FlexRay Communications System

Preliminary Central Bus Guardian Specification

Version 2.0.9
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.

All details and mechanisms concerning the bus guardian concept are defined in the FlexRay Bus Guardian Specifications. The technical concept in this specification is new and not directly succeeding and not based on the “FlexRay Communications System Bus Guardian Specification Version 2.0”. The content of the present specification is not more mature than the previous version 2.0. The previous Bus Guardian Specification V2.0 is obsolete.

The FlexRay Communications System is currently specified for a baud rate of 10 Mbit/s. It may be extended to additional baud rates.

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.
# Table of Contents

## CHAPTER 1 INTRODUCTION ..................................................................................................................... 5

1.1 Scope.................................................................................................................................................... 5  
1.2 References ........................................................................................................................................... 6
   1.2.1 FlexRay consortium documents...................................................................................................... 6
1.3 Revision history..................................................................................................................................... 6
1.4 Terms and definitions ........................................................................................................................... 6
1.5 Acronyms and abbreviations ................................................................................................................ 6
1.6 Notational conventions ......................................................................................................................... 7
1.7 SDL conventions................................................................................................................................... 7

## CHAPTER 2 GENERAL PROPERTIES AND ARCHITECTURE ................................................................. 8

2.1 General properties of the CBG ............................................................................................................. 8
2.2 Architecture of the CBG...................................................................................................................... 10

## CHAPTER 3 DESCRIPTION OF THE CBG MODULES............................................................................ 11

3.1 I/O Array ............................................................................................................................................. 11
3.2 Bit Reshaping Unit .............................................................................................................................. 11
   3.2.1 Decoding Unit................................................................................................................................ 12
   3.2.2 FIFO Buffer ................................................................................................................................... 12
   3.2.3 Encoding Unit................................................................................................................................ 12
3.3 Clock Synchronization ........................................................................................................................ 12
3.4 BGBC unit ........................................................................................................................................... 13
3.5 BG Protocol Operation Control (BG POC) ......................................................................................... 13
3.6 Schedule and Configuration Data ....................................................................................................... 13
3.7 BG Interface ........................................................................................................................................ 13

## CHAPTER 4 CBG_POC STATES .............................................................................................................. 14

4.1 Sleep................................................................................................................................................... 15
4.2 Config ................................................................................................................................................. 15
   4.2.1 Notation of branch numbers.......................................................................................................... 15
4.3 Standby............................................................................................................................................... 16
4.4 Race Arbitration .................................................................................................................................. 16
4.5 Wakeup – no explicit state.................................................................................................................. 18
4.6 Startup ................................................................................................................................................ 18
4.7 Leading Coldstarter – no explicit state............................................................................................... 20
4.8 Consistency Check – Background Process. ....................................................................................... 20
4.9 Normal Operation ............................................................................................................................... 20
4.10 Macro PUNISH(n)............................................................................................................................. 21
4.11 Release Punishment............................................................................................................................. 21
CHAPTER 5 CBG STATE “NORMAL OPERATION” ............................................................... 22
   5.1 sub-state Strict ........................................................................................................... 22
   5.2 Sub-state Relaxed (race arbitration) .............................................................. 23
   5.3 Sub-state Dynamic (race arbitration) ......................................................... 23
   5.4 Sub-state Idle ....................................................................................................... 23

CHAPTER 6 WAKEUP ......................................................................................................... 25

CHAPTER 7 STARTUP .......................................................................................................... 26
   7.1 General Startup Description ....................................................................................... 26
   7.2 Detailed Timing of Assessment Windows ............................................................ 27
      7.2.1 Notation of time distances ................................................................................. 29
   7.3 Special problem scenario “different leading coldstart nodes on different channels” .... 29
   7.4 Special scenario “CBG powers up into running communication” ......................... 31
   7.5 Special scenario “early coldstart node” ............................................................ 32
   7.6 Special scenario “resetting leading coldstart node” ............................................. 33

CHAPTER 8 CLIQUE AVOIDANCE ........................................................................................ 34

CHAPTER 9 DETAILED DESCRIPTION OF ‘BIT RESHAPING WITH VARIABLE BSS-LENGTH’ .... 35

CHAPTER 10 TEST ............................................................................................................... 37
   10.1 Built-in Self Test (BIST) – Background Process .................................................... 37
   10.2 On-line Testing ......................................................................................................... 37

CHAPTER 11 CBG PARAMETERS .......................................................................................... 38
1.1 Scope

This document describes the Central Bus Guardian (CBG) for a FlexRay system. The CBG is a central device which forwards the communication elements received on its branches according to an algorithm described in this document - that way enabling data transfer on the channel of a FlexRay system which is connected to this CBG. Several critical faults (e.g. short circuited bus lines or babbling idiot behavior of a node) can be tolerated by a FlexRay system when CBGs are used to protect its channels.

The CBG is designed to have static schedule information stored. The complete communication schedule of the FlexRay system does not necessarily need to be stored in the CBG, but the CBG should have a sufficient amount of information to protect the frames which are relevant for critical functions and of course to start up and maintain communication as a whole.

Since it is assumed that a BG is only used in critical applications and these applications or functions are closed entities where the critical part of the communication is kept unchanged (even during major parts of the development phase), the need for re-configuration of the static schedule information in the CBG is expected to be limited to major redesigns during the development phase and to end-of-line programming. The critical parts of the schedule as well as the relation of nodes to critical functions (such that the CBG knows on which branch the sender of each critical message resides) have to be specified a priori and shall be frozen. However, if this cannot be done, re-configuration of a CBG is necessary. An example for a possible network architecture using two independent CBGs in a dual channel FlexRay system including critical applications can be seen in Figure 1-1.

![Figure 1-1 Example for a dual channel FlexRay network with two independent CBGs](image-url)
1.2 References

1.2.1 FlexRay consortium documents


1.3 Revision history

<table>
<thead>
<tr>
<th>Vers.</th>
<th>Date</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.0.9</td>
<td>15-December-2005</td>
<td>First public release.</td>
</tr>
</tbody>
</table>

1.4 Terms and definitions

See FlexRay Protocol Specification (reference [PS21A]).

1.5 Acronyms and abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BG</td>
<td>Bus Guardian</td>
</tr>
<tr>
<td>BIST</td>
<td>Built-In Self Test</td>
</tr>
<tr>
<td>BSS</td>
<td>Byte Start Sequence</td>
</tr>
<tr>
<td>CAS</td>
<td>Collision Avoidance Symbol</td>
</tr>
<tr>
<td>CBG</td>
<td>Central Bus Guardian</td>
</tr>
<tr>
<td>CC</td>
<td>Communication Controller</td>
</tr>
<tr>
<td>CE</td>
<td>Communication Element</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>NIT</td>
<td>Network Idle Time</td>
</tr>
<tr>
<td>µT</td>
<td>micro tick</td>
</tr>
<tr>
<td>POC</td>
<td>Protocol Operation Control</td>
</tr>
<tr>
<td>SDL</td>
<td>Specification and Description Language</td>
</tr>
<tr>
<td>SPI</td>
<td>Serial Peripheral Interface</td>
</tr>
<tr>
<td>SOS</td>
<td>Slightly Off Specification</td>
</tr>
<tr>
<td>TSS</td>
<td>Transmission Start Sequence</td>
</tr>
<tr>
<td>WUP</td>
<td>Wakeup Pattern</td>
</tr>
<tr>
<td>WUS</td>
<td>Wakeup Symbol</td>
</tr>
</tbody>
</table>
1.6 Notational conventions

The notational conventions used in this document are the same as in the Flexray Protocol Specification (see reference [PS21A]).

1.7 SDL conventions

The SDL conventions used in this document are the same as in the Flexray Protocol Specification (see reference [PS21A]).
2.1 General properties of the CBG

The CBG is an **optional device** that can be added to a channel of a FlexRay system in order to increase fault tolerance. Several critical faults (e.g. short circuited bus lines or babbling idiot behavior of a node) can be tolerated by a FlexRay system when CBGs are used to protect its channels.

**Scalability:** If a critical application is added to a cluster performing non-critical functions, the CBG can easily be added. The existing nodes (HW and SW) do not need to be changed if a CBG is installed (exception: possible modification of the channel-specific configuration parameters related to delay compensation). The schedule of the cluster communication is stored in the CBG in a reduced form (not containing all details). If only a subset of the nodes and messages are necessary to fulfill a critical function, only their schedule has to be stored in the CBG and only these slot/branch combinations will be protected. As a result, a change in a node performing non-critical functions will not lead to a change in the CBG, as long as no new critical applications are added to the cluster. Although it is not absolutely necessary to configure the CBG with information about which nodes send sync or startup frames, such information can allow a CBG to protect the clock synchronization and the startup more effectively.

**Basic features:**

- One CBG protects one channel.
- The CBG is able to decode frames.
- The CBG is not able to encode correct frames on its own. This means no algorithm is placed on the CBG to assemble new frames, and to generate a correct CRC.
- The CBG has its own clock synchronization unit basically identical to the clock sync unit in a communication controller (CC).
- The CBG can distinguish between different frame types (e.g. startup or sync frames) by decoding them.
- The CBG shall support a given number of branches (e.g. 16).
- It is also possible to have a subbus on a branch connecting several nodes which typically would perform non-critical applications. The number of nodes connected to a subbus is not limited by the CBG.
- The CBG’s architecture and the basic algorithms are independent of the available number of branches.
- It is possible to cascade CBGs on one channel (e.g. if an implementation of a CBG does not have a sufficient number of branches for a certain system).
- Depending on the filtering functionality that is placed in the CBG, the CBG guarantees that certain errors on one branch will not propagate to other branches. Examples of such filtering functions are: semantic filtering, Byzantine (SOS) filtering. Since the CBG is separated from the nodes (CCs) errors in the nodes will not harm the CBG.

The CBG can implement a **SOS containment mechanism** that ensures a consistent broadcast to all non-faulty receivers (a message is either accepted as correct by all its non-faulty receivers or by none). A frame (or symbol) that is identified as faulty by the CBG will be invalidated in such a way that all receivers will detect this frame (or symbol) as faulty. Using a two channel star topology an asymmetric reception is only possible in the presence of multiple faults. Assuming that EMI would only affect one branch an EMI-induced inconsistency would only affect the received frame on one channel.
The CBG can additionally prevent the formation of cliques on one channel during startup and normal operation by not-forwarding or invalidating frames which are identified as faulty (e.g. with wrong frame ID or cycle count). Cliques existing on different channels though, still have to be solved by the affected nodes.

The CBG can protect the startup of the FlexRay system in the presence of faults (e.g. can prevent the “endless startup scenario” caused by a resetting leading coldstart node) and can significantly enhance the fault tolerance of the FlexRay system especially during startup. This is performed by the CBG by limiting the bandwidth a certain node or branch can access.
2.2 Architecture of the CBG

Figure 2-1: Overview of the architecture of the CBG
Chapter 3
Description of the CBG Modules

3.1 I/O Array

All branches are connected to the CBG through the I/O array. An incoming frame is received by the I/O array and will be forwarded to the decoding unit within the bit reshaping unit. Before a frame is forwarded to the other branches, the signal is reshaped by the bit reshaping unit. Only one branch is able to receive a communication element at any point in time. If the CBG allows race arbitration (no schedule based protection of the slots) the first incoming frame will be delivered to all other branches. During this transmission no frame from another branch can be received.

3.2 Bit Reshaping Unit

The CBG will have a bit reshaping unit to ensure that, independent of the input signal quality, the output signal delivers the same decoding results for all receivers. If it is a correct frame all non-faulty receivers will receive a correct frame, if it’s incorrect all will see an incorrect frame. A non-faulty CBG therefore mitigates the SOS faults created by a faulty node. Under a single fault assumption the SOS effects are that way limited to one channel. The bit reshaping unit has no ability to generate valid new frames on its own but it has the capability to invalidate frames (or symbols) which are identified as faulty or almost faulty (e.g. a TSS of length 16 \( gdBit \)). The output stream of the bit reshaping unit is generated using the clock of the CBG. Note that due to possible clock deviations between the original sender and the receiving CBG the bit reshaping unit introduces an additional (but constant) propagation delay of up to 8 bits (depending on maximal frame length and maximal relative clock deviation). The propagation delay can be reduced (and the FIFO buffer minimized) by allowing the encoding unit to compensate the clock speed difference by slightly varying the length of the high-bit in the byte start sequences (BSSs) of the frame. This alternative of ‘bit reshaping with variable BSS-length’ is discussed in more detail in Chapter 9. The general operation of bit reshaping unit can be seen in Figure 3-1.

![Figure 3-1: Operation of the bit reshaping unit](image)

The functionality of this unit is split into 3 modules:
- Decoding Unit
- FIFO Buffer
3.2.1 Decoding Unit
The functionality of this unit is basically the same as the bit strobing and decoding processes in the CC. The CBG verifies the conformance of the incoming bit-stream to the coding rules and the general rules of communication elements. If the decoding process finds an error in the incoming bitstream it may stop the current relay process and, thus, truncate (or enlarge) the communication element to invalidate it. This filtering capability (which will be made configurable / can be switched off) is located in the bus guardian protocol operation control (BG POC). The main task of this module is to deliver the digital frame data and relevant timing and status information to the clock synchronization unit and the BG POC. The status information contains the information on whether a symbol was received (CAS, WUS), whether a startup or a sync frame was received, and error information (e.g. SyntaxError, ContentError or additional checks like ‘unexpected cycle counter value’ etc.).

3.2.2 FIFO Buffer
The FIFO buffer is used to store a variable number of bits which are transferred from the decoding unit to the encoding unit. Since the clock of the sender and the clock of the CBG may run at slightly different speeds the number of bits stored within the FIFO buffer can increase or decrease during transmission of a frame. The size of the FIFO buffer typically has to be twice the size as would be required for the nominal delay of the bit reshaping unit due to the possible increase or decrease of stored bits during forwarding of a frame. The reshaping delay can be reduced (below 2 gdbit) using the bit reshaping mechanism with variable BSS-length described in Chapter 9.

3.2.3 Encoding Unit
The functionality of this unit is basically the same as the encoding process in the CC – it delivers the NRZ-encoded bitstream (or symbols) to the I/O-Array. Note that the bitstream can be overruled by the BG POC which can force the encoding unit to output an invalidated communication element. The CBG is not capable of generating valid frames on its own (i.e. it has no mechanism to generate frame or header CRC’s). In the case of a frame or symbol that is not invalidated, the encoding unit simply regenerates the bitstream that resulted from the individual bit decisions made by the decoding unit. This regeneration is based on the CBG’s local sampling clock. The encoding unit itself has no knowledge of the frame structure or encoding rules – it merely generates, on a bit-by-bit basis, the bit stream given to it by the FIFO buffer. The process is slightly more complicated if variable-length BSS’s are used; Refer to Chapter 9 for details.

3.3 Clock Synchronization
The CBG is synchronized to the global time. This is done in exactly the same way as it is done in the nodes. Since the CBG is only connected to one channel the clock synchronization is the same as for single channel non-sync nodes. Since the sync nodes in a dual-channel system are connected to both channels the offset and rate correction performed within the CBG (and any other single-channel node) can have a minor deviations from the offset and rate correction performed by dual-channel nodes. This effect is small and is kept bounded by the closed loop nature of the clock synchronization in the sync nodes, which takes into account timing information from both channels. As a result, the global time determined by a single-channel CBG does not deviate substantially from the global time of the overall system.

Errors of the clock synchronization process which are detected by the CBG are signaled to the BG POC.
3.4 BGBC unit

The bus guardian branch control (BGBC) unit is responsible for enforcing the correct schedule by opening the correct branches for transmission at the correct time. The BGBC unit supports the following BGBC modes:

- **LISTEN_TO_ALL** all branches are input, no branch is output (the CBG behaves as an inactive device);
  - status of activity of the branches is forwarded to the BG POC (where decisions on forwarding or filtering are made);
  - filtering is performed by ORing the inputs from the branches with \( \text{vInputMask} \)

- **FORWARD_FROM_BRANCH(n)** branch number n is input, the data is reshaped and forwarded to all other branches (which of course are outputs of the CBG);
  - note: if branch number n is idle, all branches will be idle

The variable \( \text{vBGBC\_Mode} \) is used to store the BGBC mode. \( \text{vBGBC\_Mode} \) is of type \text{T\_BGBC\_MODE} as defined in Definition 3-1.

```plaintext
newtype T\_BGBC\_MODE
    literals LISTEN_TO_ALL, FORWARD_FROM_BRANCH(n);
endnewtype;
```

**Definition 3-1: Formal definition of T\_BGBC\_MODE.**

The variable \( \text{vInputMask} \) is used to perform filtering of the input branches. This way the CBG is able to listen only to a subset of branches when performing race arbitration. This is useful during startup (listening only to coldstart nodes or excluding misbehaving nodes from startup (also see punishment)).

3.5 BG Protocol Operation Control (BG POC)

The bus guardian protocol operation control (BG POC) is the “brain” of the CBG. It handles the different states of the CBG (e.g. \text{CBG\_POC:startup}, \text{CBG\_POC:normal\_operation}, etc.) and the transitions between them. It also enforces the schedule by sending the appropriate commands to the BGBC unit. All detected errors are stored by the BG POC and are made available to external devices through the BG Interface.

3.6 Schedule and Configuration Data

This unit stores the communication schedule (at least the parts that need to be protected) and all necessary configuration data and makes it available to the BG POC module. The integrity of the schedule and configuration data itself is protected by a signature (e.g. a CRC) which is checked periodically. Mismatches between the data and the signature cause the CBG to transition into the Config state in which no communication elements are forwarded.

3.7 BG Interface

The BG Interface is an optional connection (implemented, for example, as an SPI link) that allows an external CPU to perform certain operations involving the CBG. The BG Interface provides the possibility for an external device to update the communication schedule and configuration data of the CBG. This interface also allows error messages, status information, and configuration data to be read out of the CBG. Note that the connection to a CPU is optional, but can be a useful way to perform configuration and extract diagnostic information from the CBG.
Chapter 4
CBG_POC States

The CBG can be in one of the following states:
1. CBG_POC:standby
2. CBG_POC:sleep
3. CBG_POC:config
4. CBG_POC:race_arbitration
5. CBG_POC:startup
6. CBG_POC:normal_operation

Figure 4-1 shows the structure of the CBG state machine and the possible transitions from one state to another.

Figure 4-1: CBG state diagram [CBG_POC]
The variable \( v_{CBG\_POC\_State} \) is used to indicate what state the CBG is in. \( v_{CBG\_POC\_State} \) is of type \( T_{CBGPOCState} \) as defined in Definition 4-1.

\[
\text{newtype } T_{CBGPOCState} \\
\quad \text{literals } CONFIG, \text{ RACE\_ARBITRATION, STARTUP, NORMAL\_OPERATION, STANDBY, SLEEP;}
\]

Definition 4-1: Formal definition of \( T_{CBGPOCState} \).

### 4.1 Sleep

In the \( CBG\_POC:sleep \) state the CBG has a minimized power consumption. A selection of transceivers (for the appropriate branches) are active in this state in order to power up the complete CBG after detecting a wakeup pattern (WUP). In this case (or after a local wakeup of the CBG) the \( CBG\_POC:config \) state is entered to initialize the CBG with the schedule and configuration data.

### 4.2 Config

In the \( CBG\_POC:config \) state the CBG is initialized. The configuration data is stored in the Schedule and Configuration Data unit. The variables used for excluding branches are initialized. If no errors occur during the built-in self test (BIST) the CBG transits into the \( CBG\_POC:race\_arbitration \) state, else it performs a re-configuration.

![Figure 4-2: Transitions from the \( CBG\_POC:config \) state [CBG_POC]](image)

#### 4.2.1 Notation of branch numbers

According to the design of the cluster a certain number of coldstart nodes (each one using a different slot for the startup frame) are present. This number is called \( g_{CBG\text{NumOfCSnodes}} \) here and for simplicity reasons is also used for numbering the branches of the CBG. It is assumed that branches 1 to \( g_{CBG\text{NumOfCSnodes}} \) are used for the coldstart nodes (in ascending order of the used slot

\[
dcl \text{vCBG\_POC\_State } T_{CBGPOCState}; \\
dcl \text{vBGBC\_Mode } T_{BGBC\_MODE}; \\
dcl \text{zWUS\_Cnt } \text{Integer}; \\
dcl \text{zFailedCounter(1..gCBG\text{NumOfCSnodes}) } \text{Integer\_array}; \\
dcl \text{vInputMask()} \text{Boolean\_array}; \\
dcl \text{vPunishMask()} \text{Boolean\_array};
\]
numbers). Branch number 1 is connected to the coldstart node with the lowest startup slot number, branch number 2 to the one with the next used startup slot number and so on.

### 4.3 Standby

When no communication is observed (all branches are idle) in the `CBG_POC:race_arbitration` or `CBG_POC:normal_operation` state for `gdCBGStandby` microticks the CBG_POC goes into the `CBG_POC:standby` state. Within this state the CBG performs no activity as long as no communication is observed. The configuration data stored in the CBG remains valid during `CBG_POC:standby`. As soon as communication resumes the CBG transits to the `CBG_POC:race_arbitration` state. If the CBG stays in `CBG_POC:standby` for longer than `gdCBGSleep` microticks the CBG shall go to the `CBG_POC:sleep` state in order to reduce power consumption.

```plaintext
Fig. 4-3: Transitions from the CBG_POC:standby state [CBG_POC]
```

### 4.4 Race Arbitration

In `CBG_POC:race_arbitration` state the CBG forwards any incoming communication element from the coldstart-node branches (marked with a ‘0’ in `gCBGCSMask`) according to the simple philosophy ‘first come – first served’ to all other branches (coldstart-node and non-coldstart-node branches). To ensure a consistent view of the sent communication elements certain bit sequences that can easily be differently interpreted by different receivers have to be avoided. This is performed by invalidating the data if necessary. To prevent a ‘babbling idiot’ from blocking all communication on the channel, the CBG limits the branches which are listened to to the branches of the startup nodes and also assesses the forwarded data and its timing according to a certain set of rules. Faulty nodes (actually faulty branches) are “punished” by excluding them from the forwarding mechanism for a specified time (`gdCBGPunish` microticks). For each assessment rule the consequence has to be specified (no
consequence, invalidate data, exclude branch for a specified time). The startup of a cluster - supported by a CBG in **CBG_POC:race_arbitration** state - is shown in Chapter 7 Startup.

![Diagram showing transitions from the CBG_POC:race_arbitration state](image)

**Figure 4-4: Transitions from the CBG_POC:race_arbitration state [CBG_POC]**
4.5 Wakeup – no explicit state

This is part of the CBG_POC:race_arbitration state. During this state a wakeup pattern sent by any node will be forwarded to all other nodes - it will be regenerated to ensure a consistent view of the communication element - that way enforcing a clean and fast wakeup. When a CAS symbol or a correct startup frame is received the CBG transits to the CBG_POC:startup state which can be handled within CBG_POC:race_arbitration state.

All errors are stored and made available via the BG Interface Link.

4.6 Startup

The CBG is in CBG_POC:startup state after the CBG has received either a CAS or a correct startup frame. During startup the CBG enforces a fast and correct startup of the nodes by choosing a leading coldstart branch and forwarding its frames so other startup nodes can join into communication. If the consistency check state shows a sufficient number of valid sync frames the CBG switches to CBG_POC:normal_operation state. More details are described in Chapter 7 Startup.

All errors are stored and made available via the BG Interface Link.
Figure 4-5: Transitions from the \texttt{CBG_POC:startup} state [CBG_POC]
4.7 Leading Coldstarter – no explicit state

This is part of the CBG_POC:startup state. Here the CBG chooses the leading coldstart branch and performs the assessments on the startup frames from this node. More details are described in Chapter 7 Startup.

All errors are stored and made available via the BG Interface Link.

4.8 Consistency Check – Background Process

The consistency check is continously performed as a background process. At the end of each cycle the number of valid sync frames received is checked. If at least \textit{gCBGMinNumValidSyncFrames} valid sync frames were received during both cycles of an even/odd cycle pair the CBG switches to the CBG_POC:normal_operation state at the end of the odd cycle. More details are described in the Chapter 7 Startup.

If the consistency check fails during CBG_POC:normal_operation state for more than \textit{gCBGMaxwoClkCorr} even/odd cycle pairs the CBG transits to CBG_POC:race_arbitration state.

All errors are stored and made available via the BG Interface Link.

4.9 Normal Operation

The CBG is working properly within its schedule without any errors. A more detailed description can be found in Chapter 5 CBG State “Normal Operation”. see Figure 5-1

All errors are stored and made available via the BG Interface Link.
4.10 Macro PUNISH(n)

This macro is used to punish a certain branch. If the according element (n) in the array \( z_{FailedCounter} \) exceeds the limit \( g_{CBGMaxTolFaults} \) (e.g. \( g_{CBGMaxTolFaults} = 2 \)) the according branch is completely cut off from being forwarded for a certain time (\( gd_{CBGPunish} \) microticks) (e.g. 15 cycles). At any point in time not more than one branch can be punished (only the according bit for this branch is set in the variable \( v_{PunishMask} \)).

![Diagram of Punishment of a certain branch n](image)

4.11 Release Punishment

This process is needed to release the punishment after a certain time (\( gd_{CBGPunish} \) microticks) (e.g. 15 cycles). For this purpose this process has to be continuously active (in parallel to all states of the CBG_POC except for CBG_POC:sleep and CBG_POC:config).

![Diagram of Release Punishment control](image)
Chapter 5
CBG State “Normal Operation”

During normal operation the CBG enforces certain temporal aspects of the communication schedule. It does this via the use of several sub-states, each of which enforces different characteristics of the communication. During the static segment the CBG operates in either the strict or relaxed sub-states, and may switch between these sub-states depending on the stored configuration information for different static slots. In the dynamic segment the CBG operates in the dynamic sub-state, and during other portions of the cycle it operates in the idle sub-state. These sub-states are described in more detail in the following sections.

Figure 5-1 shows the sub-state transitions within the state CBG_POC: normal_operation during one cycle.

![Diagram of sub-state transitions](image)

**Figure 5-1: Process sequence of sub-states during one cycle in the state CBG_POC: normal_operation [CBG_POC]**

5.1 sub-state Strict

The CBG enforces a strict schedule within the static segment by allowing only one slot/branch combination to send a frame. All other branches are blocked for transmission. The CBG disables transmission, if a frame is sent outside its timeslot. The CBG uses the sync frames to synchronize to the global time base. The CBG shall open the channel only for the duration of a frame plus a safety margin before and at the end to handle clock deviations. The CBG also shall assess the forwarded data according to a certain set of rules (semantic filtering) and shall invalidate faulty frames. For each
assessment rule (which ones are necessary is still under discussion) the consequence of violating the rule must be specified (no consequence, invalidate data, exclude branch for a specified time).

It is possible to have several nodes on one branch building a subbus. In this case a faulty node on the subbus can affect all nodes on this subbus. It is strongly recommended not to place critical nodes (also startup nodes or sync nodes) on a subbus.

5.2 Sub-state Relaxed (race arbitration)

Within the static segment the CBG opens transmission at the beginning of each relaxed slot for all nodes. The first node/branch beginning to send in this slot is allowed to use the complete slot. Note that certain (faulty) branches may be excluded from the race arbitration by a filtering mechanism. The winning node’s data is forwarded to all branches during this slot. As soon as one active branch is detected all other branches are closed. This sub-state therefore actually represents a variant of the state CBG_POC:race_arbitration. The CBG enforces a correct timing which means nodes sending too late or too long will be cut off at the end of the slot.

Note that if a babbling idiot is present in the system it could potentially consume all slots which are assigned to the relaxed sub-state.

5.3 Sub-state Dynamic (race arbitration)

This mode is applied during the dynamic segment. All branches are allowed to transmit. The CBG does not enforce any schedule or any timing related restrictions. This sub-state is also represented by the state CBG_POC:race_arbitration but the timing constraints are slightly different here. The protection always ends at the end of the frame. A node sending too long will be cut off when exceeding the maximum allowed frame length or the end of the dynamic segment.

Note that if a babbling idiot is present in the system it could potentially consume all slots which are assigned to the dynamic sub-state.

5.4 Sub-state Idle

The CBG disables all communication – no data is forwarded. This sub-state is applied during NIT. The symbol window (if used) can either be treated in a similar way as a static slot and could be assigned to a certain node or also left open (race arbitration) or may also be assigned to the sub-state idle. The symbol window may be used to test the proper function of the CBG (see also Chapter 10 Test).
Figure 5-2: Transitions from the CBG_POC:normal_operation state [CBG_POC] (only sub-states 'strict' and 'relaxed' are shown)
Chapter 6
Wakeup

One or more nodes are woken up through a local wakeup event and behave as described in the FlexRay Protocol Specification [PS21A]. The CBG could also be woken up locally but typically is woken up by the WUPs sent by one of these nodes. In any case the CBG leaves the CBG_POC:sleep state and enters the CBG_POC:config state in which the CBG is initialized. The CBG then progresses to CBG_POC:race_arbitration state. If the CBG is awake but no communication is ongoing (in CBG_POC:standby state) a received WUP is forwarded to all branches immediately since the CBG transits to CBG_POC:race_arbitration state at the beginning of the communication element. If the CBG was in CBG_POC:sleep state the WUP is forwarded to all branches after the initialization of the CBG has been performed (in CBG_POC:config state) and the CBG has entered CBG_POC:race_arbitration state. The CBG shall introduce no significant delay to the wakeup of the cluster. Depending on the characteristics of the CBG implementation (duration of the CBG_POC:config state) the number of repetitions of the wakeup symbol sent by the nodes may need to be modified.

After wakeup the CBG remains in the CBG_POC:race_arbitration state and listens to all branches for ongoing communication.

If a CBG is reset (by a temporary disruption of the power supply, for example) the CBG shall wake up locally and shall enter the CBG_POC:config state and then progress to CBG_POC:race_arbitration state. It will monitor the channel to determine if communication is ongoing. If no activity is observed (all branches are idle) for a certain time (gdCBGStandby microticks) the CBG shall enter the CBG_POC:standby state. If no activity is observed for another gdCBGSleep microticks the CBG shall enter the CBG_POC:sleep state.
7.1 General Startup Description

Startup is performed by the nodes as defined in the FlexRay Protocol Specification [PS21A]. The CBGs will forward the first received CAS on all branches immediately (race arbitration). The CBG then looks for the 'leading coldstarter' and sets up a first assessment window for the first startup frame (timing derived from the known slot number and configuration data). During the assessment window the CBG gives exclusive access to the branch that transmits the related startup frame (which is also known from the configuration data). The first assessment is fulfilled if a valid startup frame is received in the assessment window. If no valid startup frame is received the next assessment window (belonging to the next highest startup slot number) is set up, again based on the configuration data. If a valid startup frame is received the associated branch number is stored in the variable $z_{LeadingColdstarterBranch}$ and the next assessment window is set up for the same slot number in the next cycle. The next assessments are fulfilled if a valid startup frame is received in the according time window with an incremented cycle counter. This continues for four cycles, with the exclusivity of the network access protecting the leading coldstart node. If an assessment is not fulfilled the associated element in the array $z_{FailedCounter}$ is incremented and the CBG returns to race arbitration. If an element in the array $z_{FailedCounter}$ exceeds the limit $g_{CBGMaxTolFaults}$ (e.g. $g_{CBGMaxTolFaults} = 2$) the according branch is classified as "bad" and it is completely cut off from being forwarded for a certain time ($gd_{CBGPunish}$ microticks) (e.g. 15 cycles) to give a different node a chance to become leading coldstart node and finish start up.

If the leading coldstart node fulfills all assessments for four cycles the CBG opens further timing windows, providing other coldstart nodes with network access in order to let them join in into startup.

![Figure 7-1: Example of a fault-free startup with two coldstart nodes and a CBG](image)
The exact clock speed of the leading coldstart node is adopted by the CBG (just like by the other coldstart nodes). If the ‘consistency check’ shows a sufficient number of valid sync frames the CBG switches to CBG_POC:normal_operation state and then enables the complete schedule at the beginning of the 7th cycle. If the coldstart attempt of the leading coldstart node was not successful (i.e., not enough valid startup frames were received in the 5th and 6th cycle) the element of failedCounter belonging to the leading coldstart node is incremented (at the end of the according cycle) and the CBG returns to race arbitration. If an element in the array failedCounter exceeds the limit gCBGMaxTolFaults the associated branch is completely cut off from being forwarded for a certain time (gdCBGPunish microticks). This guarantees that a different active coldstart node will become the leading coldstart node when the first leading coldstarter has exceeded the limit gCBGMaxTolFaults (by performing coldstart attempts). Note that this will also happen if the other coldstart nodes are still asleep so a “temporary communication cut off” can happen to a node even though it is not faulty.

Note that the CBG must know which branch is assigned to which slot for a sufficient set of slots (especially for startup frames) to perform startup-protection, but non-essential slots may be left unassigned and can be serviced by race arbitration.

While in CBG_POC:race_arbitration state the consistency check runs in the background. If a sufficient number of valid sync frames is detected the CBG switches to CBG_POC:normal_operation state with the appropriate schedule and keeps it active as long as the clock sync can be kept up (sufficient number of valid sync frames). If clock sync is lost for more than gCBGMaxwoClkCorr even/odd cycle pairs the CBG goes back to the CBG_POC:race_arbitration state.

As long as the CBG does not detect anything reasonable (looking like a startup) it remains in CBG_POC:race_arbitration state. The punishment mechanism is used to exclude a faulty branch (e.g. babbling idiot) from communication.

### 7.2 Detailed Timing of Assessment Windows

The timing of the assessment windows is not based on a cluster-wide synchronized clock but on the local clock of the CBG. The maximum clock speed deviation between the CBG and the nodes has to be taken into account so the tolerances used for the assessment windows are larger than for normal slot timing in the static segment. Figure 7-2 reflects a detailed view on the timing. Note that the tolerances for consecutive assessment windows do not accumulate since the timing is always derived from a trigger event typically lying within the previous assessment window thus adapting the precise position of the assessment windows to the sender’s clock speed.
Figure 7-2: Detailed timing of assessment windows

a. End of correct startup frame (or end of CAS).
b. Begin of assessment window \( gdCBGt\_to\_next(n) \) after a.
c. End of correct startup frame within assessment window (trigger for next assessment window).
d. End of assessment window \( gdCBGAssessmentWindow \) after b.

Note that the tolerances of the assessment windows may be larger than the gaps between static slots and therefore startup frames should not be placed in subsequent slots.
7.2.1 Notation of time distances

The time distances relevant for startup have to be available from the configuration data. The basic timing is specified by the parameters \( gdCBGt_{to\_next}(0..gCBGNumOfCSnodes) \) and \( t_{\_window} \) where \( gdCBGt_{to\_next}(0) \) is used to specify the distance between the CAS and the first startup slot assessment window, \( gdCBGt_{to\_next}(1) \) is the distance between the first and second startup slot assessment window, and so on. \( gdCBGt_{to\_next}(gCBGNumOfCSnodes) \) is the distance between the last startup slot assessment window in the cycle and the first startup slot assessment window of the next cycle.

7.3 Special problem scenario “different leading coldstart nodes on different channels”

If two (or more) nodes are sending the CAS at nearly the same time and are noticed differently by the CBGs of the two channels, the situation will be resolved in the following manner:

The CBG on channel A sees node X but CBG on channel B sees node Y as the first node sending a CAS (this is possible, for example, due to different physical delays in the signal path). Since the CAS is designed to allow the superposition of several CAS symbols at the same time the special case can occur that both nodes X and Y would start sending startup frames.

This does not really happen, however, since the CAS symbol synchronizes the two CBGs quite closely and they will both forward the startup frame of node X (which has a lower slot number than the startup frame of node Y). The startup frame of node X will force node Y into the ABORT_STARTUP state - even before it has sent its first startup frame. Therefore the CBGs on both channels will select X as the leading coldstarter and will detect no startup frame from node Y in the fifth cycle (since node Y has not reached COLDSTART_JOIN yet). This will cause the consistency check of the CBG to fail, lead to race arbitration again and the element in \( zFailedCounter \) for the branch of node X will be incremented (unless a third coldstart node is active and joins into the leading coldstart node X in the fifth cycle). If only the two coldstart nodes X and Y are active a second coldstart attempt is needed for startup. This is the usual case for a CAS collision startup with two coldstart nodes. Node X starts earlier in the second coldstart attempt and will be joined by Y in cycle 10. So this special problem is solved. Note that a certain time distance between the slots used for startup is necessary so that in case of a CAS collision the startup frames do not also collide, and that \( gCBGMaxTolFaults \) should be larger than one so the leading coldstart node can perform two coldstart attempts in a row.
Figure 7-3: An example for the startup after a CAS-collision and different leading coldstart nodes on different channels
7.4 Special scenario “CBG powers up into running communication”

If the CBG powers up (e.g. after a temporary interruption if its power supply), the CBG runs through the `CBG_POC:config` state and then transits into `CBG_POC:race_arbitration`. The CBG monitors the coldstart-node branches and forwards all communication elements received. When the CBG detects a correct startup frame it transits to the ‘leading coldstarter’ state (within startup state) and sets up a first assessment window for the following startup frame (expected in the next cycle from the same node). Note that this behavior is quite the same as in a startup situation beginning with a CAS - only that in this case the very first correct startup frame does not have an assessment window but is the trigger for the next one. The according branch number can be stored into the variable `zLeadingColdstarterBranch` right after the trigger and the next assessment window can be set up for the same branch. After the trigger by the first correct startup frame (or a CAS) no data is forwarded until the assessment window is reached. No data is forwarded between the assessment windows. This would normally continue for four cycles and then the CBG would let other coldstart nodes join in into startup. Since the ‘consistency check’ is already active in the background the CBG switches to `CBG_POC:normal_operation` state as soon as a sufficient number of valid sync frames is detected in an even/odd pair of cycles. Therefore the complete schedule can be enabled earlier than in a normal coldstart situation. If the switchover to `CBG_POC:normal_operation` does not occur within the first six cycles the according element belonging to the leading coldstart node in the array `zFailedCounter` is incremented (at the end of the 5th and of the 6th cycle) and the CBG remains in `CBG_POC:race_arbitration`. If an element in the array `zFailedCounter` exceeds the limit `gCBGMaxTolFaults` the associated branch is completely cut off from being forwarded for a certain time (`gdCBGPunish` microticks). That way it is guaranteed that a different active node sending startup frames will be used for startup synchronization. If no activity is observed (all branches are idle) for a certain time (`gdCBGStandby` microticks) the CBG shall go to `CBG_POC:standby` state.

Figure 7-4: An example for a CBG powering up into running communication (all nodes are assumed to be in ‘normal active’ state) (m=even)

Figure 7-5: An example for a CBG powering up into disturbed communication (m=even)
7.5 Special scenario “early coldstart node”

If one node powers up several cycles earlier than the other nodes the CBG will forward the CAS and the coldstart frames from this node during the first \( g_{CBG\text{MaxTolFaults}} \) coldstart attempts. The early coldstart node will then be punished (and the following frames sent from this node will not be forwarded by the CBG) for a certain time (\( g_{CBGPunish} \) microticks) since no other coldstart node joined its coldstart attempt. The early coldstart node may (according to its configuration of \( g_{ColdstartAttempts} \)) continue to perform coldstart attempts during the punishment phase, but the frames are not forwarded by the CBG. It is expected that the host of the early coldstart node will reassign a new amount of coldstart attempts to the CC so a new set of coldstart attempts can take place after the punishment has ended. If other nodes have become active in the mean time they can initiate a coldstart themselves or (as displayed in Figure 7-6) join in to a new coldstart attempt of the early coldstart node.

---

**Figure 7-6: An example of a startup with an early coldstart node**

(node Z is active before the other nodes, \( g_{CBG\text{MaxTolFaults}} = 2 \), \( g_{ColdstartAttempts} = 4 \))
7.6 Special scenario “resetting leading coldstart node”

If one node (the leading coldstart node) performs a reset during coldstart collision resolution (due to a hardware or software fault) this node could repeatedly force the other nodes to ABORT_STARTUP and create an endless startup scenario. The CBG assesses the leading coldstart node and therefore punishes this node after \( gCBGMaxTolFaults \) coldstart attempts. During the punishment phase the frames and symbols of the resetting leading coldstart node are not forwarded by the CBG. This opens the chances for other (non-faulty) coldstart nodes of the cluster to become the leading coldstart node and to perform startup - that way allowing all nodes (or at least the non-faulty ones) to reach the normal active state (as shown in Figure 7-7).

![Diagram](image)

**Figure 7-7:** An example for a CBG filtering a resetting leading coldstart node (node X)
Chapter 8
Clique Avoidance

Since the CBG only opens the channel selectively for one node at a time according to a defined schedule, simple cliques cannot exist on a single channel. Logical cliques (resulting from more complex failures) could exist on one channel if they are not detected by the CBG (e.g. cliquing along different notions of cycle counter, slot id, etc.). Frames capable of causing such cliques can be detected and invalidated by the CBG – that way preventing such cliques. At this state of the document the invalidation capability of the CBG is only mentioned in principle and not described in detail yet.

Cliques could still exist on the different channels, and such cliques may require a clique resolution or prevention algorithm performed by the nodes since the CBGs have no information from the other channel.
Chapter 9

Detailed Description of ‘bit reshaping with variable BSS-length’

The ‘bit reshaping with variable BSS-length’ is a bit reshaping scheme optimized for short delay between the incoming and the outgoing bit stream of the CBG. The CBG will have its own decoder and encoder based on its own clock and has to cope with possible clock speed deviations of up to +/-1500 ppm. On the other hand the CBG shall forward frames and symbols with minimal delay where the delay consists of three parts:

\[ \text{transceiver\_receive\_delay} + \text{CBG\_digital\_logic\_delay} + \text{transceiver\_transmit\_delay} \]

When using ‘bit reshaping with variable BSS-length’ the signal going out of the CBG will not be exactly the same as from a normal CC as displayed (in an exaggerated way) in Figure 9-1. Note that a selection of edges in the frame all have a certain ‘standard delay’ (e.g. 2 \( gdBit \)) between received frame (RxD) and sent frame (TxD) while other edges in the frame will have slightly different delays depending on the clock speed difference between transmitting node and CBG.

The design of the CBG should guarantee a maximum delay of \( \text{CBG\_digital\_logic\_delay} < 2 \text{\( gdBit \)} \).

It is expected that the ‘bit reshaping with variable BSS-length’ can be used without a reduction of the decoding performance since the following high to low transition will be used as a bit synchronization edge by all receivers. This is still under investigation of the ‘time budget task force’ at the moment. Altering the length of the high-bit of the BSS by [-1, 0 or +1] samples (\( c\text{SamplesPerBit} = 8 \)) would be sufficient for compensating clock tolerances even up to +/-6000 ppm.

As a criteria on how a BSS bit is encoded depends on how many bits are currently stored in the FIFO. If there are more bits stored than the standard delay (sender is faster than CBG) the BSS will be encoded using 7 samples to catch up with the sender, if there are less bits stored in the FIFO than the standard delay the BSS will be encoded using 9 samples, otherwise 8 samples are used.

Note that the CBG will contain a FlexRay conformant decoder and therefore the physical layer influence on the Tx/Rx asymmetry will only be two transceivers and the wires between them for any transmitted signal - even in a cascaded CBG scenario.

The FlexRay Protocol Specification 2.1A requests a ‘\( gdBit \)-quantization’ for all communication elements - see chapter 3.1 of [PS21A]: "This means that the generated bit level is either LOW or HIGH during the entire bit time \( gdBit \).” As mentioned above this will not be precisely fulfilled by the outgoing bit stream of the CBG when ‘bit reshaping with variable BSS-length’ is enabled.
The 'bit reshaping with variable BSS-length' distinguishes between 3 categories of bits:

a.) exact "gdBit-quantization":
- all data bits, FSS, FES, low bits of all BSSs, high bit of first BSS, high bit of DTS

b.) +/-1 sample deviation from "gdBit-quantization":
- high bits of all BSS’s (except for the high bit of the first BSS)
  Note#1: The first 2 high bits of a frame shall always have exact "gdBit-quantization".
  Note#2: Goal is to keep the time distances (number of samples from one bit synchronization edge to the next) the same for the outgoing bit stream as it was for the for the incoming bit stream.

c.) "non-gdBit-quantized"
- TSS (always low bits), low bits of DTS, WUS, CAS(MTS)
  Note#3: These phases are not gdBit-quantized (= exact multiples of 8 samples) but the relevant edges are forwarded with the same timing distance as in the incoming bit stream since the 'standard delay' is used. (see Figure 9-2)
Chapter 10
Test

Testing the functionality of the CBG (especially its filtering capability) can be done in two different ways: ‘on-line testing’ and ‘built-in self test’. Both methods can also be combined.

10.1 Built-in Self Test (BIST) – Background Process

With the built-in self test the CBG constantly checks its internal circuitry and also periodically checks the validity of the schedule and configuration data (e.g. by evaluating a signature or CRC of the data). A mismatch between the data and the signature or CRC causes the CBG to transition into the `CBG_POC:config` state in which no communication elements are forwarded. The built-in self test is performed in all states except for the `CBG_POC:sleep` state.

![Diagram](image)

**Figure 10-1**: Background process PERFORM_BIST

10.2 On-line Testing

On-line testing of the CBG is performed by the nodes. It is executed during run-time of the system in communication idle times. The main goal of the on-line test is to check the filtering capability of the CBG (e.g. if the system can tolerate a babbling idiot fault). During some pre-defined communication idle time (e.g. the symbol window; other possibilities are also under consideration) a dedicated node, or several dedicated nodes are configured to transmit. The CBG shall block this communication according to its normal function. If the non-transmitting nodes receive something during this phase (and sporadic disturbances due to EMI can be excluded) the nodes have identified a faulty CBG and may need to react according to a system specific strategy. If the non-transmitting nodes do not receive anything during this phase we know that the CBG is executing a form of filtering mechanism.
## Chapter 11
### CBG Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Unit</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>gCBGNumOfCSnodes</td>
<td>-</td>
<td>number of coldstart nodes in the cluster</td>
</tr>
<tr>
<td>gCBGMaxTolFaults</td>
<td>-</td>
<td>maximum number of tolerated faults on any branch before punishment</td>
</tr>
<tr>
<td>gCBGMinNumStartupFrames</td>
<td>-</td>
<td>minimum number of needed startup frames (per cycle) during startup to avoid punishment</td>
</tr>
<tr>
<td>gCBGMinNumValidSyncFrames</td>
<td>-</td>
<td>minimum number of valid sync frames per cycle needed for transition into CBG_POC: normal operation state</td>
</tr>
<tr>
<td>gCBGToIWUS</td>
<td>-</td>
<td>maximum number of tolerated Wakeup symbols (WUS) before punishment</td>
</tr>
<tr>
<td>gCBGMaxwoClkCorr</td>
<td>-</td>
<td>maximum number of tolerated even/odd cycle pairs without clock sync</td>
</tr>
<tr>
<td>gCBGCSMask</td>
<td>-</td>
<td>bit mask for filtering branches with coldstart nodes (branches with coldstart nodes are marked with '0', others =&gt; '1')</td>
</tr>
<tr>
<td>gdCBGStandby</td>
<td>µT</td>
<td>communication timeout for transition to CBG_POC: standby state</td>
</tr>
<tr>
<td>gdCBGSleep</td>
<td>µT</td>
<td>communication timeout for transition from CBG_POC: standby state to CBG_POC: sleep state</td>
</tr>
<tr>
<td>gdCBGPunish</td>
<td>µT</td>
<td>duration of the punishment for a faulty branch</td>
</tr>
<tr>
<td>gdCBG_to_next([0.. gCBGNumOfCSnodes])</td>
<td>µT</td>
<td>array of time distances between assessment windows</td>
</tr>
<tr>
<td>gdCBGAssessmentWindow</td>
<td>µT</td>
<td>duration of the assessment window</td>
</tr>
<tr>
<td>gdStaticSlot</td>
<td>MT</td>
<td>duration of a static slot</td>
</tr>
<tr>
<td>gColdStartAttempts</td>
<td>-</td>
<td>Maximum number of times a node in the cluster is permitted to attempt to start the cluster by initiating schedule synchronization</td>
</tr>
</tbody>
</table>

* This parameter is also used in the FlexRay Protocol Specification [PS21A].