M_CAN
Modular CAN IP-module

Functionalities and Features

White Paper M_CAN_WP001

Document Revision 2.0
01.07.2015

Robert Bosch GmbH
Automotive Electronics

LEGAL NOTICE

Revision History

Version	Date	Remark
1.0	26.11.2010	First version for M_CAN 2.0
2.0	01.07.2015	New revision for M_CAN Revision 3.0.0 - 3.2.1

Conventions

The following conventions are used within this document:

Register Names	**RXBC**

References

This document refers to the following documents:

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

Terms and Abbreviations

This document uses the following terms and abbreviations:

Term	Meaning
CAN	Controller Area Network
M_CAN	Modular Controller Area Network
Rx	Receiver
Tx	Transmitter
IF	Interface
RAM	Random Access Memory
FIFO	First In First Out
CPU	Central Processing Unit
ECC	Error Correction Code
VHDL	VHSIC Hardware Description Language

Table of Contents

1 Introduction

The M_CAN white paper was created to help the first time users to understand the M_CAN functionalities and features. This document is applicable to M_CAN versions 3.0.0, 3.0.1, 3.2.0 and 3.2.1.

2 M_CAN Integration

Figure 1 shows how the M_CAN IP module can be integrated into a system. The M_CAN consists of three main parts.

- The **CAN Core** implements the functionality required to perform communication in a CAN bus system. Therefore it generates the M_CAN_Tx signal and processes the M_CAN_Rx signal.
- The **Tx Handler** in the M_CAN performs all the tasks related to message transmission.
- The **Rx Handler** in the M_CAN performs all the tasks related to message reception.

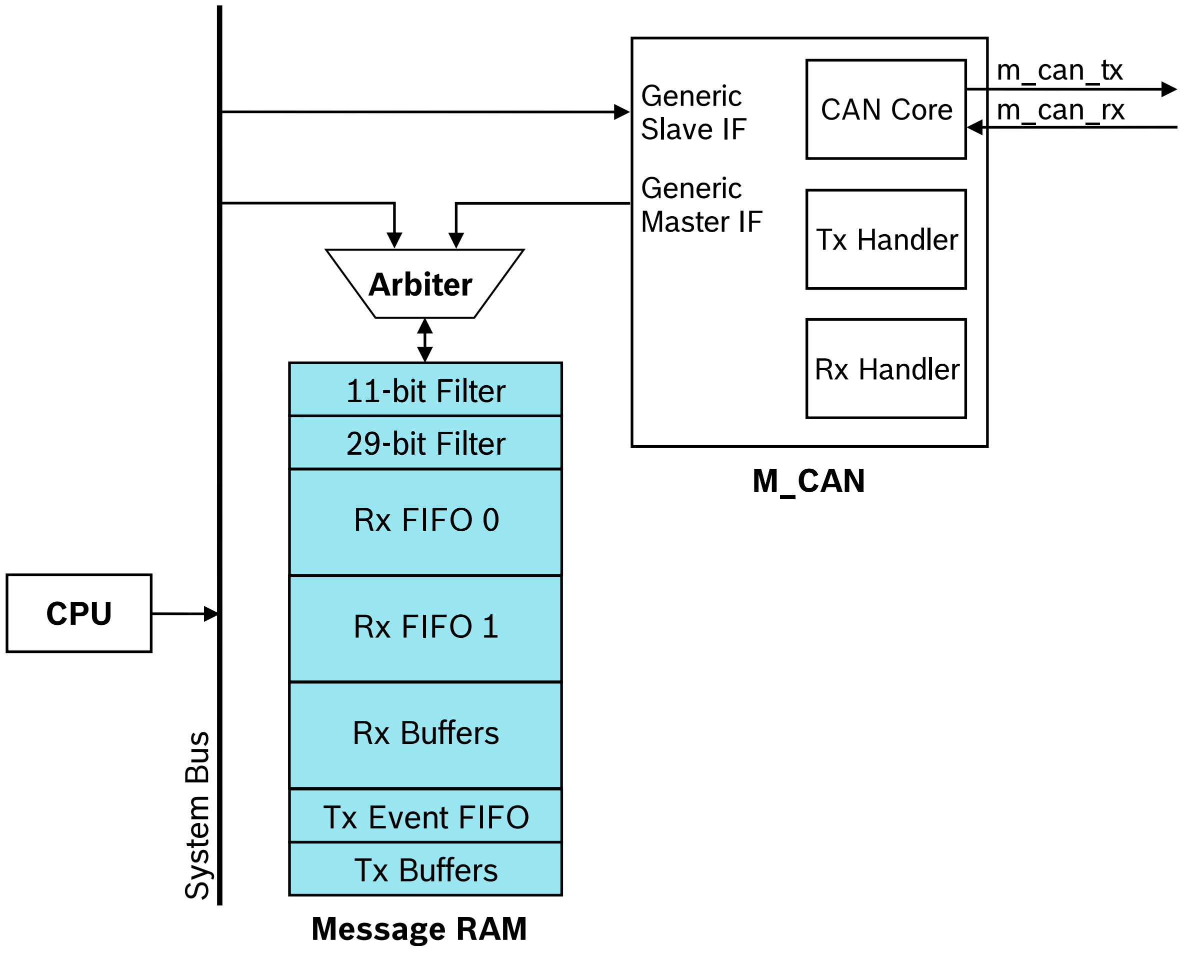

Figure 1: Integration of M_CAN IP module into a system

3 Message RAM

3.1 Introduction

The Message RAM serves as memory for the M_CAN. The following types of data can be stored in the Message RAM:

- Message ID Filter Elements
- Receive Messages (in RX FIFO0, in RX FIFO1, in RX Buffers)
- Transmit Messages (in TX Buffers)
- TX Event Element

The Message RAM needs to be accessible for the CPU and the M_CAN. If the Message RAM implementation has just a single interface, an Arbiter is required to grant access to the Message RAM. Figure 1 shows how the Message RAM can be connected to the M_CAN.

3.2 Partitioning

To operate the M_CAN the Message RAM has to be partitioned into memory sections. Partitioning is an agreement between CPU and M_CAN regarding, where in the Message RAM to store which type of elements, e.g. the M_CAN knows from where to fetch the messages to be transmitted. The Message RAM in Figure 1 contains the recommended partitioning: e.g. the first memory section should hold the 11-bit Filter elements, the second section the 29-bit Filter elements, etc.

In general the following rules should be met.

- The individual sections should be back to back. Sections are not allowed to overlap.
- The order of the sections can be arbitrary.
- Each section should be large enough to fit all according elements: e.g. if an RX FIFO1 should be able to hold 20 maximum size CAN FD frames, then dimension the section accordingly.

Bosch delivers the M_CAN with an excel sheet that allows to calculate the required Message RAM size.

- The Message RAM size required for an M_CAN that uses all features is bounded to a maximum value.
- A section in the Message RAM may also be 0 in size if the feature is not used: e.g. if RX FIFO1 is not used, then the corresponding memory section can be 0

The CPU configures the following information into the M_CAN

- Start addresses of the memory sections in the Message RAM
  The Start address is given relative to the Message RAM.
- Number of elements in each section
  E.g. the number of elements in the RX FIFO1.

- The size of the elements in some sections
  All sections that hold CAN messages can be adjusted in size, e.g. if the largest transmitted CAN FD frame will have just 32 bytes payload, this is configured. This allows saving Message RAM memory.

The default configuration of the M_CAN is that all sections contain 0 elements in the Message RAM.

3.3 Facts

a) Multiple M_CAN controllers may share the same Message RAM. Each M_CAN controller has its own Message RAM section assigned. Sharing Tx and Rx message buffers of different M_CAN controllers is not recommended. Automated gateway functions are not supported.

b) The Message RAM content is not affected by a reset of the M_CAN. The behavior of the Message RAM depends on the system implementation.

c) The Message RAM, including the parity and ECC logic, is not part of the M_CAN deliverables.

d) Arbitration logic, necessary for connecting several M_CANs to one Message RAM instance, is not part of the M_CAN deliverables.

4 Operation of the M_CAN

4.1 Basic Configuration

The CPU configures the M_CAN by reading/writing the registers in the M_CAN. The basic things to be configured are:

- Bit timings
- Enabling interrupts and selecting the interrupt lines

4.2 Message Transmission

The general steps for transmission of a message are described in this section.

Configuration

For the M_CAN to behave as a transmitting node, the CPU configures the fundamental Tx registers like **TXBC**, **TXESC** etc.

Message Transmission

1) The messages that should be transmitted are first written into the Message RAM in the section allocated for the Tx Buffers.

2) In order to request the transmission of a message, the CPU has to write to the appropriate bits in the **TXBAR** register of the M_CAN.

3) When a request to transmit a message is received by the M_CAN, the Tx handler scans the Message RAM (Tx Buffers) for the highest priority message and transmits this message.

4.3 Message Reception

The general steps for reception of a message are described in this section.

Configuration

1) For the M_CAN to behave as a receiving node, the fundamental Rx registers like **RXBC**, **RXFnC**, **and RXESC** etc. have to be configured by the CPU. These registers define among others where the messages are stored.

2) Acceptance filtering of the received messages can be done by configuring registers **GFC**, **SIDFC** and **XIDFC**.

Message Reception

3) If acceptance filtering is configured, the CPU has to write the standard/extended message ID filters into the Message RAM.

4) When a message is received by the M_CAN, the Rx handler does the acceptance filtering according to the **GFC**, **SIDFC** and **XIDFC** configurations. The Rx handler compares the ID of the received message with the ID's in the filter elements. If it finds a match, it stores/discards the received message according to the filter element configuration.

5) After filtering, if the message is to be stored, it will be stored either in an Rx FIFO or a Dedicated Rx buffer in the Message RAM, according to the filter element configuration.
6) The M_CAN can inform the CPU via interrupts (if configured) that a new message has arrived.
7) The CPU reads the Rx Buffer/FIFO status registers of the M_CAN to find out where the M_CAN had stored the message in the Message RAM.
8) The CPU reads the message from the Message RAM.
9) The CPU acknowledges the message by writing to an M_CAN register.

5 Tx Handler

a) The M_CAN supports 32 Tx buffers. The Tx buffers can be configured as dedicated Tx buffers, Tx FIFO, or Tx queue and as combination of dedicated/FIFO or dedicated/queue Tx buffers.

b) There exists no separate “message valid” flag. A message which is requested for transmission is send out as soon as the CAN bus is available.

c) Pending transmissions are signalled in an own register (“Tx Buffer Request Pending”).

d) The Tx handler scans all Tx buffers with activated transmission request, in case of the Tx FIFO the oldest pending Tx FIFO buffer. The scan result is assorted according to the priority of the message ID. The Tx message with the highest priority message ID is send out first.

e) A requested message can be cancelled (transmit cancellation flag).

f) The information about each occurred transmission, including a Tx time stamp, can be found in the Tx event FIFO. There exist two transmit events: “Tx event” (successful transmission) and “Transmission in spite of cancellation” (transmission after late Host cancellation)

g) A successfully cancelled transmission is signalized in an own register (“Tx Buffer Cancellation Finished”).

h) Remote frame handling is not implemented. The reception of a remote frame will not trigger the transmission of a message.

6 Rx Handler

a) Successfully received and filtered messages are stored in one of the two receive FIFOs or in a dedicated Rx buffer. Each receive FIFO can hold up to 64 messages. Up to 64 dedicated Rx buffers can be configured.

b) If the receive FIFO is full, the newly arriving messages are treated according to the configured mode. Two modes are available.

- Blocking Mode: No further messages are written to the Rx FIFO until at least one message has been read out from the Rx FIFO.
- Overwrite Mode: The new message accepted in the Rx FIFO will overwrite the oldest message in the Rx FIFO.

Note: When an Rx FIFO is operated in overwrite mode and an Rx FIFO full condition is signalled, reading of the Rx FIFO elements should start at least at get index + 1. The reason for that is, that it might happen, that a received message is written to the Message RAM (put index) while the CPU is reading from the Message RAM (get index).

c) The FIFO fill level is signalized. A FIFO watermark can be configured. Dedicated interrupts can be configured. The FIFO can be cleared with one write access by resetting PUT and GET index to zero (“FIFO flush”).

7 Message Filtering (Range, Dual, Classic)

a) Up to 128 filter elements for 11-bit identifiers may be configured.

b) Up to 64 filter elements for 29-bit identifiers may be configured.

c) With the high priority message filter option, received messages can trigger a high priority message interrupt. For direct access to these messages, the respective FIFO buffer index can be read from the High Priority Message Status register.

d) Each filter can be configured as acceptance or rejection filter.

8 Safety

a) The Message RAM may be secured against bit errors. If an external circuit detects and signals a Message RAM error via (**`m_can_aeim_beu`** = ‘1’) to the M_CAN, then the M_CAN sets **CCCR.INIT** to ‘1’ and enters INIT state.

b) The READY input of the generic master interface (**`m_can_aeim_ready`**) is monitored by the RAM watchdog during an access via the generic master interface (**`m_can_aeim_select`** active). In case there is no response at the READY input until the watchdog counter has counted down to zero, the counter stops and interrupt flag **IR.WDI** is set. The counter is configurable and can be disabled. The counter is clocked by the Host clock (**`m_can_hclk`**).

c) Suspend CAN operation: The Host can set the **CCCR.INIT** bit. While **CCCR.INIT** is set, message transfer from and to the CAN bus is stopped, the status of the CAN bus output can_tx is recessive (HIGH). The counters of the Error Management Logic EML are unchanged. Setting **CCCR.INIT** does not change any configuration register.

9 Miscellaneous

a) The M_CAN can be set into power down mode controlled by input signal **`m_can_clkstop_req`** or via CC Control Register **CCCR.CSR.** As long as the clock stop request signal **`m_can_clkstop_req`** is active, bit **CCCR.CSR** is read as one. When all pending transmission requests have completed, the M_CAN waits until bus idle state is detected. Then the M_CAN sets the **CCCR.INIT** to one to prevent any further CAN transfers. Now the M_CAN acknowledges that it is ready for power down by setting output signal **`m_can_clkstop_ack`** to one and **CCCR.CSA** to one. In this state, before the clocks are switched off, further register accesses can be made. A write access to **CCCR.INIT** will have no effect.

b) The M_CAN is not able to detect the baud rate autonomously by monitoring the CAN bus.

c) By hard reset, all M_CAN registers are reset to their default values. A soft reset achieving the same result is not supported.

d) The following bits/bit fields are set/reset with every read access to the register.

   - **ECR.CEL**
   - **PSR.PXE**
   - **PSR.RFDF**
   - **PSR.RBRS**
   - **PSR.RESI**
   - **PSR.DLEC**
   - **PSR.LEC**

   The destructive read accesses can be prevented with the signal **`m_can_dis_mord`** (disable modification on read).