

BS ISO 17458-2:2013



BSI Standards Publication

Road vehicles — FlexRay communications system

Part 2: Data link layer specification

bsi.

...making excellence a habit.
AVA Documents™
Document Sharing Hub

National foreword

This British Standard is the UK implementation of ISO 17458-2:2013.

The UK participation in its preparation was entrusted to Technical Committee AUE/16, Electrical and electronic equipment.

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

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

© The British Standards Institution 2013. Published by BSI Standards Limited 2013

ISBN 978 0 580 76713 5

ICS 43.040.15

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

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 31 January 2013.

Amendments issued since publication

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

INTERNATIONAL STANDARD

BS ISO 17458-2:2013

ISO
17458-2

First edition
2013-02-01

Road vehicles— FlexRay communications system —

Part 2: Data link layer specification

Véhicules routiers — Système de communications FlexRay —

Partie 2: Spécification de la couche de liaison de données



Reference number
ISO 17458-2:2013(E)

AVA Documents
Document Sharing Hub
© ISO 2013



COPYRIGHT PROTECTED DOCUMENT

© ISO 2013

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Page

Foreword	v
Introduction.....	vi
1 Scope	1
2 Normative references	1
3 Terms, definitions, symbols and abbreviated terms	1
3.1 Terms and definitions	1
3.2 Symbols.....	7
3.3 Abbreviated terms	7
4 Document overview.....	10
5 Conventions	11
5.1 General	11
5.2 Notational conventions.....	11
5.3 SDL conventions	12
5.4 Bit rates	15
5.5 Roles of a node in a FlexRay cluster.....	15
5.6 Synchronisation methods	15
5.7 Network topology considerations	19
5.8 Example node architecture.....	24
6 Protocol operation control	29
6.1 Principles.....	29
6.2 Description	31
6.3 The protocol operation control process	37
7 Coding and Decoding	59
7.1 Principles.....	59
7.2 Description	59
7.3 Coding and decoding process	77
7.4 Bit strobing process.....	96
7.5 Wakeup pattern decoding process.....	99
8 Frame Format.....	103
8.1 Overview.....	103
8.2 FlexRay header segment (5 bytes)	103
8.3 FlexRay payload segment (0 – 254 bytes)	108
8.4 FlexRay trailer segment.....	111
8.5 CRC calculation details.....	111
9 Media Access Control.....	113
9.1 Principles.....	113
9.2 Description	123
9.3 Media access control process	126
10 Frame and Symbol processing	143
10.1 Principles.....	143
10.2 Description	143
10.3 Frame and symbol processing process.....	149
11 Wakeup and Startup	161
11.1 General	161
11.2 Cluster wakeup	162
11.3 Communication startup and reintegration.....	167

12	Clock synchronisation	190
12.1	Introduction	190
12.2	Time representation	191
12.3	Synchronisation process	193
12.4	Startup of the clock synchronisation	200
12.5	Time measurement	204
12.6	Correction term calculation	208
12.7	Clock correction.....	220
12.8	Sync frame configuration	223
12.9	Time gateway interface	225
13	Controller Host Interface.....	226
13.1	Principles.....	226
13.2	Description	227
13.3	Interfaces	228
Annex A	(normative) System parameters.....	268
A.1	Protocol constants	268
A.2	Performance constants.....	270
Annex B	(normative) Configuration constraints.....	271
B.1	General.....	271
B.2	Bit rates.....	271
B.3	Parameters	272
B.4	Calculation of configuration parameters for nodes in a TT-D cluster.....	281
B.5	Configuration of cluster synchronisation method and node synchronisation role	334
B.6	Calculation of configuration parameters for nodes in a TT-L cluster	335
B.7	Calculation of configuration parameters for nodes in a TT-E cluster.....	336
Annex C	(normative) Wakeup application notes	345
C.1	Scope	345
C.2	Wakeup initiation by the host.....	345
C.3	Host reactions to status flags signalled by the communication controller.....	348
C.4	Retransmission of wakeup patterns	349
C.5	Transition to startup.....	349
C.6	Wakeup during operation	350
Bibliography		352

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

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

ISO 17458-2 was prepared by Technical Committee ISO/TC 22, *Road vehicles*, Subcommittee SC 3, *Electrical and electronic equipment*.

ISO 17458 consists of the following parts, under the general title *Road vehicles — FlexRay communications system*:

- *Part 1: General information and use case definition*
- *Part 2: Data link layer specification*
- *Part 3: Data link layer conformance test specification*
- *Part 4: Electrical physical layer specification*
- *Part 5: Electrical physical layer conformance test specification*

Introduction

The FlexRay communications system is an automotive focused high speed network and was developed with several main objectives which were defined beyond the capabilities of established standardized bus systems like CAN and some other proprietary bus systems. Some of the basic characteristics of the FlexRay protocol are synchronous and asynchronous frame transfer, guaranteed frame latency and jitter during synchronous transfer, prioritization of frames during asynchronous transfer, single or multi-master clock synchronisation, time synchronisation across multiple networks, error detection and signalling, and scalable fault tolerance.

The FlexRay communications system is defined for advanced automotive control applications. It serves as a communication infrastructure for future generation high-speed control applications in vehicles by providing:

- A message exchange service that provides deterministic cycle based message transport;
- Synchronisation service that provides a common time base to all nodes;
- Start-up service that provides an autonomous start-up procedure;
- Error management service that provides error handling and error signalling;
- Wakeup service that addresses the power management needs;

Since start of development the automotive industry world wide supported the specification development. The FlexRay communications system has been successfully implemented in production vehicles today.

The ISO 17458 series specifies the use cases, the communication protocol and physical layer requirements of an in-vehicle communication network called "FlexRay communications system".

This part of ISO 17458 has been established in order to define the protocol requirements for vehicle communication systems implemented on a FlexRay data link.

To achieve this, it is based on the Open Systems Interconnection (OSI) Basic Reference Model specified in ISO/IEC 7498-1 and ISO/IEC 10731, which structures communication systems into seven layers. When mapped on this model, the protocol and physical layer requirements specified by ISO 17458 are broken into:

- Diagnostic services (layer 7), specified in ISO 14229-1 [7], ISO 14229-4 [9];
- Presentation layer (layer 6), vehicle manufacturer specific;
- Session layer services (layer 5), specified in ISO 14229-2 [8];
- Transport layer services (layer 4), specified in ISO 10681-2 [5];
- Network layer services (layer 3), specified in ISO 10681-2 [5];
- Data link layer (layer 2), specified in ISO 17458-2, ISO 17458-3;
- Physical layer (layer 1), specified in ISO 17458-4, ISO 17458-5;

in accordance with Table 1.

Table 1 — FlexRay communications system specifications applicable to the OSI layers

Applicability	OSI 7 layers	FlexRay communications system	Vehicle manufacturer enhanced diagnostics
Seven layer according to ISO 7498-1 and ISO/IEC 10731	Application (layer 7)	vehicle manufacturer specific	ISO 14229-1, ISO 14229-4
	Presentation (layer 6)	vehicle manufacturer specific	vehicle manufacturer specific
	Session (layer 5)	vehicle manufacturer specific	ISO 14229-2
	Transport (layer 4)	vehicle manufacturer specific	ISO 10681-2
	Network (layer 3)	vehicle manufacturer specific	
	Data link (layer 2)	ISO 17458-2, ISO 17458-3	
	Physical (layer 1)	ISO 17458-4, ISO 17458-5	

Table 1 shows ISO 17458 Parts 2 – 5 being the common standards for the OSI layers 1 and 2 for the FlexRay communications system and the vehicle manufacturer enhanced diagnostics.

The FlexRay communications system column shows vehicle manufacturer specific definitions for OSI layers 3 – 7.

The vehicle manufacturer enhanced diagnostics column shows application layer services covered by ISO 14229-4 which have been defined in compliance with diagnostic services established in ISO 14229-1, but are not limited to use only with them. ISO 14229-4 is also compatible with most diagnostic services defined in national standards or vehicle manufacturer's specifications. The presentation layer is defined vehicle manufacturer specific. The session layer services are covered by ISO 14229-2. The transport protocol and network layer services are specified in ISO 10681.

Road vehicles — FlexRay communications system — Part 2: Data link layer specification

1 Scope

This part of ISO 17458 specifies the FlexRay communication protocol which is specified for a dependable automotive network. Some of the basic characteristics of the FlexRay protocol are synchronous and asynchronous frame transfer, guaranteed frame latency and jitter during synchronous transfer, prioritization of frames during asynchronous transfer, single or multi-master clock synchronisation¹⁾ time synchronisation across multiple networks, error detection and signalling, and scalable fault tolerance²⁾.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 17458-1, *Road vehicles — FlexRay communications system — Part 1: General information and use case definition*

3 Terms, definitions, symbols and abbreviated terms

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO 17458-1 and the following apply.

3.1.1

application data

data produced and / or used by application tasks

NOTE In the automotive context the term 'signal' is often used for application data exchanged among tasks.

3.1.2

bus

communication system topology in which nodes are directly connected to a single, common communication media (as opposed to connection through stars, gateways, etc.)

NOTE The term bus is also used to refer to the media itself.

-
- 1) Multi-master clock synchronisation refers to a synchronisation that is based on the clocks of several (three or more) synchronisation masters or sync nodes.
 - 2) Scalable fault tolerance refers to the ability of the FlexRay protocol to operate in configurations that provide various degrees of fault tolerance (i.e., single or dual channel clusters, clusters with many or few sync nodes, etc.).

3.1.3

channel idle

condition on the physical transmission medium when no node is transmitting, as perceived by each individual node in the network

NOTE The detection of channel idle occurs some time after all nodes have actually stopped transmitting (due to idle detection times, channel effects, ringing, etc.).

3.1.4

clique

set of communication controllers having the same view of certain systems properties

EXAMPLE The global time value or the activity state of communication controllers.

3.1.5

cluster

communication system of multiple nodes connected via at least one communication channel directly (bus topology), by active stars (star topology) or by a combination of bus and star connections (hybrid topologies)

NOTE Clusters can be coupled by gateways.

3.1.6

coldstart node

node capable of initiating the communication startup procedure on the cluster by sending startup frames

NOTE TT-D coldstart nodes, TT-L coldstart nodes, and TT-E coldstart nodes are all considered to be coldstart nodes. By definition, all coldstart nodes are also sync nodes.

3.1.7

communication slot

interval of time during which access to a communication channel is granted exclusively to a specific node for the transmission of a frame with a frame ID corresponding to the slot

NOTE FlexRay distinguishes between static communication slots and dynamic communication slots.

3.1.8

cycle-dependent slot assignment

method of assigning, for a given channel, an individual slot (identified by a specific slot number and a specific cycle counter number) or a set of slots (identified by a specific slot number and a set of communication cycle numbers) to a node

3.1.9

cycle-independent slot assignment

method of assigning, for a given channel, the set of all communication slots having a specific slot number to a node (i.e., on the given channel, slots with the specific slot number are assigned to the node in all communication cycles)

3.1.10

cycle number

positive integer used to identify a communication cycle

NOTE The cycle number of each communication cycle is one greater than the cycle number of the previous cycle, except in cases where the previous cycle had the maximum cycle number value, in which case the cycle number has the value of zero. The cycle number of the first cycle is, by definition, zero.

3.1.11

cycle time

time within the current communication cycle, expressed in units of macroticks

NOTE Cycle time is reset to zero at the beginning of each communication cycle.

3.1.12

dynamic segment

portion of the communication cycle where the media access is controlled via a mini-slotting scheme

NOTE 1 During this segment access to the media is dynamically granted on a priority basis to nodes with data to transmit.

NOTE 2 Also known as Flexible Time Division Multiple Access (FTDMA).

3.1.13

dynamic slot / dynamic communication slot

interval of time within the dynamic segment of the communication cycle consisting of one or more minislots during which access to a communication channel is granted exclusively to a specific node for transmission of a frame with a frame ID corresponding to the slot

NOTE In contrast to a static communication slot, the duration of a dynamic communication slot may vary depending on the length of the frame. If no frame is sent, the duration of a dynamic communication slot equals that of one minislot.

3.1.14

frame

structure used by the communication system to exchange information within the system

NOTE A frame consists of a header segment, a payload segment and a trailer segment. The payload segment is used to convey application data.

3.1.15

frame identifier

slot position in the static segment and priority in the dynamic segment

NOTE A lower identifier indicates a higher priority.

3.1.16

global time

combination of cycle counter and cycle time

3.1.17

Hamming distance

minimum distance (i.e., the number of bits which differ) between any two valid code words in a binary code

3.1.18

implementation dependent

behaviour that, subject to restrictions in the specification, may be chosen by an implementation designer. Implementation dependent behaviour shall be described in detail in the documentation of an implementation

3.1.19

key slot

static slot that is used by a node to transmit sync and startup frames or the slot that is used to transmit when the node is operating in key slot only mode.

3.1.20

macrotick

interval of time derived from the cluster-wide clock synchronisation algorithm

NOTE A macrotick consists of an integral number of microticks. The actual number of microticks in a given macrotick is adjusted by the clock synchronisation algorithm. The macrotick represents the smallest granularity unit of the global time.

3.1.21

microtick

interval of time derived directly from the CC's oscillator (possibly through the use of a prescaler)

NOTE The microtick is not affected by the clock synchronisation mechanisms, and is thus a node-local concept. Different nodes can have microticks of different duration.

3.1.22

minislot

interval of time within the dynamic segment of the communication cycle that is of constant duration (in terms of macroticks) and that is used by the synchronized FTDMA media access scheme to manage media arbitration

3.1.23

non-coldstart node

node that is not capable of initiating the communication startup procedure (i.e., does not transmit startup frames)

3.1.24

non-sync node

node that is not configured to transmit sync frames.

3.1.25

non-synchronized operation

operation of a node when the node does not have a notion of FlexRay time, i.e., has no knowledge of slot identifier, slot boundaries, cycle counter, or segment boundaries

3.1.26

network

combination of the communication channels that connect the nodes of a cluster

3.1.27

node

logical entity connected to the network that is capable of sending and / or receiving frames

3.1.28

null frame

frame that contains no usable data in the payload segment

NOTE A null frame is indicated by a bit in the header segment, and all data bytes in the payload segment are set to zero.

3.1.29

physical communication link

inter-node connection through which signals are conveyed for the purpose of communication

NOTE All nodes connected to a given physical communication link share the same electrical or optical signals (i.e., they are not connected through repeaters, stars, gateways, etc.). Examples of a physical communication link include a bus network or a point-to-point connection between a node and a star. A communication channel may be constructed by combining one or more physical communication links together using stars.

3.1.30

precision

worst-case deviation between the corresponding macroticks of any two synchronized nodes in the cluster

3.1.31

slot

see communication slot

3.1.32

slot ID (identifier)

see slot number

3.1.33

slot multiplexing

technique of assigning, for a given channel, slots having the same slot identifier to different nodes in different communication cycles

3.1.34

slot number

number used to identify a specific slot within a communication cycle

3.1.35

star

device that allows information to be transferred from one physical communication link to one or more other physical communication links

NOTE A star duplicates information present on one of its links to the other links connected to the star. A star can be either passive or active. For the purposes of this specification, all usages of the term "star" are references to an active star as described in ISO 17458-4.

3.1.36

startup frame

FlexRay frame whose header segment contains an indicator that integrating nodes may use timerelated information from this frame for initialisation during the startup process

NOTE Startup frames are always also sync frames.

3.1.37

static slot / static communication slot

interval of time within the static segment of the communication cycle that is constant in terms of macroticks and during which access to a communication channel is granted exclusively to a specific node for transmission of a frame with a frame ID corresponding to the slot

NOTE Unlike a dynamic communication slot, each static communication slot contains a constant number of macroticks regardless of whether or not a frame is sent in the slot.

3.1.38

static segment

portion of the communication cycle where the media access is controlled via a static Time Division Multiple Access (TDMA) scheme

NOTE During this segment access to the media is determined solely by the progression of time.

3.1.39

sync frame

FlexRay frame whose header segment contains an indicator that the deviation measured between the frame's arrival time and its expected arrival time should be used by the clock synchronisation algorithm

3.1.40

sync node

node configured to transmit sync frames

NOTE Coldstart nodes and TT-D non-coldstart sync nodes are considered to be sync nodes.

3.1.41

synchronized operation

operation of a node when the node has a notion of FlexRay time, i.e., has knowledge of slot identifier, slot boundaries, cycle counter, and segment boundaries

3.1.42

time gateway

pair of nodes attached to different clusters connected by a time gateway interface

3.1.43

time gateway interface

interface used by a time gateway source node to provide timing information for a time gateway sink node

3.1.44

time gateway sink node

node configured as TT-E coldstart node, which is connected via a time gateway interface to a time gateway source node

NOTE The time gateway sink node receives timing information from the time gateway source node.

3.1.45

time gateway source node

node connected via a time gateway interface to a time gateway sink node

NOTE The time gateway source node provides timing information for the time gateway sink node.

3.1.46

time sink cluster

cluster using the TT-E synchronisation method

NOTE The term emphasizes that the TT-E coldstart nodes of this cluster receive their timing from another cluster.

3.1.47

time source cluster

cluster that provides the timing information for a time sink cluster

3.1.48

transmission slot assignment list

structure identifying the set of all slots assigned to a node for transmission

3.1.49

TT-D cluster

cluster in which the clock synchronisation uses the TT-D synchronisation method

NOTE A TT-D cluster consists of two or more TT-D coldstart nodes, zero or more TT-D non-coldstart sync nodes and, zero or more non-sync nodes.

3.1.50

TT-D coldstart node

coldstart node operating in a TT-D cluster

NOTE This node has only a single key slot and sends a startup / sync frame in the configured key slot in each cycle on each configured channel.

3.1.51

TT-D non-coldstart sync node

node that is configured to transmit sync frames but is not capable of initiating the communication startup procedure (i.e., does not send startup frames)

3.1.52

TT-D synchronisation method

method of clock synchronisation in which the clock synchronisation is derived in a distributed manner from two or more sync nodes

NOTE Two or more coldstart nodes are required to start up a cluster using this synchronisation method.

3.1.53

TT-E cluster

cluster in which the clock synchronisation uses the TT-E synchronisation method

NOTE A TT-E cluster consists of one or more TT-E Coldstart nodes and zero or more non-sync nodes.

3.1.54

TT-E coldstart node

coldstart node operating in a TT-E cluster

NOTE This node has two key slots and sends startup / sync frames in both configured key slots in each cycle on each configured channel. A TT-E coldstart node is a time gateway sink (i.e., is configured for external synchronisation) and bases its timebase on the clock sync information derived from the time source cluster as delivered by the time gateway interface.

3.1.55

TT-E synchronisation method

method of clock synchronisation in which the clock synchronisation is derived directly from the clock synchronisation of another FlexRay cluster

NOTE In this method a single coldstart node is capable of starting up the cluster.

3.1.56

TT-L cluster

cluster in which the clock synchronisation uses the TT-L synchronisation method

NOTE A TT-L cluster consists of one TT-L Coldstart node and one or more non-sync nodes.

3.1.57

TT-L coldstart node

coldstart node operating in a TT-L cluster

NOTE This node has two key slots and sends startup / sync frames in both configured key slots in each cycle on each configured channel.

3.1.58

TT-L synchronisation method

method of clock synchronisation in which the clock synchronisation is derived from the local clock of a single sync node, and in which a single coldstart node starts up the cluster

3.2 Symbols

Σ	Summation symbol, a large upright capital Sigma
$\in$	Element, lower-case epsilon
$\forall$	For all (given any), universal quantifier symbol, a turned "A"

3.3 Abbreviated terms

μs	Microsecond
μT	Microtick
AP	Action Point
BD	Bus Driver
BIST	Built-In Self Test
BITSTRB	Bit Strobing Process

BSS	Byte Start Sequence
CAS	Collision Avoidance Symbol
CC	Communication Controller
CE	Communication Element
CHI	Controller Host Interface
CHIRP	Channel Idle Recognition Point
CODEC	Coding and Decoding Process
CRC	Cyclic Redundancy Code
CSP	Clock Synchronisation Process
CSS	Clock Synchronisation Startup Process
DTS	Dynamic Trailing Sequence
ECU	Electronic Control Unit, same as node
EMI	Electromagnetic Interference
ERRN	Error Not signal
FES	Frame End Sequence
FIFO	First In First Out
FSP	Frame and Symbol Processing
FSS	Frame Start Sequence
FTDMA	Flexible Time Division Multiple Access
FTM	Fault-Tolerant Midpoint
ID	Identifier
INH1	Inhibit signal
MAC	Media Access Control Process
MT	Macrotick
MTG	Macrotick Generation Process
MTS	Media Access Test Symbol
NIT	Network Idle Time
NM	Network Management
POC	Protocol Operation Control
RxD	Receive data signal from bus driver

SDL	Specification and Description Language
SPI	Serial Peripheral Interface
STBN	Standby Not signal
SW	Symbol Window
sync	synchronisation
TDMA	Time Division Multiple Access
TRP	Time Reference Point
TSS	Transmission Start Sequence
TT-D	Time-Triggered Local Distributed
TT-E	Time-Triggered External
TT-L	Time-Triggered Local Master
TxD	Transmit Data signal from CC
TxEN	Transmit Data Enable Not signal from CC
WUDOP	Wakeup During Operation Pattern
WUP	Wakeup Pattern
WUPDEC	Wakeup Pattern Decoding Process
WUS	Wakeup Symbol

4 Document overview

Figure 1 illustrates the document references.

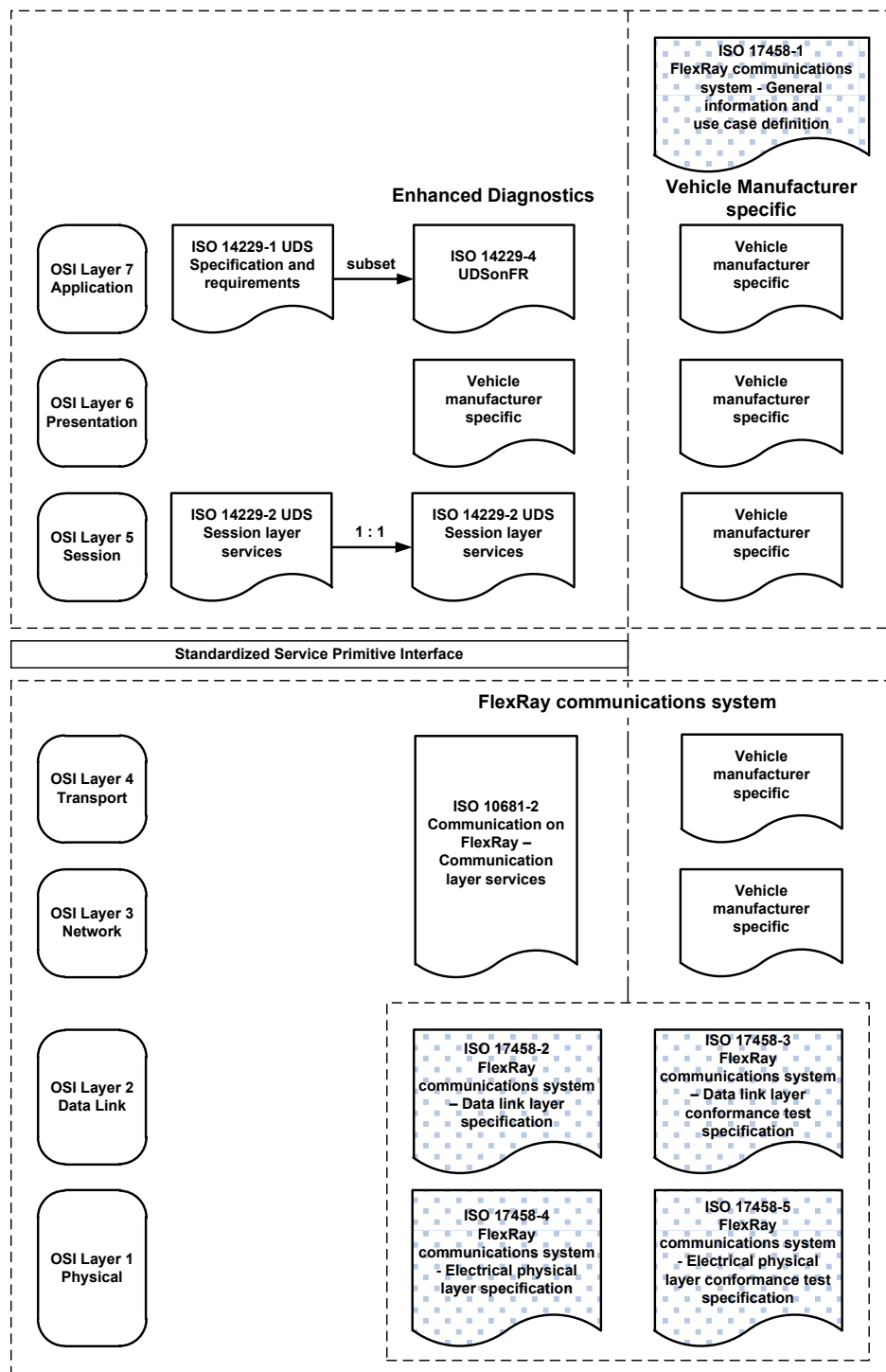


Figure 1 — FlexRay document reference according to OSI model

5 Conventions

5.1 General

ISO 17458 are based on the conventions specified in the OSI Service Conventions (ISO/IEC 10731) as they apply for physical and data link layer (protocol).

5.2 Notational conventions

5.2.1 Parameter prefix conventions

The following is a list of parameter prefix conventions:

- $\langle \text{variable} \rangle ::= \langle \text{prefix_1} \rangle [\langle \text{prefix_2} \rangle] \text{ Name};$
- $\langle \text{prefix_1} \rangle ::= a \mid c \mid v \mid g \mid p \mid z;$
- $\langle \text{prefix_2} \rangle ::= d \mid s;$

Table 2 defines the parameter $\langle \text{prefix_1} \rangle$.

Table 2 — Definition of parameter $\langle \text{prefix_1} \rangle$

Naming convention	Information type	Description
a	Auxiliary parameter	Auxiliary parameter used in the definition or derivation of other parameters or in the derivation of constraints.
c	Protocol constant	Values used to define characteristics or limits of the protocol. These values are fixed for the protocol and cannot be changed.
v	Node variable	Values that vary depending on time, events, etc.
g	Cluster parameter	Parameter that shall have the same value in all nodes in a cluster, is initialized in the <i>POC:default config</i> state, and can only be changed while in the <i>POC:config</i> state.
p	Node parameter	Parameter that may have different values in different nodes in the cluster, is initialized in the <i>POC:default config</i> state, and can only be changed while in the <i>POC:config</i> state.
z	Local SDL process variable	Variables used in SDL processes to facilitate accurate representation of the necessary algorithmic behaviour. Their scope is local to the process where they are declared and their existence in any particular implementation is not mandated by the protocol.

Table 3 defines the parameter $\langle \text{prefix_2} \rangle$.

Table 3 — Definition of parameter $\langle \text{prefix_2} \rangle$

Naming convention	Information type	Description
d	Time duration	Value (variable, parameter, etc.) describing a time duration, the time between two points in time.
s	Set	Set of values (variables, parameters, etc.).

5.2.2 Text coding

Throughout the text several types of items are highlighted through the use of an italicized font.

Parameters, constants and variables are highlighted in *italics*. An example is the parameter *gdStaticSlot*. This convention is not used within SDL diagrams, as it is assumed that such information is obvious. The meaning of the prefixes of parameters, constants, and variables is described in 5.2.1.

SDL states are also highlighted in *italics*. An example is the SDL state *POC:normal active*. This highlighting convention is not used within SDL diagrams. Further notational conventions related to SDL states are described in 5.3.2.

SDL signals are also highlighted in *italics*. An example is the SDL signal *CHIRP on A*. This convention is not used within the SDL diagrams themselves as the fact that an item is an input or output signal should be obvious.

5.2.3 Implementation dependent behaviour

While this specification defines the required behaviour of a FlexRay implementation in many respects, there are various decisions on the particulars of an implementation that, for flexibility reasons, are left up to the implementation designers. This specification defines the term "implementation dependent" to have the following meaning:

- a behaviour (or a parameter or characterization of a behaviour, such as a default value) that, subject to restrictions contained in this specification, may be chosen by an implementation designer;
- implementation dependent behaviour may vary from implementation to implementation, but the specific behaviour shall be described in detail in the documentation of the implementation.

5.3 SDL conventions

5.3.1 General

The FlexRay protocol mechanisms described in this specification are presented using a graphical method loosely based on the Specification and Description Language (SDL) technique described in [10]. The intent of this description is not to provide a complete executable SDL model of the protocol mechanisms, but rather to present a reasonably unambiguous description of the mechanisms and their interactions. This description is intended to be read by humans, not by machines, and in many cases the description is optimized for understandability rather than exact syntactic correctness.

The SDL descriptions in this specification are behavioural descriptions, not requirements on a particular method of implementation. In many cases the method of description was chosen for ease of understanding rather than efficiency of implementation. An actual implementation should have the same behaviour as the SDL description, but it need not have the same underlying structure or mechanisms.

Several SDL diagrams have textual descriptions intended to assist the reader in understanding the behaviour depicted in the SDL diagrams. Some technical details are intentionally omitted from these explanations. Unless specifically mentioned, the behaviour depicted in the SDL diagrams takes precedence over any textual description.

In SDL, transitions between states, and any processing that takes place along the paths involving these transitions, is assumed to take place in zero time. The descriptions of the protocol mechanisms rely on this zero time assumption to specify the proper behaviour of an implementation. Transitions and processing in a real implementation will not take place in zero time. The implementation designer shall comprehend any discrepancy between the implicit zero time assumption in the SDL description and the actual time taken in the chosen implementation technology and ensure that the implementation's behaviour is consistent with the behaviour described in the SDL.

5.3.2 SDL notational conventions

States that exist within the various SDL processes are shown with the state symbol shaded in light gray. These states are named with all lowercase letters. Acronyms or proper nouns that appear in a state name are capitalized as appropriate. Examples include the states "wait for sync frame" and "wait for CE start".

SDL states that are referenced in the text are prefixed with an identification of the SDL process in which they are located (for example, the state *POC:normal active* refers to the "normal active" state in the POC process). This convention is not used within the SDL diagrams themselves, as the process information should be obvious.

The definitions of an SDL process are often spread over several different figures. The caption of each figure that contains SDL definitions indicates to which SDL process the figure belongs.

5.3.3 SDL extensions

5.3.3.1 General

The SDL descriptions in this specification contain some constructs that are not a part of normal SDL. Also, some mechanisms described with constructs that are part of normal SDL expect that these constructs behave somewhat differently than is described in [10]. This subclause documents significant deviations from "standard" SDL.

5.3.3.2 Microtick, macrotick and sample tick timers

The representation of time in the FlexRay protocol is based on a hierarchy that includes microticks and macroticks (see clause 12 for details). Several SDL mechanisms need timers that measure a certain number of microticks or macroticks. This specification makes use of an extension of the SDL 'timer' construct to accomplish this.

An SDL definition of the form

```
μT timer
```

defines a timer that counts in terms of microticks. This behaviour would be similar to that of an SDL system whose underlying time unit is the microtick.

An SDL definition of the form

```
MT timer
```

defines a timer that counts in terms of macroticks. A macrotick timer uses the corrected macroticks generated by the macrotick generation process. Since the duration of a macrotick can vary, the duration of these timers can also vary, but the timers themselves remain synchronized to the macrotick-level timebase of the protocol.

In all other respects both of these constructs behave in the same manner as normal SDL timers.

In addition to the above, several SDL mechanisms used in the description of encoding make use of a timer that measures a certain number of ticks of the bit sample clock. An SDL definition of the form

```
ST timer
```

defines a timer that counts in terms of ticks of the bit sample clock (i.e., sample ticks). This behaviour would be similar to that of an SDL system whose underlying time unit is the sample tick. In all other respects this construct behaves in the same manner as a normal SDL timer.

There is a defined relationship between the "ticks" of the microtick timebase and the sample ticks of bit sampling. Specifically, a microtick consists of an integral number, $pSamplesPerMicrotick$, of sample ticks. As a result, there is a fixed phase relationship between the microtick timebase and the ticks of the sample clock.

The time expression of a timer is defined in [10] by:

`<Time expression> = now + <Duration constant expression>`

In this specification the time expression is used in the following simplified way:

`<Time expression> = <Duration constant expression>3)`

5.3.3.3 Microtick behaviour of the 'now' – expression

The behavioural descriptions of various aspects of the FlexRay system require the ability to take "timestamps" at the occurrence of certain events. The granularity of these timestamps is one microtick, and the timestamps taken by different processes need to be taken against the same underlying timebase. This specification makes use of an extension of the SDL concept of time to facilitate these timestamps.

This specification assumes the existence of an underlying microtick timebase. This timebase, which is available to all processes, contains a microtick counter that is started at zero at some arbitrary epoch assumed to occur before system startup. As time progresses, this timebase increments the microtick counter without bound⁴⁾. Explicit use of the SDL 'now' construct returns the value of this microtick counter. The difference between the timestamps of two events represents the number of microticks that have occurred between the events.

5.3.3.4 Channel-specific process replication

The FlexRay protocol described in this specification is a dual channel protocol. Several of the mechanisms in the protocol are replicated on a channel basis, i.e., essentially identical mechanisms are executed, one for channel A and one for channel B. This specification only provides SDL descriptions for the channel-specific processes on channel A - it is assumed that whenever a channel-specific process is defined for channel A there is another, essentially identical, process defined for channel B, even though this process is not explicitly described in the specification.

Channel-specific processes have names that include the identity of their channel (for example, "Clock synchronisation startup process on channel A [CSS_A]"). In addition, some signals that leave a channel-specific process have signal names that include the identity of their channel (for example, the signal *integration aborted on A*)⁵⁾.

5.3.3.5 Handling of priority input symbols

The SDL language contains certain ambiguities regarding the order of execution of processes if multiple processes have input queues that are not empty. For example, the usage of timers and clock oscillator inputs causes multiple processes to be eligible for execution at the beginning of clock edges. Generally, this poses no problem for the FlexRay specification, but for certain special cases it is not possible to specify the required behaviour in an unambiguous way without additional language constructs.

To resolve these situations the SDL priority input symbol is used, but with a slightly extended meaning. Whenever an input priority symbol is used, no other exit path of this state may be taken unless it is impossible that the priority input could be triggered on the current microtick clock edge. Effectively, the execution of the

3) If the duration time expression is zero or negative then the timer is started and expires immediately.

4) This is in contrast to the *vMicrotick* variable, which is reset to zero at the beginning of each cycle.

5) It is also possible for a signal leaving a channel-specific process to have a name that does not identify the channel. In such cases, a process that receives the SDL signal should behave identically regardless of which process sent the signal (i.e., the process receiving the signal effectively OR's the signals from all of the sending processes).

process in question is stalled until all other process have executed. Should multiple processes be in a state where they are sensitive to a priority input, all are executed last and in random order. The message queue is handled in the standard way, i.e. the signal triggering the priority input is removed from the queue while any signals placed before or after are preserved for the succeeding state.

5.3.3.6 Signals to non-instantiated processes

In various portions of the SDL behavioural descriptions the SDL sends signals that are received by a process that in some conditions may not be instantiated. As an example, the SDL in Figure 30 generates the signal *CODEC control on B (NORMAL)* even if the node is only attached to channel A (implying that the CODEC_B process is not instantiated). The sending of these signals should not be interpreted as requiring that processes that receive the signal should be instantiated - in such cases these signals should simply be ignored. This convention holds even if the only process that consumes the signal in question is a process that is not already instantiated.

5.3.3.7 Exported and imported signals

Certain features of the FlexRay protocol require that a certain direct communication between the two communication controllers of a time gateway is modeled within the SDL diagrams (for example, see 5.6.4). Signals marked with the EXP keyword are distributed within the local communication controller like any other signal, but are in addition also forwarded to a second communication controller that is represented by a separate instance of the SDL diagrams. On the receiving end, these exported signals can be received by input symbols marked with the IMP keyword. Input symbols marked in this way are only sensitive to signals emitted with the EXP keyword of the other communication controller.

5.4 Bit rates

The FlexRay Communications System specifies three standard bit rates – 10 Mbit / s (corresponding to a nominal bit duration, *gdBit*, of 100 ns), 5 Mbit / s (corresponding to a *gdBit* of 200 ns), and 2,5 Mbit / s (corresponding to a *gdBit* of 400 ns). In order to be considered FlexRay conformant, a protocol implementation is required to support all three standard bit rates.

5.5 Roles of a node in a FlexRay cluster

There are three distinct roles a node can perform.

- The role of a sync node enables a node to actively participate in the clock synchronisation algorithm performed by the cluster. Sync nodes transmit sync frames that are evaluated by all nodes of the cluster to perform an alignment of clock rate, that effectively determines the cycle length, and clock offset, that effectively determines the position of the cycle start.
- The role of a coldstart node enables a node to initiate the communication. Coldstart nodes are allowed to start transmitting startup frames in the non-synchronized state with the intent of establishing a schedule. Nodes integrate onto that new schedule by evaluating the content and timing of the received startup frames. A coldstart node is always also a sync node and a startup frame is always also a sync frame.
- A node that is neither a coldstart node nor a sync node is referred to as non-sync node. It performs no special task.

5.6 Synchronisation methods

5.6.1 General

A FlexRay node supports three different synchronisation modes. The behaviour of the cluster depends on the employed synchronisation mode of the nodes in the cluster.

5.6.2 TT-D synchronisation method

A cluster in which the coldstart nodes use the TT-D synchronisation method is a TT-D cluster. The TT-D synchronisation method uses a distributed algorithm to reduce the effect of any single failure. No critical task depends on any single node. A distributed startup instigated and carried through by two to fifteen coldstart nodes mitigates many adverse effects a single faulty coldstart node can have (see clause 11). A distributed clock synchronisation algorithm actively driven by two to fifteen sync nodes is robust against a number of Byzantine faults depending on the number of currently active sync nodes (see clause 12).

The advantage of the TT-D synchronisation method over the others is increased fault-tolerance.

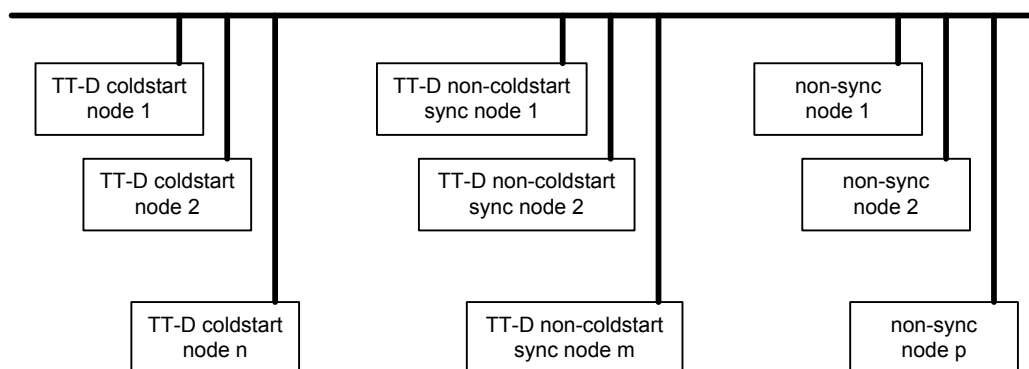


Figure 2 — TT-D cluster

Figure 2 shows the configuration of nodes in a TT-D cluster. The number of TT-D coldstart nodes n shall be equal to or greater than 2 and the sum of the number of TT-D coldstart nodes n and the number of TTD non-coldstart sync nodes m shall be equal to or less than fifteen. The number of non-sync nodes p is not limited by the protocol.

5.6.3 TT-L synchronisation method

A cluster in which the sole coldstart node uses the TT-L synchronisation method is a TT-L cluster. The TTL synchronisation method is a modification of the TT-D synchronisation method that reduces the number of required coldstart nodes from two to one. The single TT-L coldstart node in a TT-L cluster essentially behaves like two regular TT-D coldstart nodes by transmitting two startup frames. In this way non-sync nodes of the TT-L cluster will behave as if they were placed in a TT-D cluster with two TT-D coldstart nodes regularly transmitting their startup / sync frames and will integrate and operate normally, unaware of the fact that the two frames they receive actually come from the same node. The schedule and timing of such a TTL cluster will depend entirely on the single TT-L coldstart node.

The advantages of the TT-L synchronisation method are a reduced system complexity, a slightly reduced startup time, and an improved precision.

Figure 3 shows the configuration of nodes in a TT-L cluster. There exists exactly one TT-L coldstart node. The number of non-sync nodes p is not limited by the protocol.

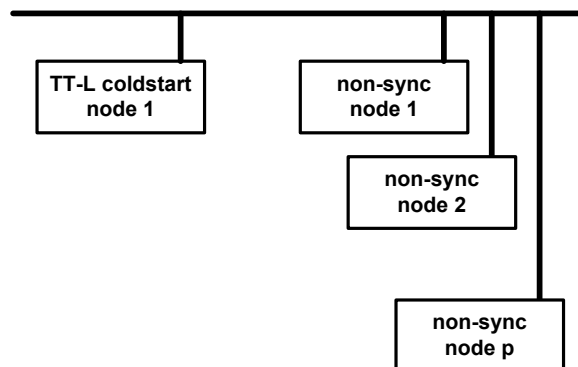


Figure 3 — TT-L cluster

5.6.4 TT-E synchronisation method

A cluster in which the coldstart nodes use the TT-E synchronisation method is a TT-E cluster. The primary intent of the TT-E synchronisation method is to synchronize the schedule of the TT-E cluster, also called a time sink cluster, to a second FlexRay cluster, which is referred to as time source cluster. To that end, each TT-E coldstart node, also called a time gateway sink node, shall be paired with a node of its time source cluster; this pair of nodes is called a time gateway. The node on the time sink cluster side is then called a time gateway sink node while the node on the time source cluster side is called time gateway source node.

Figure 4 depicts the basic setup. Depending on the synchronisation method employed by time source cluster, the time gateway source node may be a TT-D coldstart node, a TT-D non-coldstart sync node, a TT-L coldstart node, or a non-sync node⁶⁾.

The two nodes of the time gateway are connected via a time gateway interface, which is used by the time gateway source node to provide information about the schedule of the time source cluster to the time gateway sink node.

Instead of the usual distributed startup, a TT-E coldstart node derives the cycle length and position of the cycle start from its time gateway source node and directly starts transmitting according to this schedule (slightly shifted with a fixed offset of $cdT_{SrcCycleOffset}$ microticks).

Similar to the TT-L synchronisation method, each TT-E coldstart node transmits two startup frames, so that a single TT-E coldstart node suffices to start and maintain a TT-E cluster. Contrary to the TT-L synchronisation method, multiple TT-E coldstart nodes may be present in a TT-E cluster. As all TT-E coldstart nodes derive their schedule from the same cluster, they are implicitly synchronized to one another.

6) In theory it is also possible for the time gateway source node to be a TT-E coldstart node, but this would imply a time gateway that includes three or more distinct nodes. Such configurations are beyond the scope of this specification.

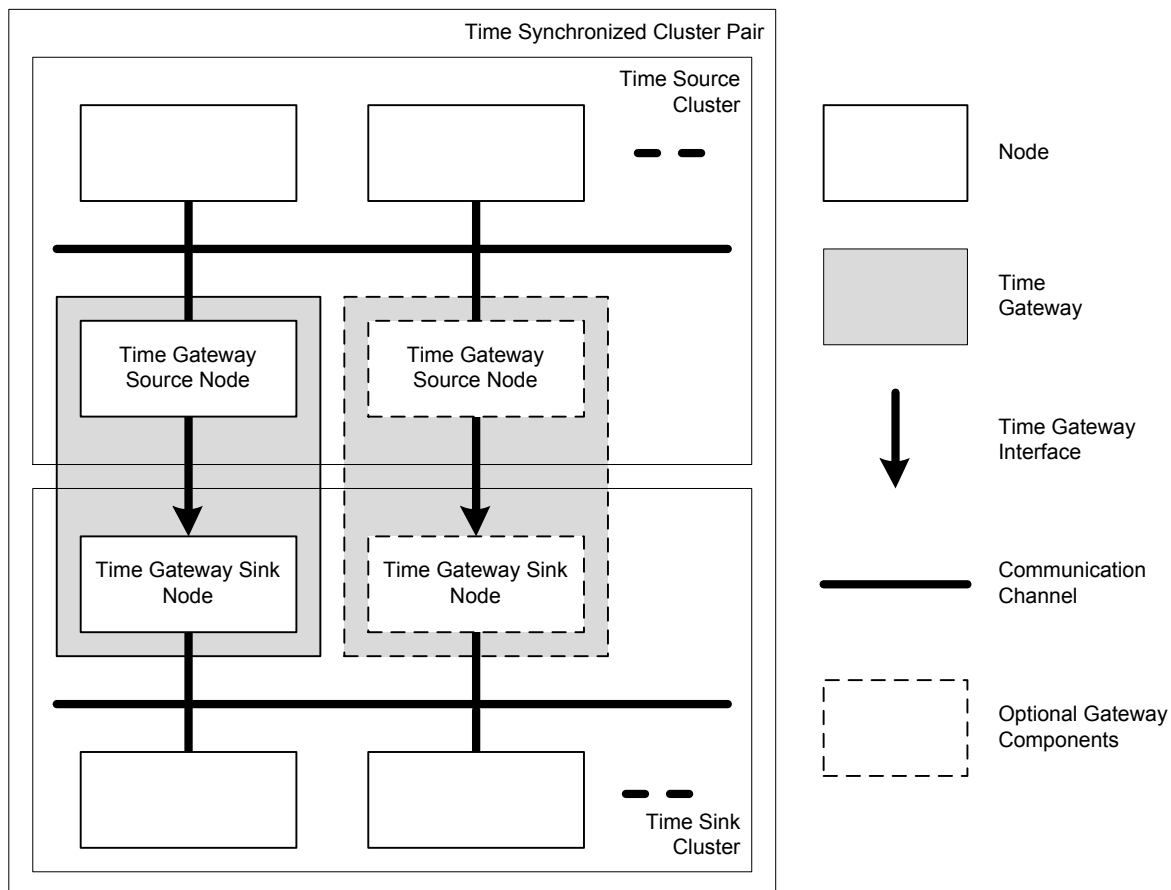


Figure 4 — Time synchronized cluster pair

The advantage of the TT-E synchronisation method is the close coupling of the schedule of a TT-E cluster to another FlexRay cluster. In this way, a single FlexRay cluster may be split into synchronized sub-clusters to avoid limits on attached nodes placed upon a single FlexRay cluster by ISO 17458-4 or to enable a separation of nodes into multiple clusters according to communication needs for a more efficient use of the available bandwidth.

Figure 5 shows the configuration of two connected clusters, the lower being a time sink cluster, the upper being a time source cluster, in this case a TT-D cluster. The TT-D cluster could also be replaced by a TT-L cluster or TT-E cluster⁷⁾. The number of TT-E coldstart nodes 'i' shall be at least one and less than or equal to 7. The number of non-sync nodes 'k' is not limited by the protocol.

The support of the TT-E synchronisation method is optional. This means that a FlexRay node may not support being a TT-E coldstart node, may not support being a time gateway source node, or may support neither of these features.

7) This specification does not provide configuration constraints for a "daisy-chain" of TT-E clusters. All configuration constraints assume that a time source cluster is either a TT-D cluster or a TT-L cluster.

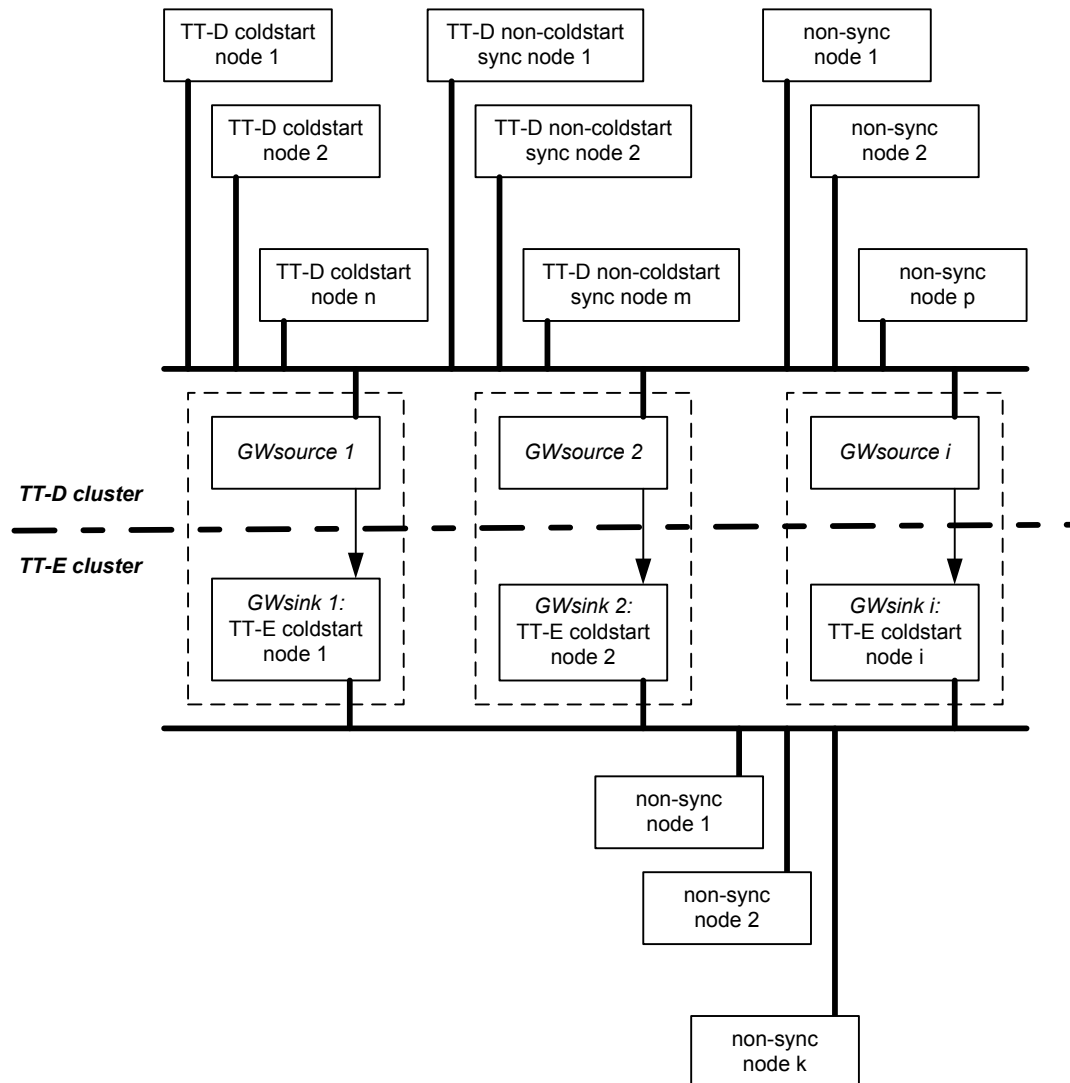


Figure 5 — TT-E cluster

Optional behaviour related to the feature of being a TT-E coldstart node is marked by dashed rectangles within the SDL diagrams. Each such rectangle is additionally annotated with the text "TT-E time gateway sink behaviour (optional)". An implementation not supporting the TT-E synchronisation method may choose not to implement the SDL content marked as optional. Further, such an implementation shall behave as if the value of the variable *pExternalSync* and in consequence also of the variable *vExternalSync* is fixed to false⁸⁾.

5.7 Network topology considerations

5.7.1 General

The following subclauses provide a brief overview of the possible topologies for a FlexRay system. This material is for reference only - detailed requirements and specifications may be found in ISO 17458-4.

8) As a result, it would be possible to redraw the SDL diagrams to remove the optional TT-E behaviour by eliminating all decision boxes involving *pExternalSync* or *vExternalSync* (which under these circumstances have a pre-determined outcome) and all optional material inside the dashed boxes. The remainder is the required behaviour for all FlexRay implementations.

There are several ways to design a FlexRay cluster. It can be configured as a single-channel or dual-channel bus network, a single-channel or dual-channel star network, or in various hybrid combinations of bus and star topologies.

A FlexRay cluster consists of at most two channels, identified as Channel A and Channel B. Each node in the cluster may be connected to either or both of the channels. In the fault free condition, all nodes connected to Channel A are able to communicate with all other nodes connected to Channel A, and all nodes connected to Channel B are able to communicate with all other nodes connected to Channel B. If a node needs to be connected to more than one cluster then the connection to each cluster shall be made through a different communication controller⁹⁾.

5.7.2 Passive bus topology

Figure 6 shows the possible topology configuration of the communication network as a dual bus. A node can be connected to both channels A and B (nodes A, C, and E), only to channel A (node D), or only to channel B (node B).

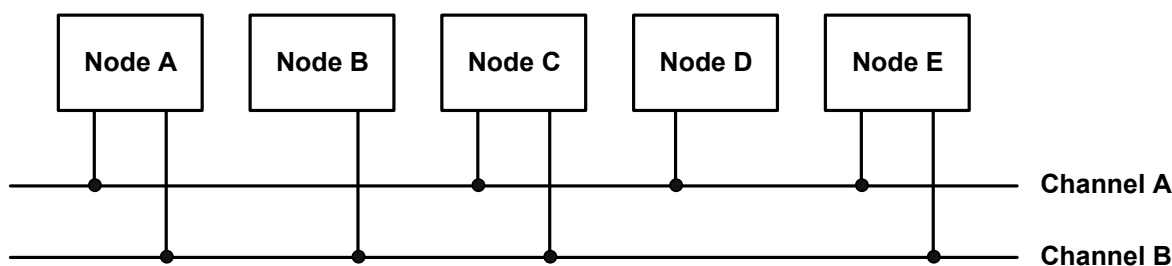


Figure 6 — Dual channel bus configuration

The FlexRay communication network can also be a single bus. In this case, all nodes are connected to this bus.

5.7.3 Active star topology

A FlexRay communication network can be built as a multiple star topology. Similar to the bus topology, the multiple-star topology can support redundant communication channels. Each network channel shall be free of closed rings, and there can be no more than two active stars on a network channel¹⁰⁾.

NOTE there may be physical layer-related restrictions that limit the number of active stars for certain bit rates - see ISO 17458-4 for details.

An incoming signal received on a branch of an active star is actively driven to all other branches of the active star.

The configuration of a single redundant star network is shown in Figure 7. The logical structure (i.e., the node connectivity) of this topology is identical with that shown in Figure 6. It is also possible to create a single, non-redundant star topology that has the same logical structure as the single bus mentioned above.

9) For example, it is not allowed for a communication controller to connect to Channel A of one cluster and Channel B of another cluster.

10) A channel with two active stars would have the stars connected to each other. Communication between nodes connected to different stars would pass through both stars (a cascaded star topology).

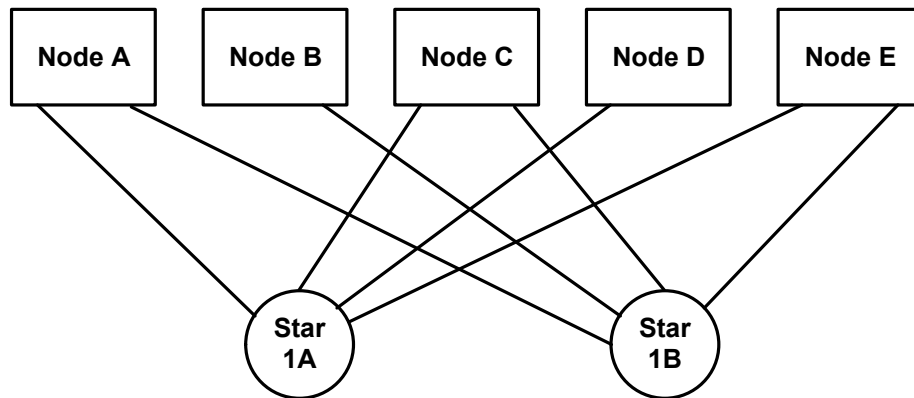


Figure 7 — Dual channel single star configuration

Figure 8 shows a single channel network built with two active stars. Each node has a point-to-point connection to one of the two active stars. The first active star (1A) is directly connected to the second active star (2A).

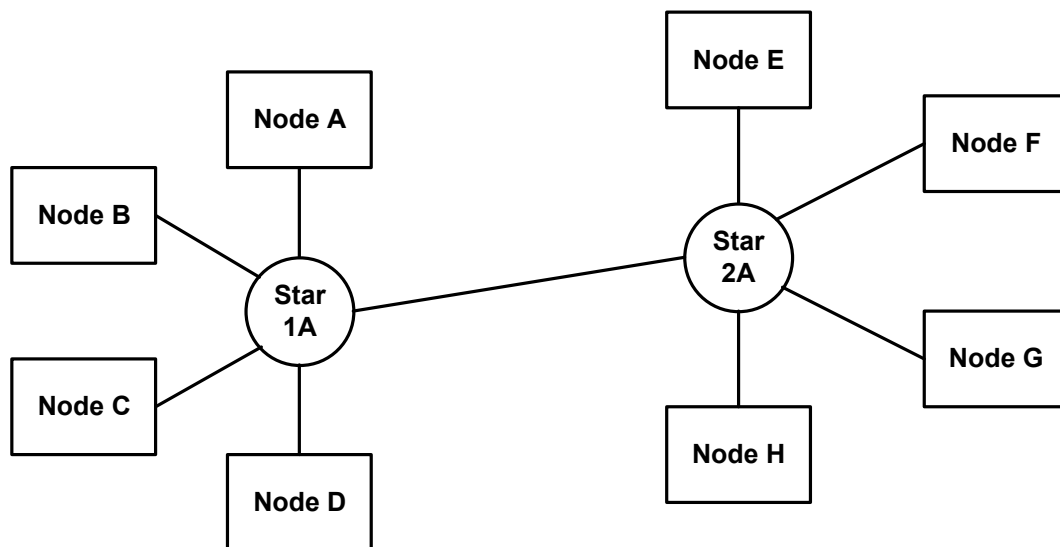


Figure 8 — Single channel cascaded star configuration

NOTE1 it is also possible to have a redundant channel configuration with cascaded stars. An example of such a configuration is Figure 9.

NOTE2 this example does not simply replicate the set of stars for the second channel - Star 1A connects nodes A, B, and C, while Star 1B connects nodes A, C, and E.

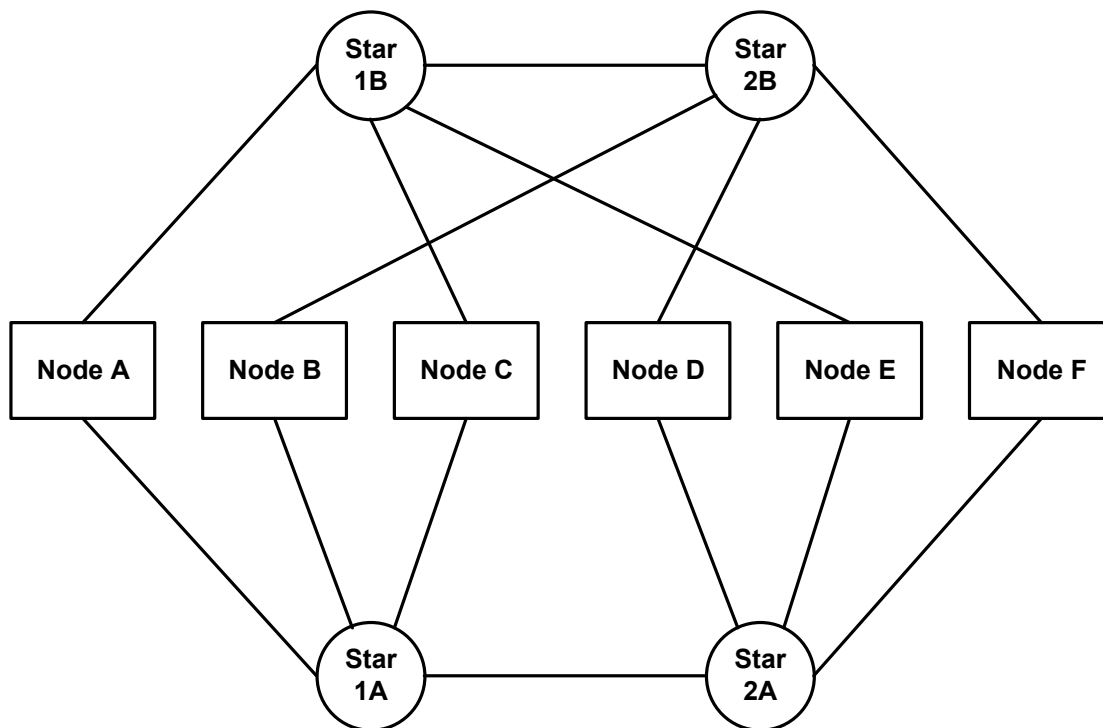


Figure 9 — Dual channel cascaded star configuration

5.7.4 Active star topology combined with a passive bus top

In addition to topologies that are composed either entirely of a bus topology or entirely of a star topology, it is possible to have hybrid topologies that are a mixture of bus and star configurations. The FlexRay system supports such hybrid topologies as long as the limits applicable to each individual topology are not exceeded. For example, the limit of two cascaded active stars also limits the number of cascaded active stars in a hybrid topology.

There are a large number of possible hybrid topologies, but only two representative topologies are shown here. Figure 10 shows an example of one type of hybrid topology. In this example, some nodes (nodes A, B, C, and D) are connected using point-to-point connections to an active star. Other nodes (nodes E, F, and G) are connected to each other using a bus topology. This bus is also connected to an active star, allowing nodes E, F, and G to communicate with the other nodes.

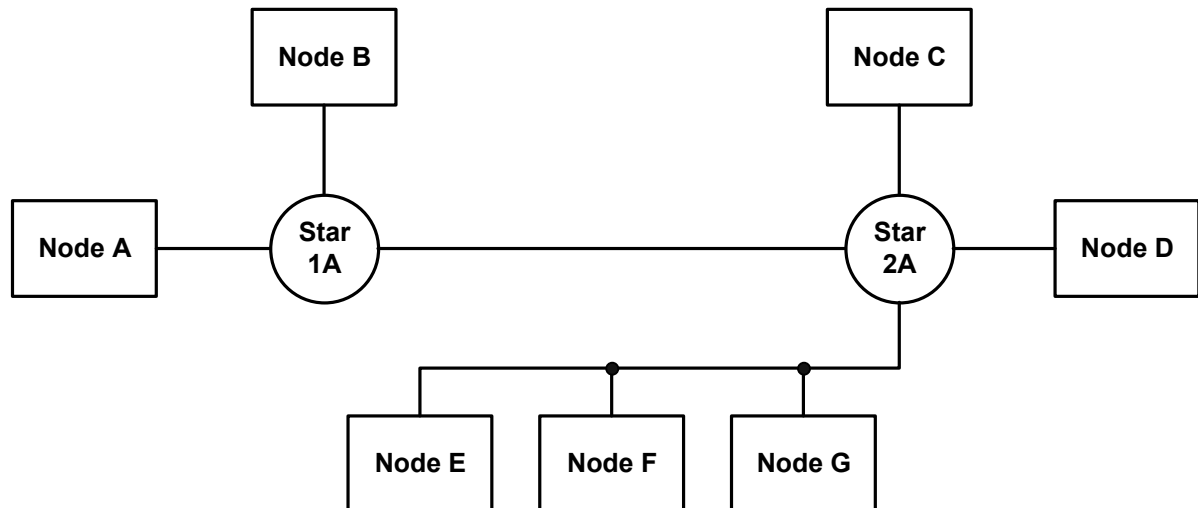


Figure 10 — Single channel hybrid example

A fundamentally different type of hybrid topology is shown in Figure 11. In this case, different topologies are used on different channels. Here, channel A is implemented as a bus topology connection, while channel B is implemented as a star topology connection.

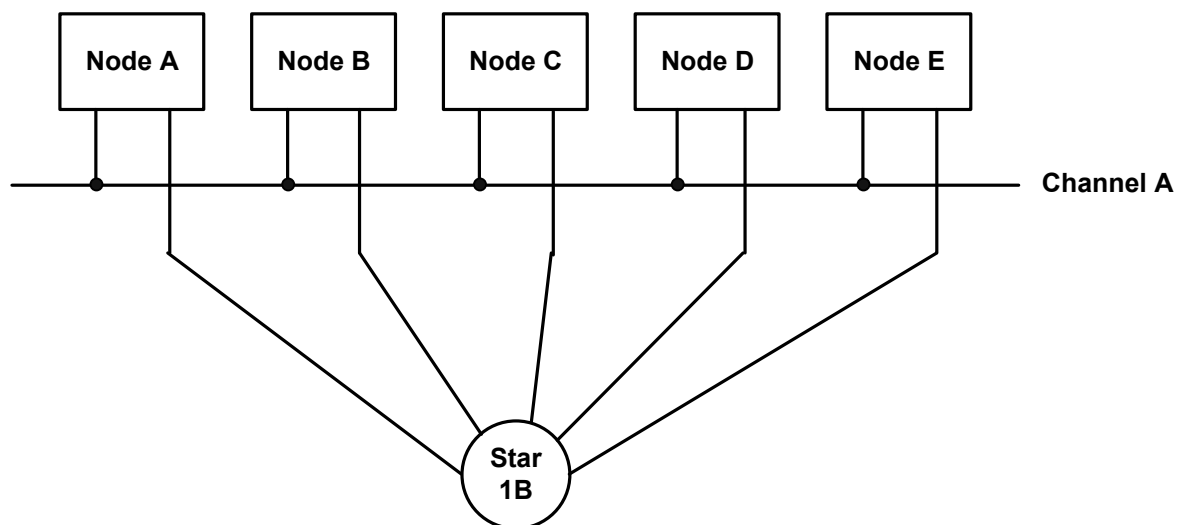


Figure 11 — Dual channel hybrid example

The protocol implications of topologies with stubs on the connection between active stars have not been fully analyzed. As a result, such topologies are not recommended and are not considered in this specification.

5.8 Example node architecture

5.8.1 Objective

This subclause is intended to provide insight into the FlexRay architecture by discussing an example node architecture and the interfaces between the FlexRay hardware devices.

The information in this subclause is for reference only. The detailed specification of the interfaces is given in the electrical physical layer specification ISO 17458-4; references are made here to appropriate text passages from this document.

Note that an active star component can also function in a role similar to a bus driver via the use of the optional BD-CC interface as specified in ISO 17458-4. The following subclauses describe the node architecture under the assumption that the CC interfaces to the channel(s) via a bus driver rather than via an active star.

5.8.2 Overview

Figure 12 depicts an example node architecture. One communication controller, one host, one power supply unit, and two bus drivers are depicted. Each communication channel has one bus driver to connect the node to the channel. In addition to the indicated communication and control data interfaces an optional interface between the bus driver and the power supply unit may exist.

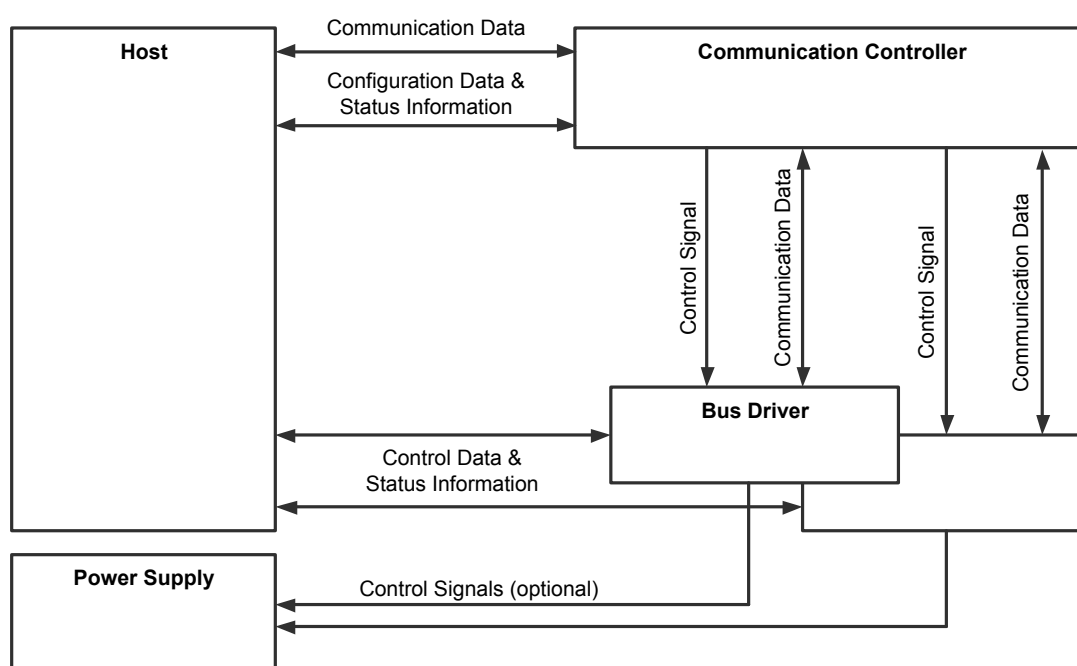


Figure 12 — Logical interfaces

5.8.3 Host - communication controller interface

The host and the communication controller share a substantial amount of information. The host provides control and configuration information to the CC and provides payload data that is transmitted during the communication cycle. The CC provides status information to the host and delivers payload data received from communication frames. Figure 13 illustrates the Host and the Communication Controller.

Details of the interface between the host and the communication controller are specified in clause 13.

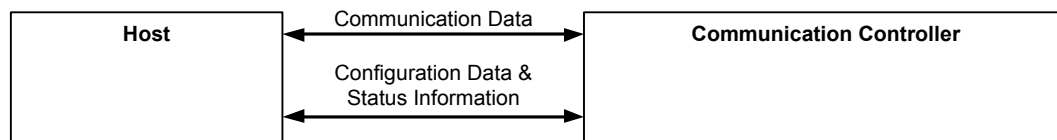


Figure 13 — Host - communication controller interfaces

5.8.4 Communication controller - bus driver interface

The interface between the BD and the CC consists of three digital electrical signals. Two are outputs from the CC (TxD and TxEN) and one is an output from the BD (RxD).

The CC uses the TxD (Transmit Data) signal to transfer the actual signal sequence to the BD for transmission onto the communication channel. TxEN (Transmit Data Enable Not) indicates the CC's request to have the bus driver present the data on the TxD line to its corresponding channel.

The BD uses the RxD (Receive Data) signal to transfer the actual received signal sequence to the CC.

Figure 14 shows the connection between the communication controller and the bus driver and the internal connection between the protocol engine and the pins. This data link layer specification does not specify a device but the behaviour of the FlexRay protocol. In the following the data link layer specification only refers to the internal signals TxD, TxEN, and RxD as depicted in Figure 14.

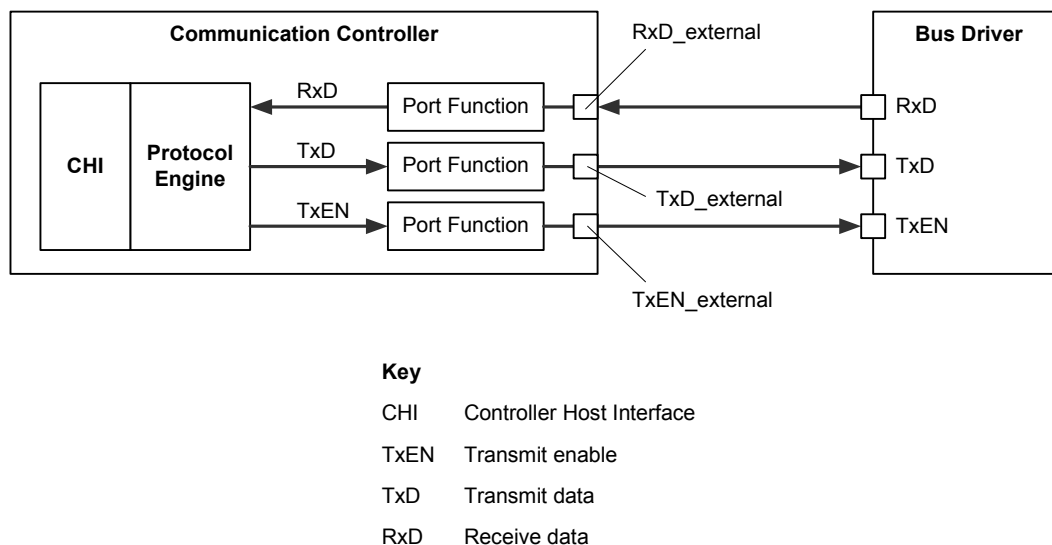


Figure 14 — Communication controller - bus driver interface

Between the internal and the external signals there are device specific port functions which are responsible for the electrical behavior of the pins, for example

- I/O voltage level,
- ESD protection,
- behaviour during power up initialization, reset, or while depowered, and
- pin multiplexing (e.g. the connection of the external pins associated with the RxD_external, TxD_external and TxEN_external signals either to the FlexRay protocol engine (i.e., the RxD, TxD, and TxEN signals, respectively), to some other function inside the CC implementation¹¹⁾, or to nothing at all).

If the pins are connected to something other than the FlexRay protocol engine this specification places no requirements on the behavior of those pins. However, the behavior during power up initialization, reset, while depowered, and the default behavior prior to the configuration of any pin multiplexing, shall ensure that the bus driver does not actively drive the FlexRay bus¹²⁾, and that the bus driver interprets the TxD signal as low¹³⁾.

Some requirements (e.g. the electrical characteristics and timing) on the port functions and the TxD_external, TxEN_external and RxD_external signals are specified in ISO 17458-4.

11) For example, if the CC implementation is part of a microcontroller it is possible that the microcontroller could be configured to use I/O pins either as the FlexRay I/O (RxD, TxD, and TxEN) or for some other purpose (perhaps general purpose I/O).

12) This could be done, for example, by ensuring that the TxEN_external output is driven to active high, provided with a weak pull up, or set to high impedance.

13) This could be done, for example, by ensuring that the TxD_external output is driven to active low, provided with a weak pull down, or set to high impedance.

5.8.5 Bus driver - host interface

5.8.5.1 Overview

The interface between the BD and the host allows the host to control the operating modes of the BD and to read error conditions and status information from the BD.

This interface can be realized using hard-wired signals (see option A in Figure 15) or by a Serial Peripheral Interface (SPI) (see option B in Figure 16).

5.8.5.2 Hard wired signals (option A)

This implementation of the BD - host interface uses discrete hard wired signals. The interface consists of at least an STBN (Standby Not) signal that is used to control the BD's operating mode and an ERRN (Error Not) signal that is used by the BD to indicate detected errors. The interface could also include additional control signals (the "EN" signal is shown as an example) that support control of optional operational modes.

Figure 15 illustrates an example of a bus driver with a host interface option A.

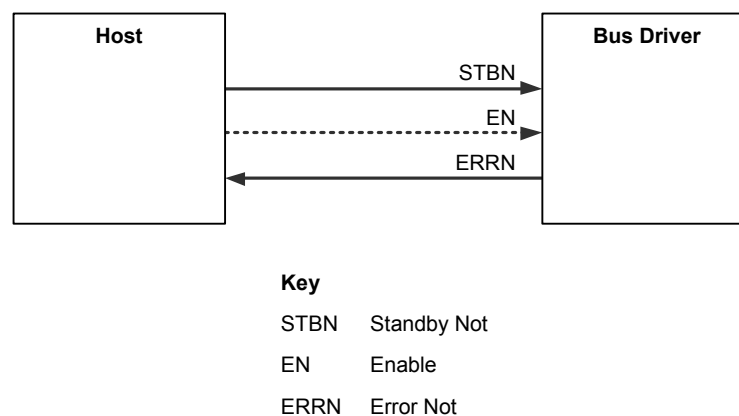


Figure 15 — Example bus driver - host interface (option A)

This interface is product specific; some restrictions are given in ISO 17458-4 that define minimum functionality to ensure interoperability.

5.8.5.3 Serial peripheral interface (SPI) (option B)

This implementation of the BD - host interface uses an SPI link to allow the host to command the BD operating mode and to read out the status of the BD. In addition, the BD has a hardwired interrupt output (INTN).

Figure 16 illustrates an example of a bus driver with a host interface option B.

The electrical characteristics and timing of this interface are specified in ISO 17458-4.

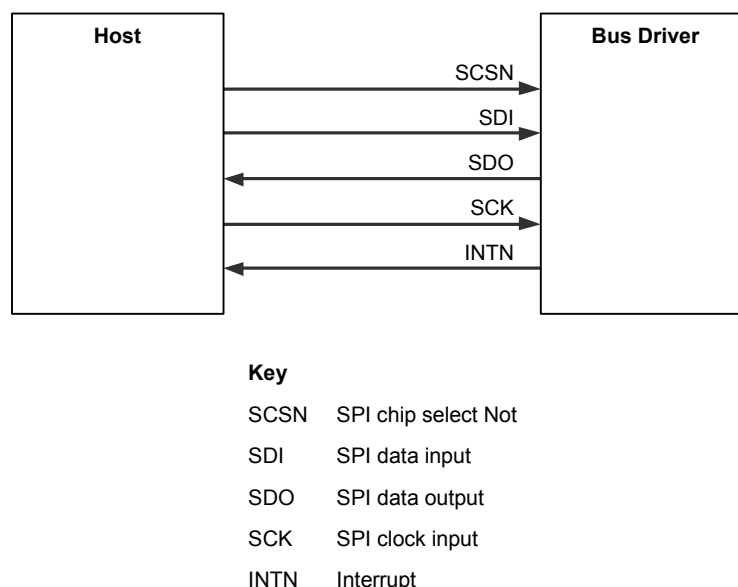


Figure 16 — Example bus driver - host interface (option B)

5.8.6 Bus driver - power supply interface (optional)

The inhibit signal (INH1) is an optional interface that allows the BD to directly control the power supply of an ECU. This signal could also be used as one of a set of signals that control the power moding of the ECU.

Figure 17 illustrates an example of a bus driver with a power supply interface.

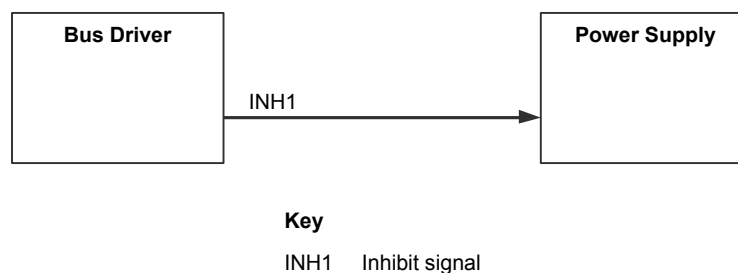


Figure 17 — Bus driver - power supply interface

The electrical characteristics and behaviour of the INH1 signal are specified in ISO 17458-4.

5.8.7 Time gateway interface

A time gateway sink node needs information on the schedule and clock synchronisation algorithm of its time gateway source node. The time gateway source node provides this information via the time gateway interface

to the time gateway sink node. This interface is unidirectional - no information flows back from the time gateway sink node to the time gateway source node. This interface is an optional feature only required to allow the node to be a time gateway source or time gateway sink node. Details of the interface between the time gateway source node and the time gateway sink node are specified in clause 12. The usage of this interface is indicated in the SDL diagrams by the EXP keyword on the transmitting end and the IMP keyword on the receiving end.

Figure 18 illustrates a time gateway interface.

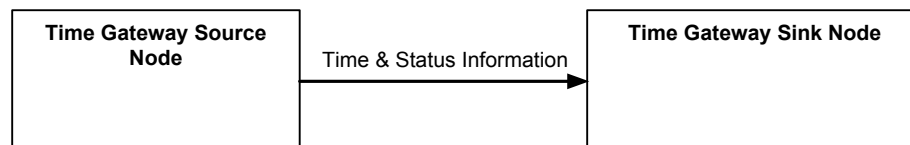


Figure 18 — Time gateway interface

5.8.8 Testability requirements

ISO 17458-3 contains additional implementation requirements. The purpose of these requirements is to facilitate testing, for example by establishing timing bounds for the availability of CHI information necessary to execute certain tests.

6 Protocol operation control

6.1 Principles

6.1.1 General

This subclause defines how the core mechanisms of the protocol are moded in response to host commands and protocol conditions.

The primary protocol behaviour of FlexRay is embodied in four core mechanisms, each of which is described in a dedicated subclause of this specification for

- Coding and Decoding (see clause 7),
- Media Access Control (see clause 9),
- Frame and Symbol Processing (see clause 10), and
- Clock Synchronisation (see clause 12)

In addition, the controller host interface (CHI) provides the mechanism for the host to interact in a structured manner with these core mechanisms and for the protocol mechanisms, including Protocol Operation Control (POC), to provide feedback to the host (see clause 13).

Each of the core mechanisms possesses modal behaviour that allows it to alter its fundamental operation in response to high-level mode changes of the node. Proper protocol behaviour can only occur if the mode changes of the core mechanisms are properly coordinated and synchronized. The purpose of the POC is to react to host commands and protocol conditions by triggering coherent changes to core mechanisms in a synchronous manner, and to provide the host with the appropriate status regarding these changes.

The necessary synchronisation of the core mechanisms is particularly evident during the wakeup, startup and reintegration procedures. These procedures are described in detail in clause 11. However, these procedures are wholly included in the POC as macros in the POC SDL models. They can be viewed as specialized extensions of the POC.

6.1.2 Communication controller power moding

Before the POC can perform its prescribed tasks the communication controller (CC) shall achieve a state where there is a stable power supply. Furthermore, the POC can only continue to operate while a stable power supply is present.

Figure 19 depicts an overview of the CC power moding operation.

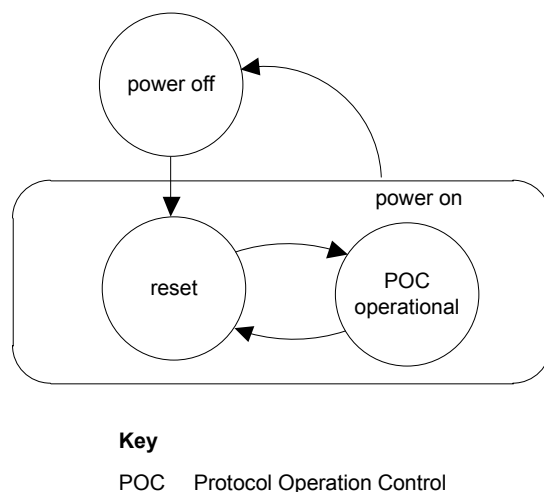


Figure 19 — Power moding of the communication controller

NOTE Figure 19 illustrates the state labelled *POC operational* which is not actually a specific state but rather a superset of all operational states of the protocol operation control (see Figure 21).

In the *power off* state there is insufficient power for the CC to operate¹⁴⁾. In the *power on* state (including both *reset* and *POC operational*) the CC shall guarantee that all pins are in prescribed states. In the *POC operational* state the CC shall drive the pins in accordance with the product specification. The POC controls the other protocol mechanisms in the manner described in this subclause while the CC is in the *POC operational* state.

14) While the CC cannot enforce specific behaviour of the pins, there shall be product-specific behaviour specified (e.g. high impedance).

6.2 Description

6.2.1 Protocol operation control context

The relationships between the CHI, POC and the core mechanisms are depicted in Figure 20¹⁵⁾.

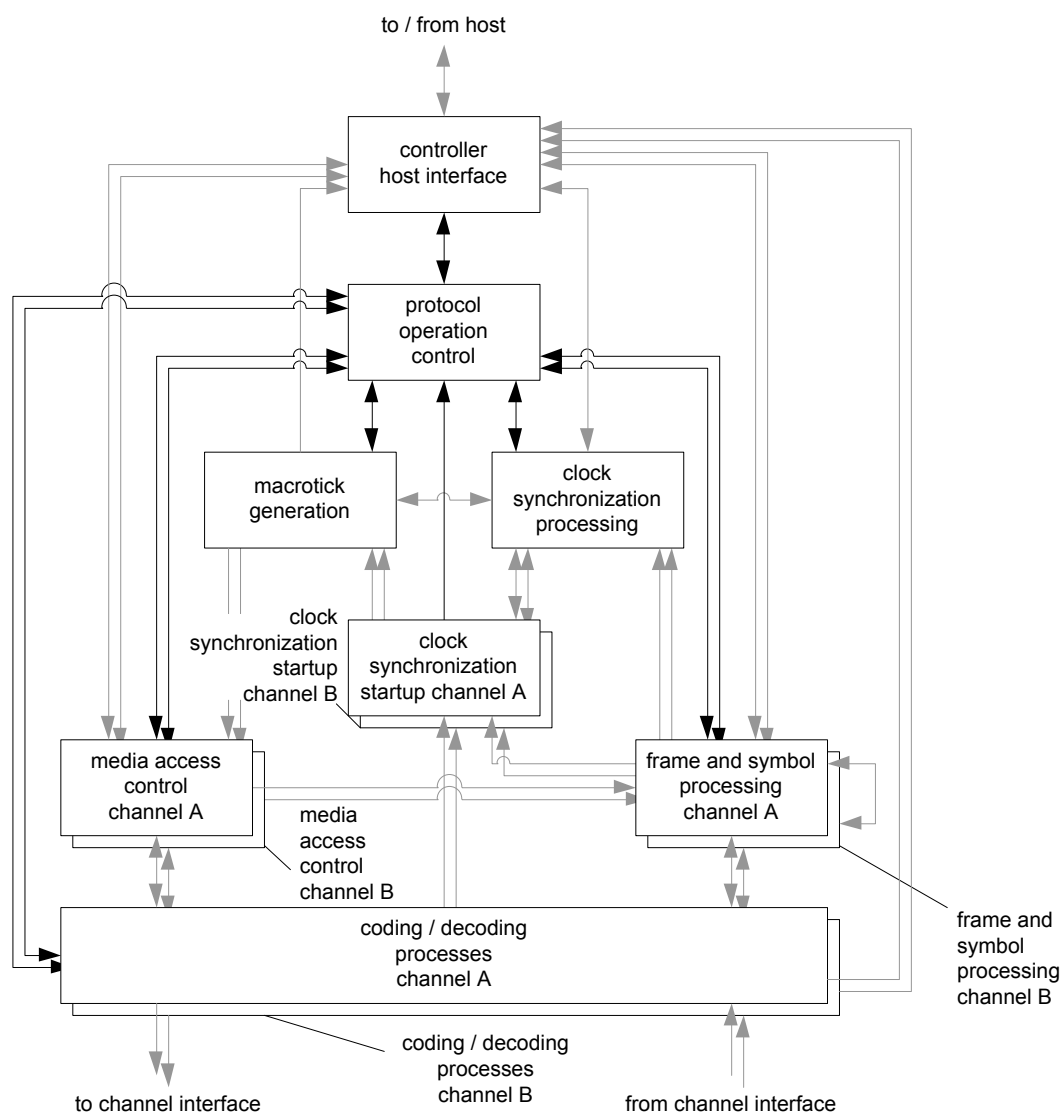


Figure 20 — Protocol operation control context

15) The dark lines represent data flows between mechanisms that are relevant to this subclause. The lighter gray lines are relevant to the protocol, but not to this clause.

6.2.2 Operational overview

6.2.2.1 General

The POC SDL process is created as the CC enters the *POC operational* state and terminated when the CC exits it. The POC process is responsible for creating the SDL processes corresponding to the core mechanisms and informing those processes when they are required to terminate. It is also responsible for changing the mode of the core mechanisms of the protocol in response to changing conditions in the node.

Mode changes of the core mechanisms occur when the POC itself changes states. Some of the POC state changes are simply a consequence of completing tasks. For example, the *POC:normal active* state (see 6.3.7) is entered as a consequence of completing the startup process. However, most of the POC state changes are a direct consequence of one of the following:

- Host commands communicated to the POC via the CHI;
- Error conditions detected either by the protocol engine or a product-specific built-in self-test (BIST) or sanity check. The host may also perform sanity checks, but the consequences of the host sanity checks are indicated to the POC as host commands.

6.2.2.2 Host commands

Strictly speaking, the POC is unaware of the commands issued by the host. Host interactions with the CC are processed by the CHI. The CHI is responsible for relaying relevant commands to the POC. While this is a minor distinction, the remainder of the POC description in this document treats the host commands as if they originated in the CHI. Similarly, status information from the POC that is intended for the host is simply provided to the CHI, which is then responsible for formatting it appropriately and relaying it to the host in a prescribed manner (see clause 13).

Some host commands result in immediate changes being reflected in the moding of the core mechanisms while mode changes are deferred to the end of the communication cycle for others. In addition, some host commands are not processed in every POC state. The detailed behaviour corresponding to each command is captured in the SDL descriptions and accompanying text (see 6.3). They are briefly summarized in Table 4. If the host issues a specific CHI command while the POC is in a state other than the states shown in the "Where processed (POC States)" column in Table 4 the command shall be ignored (i.e., it shall have no effect on the protocol engine).

Table 4 — CHI host command summary

CHI command	Where processed (POC States)	When processed
ALL_SLOTS	<i>POC:normal active, POC:normal passive</i>	End of cycle
ALLOW_COLDSTART	All except: <i>POC:default config, POC:config, POC:halt, POC:wakeup listen, POC:wakeup send, POC:wakeup detect^a</i>	Immediate
CLEAR_DEFERRED	All except: <i>POC:default config, POC:config, POC:ready, POC:halt</i>	Immediate
CONFIG	<i>POC:default config, POC:ready</i>	Immediate
CONFIG_COMPLETE	<i>POC:config</i>	Immediate
DEFAULT_CONFIG	<i>POC:halt</i>	Immediate
RUN	<i>POC:ready</i>	Immediate
WAKEUP	<i>POC:ready</i>	Immediate
FREEZE	All	Immediate
IMMEDIATE_READY	All except: <i>POC:default config, POC:config, POC:ready, POC:halt</i>	Immediate
DEFERRED_READY	All except: <i>POC:default config, POC:config, POC:ready, POC:halt, POC:normal active, POC:normal passive, POC:wakeup send, POC:coldstart collision resolution, POC:coldstart consistency check, POC:coldstart join</i>	Immediate
DEFERRED_READY	<i>POC:normal active, POC:normal passive, POC:coldstart collision resolution^b, POC:coldstart consistency check, POC:coldstart join</i>	End of cycle
DEFERRED_READY	<i>POC:wakeup send</i>	After transmission of a complete WUP or detection of wakeup collision
DEFERRED_HALT	All except: <i>POC:halt, POC:normal active, POC:normal passive, POC:wakeup send, POC:coldstart collision resolution, POC:coldstart consistency check, POC:coldstart join</i>	Immediate
DEFERRED_HALT	<i>POC:normal active, POC:normal passive, POC:coldstart collision resolution^b, POC:coldstart consistency check, POC:coldstart join</i>	End of cycle
DEFERRED_HALT	<i>POC:wakeup send</i>	After transmission of a complete WUP or detection of wakeup collision

^a The ALLOW_COLDSTART command is processed as described in Figure 31 except when the POC is in the *POC:integration listen* state, in which case it is processed by the SDL in Figure 150.

^b In the *POC:coldstart collision resolution* state a deferred command is either processed at the end of the cycle or after a frame header or a CAS is received.

6.2.2.3 Error conditions

6.2.2.3.1 General

The POC contains two basic mechanisms for responding to errors. For significant errors, the *POC:halt* state is immediately entered. The POC also contains a three-state degradation model for errors that can be endured for a limited period of time. In this case entry to the *POC:halt* state is deferred, at least temporarily, to support possible recovery from a potentially transient condition.

6.2.2.3.2 Errors causing immediate entry to the *POC:halt* state

There are three general conditions that trigger entry to the *POC:halt* state:

- Product-specific error conditions such as BIST errors and sanity checks;
- Error conditions detected by the host that result in a FREEZE command being sent to the POC via the CHI;
- Fatal error conditions detected by the FSP process.

Product-specific errors are accommodated by the POC, but not described in this specification (see 6.3.4). Similarly, host detected error strategies are supported by the POC's ability to respond to a host FREEZE command (see 6.3.4), but the host-based mechanisms that trigger the command are beyond the scope of this specification. Only errors detected by the POC or one of the core mechanisms are explicitly detailed in this specification.

6.2.2.3.3 Errors handled by the degradation model

Integral to the POC is a three-state error handling mechanism referred to as the degradation model. It is designed to react to certain conditions detected by the clock synchronisation mechanism that are indicative of a problem, but that may not require immediate action due to the inherent fault tolerance of the clock synchronisation mechanism. This makes it possible to avoid immediate transitions to the *POC:halt* state while assessing the nature and extent of the errors.

The degradation model is embodied in three POC states - *POC:normal active*, *POC:normal passive*, and *POC:halt*.

In the *POC:normal active* state the node is assumed to be either error free, or at least within error bounds that allow continued "normal operation". Specifically, it is assumed that the node remains adequately time-synchronized to the cluster to allow continued frame transmission without disrupting the transmissions of other nodes.

In the *POC:normal passive* state, it is assumed that synchronisation with the remainder of the cluster has degraded to the extent that continued frame transmissions cannot be allowed because collisions with transmissions from other nodes are possible. Frame reception continues in the *POC:normal passive* state in support of host functionality and in an effort to regain sufficient synchronisation to allow a transition back to the *POC:normal active* state.

If errors persist in the *POC:normal passive* state or if errors are severe enough, the POC can transition to the *POC:halt* state. In this state it is assumed that recovery back to the *POC:normal active* state cannot be achieved, so the POC halts the core mechanisms in preparation for reinitializing the node.

The conditions for transitioning between the three states comprising the degradation model are configurable. Furthermore, transitions between the states are communicated to the host allowing the host to react appropriately and to possibly take alternative actions using one of the explicit host commands.

6.2.2.4 POC status

In order for the host to react to POC state changes, the host shall be informed when POC state changes occur. This is the responsibility of the CHI. The POC supports the CHI by providing appropriate information to the CHI.

Definition: *T_POCStatus*

(1)

The basic POC status information is provided to the CHI using the *vPOC* data structure. *vPOC* is of type *T_POCStatus*.

```
newtype T_POCStatus
struct
    State                T_POCState;
    Freeze               Boolean;
    CHIHaltRequest       Boolean;
    CHIReadyRequest      Boolean;
    ColdstartNoise       Boolean;
    SlotMode             T_SlotMode;
    ErrorMode            T_ErrorMode;
    WakeupStatus         T_WakeupStatus;
    StartupState         T_StartupState;
endnewtype;
```

Definition: *T_POCState*

(2)

The *vPOC* structure is an aggregation of nine distinct status variables. *vPOC!State* is used to indicate the state of the POC and is based on the *T_POCState*.

```
newtype T_POCState
    literals CONFIG, DEFAULT_CONFIG, HALT, NORMAL_ACTIVE, NORMAL_PASSIVE,
    READY, STARTUP, WAKEUP;
endnewtype;
```

vPOC!Freeze is used to indicate that the POC has entered the *POC:halt* state due to an error condition requiring an immediate halt (see 6.3.4). *vPOC!Freeze* is Boolean.

vPOC!CHIHaltRequest is used to indicate that a request has been received from the CHI to halt the POC at the end of the communication cycle (see 6.3.5.1). *vPOC!CHIHaltRequest* is Boolean.

vPOC!CHIReadyRequest is used to indicate that a request has been received from the CHI to enter the *POC:ready* state at the end of the communication cycle (see 6.3.5.1). *vPOC!CHIReadyRequest* is Boolean.

vPOC!ColdstartNoise indicates noisy channel conditions during *POC:coldstart listen* if the coldstart attempt of a leading coldstart node was completed successfully (see 11.3). *vPOC!ColdstartNoise* is Boolean.

Definition: *T_SlotMode*

(3)

vPOC!SlotMode is used to indicate what slot mode the POC is in (see 6.3.5.2, 6.3.8.2.3, and 6.3.8.2.4). *vPOC!SlotMode* is based on the *T_SlotMode* formal definition.

```
newtype T_SlotMode
    literals KEYSLOT, ALL_PENDING, ALL;
```

```
endnewtype;
```

Definition: *vPOC!ErrorMode* (4)

vPOC!ErrorMode is used to indicate what error mode the POC is in (see subclauses 6.3.8.2.3 and 6.3.8.2.4). *vPOC!ErrorMode* is based on the *T_ErrorMode* formal definition.

```
newtype T_ErrorMode
  literals ACTIVE, PASSIVE, COMM_HALT;
endnewtype;
```

Definition: *vPOC!WakeupStatus* (5)

vPOC!WakeupStatus is used to indicate the outcome of the execution of the WAKEUP mechanism (see Figure 30 and 11.2.3.2). *vPOC!WakeupStatus* is based on the *T_WakeupStatus* formal definition.

```
newtype T_WakeupStatus
  literals UNDEFINED, RECEIVED_HEADER, RECEIVED_WUP, COLLISION_HEADER,
  COLLISION_WUP, COLLISION_UNKNOWN, TRANSMITTED;
endnewtype;
```

The meaning of the individual *T_WakeupStatus* values is outlined in 11.2.3.2.

Definition: *vPOC!StartupState* (6)

vPOC!StartupState is used to indicate the current substate of the startup procedure (see 11.3.5). *vPOC!StartupState* is based on the *T_StartupState* formal definition.

```
newtype T_StartupState
  literals UNDEFINED, COLDSTART_LISTEN, INTEGRATION_COLDSTART_CHECK,
  COLDSTART_JOIN, COLDSTART_COLLISION_RESOLUTION,
  COLDSTART_CONSISTENCY_CHECK, INTEGRATION_LISTEN, INITIALIZE_SCHEDULE,
  INTEGRATION_CONSISTENCY_CHECK, COLDSTART_GAP, EXTERNAL_STARTUP;
endnewtype;
```

The individual *T_StartupState* values are the states within the STARTUP mechanism in 11.3.5.

In addition to the *vPOC* data structure, the POC makes two counters available to the host via the CHI. These counters are *vClockCorrectionFailed* and *vAllowPassiveToActive*, and are described in 6.3.8.2.5.

6.2.2.5 SDL considerations for single channel nodes

FlexRay supports configurations where a node is only attached to one of the two possible FlexRay channels (see 5.7).

Process instantiation is depicted in Figure 23. The channel specific processes are readily identifiable by the "_A" or "_B" text in the process names. The POC only instantiates the channel specific processes related to channels that are actually attached.

Process termination signal generation is also depicted in Figure 23. The channel specific signals are identifiable by the "_A" or "_B" text in the signal names. Termination signals sent to a channel specific process that is not instantiated will have no effect.

Process moding signals are generated throughout the POC. Figure 22 is an example that includes all of these moding signals. The channel specific moding signals are identifiable by the "on A" or "on B" text in the signal names. Moding signals, or any other signals, sent to a channel specific process that is not instantiated will have no effect.

6.3 The protocol operation control process

6.3.1 General

This subclause contains the formalized specification of the POC process. Figure 21 depicts an overview of the POC states and how they interrelate¹⁶⁾.

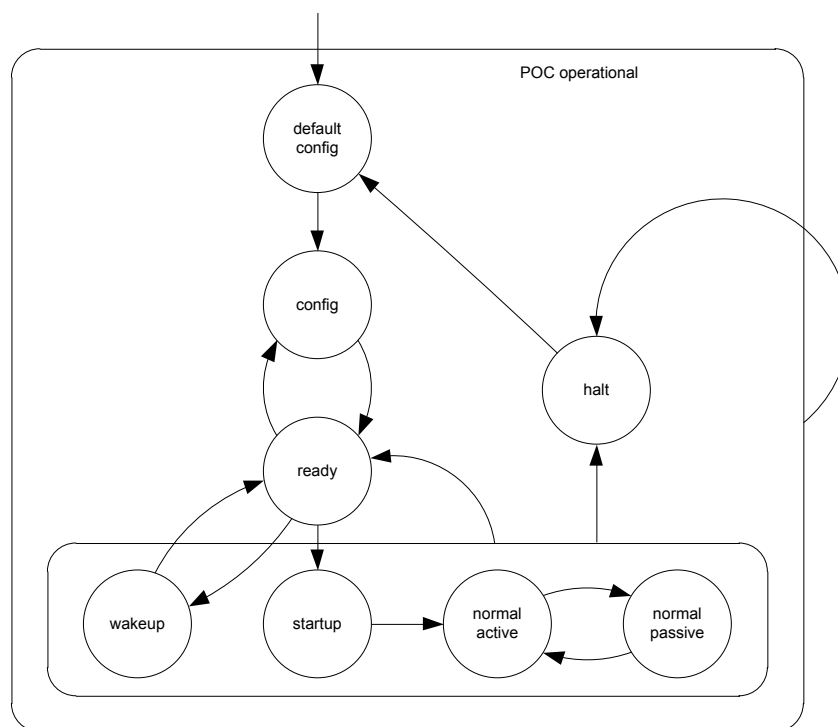


Figure 21 — Overview of protocol operation control

6.3.2 POC SDL utilities

The nature of the POC is that it performs tasks that often influence all of the core mechanisms simultaneously. From the perspective of SDL depiction these tasks are visually cumbersome. Macros are used in the POC for the sole purpose of simplifying the SDL presentation.

In the SDL that follows, there are several instances where the POC transitions to the *POC:ready* or *POC:halt* states. Prior to doing so, the core mechanisms have to be moded appropriately. The two macros in Figure 22 perform these tasks. `PROTOCOL_ENGINE_READY` modes the core mechanisms appropriately for entry to

16) The states depicted as wakeup and startup are actually procedures containing several states. The depiction is simplified for the purpose of an overview.

POC:ready, and *PROTOCOL_ENGINE_HALT* modes the core mechanisms appropriately for entry to *POC:halt*.

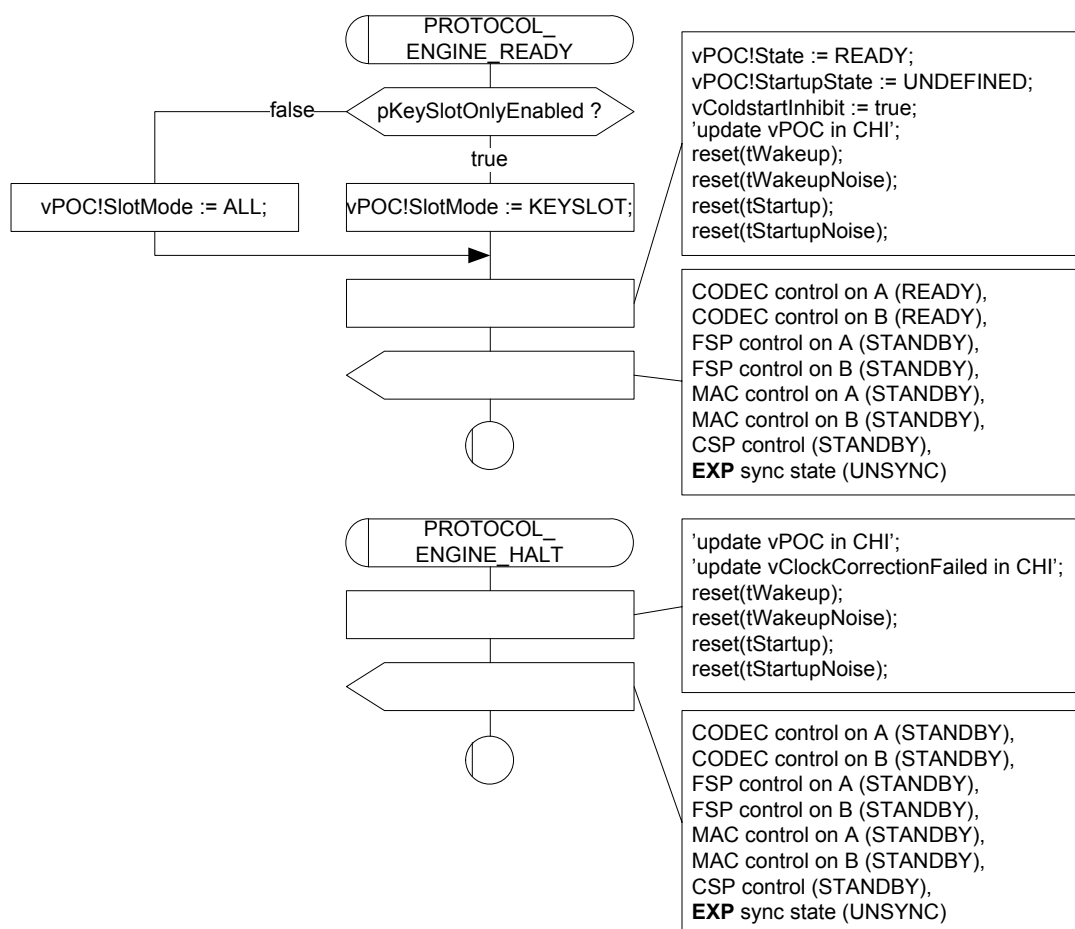


Figure 22 — Macros to mode the core mechanisms for transitions to the POC:ready and POC:halt states [POC]

The SDL processes associated with the core mechanisms are created simultaneously by the POC. While the processes shall terminate themselves, the POC is also responsible for simultaneously triggering this in all of the processes. Figure 23 depicts the macros for performing these two tasks.

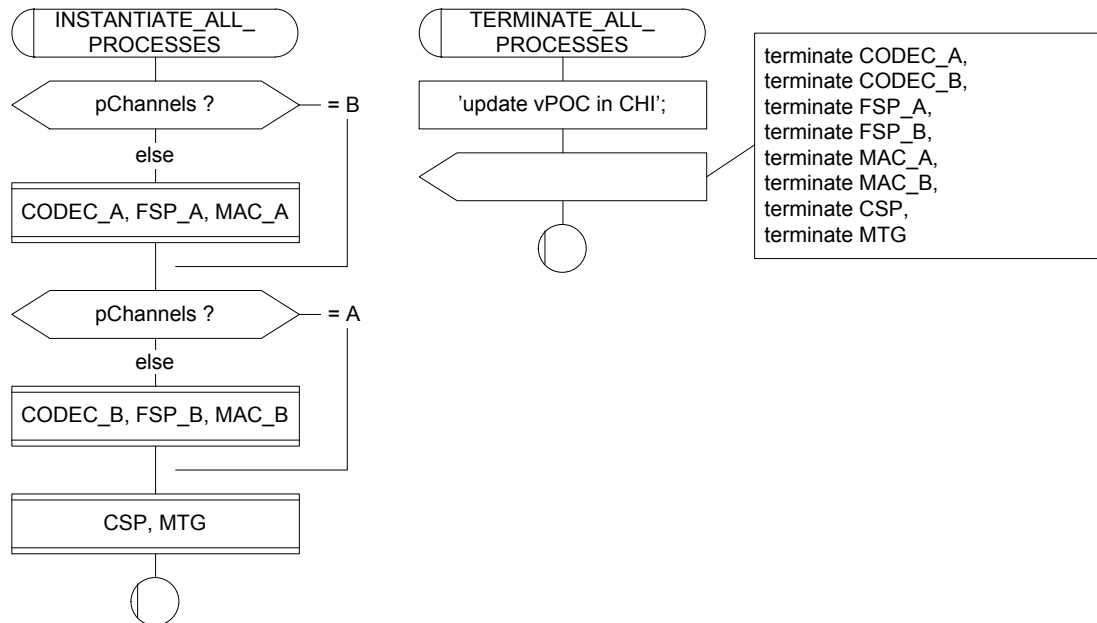


Figure 23 — Macros for creating and terminating processes [POC]

6.3.3 SDL organization

From the perspective of procedural flow, the behaviour of the POC can be loosely decomposed into four components to facilitate discussion:

- behaviours corresponding to host commands that preempt the regular behavioural flow;
- behaviour that brings the POC to the *POC:ready* state;
- behaviour leading from the *POC:ready* state to the *POC:normal active* state;
- behaviour once the *POC:normal active* state has been reached, i.e., during "normal operation".

The remainder of this subclause addresses these four components in succession, explaining the required behaviour using SDL diagrams.

6.3.4 Preempting commands

There are two commands (FREEZE and IMMEDIATE_READY) that are used to preempt the normal behavioural flow of the POC. They are depicted in Figure 24. It should be emphasized that these commands also apply to the behaviour contained in the Wakeup and Startup macros (see Figure 30) that is detailed in clause 11.

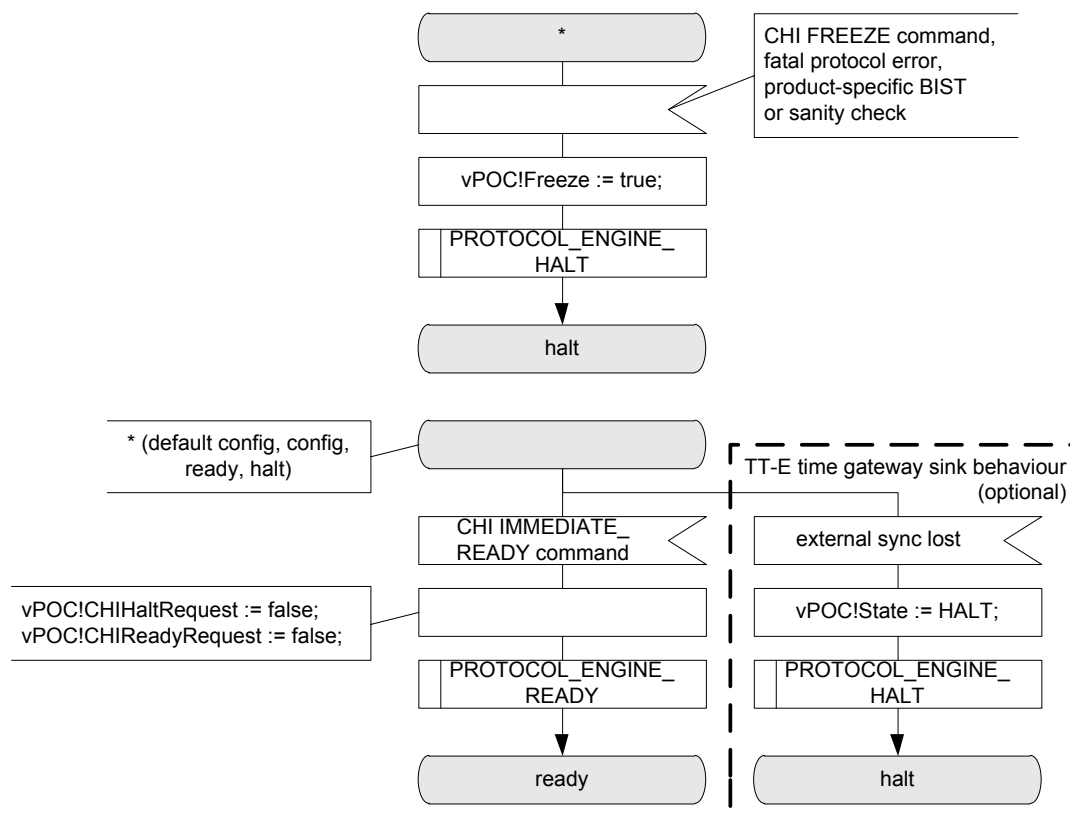


Figure 24 — POC preempting immediate commands [POC]

When a serious error occurs, the POC is notified to halt the operation of the protocol engine. For this purpose, a freeze mechanism is supported. There are three methods for triggering the freeze mechanism:

- a host FREEZE command relayed to the POC by the CHI;
- a *fatal protocol error* signalled by the FSP process;
- a product-specific error detected by a built-in self-test (BIST) or sanity check;

In all three circumstances the POC shall set *vPOC!Freeze* to true as an indicator that the event has occurred, stop the protocol engine by setting all core mechanism to the STANDBY mode, and then transition to the *POC:halt* state¹⁷⁾.

17) Values of *vPOC!State* and *vPOC!StartupState* are intentionally not altered so that the CHI can indicate to the host what state the POC was in at the time the freeze occurred.