E-Ray

FlexRay IP Module

Application Note AN001

Wakeup

Date: December 3rd, 2007

for IP Revision 1.0.2

Robert Bosch GmbH
Automotive Electronics
Semiconductors and Integrated Circuits
Digital CMOS Design Group

Copyright Notice

Disclaimer

1. About this Document

1.1 Change Control

1.1.1 Current Status

Version 1.0.4

1.1.2 Change History

Issue	Date	By	Change
Version 1.0.0	03.05.2006	Kay Hammer	Initial Draft for IP Revision 1.0 RC 1
Version 1.0.1	19.05.2006	Kay Hammer	Update for IP Revision 1.0
Version 1.0.2	03.11.2006	Kay Hammer	Correct wording in chapter 4.3.3
Version 1.0.3	01.06.2007	Kay Hammer	Update for IP Revision 1.0.1
Version 1.0.4	03.12.2007	Kay Hammer	Update for IP Revision 1.0.2

1.2 References

This document refers to the following documents:

Ref	Author(s)	Title
1	FlexRay Group	FlexRay Protocol Specification 2.1
2	Robert Bosch GmbH	E-Ray FlexRay IP-Module User’s Manual 1.2.6

1.3 Terms and Abbreviations

This document uses the following terms and abbreviations:

Term	Meaning
CHI	Controller Host Interface
CC	Communication Controller
WUP	WakeUp Pattern
WUS	WakeUp Symbol

2. Introduction

This application note describes the wakeup procedure of a FlexRay cluster, containing FlexRay E-Ray CC's.

In particular, the configuration steps necessary and the wakeup of the cluster channels are described.

The startup of the FlexRay cluster is not part of this application note.

The application note shall be used together with Ref. 1 and Ref. 2.

3. Version Control

This application note is based on the E-Ray FlexRay IP-module Revision 1.0.2. To verify the IP-module version, the Host can read out the Core Release Register (CREL). The register is read only. The correct value for the revision 1.0.2 is CREL = 0x10271031 (see Ref. 2).

4. Wakeup Procedure

4.1 Overview

In the following sections, the configuration schedule and the wakeup procedure are described.

The wakeup procedure is explained in detail in Ref. 1 and Ref. 2. For this application note, only the following states are important.

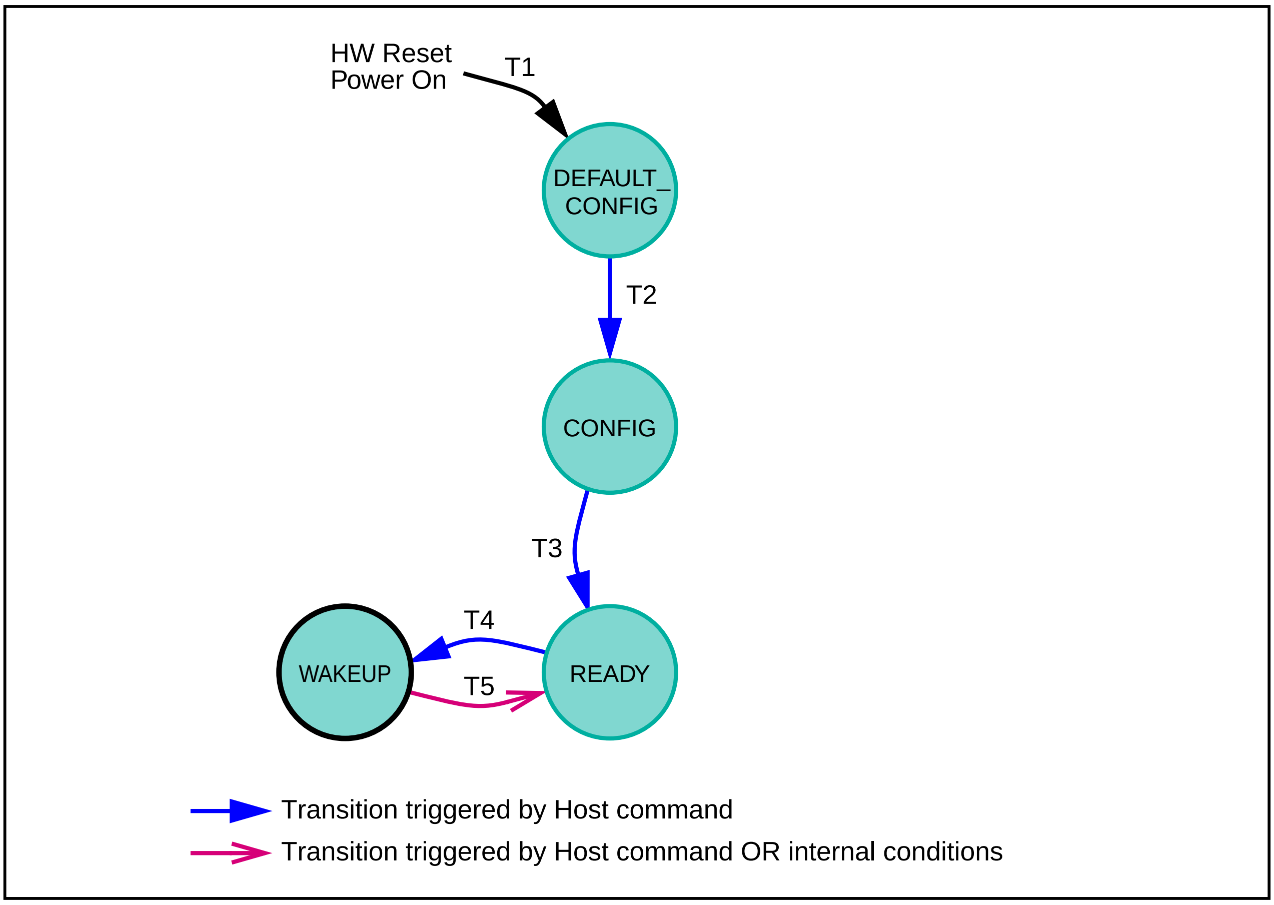
HW Reset
Power On
T1
DEFAULT_CONFIG
T2
CONFIG
T3
WAKEUP
T4
T5
READY
Transition triggered by Host command
Transition triggered by Host command OR internal conditions

Figure 1: Simplified state diagram (configuration and wakeup procedure) of E-Ray CC

T#	Condition	From	To
1	Hard reset	All States	DEFAULT_CONFIG
2	Command CONFIG, **SUCC1.CMD[3:0]** = "0001"	DEFAULT_CONFIG	CONFIG
3	Unlock sequence followed by command READY, **SUCC1.CMD[3:0]** = "0010"	CONFIG	READY
4	Command WAKEUP, **SUCC1.CMD[3:0]** = "0011"	READY	WAKEUP
5	Complete, non-aborted transmission of wakeup pattern OR received WUP OR received frame header OR wakeup collision OR command READY, **SUCC1.CMD[3:0]** = "0010"	WAKEUP	READY

Table 1: State transitions of E-Ray overall state machine (simplified)

4.2 Configuration Schedule

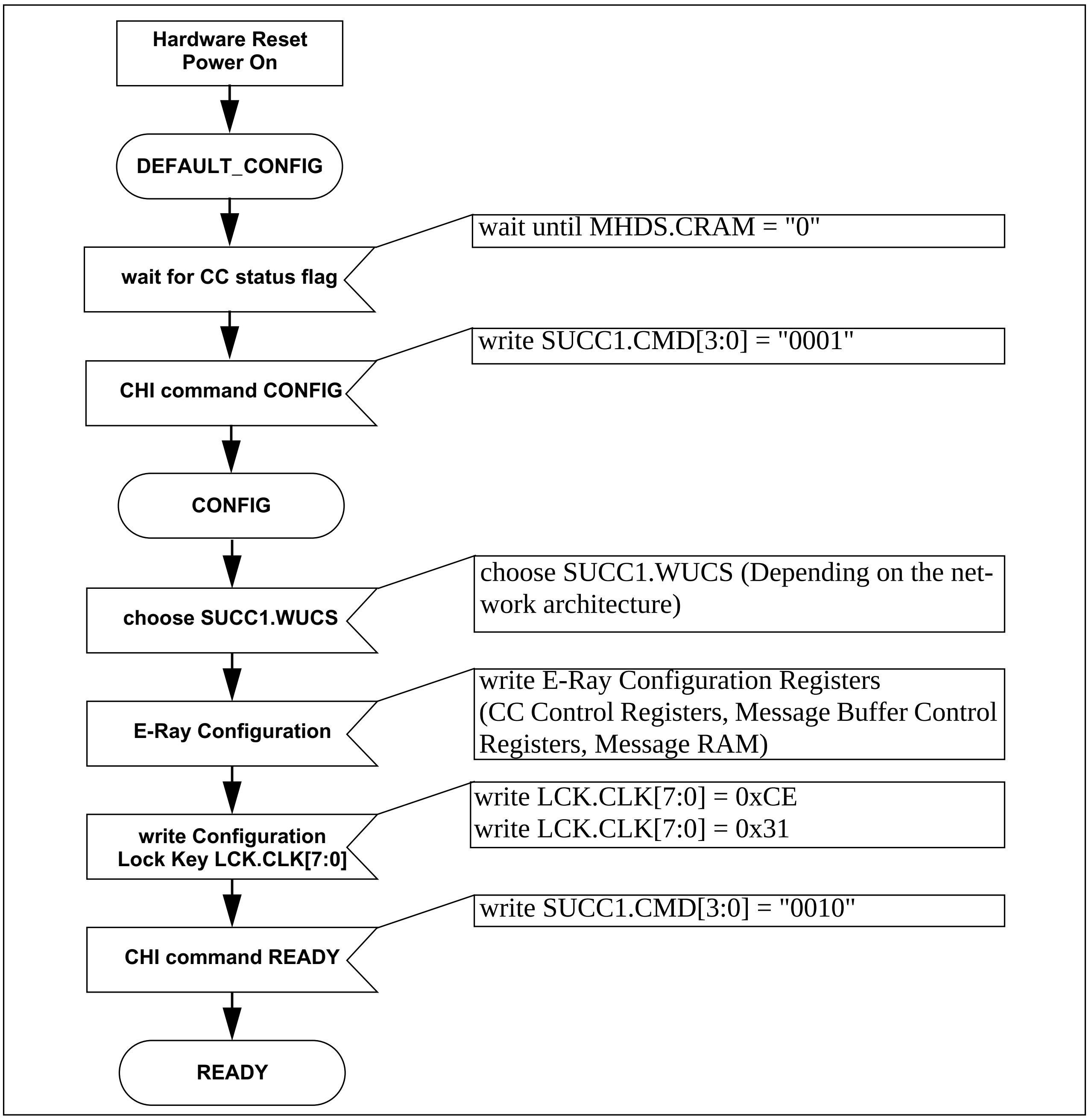
Hardware Reset Power On
DEFAULT_CONFIG
wait for CC status flag
wait until MHDS.CRAM = "0"
CHI command CONFIG
write SUCC1.CMD[3:0] = "0001"
CONFIG
choose SUCC1.WUCS
choose SUCC1.WUCS (Depending on the net-work architecture)
E-Ray Configuration
write E-Ray Configuration Registers (CC Control Registers, Message Buffer Control Registers, Message RAM)
write Configuration Lock Key LCK.CLK[7:0]
write LCK.CLK[7:0] = 0xCE
write LCK.CLK[7:0] = 0x31
CHI command READY
write SUCC1.CMD[3:0] = "0010"
READY

Figure 2: Flowchart configuration schedule for wakeup procedure (simplified)

4.3 CC Configuration

4.3.1 First Steps

The Message RAM is cleared (all bits are written to "0") after an external reset or by CHI command CLEAR_RAMS (**SUCC1.CMD[3:0]** = "1100"). During initialization of the Message RAM the flag **MHDS.CRAM** is set to "1" by the CC. The Host has to wait until the CC has reset **MHDS.CRAM** to "0" before the configuration of message buffers in the Message RAM is started.

After an external reset, the CC is in POC state DEFAULT_CONFIG (**CCSV.POCS[5:0]** = "00 0000"). It is recommend to enter POC state CONFIG with CHI command **SUCC1.CMD[3:0]** = "0001" before the configuration of the CC.

Note: POC state changes may be triggered by the Host through writing the CHI command vector to the SUC Configuration Register 1 (**SUCC1.CMD[3:0]**). The Host may write any CHI command at any time when POC Busy is LOW (**SUCC1.PBSY** = "0"), but certain commands are only enabled/allowed in certain POC states. The POC Busy Flag signals that the POC is busy and cannot accept a command from the Host. It is recommend that the Host verifies the CC is in the expected POC state and is not busy before writing a CHI command vector to the SUC Configuration Register. The actual state of operation of the CC Protocol Operation Control can be found in CC Status Vector Register (**CCSV.POCS[5:0]**).

Pseudocode Representation of Restarting a E-Ray Node:

```
// restart this Node
while (SUCC1.PBSY);
write SUCC1.CMD = FREEZE;

while (CCSV.POCS != HALT state)
// Controller is in POC state HALT

while (SUCC1.PBSY);
write SUCC1.CMD = CONFIG;

while (CCSV.POCS != DEFAULT_CONFIG state);
// Controller is in POC state DEFAULT_CONFIG
```

4.3.2 E-Ray Configuration

In POC state CONFIG (**CCSV.POCS[5:0]** = "00 1111"), the Host has to configure the Control Registers and the Message Buffer contents.

The FlexRay configuration parameters and calculation constraints are described in detail in Ref. 1 and Ref. 2. A description of the calculation of the configuration parameters is not part of this application note. An basic configuration can be found in chapter 4.4.

4.3.3 Wakeup Channel Select

Consider that a wakeup node may not be a coldstart node, or that a coldstart node may not be a wakeup node. In this application note, the wakeup nodes are designed to be coldstart nodes, too. Also the flowcharts show the wakeup procedure for nodes which are both wakeup- and coldstart nodes. Furthermore, in this application node the wakeup procedure for a dual-channel network is discussed.

Which channel is the node’s wakeup channel depends on the application and the network architecture.

The Host has to configure the wakeup channel while the CC is in POC state CONFIG by writing **SUCC1.WUCS**.

4.3.4 Leaving POC state CONFIG

If the Host wants the CC to leave POC state CONFIG, it has to proceed as described in Ref. 2, Section 4.3.3 Lock Register (LCK): To leave CONFIG state by writing **SUCC1.CMD[3:0]** = "0010" (CHI commands READY, MONITOR_MODE), the write operation has to be directly preceded by two consecutive write accesses to the Configuration Lock Key (unlock sequence). If the write sequence below is interrupted by other write accesses between the secound write to the Configuration Lock Key and the write access to the SUCC1 register, the CC remains in CONFIG state and the sequence has to be repeated.

First write:	**LCK.CLK[7:0]**	= "1100 1110" (0xCE)
Secound write:	**LCK.CLK[7:0]**	= "0011 0001" (0x31)
Third write:	**SUCC1.CMD[3:0]**	

The procedure is described in detail for different Host bus access widths in the following chapters.

The source code examples are sections of a Host controller application program. The CC is in POC state CONFIG and all E-Ray configuration registers excepting SUCC1 register are written. The source code example shows how to finalize the configuration process until POC state READY.

Note: In case that the Host uses 8/16-bit accesses to write the listed bit fields, the programmer has to ensure that no "dummy accesses" e.g. to the remaining register bytes/words are inserted by the compiler.

4.3.4.1 8-Bit Host Access

For 8-bit Host interfaces, the last write access must be **SUCC1[7:0]** with the CHI command vector value **SUCC1.CMD[3:0]** = "0010". The previous write accesses must be **LCK.CLK[7:0]** = 0xCE and **LCK.CLK[7:0]** = 0x31, respectively. Therefore, the register value **SUCC1[31:8]** must be written before **LCK.CLK[7:0]**. (The application specific configuration of the register may differ from this example):

```
/* The Communication Controller is in CONFIG state,
*  all E-Ray configuration registers excepting SUCC1 register are written.
*  To finalize the configuration process, the host controller writes
*  the unlock sequence for the Configuration Lock Key and afterwards
*  CHI command READY. */

/***************************************************************************
* Unlock sequence Configuration Lock Key                                   *
* 8 Bit Host Controller Interface                                          *
***************************************************************************/
/* write8bit(address, value); */

/* Write SUCC1[31:8] at first */
write8bit(0x0083, 0x0F);  /* write word 4 of SUCC1 register
                             CCHB = 1, CCHA = 1, MTSB = 1, MTSA = 1 */
write8bit(0x0082, 0x1F);  /* write word 3 of SUCC1 register
                             HCSE = 0, TSM = 0, WUCS = 0, PTA = 31  */
write8bit(0x0081, 0xF8);  /* write word 2 of SUCC1 register
                             CSA = 31, TXSY = 0, TXST = 0           */

/* Write Configuration Lock Key */
write8bit(0x001C, 0xCE);  /* First write: to LCK.CLK[7:0]   */
write8bit(0x001C, 0x31);  /* Second write: to LCK.CLK[7:0]  */

/* Command READY */
write8bit(0x0080, 0x02);  /* Third write: to SUCC1.CMD[3:0] */

/***************************************************************************
* Communication Controller state changes to READY                          *
***************************************************************************/
```

4.3.4.2 16-Bit Host Access

For 16-bit Host interfaces, the last write access must be **SUCC1[15:0]** with the CHI command vector value **SUCC1.CMD[3:0]** = "0010". The previous write accesses must be **LCK[15:0]** with the configuration lock key value **LCK.CLK[7:0]** = 0xCE and **LCK.CLK[7:0]** = 0x31, respectively. Therefore, the register value **SUCC1[31:16]** must be written before **LCK.CLK[7:0]**. (The application specific configuration of the register may differ from this example):

```
/* The Communication Controller is in CONFIG state,
*  all E-Ray configuration registers excepting SUCC1 register are written.
*  To finalize the configuration process, the host controller writes
*  the unlock sequence for the Configuration Lock Key and afterwards
*  CHI command READY. */

/***************************************************************************
* Unlock sequence Configuration Lock Key                                  *
* 16 Bit Host Controller Interface                                        *
***************************************************************************/
/* write16bit(address, value); */

/* Write SUCC1[31:17] at first */
write16bit(0x0082, 0x0F1F);  /* write word 3 and 4 of SUCC1 register
                               CCHB = 1, CCHA = 1, MTSB = 1, MTSA = 1
                               HCSE = 0, TSM = 0, WUCS = 0, PTA = 31  */

/* Write Configuration Lock Key */
write16bit(0x001C, 0x00CE);  /* First write: to LCK.CLK[7:0]   */
write16bit(0x001C, 0x0031);  /* Second write: to LCK.CLK[7:0]  */

/* Command READY */
write16bit(0x0080, 0xF802);  /* Third write: to SUCC1.CMD[3:0] and
                               word 2 of SUCC1 register
                               CSA = 31, TXSY = 0, TXST = 0 */

/***************************************************************************
* Communication Controller state changes to READY                         *
***************************************************************************/
```

4.3.4.3 32-Bit Host Access

For 32-bit Host interfaces, the last write access must be **SUCC1[31:0]** with the CHI command vector value **SUCC1.CMD[3:0]** = "0010". The previous write accesses must be **LCK[31:0]** with the configuration lock key value **LCK.CLK[7:0]** = 0xCE and **LCK.CLK[7:0]** = 0x31, respectively.
(The application specific configuration of the register may differ from this example):

```
//* The Communication Controller is in CONFIG state,
*  all E-Ray configuration registers excepting SUCC1 register are written.
*  To finalize the configuration process, the host controller writes
*  the unlock sequence for the Configuration Lock Key and afterwards
*  CHI command READY. */

/***************************************************************************
* Unlock sequence Configuration Lock Key                                   *
* 32 Bit Host Controller Interface                                         *
***************************************************************************/
/* write32bit(address, value); */

/* Write Configuration Lock Key */
write32bit(0x001C, 0x000000CE);  /* First write: to LCK.CLK[7:0]   */
write32bit(0x001C, 0x00000031);  /* Second write: to LCK.CLK[7:0]  */

/* Command READY */
write32bit(0x0080, 0x0F1FF802);  /* Third write: to SUCC1.CMD[3:0] and
                                    word 2 to 4 of SUCC1 register
                                    CCHB = 1, CCHA = 1, MTSB = 1, MTSA = 1
                                    HCSE = 0, TSM = 0, WUCS = 0, PTA = 31
                                    CSA = 31, TXSY = 0, TXST = 0 */

/***************************************************************************
* Communication Controller state changes to READY                          *
***************************************************************************/
```

4.3.5 Enter POC state WAKEUP

4.3.5.1 General

Depending whether the wakeup node is a coldstart node, a sync node, a single slot node or a normal node, the flags **SUCC1.TXSY** (Transmit Sync Frame in Key Slot), **SUCC1.TXST** (Transmit Startup Frame in Key Slot) and **SUCC1.TSM** (Transmission Slot Mode) must be configured accordingly.

If the wakeup node is a coldstart node, the flags must be configured with **SUCC1.TXSY** = "1" and **SUCC1.TXST** = "1".

If the wakeup node is a sync node, but no coldstart node, the flags must be configured with **SUCC1.TXSY** = "1" and **SUCC1.TXST** = "0".

If the wakeup node is a single slot node (the CC only transmits in the pre-configured key slot, see chapter 4.3.7.4), the flag **SUCC1.TSM** must be configured to "1". A single slot frame may be a sync or startup frame, too. **SUCC1.TXST** and **SUCC1.TXSY** have to be programmed accordingly.

If the wakeup node is a normal node, the flags must be configured by **SUCC1.TXSY** = "0", **SUCC1.TXST** = "0" and **SUCC1.TSM** = "0".

The configuration with **SUCC1.TXSY** = "0" and **SUCC1.TXST** = "1" is invalid.

4.3.5.2 Wakeup Procedure

The cluster wakeup must precede the communication startup in order to ensure that all nodes in a cluster are awake. The minimum requirement for a cluster wakeup is that all bus drivers are supplied with power. A bus driver has the ability to wake up the other components of its node when it receives a wakeup pattern on its channel. At least one node in the cluster needs an **external** wakeup source.

The Host completely controls the wakeup procedure. It is informed about the state of the cluster by the bus driver and the CC and configures bus guardian (if available) and CC to perform the cluster wakeup. The CC provides to the Host the ability to transmit a special wakeup pattern on each of its available channels separately. The Host has to configure the wakeup channel while the CC is in POC state CONFIG by writing **SUCC1.WUCS**. The CC needs to recognize the wakeup pattern only during POC state WAKEUP. A single CC should not wake up both channels, but this is not prohibited by Ref. 1 and may be necessary in an application.

Before the Host initiates an wakeup attempt, it should ensure that the FlexRay channels to which the node is connected are not active. Otherwise, an wakeup attempt could disturb ongoing communication.

Therefore, if all channels to which the node is connected are active, the Host shall command the CC to enter the startup state. If the node is a coldstart node, the Host resets the coldstart inhibit flag (**CCSV.CSI**) with the CHI command ALLOW_COLDSTART (**SUCC1.CMD[3:0]** = "1001").

If only the wakeup channel is active, but the other channel to which the node is connected is not, the Host may reconfigure the CC to wake up the channel which is assumed to be asleep. In that case, the Host enters POC state CONFIG with the CHI command CONFIG (**SUCC1.CMD[3:0]** = "0001"), configures the CC to wake up the missing channel (**SUCC1.WUCS**) and programs the CC to re-enter POC state READY and WAKEUP.

Alternatively, the Host commands the CC to enter POC state STARTUP with the CHI command RUN (**SUCC1.CMS[3:0]** = "0100"). If the node is a coldstart node, the Host must wait until the CC detects a wakeup pattern on the channel which is assumed to be asleep, before it resets the coldstart inhibit flag (**CCSV.CSI**) with the CHI command ALLOW_COLDSTART (**SUCC1.CMD[3:0]** = "1001").

If the wakeup channel is asleep, the Host shall awake it. If the CC is in POC state READY (**CCSV.POCS[5:0]** = "00 0001"), wakeup can entered by CHI command WAKEUP (**SUCC1.CMD[3:0]** = "0011").

The CC exits from the wakeup state after complete non-aborted transmission of wakeup pattern, after WUP reception, after detecting a WUP collision, after reception of a frame header or by writing CHI command READY (**SUCC1.CMD[3:0]** = "0010").

The CC cannot guarantee that all nodes connected to the configured channel awake upon the transmission of the wakeup pattern, since these nodes cannot give feedback until the startup phase. The wakeup procedure enables single-channel devices in a two-channel system to trigger the wakeup, by only transmitting the wakeup pattern on the single channel to which they are connected. Any coldstart node that deems a system startup necessary will then wake the remaining channel before initiating communication startup.

The wakeup procedure tolerates any number of nodes simultaneously trying to wakeup a single channel and resolves this situation such that only one node transmits the pattern. Additionally the wakeup pattern is collision resilient, so even in the presence of a fault causing two nodes to simultaneously transmit a wakeup pattern, the resulting collided signal can still wake the other nodes.

After wakeup the CC returns to READY state and signals the change of the wakeup status to the Host by setting flag **SIR.WST**. This flag is set when **CCSV.WSV[2:0]** changes to a value other than UNDEFINED. The wakeup status vector can be read from **CCSV.WSV[2:0]**. If a valid wakeup pattern was received also either flag **SIR.WUPA** or flag **SIR.WUPB** is set. **WSV[2:0]** is reset by CHI command RESET_STATUS_INDICATORS or by transition from HALT to DEFAULT_CONFIG state.

The Host has to be aware of possible failures of the wakeup and act accordingly. It is advisable to delay any potential startup attempt of the node having instigated the wakeup by the minimal time it takes another coldstart node to become awake and to be configured.

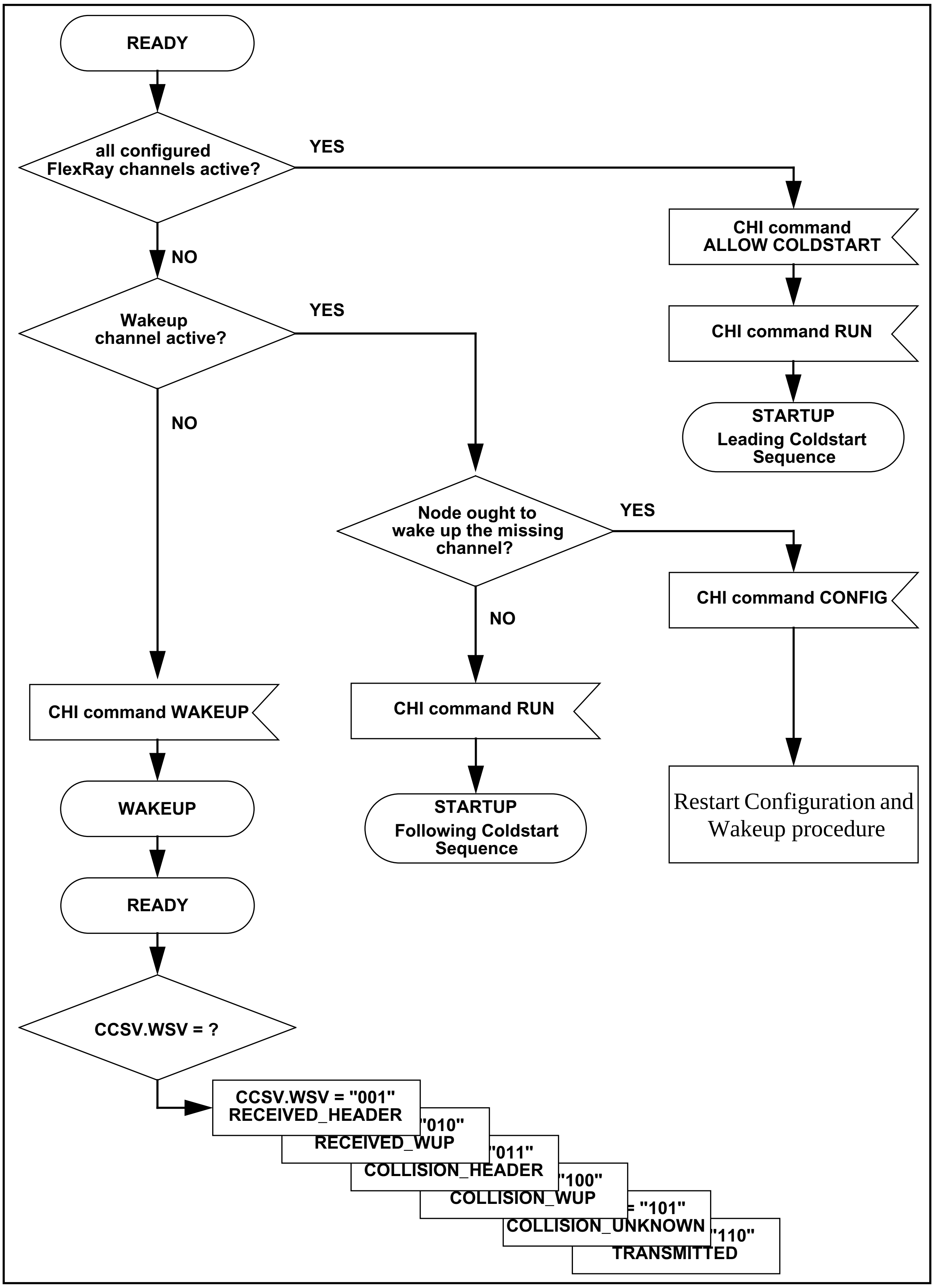
READY
all configured FlexRay channels active?
YES
NO
CHI command ALLOW COLDSTART
CHI command RUN
STARTUP Leading Coldstart Sequence
Wakeup channel active?
YES
NO
Node ought to wake up the missing channel?
YES
NO
CHI command CONFIG
CHI command WAKEUP
CHI command RUN
WAKEUP
STARTUP Following Coldstart Sequence
Restart Configuration and Wakeup procedure
READY
CCSV.WSV = ?
CCSV.WSV = "001" RECEIVED_HEADER
"010" RECEIVED_WUP
"011" COLLISION_HEADER
"100" COLLISION_WUP
= "101" COLLISION_UNKNOWN
"110" TRANSMITTED

Figure 3: Flowchart wakeup procedure (simplified)

4.3.6 Analyse Wakeup Status

4.3.6.1 RECEIVED_HEADER or COLLISION HEADER

In case the wakeup status signals RECEIVED_HEADER or COLLISION_HEADER, ongoing communication was detected by the CC on either available channel while in the POC state WAKEUP_LISTEN or POC state WAKEUP_DETECT, respectively. If the network is correctly configured, all channels must be awake due to the fact that other nodes of the cluster reached POC state STARTUP or NORMAL_ACTIVE, respectively.

The CC aborts the wakeup, even if the node's wakeup channel is still silent.

The Host shall not command the communication controller to initiate additional wakeup attempts, since this could disturb ongoing communication.

Instead, it assumes all channels to be awake and shall command the CC to enter POC state STARTUP.

If the node is a coldstart node, the Host resets the coldstart inhibit flag (**CCSV.CSI**) with the CHI command ALLOW_COLDSTART (**SUCC1.CMD[3:0]** = "1001") and enters POC state STARTUP with the CHI command RUN (**SUCC1.CMD[3:0]** = "0100") to integrate into the apparently established cluster communication.

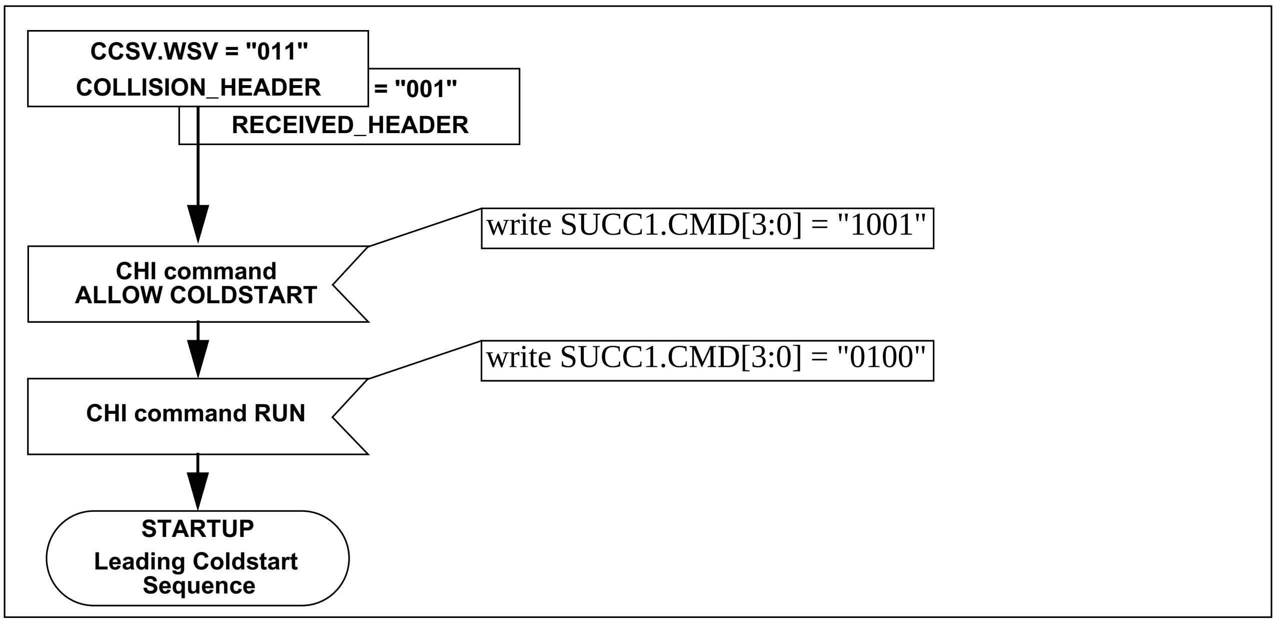
CCSV.WSV = "011"
COLLISION_HEADER
= "001"
RECEIVED_HEADER
write SUCC1.CMD[3:0] = "1001"
CHI command ALLOW COLDSTART
write SUCC1.CMD[3:0] = "0100"
CHI command RUN
STARTUP Leading Coldstart Sequence

Figure 4: Flowchart trading wakeup status RECEIVED_HEADER or COLLISION_HEADER

4.3.6.2 RECEIVED_WUP, COLLISION_WUP or TRANSMITTED

In case the wakeup status signals RECEIVED_WUP or COLLISION_WUP, the CC has received a wakeup pattern on the wakeup channel while in POC state WAKEUP_LISTEN or POC state WAKEUP_DETECT. This indicates that another node is already waking up this channel. To prevent collisions of wakeup patterns on the wakeup channel, the CC aborts the wakeup.

In case the wakeup status signals TRANSMITTED, the CC has transmitted the complete WUP on the configured wakeup channel.

In this cases, the Host shall assume that the wakeup channel is awake.

The Host shall wait the minimal time for another node to become awake, to be configured, to become ready and to enter wakeup, or rather startup, before proceeding. It cannot be assumed that all nodes need the same amount of time to become completely awake and to be configured.

If another channel is available that is not already awake, the Host must determine whether the communication controller is to wake up this channel.

If all channels to which the node is connected are active, the Host shall command the CC to enter the startup state. If the node is a coldstart node, the Host may reset the coldstart inhibit flag (**CCSV.CSI**) with the CHI command ALLOW_COLDSTART (**SUCC1.CMD[3:0]** = "1001").

If only the wakeup channel is active, but the other channel to which the node is connected is not, the Host may reconfigure the CC to wake up the channel which is assumed to be asleep. In that case, the Host enters POC state CONFIG with the CHI command CONFIG (**SUCC1.CMD[3:0]** = "0001"), configures the CC to wake up the missing channel (**SUCC1.WUCS**) and to re-enter POC state READY and WAKEUP.

Alternatively, the Host commands the CC to enter POC state STARTUP with the CHI command RUN (**SUCC1.CMS[3:0]** = "0100"). If the node is a coldstart node, the Host must wait until the CC detects a wakeup pattern on the channel which is assumed to be asleep, before it resets the coldstart inhibit flag (**CCSV.CSI**) with the CHI command ALLOW_COLDSTART (**SUCC1.CMD[3:0]** = "1001").

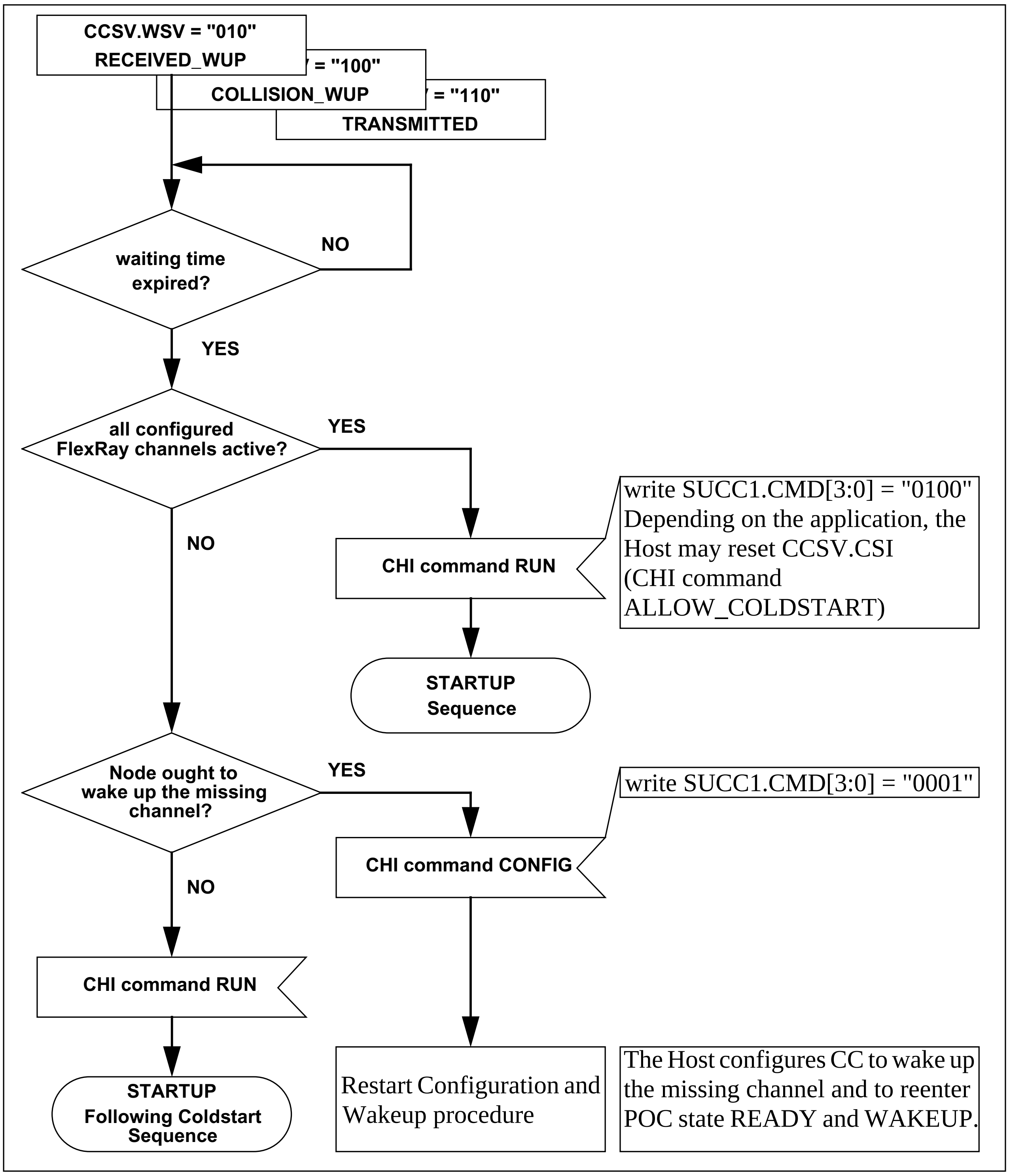
CCSV.WSV = "010"
RECEIVED_WUP
= "100"
COLLISION_WUP
= "110"
TRANSMITTED
waiting time expired?
NO
YES
all configured FlexRay channels active?
YES
NO
CHI command RUN
write SUCC1.CMD[3:0] = "0100" Depending on the application, the Host may reset CCSV.CSI (CHI command ALLOW_COLDSTART)
STARTUP Sequence
Node ought to wake up the missing channel?
YES
NO
write SUCC1.CMD[3:0] = "0001"
CHI command CONFIG
CHI command RUN
STARTUP Following Coldstart Sequence
Restart Configuration and Wakeup procedure
The Host configures CC to wake up the missing channel and to reenter POC state READY and WAKEUP.

Figure 5: Flowchart treating wakeup status RECEIVED_WUP, COLLISION_WUP or TRANSMITTED

4.3.6.3 COLLISION_UNKNOWN

In case the wakeup status signals COLLISION_UNKNOWN, the CC stops wakeup by leaving POC state WAKEUP_DETECT after expiration of the wakeup timer without receiving a valid wakeup pattern or valid frame header.

This may occur because of noise interferences or because of disturbed communication on the FlexRay channel.

The Host can neither assume the wakeup channel nor the second FlexRay channel to be awake.

The Host shall wait the minimal time for another node to become awake, to be configured, to become ready and to enter wakeup, before proceeding with a new wakeup attempt.

The Host may reconfigure the value listen timeout noise (**SUCC2.LTN[3:0]**) to enable wakeup under more difficult conditions regarding noise interference.

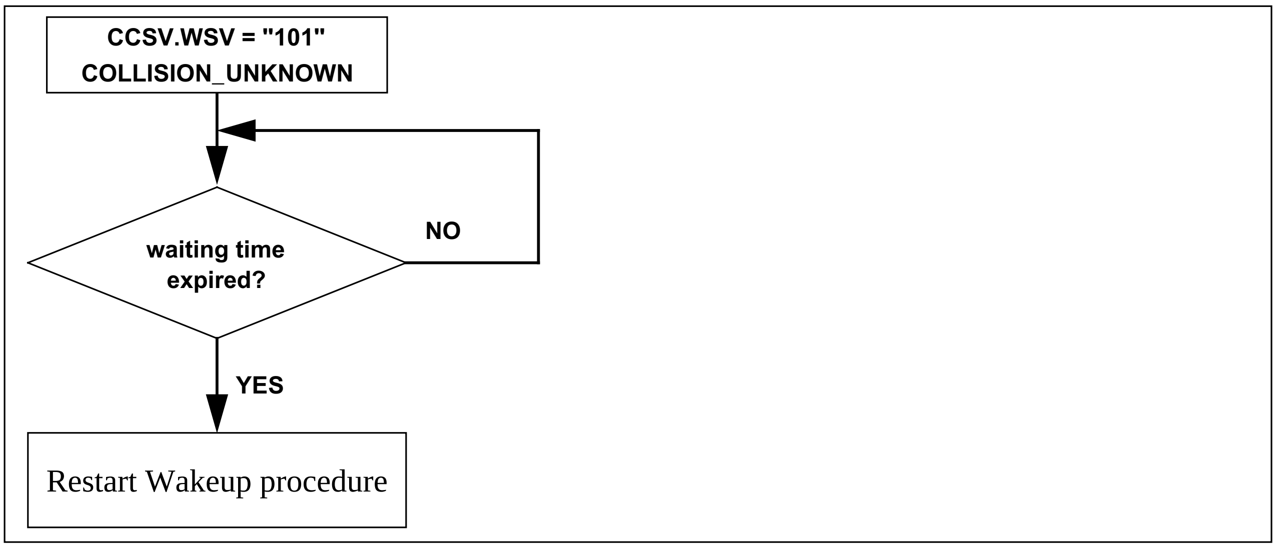

Figure 6: Flowchart treating wakeup status COLLISION_UNKNOWN

4.3.7 Implementation Hints

4.3.7.1 Command Not Accepted

If a CHI command cannot be executed by the CC, for example the command is given in the wrong POC state, the CC is still executing another CHI command, or an internal state change of the CC occurs (**EIR.CCL** = "1"), the CC returns **SUCC1.CMD[3:0]** = "0000" (command_not_accepted) and sets **EIR.CNA** = "1".

It is recommend to read the **SUCC1.CMD[3:0]** value after each written CHI command to check whether the CC accepts or rejects the given command.

4.3.7.2 Sync Frame and Startup Frame Configuration Rules

In case a FlexRay node is configured as sync node (**SUCC1.TXSY** = "1") or as coldstart node (**SUCC1.TXSY** = "1" and **SUCC1.TXST** = "1"), the node must be connected to all available channels of the cluster. To configure a sync or a coldstart node as single-channel node as part of a dual-channel network is invalid.

For example, if the node is connected to channel A and channel B, (**SUCC1.CCHA** = "1" and **SUCC1.CCHB** = "1"), the channel filter configuration of the message buffer which holds the key slot ID (message buffer 0, message buffer 1, respectively) has to be chosen accordingly (**WRHS1.CHA** = "1" and **WRHS1.CHB** = "1").

In addition, **SUCC1.TXST** and **SUCC1. TXSY** have to be programmed to "1".

4.3.7.3 Payload of Startup Frames

The configured payload of startup frames (**WRHS2.PLC[6:0]**) must be configured equal to the static frame data length (**MHDC.SFDL[6:0]**). Otherwise, the frames will not be receipt as valid startup frames and the startup fails.

4.3.7.4 Key Slot ID

The frame ID which is configured with the header section of message buffer 0 is the key slot ID.

The FlexRay protocol specification requires that each node has to send a frame in its key slot. Therefore at least message buffer 0 is reserved for transmission in key slot.

The frame configured with the key slot ID can be configured as startup frame, as sync frame, as single slot frame or as normal frame.

In case the frame configured with the key slot ID ought to be a startup frame, the flags **SUCC1.TXSY** and **SUCC1.TXST** must be configured to "1".

In case the frame configured with the key slot ID ought to be a sync frame, the flag **SUCC1.TXSY** must be configured to "1" and the flag **SUCC1.TXST** must be configured to "0".

In case the frame configured with the key slot ID ought to be a single slot frame (the CC only transmit in the pre-configured key slot), the flag **SUCC1.TSM** must be configured to "1". A single slot frame may be a sync or startup frame, too. **SUCC1.TXST** and **SUCC1.TXSY** have to be programmed accordingly. In ALL slots mode the CC may transmit in all slots. **SUCC1.TSM** is a configuration bit which can only be set/reset by the Host. The bit can be written in DEFAULT_CONFIG or CONFIG state only.

In case the frame configured with the key slot ID ought to be a normal frame, the flags **SUCC1.TXSY**, **SUCC1.TXST** and **SUCC1.TSM** must be configured to "0".
The configuration with **SUCC1.TXSY** = "0" and **SUCC1.TXST** = "1" is invalid.

Note: In case the node is configured as sync node (**SUCC1.TXSY** = "1") or for single slot mode operation (**SUCC1.TSM** = "1"), the message buffer 0 resp. 1 is reserved for sync frames or single slot frames and have to be configured with the node-specific key slot ID. In case the node is configured as sync node (**SUCC1.TXSY** = "1") or for single slot mode operation (**SUCC1.TSM** = "1"), message buffer 0 respectively message buffer 0,1 can be (re)configured in DEFAULT_CONFIG or CONFIG state only.

4.3.7.5 Different Payload for Startup, Sync and Single Slot Frames

In case the startup frame, sync frame or single slot frame shall contain different payload on channel A and channel B, two message buffers have to be configured.

The first message buffer must be message buffer 0. It has to be configured as transmit buffer with the key slot ID as frame ID, channel filter control for channel A only and the dedicated payload for channel A.

The second message buffer must be message buffer 1. It has to be configured as transmit buffer with the key slot ID as frame ID, channel filter control for channel B only and the dedicated payload for channel B.

In addition,
for a startup frame, **SUCC1.TXST** and **SUCC1.TXSY** have to be programmed to "1",
for a sync frame, **SUCC1.TXSY** has to be programmed to "1",
for a single slot frame, **SUCC1.TSM** has to be programmed to "1". A single slot frame may be a sync or startup frame, too. **SUCC1.TXST** and **SUCC1.TXSY** have to be programmed accordingly.

It is recommend to set **MRC.SPLM** = "1" to lock both message buffers 0 and 1 against reconfiguration.

4.3.7.6 Wakeup pattern Configuration Rules

The wakeup pattern is composed of at least two wakeup symbols. Wakeup symbol and wakeup pattern are configured by registers PRTC1 and PRTC2.

The WUP and the WUS are configured by the following definitions:

- Wakeup symbol must be configured identical in all nodes of a cluster
- Wakeup symbol transmit low time configured by **PRTC2.TXL[5:0]**
- Wakeup symbol idle time used to listen for activity on the bus, configured by **PRTC2.TXI[7:0]**
- A wakeup pattern composed of at least two Tx-wakeup symbols needed for wakeup
- Number of repetitions configurable by **PRTC1.RWP[5:0]** (2 to 63 repetitions)
- Wakeup symbol receive window length configured by **PRTC1.RXW[8:0]**
- Wakeup symbol receive low time configured by **PRTC2.RXL[5:0]**
- Wakeup symbol receive idle time configured by **PRTC2.RXI[5:0]**

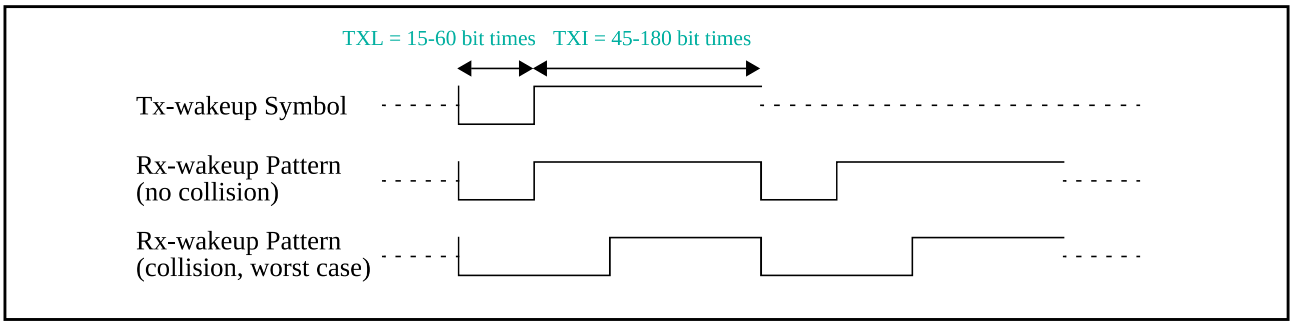
TXL = 15-60 bit times
TXI = 45-180 bit times
Tx-wakeup Symbol
Rx-wakeup Pattern (no collision)
Rx-wakeup Pattern (collision, worst case)

Figure 7: Timing of wakeup pattern

4.3.7.7 Configuration of Listen Timeout and Listen Timeout Noise

The wakeup timer and the wakeup noise timer are controlling the POC state WAKEUP_LISTEN. These two timers are controlled by the parameters listen timeout **SUCC2.LT[20:0]** and listen timeout noise **SUCC2.LTN[3:0]**.

Listen timeout enables a fast cluster wakeup in case of a noise free environment, while listen timeout noise enables wakeup under more difficult conditions regarding noise interference. (With the parameter listen timeout the upper limit for the wakeup (and startup) listen is configured, with the parameter listen timeout noise a multiplier for the listen timeout value is configured to increase the listen timeout value in the presence of noise.)

4.3.7.8 Unsuccessful Startup

Some events or conditions may prevent a cluster from waking up even though a wakeup attempt has been made. The Host detects such an error when the cluster does not start up successfully after the wakeup.

The Host may then initiate a retransmission of the wakeup pattern. Note that the Host shall wait the minimal time for another node to become awake, to be configured, to become ready and to enter wakeup, or rather startup, before proceeding with a new wakeup attempt.

Although, this might disturb ongoing communication of other nodes if the node initiating the wakeup procedure is subject to an incoming link failure or a fault in the communication controller. The Host must ensure that such a failure condition will not lead to a permanent disturbance on the bus.

4.4 Application Example

4.4.1 Message RAM Configuration

At least two coldstart nodes are necessary to start up a FlexRay cluster, named node A and node B in this example. Furthermore, these two nodes are configured to wake up the two FlexRay channels. The configuration of the message RAM of the different nodes is shown in the following figures:

RAM Word				
0..3	Message Buffer 0	TX, FID = 1, DP = 0x18		Start of Header Partition
4..7	Message Buffer 1	RX, FID = 2, DP = 0x1C		
8..11	Message Buffer 2	FIFO, DP = 0x20	⇐ **FFB**	
12..15	Message Buffer 3	FIFO, DP = 0x24		
16..19	Message Buffer 4	FIFO, DP = 0x28		
20..23	Message Buffer 5	FIFO, DP = 0x2C	⇐ **LCB**	End of Header Partition

Figure 8: Message RAM structure node A

RAM Word				
0..3	Message Buffer 0	TX, FID = 2, DP = 0x18		Start of Header Partition
4..7	Message Buffer 1	RX, FID = 1, DP = 0x1C		
8..11	Message Buffer 2	FIFO, DP = 0x20	⇐ **FFB**	
12..15	Message Buffer 3	FIFO, DP = 0x24		
16..19	Message Buffer 4	FIFO, DP = 0x28		
20..23	Message Buffer 5	FIFO, DP = 0x2C	⇐ **LCB**	End of Header Partition

Figure 9: Message RAM structure node B

4.4.2 Basic Configuration

The calculation of the configuration parameters was done according to the constrains of Ref.1.:

Parameter	Bit(field)	Value (decimal)
gColdStartAttempts	**SUCC1.CSA[4:0]**	31
gListenNoise	**SUCC2.LTN[3:0]**	15
gMacroPerCycle	**GTUC2.MPC[13:0]**	1000
gMaxWithoutClockCorrectionFatal	**SUCC3.WCF[3:0]**	15
gMaxWithoutClockCorrectionPassive	**SUCC3.WCP[3:0]**	15
gNetworkManagementVectorLength	**NEMC.NML[3:0]**	0
gNumberOfMinislots	**GTUC8.NMS[12:0]**	190
gNumberOfStaticSlots	**GTUC7.NSS[9:0]**	6
OCS	**GTUC4.OCS[13.0]**	990
gPayloadLengthStatic	**MHDC.SFDL[6:0]**	8
gSyncNodeMax	**GTUC2.SNM[3:0]**	6
gdActionPointOffset	**GTUC9.APO[5:0]**	4
gdCASRxLowMax	**PRTC1.CASM[6:0]**	71
gdDynamicSlotIdlePhase	**GTUC9.DSI[1:0]**	1
gdMinislot	**GTUC8.MSL[5:0]**	4
gdMinislotActionPoint	**GTUC9.MAPO[4:0]**	2
NIT	**GTUC4.NIT[13:0]**	989
gdSampleClockPeriod	**PRTC1.BRP[1:0**	0
gdStaticSlot	**GTUC7.SSL[9:0]**	34
gdTSSTransmitter	**PRTC1.TSST[3:0]**	4
gdWakeupSymbolRxIdle	**PRTC2.RXI[5:0]**	59
gdWakeupSymbolRxLow	**PRTC2.RXL[5:0]**	57
gdWakeupSymbolRxWindow	**PRTC1.RXW[8:0]**	301
gdWakeupSymbolTxIdle	**PRTC2.TXI[7:0]**	180
gdWakeupSymbolTxLow	**PRTC2.TXL[5:0**	60
pAllowHaltDueToClock	**SUCC1.HCSE**	1
pAllowPassiveToActive	**SUCC1.PTA[4:0]**	15
pChannels	**SUCC1.CCHA** **SUCC1.CCHB**	1 1
pClusterDriftDamping	**GTUC5.CDD[4:0]**	2
pDecodingCorrection	**GTUC5.DEC[7:0]**	28
pDelayCompensation[A]	**GTUC5.DCA[7:0]**	0
pDelayCompensation[B]	**GTUC5.DCB[7:0]**	0
pExternOffsetCorrection	**GTUC11.EOC[2:0]**	0
pExternRateCorrection	**GTUC11.ERC[2:0]**	0
pKeySlotUsedForSync	**SUCC1.TXSY**	1
pKeySlotusedForStartup	**SUCC1.TXST**	1
pLatestTx	**MHDC.SLT[12:0**	184
pMacroInitialOffset[A]	**GTUC3.MIOA[6:0]**	5
pMacroInitialOffset[B]	**GTUC3.MIOB[6:0]**	5
pMicroInitialOffset[A]	**GTUC3.UIOA[7:0]**	12
pMicroInitialOffset[B]	**GTUC3.UIOB[7:0]**	12
pMicroPerCycle	**GTUC1.UT[19:0]**	40000

Parameter	Bit(field)	Value (decimal)
pOffsetCorrectionOut	**GTUC10.MOC[13:0]**	160
pRateCorrectionOut	**GTUC10.MRC[10:0]**	121
pSamplesPerMicrotick	**PRTC1.BRP[1:0]**	0
pSingleSlotEnabled	**SUCC1.TSM**	0
pWakeupChannel	**SUCC1.WUCS**	A=0, B=1
pWakeupPattern	**PRTC1.RWP[5:0]**	63
pdAcceptedStartupRange	**GTUC6.ASR[10:0]**	77
pdListenTimeOut	**SUCC2.LT[20:0]**	A=80242, B=120363
pdMaxDrift	**GTUC6.MOD[10:0]**	121

Table 2: Basic configuration parameters

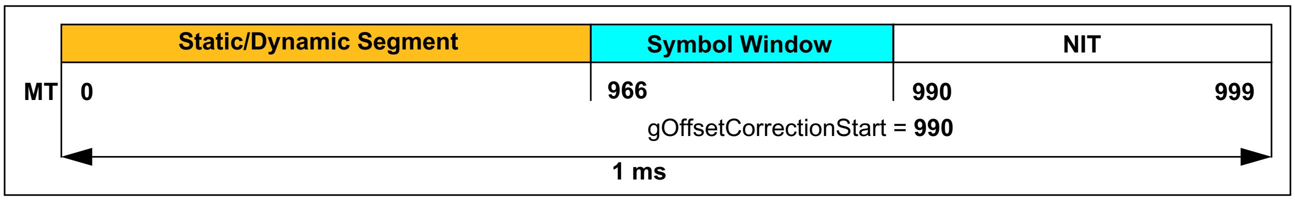

Figure 10: Structure of communication cycle

4.4.3 Configuration Procedure

The following source code is based on the configuration schedule from figure 2.

4.4.3.1 Configuration of Wakeup Node A

```
                                    // Hardware Reset Power On
                                    //
                                    // POC state DEFAULT_CONFIG
                                    //
                                    // wait for cleared CRAM flag
while((read32bit(MHDS) & 0x00000080) != 0);
                                    //
                                    // CHI command CONFIG
while ((read32bit(SUCC1) & 0x00000080) != 0);
write32bit(SUCC1, 0x00000001);
if ((read32bit(SUCC1) & 0x0000000F) == 0) return 1;
while ((read32bit(CCSV) & 0x0000003F) != 0x0F);
                                    //
                                    // POC state CONFIG
                                    //
                                    // E-Ray Configuration
write32bit(SUCC2, 0x0F013972);
write32bit(SUCC3, 0x000000FF);
write32bit(NEMC,  0x00000000);
write32bit(PRTC1, 0xFD2D0474);
write32bit(PRTC2, 0x3CB4393B);
write32bit(MHDC,  0x00B80008);
write32bit(GTUC1, 0x00009C40);
write32bit(GTUC2, 0x000603E8);
write32bit(GTUC3, 0x05050C0C);
write32bit(GTUC4, 0x03DE03DD);
write32bit(GTUC5, 0x1C020000);
write32bit(GTUC6, 0x0079004D);
write32bit(GTUC7, 0x00060022);
write32bit(GTUC8, 0x00BE0004);
write32bit(GTUC9, 0x00010204);
write32bit(GTUC10, 0x007900A0);
write32bit(GTUC11, 0x00000000);
                                    // Message RAM configuration
write32bit(MRC, 0x03050280);
write32bit(WRHS1, 0x27000001);      // transmit buffer, frame ID = 1
write32bit(WRHS2, 0x0008011B);      // payload length = 8 two-byte words
write32bit(WRHS3, 0x00000018);
                                    //
write32bit(IBCM, 0x00000005);
while ((read32bit(IBCR) & 0x00008000) != 0);
write32bit(IBCR, 0x00000000);
                                    //
write32bit(WRHS1, 0x23000002);      // receive buffer, frame ID = 2
write32bit(WRHS2, 0x00080000);      // payload length = 8 two-byte words
write32bit(WRHS3, 0x0000001C);
                                    //
write32bit(IBCM, 0x00000001);
while ((read32bit(IBCR) & 0x00008000) != 0);
write32bit(IBCR, 0x00000001);
                                    //
write32bit(WRHS1, 0x00000000);      // FIFO buffer
write32bit(WRHS2, 0x00080000);      // payload length = 8 two-byte words
```

```
write32bit(WRHS3, 0x00000020);

write32bit(IBCM, 0x00000001);
while ((read32bit(IBCR) & 0x00008000) != 0);
write32bit(IBCR, 0x00000002);
                                    //
write32bit(WRHS1, 0x00000000);      // FIFO buffer
write32bit(WRHS2, 0x00080000);      // payload length = 8 two-byte words
write32bit(WRHS3, 0x00000024);
                                    //
write32bit(IBCM, 0x00000001);
while ((read32bit(IBCR) & 0x00008000) != 0);
write32bit(IBCR, 0x00000003);
                                    //
write32bit(WRHS1, 0x00000000);      // FIFO buffer
write32bit(WRHS2, 0x00080000);      // payload length = 8 two-byte words
write32bit(WRHS3, 0x00000028);
                                    //
write32bit(IBCM, 0x00000001);
while ((read32bit(IBCR) & 0x00008000) != 0);
write32bit(IBCR, 0x00000004);
                                    //
write32bit(WRHS1, 0x00000000);      // FIFO buffer
write32bit(WRHS2, 0x00080000);      // payload length = 8 two-byte words
write32bit(WRHS3, 0x0000002C);
                                    //
write32bit(IBCM, 0x00000001);
while ((read32bit(IBCR) & 0x00008000) != 0);
write32bit(IBCR, 0x00000005);
                                    //
                                    // write Configuration Lock Key
while ((read32bit(SUCC1) & 0x00000080) != 0);
write32bit(LCK, 0x000000CE);
write32bit(LCK, 0x00000031);
                                    //
                                    // CHI command READY
write32bit(SUCC1, 0x0F8FFB02);      // SUCC1.TXSY = SUCC1.TXST = 1, WUCS = 0
if ((read32bit(SUCC1) & 0x0000000F) == 0) return 1;
while ((read32bit(CCSV) & 0x0000003F) != 0x01);
                                    //
                                    // POC state READY
```

APP_description.fm

4.4.3.2 Configuration of Wakeup Node B

```
                                    // Hardware Reset Power On
                                    //
                                    // POC state DEFAULT_CONFIG
                                    //
                                    // wait for cleared CRAM flag
while((read32bit(MHDS) & 0x00000080) != 0);
                                    //
                                    // CHI command CONFIG
while ((read32bit(SUCC1) & 0x00000080) != 0);
write32bit(SUCC1, 0x00000001);
if ((read32bit(SUCC1) & 0x0000000F) == 0) return 1;
while ((read32bit(CCSV) & 0x0000003F) != 0x0F);
                                    //
                                    // POC state CONFIG
                                    //
                                    // E-Ray Configuration
write32bit(SUCC2, 0x0F01D62B);
write32bit(SUCC3, 0x000000FF);
write32bit(NEMC,  0x00000000);
write32bit(PRTC1, 0xFD2D0474);
write32bit(PRTC2, 0x3CB4393B);
write32bit(MHDC,  0x00B80008);
write32bit(GTUC1, 0x00009C40);
write32bit(GTUC2, 0x000603E8);
write32bit(GTUC3, 0x05050C0C);
write32bit(GTUC4, 0x03DE03DD);
write32bit(GTUC5, 0x1C020000);
write32bit(GTUC6, 0x0079004D);
write32bit(GTUC7, 0x00060022);
write32bit(GTUC8, 0x00BE0004);
write32bit(GTUC9, 0x00010204);
write32bit(GTUC10, 0x007900A0);
write32bit(GTUC11, 0x00000000);
                                    // Message RAM configuration
write32bit(MRC, 0x03050280);
write32bit(WRHS1, 0x27000002);      // transmit buffer, frame ID = 2
write32bit(WRHS2, 0x00080304);      // payload length = 8 two-byte words
write32bit(WRHS3, 0x00000018);

write32bit(IBCM, 0x00000005);
while ((read32bit(IBCR) & 0x00008000) != 0);
write32bit(IBCR, 0x00000000);
                                    //
write32bit(WRHS1, 0x23000001);      // receive buffer, frame ID = 1
write32bit(WRHS2, 0x00080000);      // payload length = 8 two-byte words
write32bit(WRHS3, 0x0000001C);
                                    //
write32bit(IBCM, 0x00000001);
while ((read32bit(IBCR) & 0x00008000) != 0);
write32bit(IBCR, 0x00000001);
                                    //
write32bit(WRHS1, 0x00000000);      // FIFO buffer
write32bit(WRHS2, 0x00080000);      // payload length = 8 two-byte words
write32bit(WRHS3, 0x00000020);
                                    //
write32bit(IBCM, 0x00000001);
while ((read32bit(IBCR) & 0x00008000) != 0);
```

```
write32bit(IBCR, 0x00000002);
                                    //
write32bit(WRHS1, 0x00000000);      // FIFO buffer
write32bit(WRHS2, 0x00080000);      // payload length = 8 two-byte words
write32bit(WRHS3, 0x00000024);
                                    //
write32bit(IBCM, 0x00000001);
while ((read32bit(IBCR) & 0x00008000) != 0);
write32bit(IBCR, 0x00000003);
                                    //
write32bit(WRHS1, 0x00000000);      // FIFO buffer
write32bit(WRHS2, 0x00080000);      // payload length = 8 two-byte words
write32bit(WRHS3, 0x00000028);
                                    //
write32bit(IBCM, 0x00000001);
while ((read32bit(IBCR) & 0x00008000) != 0);
write32bit(IBCR, 0x00000004);
                                    //
write32bit(WRHS1, 0x00000000);      // FIFO buffer
write32bit(WRHS2, 0x00080000);      // payload length = 8 two-byte words
write32bit(WRHS3, 0x0000002C);
                                    //
write32bit(IBCM, 0x00000001);
while ((read32bit(IBCR) & 0x00008000) != 0);
write32bit(IBCR, 0x00000005);
                                    //
                                    // write Configuration Lock Key
while ((read32bit(SUCC1) & 0x00000080) != 0);
write32bit(LCK, 0x000000CE);
write32bit(LCK, 0x00000031);
                                    //
                                    // CHI command READY
write32bit(SUCC1, 0x0FAFFB02);      // SUCC1.TXSY = SUCC1.TXST = 1, WUCS = 1
if ((read32bit(SUCC1) & 0x0000000F) == 0) return 1;
while ((read32bit(CCSV) & 0x0000003F) != 0x01);
                                    //
                                    // POC state READY
```

4.4.4 Wakeup Procedure

4.4.4.1 Wakeup of Wakeup Node A

If the FlexRay channels are silent (bus driver has not received a wakeup pattern), the following wake-up procedure is scheduled by the wakeup node A:

The wakeup node A enters POC state WAKEUP. After transmission of the wakeup pattern on channel A it waits some time for the receiption of the wakeup pattern on channel B.

If it does not receive the wakeup pattern on channel B, it assumes that the wakeup node B does not received the wakeup pattern on channel A and reenters POC state WAKEUP to transmit a new one. It is not intended to wake up the other FlexRay channel.

```
                                  // wakeup loop
for(;;){
                                  // CHI command WAKEUP
  while ((read32bit(SUCC1) & 0x00000080) != 0);
  write32bit(SUCC1, 0x00000003);
  if ((read32bit(SUCC1) & 0x0000000F) == 0) return 1;
  while ((read32bit(SUCC1) & 0x00000030) != 0x10);
                                  //
                                  // POC state WAKEUP
                                  //
                                  // wait until POC state READY
  do{
    value32bit = (read32bit(CCSV) & 0x0000003F);
  }while (value32bit != 0x00000001);
                                  //
                                  // check status flags
  cluster_active = channel_active = 0;
                                  //
                                  // RECEIVED_HEADER
  if ((read32bit(CCSV) & 0x00070000) == 0x00010000) {
    cluster_active = 1;
    channel_active = 1;
  }
                                  // COLLISION_HEADER
  if ((read32bit(CCSV) & 0x00070000) == 0x00030000) {
    cluster_active = 1;
    channel_active = 1;
  }
                                  // RECEIVED_WUP
  if ((read32bit(CCSV) & 0x00070000) == 0x00020000) {
    cluster_active = 0;
    channel_active = 1;
  }
                                  // COLLISION_WUP
  if ((read32bit(CCSV) & 0x00070000) == 0x00040000) {
    cluster_active = 0;
    channel_active = 1;
  }
                                  // COLLISION_UNKNOWN
  if ((read32bit(CCSV) & 0x00070000) == 0x00050000) {
    cluster_active = 0;
    channel_acitve = 0;
  }
                                  // TRANSMITTED
  if ((read32bit(CCSV) & 0x00070000) == 0x00060000) {
    cluster_active = 0;
    channel_active = 1;
  }
                                  // if cluster is active, enter
                                  // POC state STARTUP with CSI reset
  if (cluster_active = 1) {
                                  //
                                  // CHI command ALLOW_COLDSTART
    while ((read32bit(SUCC1) & 0x00000080) != 0);
    write32bit(SUCC1, 0x00000009);
    if ((read32bit(SUCC1) & 0x0000000F) == 0) return 1;
    while ((read32bit(CCSV) & 0x00004000) != 0x00000000);
                                  //
                                  //
                                  // CHI command RUN
    while ((read32bit(SUCC1) & 0x00000080) != 0);
    write32bit(SUCC1, 0x00000004);
    if ((read32bit(SUCC1) & 0x0000000F) == 0) return 1;
    while ((read32bit(CCSV) & 0x00000030) != 0x20);
                                  //
                                  //
                                  // program statup procedure
                                  // continuation here
                                  // ...
    while (1);
  }
                                  //
                                  // if only wakeup channel is active, enter
                                  // POC state STARTUP with CSI set
  if (channel_active = 1) {
                                  //
                                  // CHI command RUN
    while ((read32bit(SUCC1) & 0x00000080) != 0);
    write32bit(SUCC1, 0x00000004);
    if ((read32bit(SUCC1) & 0x0000000F) == 0) return 1;
    while ((read32bit(CCSV) & 0x00000030) != 0x20);
                                  //
                                  // wait some time for WUP from other channel
    starttime_ms = gettime_ms();
    do{
      value32bit = (read32bit(SIR) & 0x01000000);
      actualtime_ms = gettime_ms();
    }while ((value32bit != 0x01000000) && (actualtime_ms < (starttime_ms+400));
                                  //
                                  // if WUP from other channel was received,
                                  // reset CSI flag with
                                  // CHI command ALLOW_COLDSTART
    if (value32bit == 0x01000000) {
      while ((read32bit(SUCC1) & 0x00000080) != 0);
      write32bit(SUCC1, 0x00000009);
      if ((read32bit(SUCC1) & 0x0000000F) == 0) return 1;
      while ((read32bit(CCSV) & 0x00004000) != 0x00000000);
    }
                                  //
                                  // program statup procedure
                                  // continuation here
                                  // ...
                                  // startup was successful
    if ((read32bit(CCSV) & 0x0000003F) == 0x00000002) while (1);
                                    //
                                    // restart wakeup if startup was
                                    // not successful
  }
                                    //
                                    // reenter POC state WAKEUP
                                    // wait and try again
  waittime_ms(800);
}
```

4.4.4.2 Wakeup of Wakeup Node B

If the FlexRay channels are silent (bus driver has not received a wakeup pattern), the following wake-up procedure is scheduled by the wakeup node B:

The wakeup node B remains in POC state READY until the WUP from the wakeup node A was received.

Afterwards, the Host triggers the CC to enter POC state WAKEUP. After successfull transmission of the WUP, the node enters POC state STARTUP.

```
                                    // wait for WUP on channel A
do{
  value32bit = (read32bit(SIR) & 0x00010000);
}while (value32bit != 0x00010000);
                                    //
                                    // wakeup procedure
                                    //
                                    // CHI command WAKEUP
while ((read32bit(SUCC1) & 0x00000080) != 0);
write32bit(SUCC1, 0x00000003);
if ((read32bit(SUCC1) & 0x0000000F) == 0) return 1;
while ((read32bit(CCSV) & 0x00000030) != 0x00000020);
                                    //
                                    // POC state WAKEUP
                                    //
                                    // wait until POC state READY
do{
  value32bit = (read32bit(CCSV) & 0x0000003F);
}while (value32bit != 0x00000001);
                                    //
                                    // check status flags
cluster_active = channel_active = 0;
                                    // RECEIVED_HEADER
if ((read32bit(CCSV) & 0x00070000) == 0x00010000) {
  cluster_active = 1;
  channel_active = 1;
}
                                    // COLLISION_HEADER
if ((read32bit(CCSV) & 0x00070000) == 0x00030000) {
  cluster_active = 1;
  channel_active = 1;
}
                                    // RECEIVED_WUP
if ((read32bit(CCSV) & 0x00070000) == 0x00020000) {
  cluster_active = 0;
  channel_active = 1;
}
                                    // COLLISION_WUP
if ((read32bit(CCSV) & 0x00070000) == 0x00040000) {
  cluster_active = 0;
  channel_active = 1;
}
                                    // COLLISION_UNKNOWN
if ((read32bit(CCSV) & 0x00070000) == 0x00050000) {
  cluster_active = 0;
  channel_acitve = 0;
}
```

```
                                    // TRANSMITTED
if ((read32bit(CCSV) & 0x00070000) == 0x00060000) {
  cluster_active = 0;
  channel_active = 1;
}
                                    //
                                    // if cluster is active, enter
                                    // POC state STARTUP with CSI reset
if (cluster_active = 1) {
                                    //
                                    // CHI command ALLOW_COLDSTART
  while ((read32bit(SUCC1) & 0x00000080) != 0);
  write32bit(SUCC1, 0x00000009);
  if ((read32bit(SUCC1) & 0x0000000F) == 0) return 1;
  while ((read32bit(CCSV) & 0x00004000) != 0x00000000);
                                    //
                                    //
                                    // CHI command RUN
  while ((read32bit(SUCC1) & 0x00000080) != 0);
  write32bit(SUCC1, 0x00000004);
  if ((read32bit(SUCC1) & 0x0000000F) == 0) return 1;
  while ((read32bit(CCSV) & 0x00000030) != 0x00000020);
                                    //
                                    //
                                    // program statup procedure
                                    // continuation here
                                    //
                                    // ...
  while (1) {};
}
                                    //
                                    // if only wakeup channel is active, enter
                                    // POC state STARTUP with CSI set
if (channel_active = 1) {
                                    //
                                    // CHI command RUN
  while ((read32bit(SUCC1) & 0x00000080) != 0);
  write32bit(SUCC1, 0x00000004);
  if ((read32bit(SUCC1) & 0x0000000F) == 0) return 1;
  while ((read32bit(CCSV) & 0x00000030) != 0x00000020);
                                    //
  do{
    value32bit = (read32bit(SIR) & 0x01000000);
  }while (value32bit != 0x01000000);
                                    //
                                    // if WUP from other channel was received,
                                    // reset CSI flag with
                                    // CHI command ALLOW_COLDSTART
  while ((read32bit(SUCC1) & 0x00000080) != 0);
  write32bit(SUCC1, 0x00000009);
  if ((read32bit(SUCC1) & 0x0000000F) == 0) return 1;
  while ((read32bit(CCSV) & 0x00004000) != 0x00000000);
                                    //
                                    // program statup procedure
                                    // continuation here
                                    // ...
  while (1) {};
}
```

5. List of Tables

6. List of Figures