DISCLAIMER
This specification as released by the FlexRay Consortium is intended for the purpose of information only. The use of material contained in this specification requires membership within the FlexRay Consortium or an agreement with the FlexRay Consortium. The FlexRay Consortium will not be liable for any unauthorized use of this Specification.

Following the completion of the development of the FlexRay Communications System Specifications commercial exploitation licenses will be made available to End Users by way of an End User’s License Agreement. Such licenses shall be contingent upon End Users granting reciprocal licenses to all Core Partners and non-assertions in favor of all Premium Associate Members, Associate Members and Development Members.

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 the publisher.

The word FlexRay and the FlexRay logo are registered trademarks.

Copyright © 2004-2005 FlexRay Consortium. All rights reserved.
## Revision history

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.0.2</td>
<td>2002-04-09</td>
<td>First release of the FlexRay Requirements Specification</td>
</tr>
<tr>
<td>2.1</td>
<td>2005-12-19</td>
<td>Second release of the FlexRay Requirements Specification</td>
</tr>
</tbody>
</table>
Table of content:

0 INTRODUCTORY REMARKS ........................................................................................................ 7
  0.1 Notational Conventions ......................................................................................................... 9
  0.2 References ............................................................................................................................. 9

1 THE PURPOSE OF THE PRODUCT ............................................................................................ 10
  1.1 The user problem or background of the project effort ............................................................ 10
  1.2 Goals of the project .............................................................................................................. 10

2 CLIENT, CUSTOMER AND OTHER STAKEHOLDERS .................................................................. 12
  2.1 The client of FlexRay Requirements Specification ............................................................... 12
  2.2 The customers of the FlexRay communication system ......................................................... 12
  2.3 Other stakeholders of the FlexRay communication system ................................................... 12

3 USERS OF THE PRODUCT ......................................................................................................... 13

4 MANDATED CONSTRAINTS ....................................................................................................... 13
  4.1 Solution constraints .............................................................................................................. 13

5 NAMING CONVENTIONS AND DEFINITIONS .......................................................................... 15

6 RELEVANT FACTS AND ASSUMPTIONS ................................................................................... 16

7 THE SCOPE OF THE WORK ....................................................................................................... 17
  7.1 The logical context of FlexRay .............................................................................................. 17
  7.2 A closer look at the logical structure ....................................................................................... 18
  7.3 The physical context of FlexRay ........................................................................................... 21
    7.3.1 Basic physical units of FlexRay ....................................................................................... 21
    7.3.2 More physical units ........................................................................................................... 23
  7.4 Relationships between logical and physical units .................................................................... 25
  7.5 Fault domains ....................................................................................................................... 27

8 THE SCOPE OF THE PRODUCT .................................................................................................. 29
  8.1 Use Case Diagram for FlexRay Processes ........................................................................... 29
  8.2 State model of a communication module .............................................................................. 31

9 DETAILED FUNCTIONAL REQUIREMENTS .............................................................................. 33
  9.1 Execute normal communication cycle ................................................................................. 33
    9.1.1 Receive communication elements .................................................................................... 40
      9.1.1.1 Receive data stream from logical link ........................................................................ 42
      9.1.1.2 Decode binary data stream ......................................................................................... 43
      9.1.1.2.1 Take samples and filter glitches ............................................................................ 44
      9.1.1.2.2 Strobe voted signals .............................................................................................. 45
      9.1.1.2.3 Decode strobed signals .......................................................................................... 45
12 PERFORMANCE REQUIREMENTS ........................................................................................................... 90

12.1 Speed and latency requirements ........................................................................................................... 90
12.2 Safety relevant requirements ................................................................................................................. 90
12.3 Precision requirements ............................................................................................................................ 91
12.4 Reliability and Availability requirements ............................................................................................... 91
12.5 Robustness requirements ....................................................................................................................... 92
12.6 Capacity requirements ............................................................................................................................ 92
12.7 Scalability or extensibility requirements .................................................................................................. 93

10 LOOK AND FEEL REQUIREMENTS ....................................................................................................... 90

11 USABILITY REQUIREMENTS ..................................................................................................................... 90
13 OPERATIONAL REQUIREMENTS...................................................................................... 93
  13.1 Expected physical environment ........................................................................ 93
  13.2 Productization Requirements ......................................................................... 94

14 MAINTAINABILITY AND SUPPORT REQUIREMENTS.................................................... 94
  14.1 Maintainance requirements ............................................................................. 94
  14.2 Special conditions that apply to the maintenance of the product ......................... 94
  14.3 Supportability requirements ............................................................................. 94
  14.4 Portability requirements .................................................................................... 94

15 SECURITY REQUIREMENTS .......................................................................................... 95
  15.1 Integrity requirements ....................................................................................... 95
  15.2 Immunity requirements ..................................................................................... 95

16 OPEN ISSUES .................................................................................................................. 96
  16.1 Issues that have been raised and do not yet have a conclusion ......................... 96

17 WAITING ROOM ............................................................................................................ 96

APPENDIX A ABBREVIATIONS AND DEFINITIONS.............................................................. 97
ABBREVIATIONS.................................................................................................................. 106

APPENDIX B UML NOTATION GUIDE .............................................................................. 107
  B.1 Use Case diagram .................................................................................................. 107
    B.1.4.1.1 Elements of a Use Case diagram ................................................................. 107
  B.2 Activity diagram ................................................................................................... 108
    B.2.4.1.1 Elements of an activity diagram ................................................................. 109
  B.3 Class diagram ....................................................................................................... 110
    B.3.4.1.1 Elements of a Class diagram .................................................................... 110
  B.4 State Transition diagram ...................................................................................... 112
    B.4.4.1.1 Elements of a State Transition diagram .................................................... 112
  B.5 Component diagram ............................................................................................. 114
    B.5.4.1.1 Elements of a Component diagram ........................................................... 114
0 Introductory Remarks

This document is a collection of requirements and constraints for the FlexRay communication system, a dependable automotive network. For the ease of reading informal paragraphs are interspersed between formal requirements.

The FlexRay requirements document is based on the VOLERE Template. We acknowledge that this document uses copyright material from the VOLERE requirements specification template. Copyright © 1995-2004 The Atlantic Systems Guild Limited. More detailed information: http://www.volere.co.uk

The terms ‘requirements’ and ‘constraints’ are used according to the following definitions:

- **requirement**
  something a system must do or a property a system must have. A requirement shall always be testable by just testing the system.

- **constraint**
  a pre-existing restriction that limits the solution a system can provide. This can be a design constraint (e.g. shall be embeddable in ...) or a project constraint (e.g. production price less than ...). Constraints are separated from requirements, because it is not possible to test a constraint by testing the system.

In addition the following further typographic characteristics are used in this Requirement Specification:

- `': A term, that is enclosed in inverted commas, describes a defined term. Defined terms used in requirements and definitions are marked this way. In describing and introducing texts it was decided not to mark defined terms due to better legibility.

  Example: ‘communication element’.

- `(': Terms in brackets are additions which serve to the comprehensibility of the document because of their additional information content. Usually these additions enclosed in brackets appear in connection with defined terms.

  Example: (received) ‘data stream’.

- **Underlines**: Underlines mark Use Case names and activity names which are further specified in the text.

  Example: execute normal communication cycle.

- **Shades**: If a shade lies behind a notation element in an activity diagram, then this activity can exist twice (one for each logical link).

In this Requirement Specification not every sentence shall be regarded as a formal requirement. A sentence might only serve as introduction and for better comprehension. Some text blocks are used to define terms to be used in further requirements. Definitions are tagged by a specific text style using the θ-symbol as enumeration type. Other text blocks are requirements with different bindingness. Requirements are marked by using a specific text style using the -symbol as
enumeration type. The bindingness of requirements is expressed in the respective requirement through a keyword according to the following table.
0.1 Notational Conventions

This specification covers five types of diagrams we use for modelling the requirement for the FlexRay communication system. These diagrams can also be used for modelling other complex systems and even business processes. Modelling is the step of analysing and describing the logical dependencies or designing a system before constructing it. The diagrams we use originate from the Unified Modelling Language (UML), an international standardized notation for systems specification, defined by the Object Management Group (OMG). The UML is a set of rules how to draw certain diagrams which are used in the system development process. Further information concerning the notation UML and the organisation OMG can be received on www.omg.org.

Creating these models for your project is an essential step in order to assure that the system (which is to be specified and later on developed) provides all the business functionality it needs. Creating these models is not a one time job, they have to be adjusted until it is assured that they are complete.

This process of creating and adjusting the models helps us to discover new requirements as well as eliminating contradictions. As the models provide a visual representation of textual requirements, you can easily recognize distinct dependencies at a glance.

In the FlexRay Requirements Specification we use:

- Use Case diagrams,
- Class diagrams,
- Activity diagrams,
- State Transition diagrams and
- Component diagrams.

A detailed description how to read these diagrams using examples is contained in Chapter 30 “UML Notation Guide”.

0.2 References

No references
1 The Purpose of the Product

1.1 The user problem or background of the project effort.

The product for which requirements are documented in this specification is the functionality of a FlexRay communication system.

1.2 Goals of the project

The basic responsibility of FlexRay is to provide communication support. The following goals are the highest level requirements for this communication support.

- **ID**: 285  
  **Type**: Requirement  
  δ ‘FlexRay’ shall provide deterministic communication with bounded latency and small latency jitter.

- **ID**: 286  
  **Type**: Requirement  
  δ ‘FlexRay’ shall support on-demand communication at run time. The portion of the overall bandwidth for this kind of communication shall be configurable, including the option to have zero bandwidth for on-demand communication.

- **ID**: 287  
  **Type**: Requirement  
  δ ‘FlexRay’ shall ensure that on-demand communication cannot interfere with the deterministic communication.

- **ID**: 288  
  **Type**: Requirement  
  δ ‘FlexRay’ shall provide scalable fault-tolerance to enable an economic design of  
  a. distributed non-fault-tolerant systems and,  
  b. distributed fault-tolerant systems, as well as  
  c. mixed systems (partly fault-tolerant systems and partly non-fault-tolerant systems).

- **ID**: 289  
  **Type**: Requirement  
  δ ‘FlexRay’ shall support a discrete set of bit rates.

- **ID**: 2018  
  **Type**: Requirement  
  δ ‘FlexRay’ shall support a bit rate of 10 Mbit/s.
FlexRay should not only support communication, but also assist the host in performing proper communication. This leads to the following requirement.

**ID** : 291  
**Type** : Requirement  
δ 'FlexRay' shall support 'host' operation by providing the additional services:  
- a network management service  
- a message ID filtering service  
- an interrupt service  
- a macrotick timer service  
- a temperature monitoring service  
- a voltage monitoring service

**ID** : 2008  
**Type** : Requirement  
δ 'FlexRay communication system' shall support the 'composability' of subsystems into larger systems. That means testing of subsystems on the 'FlexRay communication system' shall not enforce the necessity of additional test-runs within the whole system with respect to the previously tested behavior. (Composability)

**Composability**  
is a system design principle that deals with the inter-relationships of components. A highly composable system provides recombinant components that can be selected and assembled in various combinations to satisfy specific user requirements.

**ID** : 2009  
**Type** : Requirement  
δ 'FlexRay' shall provide predictable behavior at absence and at presence of error conditions. (Predictability)

**Global time**  
is the combination of cycle counter and cycle time. The global time of a cluster is the general common understanding of time inside the cluster. The FlexRay protocol does not have an absolute or reference global time; every node has its own local view of the global time.

**ID** : 2010  
**Type** : Requirement  
δ 'FlexRay communication system' shall provide a network-wide consistent view of time with a known accuracy to all 'nodes'. (Global time)
2 Client, Customer and other Stakeholders

2.1 The client of FlexRay Requirements Specification

The client of this Requirements Specification is the FlexRay Consortium, represented by the core partners.

2.2 The customers of the FlexRay communication system

The customers of the FlexRay communication system are:

- Automotive system designers, who will use FlexRay as a building block to make automotive control systems and to carry out other non-control related functionality in an automobile (body function integration, for example).
- Automotive OEM’s, as they will integrate the systems provided by the automotive system designer.
- Companies which develop generic software modules for FlexRay.
- Tier 1 suppliers who build ECU’s to the specifications of the automotive system designers.

ID : 301
Type : Requirement
δ ‘FlexRay’ ‘communication modules’ shall be embeddable into state of the art automotive environments.

2.3 Other stakeholders of the FlexRay communication system

Semiconductor manufacturers who make silicon used to implement the FlexRay modules (communication controllers, host microcontrollers, bus drivers, active stars, etc.).

Many other stakeholders will be involved with FlexRay. Among them are
- companies who create design tools for use with FlexRay.
- companies who create development and testing tools (hardware and software) for the FlexRay communication system.
3 Users of the Product

This section is considered not applicable since the FlexRay functionality will be used by applications, not by (human) users.

4 Mandated Constraints

4.1 Solution constraints

The following list is a collection of all the constraints mentioned throughout the requirements specification. They have been copied to this section to display them together in one place. They can be easier understood when reading them in their original context.

ID : 312
Type : Constraint
δ The FlexRay communication subsystem shall work when powered by state of the art automotive power supplies within (for the system) specified min and max voltage ranges.

ID : 320
Type : Constraint
δ No two ‘communication modules’ are permitted to transmit ‘frames’ in the ‘dynamic segment’ with the same priority on the same ‘logical link’ in the same ‘communication cycle’.

ID : 2052
Type : Constraint
δ ‘FlexRay’ shall support the reconfiguration of the transmission buffers in the dynamic segment during runtime.

ID : 321
Type : Constraint
δ ‘FlexRay’ shall support presence of a ‘NM Vector’ in the payload section of ‘frames’ that are transmitted in the static section.

ID : 2024
Type : Constraint
δ Presence of an ‘NM Vector’ shall be subsystem-wide configurable.

ID : 327
Type : Constraint
δ The correct functioning of bus guardian module shall be testable during operation.
ID : 328
Type : Constraint
δ ‘FlexRay’ shall support detection of presence of faults including those faults that are not immediately visible.

ID : 329
Type : Constraint
δ Each ‘protocol module’ in the ‘cluster’ shall be configured to the same nominal communication cycle length.

ID : 330
Type : Constraint
δ ‘FlexRay’ shall support to implement ‘FlexRay’ functionality with cost-effective available semiconductor technology.

ID : 331
Type : Constraint
δ ‘FlexRay’ shall support to implement ‘channels’ using cost-effective physical media (single optical fibers, shielded twisted pair, etc.).

ID : 333
Type : Constraint
δ ‘FlexRay’ shall support to add or remove ‘ECUs’ during operation as long as two nodes which are configured to transmit ‘sync frames’ are present.
5 Naming Conventions and Definitions

This section provides the definitions of the most important terms used in the FlexRay Requirements Specification. Therefore this chapter is prerequisite for reading and understanding the rest of the specification. A lot more definitions can be found in all the other chapters, because important terms are introduced right before their first usage. An alphabetic summary of all definitions can be found in Chapter 29 - FlexRay Terminology.

This document specifies requirements for a FlexRay communication system. As shown in Figure 1 the functionality of the FlexRay communication system is packaged in communication modules, optional repeater modules and optional bus guardian modules, transmitting and receiving communication elements via logical links.

- **FlexRay communication system (short: FlexRay)**
  consists of two or more ‘communication modules’, two or more ‘logical links’, optionally some ‘bus guardian modules’ and optionally some ‘repeater modules’.

- **node**
  consists of one host, one protocol module, one or two bus drivers and optionally one or two bus guardians.

- **logical link**
  is a logical connection between two or more ‘communication modules’ through which ‘communication elements’ are conveyed for the purpose of communication.

- **communication module**
  contains the main functionality of a ‘FlexRay communication system’ responsible for transmitting and receiving ‘communication elements’ on ‘logical links’.

- **repeater module**
  contains the optional routing and error handling functionality performed by ‘active stars’.

![Figure 1: The product: A FlexRay communication system](image)

Since a FlexRay communication system is supposed to support fault-tolerant and non-fault-tolerant system design the following requirements apply.
ID: 346
Type: Requirement
δ ‘FlexRay’ shall provide the capability to support single ‘logical link’ and/or dual ‘logical link’ and/or hybrid single/dual ‘logical link’ systems.

ID: 347
Type: Requirement
δ (Optionally) a ‘bus guardian module’, if present, shall supervise one or more ‘communication modules’.

6 Relevant Facts and Assumptions

This chapter has been considered not applicable so far. No assumptions were documented.
7 The Scope of the Work

7.1 The logical context of FlexRay

The FlexRay Requirements Specification covers the path from host interface to host interface in electronic control units participating in FlexRay network communication. The document specifies general behavioral and performance requirements at:

- The interfaces between host and FlexRay modules
- The interfaces between electronic control units and/or active stars to the channel(s)
- Certain interfaces inside of electronic control units and active stars.

In a physical context, wiring and connectors are out of scope for this document.

FlexRay needs to communicate with two adjacent systems:

- It communicates with hosts in order to accept configuration or control information, to forward input and output payload data and to inform hosts about important status data.
- It communicates with the environment ("the physical world") by transmitting and receiving data streams.

*host* is an application (outside the scope of FlexRay) using the ‘FlexRay communication system’ to communicate with other applications.

These logical interfaces of the three FlexRay modules are modelled in the following logical context diagram (Figure 2):

![Logical Context of FlexRay Diagram](image-url)

**Figure 2: Logical Context of FlexRay**
As perceived by the communication modules, data streams are received and transmitted via logical links. Figure 3 demonstrates that part of the environment and the optional repeater modules are considered to be part of the logical link.

Figure 3: Logical communication via logical links

7.2 A closer look at the logical structure

The communication module contains two (sub-)modules. The major data flows exchanged between these modules are named in Figure 4.

The major goal of Figure 4 is to introduce the names of the data flows exchanged between the FlexRay modules. This strict terminology will be used throughout the Requirements Specification.

Another major goal of Figure 4 is to introduce terminology for the functionality of FlexRay (i.e., the modules depicted above). Some terms have already been defined in chapter 6. Other definitions are included in the following paragraphs.
Figure 4: The logical structure of the two major modules of FlexRay

All terms ending in module refer to the functionality of FlexRay, not to hardware. A protocol module is a functional unit. It resides on an ECU as explained below!
The most important interfaces between the modules are defined as follows:

- **selected payload data**
  are ‘payload data’ filtered in the ‘protocol module’ according to configurable ‘filter criteria’.

- **communication element**
  the basic logical unit of communication of ‘FlexRay’. A ‘communication element’ is either a ‘frame’ or a ‘symbol’.

- **binary data stream**
  is a ‘communication element’ represented as a sequence of zeros and ones communicated between ‘bus driver module’ and ‘protocol module’.

- **data stream**
  is a communication element represented as electrical or optical signal transported via ‘logical links’.

With these definitions one can state the following high level requirements:

**ID**: 385  
**Type**: Requirement  
\[\text{Forward payload data shall exchange ‘payload data’ between the ‘protocol module’ and the ‘host’.} \]  
This requirement applies independent of whether a global system time is established or not.

Each ‘bus driver module’ should be connected to one ‘logical link’.

**ID**: 389  
**Type**: Requirement  
\[\text{The ‘bus driver module’ shall convert ‘binary data streams’ into (electrical or optical) ‘data streams’ to be communicated via its ‘logical link’ and vice versa.} \]

The ‘bus guardian module’ should inhibit transmissions of data streams on the ‘logical links’ outside allocated transmission slots of the ‘communication module(s)’ it guards. If the ‘communication module’ attempts to transmit outside allocated ‘slots’, then the ‘bus guardian module’ should inhibit transmission to ensure there is not any disruption of communication between fault-free ‘communication modules’.

**ID**: 392  
**Type**: Requirement  
\[\text{All ‘protocol modules’, ‘bus driver modules’, ‘bus guardian modules’ and ‘bus schedule monitoring modules’ shall provide their status to the ‘host’.} \]

**ID**: 2077  
**Type**: Requirement  
\[\text{The ‘bus driver’ shall report presence of any faults and/or significant degradations to the ‘host’.} \]

**ID**: 2078  
**Type**: Requirement  
\[\text{The ‘bus driver’ shall not provide fault and/or degradation notifications to the ‘host’ when there is no such condition present.} \]
ID : 2079
Type : Requirement
δ  The 'communication module' shall report presence of any faults and/or significant degradations to the 'host'.

ID : 2080
Type : Requirement
δ  The 'communication module' shall not provide fault and/or degradation notifications to the 'host' when there is no such condition present.

### 7.3 The physical context of FlexRay

This section explains the basic physical units, to which FlexRay functionality is deployed and shows the requirements belonging to these physical units.

Examples for physical topologies for FlexRay are given in Figure 8 to illustrate the flexibility in hardware for FlexRay. Furthermore this chapter shows the mapping between the physical terms and the logical terms of the FlexRay Requirements Specification.

#### 7.3.1 Basic physical units of FlexRay

FlexRay functionality physically resides on two types of “processors” who are responsible for executing FlexRay functionality (Figure 5):

- ECUs (electronic control units)
- active stars
q **ECU (electronic control unit)**
   a processor (or a group of processors) to which one or more ‘communication modules’ are deployed, i.e. the hardware executing the functionality of the ‘communication modules’. An ‘ECU’ can also contain one or more ‘host’ applications (and other non-FlexRay related applications).

   *ID*: 2013
   *Type*: Constraint
   δ The bus inputs of ‘FlexRay components’ shall be proof against short circuit conditions between a bus line and system supply voltage or ground potential.

q **active star**
   a device that allows ‘data streams’ received on one ‘branch’ to be transferred (or duplicated) to all other ‘branches’ connected to it.

q **channel component**
   A channel component is any component in a ‘FlexRay communication system’ that is not part of an ‘ECU’. Examples would include cabling, ‘active stars’, ECU-external ‘bus guardian modules’, etc.

‘ECUs’ should communicate via ‘channels’.

*ID*: 408
Type : Requirement
δ ‘FlexRay’ functionality shall be independent of the used physical medium; i.e. FlexRay shall be capable to operate on electrical and/or optical ‘channels’.

ID : 409
Type : Requirement
δ When an ‘active star’ has deactivated a ‘branch’, then noise and/or communication elements on that branch shall not influence the ‘static slots’ of the communication schedule of those ‘protocol modules’ that are in ‘working’ state, as long as two nodes which are configured to transmit ‘sync frames’ are present.
(For more information on branch deactivation see section 10.9)

ID : 2025
Type : Requirement
δ A non-faulty ‘channel component’ shall not affect the ability of any non-faulty ‘communication module’ to receive ‘communication elements’ sent by a non-faulty ‘communication module’.

ID : 411
Type : Requirement
δ A ‘channel component’ shall not be able to generate syntactically correct ‘frames’ on its own.

ID : 412
Type : Requirement
δ A ‘channel component’ shall not be able to modify the (syntactically correct or incorrect) ‘communication elements’ transmitted by a faulty ‘communication module’ in such a way that the resulting modified ‘communication element’ is syntactically correct.

7.3.2 More physical units

Figure 6 shows the relationships between the physical units of FlexRay.

- **Cluster**
  a set of ‘ECUs’, each of which is connected by one or two ‘channels’.

- **Channel**
  a ‘point-to-point connection’, ‘bus connection’ or a ‘star connection’ between two or more ‘ECUs’.

- **Branch**
  a direct physical connection
  - between two or more ‘ECUs’ or
  - between an ‘active star’ and one or more ‘ECUs’ or
  - between two ‘active stars’.
ID : 420
Type : Requirement
δ ‘FlexRay’ shall support presence of 2 up to 64 ‘nodes’ in a ‘cluster’.

Various implementations for FlexRay clusters are possible as indicated in Figure 6 above and detailed in the following requirements.

ID : 422
Type : Requirement
δ An ‘ECU’ shall be connected to one or both ‘channels’.

ID : 425
Type : Requirement
δ Presence of nodes on the link between ‘repeater modules’ is not required to be supported.
ID : 426
Type : Requirement
δ ‘FlexRay’ shall support presence of up to two active stars in a system.

ID : 427
Type : Requirement
δ An ‘active star’ may have two to many ‘branches’ attached. The ‘repeater module’ on the ‘active star’ shall actively transmit a ‘data stream’ received on one of them on all others, except when a branch failure condition is present.

ID : 428
Type : Requirement
δ ‘FlexRay’ shall support ‘channels’ implemented as ‘point-to-point connection’, ‘bus connections’ and ‘star connections’.

ID : 2253
Type : Requirement
δ ‘FlexRay’ shall support at least 20m for a ‘bus connection’ and for each ‘branch’ of an ‘active star’. FlexRay shall support at least 20 ‘nodes’ on a maximum length ‘bus connection’ or ‘active star’ ‘branch’. All of these apply to a bus speed of 10 Mbit/s and lower.

q **point-to-point connection**
a connection between two end points (‘ECUs’ or ‘active stars’) that is capable of transmitting ‘data streams’. It can be electrical or optical.

q **tie line**
a direct coupling of an ‘ECU’ to an ‘active star’, that is capable of transmitting ‘binary data streams’.

q **bus connection**
a broadcast connection between two or more end points (‘ECUs’ or ‘active stars’) that is capable of transmitting ‘data streams’. ‘Bus connections’ are always electrical.

q **star connection**
is a ‘channel’ which consists of one or n ‘active stars’ and two up to n+1 ‘branches’.

ID : 433
Type : Requirement
δ ‘FlexRay’ shall allow topologies containing mixed ‘channels’, such as combinations of ‘point-to-point connection’, ‘bus connections’ and ‘star connections’.

### 7.4 Relationships between logical and physical units

Figure 7 summarizes the key relationships between logical terminology and hardware terminology.
One ‘channel’ shall communicate ‘communication elements’ of one ‘logical link’.

Figure 8 is an example to demonstrate the distinction between hardware and logical terminology as defined above.
7.5 Fault domains

A goal of FlexRay communication system is to provide adequate error management mechanisms to deal with faults arising. It furthermore offers diagnosis information to the host. For this reason fault domains are defined.

- fault domains are domains/regions in which failures could happen independently (i.e. without influencing other fault domains).

- tolerable failure
  In the context of this document a tolerable failure is a failure where communication between all nodes of a cluster is maintained, even when the particular failure is present. Note, this does not imply that communication were provided in case more than one tolerable failure is present.
  List of tolerable failures:
  - is any failure of a single ‘communication module’ (resulting in transmission of any time and/or value domain fault)
  - any failure of a single ‘logical link’ (resulting in:
    - valid ‘frame’ -> invalid ‘frame’ (for all receivers or for some)
    - valid ‘frame’ -> no ‘frame’ (for all receivers or for some)
    - generation of invalid ‘frame’ (for all receivers or for some)
    - (but not generate valid ‘frames’)
    - (but not change valid ‘frames’ into other valid ‘frames’)

Figure 8: Hardware and Logical Terminology
FlexRay Requirements Specification

Chapter 7: The Scope of the Work

- (but not change invalid ‘frames’ into valid ‘frames’)
- any failure of a single ‘repeater module’

FlexRay shall provide the following fault domains:

<table>
<thead>
<tr>
<th>Functional fault domains</th>
<th>Hardware fault domains</th>
</tr>
</thead>
<tbody>
<tr>
<td>communication module</td>
<td>active star</td>
</tr>
<tr>
<td>(optional) bus guardian module</td>
<td></td>
</tr>
<tr>
<td>logical link</td>
<td></td>
</tr>
<tr>
<td>(optional) repeater module</td>
<td></td>
</tr>
</tbody>
</table>

- **single domain fault**
  - a failure occurring in only one ‘fault domain’.
  
  List of single domain faults:
  - failures of a single ‘host’
  - failures of a single ‘communication module’
  - failures of a single ‘logical link’
  - failures of a single ‘repeater module’ in a two logical link system
  - failures of a single ‘bus guardian module’.

**ID** : 466

*Type* : Requirement

δ (Optionally) ‘FlexRay’ shall support continued communication when a ‘single domain fault’ occurs at run time.

**ID** : 468

*Type* : Requirement

δ ‘FlexRay’ shall ensure, that no single faulty component (‘communication module’ or ‘logical link’) shall cause the Use Case clock synchronization among fault-free ‘communication modules’ that are connected to both ‘logical links’ to fail as long as the ‘FlexRay communication system’ is in ‘normal active’ state.

**ID** : 469

*Type* : Requirement

δ ‘FlexRay’ shall ensure, that a faulty ‘communication module’ does not prevent the Use Case execution of the normal communication cycle from operation between non faulty ‘communication modules’. (e.g. the ‘FlexRay communication system’ shall prevent a faulty ‘communication module’ from being able to affect the communications in any ‘slot’/‘logical link’ combination not assigned to it in the ‘communication schedule’.)

**ID** : 2256

*Type* : Requirement

δ A faulty ‘ECU’ shall not be able to affect (disrupt, etc.) the transmission or reception of a ‘slot’ in the ‘static segment’ that is not assigned to the faulty ‘ECU’ on more than one ‘channel’ in any given measurement period of the ‘clock synchronization’ algorithm.

**ID** : 470

*Type* : Requirement

δ As long as ‘FlexRay’ is in ‘normal active’ state, ‘FlexRay’ shall execute the normal communication cycle (transmission and reception of ‘frames’) as well as maintain clock synchronization.
8 The Scope of the Product

8.1 Use Case Diagram for FlexRay Processes

This chapter summarizes key use cases of the FlexRay communication system. The required functionality of each Use Case is detailed in chapter 10. For reading instruction for the Use Case diagram see chapter 30.1 - Use Case diagram.

![Use Case Diagram](image.png)

Figure 9: Use Cases of the FlexRay communication system

The ten main processes in the Use Case diagram give rise to the following high level functional requirements.

**ID**: 477

**Type**: Requirement

δ ‘FlexRay’ shall provide a mechanism to detect presence of wakeup patterns on a 'logical link'.

(Use Case: check for wakeup pattern)
ID : 478
Type : Requirement
 δ ‘FlexRay’ shall provide means for the ‘host’ to configure the communication module.
   (Use Case: configure communication module)

ID : 479
Type : Requirement
 δ ‘FlexRay’ shall provide mechanisms that enable a ‘host’ to initiate the startup of the ‘FlexRay
communication system’.
   (Use Case: start communication module)

ID : 480
Type : Requirement
 δ ‘FlexRay’ shall meet defined temporal characteristics at communication between fault-free
‘nodes’.
   (Use Case: execute normal communication cycle)

ID : 481
Type : Requirement
 δ The ‘FlexRay communication system’ shall support clock synchronization among all it’s fault free
‘communication modules’ to a given precision.
   (Use Case: clock synchronization)

ID : 482
Type : Requirement
 δ ‘FlexRay’ shall provide mechanisms that enable a ‘host’ to initiate a coordinated termination of
communication sessions.
   (Use Case: shutdown communication module)

ID : 483
Type : Requirement
 δ (Optionally) ‘FlexRay’ shall provide means for the ‘host’ to configure the bus guardian module.
   (Use Case: configure bus guardian module)

ID : 484
Type : Requirement
 δ (Optionally) If a ‘bus guardian’ is used, then ‘FlexRay’ shall support means to detect whether the
‘bus guardian’ is capable to suppress transmission attempts outside of configured ‘static slots’.
   (Use Case: schedule supervision)

ID : 485
Type : Requirement
 δ (Optionally) ‘FlexRay’ shall provide means to limit access to the ‘logical links’.
   (Use Case: bus guarding)

ID : 486
Type : Requirement
 δ (Optionally) ‘FlexRay’ shall provide means to deactivate ‘branches’ in case of errors on ‘logical
links’.
   (Use Case: repeating function & branch error handling)
8.2 State model of a communication module

The processes of FlexRay depend very much on the state that the protocol module is in. Therefore, the overall State transition diagram is described first, before detailing the functionality in chapter 10. For reading instruction for the State transition diagram see chapter 30.4 - State Transition diagram. States are then used as preconditions and postconditions for various FlexRay processes.

A protocol module is either in the state ‘not operational’ or ‘operational’.

As soon as the bus driver module recognizes a wakeup pattern, it informs the host. At any time the host can request to configure the communication module.

After configuration, the protocol module changes to the state ‘operational’ into the substate ‘ready’. The host can request to start communication module and the protocol module changes to the ‘starting’ state. Before being able to work in the ‘working’ state, the communication module must successfully complete the start of the communication module.
In the state ‘working’ the protocol module executes the normal communication cycle during which it also tries to keep up the clock synchronization. In case of undervoltage detection the state of the protocol module changes back to ‘not operational’.

If a protocol module detects a fatal error or the host request shutdown or freeze it looses its transmission and receiving capabilities and changes its state to ‘halted’. In case the host then wants to reconfigure the communication module, the state of the protocol module is set back to ‘not operational’.

In every state, the power off event changes the state of the protocol module to the state ‘power off.’
9 Detailed Functional Requirements

The following sections describe the detailed behavior of the FlexRay processes (shown in the Use Case diagram in Figure 9). Every Use Case is further specified with its input(s), its functionality and its output(s).

The specification of the functionality of the Use Case always consists of an informal introductory description and a set of requirements. Depending on the complexity of the behavior to be specified the requirements are written in prose or the functionality is further decomposed using an Activity diagram.

If an Activity diagram is used for the further decomposition of the functionality of a Use Case, each activity is then specified in more detail in a subchapter. Each activity is again described by naming the input(s), specifying the functionality (with an informal introductory description and a set of requirements) and by naming the output(s). If appropriate exceptions are noted in addition.

For even more complex activities a further decomposition in form of an Activity diagram is used.

If new terms are needed they are introduced in the corresponding sections. With these terms the processes are specified using the UML diagrams and prose. (More information on the diagramming style see Chapter 30).

9.1 Execute normal communication cycle

This section describes the detailed functionality, behavior and data structures of FlexRay in its main operating mode (state ‘working’ in Figure 10), i.e. the Use Case execute normal communication cycle.

This Use Case is repeatedly executed as long as the communication module is in the state ‘working’.

The structure of the communication cycle

FlexRay is based on a recurring communication cycle. Figure 11 shows the structure of the communication cycle and its constituents.

- communication cycle
  - the key periodically repeated communication structure of ‘FlexRay’. It consists of a ‘static segment’, an optional ‘dynamic segment’, an optional ‘symbol window’ and a ‘network idle time’.
Within the first two segments of the communication cycle FlexRay offers the choice of two media access schemes for communication. These are a static time division multiple access (TDMA) scheme and a dynamic mini-slotting based scheme which are explained in the following sections.

The terms used to describe the normal communication cycle and their relationships are shown in the Class diagram in Figure 12 and defined as follows:
q **communication cycle segment**

is a part of the ‘communication cycle’. There are four specific kinds of communication cycle segments: ‘static segments’, ‘dynamic segments’, ‘symbol windows’ and ‘network idle times’.

q **static segment**

a part of the ‘communication cycle’ in which ‘static frames’ are transmitted or received.

q **slot**

an interval of time during which access to the ‘logical link’ is granted exclusively to a specific ‘protocol module’ for the transmission of a ‘frame’ with a ‘frame ID’ corresponding to the ‘slot counter’. ‘FlexRay’ distinguishes between ‘static slots’ (in the ‘static segment’) and ‘dynamic slots’ (in the ‘dynamic segment’) of the ‘communication cycle’.

---

**Figure 12: The structure of the communication cycle**

- **communication cycle segment**
- **static segment**
- **dynamic segment**
- **symbol window**
- **network idle time**
- **slot**
- **minislot**
- **frame**
- **symbol**

The diagram illustrates the structure of the communication cycle with relationships between different segments and elements.
q slot counter
a counter for the current ‘slot’.
Note: The slot counter must be unique per channel per segment. The value of the slot counter can be identical for communication elements communicated in slots on different channels or in different segments.

q slot ID
identifier of a specific ‘slot’ in which the ‘communication module’ may communicate.

q static slot
is a ‘slot’ of the ‘static segment’ of the ‘communication cycle’ used to transmit or receive a ‘frame’ of fixed length.

q dynamic segment
a part of the ‘communication cycle’ in which ‘dynamic frames’ are transmitted or received.

q dynamic slot
is a ‘slot’ of the ‘dynamic segment’ of the ‘communication cycle’ used to transmit or receive a ‘dynamic frame’. It is divided in a number of ‘minislots’.

q minislot
a part of a ‘dynamic slot’. It has a constant nominal length.

q symbol window
a part of the ‘communication cycle’ used to receive or transmit one selected ‘symbol’.

q network idle time (short: NIT)
a communication-free period at the end of the ‘communication cycle’ during which the ‘protocol module’ calculates and applies clock correction terms and performs implementation specific cluster cycle related tasks.

q communication element
the basic logical unit of communication of ‘FlexRay’. A communication element is either a ‘frame’ or a ‘symbol’. On transmitting it is converted into a sequence of analog signals for the ‘logical link’. On receiving it is abstracted from sequences of analog signals arriving on the ‘logical link’.

q frame (types of)
the basic information structure transmitted or received in one ‘slot’ between ‘communication modules’. ‘FlexRay’ distinguishes two types of frames: ‘static frames’ and ‘dynamic frames’. For the structure of frames cf. the definition of ‘frames (structure of)’.

q static frame
a ‘frame’ transmitted or received in a ‘static slot’ of the ‘static segment’ of the ‘communication cycle’.

q dynamic frame
a ‘frame’ transmitted or received in a ‘dynamic slot’ of the ‘dynamic segment’ of the ‘communication cycle’. ‘Dynamic frames’ can have variable lengths.
q **symbol**
  is a ‘communication element’ of ‘FlexRay’ used to transport control information between ‘protocol modules’. ‘FlexRay’ distinguishes between three ‘symbols’: CAS (Collision Avoidance Symbol), MTS (Media Access Test Symbol) and WUS (Wakeup Symbol).

q **action point**
  designated boundaries at which a ‘communication module’ performs a specific action in alignment with its local time base, e.g. starting the transmission of a ‘frame’.

q **static slot action point**
  is the ‘action point’ of a ‘static slot’.

q **minislot slot action point**
  is the ‘action point’ of a ‘minislot’.

q **symbol window action point**
  is the ‘action point’ of a ‘symbol window’.

q **static segment**
  Within the static segment a static time division multiple access scheme is used to arbitrate the communication.

  \[ID\] : 586
  \[Type\] : Requirement
  \[\delta\] The ‘communication cycle’ shall always contain a ‘static segment’.

  \[ID\] : 588
  \[Type\] : Requirement
  \[\delta\] ‘FlexRay’ shall allow to configure the length of ‘static slots’.

  \[ID\] : 589
  \[Type\] : Constraint
  \[\delta\] ‘FlexRay’ shall support ‘static slots’ where each ‘static slot’ has the same nominal length.

q **dynamic segment**
  Within the dynamic segment a dynamic mini-slotting based scheme is used to arbitrate communication. This part contains the detailed requirements for the dynamic segment and is derived from ID 286.

  \[ID\] : 592
  \[Type\] : Requirement
  \[\delta\] The ‘communication cycle’ may contain a ‘dynamic segment’ for on-demand data communication within the ‘communication cycle’ (i.e. FlexRay shall provide a mechanism to configure whether a dynamic segment is present or not).

  \[ID\] : 2257
  \[Type\] : Requirement
  \[\delta\] The ‘FlexRay system’ shall be configurable to allow any or all ‘ECU’s’ to transmit in the ‘dynamic segment’ of the ‘communication cycle’.
ID : 593
Type : Requirement
\(\delta\) In the ’dynamic segment’, ’FlexRay’ shall support a message transmission scheme that is based on message priority, i.e. messages shall be transmitted in priority sequence with the highest priority message first.

ID : 2026
Type : Requirement
\(\delta\) The length of the ’dynamic segment’ shall be configurable and shall be an integral multiple of one ’minislot’.

ID : 595
Type : Constraint
\(\delta\) The ’dynamic segment’ shall be divided into a configurable number of ’minislots’ of equal nominal length.

ID : 2051
Type : Constraint
\(\delta\) All ’nodes’ in a ’cluster’ shall configure the ’dynamic segment’ length to the same value.

ID : 596
Type : Requirement
\(\delta\) The length of the ’dynamic slots’ in the ’dynamic segment’ shall be variable during runtime to accommodate ’frames’ of varying length.

ID : 2054
Type : Requirement
\(\delta\) When a transmitting protocol module reaches the end of the dynamic segment while frame transmission is still ongoing, then it shall automatically abort the ongoing transmission and flag an error to its host.

\(\textbf{symbol window}\)
The symbol window is the communication cycle segment in which a symbol can be received or transmitted. This part contains the detailed requirements for the symbol window.

ID : 600
Type : Requirement
\(\delta\) The ’communication cycle’ may contain a ’symbol window’ to provide a defined segment for exchanging control information for synchronizing logical link access.

ID : 601
Type : Requirement
\(\delta\) ’FlexRay’ shall support a configurable length for the ’symbol window’.

\(\textbf{network idle time}\)
The network idle time is a communication-free period that concludes each communication cycle. This part contains the detailed requirements for the network idle time.
ID: 605
Type: Requirement
δ ‘FlexRay’ shall support synchronization even when local clock speeds differ to a defined extend.

ID: 2060
Type: Requirement
δ ‘FlexRay’ shall support reception in the ‘static segment’ of all transmitted communication elements, regardless when they are scheduled to start in the ‘communication cycle’.

This could be achieved by separating the communication cycles by a phase where no transmission is allowed, e.g. NIT.
Note: The length of this phase may vary from cycle to cycle, in order to reach the configured cycle length.

The constraints on the configuration of the communication cycle are defined in chapter 10.2.

A closer look at the Use Case execute normal communication cycle reveals the concurrent activities shown in Figure 13.

Figure 13: Execute normal communication cycle

FlexRay can receive communication elements or transmit communication elements via the logical links in one communication cycle. While receiving and transmitting communication elements FlexRay also performs the following functions:

• perform additional services - from the perspective of the communications protocol non-communication related services
• clock synchronization - is responsible for synchronizing the clock of each communication module by observing the timing of transmitted sync frames from other communication modules and for generating macroticks
• (Optionally) schedule supervision - is responsible for detecting timing faults

While in one of the states ‘not operational’, ‘starting’ or ‘working’, the ‘protocol module’ should support reception of communication elements.

While in one of the states ‘starting’ or ‘normal active’, the ‘protocol module’ should support transmission of communication elements.
While in one of the state ‘working’, the ‘protocol module’ should support means to perform additional services.

The high level requirements for the other two activities (clock synchronization and schedule supervision) have already been covered in section 9.1.

9.1.1 Receive communication elements

This part of the FlexRay Requirements Specification deals with the reception of communication elements. It describes the mechanisms starting with the reception of communication elements from logical links up to the delivery of the communication elements to the host.

Input
data stream

Functionality

Figure 14 shows the four sequential activities executed upon receiving data streams on the logical links.
Since Figure 14 shows the reception only, the obvious terms “received” and “output” have been removed from the names of the data flows (e.g. data stream instead of (received) data stream and payload data instead of (output) payload data).

Note: The shaded two activities (receive data stream from logical link and decode binary data stream) are executed concurrently per logical link. The shade behind the activity symbol hints at this fact.
While in one of the state 'working', each 'bus driver module' should be able to receive 'data streams' from its 'logical link'.

_ID_ : 631
_Type_ : Requirement
δ Receive data stream from logical link shall be executed independently for each 'logical link'.

_ID_ : 633
_Type_ : Requirement
δ Decode binary data stream shall be executed independently for each 'logical link'.

_ID_ : 2086
_Type_ : Requirement
δ At frame reception, the 'communication module' shall be capable to detect presence of a burst error of arbitrary length and arbitrary position in a frame.

Output
selected payload data

### 9.1.1.1 Receive data stream from logical link

**Input**
data stream

**Functionality**
Receive data stream from logical link is responsible for the communication between the logical link and the node. The data streams (analog signals) from the logical links have to be converted into binary data streams (digital signals) and afterwards transmitted to decode binary data stream.

While in the states 'not operational', 'starting' and 'working', receive data stream from logical link should be able to receive the 'data stream' from the 'logical link'.

_ID_ : 647
_Type_ : Requirement
Receive data stream from logical link shall detect and report occurring failures on the logical links to the host.

_ID_ : 648
_Type_ : Requirement
δ Receive data stream from logical link shall detect coding failures of the 'communication element' received on the 'logical link'.

_ID_ : 649
_Type_ : Requirement
δ Receive data stream from logical link shall provide the ‘bus driver status data’ to the ‘host’, (e.g. a single indication “communication not possible” is sufficient to fulfill this FlexRay requirement.)

**Output**
binary data stream
bus driver status data
9.1.1.2 Decode binary data stream

**Input**
binary data stream

**Functionality**
Decode binary data stream is responsible for converting binary data streams to communication elements (frames and symbols). Since the binary data stream can contain glitches, take samples and filter glitches has to analyze the binary data stream for such glitches and do a majority voting to get one voted signal, align them with the local bit clock (strobe voted signals) before decode strobed signal can create communication elements. The Activity diagram in Figure 15 shows the detailed activities of decode binary data stream and their logical operating sequence. Each activity is detailed in subsections within this section.

- **glitch**
an event that changes the current condition of the ‘logical link’ such that its detected logic state is temporarily forced to a different value compared than what is being currently transmitted on the ‘logical link’ by the transmitting ‘node’ (e.g. spikes).

**ID**: 660  
**Type**: Requirement
- Decode binary data stream shall support successful bit decoding (message reception) even at presence of timing disturbances and glitches typically occurring inside vehicles (e.g. signal delay, edge jitter, baud-rate jitter).

**ID**: 2049  
**Type**: Requirement
- At frame reception, decode binary data stream shall be capable to detect burst errors of arbitrary length and arbitrary position in the frame.
9.1.1.2.1 Take samples and filter glitches

Input
binary data stream

Functionality
The purpose of take samples and filter glitches is to filter ‘glitches’ that may be contained in the binary data stream and get a voted signal with definite values of highs and lows.
q **voted signal**  
definite high or low signal used as input for **strobe voted signals**.

**ID : 672**  
**Type : Requirement**

δ The 'protocol module' shall perform a glitch filter mechanism prior to further processing of a received data stream. The filter mechanism shall work independently for each supported logical link.

**ID : 673**  
**Type : Requirement**

δ **Take samples and filter glitches** shall have a digital low pass behavior.

**Output**  
stream of voted signals

### 9.1.1.2.2 Strobe voted signals

**Input**  
stream of voted signals

**Functionality**  
Align the stream of voted signals to create a bit clock aligned stream of strobed signals.

q **local bit clock**  
defines the rate with which the voted signals are strobed.

q **strobed signals**  
definite high and low signals aligned to the local bit clock.

The 'protocol module' should determine the signal value of the received data stream at defined points in time (strobe mechanism).

**Output**  
stream of strobed signals

### 9.1.1.2.3 Decode strobed signals

Here the structure of the coded frames and symbols is analyzed and unpacked to yield the (decoded) frames and symbols.

**Input**  
stream of strobed signals (i.e. coded frames and coded symbols)

Figure 16 and the following definitions define the structure of frames and symbols including their coded forms.
Figure 16: Structure of coded and decoded communication elements

- **coded frame**
  - is a ‘frame’ augmented by a sending ‘communication module’ with a number of control elements.
  - It allows the receiving ‘communication module’ to recognize various start and end events. Control elements are: ‘TSS’ (transmission start sequence), ‘FSS’ (frame start sequence), ‘BSS’ (byte start sequence), ‘FES’ (frame end sequence), and ‘DTS’ (dynamic trailing sequence).

- **coded symbol**
  - is a ‘symbol’ augmented by a sending ‘communication module’ with a ‘TSS’ (transmission start sequence).
  - Note: WUS is sent without TSS.

- **TSS (transmission start sequence)**
  - is used to initiate a proper connection setup through the ‘FlexRay communication system’. The purpose of the TSS is to “open the gates” of an ‘active star’, i.e., to cause the ‘active star’ to properly set up ‘branches’. During this set up, an ‘active star’ truncates a number of bits at the beginning of a ‘communication element’. The TSS also prevents the content of the ‘frame’ or ‘symbol’ from being truncated by the receiving ‘communication module’.

**ID**: 696
**Type**: Requirement

δ When starting the transmission of a ‘communication element’, the ‘protocol module’ shall first send a ‘TSS’ such that receiving ‘nodes’ can prepare for the arrival of a new ‘communication element’.

**ID**: 697
**Type**: Requirement

δ The structure and length of the ‘TSS’ shall be such that it enables receiving ‘protocol modules’ to detect whether a ‘frame’ or a ‘symbol’ is present.
q **FSS (frame start sequence)**
   is used to truncate TSS and compensate a possible quantization error.

q **BSS (byte start sequence)**
   is used to provide a bit stream timing information to the receiving ‘communication module’.

q **FES (frame end sequence)**
   is used to mark the end of the last byte sequence of a ‘frame’.

q **DTS (dynamic trailing sequence)**
   is used to indicate the exact end of the ‘dynamic frame’, to prevent premature channel idle
detection. This is required for the stability of certain types of physical layers.

**Functionality**
wait for start of communication element by detecting TSS
depending on the length of TSS, decide if frame or symbol has been received:
   if a frame has been received:
      detect FSS/BSS and report potential frame start to start communication module
      take a timestamp and forward it as primary time reference point to measure
deviation values
      detect header segment and report header received to wakeup channel
      detect FES and optionally DTS to determine the end of the frame
      forward frame to check communication elements for errors
   if a symbol has been received:
      forward symbol to check communication elements for errors

**ID** : 713
**Type** : Requirement
δ **Decode strobed signals** shall provide a mechanism to extract ‘frames’ and ‘symbols’ from the
stream of ‘voted signals’.

**Decode strobed signals** should forward decoded ‘frames’ to check communication elements for
errors.

**ID** : 722
**Type** : Requirement
**Decode strobed signals** shall detect and report errors occurring during the decoding process.

**ID** : 2090
**Type** : Requirement
δ The ‘communication module’ shall not terminate transmission and reception in the static segment
when any kind of decoding errors occur in the dynamic segment.

**ID** : 723
**Type** : Requirement
δ **Decode strobed signals** shall detect bit errors within the ongoing ‘communication element’. Upon
detection of a bit error **decode strobed signals** shall mark the ‘communication element’
immediately as invalid.
ID: 724  
Type: Requirement  

δ  If a decoding error is detected, **decode strobed signal** shall not forward the erroneous 'communication element'.

**Decode strobed signals** should transmit recognized decoding errors as 'status data' to **syntactic and semantic checks**.

**Output**  
communication element (frame or symbol)  
status data (potential frame start, header received, primary time reference point, decoding error)

### 9.1.1.3 Check communication elements for errors

**Input**  
communication elements (frames and symbols)  
status data (decoding error)

**Functionality**  
**Check communication elements for errors** checks the structure of the received communication elements (especially the frame structure) against significant events of the communication cycle. In case of syntactic or semantic errors it reports those to the host. Otherwise the correct payload data are offered to inform the host.

Before this functionality is refined in the Activity diagram in Figure 19 the structure of frames that are extracted are explained.

q  **frame (structure of)**

consists of a 'header segment', a 'payload segment' and a 'trailer segment'. For the types of frames cf. the definition of 'frames (type of)'.

Figure 17 shows the constituents of the FlexRay frame format and their dependencies in form of a UML Class diagram. Figure 18 shows the sequence and adds the numbers of bits and bytes for the constituents.
**Figure 17: Structural view of FlexRay frames**

- **header segment**: is the part of a ‘frame’ in which the control information for the ‘frame’ is contained, e.g. the ‘frame ID’.

- **payload segment**: is the part of a ‘frame’ in which ‘payload data’ is transmitted or received.

- **static payload segment**: is a ‘payload segment’ of a ‘frame’ in which a fixed length ‘payload data’ is transmitted or received in the ‘static segment’ of the ‘communication cycle’.

- **dynamic payload segment**: is a ‘payload segment’ of a ‘frame’ in which ‘payload data’ of variable length is transmitted or received in the ‘dynamic segment’ of the ‘communication cycle’.

- **trailer segment**: is the last part of a ‘frame’ containing a single field, the 24-bit ‘trailer CRC’ for the ‘frame’.

- **trailer CRC**: a cyclic redundancy check code computed over the ‘header segment’ and the ‘payload segment’ of the ‘frame’.

- **payload preamble indicator**: indicates whether (indicator set to 1) or not (indicator set to 0) a ‘NM Vector’ or ‘message ID’ is contained within the ‘payload segment’ of the transmitted ‘frame’.

- **sync frame indicator**: indicates whether or not the ‘frame’ is a ‘sync frame’, i.e. a ‘frame’ that is utilized for system wide synchronization of communication.

- **startup frame indicator**: indicates whether or not a frame is a ‘startup frame’. ‘Startup frames’ serve a special role in the Use Case start communication module.

- **frame ID**: defines the slot in which the frame is intended to be transmitted.

  - **ID**: 750
  - **Type**: Requirement
  - δ Each ‘frame’ shall contain a ‘frame ID’.

- **payload length**: is an indicator for the size of the ‘payload segment’.
**cycle count**
the number of the ‘communication cycle’ in which this ‘frame’ shall be transmitted.

The formal structure of a frame can also be shown as in Figure 18. The length of the fields are examples for a potential implementation of the structure.

![Figure 18: FlexRay frame format: another view](image)

**ID**: 2083  
**Type**: Requirement  
δ The message header CRC shall be adequate to reveal errors in headers with a probability of $10^{-10}$ for all credible errors, e.g. bit drop/flip, header truncation by BG and burst noise.

**ID**: 2084  
**Type**: Requirement  
δ The message body CRC shall be adequate to reveal errors in the frame body with a probability of $10^{-10}$ for all credible frame errors, e.g. bit drop/flip, message tail truncation by BG and burst noise.

**ID**: 2085  
**Type**: Requirement  
δ The message body CRC shall provide a Hamming Distance of at least 6 for messages with a payload length of up to 64 bytes.

With these definitions the following activities can be explained.
Check communication elements for errors should check the syntactic and semantic correctness of ‘communication elements’.

q **syntactic correctness**
   a ‘communication element’ is syntactically correct, if it’s constituents are time aligned with the segments and ‘slots’ of the ‘communication cycle’.

q **semantic correctness**
   a ‘communication element’ is semantically correct, if the elements of the ‘frame structure’ are used in accordance with the Use Cases of ‘FlexRay’.

9.1.1.3.1 **Provide significant events**

**Input**
cycle start signal
cycle counter

**Functionality**
Provide significant events is responsible for detecting significant timing events and informing perform syntactic and semantic checks about these events.
significant timing event
is any event that indicates the start of a new ‘communication cycle’, any ‘communication cycle segment’ or any ‘slot’ in the segment.

for each communication cycle:
provide cycle counter,
provide start of static segment,
provide static slot boundary,
provide start of dynamic segment,
provide dynamic slot boundary,
provide start of symbol window,
provide start of the NIT.

Provide significant events should determine ‘significant timing events’.

When provide significant events has determined a ‘significant timing event’ it should inform perform syntactic and semantic checks prior to consideration of the event or notification of the host.

Output
significant timing event

9.1.1.3.2 Perform syntactic and semantic checks

Input
significant timing event
communication elements (frames and symbols)
status data (decoding error)

Functionality
In order to detect and indicate errors, perform syntactic and semantic checks perform specific syntactic checks according to the following requirements:

ID : 788
Type : Requirement
Perform syntactic and semantic checks shall check if the ‘communication module’ starts transmitting while the ‘logical link’ is not idle.

ID : 790
Type : Requirement
Perform syntactic and semantic checks shall check whether unexpected 'communication elements' were present in each of the message slots.

ID : 791
Type : Requirement
Perform syntactic and semantic checks shall check whether a 'symbol' is received in the 'static segment', in the 'dynamic segment', or in the 'network idle time'.

In order to detect and indicate semantic errors, perform syntactic and semantic checks perform specific semantic checks according to the following requirements:

ID : 794
Type : Requirement
Perform syntactic and semantic checks shall check whether the 'payload length' being indicated in the header of a 'frame' that was received in the 'static segment' does match the configured 'payload length' for ‘frames’ in the 'static segment'. 
ID : 795
Type : Requirement
δ Perform syntactic and semantic checks shall check in the ‘static segment’ whether the ‘startup frame indicator’ is set while the ‘sync frame indicator’ is not set.

ID : 796
Type : Requirement
δ Perform syntactic and semantic checks shall check in the ‘static segment’ whether the ‘frame ID’ contained in the header of the ‘communication element’ does match the current value of the ‘slot counter’.

ID : 797
Type : Requirement
δ Perform syntactic and semantic checks shall check in the ‘static segment’ or in the ‘dynamic segment’ if the ‘cycle count’ contained in the header of the ‘communication element’ does match the current value of the ‘cycle counter’.

ID : 798
Type : Requirement
δ Perform syntactic and semantic checks shall check in the ‘dynamic segment’ whether the ‘sync frame indicator’ contained in the header of the ‘communication element’ is set.

ID : 799
Type : Requirement
δ Perform syntactic and semantic checks shall check in the ‘dynamic segment’ whether the ‘startup frame indicator’ contained in the header of the ‘communication element’ is set.

If none of the above errors occurred the frame or symbol is valid and the payload data can be transmitted to inform the host.

When the ‘protocol module’ has verified that there is no syntactic or semantic error present with a significant timing event, then it should extract the ‘payload data’ out of the ‘communication element’ and forward it to the ‘host’.

Perform syntactic and semantic checks should report the ‘slot status’ as ‘status data’ to the host.

Slot status
is a structure consisting of the following information:
- ‘channel identifier’ of the receiving ‘logical link’,
- ‘slot ID’ of the ‘slot’ in which the ‘communication element’ was received;
- ‘cycle counter’ of the ‘communication cycle’,
- validity of ‘communication element’,
- type of ‘communication element’ (‘frame’ or ‘symbol’),
- type of segment in which the ‘communication element’ was received (unsynchronized startup phase, ‘static segment’, ‘dynamic segment’, ‘symbol window’, ‘NIT’).

ID : 804
Type : Requirement
If any error is present in slot status and/or if a decoding error has been reported by decode binary data stream, perform syntactic and semantic checks shall report an error data to inform the host.
Output
payload data
status data (syntactic or semantic error data, decoding error, slot status)

9.1.1.4 Inform the host

Input
payload data, status data

Functionality
Inform the host has to provide means for transferring payload data from the protocol module to the host ranging from hosts that are temporally time aligned to those that are temporally fully decoupled.
The Activity diagram in Figure 20 shows the detailed activities of inform the host and their logical operating sequence. Every activity is furthermore detailed in an additional subchapter within this chapter.

- message buffer
  stands for the idea of a storage mechanism to decouple the 'host' from the 'communication module'. A message buffer does not only contain the payload but also information about the 'logical link', the 'slot' and the cycle in which they have been received or shall be transmitted.

- valid payload data
  data that was successfully received from the particular 'logical link' (receive message buffer) or that was successfully transferred from the 'host' to the 'protocol module' for transmission purpose (transmit message buffer)

ID: 828
Type: Requirement
δ For each 'message buffer' there shall be a dedicated notification whether or not the buffer currently contains 'valid payload data'.

- filter criteria
  consists of 'slot ID', 'channel ID', 'cycle counter' (and for reception additionally an (optional) 'message ID' (cf. section 10.1.3.2) criteria used to filter output and input 'payload data' inside the 'protocol module'.

- slot ID
  identifier of a specific 'slot' in which the 'communication module' may communicate.

- channel ID
  identifier of a specific 'channel' (A or B) in which the 'communication module' may communicate.

- message ID
  if a 'frame' is tagged to contain a message ID, they are the first two bytes of the 'payload segment' of the 'frame' communicated in the 'dynamic segment'.

Optionally the protocol module should perform filter mechanisms on the 'payload data' and should forward 'payload data' and reception notifications to the 'host' according to the current 'filter criteria' settings as commanded by the host.

ID: 2112
Type: Requirement
δ The ‘communication module’ shall store and make available to the ‘host’ all data of those
messages being identified in the ‘communication module’s’ message reception table.

ID: 2113
Type: Requirement
δ For all messages being identified in the ‘communication module’s’ message reception table, the
‘communication module shall monitor the status of the ‘message buffers’ and indicate to the ‘host’
when new data was received before the ‘host’ has accessed the particular ‘message buffer’
(message buffer overflow condition present).

Output
selected payload data
protocol status data

9.1.2 Transmit communication elements

This part of the FlexRay Requirements Specification deals with the transmission of communication
elements. It describes all transmission related activities, starting from the reception of communication
elements from the host down to the delivery of communication elements to the logical links.

Input
payload data

Functionality
The Activity diagram in Figure 21 shows the detailed activities of transmit communication element
and their logical operating sequence. Since Figure 21 shows the transmission only, the obvious terms
“transmitted” and “input” have been removed from the names of the data flows (e.g. data stream
instead of (transmitted) data stream and payload data instead of (input) payload data). Each activity
is detailed in an additional subsection within this section.

Note: The shaded three activities (encode communication element, transmit data stream to logical
link and bus guarding) are executed concurrently per logical link.
The ‘protocol module’ should obtain payload data from the ‘host’.

After obtaining ‘payload data’, the ‘FlexRay communication system’ should assemble communication element.

After managing the access to the ‘logical link’, the ‘FlexRay communication system’ should encode communication elements.

**ID**: 860

**Type**: Requirement

δ For each ‘logical link’ encode communication element shall be independent from other ‘logical links’.

After encoding the ‘communication element’, the ‘FlexRay communication system’ should transmit data streams to the logical link.

**ID**: 862

**Type**: Requirement
δ For each 'logical link' transmit data stream to logical link shall be independent from other 'logical links'.

Output
data stream

9.1.2.1 Obtain payload data

Input
payload data

Functionality
Obtain payload data has to provide means for transferring payload data from the host to the protocol module ranging from hosts that are temporally time aligned to those that are temporally fully decoupled.

For each transmission slot assigned to the communication module:
   check if payload data is available for one or both of the logical links and the current cycle counter
   perform a consistent read of the payload data (in the corresponding transmission message buffer)
   provide selected payload data to assemble communication element

ID : 876
Type : Constraint
δ A transmission 'message buffer' shall be assigned to a transmission 'slot' based on the 'slot ID' of the transmission 'slot' and the 'channel ID' on which the transmission shall occur.

ID : 877
Type : Constraint
δ The protocol shall support assignment of more than one transmit and/or receive 'message buffer' to a particular communication slot. In case more than one 'message buffer' is assigned, then a communication cycle controlled counter shall control which buffer is currently pending for transmission and/or reception.

In each assigned transmission 'slot' of the 'communication module', obtain payload data should check whether 'payload data' are available for one or both of the 'logical links' and the current 'cycle counter'.

When the 'host' has indicated to the 'protocol module' that a valid buffer content is available, then the 'protocol module' should transmit that data according to its configured transmission schedule. The 'protocol module' should only perform a consistent read mechanisms on 'message buffer' contents.

ID : 2091
Type : Requirement
δ The 'communication module' shall provide read access for the transmission schedule to the host or the 'communication module' shall calculate a protection value over the transmission schedule parameter space and shall make this value accessible to the 'host' at run time.

ID : 2092
Type : Requirement
δ The 'communication module' shall provide read access for the protocol parameters to the host or the 'communication module' shall calculate a protection value over the protocol parameter space and shall make this value accessible to the 'host' at run time.
**ID**: 2099  
**Type**: Requirement  

δ The ‘communication module’ shall provide read access for the message reception table to the host or the ‘communication module’ shall calculate a protection value over the message reception table parameter space and shall make this value accessible to the ‘host’ at run time.

Obtain payload data should provide assemble communication element the ability to request ‘selected payload data’ at the start of each ‘slot’ assigned to the ‘communication module’ for transmission.

**Output**  
selected payload data

### 9.1.2.2 Assemble communication element

**Input**  
selected payload data

**Functionality**  
In each communication cycle the following functions are performed within the activity assemble communication element:

- assemble static frame
- assemble dynamic frame
- assemble symbol
- wait for cycle end

The Activity diagram in Figure 22 shows the detailed activities of assemble communication element and their logical operating sequence. Each activity is detailed in an additional subsection within this section.
**Figure 22: Activities of assemble communication element**

*ID*: 930  
*Type*: Constraint  
\[ \diamond \text{ A certain 'frame ID' shall be transmitted at maximum one time in a certain 'communication cycle' on a certain 'channel'.} \]

*ID*: 2033  
*Type*: Constraint  
\[ \diamond \text{ In the 'static segment' each 'frame ID' shall be assigned to at maximum of one 'communication module' in a 'cluster'.} \]

**Assemble communication element** should maintain a ‘slot counter’.

**Output**  
communication element
9.1.2.2.1 Assemble static frame

**Input**
selected payload data

**Functionality**
While in normal active state: wait for the cycle start (= static segment start)
for each static slot in the static segment
wait for the action point
if this slot is assigned to this communication module and selected payload data for this slot exist,
assemble static frame, i.e. set attributes of header segment and add a defined padding pattern
if this slot is assigned to this communication module but no selected payload data for this slot exist,
assemble static frame, i.e. set attributes of header segment and set all payload data to a defined padding pattern
forward static frame to encode communication element
increase static slot counter

If ‘selected payload data’ for a specific ‘slot’ exists, assemble static frame should assemble the ‘selected payload data’ into a ‘static frame’, i.e. set attributes of ‘header segment’ and add a defined padding pattern to fill the missing bytes if the ‘payload data’ is smaller than the configured length of the ‘payload segment’.

**ID** : 914

**Type** : Requirement

δ If no new ‘payload data’ was provided by the ‘host’ since the last transmission of the particular ‘frame’, then assemble static frame shall assemble a ‘static frame’, i.e. shall set an explicit indication in the header that indicates the payload data is not new and set all ‘payload data’ to a defined padding pattern.

After the ‘static frame’ was assembled, assemble static frame should increase the static ‘slot counter’.

**Output**
communication element

9.1.2.2.2 Assemble dynamic frame

**Input**
selected payload data

**Functionality**
While in normal active state:
for each dynamic slot in the dynamic segment
wait for the action point
if this slot is assigned to this communication module and selected payload data for this slot exist,
assemble dynamic frame, i.e. set attributes of header segment
forward dynamic frame to encode communication element
inform encode communication element module about time to send DTS
increase dynamic slot counter

If ‘selected payload data’ for a specific ‘slot’ exists, assemble dynamic frame should assemble the ‘selected payload data’, according to the priority within the ‘frame ID’, into a ‘dynamic frame’.

After the ‘dynamic frame’ was assembled, assemble dynamic frame should increase the dynamic ‘slot counter’.
**ID**: 2034  
**Type**: Requirement  
δ After sending a ‘DTS’ the ‘protocol module’ shall increment the dynamic ‘slot counter’.  

Output  
communication element

9.1.2.2.3 **Assemble symbol**

**Input**  
symbol

**Functionality**  
While in normal active state:  
wait for the symbol window action point  
if a symbol for this symbol window exists,  
forward symbol to **encode communication element**.

If a 'symbol' for a specific 'slot' exists, assemble symbol should forward the 'symbol' to encode communication element.

Output  
communication element

9.1.2.3 **Encode communication element**

**Input**  
communication element

**Functionality**  
**Encode communication element** is responsible for encoding communication elements into binary data streams. It consists of two different encoding mechanisms, one to encode symbols and the other to encode frames. The Activity diagram in Figure 23 shows the detailed activities of **encode communication element** and their logical operating sequence. Each activity is detailed in an additional subsection within this section.

q **transmission enable signal**  
is this signal is used by the ‘communication module’ to signal whether the transmitted ‘communication element’ from ‘protocol execution module’ is valid or not, i.e. shall be transmitted or not.
**Encode communication element** should provide a mechanism to encode 'frames' and 'symbols'.

The 'protocol module' should not consider ist own 'sync frames' for performing 'clock synchronization'.

**Encode communication element** should have the possibility to enable and disable the functionality of transmit data stream to logical link, e.g. via the 'transmission enable signal'.

*ID*: 2062  
*Type*: Requirement  
δ The frame and symbol encoding shall allow a robust decoding, even at the presence jitter, bit errors and burst errors.

*ID*: 2050  
*Type*: Requirement  
δ **Encode communication element** shall provide a Hamming Distance of at least 6.

**Output**

binary data stream  
transmission enable signal

### 9.1.2.3.1 Encode frames
Input
communication element (frame)

Functionality
For each slot of the communication cycle:
- wait for frame from assemble communication element
- break down frame into individual bytes
- prepend TSS at start of the bit stream
- append FSS at end of the TSS
- create extended byte sequence by adding a BSS before each frame data byte
- assemble continuous binary data stream by concatenating the extended byte sequence
- calculate trailer CRC (over header and payload segment) and create extended byte sequence out of them
- append FES at end of the binary data stream
- for frames in the dynamic segment only:
  - append DTS after FES when assemble communication element commands it
- transmit binary data stream to transmit data stream to logical link

After encode frames has received a ‘frame’, it should break down the ‘frame’ into individual bytes.

ID : 991
Type : Requirement
δ Encode frames shall calculate a ‘trailer CRC’ (computed over ‘header segment’ and ‘payload segment’) and append it to the ‘binary data stream’.

Encode frames should transmit the ‘binary data stream’ to the ‘bus driver module’ and enable its transmission functionality.

Output
binary data stream
transmission enable signal

9.1.2.3.2 Encode symbols

Input
communication element (symbol)

Functionality
If symbol is received from assemble communication element
- prepend TSS to symbol
- transmit symbol as binary data stream to transmit data stream to logical link

Encode symbols should transmit the ‘symbol’ as a ‘binary data stream’ to the ‘bus driver module’ and enable its transmission functionality.

Output
binary data stream
transmission enable signal

9.1.2.4 Transmit data stream to logical link

Input
binary data stream
transmission enable signal

Functionality
Transmit data stream to logical link is responsible for the communication between the node and the
logical links. The binary data streams (digital signals) have to be converted into data streams (analog signals) and afterwards transmitted to the logical links.

**Output**
data stream

### 9.1.3 Perform additional services

This chapter contains requirements for services provided by FlexRay which are not part of the key functionality of FlexRay (transmission and reception of communication elements).

![Figure 24: Additional services](image)

Additional services, from the perspective of the communications protocol are non-communication related services.

#### 9.1.3.1 Network management service

This service is optional.

**Input**
- payload data
- protocol configuration data
- protocol status data

**Functionality**
The optional network management service provides means for exchanging and processing network management data. It supports high-level host-based network management protocols that provide cluster-wide coordination of startup and shutdown decisions based on the actual application state.

**network management vector (short: NM Vector)**
A number of bytes in the ‘payload segment’ of the FlexRay ‘frame’ format in a ‘frame’ transmitted in the ‘static segment’ can be used as network management vector.

To support this service minimally the following data needs to be maintained by the ‘protocol module’:

**ID**: 1039
Type : Requirement
δ The protocol module shall provide a configuration feature such that the host can configure which messages contain a ‘NM Vector’.

ID : 1041

Type : Requirement
δ The ‘protocol status data’ shall contain a snapshot of the cumulated ‘NM Vector’ that shall be updated at the end of each ‘communication cycle’ and shall remain available and valid until it is overwritten by the next snapshot at the end of the subsequent ‘communication cycle’.

Output
selected payload data

9.1.3.2 Message ID filtering service

This service is optional.

Input
payload data

Functionality
The optional message ID filtering service provides an additional filter criteria for selecting message buffers for the dynamic segment within the reception phase. This service is part of the functionality of the protocol module.

message ID
are the first two bytes of the ‘payload segment’ of the ‘frame’ communicated in the ‘dynamic segment’.

(Optionally) The protocol module should provide a mechanism to optionally filter received messages based on the ‘message ID’ contained in the particular message.

ID : 2063

Type : Requirement
δ The length of the ‘message ID’ shall be 16 bit.

ID : 1054

Type : Constraint
δ ‘Message IDs’ shall be unique in a ‘FlexRay communication system’

Output
selected payload data

9.1.3.3 Interrupt service

This service is optional.

Input
Protocol control data (interrupt masking, request for interrupt status, timer setting)

interrupt
signal generated by FlexRay to inform the host about errors
Functionality
The optional interrupt service provides a configurable interrupt mechanism to the host. Errors caused by different modules are provided to the host and the host selects which ones it is interested in. This service is part of the functionality of the protocol module.

ID : 1064  
Type : Requirement  
δ Interrupt service shall provide the ‘host’ the ability to enable and disable each interrupt individually.

ID : 1065  
Type : Requirement  
δ Interrupt service shall provide the ‘host’ the ability to enable and disable all interrupts together without having to enable or disable each interrupt individually.

ID : 1066  
Type : Requirement  
δ Interrupt service shall provide the ‘host’ the ability to clear each interrupt individually.

ID : 1067  
Type : Requirement  
δ Interrupt service shall provide an interrupt status of each interrupt to the ‘host’.

ID : 1068  
Type : Requirement  
δ Interrupt service shall provide the ‘host’ the ability to set independent timers and receive interrupts upon their timeouts.

Output
Interrupt status

9.1.3.4 Macrotick timer service

This service is optional.

Input  
macrotick counter

Functionality
The purpose of the optional macrotick timer service is to provide absolute and relative timers for the use within the FlexRay communication system. This service is part of the functionality of the protocol module.

ID : 1077  
Type : Requirement  
δ Macrotick timer service shall provide one or more absolute timers set to an absolute time in terms of cycle count and macrotick, i.e. the timer is set to expire at a determined macrotick in a determined ‘communication cycle’.

ID : 1078  
Type : Requirement
Macrotick timer service shall provide one or more relative timers set to a relative time in terms of macroticks.

Output
timeout interrupts

9.1.3.5 Temperature monitoring service

Input
temperature information

Functionality
The over temperature protection is responsible for monitoring the junction temperature on the silicon die. This service is part of the functionality of the bus driver module.

Temperature monitoring service should monitor the temperature of the ‘bus driver module’.

ID : 1087
Type : Requirement
δ When temperature monitoring service detects presence of an overtemperature condition, then the ‘bus driver module’ shall disable its transmission functionality, but the ‘bus driver module’ shall uphold receive functionality as long as possible, specifically it shall not enter a low power mode.

ID : 1088
Type : Requirement
δ When overtemperature condition is reached, then temperature monitoring service shall transmit an error to the ‘host’.

ID : 1089
Type : Requirement
δ Temperature monitoring service shall provide the temperature status data to the ‘host’.

Output
temperature status data

9.1.3.6 Voltage monitoring service

Input
power supply status data

Functionality
Voltage monitoring service is responsible for retrieving diagnostic information from the power supply and present this information to the host. This service is part of the functionality of the bus driver module.

q low power operating mode
a specific power operating mode in which the ‘communication module’ is not able to either transmit or receive data on the ‘logical link’ except for detecting the wakeup pattern (cf. section 9.2).

Voltage monitoring service should supervise the voltage level of the power supply voltage and notify the ‘host’ when an out of range condition is present or becomes present.
ID : 1099
Type : Requirement
δ When the voltage falls below a certain threshold (undervoltage), then voltage monitoring service shall autonomously switch to a 'low power operating mode'.

ID : 1100
Type : Requirement
δ When the voltage falls below a certain threshold, then voltage monitoring service shall transmit an error to the 'host'.

ID : 1101
Type : Requirement
δ When undervoltage is detected, then voltage monitoring service shall delay the reaction for a specific time (undervoltage detection filter time).

ID : 1102
Type : Requirement
δ Voltage monitoring service shall enable the 'host' to access the voltage status data.

Output
voltage status data $V_{CC}$

### 9.1.4 Other operating states

The flow of execute normal communication cycle (i.e. the functionality in the state ‘normal active’) is shown in sections 9.1.1 and 9.1.2. The details of errors, occurring during the normal communication cycle, are explained in the sections 9.1.1, 9.1.2 and 9.1.3 as exceptions. This section explains what happens if these errors occur and what state changes of the FlexRay protocol module are executed and what options the FlexRay protocol module has to get back to ‘normal active’ state.

Figure 25 shows the state changes of the FlexRay protocol module triggered by different events.
normal active
This is the state in which execute normal communication cycle is executed. The ‘communication module’ is either error free, or at least within error bounds that allow continued normal operation. Specifically, it is assumed that the ‘communication module’ remains adequately time-synchronized to the ‘cluster’ to allow continued transmission of ‘communication elements’ without disrupting the transmission of other ‘communication modules’.

normal passive
In this state synchronization with the remainder of the ‘cluster’ has degraded to the extent that continued transmission of ‘communication elements’ cannot be allowed because collisions with transmissions from other ‘communication modules’ are possible. ‘communication element’ reception continues (as described in 9.1.1) in this state in support of host functionality and in an effort to regain sufficient synchronization to allow a transition back to ‘normal active’.

halted
In this state recovery back to ‘normal active’ cannot be achieved, so execute normal communication cycle is halted in preparation for (re)configure communication module and (re)start communication module.

ready
In this state the ‘FlexRay communication system’ waits for the startup command from the ‘host’.

starting
In this state ‘the FlexRay communication system’ tries to bring up all ‘communication modules’ to ‘normal active’ state. The process start communication module is executed.
FlexRay Requirements Specification  
Chapter 9: Detailed Functional Requirements

$q$ not operational
is the state before communication between 'communication modules' can happen, i.e. the state where the 'ECU' to which a 'communication module' is deployed is still on low power or the 'communication module' is not yet configured. Only checking for wakeup patterns or configure communication module is possible in this state.

$q$ operational
contains 'ready', 'starting', 'working and 'halted'.

$q$ working
contains 'normal active' and 'normal passive'.

While the FlexRay communication system is in the state 'normal active' the following state transitions can be executed:
• state transition to 'normal passive state' after the occurrence of a clock synchronization error,
• host requests a transition to 'halted state'.

While the FlexRay communication system is in the state 'normal passive' the following state transitions can be executed:
• state transition to 'normal active' when the clock synchronization is working correctly again and transitioning back to 'normal active' is configured,
• state transition to 'halted' when the clock synchronization failure is persistent,
• host requests a transition to 'halted'.

$ID: 1131$
$Type: Requirement$
$\delta$ The 'protocol module' shall provide a configuration mechanism to enable/disable automatic return to 'normal active' as soon as the synchronization to other 'nodes' works properly.

$ID: 1132$
$Type: Requirement$
$\delta$ If a clock synchronization error persists over a specific time, then the 'FlexRay communication system' shall transition from 'normal passive state' to 'halted'.

While the FlexRay communication system is in the state 'working' the following state transitions can be executed:
• state change to 'not operational' after monitor voltage has detected a undervoltage condition (cf. section 9.1.3.6),
• host requests a transition to 'ready'.

$ID: 1140$
$Type: Requirement$
$\delta$ When monitor voltage has detected presence of an undervoltage condition, then the 'FlexRay communication system' shall transition from 'working' to 'not operational'.

$ID: 1141$
$Type: Requirement$
$\delta$ The 'FlexRay communication system' shall transition from 'working' to 'halted' when the 'host' requests so.

$ID: 1142$
$Type: Requirement$
The ‘FlexRay communication system’ shall transition from ‘working’ to ‘ready’ when the ‘host’ requests so.

### 9.2 Check for wakeup pattern

FlexRay uses a special symbol, i.e. the wakeup symbol, to wake up the communication module. When a communication module is in low power mode the bus driver module is responsible for detecting sequences of such symbols on its logical link. For other ways of starting communication in non low power mode see section 9.6 (Start communication module) below.

The refinement of the not operational state of Figure 10 is given in Figure 26.

**Figure 26: Refinement of state 'not operational'**

- **Input**
  - wakeup pattern

- **q** *wakeup symbol (WUS)*
  - is a symbol to inform other 'communication modules' that they should wakeup.
  - It consists of a defined number of low-bits followed by a defined number of high bits.

- **q** *wakeup pattern*
  - is a sequence of a configurable number of (at least two) consecutive 'wakeup symbols'.

**Functionality**

While the communication module is in its low power mode:
- monitor for presence of wakeup patterns
- notify host when a pattern was detected.

While in low power mode the *check for wakeup pattern* should be able to detect 'wakeup patterns'.

**ID**: 2270
**Type**: Requirement

δ ‘FlexRay’ shall support low power modes with the capability to activate cluster operation via the
'channels'.

\[ ID : 2271 \]
Type : Requirement
\[ \delta \] The 'bus driver' shall provide the capability to wake up the other components of its 'node' when it receives a 'wakeup pattern' on its 'channel'.

\[ ID : 2272 \]
Type : Requirement
\[ \delta \] One or more nodes in the 'cluster' shall provide the capability to wakeup when triggered by an external wakeup source.

\[ ID : 1157 \]
Type : Requirement
\[ \delta \] Upon detection of a complete 'wakeup pattern' the check for wakeup pattern shall inform the 'host' about it, so that it can configure the 'communication module' or trigger the 'communication module' to proceed in the starting process.

Output
bus driver status data (wakeup pattern detected) to the host

9.3 Configure communication module

The host can trigger the configuration of a communication module. This normally happens
\[ • \] either after being informed by the bus driver module that a wakeup pattern has been detected
\[ • \] or after the communication module came to a halt
\[ • \] or any time the communication module is in its substate ready of the state operational

A prerequisite for the configuration of a communication system is that the ECU to which it is deployed is completely powered up for a predefined time period (cf. state not operational/non low power mode in Figure 26). As soon as this state is reached the host is in full control of the further steps of bringing the communication module to its normal active state. Proper configuration is the first of these steps. For the additional steps refer to the process start communication module in section 9.6.

Among the most important configuration parameters is the communication schedule.

\[ q \] communication schedule
defines on which 'logical link' and in which (static and dynamic) 'slots' a 'communication module' is allowed to transmit 'communication elements'. (cf. Figure 27)
ID : 1170
Type : Requirement
δ Each combination of 'channel ID' and 'slot ID' in the 'static segment' shall at most be assigned to one 'communication module' for transmitting.

ID : 2114
Type : Requirement
δ The configuration mechanism shall ensure that the local 'communication module's' message reception table identifies all messages that are required by the host's local application.

ID : 2115
Type : Requirement
δ The configuration mechanism shall ensure that the FlexRay protocol parameters meet Flex Ray configuration constraints.

Input
protocol configuration data

Functionality
The communication module is first initialized with global FlexRay parameters. Then the communication module (and the bus guardian module) is initialized with specific communication module parameters.

'FlexRay' should enable the host to configure a 'communication module' with a set of system parameters.

ID : 2053
Type : Requirement
δ The 'FlexRay' configuration mechanism shall assure the consistency of all global parameters

ID : 2095
Type : Requirement
δ The 'communication module' shall disable transmission of communication elements (meaning
patterns, symbols, frames) until the host has signalled to the 'communication module' that its configuration is complete and correct.

**ID**: 2098  
**Type**: Requirement  
δ The 'communication module' shall not enable the 'bus driver' for transmission until the 'host' has signalled to the 'communication module' that the 'communication modules' configuration is complete and correct.

**ID**: 2261  
**Type**: Requirement  
δ The design-time removal of a 'frame' in a 'static slot' shall not require configuration changes in 'ECU' components of the 'FlexRay system' that do not explicitly receive the data in the removed 'frame'.

**ID**: 2260  
**Type**: Requirement  
δ The design-time addition of a 'frame' in a 'static slot' that was present in the 'communication schedule', but previously unoccupied, shall not require configuration changes in 'ECU' components of the 'FlexRay system' that do not explicitly receive the data in the added 'frame'.

**ID**: 2262  
**Type**: Requirement  
δ The design-time reassignment of the transmitter of a 'frame' from one 'ECU' to another 'ECU' shall not require configuration changes in 'ECU' components of the 'FlexRay system' other than the two 'ECUs' directly involved in the reassignment. Specifically, 'ECUs' that explicitly receive the data in the message, but are not one of the two 'ECUs' involved in the reassignment, shall not require configuration changes as a result of the reassignment.

**ID**: 2263  
**Type**: Requirement  
δ Within the limits of the rules related to 'sync frames', the design-time modification of the sync status of a 'frame' (i.e., whether a 'frame' is a 'sync frame' or not) shall not require configuration changes to any 'ECU' component of the 'FlexRay system' other than the 'ECU' that transmits the 'frame'.

**ID**: 2264  
**Type**: Requirement  
δ Within the limits of the rules related to 'startup frames', the design-time modification of the startup status of a 'frame' (i.e., whether a 'frame' is a 'startup frame' or not) shall not require configuration changes to any 'ECU' component of the 'FlexRay system' other than the 'ECU' that transmits the 'frame'.

**ID**: 1390  
**Type**: Requirement  
δ When the 'communication module' detects a reset condition, then it shall automatically mark its configuration state as "not configured".

**Output**
protocol status data (status data about the configuration to the host)
9.4 Configure bus guardian module

*ID*: 1227  
*Type*: Requirement  
δ The configuration of the ‘bus guardian module’ shall be independent from the configuration of the ‘communication module’. The ‘communication module’ configuration shall be distinct from the ‘bus guardian module’ configuration, specifically the ‘protocol module’ shall not configure the ‘bus guardian module’.

9.5 Clock synchronization

In a distributed communication system each communication module has its own clock. Because of temperature fluctuation, voltage fluctuations, and production tolerances of the timing sources, the internal time bases of the various communication modules diverge after a short time, even if all the internal time bases are initially synchronized.

A basic assumption for a time-triggered system is that every FlexRay communication module in the cluster has approximately the same view of time and this common global time is used as the basis for communication timing for each communication module.

FlexRay uses a distributed fault tolerant clock synchronization mechanism in which each communication module individually synchronizes itself to the FlexRay communication system by observing the timing of transmitted sync frames from other communication modules.

In order to define the requirements for clock synchronization we define the following terms:

- **offset**  
  is the difference between two corresponding points in time measured by means of a reference clock of an outside observer (e.g. in "real-time"). The offset is measured and expressed in microticks. If the clock of one communication module is at microtick 12 in cycle 3 and the clock of another communication module is at microtick 17 in cycle 3 at a corresponding (globally agreed) macrotick, their offset is 5 microticks.

- **rate difference**  
  is the difference in the number of 'micro ticks' that a local clock of one 'communication module' needs for a 'communication cycle' compared to the local clock of another 'communication module'. The number of 'microticks' is set during (local) configuration of a communication module and can be adjusted by the clock synchronization process.

- **precision** (of a FlexRay communication system)  
  is the maximum difference in real-time(observed by an outside observer) between any two corresponding events in the 'FlexRay communication system'. (Corresponding events are e.g. cycle start, start of static slot, start of dynamic segment, ....)

With these definitions we can phrase the following high level requirement for clock synchronization:

*ID*: 1264  
*Type*: Requirement  
δ Clock synchronization shall attempt to synchronize the local clock to the clock information contained in the received 'sync frames'.

*ID*: 2064  
*Type*: Requirement
δ Synchronisation process shall tolerate failures of clocks in 'communication modules' within the specified boundaries of the applied synchronisation algorithm.

ID : 2065
Type : Requirement
δ Failures in the synchronisation process shall be reported to the 'host'.

ID : 2100
Type : Requirement
δ The 'communication module' shall provide status information to the 'host' about whether global synchronisation has been achieved. (i.e. whether the CC clock is running unsynchronized or synchronised).

The activity diagram in Figure 30 shows the most important steps of clock synchronization. Details are described in the following text.

Figure 30: clock synchronization
Functionality
In parallel to generate macroticks the following activities are performed (depending on the value of the cycle counter:

**For even cycles** (i.e. cycle counter is an even number)
- during the static segment: **measure deviation values** (resulting in a deviation table for the even cycle)

**For odd cycles** (i.e. cycle counter is an odd number)
- during the static segment: **measure deviation values** (resulting in a deviation table for the odd cycle)
- after static segment: **calculate offset correction** (result: a number of microticks, based on values measured in this odd cycle)  
- **calculate rate correction** (result: a positive or negative number of microticks, based on measurements over a double cycle, i.e. both deviation tables from the even and the odd cycle)

- during NIT: (but of course after the calculations are finished) **apply offset correction**

**ID**: 1282

**Type**: Requirement

- Apply offset correction and calculate rate correction shall both be finished before the next ‘communication cycle’ starts.

### 9.5.1 Measure deviation values

**Input**
- action point in static slot reached: This is the expected arrival time of a frame
- primary time reference point: This is the actual arrival time of a frame

**q sync frame**
- a ‘frame’ whose ‘header segment’ contains a ‘sync frame indicator’. The deviation measured between such a ‘frame’ arrival time and its expected arrival time is used by the clock synchronization algorithm.

**q startup frame**
- a ‘frame’ whose ‘header segment’ contains a ‘startup frame indicator’. Other ‘nodes’ may use the time related information from such a ‘frame’ to initialize their ‘communication cycle’. Startup frames are always also ‘sync frames’.

**Functionality**
For each slot in the static segment of the communication cycle:
- When action point of the slot is reached take a time stamp for the expected arrival time
- When primary time reference point is received: calculate the time difference between the expected arrival time and the actual arrival time
- Store this time difference together with the information, whether it is a valid frame and whether it is a sync frame in the even or the odd part of the deviation table.

**Exception**
- If more than the configured number of sync frames per logical link is received in one cycle
  - use the configured number of sync frames for rate and offset correction
  - inform host about this error

Clock synchronization shall calculate the time difference between the expected arrival time and the actual arrival time of each sync ‘frame’ in the ‘static segment’ of the ‘communication cycle’. 
ID : 1301  
Type : Requirement  
δ Clock synchronization shall report an error if more than the maximum configured number of 'sync frames' per 'logical link' are received in one cycle. In such a case it shall use the time differences of the configured number of 'sync frames' for the calculation of rate and offset correction.

ID : 2035  
Type : Requirement  
δ When clock synchronization has reported an error as stated above, it shall use the time differences of the configured number of sync frames for the calculation of rate and offset correction.

Output  
deviation table (for the even or the odd cycle), consisting of the three values for each slot and for both logical links:  
• time difference (in microticks)  
• valid frame (true/false)  
• sync frame (true/false)

9.5.2 Calculate offset correction

q offset correction is a signed integer that indicates how many microticks the 'communication module' should shift the start of its cycle. Negative values mean that the 'network idle time' should be shortened (making the next cycle start earlier). Positive values mean the 'network idle time' should be lengthened (making the next cycle start later).

Input  
deviation table (of the odd cycle)

Functionality  
for all valid sync frames received during the odd cycle:  
create a list of the deviation times:  
if only one logical link has a valid sync frame  
take its time difference  
if both logical links have valid sync frames  
use the smaller value of the time differences.  
execute a fault tolerant midpoint algorithm with this list of deviation times, creating the most likely offset correction.

Exception  
If the offset correction is outside predefined limits (set during configuration) raise an error and use maximum or minimum correction value  
If necessary an external correction value (e.g. from the host) can be added to the calculated correction value to synchronize a FlexRay communication system with other FlexRay communication systems or an external time base (e.g. GPS-time).

Calculate offset correction shall create a list of deviation times for each static slot in which a sync frame was received according to the following rules:  
• - if a sync frame has been received on only one logical link, store its time difference  
• - if a sync frame has been received on both logical links, store the smaller value of the time differences

Calculate offset correction shall execute a fault tolerant midpoint algorithm with the list of deviation times to determine the most likely 'offset correction'
ID : 1322
Type : Requirement
δ When the 'protocol module' detects a problem to synchronize its clock to other 'nodes', then it shall notify the 'host'.

ID : 1323
Type : Requirement
δ The 'FlexRay communication system' shall enable the host to modify the calculated 'offset correction' by adding an external correction value.

Output
offset correction
protocol status data (offset correction outside configured limits)

9.5.3 Calculate rate correction

q rate correction
a signed integer indicating how many microticks the 'communication module's' cycle length should be changed. Negative values mean the cycle should be shortened; positive values mean the cycle should be lengthened.

Input
development table (of the even and the odd cycle)

q valid sync frame
every frame where the following conditions apply: a valid frame was received, the sync frame indicator of the received frame is set, and no content error was detected.

Functionality
determine pairs of valid sync frames received in the same slot on the same logical link on two consecutive cycles
calculate their difference in time deviation
If such pairs exist on both logical links take the average of the two values.
if no pair was found report this error
otherwise apply midpoint algorithm to all values

Calculate rate correction shall determine the time differences between pairs of 'valid sync frame' received over an even and an odd communication cycle.

If pairs of 'valid sync frames' have been received over the last even and odd 'communication cycle', then calculate rate correction shall store the average value in their time deviations in a deviation list

If no pairs of 'valid sync frames' have been received over the last even and odd 'communication cycle', then calculate rate correction shall inform the host about this fact.

Calculate rate correction shall use a midpoint algorithm on the deviation list to calculate the most likely 'rate correction'.

Output
rate correction (to be used by generate macroticks from the next cycle start onwards)
protocol status data (rate correction cannot be determined)
9.5.4 Apply offset correction

**Input:**
offset correction, i.e. a number of microticks, based on values measured in a single cycle

**Functionality**
use the offset correction to change the number of microticks per cycle during NIT

**Apply offset correction** shall change the number of microticks per cycle by the calculated ‘offset correction’ during the ‘network idle time’ before the next cycle start.

**Output**
new value of microticks per cycle, to be used from the next cycle start onwards.

9.5.5 Generate macroticks

**Input**
microtick
rate correction

**Functionality**
at the beginning of each communication cycle increase the cycle counter (modulo 64) and reset the macroticks (as an atomic action).
evenly distribute the configured number of microticks per cycle and the rate correction over the communication cycle.
create macroticks at appropriate microticks.

At the beginning of each ‘communication cycle’ **generate macroticks** shall increase the ‘cycle counter’ by one (modulo 64)

**Generate macroticks** shall evenly distribute the configured number of microticks and the calculated ‘rate correction’ (from the previous even and odd cycle) over the ‘communication cycle’.

**Generate macroticks** shall create a macrotick signal after the evenly distributed number of microticks.

**Generate macroticks** shall create a cycle start signal at the start of each new ‘communication cycle’.

**Output**
cycle start signal
cycle counter
macroticks

9.6 Start communication module

A TDMA based communication scheme requires synchrony and alignment of all communication modules that participate in the communication. A fault-tolerant, distributed startup strategy is required for initial synchronization of all communication modules.

The execution of this process **Start communication module** depends on the configuration of the communication module. They can be configured as coldstart nodes or non-coldstart nodes as shown in Figure 31.
The action of initiating the startup process is called coldstart. Only a limited number of communication modules may initiate a startup, that are called coldstart nodes.

coldstart node

a ‘node’ capable of initiating the communication startup procedure of the ‘FlexRay communication system’ by sending ‘startup frames’. There are two different kinds of coldstart nodes, the ‘leading coldstart node’ and the ‘following coldstart node’.

ID : 1376
Type : Requirement
δ The ‘FlexRay’ configuration mechanism shall verify that ‘coldstart nodes’ are connected to all configured ‘logical links’.

ID : 1377
Type : Constraint
δ ‘FlexRay’ shall support system startup and operation when two or more nodes are present in the system and at least one of these nodes is configured to be a coldstart node.

ID : 2020
Type : Constraint
δ ‘FlexRay’ shall support startup and operation of a 3-or-more-node system when one of the nodes in the system erroneously does not participate in communication.

ID : 2021
Type : Constraint
δ ‘FlexRay’ shall support startup and operation between nodes which are attached to two channels when one of the two channels is not available for communication because of presence of a fault condition.

ID : 2103
Type : Requirement
A failure during the start-up procedure shall be detected, localized and notified in the FlexRay communication system.

**ID:** 2111  
**Type:** Requirement

The wake-up and start-up of the 'communication system', the integration of 'nodes' powered on later and the reintegration of failed 'nodes' shall be fault-tolerant against:

- the temporary/permanent failure of one or more 'communication modules' (down to one module sending in the static part for mixed or pure static configurations),
- the temporary/permanent failure of one or more communication 'channel(s)' in a redundant configuration, and
- the loss of one or more 'frames'.

To say that “a cluster is able to start up in the presence of a fault” has a meaning that depends on the type of the fault.

- **Fault class 1:** The fault is associated to a channel or a star: All nodes which are intended to participate in communication and are connected to the other channel reach a state where they communicate to one another as scheduled. They reach this state within a defined maximum time.
- **Fault class 2:** The fault is associated to a node: All fault-free nodes which are intended to participate in communication reach a state where they communicate to one another as scheduled. They reach this state within a defined maximum time.
- **Fault class 3:** Transient fault: All nodes which are intended to participate in communication reach a state where they communicate to one another as scheduled. They reach this state within a defined maximum time (For a value see the requirements specification).

**List of faults**

Whether startup should work in presence of such a fault is indicated in the column “Startup should work”, according to the criteria above. If, for a certain fault, startup is not required, it may nevertheless work. In particular, presence of Bus Guardians may enable startup in such cases anyhow.

The listed faults are not necessarily independent from one another.

<table>
<thead>
<tr>
<th>Nr</th>
<th>Description of faulty behaviour</th>
<th>Possible physical cause(s)</th>
<th>Startup should work</th>
<th>Fault class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>One of two channels is in a frozen logic state. Channel is locked in one of the bus conditions: &quot;idle&quot;, &quot;idle_LP&quot;, &quot;Data_0&quot;, &quot;Data_1&quot;. (Hint: terms are defined in chapter 6 of the physical layer specification).</td>
<td>A short-circuit condition is present</td>
<td>For 2-channel systems</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>A single or both bus wires of one channel is/are shorted to a fixed voltage.</td>
<td>A short-circuit condition is present</td>
<td>For 2-channel systems</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>depends on specific pin that is interrupted</td>
<td>A single arbitrary I/O signal/pin connecting two of the components communication module, bus guardian or bus driver has become disconnected inside one of the nodes in the cluster. e.g. bus driver’s Transmit Enable input disconnected.</td>
<td>For 1+2 channel systems</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Requirement</td>
<td>Cause/Effect</td>
<td>Impact</td>
<td>System Configurations</td>
</tr>
<tr>
<td>----</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>3b</td>
<td>Signal wave form degraded on one channel</td>
<td>A single bus termination is unit disconnected.</td>
<td>For 2-channel systems</td>
<td>1</td>
</tr>
<tr>
<td>3b</td>
<td>Signal wave form degraded on one channel</td>
<td>A single bus termination is unit disconnected.</td>
<td>For 1 channel systems: under certain conditions (topology, bit rate)</td>
<td>2</td>
</tr>
</tbody>
</table>
| 4  | A single arbitrary node in the cluster is not communicating on all attached channels for whatever reason. It doesn't transmit anything, or it starts to transmit somewhat later than the other nodes. | (A) Transmit data line between communication module and bus driver interrupted.  
(B) bus driver enable line interrupted.  
(C) An incoming link failure is present.  
(D) outgoing link failure (E) interruption in connector  
(F) plug becomes disconnected | For 1+2 channel systems                                                   | 2                     |
| 5  | A single clock oscillator in the cluster erroneously runs at a wrong frequency. At least the following six cases shall be covered: The faulty oscillator runs at 1/3, 5/3 and 7/3 times its nominal frequency. The oscillator doesn't run at all. The oscillator runs just above or below the specified frequency tolerance band (e.g. 2000 ppm). | (A) Degradation of oscillator present.  
(B) Oscillator circuit sensitive to value changes.  
(C) oscillator runs at a wrong harmonic frequency. | For 1+2 channel systems                                                   | 2                     |
| 6  | A node (e.g. coldstart node) cannot receive any communication element on all its attached channels | Receive data line between bus driver and communication module interrupted | For 1+2 channel systems                                               | 2                     |
| 7  | A two-channel-node node (e.g. coldstart node) cannot receive any communication element on one of its channels | Receive data line between bus driver and communication module interrupted | For 2-channel systems                                               | 2                     |
| 8  | A star cannot receive any communication element of a certain branch         | Receive comparator in the bus driver doesn’t work                           | For 2-channel systems                                               | 1                     |
| 8  | A star cannot receive any communication element of a certain branch         | Receive comparator in the bus driver doesn’t work                           | For 1 channel systems: under certain conditions (e.g. topology)     | 2                     |
| 9  | A periodic reset event is present in a node                                | Supply voltage at lower limit                                               | For 1+2 channel systems                                               | 2                     |
| 10 | A periodic reset event is present in an active star                         | Supply voltage at lower limit                                               | For 2-channel systems                                               | 1                     |
| 11 | One of the bus wires of one channel is interrupted (no star in the system) | An interruption at some (inline) connector is present                       | For 2-channel systems                                               | 1                     |
| 12 | One single bit of a communication element on one channel flips             | EMI                                                                          | For 1+2 channel systems                                               | 3                     |
| 13 | For a given time of less than one frame length, all present channels are forced to an arbitrary pattern | EMI                                                                          | For 1+2 channel systems                                               | 3                     |
| 14 | A bus driver in a node cannot receive anything                             | Transceiver can’t receive anything due to an internal fault                 | For 1+2 channel systems                                               | 2                     |
### FlexRay Requirements Specification

**Chapter 9: Detailed Functional Requirements**

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
<th>Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>14b</td>
<td>A bus driver in a star cannot receive anything</td>
<td>For 2 channel systems</td>
</tr>
<tr>
<td>15</td>
<td>A bus driver in a node cannot transmit anything</td>
<td>For 1+2 channel systems</td>
</tr>
<tr>
<td>15b</td>
<td>A bus driver in a star cannot transmit anything</td>
<td>For 2 channel systems</td>
</tr>
<tr>
<td>16</td>
<td>A node sends sporadically WUSs. After occurrence, the fault does not manifest itself for at least 10 communication cycles.</td>
<td>For 1+2 channel systems</td>
</tr>
<tr>
<td>17</td>
<td>A coldstart node sends sporadically CASs. After occurrence, the fault does not manifest itself for at least 10 communication cycles.</td>
<td>For 1+2 channel systems</td>
</tr>
<tr>
<td>18</td>
<td>No node is operational except of 2 fault free nodes and these two nodes are assigned to perform startup (“coldstart nodes”). (Req ID 326).</td>
<td>For 1+2 channel systems</td>
</tr>
</tbody>
</table>

#### Leading coldstart node
- The first ‘coldstart node’ that is triggered by its ‘host’ to initially start communication.

#### Following coldstart node
- A ‘coldstart node’ that integrates upon a ‘leading coldstart node’.

**ID**: 1380  
**Type**: Requirement  
- The ‘FlexRay’ configuration mechanism shall ensure that only ‘coldstart nodes’ transmit ‘startup frames’.

#### Non-coldstart node
- A non-coldstart node requires at least two ‘startup frames’ from distinct ‘coldstart nodes’ for integration. This condition ensures that each ‘non-coldstart node’ always joins the majority of the ‘coldstart nodes’.

**Input**  
startup request from host

**Functionality**  
- If the node is a coldstart node: c.f. section 9.6.1  
- If the node is a non-coldstart node c.f. section 9.6.2

**ID**: 2037  
**Type**: Requirement  
- When a reset condition occurs, then ‘FlexRay’ shall disable start communication module.

**ID**: 2038
Type : Requirement
δ In 'working' state, 'FlexRay' shall periodically provide a protection value (e.g. CRC or similar) to the 'host' for detection of configuration data inconsistencies.

ID : 1391
Type : Requirement
δ The 'protocol module' shall perform start communication module simultaneously on all 'logical links'.

ID : 1392
Type : Requirement
δ When a 'node' attempts late to integrate to the communication, the 'communication module' shall provide a mechanism that this shall not disturb the startup process of the 'nodes' that have started early to integrate.

ID : 1393
Type : Requirement
δ Start communication module shall not disturb other 'nodes' working in the state 'normal active'.

Output
startup complete

9.6.1 State: Starting coldstart nodes

Input
startup request from host

Functionality
Starting coldstart nodes is responsible for initiating the startup process for leading coldstart node and integrating following coldstart nodes, thus bringing these nodes to the state normal active. The coldstart nodes start in the coldstart inhibit mode to prevent initiation of the startup, until the host triggers the mode change. Each of the coldstart nodes finishes this process as soon as stable communication with one of the other coldstart nodes is established.

Output
status data to host
Each of the ‘coldstart nodes’ shall transmit ‘startup frames’ until a stable communication with one of the other ‘coldstart nodes’ is established, until the host withdraws the request to startup communication or after a defined and limited time is exceeded.

The ‘coldstart node’ shall listen to potential communication on the ‘logical links’ to determine whether it shall act as ‘leading coldstart node’ or as ‘following coldstart node’.

If the ‘coldstart node’ has not received a valid startup frame within a configurable time, it shall act as ‘leading coldstart node’ and shall send startup frames on the ‘logical links’.

If the ‘coldstart node’ has received a valid startup frame within a configurable time, it shall act as ‘following coldstart node’ and shall use the startup frames to synchronize its clock.

The ‘communication module’ shall automatically terminate any attempt to startup communication
after a defined and limited time. The 'communication module' shall do so even if it is configured to be a 'coldstart node' and when multiple nodes initiate a coldstart simultaneously. In addition, the 'communication module' shall do so when due to a local fault condition the 'communication module' cannot detect any 'communication elements' being present on a 'channel'.

9.6.2 State: Integrating non-coldstart nodes

Input
startup request from host

Functionality
After at least two coldstart nodes have completed their startup process, non-coldstart nodes try to integrate into the startup process as shown in Figure 34. This ensures, that non-coldstart nodes always join the majority of the 'coldstart nodes'.

ID : 1477
Type : Requirement
δ Integrating 'non-coldstart nodes' may start their integration before 'coldstart nodes' have finished their startup.

ID : 2069
Type : Requirement
δ If configured to be a 'non-coldstart node', the 'protocol module' shall wait for the occurrence of defined conditions before starting its integration into the 'cluster'.

ID : 1478
Type : Requirement
δ Integrating 'non-coldstart nodes' shall not finish their startup until at least two 'coldstart nodes' have finished their startup.

ID : 2070
Type : Requirement
δ An 'non-coldstart node' shall listen to the communication on the 'logical links' until it detects valid startup frames. The 'non-coldstart node' shall use these 'frames' to initialize its communication cycle timing.

Output
status data to host

9.7 Bus guarding

This functionality is optional in FlexRay.

ID : 2266
Type : Requirement
δ The 'FlexRay system' shall meet 'bus guardian' requirements for systems that support 'communication schedules' in which 'ECUs' transmit in up to 16 distinct 'static slots'.
Clarification - this requirement does not preclude solutions that support more than 16 transmission slots per node. It merely states that any acceptable solution must work in system where an ECU transmits in 16 slots
ID : 2258  
**Type** : Requirement  
δ Faulty operation of any 'channel component' of the system shall not affect / disrupt the transmission and reception capability of any non-faulty 'ECU' on more than one 'channel'.

ID : 2267  
**Type** : Requirement  
δ A single faulty 'channel component' shall not cause an operating 'FlexRay communication system' to prematurely shut down.

ID : 2259  
**Type** : Requirement  
δ The 'FlexRay system' shall prevent a 'communication module' from transmitting during the 'NIT'.

ID : 2268  
**Type** : Requirement  
δ The 'FlexRay system' shall prevent a 'communication module' from transmitting 'frames' during the 'symbol window'.

### 9.8 Shutdown communication module

**Input**
host requests shutdown

**Functionality**

ID : 1611  
**Type** : Requirement  
δ If the 'host' requests shutdown, 'FlexRay' shall initiate a coordinated termination of communication sessions.

**Output**
none

### 9.9 Repeating function & branch error handling

This Use Case describes the functionality performed on active stars. The main functionality is forwarding incoming data streams on one branch without any change to all other connected branches.

**Input**
data streams on a branch

**Functionality**
Forward incoming data stream on all other connected branches. If errors are detected they shall be reported and the erroneous branch shall be deactivated.

ID : 1622  
**Type** : Requirement  
δ An 'active star' shall deactivate a certain 'branch' when on this 'branch' one of the following
conditions is present:
1) permanent “0” on the ‘logical link’ or
2) permanent “1” on the ‘logical link’ or
3) permanent noise on the ‘logical link’.

ID : 1623
Type : Requirement
δ Repeating function & branch error handling shall activate a certain ‘branch’ again when none of
the conditions stated above (in Req 1622) is present any more.

Output
Unmodified data streams on all other branches
10 Look and Feel Requirements

This chapter is considered not applicable since FlexRay does not have a user interface.

11 Usability Requirements

This chapter is considered not applicable since FlexRay is not used by humans, but applications.

12 Performance Requirements

12.1 Speed and latency requirements

\[ ID : 1630 \]
\[ Type : Requirement \]
\[ \delta \] When 'FlexRay' is in the state 'working', then the differences in the view of time between the 'nodes' shall be less than a pre-defined limit.

\[ ID : 1634 \]
\[ Type : Requirement \]
\[ \delta \] The 'FlexRay communication system' shall startup and shutdown within a time that is similar as for state of the art automotive bus systems (e.g. CAN, LIN)

\[ ID : 1637 \]
\[ Type : Requirement \]
\[ \delta \] The timing behavior of the 'static segment' of 'FlexRay' shall not change when 'communication modules' are added or removed while the 'FlexRay communication system' is in the state 'working' and two synchronisation nodes are present.

12.2 Safety relevant requirements

The following requirements are draft collection. They will be detailed by the safety working group.

\[ ID : 1651 \]
\[ Type : Requirement \]
\[ \delta \] The 'FlexRay communication system' shall not modify 'payload data'.
ID : 2087
Type : Requirement
δ The probability of a CCF between the two 'bus drivers' of a 'node' shall be below a defined limit.

ID : 2088
Type : Requirement
δ The probability of a CCF between the 'communication module' and the 'bus driver' of a 'node' shall be below a defined limit.

ID : 2089
Type : Requirement
δ The probability of a CCF between the 'host' and the 'bus drivers' of a node shall be below a defined limit.

ID : 2292
Type : Requirement
δ The 'protocol module' shall not transmit a non-safety relevant message as a safety relevant message to the 'channel'.

ID : 2293
Type : Requirement
δ The 'protocol module' shall not present a safety relevant message as a different safety relevant message to its 'host'.

12.3 Precision requirements

ID : 1654
Type : Requirement
δ 'FlexRay' shall support operation when oscillators are used with a frequency tolerance of up to 0.15%.

12.4 Reliability and Availability requirements

ID : 1656
Type : Requirement
δ The 'FlexRay communication system' shall ensure a failure rate of less than $10^{-10}$ failures per hour while in the state 'working'.

ID : 1657
Type : Requirement
δ The 'FlexRay communication system' shall ensure a failure rate of less the $10^{-6}$ failures while in the state 'starting'.

ID : 2104
Type : Requirement
There shall be a 99% probability of detecting communication loss or degradation or fail operational states.

**ID : 2106**  
**Type : Requirement**

A failure during the wake-up-, initialisation- or start-up procedure shall be detected, localized and notified in the communication system.

### 12.5 Robustness requirements

**ID : 1661**  
**Type : Requirement**

- ‘FlexRay’ hardware shall be robust against typical automotive environmental conditions such as jitter, cause by injection of RF-fields, common mode voltages, high temperature, electrostatic discharge.

**ID : 1662**  
**Type : Requirement**

- The non-faulty ‘communication modules’ of a ‘FlexRay communication system’ shall be able to communicate correctly on at least one ‘logical link’ even if there is an arbitrary fault in a ‘protocol module’, a ‘bus driver module’ or a ‘bus guardian module’.

**ID : 1663**  
**Type : Requirement**

- A ‘FlexRay communication system’ shall maintain the precision of the clock synchronization between ‘communication modules’ in the presence of arbitrary faults disturbing the communication on one ‘logical link’.

**ID : 2108**  
**Type : Requirement**

- No single failure shall inhibit communication on both ‘channels’ / ‘stars’ for more than a defined time.

**ID : 2109**  
**Type : Requirement**

- ‘Nodes’ shall not propagate a communication error from one FlexRay ‘channel’ to the other such that the communication is inhibited for more than a defined time.

It is recommended to design systems such that a short-circuit between the ‘channels’ of a two-channel ‘FlexRay system’ is prevented.

### 12.6 Capacity requirements

**ID : 1665**  
**Type : Requirement**

- FlexRay shall be capable to receive and transmit substantially more ‘payload data’ in a single
'slot' compared to a CAN protocol frame, i.e. substantially more than 8 bytes.

12.7 Scalability or extensibility requirements

One ‘repeater module’ should be deployed to one ‘active star’.

ID : 1679
Type : Requirement
\[ \delta \] The interface between ‘FlexRay modules’ and the ‘host’ should not be influenced if future bandwidth exceeds 10MBit/s.

ID : 1680
Type : Requirement
\[ \delta \] FlexRay shall allow flexible adaptation of the following parameters:
- number of frames transmitted by a ‘communication module’ (varies from none to 16 or more)
- length of ‘frames’ in the ‘static segment’
- length of ‘frames’ in the ‘dynamic segment’
- overall duration of the ‘communication cycle’
- duration of the segments of the communication cycle (static segment, dynamic segment, symbol window, NiT)

ID : 1681
Type : Requirement
\[ \delta \] ‘FlexRay’ shall support single channel operation.

Notes: Functions which need presence of a second channel do not have to be supported in single channel operation. It is acceptable that the fault-tolerance is lower in single channel operation compared to dual channel operation.

13 Operational Requirements

13.1 Expected physical environment

ID : 1687
Type : Requirement
\[ \delta \] ‘ECUs’ to which a ‘FlexRay communication system’ is deployed shall meet automotive temperature requirements. General temperature requirements include a range of -40 to +125 degrees Celsius. Special applications may require higher temperatures, e.g. near brake actuators.

ID : 1688
Type : Requirement
\[ \delta \] ‘ECUs’ to which a ‘FlexRay communication system’ is deployed shall meet automotive requirements in the areas temperature cycling, pressure, vibration.
13.2 Productization Requirements

No requirements for this section have been identified.

14 Maintainability and Support Requirements

14.1 Maintainance requirements

No requirements for this section have been identified.

14.2 Special conditions that apply to the maintenance of the product

This section is considered not applicable (Application level support should be considered elsewhere)

14.3 Supportability requirements

This section is considered not applicable (Application level support should be considered elsewhere)

14.4 Portability requirements

ID : 1708
Type : Requirement
δ The functionality and the behavior of a 'communication module' shall be independent of the implementation of the physical layer (as far as possible).
ID : 1709  
Type : Requirement  
δ The functionality and the behavior of a ‘communication module’ shall be independent of the bitrate used to communicate. (as far as possible).

ID : 1710  
Type : Requirement  
δ Future versions of the FlexRay functionality and the behavior should be backward compatible with earlier versions.

Remark: This is seen as a requirement as long as there is no severe deficiency identified within the FlexRay Specification v2.1 that necessitates a break of the backward compatibility.

ID : 1711  
Type : Requirement  
δ FlexRay shall allow to add a listen-only ‘communication module’ to an existing ‘FlexRay communication system’ without having to change anything in the existing ‘communication modules’.

15 Security Requirements

15.1 Integrity requirements

No requirement for this section have been identified.

15.2 Immunity requirements

ID : 1721  
Type : Requirement  
δ Check for wakeup pattern shall shall not indicate a wakeup event when only EMC disturbance is present.
16 Open Issues

16.1 Issues that have been raised and do not yet have a conclusion

Functionality and deployment of the bus guardian module, including related issues of the bus guardian schedule monitoring module

17 Waiting Room

This list of the following requirements is postponed:

When monitor temperature has detected an overheat condition and the 'host' has enabled a state transition, then 'FlexRay communication system' should transition from 'normal active state' to 'receive only state'.

When the protocol module aborts transmission in the dynamic segment due to a frame length error condition, then it shall continue regular synchronized communication, e.g. it shall not change its operating mode because of such a condition.

When a receiving protocol module detects presence of an aborted frame in the dynamic segment, then it shall signal an error condition to it's host and shall continue regular synchronized operation, e.g. it shall not change its operating mode because of such a condition.
Appendix A Abbreviations and Definitions

q **active star**
   a device that allows ‘data streams’ received on one ‘branch’ to be transferred (or duplicated) to all other ‘branches’ connected to it.

q **action point**
   designated boundaries at which a ‘communication module’ performs a specific action in alignment with its local time base, e.g. starting the transmission of a ‘frame’.

q **binary data stream**
   is a ‘communication element’ represented as a sequence of zeros and ones communicated between ‘bus driver module’ and ‘protocol module’.

q **branch**
   a direct physical connection
   - between two or more ‘ECUs’ or
   - between an ‘active star’ and one or more ‘ECUs’ or
   - between two ‘active stars’.

q **BSS (byte start sequence)**
   is used to provide a bit stream timing information to the receiving ‘communication module’.

q **bus connection**
   a broadcast connection between two or more end points (‘ECUs’ or ‘active stars’) that is capable of transmitting ‘data streams’. ‘Bus connections’ are always electrical.

q **channel**
   a ‘point-to-point connection’, ‘bus connection’ or a ‘star connection’ between two or more ‘ECUs’.

q **channel component**
   A channel component is any component in a ‘FlexRay communication system’ that is not part of an ECU. Examples would include cabling, ‘active stars’, ECU-external ‘bus guardian modules’, etc.

q **channel ID**
   identifier of a specific ‘channel’ (A or B) in which the ‘communication module’ may communicate.

q **cluster**
   a set of ‘ECUs’, each of which is connected by one or two ‘channels’.

q **coded frame**
   is a ‘frame’ augmented by a sending ‘communication module’ with a number of control elements. It allows the receiving ‘communication module’ to recognize various start and end events. Control elements are: ‘TSS’ (transmission start sequence), ‘FSS’ (frame start sequence), ‘BSS’ (byte start sequence), ‘FES’ (frame end sequence), and ‘DTS’ (dynamic trailing sequence).

q **coded symbol**
   is a ‘symbol’ augmented by a sending ‘communication module’ with a ‘TSS’ (transmission start sequence).
   Note: WUS is sent without TSS.

q **coldstart node**
   a ‘node’ capable of initiating the communication startup procedure of the ‘FlexRay
communication system’ by sending ‘startup frames’. There are two different kinds of coldstart nodes, the ‘leading coldstart node’ and the ‘following coldstart node’.

q **communication cycle**
the key periodically repeated communication structure of ‘FlexRay’. It consists of a ‘static segment’, an optional ‘dynamic segment’, an optional ‘symbol window’ and a ‘network idle time’.

q **communication cycle segment**
is a part of the ‘communication cycle’. There are four specific kinds of communication cycle segments: ‘static segments’, ‘dynamic segments’, ‘symbol windows’ and ‘network idle times’.

q **communication element**
the basic logical unit of communication of ‘FlexRay’. A ‘communication element’ is either a ‘frame’ or a ‘symbol’.

q **communication module**
contains the main functionality of a ‘FlexRay communication system’ responsible for transmitting and receiving ‘communication elements’ on ‘logical links’.

q **communication schedule**
defines on which ‘logical link’ and in which (static and dynamic) ‘slots’ a ‘communication module’ is allowed to transmit ‘communication elements’. (cf. Figure 27)

q **Composability**
is a system design principle that deals with the inter-relationships of components. A highly composable system provides recombinant components that can be selected and assembled in various combinations to satisfy specific user requirements.

q **constraint**
a pre-existing restriction that limits the solution a system can provide. This can be a design constraint (e.g. shall be embeddable in ...) or a project constraint (e.g. production price less than ...). Constraints are separated from requirements, because it is not possible to test a constraint by testing the system.

q **cycle count**
the number of the ‘communication cycle’ in which this ‘frame’ shall be transmitted.

q **data stream**
is a communication element represented as electrical or optical signal transported via ‘logical links’.

q **DTS (dynamic trailing sequence)**
is used to indicate the exact end of the ‘dynamic frame’, to prevent premature channel idle detection. This is required for the stability of certain types of physical layers.

q **dynamic frame**
a ‘frame’ transmitted or received in a ‘dynamic slot’ of the ‘dynamic segment’ of the ‘communication cycle’. ‘Dynamic frames’ can have variable lengths.

q **dynamic payload segment**
is a ‘payload segment’ of a ‘frame’ in which ‘payload data’ of variable length is transmitted or received in the ‘dynamic segment’ of the ‘communication cycle’.

q **dynamic segment**
a part of the ‘communication cycle’ in which ‘dynamic frames’ are transmitted or received. Within the dynamic segment a dynamic mini-slotting based scheme is used to arbitrate communication.
q **dynamic slot**
   is a ‘slot’ of the ‘dynamic segment’ of the ‘communication cycle’ used to transmit or receive a ‘dynamic frame’. It is divided in a number of ‘minislots’.

q **ECU (electronic control unit)**
   a processor (or a group of processors) to which one or more ‘communication modules’ are deployed, i.e. the hardware executing the functionality of the ‘communication modules’. An ‘ECU’ can also contain one or more ‘host’ applications (and other non-FlexRay related applications).

q **fault domains**
   are domains/regions in which failures could happen independently (i.e. without influencing other fault domains).

q **FES (frame end sequence)**
   is used to mark the end of the last byte sequence of a ‘frame’.

q **filter criteria**
   consists of ‘slot ID’, ‘channel ID’, ‘cycle counter’ (and for reception additionally an (optional) ‘message ID’ (cf. section 10.1.3.2) criteria used to filter output and input ‘payload data’ inside the ‘protocol module’.

q **FlexRay communication system (short: FlexRay)**
   consists of two or more ‘communication modules’, two or more ‘logical links’, optionally some ‘bus guardian modules’ and optionally some ‘repeater modules’.

q **following coldstart node**
   a ‘coldstart node’ that integrates upon a ‘leading coldstart node’.

q **frame (types of)**
   the basic information structure transmitted or received in one ‘slot’ between ‘communication modules’. ‘FlexRay’ distinguishes two types of frames: ‘static frames’ and ‘dynamic frames’. For the structure of frames cf. the definition of ‘frames (structure of)’.

q **frame (structure of)**
   consists of a ‘header segment’, a ‘payload segment’ and a ‘trailer segment’. For the types of frames cf. the definition of ‘frames (type of)’.

q **frame ID**
   defines the slot in which the frame is intended to be transmitted.

q **FSS (frame start sequence)**
   is used to truncate TSS and compensate a possible quantization error.

q **glitch**
   an event that changes the current condition of the ‘logical link’ such that its detected logic state is temporarily forced to a different value compared than what is being currently transmitted on the ‘logical link’ by the transmitting ‘node’ (e.g. spikes).

q **Global time**
   is the combination of cycle counter and cycle time. The global time of a cluster is the general common understanding of time inside the cluster. The FlexRay protocol does not have an absolute or reference global time; every node has its own local view of the global time.

q **halted**
   In this state recovery back to ‘normal active’ cannot be achieved, so execute normal communication cycle is halted in preparation for (re)configure communication module and (re)start communication module.
header segment
is the part of a ‘frame’ in which the control information for the ‘frame’ is contained, e.g. the ‘frame ID’.

host
is an application (outside the scope of FlexRay) using the ‘FlexRay communication system’ to communicate with other applications.

interrupt
signal generated by FlexRay to inform the host about errors

leading coldstart node
the first ‘coldstart node’ that is triggered by its ‘host’ to initially start communication.

local bit clock
defines the rate with which the voted signals are strobed.

logical link
is a logical connection between two or more ‘communication modules’ through which ‘communication elements’ are conveyed for the purpose of communication.

low power operating mode
a specific power operating mode in which the ‘communication module’ is not able to either transmit or receive data on the ‘logical link’ except for detecting the wakeup pattern (cf. section 9.2).

node
consists of one host, one protocol module, one or two bus drivers and optionally one or two bus guardians.

message buffer
stands for the idea of a storage mechanism to decouple the ‘host’ from the ‘communication module’. A message buffer does not only contain the payload but also information about the ‘logical link’, the ‘slot’ and the cycle in which they have been received or shall be transmitted.

message ID
if a ‘frame’ is tagged to contain a message ID, they are the first two bytes of the ‘payload segment’ of the ‘frame’ communicated in the ‘dynamic segment’.

minislot
a part of a ‘dynamic slot’. It has a constant nominal length.

minislot slot action point
is the ‘action point’ of a ‘minislot’.

network idle time (short: NIT)
a communication-free period at the end of the ‘communication cycle’ during which the ‘protocol module’ calculates and applies clock correction terms and performs implementation specific cluster cycle related tasks.

network management vector (short: NM Vector)
A number of bytes in the ‘payload segment’ of the FlexRay ‘frame’ format in a ‘frame’ transmitted in the ‘static segment’ can be used as network management vector.

non-coldstart node
a non-coldstart node requires at least two ‘startup frames’ from distinct ‘coldstart nodes’ for integration. This condition ensures that each ‘non-coldstart node’ always joins the majority of the ‘coldstart nodes’. (Note: This is the case under the assumption that the cluster contains exactly the three recommended ‘coldstart nodes’.)
q **normal active**
   This is the state in which the execute normal communication cycle is executed. The 'communication module' is either error free, or at least within error bounds that allow continued normal operation. Specifically, it is assumed that the 'communication module' remains adequately time-synchronized to the 'cluster' to allow continued transmission of 'communication elements' without disrupting the transmission of other 'communication modules'.

q **normal passive**
   In this state, synchronization with the remainder of the 'cluster' has degraded to the extent that continued transmission of 'communication elements' cannot be allowed because collisions with transmissions from other 'communication modules' are possible. 'Communication element' reception continues (as described in 9.1.1) in this state in support of host functionality and in an effort to regain sufficient synchronization to allow a transition back to 'normal active'.

q **not operational**
   is the state before communication between 'communication modules' can happen, i.e. the state where the 'ECU' to which a 'communication module' is deployed is still on low power or the 'communication module' is not yet configured. Only checking for wakeup patterns or configure communication module is possible in this state.

q **offset**
   is the difference between two corresponding points in time measured by means of a reference clock of an outside observer (e.g. in "real-time"). The offset is measured and expressed in microticks. If the clock of one communication module is at microtick 12 in cycle 3 and the clock of another communication module is at microtick 17 in cycle 3 at a corresponding (globally agreed) macrotick, their offset is 5 microticks.

q **offset correction**
   is a signed integer that indicates how many microticks the 'communication module' should shift the start of its cycle. Negative values mean that the 'network idle time' should be shortened (making the next cycle start earlier). Positive values mean the 'network idle time' should be lengthened (making the next cycle start later).

q **operational**
   contains 'ready', 'starting', 'working' and 'halted'.

q **payload segment**
   is the part of a 'frame' in which 'payload data' is transmitted or received.

q **payload preamble indicator**
   indicates whether (indicator set to 1) or not (indicator set to 0) a 'NM Vector' or 'message ID' is contained within the 'payload segment' of the transmitted 'frame'.

q **payload length**
   is an indicator for the size of the 'payload segment'.

q **point-to-point connection**
   a connection between two end points ('ECUs' or 'active stars') that is capable of transmitting 'data streams'. It can be electrical or optical.

q **precision** (of a FlexRay communication system)
   is the maximum difference in real-time(observed by an outside observer) between any two corresponding events in the 'FlexRay communication system'. (Corresponding events are e.g. cycle start, start of static slot, start of dynamic segment, ....)

q **rate correction**
   a signed integer indicating how many microticks the 'communication module's' cycle length should be changed. Negative values mean the cycle should be shortened; positive values mean the cycle should be lengthened.
**rate difference**
is the difference in the number of 'micro ticks' that a local clock of one 'communication module' needs for a 'communication cycle' compared to the local clock of another 'communication module'. The number of microticks is set during (local) configuration of a communication module and can be adjusted by the clock synchronization process.

**ready**
In this state the ‘FlexRay communication system’ waits for the startup command from the ‘host’.

**repeater module**
contains the optional routing and error handling functionality performed by ‘active stars’.

**requirement**
something a system must do or a property a system must have. A requirement shall always be testable by just testing the system.

**selected payload data**
are ‘payload data’ filtered in the ‘protocol module’ according to configurable ‘filter criteria’.

**semantic correctness**
a ‘communication element’ is semantically correct, if the elements of the ‘frame structure’ are used in accordance with the Use Cases of ‘FlexRay’.

**significant timing event**
is any event that indicates the start of a new ‘communication cycle’, any ‘communication cycle segment’ or any ‘slot’ in the segment.

**single domain fault**
a failure occurring in only one ‘fault domain’.  
List of single domain faults:
- failures of a single ‘host’
- failures of a single ‘communication module’
- failures of a single ‘logical link’
- failures of a single ‘repeater module’ in a two logical link system
- failures of a single ‘bus guardian module’.

**slot**
an interval of time during which access to the ‘logical link’ is granted exclusively to a specific ‘protocol module’ for the transmission of a ‘frame’ with a ‘frame ID’ corresponding to the ‘slot counter’. ‘FlexRay’ distinguishes between ‘static slots’ (in the ‘static segment’) and ‘dynamic slots’ (in the ‘dynamic segment’) of the ‘communication cycle’.

**slot counter**
a counter for the current ‘slot’.  
Note: The slot counter must be unique per channel per segment. The value of the slot counter can be identical for communication elements communicated in slots on different channels or in different segments.

**slot ID**
identifier of a specific ‘slot’ in which the ‘communication module’ may communicate.

**slot status**
is a structure consisting of the following information:
- ‘channel identifier’ of the receiving ‘logical link’,
- ‘slot ID’ of the ‘slot’ in which the ‘communication element’ was received;
- ‘cycle counter’ of the ‘communication cycle’,
- validity of ‘communication element’,
- type of ‘communication element’ (‘frame’ or ‘symbol’),
- type of segment in which the ‘communication element’ was received (unsynchronized startup phase, ‘static segment’, ‘dynamic segment’, ‘symbol window’, ‘NIT’).
q **star connection**
   is a ‘channel’ which consists of one or n ‘active stars’ and two up to n+1 ‘branches’.

q **starting**
   In this state ‘the FlexRay communication system’ tries to bring up all ‘communication modules’ to
   ‘normal active’ state. The process **start communication module** is executed.

q **startup frame**
   a ‘frame’ whose ‘header segment’ contains a ‘startup frame indicator’. Other ‘nodes’ may use the
   time related information from such a ‘frame’ to initialize their ‘communication cycle’. Startup
   frames are always also ‘sync frames’.

q **startup frame indicator**
   indicates whether or not a frame is a ‘startup frame’. ‘Startup frames’ serve a special role in the
   Use Case **start communication module**.

q **static frame**
   a ‘frame’ transmitted or received in a ‘static slot’ of the ‘static segment’ of the ‘communication
   cycle’.

q **static payload segment**
   is a ‘payload segment’ of a ‘frame’ in which a fixed length ‘payload data’ is transmitted or
   received in the ‘static segment’ of the ‘communication cycle’.

q **static segment**
   a part of the ‘communication cycle’ in which ‘static frames’ are transmitted or received. Within the
   static segment a static time division multiple access scheme is used to arbitrate the
   communication.

q **static slot**
   is a ‘slot’ of the ‘static segment’ of the ‘communication cycle’ used to transmit or receive a ‘frame’
   of fixed length.

q **static slot action point**
   is the ‘action point’ of a ‘static slot’.

q **strobed signals**
   definite high and low signals aligned to the local bit clock.

q **symbol**
   is a ‘communication element’ of ‘FlexRay’ used to transport control information between ‘protocol
   modules’. ‘FlexRay’ distinguishes between three ‘symbols’: CAS (Collision Avoidance Symbol,
   MTS (Media Access Test Symbol) and WUS (Wakeup Symbol).

q **symbol window**
   The symbol window is the communication cycle segment in which a symbol can be received or
   transmitted. This part contains the detailed requirements for the symbol window.

q **symbol window action point**
   is the ‘action point’ of a ‘symbol window’.

q **sync frame indicator**
   indicates whether or not the ‘frame’ is a ‘sync frame’, i.e. a ‘frame’ that is utilized for system wide
   synchronization of communication.

q **sync frame**
   a ‘frame’ whose ‘header segment’ contains a ‘sync frame indicator’. The deviation measured
   between such a ‘frames’ arrival time and its expected arrival time is used by the clock
   synchronization algorithm.
q **syntactic correctness**
a ‘communication element’ is syntactically correct, if its constituents are time aligned with the segments and ‘slots’ of the ‘communication cycle’.

q **tie line**
a direct coupling of an ‘ECU’ to an ‘active star’, that is capable of transmitting ‘binary data streams’.

q **tolerable failure**
In the context of this document a tolerable failure is a failure where communication between all nodes of a cluster is maintained, even when the particular failure is present. Note, this does not imply that communication were provided in case more than one tolerable failure is present.
List of tolerable failures:
- any failure of a single ‘communication module’ (resulting in transmission of any time and/or value domain fault)
- any failure of a single ‘logical link’ (resulting in:
  - valid ‘frame’ -> invalid ‘frame’ (for all receivers or for some)
  - valid ‘frame’ -> no ‘frame’ (for all receivers or for some)
  - generation of invalid ‘frame’ (for all receivers or for some)
  - (but not generate valid ‘frames’)
  - (but not change valid ‘frames’ into other valid ‘frames’)
  - (but not change invalid ‘frames’ into valid ‘frames’)
any failure of a single ‘repeater module’

**trailer CRC**
a cyclic redundancy check code computed over the ‘header segment’ and the ‘payload segment’ of the ‘frame’.

q **trailer segment**
is the last part of a ‘frame’ containing a single field, the 24-bit ‘trailer CRC’ for the ‘frame’.

q **transmission enable signal**
this signal is used by the ‘communication module’ to signal whether the transmitted ‘communication element’ from ‘protocol execution module’ is valid or not, i.e. shall be transmitted or not.

q **TSS (transmission start sequence)**
is used to initiate a proper connection setup through the ‘FlexRay communication system’. The purpose of the TSS is to “open the gates” of an ‘active star’, i.e., to cause the ‘active star’ to properly set up ‘branches’. During this set up, an ‘active star’ truncates a number of bits at the beginning of a ‘communication element’. The TSS also prevents the content of the ‘frame’ or ‘symbol’ from being truncated by the receiving ‘communication module’.

q **valid payload data**
data that was successfully received from the particular ‘logical link’ (receive message buffer) or that was successfully transferred from the ‘host’ to the ‘protocol module’ for transmission purpose (transmit message buffer)

q **valid sync frame**
every frame where the following conditions apply: a valid frame was received, the sync frame indicator of the received frame is set, and no content error was detected.

q **voted signal**
definite high or low signal used as input for **strobe voted signals**.

q **wakeup symbol (WUS)**
is a symbol to inform other ‘communication modules’ that they should wakeup.
It consists of a defined number of low-bits followed by a defined number of high bits.

q **wakeup pattern**
is a sequence of a configurable number of (at least two) consecutive ‘wakeup symbols’.
q  working
contains ‘normal active’ and ‘normal passive’.
### Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM</td>
<td>Channel Guardian Arm Signal</td>
</tr>
<tr>
<td>BSS</td>
<td>Byte Start Sequence</td>
</tr>
<tr>
<td>CAS</td>
<td>Collision Avoidance Symbol</td>
</tr>
<tr>
<td>CHI</td>
<td>Controller Host Interface</td>
</tr>
<tr>
<td>CM</td>
<td>Communication Module</td>
</tr>
<tr>
<td>CRC</td>
<td>Cyclic Redundancy Code</td>
</tr>
<tr>
<td>DTS</td>
<td>Dynamic Trailing Sequence</td>
</tr>
<tr>
<td>ECU</td>
<td>Electronic Control Unit.</td>
</tr>
<tr>
<td>FES</td>
<td>Frame End Sequence</td>
</tr>
<tr>
<td>FSS</td>
<td>Frame Start Sequence</td>
</tr>
<tr>
<td>ID</td>
<td>Identifier</td>
</tr>
<tr>
<td>MT</td>
<td>Macrotick</td>
</tr>
<tr>
<td>MTS</td>
<td>Media Access Test Symbol</td>
</tr>
<tr>
<td>NIT</td>
<td>Network Idle Time</td>
</tr>
<tr>
<td>NM</td>
<td>Network Management</td>
</tr>
<tr>
<td>RM</td>
<td>Repeater module</td>
</tr>
<tr>
<td>TDMA</td>
<td>Time Division Multiple Access (media access method)</td>
</tr>
<tr>
<td>TSS</td>
<td>Transmission Start Sequence</td>
</tr>
<tr>
<td>WUS</td>
<td>Wakeup Symbol</td>
</tr>
</tbody>
</table>
Appendix B UML Notation Guide

This UML Notation Guide covers five types of diagrams used for modelling the requirements of the FlexRay communication system. Modelling is the step of analysing and describing the logical dependencies before constructing a system. The diagrams used originate from the Unified Modelling Language (UML), an international standardized notation for systems specification, defined by the Object Management Group (OMG). The UML is a set of rules describing how to draw certain diagrams used in the system development process. Further information concerning the UML and the organisation OMG can be received on www.omg.org.

Creating these models for Flex Ray is an essential step in order to assure that the system provides all the functionality needed. Creating these models is not a one time job, they have to be adjusted until it is assured that they are complete. This process of creating and adjusting the models helps to discover new requirements as well as eliminating contradictions. As the models provide a visual representation of textual requirements, one can easily recognize distinct dependencies at a glance.

In the FlexRay Requirements Specification the following diagrams are used:

- Use Case diagrams,
- Class diagrams,
- Activity diagrams,
- State Transition diagrams and
- Component diagrams.

B.1 Use Case diagram

A Use Case describes a typical interaction between an external partner (e.g. a user of the system or an external system) and the system to be specified. So the content of a Use Case is the desired behavior of the system from an external point of view. Therefore Use Cases represent some very high level requirements the system has to fulfil. For the FlexRay communication system the typical user is a system (e.g. the host) communicating with FlexRay.

The Use Case diagram is used to find the fundamental processes and elements of a system and to structure the requirements in the Requirements Specification.

A Use Case diagram consists of two main elements: “actors” and “Use Cases” and shows the relationship between actors and Use Cases. The diagram allows to determine manageable parts of a system and. On the basis of these parts it is possible to go further into detail.

The Use Case diagram gives an overview of the functional behaviour of a system from an outside view. It is held very simple and does not contain any implementation specific jargon, so that it is easy to understand although it already defines requirements for the system being modelled.

It is the goal of a Use Case diagram to discover all processes on a high level of abstraction and thus it helps in the creation of a complete and well structured Requirements Specification.

B.1.4.1.1 Elements of a Use Case diagram

Each Use Case represents a part of the systems functionality. They are drawn as ellipses containing the name of the functionality. Any functionality described in a Use Case can in turn have more detailed sub-functionalities which may be described in more detail e.g. by Activity diagrams or prose requirements.
Exceptions of the normal behaviour of a Use Case are modelled as extending Use Cases and are connected to the Use Case containing the standard behavior with an «extend»-Association. This extending Use Case only contains the specific behavior that is executed in the special case (e.g. when a bus guardian module exists).

Actors perform certain roles which interact with the system. They are placed outside the system boundary. Actors do not have to be real persons, they can also represent sensors, input devices or other kinds of interfaces which are drawn for example as a box or a clock symbol.

The system boundary is shown as a rectangle which includes all Use Cases of the system. It defines the scope of the system and represents the whole system as defined in the problem statement.

Associations connect actors and Use Cases. They determine which actor interacts with an Use Case.

Figure 39 shows the Use Case diagram used in the FlexRay Requirements Specification.

![Use Case Diagram](image)

**Figure 39: Example of a Use Case Diagram**

### B.2 Activity diagram
An Activity diagram represents the operational workflows within a system or within parts of a system (e.g. within a Use Case) and can be modelled with a varying level of detail. As there are just a few elements it is easy to learn and quite understandable.

B.2.4.1.1 Elements of an activity diagram

An Activity diagram contains a starting point (initial activity) and an end point (final activity). The initial activity is the first activity that has to be executed.

Activities are represented by rectangles with rounded edges. The execution of an activity can start, when the execution of the previous activity has finished.

Data elements used within the flow of activities are shown as rectangles.

A diamond describes a decision which has to be made to show alternative flows. Only one of the alternative activities is executed. In the example, either “encode frame” or “encode symbol” is executed.

Parallel activities can also be modelled and are called concurrent activities. A splitting bar (a horizontal or vertical thick bar) shows that one incoming event is divided up into two or more outgoing events. The subsequent activities are executed in parallel. In the example, the activity “bus guarding” is executed in parallel to one of the activities “obtain payload data” and “assemble communication element”.

Another thick bar called the synchronization bar is used to synchronize all parallel flows. The synchronization bar shows, that the parallel activities are joined.

Swimlanes (areas limited by dotted lines) are used to distinguish between different regions of the system.

Figure 40 shows an Activity diagram containing all notation elements used in different Activity diagrams of the FlexRay Requirements Specification.
A Class diagram is a static and pictorial view of the logical system structure. It shows a collection of classes along with the relationships between them. It assembles all elementary modelling constructs of the UML.

B.3.4.1.1 Elements of a Class diagram

The main elements of a Class diagram are classes. A class describes a number of similar objects which arise when the system is executed. By this it forms a template for all objects that are members of the class.

A class is described by a rectangle divided into up to three parts. The upper part contains the class name, the middle part contains the attributes and data structures that the class needs to work with,
the lower part encapsulates the functionalities of a class, e.g. do some calculations or modify its attributes. (In FlexRay only the upper and middle parts are used.)

Classes may be associated through associations with other classes. This means an object of a class knows one or more objects of the other class so it can make use of the functionality of this objects. The multiplicity determines how many objects of the associated class are known of one object of the class that is the source of the relation. An association is a bidirectional relation: One has to consider both directions in the way described above.

In the figure below, we see that each object of the class “dynamic frame” has an association to one up to many minislots (because it is transmitted or received in minislots). Because of the other direction of this association, each minislots has an association to none or one dynamic frame.

The generalization relation (marked by a triangle) allows to specialize the classes. Using this kind of relation, the specialization of the upper class is defined. In the example, a symbol or a frame is a special kind of communication element.

The aggregation relation (marked by the diamond symbol) shows that an object of one class (the aggregation) exists of other objects (the parts), e.g. a communication cycle exists of one static segment, none or one dynamic segment, none or one symbol window and the network idle time.

Figure 41 shows one of the Class diagrams used in the FlexRay Requirements Specification.
B.4 State Transition diagram

State transition diagrams are used to describe complex systems in detail but still manageable, similar to an activity diagram describing a Use Case diagram in detail. One of the outstanding advantages of a state transition diagram lies in the possibility to split the behaviour of a system up into parts which become smaller and smaller. This simplifies development as well as testing, because the smaller the unit to be tested is, the less complex the input and output conditions are.

B.4.4.1.1 Elements of a State Transition diagram
The initial state is the starting point of the diagram. It is represented by a solid circle. The final state ends the State Transition diagram, and is shown by a bull’s eye symbol.

A state symbol represents the invariant state of a system (or a part of a system) at an instant of time. A state is denoted as a rectangle with rounded corners and compartments.

The compartments contain the name of the state and optionally a do-activity. As long as a certain state is upheld the do-activity is being performed.

A state change is described with an arrow pointing from the former to the succeeding state. This is called a transition and occurs if a trigger causes a state change. Additionally it is possible to perform an activity (a not divisible activity, also called action) when a transition occurs. This is modelled with a slash next to the transition arrow.

Sometimes it is necessary to choose between transitions to several possible following states depending on certain conditions. These conditions are called “guard conditions” and denoted in squared brackets in front of an activity.

Choices between two or more different following states are drawn as a diamonds. In the example below we can see a choice where it is determined if a certain button (in this case the power button) is pressed.

In a state diagram it is also possible to describe a state which needs to be described in more detail, using several new, so called sub states. States compiled out of sub states are called composite states. A composite state is entered either by a default entry (the transition points to the composite state), which uses the starting point to determine the first state, or by an explicit entry (the transition points directly to a sub state). A composite state is left if a trigger causes a transition which leaves the composite state. As seen in the example a transition which leaves a composite state can start at a distinct sub state or it can start at the composite state. The first trigger is only set off if the sub state to the corresponding transition is active, the second may be set off as long as the whole composite state is currently active.

Figure 42 shows one of the State Transition diagrams used in the FlexRay Requirements Specification.
B.5 Component diagram

A Component diagram illustrates the run-time organisation of different elements of the system - called components. The Component diagram is typically used to show the existing components, the interfaces they provide and need and the flow of information and events between them.

B.5.4.1.1 Elements of a Component diagram

A Component diagram uses notation elements of the class diagram.

The component is modelled by a class symbol with the stereotype component and optional with a component symbol in the upper right corner. Due to better legibility, we left out the stereotype notation in the Component diagrams. A component represents an application within the system and offers a functionality to other components. It encapsulates its inner life from other components.

The functionality of a component is offered to other components by interfaces. Interfaces are displayed with the ball- and socket-notation. The interface notations show provided and needed interfaces of the component. The provided interface is notated with a circle (called ball) and the needed using a half circle (called the socket). If it is not possible to define, which interface will be realized, a constraint can show that it is possible to choose between two or more interfaces. These
constraints are noted with a dashed line between the interfaces, including the constraint text (for example {XOR}).

Exchange of data or events between components is shown by arrows. The kind of data or event is displayed as text next to these arrows.

Figure 43 shows a part of a Component diagram used in the FlexRay Requirements Specification.

Figure 43: An example of a Component diagram