



**BOSCH**

Invented for life

---

# **M\_CAN**

**Modular CAN IP-module**

## **Reception Handling**

**Application Note M\_CAN\_AN001**

**Document Revision 2.3**  
**28.06.2021**



Robert Bosch GmbH  
Automotive Electronics

---

## LEGAL NOTICE

© Copyright 2021 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	07.06.2011	First version for M_CAN 2.0
2.0	15.07.2015	New revision for M_CAN Revision 3.0.0 - 3.2.1
2.1	28.09.2015	<ul style="list-style-type: none"> <li>- Titel changed to "Reception Handling"</li> <li>- Updated software examples (c code)</li> <li>- Rephrased some descriptions</li> <li>- Added a new section: Timeout Counter</li> </ul>
2.2	29.06.2018	Source Code updated
2.3	28.06.2021	Source Code updated

## Conventions

The following conventions are used within this document:

Register Names	<b>RXBC, SIDFC</b>
Names of files and directories	directoryname/filename
Source code	m_can_rx_fifo_init(..)

## 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
BRP	Baud Rate Prescaler
CAN	Controller Area Network
CRC	Cyclic Redundancy Check
DLC	Data Length Code

## Table of Contents

<b>1</b>	<b>Target .....</b>	<b>1</b>
<b>2</b>	<b>Receive Buffers .....</b>	<b>2</b>
2.1	Rx FIFOs.....	2
2.2	Dedicated Rx Buffers.....	5
<b>3</b>	<b>Acceptance Filtering .....</b>	<b>8</b>
3.1	Introduction.....	8
3.2	Functionality .....	8
3.2.1	Overview.....	8
3.2.2	Range Filter .....	9
3.2.3	Filter for dedicated IDs .....	9
3.2.4	Classic Bit Mask Filter.....	9
3.3	Standard Message ID Filtering .....	10
3.4	Extended Message ID Filtering .....	11
3.5	Example .....	12
<b>4</b>	<b>High Priority Messages.....</b>	<b>14</b>
<b>5</b>	<b>Timeout Counter .....</b>	<b>16</b>
<b>6</b>	<b>Software Example .....</b>	<b>17</b>
<b>7</b>	<b>List of Tables .....</b>	<b>18</b>
<b>8</b>	<b>List of Figures.....</b>	<b>19</b>

# 1 Target

This application note describes the handling of received messages in the **M\_CAN versions 3.0.0 up to 3.3.0**

The topics included are:

- Rx Buffer configuration – Rx FIFO and dedicated Rx buffers
- Acceptance filtering – for Standard and Extended message IDs
- High priority message signalling

**Note: Software examples in this application note are only for illustration purposes. Use the examples on own risk.**

## 2 Receive Buffers

### 2.1 Rx FIFOs

The M\_CAN provides two Rx FIFOs called Rx FIFO0 and Rx FIFO1. Each FIFO is capable to store up to 64 messages. Each message is stored in one Rx FIFO element. The M\_CAN User's manual describes the structure of an Rx FIFO element. The size of an Rx FIFO element can be configured via register **RXESC** for each FIFO individually. The Rx FIFO element size defines how many data field bytes of a received message can be stored.

The size of an Rx FIFO element is:

$2 * 4$  byte header information + data field size configured in **RXESC**

Example for a 64 byte data field size:  $(2 * 4 + 64) = 72$  bytes or 18 words. The memory requirement for an Rx FIFO in the Message RAM depends on the configured Rx FIFO element size **RXESC.FnDS**. Configuration of the two Rx FIFOs is done via registers **RXF0C** and **RXF1C**.

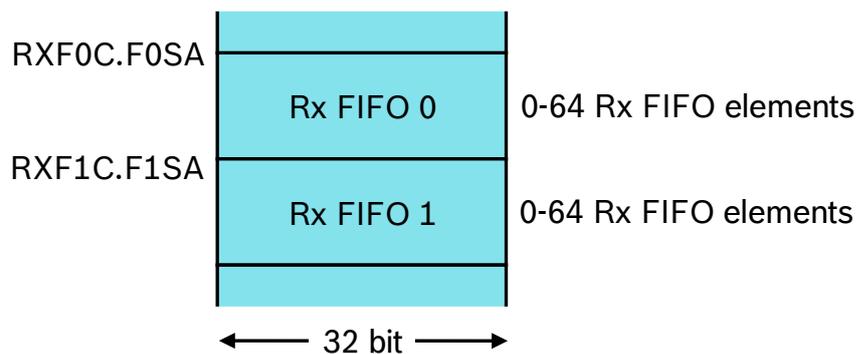


Figure 1: Message RAM Rx FIFO section

Figure 1 shows the section of the Message RAM with the two Rx FIFOs and their respective start addresses. The start address of a FIFO is the address of the first word of the first Rx FIFO element. Received messages that passed acceptance filtering are stored in the appropriate Rx FIFO based on the matching filter element. Acceptance filtering configurations are explained in chapter 3.

If the receive FIFO is full, the newly arriving messages can be handled according to two different modes. The used mode is configured in **RXFnC.FnOM**.

- **Blocking Mode:** No further messages are written to the Rx FIFO until at least one message has been read out from the Rx FIFO. This is the default operation mode of the Rx FIFOs.
- **Overwrite Mode:** The new message accepted in the Rx FIFO will overwrite the oldest message in the Rx FIFO and the put and get indices of the FIFO are incremented by one.

The following interrupt flags exist, to inform the CPU that the status of an Rx FIFO has changed:

- **IR.RFnN:** Rx FIFO *n* New Message  
When a new message has arrived in the Rx FIFO, this interrupt flag is set.
- **IR.RFnW:** Rx FIFO *n* Watermark reached  
To avoid an Rx FIFO overflow, the Rx FIFO watermark can be used. When the Rx FIFO fill level reaches the Rx FIFO watermark configured by **RXFnC.FnWM**, this interrupt flag is set.
- **IR.RFnF:** Rx FIFO *n* Full  
When an Rx FIFO full condition is reached i.e. the get and put indices of the FIFO becomes equal (**RXFnS.FnPI = RXFnS.FnGI**), this interrupt flag is set.
- **IR.RFnL:** Rx FIFO *n* Message Lost  
If a message is received while the corresponding Rx FIFO is full, this interrupt flag is set.

To read a message from an Rx FIFO, the CPU has to perform the following steps:

1. Read the register **RXFnS** to know the status of the FIFO.
2. Calculate the address of the oldest message in the Message RAM.  
**RXFnS.FnGI** addresses the oldest message. The Address is calculated according to the Formula:

$$\text{Message\_RAM\_base\_address} + \mathbf{RXFnC.FnSA} \\ + \mathbf{RXFnS.FnGI} * \text{Rx\_FIFO\_element\_size}$$

Example (byte address):  $0x100000 + 0x200 + 2 * (18 * 4)$

3. Read the message from the calculated address

After the host has read a message or a sequence of messages from the Rx FIFO, it has to acknowledge the read. After acknowledgement, the M\_CAN can reuse the corresponding Rx FIFO buffer for a new message. To acknowledge one or more messages, write the buffer index of the last element read from Rx FIFO to register **RXFnA**. As a consequence, the M\_CAN updates the FIFO fill level and the Get index.

Summary of registers used for Rx FIFO operation:

- **RXFnC:** Rx FIFO 0/1 Configuration
- **RXFnS:** Rx FIFO 0/1 Status
- **RXESC:** Rx Buffer/FIFO Element Size Configuration
- **RxFnA:** Rx FIFO 0/1 Acknowledge

## Software Example

Table 1 lists C functions that demonstrate Rx FIFO operations. The functions are provided with this application note.

Table 1: C functions that demonstrate Rx FIFO operations

<b>Name:</b>	<code>m_can_rx_fifo_init(..)</code>
<b>File:</b>	<code>../m_can/m_can.c</code>
<b>Description:</b>	This function configures an RX FIFO of the M_CAN. M_CAN has to be in configuration change enable mode ( <b>CCCR.CCE='1'</b> ) when this function is called.
<b>Name:</b>	<code>m_can_rx_fifo_copy_msg_to_msg_list(..)</code>
<b>File:</b>	<code>../m_can/m_can.c</code>
<b>Description:</b>	Function copies and acknowledges all the received messages from the Rx FIFO section of the Message RAM. This function serves as an interrupt service routine for the interrupts <b>IR.RFnN</b> .
<b>Name:</b>	<code>m_can_read_msg_from_msg_ram(..)</code>
<b>File:</b>	<code>../m_can/m_can.c</code>
<b>Description:</b>	Function that actually copies an Rx FIFO element/RX buffer from a calculated address in the Message RAM to an own data structure in the software.

## 2.2 Dedicated Rx Buffers

### Overview

The M\_CAN supports up to 64 dedicated Rx buffers. Each dedicated Rx buffer can store one CAN message. The M\_CAN User's manual describes the structure of an Rx buffer element. The size of a dedicated Rx buffer can be configured via the register **RXESC**. The Rx buffer size defines how many data field bytes of a received message can be stored.

The size of a dedicated Rx buffer is:

$2 * 4$  byte header information + data field size configured in **RXESC**.

Example for a 64 byte data field:  $(2 * 4 + 64) = 72$  bytes or 18 words. The memory requirement for a dedicated Rx buffer in the Message RAM depends on the configured Rx buffer data field size **RXESC.RBDS**.

Figure 2 shows the Rx buffer section in the Message RAM. The user has to configure the start address of the section in the M\_CAN register **RXBC**.

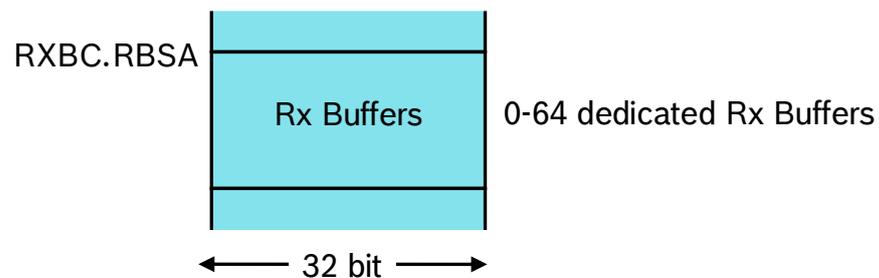


Figure 2: Message RAM dedicated Rx Buffer section

To let the M\_CAN store a message in an Rx buffer, the user has to configure a Standard or Extended ID Filter. A filter contains the information where to store a message in case when the filter matches. This means a filter can reference an Rx buffer. Chapter 3 describes how to use acceptance filtering in the M\_CAN.

## Arrival of New Messages

When a message is stored in a dedicated Rx buffer, the M\_CAN sets the interrupt flag **IR.DRX** and the corresponding bit in **NDAT1/2**. When bits in **NDAT1/2** are set the respective Rx buffer is locked (will not be overwritten by a new message) and the corresponding filter will not match. After reading a message, the host has to reset the bit in **NDAT1/2**, in order to unlock the respective Rx buffer.

To read a message from a dedicated Rx buffer, the CPU has to perform the following steps:

1. Check the bits in NDAT1/2 to know if a new message has arrived in a dedicated Rx buffer.
2. Calculate the address of the message in the Message RAM according to formula:

$$\text{Message\_RAM\_base\_address} + \mathbf{RXBC.RBSA} \\ + \text{dedicated Rx buffer index} * \text{Rx\_Buffer\_element\_size}$$

Example (byte address):  $0x100000 + 0x2800 + * (18 * 4)$

3. Read the message from the calculated address

## Configure the number of used Rx Buffers

The number of Rx buffers to be used by the M\_CAN is configured indirectly by the standard and extended filters elements (see Chapter 3). This means, the M\_CAN contains no register to configure the number of Rx buffers.

A standard or extended filter element can reference Rx buffers 0 to 63 as destination for a received message. The M\_CAN will only perform a write to a referenced Rx buffer location, if the corresponding filter matches. In other words, the M\_CAN will not write to unreferenced Rx buffer locations.

**Good example:** If the used standard and extended filter elements only reference to Rx buffer 0 to 15, then the Rx buffer section in the Message RAM can be dimensioned to the following size:  $16 * \text{Rx\_Buffer\_element\_size}$ .

The size of the Rx buffer section in the Message RAM depends on the highest referenced Rx buffer. To keep the Rx buffer section size small start using Rx buffer 0 up to the required number of Rx buffers, e.g. Rx buffers 0 to 7.

**Bad example:** If two filter elements are used, where the first references Rx buffer 0 and the second Rx buffer 63, then the Rx buffer section size is  $64 * \text{Rx\_Buffer\_element\_size}$ . If the second filter references Rx buffer 1, then the Rx buffer section size is just  $2 * \text{Rx\_Buffer\_element\_size}$ .

## Register Summary

Summary of registers used for a dedicated Rx Buffer operation:

- **RXBC:** Rx Buffer Configuration
- **RXESC:** Rx Buffer/FIFO Element Size Configuration
- **NDAT1:** New Data 1
- **NDAT2:** New Data 2

## Software Example

Table 2 lists C functions that demonstrate operation of dedicated Rx Buffers. The functions are provided with this application note.

Table 2: C functions that demonstrate dedicated Rx buffer operations

<p><b>Name:</b> <code>m_can_rx_dedicated_buffers_init(..)</code></p> <p><b>File:</b> <code>../m_can/m_can.c</code></p> <p><b>Description:</b> This function configures the usage of dedicated Rx buffers in the M_CAN. M_CAN has to be in the configuration change enable mode (<b>CCCR.CCE</b>='1') when this function is called.</p>
<p><b>Name:</b> <code>m_can_rx_dedicated_buffer_process_new_msg(..)</code></p> <p><b>File:</b> <code>../m_can/m_can_irq_handling.c</code></p> <p><b>Description:</b> This function reads received messages from the dedicated Rx buffer section of the Message RAM based on bits set in registers <b>NDAT1/2</b>. After a successful read, the corresponding bits in <b>NDAT1/2</b> are reset. This function serves as an interrupt service routine for the interrupt <b>IR.DRX</b>.</p>
<p><b>Name:</b> <code>m_can_rx_dedicated_buffer_copy_msg_to_msg_list(..)</code></p> <p><b>File:</b> <code>../m_can/m_can.c</code></p> <p><b>Description:</b> Function copies a message from the Rx Buffer section of the Message RAM to a message list in the software. Therefore it calculates the address of the desired RX buffer.</p>
<p><b>Name:</b> <code>m_can_read_msg_from_msg_ram(..)</code></p> <p><b>File:</b> <code>../m_can/m_can.c</code></p> <p><b>Description:</b> Function that actually copies an Rx FIFO element/RX buffer from a calculated address in the Message RAM to an own data structure in the software.</p>

## 3 Acceptance Filtering

### 3.1 Introduction

The M\_CAN supports acceptance filtering in hardware. This means the user can configure the so called filter elements. Based on these, the M\_CAN stores or rejects received messages. Up to 128 filter elements can be configured for 11-bit standard IDs and up to 64 filter elements can be configured for 29-bit extended IDs. The M\_CAN User's manual describes the size and structure of a filter element. Figure 3 shows a section of the Message RAM with the filter elements and their start addresses.

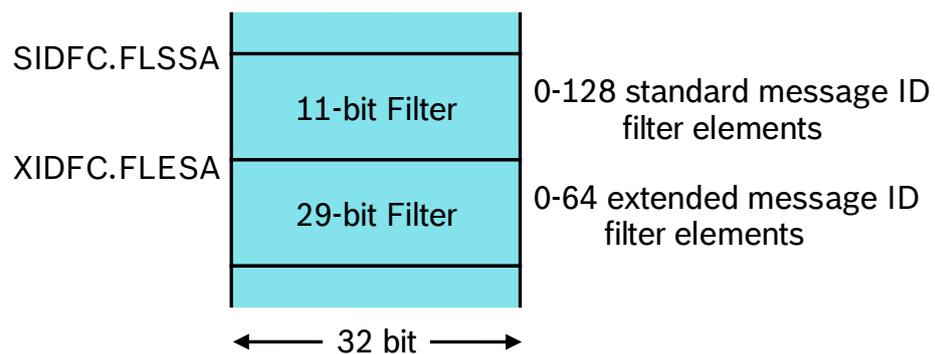


Figure 3: Message RAM Filters section

The registers used to configure the acceptance filters are:

- **GFC:** Global Filter Configuration  
GFC controls the filter path for standard and extended messages as described in Figure 4 and Figure 5.
- **SIDFC:** Standard ID Filter Configuration  
This register configures the setups for 11-bit standard message ID filtering
- **XIDFC:** Extended ID Filter Configuration  
This register configures the setups for 29-bit extended message ID filtering
- **XIDAM:** Extended ID AND Mask  
For acceptance filtering of extended frames the Extended ID AND Mask is ANDed with the Message ID of a received frame. Intended for masking of 29-bit IDs in SAE J1939. With the reset value of all bits set to one, the mask is not active.

### 3.2 Functionality

The Rx Handler controls the acceptance filtering, the transfer of received messages to the Rx FIFOs and the dedicated Rx buffers as well as the Put and Get Indices.

#### 3.2.1 Overview

The M\_CAN offers the possibility to configure two sets of acceptance filters, one for standard identifiers and one for extended identifiers. These filters can be assigned to the Rx FIFO's or to the dedicated Rx buffers. When the M\_CAN performs acceptance filtering, it starts always at filter element #0 and proceeds through the filter list to find a matching element. Acceptance filtering stops at the first matching element and the

following filter elements are not evaluated for this message. Therefore the sequence of configured filter elements has a significant impact on the performance of the filtering process.

The main features are:

- Each filter element can be configured as
  - Range filter (from – to)
  - Filter for one or two dedicated IDs
  - Classic bit mask filter
- Each filter element is configured for acceptance or rejection filtering
- Each filter element can be enabled / disabled individually
- Filters are checked sequentially starting at element #0; execution stops with the first matching element

Depending on the configuration of the filter element (**SFEC/EFEC**) a match triggers one of the following actions:

- Store received frame in FIFO 0 or FIFO 1 or a dedicated Rx Buffer
- Reject received frame
- Set High Priority Message interrupt flag **IR.HPM**
- Set High Priority Message interrupt flag **IR.HPM** and store received frame in FIFO 0 or FIFO 1
- Store received frame in an Rx buffer or as a debug message

### 3.2.2 Range Filter

The filter matches for all frames with Message IDs in the range defined by **SF1ID/SF2ID** respectively **EF1ID/EF2ID**.

### 3.2.3 Filter for dedicated IDs

A filter element can be configured to filter for one or two specific Message IDs. To filter for one specific Message ID, the filter element has to be configured with **SF1ID=Sf2ID** resp. **EF1ID=EF2ID**.

### 3.2.4 Classic Bit Mask Filter

Classic bit mask filtering is intended to filter groups of Message IDs by masking single bits of a received Message ID. With classic bit mask filtering **SF1ID/EF1ID** is used as Message ID filter, while **SF2ID/EF2ID** is used as filter mask.

A zero bit at the filter mask will mask out the corresponding bit position of the configured ID filter, e.g. the value of the received Message ID at that bit position is not relevant for acceptance filtering. Only those bits of the received Message ID where the corresponding mask bits are one are relevant for acceptance filtering. In case all bits are one, a match occurs only when the received Message ID and the Message ID filter are identical. If all mask bits are zero, all Message IDs match.

### 3.3 Standard Message ID Filtering

Figure 4 shows the flow for standard Message ID (11-bit Identifier) filtering. The standard filtering is controlled by the Global Filter Configuration (**GFC**) and the Standard ID Filter Configuration (**SIDFC**). The list of configured filter elements is compared against the following components of the received frames:

- Message ID
- Remote Transmission Request (RTR)
- Identifier Extension Bit (IDE)

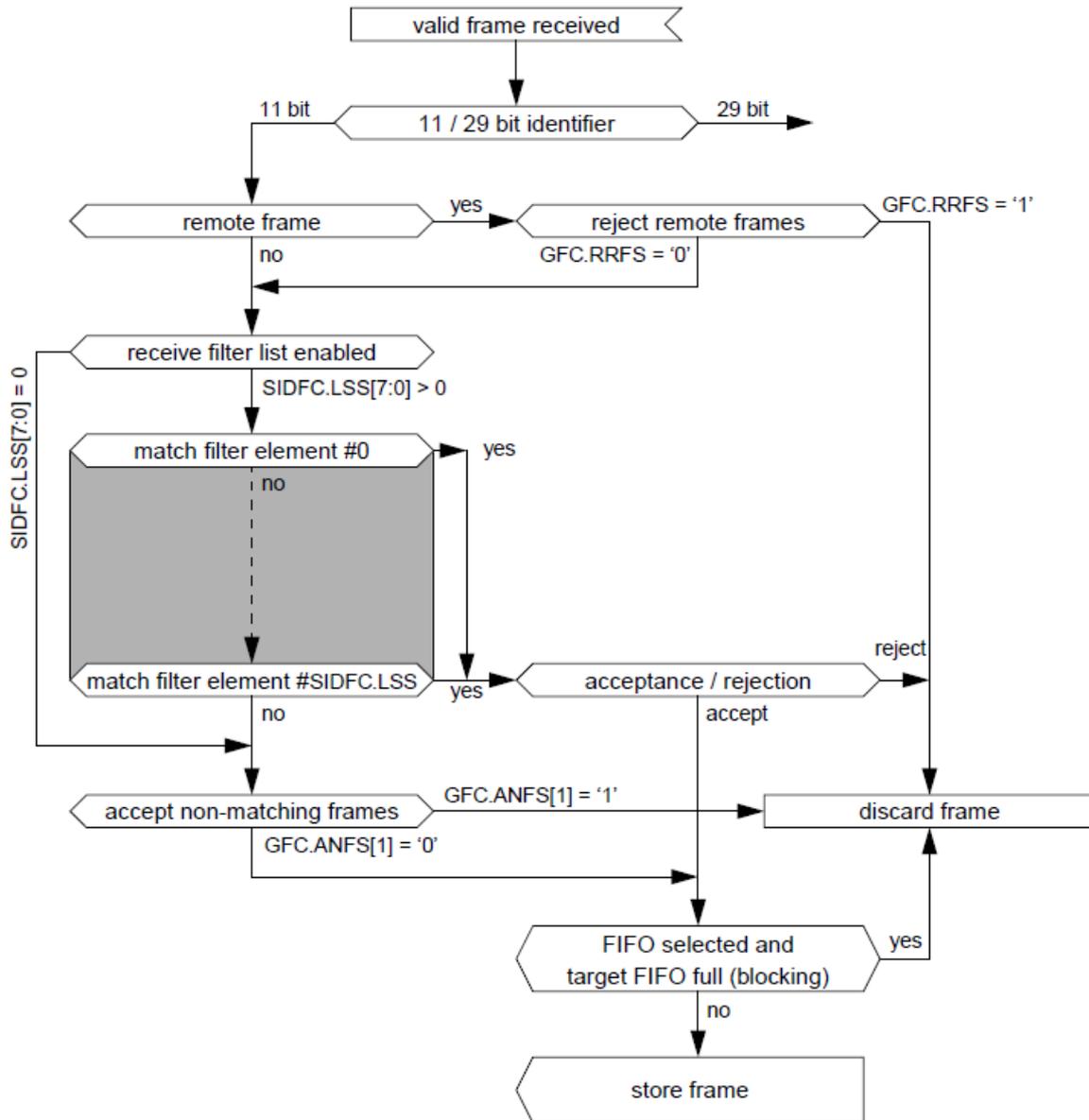


Figure 4: Standard Message ID Filter Path

### 3.4 Extended Message ID Filtering

Figure 5 shows the flow for extended Message ID (29-bit Identifier) filtering. This filtering is controlled by Global Filter Configuration, **GFC** and Extended ID Filter Configuration, **XIDFC**. The Extended ID AND Mask **XIDAM** is ANDed with the received identifier before the filter list is executed. The list of configured filter elements is compared against the following components of the received frames:

- Message ID
- Remote Transmission Request (RTR)
- Identifier Extension Bit (IDE)

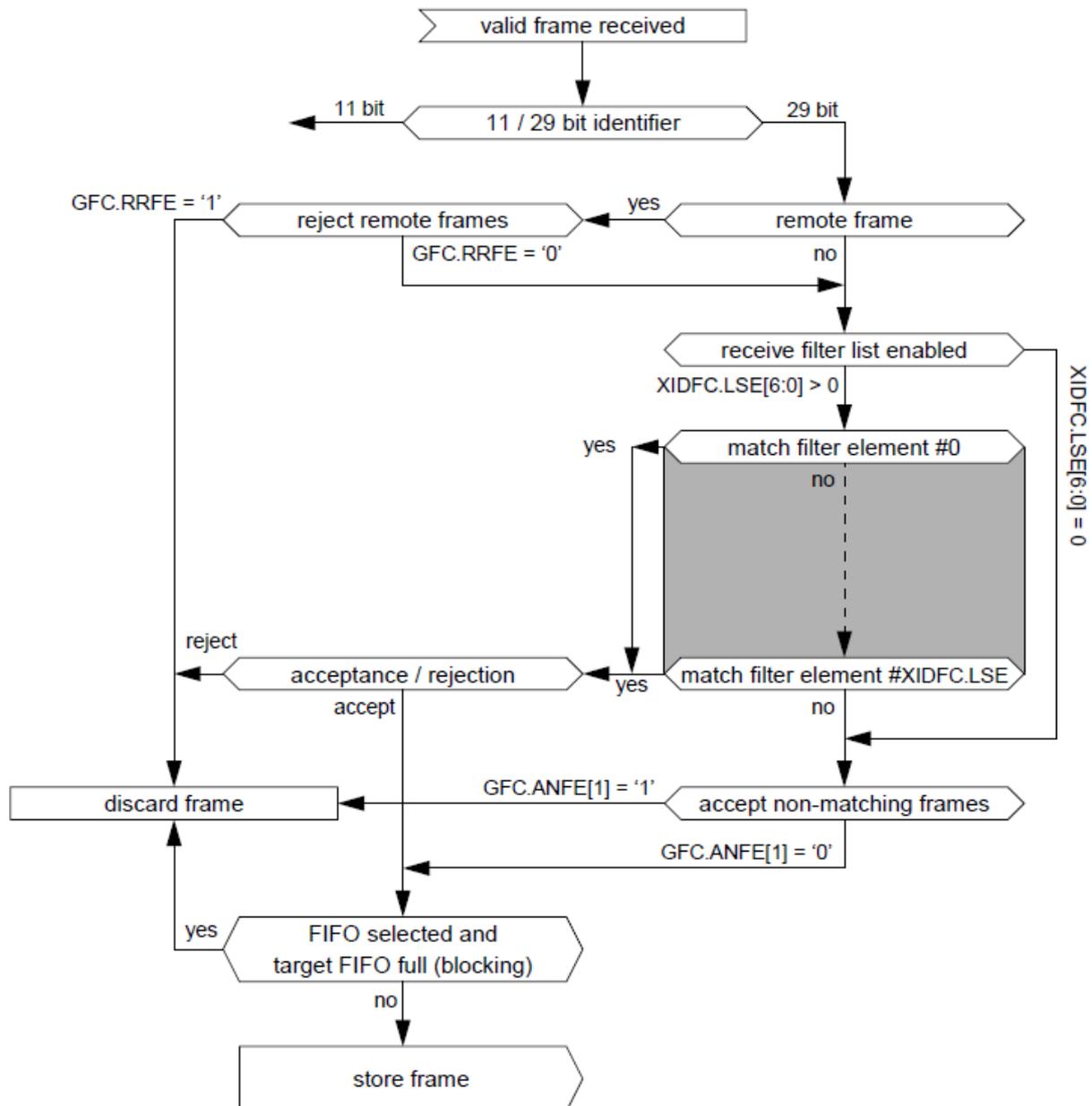


Figure 5: Extended Message ID Filter Path

### 3.5 Example

The numerous filter possibilities of the M\_CAN allow a complex message filtering in hardware. This makes software filtering redundant and saves CPU resources.

Table 3 shows different standard 11-bit message ID filters. These filters are used as an example to illustrate acceptance filtering.

Table 3: Filter configuration

Filter	SFT	SFEC	SFID1	SFID2
0	Range Filter	reject	0x017	0x019
1	Range Filter	Store in FIFO 0	0x014	0x01A
2	Dual ID	Store in FIFO 0	0x0184	0x0187
3	Dual ID	Store in FIFO 0	0x0189	0x0189
4	Classic Filter	Store in FIFO 0	0x200	0x39F
5	Classic Filter	reject	0x201	0x39F
6	Not applicable	Store in Rx buffer	0x325	0x02 (ded. buffer index)
7	Not applicable	Store in Rx buffer	0x326	0x05 (ded. buffer index)

Table 4 shows the result of the acceptance filtering based on the filter elements in Table 3.

Table 4: Result of message ID Filtering

Identifier	accept/reject	Matching Filter	Store in
0x014	accept	Filter 1	FIFO0
0x015	accept	Filter 1	FIFO0
0x016	accept	Filter 1	FIFO0
0x017	reject	Filter 0	
0x018	reject	Filter 0	
0x019	reject	Filter 0	
0x01A	accept	Filter 1	FIFO0
0x184	accept	Filter 2	FIFO1
0x187	accept	Filter 2	FIFO1
0x189	accept	Filter 3	FIFO0
0x200	accept	Filter 4	FIFO0
0x201	reject	Filter 5	
0x220	accept	Filter 4	FIFO0
0x221	reject	Filter 5	
0x240	accept	Filter 4	FIFO0
0x241	reject	Filter 5	
0x260	accept	Filter 4	FIFO0
0x261	reject		
0x325	accept	Filter 6	Rx buffer at index 2
0x326	accept	Filter 7	Rx buffer at index 5
others	accept	accept non matching frames	FIFO 1

## Software Example

Table 5 lists C functions that demonstrate how to configure the various filter elements. The functions are provided with this application note.

Table 5: C functions that demonstrate acceptance filter configurations

Name:	<code>m_can_global_filter_configuration(..)</code>
File:	<code>../m_can/m_can.c</code>
Description:	This function configures the <b>GFC</b> register for global filters. M_CAN has to be in the configuration change enable mode ( <b>CCCR.CCE</b> ='1').
Name:	<code>m_can_filter_init_standard_id(..)</code>
File:	<code>../m_can/m_can.c</code>
Description:	This function configures the <b>SIDFC</b> register for standard 11-bit message ID filter usage. M_CAN has to be in the configuration change enable mode ( <b>CCCR.CCE</b> ='1').
Name:	<code>m_can_filter_init_extended_id(..)</code>
File:	<code>../m_can/m_can.c</code>
Description:	This function configures the <b>XIDFC</b> register for extended 29-bit message ID filter usage. M_CAN has to be in the configuration change enable mode ( <b>CCCR.CCE</b> ='1').
Name:	<code>m_can_filter_write_standard_id(..)</code>
File:	<code>../m_can/m_can.c</code>
Description:	This function writes a standard 11-bit message ID filter element into the Message RAM.
Name:	<code>m_can_filter_write_extended_id(..)</code>
File:	<code>../m_can/m_can.c</code>
Description:	This function writes an extended 29-bit message ID filter element into the Message RAM.

## 4 High Priority Messages

The M\_CAN can notify the user, that a high priority message was received. This can be used to monitor the status of incoming high priority messages and to enable fast access to these messages.

### Overview

Figure 6 shows the sequence of events when a high priority message is received.

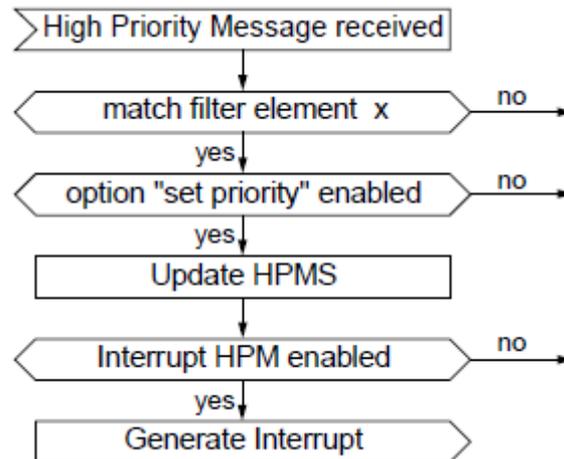


Figure 6: Handling of High Priority Messages

### Configuration

The M\_CAN detects a high priority message with help of a message filter. A filter element (standard or extended ID) provides the following settings related to high priority messages. This settings are made in **SFEC** (standard ID) or **EFEC** (extended ID) field of the filter element configuration.

- **“Set priority if filter matches”**: If this message filter matches, the M\_CAN notifies about high priority message arrival but **does not store the message**.
- **“Set priority and store in FIFO 0 if filter matches”**: If this messages filter matches, the M\_CAN Informs about high priority message arrival and stores message in RX FIFO0.
- **“Set priority and store in FIFO 1 if filter matches”**: Same as the last item, but the M\_CAN stores the message in RX FIFO1.

### Notification

When the M\_CAN finds a matching filter and this is configured to generate a priority event, then the M\_CAN does the following.

- The M\_CAN updates **HPMS** (High Priority Message Status) register. **HPMS** contains information about the filter element that matched and where the M\_CAN stored the message.
- The M\_CAN sets **IR.HPM** (High Priority Message Interrupt) interrupt flag.

### Hint 1: General usage of high priority messages

The M\_CAN overwrites the register **HPMS** and sets the **IR.HPM** flag each time when a message ID filter element matches, that has the according configuration. In high load situation, if two high priority messages arrive back to back, it is possible that the CPU misses to execute the interrupt before the second high priority message arrives. As the register **HPMS** is overwritten, it is not possible to find out that there was an unhandled **IR.HPM** interrupt before.

This means the application should tolerate that it misses some of the M\_CAN notifications about high priority messages. This potential “miss” refers only to the notification of the high priority message. The M\_CAN stores the message (if configured) and the user can access it as any other received message.

For applications that need to get informed about the arrival of every high priority message **in time**, we recommend to use one of the following setups.

- Use dedicated RX buffers for important messages.
- In case RX FIFOs are used, use RX FIFO0 for low priority messages and RX FIFO1 for high priority messages.

The high priority message feature of the M\_CAN is not required in these cases, because the user knows that all messages in RX FIFO1 are high priority messages.

### Hint 2: Reading a priority message from an RX FIFO

A High Priority Message is indexed via **HPMS.BIDX** instead of the FIFO read index pointer (Get Index). The Host has to take care about the acknowledging of a read message because the **HPMS.BIDX** and **Rx FIFO 0/1 Get Index** will be usually different. If a wrong acknowledgment is done, the Get index of the FIFO would be set wrong and some of the older FIFO elements could be lost.

Possibilities for acknowledging a High Priority Message.

1. Read the high priority message from Rx FIFO and acknowledge the read later
  - Read the high priority message
  - Remember the Get Index of the high priority message
  - Do not acknowledge the high priority message, if this is not the oldest message in the FIFO
  - Later, when the “Get Index of the FIFO” == ”Get index high priority message”, acknowledge this message without reading it again.
2. Read and acknowledge all messages in the FIFO up to the high priority message. This means the **HPM** interrupt is used as trigger to read the RX FIFO.

## 5 Timeout Counter

### Overview

M\_CAN provides a 16-bit timeout counter to signal a timeout condition. It operates as down counter. When the counter reaches zero, the M\_CAN sets interrupt flag **IR.TOO**.

This feature is **not recommended for CAN FD operation with bit rate switching**. This is because the internal timeout counter in the M\_CAN counts “CAN bit times”. In CAN FD with bit rate switching, the bit times in data phase and arbitration phase typically have different lengths. Consequently, the “time” would progress faster in the data phase.

### Configuration

- The timeout counter can be configured via register **TOCC** (Timeout Counter Configuration)
- **Note:** The Timeout Counter feature uses the same prescaler as the timestamp counter **TSCC.TCP**

### Operation

Two different operation modes can be set in **TOCC.TOS**.

- **When operating in Continuous Mode:** The counter starts when **CCCR.INIT** is reset. A write to register **TOCV** (Timeout Counter Value) presets the counter to the value configured by **TOCC.TOP**. After preset the counter continues down-counting. This mode could be used in a setup where the M\_CAN is periodically polled by the host CPU to check for new messages.
- **Controlled by Rx FIFO0, Rx FIFO1, or TX Event FIFO:** When one of the FIFOs controls the Timeout Counter, an empty FIFO presets the counter to the value configured by **TOCC.TOP**. Counting down is started when the first FIFO element is stored. This mode can be used as a watchdog to detect if the Host CPU didn't read **all** FIFO elements in a specific time frame.

### Software Example

Table 6 lists C functions that demonstrate timeout counter usage. The functions are provided with this application note.

Table 6: C functions that demonstrate timeout counter usage

Name:	<code>m_can_timestampcounter_and_timeoutcounter_init(..)</code>
File:	<code>../m_can/m_can.c</code>
Description:	Function configures the timeout usage and the timestamp usage.

## 6 Software Example

The software examples were written for **M\_CAN version 3.2.1**.

Table 7 lists examples that demonstrate handling of Rx messages and acceptance filtering. The functions are provided with this application note.

Table 7: Example C functions that demonstrate Rx handling.

Name:	<code>m_can_an001_receive_messages_simple(..)</code>
File:	<code>../app_notes/app_note_001_rx_handling.c</code>
Description:	This function demonstrates message reception by an M_CAN node. Two M_CAN nodes participate in the example. M_CAN_0 transmits messages and all these messages are received by M_CAN_1 in its Rx FIFO0.
The example demonstrates	Rx FIFO configurations, Global filter configuration
Name:	<code>m_can_an001_receive_messages_with_filtering(..)</code>
File:	<code>../app_notes/app_note_001_rx_handling.c</code>
Description:	This function demonstrates message reception with acceptance filtering. This example is similar to the example in chapter 3.5. Two M_CAN nodes participate in the test. M_CAN_0 transmits messages and M_CAN_1 receives messages according to the configured filter elements.
The example demonstrates:	Rx FIFO configurations, dedicated Rx buffer configuration, Global filter configurations, standard and extended ID message filtering
Name:	<code>m_can_an001_high_priority_message_handling(..)</code>
File:	<code>../app_notes/app_note_001_rx_handling.c</code>
Description:	This function demonstrates one approach to handle a High Priority message. <b>IR.HPM</b> interrupt is used as a trigger to read the Rx FIFO. Additionally <b>IR.RFOWL</b> is also used as a trigger to read and empty the Rx FIFO.
The example demonstrates:	Rx FIFO configurations, M_CAN interrupt configurations, Handling of a High Priority Message, Global filter configuration, Standard ID filter configuration.
Name:	<code>m_can_an001_timeoutcounter_usage_with_rx_fifo(..)</code>
File:	<code>../app_notes/app_note_001_rx_handling.c</code>
Description:	Function demonstrates timeout counter operation when controlled by Rx FIFO0.

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

## 7 List of Tables

Table 1: C functions that demonstrate Rx FIFO operations .....	4
Table 2: C functions that demonstrate dedicated Rx buffer operations .....	7
Table 3: Filter configuration .....	12
Table 4: Result of message ID Filtering .....	12
Table 5: C functions that demonstrate acceptance filter configurations .....	13
Table 6: C functions that demonstrate timeout counter usage .....	16
Table 7: Example C functions that demonstrate Rx handling. ....	17

## 8 List of Figures

Figure 1: Message RAM Rx FIFO section.....	2
Figure 2: Message RAM dedicated Rx Buffer section .....	5
Figure 3: Message RAM Filters section .....	8
Figure 4: Standard Message ID Filter Path .....	10
Figure 5: Extended Message ID Filter Path.....	11
Figure 6: Handling of High Priority Messages.....	14