E-Ray

FlexRay IP Module

Application Note AN006

Reconfiguration of Message Buffers

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

1. About this Document

1.1 Change Control

1.1.1 Current Status

Version 1.0.0

1.1.2 Change History

Issue	Date	By	Change
Version 1.0.0	03.12.2007	K. Hammer	Initial Draft 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
3	Robert Bosch GmbH	E-Ray Application Note 003 Message RAM Configuration
4	Robert Bosch GmbH	E-Ray Application Note 004 FIFO Function
5	Robert Bosch GmbH	E-Ray Application Note 005 Handling of Parity Errors

1.3 Terms and Abbreviations

This document uses the following terms and abbreviations:

Term	Meaning
CC	Communication Controller
FIFO	First In First Out

APP_about.fm

2. Introduction

This application note describes the dynamic reconfiguration of Message Buffers during FlexRay operation of an E-Ray Communication Controller. As introduction, access of Host and Protocol Controller to the Message RAM is described.

For information about the configuration of the E-Ray Message RAM consult Ref. 3. For information about the configuration of the FIFO message RAM and the FIFO function consult Ref. 4.

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. Message Handling

The Message Handler controls data transfers between the Input (IBF) / Output (OBF) Buffer and the Message RAM and between the Message RAM and the two Transient Buffer RAMs (TBF A, B). All accesses to the internal RAMs are 32+1 bit accesses. The additional bit is used for parity checking (see Ref. 5 for further information).

Access to the message buffers stored in the Message RAM is done under control of the Message Handler state machine. This avoids conflicts between accesses of the two FlexRay channel protocol controllers and the Host to the Message RAM.

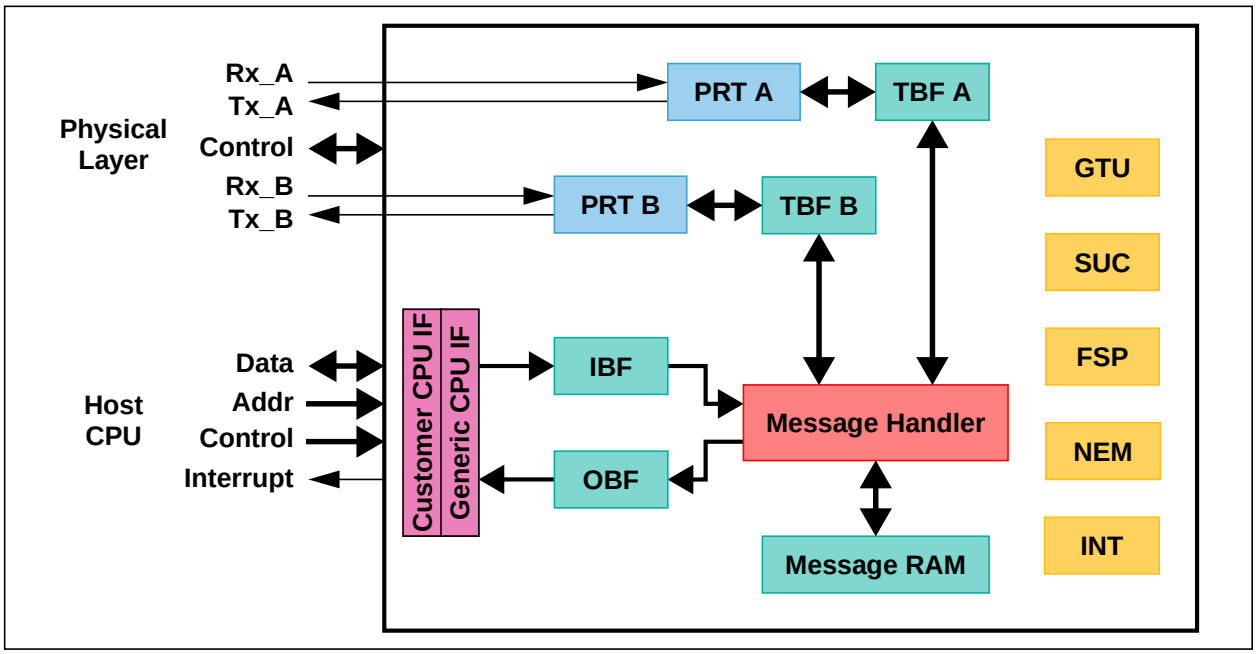

Figure 1: E-Ray block diagram

4.1 FlexRay Protocol Controller access to MRAM

The two Transient Buffer RAMs (TBF A,B) are used to buffer the data for transfer between the two FlexRay Protocol Controllers and the Message RAM.

Each Transient Buffer RAM is build up as a double buffer, able to store two complete FlexRay messages. There is always one buffer assigned to the corresponding Protocol Controller while the other one is accessible by the Message Handler.

If e.g. the Message Handler writes the next message to be send to Transient Buffer Tx, the FlexRay Channel Protocol Controller can access Transient Buffer Rx to store the message it is actually receiving. During transmission of the message stored in Transient Buffer Tx, the Message Handler transfers the last received message stored in Transient Buffer Rx to the Message RAM (if it passes acceptance filtering) and updates the respective message buffer.

Data transfers between the Transient Buffer RAMs and the shift registers of the FlexRay Channel Protocol Controllers are done in words of 32 bit. This enables the use of a 32 bit shift register independent of the length of the FlexRay messages.

4.2 Host Access to Message RAM

The message transfer between Input Buffer and Message RAM as well as between Message RAM and Output Buffer is triggered by the Host by writing the number of the target / source message buffer to be accessed to IBCR or OBCR register.

The IBCM and OBCM registers can be used to write / read header and data section of the selected message buffer separately.

If bit **IBCM.STXR** is set to = ’1’, the transmission request flag TXR of the selected message buffer is automatically set after the message buffer has been updated. If bit **IBCM.STXR** is reset to ’0’, the transmission request flag TXR of the selected message buffer is reset. This can be used to stop transmission from message buffers operated in continuous mode.

Input Buffer (IBF) and Output Buffer (OBF) are build up as a double buffer structure. One half of this double buffer structure is accessible by the Host (IBF Host / OBF Host), while the other half (IBF Shadow / OBF Shadow) is accessed by the Message Handler for data transfers between IBF / OBF and Message RAM.

4.2.1 Data Transfer from Input Buffer to Message RAM

To configure / update a message buffer in the Message RAM, the Host has to write the data to WRDSn and the header to WRHS1…3. The specific action is selected by configuring the Input Buffer Command Mask IBCM.

If the configured payload length of a static transmit buffer is shorter than the payload length configured for the static segment by **MHDC.SFDL[6:0]**, the CC generates padding bytes to ensure that frames have proper physical length. The padding pattern is logical zero.

Note: In case of an odd payload length (**PLC** = 1,3,5,...) the application has to write zero to the last 16 bit of the message buffers data section to ensure that the padding pattern is all zero.

When the Host writes the number of the target message buffer in the Message RAM to **IBCR.IBRH[6:0]**, IBF Host and IBF Shadow are swapped (see Figure 2).

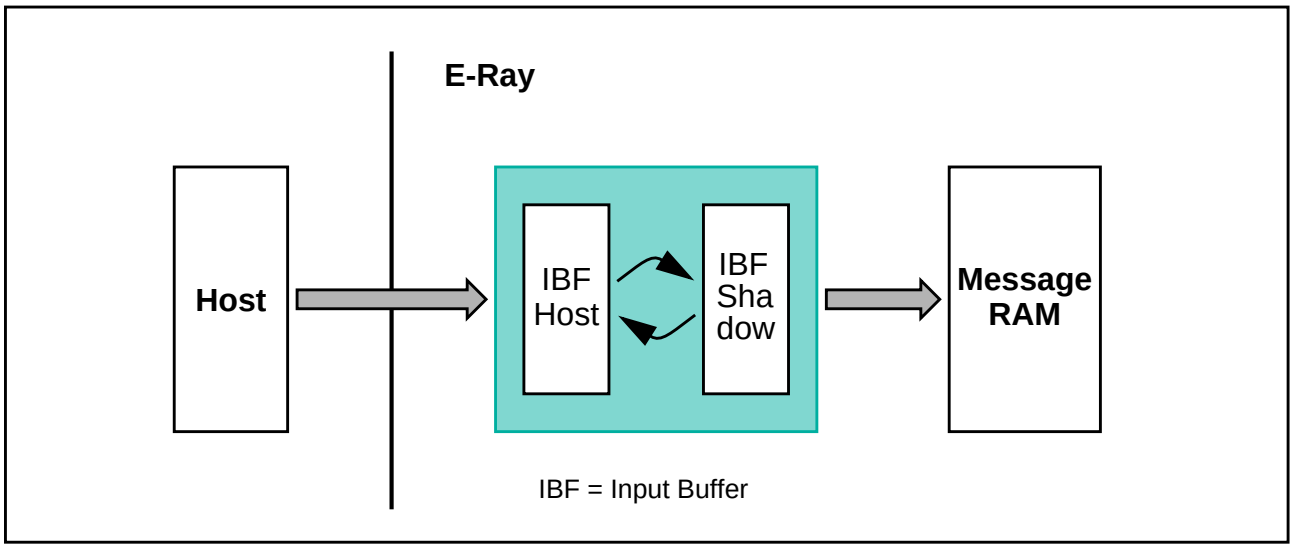

Figure 2: Double buffer structure Input Buffer

In addition the bits in the IBCM and IBCR registers are also swapped to keep them attached to the respective IBF section (see Figure 3).

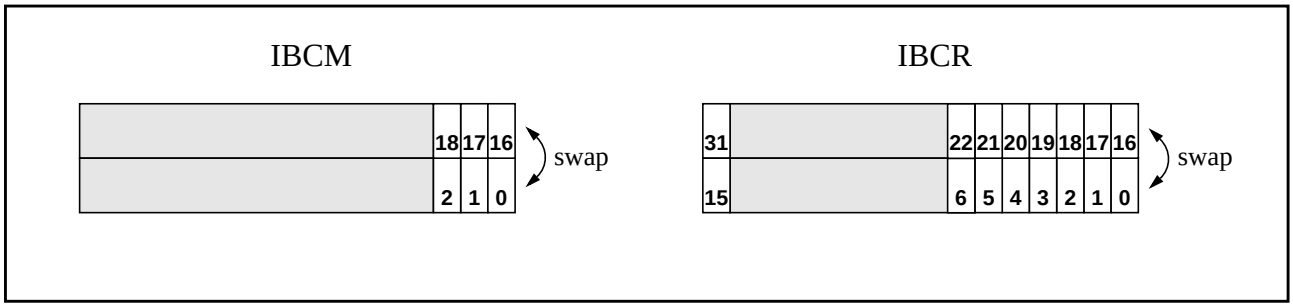

Figure 3: Swapping of IBCM and IBCR bits

With this write operation bit **IBCR.IBSYS** is set to ’1’. The Message Handler then starts to transfer the contents of IBF Shadow to the message buffer in the Message RAM selected by **IBCR.IBRS[6:0]**.

While the Message Handler transfers the data from IBF Shadow to the target message buffer in the Message RAM, the Host may write the next message to IBF Host. After the transfer between IBF Shadow and the Message RAM has completed, bit **IBCR.IBSYS** is set back to ’0’ and the next transfer to the Message RAM may be started by the Host by writing the respective target message buffer number to **IBCR.IBRH[6:0]**.

If a write access to **IBCR.IBRH[6:0]** occurs while **IBCR.IBSYS** is ’1’, **IBCR.IBSYH** is set to ’1’. After completion of the ongoing data transfer from IBF Shadow to the Message RAM, IBF Host and IBF Shadow are swapped, **IBCR.IBSYH** is reset to ’0’, **IBCR.IBSYS** remains set to ’1’, and the next transfer to the Message RAM is started. In addition the message buffer numbers stored under **IBCR.IBRH[6:0]** and **IBCR.IBRS[6:0]** and the command mask flags are also swapped.

Example of a 8/16/32-bit Host access sequence:

Configure / update n-th message buffer via IBF

- Wait until **IBCR.IBSYH** is reset
- Write data section to WRDSn
- Write header section to WRHS1…3
- Write Command Mask: write **IBCM.STXRH**, **IBCM.LDSH**, **IBCM.LHSH**
- Demand data transfer to target message buffer: write **IBCR.IBRH[6:0]**

Configure / update (n+1)th message buffer via IBF

- Wait until **IBCR.IBSYH** is reset
- Write data section to WRDSn
- Write header section to WRHS1…3
- Write Command Mask: write **IBCM.STXRH**, **IBCM.LDSH**, **IBCM.LHSH**
- Demand data transfer to target message buffer: write **IBCR.IBRH[6:0]**

…

Note: Any write access to IBF while **IBCR.IBSYH** is ’1’ will set error flag **EIR.IIBA** to ’1’. In this case the write access has no effect.

Pos.	Access	Bit	Function
18	r	STXRS	Set Transmission Request Shadow ongoing or finished
17	r	LDSS	Load Data Section Shadow ongoing or finished
16	r	LHSS	Load Header Section Shadow ongoing or finished
2	r/w	STXRH	Set Transmission Request Host
1	r/w	LDSH	Load Data Section Host
0	r/w	LHSH	Load Header Section Host

Table 1: Assignment of IBCM bits

Pos.	Access	Bit	Function
31	r	IBSYS	IBF Busy Shadow, signals ongoing transfer from IBF Shadow to Message RAM
22…16	r	IBRS[6:0]	IBF Request Shadow, number of message buffer currently / lately updated
15	r	IBSYH	IBF Busy Host, transfer request pending for message buffer referenced by IBRH[6:0]
6…0	r/w	IBRH[6:0]	IBF Request Host, number of message buffer to be updated next

Table 2: Assignment of IBCR bits

4.2.2 Data Transfer from Message RAM to Output Buffer

To read a message buffer from the Message RAM, the Host has to write to register OBCR to trigger the data transfer as configured in OBCM. After the transfer has completed, the Host can read the transferred data from RDDSn, RDHS1…3, and MBS.

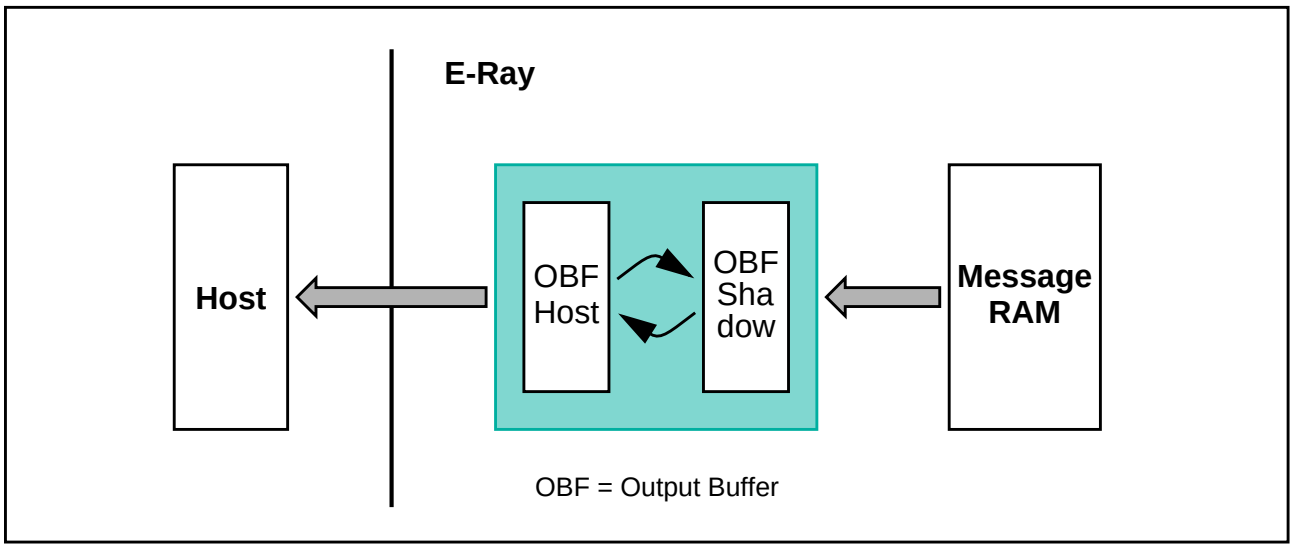

Figure 4: Double buffer structure Output Buffer

OBF Host and OBF Shadow as well as bits **OBCM.RHSS**, **OBCM.RDSS**, **OBCM.RHSH**, **OBCM.RDSH** and bits **OBCR.OBRS[6:0]**, **OBCR.OBRH[6:0]** are swapped under control of bits **OBCR.VIEW** and **OBCR.REQ**.

Setting bit OBCR.REQ:

Writing bit **OBCR.REQ** to ’1’ copies bits **OBCM.RHSS**, **OBCM.RDSS** and bits **OBCR.OBRS[6:0]** to an internal storage (see Figure 5).

After setting **OBCR.REQ** to ’1’, **OBCR.OBSYS** is set to ’1’, and the transfer of the message buffer selected by **OBCR.OBRS[6:0]** from the Message RAM to OBF Shadow is started. After the transfer between the Message RAM and OBF Shadow has completed, the **OBCR.OBSYS** bit is set back to ’0’. Bits **OBCR.REQ** and **OBCR.VIEW** can only be set to ’1’ while **OBCR.OBSYS** is ’0’.

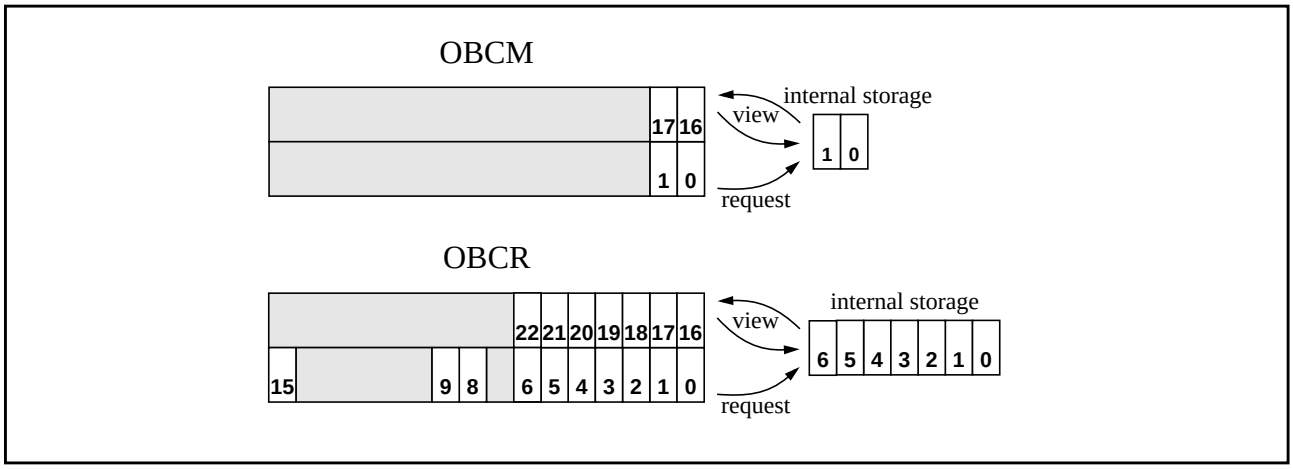

Figure 5: Swapping of OBCM and OBCR bits

Setting bit OBCR.VIEW:

OBF Host and OBF Shadow are swapped by setting bit **OBCR.VIEW** to ’1’ while bit **OBCR.OBSYS** is ’0’ (see Figure 4).

In addition bits **OBCR.OBRH[6:0]** and bits **OBCM.RHSH**, **OBCM.RDSH** are swapped with the registers internal storage thus assuring that the message buffer number stored in **OBCR.OBRH[6:0]** and the mask configuration stored in **OBCM.RHSH**, **OBCM.RDSH** matches the transferred data stored in OBF Host (see Figure 5).

Now the Host can read the transferred message buffer from OBF Host while the Message Handler may transfer the next message from the Message RAM to OBF Shadow.

If bits **REQ** and **VIEW** are set to ’1’ with the same write access while **OBSYS** is ’0’, **OBSYS** is automatically set to ’1’ and OBF Shadow and OBF Host are swapped. Additionally mask bits **OBCM.RDSH** and **OBCM.RHSH** are swapped with the registers internal storage to keep them attached to the respective Output Buffer transfer.

Afterwards **OBRS[6:0]** is copied to the register internal storage, mask bits **OBCM.RDSS** and **OBCM.RHSS** are copied to register OBCM internal storage, and the transfer of the selected message buffer from the Message RAM to OBF Shadow is started. While the transfer is ongoing the Host can read the message buffer transferred by the previous transfer from OBF Host. When the current transfer between Message RAM and OBF Shadow has completed, this is signalled by setting **OBSYS** back to ’0’.

Example of an 8/16/32-bit Host access to a single message buffer:

If a single message buffer has to be read out, two separate write accesses to **OBCR.REQ** and **OBCR.VIEW** are necessary:

- Wait until **OBCR.OBSYS** is reset
- Write Output Buffer Command Mask **OBCM.RHSS**, **OBCM.RDSS**
- Request transfer of message buffer to OBF Shadow by writing **OBCR.OBRS[6:0]** and **OBCR.REQ** (in case of and 8-bit Host interface, **OBCR.OBRS[6:0]** has to be written **before OBCR.REQ**).
- Wait until **OBCR.OBSYS** is reset
- Toggle OBF Shadow and OBF Host by writing **OBCR.VIEW** = ’1’
- Read out transferred message buffer by reading RDDSn, RDHS1…3, and MBS

Example of an 8/16/32-bit Host access sequence:

Request transfer of 1st message buffer to OBF Shadow

- Wait until **OBCR.OBSYS** is reset
- Write Output Buffer Command Mask **OBCM.RHSS**, **OBCM.RDSS** for 1st message buffer
- Request transfer of 1st message buffer to OBF Shadow by writing **OBCR.OBRS[6:0]** and **OBCR.REQ** (in case of an 8-bit Host interface, **OBCR.OBRS[6:0]** has to be written **before OBCR.REQ**).

Toggle OBF Shadow and OBF Host to read out 1st transferred message buffer and request transfer of 2nd message buffer:

- Wait until **OBCR.OBSYS** is reset
- Write Output Buffer Command Mask **OBCM.RHSS**, **OBCM.RDSS** for 2nd message buffer
- Toggle OBF Shadow and OBF Host and start transfer of 2nd message buffer to OBF Shadow simultaneously by writing **OBCR.OBRS[6:0]** of 2nd message buffer, **OBCR.REQ**, and **OBCR.VIEW** (in case of and 8-bit Host interface, **OBCR.OBRS[6:0]** has to be written **before OBCR.REQ** and **OBCR.VIEW**).
- Read out 1st transferred message buffer by reading RDDSn, RDHS1…3, and MBS

…

Demand access to last requested message buffer without request of another message buffer:

- Wait until **OBCR.OBSYS** is reset
- Demand access to last transferred message buffer by writing **OBCR.VIEW**
- Read out last transferred message buffer by reading RDDSn, RDHS1…3, and MBS

Pos.	Access	Bit	Function
17	r	RDSH	Data Section available for Host access
16	r	RHSH	Header Section available for Host access
1	r/w	RDSS	Read Data Section Shadow
0	r/w	RHSS	Read Header Section Shadow

Table 3: Assignment of OBCM bits

Pos.	Access	Bit	Function
22…16	r	OBRH[6:0]	OBF Request Host, number of message buffer available for Host access
15	r	OBSYS	OBF Busy Shadow, signals ongoing transfer from Message RAM to OBF Shadow
9	r/w	REQ	Request Transfer from Message RAM to OBF Shadow
8	r/w	VIEW	View OBF Shadow, swap OBF Shadow and OBF Host
6…0	r/w	OBRS[6:0]	OBF Request Shadow, number of message buffer for next request

Table 4: Assignment of OBCR bits

5. Reconfiguration of Message Buffers

In case that an application needs to operate with more than 128 different messages, static and dynamic message buffers may be reconfigured during FlexRay operation. This is done by updating the header section of the respective message buffer via Input Buffer registers WRHS1...3.

Reconfiguration has to be enabled via control bits **MRC.SEC[1:0]** in the Message RAM Configuration register.

If a message buffer has not been transmitted / updated from a received frame before reconfiguration starts, the respective message is lost.

The point in time when a reconfigured message buffer is ready for transmission / reception according to the reconfigured frame ID depends on the actual state of the slot counter when the update of the header section has completed. Therefore it may happen that a reconfigured message buffer is not transmitted / updated from a received frame in the cycle where it was reconfigured.
The Message RAM is scanned according to table 5 below:

Start of Scan in Slot	Scan for Slots
1	2…15, 1 (next cycle)
8	16…23, 1 (next cycle)
16	24…31, 1 (next cycle)
24	32…39, 1 (next cycle)
…	…

Table 5: Scan of Message RAM

A Message RAM scan is terminated with the start of NIT regardless whether it has completed or not. The scan of the Message RAM for slots 2 to 15 starts at the beginning of slot 1 of the actual cycle. The scan of the Message RAM for slot 1 is done in the cycle before by checking in parallel to each scan of the Message RAM whether there is a message buffer configured for slot 1 of the next cycle.

The number of the first dynamic message buffer is configured by **MRC.FDB[7:0]**. In case a Message RAM scan starts while the CC is in dynamic segment, the scan starts with the message buffer number configured by **MRC.FDB[7:0]**.

In case a message buffer should be reconfigured to be used in slot 1 of the next cycle, the following has to be considered:

- If the message buffer to be reconfigured for slot 1 is part of the "Static Buffers", it will only be found if it is reconfigured before the last Message RAM scan in the static segment of the actual cycle evaluates this message buffer.
- If the message buffer to be reconfigured for slot 1 is part of the "Static + Dynamic Buffers", it will be found if it is reconfigured before the last Message RAM scan in the actual cycle evaluates this message buffer.
- The start of NIT terminates the Message RAM scan. In case the Message RAM scan has not evaluated the reconfigured message buffer until this point in time, the message buffer will not be considered for the next cycle.

Note: Reconfiguration of message buffers may lead to the loss of messages and therefore has to be

used very carefully. In worst case (reconfiguration in consecutive cycles) it may happen that a message buffer is never transmitted / updated from a received frame.

5.1 Temporary Unlocking of Header Section

Several message buffers are protected from reconfiguration outside POC state DEFAULT_CONFIG or CONFIG. This message buffers belongs for example to the Secured Buffer section (see Ref. 2, chapter 4.7.1 Message RAM Configuration (MRC)), to the key slot or single slot ID, or to the FIFO message buffers.

If such a protected message buffer needs to be reconfigured, the application should allow the Host to bring the CC to POC state CONFIG for reconfiguration the protected message buffer. Afterwards, the node can be re-integrated to the FlexRay cluster.

A message buffer in the header section of a locked message buffer can also be reconfigured by a transfer from the Input Buffer to the locked buffer Header Section. For this transfer, the write access to the IBCR (specifying the message buffer number) must be immediately preceded by the unlock sequence normally used to leave CONFIG state (see Ref. 2, chapter 4.3.3 Lock Register (LCK)).

For that single transfer the respective message buffer is unlocked, regardless whether it belongs to the FIFO or whether its locking is controlled by **MRC.SEC[1:0]**, and will be updated with new data.

6. Application Hints

6.1 Slot Multiplexing in Dynamic Segment

If a dynamic slot shall be multiplexed in a way that frames shall be received and transmitted in or from that slot depending on the cycle counter, the transmit buffers must have a lower buffer number (e.g. N) that the receive buffer (e.g. N+1).

The reason is that the Message Handler search algorithm starts at buffer 0 and ends at last configured buffer and that the search algorithm selects the first message buffer with matching cycle counter, regardless whether it is a receive or transmit message buffer, and independently of the existence of further receive or transmit message buffers for a dedicated slot.

For example, the message from dynamic slot 10 shall be stored in every cycle to a message buffer. In every 8th cycle a message shall be send from another message buffer in dynamic slot 10.

Then the transmit message buffer which has to send a message in every 8th cycle must have the lower message buffer number, e.g. message buffer number 12. The receive message buffer for slot number 10 must have a higher message buffer number, at least message buffer number 13.

6.2 Padding Pattern in Static Segment

The payload length field of a message buffer (**WRHS2.PLC[6:0]**) configures the payload length in 2-byte words. If the configured payload length of a static transmit buffer is shorter than the payload length configured for the static segment my **MHDC.SFDL[6:0]**, the CC generates padding bytes to ensure that frames have proper physical length. The padding pattern is logical zero.

Note: The Header-CRC has to be calculated with the value of the programmed static frame payload length (**MHDC.SFDL[6:0]**), regardless of the value of the payload length configured (**WRHS2.PLC[6:0]**) for the static frame.

In case of an odd payload (PLC = 1,3,5,...) the application has to write zero to the last 16 bit of the corresponding message buffer data section to ensure that the padding pattern are all zero.

For example, if the Host writes to the Input buffer RAM with 16-bit or 8-bit accesses, the application has to ensure that all 32 bit of the Write Data Section are written with the payload or with zero.

In case of 8-bit access and a payload configured PLC=3, the application has to perform eight write accesses to the Data Section, whereas the write access seven and eight shall write 00h to the Data Section.

In case of 16-bit access and a payload configured PLC=3, the application has to perform four write accesses to the Data Section, whereas the fourth write access shall write 0000h to the Data Section.

7. List of Tables

8. List of Figures