BOSCH
Invented for life

M_CAN

Modular CAN IP-module

Transmission Handling

Application Note M_CAN_AN002

Document Revision 2.1
29.06.2018

Robert Bosch GmbH
Automotive Electronics

LEGAL NOTICE

Revision History

Version	Date	Remark
1.0	08.06.2011	First version for M_CAN 2.0
2.0	28.09.2015	New revision for M_CAN Revision 3.1.0 - 3.2.1
2.1	29.06.2018	Source Code updated

Conventions

The following conventions are used within this document:

Register names	**TXBC, TXESC**
Names of files and directories	`directoryname/filename`
Source code/function names	`m_can_tx_fifo_queue_msg_write()`

References

This document refers to the following documents:

Ref	Author	Title
[1]	AE/PJ-SCI	M_CAN User's Manual
[2]	AE/PJ-SCI	M_CAN System Integration Guide

Terms and Abbreviations

This document uses the following terms and abbreviations:

Term	Meaning
BRP	Baud Rate Prescaler
CAN	Controller Area Network
CRC	Cyclic Redundancy Check
DLC	Data Length Code

Table of Contents

1 Target

This application note describes transmit message handling in the **M_CAN versions 3.1.0 up to 3.2.x**

The topics included are:

- Tx Buffer configuration – Tx FIFO, Tx Queue and dedicated Tx Buffers
- Tx Event FIFO configuration and Tx Event handling
- Cancellation of a transmit request

Software examples delivered with this application note are only for illustration purposes. Use the examples on own risk.

2 Tx Buffers

Configuration

The M_CAN supports a maximum of 32 Tx Buffers. They can be configured as

- Tx FIFO
- Tx Queue
- Dedicated Tx Buffers
- Combination of dedicated Tx Buffers with Tx FIFO
- Combination of dedicated Tx Buffers with Tx Queue

Table 1 shows all the possible Tx Buffer combinations.

Table 1: Tx Buffer combinations

Dedicated Tx Buffers	Tx Queue	Tx FIFO	Configurable
-	-	-	Yes
-	-	X	Yes
-	X	-	Yes
-	X	X	No
X	-	-	Yes
X	-	X	Yes
X	X	-	Yes
X	X	X	No

Each message to be transmitted is stored in one Tx Buffer element. The M_CAN User's manual describes the structure of a Tx Buffer element. The size of the Tx Buffer elements can be configured via **TXESC.TBDS**. It defines the number of data field bytes for all Tx Buffer elements. A Tx Buffer element has the size:

2 * 4 byte header information + data field size configured in **TXESC.TBDS**

Example for a 64 byte data field size: (2 * 4 + 64) = 72 bytes or 18 words.

Figure 1 shows the section of the Message RAM with the Tx Buffers and its start address.

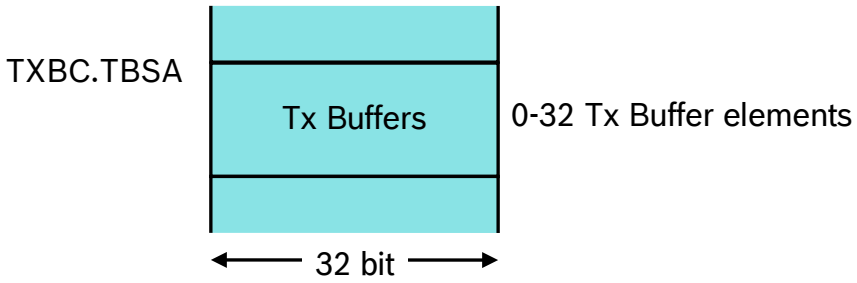

Figure 1: Section of Tx Buffers in the Message RAM

The memory requirement for the Tx Buffers depends on the configured Tx Buffer element size **TXESC.TBDS.** Configuration of the Tx Buffers is done via register **TXBC**.

In case that the Tx Buffers section is shared by dedicated Tx Buffers and a Tx FIFO/Queue, the dedicated Tx Buffers start at the beginning of the Tx Buffers section followed by the Buffers assigned to the Tx FIFO or Tx Queue. The Tx handler distinguishes between dedicated Tx Buffers and Tx FIFO/Queue by evaluating the Tx Buffer configuration **TXBC.TFQS** and **TXBC.NDTB.** The Tx handler controls the message transfer from the external Message RAM to the CAN Core.

The CAN mode for transmission (Classic CAN or CAN FD) can be configured separately for each Tx Buffer element. Based on these configurations, the transmitted frame format can be:

- Classic CAN frame
- CAN FD frame with Bit Rate Switching
- CAN FD frame without Bit Rate Switching

Message Transmission

Two steps are required to transmit a message:

1. Write the message into a Tx Buffer in the Message RAM. This step may require to check if there is room for a new message. (eg. Read the Tx FIFO/Queue status **TXFQS** or Tx Request Pending **TXBRP** register).

2. Add request for this Tx Buffer by setting the appropriate bit in **TXBAR.AR**.

The requested messages arbitrate internally and externally with the messages on the CAN bus and are sent out according to their Message ID. The message with the lowest Message ID is transmitted first.

Internal Arbitration

Priority of messages during internal arbitration for different Tx Buffer configurations:

- **Dedicated Tx Buffers**: The message with lowest ID wins the arbitration
- **Tx FIFO**: The first/oldest message in the FIFO wins the arbitration
- **Tx Queue**: Message in the queue with the lowest message ID wins the arbitration
- **Dedicated Tx Buffers and Tx FIFO**: The first/oldest element in FIFO and all the elements in dedicated Buffers are checked, and the message with lowest ID among them wins the arbitration.
- **Dedicated Tx Buffers and Tx Queue**: All the elements in Queue and all the elements in dedicated Buffers are checked, and the message with lowest ID among them wins the arbitration.

In case multiple messages with same message ID are present in dedicated Buffers or Tx Queue, then the message with the lowest Buffer index will be selected for arbitration. However, it is better to use Tx FIFO in such cases as Tx FIFO enables transmission of messages with same message ID in the order these messages had been written to the Tx FIFO.

Interrupt Flags

Interrupt flag **IR.TC** can be used to inform the CPU that a transmission is completed. Interrupt flag **IR.TCF** signals that a transmission cancellation has finished.

Summary of registers used for Tx Buffer operation

- **TXBC**: Tx Buffer Configuration
- **TXFQS**: Tx FIFO/Queue Status
- **TXESC**: Tx Buffer Element Size Configuration
- **TXBAR**: Tx Buffer Add Request
- **TXBCR**: Tx Buffer Cancellation request
- **TXBRP**: Tx Request Pending
- **TXBTO**: Tx Buffer Transmission Occurred
- **TXBCF**: Tx Buffer Cancellation Finished

Software Example

Table 1 lists C functions that demonstrate Tx Buffer operation. The functions are provided with this application note.

Table 1: C functions that demonstrate Tx Buffer operation

Name: File:	`m_can_tx_buffer_init(..)` `m_can/m_can.c`
Description:	Function configures the Tx Buffer section. Tx Buffer element size and the Tx Buffer combination is configured. M_CAN has to be in configuration change enable mode when this function is called.
Name: File:	`m_can_tx_is_tx_buffer_req_pending(..)` `m_can/m_can.c`
Description:	Function checks if a Tx Buffer has a pending Tx request.
Name: File:	`m_can_tx_write_msg_to_tx_buffer(..)` `m_can/m_can.c`
Description:	Function copies transmit message to Tx Buffer - NO CHECK is performed if the buffer has already a pending Tx request and NO transmission is requested.
Name: File:	`m_can_tx_msg_request_tx(..)` `m_can/m_can.c`
Description:	Function requests transmission of Tx message in Tx Buffer - NO CHECK is performed if the buffer has already a pending Tx request.

Name:	`m_can_tx_write_msg_to_tx_buffer_and_request_tx(..)`
File:	`m_can/m_can.c`
Description:	Function copies transmit message to Tx Buffer and requests transmission.
Name:	`m_can_tx_dedicated_msg_transmit(..)`
File:	`m_can/m_can.c`
Description:	Function copies a transmit message to a dedicated Tx Buffer and requests it. It CHECKs if the buffer is free.
Name:	`m_can_tx_fifo_queue_msg_transmit(..)`
File:	`m_can/m_can.c`
Description:	Function copies transmit message into FIFO/Queue and requests transmission.
Name:	`m_can_tx_fifo_get_num_of_free_elems(..)`
File:	`m_can/m_can.c`
Description:	Function gets the number of Free Elements in Tx FIFO.
Name:	`m_can_tx_buffer_transmission_occured(..)`
File:	`m_can/m_can.c`
Description:	Function checks if a Tx buffer transmission has occurred or not.

3 Tx Event FIFO

Overview

By using a Tx Event FIFO the Host (CPU) gets the following information about sent messages:

- in which order were the messages transmitted
- for each message the local time when the frame was transmitted

Configuration

The M_CAN provides a Tx Event FIFO. The use of this Tx Event FIFO is optional. After the M_CAN has transmitted a message successfully on the CAN bus, it can store the Message ID and timestamp in a Tx Event FIFO element. A Tx Event FIFO element is a data structure that stores information about transmitted messages. The M_CAN User's manual describes the structure of a Tx Event FIFO element. The Tx Event FIFO can be configured via the register **TXEFC** (Tx Event FIFO Configuration). The FIFO can store up to a maximum of 32 elements.

Figure 2 shows a section of the Message RAM where the Tx Event FIFO elements are stored.

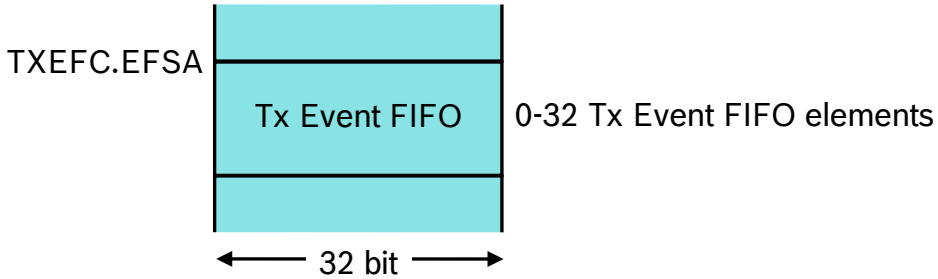

Figure 2: Section of Tx Event FIFO in the Message RAM.

The address of a Tx Event FIFO element in the Message RAM is given by the formula:

Message_RAM_base_address + **TXEFC_EFSA** + **TXEFS.EFGI** *Tx_event_fifo_element_size_in_bytes

Example (byte address of an element at index 7): 0x100000 + 0xE80 + 7 * (2 * 4). The size of a Tx Event FIFO element has a constant value of 2 words (2*4 = 8 bytes).

Operation

- Status information about the Tx Event FIFO can be obtained from the register **TXEFS** (Tx Event FIFO Status)
- To link a Tx event to a Tx Event FIFO element, the Message Marker from the transmitted Tx Buffer is copied into the Tx Event FIFO element.
- Events are stored in Tx Event FIFO only if EFC bit (Store Tx events) in the Tx Buffer element is '1'.

- When the Tx Event FIFO is full, no further elements are written to the Tx Event FIFO until at least one element has been read out and the Tx Event FIFO Get Index has been incremented. In case a Tx event occurs while the Tx Event FIFO is full, this event is discarded. To avoid a Tx Event FIFO overflow, the Tx Event FIFO watermark can be used.
- After the host has read an element or a sequence of elements from the Tx Event FIFO it has to acknowledge the read. Therefore it writes the index of the last element read from Tx Event FIFO to **TXEFA.EFAI** (Event FIFO Acknowledge Index). The acknowledged FIFO elements can now be overwritten by new ones.

Interrupt Flags

The following interrupt flags exist, to inform the CPU when the status of the Tx Event FIFO changes:

- **IR.TEFW**: Tx Event FIFO Watermark Reached
  When the Tx Event FIFO fill level reaches the Tx Event FIFO watermark configured by **TXEFC.EFWM**, this interrupt flag is set.
- **IR.TEFF**: Tx Event FIFO Full
  When the Tx Event FIFO is full, this interrupt flag is set.
- **IR.TEFL**: Tx Event FIFO Element Lost
  This interrupt flag is set either when a Tx Event FIFO element is lost or after a write attempt to Tx Event FIFO of zero size.
- **IR.TEFN**: Tx Event FIFO New Entry
  This interrupt flag is set when there is a new element in the Tx Event FIFO.

Summary of Registers for Operation and Configuration

- **TXEFC**: Tx Event FIFO Configuration
- **TXEFS**: Tx Event FIFO Status
- **TXEFA**: Tx Event FIFO Acknowledge

Time Stamp Generation

Each Tx FIFO Element has a timestamp information, **TXTS.** This contains the value of the timestamp counter captured at the start of a frame transmission.

Register **TSCC** (Timestamp Counter Configuration) configures the timestamp counter and the counter value is readable via **TSCV.TSC**.

- **For Classical CAN**: M_CAN has a 16-bit wrap-around counter for timestamp generation. A prescaler **TSCC.TCP** can be configured to clock the counter in **multiples of CAN bit times**.
- **For CAN FD**: An external counter is recommended for timestamp generation. By programing bit **TSCC.TSS** an external 16-bit timestamp can be used.
  Reason: The internal counter in the M_CAN counts “CAN bit times”. In CAN FD with bit rate switching, the bit times in data phase and arbitration phase are different. Consequently if the internal counter is used, the “time” would progress faster in the data phase.

Software Example

Table 2 lists C functions that demonstrate Tx Event FIFO operations. The functions are provided with this application note.

Table 2: C functions that demonstrate Tx Event FIFO operation.

Name: File:	`m_can_tx_event_fifo_init(..)` `m_can/m_can.c`
Description:	Function configures the Tx Event FIFO via register **TXEFC**. M_CAN has to be in configuration change enable mode when this function is called.
Name: File:	`m_can_tx_event_fifo_copy_element_to_tx_event_list(..)` `m_can/m_can.c`
Description:	Function copies all the elements in the Tx Event FIFO from the Tx Event FIFO section of the Message RAM. This function serves as an interrupt service routine for the interrupt **IR.TEFN.**
Name: File:	`m_can_copy_tx_event_element_from_msg_ram(..)` `m_can/m_can.c`
Description:	Function that actually copies a Tx Event FIFO element from a calculated address in the Message RAM.
Name: File:	`copy_event_element_from_msg_ram_to_tx_event_list(..)` `m_can/m_can_helper_func.c`
Description:	Function copies all the Tx Event FIFO elements into a data structure defined in the software.
Name: File:	`print_tx_event_fifo_element(..)` `m_can/m_can_helper_func.c`
Description:	Function prints the content of one Tx Event FIFO element.
Name: File:	`m_can_timestampcounter_and_timeoutcounter_init(..)` `m_can/m_can.c`
Description:	Function configures the Timestamp usage and Timeout usage.

4 Transmit Cancellation

The M_CAN supports transmit cancellation.

- **Tx Buffer, Tx Queue:** To cancel a requested transmission from a dedicated Tx Buffer or a Tx Queue Buffer, the Host has to write a '1' to the corresponding bit position (=number of Tx Buffer) of register **TXBCR** (Tx Buffer Cancellation Request).
- **Tx FIFO:** Transmit cancellation is not intended for Tx FIFO operation. This means the host is **not allowed** to cancel individual transmission request from the Tx FIFO. But the host is **allowed** to cancel all requested transmissions in a Tx FIFO at once, i.e. with a single write to **TXBCR**.

When a cancellation is successful, the M_CAN sets the interrupt flag **IR.TCF** (Transmission Cancellation Finished) and the corresponding bit of register **TXBCF** (Tx Buffer Cancellation Finished) to '1'.

A transmit cancellation request resets the corresponding bit in **TXBRP** and then sets the corresponding bit in **TXBCF**. In case a transmit cancellation is requested while a transmission from a Tx Buffer is already ongoing, the corresponding **TXBRP** bit remains set as long as the transmission is in progress.

- If the transmission was successful, the corresponding **TXBTO** and **TXBCF** bits are set.
- If the transmission was not successful, it is not repeated and only the corresponding **TXBCF** bit is set.

When a transmission request for the Tx Buffer referenced by the Get Index is cancelled, the Get Index is incremented to the next Tx Buffer with pending transmission request and the Tx FIFO free level is recalculated. When transmission cancellation is applied to any other Tx Buffer, the Get Index and the FIFO free level remain unchanged.

Summary of registers used for transmit cancellation operation are:

- **TXBCR:** Tx Buffer Cancellation Request
- **TXBCF:** Tx Buffer Cancellation Finished
- **TXBTO:** Tx Buffer transmission Occurred

Software Example

Table 3 lists C functions that demonstrates transmit cancellation. The functions are provided with this application note.

Table 3: C functions that demonstrate transmit cancellation

Name:	`m_can_tx_buffer_request_cancelation(..)`
File:	`m_can/m_can.c`
Description:	Function executes one cancellation request, by writing to the appropriate bit in the register **TXBCR**, corresponding to the Tx Buffer index.
Name:	`m_can_tx_buffer_is_cancelation_finshed(..)`
File:	`m_can/m_can.c`
Description:	Function returns a Boolean depending on the cancellation status of a Tx Buffer.

5 Examples

5.1 Dedicated Tx Buffers

Figure 3 shows the working principle of eight dedicated transmit Buffers. The buffers are configured by setting **TXBC.TFQS** to '0' and **TXBC.NDTB to** '8'. Dedicated Tx Buffers are intended for message transmission under complete control of the Host CPU. Each dedicated Tx Buffer is configured with a specific message ID. If the data section has been updated, a transmission is requested by an Add Request via **TXBAR.ARn.** The requested messages arbitrate externally with messages on the CAN bus, and are sent out according to their message ID (Lowest ID first).

The memory requirements for the dedicated Tx Buffers depends on the Tx Buffer element size. The Tx Buffer element size defines the number of data bytes belonging to a Tx Buffer. The address of a dedicated Tx Buffer in the Message RAM is calculated using the formula:

Message_RAM_base_address + **TXBC.TBSA**
+ Tx_Buffer_Index * Tx_Buffer_element_size

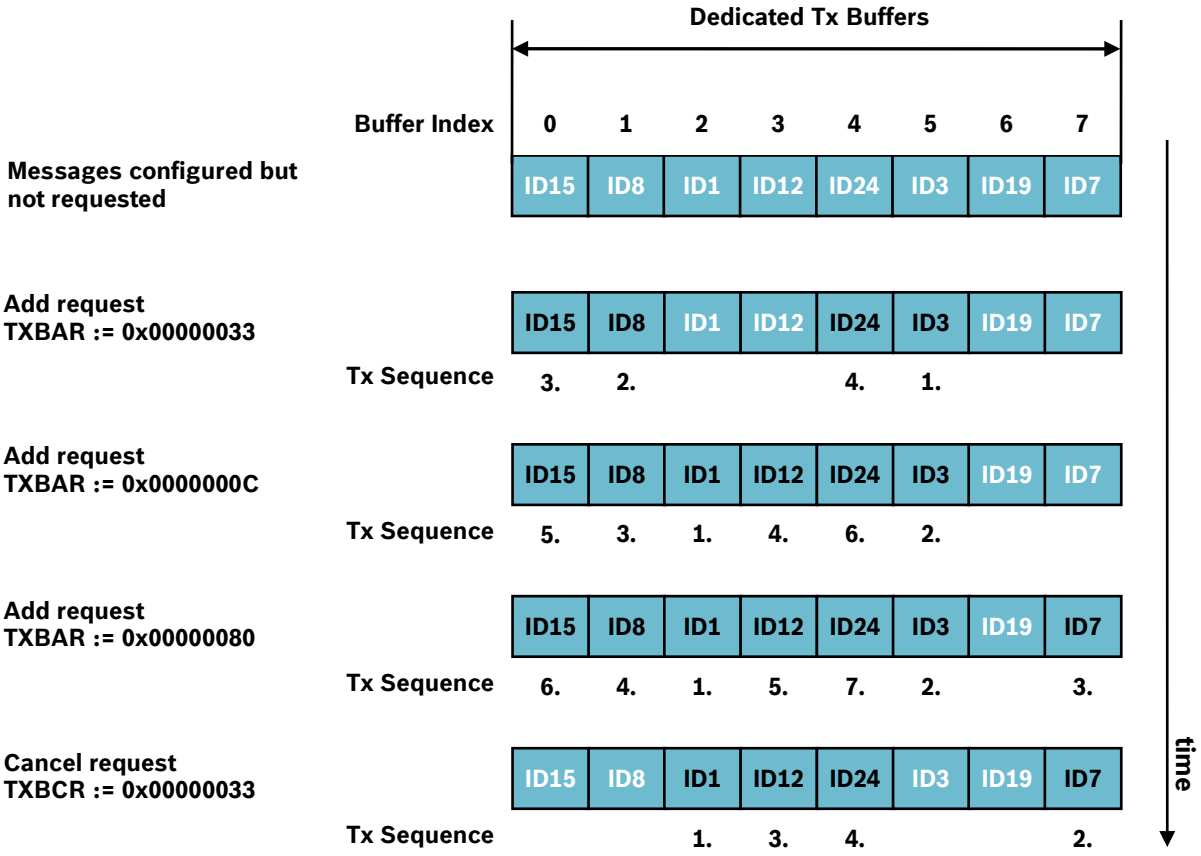

Figure 3: Working principle of Dedicated Tx Buffer

5.2 Tx FIFO

Tx FIFO operation is configured by programming **TXBC.TFQM** to '0'. Messages stored in the Tx FIFO are transmitted starting with the message referenced by the Get Index **TXFQS.TFGI**. After each transmission, the Get Index is incremented cyclically until the Tx FIFO is empty. The Tx FIFO enables transmission of messages with the same message ID from different Tx Buffers in the order these messages had been written to the Tx FIFO. If messages with different IDs are transmitted from the Tx FIFO, the order of transmission is independent of the priority of the respective message IDs because the first/oldest message in the FIFO will be sent out first. The M_CAN calculates the Tx FIFO free level **TXFQS.TFFL** as difference between Get and Put Index. This value indicates the number of available (free) Tx FIFO elements.

Figure 4 shows the working principle of the Tx FIFO and demonstrates how the Get and Put indices are used to access the elements in the Tx FIFO.

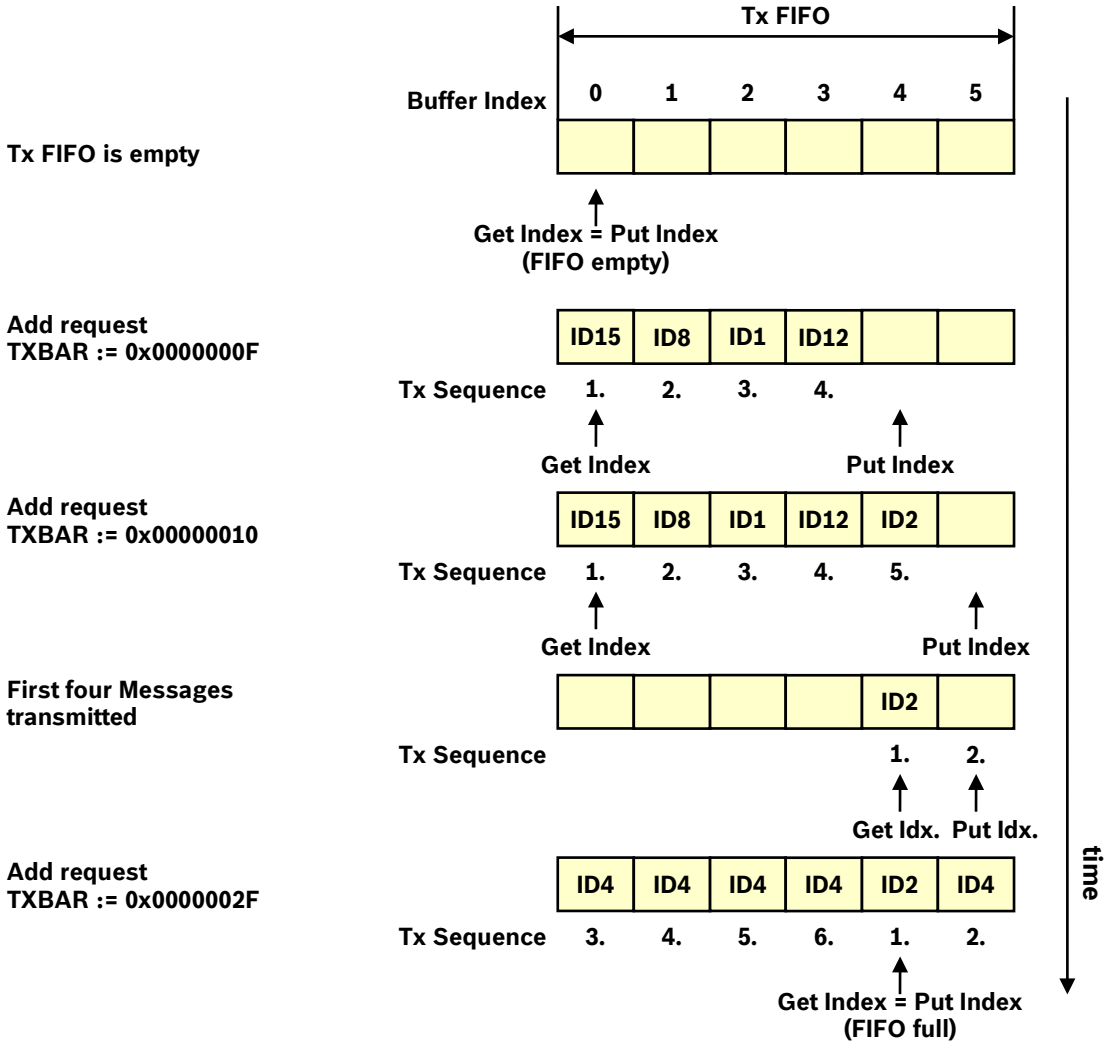

Figure 4: Working principle of Tx FIFO

New transmit messages have to be written to the Tx FIFO starting with the Tx Buffer referenced by the Put Index **TXFQS.TFQPI**. The address of a next free Tx FIFO Buffer in the Message RAM is calculated by the formula:

Message_RAM_base_address + **TXBC.TBSA** + **TXFQS.TFQPI** * Tx_FIFO_element_size

An Add request increments the Put index to the next free Tx FIFO element. When the Put index reaches the Get index, Tx FIFO full (**TXFQS.TFQF** = '1') is signalled. In this case no further messages should be written to the Tx FIFO until the next message has been transmitted and the Get index has been incremented.

When a single message is added to the Tx FIFO, the transmission is requested by writing a '1' to the **TXBAR** bit related to the Tx Buffer referenced by the Tx FIFO's Put index. When multiple (n) messages are added to the Tx FIFO, they are written to n consecutive Tx Buffers starting with the Put index. The transmissions are then requested via **TXBAR**. When transmitting the messages, the Put index is then cyclically incremented by n. The number of requested Tx Buffers should not exceed the number of free Tx Buffers as indicated by the Tx FIFO free level.

5.3 Tx Queue

Tx Queue operation is configured by programming **TXBC.TFQM** to '1'. Messages stored in the Tx Queue are transmitted starting with the message with the lowest ID (highest priority). In case that multiple Queue Buffers are configured with the same message ID, the Queue Buffer with the lowest Buffer number is transmitted first.

In contrast to dedicated Tx Buffers, the message ID is not fixed to a predefined Tx Buffer index. The host CPU has to copy each message completely to the Message RAM. The Tx Queue is recommended when the number of different IDs used for transmission exceeds the maximum of 32 Tx Buffers.

New messages have to be written to the Tx Buffer referenced by the Put Index **TXFQS.TFQPI**. An Add request cyclically increments the Put Index to the next free Tx Buffer. In case that the Tx Queue is full (**TXFQS.TFQF** = '1'), no further message should be written to the Tx Queue until at least one of the requested messages has been sent out or a pending transmission request has been cancelled. The application may use register **TXBRP** instead of the Put Index and may place messages to any Tx Buffer without pending transmission request.

The memory requirements for the Tx Buffer depends on the Tx Buffer element size. The Tx Buffer element size defines the number of data bytes belonging to a Tx Buffer. The address of a next available free Tx Queue Buffer in the Message Ram can be calculated by the formula:

Message_RAM_base_address + **TXBC.TBSA** + **TXFQS.TFQPI** * Tx_Buffer_element_size

Figure 5 shows the operation of a Tx Queue.

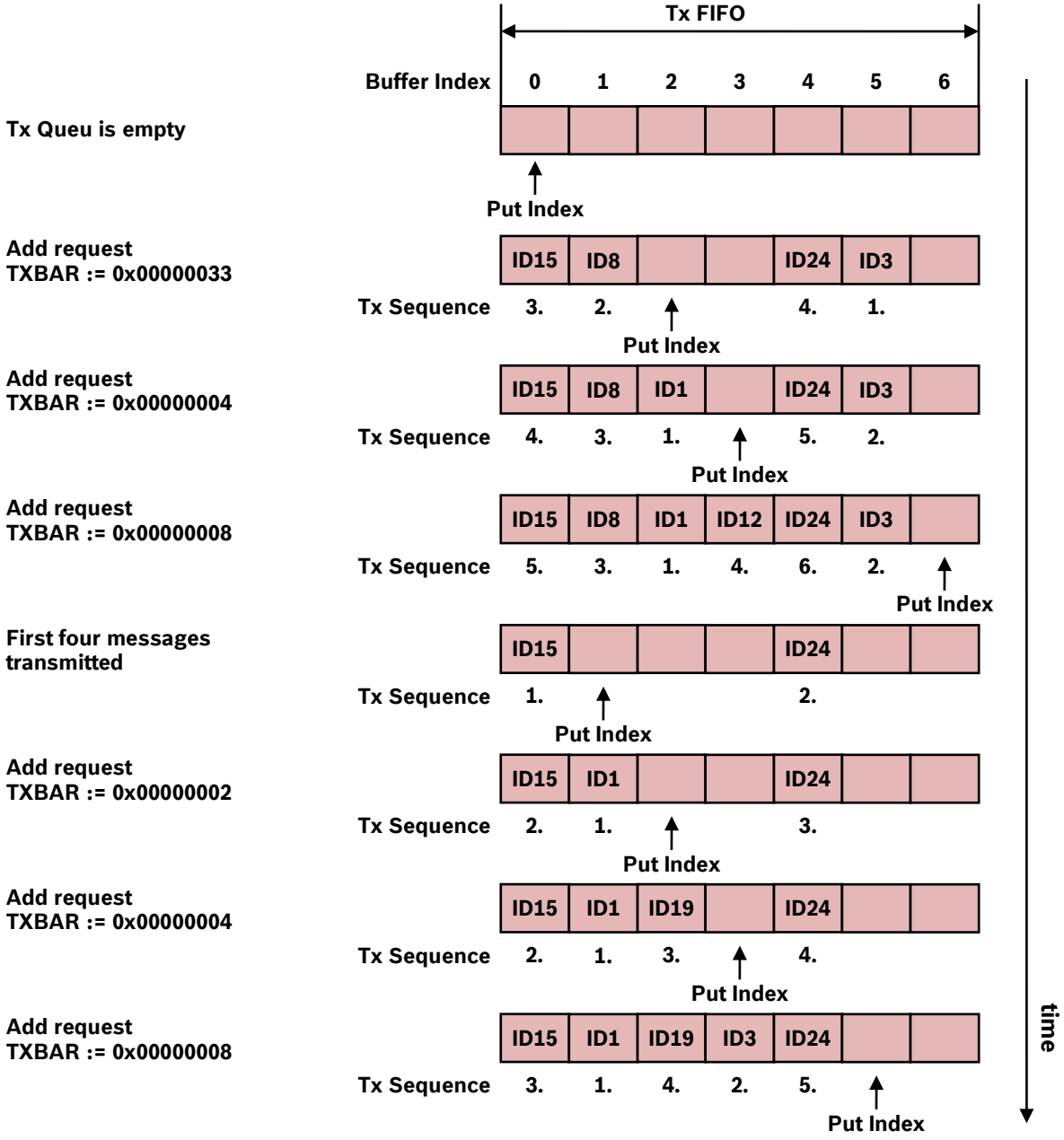

Figure 5: Working principle of Tx Queue

5.4 Dedicated Tx Buffers + Tx FIFO

Figure 6 shows a mixed configuration where the Tx Buffers section in the Message RAM is subdivided into a set of dedicated Tx Buffers and a Tx FIFO. The number of dedicated Tx Buffers is configured via register **TXBC.NDTB**. The number of Tx Buffers assigned to the Tx FIFO is configured via register **TXBC.TFQS**.

The Tx handler scans the dedicated Tx Buffers and the oldest pending Tx FIFO Buffer (referenced by **TXFS.TFGI**). The Buffer with the lowest message ID gets highest priority and is transmitted next.

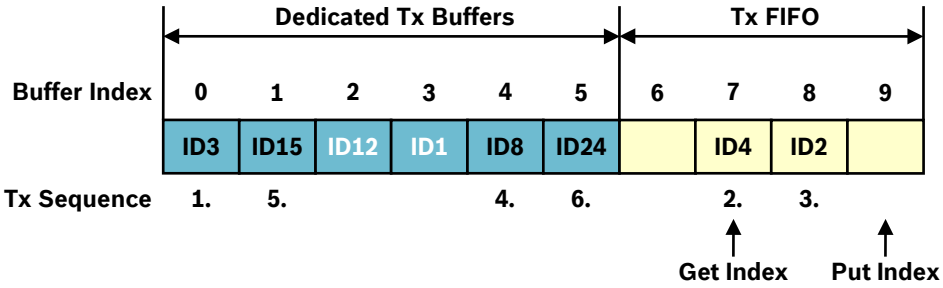

Figure 6: A mixed configuration with Dedicated Tx Buffers and Tx FIFO

5.5 Dedicated Tx Buffers + Tx Queue

Figure 7 shows a mixed configuration where the Tx Buffers section in the Message RAM is subdivided into a set of dedicated Tx Buffers and Tx Queue. The number of dedicated Tx Buffers is configured via register **TXBC.NDTB.** The number of Tx Queue Buffers is configured via register **TXBC.TFQS**.

The Tx handler scans all the Tx Buffers with activated transmission request. The Tx Buffer with the lowest message ID gets the highest priority and is transmitted next.

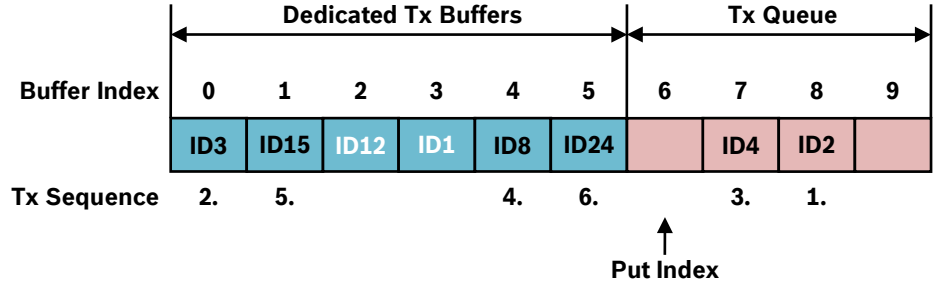

Figure 7: A mixed configuration with Dedicated Tx Buffers and Tx Queue

6 Software Example

The software examples were written for **M_CAN version 3.2.1.**Table 4 lists a few examples and some C functions that are used in the examples provided with this application note.

Table 4: Examples and some basic C functions used in all examples

Name:	`m_can_an002_transmit_messages_tx_fifo()`
File:	`m_can/app_note_002_tx_handling.c`
Description:	The example demonstrates how to configure and use Tx FIFO. Messages transmitted by M_CAN-0 via its Tx FIFO are received by M_CAN-1 in its Rx FIFO0.
Name:	`m_can_an002_transmit_messages_tx_queue_and_ded_tx_buffers()`
File:	`m_can/app_note_002_tx_handling.c`
Description:	The example demonstrates how to configure and use Tx Buffers as a combination of Tx Queue and dedicated Tx Buffers. Messages are transmitted by M_CAN-0 via Tx Queue and dedicated Tx buffers. These messages are received by M_CAN-1 in its Rx FIFO0.
Name:	`m_can_an002_tx_event_fifo_handling()`
File:	`m_can/app_note_002_tx_handling.c`
Description:	The example demonstrates working of a Tx Event FIFO. Two M_CAN nodes participate in the test. M_CAN-0 transmits messages with proper message markers. All these messages are received by M_CAN-1 in its Rx FIFO0. Information about transmitted messages stored in the Tx Event FIFO is displayed and can be identified by their respective message marker.
The example demonstrates:	Tx FIFO configuration, Tx Event FIFO configuration and transmission of message via Tx FIFO.
Name:	`m_can_an002_tx_buffer_cancellation()`
File:	`m_can/app_note_002_tx_handling.c`
Description:	The example demonstrates how a transmission request can be cancelled. Two M_CAN nodes participate in the test. Received messages and the Tx Buffer indices whose cancellation were successful are displayed.
The example demonstrates:	Tx Queue configuration, Dedicated Tx Buffer configuration, transmission of message via Dedicated Tx Buffers and Tx Queue, transmission cancellation.

This application note contains all C-source files that are necessary to compile the examples. The file `_info.txt` contains a short description of each provided source file.

7 List of Tables

8 List of Figures