



BOSCH
Invented for life

M_CAN

Modular CAN IP-module

DMU Handling

Application Note M_CAN_AN004

Document Revision 1.0
17.08.2023



Robert Bosch GmbH
Automotive Electronics

LEGAL NOTICE

© Copyright 2023 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	17.08.2023	First version of DMU App Note

Conventions

The following conventions are used within this document:

Register names
 Names of files and directories
 Source code/function names

References

This document refers to the following documents:

Ref	Author	Title
[1]	AE/EY	M_CAN User's Manual
[2]	AE/EY	M_CAN System Integration Guide
[3]	AE/EY	User Manual DMU
[4]	AE/EY	User Manual TSU

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
C	With "C" language programmed functions
CRAM	Central Memory of the SoC, bound to DMA and CPU
MRAM	Message memory of the M_CAN
TSU	Timestamping Unit
TTCAN	Time-Triggered CAN

Table of Contents

1	Overview of DMU integration into M_CAN.....	1
1.1	Host interface for DTM CAN access (AEI Slave)	1
1.2	Host interface for direct MRAM access.....	1
1.3	M_CAN interface for MRAM access	1
1.4	DMU interface for MRAM access.....	2
2	Target of this Application Note	3
3	Introduction to Message Handling with DMU	4
4	DMU configuration and Memory map	6
4.1	M_CAN and DMU Configuration.....	6
4.2	DMU Memory address map and DMU register	6
5	DMU enqueue of TX Element	8
5.1	Enqueue Tx Message	9
6	DMU dequeuing of RX Element.....	10
6.1	Dequeuing of an Rx message.....	11
7	Using Time stamping with DMU.....	12
7.1	Usage of TSU	12
7.2	Configuration of TSU	12
7.3	Reception of timestamped messages	12
7.4	Transmission of time stamped Messages.....	13
8	Example of TX message enqueueing with DMU.....	15
8.1	Tx Message Transmission with DMU.....	15
9	Example of RX message dequeue with DMU and TSU	18
9.1	Rx dequeue Rx Message Transmission with DMU and TSU.....	18
9.2	Hints for using time stamping with TSU	20
10	List of Tables	21
11	List of Figures	22

1.4 DMU interface for MRAM access

The **dmu_mram_aeim** Master Interface connects the DMU to the same external 32-bit Message RAM in parallel to the M_CAN **m_can_aeim** master interface. It is used for pass-through of **mram_aei** accesses and for address redirected accesses via **dmu_aei**.

2 Target of this Application Note

This application note describes the DMU message handling in the **M_CAN version 3.3.1, DMU 1.0.1 and TSU 1.0.0**.

The topics of this Application Note are:

- Introduction to DMU
- DMU message handling
 - Enqueue a message
 - Dequeue a message
- Introduction of timestamping with TSU via DMU
- Example of TX handling with TX-FIFO and DMU
- Example of RX handling with RX-FIFO with DMU and TSU

Important Note:

Software examples delivered with this application note are only for illustration purposes.

3 Introduction to Message Handling with DMU

The DMU uses Virtual Buffers for CAN message transfer. The DMU redirects accesses to these Virtual Buffers dynamically to the MRAM. Redirections are controlled by the FIFO pointers of the **M_CAN**. The DMU supports message transfers from the CRAM to the TX FIFO/Queue and from the RX FIFOs / TX Event FIFO to the CRAM. See table 1 with an overview of the possible MRAM buffer types.

MRAM buffer type	Data transfer direction	Buffer condition	DMU Request signal
TX FIFO/Queue	CRAM →MRAM	not full	dmu_txr
RX FIFO 0	MRAM →CRAM	not empty	dmu_rx0r
RX FIFO 1	MRAM →CRAM	not empty	dmu_rx1r
TX Event FIFO	MRAM →CRAM	not empty	dmu_txer

Table 1: Supported M_CAN message buffers

The **M_CAN** FIFOs have to be configured as described in [1], e.g. base addresses and data field sizes. When the **M_CAN** is setup and running, the CAN messages have to be interchanged via dedicated DMU Virtual Buffers as shown in Figure 2. These sections are bound to the Tx FIFO respective Rx FIFO head elements, i.e. the elements, pointed by **M_CAN PUT_INDEX** respective **GET_INDEX**. The DMU just redirects the accesses to the MRAM.

The data structure of the FIFO elements is described in [1], chapter 2.4 Message RAM. Each DMU Virtual Buffer is dimensioned **to the maximal M_CAN element size**, i.e. **72 bytes for Tx and Rx FIFO elements** respective **8 byte for Tx Event FIFO element**.

Transfer requests of the external DMA controller will be indicated by separate DMU Request signals, one for each Virtual Buffer, see Table 1. The external DMA interchange data will be transferred by a linear access to the dedicated Virtual Buffer.

With this linear data transfer, everything internal is implicitly handled, e.g. that a Tx message will be sent by the M_CAN or that an Rx FIFO element will be acknowledged and disengaged at the **M_CAN**.

For software debugging purposes, the Virtual Buffers can be accessed via debug section (DMU debug), without affecting the dataflow of normal operation.

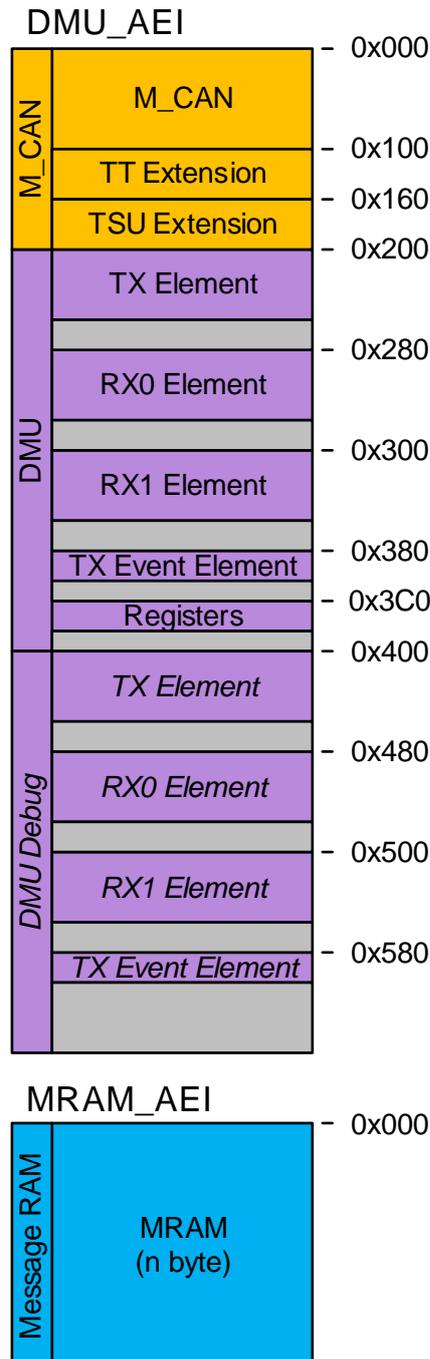


Figure 2: DMU Address Map

4 DMU configuration and Memory map

4.1 M_CAN and DMU Configuration

The M_CAN has to be configured for your target CAN node application. This include all necessary settings to operate as CAN node and is the same configuration as used without DMU usage. This includes e.g. the CAN node protocol type (CAN FD or CAN Classic, CAN Bit timing, the TX/RX-FIFOs and TX-Queue buffer configuration, the filtering, the interrupt usage, etc.

4.2 DMU Memory address map and DMU register

The DMU provides a memory mapped access to the embedded M_CAN with optional TTCAN registers and optional TSU at the first 512 addresses. The following addresses for the DMU are listed, i.e. the indirect addressing of the M_CAN head elements and the configuration registers of the DMU. Data must be accessed via 32 bit (word access).

ADDRESS	SYMBOL	NAME	RESET	Access
0x000 – 0FC		M_CAN address range	See [1]	See [1]
0x100 – 15C		TTCAN Extension	See [1]	See [1]
0x160 – 1FC		TSU Extension	See [2]	See [2]
0x200 – 244		DMU TX Element (18)	MRAM ¹	W ²
0x248 – 27C		Reserved (14)	MRAM ¹	R
0x280 – 2C4		DMU RX0 Element (18)	MRAM ¹	R ²
0x2C8 – 2FC		Reserved (14)	MRAM ¹	R
0x300 – 344		DMU RX1 Element (18)	MRAM ¹	R ²
0x348 – 37C		Reserved (14)	MRAM ¹	R
0x380 – 388		DMU TX Event Element (3)	MRAM ¹	R ²
0x38C – 3BC		Reserved (13)	MRAM ¹	R
0x3C0	DMUCR	DMU Core Release	rrrd dddd	R
0x3C4	DMUI	DMU Internals	0007 0000	RW
0x3C8	DMUQC	DMU Queueing Counter	0000 0000	RC
0x3CC	DMUIR	DMU Interrupt Register	0000 0000	RC1
0x3D0	DMUIE	DMU Interrupt Enable	0000 0000	RW
0x3D4	DMUC	DMU Configuration	0000 0000	RP
0x3D8 – 3FC		Reserved (10)	0000 0000	R
0x400 – 444		DMU TX Element Debug (18)	MRAM ¹	R ³
0x448 – 47C		Reserved (14)	MRAM ¹	R
0x480 – 4C4		DMU RX0 Element Debug (18)	MRAM ¹	R ³
0x4C8 – 4FC		Reserved (14)	MRAM ¹	R
0x500 – 544		DMU RX1 Element Debug (18)	MRAM ¹	R ³
0x548 – 57C		Reserved (14)	MRAM ¹	R
0x580 – 588		DMU TX Event Element Debug (3)	MRAM ¹	R ³
0x58C – 5FC		Reserved (29)	MRAM ¹	R
0x600 – 7FC		Reserved (128)	0000 0000	R

Table 2: DMU address map

In case the application software accesses one of the reserved addresses in the DMU register map (read or write access), the interrupt flag **M_CAN IR.ARA** is set. In addition accesses to reserved addresses are directly signaled to the Host CPU by status signals of the DMU interface.

Regarding the SW implementation the DMU address map with memory segments should be used with the corresponding header file, see M_CAN.h file as an example.

```
#define DMU_TX_ELEMENT_BASE_ADDRESS 0x200 // DMU TX Element base address
#define DMU_RX0_ELEMENT_BASE_ADDRESS 0x280 // DMU RX0 Element base address
#define DMU_RX1_ELEMENT_BASE_ADDRESS 0x300 // DMU RX1 Element base address
#define DMU_TXE_ELEMENT_BASE_ADDRESS 0x380 // DMU TXE Element base address
```

Figure 3: DMU address map segments for M_CAN.h file

The **address calculation** for the DMU r/w access operation is:
Base address of M_CAN module + DMU address segment address

The **M_CAN.h** also include the register positions for DMUIR.

5 DMU enqueue of TX Element

Overview

The TX Virtual Buffer of the DMU allows to enqueue messages to the Tx FIFO/Queue of the M_CAN in a simplified way. Both queue types (Fifo or Queue) of the M_CAN are supported.

The CAN message format of the **DMU TX Element** is the format described in [1], chapter 2.4.3 Tx Buffer Element. See following figure as an overview about TX Buffer Element.

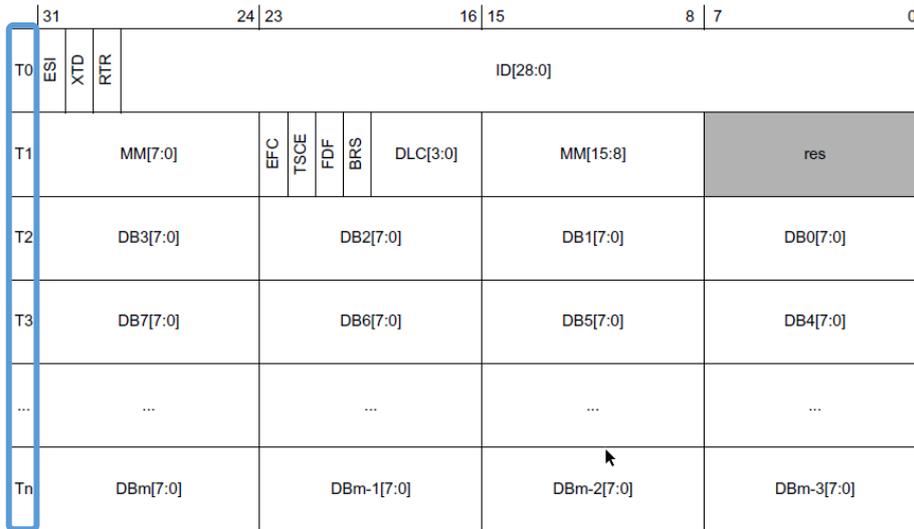


Figure 4: TX buffer element

Start Address

For each message that is transferred to the DMU, the DMU expects the first write access to be to the first address (called start address) of the TX Element section.

Trigger Address

The last element word Tn of the particular CAN Message, with $n \in \{1, 2, 3, 4, 5, 6, 7, 9, 13, 17\}$, which is written by the external DMA controller. This is called the Trigger Address.

5.1 Enqueue Tx Message

To enqueue a Tx message, a strict write access sequence has to be processed. The SW Application or the external DMA controller has to start by writing the **T0** word of the CAN message to the first address of the **TX Virtual Buffer (Base address + 0x200)**, which is called the **start address**. This **triggers the DMU** to expect a CAN Tx message transfer. The complete CAN message has to be transferred linearly to the ascending and consecutive addresses (word by word) by the external DMA controller.

The DMA transfer is complete by writing the last element word **T_n** of the CAN Message (with $n \in \{1, 2, 3, 4, 5, 6, 7, 9, 13, 17\}$) which is the **Trigger Address**. The last required write access to the TX Element Section for a particular CAN message **triggers the scheduling** of that TX message element inside of the M_CAN.

The DMU derives the **Trigger Address** from the **RTR bit** in **T0**, the **FDf bit** in **T1**, and the **Data Length Code (DLC)** in **T1**. The DMU derives the Trigger Address **dynamically for each message**, depending on the **individual payload (-> DLC value)** to be transferred. In case of a CAN message with DLC = 4 the Trigger Address is **T2**.

After the completion of the first enqueued message scheduling the next access to the DMU TX Element section is started always at the same **start address of the DMU TX Element (base address + 0x200)**.

The **DMU TX Element handling** is also compatible to DMA with a fixed transfer length/size see flag **DMUIR.TXEWATA**. In this case of fixed DMU transfer length the DMU TX handler will ignore all data above the payload size of the CAN message. The DMU accepts only the CAN message data until end of payload of the CAN message and transfers it to **M_CAN**. The DMA data which is transferred above the payload size address (-> DLC Value) to **DMU TX Element section** is discarded and not transferred to **M_CAN**.

Status flags in the Interrupt Register **DMUIR** provide detailed information, when one of the access rules is violated. It is highly recommended, to carefully check all **DMUIR.TXExxx** status flags during runtime for unintended activations.

It is recommended to monitor the following interrupt flags, which should never be active during the normal operation. You need to use a interrupt function which is monitoring the DMUIR Interrupt registers.

- **DMUIR.TXEEIW:** TX Event Element Illegal Write
- **DMUIR.TXEEIAS:** TX Event Element Illegal Access Sequence
- **DMUIR.TXEEID:** TX Event Element Illegal Dequeuing
- **DMUIR.TXEENSA:** TX Event Element Not Start Address

6 DMU dequeuing of RX Element

Overview

The RX Virtual Buffer of the DMU allows to dequeue messages to the Rx FIFO 0/1 of the M_CAN in a simplified way.

The CAN message format for DMU RX Element 0/1 is the format described in [1], chapter 2.4.2. Rx Buffer and FIFO Element. See following figure as an overview about RX Buffer and FIFO Element.

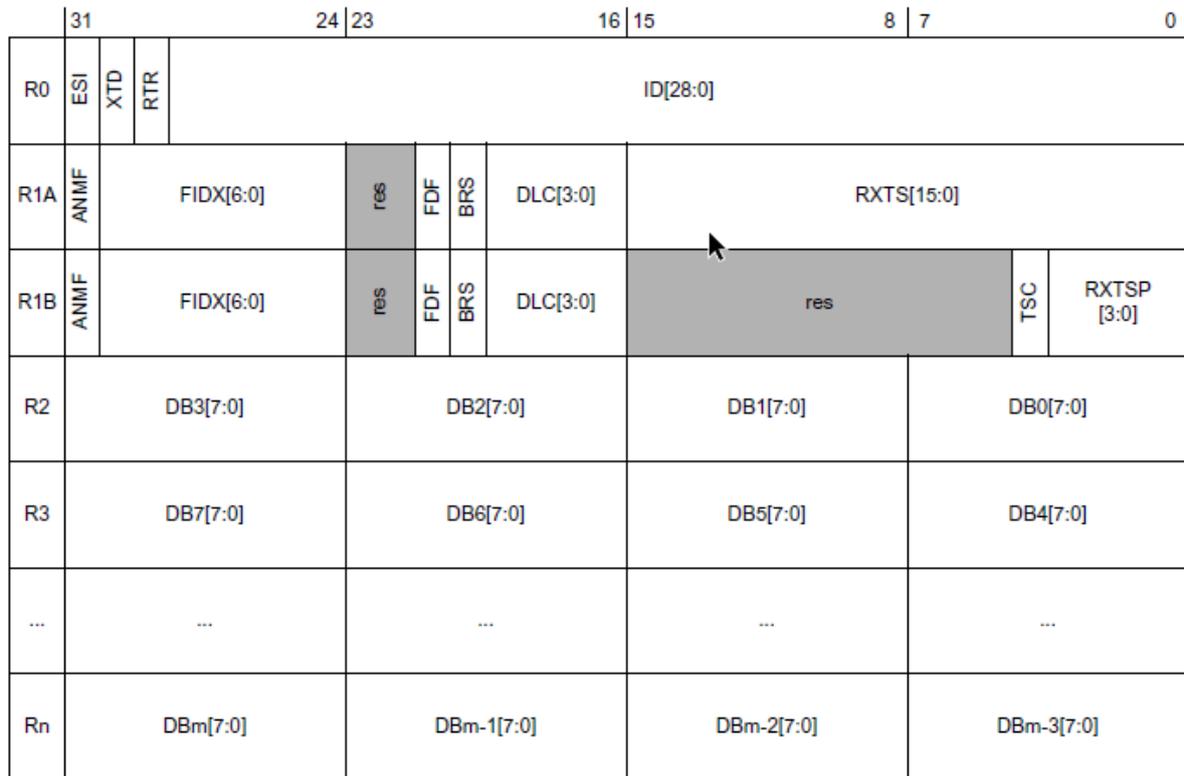


Figure 5: RX Buffer / FIFO Element

The DMU RX Element behaves similar to the previously described TX Element. But the main differences are the transfer direction (**MRAM -> CRAM**) and that the transfer size is **constant/fixed** (according M_CAN:RXESC.FnDS) and does not dynamically change by the **DLC value** of a given CAN message.

Start Address

The RX Element section addresses for RX Element 0/1 (R0: **base address + 0x280** and R1: **base address + 0x300**) are the start addresses.

Trigger Address

The trigger address is derived by the configuration of the dedicated M_CAN **Rx FIFO Element Size**. The Rx Buffer / FIFO Element Size is configured with the **M_CAN:RXESC.F0DS[2:0]** and **M_CAN:RXESC.F1DS[2:0]** registers. The trigger address doesn't change with the DLC value of the CAN message.

6.1 Dequeuing of an Rx message

Dequeuing of an Rx message with the DMU RX Element should only be started, if this is requested by the DMU. The DMU signals **dmu_rx0r** and **dmu_rx1r** indicate the active request. This means that the **dmu_rx0r / dmu_rx1r** signals must be checked for active readiness before the dequeuing process starts.

An Rx message is dequeued by a strict read access sequence. It has to start by reading the R0 word of the CAN message from the first address of the **DMU RX Element (R0: base address + 0x280 and R1: base address + 0x300)**, which is called the start address. This triggers the DMU to expect a CAN Rx message transfer. The CAN message has to be transferred by a linear addressing to ascending and consecutive addresses, word by word. The transfer is completed by reading the **Trigger Address R_n, with $n \in \{3, 4, 5, 6, 7, 9, 13, 17\}$** .

The Trigger Address is derived from the configuration of the dedicated M_CAN Rx-FIFO **n Element Size**. E.g. if **M_CAN:RXESC.F0DS = 0** and **M_CAN:RXESC.F1DS = 7**. The Trigger Address of DMU RX 0 Element is 0x28C (Rx FIFO element word R3) and Trigger Address of DMU RX 1 Element is 0x344 (Rx FIFO element word R17). After the dequeuing process is completed, the next expected access to the DMU Element can be started at the start address of the DMU RX Element.

The following flags **DMUIR.RXnExxx** should be monitored and they should be never active during normal operation. An interrupt routine has to be used for this purpose which is monitoring the DMUIR interrupt registers:

- **DMUIR.RX0,1EIO:** RX Element Illegal Overwrite
- **DMUIR.RX0,1EIW:** RX Element Illegal Write
- **DMUIR.RX0,1EIAS:** RX Element Illegal Access Sequence
- **DMUIR.RX0,1EID:** RX Element Illegal Dequeuing
- **DMUIR.RX0,1ENSA:** RX Element Not Start Address

7 Using Time stamping with DMU

The M_CAN supports HW - time stamping. For the time stamping in this application the TSU unit is used, which is an Add-On of the M_CAN. The hardware time stamping is supported according CiA 603. See more info about TSU in [5].

The target of this application note is focused to use the combination of DMU with timestamping.

7.1 Usage of TSU

The TSU is enabled with the **CCCR.UTSU register = '1'**. In case **CCCR.UTSU = '0'** (default value), the **M_CAN's** internal 16-bit timestamp counter is used.

7.2 Configuration of TSU

The TSU is configured with the TSCFG registers Register, the TSCFG is only writeable while input `m_can_cce = '1'`. The following register bits must be configured:

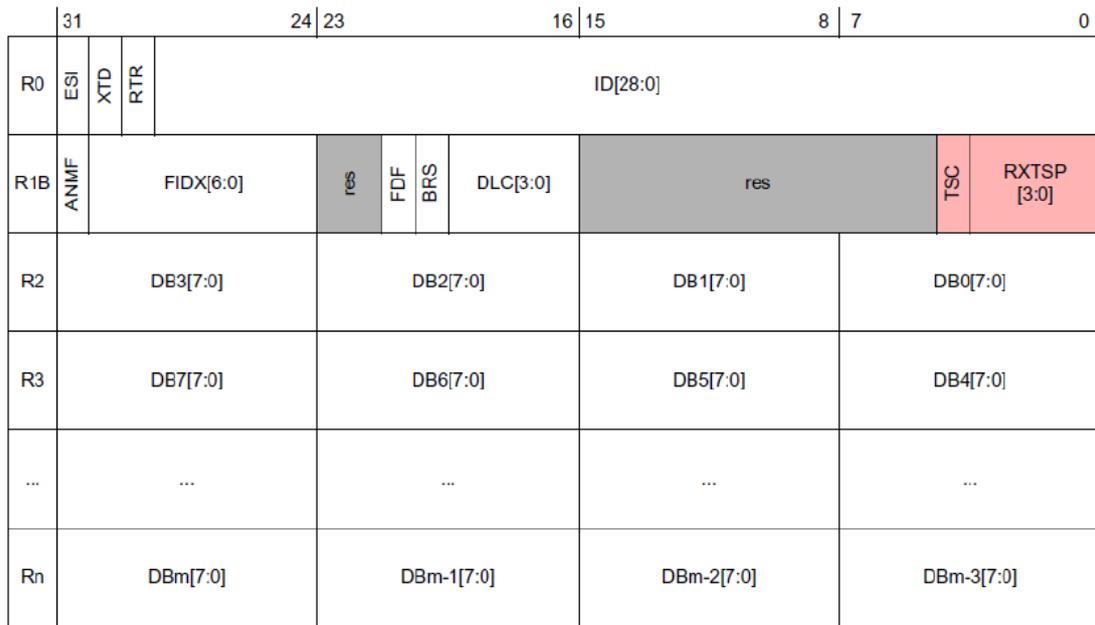
- TBPRES[7:0] Timebase Prescaler
- SCP Capturing Position
- TBCS Timebase Counter Select
- TSUE Timestamp Unit Enable

See C-function [tsu_config\(\)](#) in the application SW code example how the TSU configuration can be done.

7.3 Reception of timestamped messages

For reception of time stamped messages, a **Standard Sync Message/Extended Sync Message ID Filter Element** must be set with the **S0.SSYNC = '1'** and **F1.ESYNC = '1'**. Without an active filter, the message data are not received.

In case the M_CAN is used together with the TSU, the Rx Buffer and FIFO Element is modified according to following figure.



I

Figure 6: Rx Buffer and FIFO Element with TSU enabled (CCCR.UTSU = '1')

For message reception, the value of `m_can_tsp[3:0]` is stored on position **R1B.RXTSP[3:0]** of the M_CAN's Rx Buffer or Rx FIFO element, which the messages with timestamp capturing are stored. **R1B.TSC** is set to '1' when a time stamp has been captured by the TSU and **R1B.RXTSP[3:0]** holds a valid time stamp pointer.

See more info in [3] chap. 4.2.3 and how to use Tx Event FIFO Element with TSU.

7.4 Transmission of time stamped Messages

To support hardware time stamping by the TSU, bit **T1.TSCE** is added to the M_CAN's TX Buffer element as shown in following Figure 7 below.

When bit **T1.TSCE** (Timestamp Capture Enable) is set, a successful transmission of the Tx Buffer element triggers the capture of the TSU's timebase value as time stamp.

	31		24	23		16	15		8	7		0
T0	ESI	XTD	RTR	ID[28:0]								
T1	MM[7:0]			EFC	TSCE	PDF	BRS	DLC[3:0]	MM[15:8]		res	
T2	DB3[7:0]			DB2[7:0]				DB1[7:0]		DB0[7:0]		
T3	DB7[7:0]			DB6[7:0]				DB5[7:0]		DB4[7:0]		
...		
Tn	DBm[7:0]			DBm-1[7:0]				DBm-2[7:0]		DBm-3[7:0]		

Figure 7: Modified Tx Buffer Element

8 Example of TX message enqueueing with DMU

The C-function `can_an004_dmu_tx_enqueue()` demonstrates the principle of TX message enqueueing with the DMU.

This sample C-function uses 2 x M_CAN nodes in a CAN-FPGA demonstrator. For this demo, CAN messages are transmitted from node **M_CAN_0** with DMU to node **M_CAN_1** with DMU. The M_CAN module nodes are configured as “normal” node as without DMU.

TX-node: The TX node (**M_CAN_0**) is configured as TX-FIFO in Queue mode. CAN-FD TX messages are pre-defined in a C-Array.

At the TX-node, the DMU is used to transfer/enqueue CAN message to the CAN bus.

The application SW or the DMA controller writes/enqueues CAN messages to the TX Element of the DMU (Figure 2). The DMU redirects these CAN messages to the MRAM. When the application SW or the DMA controller writes until the Trigger Address, the M_CAN transmits the CAN messages to the CAN bus.

RX-node: The RX node (**M_CAN_1**) is configured as RX-FIFO and use also the DMU. In this example the received CAN messages are printed out to the console of the FPGA demonstrator.

8.1 Tx Message Transmission with DMU

Following steps are required to transmit a message with a DMU:

1. Configure your DMA controller for a word (four bytes -> 32bit) access. The transfer size should be calculated according to the payload (DLC value of the CAN message) of the Tx message. If the DMA supports only fixed transfer length/size, then this is also possible. In this case, transfer length must be the length of the maximum Tx message (incl. max. payload), which should be transmitted.

The write address destination location of the DMA must be calculated as follows:

Start Address: *M_CAN Module address + DMU_TX_ELEMENT_ADDRESS (0x200).*

2. The DMA controller has to transfer/enqueue the Tx message from the source address (CRAM) to the destination address, which is the start address. The Tx message must be a 32-bit-word as described in the in [1], chapter 2.4.3 Tx Buffer Element.
3. The signal **dmu_txr** must be checked for readiness before the SW application or the DMA controller transfers/enqueues Tx message to DMU virtual buffer.
4. The enqueue process can only be started when the DMU Request signal **dmu_txr** is set. This signal is set and cleared by the DMU.

- The enqueue process should be monitored via a C-interrupt function, which checks the Interrupt DMUIR.TXExxx status flags. In case of an error the enqueue processing must be stopped and the error handling must be started.

The DMU redirects the Tx messages, which are written to its TX Element virtual buffer, to the MRAM. After the Trigger Address is reached, the M_CAN considers the arbitration among Tx messages internally in the MRAM and externally with the messages on the CAN bus. Then, the messages are sent out according to their Message ID.

The Application Code C-functions with DMU have different switches and additional functions integrated to handle the DMU. The following parameter switches are used:

- **USE_DMU:** If the DMU is used, then this define/parameter must be '1'. /* defined in global_defines.h */
- **USE_VARIABLE_PAYLOAD_BY_DLC:** Value = '1', if CAN Message DLC value is used for the TX transfer. Value = '0', if a fixed size for the Tx transfer is used.

Software Examples

Table 1 lists C - functions which are used to demonstrate Tx Enqueue operation. The functions are provided with this application note.

Table 1: List of C functions which are used for Tx Enqueue demonstration

Name:	<code>m_can_an004_dmu_tx_enqueue(..)</code>
File:	<code>../appnote/app_note_004_dmu_TX_enqueue_handling.c</code>
Description:	Demonstrates the principle of TX enqueueing with the DMU. This sample C-function uses 2 x M_CAN nodes in a CAN-FPGA demonstrator. For this demo CAN messages are transmitted from node M_CAN_0 with DMU to node M_CAN_1 without DMU . The M_CAN module nodes are configured as "normal" as above explained.
Name:	<code>m_can_tx_buffer_init(..)</code>
File:	<code>../m_can/m_can.c</code>
Description:	This 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:	<code>m_can_tx_write_msg_to_tx_buffer(..)</code>
File:	<code>../m_can/m_can.c</code>
Description:	Function to copy Tx message to Virtual DMU Tx Buffer. The TX Buffer elements T0 ,T1 and T2 are adjusted (into the <code>dmu_word</code> variable) for the subsequent DMU memory write process. This Function transfers every T0 ,T1 and T2 buffer elements to DMU virtual buffer.

Name: `m_can_tx_fifo_queue_msg_transmit(..)`

File: `../m_can/m_can.c`

Description: The function includes a switch parameter **USE_DMU** inside the function.

For M_CAN without DMU: (If `USE_DMU = 0`), function copies Tx message to FIFO/Queue on MRAM, and requests transmission. Function only makes these when the Tx FIFO is not full.

For M_CAN with DMU: (If `USE_DMU = 1`), functions copies TX messages to DMU virtual buffer. The DMU redirect the messages to FIFO/Queue on MRAM. When the application SW or the DMA controller copies the message until the Trigger Address, the M_CAN transmits the CAN messages to the CAN bus.

Function only makes the copying when the Tx FIFO is not full. This is done indirectly by checking bit TXR in the register DMUI. The behaviour of TXR bit is similar to the `dmu_txr` signal, which is used to trigger an external DMA access.

9 Example of RX message dequeue with DMU and TSU

The C-function [m_can_an004_dmu_rx_with_time_stamp](#) demonstrates the principle of RX dequeuing with the DMU. This example shows also, the usage of timestamping with the TSU.

This sample C-function uses **2 x M_CAN** nodes in a CAN-FPGA demonstrator. As above CAN messages are transmitted from node **M_CAN_0 with DMU** to node **M_CAN_1 with DMU**, which is the receiving node. The M_CAN module nodes are configured as “normal” node as node without DMU.

This example also shows the usage of the TSU. The TSU with the hardware time stamping must be configured according the description in the TSU user manual[5].

Tx-node: The Tx node (**M_CAN_0**) is configured as **Tx-FIFO**. CAN-FD Tx messages are pre-defined in a C-Array.

At the TX-node, a DMU is used to transfer CAN message to the CAN bus. The application SW or the DMA controller writes/enqueues CAN messages to the TX Element of the DMU (Figure 2). The DMU redirects these CAN messages to the MRAM. After that, the M_CAN implicitly transmits the CAN messages to the CAN bus.

RX-node: The RX node (**M_CAN_1**) is configured as RX-FIFO and use also DMU. . In this example the received CAN messages are printed out to the console of the FPGA demonstrator.

At the receiving node the TSU is enabled. The internal time base counter is enabled the timestamp is captured for Sync at EOF (End of Frame) and the Timebase prescaler TSCFG.TBPRES[7:0] is set to '0'.

To configure the M_CAN for reception of timestamped messages, a Standard/Extended Message ID Filter Element has to be set up by configuration of **S0.SSYNC = '1'** respectively **F1.ESYNC = '1'**.

9.1 Rx dequeue Rx Message Transmission with DMU and TSU

Following steps are required to receive a message with M_CAN + DMU:

1. Configure a DMA controller for a word (four bytes -> 32bit) access. For the Rx transmission the transfer size is fixed. The DMA transfer length/size must be the length of the complete **RX FIFO 0/1**. For e.g. in case of a RX-Fifo 0/1 with 64 bytes then transfer size must also 64byte, which are 16 word (32Bit) for the DMA.

The read address of the DMA source location must be calculated as follows:

Start Address: *M_CAN Module address + DMU_RX_ELEMENT_ADDRESS*
(R0: base address + 0x280 and R1: base address + 0x300)

2. The DMA controller has to transfer the Rx message from the source address (start address of DMU Virtual Rx-Element) to the destination address (CRAM).

The Rx message must be a 32Bit word as described in the in [1], chapter 2.4.2 Rx Buffer Element.

3. The dequeue process should only be started when the DMU Request signal **dmu_rxr0** (for FIFO0) / **dmu_rxr1** (for FIFO1) or bit RX0R / RX1R in register DMUI) is 1. Otherwise, exception recovery will start (in [3], chapter 3.4 Exception Recovery at DMU Virtual Buffer). This signal is set and cleared by the DMU
4. The dequeue process should be monitored via a C-interrupt function, which checks the Interrupt **DUMIR.RXxxx** status flags. In case of an error the dequeue processing must be stopped and the error handling must be started.
5. The DMU transfer the messages from the MRAM to the DMU Rx-Element Virtual Buffer. The application SW or the external DMA controller must transfer the received message from RX-Element section to the CRAM until the complete Rx-FIFO size is transferd.

The Application Code C-functions with DMU have different switches and additional functions integrated to handle the DMU. The following parameter switches are used:

- **USE_DMU:** If the DMU is used, then this define/parameter must be '1'. /* defined in global_defines.h */

Software Examples

Table 2 lists C - functions which are used to demonstrate Tx dequeue operation. The functions are provided with this application note.

Table 2 : List of C functions which are used for Rx dequeue demonstration

Name:	<code>m_can_an004_dmu_rx_with_time_stamp (..)</code>
File:	<code>../app_notes/app_note_004_dmu_RX_dequeue_with_time_s</code>
Description:	<code>tamp_handling.c</code>
Description:	<p>Demonstrates the principle of RX dequeuing with the DMU with the usage of TSU. This sample C-function uses 2 x M_CAN nodes in a CAN-FPGA demonstrator.</p> <p>TX_Node (M_CAN_0): configured for transmission (TX_FIFO), transmits CAN FD frames (messages are filled with data in a while-loop) using a time stamp from the external TSU.</p> <p>RX_Node (M_CAN_1): configured for reception (RX_FIFO). All received messages (Messages are received in while-loop) as well as the time stamp from the TSU and the time stamp from the RX message are printed on the std_out.</p>
Name:	<code>m_can_TSU_set_usage (..)</code>
File:	<code>../m_can/m_can.c</code>
Description:	Enable or Disable the TSU usage with the M_CAN

Name:	<code>tsu_config (..)</code>
File:	<code>../tsu/m_can.c</code>
Description:	Configure the TSU. Parameter configuration with function <code>tsu_config()</code> Parameters: TSU enable / TimeBase Counter Select / Select Capture Position / TimeBase pre-scaler

9.2 Hints for using time stamping with TSU

See following steps and hints to use hardware time stamping:

1. For the usage of hardware time stamping the TSU is needed. The `CCCR.UTSU` must be set (=1), if TSU is enabled. See C-function `m_can_TSU_set_usage()`
2. The payload for messages with timestaming is reduced to maximum 48 Bytes with `DLC =14`, because otherwise overwrite occur. This is needed because of the timestamping data are inserted into the TX-Frame at `R1B.RXTSP[3:0]` for RX-Buffer and FIFO 0/1 configuration. For TX Event-FIFO it is inserted into `E1B.TXTSP[3:0].I`
3. Configure the TSU according to our application needs. See C-Function `tsu_config ()`, how to configure the TSU.
4. Set and configure filter for your RX configuration (FIFO0 / FIFO1). See C-functions `m_can_global_filter_configuration()`, `m_can_filter_init_standard_id()` or `m_can_filter_init_extended_id()`.
5. Write ID filter (Standard/Extended) to message RAM. For Standard ID Filter see `m_can_filter_write_standard_id()` and `S0.SSYNC = '1'` must be set.

For extended ID Filter see `m_can_filter_write_extended_id()` and `F1.ESYNC = '1'` must be set.

10 List of Tables

TABLE 1: LIST OF C FUNCTIONS WHICH ARE USED FOR TX ENQUEUE DEMONSTRATION..... 16
TABLE 2 : LIST OF C FUNCTIONS WHICH ARE USED FOR RX DEQUEUE DEMONSTRATION 19

11 List of Figures

- FIGURE 1: BLOCK DIAGRAM M_CAN WITH DMU AND TSU 1
- FIGURE 2: DMU ADDRESS MAP 5
- FIGURE 3: DMU ADDRESS MAP SEGMENTS FOR M_CAN.H FILE 7
- FIGURE 4: TX BUFFER ELEMENT 8
- FIGURE 5: RX BUFFER / FIFO ELEMENT 10
- FIGURE 6: RX BUFFER AND FIFO ELEMENT WITH TSU ENABLED (CCCR.UTSU = '1') 13
- FIGURE 7: MODIFIED TX BUFFER ELEMENT 14