



BSI Standards Publication

Space engineering — SpaceWire — Links, nodes, routers and networks

National foreword

This British Standard is the UK implementation of EN 16603-50-12:2020.

The UK participation in its preparation was entrusted to Technical Committee ACE/68, Space systems and operations.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

© The British Standards Institution 2020
Published by BSI Standards Limited 2020

ISBN 978 0 539 05707 2

ICS 49.140

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 31 March 2020.

Amendments/corrigenda issued since publication

Date	Text affected
------	---------------

English version

Space engineering - SpaceWire - Links, nodes, routers and networks

Ingénierie spatiale - SpaceWire - Liaisons, noeuds,
routeurs et réseaux

Raumfahrttechnik - SpaceWire - Verbindungen,
Knoten, Router und Netzwerke

This European Standard was approved by CEN on 29 December 2019.

CEN and CENELEC members are bound to comply with the CEN/CENELEC Internal Regulations which stipulate the conditions for giving this European Standard the status of a national standard without any alteration. Up-to-date lists and bibliographical references concerning such national standards may be obtained on application to the CEN-CENELEC Management Centre or to any CEN and CENELEC member.

This European Standard exists in three official versions (English, French, German). A version in any other language made by translation under the responsibility of a CEN and CENELEC member into its own language and notified to the CEN-CENELEC Management Centre has the same status as the official versions.

CEN and CENELEC members are the national standards bodies and national electrotechnical committees of Austria, Belgium, Bulgaria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Netherlands, Norway, Poland, Portugal, Republic of North Macedonia, Romania, Serbia, Slovakia, Slovenia, Spain, Sweden, Switzerland, Turkey and United Kingdom.



CEN-CENELEC Management Centre:
Rue de la Science 23, B-1040 Brussels

Table of contents

European Foreword	7
1 Scope	8
2 Normative references	9
3 Terms, definitions and abbreviated terms	10
3.1 Terms from other standards	10
3.2 Terms specific to the present standard	10
3.3 Abbreviated terms	20
3.4 Conventions	21
3.4.1 Numbers	21
3.4.2 Differential signals	21
3.4.3 Order of sending bits in symbols	21
3.4.4 Graphical representation of packets	22
3.4.5 State diagram notation	22
3.4.6 UML diagram notation	23
3.5 Nomenclature	24
4 Overview of SpaceWire	25
4.1 Introduction	25
4.2 SpaceWire Spacecraft Data-Handling Network	25
4.2.1 The Rationale for SpaceWire	25
4.2.2 Example SpaceWire Application	26
4.2.3 How SpaceWire Works	28
5 Requirements	33
5.1 Overview	33
5.2 Protocol stack and interface architecture	33
5.2.1 Protocol stack	33
5.2.2 Network layer	34
5.2.3 Data Link layer	34
5.2.4 Encoding layer	35
5.2.5 Physical layer	35

5.2.6	Management Information Base	36
5.2.7	Service interfaces	36
5.2.8	SpaceWire Port architecture	36
5.3	Physical layer	37
5.3.1	Introduction	37
5.3.2	Cables.....	38
5.3.3	Connectors	41
5.3.4	Cable assemblies.....	45
5.3.5	PCB tracks.....	50
5.3.6	Line drivers and receivers	51
5.3.7	Data-Strobe skew	59
5.3.8	Physical layer management parameters	62
5.4	Encoding layer.....	63
5.4.1	Introduction	63
5.4.2	Serialisation and de-serialisation.....	63
5.4.3	Character and control code encoding.....	63
5.4.4	Data strobe encoding and decoding.....	66
5.4.5	First Null.....	68
5.4.6	Null detection	68
5.4.7	Parity error	69
5.4.8	Disconnect	69
5.4.9	ESC error.....	69
5.4.10	Data signalling rate	70
5.4.11	Encoding layer management parameters.....	71
5.5	Data link layer.....	71
5.5.1	Introduction	71
5.5.2	Data link layer interfaces.....	71
5.5.3	Data link layer management interface	72
5.5.4	Flow control.....	73
5.5.5	Flow control errors	74
5.5.6	Sending priority	75
5.5.7	Link initialisation behaviour	75
5.5.8	Link error recovery	80
5.5.9	Accepting broadcast codes for sending.....	82
5.6	SpaceWire network layer	82
5.6.1	Introduction	82
5.6.2	SpaceWire packets	83

5.6.3	Broadcast codes	83
5.6.4	SpaceWire time-codes	84
5.6.5	SpaceWire distributed interrupts	87
5.6.6	SpaceWire nodes	94
5.6.7	SpaceWire node management parameters	96
5.6.8	SpaceWire routing.....	96
5.6.9	SpaceWire routing switch management parameters	104
5.6.10	SpaceWire network	104
5.7	SpaceWire management information base	106
5.7.1	Introduction	106
5.7.2	General	106
5.7.3	Physical layer management parameters	106
5.7.4	Encoding layer management parameters.....	106
5.7.5	Data link layer management parameters.....	106
5.7.6	Network layer management parameters.....	106
6	Service interfaces.....	107
6.1	Network layer service interface	107
6.1.1	Packet service interface	107
6.1.2	Time-code service interface	108
6.1.3	Distributed interrupt service interface	109
6.2	Data link layer service interface	111
6.2.1	N-Char service interface.....	111
6.2.2	Broadcast code service interface	113
6.3	Encoding layer service interface	114
6.3.1	Encoding service interface	114
6.3.2	Decoding service interface	115
6.4	Physical layer service interface.....	117
6.4.1	Line transmit service interface.....	117
6.4.2	Line receive service interface	118
6.5	Management information base service interface.....	119
6.5.1	Set parameter service interface.....	119
6.5.2	Get parameter service interface	119
Annex A	(informative) Technical Changes.....	121
Bibliography		123

Figures

Figure 3-1: Convention for first bit to be sent	21
Figure 3-2: Graphical packet notation	22
Figure 3-3: State diagram style	22
Figure 3-4: UML notation	23
Figure 4-1: Example SpaceWire Architecture without redundancy	27
Figure 4-2: SpaceWire Packet Format	29
Figure 4-3: Path Addressing	31
Figure 4-4 Logical Addressing	32
Figure 5-1: SpaceWire protocol stack	34
Figure 5-2: SpaceWire port architecture	37
Figure 5-3: SpaceWire connector contact identification	44
Figure 5-4: SpaceWire cable assembly Type A	47
Figure 5-5: SpaceWire cable assembly Type AL	49
Figure 5-6: SpaceWire LVDS	53
Figure 5-7: LVDS line driver output signals	54
Figure 5-8: LVDS line driver differential output signal	56
Figure 5-9: Physical layer components	59
Figure 5-10: Skew and jitter	62
Figure 5-11: Data character encoding	64
Figure 5-12: Control character encoding	65
Figure 5-13: Null control code encoding	65
Figure 5-14: Broadcast code encoding	66
Figure 5-15: Parity coverage	66
Figure 5-16: Data-Strobe (DS) encoding	67
Figure 5-17: Data and strobe signals for first Null	68
Figure 5-18: Null detection sequence	69
Figure 5-19: Link initialisation behaviour	76
Figure 5-20: Link error recovery behaviour	81
Figure 5-21: SpaceWire packet format	83
Figure 5-22: Specialisations and relationships of a SpaceWire broadcast code	84
Figure 5-23 Network layer time-code	85
Figure 5-24: Network layer interrupt	88
Figure 5-25: Network layer interrupt acknowledgement code	89
Figure 5-26: Components and specialisations of a SpaceWire node	95
Figure 5-27: Components of a SpaceWire routing switch	97
Figure 5-28: Components of a SpaceWire network	105

Tables

Table 5-1: Constants for a SpW cable using 28 AWG differential pairs	39
Table 5-2: Insertion loss values 28AWG SpW cable.....	40
Table 5-3: Constants for a SpaceWire cable using 26 AWG differential pairs	40
Table 5-4: Insertion loss values for 26 AWG SpaceWire cable	40
Table 5-5: Cable PSNEXT specification	41
Table 5-6: Cable PSELFEXT specification	41
Table 5-7: Connector contact identification.....	43
Table 5-8: Cable assembly Type A signal wire connections	47
Table 5-9: Cable assembly Type AL signal wire connections	49
Table 5-10 Example calculation of maximum bit rate.....	61
Table 5-11: Address function.....	99

European Foreword

This document (EN 16603-50-12:2020) has been prepared by Technical Committee CEN-CENELEC/TC 5 "Space", the secretariat of which is held by DIN.

This standard (EN 16603-50-12:2020) originates from ECSS-E-ST-50-12C Rev.1.

This European Standard shall be given the status of a national standard, either by publication of an identical text or by endorsement, at the latest by September 2020, and conflicting national standards shall be withdrawn at the latest by September 2020.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. CEN [and/or CENELEC] shall not be held responsible for identifying any or all such patent rights.

This document has been prepared under a standardization request given to CEN by the European Commission and the European Free Trade Association.

This document has been developed to cover specifically space systems and has therefore precedence over any EN covering the same scope but with a wider domain of applicability (e.g. : aerospace).

According to the CEN-CENELEC Internal Regulations, the national standards organizations of the following countries are bound to implement this European Standard: Austria, Belgium, Bulgaria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Former Yugoslav Republic of Macedonia, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Netherlands, Norway, Poland, Portugal, Romania, Serbia, Slovakia, Slovenia, Spain, Sweden, Switzerland, Turkey and the United Kingdom.

1

Scope

SpaceWire technology has grown from the needs of spacecraft on-board data handling 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.

This standard may be tailored for the specific characteristics and constraints of a space project in conformance with ECSS-S-ST-00.

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 revision 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 more recent editions of the normative documents indicated below. For undated references, the latest edition of the publication referred to applies.

EN reference	Reference in text	Title
EN 16601-00-01	ECSS-S-ST-00-01	ECSS system - Glossary of terms
EN 16602-70-08	ECSS-Q-ST-70-08	Space product assurance - Manual soldering of high-reliability electrical connections
EN 16602-70-26	ECSS-Q-ST-70-26	Space product assurance - Crimping of high-reliability electrical connections
	ESCC 3401	Connectors, electrical, non-filtered circular and rectangular, ESCC Generic Specification no. 3401, Issue 5, March 2018
	ESCC 3401/029:2017	Connectors, Electrical, Rectangular, Microminiature, based on type MDMA, ECSS Detail Specification no. 3401/029, Issue 15, November 2017.
	ESCC 3401/077:2016	Connectors, Electrical, Rectangular, Microminiature, Removable Crimp Contacts, based on type MDMA, ECSS Detail Specification no. 3401/077, Issue 7, April 2016.
	ESCC 3902/003:2014	Cable, "SpaceWire", Round, Quad using Symmetric Cables, Flexible, -200 to +180 °C, Detail Specification no. 3902/003, Issue 4, November 2014.
	ESCC 3902/004:2014	Cable, Low Mass, "SpaceWire", Round, Quad using Symmetric Cables, Flexible, -100 to +150 °C, Detail Specification no. 3902/004, Issue 1, October 2014.
	MIL-DTL-17J:2014	Military Specification: Cables, Radio, Frequency, Flexible and Semirigid, General Specification for, 10 th February 2014.
	TIA-644-A:2012	TIA-644-A, Electrical Characteristics of Low Voltage Differential Signalling (LVDS) Interface Circuits, Revision A, Reaffirmed 12/07/12, Telecommunications Industry Association, 2012.

Terms, definitions and abbreviated terms

3.1 Terms from other standards

For the purpose of this Standard, the terms and definitions from ECSS-S-ST-00-01 apply.

3.2 Terms specific to the present standard

The UML diagrams of Figure 5-22, Figure 5-26, Figure 5-27 and Figure 5-28 in clause 5.6 illustrate the relationships between various terms used within this standard.

3.2.1 allocated output port

output port that a **packet** is routed through

3.2.2 after 6,4 μ s

delay of 6,4 μ s (nominal) measured from when a state is entered

3.2.3 after 12,8 μ s

delay of 12,8 μ s (nominal) measured from when a state is entered

3.2.4 AutoStart

management parameter set by hardware or software which when asserted allows an enabled **SpaceWire port** to start the SpaceWire **link** only when a **Null** is received

3.2.5 bit error rate

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

3.2.6 broadcast code

time-code or **distributed interrupt code**

3.2.7 broadcast code identifier

two-bit code that identifies the type of **broadcast code**: 0b00 identifies a **time-code** and 0b10 identifies a **distributed interrupt code**

3.2.8 byte

eight bits

3.2.9 cargo

some information for transfer from a **source** to a **destination** which is encapsulated in a **packet**

3.2.10 character

control character or data character

3.2.11 coding

act of translating a set of bits into another set of bits which are more appropriate for transmitting across a medium

3.2.12 configuration port

port in a **routing switch** or **node** that gives access to a **configuration node**

3.2.13 configuration node

type of **node** whose purpose is to configure the **routing switch** or **node** that it is part of

3.2.14 control character

ESC, FCT, EOP or EEP **character** that is used to pass control information across a **link**

3.2.15 control code

sequence of an ESC followed by an FCT forming a **Null** which is used to keep a **link** active, or a sequence of an ESC **character** followed by a **data character** forming a **broadcast code** which is used to broadcast **time-codes** and **distributed interrupt codes** over a **SpaceWire network**

3.2.16 control symbol

control character encoded in 4-bits for transfer across a **link**

3.2.17 data character

character that is used to pass 8 bits of data across a **link**

3.2.18 data link layer

protocol layer which is responsible for the initialisation of a SpaceWire **link**, for transferring **packets** and **broadcast codes** over the **link** and for recovery from errors on the **link**

3.2.19 data rate

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

3.2.20 data signalling rate

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

3.2.21 data-strobe

sequence of data bits and bit clock encoded into two signals, one containing the data bit sequence (data) and the other changing state whenever the data bit sequence does not (strobe)

3.2.22 data symbol

data character encoded in 10 bits for transfer across a **link**

3.2.23 decoding

act of translating an encoded set of bits back into the original set of bits prior to encoding

3.2.24 de-serialisation

transformation of a serial bit stream into a sequence of **control** or **data symbols**

3.2.25 destination

end-point that a **packet** is being sent to

3.2.26 destination address

route, described by a **path address**, to be taken by a **packet** in moving from **source** to **destination** or an identifier, in the form of a **logical address**, specifying the **destination**

3.2.27 destination node

node that is the **destination** of one or more **SpaceWire packets**

3.2.28 disconnect

indication from a **receiver** that there has been no edge on the data or strobe signals for the past 850 ns (nominal) indicating that the **link** is disconnected

3.2.29 distributed interrupt

interrupt that is distributed to all or many **nodes** on the **network**

3.2.30 distributed interrupt code

broadcast code used to distribute interrupts over a **SpaceWire network** which is either an **interrupt code** or an **interrupt acknowledgement code**

3.2.31 driver

electronic circuit that transmits signals across a particular transmission medium

3.2.32 driver ground

ground at the 0V pin or pins of the **driver** differential outputs

NOTE If the driver has one 0V for signals such as TTL or CMOS, and a separate 0V for the differential outputs, the driver ground is the 0V for the differential outputs. The driver ground is a ground plane or planes which provide

controlled impedance for the driver's differential pairs.

3.2.33 encoding layer

protocol layer which is responsible for the encoding of **characters** into **symbols**, **symbol** serialisation, data-strobe encoding of the serial bit stream, data-strobe decoding, **symbol** de-serialisation and decoding of **symbols** into **characters**

3.2.34 end of packet marker

control character which indicates the end of a **packet**

3.2.35 end-point

interface between the **network** and a **host system** providing a single **port** into the **network**

3.2.36 error recovery scheme

method for handling errors detected within a SpaceWire **link**

3.2.37 ESC

control character which is followed by another **control character** or **data character** to form a **control code**

3.2.38 ESC error

invalid ESC sequence, formed from an ESC followed immediately by an EOP, EEP, or ESC

3.2.39 FIFO port

port that has a FIFO interface rather than a SpaceWire **interface**

3.2.40 fall time

time during which a falling signal voltage is within the range of 80 % to 20 % of the difference between the two steady state values

3.2.41 first Null

initial **Null** received without a parity error when the link state machine is not in the ErrorReset state

3.2.42 flow control token

control character used to manage the flow of data across a **link**, and being exchanged for eight N-Chars

3.2.43 gotBC

sometime after the first **Null** was received, a **broadcast code** has been received without a parity error

3.2.44 gotFCT

sometime after the first **Null** was received, an FCT has been received without a parity error

3.2.45 gotNchar

sometime after the first **Null** was received, an **N-Char** has been received without a parity error

3.2.46 gotNull

first **Null**, after the **receiver** has been enabled, has been received without a parity error

3.2.47 group adaptive routing

assignment of a set of several **ports** to a **logical** or **path address** so that when a **packet** arrives with that address it is switched to one of the set of several **ports** that is currently available to accept the **packet** or that becomes available first

3.2.48 host interface

interface to a **host system**

3.2.49 host system

system that is connected to a **SpaceWire network** via an **end-point** and which uses the services of that **SpaceWire network**

3.2.50 input port

receive side of a **port**

3.2.51 interrupt acknowledgement code

code used to confirm that a **distributed interrupt** has reached the appropriate **interrupt handler**

3.2.52 interrupt code

code used to distribute an interrupt over a **SpaceWire network**

3.2.53 interrupt destination

node that receives and handles a **distributed interrupt**

3.2.54 interrupt handler

node that is responsible for handling an **interrupt code** with a specific value of **interrupt identifier**

3.2.55 interrupt identifier

5-bit value representing one of 32 possible interrupts, which is held in the value field of an **interrupt code** and which identifies the particular interrupt being carried by the **interrupt code** or being acknowledged by an **interrupt acknowledgement code**

3.2.56 interrupt relay register

register in a **routing switch** that holds the current state of the **distributed interrupt** where the i^{th} bit of the register relates to the **interrupt identifier** of value i , and is used to prevent repeated circular propagation of **distributed interrupt codes**

3.2.57 interrupt source

node that generates a **distributed interrupt**

3.2.58 jitter

random errors in the timing of a signal

3.2.59 L-Char

link character

3.2.60 leading data character

very first **data character** sent over a link after initialisation or the first **data character** following the EOP or EEP that terminated the previous packet

3.2.61 line driver

electronic circuit that drives signals across a particular transmission medium

3.2.62 line receiver

electronic circuit that receives signals sent across a particular transmission medium

3.2.63 link

bi-directional connection between two **ports** used to transfer **packets** and **broadcast codes** between the two **ports**

3.2.64 link character

control character or **control code** which appears on the **link** only and is not passed between the data link and network layers

NOTE The **FCT** and **Null** are the only link characters

3.2.65 LinkDisabled

management parameter which when asserted prevents a **SpaceWire port** from operating

3.2.66 link error

disconnect error, parity error, **ESC** error or credit error

3.2.67 link receiver

receiver at an end of a SpaceWire **link**

3.2.68 LinkStart

management parameter set by hardware or software which when asserted causes an enabled **SpaceWire port** to attempt to start the SpaceWire **link** by sending **Nulls**

3.2.69 link transmitter

transmitter at an end of a SpaceWire **link**

3.2.70 logical address

data character which identifies the **destination** for the **packet**

3.2.71 low voltage differential signalling

particular form of differential signalling using low voltage signals

3.2.72 management parameter

configuration parameter, control variable or status variable of a **SpaceWire node** or **routing switch** used to manage its operation

3.2.73 multicast

sending of the same **packet** through two or more output **ports** of a **routing switch** concurrently

3.2.74 multicast set

set of **output ports** assigned to a **logical address** through which a **packet** with that **logical address** is forwarded concurrently

3.2.75 N-Char

normal-character

3.2.76 network level

protocol level responsible for transferring **packets** across a **SpaceWire network** from **source node** to **destination node** via **links** and **routers**

3.2.77 node

source or **destination** of **SpaceWire packets** and **broadcast codes** comprising one or more **end-points** each providing an interface between a **port** and a **host system**

3.2.78 normal-character

data character, **EOP** or **EEP**

3.2.79 Null

control code made up of an **ESC** followed by an **FCT** which is sent to keep the data **link** active when there are no **data** or **control characters** to send, thus preventing a disconnect

3.2.80 output port

transmit side of a **port**

3.2.81 packet

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

3.2.82 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** from **source** to **destination**

3.2.83 physical layer

protocol layer which specifies the cables, connectors, cable assemblies, **line drivers** and **line receivers**

3.2.84 port

SpaceWire interface or FIFO interface comprising an **input port** and an **output port**

3.2.85 port reset

reset signal that resets a single **SpaceWire port**

3.2.86 receive error

error detected when receiving a **symbol**

NOTE This is a parity error or invalid code error

3.2.87 receive FIFO

FIFO memory which stores received **N-Chars** until they can be read by the application via the **SpaceWire port** interface

3.2.88 receiver

circuit that receives signals from the **line receiver** and decodes those signals into a stream of **characters**

3.2.89 receiver ground

ground at the 0V pin or pins of the **receiver** differential inputs

NOTE If the driver has one 0V for signals such as TTL or CMOS, and a separate 0V for the differential inputs, the receiver ground is the 0V for the differential inputs. The receiver ground is a plane or planes which provide controlled impedance for the receiver's differential pairs.

3.2.90 reset

power on reset, other hardware reset or software commanded reset

3.2.91 rise time

time during which a rising signal voltage is within the range of 20 % to 80 % of the difference between the two steady state values

3.2.92 router

routing switch

NOTE The term "router" is used because of the heritage of SpaceWire which is based on IEEE1355-1995 which is in turn based on earlier work on Transputer technology. This work predated the Internet and the use of the term "router" to mean a device that determines the

route to a destination as opposed to a “switch” that switches a packet to an output port based on an address. In SpaceWire the term “router” is used to mean a routing switch. This is widely understood terminology in the SpaceWire community.

3.2.93 routing switch

switch matrix connected to one or more **SpaceWire ports**, a **configuration port** and a local **broadcast code** register, which optionally broadcasts **broadcast codes** and which switches **packets** from one **port** to one or more other **ports** where the **destination address** of each **packet** is used by the **routing switch** to determine which **port** the **packet** is forwarded through

3.2.94 routing table

table in a **routing switch** which is indexed by the leading **data character** of the **packet** to look-up the **output port** for forwarding a **packet** through

3.2.95 serialisation

transformation of a sequence of **control** or **data symbols** into a serial bit stream

3.2.96 signal

measurable quantity that varies with time and propagates along a transmission medium to transfer information across that medium

3.2.97 skew

difference in time between the expected position of the rising or falling edge of a signal and the actual position of that signal

3.2.98 source

originator of a **packet**, signal or other form of information

3.2.99 source node

node that is the **source** of one or more SpaceWire **packets**

3.2.100 SpaceWire interface

interface comprising a transmitter for sending information across a SpaceWire **link**, and a receiver for receiving information from that SpaceWire **link**

3.2.101 SpaceWire network

two or more **nodes** connected together via one or more **links** and zero or more **routing switches**

NOTE A point-to-point link between two nodes is therefore regarded as a network.

3.2.102 SpaceWire node

node

3.2.103 SpaceWire port

port which has a SpaceWire interface

3.2.104 switch matrix

non-blocking, worm-hole **routing switch** that switches a **packet** arriving at an **input port** of a **routing switch** to the appropriate **output port**, or to several **output ports** in case of multicasting

3.2.105 symbol

encoded **data character**, encoded **control character** or encoded **control code**

3.2.106 time-code

code used to distribute synchronisation information over a SpaceWire network

3.2.107 time-code master

host application on a **node** that is responsible for periodically sending out **time-codes** which are broadcast across the SpaceWire network

3.2.108 time-code master node

node on which the **time-code master** resides and which when requested by a **time-code master** sends **time-codes** out of one or more **end-points**

3.2.109 time-code register

register in a **node** or **routing switch** that holds the value of the last **time-code** received

3.2.110 time-code value

six bits of information contained in a **time-code**

3.2.111 transmission medium

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

3.2.112 transmit FIFO

FIFO memory which stores **N-Chars** until they can be sent across the **link**

3.2.113 transmitter

circuit that encodes the **characters** being sent across a **link**, serialises them, data-strobe encodes them and passes the resulting data and strobe signals to **line drivers**

3.2.114 unit

entity, like an instrument, processor or mass memory, which contains zero or more **nodes** and zero or more **routing switches**, and which contains at least one **node** or one **routing switch**

3.3 Abbreviated terms

The following abbreviations are defined and used within this standard:

Abbreviation	Meaning
AWG	American Wire Gauge
BC	broadcast code
BER	bit error rate
DC	direct current
DS	Data-Strobe
ECSS	European Cooperation for Space Standardization
EEP	error end of packet
EOP	end of packet
ESA	European Space Agency
ESC	escape character
ESCC	European Space Components Coordination
FCT	flow control token
FIFO	first in first out
ID	identifier
IID	Interrupt identifier
LS	least-significant
LSB	least significant bit
LVDS	low voltage differential signalling
LVTTL	low voltage transistor-transistor logic
Mbps	Megabits per second
MS	most-significant
MSB	most-significant bit
PCB	printed circuit board
PSELFEXT	Power Sum Equal Level Far-End Cross-Talk
PSNEXT	Power Sum Near End Cross-Talk
RX	receive
SpW	SpaceWire
TDR	time domain reflectometer
TX	transmit
UML	Unified Modelling Language

3.4 Conventions

3.4.1 Numbers

In this document hexadecimal numbers are written with the prefix 0x, for example 0x34 and 0xDF15.

Binary numbers are written with the prefix 0b, for example 0b01001100 and 0b01.

Decimal numbers have no prefix.

3.4.2 Differential signals

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 are designated Dout+, Dout- and Sout+, Sout- for Data and Strobe, respectively. Similarly, the signals at the input end of a SpaceWire link are Din+, Din- and Sin+, Sin-.

3.4.3 Order of sending bits in symbols

The first bit of a data symbol or control symbol to be sent is the parity bit.

The data bits of a data symbol are sent least significant bit first.

An arrow is added to relevant diagrams to illustrate the direction that the data moves, and which bit is sent first as shown in Figure 3-1.

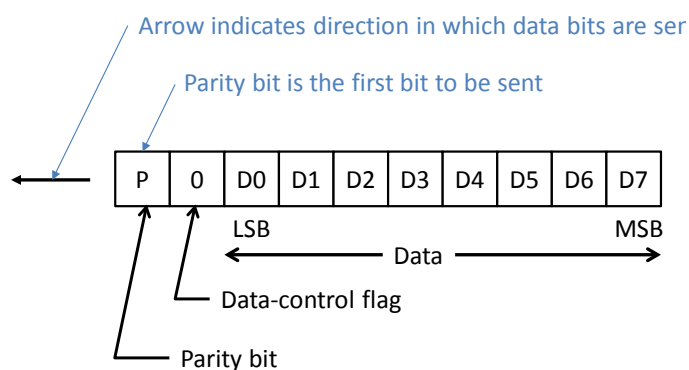


Figure 3-1: Convention for first bit to be sent

3.4.4 Graphical representation of packets

Packets and their component fields are represented graphically as shown in Figure 3-2. The arrow indicates the direction that the packet is travelling, so the destination field is sent first.

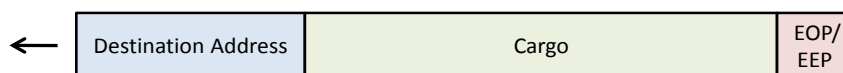


Figure 3-2: Graphical packet notation

3.4.5 State diagram notation

State diagrams are used to represent the behaviour of several parts of the SpaceWire protocol. All state diagrams in this Standard use the style shown in Figure 3-3. 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.

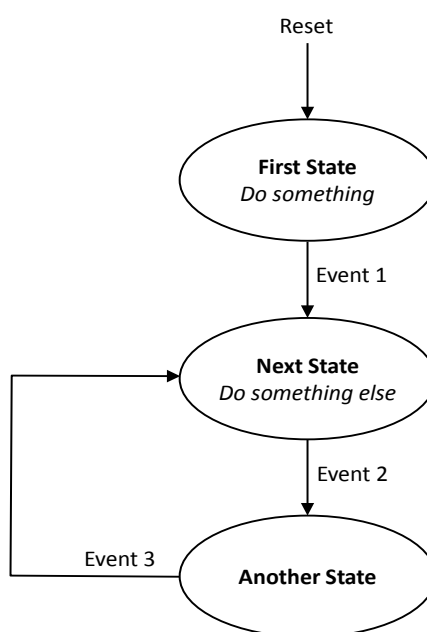


Figure 3-3: State diagram style

3.4.6 UML diagram notation

UML diagrams in this document use the notation illustrated in Figure 3-4.

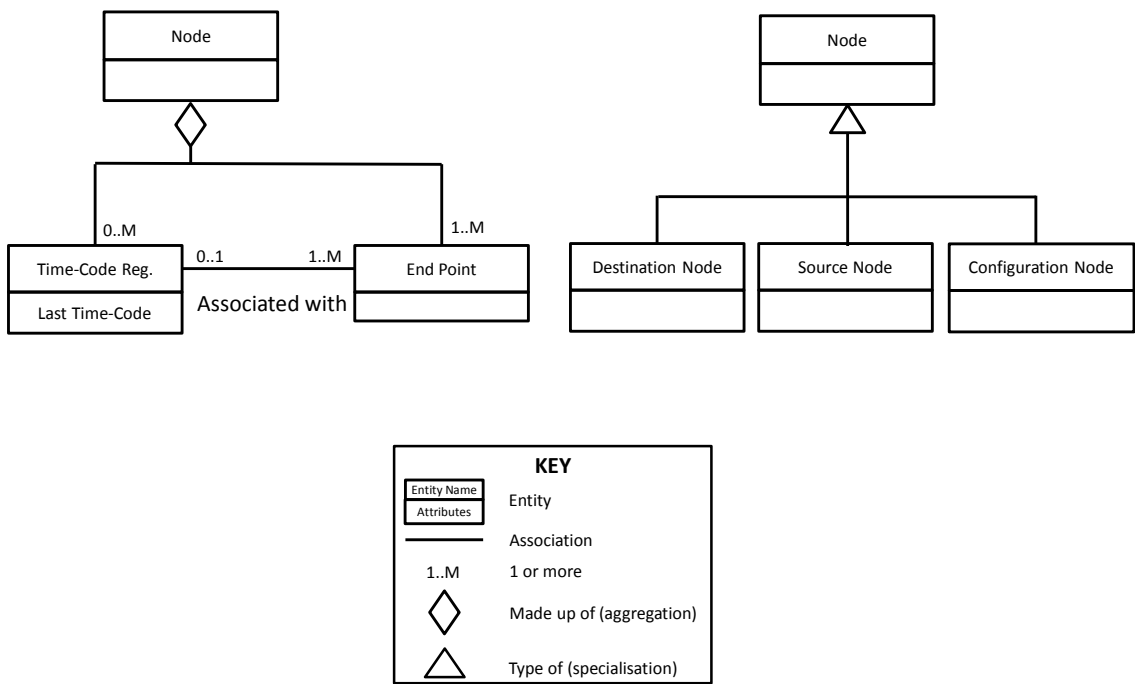


Figure 3-4: UML notation

Each entity (or object) is shown as a rectangle divided in two by a horizontal line. The name of the entity is at the top above the line and a list of key attributes is below the line. Only those attributes that are particularly relevant to understanding a diagram are shown, for example “Last Time-Code” in the Time-Code Register.

A diamond is used to represent aggregation and is translated as “is made up of”, hence in Figure 3-4 a node is made up of one or more (1..M) end-points and zero or more (0..M) time-code registers. The number at the end of the line shows the cardinality of the relationship and is either a number (e.g. 1) or a range (e.g. 1..M) with M representing more or many.

A triangle represents specialisation in one direction (reading in the direction the triangle is pointing) and generalisation in the other. So, in Figure 3-4, a destination node is a special type of node. Reading in the other direction, a node is a generalisation of destination node, source node and configuration node.

Other associations between entities are shown as lines connecting the two objects. A name capturing the intent of the association is written alongside the line. If the association is general, no name needs to be given to it. Hence, in Figure 3-4, each end-point is associated with zero or more time-code registers and each time-code register is associated with 1 or more end-points.

3.5 Nomenclature

The following nomenclature applies throughout this document:

- a. The word “shall” is used in this Standard to express requirements. All the requirements are expressed with the word “shall”.
- b. The word “should” is used in this Standard to express recommendations. All the recommendations are expressed with the word “should”.

NOTE It is expected that, during tailoring, recommendations in this document are either converted into requirements or tailored out.

- c. The words “may” and “need not” are used in this Standard to express positive and negative permissions, respectively. All the positive permissions are expressed with the word “may”. All the negative permissions are expressed with the words “need not”.
- d. The word “can” is used in this Standard to express capabilities or possibilities, and therefore, if not accompanied by one of the previous words, it implies descriptive text.

NOTE In ECSS “may” and “can” have completely different meanings: “may” is normative (permission), and “can” is descriptive.

- e. The present and past tenses are used in this Standard to express statements of fact, and therefore they imply descriptive text.

Overview of SpaceWire

4.1 Introduction

SpaceWire is a data-handling network for use on-board spacecraft, which connects together instruments, mass-memory, processors, downlink telemetry, and other on-board sub-systems. SpaceWire provides 2 Mbps to 200 Mbps, bi-directional, full-duplex data-links, which connect together SpaceWire-enabled equipment. Data rates of over 400 Mbps are possible when matched impedance connectors are used. It delivers high-speed, low-power, simplicity, relatively low implementation cost, and its architectural flexibility makes it ideal for many space missions. Data-handling networks can be built to suit particular applications using point-to-point data-links and routing switches [ESA Bulletin Vol. 145].

Since the SpaceWire standard was published by the European Cooperation for Space Standardization in January 2003, it has been adopted by ESA, NASA, JAXA and Roscosmos for many missions and is being widely used on scientific, Earth observation, commercial and other spacecraft. High-profile missions using SpaceWire include: Gaia, ExoMars, Bepi-Colombo, James Webb Space Telescope, GOES-R, Lunar Reconnaissance Orbiter and Astro-H.

SpaceWire has been developed in an extremely successful collaboration between ESA, academia and space industry, involving spacecraft engineers from across the world. This revision of the SpaceWire standard aims to correct errors, remove ambiguities and make clarifications to the previous version. It adjusts the SpaceWire protocol stack to make the layer interfaces clear and provides abstract service interface descriptions for each layer. Distributed interrupts are added to SpaceWire and time-codes are generalised into broadcast codes which comprise time-codes and distributed interrupt codes.

4.2 SpaceWire Spacecraft Data-Handling Network

4.2.1 The Rationale for SpaceWire

Before SpaceWire became a standard, many spacecraft primes and equipment manufacturers had to use ad hoc or their own proprietary interfaces for inter-unit communications, e.g. connecting high data-rate instruments to mass-memory units. This resulted in several different types of communication links being used on a spacecraft, increasing the cost and extending the time required

for spacecraft integration and test. There was a clear need for a standard on-board communication link to simplify spacecraft development.

SpaceWire aims to:

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

Use of the SpaceWire standard ensures that equipment is compatible at both the component and sub-system levels. Instruments, processing units, mass-memory devices and down-link telemetry systems using SpaceWire interfaces that were developed for one mission can be readily used on another mission. This reduces the cost of development (**Cheaper**) and reduces development timescales (**Faster**) while improving reliability (**Better**) and maximising the amount of science and data return that can be achieved within a given budget (**More**).

4.2.2 Example SpaceWire Application

SpaceWire is able to support many different payload processing architectures using point-to-point links and SpaceWire routing switches. The data-handling architecture can be constructed to suit the requirements of a specific mission, rather than having to force the application onto a constrained bus or network with restricted topology.

An example SpaceWire architecture is shown in Figure 4-1. It uses two SpaceWire routing switches to provide the interconnectivity between instruments, memory and processing modules.

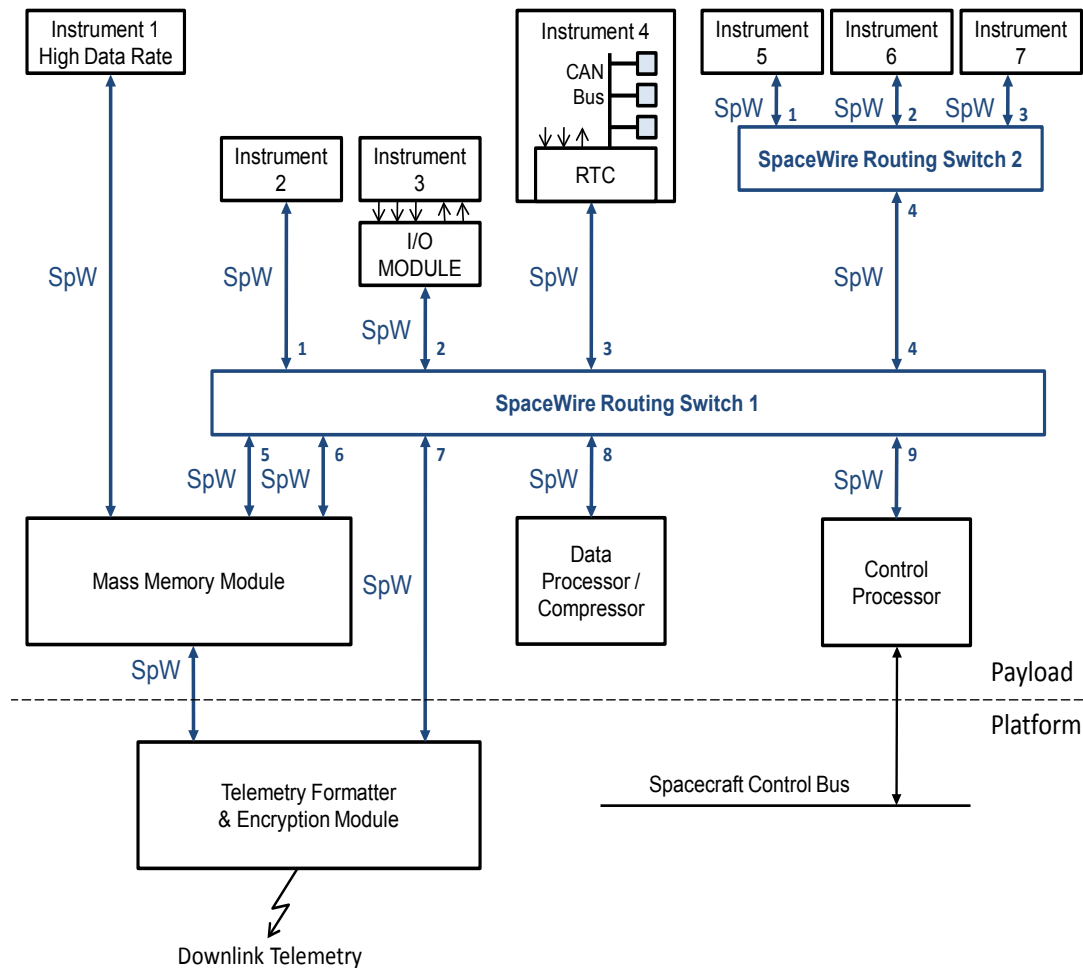


Figure 4-1: Example SpaceWire Architecture without redundancy

Instrument 1 in the top left-hand corner is a high data-rate instrument (e.g. producing data at a rate of 150 Mbps). A SpaceWire point-to-point link is used to stream data from this instrument directly into the Mass Memory Module. If a single SpaceWire link is insufficient to handle the data-rate from this instrument, two or more links can be used in parallel.

Instrument 2 is of lower average data-rate than instrument 1. Its data is passed through SpaceWire Routing Switch 1 to the Mass Memory Module.

Instrument 3 does not have a SpaceWire interface so an input/output (I/O) module is used to connect the instrument to the SpaceWire router. Its data is then be sent over the SpaceWire network to the Mass Memory Module.

Instrument 4 is a complex instrument containing a number of sub-modules which are interconnected using the CAN bus. A Remote Terminal Controller (RTC) is used to bridge between the CAN bus and SpaceWire. Other signals from the instrument are also connected to the RTC, which contains a processor for performing the bridging and local instrument control functions.

Instruments 5, 6 and 7 are located in a remote part of the spacecraft. To avoid having three SpaceWire cables running to this remote location a second routing switch (SpaceWire Routing Switch 2) is used to concentrate the information

from these three instruments and send it over a single SpaceWire link to Routing Switch 1 and then on to the Mass Memory Module.

This Mass Memory Module can receive data from any of the instruments either directly, as is the case for Instrument 1, or indirectly via Routing Switch 1. Data stored in the Mass Memory Module can be sent to the Telemetry Formatter/Encryption Module for sending to Earth, or it is first be sent to a Data Processing or Data Compression Unit. This unit returns the processed/compressed data to the Mass Memory Module or send it straight to the Telemetry Module via Routing Switch 1.

The Control Processor is responsible for controlling all the Instruments, the Mass Memory Module and the Telemetry unit. The SpaceWire Network provides access to all these modules so that the Control Processor can configure, control and read housekeeping and status information from them. The Control Processor is also attached to the spacecraft control bus over which it can receive telecommands and forward housekeeping information.

With several instruments and the Data Processor/Compressor sending data to the Mass Memory Module via Routing Switch 1, a single link from that Routing Switch to the Mass Memory Module can be insufficient to handle all the data, so a second link has been added to provide more bandwidth. In a SpaceWire network links can be added to provide additional bandwidth or to add fault tolerance to the system. In Figure 4-1 no redundancy has been included for clarity. In a spaceflight application, an additional pair of routing switches is included with duplicate links to the modules to provide redundancy. It is straightforward to support traditional cross-strapped, redundant modules using SpaceWire.

4.2.3 How SpaceWire Works

4.2.3.1 Overview

Having looked at the way in which SpaceWire can be used to provide a spacecraft data-handling system, this section examines SpaceWire in more detail to provide an understanding of how SpaceWire works. The normative clauses are presented in clause 5.

4.2.3.2 SpaceWire Links

SpaceWire links are point-to-point data links that connect a SpaceWire node (e.g. instrument, processor, mass-memory unit) to another node or to a router. Information can be transferred in both directions of the link at the same time. Each full-duplex, bi-directional, serial data link can operate at data-rates between 2 Mbps and 200 Mbps. When matched impedance connectors are used, SpaceWire is able to run from 2 Mbps to rates above 400 Mbps. It sends information as a serial bit stream using two signals in each direction (data and strobe). These signals are normally driven across a link using Low Voltage Differential Signalling (LVDS), which requires two wires for each signal, resulting in a SpaceWire cable containing four screened twisted pairs.

Bit synchronisation in SpaceWire is achieved by sending a clock signal along with the serial data. To reduce the maximum clock to data skew requirements

the clock signal is encoded into a strobe signal in such a way that XORing the data and strobe signal recovers the clock signal. Data is transferred as a stream of elementary data and control characters. Character synchronisation is only performed once, when the link is started. If character synchronisation is ever lost, leading to parity errors, this is detected and the link restarted to recover initial character synchronisation.

SpaceWire provides a simple mechanism for starting a link, keeping the link running, sending data over the link, ensuring that data is not sent if the receiver at the other end of the link is not ready for it, and for recovering from any errors on the link. All this is handled by state-machines in the SpaceWire interface and is transparent to the user application.

4.2.3.3 SpaceWire Packets

Information is transferred across a SpaceWire link in distinct packets. Packets can be sent in both directions of the link, provided that there is room in the receiver for more data.

A packet is formatted as illustrated in Figure 4-2.



Figure 4-2: SpaceWire Packet Format

The "Destination Address" at the beginning of the packet contains either the identity of the destination node or the path the packet has to take through a SpaceWire network to reach to the destination node. In the case of a point-to-point link connection directly between two nodes (no routing switches in between), the destination address is not necessary.

The "Cargo" is the data to be transferred from source to destination. An arbitrary number of data bytes can be transferred in the cargo of a SpaceWire packet.

The End of Packet marker (EOP or EEP) is the last character in the packet. The data character following an End of Packet is the start of the next packet. There is no limit on the size of a SpaceWire packet.

As can be seen the SpaceWire packet format is very simple. It is, however, also very powerful, allowing SpaceWire to be used to carry a wide range of application protocols, with minimal overhead.

4.2.3.4 SpaceWire Networks

4.2.3.4.1 General

SpaceWire networks are constructed using SpaceWire point-to-point links and routing switches.

4.2.3.4.2 SpaceWire Routing Switches

A SpaceWire routing switch connects together many nodes using SpaceWire links, providing a means of routing packets from one node to any of the other nodes or routing switches attached to the router. A node is simply the source or destination of a SpaceWire packet. A SpaceWire routing switch comprises a number of SpaceWire ports and a switch matrix. The switch matrix enables packets arriving at one port to be transferred to and sent out through another port on the router.

Each port can be considered as comprising an input port (the port receiver) and an output port (the port transmitter). A SpaceWire routing switch transfers packets from the input port of the routing switch where the packet arrives, to a particular output port determined by the packet destination address. A routing switch uses the leading data character of a packet (one of the destination address characters) to determine the output port of the routing switch to which the packet is routed. If there are two input ports waiting to use a particular output port when the previous packet has finished being sent, an arbitration mechanism decides which input port is served next. The packet from the other input port is queued until the output port becomes available again. This so called “wormhole” switching requires a minimum amount of buffer memory, which enables a SpaceWire routing switch to be implemented even in smaller, space-qualified chip technologies.

4.2.3.4.3 Packet Addressing

The destination address at the front of a SpaceWire packet is used to route the packet through a network from the source node to the destination. There are two forms of addressing used in SpaceWire networks: path addressing and logical addressing.

Path addressing can best be understood using the simple analogy of providing directions to someone driving a car. To reach the destination it is suggested that the driver takes exit 2 at the first road-junction, exit 1 at the next road-junction, and finally exit 3 on the third road-junction. The driver then reaches the required destination (see Figure 4-3). There is a command to follow at each road-junction (take a particular exit). Together these commands describe the path from the initial position to the required destination. There is a list of commands to follow; one for each road-junction. Once a command has been followed it is dropped from the list and the next command is followed at the next road-junction.

In a SpaceWire network the road-junctions are routing switches and the roads connecting the road-junctions are the links connecting the routing switches (see Figure 4-3). The list of commands is attached to the front of the SpaceWire packet forming the destination address. The first command is followed when the first routing switch is encountered. This command is simply a data character that specifies which port of the routing switch the packet is forwarded

through. A routing switch can have a maximum of 31 external ports (port numbers 1 to 31) and one internal configuration port (port number 0). The leading data character at the front of the packet is used to specify the port that the packet is forwarded through: for instance, if the leading data character is 3, the packet is forwarded to port 3 of the router. Once the leading data character has been used to forward a packet, it is removed as it is no longer needed. This reveals the next data character in the path address for routing the packet at the next router. Figure 4-3, shows how the path address at the start of the packet is modified as the packet goes through the network. Since a routing switch can have a maximum of 31 ports along with an internal configuration port, each data character forming a path address is in the range 0 to 31.

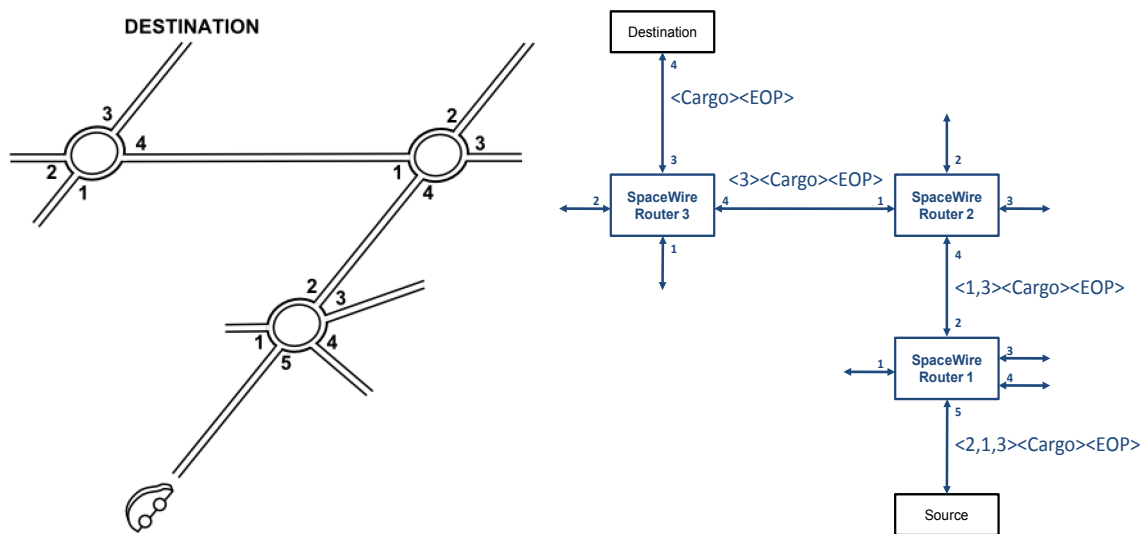


Figure 4-3: Path Addressing

Logical addressing can also be understood using the analogy of giving directions to a car driver. Logical addressing is like saying to the driver, “follow the signs to the City Centre at each of the road-junctions”. This, of course, requires appropriate signs to be placed at each road-junction and each destination has to have a name or identifier so that it can be recognised on the signs. The logical address analogy is illustrated in Figure 4-4.

In a SpaceWire network using logical addressing, each destination is given an identifier, which is a number in the range 32 to 255. Each routing switch in the network has a routing table (like the sign at a road-junction) which specifies the port a packet is forwarded through for each possible destination identifier. The leading data character of a packet is set to the required destination identifier (e.g. 44 for City Centre). At each routing switch the leading data character is used to look up the appropriate directions from the routing table and the packet is forwarded accordingly. For logical addressing the leading data character is not discarded at each router, since it is needed to look up the path to follow at the next routing switch encountered. The use of logical addressing is illustrated in Figure 4-4. Logical addressing uses just a single data character to identify the destination which is in the range 32 to 255 so that it is not confused with path addresses.

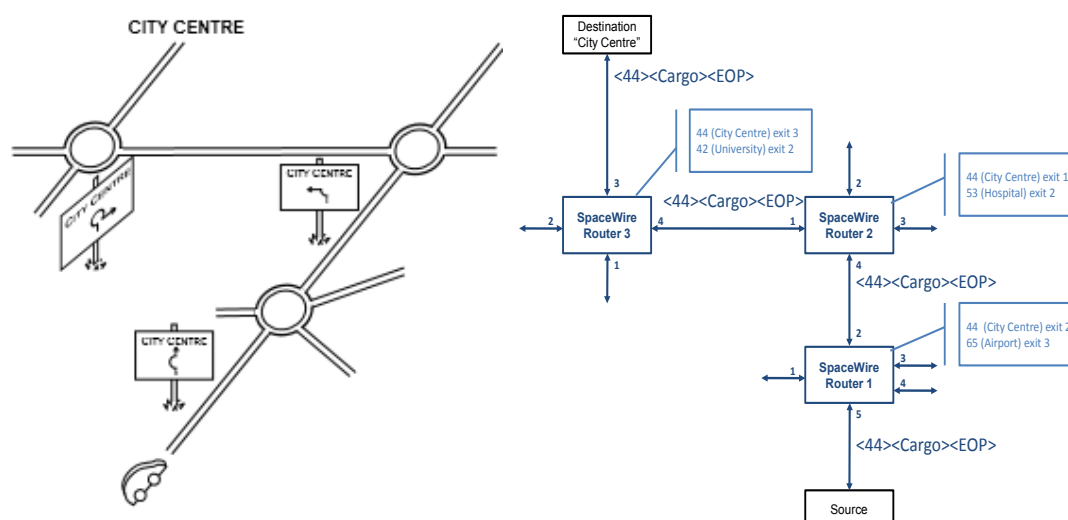


Figure 4-4 Logical Addressing

Logical addressing has the advantage that only a single address byte is required, but it requires the routing tables to be configured before it can be used. Path addressing requires a byte for each routing switch and does not depend on routing tables in the routers.

4.2.3.5 Time-Codes

SpaceWire time-codes provide a means of synchronising units across a SpaceWire system with low jitter. This time information is provided as “ticks”, based on an incrementing value that is possibly synchronized to spacecraft time. The time-codes are transmitted over the SpaceWire network with high priority, alleviating the possible need for a separate time distribution network.

4.2.3.6 Distributed interrupts

SpaceWire distributed interrupts are used to signal events across a SpaceWire network with low jitter. Up to 32 different interrupts can be broadcast over the network with high priority.

5

Requirements

5.1 Overview

This clause provides the normative requirements for SpaceWire. It is separated into several functional layers.

- Clause 5.1 is a short overview of the following sub-clauses.
- Clause 5.2 describes the SpaceWire protocol stack and architecture of a SpaceWire Port.
- Clause 5.3 specifies the Physical layer of SpaceWire which covers the specification of the cables, connectors, cable assemblies, line drivers and line receivers.
- Clause 5.4 specifies the Encoding layer which covers the encoding of characters into symbols, symbol serialisation, data-strobe encoding of the serial bit stream, data-strobe decoding, symbol de-serialisation and decoding of symbols into characters.
- Clause 5.5 specifies the Data Link Layer of SpaceWire which is responsible for transferring packets and broadcast codes over a link, initialising the SpaceWire link, and managing the recovery from errors on the link.
- Clause 5.6 describes the Network layer which covers SpaceWire packets, nodes, routing switches, networks, time-code broadcasting and distributed interrupt operation.

5.2 Protocol stack and interface architecture

5.2.1 Protocol stack

- a. The SpaceWire protocol stack shall have a Network layer, Data Link layer, Encoding layer, Physical layer and Management information base.

NOTE The SpaceWire protocol stack is illustrated in Figure 5-1.

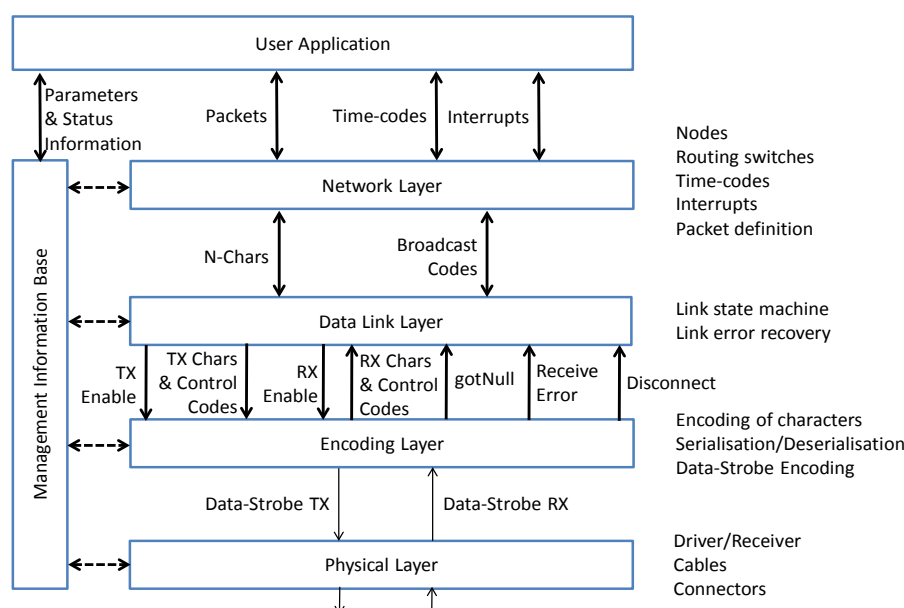


Figure 5-1: SpaceWire protocol stack

5.2.2 Network layer

- a. The SpaceWire Network layer shall provide three principal services:
 1. A packet service which sends and receives packets over a SpaceWire network.
 2. A time-code service which sends and receives time-codes over a SpaceWire network.
 3. A distributed interrupt service which sends and receives distributed interrupts over a SpaceWire Network.
- b. A SpaceWire implementation shall support the packet service, optionally support the time-code service and optionally support the distributed interrupt service.
- c. The SpaceWire Network layer shall accept service requests from user applications.
- d. The SpaceWire Network layer shall be responsible for transferring SpaceWire packets, time-codes and distributed interrupts over a SpaceWire Network.
- e. The SpaceWire Network layer shall use the services of the SpaceWire Data Link Layer.

5.2.3 Data Link layer

- a. The SpaceWire Data Link layer shall provide two services:
 1. An N-Char service which sends and receives N-Chars (the components of packets) over a SpaceWire link.

2. A broadcast code service which sends and receives broadcast codes (time-codes and distributed interrupt codes) over a SpaceWire link.
- b. The Data Link layer shall accept service requests from the network layer.
- c. The Data Link layer shall be responsible for establishing communications across the SpaceWire link, for controlling the flow of information over the link, for sending and receiving N-Chars, for sending and receiving broadcast codes and for re-establishing communications across the link after errors that occur over the link.
- d. The Data Link layer shall use the services of the SpaceWire encoding layer.

5.2.4 Encoding layer

- a. The SpaceWire encoding layer shall provide two services:
 1. A character encoding service which encodes characters and control codes into symbols, serialise those symbols and data-strobe encode them ready for transmission over the SpaceWire physical layer.
 2. A character decoding service which recovers the data bit stream from the data-strobe signals received from the physical layer, de-serialise that bit-stream, and decode the resulting symbols into characters and control codes.
- b. The Encoding layer shall accept service requests from the SpaceWire data link layer.
- c. The Encoding layer shall be responsible for encoding and decoding of characters into a form suitable for sending over the SpaceWire physical layer.
- d. The Encoding layer shall use the services of the SpaceWire physical layer.

5.2.5 Physical layer

- a. The SpaceWire Physical layer shall provide two services:
 1. A transmit service which transmits the data and strobe signals from the encoding layer over the physical medium.
 2. A receive service which receives the data and strobe signals from the physical medium and passes them to the encoding layer.
- b. The SpaceWire Physical layer shall accept service requests from the encoding layer.
- c. The SpaceWire Physical layer shall be responsible for transmitting and receiving the data and strobe signals over PCB tracks, connectors and cable assemblies.

5.2.6 Management Information Base

- a. The SpaceWire Management Information Base shall provide two services:
 - 1. A set parameter service which writes control or configuration information to the other SpaceWire layers.
 - 2. A get status service which reads the status or current configuration or control values of the other SpaceWire layers.
- b. The SpaceWire Management Information Base shall accept service requests from a user application.
- c. The SpaceWire Management Information Base shall be responsible for configuring, controlling and monitoring the operation of the other SpaceWire layers.
- d. The SpaceWire Management Information Base shall have direct access to the relevant configuration parameters, control parameters and status parameters in the network layer, data link layer, encoding layer and physical layer.

5.2.7 Service interfaces

The Service interfaces for each layer of the SpaceWire protocol stack are specified in clause 6.

5.2.8 SpaceWire Port architecture

- a. A SpaceWire Port shall incorporate a SpaceWire port interface which comprises a packet transmit interface, a packet receive interface, an optional broadcast code transmit interface and an optional broadcast code receive interface.
- b. A SpaceWire Port shall incorporate a transmit FIFO (TX FIFO) which stores N-Chars provided by the application via the SpaceWire port interface until they can be sent across the link.
- c. A SpaceWire Port shall incorporate a receive FIFO (RX FIFO) which stores received N-Chars until they can be read by the application via the SpaceWire port interface.
- d. A SpaceWire Port shall incorporate a flow control manager which manages the flow of data over the link, preventing data from being sent when there is no space for it in the receive FIFO.
- e. A SpaceWire Port shall incorporate a link state machine which is responsible for controlling the starting of a link and its recovery from errors.
- f. A SpaceWire Port shall incorporate a transmitter which is responsible for encoding the characters to be sent over the link into symbols, serialising those symbols into a bit stream and encoding the bit stream into a data and strobe pair of signals.
- g. A SpaceWire Port shall incorporate a receiver which is responsible for decoding the received data and strobe pair of signals into a data bit