BOSCH
Invented for life

XS_CAN

Slim CAN XL IP module

Transmission Handling Full Message Mode (FMM)

Application Note XS_CAN_AN002

Document Revision 1.1
27.04.2026

Robert Bosch GmbH
Mobility Electronics

LEGAL NOTICE

© Copyright 2026 by Robert Bosch GmbH and its licensors. All rights reserved.

“Bosch” is a registered trademark of Robert Bosch GmbH.

The content of this document is subject to continuous developments and improvements. All particulars and its use contained in this document are given by BOSCH in good faith.

NO WARRANTIES: TO THE MAXIMUM EXTENT PERMITTED BY LAW, NEITHER THE INTELLECTUAL PROPERTY OWNERS, COPYRIGHT HOLDERS AND CONTRIBUTORS, NOR ANY PERSON, EITHER EXPRESSLY OR IMPLICITLY, WARRANTS ANY ASPECT OF THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO, INCLUDING ANY OUTPUT OR RESULTS OF THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO UNLESS AGREED TO IN WRITING. THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO IS BEING PROVIDED "AS IS", WITHOUT ANY WARRANTY OF ANY TYPE OR NATURE, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND ANY WARRANTY THAT THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO IS FREE FROM DEFECTS.

ASSUMPTION OF RISK: THE RISK OF ANY AND ALL LOSS, DAMAGE, OR UNSATISFACTORY PERFORMANCE OF THIS SPECIFICATION (RESPECTIVELY THE PRODUCTS MAKING USE OF IT IN PART OR AS A WHOLE), SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO RESTS WITH YOU AS THE USER. TO THE MAXIMUM EXTENT PERMITTED BY LAW, NEITHER THE INTELLECTUAL PROPERTY OWNERS, COPYRIGHT HOLDERS AND CONTRIBUTORS, NOR ANY PERSON EITHER EXPRESSLY OR IMPLICITLY, MAKES ANY REPRESENTATION OR WARRANTY REGARDING THE APPROPRIATENESS OF THE USE, OUTPUT, OR RESULTS OF THE USE OF THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO IN TERMS OF ITS CORRECTNESS, ACCURACY, RELIABILITY, BEING CURRENT OR OTHERWISE. NOR DO THEY HAVE ANY OBLIGATION TO CORRECT ERRORS, MAKE CHANGES, SUPPORT THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO, DISTRIBUTE UPDATES, OR PROVIDE NOTIFICATION OF ANY ERROR OR DEFECT, KNOWN OR UNKNOWN. IF YOU RELY UPON THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO, YOU DO SO AT YOUR OWN RISK, AND YOU ASSUME THE RESPONSIBILITY FOR THE RESULTS. SHOULD THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL LOSSES, INCLUDING, BUT NOT LIMITED TO, ANY NECESSARY SERVICING, REPAIR OR CORRECTION OF ANY PROPERTY INVOLVED TO THE MAXIMUM EXTEND PERMITTED BY LAW.

DISCLAIMER: IN NO EVENT, UNLESS REQUIRED BY LAW OR AGREED TO IN WRITING, SHALL THE INTELLECTUAL PROPERTY OWNERS, COPYRIGHT HOLDERS OR ANY PERSON BE LIABLE FOR ANY LOSS, EXPENSE OR DAMAGE, OF ANY TYPE OR NATURE ARISING OUT OF THE USE OF, OR INABILITY TO USE THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO, INCLUDING, BUT NOT LIMITED TO, CLAIMS, SUITS OR CAUSES OF ACTION INVOLVING ALLEGED INFRINGEMENT OF COPYRIGHTS, PATENTS, TRADEMARKS, TRADE SECRETS, OR UNFAIR COMPETITION.

INDEMNIFICATION: TO THE MAXIMUM EXTEND PERMITTED BY LAW YOU AGREE TO INDEMNIFY AND HOLD HARMLESS THE INTELLECTUAL PROPERTY OWNERS, COPYRIGHT HOLDERS AND CONTRIBUTORS, AND EMPLOYEES, AND ANY PERSON FROM AND AGAINST ALL CLAIMS, LIABILITIES, LOSSES, CAUSES OF ACTION, DAMAGES, JUDGMENTS, AND EXPENSES, INCLUDING THE REASONABLE COST OF ATTORNEYS' FEES AND COURT COSTS, FOR INJURIES OR DAMAGES TO THE PERSON OR PROPERTY OF THIRD PARTIES, INCLUDING, WITHOUT LIMITATIONS, CONSEQUENTIAL, DIRECT AND INDIRECT DAMAGES AND ANY ECONOMIC LOSSES, THAT ARISE OUT OF OR IN CONNECTION WITH YOUR USE, MODIFICATION, OR DISTRIBUTION OF THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO, ITS OUTPUT, OR ANY ACCOMPANYING DOCUMENTATION.

GOVERNING LAW: THE RELATIONSHIP BETWEEN YOU AND ROBERT BOSCH GMBH SHALL BE GOVERNED SOLELY BY THE LAWS OF THE FEDERAL REPUBLIC OF GERMANY. THE STIPULATIONS OF INTERNATIONAL CONVENTIONS REGARDING THE INTERNATIONAL SALE OF GOODS SHALL NOT BE APPLICABLE. THE EXCLUSIVE LEGAL VENUE SHALL BE DUESSELDORF, GERMANY.

MANDATORY LAW SHALL BE UNAFFECTED BY THE FOREGOING PARAGRAPHS.

INTELLECTUAL PROPERTY OWNERS/COPYRIGHT OWNERS/CONTRIBUTORS:
ROBERT BOSCH GMBH, ROBERT BOSCH PLATZ 1, 70839 GERLINGEN,
GERMANY AND ITS LICENSORS.

Revision History

Version	Date	Remark
1.0	31.03.2026	Initial version for XS_CAN 1.0.0 – 1.1.0
1.1	27.04.2026	Correct the TXFQ end address calculation

Conventions

The following conventions are used within this document:

Register names	`TXFQ_LMEM_SA, TXFQ_CFG`
Names of files and directories	`directoryname/filename`
Source code/function names	`xs_can_mh_tx_fifo_queue_enqueue_msg()`

References

This document refers to the following documents:

Ref	Author	Title
[1]	MS/EEJ	XS_CAN User's Manual
[2]	MS/EEJ	XS_CAN System Integration Guide
[3]	MS/EEJ	calc_min_host_clk_freq_for_XS_CAN.xlsx
[4]	MS/EEJ	XS_CAN_Local_Memory_size_calculator.xlsx

Terms and Abbreviations

This document uses the following terms and abbreviations:

Term	Meaning
CAN	Controller Area Network
CAN CC	Controller Area Network Classical CAN
CAN FD	Controller Area Network with Flexible Data Rate
CAN XL	Controller Area Network with Extended frame Length
CTB	Cut Through Buffer
CTM	Cut Through Mode
DLC	Data Length Code
FMM	Full Message Mode
HOST	This is the CPU which is hosting the X_CAN
IP	Intellectual property
IRC	Interrupt Controller
LMEM	Local Memory
MH	Message Handler
PRT	Protocol Controller
PWM	Pulse Width Modulation
RAM	Random Access Memory

Term	Meaning
RX	Receive
RXFQ	Receive FIFO Queue
SMEM	System Memory
TEFQ	Transmit Event FIFO Queue
TEQE	Transmit Event FIFO Queue Element
TS	Timestamp
TX	Transmit
TXEF	Transmit Event FIFO
TXPQ	Transmit Priority Queue
TXFQ	Transmit FIFO Queue
TXQE	Transmit FIFO Queue Element
VBM	Virtual Buffer Manager
XS_CAN	Slim CAN XL IP

Table of Contents

1 Target 1
2 LMEM configuration and interface 2
2.1 LMEM Size 3
2.2 Host Clock frequency 3
2.3 Queue memory space in LMEM configuration 3
2.4 TXFQ (TX FIFO Queue) 5
2.5 TXPQ (TX Priority Queue) 6
2.6 TEFQ (TX Event FFIFO Queue) 9
3 Virtual Buffer Manager 11
3.1 Virtual buffer access order 12
4 Message Enqueue for transmission 13
4.1 General flowcharts for TXFQ and TXPQ 16
4.2 Example of Messages Enqueue for TXFQ and TXPQ 17
4.3 Internal Prioritization, and transmision after the enqueue: 19
5 TX Event Message Dequeue 20
5.1 General Flowcharts for TEFQ 22
5.2 Example of Event Messages Dequeue from TEFQ 23
6 Interrupt Flags 26
6.1 TX Functional Raw Event Status register (TX_FUNC_RAW) 26
6.2 Error Raw Event Status register (ERR_STS_RAW) 26
6.3 Safety Raw Event Status register (SAFETY_RAW) 26
7 Software Examples 27

List of Figures:

FIGURE 1: LMEM CONFIGURATION DIAGRAM EXAMPLE 2
FIGURE 2: ADRESS SPACE OF A VIRTUAL BUFFER (VB) 12
FIGURE 3 : ENQUEUE FLOW DIAGRAM FOR TXFQ AND TXPQ. 16
FIGURE 4 : ENQUEUE FLOW DIAGRAM FOR TEFQ. 22

List of Tables:

TABLE 1: OVERVIEW OF TX MESSAGE FRAME FORMAT AND SIZE 4
TABLE 2: TXFQ EXAMPLE CONFIGURATION 6
TABLE 3: TXPQ EXAMPLE CONFIGURATION 8
TABLE 4: TEFQ EXAMPLE CONFIGURATION 10
TABLE 5: VBM VIRTUAL ADDRESS MAP (FMM) 11
TABLE 6 : MESSAGE DLC AND PAYLOAD LENGTH 15
TABLE 7: DETAILED STEPS OF MESSAGE ENQUEUE FOR TXFQ AND TXPQ 18
TABLE 8: DETAILED STEPS OF DEQUEING OF EVENT MESSAGES FROM TEFQ (TEFQ_CFG.TEFE_LARGE = 1) 24
TABLE 9: DETAILED STEPS OF DEQUEING OF EVENT MESSAGES FROM TEFQ (TEFQ_CFG.TEFE_LARGE = 0) 25
TABLE 10: LIST OF SW EXAMPLE FUNCTIONS FOR TX MESSAGE HANDLING 28

1 Target

This application note describes transmit message handling in the **XS_CAN versions 1.0.0 through 1.1.0**

The topics covered include:

- Local Memory configuration for Queues (TXFQ, TXPQ, TEFQ)
- TX Message enqueue for transmission (TXFQ, TXPQ)
- TX Event dequeue after transmission (TEFQ)

Important Note:
The software examples delivered with this application note are for illustration purposes only. Use the examples at your own risk.

2 LMEM configuration and interface

The XS_CAN requires external local memory to store all messages (transmitted and received), TX Event information, and RX filter elements. This external memory is called Local Memory (LMEM).

Up to 256KB of LMEM is accessible by a single XS_CAN IP instance via the LMEM interface. The address width is 18 bits and the data width is one 32-bit word. Data is stored into LMEM as words (32 bits). The following Figure 1 shows a LMEM configuration example.

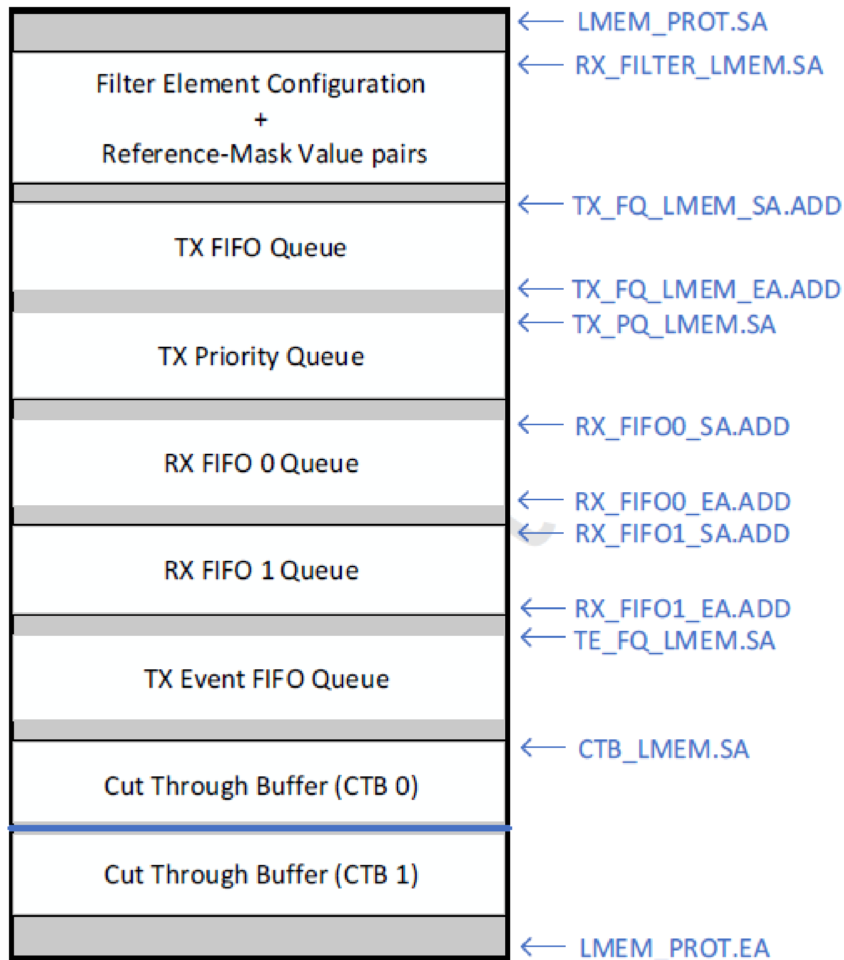

Figure 1: LMEM configuration diagram example

The XS_CAN provides a “LMEM protection configuration” (LMEMPC) - feature to configure LMEM protection for all accesses to the LMEM. XS_CAN allows accesses to LMEM only in the range of start and end addresses.

The start address (LMEMPC_START_ADDR) and the end address (LMEMPC_END_ADDR) are 18-bit width and must be configured in 64-byte

granularity. These are static inputs and must be provided before configuring and starting the
XS_CAN.
Important hint: The start address of every Queue must be 64-byte aligned. Therefore, the lower 6 bits (5:0) of the 18-bit address are not used and are treated as '0'.

XS_CAN allows accesses to LMEM only in the range of start and end addresses. Both addresses are included in this range. Since the start and end addresses are in 64-byte granularity an access is granted in the following address range: LMEMPC_START_ADDR to LMEMPC_END_ADDR. This granularity introduces limitation where the entire LMEM space may not be fully utilized. For example, for a 4KB memory with start address 0x000, the usable range would be from 0x00 to 0xFC0 (included). Any access to an address greater than 0xFC0 will result in the setting of an error event flag and triggering of an interrupt. Depending on the source of access (host, TX Handler, RX handler, Transparent access), behaviour of the IP is different. Refer Section 1.5.8 Error and Exception handling in [1] for details.

Messages to be transmitted are stored in the **TXFQ** and/or the **TXPQ** depending on the use case of the application.

TX messages are stored in LMEM and consist of a Message Header and a Message Payload. [1] describes the TX Message Header's structure and how it must be stored in the TXFQ/TXPQ. The Message Header's data structure depends on the CAN Frame Format (Classical CAN, CAN FD and CAN XL) and the mode of operation (FMM or CTM).

The XS_CAN supports single TXFQ with up to 255 messages, up to 32 TXPQ slots and single TEFQ with up to 63 elements.

2.1 LMEM Size

For the LMEM size calculation we provide with the XS_CAN IP an LMEM calculator [4]. For the LMEM size calculation the LMEM size calculator [4] should be used. This LMEM size calculator provide already different memory example configurations for FMM or CTM mode.

2.2 Host Clock frequency

For the Host clock calculation, we provide with the XS_CAN IP an Host clock calculator [3]. This Host clock calculation should be used to calculate Host clock frequency. This [3] is to check if a specific combination of bit rates (nominal bit rate, FD Data bit rate, XL Data bit rate) and SMEM/LMEM latencies is functional under specific system parameters

2.3 Queue memory space in LMEM configuration

The size and location of Queues in LMEM can be configured via their respective configuration registers, depending on the Queue type. See [1] for more information.
Hint:

The maximum size of TX message varies according to its CAN frame format.
See the following Table 1 with the overview of TX message frame format and size

	TX Message				total size of one message	
	Header		maximum Payload			
	(byte)	(word)	(byte)	(word)	(byte)	(word)
CC	8 (T0,T1)	2 (T0,T1)	8	2	16	4
FD	8 (T0,T1)	2 (T0,T1)	64	16	72	18
XL	12 (T0,T1,T2)	3 (T0,T1,T2)	2048	512	2060	515

Table 1: Overview of TX message frame format and size

2.4 TXFQ (TX FIFO Queue)

The XS_CAN supports one TX FIFO Queue (TXFQ) defined in LMEM. A TX FIFO Queue holds TX messages to be sent in order to the PRT. Each message inside TXFQ is called a TX Queue Element (TXQE). Each TXQE consists of header and a payload. See more information in [1], chapter TX Message Header Definition.

The XS_CAN provides register **TXFQ_LMEM_SA.ADD** for configuring the start address and **TXFQ_LMEM_EA.ADD** for configuring the end address of TXFQ in LMEM.

See the following parameter overview:

TXFQ_LMEM_SA.ADD	defines the TXFQ's start addresses within LMEM.
TXFQ_LMEM_EA.ADD	defines the TXFQ's end addresses within LMEM.

Their values must always be 64-byte aligned address; therefore, only the higher 12 bits (17:6) of the 18-bit address are considered.

Example configuration for TXFQ

In this configuration example, the TXFQ starts at address 0x02000 (byte address) in the LMEM.

Number of messages in TXFQ	Maximum message length	Calculation and configuration
10	Classical CAN 8-byte payload	required size = 160 bytes SA = 0x02000 (byte address) EA = 0x0209F (byte address) configurable size = 192 bytes (multiple of 64 bytes) SA = 0x02000 (64-byte aligned address) EA = 0x020C0 (64-byte aligned address) **TXFQ_LMEM_SA.ADD = 0x0080** **TXFQ_LMEM_EA.ADD = 0x0083**
10	CAN FD 8-byte payload	required size = 160 bytes SA = 0x02000 (byte address) EA = 0x0209F (byte address) configurable size = 192 bytes (multiple of 64 bytes) SA = 0x02000 (64-byte aligned address) EA = 0x020C0 (64-byte aligned address) **TXFQ_LMEM_SA.ADD = 0x0080** **TXFQ_LMEM_EA.ADD = 0x0083**

Number of messages in TXFQ	Maximum message length	Calculation and configuration
10	CAN FD 64-byte payload	required size = 720 bytes SA = 0x02000 (byte address) EA = 0x022CF (byte address) configurable size = 768 bytes (multiple of 64 bytes) SA = 0x02000 (64-byte aligned address) EA = 0x02300 (64-byte aligned address) **TXFQ_LMEM_SA.ADD = 0x0080** **TXFQ_LMEM_EA.ADD = 0x008C**
10	CAN XL 512-byte payload	required size = 5240 bytes SA = 0x02000 (byte address) EA = 0x03477 (byte address) configurable size = 5248 bytes (multiple of 64 bytes) SA = 0x02000 (64-byte aligned address) EA = 0x03480 (64-byte aligned address) **TXFQ_LMEM_SA.ADD = 0x0080** **TXFQ_LMEM_EA.ADD = 0x00D2**
10	CAN XL 2048-byte payload	required size = 20600 bytes SA = 0x02000 (byte address) EA = 0x07077 (byte address) configurable size = 20608 bytes (multiple of 64 bytes) SA = 0x02000 (64-byte aligned address) EA = 0x07080 (64-byte aligned address) **TXFQ_LMEM_SA.ADD = 0x0080** **TXFQ_LMEM_EA.ADD = 0x01C2**

Table 2: TXFQ example configuration

2.5 TXPQ (TX Priority Queue)

The XS_CAN IP supports one TX Priority Queue (TXPQ). It supports up to 32 priority queue slots, and the number of slots are user configurable. See more information in [1].

The IP provides a register **TXPQ_LMEM.SA** for configuring the TXPQ start address in the LMEM. The address value is always 64-byte aligned.

The TXPQ Configuration register provides **TXPQ_CFG.SLOT_NUM** for configuring the number of slots in TXPQ. User can store up to 32, and **TXPQ_CFG.SLOT_SIZE** to define the size of a TXPQ slot.

See the following parameter overview:

TXPQ_LMEM.SA	defines the TXPQ's start addresses within LMEM.
TXPQ_CFG. SLOT_SIZE	defines the size of a TXPQ slot in 32-bit word.
TXPQ_CFG.SLOT_NUM	defines the number of TXPQ slots, up to 32.

The size of the TXPQ in LMEM is = **SLOT_NUM x SLOT_SIZE**

The start address of Slot n where $n \in \{1,2,3,\ldots,32\}$ in TXPQ is calculated as:
(TXPQ_LMEM.SA) + (n*TXPQ_CFG.SLOT_SIZE)

End address of each slot (n) is Start address of slot (n+1) - 4.

Example configuration for TXPQ

In this configuration example, the TXPQ starts at address 0x03000 (byte address) in the LMEM.

Number of slots in TXPQ	Maximum message length	Calculation and configuration
10	Classical CAN 8-byte payload	required slot size = 16 bytes configurable slot size = 4 words SA = 0x03000 (64-byte aligned address) **TXPQ_LMEM.SA = 0x00C0** **TXPQ_CFG. SLOT_SIZE = 4** **TXPQ_CFG.SLOT_NUM = 10** total TXPQ size = 40 words
10	CAN FD 64-byte payload	required slot size = 72 bytes configurable slot size = 18 words SA = 0x03000 (64-byte aligned address) **TXPQ_LMEM.SA = 0x00C0** **TXPQ_CFG. SLOT_SIZE = 18** **TXPQ_CFG.SLOT_NUM = 10** total TXPQ size = 180 words
10	CAN XL 512-byte payload	required slot size = 524 bytes configurable slot size = 131 words SA = 0x03000 (64-byte aligned address) **TXPQ_LMEM.SA = 0x00C0** **TXPQ_CFG. SLOT_SIZE = 131** **TXPQ_CFG.SLOT_NUM = 10** total TXPQ size = 1310 words
10	CAN XL 2048-byte payload	required slot size = 2060 bytes configurable slot size = 515 words SA = 0x03000 (64-byte aligned address) **TXPQ_LMEM.SA = 0x00C0** **TXPQ_CFG. SLOT_SIZE = 515** **TXPQ_CFG.SLOT_NUM = 10** total TXPQ size = 5150 words

Table 3: TXPQ example configuration

2.6 TEFQ (TX Event FFIFO Queue)

The XS_CAN IP supports one TX Event FIFO Queue (TEFQ). It stores one TX Event FIFO Queue Element (TEQE) to the TEFQ per TX message transmitted if EFC (Event FIFO control) bit in the TXFQ header is enabled.

The XS_CAN provides a register **TEFQ_LMEM.SA** for configuring the TEFQ start address where the TEQE are defined in LMEM. The address is always 64-byte aligned.

The TEFQ LMEM Configuration register **TEFQ_CFG.TEQE_LARGE** decides about the size of TEQE. When it is set to ‘1‘, the TEQE is of 5 words else it is of 4 words for all frame formats. **TEFQ_CFG.TEQE_NUM** is also provided to define the maximum number (up to 63 elements) of TEQE that will be stored in LMEM.

See the following parameter overview:

TEFQ_LMEM.SA	defines the TEFQ's start addresses within LMEM.
TEFQ_CFG. TEFE_LARGE	defines the size of a TEFQ element. If **TEFE_LARGE** is ‘1’, an element is 5 32-bit words. If **TEFE_LARGE** is ‘0’, an element is 4 32-bit words.
TEFQ_CFG.TEFE_NUM	defines the number of TEFQ elements, up to 63.

The size of the TEFQ in LMEM is = **TEFE_NUM x TEFE size**

Example configuration for TEFQ

In this configuration example, the TEFQ starts at address 0x06000 (byte address) in the LMEM.

Number of elements in TEFQ	TX message type and Option	Calculation and configuration
10	Classical CAN or/and CAN FD	configurable element size = 4 words SA = 0x06000 (64-byte aligned address) **TEFQ_LMEM.SA = 0x0180** **TEFQ_CFG.TEFE_LARGE = 0** **TEFQ_CFG.SLOT_NUM = 10** total TEFQ size = 40 words
10	Classical CAN or/and CAN FD and CAN XL no Acceptence Field of CAN XL capturing required	configurable element size = 4 words SA = 0x06000 (64-byte aligned address) **TEFQ_LMEM.SA = 0x0180** **TEFQ_CFG.TEFE_LARGE = 0** **TEFQ_CFG.SLOT_NUM = 10** total TEFQ size = 40 words

Number of elements in TEFQ	TX message type and Option	Calculation and configuration
10	Classical CAN or/and CAN FD and CAN XL Acceptence Field of CAN XL capturing required	configurable element size = 4 words SA = 0x06000 (64-byte aligned address) **TEFQ_LMEM.SA = 0x0180** **TEFQ_CFG.TEFE_LARGE = 1** **TEFQ_CFG.SLOT_NUM = 10** total TEFQ size = 50 words

Table 4: TEFQ example configuration

3 Virtual Buffer Manager

The **Virtual Buffer Manager (VBM)** in MH handles the conversion of virtual addresses to physical LMEM addresses. It manages the enqueuing of messages into TXFQ, and TXPQ slots, as well as the dequeuing of messages from RXFQs and the TEFQ.

The host can access the full 256KB LMEM address space using **LMEM Transparent Access**, **LMEM TA**, the address range 0x40000 to 0x7FFFC (the byte address bit 18 (the 19th bit) is '1').

However, the actual memory space which is allocated to an XS_CAN IP is indicated in **LMEM_PROT.SA** (Start Address of LMEM space) and **LMEM_PROT.EA** (End Address of LMEM space). Transparent accesses outside this range of start and end address will generate **SAFETY_RAW.MH_LMEM_PROT_ERR** interrupt. See LMEM Access Protection chapter [1] for more information.

The host interacts with LMEM via the VBM interface. To simplfiy the LMEM access for the host, virtual buffers are set up for each Queue type: **TXFQ, TXPQ, RXFQ0, RXFQ1, and TEFQ**. The address range for these virtual buffers is fixed, matching the size of the largest message that can be stored in any given Queue.

All virtual addresses are 64-byte aligned. Each virtual buffer has its own **start, trigger, and end addresses**. The XS_CAN-FMM's virtual buffer address map is shown in the following table.

Start location Address	End locationAddress	Symbol	Name
0x01000	0x01808	TXFQ	TX FIFO Queue
0x0180C	0x01FFC	reserved	reserved
0x02000	0x02808	TXPQ	TX Priority Queue
0x0280C	0x02FFC	reserved	reserved
0x03000	0x03810	RXFQ0	RX FIFO 0 Queue
0x03814	0x03FFC	reserved	reserved
0x04000	0x04810	RXFQ1	RX FIFO 1 Queue
0x04814	0x04FFC	reserved	reserved
0x05000	0x05010	TEFQ	TX Event FIFO Queue
0x05014	0x3FFFC	reserved	reserved
0x40000	0x7FFFC	LMEM TA	LMEM Transparent Access

Table 5: VBM Virtual Address Map (FMM)

The VBM processes transactions to virtual buffers only when the **MH_CTRL.ENABLE** bit is set. However, transparent LMEM access remains possible even without setting the **MH_CTRL.ENABLE** bit.

3.1 Virtual buffer access order

The host issues address in linear incrementing order (4-byte alignment), starting from the start address until the trigger address.

Any other order of addresses issued is considered as a nonlinear access and the Error- and Exception Handling behaviour starts.

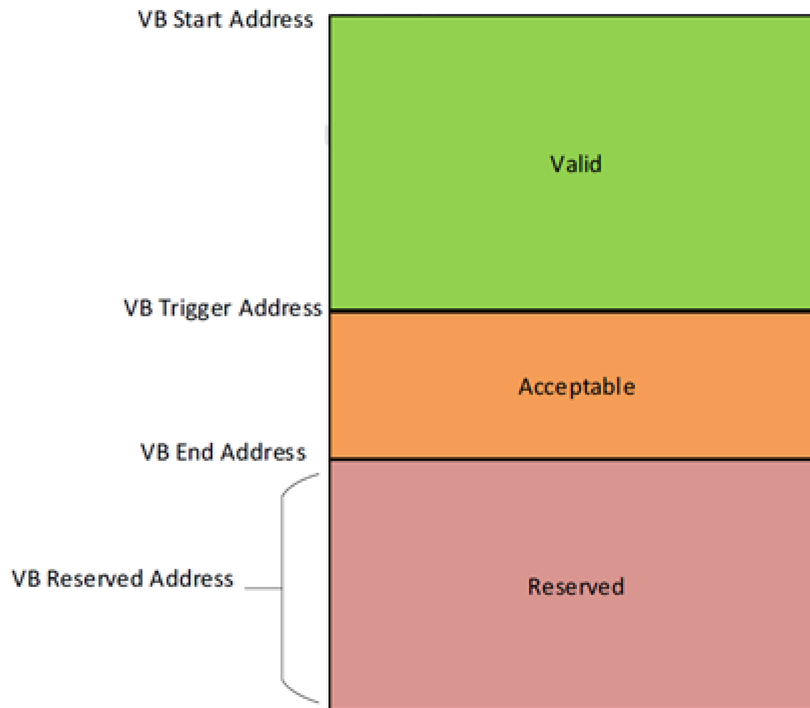

Figure 2: Adress space of a Virtual Buffer (VB)

To start the enqueue and dequeue of a message, the host issues the start address of the VB. Trigger address corresponds to the last word of the message. Trigger address and end address can be same if the message has the maximum payload possible which is a CAN XL with 2KB payload. See more information in [1].

4 Message Enqueue for transmission

The XS_CAN supports a single TXFQ with up to 255 messages, and up to 32 TXPQ slots.

Before using the TXFQ and TXPQ, they must be enabled by setting **MH_CFG.TXFQE= '1'** and **MH_CFG.TXPQE='1'**.

To transmit a message, the following steps are required:

- **Step 1: Ensure sufficient memory space for Message Enqueueing**

When a Host is used for enqueuing, check the following FIFO Status signals before Enqueue is started.

  - **TXFQ:** check **TXFQ_STS.FULL** to ensure the TXFQ is not full.
  - **TXPQ:** check **TXPQ_STS1.TXPQ_FULL** to ensure not all TXPQ slots are occupied.

The enqueue operation will only proceed if the **TXFQ is not full** or the **TXPQ slot is available**.

When an external DMA controller is used for enqueuing instead of the host interface, the following DMA request signals shall be used for checking status changes. Monitor the following Transfer Request Signals to ensure that message write is accepted by VBM.

  - **TXFQ: TXFQ_WREQ** (TX FIFO Queue write request from VBM) signal.
  - **TXPQ: TXPQ_WREQ** (TX Priority Queue write request from VBM) signal.

See more information in [1], chapter Transfer Request Signals.

- **Step 2: Start Enqueuing of TXFQ/TXPQ**

To start Enqueuing a new message into either the **TXFQ** or **TXPQ**, write the message header **T0** (the first word) to **VB start address**. This address is **0x01000** for TXFQ or **0x02000** for TXPQ. This action triggers the VBM to initiate the enqueue process for the respective Queue.

The TXFQ and TXPQ handling module within the VBM converts the virtual buffer address to the actual LMEM address where the message will be enqueued.

After that, write subsequent words, beginning with T1, linearly at increments of +4 from the start address of TXFQ/TXPQ.

The VBM calculates a trigger address based on the frame type (determined by the FDF and XLF bits in T0), and the RTR and DLC bits in T1. This trigger address corresponds to the last word of the enqueued message.

After the trigger address is accessed, the next expected address is the start address of the VB again. In case if this is violated, status flags are updated. Refer the registers MH_STS0, MH_STS1 or INTERRUPTS, see more information in [1].

- ### Step 3: End of Enqueuing

After enqueuing the following registers are updated by VBM:

For TXFQ:

1) Write pointer register **TXFQ_WPTR.WPTR_ADD** with the actual LMEM address corresponding to the last word of TXQE that is enqueued.
2) The fill level of the TXFQ is incremented by one and is updated into the register **TXFQ_STS.FILL_LEVEL**.

For TXPQ:

1) Write pointer register **TXPQ_WPTR.WPTR_ADD** with the actual LMEM address corresponding to the last word of TXQE that is enqueued.
2) The slot occupied status into the register **TXPQ_STS0.SLOT_OCC[n]**.
3) The fill level of the TXPQ is incremented by one and is update into the register **TXPQ_STS1.FILL_LEVEL**

The Application SW can use these registers to monitor the Enqueue process.

See the following table with the message DLC and payload length.

DLC	data field in bytes			
	classical CAN		CAN FD	CAN XL
	remote frame	data frame		
0	0	0	0	1
1	0	1	1	2
2	0	2	2	3
3	0	3	3	4
4	0	4	4	5
5	0	5	5	6
6	0	6	6	7
7	0	7	7	8
8	0	8	8	9
9	0	8	12	10
10	0	8	16	11
11	0	8	20	12
12	0	8	24	13
13	0	8	32	14
14	0	8	48	15
15	0	8	64	16
16	n/a	n/a	n/a	17
...	n/a	n/a	n/a	...
n	n/a	n/a	n/a	n+1
...	n/a	n/a	n/a	...
2047	n/a	n/a	n/a	2048

Table 6 : Message DLC and payload length

In some applications, message transfers with dynamic lengths, controlled by the message DLC are not always applicable. In such cases, the DMA controller may transfer (Enqueue) with the size of the largest possible TX message (maximum 2060 bytes), regardless of the actual message length. The VBM tolerates this by simply discarding these accesses. The status flags **MH_STS0.TXFQ_VB_NO_SA** or **MH_STS0.TXPQ_VB_NO_SA** will be updated, when this occurs.

4.1 General flowcharts for TXFQ and TXPQ

These two flow diagram show the general flow of Message Enqueueing for TXFQ and TXPQ.

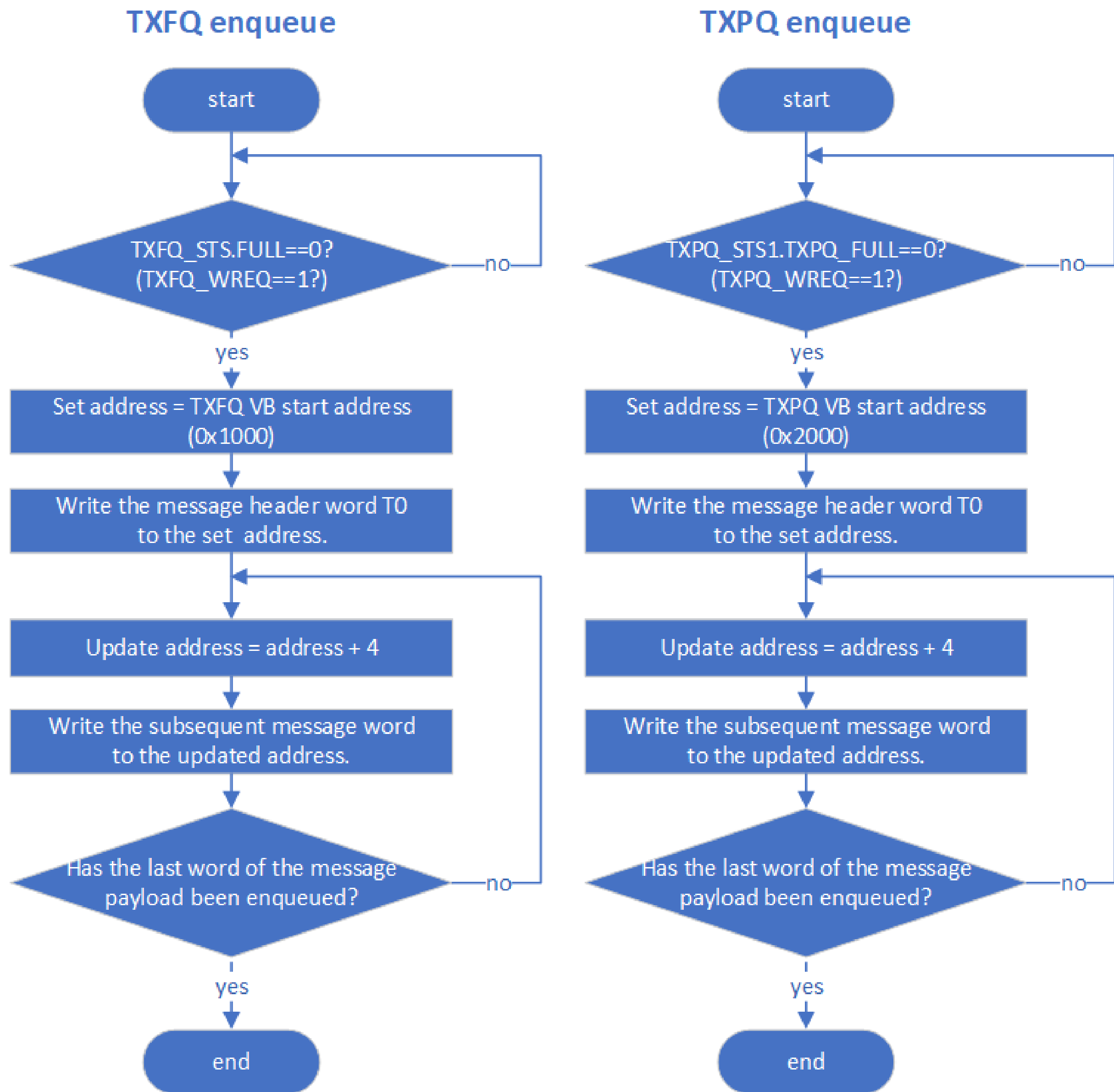

Figure 3 : Enqueue flow diagram for TXFQ and TXPQ.

4.2 Example of Messages Enqueue for TXFQ and TXPQ

The following examples show the Enqueue of TX Messages in detailed steps for TXFQ and TXPQ.

TXFQ example configuration:

Message 1:	A CAN FD frame with DLC=4, trigger address will be at word T2(0x1008)
Message 2:	A CAN XL frame with DLC=6, trigger address will be at word T4(0x1010)

TXPQ example configuration:

Message 3:	A CAN FD frame with DLC=4, trigger address will be at word T2(0x2008)
Message 4:	A CAN XL frame with DLC=6, trigger address will be at word T4(0x2010)

The following steps detail how to enqueue **msg1** and **msg2** into TXFQ, and **msg3** and **msg4** into TXPQ.

Step	Description	Enqueue to TXFQ	Enqueue to TXPQ
1	Ensure sufficient space for enqueue	**TXFQ_STS.FULL**==0? (TXFQ_WREQ==1?)	**TXPQ_STS1.TXPQ_FULL** ==0? (TXPQ_WREQ==1?)
2	If space is available, proceed with enqueuing; otherwise, wait until space becomes available.		
3	Begin enqueuing **msg1/msg3** at the VB start address.	Write T0 (header word 0) to address **0x1000**	Write T0 (header word 0) to address **0x2000**
4	Continue enqueuing **msg1/msg3** the subsequent word	Write T1 (header word 1) to address **0x1004**	Write T1 (header word 1) to address **0x2004**
5	Continue enqueuing **msg1/msg3** the last word	Write T2 (payload byte 0 – 3) to address **0x1008** = trigger address	Write T2 (payload byte 0 – 3) to address **0x2008** = trigger address

Step	Description	Enqueue to TXFQ	Enqueue to TXPQ
6	Ensure sufficient space for enqueue	**TXFQ_STS.FULL**==0? (TXFQ_WREQ==1?)	**TXPQ_STS1.TXPQ_FULL** ==0? (TXPQ_WREQ==1?)
7	If memory space is available, proceed with enqueuing; otherwise, wait until space becomes available.		
8	Begin enqueuing **msg2/msg4** at the VB start address.	Write T0 (header word 0) to address **0x1000**	Write T0 (header word 0) to address **0x2000**
9	Continue enqueuing **msg2/msg4** the subsequent word	Write T1 (header word 1) to address **0x1004**	Write T1 (header word 1) to address **0x2004**
10	Continue enqueuing **msg2/msg4** the subsequent word	Write T2 (acceptance field) to address **0x1008**	Write T2 (acceptance field) to address **0x2008**
11	Continue enqueuing **msg2/msg4** the subsequent word	Write T3 (payload byte 0 – 3) to address **0x100C**	Write T3 (payload byte 0 – 3) to address **0x200C**
12	Continue enqueuing **msg2/msg4** the last word	Write T4 (payload byte 4 – 7) to address **0x1010** = trigger address Note: since DLC = 6, byte 7 is written but not used.	Write T4 (payload byte 4 – 7) to address **0x2010** = trigger address Note: since DLC = 6, byte 7 is written but not used.

Table 7: Detailed steps of Message Enqueue for TXFQ and TXPQ

4.3 Internal Prioritization, and transmission after the enqueue:

Inside the XS_CAN, the internal TX Scan module indentifies which enqueued message in LMEM to be transmitted first. The internal arbitration process selects the message with the lowest ID, which has the highest priority. Only the first message header element T0 from enqueued message is used to identify the priority during the TX Scan. See more information in [1] chapter TX Scan.

- **TXFQ (only)**: The oldest message (first-enqueued) in the TXFQ is transmitted first. (A)
- **TXPQ (only)**: Among the messages currently in the TXPQ slots, the message with the lowest ID is transmitted first. (B)

  Note:

  If **TXPQ_CFG.TX_MSG_SEQE** (Transmit Messages Sequence Enable) is set to '1', messages in the TXPQ that have the same priority are transmitted in a user-defined sequence. This sequence is defined by the user via the **MM** (Message Marker) in the TX Message Header T1. The message with the lowest **MM** is transmitted first.

- **TXFQ and TXPQ (combined):** When both Queues (TXFQ/TXPQ) contain messages, then the message with the lower ID between the oldest TXFQ message (A) and the lowest ID TXPQ message (B) is transmitted first.

  If messages have the same priority, they are transmitted in the following order, first TXPQ slot elements from 0 to 31 followed by TXFQ oldest element.

Hint: The Remote frame and the Data frame of the Classical CAN frame have the same priority.

5 TX Event Message Dequeue

The XS_CAN supports one TX Event FIFO (TEFQ) with up to 63 elements.

The TEFQ stores the feedback information of the TX message including whether the message is successfully transmitted or not, the source of the TX message (TXFQ or TXPQ), as well as the timestamp.

Before using the **TEFQ**, the **MH_CFG.TEFQE** must be enabled by setting to 1.

To enable XS_CAN to generate the transmitted message information (TX Event), the **Event FIFO Control** (EFC) bit in the TX message header element **T1** must be enabled by setting to 1.

To dequeue an event message, the following steps are required:

- **Step 1: Ensure there is event message in the TEFQ available.**

When a Host is used for dequeuing, check the fill level, that the **TEFQ_STS** → **FILL_LEVEL is > 0**.

The dequeue operation shall only proceed if the TEFQ fill level is greater than zero.

Alternatively, the interrupt **TX_FUNC_RAW.MH_TEFQ_ENQ** can also be used; it indicates a new event message was enqueued into the TEFQ.

When an external DMA controller is used for dequeuing instead of the host interface, the DMA request signal **TXEF_RREQ** (TX Event FIFO Queue read request from VBM) shall be used for monitoring status changes.

- **Step 2: Start dequeuing**

To start dequeuing an Event Message from the TEFQ, read the Event Message element **E0** (the first word) from VB start address **0x05000**. This action triggers the VBM to initiate the dequeue process.

The TEFQ handling module within the VBM converts the virtual buffer address to the actual LMEM address where the message will be dequeued.

After that, read subsequent words, beginning with E1, linearly at increments of +4 from the start address.

The VBM calculates a trigger address based on **TEFQ_CFG. TEFE_LARGE**. If **TEFQ_CFG.TEQE_LARGE**=1, the trigger address is **0x05010**; otherwise, it is **0x0500C**.

After the trigger address is accessed, the next expected address is the start address of the VBM.

- **Step 3: End of dequeuing**

After dequeuing the following registers are updated by VBM:

1) Read pointer register **TEFQ_RPTR.RPTR_ADD** with the actual LMEM address corresponding to the last word of TEQE that is dequeued.
2) The fill level of TEFQ is decremented by one and is updated into the register **TEFQ_STS.FILL_LEVEL**.

The Application SW can use these registers to monitor the Enqueue process.

5.1 General Flowcharts for TEFQ

These two flow diagrams show the general flow of Message Enqueueing for TEFQ.

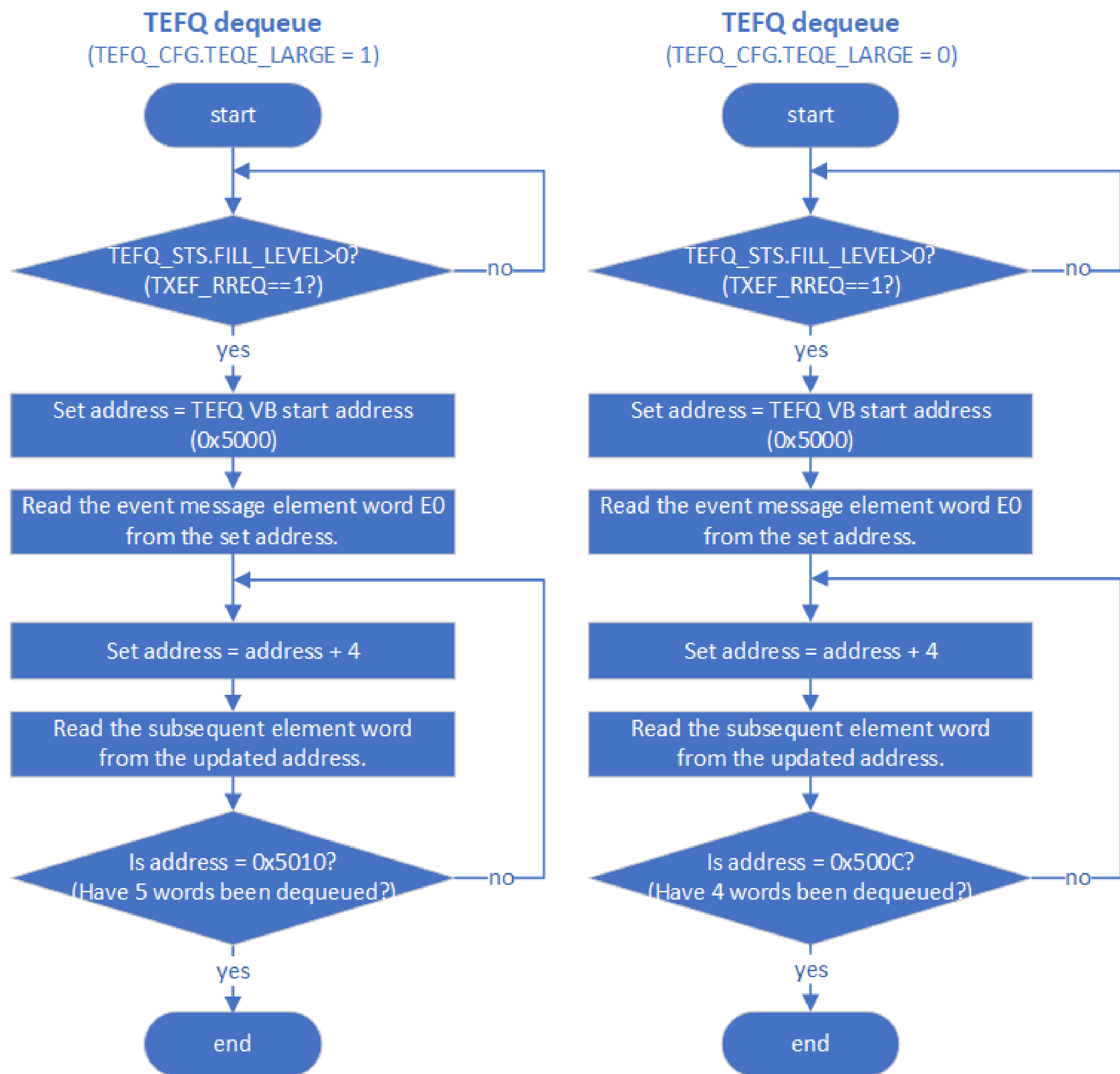

Figure 4 : Enqueue flow diagram for TEFQ.

5.2 Example of Event Messages Dequeue from TEFQ

Dequeue event messages from TEFQ when **TEFQ_CFG.TEFE_LARGE** = **1** and **TEFQ_CFG.TEFE_LARGE** = **0**

The following steps show in detail how to dequeue Event Messages consecutively, if **TEFQ_CFG.TEQE_LARGE = 1**.

Step	Description	TEFQ_CFG.TEQE_LARGE = 1
1	Ensure sufficient space for enqueue	**TEFQ_STS. FILL_LEVEL**>0? (TXEF_RREQ==1?)
2	If event message is available, proceed with dequeuing; otherwise, wait until event message becomes available.	
3	Start dequeuing **event msg1** at the VB start address.	Read E0 (word 0) from address 0x5000
4	Continue dequeuing **event msg1** the subsequent word	Read E1 (word 1) from address 0x5004
5	Continue dequeuing **event msg1** the subsequent word	Read E2 (word 2) from address 0x5008
6	Continue dequeuing **event msg1** the subsequent word	Read E3 (word 3) from address 0x500C
7	Continue dequeuing **event msg1** the last word	Read E4 (word 4) from address 0x5010 = trigger address
8	Ensure sufficient space for enqueue	**TEFQ_STS. FILL_LEVEL**>0? (TXEF_RREQ==1?)
9	If event message is available, proceed with dequeuing; otherwise, wait until event message becomes available.	
10	Start dequeuing **event msg2** at the VB start address.	Read E0 (word 0) from address 0x5000
11	Continue dequeuing **event msg2** the subsequent word	Read E1 (word 1) from address 0x5004
12	Continue dequeuing **event msg2** the subsequent word	Read E2 (word 2) from address 0x5008

Step	Description	TEFQ_CFG.TEQE_LARGE = 1
13	Continue dequeuing **event msg2** the subsequent word	Read E3 (word 3) from address 0x500C
14	Continue dequeuing **event msg2** the last word	Read E4 (word 4) from address 0x5010 = trigger address

Table 8: Detailed steps of dequeuing of event messages from TEFQ (TEFQ_CFG.TEFE_LARGE = 1)

The following steps show in detail how to dequeue Event Messages consecutively, if **TEFQ_CFG.TEQE_LARGE = 0**.

Step	Description	TEFQ_CFG.TEQE_LARGE = 0
1	Ensure sufficient space for enqueue	**TEFQ_STS. FILL_LEVEL**>0? (TXEF_RREQ==1?)
2	If event message is available, proceed with dequeuing; otherwise, wait until event message becomes available.	
3	Start dequeuing **event msg1** at the VB start address.	Read E0 (word 0) from address 0x5000
4	Continue dequeuing **event msg1** the subsequent word	Read E1 (word 1) from address 0x5004
5	Continue dequeuing **event msg1** the subsequent word	Read E2 (word 2) from address 0x5008
6	Continue dequeuing **event msg1** the last word	Read E4 (word 4) from address 0x500C = trigger address
7	Ensure sufficient space for enqueue	**TEFQ_STS. FILL_LEVEL**>0? (TXEF_RREQ==1?)
8	If event message is available, proceed with dequeuing; otherwise, wait until event message becomes available.	
9	Start dequeuing **event msg2** at the VB start address.	Read E0 (word 0) from address 0x5000
10	Continue dequeuing **event msg2** the subsequent word	Read E1 (word 1) from address 0x5004

Step	Description	TEFQ_CFG.TEQE_LARGE = 0
11	Continue dequeuing **event msg2** the subsequent word	Read E2 (word 2) from address 0x5008
12	Continue dequeuing **event msg2** the last word	Read E4 (word 4) from address 0x500C = trigger address

Table 9: Detailed steps of dequeuing of event messages from TEFQ (TEFQ_CFG.TEFE_LARGE = 0)

6 Interrupt Flags

6.1 TX Functional Raw Event Status register (TX_FUNC_RAW)

The following flags are relevant to message transmission.

Bit field/flag	Description
PRT_TX_EVT	A valid CAN message was transmitted by PRT.
MH_TEFQ_DEQ	A message is dequeued from TEFQ.
MH_TEFQ_ENQ	A message is enqueued into TEFQ.
MH_TXPQ_ENQ	A message is enqueued into TXPQ.
MH_TXFQ_ENQ	A message is enqueued into TXFQ.

6.2 Error Raw Event Status register (ERR_STS_RAW)

See the following relevant flags to message transmission.

Bit field/flag	Description
PRT_IFF	An invalid Frame Format at TX_MSG detected by PRT.
MH_TEFQ_DROP	A new TEQE is dropped
MH_TXPQ_DEQ_NO_TX	A message has been dequeued from TXPQ but is not transmitted.
MH_TXFQ_DEQ_NO_TX	A message has been dequeued from TXFQ but is not transmitted.

6.3 Safety Raw Event Status register (SAFETY_RAW)

See the following relevant flags to message transmission.

Bit field/flag	Description
PRT_TX_DU	PRT detected underrun condition at TX_MSG sequence.
PRT_USOS	PRT detected unexpected Start of Sequence during TX_MSG sequence.
PRT_TX_ABORTED	PRT detected stop of TX_MSG sequence by TX_MSG_WUSER code ABORT.
MH_TX_ABORT	MH has to abort an ongoing transmission of a message to PRT.

See more information in [1], chapter 1.8 IRC - Interrupt Controller

These flags are typically processed by the host SW driver interrupt routine.

7 Software Examples

The following table provides examples of the 'C' SW-functions that are used as demonstrated in this application note.

Name:	`xs_can_app_note_tx_rx()`
File:	`../xs_can/app_notes/app_note_tx_rx.c`
Description:	The example demonstrates how to use the TXFQ and the RXFQ to transmit and receive Classical CAN, CAN FD and CAN XL messages.
Name:	`xs_can_app_note_tx_rx_transceiver_mode_switch_on()`
File:	`../xs_can/app_notes/app_note_tx_rx.c`
Description:	The example demonstrates how to use the TXFQ and the RXFQ to transmit and receive CAN XL message when the **Transceiver Mode Switching** is enabled. In this mode, Pulse Width Mode (PWM) is applied on the TXD pin of the transceiver.
Name:	`xs_can_app_note_txpq()`
File:	`../xs_can/app_notes/app_note_txpq.c`
Description:	The example demonstrates how to use the TXPQ and the RXFQ to transmit and receive Classical CAN, CAN FD and CAN XL messages.
Name:	`xs_can_app_note_txpq_with_message_sequence_enabled()`
File:	`../xs_can/app_notes/app_note_txpq.c`
Description:	The example demonstrates how to use the TXPQ with the use of **Message Marker** (**MM**) and the RXFQ to transmit and receive Classical CAN, CAN FD and CAN XL messages.
Name:	`xs_can_app_note_tefq()`
File:	`../xs_can/app_notes/app_note_tefq.c`
Description:	The example demonstrates how to get the event message of the transmitted message from the TEFQ.

Name:	`xs_can_mh_set_config()`
File:	`../xs_can/mh/xs_can_mh.c`

Description:	This function demonstrates how to configure all of the relevant MH configuration registers.
	`MH_CFG`
	`RX_FILTER_CFG`
	`RX_FILTER_LMEM`
	`TXFQ_LMEM_SA`
	`TXFQ_LMEM_EA`
	`TXFQ_CFG`
	`TXPQ_CFG`
	`TXPQ_LMEM`
	`TEFQ_LMEM`
	`TEFQ_CFG`
	`RXFQ0_SA`
	`RXFQ0_EA`
	`RXFQ1_SA`
	`RXFQ1_EA`

Name:	`xs_can_mh_tx_fifo_queue_enqueue_msg()`
File:	`../xs_can/mh/xs_can_mh.c`
Description:	This functon demonstrates how to enqueue a message into the TXFQ.
Name:	`xs_can_mh_tx_priority_queue_enqueue_msg()`
File:	`../xs_can/mh/xs_can_mh.c`
Description:	This functon demonstrates how to enqueue a message into the TXPQ.
Name:	`xs_can_mh_tx_event_fifo_queue_dequeue_msg()`
File:	`../xs_can/mh/xs_can_mh.c`
Description:	This functon demonstrates how to dequeue an event message from the TXFQ.

Table 10: List of SW example functions for TX message handling

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