E-Ray

FlexRay IP Module

Application Note AN002

Startup

Date: December 3rd, 2007

for IP Revision 1.0.2

APP_cover.fm

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

Copyright Notice

Disclaimer

APP_TOC.fm

APP_TOC.fm

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	Update for E-Ray Specification 1.2.3
Version 1.0.3	01.06.2007	Kay Hammer	Update for E-Ray Specifiaction 1.2.5
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
CAS	Collision Avoidance Symbol
CHI	Controller Host Interface
CC	Communication Controller

2. Introduction

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

In particular, the configuration steps necessary to perform the startup of the cluster are described. This application note assumes that all nodes of the cluster are already awake.

This 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. Startup Procedure

4.1 Overview

In the following sections, the configuration schedule and the startup procedure for integrating and coldstart nodes are described.

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

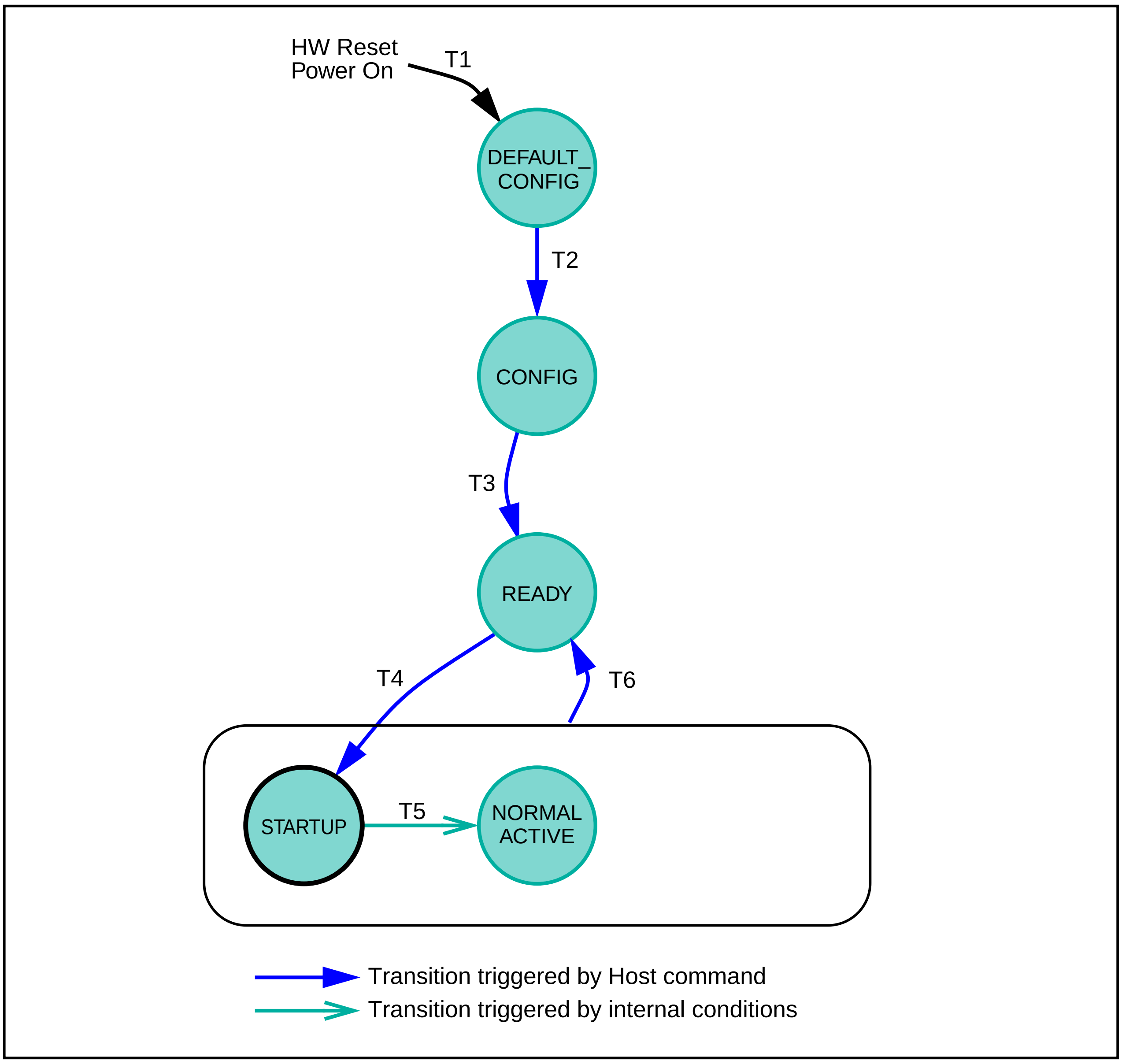
HW Reset
Power On
T1
DEFAULT_
CONFIG
T2
CONFIG
T3
READY
T4
T6
STARTUP
T5
NORMAL
ACTIVE
Transition triggered by Host command
Transition triggered by internal conditions

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

T#	Condition	From	To
1	Externem 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 RUN, **SUCC1.CMD[3:0]** = "0100"	READY	STARTUP
5	Successful startup	STARTUP	NORMAL_ACTIVE
6	Command READY, SUCC1.CMD[3:0] = "0010"	STARTUP, NORMAL_ACTIVE	READY

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

4.2 Configuration Schedule

Besides of some specific configuration parameters, the configuration schedule is identical for integrating and coldstart nodes.

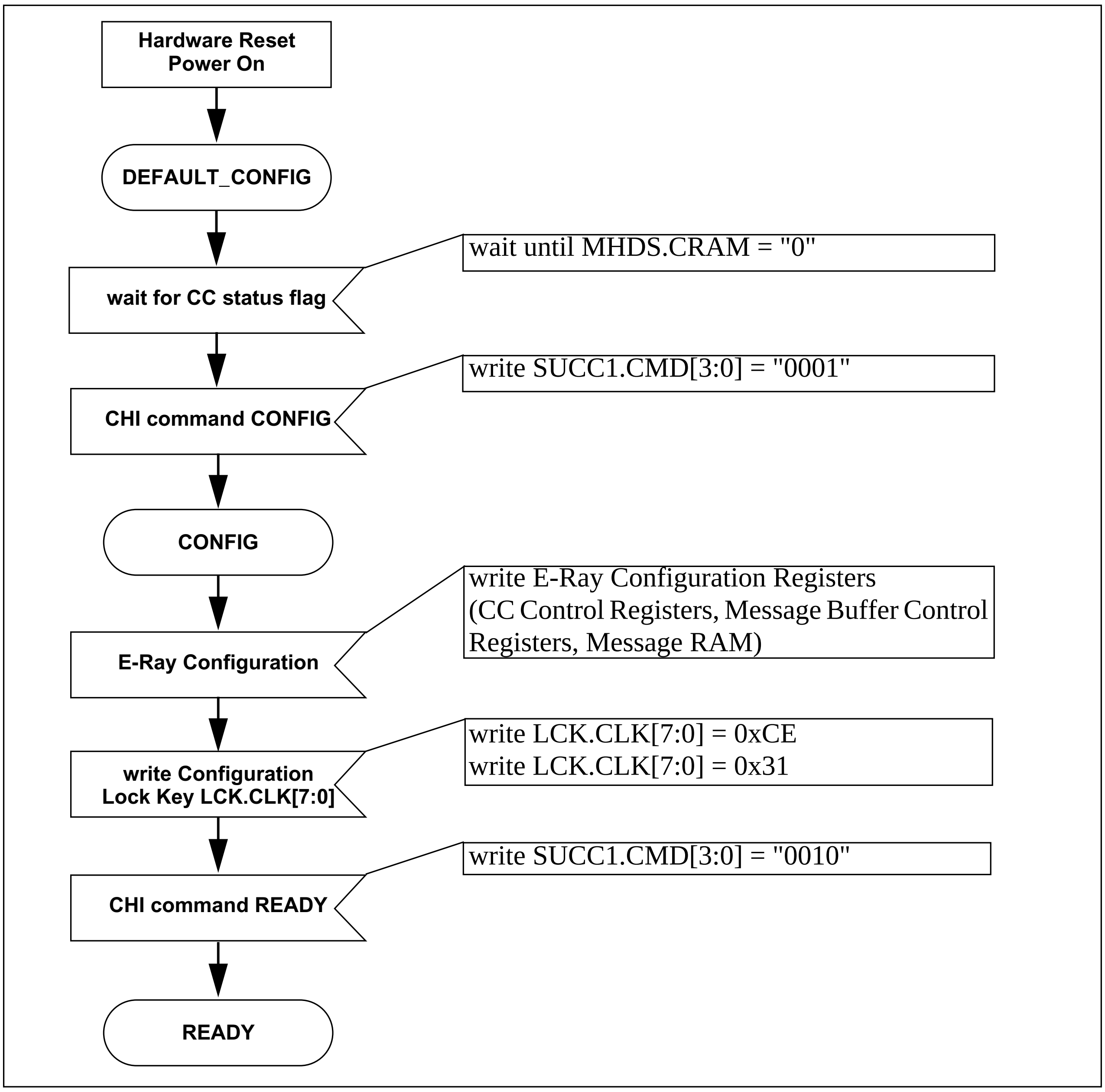
Hardware Reset
Power On
DEFAULT_CONFIG
wait for CC status flag
wait until MHDS.CRAM = "0"
CHI command CONFIG
write SUCC1.CMD[3:0] = "0001"
CONFIG
E-Ray Configuration
write E-Ray Configuration Registers (CC Control Registers, Message Buffer Control Registers, Message RAM)
write Configuration Lock Key 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 startup 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 the 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 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.

4.3.3.1 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                         *
***************************************************************************/
```

4.3.3.2 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                        *
*************************************************************************/
```

4.3.3.3 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                          *
***************************************************************************/
```

4.3.4 Enter POC state STARTUP

4.3.4.1 General

Before entering the STARTUP state, the nodes must ensure that the FlexRay channels to which they are connected are not in sleep mode. At least two coldstart nodes are necessary to start up a FlexRay cluster.

As explained in Ref. 1 and 2, a node can enter startup on three possible paths - as leading coldstarter, as following coldstarter or as integrating node.

Each node can be configured as coldstart node, as sync node, as single slot node or as normal node by the flags **SUCC1.TXSY** (Transmit Sync Frame in Key Slot), **SUCC1.TXST** (Transmit Startup Frame in Key Slot) and **SUCC1.TSM** (Transmission Slot Mode).

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

A sync node which is not a coldstart node must be configured by **SUCC1.TXSY** = "1" and **SUCC1.TXST** = "0".

For a single slot node (the CC only transmit in the pre-configured key slot, see chapter 4.3.5.4), the flag **SUCC1.TSM** must be configured to "1". A single slot mode may be a sync or a coldstart node, too. **SUCC1.TXST** and **SUCC1.TXSY** have to be programmed accordingly.

A node which is neither a coldstart node nor a sync node must be configured by **SUCC1.TXSY** = "0" and **SUCC1.TXST** = "0".

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

The startup procedure controlled by the Host is described in the following sub chapters.

4.3.4.2 Coldstart Node

If the CC is in POC state READY (**CCSV.POCS[5:0]** = "00 0001"), startup can be entered by CHI command RUN (**SUCC1.CMD[3:0]** = "0100").

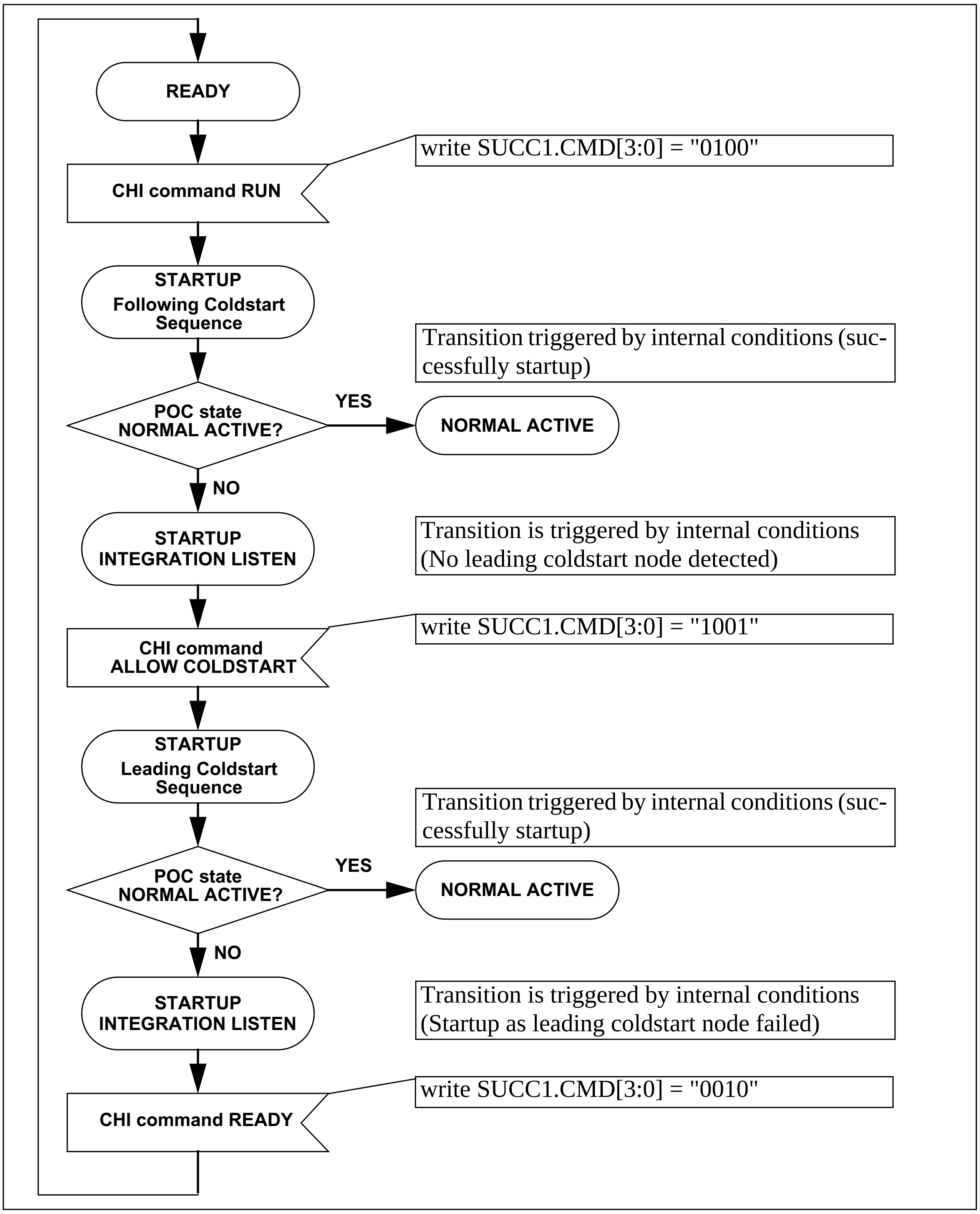
READY
write SUCC1.CMD[3:0] = "0100"
CHI command RUN
STARTUP Following Coldstart Sequence
Transition triggered by internal conditions (successfully startup)
POC state NORMAL ACTIVE?
YES
NORMAL ACTIVE
NO
STARTUP INTEGRATION LISTEN
Transition is triggered by internal conditions (No leading coldstart node detected)
write SUCC1.CMD[3:0] = "1001"
CHI command ALLOW COLDSTART
STARTUP Leading Coldstart Sequence
Transition triggered by internal conditions (successfully startup)
POC state NORMAL ACTIVE?
YES
NORMAL ACTIVE
NO
STARTUP INTEGRATION LISTEN
Transition is triggered by internal conditions (Startup as leading coldstart node failed)
write SUCC1.CMD[3:0] = "0010"
CHI command READY

Figure 3: Flowchart startup loop for coldstart nodes (simplified)

The number of coldstart attempts that a node is allowed to perform is configured by **SUCC1.CSA[4:0]**. For coldstart nodes, the configured number of coldstart attempts must be greater than one and identical in all coldstart nodes of a cluster.

A coldstart node should enter startup with coldstart inhibit (**CCSV.CSI** = "1") to ensure that an ongoing FlexRay communication is not disturbed by the coldstart node. Thus, after CHI command RUN, the CC enters the INTEGRATION_LISTEN state and listens to its attached channels.

If no communication is detected, the Host must enable the coldstart capability by resetting CCSV.CSI with CHI command ALLOW_COLDSTART (**SUCC1.CMD[3:0]** = "1001"). With remaining coldstart attempts, the coldstart node enters COLDSTART_LISTEN and listens to its attached channels.

Path of leading Coldstart Node (initiating coldstart)

If no communication is detected, the node enters the COLDSTART_COLLISION_RESOLUTION state and commences a coldstart attempt. The initial transmission of a CAS symbol is succeeded by the first regular cycle. This cycle has the number zero.

From cycle zero on, the node transmits its startup frame. Since each coldstart node may perform a coldstart attempt, it may occur that several nodes simultaneously transmit the CAS symbol and enter the coldstart path. This situation is resolved during the first four cycles after CAS transmission.

As soon as a node that initiates a coldstart attempt receives a CAS symbol or a frame header during these four cycles, it re-enters the COLDSTART_LISTEN state. Thereby, only one node remains in this path. In cycle four, other coldstart nodes begin to transmit their startup frames.

After four cycles in COLDSTART_COLLISION_RESOLUTION state, the node that initiated the coldstart enters the COLDSTART_CONSISTENCY_CHECK state. It collects all startup frames from cycle four and five and performs the clock correction. If the clock correction does not deliver any errors and it has received at least one valid startup frame pair, the node leaves COLDSTART_CONSISTENCY_CHECK and enters NORMAL_ACTIVE state.

The number of coldstart attempts that a node is allowed to perform is configured by **SUCC1.CSA[4:0]**. The number of remaining coldstarts attempts can be read from **CCSV.RCA[4:0]**. The number of remaining coldstart attempts is reduced by one for each attempted coldstart. A node may enter the COLDSTART_LISTEN state only if this value is larger than one and it may enter the COLDSTART_COLLISION_RESOLUTION state only if this value is larger than zero. If the number of coldstart attempts is one, coldstart is inhibited but integration is still possible.

Path of following Coldstart Node (responding to leading Coldstart Node)

When a coldstart node enters the COLDSTART_LISTEN state, it tries to receive a valid pair of startup frames to derive its schedule and clock correction from the leading coldstart node.

As soon as a valid startup frame has been received the INITIALIZE_SCHEDULE state is entered. If the clock synchronization can successfully receive a matching second valid startup frame and derive a schedule from this, the INTEGRATION_COLDSTART_CHECK state is entered.

In INTEGRATION_COLDSTART_CHECK state it is assured that the clock correction can be performed correctly and that the coldstart node from which this node has initialized its schedule is still available. The node collects all sync frames and performs clock correction in the following double-cycle. If clock correction does not signal any errors and if the node continues to receive sufficient frames from the same node it has integrated on, the COLDSTART_JOIN state is entered.

In COLDSTART_JOIN state following coldstart nodes begin to transmit their own startup frames and continue to do so in subsequent cycles. Thereby, the leading coldstart node and the nodes joining it can check if their schedules agree with each other. If the clock correction signals any error, the node aborts the integration attempt. If a node in this state sees at least one valid startup frame during all even cycles in this state and at least one valid startup frame pair during all double cycles in this state, the node leaves COLDSTART_JOIN state and enters NORMAL_ACTIVE state. Thereby it leaves STARTUP at least one cycle after the node that initiated the coldstart.

Error Handling

If the coldstart fails, it is recommend to read out the available status and error registers. The Host may re-enter startup by CHI command READY (**SUCC1.CMD[3:0]** = "0010"), followed by CHI command RUN (**SUCC1.CMD[3:0]** = "0100"). The application should ensure that an ongoing FlexRay communication is not disturbed by the new startup attempt, e.g. by observing a waiting time or by using the coldstart inhibit flag **CCSV.CSI**.

4.3.4.3 Non-Coldstart node

The following figure shows the flowchart of the startup procedure for an integrating node. The Host has to release the CHI command RUN to trigger the startup procedure. Afterwards, the Host waits until POC state NORMAL_ACTIVE is displayed be the CC.

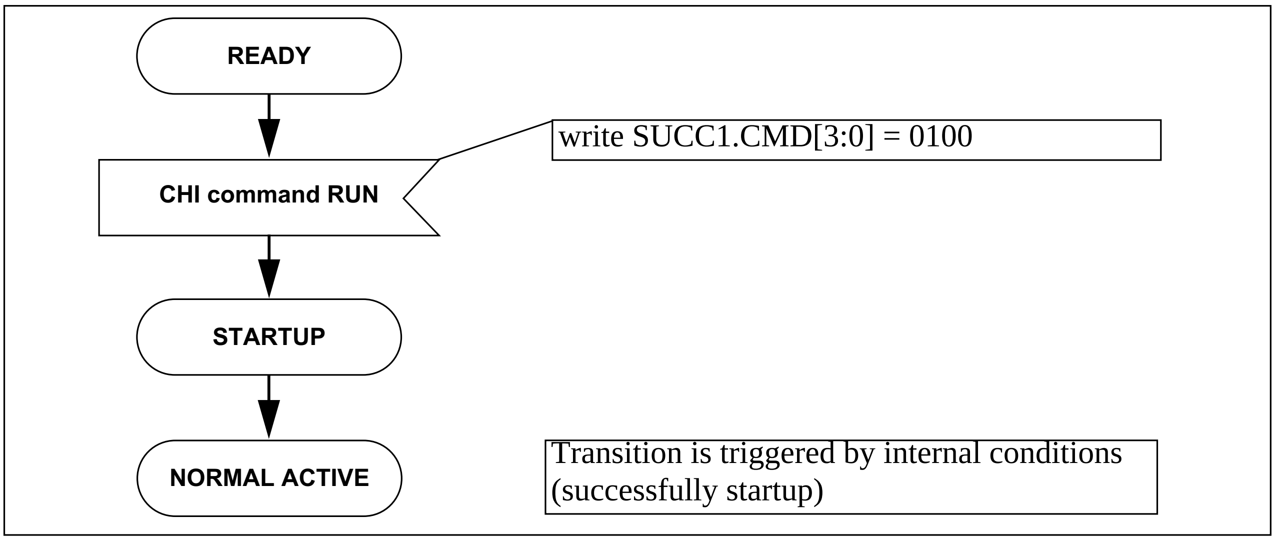
READY
CHI command RUN
write SUCC1.CMD[3:0] = 0100
STARTUP
NORMAL ACTIVE
Transition is triggered by internal conditions (successfully startup)

Figure 4: Flowchart startup procedure non-coldstart node (simplified)

If the CC is in POC state READY (**CCSV.POCS[5:0]** = "00 0001"), startup can be entered by CHI command RUN (**SUCC1.CMD[3:0]** = "0100").

A non-coldstart node is an integrating node. After CHI command RUN, the CC enters the INTEGRATION_LISTEN state and listens to its attached channels.

As soon as a valid startup frame has been received, the INITIALIZE_SCHEDULE state is entered. If the clock synchronization can successfully receive a matching second valid startup frame and derive a schedule from this, the INTEGRATION_CONSISTENCY_CHECK state is entered.

In INTEGRATION_CONSISTENCY_CHECK state the node verifies that the clock correction can be performed correctly and that enough coldstart nodes (at least 2) are sending startup frames that agree with the node's own schedule. Clock correction is activated, and if any errors are signalled, the integration attempt is aborted.

During the first even cycle in this state, either two valid startup frames or the startup frame of the node that this node has integrated on must be received; otherwise the node aborts the integration attempt.

During the first double-cycle in this state, either two valid startup frame pairs or the startup frame pair of the node that this node has integrated on must be received; otherwise the node aborts the integration attempt.

If after the first double-cycle less than two valid startup frames are received within an even cycle, or less than two valid startup frame pairs are received within a double-cycle, the startup attempt is aborted.

Nodes in this state need to see two valid startup frame pairs for two consecutive double-cycles each to be allowed to leave STARTUP and enter NORMAL_OPERATION. Consequently, they leave startup at least one double-cycle after the node that initiated the coldstart and only at the end of a cycle with an odd cycle number.

4.3.5 Implementation Hints

4.3.5.1 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.CLL** = "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.

4.3.5.2 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 coldstart node as single-channel node as part of a dual-channel cluster 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 resp. message buffer 1) 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".

4.3.5.3 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.

4.3.5.4 Key Slot ID

The frame ID which is configured within 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.

4.3.5.5 Different Payload for Startup, Sync and Single Slot Frames

In case a 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.

4.4 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. Node C is configured as integrating node. Node A and B are configured to send startup frames in slot 1 and 2, node C to send a sync frame in slot 3. 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	RX, FID = 3, DP = 0x20		
12..15	Message Buffer 3	FIFO, DP = 0x24	⇐ **FFB**	
16..19	Message Buffer 4	FIFO, DP = 0x28		
20..23	Message Buffer 5	FIFO, DP = 0x2C	⇐ **LCB**	End of Header Partition

Figure 5: 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	RX, FID = 3, DP = 0x20		
12..15	Message Buffer 3	FIFO, DP = 0x24	⇐ **FFB**	
16..19	Message Buffer 4	FIFO, DP = 0x28		
20..23	Message Buffer 5	FIFO, DP = 0x2C	⇐ **LCB**	End of Header Partition

Figure 6: Message RAM structure node B

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

Figure 7: Message RAM structure node C

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**	A=B=1, C=0
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**	0
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

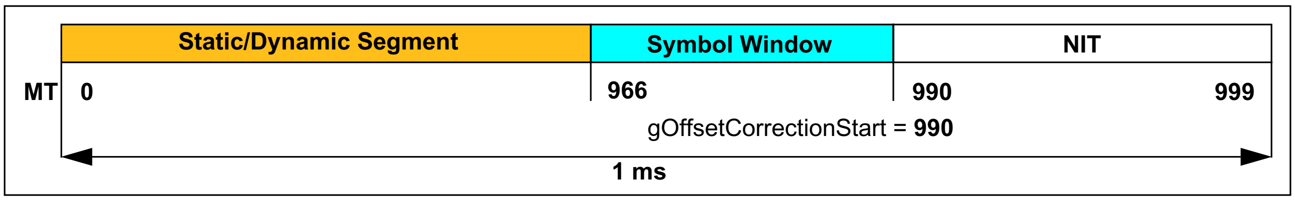

Figure 8: Structure of communication cycle

4.4.3 Configuration Procedure

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

4.4.3.1 Configuration of Coldstart 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, 0x03050380);
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, 0x23000003);      // receive buffer, frame ID = 3
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
if ((read32bit(SUCC1) & 0x0000000F) == 0) return 1;
while ((read32bit(CCSV) & 0x0000003F) != 0x00000001);
                                    //
                                    // POC state READY
```

4.4.3.2 Configuration of Coldstart 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) != 0);
                                    //
                                    // 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, 0x03050380);
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, 0x23000003);      // receive buffer, frame ID = 3
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
if ((read32bit(SUCC1) & 0x0000000F) == 0) return 1;
while ((read32bit(CCSV) & 0x0000003F) != 0x00000001);
                                    //
                                    // POC state READY
```

4.4.3.3 Configuration of Integrating Node C

```
                                    // 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) != 0);
                                    //
                                    // 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, 0x03050380);
write32bit(WRHS1, 0x27000003);      // transmit buffer, frame ID = 3
write32bit(WRHS2, 0x000805D2);      // 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, 0x23000002);      // receive buffer, frame ID = 2
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);


write32bit(IBCM, 0x00000005);
write32bit(IBCR, 0x00000000);
while ((read32bit(IBCR) & 0x00008000) != 0);
                                    //
                                    // write Configuration Lock Key
while ((read32bit(SUCC1) & 0x00000080) != 0);
write32bit(LCK, 0x000000CE);
write32bit(LCK, 0x00000031);
                                    //
                                    // CHI command READY
write32bit(SUCC1, 0x0F8FFA02);      // SUCC1.TXSY = 1, SUCC1.TXST = 0
if ((read32bit(SUCC1) & 0x0000000F) == 0) return 1;
while ((read32bit(CCSV) & 0x0000003F) != 0x00000001);
                                    //
                                    // POC state READY
```

4.4.4 Startup Procedure

4.4.4.1 Startup of Coldstart node, node A and B

The following source code is based on the configuration schedule figure 3.

```
                                    // coldstart loop
for(;;){
                                    //
                                    // CHI command RUN
  while ((read32bit(SUCC1) & 0x00000080) != 0);
  write32bit(SUCC1, 0x00000004);
  if ((read32bit(SUCC1) & 0x0000000F) == 0) return 1;
  while ((read32bit(CCSV) & 0x00000030) != 0x00000020);
                                    //
                                    // POC state STARTUP
                                    //
                                    // try to startup as following coldstarter
                                    // CSI set,
                                    // until POC state NORMAL_ACITVE
                                    // or expired timer (400 ms)
  starttime_ms = gettime_ms();
  do{
    value32bit = (read32bit(CCSV) & 0x0000003F);
    actualtime_ms = gettime_ms();
  }while ((value32bit != 0x00000002) && (actualtime_ms < (starttime_ms + 400)));
                                    //
                                    // integration was not successfull?
                                    // (POC state INTEGRATION_LISTEN)
  if (value32bit == 0x00000027){
    value32bit_2 = (read32bit(CCSV) & 0x00FF0000);
                                    //
                                    // remaining coldstart attempts?
    if (value32bit_2 != 0){
                                    // reset CSI bit to allow coldstart
      while ((read32bit(SUCC1) & 0x00000080) != 0);
      write32bit(SUCC1, 0x00000009);
      if ((read32bit(SUCC1) & 0x0000000F) == 0) return 1;
    }
  }
                                    // try to startup as leading coldstarter,
                                    // CSI flag reset,
                                    // until POC state NORMAL_ACITVE
                                    // or INTEGRATION_LISTEN
  do{
    value32bit = (read32bit(CCSV) & 0x0000003F);
  }while ((value32bit != 0x00000002) && value32bit != 0x00000027);
                                    //
                                    // startup successfull?
                                    //
  if(value32bit == 0x00000002){
                                    // NORMAL_ACTIVE loop
    do{
      value32bit = (read32bit(CCSV) & 0x0000003F);
    }while (value32bit == 0x00000002);
                                    //
                                    // left loop at NORMAL_ACTIVE
    while (1){};
  }
```

```
                                    //
                                    // startup not successfull, wait
                                    // and try again
  waittime_ms(200);
                                    //
                                    // set CHI command READY
  while ((read32bit(SUCC1) & 0x00000080) != 0);
  write32bit(SUCC1, 0x00000002);
  if ((read32bit(SUCC1) & 0x0000000F) != 0) return 1;
  while ((read32bit(CCSV) & 0x0000003F) != 0x00000001);
                                    //
                                    // POC state READY
                                    //
                                    // coldstart loop starts again
}
```

4.4.4.2 Startup of Non-Coldstart Node, Node C

The following source code is based on the configuration schedule figure 4.

```
                                    // CHI command RUN
while ((read32bit(SUCC1) & 0x00000080) != 0);
write32bit(SUCC1, 0x00000004);
if ((read32bit(SUCC1) & 0x0000000F) == 0) return 1;
while ((read32bit(CCSV) & 0x00000030) != 0x00000020);
                                    //
                                    // POC state STARTUP
                                    //
                                    // wait until POC state NORMAL_ACTIVE
do{
  value32bit = (read32bit(CCSV) & 0x0000003F);
}while (value32bit != 0x00000002);
                                    //
                                    // POC state NORMAL_ACTIVE
                                    //
                                    // NORMAL_ACTIVE loop
do{
  value32bit_4 = read32bit(CCSV) & 0x0000003F);
}while (value32bit_4 == 0x00000002);
                                    //
                                    // left loop at NORMAL_ACTIVE
while (1){};
```

5. List of Tables

6. List of Figures