BOSCH
Invented for life

X_CAN

Modular CAN IP-module

Transmission and Reception Handling with FIFO Queue

Application Note X_CAN_AN001

Document Revision 1.0

Robert Bosch GmbH
Automotive Electronics

LEGAL NOTICE

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	11.12.2024	First version for X_CAN version 1.0.0 – 1.1.0

Conventions

The following conventions are used within this document:

Register names	`TX_FQ_START_ADDn, TX_FQ_CTRL0`
Names of files and directories	`directoryname/filename`
Source code/function names	`xcand_mh_tx_fifo_enqueue_msg()`

References

This document refers to the following documents:

Ref	Author	Title
[1]	ME-IC/PAY	XCAN user manual version 3.90

Terms and Abbreviations

This document uses the following terms and abbreviations:

Term	Meaning
AXI	Advanced eXtensible Interface
C	Software example functions programmed in “C”
CAN	Controller Area Network
CAN CC	CAN classic
CAN FD	CAN flexible data rate
CAN XL	CAN extended data Length
DESC	Descriptor
FQ	FIFO Queue
HOST	This is the CPU which is hosting the X_CAN
HW	Hardware
IRC	Interrupt Request Controller
L_MEM	Local Memory
MH	Message Handler
NA	Not applicable
PRT	Protocol Controller
RX	Receive
S_MEM	System Memory
SW	Software
TX	Transmit

Table of Contents

1 Target of this Application note

This application note describes transmit message handling and receive message handling using FIFO queue and Data Container in the **X_CAN versions 1.0.0 to 1.1.0**

The topics included are:

- Introduction of MH, L_MEM, S_MEM, TX Descriptor and RX Descriptor.
- General preparation of MH before using the FIFO Queue
- Preparation and transmission of TX message using TX FIFO Queue
- Preparation and reading out the RX message using RX FIFO Queue

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

2 MH Message Handler

2.1 MH Introduction

The MH is located in between the main interconnect and the PRT.

All functions concerning the storage and scheduling of CAN messages are implemented in the **MH**. The TX path supports the storage of CAN messages in up to **8 TX FIFO Queues** and **1 TX Priority Queue**. The RX path provides **8 RX FIFO Queues**.

FIFO data is physically stored in S_MEM and managed by the Descriptors.

TX and RX Filters provide methods to accept or deny CAN Messages and (for RX only), to determine the target RX FIFO for data storage.

The MH is configured and controlled by HOST CPU via HOST_AXI interface. CAN messages and Descriptors are transported between S_MEM and L_MEM autonomously by an internal DMA, which is connected to DMA_AXI. For fast access, the MH needs a L_MEM which is connected via MEM_AXI interface.

Depending on the chosen SoC integration, multiple X_CAN IPs can share the same L_MEM.

See more details of MH in [1].

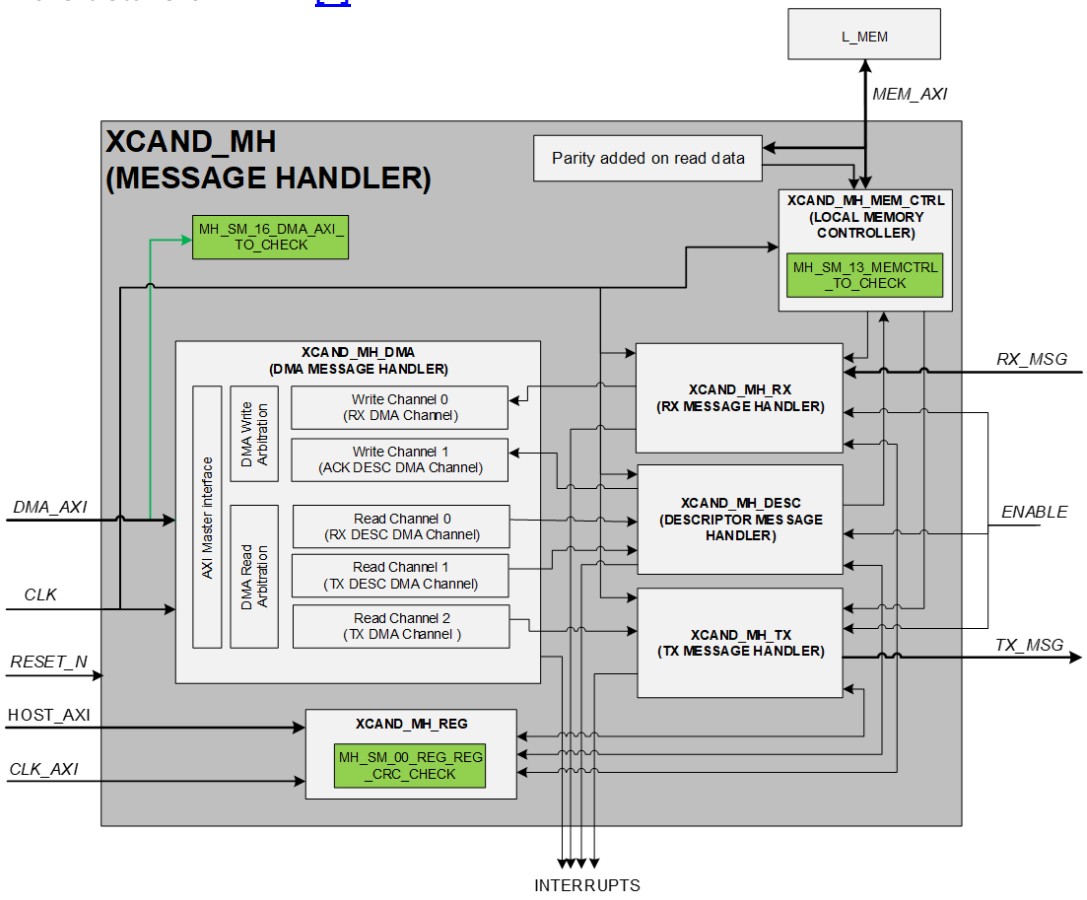


Figure: Block diagram MH

2.2 MH - AXI interfaces Overview

The MH is configured and controlled by the HOST via the HOST_AXI interface, which is an AXI4-Lite slave interface.

The MH uses the **DMA_AXI** interface to interchange the CAN messages and descriptors with the S_MEM and/or L_MEM via its embedded DMA, which is an AXI4 master interface.

The MH-AXI interfaces comply to AMBA 4 ARM Ltd protocol (see ARM IHI 0022E (ID022613)).

See more information about AXI interfaces and AXI parameters in [1].

2.3 MH Configuration

Please note that, before using the X_CAN MH to transmit and receive message, some general configuration of the MH have to be configured and started.

See the following registers

Register (short name)	Description
MH_CFG	Message Handler Configuration register
AXI_PARAMS	AXI parameter register
RX_FILTER_MEM_ADD	RX Filter Base Address register
TX_DESC_MEM_ADD	TX Descriptor Base Address register
MH_CTRL	Message Handler Control register

More details of registers, please see in [1].

An example of these configurations can be found in the following functions in the provided example software.

Name:	`xcand_mh_init(..)`
File:	`../xcand/xcand_mh/xcand_mh.c`

Description:	Include the initialization of following MH functions:
	- TX FIFO Queue
	- TX PRIO Queue
	- RX FIFO Queue
	- RX Filter Config
	
	This function initializes all related features of MH and allocate the memory which is used by X_CAN, by calling other sub-functions such as
	`xcand_mh_set_global_config()`
	`xcand_mh_set_tx_fifo_config()`
	`xcand_mh_set_rx_fifo_config()`

Name:	`xcand_mh_set_global_config(..)`
File:	`../xcand/xcand_mh/xcand_mh.c`
Description:	This function configures the following items: - global configuration of MH such as Message Retransmision option and RX FIFO Queue mode (Normal/Continuous) - AXI parameters - the base address of L_MEM
Name:	`xcand_mh_start(..)`
File:	`../xcand/xcand_mh/xcand_mh.c`
Description:	This function starts the MH, which has to be done before the FIFO Queues and FIFO Priority Queues can be started.

Name:
File:
Description:
	`xcand_mh_stop(..)`
	`../xcand/xcand_mh/xcand_mh.c`
	This function stops the MH with a step-wise process by clearing the START bit.

Name:
File:
Description
	`xcand_config_and_start(..)`
	`../xcand/xcand.c`
	This function configures the X_CAN (PRT, MH, IRC), and start the X_CAN (MH, PRT)

3 L_MEM , S_MEM and TX FIFO Queue Descriptor Memory Organisation

3.1 L_MEM (Local Memory)

The MH needs a L_MEM which is connected via MEM_AXI interface.
Up to 64Kbytes can be used as L_MEM. The L_MEM size depends only on:

- the number of TX FIFO Queue
- the number of TX Priority Queue slot
- the number of RX filter element and reference/mask pair

The L_MEM is needed for:

- performing the TX-SCAN and the RX filtering
- storing the current Header Descriptor and the next Header Descriptor for active TX FIFO Queues
- storing the Header Descriptor for active TX Priority Queues
- storing all RX filter elements and reference/mask pairs

See following figure as overview about X_CAN architecture.

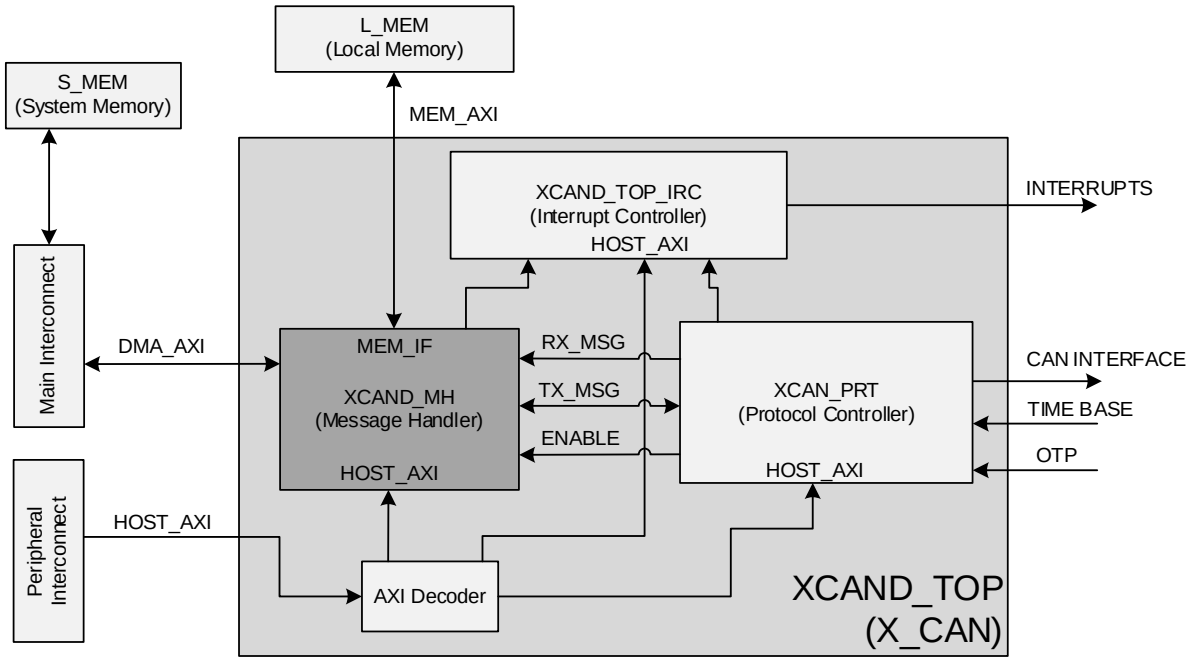


Figure: X_CAN architecture (top view)

3.2 S_MEM (System Memory)

S_MEM is the System Memory and it is used to:

- Define linked-list of TX Descriptors for every TX FIFO Queue
- Define TX Descriptor for every TX Priority Queue slot
- Define linked-list of RX Descriptors for every RX FIFO Queue
- Store RX messages (header and payload data) at the defined RX FIFO Queue address
- Declare TX message data payload attached to TX Descriptor (where the header and other information are defined.)

3.3 TX FIFO Descriptor memory organization

The TX FIFO Queue Descriptors are organized into the **L_MEM** starting at the base address defined in TX_DESC_MEM_ADD.**FQ_BASE_ADDR[15:0]** bit in **TX_DESC_MEM_ADD** register. One memory location of size 8*32bit is required to hold the TX Header Descriptor of every TX FIFO Queues.

The TX Descriptor elements are organized in 32bit word, and therefore any offset would be a multiple of 4.

Memory Base Address	Offset	Name	Bit Field	Description
FQ_BASE_ADDR[15:0]	0x0+0x40*n	TX FIFO Queue n (current/next TX Header Descriptor) (0<= n <N)	Element 0	TX Header Descriptor, see TX descriptor, TX Message and TX FIFO Queue chapters
	0x4+0x40*n		Element 1	
	0x8+0x40*n		Element 2: TS0	
	0xC+0x40*n		Element 3: TS1	
	0x10+0x40*n		Element 4: T0	
	0x14+0x40*n		Element 5: T1	
	0x18+0x40*n		Element 6: T2/TD0	
	0x1C+0x40*n		Element 7: TX_AP/TD1	
	0x20+0x40*n	TX FIFO Queue n (next/current TX Header Descriptor) (0<= n <N)	Element 0	TX Header Descriptor, see TX descriptor, TX Message and TX FIFO Queue chapters
	0x24+0x40*n		Element 1	
	0x28+0x40*n		Element 2: TS0	
	0x2C+0x40*n		Element 3: TS1	
	0x30+0x40*n		Element 4: T0	
	0x34+0x40*n		Element 5: T1	
	0x38+0x40*n		Element 6: T2/TD0	
	0x3C+0x40*n		Element 7: TX_AP/TD1	

Table: Memory organization of the TX descriptors considering N TX FIFO Queue

Every **TX FIFO Queue**, when active, has its current and next Descriptor defined in the **L_MEM** for the TX-SCAN process. This means, for a given TX FIFO Queue, memory space must be double the size. The current and the next TX Header Descriptor are used for the TX-SCAN.

If [n] TX FIFO Queues are defined, the required L_MEM memory is [n] x 2 x TX Descriptor size (32bytes).

Example for TX-FIFO, L_MEM memory usage:
If 8 x TX FIFO Queues are used, then 512 bytes are required in the L_MEM.

4 TX FIFO Queue

The X_CAN supports a maximum of 8 TX FIFO queues. The needed amount of 0..N-1 TX-FIFO Queues can be defined by Software. A TX FIFO Queue is a list of TX messages to be sent in order to the PRT.

Every TX FIFO has its **current** TX Descriptor locally stored in the L_MEM. Furthermore, the **next** TX Descriptor will be uploaded if the current TX Descriptor is selected with the TX Scan for the CAN Bus arbitration.

TX messages are sent according to the priority of the message, namely message ID. This priority are evaluate during the TX Scan against candidates from other active TX FIFO Queues. This means, every TX FIFO Queue will progress differently according to the priority of their TX messages.

The lower number of TX FIFO Queue is the higher the priority. The TX FIFO Queue number 0 has the highest priority. The TX FIFO Queue number 7 has the lowest priority. This means, if there are messages with the same message ID from different TX FIFO Queues during the TX Scan, the message from the lower number of TX FIFO will be sent first.

The TX message payload data is always stored in the S_MEM. The TX FIFO start address, and size are defined in the MH registers.

When the TX FIFO Queue is busy (either running or on hold), its configuration registers are write-protected.

TX Data Container must be aligned on the burst of 8x32bit whenever possible.

In order to transmit a message, the TX Descriptor and Data Container have to be prepared.

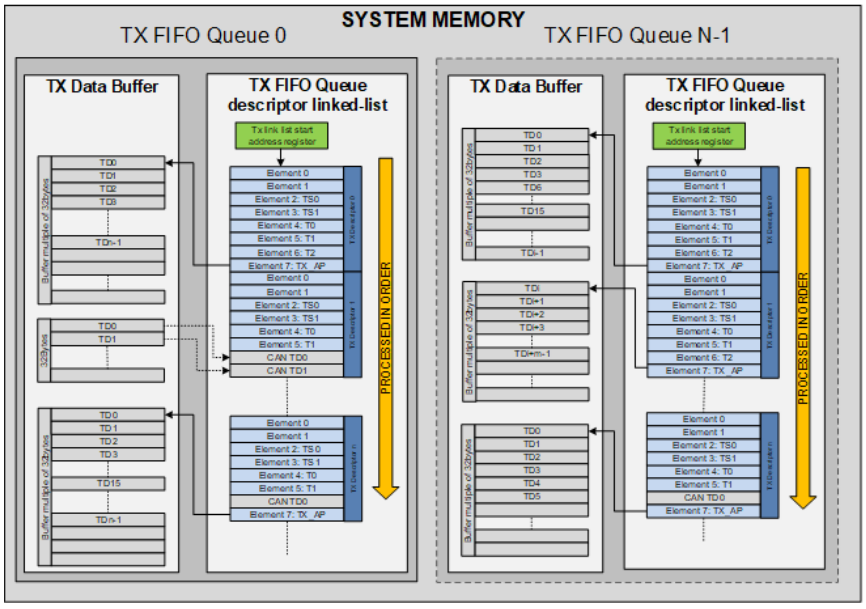


Figure: TX-FIFO Queue memory organization in S_MEM

4.1 TX Descriptor (TX FIFO Queue)

The MH uses TX Descriptors for TX-FIFO Queue. A TX message and a TX Descriptor are defined as follows:

1 TX message = 1 TX Descriptor (+ 1 TX Buffer/TX Data Container).
1 TX Descriptor is made of 8 elements of 32bit word. (e.g. 8 x 32 bits)

The TX Descriptor contains the information for/from the MH such as control bits, interrupt option, the status of the transmission of the TX message, TX message Timestamp, TX message header and TX message payload or the address pointer to the TX message payload.

TX Data Container is the memory space, which is allocated by the SW. This Data Container is used to hold the TX message payload. In most cases, the size of this TX Data Container should be identical to the TX buffer size defined in the TX Descriptor. And these should be big enough for the maximum message payload which will be transmitted.

The following table shows the TX-FIFO Queue Descriptor structure.

TX FIFO QUEUE DESCRIPTOR	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**[24:16]**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**[8:4]**	**3**	**2**	**1**	**0**
DMA Info Ctrl 1	VALID	**HD (set to 1) (Message Header)**	WRAP	**NEXT (set to 0)**	IRQ (Interrupt)	**PQ (set to 0)**	END	CRC[8:0]	FQN[3:0] (FIFO Queue Number)				**Not Used (set to 0)**	**Not Used (set to 0)**		RC4[:0] (Rolling Counter)	STS[3:0] (TX Message Status)			
DMA Info Ctrl 2	**Not Used (set to 0)**					PLSRC	SIZE[9:0] (TX Buffer size)		IN[2:0] (Instance Number)			**Not Used (set to 0)**	**NHDO[9:0] (set to 1)**							**Not Used (set to 0)**
TS0	TS0[31:0] (TimeStamp[31:0])																			
TS1	TS1[31:0] (TimeStamp[63:32])																			
T0	T0[31:0] (TX Message Header Information)																			
T1	T1[31:0] (TX Message Header Information)																			
T2 / TD0	T2[31:0] / TD0[31:0] (TX Message Header Information / First TX Data Payload)																			
TX_AP / TD1	TX_AP[31:0] / TD1[31:0] (TX Payload Data Address Pointer / Second TX Data Payload)																			

Managed by SW and HW

Table: Deatiled list of TX-FIFO Queue Descriptor

Element No.	Element Name	Element description
0	DMA Info Ctrl1	Include the DMA TX Element Descriptor information and the status report of the transaction
1	DMA Info Ctrl2	
2	TS0	**Timestamping 0:** LSB of the 64bits timestamp, the Timestamp[31:0]
3	TS1	**Timestamping 1**: MSB of the 64bits timestamp, the Timestamp[63:32]
4	T0	**TX Message Header 0**
5	T1	**TX Message Header 1**
6	T2 (XL)	**TX Message Header 2**
	TD0 (CC, FD)	**TX Message Data 0:** payload byte 0 – byte 3
7	TX_AP (XL, FD)	**Address Pointer** to TX Data Container (FD: when payload is longer than 4 bytes)
	TD1 (CC)	**TX Message Data 1**: payload byte 4 – byte 7

Table: List with Overview of the TX Descriptor elements

4.2 TX Data Container

The configuration of the Data Container of the TX Descriptor is different, depending on the CAN Frame Format.

CAN Frame Format	TX Data Container usage
Classical CAN (CC)	TX Data Container is **not necessary**. The payload of the frame are described in TD0 and TD1. This means **PLSRC** bit (Payload Source) in Element No. 1 must be set to **0**.
CAN FD	TX Data Container is only necessary when the payload of the frame a longer than 4 bytes. This means, **PLSRC** bit in Element No. 1 is **depends on the payload length**.
CAN XL	TX Data Container is a must. This means **PLSRC** bit in Element No. 1 must be set to **1**.

Table: Data Container for used CAN frame format

4.3 TX Message Header

The TX Descriptor contains also the description of TX Message Header. The structure of the TX Message Header depends on the CAN Frame Format.

The following tables describe 3 different data structures of the TX Message Header.

For Classical CAN:

Tn	Bits	Name	Description/Constraints
T0	[31]	FDF	FD Format: **must be set to 0.**
	[30]	XLF	XL Format: **must be set to 0.**
	[29]	XTD	Extended Identifier
	[28:18]	BaseID [28:18]	Base ID
	[17:0]	ExtID [17:0]	Extended ID
T1	[31]	Reserved	Not Applicable
	[30]	FIR	Fault Injection Request
	[29:27]	Reserved	Not Applicable
	[26]	RTR	Remote Transmission Request
	[25:20]	Reserved	Not Applicable
	[19:16]	DLC[3:0]	Data Length Code
	[15:0]	Reserved	Not Applicable

Table: Classical CAN TX Message Header definition

Note: Classical CAN frames (CBDF, CEDF, CBRF, CERF) require T0.FDF = 0 and T0.XLF = 0. The header consists of T0 and T1.

For CAN FD:

Tn	Bits	Name	Description/Constraints
T0	[31]	FDF	FD Format: **must be set to 1.**
	[30]	XLF	XL Format: **must be set to 0.**
	[29]	XTD	Extended Identifier
	[28:18]	BaseID [28:18]	Base ID
	[17:0]	ExtID [17:0]	Extended ID
T1	[31]	Reserved	Not Applicable
	[30]	FIR	Fault Injection Request
	[29:27]	Reserved	Not Applicable
	[26]	Must be set to 0	Not Applicable
	[25]	BRS	Bit Rate Switch
	[24:21]	Reserved	Not Applicable
	[20]	ESI	Error State Indicator
	[19:16]	DLC[3:0]	Data Length Code
	[15:0]	Reserved	Not Applicable

Table: CAN FD TX Message Header definition

Note: CAN FD frames (FBDF, FEDF) require T0.FDF = 1 and T0.XLF = 0. The header consists of T0 and T1.

For CAN XL:

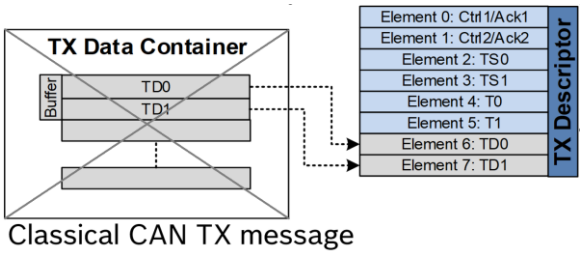
Tn	Bits	Name	Description/Constraints
T0	[31]	FDF	FD Format: **must be set to 1.**
	[30]	XLF	XL Format: **must be set to 1.**
	[29]	XTD	Extended Identifier: **must be set to 0.**
	[28:18]	Priority ID[28:18]	Priority identifier
	[17]	RRS	Remote Request Substitution
	[16]	SEC	Simple Extended Content
	[15:8]	VCID[7:0]	Virtual CAN Network ID
	[7:0]	SDT[7:0]	SDU Type
T1	[31]	Reserved	Not Applicable
	[30]	FIR	Fault Injection Request
	[29:27]	Reserved	Not Applicable
	[26:16]	DLC-XL[10:0]	Data Length Code with CAN XL encoding
	[15:0]	Reserved	Not Applicable
T2	[31:0]	AF[31:0]	Acceptance Field

Table: CAN XL TX Message Header definition

Note: CAN XL frames (XLFF) require T0.FDF = 1, T0.XLF = 1 and T0.XTD = 0. The header consists of T0, T1 and T2.

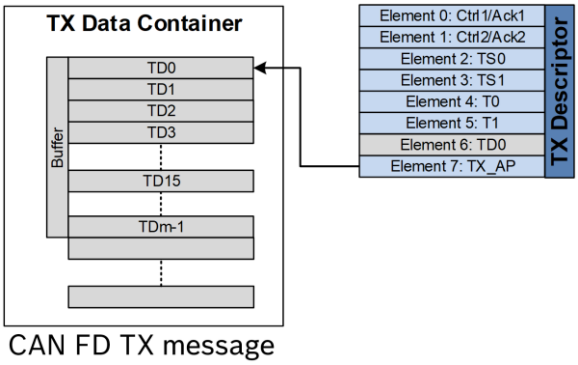
Following figures show the linking between the TX Descriptors and the TX Data Containers for different CAN Frame Format.

CAN CC:



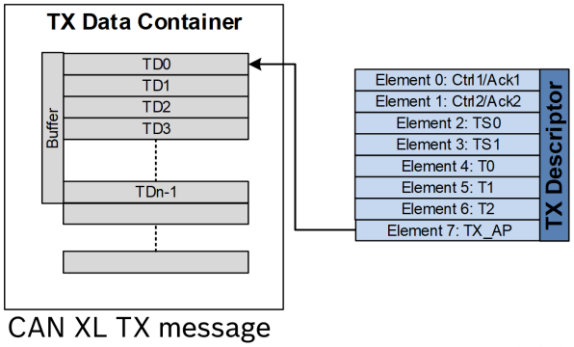
For Classic CAN **there is no additionally** TX Data Container necessary. The payload data is included in the TX Descriptor. (Element 6 TD0 and Element 7 TD1)

CAN FD:



For CAN FD which contains the payload ≤ 4 bytes, the TX Data Container is not required. The payload is included in the TX Descriptor. (Element 6 TD0)

CAN XL:



For CAN XL no payload data can be defined in TX Descriptor. The payload data is always in the Data Container.

Figure: Links between TX Descriptor and TX Data Container

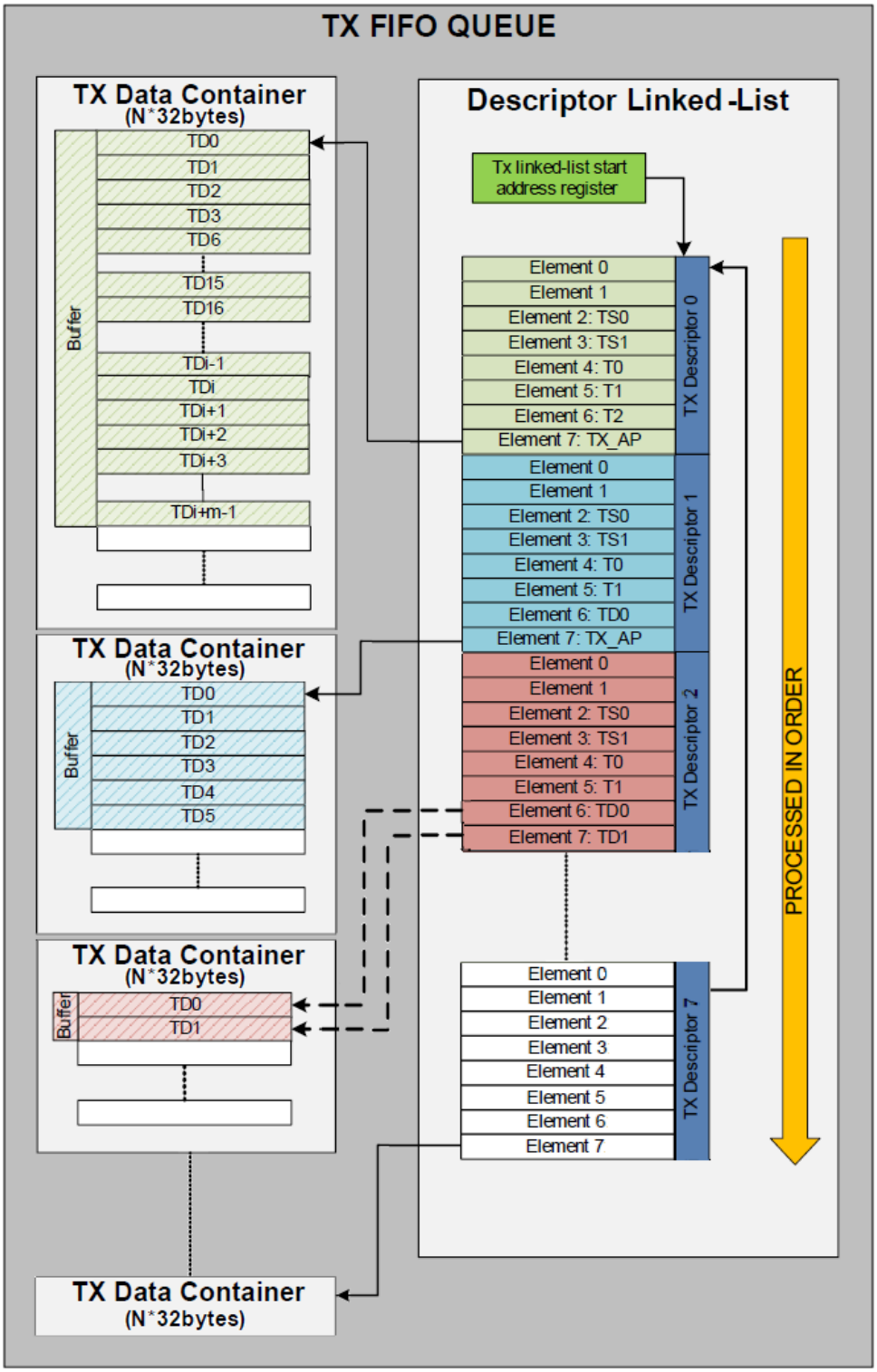


Figure: Struture of TX-FIFO Queue in the S_MEM as used in the example SW

The above figure shows the structure of TX FIFO Queue on the S_MEM which is used in the example software. The TX FIFO Queue has the size of 8 elements. One TX Descriptor is = one TX message.

Note: The X_CAN supports a maximum number of **1024** Tx descriptors.

4.4 Prepare and transmit message with TX FIFO queue

Before the TX FIFO Queue can be used to transmit a TX message, following registers have to be configured.

TX_DESC_MEM_ADD	TX Descriptor Base Address register (address on L_MEM)
TX_FQ_START_ADD [n]	TX FIFO Queue [n] Start Address register (address on S_MEM)
TX_FQ_SIZE [n]	TX FIFO Queue *n* Size register
TX_FQ_CTRL2	TX FIFO Queue Control register 2

Now the **TX FIFO Queue** is configured by the SW (Host), following sequence need to be applied to prepare and transmit a TX message.

1. Check if the **VALID** bit in the Element 0 of the TX Descriptor is already VALID bit = 1 or not (VALID bit = 0). If it is not valid , this means the MH has not yet proceeded this TX Descriptor. If the VALID bit = 0, then the SW- Application may continue writing.
2. The SW needs to prepare/write all TX message information in the elements of the the TX Descriptor.
   Note: Please make sure, that the **VALID** bit in the element 0, shall be set to 1 at the end. This VALID bit = 1 tells the MH that the configuration of the TX message is ready for the the MH.
3. Start the TX FIFO Queue (if not started or on hold), by writing the **START** bit = 1 of the TX FIFO Queue *n* in the **TX_FQ_CTRL0** register.

4.5 Abort TX FIFO Queue

Aborting a TX FIFO Queue does makes sense if it is active (**BUSY** bit of TX FIFO Queue *n* in **TX_FQ_STS0** = 1) otherwise nothing is done. This action can be taken at any time and will terminate with various delays depending on the MH states, Aborting a TX FIFO Queue does not affect the other TX FIFO Queues currently running.

This kind of hard stop on a TX FIFO Queue would be mainly used for:

- **Restart** a TX FIFO Queue when an error or issue has been detected while running properly.
- **Stop** the MH completely.

Aborting TX FIFO Queue can be done by doing the following

1. Write the unlock key sequence (0x1234 and 0x04321, see [1], chapter MH_LOCK) to **MH_LOCK** register. (Writing the **ABORT** bits in **TX_FQ_CTRL1** register is protected.)
2. Abort the TX FIFO Queue by setting **ABORT** bit of the TX FIFO Queue *n* in the **TX_FQ_CTRL1** register.
3. Wait until the **BUSY** bit of the TX FIFO Queue *n* in the **TX_FQ_STS0** register is 0.
4. Clear **ABORT** bit of the TX FIFO Queue *n* in the **TX_FQ_CTRL1** register.

5. Write **ENABLE** bit of the TX FIFO Queue n in the **TX_FQ_CTRL2** register back to 0 to protect the TX FIFO Queue n from being restarted.

See also [1] Chapter Aborting a TX FIFO Queue

4.6 Restart TX FIFO Queue

In case that the SW does not provide new TX messages in time, before the MH gets to the last valid TX Descriptor. This could happen when the SW is too slow compared to the HW (X_CAN-MH).

This means, the MH has transmitted a TX message, and there is no valid TX Descriptor (**VALID** bit = 0), the TX FIFO Queue is then put on hold. (**TX_FQ_STS0.BUSY[n]** =1 and **TX_FQ_STS0.STOP[n]** =1).

Therefore, the **MH_TX_FQn_IRQ[n]** bit in **FUNC_RAW** register is then set, and the interrupt **TX_FQ_IRQ[n]** notifies the SW of such state. In this case, the **UNVALID[n]** bit in **TX_FQ_INT_STS** register is set. This is normal behavior.

The SW can read the address of the TX Descriptor where the MH has stopped in **TX_FQ_ADD_PT[n]** register, and then start the TX FIFO Queue again.

See also the description about restart sequnence in [1], chapter Restarting a TX FIFO queue.

In the example software, this interrupt service can be found in function `xcand_process_irq_func(..)` in file `xcand.c`

4.7 Summary of registers for TX-FIFO Queue

This table lists X_CAN registers for the TX-FIFO Queue.

Register (short name)	Description
TX_DESC_MEM_ADD	TX Descriptor Base Address register (address of L-MEM)
TX_DESC_ADD_PT	TX Descriptor current address pointer register
TX_STATISTICS	TX Message Counter
TX_FQ_STS0	TX FIFO Queue Status register 0
TX_FQ_STS1	TX FIFO Queue Status register 1
TX_FQ_CTRL0	TX FIFO Queue Control register 0
TX_FQ_CTRL1	TX FIFO Queue Control register 1
TX_FQ_CTRL2	TX FIFO Queue Control register 2
TX_FQ_ADD_PT*n*	TX FIFO Queue *n* Current Address Pointer register
TX_FQ_START_ADD[n]	TX FIFO Queue *n* Start Address register (address on S_MEM, where the first Descriptor is.)
TX_FQ_SIZE[n]	TX FIFO Queue *n* Size register

Table: X_CAN registers for the TX-FIFO Queue

More details of registers, please see in [1].

4.8 Example software with functions for TX FIFO Queue

List of C functions in example software which demonstrate the Transmission Handling with TX FIFO Queue for the X_CAN.

Name:	`xcand_an01_tx_rx_fifo_queue(..)`
File:	`../xcand/app_notes/xcand_an01_tx_rx_fifo_queue.c`
Description:	Function illustrates how to use TX FIFO Queue and RX FIFO Queue to transmit and receive messages.
Name:	`xcand_mh_set_tx_fifo_config(..)`
File:	`../xcand/xcand_mh/xcand_mh.c`
Description:	Function configures - the starts address of the TX IFO Queue link list Descriptor (address of first Descriptor) in the S_MEM - the size of the TX FIFO Queue - the ENABLE bit, to enable the TX FIFO Queue.

Name:	`xcand_mh_tx_fifo_enqueue_msg(..)`
File:	`../xcand/xcand_mh/xcand_mh.c`
Description:	Function does the following sequence, - checks if TX FIFO Queue is not full and has space for the message. - prepare the contents of the TX Descriptor and Data Container. - sets VALID bit in TX Descriptor. - if TX FIFO Queue has not been started, sets START bit for the TX FIFO Queue. - updates putindex and rolling counter for the next message.
Name:	`xcand_mh_tx_fifo_start(..)`
File:	`../xcand/xcand_mh/xcand_mh.c`
Description:	Function starts a TX FIFO Queue, sets START bit for the TX FIFO Queue.
Name:	`xcand_mh_tx_fifo_abort(..)`
File:	`../xcand/xcand_mh/xcand_mh.c`
Description:	Function aborts a TX FIFO Queue, and disables a TX FIFO Queue
Name:	`xcand_mh_tx_fifo_is_full(..)`
File:	`../xcand/xcand_mh/xcand_mh.c`
Description:	Function checks if TX FIFO Queue is full or not.
Name:	`xcand_mh_tx_fifo_is_empty(..)`
File:	`../xcand/xcand_mh/xcand_mh.c`
Description:	Function checks if TX FIFO Queue is empty or not.
Name:	`xcand_process_irq_func(..)`
File:	`../xcand/xcand.c`
Description:	Function checks and processes the individual Interrupt Flags For TX FIFO Queue releated: when the UNVALID and SENT interrupts have happened.

5 RX FIFO QUEUE

The X_CAN supports a maximum of 8 RX FIFO Queues. The SW needs to configure the required RX FIFO Queues[n], so that only the RX FIFO Queue number from 0 to N-1 can be used. A RX FIFO Queue is a list of RX Descriptors pointing to an RX Data Container to store the RX messages received by the PRT.

The mechanism to manage RX FIFO Queues is based on the concept of the linked list. Each RX FIFO Queue uses a linked list of RX Descriptors and RX Data Containers. Those containers are used for writing/storing the RX message **Header** and **Data** on the S_MEM, and have the fixed size over the entire RX FIFO Queue.

In order to receive a message, the RX Descriptor and the Data Container have to be prepared.

The example software of the application note demonstrates how to receive message with **RX FIFO Queue in normal mode** (**RX_CONT_DC** bit in **MH_CFG** register = 0), with **single RX Descriptor**.

5.1 RX Descriptor (single Descriptor, normal mode)

- **1 RX message** => 1 RX Descriptor + 1 RX Buffer/RX Data Container (This is as single Descriptor)
- **1 RX Descriptor** is made of 4 elements of 32bit word. They provide following information in the table below.

Note: The Rx FIFO Queue in Continuous Mode usage is described in [1], chapter RX FIFO Queue in Continuous Mode

RX FIFO QUEUE DESCRIPTOR (Normal Mode)	31	30	29	28	27	26	25	[24:16]	[15:12]	[11:9]	[8:4]	[3:0]
DMA info Ctrl 1	VALID	HD (Message Header)	**Not Used (set to 0)**	NEXT	IRQ	**Not Used (set to 0)**		CRC[8:0]	FQN[3:0] (RX FIFO Queue Number)	IN[2:0] (Instance Number)	RC[4:0] (Rolling Counter)	STS[3:0] (TX Message Status)
RX_AP	RX_AP[31:0] (RX Address Pointer)											
TS0	TS0[31:0] (TimeStamp[31:0])											
TS1	TS1[31:0] (TimeStamp[63:32])											

Managed by SW and HW

Table: Deatiled list of RX Descriptor (Normal mode)

Element No.	Element Name	Element description
0	DMA Info Ctrl1	Information for linked-list execution, and the status report of the transaction
1	RX_AP	**Address Pointer** to RX Data Container
2	TS0	**Timestamping 0**: LSB of the 64bits timestamp
3	TS1	**Timestamping 1**: MSB of the 64bits timestamp

Table: List with Overview of the RX Descriptor elements

See also in [1], the chapter RX Descriptor, which explains more in detail the usage of the RX Descriptor register bits.

The following lists shows the element in RX Descriptor which is managed by MH or SW. Please use this information accordingly for your SW driver design.

Note: for Single Descriptor, there is only Header Descriptor, and there is no Trailing Descriptor. Since 1 message = 1 Descriptor.

	SW to write information to MH	SW to read information from MH	
Element Number	***RX Descriptor***	**Header Descriptor**	**Trailing Descriptor**
0	Mandatory	Mandatory	Mandatory in Normal mode
1	Mandatory in Normal mode NA in Continuous mode (must be set to 0)	Mandatory	Mandatory in Normal mode
2	NA (must be set to 0)	Mandatory	NA (must be equal to 0)
3	NA (must be set to 0)	Mandatory	NA (must be equal to 0)

Table: Elements managed by the SW

	MH to write information to SW		MH to read information from SW
Element Number	***Header Descriptor***	**Trailing Descriptor**	**RX Descriptor**
0	Mandatory	Not updated	Mandatory
1	Not updated in Normal mode Mandatory in Continuous mode	Not updated	Mandatory in Normal mode NA in Continuous mode
2	Mandatory	Not updated	NA
3	Mandatory	Not updated	NA

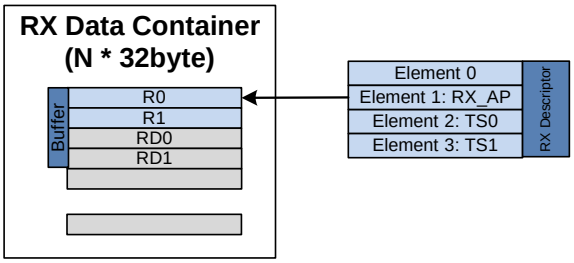
Table: Elements managed by MH

5.2 Single RX Descriptor and Data Container

The needed size of the Data Container changes according to the CAN Frame Format of the received message.

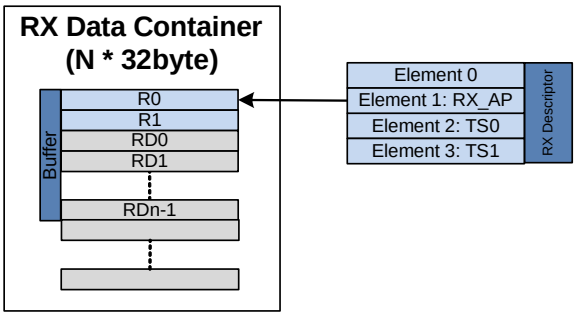
Following figures show the links between the RX Descriptors and the RX Data Containers for different CAN Frame Format.

CAN CC:



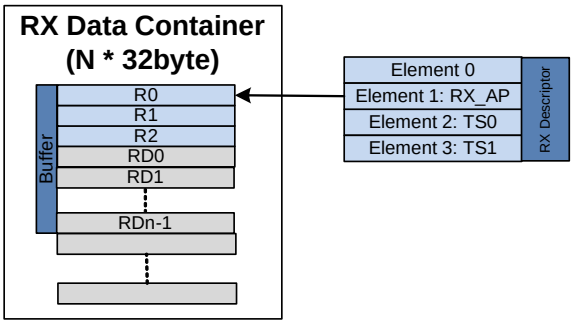
For Classical CAN, header and payload data can be directly written into a 32byte Data Container (N = 1)

CAN FD:



For CAN FD, a larger data buffer is required to hold up to 64byte of payload data and the header message data. In this case, a Data Container of 96byte (N = 3) is allocated to support CAN FD frame format.

CAN XL:



For CAN XL a Data Container size of more than 2048byte (N=65) is required. However, quite some memory space is lost in the Data Container (when configure to support CAN XL payload size) when receiving Classical CAN or CAN FD messages. To solve this issue, multiple RX Descriptors can be used, see next chapter.

Figure: Overview of Links between the RX Descriptors and the RX Data Containers

See more info in [1], chapter RX Single Descriptor.

5.3 RX Message Header

RX Data Container contains the RX Message Header (R0, R1 and R2) and RX Message Data (RDn)of the received frame.

Following tables shows the structure of the RX Message Header for the dedicated CAN protocol type.

For Classical CAN:

Rn	Bits	Name	Source	Description/Constraints
R0	[31]	FDF	CAN	FD Format **(=0)**
	[30]	XLF	CAN	XL Format **(=0)**
	[29]	XTD	CAN	Extended Identifier
	[28:18]	BaseID [28:18]	CAN	Base ID
	[17:0]	ExtID [17:0]	CAN	Extended ID
R1	[31:27]	na	na	reserved
	[26]	RTR	CAN	Remote Transmission Request
	[25:20]	na	na	reserved
	[19:16]	DLC[3:0]	CAN	Data Length Code
	[15:11]	na	na	reserved
	[10]	FAB	MH	Filter Aborted: when set to 1, the RX filtering process was ending before completing with no match
	[9]	BLK	MH	Black List: When set to 1, the RX message filtered belongs to a blacklist
	[8]	FM	MH	Filter Match: When set to 1 one of the filter elements (defined by FIDX[7:0]) has detected a match
	[7:0]	FIDX[7:0]	MH	Filter index: provide the information of the filter index which has been triggered
R2	[31:0]	na	na	reserved

Table: Classical CAN RX Header definition

Note: Classical CAN frames (CBDF, CEDF, CBRF, CERF) can be identified by **R0.FDF** = 0 and **R0.XLF** = 0.

For CAN FD:

Rn	Bits	Name	Source	Description/Constraints
R0	[31]	FDF	CAN	FD Format **(=1)**
	[30]	XLF	CAN	XL Format **(=0)**
	[29]	XTD	CAN	Extended Identifier
	[28:18]	BaseID [28:18]	CAN	Base ID
	[17:0]	ExtID [17:0]	CAN	Extended ID
R1	[31:26]	na	na	reserved
	[25]	BRS	CAN	Bit Rate Switch
	[24:21]	na	na	reserved
	20	ESI	CAN	Error State Indicator
	[19:16]	DLC[3:0]	CAN	Data Length Code
	[15:11]	na	na	reserved

Rn	Bits	Name	Source	Description/Constraints
	[10]	FAB	MH	Filter Aborted: when set to 1, the RX filtering process was ending before completing with no match
	[9]	BLK	MH	Black List: When set to 1, the RX message filtered belongs to a blacklist
	[8]	FM	MH	Filter Match: When set to 1 one of the filter elements (defined by FIDX[7:0]) has detected a match
	[7:0]	FIDX[7:0]	MH	Filter index: provide the information of the filter index which has been triggered
R2	[31:0]	na	na	reserved

Table: CAN FD RX Header definition

Note: CAN FD frames (FBDF, FEDF) can be identified by **R0.FDF** = 1 and **R0.XLF** = 0.

For CAN XL:

Rn	Bits	Name	Source	Description/Constraints
R0	[31]	FDF	CAN	FD Format **(=1)**
	[30]	XLF	CAN	XL Format **(=1)**
	[29]	na	na	reserved
	[28:18]	Priority ID[28:18]	CAN	Priority identifier
	[17]	RRS	CAN	Remote Request Substitution
	[16]	SEC	CAN	Simple Extended Content
	[15:8]	VCID[7:0]	CAN	Virtual CAN Network ID
	[7:0]	SDT[7:0]	CAN	SDU Type
R1	[31:27]	na	na	reserved
	[26:16]	DLC-XL[10:0]	CAN	Data Length Code with CAN XL encoding
	[15:11]	na	na	reserved
	[10]	FAB	MH	Filter Aborted: when set to 1, the RX filtering process was ending before completing with no match
	[9]	BLK	MH	Black List: When set to 1, the RX message filtered belongs to a blacklist
	[8]	FM	MH	Filter Match: When set to 1 one of the filter elements (defined by FIDX[7:0]) has detected a match
	[7:0]	FIDX[7:0]	MH	Filter index: provide the information of the filter index which has been triggered
R2	[31:0]	AF[31:0]		Acceptance Field

Table: CAN XL RX Header definition

Note: CAN XL frames (XLFF) could be identified by **R0.FDF** = 1 and **R0.XLF** = 1.

Note: FAB, **BLK**, **FM** and **FIDX** bits in **R1** are the outputs of RX Filtering if the RX Filter Element is used. If the RX Filter Element is not used, these bits have no usage.

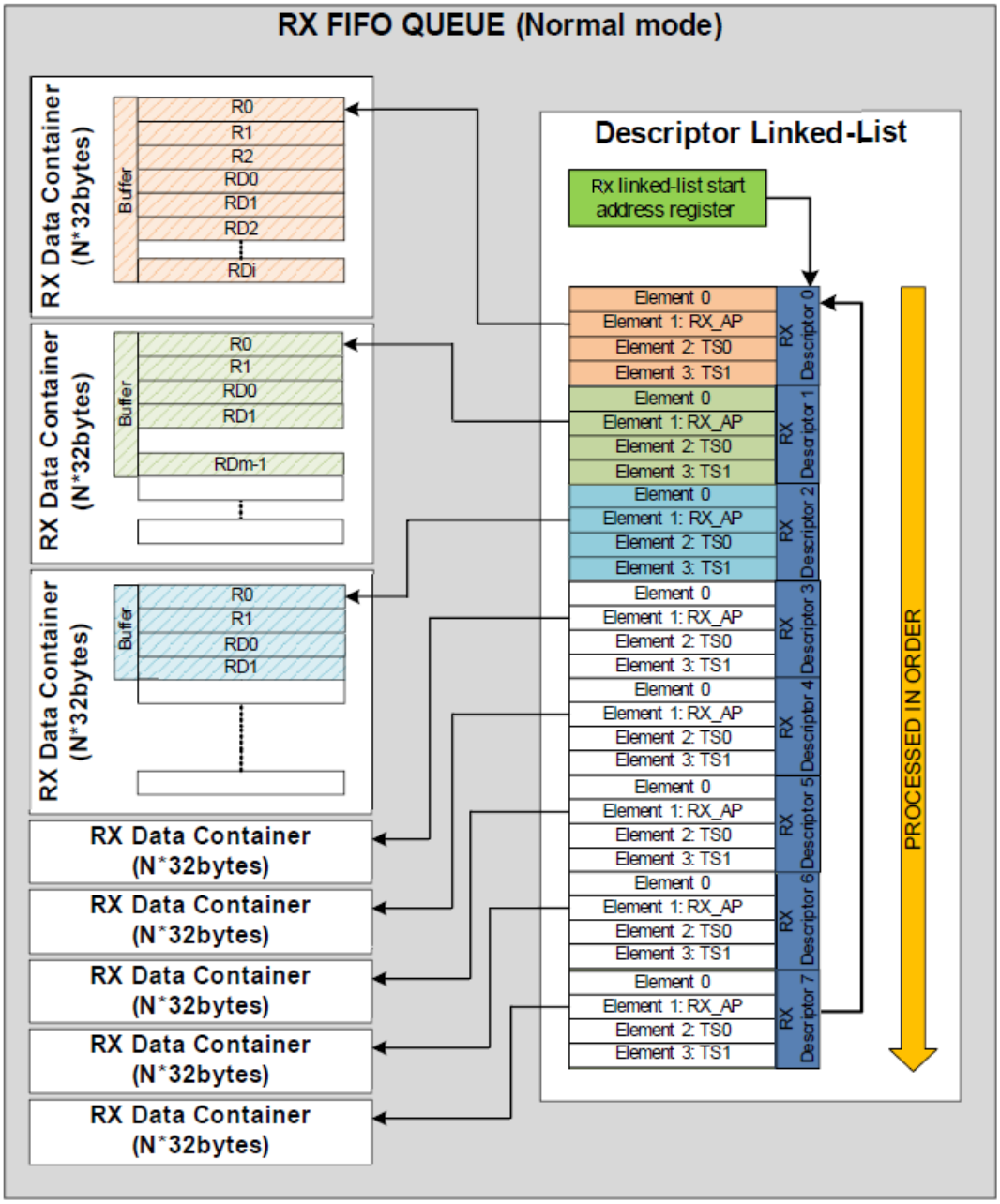


Figure: Memory organization structure of the RX FIFO Queue in Normal Mode, Single Descriptor

The above figure shows the structure of the RX FIFO Queue in Normal Mode on the S_MEM which is used in the example software. The RX FIFO Queue has the size of 8 elements and 1 Descriptor is for 1 RX Message.

The linked list contains the Descriptors, where a Descriptor is defined by several data elements of the same size, the element is 32bit word. A Descriptor is built by the SW, but it will be read and executed by the MH. Every Descriptor is of the same size, pointing to a Data Container into the S_MEM.

Note: The X_CAN supports a maximum number of 1024 RX Descriptor.

5.4 Read received message

Please note, it is assumed that the RX Filters have already been configured. In the example software, the RX Filters are configured as all message is accepted and stored in the RX FIFO Queue[n].

Before RX FIFO Queue can be used to receive a message, following registers have to be configured.

RX_FQ_START_ADD[n]	RX FIFO Queue *n* Start Address register (address on S_MEM, where the first Descriptor is.)
RX_FQ_SIZE[n]	RX FIFO Queue *n* Size register
RX_FQ_CTRL2	RX FIFO Queue Control register 2

Note: The size of Data Container for the received message is defined in RX_FQ_SIZE[n] and this must big enough for **RX Message Header** + **RX Message Data.**

Now the **RX FIFO Queue** is configured, following sequence can be applied to receive and read out a message.

1. The SW check the **VALID** bit in the RX Descriptor Element 0, if it is 1. This means the MH has written the message to that RX Descriptor and the Data Container described in that RX Descriptor.
2. Read the information from the RX Descriptor and Data Container.
3. Write bit **VALID bit to 0**, to notify the MH that the RX Descriptor has been read and proceeded.
4. **Start the RX FIFO Queue** (if not started or on hold), by writing the **START** bit to 1 of the corresponding RX FIFO Queue number in the **RX_FQ_CTRL0** register.

5.5 Abort RX FIFO Queue

Aborting a RX FIFO Queue makes sense if it is active (**BUSY bit** in **RX_FQ_STS0** bit of RX FIFO Queue *n* = 1) otherwise nothing is done. This action can be taken at any time, and will terminate with various delays depending on the MH states. Aborting a **RX FIFO Queue** does not affect the other ones currently running.

This kind of hard stop on a RX FIFO Queue would be mainly used for:

- Restarting properly a RX FIFO Queue when an error or issue has been detected while running
- Stop completely the MH

Aborting RX FIFO Queue can be done by doing the following

1. Write the unlock key sequence to **MH_LOCK** register. (Writing **ABORT** bits in **RX_FQ_CTRL1** register is protected.)

2. Abort RX FIFO Queue by setting **ABORT** bit of the RX FIFO Queue n in the **RX_FQ_CTRL1** register.
3. Wait until the **BUSY** bit of the RX FIFO Queue n in the **RX_FQ_STS0** is 0.
4. Set **ABORT** bit of the RX FIFO Queue n in the **RX_FQ_CTRL1** register back to 0.
5. Set the RX_FQ_CTRL2.ENABLE[n] bit register back to 0 to protect the RX FIFO Queue n from being restarted

See also [1] chapter Aborting a RX FIFO Queue

5.6 Summary of registers

This table lists X_CAN registers for the RX-FIFO Queue.

Register (short name)	Description
MH_CFG	Message Handler Configuration register
RX_DESC_ADD_PT	RX Descriptor current address pointer register
RX_STATISTICS	RX Message Counter
RX_FQ_STS0	RX FIFO Queue Status register 0
RX_FQ_STS1	RX FIFO Queue Status register 1
RX_FQ_STS2	RX FIFO Queue Status register 2 (only for continuous mode)
RX_FQ_CTRL0	RX FIFO Queue Control register 0
RX_FQ_CTRL1	RX FIFO Queue Control register 1
RX_FQ_CTRL2	RX FIFO Queue Control register 2
RX_FQ_ADD_PT*n*	RX FIFO Queue *n* Current Address Pointer register
RX_FQ_START_ADD*n*	RX FIFO Queue *n* Start Address register (address on S_MEM)
RX_FQ_SIZE*n*	RX FIFO Queue *n* Size register
RX_FQ_DC_START_ADD*n*	RX FIFO Queue *n* Data Container Start Address (only for continuous mode)
RX_FQ_RD_ADD_PT*n*	RX FIFO Queue *n* Read Address Pointer (only for continuous mode)

Table: X_CAN registers for RX-FIFO Queue

More details of registers, please see in [1].

5.7 Example software with functions for RX-FIFO queue

List of C example functions which demonstrate Reception Handling with RX FIFO Queue for the X_CAN.

Name:
File:
Description:
	`xcand_an01_tx_rx_fifo_queue(..)`
	`../xcand/app_notes/xcand_an01_tx_rx_fifo_queue.c`
	- Function illustrates how to use TX FIFO Queue and RX FIFO Queue to transmit and receive messages.

Name:
File:
Description:
	`xcand_mh_set_rx_fifo_config(..)`
	`../xcand/xcand_mh/xcand_mh.c`
	Function configures
	- the starts address of the RX FIFO Queue link list Descriptor (address of first Descriptor) in the S_MEM
	- The size of the RX FIFO Queue
	- The ENABLE bit, to enable the RX FIFO Queue.

Name:
File:
Description:
	`xcand_mh_rx_fifo_dequeue_msg(..)`
	`../xcand/xcand_mh/xcand_mh.c`
	Function does the following sequence,
	- checks if RX FIFO Queue is empty or not.
	- reads out a message/contents from RX Descriptor and Data Container to message variable (struct).
	- clears VALID bit in RX FIFO Queue Descriptor.
	- if RX FIFO Queue has not been started, sets START bit for the RX FIFO Queue.
	- updates get index and rolling counter for the next message.

Name:
File:
Description:
	`xcand_mh_rx_fifo_start(..)`
	`../xcand/xcand_mh/xcand_mh.c`
	Function starts a RX FIFO Queue [n], sets START bit for the RX FIFO Queue[n].

Name:
File:
Description:
	`xcand_mh_rx_fifo_abort(..)`
	`../xcand/xcand_mh/xcand_mh.c`
	Function aborts a RX FIFO Queue, and disables a RX FIFO Queue.

Name:
File:
Description:
	`xcand_mh_rx_fifo_is_empty(..)`
	`../xcand/xcand_mh/xcand_mh.c`
	Function checks if RX FIFO Queue is empty or not.