M_CAN

M_CAN add-ons

White Paper

Revision: 1.0.0

15.03.2019

Robert Bosch GmbH

Automotive Electronics (AE)

LEGAL NOTICE

REVISION HISTORY

Revision	Date	Notes
1.0.0	15.03.2019	Initial release

References

[1] Time-stamping of CAN frames, CAN Newsletter 2/2017: https://can-newsletter.org/uploads/media/raw/3ebef2c5e26451ab4aa386e0e67ce5f2.pdf

Terms and Abbreviations

This document uses the following terms and abbreviations:

Term	**Meaning**
CAN	Controller Area Network
CRAM	Central RAM; common memory of DMA and CPU Core
DMA	Direct Memory Access
DMU	DMA Interface Unit; Add-on of M_CAN of BOSCH
EoF	End of Frame; distinct position at end of CAN frames
GTM	Generic Timer Module
M_CAN	CAN Communication Controller; IP module of BOSCH
MRAM	Message RAM; local memory of M_CAN
NoC	Network on Chip
NVM	Non-Volatile Memory; e.g. Flash Memory
SoC	System on Chip
SoF	Start of Frame; distinct position at start of CAN frames
TCM	Tightly Coupled Memory
TSU	Time Stamping Unit; add-on for M_CAN of BOSCH

Table of Contents

1 Overview

To support modern CAN networks, Bosch has extended the functionality of M_CAN in form of two hardware add-ons. The DMU add-on allows reduction of CPU load by offloading the transport of CAN messages to a DMA controller. The TSU expansion module enables hardware based and AUTOSAR compatible time synchronization with the best possible precision.

1.1 DMU add-on

1.1.1 Motivation to use a DMU

When exchanging CAN messages between the CPU Core and the M_CAN, there are some issues to consider, especially for complex SoCs. Due to the high complexity of modern SoC architectures, the OnChip communication paths are divided into several domains of different performance. The bridging between domains additionally slows down performance, e.g. due to clock-domain crossings.

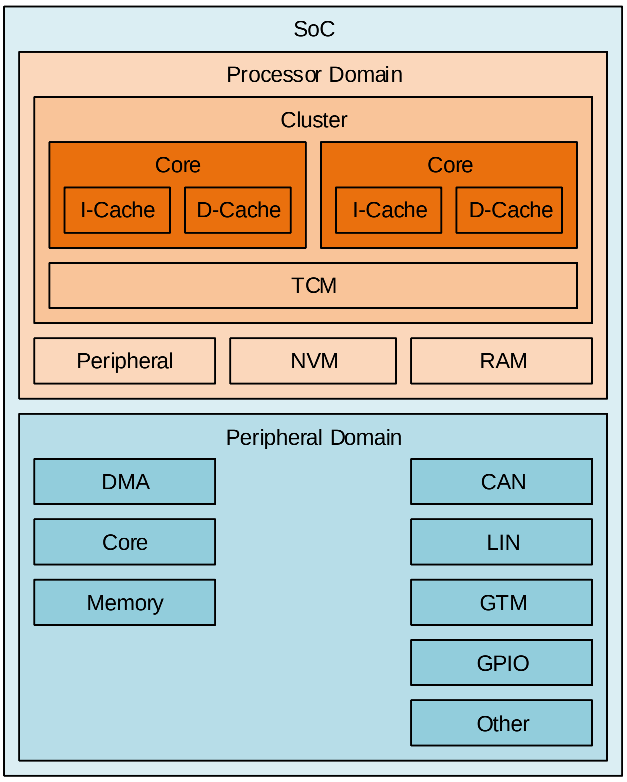

Figure 1: Domains of a SoC

The heat-map (red = fast, blue = slow) in Figure 1 illustrates the speed of data transfers initiated by CPU Core in the Processor Domain. Thus, the connection of the CPU cores to the dedicated caches is the fastest, followed by the TCM within the cluster. When creating the SW, it must be ensured that these memories can be used efficiently.
In extreme contrast to this are the accesses to components in the peripheral domain. For single access, performance can be 30 times slower.

If, for example, the continuous exchange of CAN messages between the CPU core and an M_CAN is considered, the following interactions are typically required:

- Check the status register of M_CAN when asserting an interrupt
- Optionally transfer the CAN messages
- Optionally signal to the M_CAN the completion message transfers

If the CPU cores in the Processor Domain would perform these interactions, they would be significantly slowed down by the NoC. For example, such interactions can also be done via a CPU core within the Peripheral Domain, if available. However, this article focuses on a different approach that does not allocate computational resources of any CPU core.
Bosch offers an add-on for the M_CAN called DMU. With this, the continuous exchange of CAN messages can be completely outsourced to a DMA controller. The add-on is based on the concept of virtualizing the FIFO head elements, s. Figure 2.

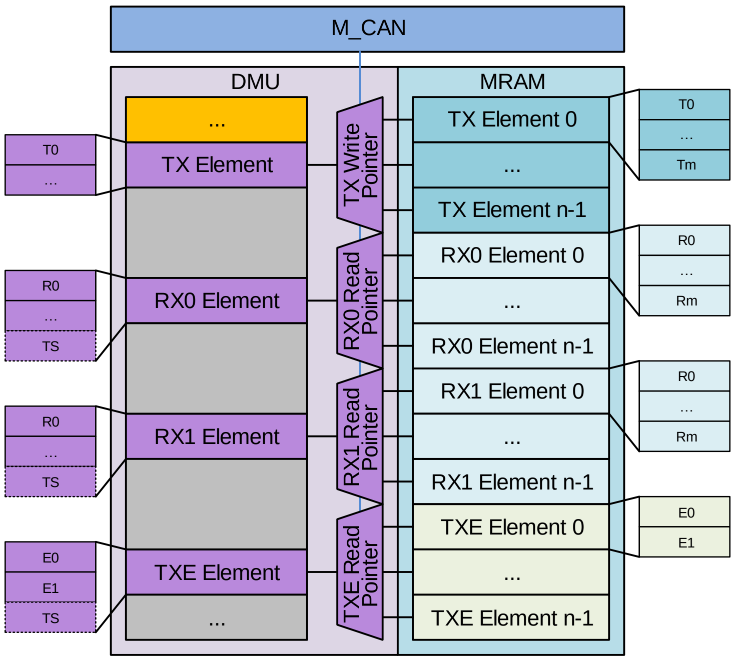

Figure 2: Functional block diagram of DMU

The M_CAN has an associated message RAM (MRAM), which i.a. contains the elements (CAN messages) organized in FIFOs, shown on the right. To access the memory segment of the current message (head element) within these FIFOs by the CPU, the respective pointers (Read / Write Pointer) from the M_CAN must previously be queried. To avoid this, accesses to fixed address areas are virtualized. The DMU dynamically redirects these accesses to the head elements in the MRAM. The redirection is controlled invisibly within the DMU by the FIFO pointers in the M_CAN. The size of the reserved areas correspond to the largest possible message elements, which are 18 words for the TX, RX0 and RX1 elements and 2 words for the TX Event element (3 words if the TSU Timestamp is also transferred). The transfer of a last element word activates a process in the DMU, in which for TX elements the transmit request is set in M_CAN, or for the other elements (RX0, RX1, TXE) the dedicated

FIFO-Acknowledge is set in M_CAN. Thus, writing / reading the CAN messages via DMU elements completes the whole enqueueing / dequeueing process in the M_CAN. The DMU supports messages transfers from the CRAM to the TX FIFO / Queue and vice versa, from the RX FIFOs / TX Event FIFO to the CRAM.
The following block diagram (Figure 3) shows the M_CAN with the add-ons DMU and TSU. The CPU accesses to the M_CAN are routed through both add-on modules.

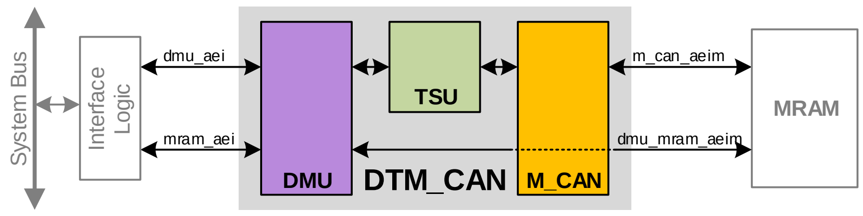

Figure 3: M_CAN subsystem with DMU and TSU

Figure 4 shows the memory map of the DMU. In the yellow marked memory area starting at address 0, the registers of the M_CAN and the TSU are memory mapped. Afterwards, the purple coloured Virtual Buffers are shown, which are accessed by the DMA in order to transport the head elements of the M_CAN message FIFOs.

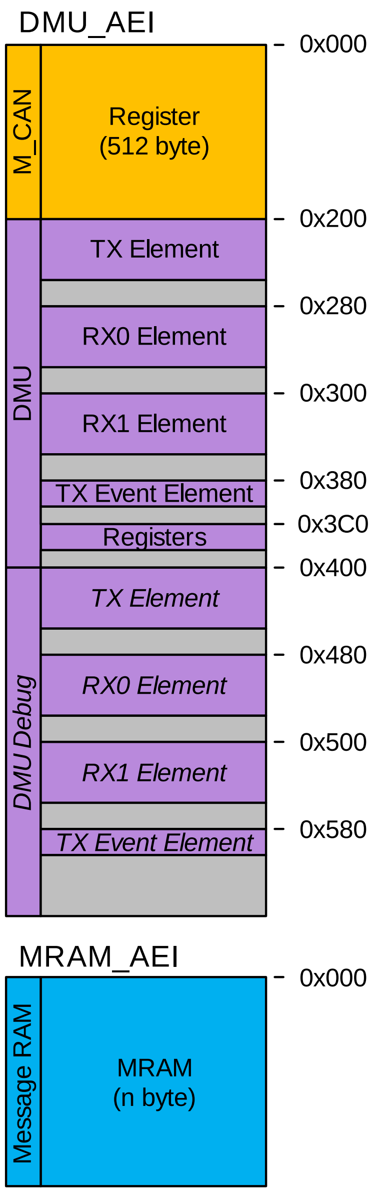

Figure 4: DMU address map

1.1.2 TX element

Here the CAN message elements are written by the DMA controller to be added in the TX FIFO / Queue. When writing the last element word, the TX request is automatically set for this element, so that the M_CAN sends this. The DMU requests further CAN message elements from the DMA controller as long as the TX FIFO / Queue is not full.

1.1.3 RX0 / RX1 elements

Here the CAN message elements are read by the DMA controller which are located in the receive FIFOs 1 or 2 of the M_CAN. When the last element word is read, the dequeuing is communicated to the M_CAN by setting the dedicated acknowledge index by the DMU. Optionally, the time stamp of the TSU can also be transmitted. The DMU triggers the DMA to dequeue further CAN message elements as long as the RX FIFO is not empty.

1.1.4 TX event element

Like the RX0 / RX1 elements, but here the TX events are read, optionally with the time stamp of the TSU.

1.1.5 DMU register

The DMU gets most of the configuration parameters from the M_CAN, only the transport of the HW timestamp of the TSU can be switched on / off here. Various status information provides feedback on whether the access to the virtual elements were correct or, if not, what problem occurred. This is particularly helpful when debugging the DMA routines, but should also be monitored during normal operation for reasons of Functional Safety.

1.1.6 DMU Debug section

When debugging the software, the DMU elements can be accessed by reading, without affecting the enqueuing of dequeuing of the DMA. For the TX element, the last element written is read, for the RX0, RX1, TXE elements the current element is read. These accesses do not trigger automatisms of the DMU, like the acknowledgment in M_CAN.

1.1.7 Data flow within the SoC

The following approach is recommended for the data flow within the SoC: A RAM has to be selected to which the desired CPU core can access with the highest possible read / write performance and to which the DMA controller also has direct access. This CRAM is then used to exchange the CAN messages, with the DMA controller taking over the slow transfers of the CAN messages across large distances of the NoC and storing the messages in the CRAM close to the CPU core, which then access without performance loss.

1.2 TSU add-on

For the AUTOSAR (Automotive Open System Architecture) compatible synchronization of time bases between CAN nodes, only software implementations had been used due to a lack of special hardware. To further increase the time accuracy, special hardware is required. Document CiA 603 specifies a hardware-based concept, which is now available as an M_CAN add-on called TSU. This approach is independent of interrupt response times and thus achieves the best possible accuracy. The TSU may operate on its own internal timebase, or uses an external time base, e.g. a reference time base within the SoC.

1.2.1 Receipt of messages with timestamp

To configure the M_CAN to receive a time-stamped message, a base / extended message ID filter element must be configured accordingly, i.e. S0.SSYNC = 1 or F1.ESYNC = 1. Upon receipt of a valid message matching the filter, the time stamp is stored in the TSU. Since the TSU stores several time stamps, a pointer is written into the CAN message (R1.RXTSP), which points to the corresponding entry in the TSU. When reading out such an RX messages with the DMU, the TSU time stamp can be automatically attached.

1.2.2 Sending messages with timestamp

If a time stamp has to be generated when a CAN message is sent, the following bits must be set in the message: T1.TSCE = 1 and T1.EFC = 1. Upon successful transmission, a time stamp is stored in the TSU and a TX Event message is generated, which refers to the corresponding entry in the TSU, i.e. field TXTSP in event message word E1 (E1.TXTSP). When using DMU, the time stamp of the TSU can also be automatically attached.

1.2.3 Synchronization process after AUTOSAR

After successful configuration (see above) of all participating CAN nodes, the timers of the time slaves can be corrected with a two-step synchronization process. In the first step, the current time T0 is latched in the time master, and the part of T0, which represents the seconds, is sent to the time slaves with the SYNC message. If the EoF is reached when sending this SYNC message, then timestamps are stored in the TSUs of all nodes, i.e. in the Time Master and all Time Slaves. This is the common point of time reference. In the second step, the time master adds its TSU time stamp to the nanosecond part of T0 and sends it to the time slaves via FollowUp message. From this, the time slaves can derive their own time base. A detailed description of the time synchronization is described here [1].

1.2.4 Saving time stamps at SoF

For non-AUTOSAR applications, the TSU can also be configured to store timestamps at the start of a CAN message (SoF bit).