Proposal of ECSS—E-50-xx Draft 2.1
May 2005

EUROPEAN COOPERATION
ECSS
FOR SPACE STANDARDIZATION

Space Engineering Standards

Recommendations for CAN Bus in Spacecraft Onboard Applications

ECSS Secretariat
ESA-ESTEC
Requirements & Standards Division
Noordwijk, 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 must be accomplished, rather than in terms of how to organise and perform the necessary work. This allows existing organisational 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 ESTEC CAN Working Group charged with the analysis and development of the CAN (Controller Area Network) data bus for spacecraft applications.

Contents List

Foreword ... 3
Contents List ... 5
1 Scope ... 8
2 References ... 9
2.1 Normative References ... 9
3 Definitions, abbreviations, and conventions ... 10
3.1 Definitions ... 10
3.2 Abbreviations ... 12
3.3 Conventions ... 12
3.3.1 Bit Numbering Convention ... 12
3.3.2 Requirement identification ... 13
4 Overall description (informative) ... 14
4.1 An overview of the CAN bus ... 14
4.2 An overview of spacecraft onboard data characteristics ... 18
4.2.1 Future Trends in Onboard Bus Use ... 19
4.2.2 Summary of Onboard Bus Requirements ... 20
4.3 Content of the standard ... 20
4.4 The CAN bus physical layer specification ... 21
4.5 CANopen higher layer protocol ... 21
4.5.1 Object dictionary ... 22
4.5.2 Electronic Data Sheets ... 22
4.5.3 Communication objects ... 22
4.5.4 Device profiles ... 28
4.6 Synchronous data transfers over CAN bus ... 28
4.6.1 General protocol for synchronous data transfers ... 28
4.6.2 Communication slot organization ... 28
4.7 Transfer of large data units ... 29
4.8 Time distribution ... 30
4.9 CAN object identifier assignments ... 30
4.10 Redundancy management ... 31
5 Physical layer (normative) ... 32
5.1 Introduction ... 32
5.1.1 Scope ... 32
5.2 Topology ... 32
5.2.1 Physical topology (R) ... 32
5.2.2 Maximum bus length and drop length (R) ... 33
5.2.3 Minimum number of network devices (R) ... 33
5.3 Medium ... 34

5.3.1 Cable requirements 34
5.3.2 Connector 34
5.3.3 Shield Grounding – system specific (R) 35
5.4 Transceiver Characteristics 35
5.4.1 Transceiver electrical characteristics (M) 35
5.4.2 Resistance to electrical CAN bus faults (R) 35
5.4.3 Optical isolation (R) 35
5.4.4 Transceiver implementation based on RS-485 transceivers (O) 36
5.5 Bit Timing 36
5.5.1 Bit rate 1 Mbps (M) 36
5.5.2 Other bit rates (O) 37
5.5.3 Bit Timing (M) 37
5.6 Electromagnetic (EMC) Compatibility (R) 37
5.7 Data Link Layer (M) 37
5.7.1 ISO 11898 Compliance (M) 38
5.7.2 Fault confinement (M) 38
6 CANopen higher layer protocol (normative) 39
6.1 General (M) 39
6.2 Communication Objects 39
6.2.1 Service Data Objects (M) 39
6.2.2 Process Data Objects (O) 39
6.2.3 Synchronisation object (O) 40
6.2.4 Emergency object (O) 40
6.2.5 Network management objects (M) 40
6.3 Electronic Data Sheets (M) 41
6.4 Device & Application Profiles (O) 41
6.5 Object Dictionary (M) 41
7 Synchronous data transfers (normative) 42
7.1 Synchronous communications (O) 42
8 Transfer of large data units (normative) 43
8.1 Introduction 43
8.2 Identifier encoding (D) 43
8.2.1 Function ID field (D) 44
8.2.2 Priority field (D) 44
8.2.3 Frame Type Field (D) 44
8.2.4 Source address field (D) 45
8.2.5 Destination address field (D) 45
8.2.6 Protocol ID field (D) 45
8.2.7 Toggle Bit field (D) 45
8.3 Protocol control frames (O) 45
8.3.1 Acknowledge frame (O) 46
8.3.2 Stop frame (O) 46
8.3.3 Resume frame (O) 46
8.3.4 Abort frame (O) 46
8.4 Selective acknowledgement for unsegmented transfers (O) 46
9 Time distribution (normative) 48
9.1 Time objects (O) 48
9.1.1 Time code formats (D) 48
9.1.2 Spacecraft elapsed time objects (D) 48
9.1.3 Spacecraft universal time coordinated objects (D) 49
9.2 Time distribution and synchronization protocols (O) 50
9.2.1 Time distribution protocol (D) 50
9.2.2 High-resolution time distribution protocol (O) 51
10 CAN bus object identifier assignments (normative) 53
10.1 CAN bus version (M) 53
10.2 COB-ID assignment (M) 53
11 Redundancy Management (normative) 55
11.1 General (O) 55

11.2 Node internal bus redundancy architectures (D) 55
11.2.1 Selective bus access architecture (D) 55
11.2.2 Parallel bus access architecture (D) 56
11.3 Bus monitoring and reconfiguration management (D) 57
11.3.1 Bus redundancy management parameters (D) 57
11.3.2 Startup procedure (D) 58
11.3.3 Bus montoring protocol (D) 59
11.3.4 Bus selection process 60
Annex A Recommended connectors and pin assignments (normative) 62
A.2.1 MIL-C D38999 configuration B: Dual CAN bus 62
A.2.2 MIL-C D38999 configuration D: Single CAN bus 63
A.3.1 Micro-miniature D Shell: Dual CAN bus 63
A.3.1 Micro-miniature D Shell: Single CAN bus 63
Annex B Guidelines for implementing bus redundancy management (informative) 65
B.1 Bus monitoring and reconfiguration management 65
B.1.1 Bus redundancy management parameters 65
B.1.2 Startup procedure (D) 66
B.1.3 Bus monitoring protocol (D) 67
B.1.4 Bus selection process 67
Annex C Minimalist implementation of CANopen (informative) 68
C.1 Communication Objects 68
C.1.1 Service Data Objects 68
C.1.2 Network management objects 68
C.2 Object Dictionary 68
Annex D Process for adoption of COB-Ids (informative) 69
Annex E CAN system design issues (informative) 73
Annex F PHY Layer design considerations (informative) 79
Annex G Compliance pro-forma (informative) 80
Bibliography 81

1 Scope

This standard is aimed at spacecraft development projects that opt to use the CAN bus for spacecraft onboard communications and control. This standard specifies the protocols and services that are to be provided on top of the basic CAN bus specification, and indicates how those protocols and services can be implemented on a CAN bus.

This standard does not modify the basic CAN bus specification in any way. Instead the CAN bus is used exactly as defined in ISO 11898-1/-2:2003. This standard merely defines how spacecraft specific requirements can be achieved using protocol extensions running over the CAN bus.

The intention of the standard is to meet the vast majority of the onboard data bus requirements for a broad range of different mission types. However, there may be some cases where a mission has particularly constraining requirements that might not be met by this standard. In those cases it is recommended that this standard should still be used as the basis for the use of CAN bus, with extensions and special amendments only being applied as absolutely needed.

2 References

2.1 Normative References

This ECSS standard incorporates by dated or undated reference, provisions from other publications. These normative references are cited at the appropriate places in the text and publications are listed hereafter. For dated references, subsequent amendments to or revisions of any of these apply to this ECSS standard only when incorporated in it by amendment or revision. For undated references, the latest edition of the publication referred to applies.

ISO 11898-1:2003	Road vehicles – Controller Area Network (CAN) Part 1: Data link layer and physical signalling
ISO 11898-2:2003	Road vehicles – Controller Area Network (CAN) Part 2: High-speed medium access unit
CiA Draft Standard 301 Version 4.02	CANOpen Application Layer and Communication Profile, CAN in Automation e. V.

3
Definitions, abbreviations, and conventions

3.1 Definitions

For the purposes of this standard, the definitions given in ECSS-P-001 Issue 1 apply. The following terms and definitions are specific to this standard and shall be applied.

The specification of data type and encoding rules according to CANopen shall apply.

COB-ID	The Communication Object Identifier is the 11- or 29-bit identifier in the arbitration and control field of the CAN frame.
Cold redundant bus	A redundant bus system where data is only transmitted on one of the available buses.
Dominant bit level	A logical level that when applied to a bus forces the entire bus to the same logical level.
Hot redundant bus	A redundant bus where data is transmitted simultaneously on all of the available buses.
Large Data Unit	Any data unit that requires segmentation to be transferred over the CAN bus, i.e. a data unit of more than eight octets.
Local SCET:	A time counter implemented and maintained in a node, that is correlated with the SCET.
NMT Master:	The node in a CANopen network, responsible for managing all other nodes on the bus using the NMT services.

NMT Slave:	The nodes in a CANopen network that are managed by the NMT Master using the NMT services.
Recessive bit level	A logical level that when applied to a bus only has effect on the level of the bus if there is no driver that simultaneously applies a dominant bit level.
Redundancy master	A dedicated node responsible for managing the bus redundancy. In particular this includes controlling the switching from a nominal to a redundant bus in a cold redundant bus system.
Redundant bus	A bus system that consists of two or more identical physical communication channels to increase the bus reliability or availability.
Redundant node	A node that provides identical functionality as another node connected on the same physical bus.
Spacecraft Elapsed Time (SCET):	A central time reference that is maintained onboard the spacecraft. The SCET may be correlated to the ground time, and may be distributed to other onboard nodes.

3.2 Abbreviations

The following abbreviations are defined and used within this standard.

Abbreviation	**Meaning**
CAN	Controller Area Network
CCSDS	Consultative Committee for Space Data Systems
COB-ID	Communication Object Identifier
ECSS	European Cooperation for Space Standardisation
EDS	Electronic Data Sheet
EMC	Electromagnetic Compatibility
FIFO	First In First Out
LDUT	Large Data Unit Transfer
LSB	Least Significant Bit
MSB	Most Significant Bit
NMT	Network Management
OSI	Open Systems Interconnection
PCB	Printed Circuit Board
PDU	Protocol Data Unit
PDO	Process Data Object
RPDO	Receive PDO
RTR	Remote Transmission Request
SAP	Service Access Point
SCET	Spacecraft Elapsed Time
SDO	Service Data Object
SYNC	Synchronisation Object
TPDO	Transmit PDO

3.3 Conventions

3.3.1 Bit Numbering Convention.

The most significant bit of an n-bit field shall be:

- numbered bit n-1,
- the first bit transmitted,
- the leftmost bit on a format diagram.

The least significant bit of an n-bit field shall be:

- numbered bit 0 (zero),
- the last bit transmitted,
- the rightmost bit on a format diagram.

This is illustrated in Figure 1.

Note: This convention is the opposite of most ECSS and CCSDS documents. Its choice is dictated by the bit numbering convention used in the CAN bus specification.

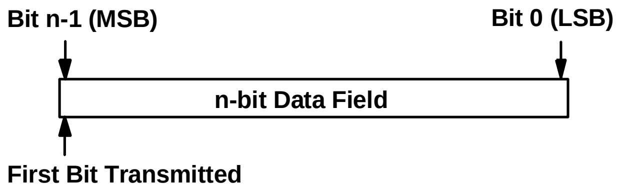


Figure 1: Bit numbering convention

3.3.2 Requirement identification

Requirements specified within this standard fall into one of three categories, namely mandatory, M, optional, O, or option dependent, D. Mandatory requirements must be implemented by all systems that comply with this standard. Optional requirements can be applied at the discretion of the implementer or at the insistence of the system specifying system requirements. Dependent requirements are requirements that must be applied when a particular optional requirement is selected.

This standard also includes recommendations, which do not bear the weight of requirements but should indicate the preferred implementation to be used.

Each requirement or recommendation in the standard has an associated letter indicating its type. Mandatory requirements are indicated by an (M), optional requirements by an (O), dependent requirements by a (D), and recommendations by an (R).

In order to standardise the way spacecraft CAN bus implementations are characterised, a compliance pro-forma is provided in Annex G. This lists all the requirements and recommendations in the standard, and allows requirement dependencies to be traced. By completing this pro-forma, the implementer not only states his compliance to the standard but also characterises his implementation by indicating which options have been selected.

4 Overall description (informative)

This standard contains recommendations for the use of the CAN (Controller Area Network) data bus in spacecraft onboard applications. These recommendations extend the CAN bus specification to cover aspects that are required to satisfy special needs that have been identified as being commonly required onboard spacecraft.

At the time of preparation of this standard there was limited experience with the use of the CAN bus as the principle onboard data bus for spacecraft applications. However, this limited experience has been taken fully into account during the development of these recommendations. By contrast, there is very extensive experience of the use of CAN bus in terrestrial applications, such as automobiles and factory process control. Often these applications have demanding safety and reliability requirements, and operate in hostile environments that have similarities to spacecraft onboard applications. This experience has also been taken fully into account in the preparation of this standard.

4.1 An overview of the CAN bus

The ISO11898 Part 1 standard specifies the Data Link Layer and Physical Signalling for CAN. Parts 2 and 3(draft) of ISO11898 specify 'high-speed' and 'low-speed' Medium Access Units for CAN respectively. The protocol specifications describe the data-link layer and physical layer requirements for CAN as illustrated in Figure 2. The remaining higher layer implementations have up until now been left to the designers' discretion and as illustrated a number of commercial specifications have evolved at the application level.

ISO / OSI Layer Model[1]	CAN Bus Layer			
Layer 8 Example User Applications	CANopen	CAN Kingdom	DeviceNet	Smart Distributed System (SDS)

ISO OSI Protocol Stack	Layer 7 Application Layer	CAL: CAN Application Layer & CANopen	CAN Kingdom	DeviceNet Specifications	SDS Specifications	
	Layer 6 Presentation Layer	Not explicitly defined				
	Layer 5 Session Layer	“Time-Triggered CAN” ISO 11898-4 (Draft)				
	Layer 4 Transport Layer	Not explicitly defined				
	Layer 3 Network Layer	Not explicitly defined				
	Layer 2 Data Link Layer[2]	ISO 11898-1 LLC: Logical Link Control Acceptance Filtering Overload Notification Recovery Management MAC: Medium Access Control Data En-/Decapsulation Frame Coding (De-/Stuffing) Medium Access Management Error Detection and Signalling Acknowledgement De-/Serialisation Specification Variations: CAN 2.0A, Standard CAN (11 bit identifier field) CAN 2.0B, Extended CAN (29 bit identifier field)				Defined ISO CAN Layers
	Layer 1 Physical Layer[3]	ISO 11898-1 Physical Signalling Bit En-/Decoding Bit Timing Synchronisation				
		“Low Speed CAN” ISO 11898-3 (Draft) (up to 125kbit/s)	“High-Speed CAN” ISO 11898-2 (up to 1Mbit/s)			

Layer 0 Transmission Medium	Some of the mediums on which CAN has been used: Twisted Pair, Shielded Twisted Pair, Single Wire, Fibre Optic, Co-Axial Cable, Radio Band, Infrared

Figure 2: CAN stack mapped to the ISO, OSI reference model [23]

1 A liberty has been taken in the inclusion of Layer 0 and Layer 8 which are not defined in the International Standards Organisation Open system Interconnection ISO 7498.
2 The Data Link Layer is specified in accordance with ISO 8802-2
3 The Physical Layer is specified in accordance with ISO 8802-3

Starting with CAN's physical layer the network topology is a line-drop configuration, whereby all nodes are connected by means of stubs to the network bus. The network medium itself is specified to be a pair of electrical wires. These two wires are referred to as 'CAN High' and 'CAN Low' as illustrated in Figure 3 and which show the physical signals on the lines for logic zero and logic one bit levels. The transceiver electrical configuration is such that the network medium performs a wired-and logical function, when two or more nodes try to drive a bit level on the network. This is why a logic zero is also referred to as a dominant level, and a logic one as a recessive level.

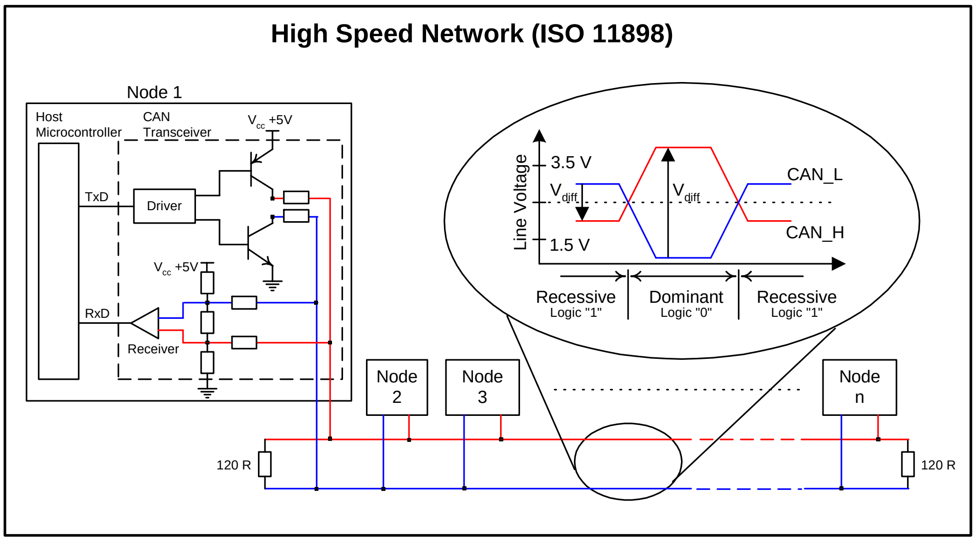


Figure 3: High speed CAN network [23]

Like many serial communications protocols, the CAN protocol transmits frames of data as a temporal sequence of bit time durations, whereby information is encoded by alternating the medium between two possible voltage levels. However, CAN also differentially encodes the Non-Return-to-Zero (NRZ) bit stream such that the electrical potential difference between the 'CAN High' and 'CAN Low' lines flips in polarity between the two alternative bits values. This technique contributes considerable additional tolerance to the low signal-to-noise ratios experienced in automotive and industrial applications.

The CAN protocol defines a number of fixed frame formats for the following messages types:

- Data Frames
- Remote Transmission Request Frames
- Active Error Frames
- Passive Error Frames
- Overload Frames

Figure 4 illustrates the data frame format for Standard and Extended CAN. The only difference between these two protocol variants lies in the control field portion of the frame. The significant difference exists in the length of the identifier sequence, being 11 bits in Standard CAN (version 2.0A, refer to [1]) and 29 bits in Extended CAN (version 2.0B, refer to [1]). The identifier field itself serves two significant functions, one being to reflect the content of the data frame and the other being to determine the priority of the frame during bus conflict situations.

CAN uses a content based addressing scheme, which supports both peer-to-peer and broadcast communication. The identifier field indicates the data type contained in the data frame and this is seen by all nodes attached to the network. If a node is interested in the data content as indicated by the identifier, it will process the data frame as appropriate.

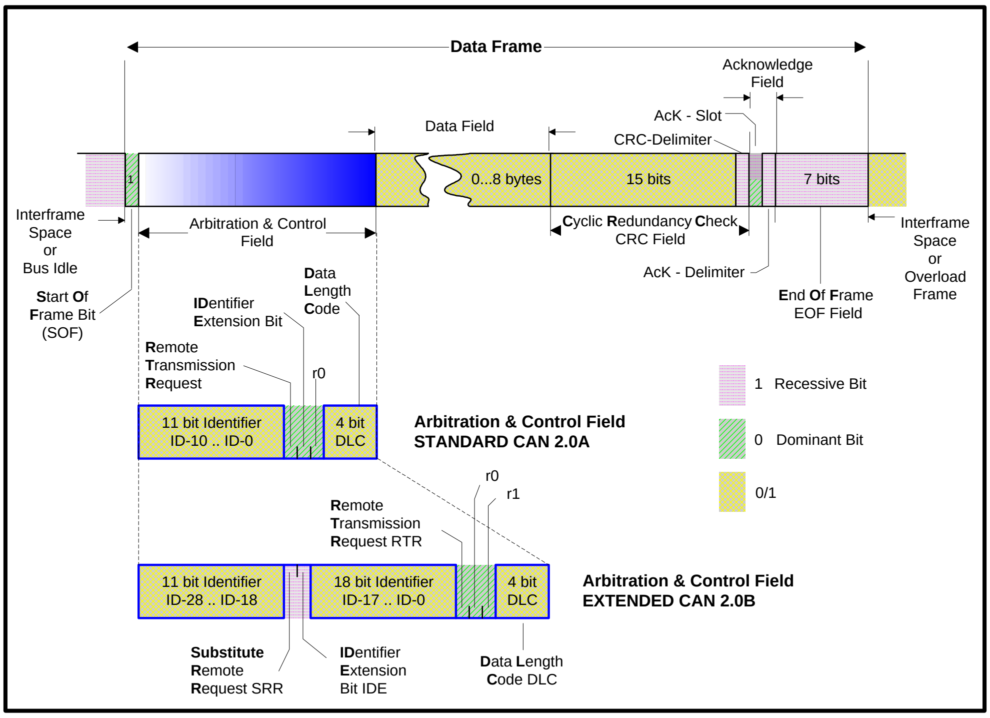

Data Frame
Acknowledge Field
AcK - Slot
CRC-Delimiter
Data Field
0...8 bytes
15 bits
7 bits
1
Interframe Space or Bus Idle
Arbitration & Control Field
Cyclic Redundancy Check CRC Field
Interframe Space or Overload Frame
AcK - Delimiter
End Of Frame EOF Field
Start Of Frame Bit (SOF)
IDentifier Extension Bit
Data Length Code
Remote Transmission Request
r0
11 bit Identifier ID-10 .. ID-0
4 bit DLC
Arbitration & Control Field STANDARD CAN 2.0A
1 Recessive Bit
0 Dominant Bit
0/1
r0
r1
Remote Transmission Request RTR
11 bit Identifier ID-28 .. ID-18
18 bit Identifier ID-17 .. ID-0
4 bit DLC
Arbitration & Control Field EXTENDED CAN 2.0B
Substitute Remote Request SRR
IDentifier Extension Bit IDE
Data Length Code DLC

Figure 4: Standard and Extended CAN frames [23]

The CAN protocol realises a multi-master architecture, whereby any node may arbitrarily transmit a message on the network, provided that the network is free at the time when the transmission commences. Unlike other protocols such as Ethernet, when two nodes on the network simultaneously transmit a message the messages are not destroyed. CAN uses a technique known as non-destructive bitwise arbitration to resolve such bus conflicts.

In this method, during the arbitration phase of message transmission, the value of the CAN frame identifier field determines which frame is allowed to transfer on the network and which frames 'yield-right-of-way' postponing their attempt to transmit. The lower the numeric value of the identifier field the higher the priority of the frame during the arbitration process.

If two or more nodes transmit a frame simultaneously the first to transmit a zero bit within its arbitration field, while another attempts to transmit a one, will win control over the network. The winning node will then complete its transmission while the unsuccessful node backs-off and attempts to re-transmit again when the network becomes idle. Figure 5 illustrates the process with an example.

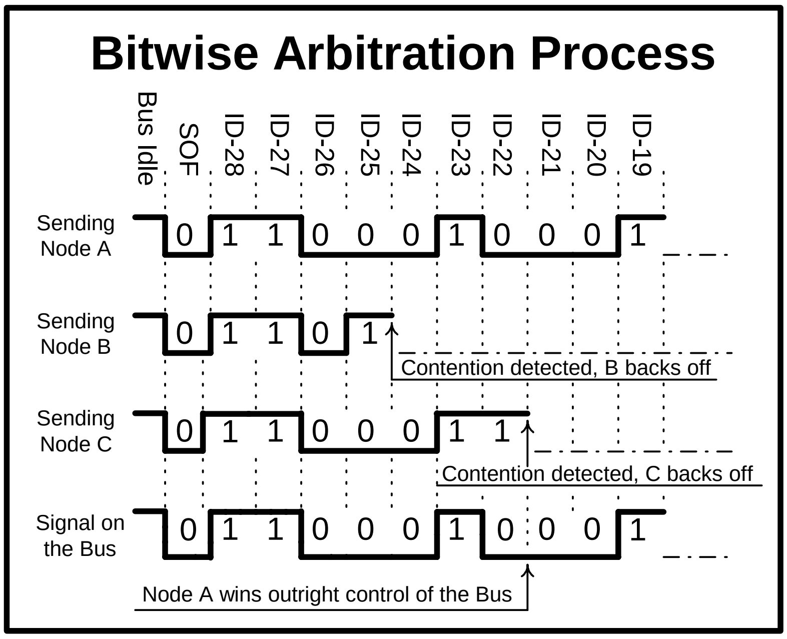


Figure 5: Network Arbitration [23]

4.2 An overview of spacecraft onboard data characteristics

The spacecraft onboard bus is used for three principal functions:

- The acquisition of data from simple sensors and the commanding of simple actuators,
- The transfer of packets of data between onboard instruments and control computers,
- The distribution of time information.

The ***acquisition of data from simple sensors and the commanding of simple actuators*** was the original role of the spacecraft onboard bus. Typically, the bus comprised a central controller with a number of remote terminals attached. Each remote terminal implemented a certain number of interfaces to simple sensors and actuators, such as thermistors and on/off switches.

To read a given sensor, the bus controller issued a command to the remote terminal to which that sensor was attached, and the remote terminal then read the sensor and transmitted the result back to the bus controller. Similarly, to write to a device, e.g. to operate a switch or relay, the bus controller issued a command to the appropriate remote terminal, which then wrote to the device itself.

The commands issued by the bus controller were generally very short, typically using only 16 bits, and responses from remote terminals were usually also short. Addressing and control information was needed in addition to the command and response data, but even with this, commands and their associated responses usually occupied 32 bits or less.

The need to ***transfer packets of data across the spacecraft bus*** has arisen as more capable microcontrollers and microprocessors become available for use in remote terminals. Firstly, this enables larger and more complex commands to be sent to the remote terminals, where they can then be decoded and may result in the acquisition of data from several sensors, or the execution of a series of actuator commands. Secondly, it becomes possible to run software applications in the remote terminals. These applications can perform simple automatic operations, such as the periodic acquisition and formatting of data from several sensors without waiting for a command from the bus controller.

Another growing requirement is for the initial programme loading of remote terminals, and occasional loading of configuration tables during operations.

Finally, the need for the ***distribution of time information*** is a consequence of the increasing autonomy and capabilities of the remote terminals, and the devolution of control functions to them. Such a devolved system can be considered as a set of independent, asynchronous processes. However, for operational purposes it is necessary that all of those processes can access a common, coherent time reference. One obvious need for this is in the time stamping of locally acquired data so that a complete event time-line can be reconstituted from the spacecraft telemetry. Another example is in the synchronisation of control actions, such as synchronisation of a spacecraft attitude control manoeuvre with the acquisition of an image.

Time distribution involves the transfer of time data with the appropriate precision, and the distribution of a time reference pulse or tick that indicates exactly when the time data is valid. While all onboard buses have the capability of transferring the time data, they vary considerably in their ability to transfer the reference pulse. If it is not possible to distribute the reference pulse with sufficient accuracy through the onboard bus, it becomes necessary to use an external means of transferring this pulse, e.g. by adding a dedicated time distribution bus, which increases the spacecraft harness.

4.2.1 Future Trends in Onboard Bus Use

As the capability of spacecraft microprocessors increases, there is a growing trend for distribution of the spacecraft control applications amongst remote terminals on the bus. This trend is seen both in payloads, which are increasingly autonomous and handle more of their own data processing, and in the spacecraft sub-systems such as the power management system. This leads to two main effects.

Firstly, there is an increase in the proportion of data packet traffic on the onboard bus. Secondly, as remote terminals become more “intelligent”, they expect better services from the onboard bus. In particular, they expect to be able to access the bus to transfer data packets *on demand*, and many modern software architectures are based on messaging capabilities, where applications communicate with each other variable length messages that are generated asynchronously.

Another trend is the steadily increasing volumes of data, especially from payloads. Any given bus has a limit to its data throughput capacity, making it more and more difficult to devise bus scheduling tables that accommodate all of the asynchronous bus traffic while meeting the deadlines required by synchronous transfers. One answer to this is to use more than one bus, or to use dedicated high speed links for high volume data paths. However, this introduces its own problems of bus control and management, particularly when data has to be transferred across more than one bus.

The steadily increasing “intelligence” in remote terminals, their demands for more comprehensive communication services, and the need to support more elaborate, multi-bus architectures, results in an increased use of higher level protocols across the onboard bus. This in turn may increase the volume of traffic, particularly asynchronous traffic, but more importantly emphasises the need for a symmetric medium access service to be provided by the onboard bus.

While there is a steady increase in the asynchronous packet traffic across spacecraft buses, there is no decrease in the number of simple sensors and actuators that must be serviced through the bus. In fact, there is clear evidence to suggest that the number of such simple devices is also steadily increasing.

4.2.2 Summary of Onboard Bus Requirements

Given the foregoing discussion, the requirements that must be met by the spacecraft onboard bus can be summarised as:

1. Ability to acquire data synchronously and in real time from sensors,
2. Ability to transmit commands synchronously and in real time to actuators,
3. Ability to transfer asynchronous data packets between nodes,
4. Provide a symmetric medium access control service to nodes (i.e. each node can access the bus on demand),
5. Provide accurate distribution of time data and time reference pulse.

4.3 Content of the standard

The normative part of this standard defines:

- The CAN bus physical layer specification for spacecraft applications,
- A generic higher layer protocol (CANopen) for use over CAN bus in spacecraft applications,
- Techniques for synchronous data transfers over CAN bus for real time control applications based on CANopen,
- Techniques for the transfer of large data units, e.g. packets, over CAN bus using a protocol complementary to CANopen,
- Time distribution over CAN bus for spacecraft applications based on CANopen,
- CAN bus frame identifier assignments,
- CAN bus redundancy management.

Each of these aspects is addressed in a separate normative section of the standard. The remainder of this clause provides brief introductory descriptions of each of these aspects.

In addition to these normative sections, annexes are provided describing

- Recommended connectors and pin assignments,
- Guidelines for implementing bus redundancy management,
- Minimalist implementation of CANopen,
- Process for adoption of COB-IDs,
- CAN system design issues,
- Physical layer design considerations,
- Compliance pro-forma.

4.4 The CAN bus physical layer specification

The CAN physical layer specification in this document aims to be suitable for the overwhelming majority of spacecraft missions. However, extreme mission environment conditions may require a specific mission-driven physical layer implementation that is not covered by this standard.

The physical layer specification inherits as much as possible from commercial industry, primarily the automotive and related industries.

The specification aims to ensure device-on-bus electrical compatibility and device-across-the-industry electrical compatibility by full adherence to ISO 11898-1:2003 and ISO 11898-2:2003 as the physical layer electrical reference requirement. This includes full compliance with ISO 11898-2 section 7.6, Bus Failure Management. 'Low- speed' CAN as specified by ISO 11898-3 Draft (previously by 11519-2) has not been considered in this specification.

Optionally, and in order to ensure the primary objective to be suitable for the majority of spacecraft missions, implementations of the physical layer based on RS-485 transceivers can be considered. Connecting these transceivers in a special way, it is possible to emulate the behaviour of ISO11898 transceivers, as it has already been the case in some space missions.

The physical layer defined in this standard includes specifications that can be met using off-the-shelf components that are either space qualified or likely to be qualifiable in terms of component performance and regarding radiation tolerance and other key space environment concerns.

The physical layer for spacecraft onboard applications is specified in clause 5.

4.5 CANopen higher layer protocol

The ISO 11898 standard specifies the CAN bus physical and data link layers. However, for the application layer there are several higher layer protocols defined as can be seen in Figure 2.

This standard recommends the use of CANopen, as basis for the higher layer communication protocol over the CAN bus.

CANopen is an open standard widely used in automation and industrial applications, including safety critical and maritime applications implementing redundant communication buses.

The most important part of a CANopen device is the Object Dictionary. It describes the data types, the communication functionality and the application data used in the device. It is also the interface between the application and the CAN bus.

CANopen defines standard communication mechanisms to transport data over the CAN bus by means of communication objects. These objects are defined in the CANopen communication profile area of the Object Dictionary and they provide a set of services for device configuration, data transfer, some special functions such as synchronization, time stamping, emergency notification and network management.

CANopen also specifies application data objects that are manipulated by the device's application program. These application objects are defined in the device profiles area of the Object Dictionary. They describe the default behavior and optional functionality of the devices.

A key feature of CANopen is the scalability. While the range of objects and services is broad the number of mandatory requirements in the CANopen standard is reasonably low and allows for simplified implementations in nodes not requiring the full CANopen capability.

4.5.1 Object dictionary

The object dictionary is essentially a grouping of objects accessible via the network in an ordered pre-defined fashion. Each object is addressed using an index and a sub index.

The object dictionary provides access to all data types used in the device; to the communication objects and the application objects.

The overall layout of the standard Object dictionary is summarized in Table 1.

Table 1: Organisation of the object dictionary

Index	Object	
0000	Not used	
0001-001F	Static data types	Data types
0020-003F	Complex data types	
0040-005F	Manufacturer specific data types	
0060-007F	Device profile static data types	
0080-009F	Device profile complex data types	
00A0-0FFF	Reserved	
1000-1FFF	Communication profile area	Communication parameters
2000-5FFF	Manufacturer specific profile area	Application data
6000-9FFF	Standardized profile area	
A000-FFFF	Reserved	

The communication profile is common to all devices connected to the network (with different values). It defines the communication objects and the associated parameters. The device profile part is specific to each device. It contains the used application objects.

The knowledge of the object dictionary of a device is sufficient to know the behavior of a device on the network.

4.5.2 Electronic Data Sheets

The description of the object dictionary of a device is provided in the form of an electronic data sheet (EDS), which is an ASCII file with a strictly defined syntax. The EDS is a standardized way to describe a device, it facilitates the exchange of information. It can be used in various configuration tools and helps for designing networks with CANopen devices. The use of EDS can be part of the COB-ID allocation process described in Annex D.

4.5.3 Communication objects

Two general categories of communication objects can be identified: the objects responsible for application data transfer and objects for network management.

Service data objects provide indexed access to all objects in a device via the object dictionary and process data objects provide direct access to application objects making it possible to implement real time data exchange mechanisms.

The network management communication objects are used to control the initialization and the communication state of nodes and they enable continuous supervision of the communication status of nodes.

4.5.3.1 Service Data Objects (SDO)

The CANopen Service Data Objects (SDOs) are used to access the object dictionary of a device. Since the object dictionary entries may contain data of arbitrary size and type, an SDO can be used to transfer data of any type (variables, programs, etc.) and any size. SDOs are especially used for the configuration of devices.

SDO communication follows a client/server model and is used to establish a peer-to-peer connection between two nodes. The node requesting the access to a remote object dictionary is the client and the owner of the accessed OD is the server. The client can request data download to the server, data upload from the server, and both can abort a transfer. The download/upload services are confirmed, which means client and server uses 2 dedicated CAN-messages (one for request, the other for reply).

When more than 7-octets of data are to be transferred, the data is segmented and transferred in several CAN frames. The data field of the SDO request and reply frames always contains 8 octets (even if it contains no meaningful data).

4.5.3.2 Process Data Objects (PDO)

The CANopen Process Data Objects (PDOs) are used for transmitting real time process data. PDOs transfer up to 8 octets of data without protocol overhead using unconfirmed service.

PDO communication follows a producer/consumer model. A PDO is transferred from a single device (producer) to one or more devices (consumers). The producer can request the unconfirmed Write PDO service to send data to consumer(s); consumer(s) can request the Read PDO service by issuing a Remote Transmit Request. The producer sends a Transmit PDO (TPDO) with pre-defined COB-ID and contents corresponding to those defined in the Receive PDO (RPDO) of one or more consumers.

The content of a PDO is precisely defined by means of the PDO mapping parameters. The mapping parameters indicate which application objects in the object dictionary are to be mapped into the PDO and what are their attributes (type, size in bit, etc.). This mapping can be defined statically or configured by means of SDO messages if the device supports “variable mapping” capability. Typically, the PDO can be used for the acquisition of values (e.g. thermistor values). For example, a PDO can be used to transfer 64 bits of digital data, or 4 analogue inputs of 16 bits each.

The communication behavior and priority of a PDO are defined by means of the PDO communication parameters in the Object Dictionary of the device. They define the COB-ID, transmission type and optionally a minimum time between two consecutive PDOs and an event timer. The transmission type parameter specifies the conditions under which the TPDO content is updated as well as the criteria for PDO transmission.

PDO transmission can be synchronized by means of a synchronization object or it can be event driven and thus asynchronous. The transmission type parameter specifies for RPDO the condition under which the received message is passed to the application. TPDOs and RPDOs have a number of transmission types summarized in Table 2.

Table 2: PDO transmission types

TPDO	
Transmission mode	Triggering conditions

Synchronous	Cyclic	**SYNC Object only**. At reception of the SYNC object the content of PDO is updated and it is sent.
	Acyclic	**SYNC Object + Event**. The content of the PDO is updated by the occurrence of a device specific event (e.g. a change of a value of a specific parameter) and the PDO is transmitted synchronously after reception of the next SYNC object.
	After RTR	**SYNC object + RTR**. The content of the PDO is updated at the reception of each SYNC object; and the PDO is transmitted only after reception of a remote transmit request.
Asynchronous	After RTR	**RTR only**. At the reception of a remote transmit request the content of the PDO is updated and it is transmitted.
	On event	**Event only**. The PDO content is updated and it is sent by the occurrence of a device specific event or a timer event.
RPDO		
Reception mode		Update condition
Synchronous		**SYNC object**. The data of a synchronous RPDO received after the occurrence of a SYNC is passed to the application at reception of the next SYNC message.
Asynchronous		**None**. The data of asynchronous RPDOs is passed directly to the application when received.

Figure 6 illustrates the cyclic transmission mode seen from the producer side. In this example the PDO$_1$ is defined as cyclic and PDO$_2$ is defined as cyclic with a period of two.

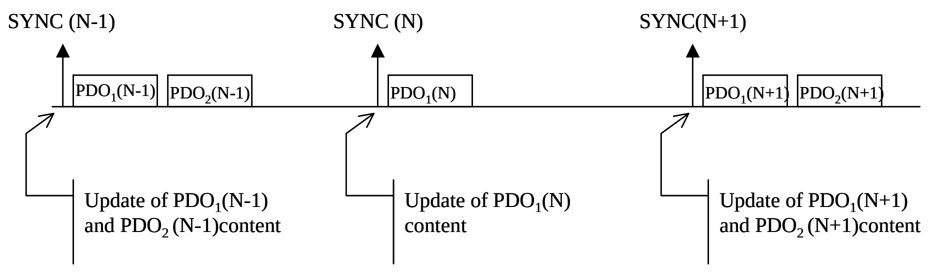

SYNC (N-1)
SYNC (N)
SYNC(N+1)
PDO1(N-1)
PDO2(N-1)
PDO1(N)
PDO1(N+1)
PDO2(N+1)
Update of PDO1(N-1) and PDO2 (N-1)content
Update of PDO1(N) content
Update of PDO1(N+1) and PDO2 (N+1)content

Figure 6: Cyclic transmission mode (Producer).

Figure 7 illustrates the synchronous and asynchronous reception modes seen from the consumer side. PDO$_1$ is defined as synchronous and PDO$_2$ asynchronous reception mode.

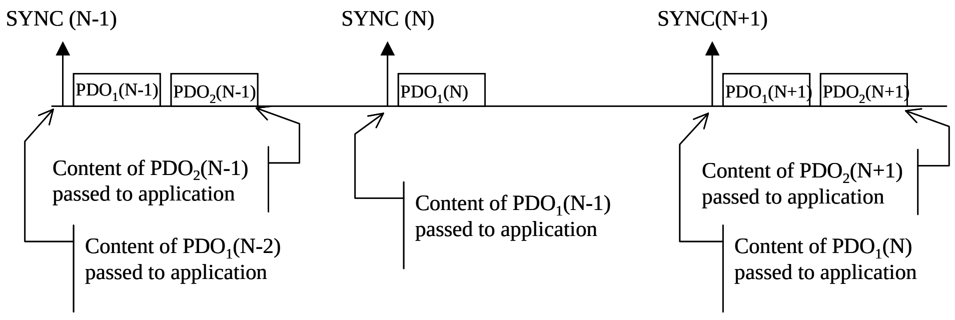

SYNC (N-1)
SYNC (N)
SYNC(N+1)
PDO$_1$(N-1)
PDO$_2$(N-1)
PDO$_1$(N)
PDO$_1$(N+1)
PDO$_2$(N+1)
Content of PDO$_2$(N-1) passed to application
Content of PDO$_1$(N-2) passed to application
Content of PDO$_1$(N-1) passed to application
Content of PDO$_2$(N+1) passed to application
Content of PDO$_1$(N) passed to application

Figure 7: Synchronous and asynchronous reception mode (Consumer).

Figure 8 illustrates the acyclic synchronous transmission of a PDO based on an application specific event.

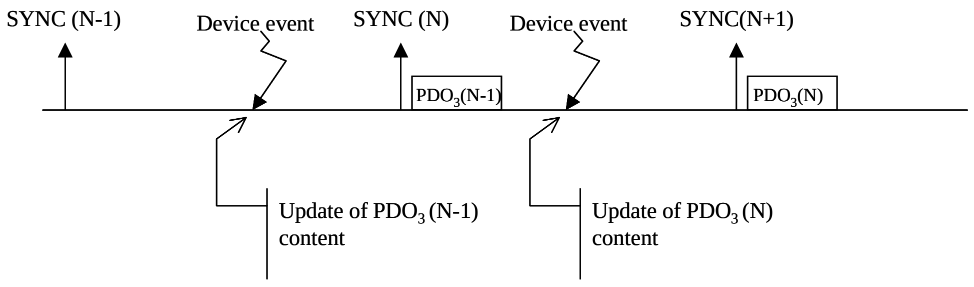


Figure 8: PDO with acyclic transmission mode (Producer)

Figure 9 illustrates the synchronous updating of the content of a PDO.

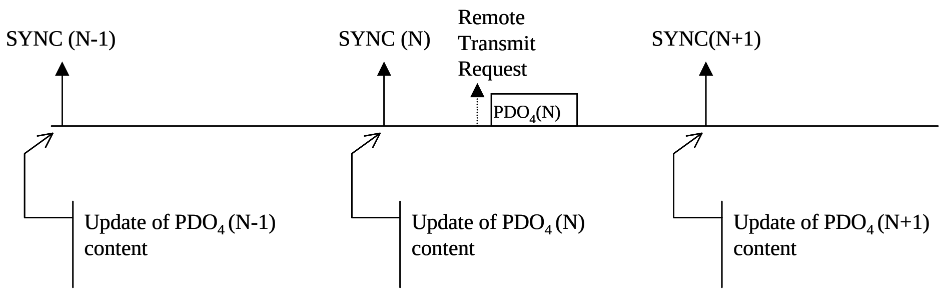

SYNC (N-1)
SYNC (N)
Remote Transmit Request
SYNC(N+1)
PDO$_4$(N)
Update of PDO$_4$ (N-1) content
Update of PDO$_4$ (N) content
Update of PDO$_4$ (N+1) content

Figure 9: PDO with synchronous RTR transmission mode (Producer)

4.5.3.3 Synchronization object (SYNC)

The CANopen SYNC object is used to synchronize devices connected to the network.

The SYNC object is periodically issued by a SYNC producer which provides the synchronization signal for the SYNC consumers. Upon reception of the signal the latter carry out their synchronous tasks. It could be application specific actions, or standard actions such as triggering of the transmission of PDO as shown in the previous section.

In order to guarantee timely access to the CAN bus the SYNC object is given a very high priority identifier. By default, the SYNC object does not carry any data.

There can be a time jitter in the transmission of the SYNC object. Typically the jitter would be caused by another message being transmitted on the bus just before the SYNC object. As such the uncertainty in the delay in the transmission of the SYNC object would be in the order of one CAN message. A time jitter may also be introduced in case of priority inversion.

The SYNC object does not provide any dedicated mechanism to allow for detection of double frames.

Priority inversion and double frames are further described in Annex E.

4.5.3.4 Emergency object (EMCY)

The CANopen emergency object is used to transmit information about the status of a device. A device can use emergency objects to notify other devices when it encounters an internal error situation.

The emergency object contains the content of an error register, the standardized error codes defined in the communication profile, as well as device specific error codes specified in device profile.

The emergency object is optional and if a node supports it, it must at least implement two error codes (Error Reset/No Error and Generic Error). A device supporting the emergency object will be in one of two states, Error Occurred or Error Free. Emergency messages will be transmitted when the node (re)enters either of the two states.

4.5.3.5 Network management objects (NMT)

The Network Management (NMT) Objects provide services for network initialization, control of the communication state of each node as well as error- and device status control.

The network management follows a master-slave structure. One device within the network fulfils the function of the NMT master, controlling all the slave nodes.

The CANopen NMT services include the services listed below. The applicability of these services are specified in the normative clause 6.

Module Control Services

The communication state of a node is based on the state diagram of a CANopen node and indicated in Figure 10. The state diagram defines the state of a node with regard to its communications capability. In Figure 10 the active communication objects are indicated for each state.

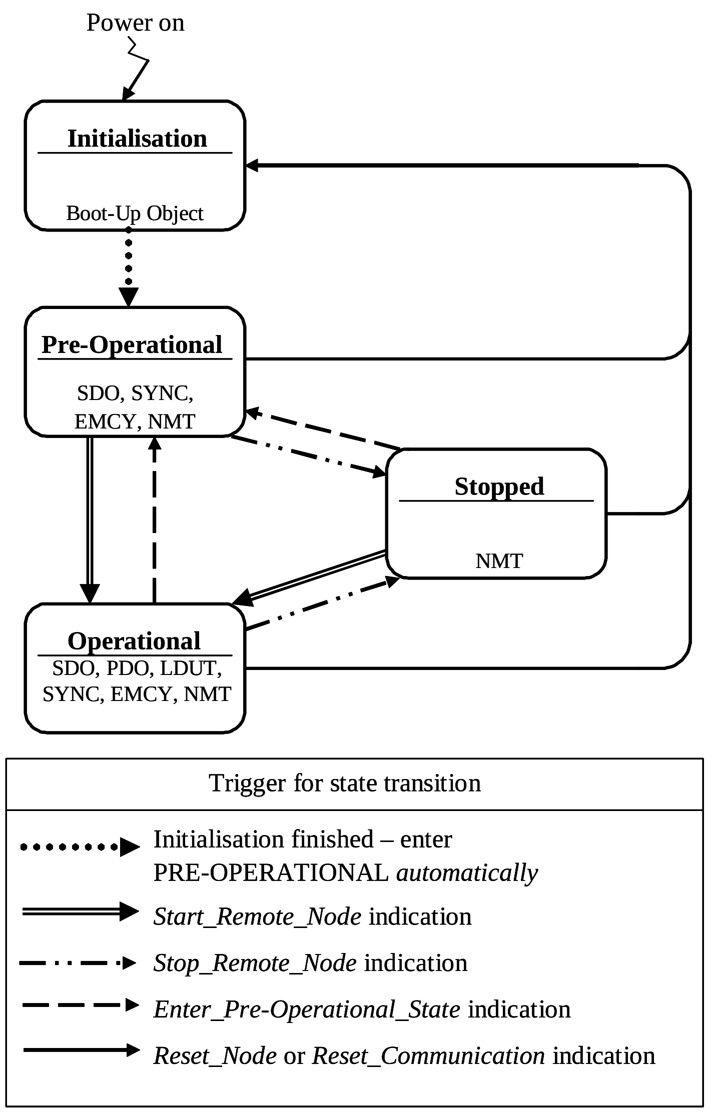

Power on
Initialisation
Boot-Up Object
Pre-Operational
SDO, SYNC, EMCY, NMT
Stopped
NMT
Operational
SDO, PDO, LDUT, SYNC, EMCY, NMT
Trigger for state transition
Initialisation finished – enter PRE-OPERATIONAL *automatically*
Start_Remote_Node indication
Stop_Remote_Node indication
Enter_Pre-Operational_State indication
Reset_Node or *Reset_Communication* indication

Figure 10: State Diagram of a CANopen device.

The state transitions are controlled by the NMT master by means of the Module Control Services: Start Remote Node, Stop Remote Node, Enter Pre-Operational state, Reset Node and Reset Communication.

Error Control Services

The Error Control Services provide mechanisms to detect failures in a CAN based network. The Error Control Services include Node Guarding, Life Guarding and Heartbeat services. The Node Guarding and Life Guarding services are based on requests issued by an NMT Master.

In Node Guarding, the NMT Master issues a request to an individual node and expects a reply within a certain time. In Life Guarding, a remote node expects guarding requests from the NMT Master within a certain time.

The Heartbeat service is not based on requests. Instead each node transmits a heartbeat message autonomously with a certain periodicity. One or several nodes on the network will listen to this heartbeat and take action if the heartbeat message is not received within the defined time. The Heartbeat message can be produced or consumed in all NMT states except the Initialisation.

Bootup Service

When a node enters the Pre-Operational state from the Initialising state, the node transmits a Bootup Event message to notify an NMT Master of the bootup.

4.5.4 Device profiles

A device profile defines the basic functionality every device within a device class must support. It also defines the way the functionality is made accessible through the CAN bus and what CANopen communication objects are needed to access this functionality.

Devices profiles are built on top of the CANopen communication profile. They specify supported application objects (input and output signals, device configuration parameters, device functions, etc.), additional data types, additional error codes, and default value of communication objects (default PDO mapping and communication parameters).

The exchangeability of devices from different manufacturers is supported by the specification of standard device profiles. Profile for Inputs/Outputs modules and encoders are two examples that are exiting in the industrial community.

Standardization gives an understandable and unique behavior of devices on the CAN network, enabling for example easy configuration of devices over the bus. There is also the possibility to define manufacturer specific device profile for manufacturer specific functions. For example considering an I/O device, the basic read analogue inputs function is defined in the I/O standardized device profile, and specific ADC configuration function can be defined in manufacturer specific device profile.

This standard does not specify device profiles. It is however recommended that the device profiles specified by the industrial community are considered when specifying and designing CAN devices.

4.6 Synchronous data transfers over CAN bus

4.6.1 General protocol for synchronous data transfers

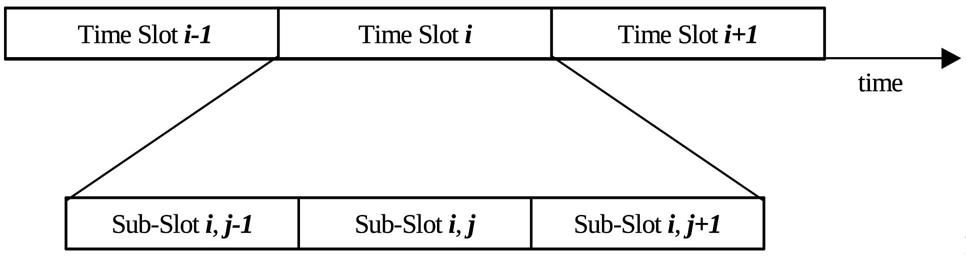
The command and control applications on spacecraft require the ability to perform synchronous data transfers. Synchronous transfers are those with rigorous timing constraints, and are typically used for periodic data acquisition from sensors and precisely timed commanding of actuators for real time control functions, e.g. attitude control, power management and thermal regulation and for periodic housekeeping acquisitions.

This standard specifies how to perform synchronous data transfers over CAN bus. The specified scheme uses the CANopen process data objects in conjunction with the SYNC object to achieve synchronous data transfers.

The SYNC object is produced periodically and the concerned nodes react upon the reception of the synchronizing message by issuing synchronous PDOs.

4.6.2 Communication slot organization

Some buses used onboard spacecraft, like MIL-STD-1553 or OBDH, use fixed timing structures to organize the bus traffic in communication slots. Activities to be performed within each slot time interval can then be pre-defined. The timing structure may be defined with slots and possibly sub-slots depending the timing granularity required as shown in Figure 11.



F

igure 11: Framing organisation.

Since the CAN bus is inherently asynchronous, there is no notion of slots. However, a synchronous slotting scheme can be implemented by means of CANopen communication objects.

If a slotted communication scheme is desirable the slot structure can be defined by the periodical transmission of SYNC objects. However, slot number information cannot be carried by the SYNC object because it conveys no data. Instead PDOs can be used to periodically broadcast the number of the upcoming slot before the reception of the next SYNC object.

While a detailed scheme for such communication is out of scope for this standard, it can be easily implemented using configurable features of CANopen.

4.7 Transfer of large data units

A common requirement on modern spacecraft is to transfer formatted data units such as packets or messages between onboard applications. Where the applications reside on different onboard nodes, these data units must be transferred across the onboard bus. Systems that implement CANopen can use the techniques defined within that standard to transfer large data units.

However, this standard specifies an alternative large data unit transfer technique that can be used without requiring CANopen segmented-SDO mechanisms to be implemented and can co-exist with CANopen on a single bus.

The independence of CANOpen and LDUT is illustrated in Figure 12.

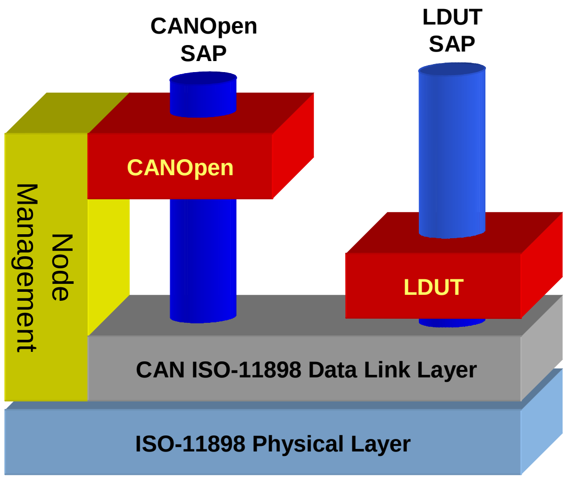


Figure 12: Relationship between CANOpen and LDUT

The CAN bus limits the maximum number of data octets to eight per frame, so any data unit larger than this must be segmented and transferred in several frames. Any data unit requiring segmentation is considered to be a ***large data unit***.

To make the transfer of large data units reliable requires the exchange of protocol control information that indicates the relationship of the segments making up the large data unit, and assists in the receiving process. This protocol control information must be encoded into CAN bus frames. The encoding technique that is proposed here makes use of the 29-bit identifier of the CAN 2.0B extended frame format, so that up to eight octets of segment data can be carried in each frame. The encoding scheme has been selected to enable this protocol to co-exist with CANopen on a single CAN bus.

The large data unit transfer scheme is defined in clause 8.

4.8 Time distribution

Virtually all space missions require that the spacecraft provide the capability to maintain and distribute time information on board. The time information is used for a variety of functions including time stamping of measurement data and control and scheduling of delayed execution of telecommands.

The on board time is implemented as a centralized time reference. However, there are typically many items of onboard equipment that also need to maintain local time information. A mechanism to distribute the central time and to keep the local times synchronized with the central time reference is therefore required.

This standard specifies two protocols that allows for time distribution and synchronization over the CAN bus. The accuracy of the time distribution and synchronization depends on the bit rate of the CAN bus as well as on the actual implementation of the protocols.

A time distribution protocol is specified that does not provide any special means to control the accuracy of the time distribution. As such the accuracy is application and implementation dependent. Accuracy in the millisecond range can be achieved using this protocol.

In addition, a high-resolution time distribution protocol is specified. The high-resolution time distribution protocol provides the best possible synchronization accuracy that can be achieved when distributing time over the CAN bus. Disregarding implementation uncertainties and delays the specified protocol allows for synchronization accuracy in the microsecond range. The high-resolution time distribution protocol is based on the CANopen High Resolution Synchronization protocol.

It should be noted that CANopen specifies two optional protocols and time formats for time distribution and synchronization using the so-called TIME object. Neither of these is directly suitable to be applied for space applications due to differences in the data types used for representing the time in CANopen compared to existing standards for spacecraft on board time formats.

The recommended time distribution techniques are defined in clause 9.

4.9 CAN object identifier assignments

The CAN object identifiers (COB-IDs) determines the priorities of the communication objects. The assignment of the COB-IDs to the different communication objects (CAN frames) has consequently a direct impact on the real- time characteristics of the bus communication.

All implementations of the CAN bus uses either 11-bit or 29-bit identifiers. 11- and 29-bit identifiers can be used simultaneously in the same system.

In order to allow for system optimizations and to minimize the impact of system modifications, it is important that all devices provide some possibility to modify the COB-IDs. This standard specifies the minimum requirements needed to allow for late modifications to COB-ID assignments.

CANopen specifies a mandatory pre-assignment of COB-IDs known as the pre-defined connection set. The purpose of this pre-assignment is to guarantee off-the shelf compatibility before system integration. At system integration time updated COB-IDs can be stored, e.g. in a non-volatile memory, in each device if the device supports this capability. This standard considers the use of the pre-defined connection set as optional, and proposes how to perform COB-ID assignment.

This standard also addresses how to perform COB-ID assignment for systems implementing two or more identical devices that are connected to the same physical bus and operates simultaneously in hot redundancy.

The requirements for identifier assignment are specified in clause 10.

4.10 Redundancy management

Spacecraft onboard communication buses are typically implemented using a redundant physical media topology that is resilient to single point failure on cabling or connectors faults.

Neither the CAN bus specification ISO 11898 nor the CANopen standard defined redundancy management at the moment of the preparation of this recommendation. However, both specifications define capabilities that are suitable when implementing a highly reliable bus system.

A redundant communication system requires redundant communication channels and also a redundancy management scheme. Components that may be redundant are the bus i.e. the physical connection, the bus transceiver and the bus controller.

This standard specifies a selective bus access architecture that allows a node to communicate on one bus at a time. In addition, a parallel bus access architecture is specified that allows a node simultaneous communication on both a nominal and a redundant bus.

A redundancy management scheme for a cold redundant bus system is specified compatible with the defined bus interface architectures. Redundancy management for hot redundant bus systems as well as node level redundancy management are very application specific and therefore outside of the scope of this document.

The redundancy management scheme defined in this standard uses the CANopen Heartbeat object as a means to monitor and control the bus system.

The redundancy management is discussed in clause 11. Additionally some implementation guidelines are included in Annex B.

5

Physical layer (normative)

5.1 Introduction

5.1.1 Scope

The CAN physical layer is comprised of the electrical, mechanical and performance characteristics of the cabling, connectors, termination resistors, CAN transceivers, and optional optical couplers that may be used in a spacecraft application.

The philosophy of this document is to make reference to ISO 11898-1 and ISO11898-2 wherever possible; detailing only the specific deviations or additions required to satisfy the requirements for spacecraft applications.

For convenience, the Data Link layer description is included in this document within the physical layer clause. The Data Link conforms to ISO 11898-1, which is based on the Bosch CAN 2.0B (1991) specification, [1].

5.2 Topology

5.2.1 Physical topology (R)

Two physical topologies are recommended:

- Linear multi-drop
- Daisy chain

Multi-drop devices, Figure 13, feature only one bus connector per CAN bus and are linked to the main bus via a drop cable stub.

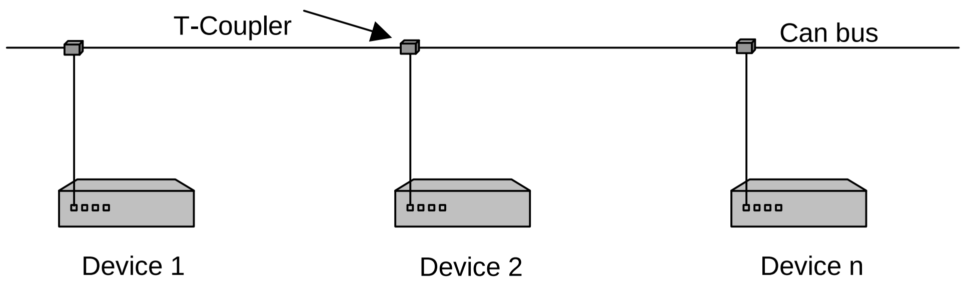


Figure 13: Linear Multi-drop Topology

Daisy chain devices, Figure 14, feature one input bus connector and one output bus connector and are connected in series.

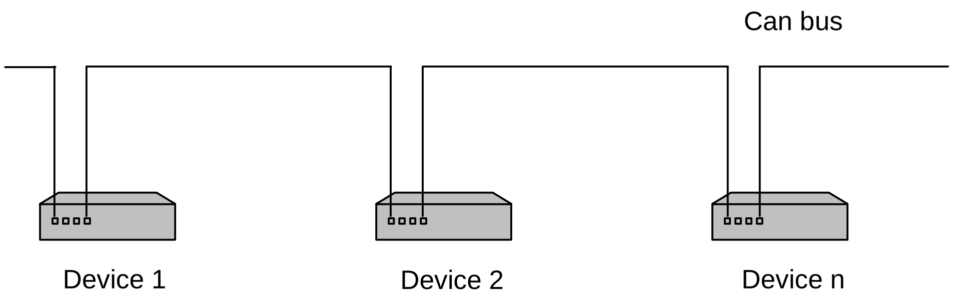


Figure 14: Daisy Chain Topology.

The two topology options can be employed exclusively or in combination.

a. Devices supporting the daisy-chain topology shall be capable of use in the multi-drop topology, utilising only the daisy chain input bus connector.

NOTE: It is recommended that equal cable lengths between devices are avoided to minimise standing waves. Similarly drop (stub) cable lengths should generally not be equal.

NOTE: Nominally, a single 120-ohm (or ideally, split – matched pair nominal 60-ohm resistors if employing a transceiver that supports a split bus configuration) terminating resistor should be installed at each extreme end of each bus segment.

NOTE: The precise values of terminating resistors should be calculated, based on the particular CAN bus implementation, length, and topology. It is recommended that the issues raised in [5] are considered.

5.2.2 Maximum bus length and drop length (R)

It is recommended that the maximum bus and drop lengths are implemented as per ISO 11898-2.

Additional considerations and requirements are addressed in 5.5.3.

5.2.3 Minimum number of network devices (R)

It is recommended that the CAN interfaces of each node is designed such that it can be incorporated in a system with at least 64 network nodes.

NOTE: The maximum number of network devices in a system is a function of the electrical properties of the devices and circuits of the CAN transceivers. Requirements for the CAN transceivers are specified in clause 5.4.

5.3 Medium

5.3.1 Cable requirements

5.3.1.1 CAN primary bus (M)

a. Each CAN node shall implement provision to carry one CAN bus in accordance with the physical medium specification of ISO 11898-2.

5.3.1.2 CAN redundant bus (M)

a. If implemented, the second (redundant) CAN bus shall also be carried in accordance with the physical medium specification of ISO11898-2.

5.3.1.3 CAN bus cable (M)

a. The CAN bus cable shall be compliant with ISO 11898-2.

NOTE: The specific CAN bus cable selection is left to the system engineer.

5.3.1.4 Shield – system specific (R)

a. It is recommended that all implementations employ shielded cables.

NOTE: The specific details of each implementation are left to the system designer to satisfy specific system requirements. However, the recommendation is to shield each CAN signal pair with individual shields. In most spacecraft applications it is desirable to include an overall cable shield.

5.3.2 Connector

5.3.2.1 Connector type – system specific (R)

It is not practical to define a connector to suit all applications. Therefore, this specification maintains a list of recommended connector configurations in Annex A. When a suitable connector from Annex A is selected, device-on-bus compatibility can be maintained by implementing the recommended configuration and signal pin-out for that connector.

5.3.2.2 Receptacles (M)

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

b. Receptacles shall be equipped with male contacts.

c. Receptacles with flying leads shall 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.

5.3.2.3 Plugs (M)

a. Plugs shall be used on cable assemblies.

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

   1. The conductors shall be directly soldered or crimped to the contacts.
   2. The overall shield 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.

5.3.2.4 Reserved pins (M)

a. Any pins designated as 'Reserved' in Annex A are not for use by system designers, as they may be assigned a specification in the future. They shall not be used for any purpose.

5.3.2.5 Bus terminators (M)

a. Bus terminators shall be implemented according to ISO11898-2.

b. Terminating resistors shall not be incorporated inside equipments.

NOTE 1: Terminating resistors may be incorporated inside a CAN bus cable harness or CAN bus cable connector.

NOTE 2: A termination resistor implemented outside the equipments eases tuning of the actual resistor value in a particular system configuration.

5.3.3 Shield Grounding – system specific (R)

a. There shall be a connection of the CAN signal shield at the digital ground of each device's interface circuitry.

NOTE: Each implementation will have specific requirements that shall be taken into account by the system designer.

5.4 Transceiver Characteristics

5.4.1 Transceiver electrical characteristics (M)

a. Transceivers shall satisfy the physical medium attachment sub-layer specification of ISO 11898-2.

5.4.2 Resistance to electrical CAN bus faults (R)

a. Full compliance with ISO 11898-2 section 7.6, Bus Failure Management, is required.

5.4.3 Optical isolation (R)

To minimize electrical disturbances it is recommended that an optical isolation of the CAN signal is implemented between the CAN transceiver and the CAN controller. For such implementations an isolated power supply should power the components on the bus-side of the optical couplers.

Figure 15 illustrates the principles of optical isolation. The detailed specification or design of the optical isolation is however outside the scope of this standard.

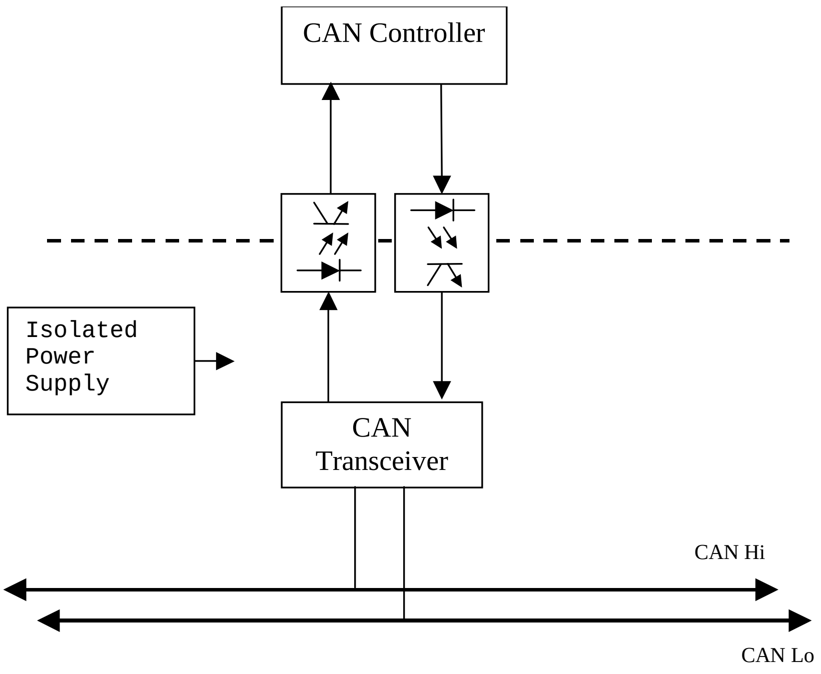


Figure 15: Principle of optically isolated bus interface

a. Equipments implementing an optically isolated bus interface shall comply with the specifications of this standard.

NOTE 1: This ensures that it is possible to connect both optically isolated and non-isolated nodes to the same network.

NOTE 2: Opto-isolators will impose an additional propagation delay that could restrict the maximum possible bus length. The design of the bus interface should thus be made such that the maximum round trip interface delay time for a device is still compliant with the bit timing requirements of section 5.5.

5.4.4 Transceiver implementation based on RS-485 transceivers (O)

Optionally, and in order to ensure the primary objective to be suitable for the majority of spacecraft missions, implementations of the physical layer based on RS-485 transceivers can be considered.

a. ISO11898 and RS-485 devices shall never be used on the same bus.

b. This choice shall be made on a project basis, only if ISO11898 compliant components do not fulfil the specific mission requirements.

NOTE: Please note that this option precludes the highly desirable “device-across-the-industry electrical compatibility” so this option should be avoided whenever possible.

5.5 Bit Timing

5.5.1 Bit rate 1 Mbps (M)

a. All devices shall support the 1 Mbps high-speed CAN bit rate.

NOTE: ISO 11898 specifies high-speed CAN as 125 Kbps to 1 Mbps.

5.5.2 Other bit rates (O)

A device may optionally support one or more other high-speed CAN rates.

a. If a device supports more bit rates than specified in clause 5.5.1 it shall support one or more of the following rates:
   - 500Kbps
   - 250Kbps
   - 125Kbps

b. A device may implement bit rates not specified in this standard if the device also implements all bit rates specified in this standard.

5.5.3 Bit Timing (M)

Bit timing parameters are specified to maintain compatibility with devices implementing parameters specified in the CANOpen specification. Further information about bit timing parameters are available in [5].

It is important that the oscillator or any other source used for deriving the nodes' bit timing support reliable 1Mbps data communication. The choices made are critical for stable, reliable and error-free CAN bus operation considering frequency drift due to: aging, temperature instability, jitter, etc.

a. For the 1Mbps option, the bit sample point shall be between 75% and 99% of the bit time.

NOTE: It is strongly recommended to use a sampling time of 80% or later.

b. For the 250 Kbps option, the bit sample point shall be between 88% and 99% of the bit time.

c. For the 500Kbps and 125Kbps options, the bit sample time shall be determined by consulting [5].

d. The synchronisation jump width shall be 1 time quanta.

e. Synchronisation shall be performed on 'recessive to dominant' edges only.

f. For the 1Mbps bit rate, each device's CAN interface round trip propagation time shall be less than 210ns.

g. For the 250Kbps option, each device's CAN interface round trip propagation time shall be less than 300ns.

h. For the 500Kbps and 125Kbps options, the round trip propagation time shall be determined by consulting [5].

5.6 Electromagnetic (EMC) Compatibility (R)

The specification of requirements for EMC is out of scope for this standard. The specification of EMC related requirements are the responsibility of the equipment and system designers.

ECSS-E-20A Space engineering, Electrical and electronic [21] specifies general requirements for EMC control.

5.7 Data Link Layer (M)

5.7.1 ISO 11898 Compliance (M)

a. The CAN Data Link layer shall comply with ISO 11898-1.

The ISO 11898-1 document inherits data link layer definitions from the Bosch specification, [1].

5.7.2 Fault confinement (M)

a. The Data Link layer shall implement fault confinement, as specified in ISO 11898-2 section 7.6, Fault Confinement.

The ISO 11898-2 document inherits fault confinement definitions from the Bosch specification, [1]. It is left to the system engineer how to best utilize the fault containment mechanisms provided for in section 7.6 of the ISO reference.

The ISO 11898-1, specifies that Bus Off and Normal Mode indications shall be provided to the application. In addition to these indications it is recommended that indication of Error Active and Error Passive are also provided to the application.

It is further recommended that implementers provide read access to the error counters to allow assessment of the quality of the bus.

6

CANopen higher layer protocol (normative)

6.1 General (M)

a. The implementation of the application layer protocol shall be according to CANopen.

NOTE: This is an overall general requirement. It shall be noted that this standard also specifies additional details or constraints in addition to those specified by CANopen. The following sections detail these requirements further.

6.2 Communication Objects

6.2.1 Service Data Objects (M)

a. All devices shall implement at least one server SDO.

b. The SDO shall be implemented as specified in CANopen.

NOTE 1: This is in order to ensure access to the device Object Dictionary. In systems only requiring configuration on ground, the Client SDO can be implemented in a PC or similar. SDOs could be used for configuration of the CAN nodes in-flight. This choice is left to the particular application and out of scope for this standard.

NOTE 2: An SDO can be used to transfer large data units. Optionally the Large Data Unit Transfer protocol specified in clause 8 can be used instead.

6.2.2 Process Data Objects (O)

CANopen restricts the number of different PDOs to 512 Transmit PDOs and 512 Receive PDOs per node. A device can implement an arbitrary number of PDOs up to the limit defined by the CANopen standard.

a. PDOs shall be implemented according to the CANopen specification.

NOTE: Optionally, variable PDO mapping and multiplexed PDOs can be implemented and used.

6.2.3 Synchronisation object (O)

a. All devices requiring synchronous communication shall implement a SYNC object.

NOTE: A device might be a SYNC producer or SYNC consumer depending on its functionality.

b. The SYNC object shall be implemented according to CANopen.

6.2.4 Emergency object (O)

CANopen specifies an optional emergency object. This standard specifies an optional use of the Emergency object within the bus redundancy management scheme. In addition the Emergency object can be used for other error reporting.

a. Error reporting shall be done by means of the Emergency object.

b. The Emergency object shall be implemented as specified in CANopen.

NOTE: CANopen specifies a number of error codes covering a wide range of error cases.

6.2.5 Network management objects (M)

CANopen specifies services for management of nodes (NMT) connected to the network. These services are based on communication objects for starting, resetting and stopping a node as well as for monitoring node status via the network.

6.2.5.1 Module Control Services (M)

a. All devices shall implement the following CANopen NMT Module Control Services:

- Start Remote Node
- Stop Remote Node
- Enter Pre-Operational
- Reset Node
- Reset Communication

b. The NMT Module Control Services shall be implemented as specified in CANopen.

6.2.5.2 Error Control Services (M)

CANopen specifies that it is mandatory to implement either the Node/Life Guarding or the Heartbeat services and protocols. This standard is more restrictive and specifies the Heartbeat service and protocol as mandatory.

a. All devices shall implement the Heartbeat service as specified in CANopen.

NOTE: This implies that, as a minimum, all nodes shall provide the capability to generate Heartbeat messages. In case the Heartbeat is used in a particular system, there need to be at least one Heartbeat consumer. It is left to the system designer to specify the need for a Heartbeat consumer.

b. All devices implementing the redundancy scheme defined in the clause 11 of this standard shall implement a Heartbeat consumer.

NOTE: The bus redundancy management scheme defined in this standard requires all the slave nodes to consume the redundancy master Heartbeat message.

6.2.5.3 Bootup Service (M)

a. All devices shall implement the Bootup service as specified in CANopen.

6.2.5.4 Node state diagram (M)

a. All devices shall implement the NMT state diagram as specified in CANopen.

NOTE: The NMT state diagram is further indicated in Figure 10 (page 27).

6.3 Electronic Data Sheets (M)

a. For each device, the device manufacturer shall provide an Electronic Data Sheet in line with the CANopen standard.

6.4 Device & Application Profiles (O)

The detailed definition of CANopen Device and Application profiles is outside of the scope for this standard.

a. When a device profile is to be defined, the device profile designer shall coordinate the allocation and definition of the profile with CiA for inclusion in their database.

NOTE: This is in order to ensure that any device profile defined within the space community is compatible with the CANopen device profile requirements.

b. Each device that implements a built-in self-test functionality shall provide the means to start the test by means of access to an object in the device object dictionary.

c. The result of the test shall be accessible in a dedicated object.

6.5 Object Dictionary (M)

a. All devices shall implement an object dictionary according to CANopen.

7 Synchronous data transfers (normative)

7.1 Synchronous communications (O)

a. Synchronous data transfers shall be performed using synchronous PDOs.

NOTE: Synchronous PDOs's transfers are synchronized by receipt of SYNC object.

b. The SYNC producer shall be capable of sending the SYNC object according to CANOpen.

NOTE: One device within the network acts as SYNC producer by issuing periodically the SYNC object.

c. SYNC consumers implementing the SYNC capability shall react on SYNC object's reception according to CANOpen.

8 Transfer of large data units (normative)

8.1 Introduction

In order to support the transfer of large data units, i.e. data units that consist of more than eight octets, it is necessary to implement a frame spanning mechanism over the basic CAN bus. CANOpen provides SDOs to perform large data unit transfers, but for nodes that do not implement the CANOpen segmented-SDO mechanism, the Large Data Unit Transfer (LDUT) protocol specified in this section can be used instead.

Frame spanning allows a sending node to segment a large data unit and transmit it as a series of CAN frames. The receiving node can then reassemble the large data unit from these frames.

This protocol uses the extended COB-ID of the CAN 2.0B frame format to convey protocol control information, thereby enabling a full cargo of 8-octets to be transferred in every segment. The protocol preserves the inherent priority based scheduling of CAN bus and combines this with node addressing to support traditional messaging requirements. The protocol engines required for the proposed protocol could be implemented in hardware or software[4].

The protocol can co-exist with CANopen on a single CAN bus network, provided that the assignment of COB-IDs is correctly profiled to allow this. The protocol uses a single CANopen PDO pre-defined connection set function code value that must not be used by any CANopen node. In general, if this protocol is used, it must be ensured that no COB-ID values that could conflict with this protocol are assigned for other purposes.

8.2 Identifier encoding (D)

a. The elements of protocol control information that shall be conveyed are:
   - The function ID,
   - The priority indicator,
   - The frame type,
   - The source address,

4 Software implementations of CAN bus protocols will always be relatively demanding of processor resources because of the small size of the CAN frame. However, prototype implementations in software are none the less feasible.

- The destination address,
- The Protocol ID,
- A Toggle Bit.

b. Each segment of the data unit shall be transmitted in sequence.

c. The encoding of the protocol control information in the 29-bit COB-ID field shall be as shown in Figure 16.

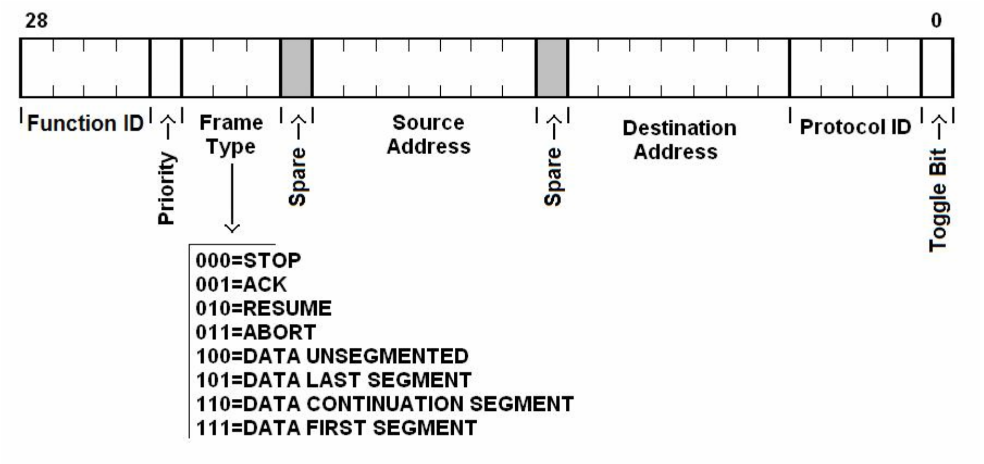


Figure 16: LDUT COB-ID encoding

8.2.1 Function ID field (D)

The *function ID* field is the most significant field for bus medium arbitration. Therefore all messages transferred using this protocol will have a well defined priority with respect to any CANopen services being operated on the bus.

a. The value assigned to the *function ID* field shall, on a given CAN network, be unique to the large data unit transfer protocol and not used for any other purpose.

NOTE 1: This ensures that there will be no other frames on the CAN network that will have the same COB-IDs as the large data unit transfer protocol.

NOTE 3: It is possible to assign more than one value to the *function ID* field thus implementing more than one large data unit transfer service.

8.2.2 Priority field (D)

The 1-bit *priority* field is the next most significant bit in bus medium arbitration, and the value used in this field therefore determines the priority of handling of the frame within this service. This bit allows selecting between expedited and non-expedited data transfer.

- '0': Expedited
- '1': Non-Expedited

8.2.3 Frame Type Field (D)

The frame type field permits to distinguish among the different types of frames used by the LDUT protocol.

The protocol control frames types are detailed in section 8.3.

Using different encoding for first, continuation, last and unsegmented data provides a fully delimited transfer protocol. This enables unsegmented data to be interleaved with segmented data from a given source node.

The use of a fully delimited protocol implies also very simple protocol engines at both the sending and receiving ends of a transfer while allowing a variety of transfer errors to be detected.

a. The frame type field encoding shall be as defined in Figure 16.

8.2.4 Source address field (D)

a. The 7-bit *source address* field shall indicate the source of the data segment.

b. The 7-bit *source address* fields shall correspond to the CANopen node ID.

NOTE: Carrying the source node identifier in the protocol control information allows a receiving node to distinguish between sequences of segments simultaneously arriving from different nodes on the network.

8.2.5 Destination address field (D)

a. The 7-bit *destination address* field shall indicate the destination of the data segment.

b. The 7-bit *destination address* fields shall correspond to the CANopen node ID.

8.2.6 Protocol ID field (D)

The 4-bit *Protocol ID* field serves to label the content of the Large Data Unit Transfer in order to distinguish among 16 different types of service data to be carried over LDUT. The actual definition of the meaning of these bits is left for the upper layers.

8.2.7 Toggle Bit field (D)

This toggling bit mechanism allows the receiver to detect missing and duplicated segments.

a. This bit shall be toggled every two consecutive messages of the same LDUT transmission.

NOTE: By using the segment count field it is possible to detect if double frames are received.

8.3 Protocol control frames (O)

Optionally, the large data unit transfer protocol can use a simple stop/resume/abort mechanism to implement flow control and error recovery in order to provide a reliable data transfer service at the data link layer.

The extremely low probability of a frame being dropped over the CAN bus, coupled with the fact that higher layer spacecraft protocols implement acknowledgement and retry services, mean that there is no requirement to acknowledge the receipt of each segment. Therefore this protocol does not provide for it.

It shall be noted that this standard does not specify response/reaction times for the protocol control frames. Care must thus be taken to ensure that a fast frame producer could not send too many frames to a consumer before the producer receives a control frame to stop transmitting.

8.3.1 Acknowledge frame (O)

a. Once a complete data unit has been successfully received, the receiver shall transmit an acknowledgement frame to the sender

NOTE: This enables the sender to clear its transmit buffer for this transfer.

b. The acknowledge frame format shall be according to Figure 16, with the ACK frame type.

8.3.2 Stop frame (O)

a. If the receiver detects that it cannot receive the complete data unit, it shall transmit a stop frame back to the sender.

NOTE: This could happen e.g. due to limited buffer space, or detection of an error in the sequencing of segments.

b. The sender shall stop transmitting segments for this transfer on receipt of the stop frame.

c. The format of the stop frame shall be according to Figure 16 with the STOP frame type.

8.3.3 Resume frame (O)

a. When the receiver is able to accept more segments, or to recover from a sequencing error, it shall transmit a resume frame back to the sender.

b. The resume frame shall indicate in the segment could field the segment number the sender should resume from.

c. The resume frame format shall be according to Figure 16 with the RESUME frame type.

NOTE: This standard does not specify retry policy issues, such as the number of retries that shall be attempted, or whether time-outs are used at the sending end.

8.3.4 Abort frame (O)

a. In the event that a receiver cannot receive the data unit, or wants to stop the sender from transmitting further segments for this transaction, it shall transmit an abort frame back to the sender.

b. On receipt of an abort frame, the sender shall clear its transmit buffer for this data unit and notify the user that the data unit could not be transferred.

c. The format of the abort frame shall be according to Figure 16 with the ABORT frame type.

8.4 Selective acknowledgement for unsegmented transfers (O)

For unsegmented transfers, i.e. for data units of less than 8 octets, the protocol provides a selective acknowledgement mechanism as follows:

a. If the unsegmented transfer is to be acknowledged, the sender shall set all bits in the segment count field to 1.

b. On receipt of an unsegmented data transfer with all bits in the segment count field set to one, the receiver shall transmit an acknowledge frame back to the sender.

c. If the unsegmented transfer is not to be acknowledged (default), the sender shall set all bits in the segment count field to 0.

d. On receipt of an unsegmented data transfer with all bits in the segment count field set to zero, the receiver shall not transmit an acknowledge frame.

9 Time distribution (normative)

9.1 Time objects (O)

9.1.1 Time code formats (D)

The definition of the Spacecraft Elapsed Time (SCET) and the corresponding absolute Spacecraft Universal Time Coordinated (UTC) is addressed in PSS-04-106, Issue 1 [22].

The time code format of the SCET is the CCSDS Unsegmented Time Code (CUC) format. The CUC is an unsegmented binary count of seconds and binary powers of sub-seconds. The SCET is thus a free running counter with a MSB of up to 2^{32} seconds and LSB sub-second representations down to 2^{-8}, 2^{-16} or 2^{-24}.

If the spacecraft provides the optional service of maintaining the UTC on board, the format of the UTC shall be that of the CCSDS Day Segmented time code (CDS). The CDS is a 16 or 24 bit binary representation of number of days elapsed from a predefined epoch, 32 bits represent the number of ms and an optional 16 or 32 bit field represents the sub-milliseconds.

This standard allows for either of the time code formats to be used as long as all devices in the network support the selected time code format. There are however, some limitations in terms of resolution and size of the fields.

a. Each device on the network that maintains time information shall comply with the time object specifications in this section.

NOTE: It is not permitted to use both the Spacecraft elapsed time objects defined in 9.1.2 and the Spacecraft universal time coordinated (UTC) objects defined 9.1.3 on the same CAN bus.

9.1.2 Spacecraft elapsed time objects (D)

a. The Spacecraft elapsed time objects (SCET) shall be implemented in all devices that maintain time information.

b. Each device shall implement one *Local SCET Set* and one *Local SCET Get* object in the device Object Dictionary.

c. The *Local SCET Set* object shall allow setting the local time of the node via the CAN bus.

d. The *Local SCET Get* object shall allow reading the local time of the node via the CAN bus.

e. The *Local SCET Set* and *Local SCET Get* objects shall be of a compound data type according to the following SCET definition:

STRUCT OF

Unsigned 32	Coarse Time
Unsigned 24	Fine Time (sub seconds)

"SCET"

f. The *Local SCET Set* and *Local SCET Get* objects shall be mapped into the CAN frame according to Figure 17.

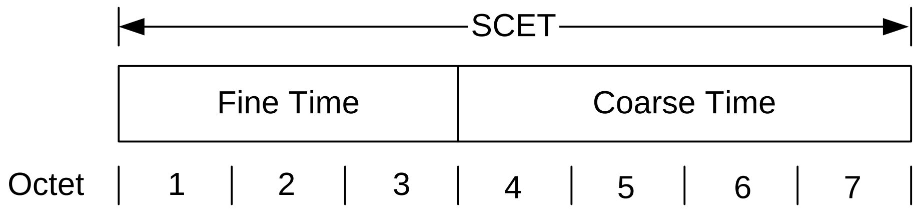


Figure 17: Format for objects containing the SCET

It is up to the implementation to decide the implemented size of the Fine Time counter in a particular device.

g. If a device supports less than 24 bits of fine time, the unused least significant bits shall be set to zeroes whenever the SCET is transmitted from a device

h. If a device supports less than 24 bits of fine time, the unused least significant bits shall be interpreted as don't-care whenever a device receives the SCET.

9.1.3 Spacecraft universal time coordinated objects (D)

a. The Spacecraft universal time coordinated (UTC) objects are optional.

b. Each device supporting UTC shall implement one *Local UTC Set* and one *Local UTC Get* object in the device Object Dictionary.

c. The *Local UTC Set* object shall allow setting the local time of the node via the CAN bus.

d. The *Local UTC Get* object shall allow reading the local time of the node via the CAN bus.

e. The *Local UTC Set* and *Local UTC Get* objects shall be of a compound data type according to the following UTC definition:

STRUCT OF

Unsigned 16	Day
Unsigned 32	ms of day
Unsigned 16	submilliseconds of ms

"UTC"

f. The *Local UTC Set* and *Local UTC Get* objects shall be mapped into the CAN frame according to Figure 18.

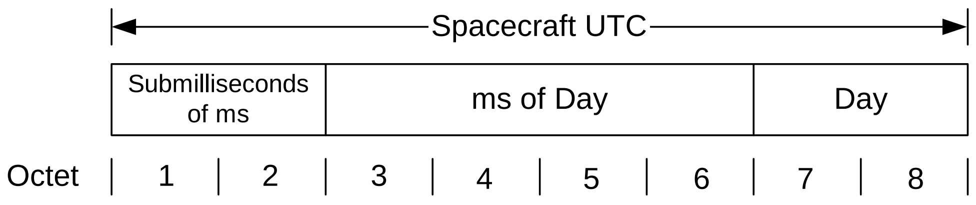


Figure 18: Format for objects containing the Spacecraft UTC

It is up to the implementation to decide if the "submilliseconds of ms" field is used in a particular device.

g. If a device does not support the "submilliseconds of ms" field, the unused least significant bits shall be set to zeroes whenever the time object is transmitted from a device.

h. If a device does not support the "submilliseconds of ms" field, the unused least significant bits shall be interpreted as don't-care whenever a device receives the time object.

9.2 Time distribution and synchronization protocols (O)

The time distribution and synchronization protocols specified below distributes a sample of the central time to devices maintaining a local time.

The time distribution protocol provides a mechanism to transfer and read back the time with no specific requirements on accuracy. The optional high-resolution time distribution protocol allows for the best possible time synchronization via the CAN bus. It is possible to use both protocols on the same CAN bus.

This standard makes no assumptions on the relationship or maximum time error between the central time and the local time. In particular, the frequency of local time updating and any associated mechanisms for adjusting the local time without loss or duplication of time codes are device and application specific and outside the scope of this standard.

a. Each device on the network that maintains time information shall be compliant to the Time distribution protocol (9.2.1) specification in this section.

9.2.1 Time distribution protocol (D)

This protocol is based on a single PDO transmission from the Time producer to one or several Time consumers. Read-back of the local times are facilitated by means of dedicated PDOs. There are no specific features in this protocol to control the accuracy of the time distribution.

a. The Time Producer shall map the *Local SCET Get* object (9.1.2) or the *Local UTC Get* object (9.1.3) to a dedicated *Spacecraft Time PDO* transmit PDO.

b. The Time Producer shall use the *Spacecraft Time PDO* to convey its local time to the time consumers.

c. The Time Consumers shall map the *Local SCET Set* or *Local UTC Set* objects to the *Spacecraft Time PDO* receive PDO.

d. There shall be only one *Spacecraft Time PDO* in a particular system.

e. Each Time consumer shall map the *Local SCET Get* object (9.1.2) or the *Local UTC Get* object (9.1.3) to a dedicated *Local Time PDO.*

f. The Time consumers shall use its *Local Time PDO* to convey its local time on the CAN bus.

g. There shall be one *Local Time PDO* for each node maintaining local time.

NOTE 1: The various transmission types defined in CANopen makes it possible to use a number of different methods to transmit the Spacecraft Time PDO and Local Time PDOs. However, the details are application specific and out of scope for this standard.

NOTE 2: If there is a need to distribute time during pre-operational state (before entering the operational state) it can be done by updating the time objects in the OD of the slaves by means of SDOs.

9.2.2 High-resolution time distribution protocol (O)

This protocol uses the SYNC object to achieve the highest possible synchronization accuracy when distributing time over the CAN bus. Read-back of the local times are facilitated by means of dedicated PDOs. The actual accuracy of the time distribution is implementation dependent. Accuracy in the microsecond range can be achieved with a bit rate of 1 Mbps.

a. The Time Producer shall map the *Local SCET Get* object (9.1.2) or the *Local UTC Get* object (9.1.3) to a dedicated *Spacecraft Time PDO* transmit PDO.

b. The Time Producer shall use the *Spacecraft Time PDO* to convey its local time to the time consumers.

c. The value of the *Spacecraft Time PDO* shall correspond to the spacecraft time when the last preceding SYNC object was successfully transmitted.

d. The High Resolution Time Distribution protocol shall be implemented according to the CANopen High Resolution Synchronisation Protocol with the following exceptions:

- The SYNC Producer and the Time Producer can be implemented as separate entities.

  NOTE: The text in $9.3.2 in the CANopen specification implies that the SYNC producer is identical to the Time Producer. This standard does not impose such limitations.

- In cases where the SYNC Producer and the Time Producer are separate entities the Time Producer shall implement the *Spacecraft Time PDO* with a transmission type of 1-240.

  NOTE: Transmission types 1-240 imply that the Time Producer will sample its local time on successful receipt of the SYNC object and thereafter send the *Spacecraft Time PDO*.

- The *Spacecraft Time PDO* defined in a. above shall be used for conveying the time information, i.e. the Time object as specified in CANopen shall not be used.

e. The Time Consumers shall map the *Local SCET Set* or *Local UTC Set* objects to the *Spacecraft Time PDO* receive PDO.

e. Each Time consumer shall map the *Local SCET Get* object (9.1.2) or the *Local UTC Get* object (9.1.3) to a dedicated *Local Time PDO* transmit PDO.

f. Each Time consumer shall use its *Local Time PDO* to convey its local time on the CAN bus.

g. There shall be one *Local Time PDO* for each node maintaining local time.

h. The value of the *Local Time PDO* shall correspond to the local time when the last preceding SYNC object was successfully received.

The detailed mechanisms for transmission of the *Local Time PDOs* are out of scope for this standard.

It is left to the implementer to define if the sampling of the local time at the occurrence of the SYNC object are to be implemented in H/W or S/W in order to achieve the required accuracy. It is nevertheless recommended that generic H/W implementations always provide the possibility to signal, e.g. by means of an interrupt that the SYNC object has been successfully transmitted/received.

10 CAN bus object identifier assignments (normative)

10.1 CAN bus version (M)

a. The implementation of the CAN bus shall be compliant to ISO 11898-1:2003.

NOTE: This implies that both 11-bit and 29-bit identifiers could be used on the CAN bus

10.2 COB-ID assignment (M)

a. All devices shall allow COB-ID assignment as specified by CANopen.

b. COB-IDs restricted for a specific purpose in CANopen must not be used other than for that particular purpose.

c. It shall be possible to change any non-restricted COB-ID of a device at any time up to completion of system level test without need for a re-qualification of the device.

NOTE: The COB-ID reassignment capability can be implemented by using SDOs but also other means such as PROM boxes, straps in connectors, S/W updates or similar could be considered if acceptable for the particular application.

d. Each node connected on the CAN bus shall have a unique CANopen Node ID.

NOTE: The CANopen Node ID is used by the NMT services.

e. Each hot redundant node connected to the same physical CAN bus shall use different COB-IDs.

NOTE: In cases where two or more hot redundant nodes are connected and actively operates on the same physical CAN bus, the communication objects of each redundant node must be uniquely identifiable.

f. For each device, the default COB-ID allocation to be implemented before delivery of the device shall be agreed between the customer and the supplier.

NOTE: The COB-ID assignment upon device delivery is thus not restricted to the CANopen pre-defined connection set. A recommended flow for the COB-ID adoption process is presented in Annex D.

11 Redundancy Management (normative)

11.1 General (O)

This clause specifies the implementation and protocol requirements related to redundancy management at bus level. The redundancy management scheme is based on the assumption that there is always one dedicated node responsible for the bus redundancy management. In the following this node is denoted Redundancy Master. The nodes not managing the bus redundancy are denoted Slave nodes.

a. In any system implementing bus redundancy there shall be one and only one active node assigned to be the Redundancy master.

b. The Redundancy Master shall be identical to the CANopen NMT master.

11.2 Node internal bus redundancy architectures (D)

This standard specifies two alternative architectures for the implementation of the bus interfaces in the CAN nodes.

The selective bus access architecture allows communication on only one bus at a time whereas the parallel bus access architecture allows simultaneous communication on both a nominal and a redundant bus. Both architectures can be used simultaneously in the same system.

a. Each CAN node interfacing redundant CAN buses shall be implemented according to one of the two schemes specified in this section.

b. The Redundancy Master node shall implement the parallel bus access architecture as specified in 11.2.2.

NOTE: This is to allow the bus Redundancy Master to passively listen for messages on the redundant (not active) bus.

11.2.1 Selective bus access architecture (D)

The selective bus access architecture implements a single CAN controller and interfaces the redundant CAN bus via two transceivers as illustrated in Figure 19. A bus selection mechanism is implemented between the CAN controller and the transceivers allowing the application to select the bus to be used for communication. This architecture allows communication on only one of the two buses at a time.

a. Nodes implementing the selective bus access architecture shall provide a mechanism allowing the application to select which bus to use.

NOTE 1: The details of this selection mechanism is implementation specific and out of scope for this standard.

NOTE 2: This scheme is not recommended for hot redundant bus implementations since only one bus can be active at a time.

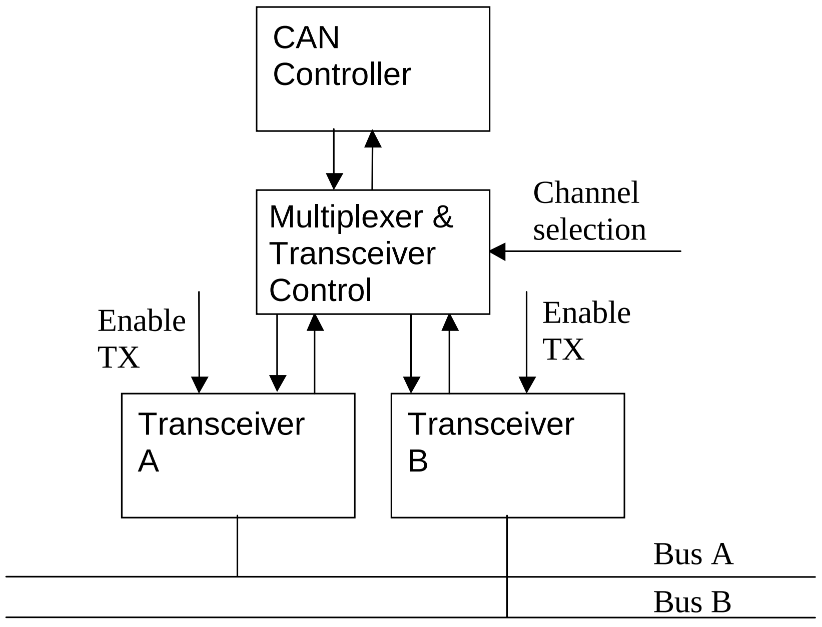


Figure 19: Selective bus access architecture

11.2.2 Parallel bus access architecture (D)

The parallel bus access architecture interfaces a redundant CAN bus through a pair of CAN controllers as illustrated in Figure 20. This implies that each CAN bus is fully and independently accessible via a dedicated CAN controller. The detailed mechanisms for communication on the two buses are highly application specific and out of scope for this standard.

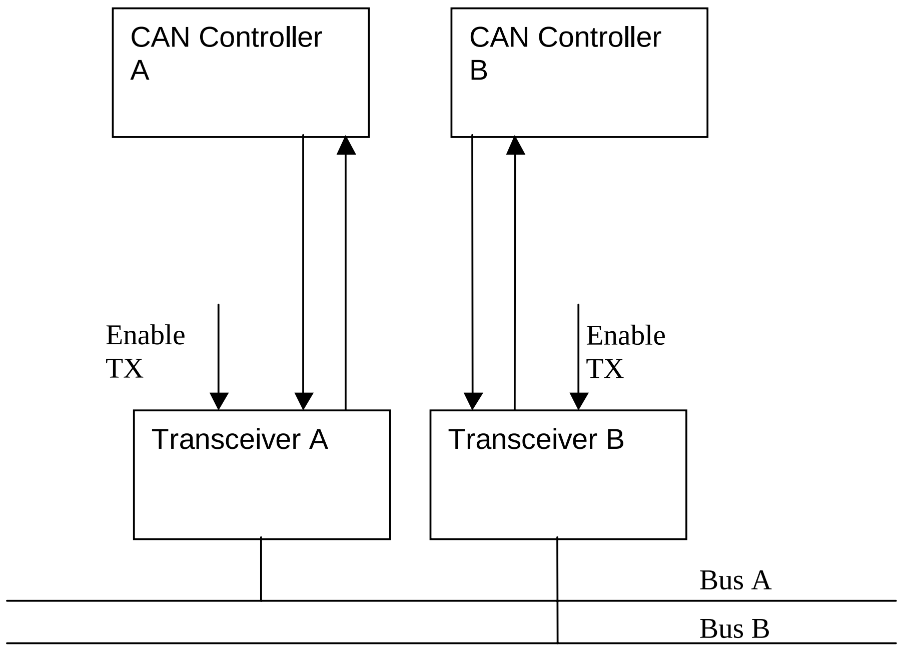


Figure 20: Parallel bus access architecture

11.3 Bus monitoring and reconfiguration management (D)

The bus monitoring and reconfiguration management protocol defined in this standard makes use of CANopen NMT objects (and in particular the CANopen Heartbeat message) to determine the active bus.

The Redundancy Master defines which bus shall be considered active by periodic transmission of CANopen Heartbeat messages on the active bus. The slave nodes monitor the presence of the Heartbeat message from the master to determine the active bus.

In addition to the Master Heartbeat messages the slave nodes also detects if another CANOpen NMT message is received from the master. If so, the bus on which the NMT message was received is considered the active one.

Nodes implementing the selective bus access architecture are not capable of monitoring both buses simultaneously. Therefore, the protocol allows nodes to toggle between the nominal and redundant bus when searching for the Heartbeat of the Redundancy Master.

11.3.1 Bus redundancy management parameters (D)

This section specifies the parameters that define the characteristics of the reconfiguration protocol.

a. Each slave node shall implement the objects listed in Table 3 in its object dictionary.

Table 3: Parameters for bus redundancy management

Parameter	Remark
Master Consumer Heartbeat Time *Index: 1016h*	The *Master Consumer Heartbeat Time* parameter is specified by CANopen. The parameter defines the maximum time allowed between two

Subindex: 01h	subsequent Master Heartbeat messages. In case the *Master Consumer Heartbeat Time* elapsed a slave node takes actions as specified in this document.
Ttoggle *Index: TBD* *Subindex: TBD*	Parameter used during bus selection when searching for a Master Heartbeat or an other NMT message. *Ttoggle* specifies the number of *Master Consumer Heartbeat Time* during which the node shall listen for an NMT message on a particular bus before switching to the other bus.
Ntoggle *Index: TBD* *Subindex: TBD*	Defines the number of togglings between the Nominal and Redundant bus in case of no NMT message detected. Each switch from one bus to the other shall be considered as one toggling. After *Ntoggle* togglings, in case no NMT message has been detected on any bus, the node stops toggling and stays on the latest selected bus. The value of *Ntoggle* shall be an even number to ensure that after a sequence of togglings the bus selected will be the one defined by the *Bdefault* parameter. In case *Ntoggle* is set to 0 no bus toggling shall be performed and the bus defined by the *Bdefault* parameter shall be used as the active bus.
Ctoggle *Index: TBD* *Subindex: TBD*	Counter of Ntoggles. Shows the count of the number of toggles that have already been done by the device.
Bdefault *Index: TBD* *Subindex: TBD*	Defines the bus to be considered active after a node power-on, node hardware reset and after maximum number of bus togglings have been performed.

b. The objects defined in Table 3 shall be programmable by means of SDOs.

NOTE 1: This is to allow for various in system configurations of the object dictionaries as a result of failure investigations on system level. Recommended values and procedures for setting of the above parameters are found in Annex B.

NOTE 2: This also makes it possible to implement a command to order any slave to switch bus by configuring the default bus on its object dictionary, setting Ntoggle to 0 and resetting the slave.

11.3.2 Startup procedure (D)

a. After a node power-on or after hardware reset, the node shall use the bus defined by the *Bdefault* parameter as the active bus.

NOTE: This implies that the CANopen Boot-up message will be transmitted on the default bus.

b. When in CANopen Pre-operational state, each slave node shall listen for NMT message, alternatively on the two buses according to the toggling mechanism specified in section 11.3.3.

The nodes start-up procedure is illustrated in Figure 21. The start-up procedure ends with an entry to the bus selection process specified in 11.3.4.

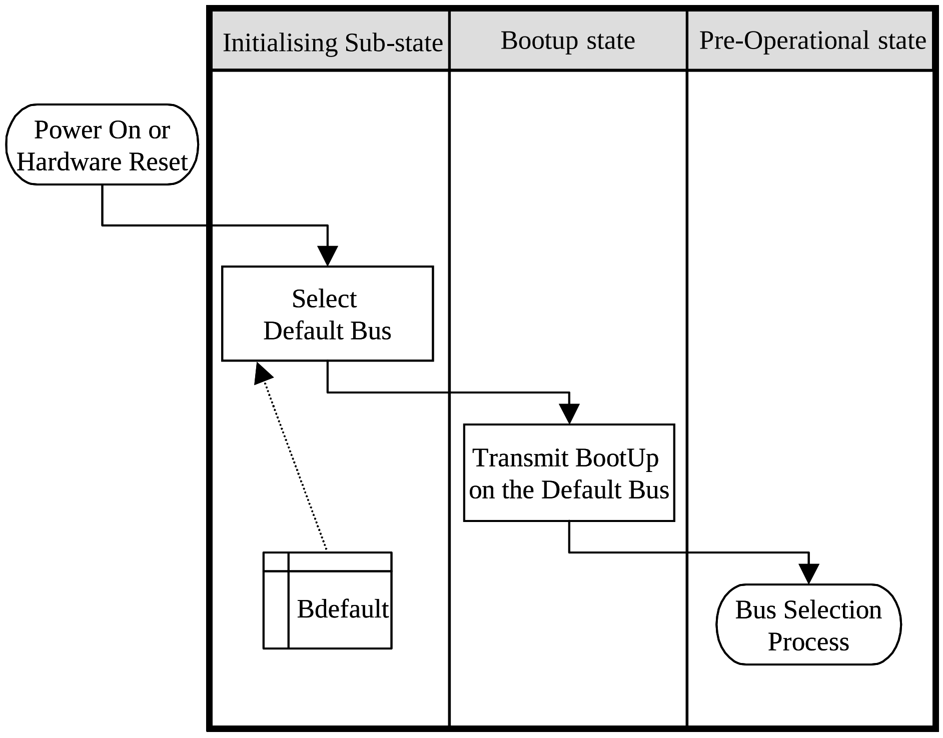


Figure 21: Node start up procedure

11.3.3 Bus montoring protocol (D)

a. The Redundancy Master shall periodically produce CANopen Heartbeat messages on the active bus.

NOTE: This implies that in case the Redundancy Master wishes to switch bus it stops transmitting heartbeat messages on the active bus and starts transmitting on the previously passive bus. This makes the previously passive bus the active one.

b. Each Slave node shall be a consumer of the Master Heartbeat message sent by the Redundancy Master.

c. Each Slave node shall implement the possibility to periodically transmit CANopen Heartbeat messages on the bus it considers being the active.

NOTE: If suitable for the application, the Redundancy Master can be a consumer of the heartbeat messages from the slaves as a means to detect bus or node malfunctions. However, the detailed use of the slave heartbeat messages are application specific and out of scope for this standard.

d. In case a slave node misses Ttoggle times the Master Heartbeat on the assumed active bus it shall enter the CANopen Pre-operational state, switch to the assumed inactive bus and optionally start producing heartbeat on this bus.

NOTE: The CANopen Pre-operational state affects the communication on the CAN bus in the sense that PDOs are no longer transmitted by the node. However, the mode of the application in the node is not necessarily affected.

NOTE 2: The Pre-operational state shall be entered to prevent issuing of PDOs during bus switching. Consequently, the master will need to start any slave node that has performed a bus switching and entered pre-operational state before normal operations can be resumed.

e. Optionally, in case a node misses the Master Heartbeat the node can send an EMCY message on the active bus before switching bus.

Figure 22 illustrates the bus monitoring protocol.

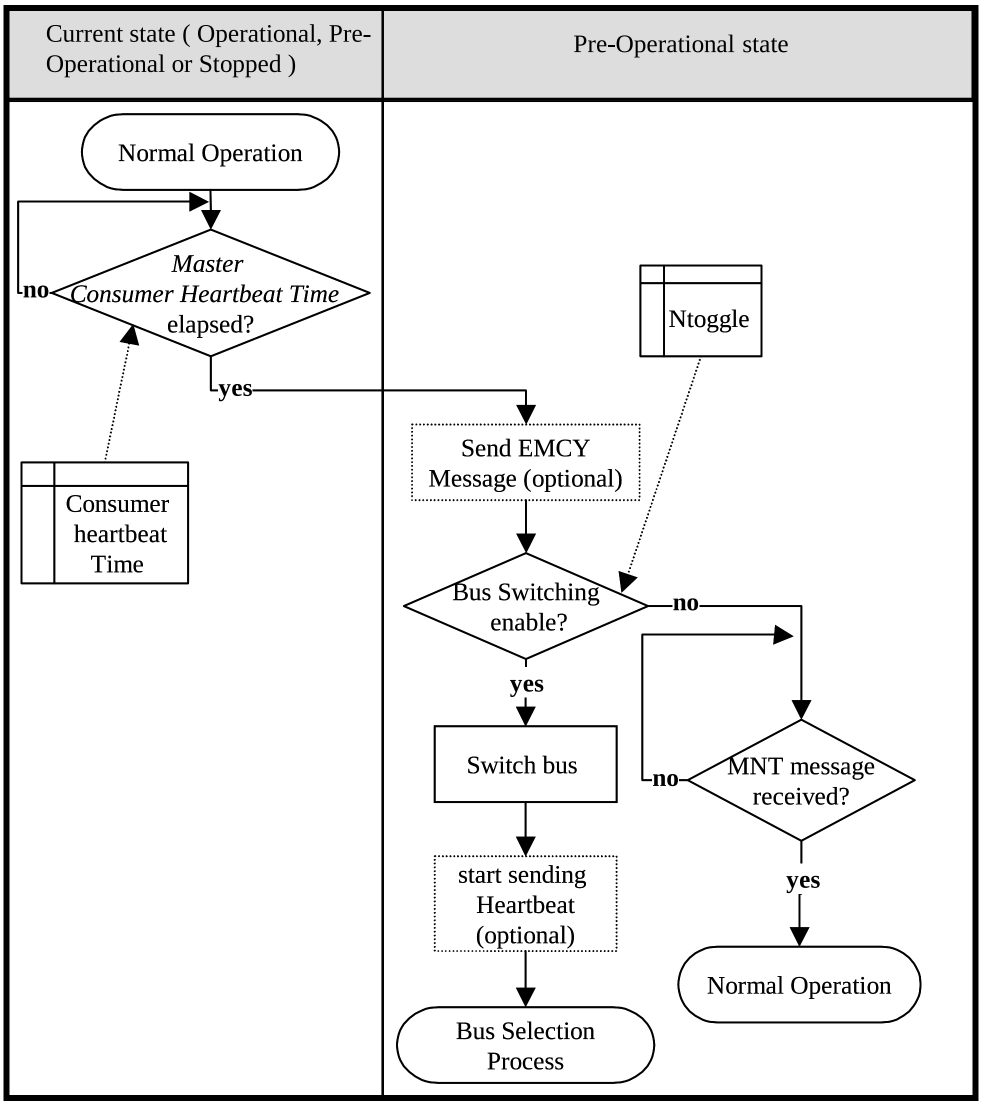


Figure 22: Bus monitoring protocol

11.3.4 Bus selection process

a. When a NMT message has been detected on a particular bus, the slave shall consider this bus the active one.

b. In case *Ttoggle* elapsed and no NMT message has been received on the selected bus a slave node shall switch to the other bus (while still in Pre-operational), optionally start producing heartbeat on this bus, and listen for NMT messages.

c. A slave node shall perform bus toggling as specified in b. for a predefined number of times (*Ntoggle*).

d. If no NMT message has been detected after *Ntoggle* togglings the node stops toggling and shall remain listening on the bus which is, after an even number of toggling, the one defined by the *Bdefault* parameter.

e. After the bus selection process ends if an active bus is found, the value of the Bdefault parameter shall be updated to the new active bus.

Figure 23 illustrates the overall bus selection process.

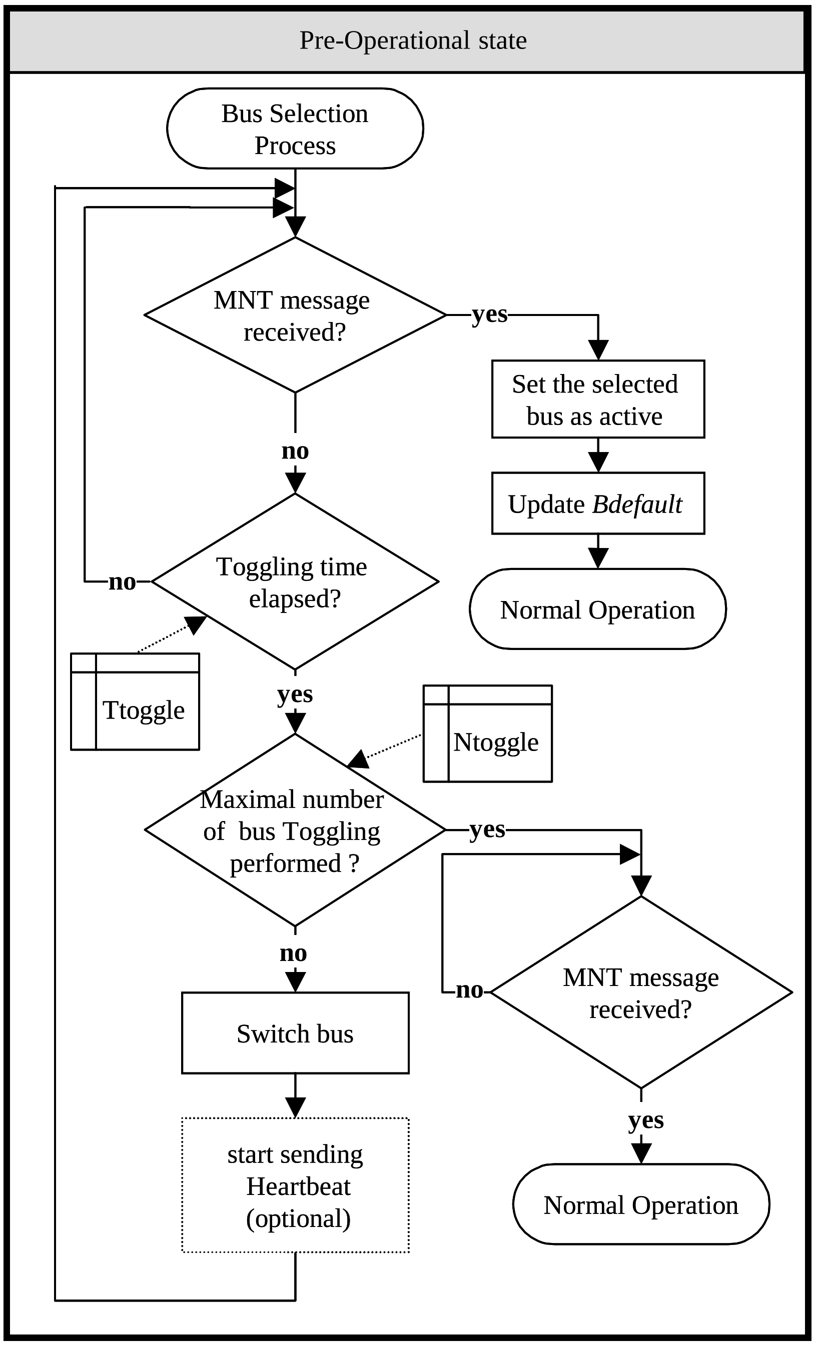


Figure 23: Slave bus selection process, toggling mechanism

Annex A
Recommended connectors and pin assignments (normative)

A.1 General

This annex specifies the connector types and pin allocations to be used for connecting the CAN bus to the CAN nodes.

A.2 Circular connectors

A.2.1 MIL-C D38999 configuration B: Dual CAN bus

a. The connector shall be compliant to specification: MIL-C D38999/ffeA35zA Series 3.

NOTE: There is no restriction on fixing type 'ff' or exterior finish 'e'.

c. The shell size shall be A and the pin layout shall be 35 (6 pin).

d. The key orientation shall be A.

e. The pin function shall be according to Table 4.

Table 4: Pin function for MIL-C D38999 configuration B

Pin No	Function
1	CAN_H (PRIMARY)
2	CAN_L (PRIMARY)
3	Shield CAN (PRIMARY)
4	CAN_H (REDUNDANT)
5	CAN_L (REDUNDANT)
6	Shield CAN (REDUNDANT)

The appropriate gender selection 'z' is specified in section 5.3.2.

A.2.2 MIL-C D38999 configuration D: Single CAN bus

a. The connector shall be compliant to specification: MIL-C D38999/ffeA98zN Series 3.

NOTE: There is no restriction on fixing type 'ff' or exterior finish 'e'.

b. The shell size shall be A and the pin layout shall be 98 (3 pin).

c. The key orientation shall be N.

d. The pin function shall be according to Table 5.

Table 5: Pin function for MIL-C D38999 configuration D

Pin No	Function
1	CAN_H (PRIMARY)
2	CAN_L (PRIMARY)
3	Shield CAN (PRIMARY)

The appropriate gender selection 'z' is specified in section 5.3.2.

A.3 Micro-miniature D Shell connectors

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

A.3.1 Micro-miniature D Shell: Dual CAN bus

a. Connector shall be compliant to specification: micro-miniature D type.

b. The pin function shall be according to Table 6.

Table 6: Pin function for micro-miniature D-type with dual CAN bus

Pin No	Function
1	CAN_H (PRIMARY)
2	Reserved
3	CAN_SHLD
4	Reserved
5	CAN_H (REDUNDANT)
6	CAN_L (PRIMARY)
7	CAN Ground (Required)
8	CAN Ground (Optional)
9	CAN_L (REDUNDANT)

The appropriate gender selection 'z' is specified in section 5.3.2.

A.3.1 Micro-miniature D Shell: Single CAN bus

a. Connector shall be compliant to specification: micro-miniature D type.

b. The pin function shall be according to Table 7.

Table 7: Pin function for micro-miniature D-type with single CAN bus

Pin No	Function
1	CAN_H (PRIMARY)
2	Reserved
3	CAN_SHLD
4	Reserved
5	Reserved
6	CAN_L (PRIMARY)
7	CAN Ground (Required)
8	CAN Ground (Optional)
9	Reserved

The appropriate gender selection ‘z’ is specified in section 5.3.2.

Annex B
Guidelines for implementing bus redundancy management (informative)

B.1 Bus monitoring and reconfiguration management

Annex B intends to provide some guidelines for implementing bus redundancy management. Clause 11.3 defines the mechanisms for bus switching from the slave point of view; this annex provides clues about the master behaviour.

B.1.1 Bus redundancy management parameters

This section provides recommendations on how to configure the parameters involved in the reconfiguration protocol. These recommendations are the result of simulation and implementation.

- *Master Consumer Heartbeat Time*

The value of the *Master Consumer Heartbeat Time* depends on the reactivity required in case of errors.

The *CANopen Framework for Maritime Electronics* [15], that integrates safety considerations, proposes the following values:

Master Consumer Heartbeat Time equals 1 to 1.5 seconds considering that the *Master heartbeat producer time equals 500ms*.

It is reasonable to consider for non critical system values in the range of few seconds for the *Master heartbeat producer time* and the double or third of this value for the *Master Consumer Heartbeat Time.*

The following TBD values have been implemented on the demonstrator with the TBD results.

Description: Consumer Heartbeat Time is defined in CANopen Application Layer and Communication Profile as an UNSIGNED16. The time has to be a multiple of 1ms.

- *Ttoggle*

Ttoggle is a multiple of *Master Consumer Heartbeat Time*. Its value should be long enough to enable the slaves to detect NMT messages before toggling to the other bus. Nevertheless the time to recover from a fault may increase in case this value is too high.

Values between 2 and 10 seem reasonable. Considering the values given above for the *Master Consumer Heartbeat Time*, it would implies that a slave would wait between 2 and 10 seconds on one bus before toggling to the other one.

Description: Ttoggle may be an UNSIGNED8, and then have up to 255 times the value of *Master Consumer Heartbeat Time.*

- *Ntoggle*

Ntoggle determines the number of bus switching a slave node executes before remaining on the default bus.

This parameter indicates the number of opportunities a slave node has to find some activity on the bus.

The value 0 means no bus switching is enable. This value may be set in case one of the bus is definitely out of service, to avoid the slaves to switch to this bus even in case a Master Heartbeat is missed on the other one.

The value 2 gives only one opportunity to find the master on the redundant bus. In case a major problem occurs that implies some reconfiguration, this value may be too small.

The value is to be chosen depending on the time needed for a spacecraft reconfiguration. In the case 1 minute is needed before communication can resume on the bus, with Ttoggle=5 seconds, Ntoggle may be set to 15.

Description: *Ntoggle* may be an UNSIGNED8, this allows to toggle up to 255 times.

- *Bdefault*

Bdefault is the bus a node is first listening to after a power-on or hardware reset. It is also the bus a node remains listening to after it performed the maximum number of bus toggling (Ntoggle).

It shall be set to the Nominal Bus, and may be change during operation if the nominal bus goes faulty for example.

B.1.2 Startup procedure (D)

The clause 11.3.2 defines the start-up procedure from the slave node point of view. This annex provides some elements about the master behaviour.

After power-on the master shall check that all the slaves are present on the same bus before going to operational state. The basic boot-up procedure of the Redundancy Master (which is also the NMT manager) is described in the *CANopen Framework for CANopen Managers and Programmable CANopen Devices* [16].

After the master powers on it issues its bootup message and go in Pre-Operational, it then sends a “NMT Reset Communication all Nodes” command to put the slaves in a state, where the setting of all parameters are well defined. The master begins the boot up procedure for each slave.

If the master is a consumer of the Heartbeat of various slave nodes it can use this information to know which slave is present on the bus. It may also use the bootup message sent by the slave nodes before entering the Pre-Operational state.

It is also possible for the master to detect the presence of a slave node on the bus by simply trying to access the *Device Type* (object 1000h) via SDO. The device type is one of the 3 mandatory objects of the Object dictionary.

B.1.3 Bus monitoring protocol (D)

The clause 11.3.3 defines the actions to be taken by a slave node in case the *Master Consumer Heartbeat Time* elapsed.

The master has also the capability to be a heartbeat consumer of each slave node and it can use this feature to monitor the health of the bus and the health of nodes. Missing a slave node heartbeat has for consequence a series of actions that are application specific. It could be for example the transmission of a "NMT stop all Nodes" command or "NMT Reset Node" command, or the action to switch bus by stopping the transmission of the Heartbeat on the active bus and starting its transmission on the previous inactive bus. The master could eventually actively notify the slave nodes about its will of switching bus by transmitting a dedicated "switch bus" command.

B.1.4 Bus selection process

After the master power-on, if no activity is detected on the bus defined by *Bdefault* in its object dictionary, it should switch to the other bus after a certain amount of time

The Redundancy Master has the following parameters defined in the object dictionary: *Ttoggle, Ntoggle,* and *Bdefault*. It enables the Redundancy Master to toggle to determine the active bus.

Usually the active bus is set by the Redundancy Master. In an extreme case where for example the nominal bus is faulty, an the redundant master is powered on, if its *Bdefault* parameter is set to Nominal bus, it should be possible for the redundant master to toggle to the redundant bus. It can do it after Ttoggle elapsed.

Annex C Minimalist implementation of CANopen (informative)

INCLUDE Annex with Matrix listing the features used from CANopen (referencing the section in this standard and the one in CANopen standard)

A key feature of CANopen is the scalability. While the range of objects and services is broad the number of mandatory requirements in the CANopen standard is reasonably low and allows for simplified implementations in nodes not requiring the full CANopen capability.

C.1 Communication Objects

C.1.1 Service Data Objects

All devices shall implement at least one server SDO.

C.1.2 Network management objects

All the devices shall implement the NMT state diagram and all the CANopen NMT objects. This includes:

- Module Control Services
- Error Control Services (via Heartbeat)
- Bootup Service

C.2 Object Dictionary

All devices shall implement an object dictionary according to CANopen. Only 3 entries are mandatory:

- Device type (1000h)
- Error register (1001h)
- Identity object (1018h)

Annex D
Process for adoption of COB-Ids
(informative)

D.1 Overview

In a CAN system the assignment of the CAN frame identifiers (COB-IDs) is a very important task since the priorities of the individual messages has a direct influence on the real-time behaviour of the system.

The system designer has the full knowledge of the entire system and thus only the system designer is in a position to allocate unique identifiers to each message that appears on the CAN network. The system designer will thus need to be responsible for the scheduling of the CAN message communications as this is a system level function and not possible with only a partial sub-system overview.

A recommended methodology is to define in a system specification all specific identifiers and communication properties that shall be implemented as default values by each device supplier. This implies that when a device is delivered to the customer for system integration, it will already contain agreed default COB-IDs of all communication objects. The specification of the communication characteristics of each device could be performed by means of Electronic Data Sheets (EDS).

Note that during the individual device development phase, each device supplier may be supplied with a complete communications matrix from the system designer to assist in the development and test of the device. This could prove useful since the entire network communication could be simulated using suitable tools.

Since this standard requires that it shall be possible to reprogram the COB-ID values of all devices, it will be possible for the system designer to modify the COB-IDs during system integration should that prove necessary.

Figure 24 illustrates a recommended development methodology for CAN based spacecraft devices/instruments.

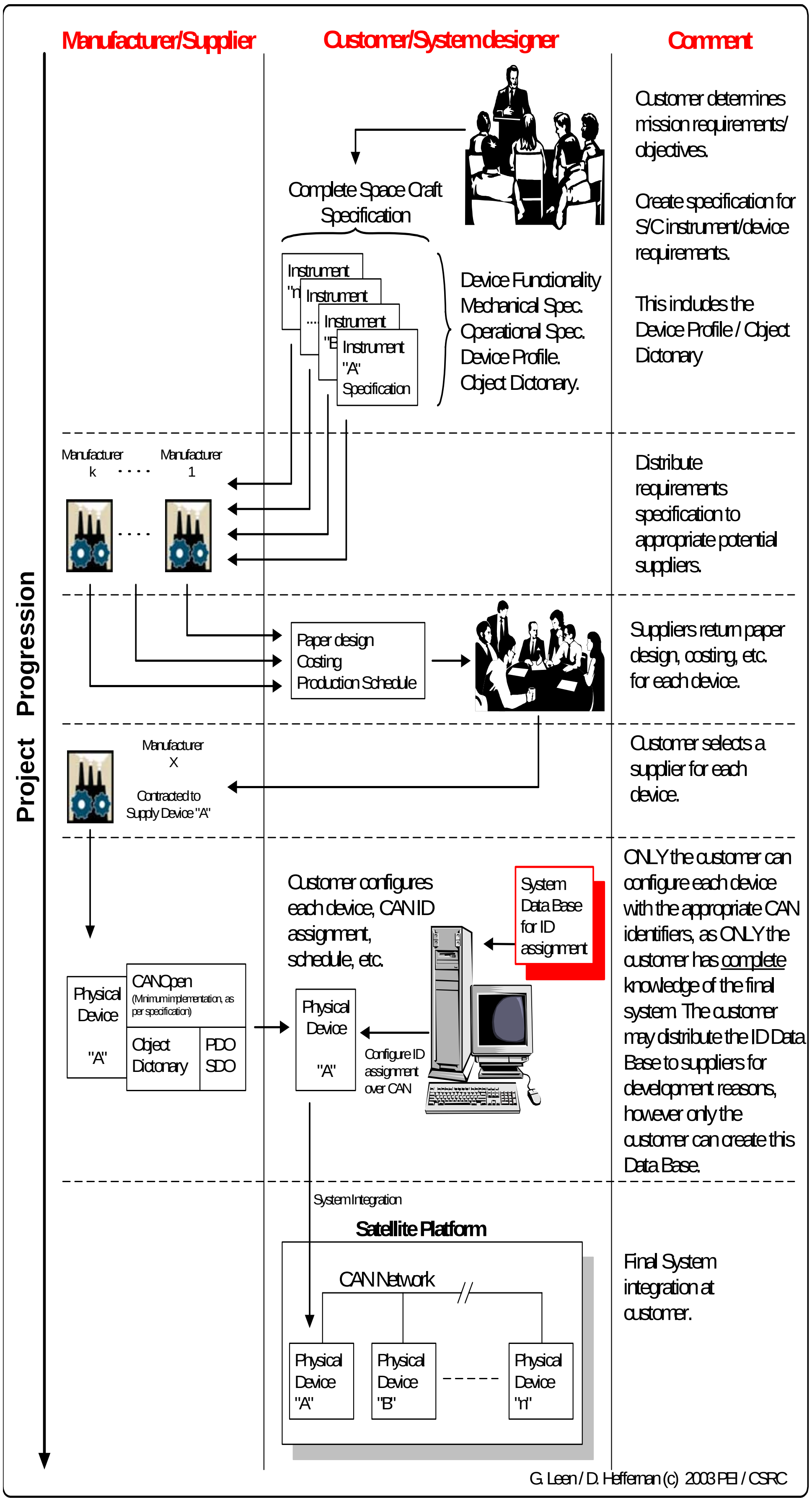

Manufacturer/Supplier
Customer/System designer
Comment
Project Progression
Complete Space Craft Specification
Instrument "n"
Instrument ...
Instrument "B"
Instrument "A" Specification
Device Functionality
Mechanical Spec.
Operational Spec.
Device Profile.
Object Dictionary.
Customer determines mission requirements/ objectives.
Create specification for S/C instrument/device requirements.
This includes the Device Profile / Object Dictionary
Manufacturer k
Manufacturer 1
Distribute requirements specification to appropriate potential suppliers.
Paper design
Costing
Production Schedule
Suppliers return paper design, costing, etc. for each device.
Manufacturer X
Contracted to Supply Device "A"
Customer selects a supplier for each device.
Customer configures each device, CAN ID assignment, schedule, etc.
System Data Base for ID assignment
ONLY the customer can configure each device with the appropriate CAN identifiers, as ONLY the customer has complete knowledge of the final system. The customer may distribute the ID Data Base to suppliers for development reasons, however only the customer can create this Data Base.
Physical Device "A"
CANOpen (Minimum implementation, as per specification)
Object Dictionary
PDO SDO
Physical Device "A"
Configure ID assignment over CAN
System Integration
Satellite Platform
CAN Network
Physical Device "A"
Physical Device "B"
Physical Device "n"
Final System integration at customer.
G. Leen / D. Heffernan (c) 2003 PEI / CSRC

Figure 24: CAN bus COB-ID allocation flow

D.2 CANopen Identifier Allocation

The CANopen communications profile specifies a default scheme for the allocation of identifiers. This default allocation predefines a Master/Slave connection set allowing the implementation of peer-to-peer communication between an Application Master device and the Slave nodes without the need for an identifier distribution process. This default configuration is available in each device after the initialisation phase, when the device enters the preoperational state.

These default message identifiers used by slave devices for communication with the Application Master are divided into two parts, as illustrated in

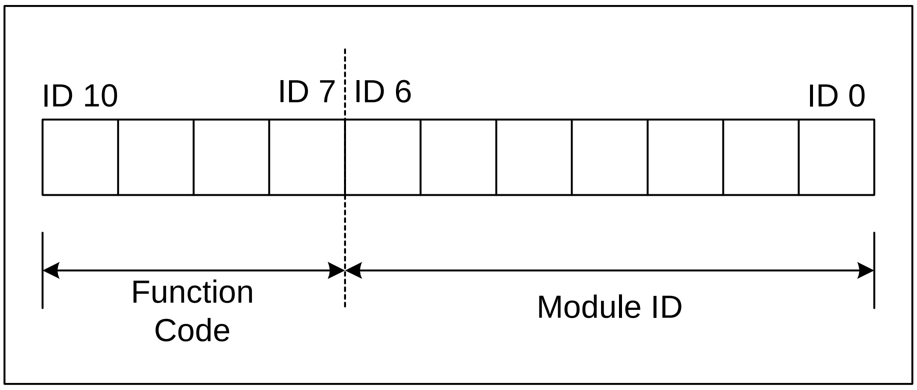


Figure 25: Default identifier functional structure for an 11 bit identifier

The functional component of the identifier determines the message priority, while the Module ID component helps distinguish physically distinct network nodes using exchanging massages in the same functional group. Table 8 illustrates the default set of identifiers for communication between Application Master and slaves.

Object		Function Code (binary)	Identifier (decimal)
Emergency Object		0001	129-255
PDO1	transmit	0011	385-511
PSO1	receive	0100	513-639
PDO2	transmit	0101	641-767
PSO2	receive	0110	769-895
PDO3	transmit	0111	897-1023
PSO3	receive	1000	1025-1151
PDO4	transmit	1001	1153-1279
PSO4	receive	1010	1281-1407
SDO	transmit	1011	1409-1535
SDO	receive	1100	1537-1663
NMT		1110	1793-1919

Table 8: Default peer-to-peer identifier allocation

This allocation provides each device with eight PDO’s (four transmit and four receive), one SDO (two identifiers are required for this purpose) and an Emergency object. Also allocated is an identifier for NMT Error Control (Node

Guarding and Heartbeat) and Boot-up Services, which share the same identifier.

In addition identifiers are allocated to allow the Application Master to communicate with the slaves on a peer-to-peer basis. Finally an additional set of identifiers is defined for message broadcast.

Object	Function Code (binary)	Identifier [decimal] (hex)
NMT Module Control	0000	0
Synchronisation Object	0010	[128] (0x80)
Time Stamp	0011	[256] (0x100)

Table 9: Broadcast object predefined identifier allocation

If one starts with the default CANopen identifier allocation and subsequently wishes to alter this allocation using a configuration tool for example, then CANopen Communication Profile provides guidelines for doing so. These guidelines are summarised in

Priority	Communication Object
Highest (low identifier value)	➢ SYNC
	➢ Emergency
	➢ Network timing (Time Stamp)
	➢ Synchronisation messages (other)
	➢ Synchronous PDOs
	➢ Asynchronous PDOs
Lowest (high identifier value)	➢ SDOs

Table 10: Guidelines for identifier allocation

The SYNC Objects are given the highest priority in order to minimise drift in the communication cycle period. Emergency Objects are also assigned high priority for error signal handling. Similarly Time Stamp messages are given a high network priority, and so on.

Annex E
CAN system design issues
(informative)

E.1 Overview

The CAN bus has characteristics that are significantly different from most of the control buses traditionally used on-board spacecraft such as Mil-std-1553 OBDH. The CAN bus is an asynchronous multi-master bus where the Medium Access Control is performed by means of a Non-Destructive Bitwise Arbitration (NDBA) technique.

This annex highlights some areas where the characteristics of the CAN bus is different from synchronous buses such as Mil-Std-1553 and OBDH. It shall be noted that the information contained in this annex is not complete or exhaustive. The intention is only to highlight a few areas where particular attention might be required.

The topics discussed below are:

Message latency	The CAN message latency section illustrates, with an example, how the allocation of identifiers and the network traffic generated by each network module could affect the message latency.
Babbling idiot	Due to the NDBA technique there is the possibility that a high-priority message sent continuously could block all other bus traffic.
Double frames	In case of a bit error at the end of a frame there is the possibility that the frame will be retransmitted, causing two identical frames to be sent sequentially on the network.
Priority inversion	Priority inversion may occur when the CAN controller implements only one transmit buffer. A low priority message stored in the buffer could prevent a message of higher priority from being transmitted over the bus.

E.2 Message latency

The Non-Destructive Bitwise Arbitration (NDBA) technique utilised in CAN need to be well understood by the system designer. In effect the NDBA mechanism creates a distributed network-wide message queue whereby the priority of the message placed in the queue defines precedence rather than any temporal attribute, i.e. how long the message has been waiting in the queue.

Network messages with low relative priorities may be delayed by messages with a higher relative priority. Each priority level (identifier value) is uniquely assigned to a network frame on a network-wide basis. This feature leads to possible situations where some frames, despite their immediate importance at an application or system level, may experience delays resulting from 'higher priority' messages present on the network.

Figure 26 illustrates an example situation where a message with identifier 8 becomes ready for transmission, and in order for the application requirements to be fulfilled, it must be successfully transferred on the network before a specific point in time known as its deadline.

In this example message 8 becomes ready for transmission, while at the same instance in another node a message with identifier 6 becomes ready. The NDBA mechanism ensures that message 6 is allowed to transmit while message 8 is forced to wait for the bus to become idle. As the diagram illustrates a pattern of events may occur whereby message 8 is prevented from being transmitted and eventually misses its deadline.

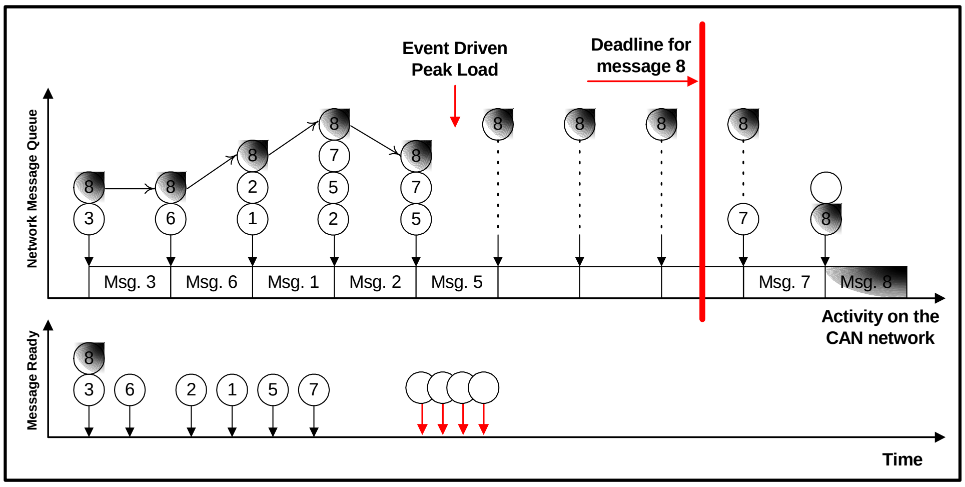


Figure 26: Distributed queue on a CAN network

This example illustrates two points that need to be particularly considered when using CAN: one being the possible missed deadline scenario, and the other the indeterminable message transmission latency time which may be experienced in a loosely coupled un-synchronised network.

Several analytical and scheduling techniques have been developed which place an upper bound on such transmission latency times. However, all such techniques fail to resolve the high degree of temporal tolerance experienced in a frame transmission. Despite this, there are scheduling techniques that do guarantee, for a given configuration and minimum message release period, that message deadline constraints are satisfied. These scheduling techniques usually sacrifice bandwidth achieving this objective.

There are many papers available on the subject of event-triggered CAN networks, with regard to the best technique for scheduling such a network.

Tindell's research focuses on fixed priority scheduling [1]. Shin [7] supports Earliest Deadline (ED) scheduling. Zuberi et al. favour Mixed Traffic Scheduling [8] where ED is used for high-speed messages and Deadline Monotonic (DM) for low speed messages. Cena and Valenzano suggests Priority Promotion (PP) and Distributed Priority Queue (DPQ) schemes [9]. Hong presents a scheduling algorithm for mixed control and communication traffic [10].

Much work has been done to establish the bounded response time of data frames and remote frames in an event triggered Controller Area Network. For example, Tindell et al. [11], Navet et al. presents an analysis for CAN networks in the presence of errors [12], while Rudiger summarises many of these techniques and introduces the notion of a cost function [13].

Current fully event-triggered CAN implementations have relatively low network utilisation factors. For non-critical systems the network bus utilisation factor rarely exceeds 50% [14]. For hard real-time systems, (e.g. engine management and power train control) the utilisation factor is much lower at around 20 - 30%. This low utilisation factor for hard real-time systems is necessary to allow for the overhead produced when a transmission error occurs and a retransmission is necessary. This factor sets a pessimistic upper bound on the bandwidth utilisation factor. In this approach the utilisation factor is strongly dependent on the ratio of the message length to the size of an error frame as indicated in Equation 1.

$$\frac{Message \quad Size}{(2 \quad x \quad Message \quad Size) + Error \quad Frame \quad Size} = Utilisation \quad Factor$$

Equation 1: CAN bus utilisation factor

Despite this, high network utilisation factors may be attained in applications where the probability of network errors is very low, as is typically the case on-board spacecraft.

E.3 Babbling idiot

CAN is a shared medium communication protocol whereby communication bandwidth is assigned in its entirety to the transmitting nodes as determined by CAN's native bitwise arbitration mechanism. Essentially the communication channel is multiplexed in the time domain between nodes wishing to communicate based on the competing messages relative priority.

One particular class of fault known as "babbling idiot" may potentially occur in any shared channel communication protocol including CAN whereby a faulty node on the network monopolises the available communication bandwidth exclusively for its own use. This failure arises when a faulty node continuously tries to transmit a particular message (or messages) which is not intended behaviour for a correctly functioning node.

This particular class of failure may be exhibited to varying degrees of severity depending on the relative priority of the message being transmitted. A message with a higher priority relative to other message attempting to gain access to the medium will block those lower priority messages. The higher the priority of the faulty message the more critical the problem becomes. The situation may also be aggravated if the transmit message queues of the correctly operating nodes do not priority order messages for transmission thus a blocking situation may occur.

A number of strategies have been proposed to help prevent this failure mode, such as a hardware bus guardian allowing a given node access to the medium only at specifically defined time intervals and/or for a specific time duration. Depending on the level of fault tolerance one desires this mechanism may need to be implemented in a separate physical device whereby if the transmitter attempts to transmit outside the synchronised time window allocated, an error is flagged. However, this method will most likely require global synchronisation of the nodes and transmission schedule.

Another solution is to monitor the frequency of message transmissions and to compare this to the expected maximum release periods for the particular messages.

E.4 Double frames

On a CAN network, as with many other control networks, it may be possible, under fault conditions, to send or receive a second copy of the same message. CAN has been designed such that all correctly received messages are acknowledged by the receiving node (or nodes), by writing a dominant bit in the acknowledgement portion (ACK slot) of the transmitted frame; thus overwriting the sent recessive level in the ACK slot to indicate successful reception. If the acknowledgement bit is missing or an error has been detected earlier in the frame, both situations resulting in an error frame, then the message content received thus far in the receiving node's CAN protocol engine are destroyed and the transmitting node will proceed to re-transmit the message again.

However, if for some reason a node's transmission operation fails to work correctly such that a message is sent twice, or if a failure occurs in the reception process such that a node is under the impression that a message is received twice, then the consequences of this double message sending error or double message receiving error is a function of both the content and context of the message involved.

Evaluation of the problem must consider whether the failure is transient or permanent in nature, whether the message is a status message, control message, RTR message, ADC signal or otherwise. A node's application software (or hardware) can be programmed to know the expected message sequence and thus be aware of message duplications. A further detailed discussion of these failure modes is beyond the scope of this document.

E.5 Priority inversion

Priority inversion may occur in one node when the CAN controller implements only one transmit buffer. A low priority message stored in the buffer could prevent a message of higher priority from being transmitted over the bus.

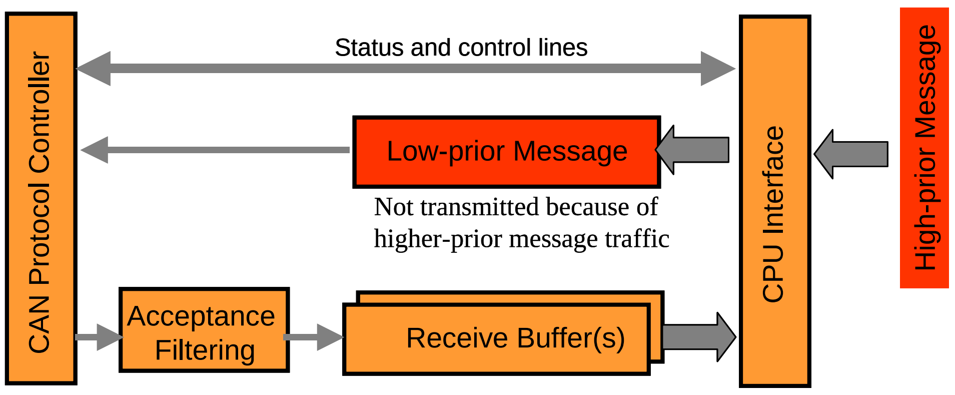


Figure 27: Inner priority inversion

One possible solution is to use CAN controllers that provide multiple transmit buffers with different priorities. Messages can then be queued by the higher layer protocol to one buffer or another depending on their priority. Anyway, this might not be the optimum solution, because if all buffers are full, and a new message with higher priority is to be sent, it will still be blocked.

A better way to solve the problem would be to use a single transmit buffer and sort the messages by priority outside the controller. This way, the higher layer protocol has to implement a priority ordered transmission queue, instead of the typical FIFO. In order for this to work, the controller must offer the possibility to clear the transmit buffer if the arbitration was lost. The higher layer protocol should then keep that low priority message in the queue and pass the new highest priority message to the controller.

Annex F
PHY Layer design considerations (informative)

The large amount of propagation time in the 125Kbps and 250Kbps configurations facilitates slower slopes and input filtering.

Transceivers may be used in conjunction with filter circuits. However, the increased signal slope times and propagation delays shall be included in each device's CAN interface propagation delay time. The total shall still comply with the bit timing specification of section 5.5.3.

It is possible to use compliant devices, i.e. those with bit-timing characteristics meeting this document's requirements, on the same bus as devices that use an earlier (non-compliant) sample point. However, the potential maximum bus length is shortened accordingly.

It is possible to use compliant devices, i.e. those with bit-timing characteristics meeting this document's requirements, on the same bus as devices that use CAN interfaces with (non-compliant) larger propagation delays. However the bus length shall be shortened accordingly.

It may not possible to use compliant devices on the same bus as devices that have wider oscillator tolerances than the tolerance described in this document.

Specific Maximum Bus Length and Stub (Drop) Length are considered to be system specific parameters that are not constrained by this specification and are therefore left to the system developer to optimise for any specific system. Guidance on the specification of these parameters can be obtained by referring to an Application Note from Philips components, see reference Philips AN123, and ISO 11898 and SAE J2284.

Annex G
Compliance pro-forma (informative)

TBW when normative section has been approved.

Bibliography

[1] Controller Area Network (CAN), CAN Specification 2.0B, Robert Bosch GmbH

[2] Physical Layer Recommendations for CAN bus in Heavy Vehicles, Society of Automotive and Aerospace Engineers: SAE J2284

[3] Philips Semiconductor Application Note, CAN bus Timing:
Philips AN4781NG

[4] CANOpen Cabling and Connector Pin Assignment:
CiA Draft Recommendation DR-303-1

[5] CAN Wiring – Notes on the wiring of CAN-Bus Systems and Cable Selection, Version 3.4, ESD Electronic System Design Gmbh, Hanover Germany

[6] K.W. Tindell , H. Hansson and A.J. Wellings, Analysing Real-Time Communications: Controller Area Network (CAN), Real-Time Systems Symposium, 1994., Proceedings pp. 256 - 263, 1994

[7] K. G. Shin, Real-time communications in a computer-controlled work-cell. IEEE Transactions on Robotics and Automation, Vol. 7 pp. 105 - 113, Feb. 1991

[8] K. M. Zuberi, and K.G Shin, Scheduling Messages on Controller Area Network for Real-Time CIM Applications, IEEE Transactions on Robotics and Automation, Vol. 13, No. 2, pp. 310 - 314, April. 1997.

[9] G. Cena and H. Valenzano, An improved CAN fieldbus for industrial applications, IEEE\ Transactions on Industrial Electronics, 1997, 44 (4), pp. 553 - 564

[10]S.H. Hong, Scheduling algorithm of data sampling times in the integrated communication and control system, IEEE\ Transactions on Control System Technology, 3 (2), 1995 pp. 225 - 230

[11]K. Tindell, Calculating Controller Area (CAN) message response times, Contr. Eng. Practice, Vol. 3, no. 8. pp. 1163-1169, 1995

[12]N. Navet and Y.-Q. Song ,Design of reliable real-time applications distributed over CAN (Controller Area Network) (INCOM'98 - IFAC)

Int. Symp. On Information Control in Manufacturing, pp. 391 -396, 1998

[13]R. Rudiger, Evaluating the temporal behavior of CAN based systems by means of a cost functional Proc. 5th international CAN Conference '98, pp. 10.09-10.26, San Jose, CA, USA, 3-5 November 1998

[14]J-A Yung, S-W. Nam, K-W. Kim, S. Lee, M.H. Lee. J.M. Lee and J.H. Kim, Performance Evaluation of Multiplexing Protocols, International Congress and Exposition, Detroit, Michigan. Feb. 23-26, 1998. SAE paper reference number 981105

[15]*CANopen Framework for Maritime Electronics.* CiA Draft Standard Proposal 307 Version 1.0.1, 8.November 2002

[16]*CANopen Framework for CANopen Managers and Programmable CANopen Devices.* CiA Draft Standard Proposal 302 Version 3.3.0

[17]Etschberger, K.: CAN-based Higher Layer Protocols and Profiles; Proceedings of the 4th International CAN Conference; CAN-in-Automation; Berlin; 1997

[18]Etschberger, K.: Controller Area Network; IXXAT Press; 2001; ISBN 3-00-007376-00

[19]Lawrenz, W.: CAN System Engineering, from theory to practical applications; Springer-Verlag; 1997; ISBN 0-387-94939-9

[20]Barbosa, M.: CANopen Implementation; Research Studies Press ltd.; 2000; ISBN 0-86380-247-8

[21] ECSS-E-20A, Space Engineering – Electrical and electronic

[22] PSS-04-106, Packet Telemetry Standard

[23] Leen, G. and Heffernan, D.: Time-triggered Controller Area Network, IEEE Computing and Control Engineering Journal. Vol. 12, Issue 6, Dec. 2001, pp. 245 - 256.