node then the error shall be flagged up to the application level.

NOTE 1 If the error is detected within a routing switch, then the error may be flagged to a pin on the routing switch device or to an internal status register within the routing switch.

NOTE 2 Flagging to an external pin or internal status register can be useful for system debugging and monitoring.

b. The current packet being transmitted shall stop being transmitted, i.e. the part of the packet that is not yet transmitted shall not be transmitted, and therefore N-Chars up to and including the next EOP or EEP, shall be discarded by the link interface without being transmitted.

NOTE This is known as spilling the packet.

c. The current packet being received shall stop being received and the part of the packet that has been received already shall be terminated by an EEP.

NOTE 1 If a complete packet has just been received when the error occurs, i.e. last character received before error was an EOP or EEP, then the link interface needs not add an EEP to the receive buffer, but one may be added.

NOTE 2 If the receiver buffer (e.g. FIFO) is full the EEP can be unable to be written into the receive buffer. If there is not room for at least 9 N-Chars (the EEP and 8 other N-Chars) then the link interface is not able to send an FCT. In either of these two cases the link cannot restart after an error until some N-Chars are read out of the receive buffer (to make room for at least 9 N-Chars). This is a better solution than automatically starting after an error when the receive buffer remains full, because the link sends a few N-Chars and then become blocked again. If this happens, the link repeatedly blocks part of the network.

d. Packets terminated by an EEP can flow through routing switches within a network (see subclause 9.2.3). When an EEP is received at a destination node the application level shall be informed of the occurrence of the error indicated by the EEP.

NOTE 1 The application level at a source node can re-send the packet which was being sent when the link error was reported. The application level at a destination node may discard a packet terminated by an EEP, or may decide to use it.

NOTE 2 Because the remainder of a packet is discarded after an error, very large packets can cause the link to halt for a long period of time while the packet is spilt. To bear this in mind is of particular importance when determining the size of packets to use in a particular application.

10.6.3 EEP received

10.6.3.1 General

An EEP at the end of a packet means that an error occurred within the transmitted packet and that the packet was consequently terminated prematurely. The data in the packet can be valid but the complete packet has not been transferred successfully.

If an EEP is received the action taken depends on whether the link interface is within a source-destination node or within a network routing switch, as described in the following subclauses.

10.6.3.2 Routing switch

An EEP received by a routing switch shall be transferred through the routing switch in the same way as an EOP, i.e. EEP shall be treated in the same way as an EOP within routing switches.

10.6.3.3 Node

An EEP received by a link interface within a destination node shall be reported to the application level.

10.6.4 Invalid destination address

10.6.4.1 General

A logical address whose routing table entry refers to a non-existent output port shall be regarded as an invalid address.

NOTE A destination address can be unrecognisable by a routing switch or node. For example, when a path address of 9 is specified in a routing switch with only eight output ports. A NULL logical address in a routing switch means that the location in the routing table is not configured, so is invalid.

10.6.4.2 Routing Switch

If a packet arriving at a routing switch has an invalid destination address then that packet shall be discarded (spilt) and the N-Chars arriving at the input port where the invalid destination address was detected shall be discarded until and including the next EOP or EEP.

NOTE The invalid destination address error may be flagged either to external status pins on the routing switch device or to a status register within the routing switch.

10.6.4.3 Node

If a packet arrives at a node with an unexpected destination address then that packet shall be discarded.

NOTE 1 Whether a particular address is expected or not within a node depends upon the host system and its use of virtual channels.

NOTE 2 The unexpected destination address may be flagged to the host system.

10.6.4.4 Double EOP or EEP

An EOP or EEP received immediately after an EOP or EEP represents an empty packet which does not have a destination address. A routing switch receiving an empty packet shall discard it by deleting the second EOP or EEP.

10.7 Example networks

10.7.1 General

Several examples are provided in this subclause to clarify the operation of packet routing and header deletion.

A SpaceWire network is illustrated in Figure 30. The SpaceWire nodes are numbered N1 to N9 and each has just one SpaceWire link interface. The SpaceWire routing switches are numbered R1 to R4 and each has four link interfaces.

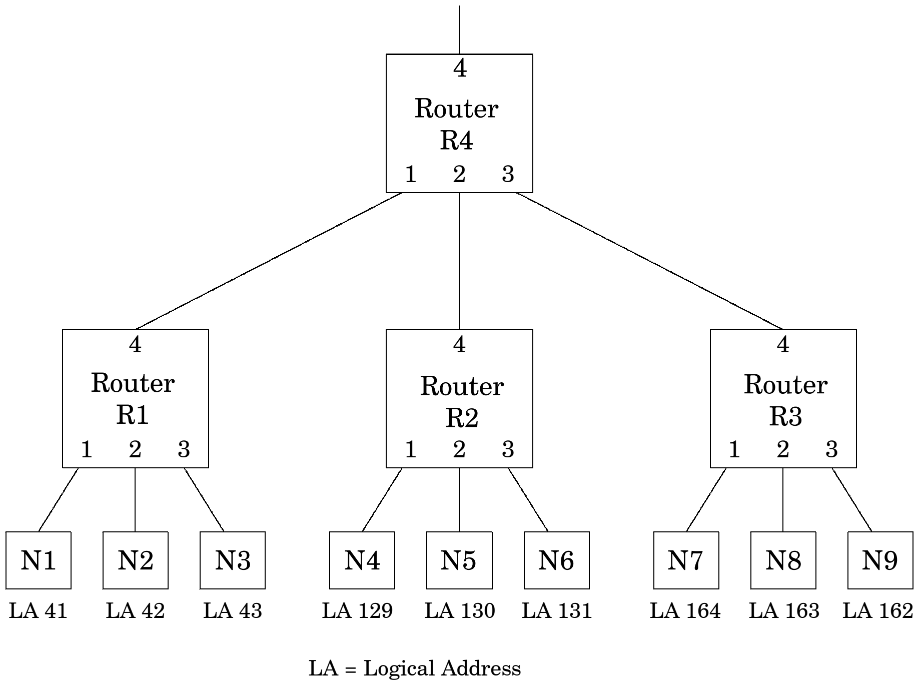


Figure 30: Example SpaceWire network

10.7.2 Path addressing

To transfer a cargo from node N1 to node N3 using path addressing the following packet is sent:

<3><cargo><EOP>

To transfer a cargo from node N1 to node N8 using path addressing the following packet is sent:

<4><3><2><cargo><EOP>

10.7.3 Logical addressing

To transfer a cargo from node N1 to node N3 using logical addressing the following packet is sent:

<43><cargo><EOP>

To transfer a cargo from node N1 to node N8 using logical addressing the following packet is sent:

<163><cargo><EOP>

The routing table contents are shown for each routing switch in Figure 31.

Routing table switch 1	
...	
41	1
42	2
43	3
...	
129	4
130	4
131	4
...	
162	4
163	4
164	4

Routing table switch 2	
...	
41	4
42	4
43	4
...	
129	1
130	2
131	3
...	
162	4
163	4
164	4

Routing table switch 3	
...	
41	4
42	4
43	4
...	
129	4
130	4
131	4
...	
162	3
163	2
164	1

Routing table switch 4	
...	
41	1
42	1
43	1
...	
129	2
130	2
131	2
...	
162	3
163	3
164	3

Figure 31: Example SpaceWire network routing table contents

10.7.4 Regional addressing

In this subclause the SpaceWire network shown in Figure 30 is altered slightly to illustrate regional logical addressing i.e. logical addressing with header deletion on output ports that drive links between two regions. The altered SpaceWire network is shown in Figure 32. The network is split into two regions. The nodes in region 2 are renumbered to support the example, but the local logical address of each node is the same as before. The structure of the network is identical to that in Figure 30. A single data character is used to define the logical address within a region. Up to 224 nodes can be addressed within a single region. Some logical addresses within a region are used as the bridge (or route) to other regions. An unlimited number of regions can be supported.

To transfer a cargo from node N1 to node N3 (both in region 1) the following packet is sent:

<43><cargo><EOP>

To transfer a cargo from node N1 to node N5 (both in region 1) the following packet is sent:

<130><cargo><EOP>

To transfer a cargo from node N1 (region 1) to node N3 (region 2) the following packet is sent:

<109><162><cargo><EOP>

<109> is the logical address given to the link between region 1 and region 2 (R4 port 3 to R3 port 4). R4 deletes the header character on any packet leaving via port 3 or port 4. R3 deletes the header character on any packet leaving via port 4.

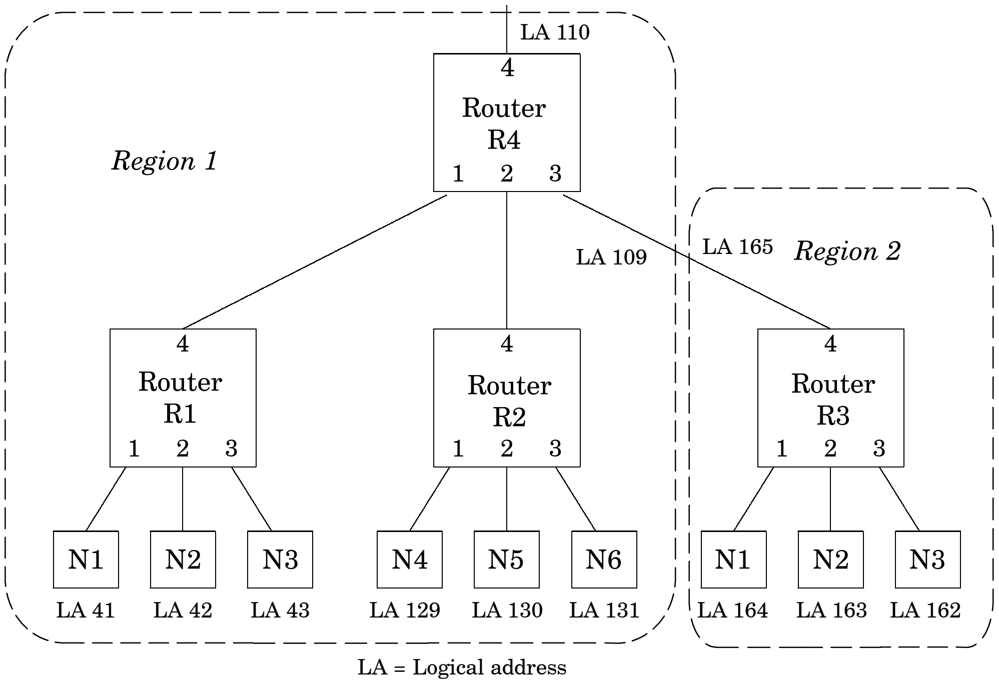


Figure 32: Example SpaceWire network with local logical address regions

(This page is intentionally left blank)

11

Error recovery scheme

11.1 Purpose

The error recovery scheme is split across the exchange level and network level. This clause aims to describe the error recovery scheme as a whole, to aid comprehension.

11.2 Exchange level errors

The following errors (see subclause 8.9) can be reported at the exchange level:

- Disconnect error,
- Parity error,
- Escape sequence error,
- Character sequence error,
- Credit error.

The response to any of these errors is as follows:

a. Detect error,

b. Disconnect link,

c. Report error to network level,

d. Attempt to reconnect link if link interface still enabled.

11.3 Network level errors

11.3.1 General

- The following errors (see subclause 10.6) can be reported at the network level:
- Link error (exchange level error),
- EEP received,
- Invalid destination address.

11.3.2 Link error

If a link error, i.e. error detected in exchange level, is reported to the network level then the network level responds as follows:

a. Error reported to network level.

b. Terminate current receive packet with EEP.

c. Spill current transmit packet until and including next EOP (or EEP).

d. If the error occurs in a link interface within a source or destination node then flag the error to the application layer.

e. If the error occurs in a link interface within a routing switch, then the error may be flagged to external status pins on the routing switch device or to a status register within the device.

This sequence of events is illustrated and described in more detail in subclause 11.4.

11.3.3 EEP received

If an EEP is received the action taken depends on whether the link interface is within a destination node or within a network routing switch. In a destination node, the reception of an EEP is flagged to the application level. The packet terminated by the EEP is otherwise transferred as normal to the application level. In a routing switch no special action is taken when an EEP is received. The EEP is treated in exactly the same way as an EOP.

11.3.4 Invalid destination address

A packet that arrives at a routing switch with an invalid address, i.e. an address that is not recognized by the routing switch, is discarded.

11.4 Link error recovery

If any form of error is detected within the link interface then the following sequence of events occurs to recover from the error (see Figure 33):

a. Detect error (disconnect, parity, escape sequence, character sequence, credit).

b. Disconnect link.

c. If previous character was NOT EOP then add EEP (error end of packet) to the receiver buffer (i.e. the receive FIFO in Figure 33).

d. Delete data in the transmitter buffer (i.e. transmit FIFO in Figure 33) until the next EOP (End of Packet).

e. Reconnect.

f. Send next packet.

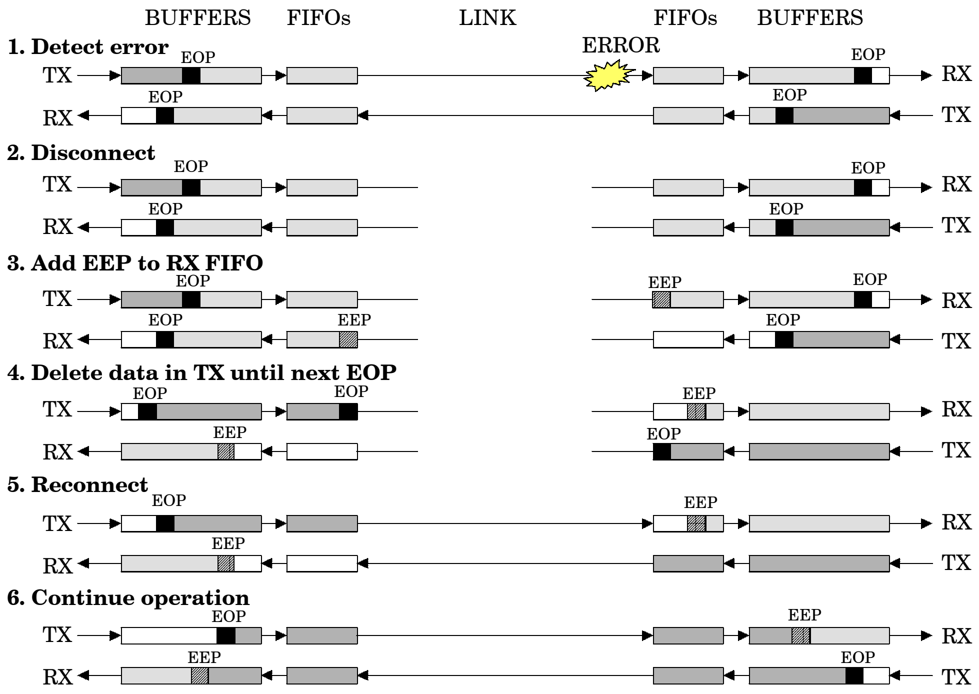


Figure 33: Link interface error detection and recovery operation

Figure 33 shows transmit and receive buffers and FIFOs. The reason for this is to illustrate a typical host system where buffering is used to hold outgoing and incoming data and FIFOs are used for data rate regulation. In this Standard both buffers and FIFOs are regarded as part of the host system. This is so that this Standard can be specified in a more general way than if the FIFOs were included within the Standard (see subclause 8.4.8). It also simplifies the definition of the error recovery scheme with the link interface error recovery being defined by the exchange level and the FIFO and buffer management being defined by the network level.

The decision about what to do with the packet that terminates with the EEP is left up to the higher level application layer. For example if the packet is a line from an image the part of the packet that was received correctly can be used, or alternatively the packet can be thrown away and the process can carry on. If the packet was part of some important control information (e.g. program code) it is important that the packet is resent.

The above protocol works across networks comprised of a number of routing switches. Only the link on which the error occurs is reset (disconnect or reconnect). All the other links continue operation. Normally only the packet in which the error occurred is partly lost and all other packets remain valid. In some cases more than one packet can be lost due to the time taken for link disconnection.

If the header byte (i.e. first byte after an EOP or EEP) is corrupted then the entire packet is lost and the data is not propagated across a network. The routing switch simply disposes of the packet. This can cause a problem at the destination of the packet because the fact that a packet is missing is not reported. The application level ensures then that the proper sequence of packets is received. In the event of a receiver at a destination node detecting an error in a header byte it can report this fact to the application level.

If the error occurs in a EOP (or EEP) then two packets are affected – the one before the EOP where all the data are sent but no EOP is received, and the following one because the link transmitter “spills” the packet until the next EOP (or EEP).

If the error occurs in a NULL or FCT inserted in the data stream for a packet then the packet being sent is discarded from that point on. This is because it is not known what the character was before it was corrupted.

11.5 Application level error handling

11.5.1 General

The application interface is not defined in this Standard. However, a typical application interface comprises the following services:

- Open link – Starts a link interface and attempts to establish a connection with the link interface at the other end of the link.
- Close link – Stops a link and breaks the connection.
- Write packet – Sends a packet out of the link interface.
- Read packet – Reads a packet from the link interface.
- Status and Configuration – Reads the current status of the link interface and sets the link configuration.

Several error checks can be implemented at the application level. These are described below.

11.5.2 Link initialization timeout error

When an application attempts to open a link, it can set a timeout period for link connection. If the link connection has not been established within the specified timeout period then the link can be considered unusable and an alternative link used for the requested communication.

11.5.3 Packet transmit timeout error

When an application tries to write a packet to a link interface, it can set a timeout period for transmission of the packet. If the complete packet has not been transmitted when the timeout period expires then the transmitter is assumed blocked. The link interface is then disabled to cause a disconnect error and reset of the link, and then enabled again to allow the link to start and reconnect.

11.5.4 Packet receive timeout error

When an application tries to read a packet from a link interface, it can set a timeout period for reception of the packet. If the complete packet has not been received when the timeout period expires then the receiver is assumed blocked. The link interface is then disabled to cause a disconnect error and reset of the link, and then enabled again to allow the link to start and reconnect.

12 Conformance criteria (normative)

12.1 Conformance statements

Several SpaceWire compatible subsets (products) can be identified each of which implements only a part of this Standard:

- SpaceWire cable, i.e. unterminated cable.
- SpaceWire connector, i.e. connectors for cable termination and PCB mounting.
- SpaceWire cable assembly, i.e. cable terminated with connectors at each end.
- SpaceWire interface, i.e. complete interface for connection of SpaceWire to some host system. The interface includes the encoder or decode device, the LVDS drivers and receivers and the SpaceWire connectors.
- SpaceWire encoder-decoder, i.e. encoder-decoder device with CMOS level DS signals and with the LVDS drivers and receivers implemented by external sources.
- SpaceWire LVDS encoder-decoder, i.e. encoder-decoder device with internal LVDS drivers and receivers.
- SpaceWire routing switch, i.e. routing switch device with CMOS level DS signals and with the LVDS drivers and receivers implemented by external sources.
- SpaceWire LVDS routing switch, i.e. routing switch device with internal LVDS drivers and receivers.
- SpaceWire routing switch unit, i.e. routing switch unit including the LVDS drivers and receivers and the SpaceWire connectors.
- SpaceWire network, i.e. system comprising SpaceWire links, nodes and routing switches.

Corresponding subsets of this Standard are defined to which implementations may claim conformance. The form of the conformance statement to use is the one given by the appropriate subset definition in the following subclauses.

12.2 Definition of subsets

12.2.1 SpaceWire cable

An implementation of SpaceWire cable shall conform to all of the requirements given in all subclauses listed in Table 16.

NOTE A cable meeting this specification may use the following conformance statement:

"This cable conforms to the SpaceWire cable specification ECSS-E-50-12A."

Table 16: SpaceWire cable conformance

Relevant clause or subclause	Title
5.2	Cables

12.2.2 SpaceWire connector

An implementation of a SpaceWire connector shall conform to all of the requirements given in all subclauses listed in Table 17.

NOTE A connector meeting this specification may use the following conformance statement:

"This connector conforms to the SpaceWire connector specification ECSS-E-50-12A."

Table 17: SpaceWire connector conformance

Relevant clause or subclause	Title
5.3	Connectors

12.2.3 SpaceWire cable assembly

An implementation of a SpaceWire cable assembly shall conform to all of the requirements given in all subclauses listed in Table 18.

NOTE A cable assembly meeting this specification may use the following conformance statement:

"This cable assembly conforms to the SpaceWire cable assembly specification ECSS-E-50-12A."

Table 18: SpaceWire cable assembly conformance

Relevant clause or subclause	Title
5.2	Cables
5.3	Connectors
5.4	Cable assembly

12.2.4 SpaceWire interface

a. An implementation of a SpaceWire interface shall conform to all of the requirements given in all subclauses listed in Table 19.

NOTE A product fitted with an interface meeting this specification may use the following conformance statement:

"This product conforms to the SpaceWire interface specification ECSS-E-50-12A."

Table 19: SpaceWire interface conformance

Relevant clause or subclause	Title
5.3	Connectors
5.5	PCB tracks
6	Signal level
7	Character level
8	Exchange level
9	Packet level
10.4	SpaceWire nodes
10.6	Network level errors

b. Together with the above conformance statement the following parameters shall be specified for the interface:

1. Total transmitter data-strobe skew (worst case over process, temperature, voltage range and irradiation) measured at the interface connector (Dout to Sout skew).
2. Total transmitter data jitter (worst case) measured at the interface connector (Dout jitter).
3. Total transmitter strobe jitter (worst case) measured at the interface connector (Sout jitter).
4. Total receiver minimum separation between Data and Strobe (worst case) measured at the interface connector (Din to Sin minimum separation). That includes all D-S skew, D jitter and S jitter between the interface connector and the decoder device.

NOTE A detailed explanation of the above parameters is provided in clause 6.

c. If typical figures for the above parameters are also provided, the conditions applying for the figure measurements shall be clearly stated (e.g. temperature, operating voltage).

NOTE If a SpaceWire encoder-decoder supports system time distribution as defined in subclause 8.12 then the following conformance statement may be used:

"This product supports SpaceWire system time distribution according to ECSS-E-50-12A."

12.2.5 SpaceWire encoder-decoder

a. An implementation of a SpaceWire encoder-decoder shall conform to all of the requirements given in all subclauses listed in Table 20.

NOTE A product fitted with an interface meeting this specification may use the following conformance statement:

"This product conforms to the SpaceWire encoder-decoder specification ECSS-E-50-12A."

Table 20: SpaceWire encoder-decoder conformance

Relevant clause or subclause	Title
6.3	Signal coding
6.5	SpaceWire link
6.6	Data signalling rate
7	Character level
8	Exchange level
9	Packet level

b. Together with the above conformance statement the following parameters shall be specified for the interface:

1. Encoder data-strobe skew (worst case over process, temperature, voltage range and irradiation) measured at the output of the encoder device.
2. Encoder data jitter (worst case) measured at the output of the encoder device.
3. Encoder strobe jitter (worst case) measured at the output of the encoder device.
4. Decoder minimum separation between Data and Strobe (worst case) measured at the input of the decoder device.

NOTE A detailed explanation of the above parameters is provided in clause 6.

c. If figures for the above parameters are also provided, the conditions applying for the figure measurements shall be clearly stated (e.g. temperature, operating voltage).

NOTE If a SpaceWire encoder-decoder supports system time distribution as defined in subclause 8.12 then the following conformance statement may be used:

"This product supports SpaceWire system time distribution according to ECSS-E-50-12A."

12.2.6 SpaceWire LVDS encoder-decoder

a. An implementation of a SpaceWire encoder-decoder which include the LVDS drivers and receivers shall conform to all of the requirements given in all subclauses listed in Table 21.

NOTE A product fitted with an interface meeting this specification may use the following conformance statement:

"This product conforms to the SpaceWire LVDS encoder-decoder specification ECSS-E-50-12A."

Table 21: SpaceWire LVDS encoder-decoder conformance

Relevant clause or subclause	Title
6	Signal level
7	Character level
8	Exchange level
9	Packet level

b. Together with the above conformance statement the following parameters shall be specified for the interface:

1. Encoder data-strobe skew (worst case over process, temperature, voltage range and irradiation) measured at the output of the encoder device.
2. Encoder data jitter (worst case) measured at the output of the encoder device.
3. Encoder strobe jitter (worst case) measured at the output of the encoder device.
4. Decoder minimum separation between Data and Strobe (worst case) measured at the input of the decoder device.

NOTE A detailed explanation of the above parameters is provided in clause 6.

c. If figures for the above parameters are also be provided, the conditions applying for the figure measurements shall be clearly stated (e.g. temperature, operating voltage).

NOTE If a SpaceWire LVDS encoder-decoder supports system time distribution as defined in subclause 8.12 then the following conformance statement may be used:

"This product supports SpaceWire system time distribution according to ECSS-E-50-12A."

12.2.7 SpaceWire routing switch

a. An implementation of a SpaceWire routing switch shall conform to all of the requirements given in all subclauses listed in Table 22.

NOTE A routing switch meeting this specification may use the following conformance statement:

"This product conforms to the SpaceWire routing switch specification ECSS-E-50-12A."

Table 22: SpaceWire routing switch conformance

Relevant clause or subclause	Title
6.3	Signal coding
6.5	SpaceWire link
6.6	Data signalling rate
7	Character level
8	Exchange level
9	Packet level
10.3	SpaceWire routing switches
10.6	Network level errors

b. Together with the above conformance statement the following parameters shall be specified for the routing switch:

1. SpaceWire link encoder data-strobe skew (worst case over process, temperature, voltage range and irradiation) measured at an output of the routing switch device.
2. SpaceWire link encoder data jitter (worst case) measured at an output of the routing switch device.

3. SpaceWire link encoder strobe jitter (worst case) measured at an output of the routing switch device.
4. SpaceWire link decoder minimum separation between Data and Strobe (worst case) measured at an input of the routing switch device.

   NOTE A detailed explanation of the above parameters is provided in clause 6.

c. If figures for the above parameters are also be provided, the conditions applying for the figure measurements shall be clearly stated (e.g. temperature, operating voltage).

   NOTE If a SpaceWire routing switch supports system time distribution as defined in subclause 8.12 then the following conformance statement may be used:

   "This product supports SpaceWire system time distribution according to ECSS-E-50-12A."

12.2.8 SpaceWire LVDS routing switch

a. An implementation of a SpaceWire routing switch which includes the LVDS drivers and receivers shall conform to all of the requirements given in all subclauses listed in Table 23.

   NOTE A routing switch meeting this specification may use the following conformance statement:

   "This product conforms to the SpaceWire LVDS routing switch specification ECSS-E-50-12A."

Table 23: SpaceWire LVDS routing switch conformance

Relevant clause or subclause	Title
6	Signal level
7	Character level
8	Exchange level
9	Packet level
10.3	SpaceWire routing switches
10.6	Network level errors

b. Together with the above conformance statement the following parameters shall be specified for the routing switch.

1. SpaceWire link encoder data-strobe skew (worst case over process, temperature, voltage range and irradiation) measured at an output of the routing switch device.
2. SpaceWire link encoder data jitter (worst case) measured at an output of the routing switch device.
3. SpaceWire link encoder strobe jitter (worst case) measured at an output of the routing switch device.
4. SpaceWire link decoder minimum separation between Data and Strobe (worst case) measured at an input of the routing switch device.

   NOTE A detailed explanation of the above parameters is provided in clause 6.

c. If figures for the above parameters are also be provided, the conditions applying for the figure measurements shall be clearly stated (e.g. temperature, operating voltage).

NOTE If a SpaceWire LVDS routing switch supports system time distribution as defined in subclause 8.12 then the following conformance statement may be used:

"This product supports SpaceWire system time distribution according to ECSS-E-50-12A."

12.2.9 SpaceWire routing switch unit

a. An implementation of a SpaceWire routing switch unit shall conform to all of the requirements given in all subclauses listed in Table 24.

NOTE A routing switch unit meeting this specification may use the following conformance statement:

"This product conforms to the SpaceWire routing switch unit specification ECSS-E-50-12A."

Table 24: SpaceWire LVDS routing switch unit conformance

Relevant clauses or subclauses	Title
5.3	Connectors
5.5	PCB tracks
6	Signal level
7	Character level
8	Exchange level
9	Packet level
10.3	SpaceWire routing switches
10.6	Network level errors

b. Together with the above conformance statement the following parameters shall be specified for the routing switch:

1. SpaceWire link encoder data-strobe skew (worst case over process, temperature, voltage range and irradiation) measured at an output of the routing switch device.
2. SpaceWire link encoder data jitter (worst case) measured at an output of the routing switch device.
3. SpaceWire link encoder strobe jitter (worst case) measured at an output of the routing switch device.
4. SpaceWire link decoder minimum separation between Data and Strobe (worst case) measured at an input of the routing switch device.

NOTE A detailed explanation of the above parameters is provided in clause 6.

c. If figures for the above parameters are also be provided, the conditions applying for the figure measurements shall be clearly stated (e.g. temperature, operating voltage).

NOTE If a SpaceWire routing switch unit supports system time distribution as defined in subclause 8.12 then the following conformance statement may be used:

"This product supports SpaceWire system time distribution according to ECSS-E-50-12A."

12.2.10 SpaceWire network

An implementation of a SpaceWire network shall conform to all of the requirements given in all subclauses listed in Table 25.

NOTE 1 A network meeting this specification may use the following conformance statement:

"This network conforms to the SpaceWire network specification ECSS-E-50-12A."

NOTE 2 If a SpaceWire network supports system time distribution as defined in subclause 8.12 then the following conformance statement may be used:

"This network supports SpaceWire system time distribution according to ECSS-E-50-12A."

Table 25: SpaceWire network conformance

Relevant clause	**Title**
5	Physical level
6	Signal level
7	Character level
8	Exchange level
9	Packet level
10	Network level

Annex A (informative)

Differences between SpaceWire and IEEE Standard 1355-1995

A.1 General

There are several differences between SpaceWire and IEEE Standard 1355-1995 [1]. Improvements are made in the present Standard to improve ruggedness, power consumption, EMC performance, and to eliminate problems and ambiguities that exist with IEEE Standard 1355-1995. IEEE Standard 1355-1995 contains several sub-standards, the comparison here is with the DS part of IEEE Standard 1355-1995.

The differences between the two standards and the reasons for them are detailed in the following subclauses, looking at each level of this Standard in turn.

A.2 Physical level

IEEE Standard 1355-1995	**ECSS-E-50-12 SpaceWire Standard**
Cable is not suitable for space applications.	Cable is designed to be suitable for space applications.
Connector is not rugged enough for space use.	Connectors are 9-way micro-miniature D-type connectors which are used in space applications.

A.3 Signal level

IEEE Standard 1355-1995	**ECSS-E-50-12 SpaceWire Standard**
PECL does not support failsafe operation.	LVDS adopted for SpaceWire provides improved electromagnetic emission characteristics compared to the PECL signals used in IEEE Standard 1355-1995. LVDS supports failsafe operation. LVDS can be implemented in a range of semiconductor technologies. This enables integration of completed SpaceWire interfaces with other system functions.
There are no requirements for tolerance of simultaneous transitions on Data and Strobe signals: this can lead to faults occurring within the interface.	The DS encoding used is identical to IEEE Standard 1355-1995 with the exception that SpaceWire interfaces are specified to be tolerant of simultaneous transitions on Data and Strobe signals.
Only considers skew.	The timing specification is tightened up compared to that in IEEE Standard 1355-1995. Considers both jitter and skew.

A.4 Character level

IEEE Standard 1355-1995	**ECSS-E-50-12 SpaceWire Standard**
	The character level protocol for SpaceWire is in complete agreement with that in IEEE Standard 1355-1995.
It does not define any other use of the ESC sequence besides NULL. It leaves the use of ESC vague. See also minor differences in A.9.	it uses ESC in the NULL control code (i.e. ESC or FCT), in the Time-Code and in the reserved ESC or N-Char codes. It is specified not to use ESC in any other way. See also minor differences in A.9.

A.5 Exchange level

IEEE Standard 1355-1995	ECSS-E-50-12 SpaceWire Standard
Subclause 5.7.2 states "Thereafter it shall send only NULLs unless and until at least one character has been received by the corresponding link input since reset. After the link output has been started and at least one character has been received by the corresponding link input since reset, the link shall begin normal operation." The state diagram in Figure 5-11 and timing sequence diagram in Figure 5-12 show a different situation. Instead of one character being received it is specifically a NULL that is expected.	This ambiguity is resolved in subclause 8.5 – a NULL is expected.
Subclause 5.7.4.2 states "If a link interface detects a disconnect error before it has started, it shall start, transmit at least one character, and then halt, to ensure that a disconnection error is also detected by the other end 0." The state diagram in Figure 5-11 shows a different reaction to a disconnect error before the link has started. The link is simply halted, it is NOT started and a character is NOT sent.	This ambiguity is resolved in this Standard by modification of the state machine (see subclause 8.5). If a disconnect error is detected before the link has started then the link resets immediately – it does not send a character first. This implies modification to the state machine to work reliably.
Subclause 5.7.2 states "After reset, a DS-SE link output shall maintain both signals at their reset level until started, i.e., instructed to begin operation (note that receipt of a character by the corresponding link input can be taken as such an instruction)." The state diagram in Figure 5-11 specifies the *LinkStart* command even when a character (specifically a NULL) has been received. When a NULL is received in the *Ready* state the link moves to the *NULLReceived* state. It does not move on to the *Run* state until it receives the *LinkStart* command.	This Standard specifies the reception of the *LinkEnable* (equivalent to *LinkStart*) command before a link can move to the *Run* state (subclause 8.6).
The state machine illustrated in Figure 5-11 hangs up if the *ResetLinkCommand* is given to both ends of the link while they are both in the *Ready* state. In the *Ready* state no characters have been sent so the disconnect timeout has not started (disconnect timeout is started only after a bit is received – subclause 5.7.2). When the *ResetLinkCommand* is given the state machine moves to the *WaitInStop* state where it waits for a disconnect error – that cannot occur. Further application of the *ResetLinkCommand* does not resolve this problem, the state machine remains firmly stuck in the *WaitInStop* state until a *PowerOnReset* is applied.	This Standard has a modified state machine in the exchange level that resolves this problem. The *WaitInStop* state is removed completely and any Reset command causes a transition to the *ErrorReset* state.

IEEE Standard 1355-1995	ECSS-E-50-12 SpaceWire Standard
A similar problem as above can occur if one end of the link (end A) starts (in *Started* state) and the other end (end B) is in the *NULLsReceived* state, but is not yet commanded to start. If end A receives a *ResetLinkCommand* it moves to the *WaitInStop* state. In the *WaitInStop* state the outputs are halted so end A stops transmitting NULLs and this is detected as a disconnect error at end B. End B then moves through the *ErrorReset* and *ErrorWait* states ending up in the *Ready* state. Meanwhile end A remains in the *WaitInStop* state unable to detect the disconnect error because it has not been sent a character.	The modified state machine specified in this Standard resolves this problem.
Allows the reception of an FCT before a NULL is received.	Specifies the reception of a NULL before an FCT is received.
	It has been extended to include support for system time distribution using the Time-Code.

A.6 Packet level

IEEE Standard 1355-1995	ECSS-E-50-12 SpaceWire Standard
Subclause 9.2.1 is unclear in the case of a destination being null. It is not clear whether this means a destination address of zero, which is a valid destination address, or whether it means a non-existent destination, i.e. the destination list is empty (contains zero destination identifiers).	The latter case for IEEE Standard 1355-1995 is the one specified in this Standard.
Subclause 9.2.1 is also ambiguous in its definition of a dest_id. It is defined as a fixed size field, its size being known to the (sub)network. It is not clear whether a network comprising several sub-networks can have different size dest_ids. E.g. the first sub-network encountered by a packet can expect a dest_id of 2 bytes and the next sub network encountered can expect a dest_id of 1 byte.	In this Standard it is specified that a destination list contains dest_ids of one data-character each. Each routing switch knows how many dest_ids to strip off. The packet source knows the destination address across the network. Alternative paths to the destination available to the source can have different format destination lists. Alternative paths to a destination determined by an intelligent routing device have the same format destination lists.

A.7 Network level

IEEE Standard 1355-1995	ECSS-E-50-12 SpaceWire Standard
There is no network level.	Network level is described in clause 10.

A.8 Error recovery scheme

IEEE Standard 1355-1995	ECSS-E-50-12 SpaceWire Standard
There is no error recovery Scheme defined.	Error recovery scheme is described in clause 11.

A.9 Other minor differences

IEEE Standard 1355-1995	**ECSS-E-50-12 SpaceWire Standard**
FCT and FCC have been used interchangeably.	The name flow control token (FCT) is used consistently throughout this Standard.
EOP tokens are colled EOP-1 and EOP-2.	The two EOP tokens of IEEE Standard 1355-1995 (EOP-1 and EOP-2) are renamed EOP (End of Packet) and EEP (Error End of Packet). EOP of SpaceWire and EOP-1 of IEEE Standard 1355-1995 have the same function. The EEP of SpaceWire is used specifically for error recovery purposes and terminates a packet that has ended prematurely due to a link error.

(This page is intentionally left blank)

Annex B (informative)

State exit conditions for encoder-decoder state machine

The state exit conditions for encoder-decoder state machine are shown in Table B-1.

Table B-1: State exit conditions for encoder-decoder state machine

Current state and exit conditions				**Comments**
ErrorReset **STATE**				Entered when the link interface is reset, when the link is disabled [Link Disabled] in the *Run* state, or when error conditions occur in any other state.
Move to *ErrorWait* state when				
After 6,4 µs				
ErrorWait **STATE**				Entered from *ErrorReset* state after being in *ErrorReset* state for 6,4 µs (After 6,4 µs condition TRUE)
Move to *ErrorReset* state when				
First Bit Received	**AND**	Disconnect Error		
OR				
First NULL Received	**AND**	Parity Error	**OR**	
		Escape Error	**OR**	
		gotFCT	**OR**	
		gotN-Char	**OR**	
		gotTime-Code		
Move to *Ready* state when				
After 12,8 µs				

Current state and exit conditions				Comments
***Ready* STATE**				Entered from *ErrorWait* state after being in *ErrorWait* state for 12,8 µs (After 12,8 µs condition TRUE)
Move to *ErrorReset* state when				
First Bit Received	**AND**	Disconnect Error		
OR				
First NULL Received	**AND**	Parity Error	**OR**	
		Escape Error	**OR**	
		gotFCT	**OR**	
		gotN-Char	**OR**	
		gotTime-Code		
Move to *Started* state when				
[Link Enabled]				
***Started* STATE**				Entered from *Ready* state when [Link Enabled] guard is TRUE
Move to *ErrorReset* state when				
After_12,8 µs				GotNULL Timeout
OR				
First Bit Received	**AND**	Disconnect Error		
OR				
First NULL Received	**AND**	Parity Error	**OR**	
		Escape Error	**OR**	
		gotFCT	**OR**	
		gotN-Char	**OR**	
		gotTime-Code		
Move to *Connecting* state when				
GotNULL				
***Connecting* STATE**				Entered from *Started* state on receipt of gotNULL (which also satisfies First Bit Received)
Move to *ErrorReset* state when				
After_12,8 µs	**OR**			gotFCT Timeout
Disconnect Error	**OR**			First Bit Received as part of the gotNULL
Parity Error	**OR**			First NULL Received is already true in order to enter this state
Escape Error	**OR**			First NULL Received is already true in order to enter this state
gotN-Char	**OR**			First NULL Received is already true in order to enter this state
gotTime-Code				First NULL Received is already true in order to enter this state
Move to *Run* state when				
GotFCT				

Current state and exit conditions				**Comments**
Run STATE				Entered from *Connecting* state when FCT received. First Bit Received and gotNULL conditions are TRUE since they were true in *Connecting* state.
Move to *ErrorReset* state when				
Disconnect Error	**OR**			First Bit Received is already true since passed through *Connecting* State
Parity Error	**OR**			First NULL Received is already true since passed through *Connecting* State
Escape Error	**OR**			First NULL Received is already true since passed through *Connecting* State
Credit Error	**OR**			First NULL Received is already true since passed through *Connecting* State
Link Disabled				

(This page is intentionally left blank)

Annex C (informative)

Availability of referenced documents

At the time of publication of this Standard, electronic copies of the normative references (see clause 2) can be found at:

- ANSI publications: http://www.ansi.org.
- ESCC publications: https://escies.org
  ESCC 3902/003 and ESCC 3401/071 are also posted at
  http://www.estec.esa.nl/tech/spacewire/literature
- ECSS publications: http://www.ecss.nl

At the time of publication of this Standard, electronic copies of the bibliogarpy documents can be found at:

- IEEE publications: http://www.standards.ieee.org/
- http://www.estec.esa.nl/tech/spacewire/literature

Paper copies of the documents referenced in this Standard are available as follows:

- ANSI publications are available from the Sales Department, American National Standards Institute, 25 West, 43rd Street, 4th Fl., New York, NY 10036, USA.
- IEEE publications are available from the Institute of Electrical and Electronics Engineers, 455 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA.
- ESCC publications are available from the ESCC Secretariat, ESTEC TOS-QCS), P.O. Box 299, 2200 AG Noordwijk, The Netherlands.
- ECSS publications are available, against a nominal charge, from ESA Publications Division, ESTEC, P.O. Box 299, 2200 AG Noordwijk, The Netherlands.

(This page is intentionally left blank)

Bibliography

[1] IEEE Computer Society, *"IEEE Standard for Heterogeneous Interconnect (HIC) (Low-Cost, Low-Latency Scalable Serial Interconnect for Parallel System Construction)", IEEE Standard 1355-1995, IEEE, June 1996.*

[2] IEEE Computer Society, *"IEEE Standard for Low-Voltage Differential Signals (LVDS) for Scalable Coherent Interface (SCI)", IEEE Standard 1596.3-1996, IEEE, July 1996.*

[3] Konig W, *"Rosetta EMC Requirements Specification", RO-DSS-RS-1002, Issue 2, Daimler-Benz Aerospace, January 1998.*

[4] Parkes SM, Allinniemi T and Rastetter P, *"Digital Interface Circuit Evaluation Study Final Report", ESA Contract No. 12693/97/NL/FM, University of Dundee, March 2001.*

[5] Parkes SM, *"High-Speed, Low-Power, Excellent EMC: LVDS for On-Board Data Handling", Proceedings of the 6th International Workshop on Digital Signal Processing Techniques for Space Applications, ESTEC, Sept. 1998.*

[6] IEEE Computer Society, *"IEEE Standard for a High Performance Serial Bus", IEEE Standard 1394-1995, IEEE, August 1996.*

[7] May MD, Thompson PW & Welch PH, *"Networks, Routers & Transputers: Function, Performance and Application", IOS Press, ISBN 90 5199 129 0, 1993.*

[8] ISO/IEC Directives, Part 3, *Rules for the structure and drafting of International Standards, 3rd edition, 1997.*

(This page is intentionally left blank)

ECSS Document Improvement Proposal

1. Document I.D. ECSS-E-50-12A	**2. Document date** 24 January 2003	**3. Document title** SpaceWire – Links, nodes, routers and networks
4. Recommended improvement (identify clauses, subclauses and include modified text or graphic, attach pages as necessary)		
5. Reason for recommendation		
6. Originator of recommendation		
Name:	Organization:	
Address:	Phone: Fax: e-mail:	**7. Date of submission:**
8. Send to ECSS Secretariat		
Name: W. Kriedte ESA-TOS/QR	Address: ESTEC, P.O. Box 299 2200 AG Noordwijk The Netherlands	Phone: +31-71-565-3952 Fax: +31-71-565-6839 e-mail: Werner.Kriedte@esa.int

Note: The originator of the submission should complete items 4, 5, 6 and 7.

An electronic version of this form is available in the ECSS website at: http://www.ecss.nl/
At the website, select “Standards” – “ECSS forms” – “ECSS Document Improvement Proposal”

(This page is intentionally left blank)