TMS320F28x Enhanced Controller Area Network (eCAN) Peripheral Reference Guide

Preliminary

Literature Number: SPRU074
May 2002

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of that third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265

Contents

1 Architecture 1-1

Describes the architecture of the enhanced controller area network (eCAN).

1.1 CAN Overview 1-2
1.1.1 Features 1-2
1.1.2 Block Diagram 1-3
1.1.3 eCAN Compatibility With Other TI CAN Modules 1-4
1.2 The CAN Network and Module 1-5
1.2.1 CAN Protocol Overview 1-5
1.3 eCAN Controller Overview 1-8
1.3.1 SCC-Compatible Mode 1-9
1.3.2 Memory Map 1-9
1.3.3 eCAN Registers 1-11
1.4 Message Objects 1-12
1.5 Message Mailbox 1-13
1.5.1 Transmit Mailbox 1-15
1.5.2 Receive Mailbox 1-15
1.6 CAN Power-Down Mode 1-17
1.6.1 Entering and Exiting Low-Power Mode 1-17
1.6.2 Local Power-Down 1-17
1.6.3 Global Power-Down 1-18
1.6.4 Enabling/Disabling Clock to the CAN Module 1-18
1.7 Timer Management Unit 1-19
1.7.1 Time Stamp Functions 1-19
1.7.2 Time-Out Functions 1-20

2 eCAN Configuration 2-1

Describes how to initialize and configure the eCAN module.

2.1 CAN Module Initialization 2-2
2.1.1 CAN Bit-Timing Configuration 2-3
2.1.2 CAN Bit Rate Calculation 2-4
2.2 Steps to Configure eCAN 2-6
2.2.1 Configuring a Mailbox for Transmit 2-7
2.2.2 Transmitting a Message 2-7
2.2.3 Configuring Mailboxes for Receive 2-8
2.2.4 Receiving a Message 2-9

2.2.5 Handling of Overload Situations 2-9
2.3 Handling of Remote Frame Mailboxes 2-11
2.3.1 Requesting Data From Another Node 2-11
2.3.2 Answering a Remote Request 2-11
2.3.3 Updating the Data Field 2-12
2.4 Interrupts 2-13
2.4.1 Interrupts Scheme 2-15
2.4.2 Message Object Interrupt 2-15
2.4.3 Interrupt Handling 2-17
2.4.4 Configuring for Interrupt Handling 2-17
2.4.5 Handling Mailbox Interrupts 2-17

3 eCAN Registers 3-1
Describes the eCAN registers.

3.1 Mailbox Registers 3-2
3.1.1 Transmission Request Set Register (CANTRS) 3-3
3.1.2 Transmission Request Reset Register (CANTRR) 3-4
3.1.3 Transmission Acknowledge Register (CANTA) 3-5
3.1.4 Abort Acknowledge Register (CANAA) 3-5
3.1.5 Message Data Registers (MDL, MDH) 3-6
3.1.6 Receive Message Pending Register (CANRMP) 3-8
3.1.7 Receive Message Lost Register (CANRML) 3-9
3.1.8 Handling of Remote Frames 3-9
3.1.9 CPU Message Mailbox Access 3-10
3.1.10 Remote Frame Pending Register (CANRFP) 3-13
3.2 Acceptance Filter 3-15
3.2.1 Local Acceptance Masks (CANLAM) 3-15
3.2.2 Global Acceptance Mask Register (CANGAM) 3-17
3.3 Master Control Register (CANMC) 3-19
3.3.1 CAN Module Action in SOFT Mode 3-21
3.4 Bit-Timing Configuration Register (CANBTC) 3-23
3.5 Error Registers 3-27
3.5.1 Error and Status Register (CANES) 3-27
3.5.2 CAN Error Counter Registers (CANTEC/CANREC) 3-30
3.6 Interrupt Registers 3-32
3.6.1 Global Interrupt Flag Registers (CANGIF0 / CANGIF1) 3-32
3.6.2 Global Interrupt Mask Register (CANGIM) 3-36
3.6.3 Mailbox Interrupt Mask Register (CANMIM) 3-38
3.6.4 Mailbox Interrupt Level Register (CANMIL) 3-39
3.6.5 Overwrite Protection Control Register (CANOPC) 3-40
3.7 eCAN I/O Control Registers (CANTIOC, CANRIOC) 3-41

Figures

1–1 eCAN Block Diagram and Interface Circuit 1-3
1–2 CAN Data Frame 1-6
1–3 Architecture of the eCAN Module 1-7
1–4 Memory Map 1-10
1–5 Local Network Time Register (LNT) 1-19
1–6 Message Object Time Stamp Registers (MOTS) 1-20
1–7 Message Object Time-Out Registers (MOTO) 1-21
1–8 Time Out Control-Register (TOC) 1-21
1–9 Time Out Status Register (TOS) 1-21
2–1 Initialization Sequence 2-2
2–2 CAN bit timing 2-4
2–3 Interrupts Scheme 2-14
3–1 CAN Local Network Time Register Bits 3-2
3–2 TOS Register 3-2
3–3 Mailbox Enable Register (CANME) 3-3
3–4 Mailbox Direction Register (CANMD) 3-3
3–5 Transmit Request Set Register 3-4
3–6 Transmit Request Reset Register Bits 3-5
3–7 Abort Acknowledge Register Bits 3-6
3–8 Message Data Low Register with DBO = 0 (MDL) 3-7
3–9 Message Data High Register with DBO = 0 (MDH) 3-7
3–10 Message Data Low Register with DBO = 1 (MDL) 3-7
3–11 Message Data High Register with DBO = 1 (MDH) 3-7
3–12 Receive Message Pending Register Bits 3-8
3–13 Receive Message Lost Register Bits 3-9
3–14 Message Identifier Register (MID) 3-11
3–15 Message Control Field Register (MCF) 3-12
3–16 Remote Frame Pending Register Bits 3-13
3–17 Acceptance Filter 3-15
3–18 Local acceptance mask register (LAMn) 3-16
3–19 CANGAM Register Bits (0x24h) 3-17
3–20 CANMC Register Bits 3-19
3–21 CANBTC Register Bits 3-23
3–22 CANES Register Bits 3-27
3–23 Transmit Error Counter Register - CANTEC 3-30
3–24 Receive Error Counter Register - CANREC 3-30

3–25 Global Interrupt Flag 0 Register - CANGIF0 3-33
3–26 Global Interrupt Flag 1 Register - CANGIF1 3-33
3–27 Global Interrupt Mask Register (CANGIM) Bits 3-36
3–28 Mailbox Interrupt Level Register Bits 3-39
3–29 Overwrite Protection Control Register Bits 3-40
3–30 TX I/O Control Register - CANTIOC 3-41
3–31 RX I/O Control Register - CANRIOC 3-42

Tables

1–1 Register Map 1-11
1–2 Message Object Description 1-12
1–3 Message Object Behavior Configuration 1-13
1–4 Mailbox RAM Layout 1-14

Notes

Unused Message Mailboxes 1-9
Unused Message Mailboxes 1-12
First Message Received During Power-Down Mode 1-18
Bit-Timing Configuration (CANBTC) Register With Zero Value 2-2
Enter / Exit Initialization Mode 2-3
Protected Registers Access 2-6
Additional conditions that set the AA[n] flag 3-6
Data Field 3-7
Remote Transmission Request Bit 3-10
Forbidden Configuration Values 3-23

Chapter 1

Architecture

The enhanced 'Controller Area Network' (eCAN) module implemented in the 28x™ DSP is compatible with the CAN 2.0B standard (active). It uses established protocol to communicate serially with other controllers in electrically noisy environments. With 32 fully configurable mailboxes and time stamping feature, the eCAN module provides a versatile and robust serial communication interface.

Topic **Page**

1.1	CAN Overview	1-2
1.2	The CAN Network and Module	1-5
1.3	eCAN Controller Overview	1-8
1.4	Message Objects	1-12
1.5	Message Mailbox	1-14
1.6	CAN Power-Down Mode	1-18
1.7	Timer Management Unit	1-20

28x is a trademark of Texas Instruments.

1.1 CAN Overview

The eCAN uses the CAN protocol kernel (CPK) module to perform the basic CAN protocol tasks. Figure 1–1 shows the major blocks of the eCAN and the interface circuits.

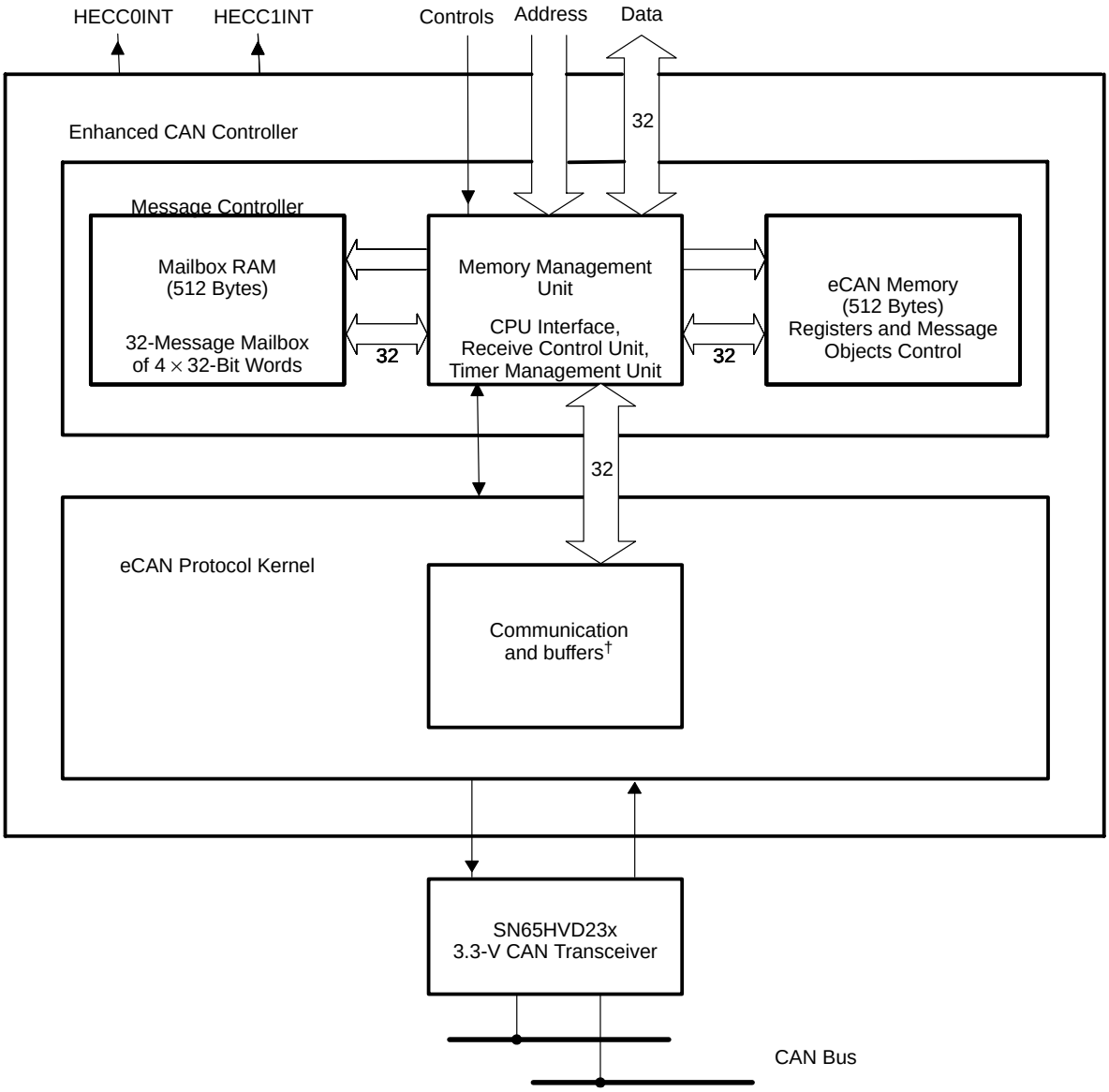
1.1.1 Features

The eCAN module has the following features:

- Fully compliant with CAN protocol, version 2.0B
- Supports data rates up to 1 Mbps
- Thirty-two mailboxes, each with the following properties:
  - Configurable as receive or transmit
  - Configurable with standard or extended identifier
  - Has a programmable receive mask
  - Supports data and remote frame
  - Composed of 0 to 8 bytes of data
  - Uses a 32-bit time stamp on receive and transmit message
  - Protects against reception of new message
  - Holds the dynamically programmable priority of transmit message
  - Employs a programmable interrupt scheme with two interrupt levels
  - Employs a programmable alarm on transmission or reception time-out
- Low-power mode
- Programmable wake-up on bus activity
- Automatic reply to a remote request message
- Automatic retransmission of a frame in case of loss of arbitration or error
- 32-bit local network time counter synchronized by a specific message (communication in conjunction with mailbox 16)
- Self-test mode
  - Operates in a loopback mode receiving its own message. A “dummy” acknowledge is provided, thereby eliminating the need for another node to provide the acknowledge bit.

1.1.2 Block Diagram

Figure 1–1. eCAN Block Diagram and Interface Circuit



† The communication buffers are transparent to the user and are not accessible by user code.

1.1.3 eCAN Compatibility With Other TI CAN Modules

The eCAN module is identical to the high-end CAN controller (HECC) used in the TMS470™ series microcontrollers from Texas Instruments. The eCAN module features several enhancements (such as increased number of mailboxes with individual acceptance masks, time stamping, etc) over the CAN module featured in 240x™ series of DSPs. For this reason, code written for 240x CAN modules cannot be directly ported to eCAN. However, eCAN follows the same register structure and bit functionality as that of 240x CAN (i.e., many registers and bits perform exactly identical functions across these two platforms. This makes code migration a relatively easy task, more so with code written in C language.)

240x and TMS470 are trademarks of Texas Instruments.

1.2 The CAN Network and Module

The controller area network (CAN) uses a serial multimaster communication protocol that efficiently supports distributed real-time control, with a very high level of security, and a communication rate of up to 1 Mbps. The CAN bus is ideal for applications operating in noisy and harsh environments, such as in the automotive and other industrial fields that require reliable communication or multiplexed wiring.

Prioritized messages of up to eight bytes in data length can be sent on a multimaster serial bus using an arbitration protocol and an error-detection mechanism for a high level of data integrity.

1.2.1 CAN Protocol Overview

The CAN protocol supports four different frame types for communication:

- Data frames that carry data from a transmitter node to the receiver nodes
- Remote frames that are transmitted by a node to request the transmission of a data frame with the same identifier
- Error frames that are transmitted by any node on a bus-error detection
- Overload frames that provide an extra delay between the preceding and the succeeding data frames or remote frames.

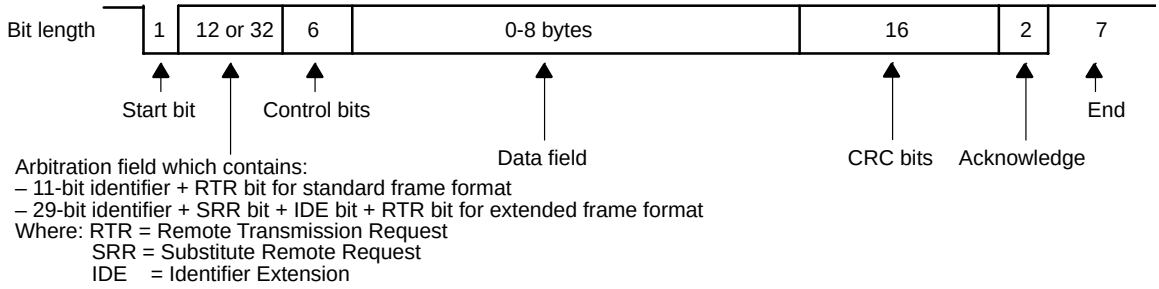
In addition, CAN specification version 2.0B defines two different formats that differ in the length of the identifier field: standard frames with an 11-bit identifier and extended frames with 29-bit identifier.

CAN standard data frames contain from 44 to 108 bits and CAN extended data frames contain 64 to 128 bits. Furthermore, up to 23 stuff bits can be inserted in a standard data frame, and up to 28 stuff bits in an extended data frame, depending on the data-stream coding. The overall maximum data frame length is then 131 bits for a standard frame and 156 bits for an extended frame.

Bit fields within the data frame, shown in Figure 1–2, identify:

- Start of the frame
- Arbitration field containing the identifier and the type of message being sent
- Control field containing the number of data
- Up to 8 bytes of data
- Cyclic redundancy check (CRC)
- Acknowledgment
- End-of-frame bits

Figure 1–2. CAN Data Frame

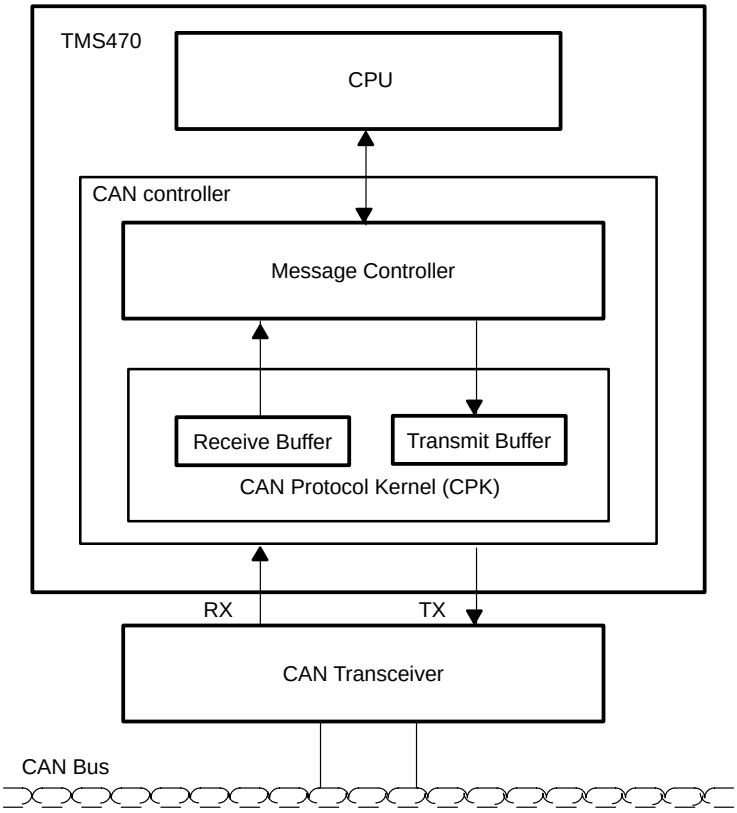


Note: Unless otherwise noted, numbers are amount of bits in field.

The TMS320F28x CAN controllers provide the CPU with full functionality of the CAN protocol, version 2.0B. The CAN controller minimizes the CPU's load in communication overhead and enhances the CAN standard by providing additional features.

The architecture of eCAN module, shown in Figure 1–3, is composed of a CAN protocol kernel (CPK) and a message controller.

Figure 1–3. Architecture of the eCAN Module



Two functions of the CPK are to decode all messages received on the CAN bus according to the CAN protocol and to transfer these messages into a receive buffer. Another CPK function is to transmit messages on the CAN bus according to the CAN protocol.

The message controller of a CAN controller is responsible for determining if any message received by the CPK must be preserved for the CPU use or be discarded. At the initialization phase, the CPU specifies to the message controller all message identifiers used by the application. The message controller is also responsible for sending the next message to transmit to the CPK according to the message's priority.

1.3 eCAN Controller Overview

The eCAN is a new-generation, Texas Instruments, advanced CAN controller with an internal 32-bit architecture.

The eCAN module consists of:

- The CAN protocol kernel (CPK)
- The message controller comprising:
  - The memory management unit (MMU), including the CPU interface and the receive control unit (acceptance filtering), and the timer management unit
  - 512 bytes of mailbox RAM enabling the storage of 32 messages
  - 512 bytes of memory comprising the registers and the message objects control

After the reception of a valid message by the CPK, the receive control unit of the message controller determines if the received message must be stored into one of the 32 message objects of the mailbox RAM. The receive control unit checks the state, the identifier, and the mask of all message objects to determine the appropriate mailbox location. The received message is stored into the first mailbox passing the acceptance filtering. If the receive control unit could not find any mailbox to store the received message, the message is discarded.

A message is composed of an 11- or 29-bit identifier, a control field, and up to 8 bytes of data.

When a message must be transmitted, the message controller transfers the message into the transmit buffer of the CPK in order to start the message transmission at the next bus-idle state. When more than one message must be transmitted, the message with the highest priority (defined by the message-object-priority register) that is ready to be transmitted is transferred into the CPK by the message controller.

The timer management unit comprises a local network time counter and apposes a time stamp to all messages received or transmitted. The timer management unit controls all message reception and transmission, and generates an alarm when a message has not been received or transmitted during an allowed period of time (time-out).

To initiate a data transfer, the transmission request bit has to be set in the corresponding control register. The entire transmission procedure and possible error handling are then performed without any CPU involvement. If a mailbox has been configured to receive messages, the CPU easily reads its data registers using CPU read instructions. The mailbox may be configured to interrupt the CPU after every successful message transmission or reception.

1.3.1 SCC-Compatible Mode

The eCAN can be used in SCC mode. In this mode, all functions specific to the eCAN are not available and the eCAN behaves exactly as the SCC. This mode is selected by default to allow any application software written for the SCC to run on the eCAN without any modification.

The SCC-compatible mode is selected with bit SCM (MC.13).

1.3.2 Memory Map

The HECC module has two different offset addresses mapped in the TMS320F28x memory. The first offset address, *HECC_Offset*, is used to access the control register, the status register, the acceptance mask, the time stamp, and the time-out of the message objects. The access to the control and status registers (memory range: *HECC_Offset* ... *HECC_Offset* + *0x07C*) is limited to 32-bit wide accesses. The local acceptance masks, the time stamp registers, and the time-out registers can be accessed 8-bit, 16-bit and 32-bit wide. The second offset address, *Mailbox_RAM_Offset,* is used to access the mailboxes. This memory range can be accessed 8-bit, 16-bit and 32-bit wide. Each of these two memory blocks, shown in Figure 1–4, uses 512 bytes of address space.

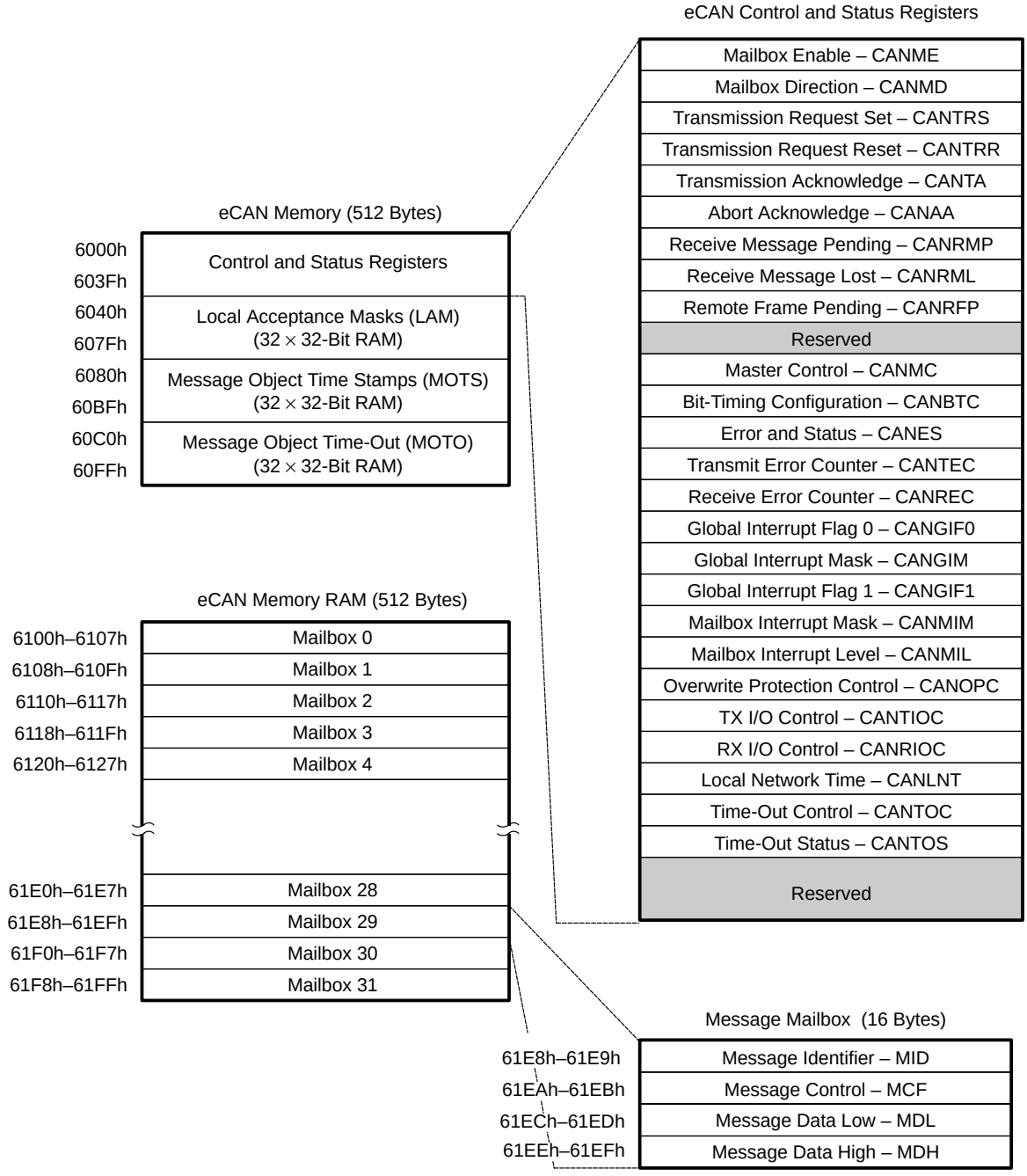
The message storage is implemented by a RAM that can be addressed by the CAN controller or the CPU. The CPU controls the CAN controller by modifying the various mailboxes in the RAM or the additional registers. The contents of the various storage elements are used to perform the functions of the acceptance filtering, message transmission, and interrupt handling.

The mailbox module in the HECC provides 32 message mailboxes of 8-byte data length, a 29-bit identifier, and several control bits. Each mailbox can be configured as either transmit or receive. In the HECC, each mailbox has its individual acceptance mask.

Note: Unused Message Mailboxes

All RAM areas are byte-writable and can be used as normal memory. In this case, the user must ensure that no CAN function uses the RAM area. This assurance is reached by disabling the corresponding mailbox or by disabling the corresponding functions.

Figure 1–4. Memory Map



1.3.3 eCAN Registers

The eCAN registers listed in Table 1–1 are used by the CPU to configure and control the CAN controller and the message objects.

Table 1–1. Register Map

Register Name	Address	Description
CANME	0x00 6000	Mailbox enable
CANMD	0x00 6002	Mailbox direction
CANTRS	0x00 6004	Transmit request set
CANTRR	0x00 6006	Transmit request reset
CANTA	0x00 6008	Transmission acknowledge
CANAA	0x00 600A	Abort acknowledge
CANRMP	0x00 600C	Receive message pending
CANRML	0x00 600E	Receive message lost
CANRFP	0x00 6010	Remote frame pending
Reserved	0x00 6012	Reserved
CANMC	0x00 6014	Master control
CANBTC	0x00 6016	Bit-timing configuration
CANES	0x00 6018	Error and status
CANTEC	0x00 601A	Transmit error counter
CANREC	0x00 601C	Receive error counter
CANGIF0	0x00 601E	Global interrupt flag 0
CANGIM	0x00 6020	Global interrupt mask
CANGIF1	0x00 6022	Global interrupt flag 1
CANMIM	0x00 6024	Mailbox interrupt mask
CANMIL	0x00 6026	Mailbox interrupt level
CANOPC	0x00 6028	Overwrite protection control
CANTIOC	0x00 602A	TX I/O control
CANRIOC	0x00 602C	RX I/O control
CANLNT	0x00 602E	Local network time (Reserved in SCC mode)
CANTOC	0x00 6030	Time-out control (Reserved in SCC mode)
CANTOS	0x00 6032	Time-out status (Reserved in SCC mode)

† These registers are mapped to Peripheral Frame 1. This space allows 16-bit and 32-bit accesses.

1.4 Message Objects

The message controller can handle 32 different message objects.

Each message object can be configured to either transmit or receive. Each message object has its individual acceptance mask.

A message object consists of 28 bytes of RAM distributed, as shown in Table 1–2, as:

- A message mailbox comprising
  - The 29-bit message identifier
  - The message control register
  - 8 bytes of message data
- A 29-bit acceptance mask
  - A 32-bit time stamp
  - A 32-bit time-out

Furthermore, corresponding control and status bits located in the registers allow control of the message objects.

Table 1–2. Message Object Description

Offset / Address†	Mnemonic	Name
Mailbox_RAM_Offset + (Object# 0x10) + 0x00	MID	Message identifier
Mailbox_RAM_Offset + (Object# 0x10) + 0x04	MCF	Message control field
Mailbox_RAM_Offset + (Object# 0x10) + 0x08	MDL	Message data low word
Mailbox_RAM_Offset + (Object# 0x10) + 0x0C	MDH	Message data high word
HECC_Offset + (Object# 4) + 0x80	LAM	Local acceptance mask
HECC_Offset + (Object# 4) + 0x100	MOTS	Message object time stamp
HECC_Offset + (Object# 4) + 0x180	MOTO	Message object time-out

† The actual addresses of the message objects are device-specific. See the specific device data sheet to verify the HECC module memory offset and RAM offset.

Note: Unused Message Mailboxes

Message mailboxes not used by the application for CAN message (disabled in the CANME register) may be used as general memory by the CPU.

1.5 Message Mailbox

The message mailboxes are the RAM area where the CAN messages are actually stored after they were received or before they are transmitted.

The CPU may use the RAM area of the message mailboxes that are not used for storing messages as normal memory. This RAM area, unlike the register area, can be accessed by byte.

Each mailbox contains:

- The message identifier
  - 29 bits for extended identifier
  - 11 bits for standard identifier
- The identifier extension bit, IDE (MID.31)
- The acceptance mask enable bit, AME (MID.30)
- The auto answer mode bit, AAM (MID.29)
- The remote transmission request bit, RTR (MCF.4)
- The data length code, DLC (MCF.3-0)
- Up to eight bytes for the data field
- A transmit priority level, TPL, register on the eCAN (MCF.12:8)

Each of the mailboxes can be configured as one of four message object types (see Table 1–3). Transmit and receive message objects are used for data exchange between one sender and multiple receivers (1 to n communication link), whereas request and reply message objects are used to set up a one-to-one communication link. Table 1–4 lists the mailbox RAM layout.

Table 1–3. Message Object Behavior Configuration

Message Object Behavior	Mailbox Direction Register (CANMD)	Auto-Answer Mode Bit (AAM)	Remote Transmission Request Bit (RTR)
Transmit message object	0	0	0
Receive message object	1	0	0
Request message object	1	0	1
Reply message object	0	1	0

Table 1–4. Mailbox RAM Layout

MB	MID MIDL – MIDH	MCF MCF – Rsvd	MDL MDL_L–MDL_H	MDH MDH_L–MDH_H
0	6100 - 6101	6102 - 6103	6104 - 6105	6106 - 6107
1	6108 - 6109	610A - 610B	610C - 610D	610E - 610F
2	6110 - 6111	6112 - 6113	6114 - 6115	6116 - 6117
3	6118 - 6119	611A - 611B	611C - 611D	611E - 611F
4	6120 - 6121	6122 - 6123	6124 - 6125	6126 - 6127
5	6128 - 6129	612A - 612B	612C - 612D	612E - 612F
6	6130 - 6131	6132 - 6133	6134 - 6135	6136 - 6137
7	6138 - 6139	613A - 613B	613C - 613D	613E - 613F
8	6140 - 6141	6142 - 6143	6144 - 6145	6146 - 6147
9	6148 - 6149	614A - 614B	614C - 614D	614E - 614F
10	6150 - 6151	6152 - 6153	6154 - 6155	6156 - 6157
11	6158 - 6159	615A - 615B	615C - 615D	615E - 615F
12	6160 - 6161	6162 - 6163	6164 - 6165	6166 - 6167
13	6168 - 6169	616A - 616B	616C - 616D	616E - 616F
14	6170 - 6171	6172 - 6173	6174 - 6175	6176 - 6177
15	6178 - 6179	617A - 617B	617C - 617D	617E - 617F
16	6180 - 6181	6182 - 6183	6184 - 6185	6186 - 6187
17	6188 - 6189	618A - 618B	618C - 618D	618E - 618F
18	6190 - 6191	6192 - 6193	6194 - 6195	6196 - 6197
19	6198 - 6199	619A - 619B	619C - 619D	619E - 619F
20	61A0 - 61A1	61A2 - 61A3	61A4 - 61A5	61A6 - 61A7
21	61A8 - 61A9	61AA - 61AB	61AC - 61AD	61AE - 61AF
22	61B0 - 61B1	61B2 - 61B3	61B4 - 61B5	61B6 - 61B7
23	61B8 - 61B9	61BA - 61BB	61BC - 61BD	61BE - 61BF
24	61C0 - 61C1	61C2 - 61C3	61C4 - 61C5	61C6 - 61C7
25	61C8 - 61C9	61CA - 61CB	61CC - 61CD	61CE - 61CF
26	61D0 - 61D1	61D2 - 61D3	61D4 - 61D5	61D6 - 61D7
27	61D8 - 61D9	61DA - 61DB	61DC - 61DD	61DE - 61DF
28	61E0 - 61E1	61E2 - 61E3	61E4 - 61E5	61E6 - 61E7
29	61E8 - 61E9	61EA - 61EB	61EC - 61ED	61EE - 61EF
30	61F0 - 61F1	61F2 - 61F3	61F4 - 61F5	61F6 - 61F7
31	61F8 - 61F9	61FA - 61FB	61FC - 61FD	61FE - 61FF

1.5.1 Transmit Mailbox

The CPU stores the data to be transmitted in a mailbox configured as transmit mailbox. After writing the data and the identifier into the RAM, the message is sent if the corresponding TRS[*n*] bit has been set.

If more than one mailbox is configured as transmit mailbox and more than one corresponding TRS[*n*] is set, the messages are sent one after another in falling order beginning with the mailbox with the highest priority.

In the SCC-compatibility mode, the priority of the mailbox transmission depends on the mailbox number. The highest mailbox number (=15) comprises the highest transmit priority.

In the eCAN, the priority of the mailbox transmission depends on the setting of the TPL field in the message control field (CANMCF) register. The mailbox with the highest value in the TPL is transmitted first. Only when two mailboxes have the same value in the TPL register is the higher numbered mailbox transmitted first. See Section 3.5.3.6 on page 3-28 for more information about the CANMCF.

If a transmission fails due to a loss of arbitration or an error, the message transmission will be reattempted. Before reattempting the transmission, the CAN module checks if other transmissions are requested and then transmits the mailbox with the highest priority.

1.5.2 Receive Mailbox

The identifier of each incoming message is compared to the identifiers held in the receive mailboxes using the appropriate mask. When equality is detected, the received identifier, the control bits, and the data bytes are written into the matching RAM location. At the same time, the corresponding receive-message-pending bit, RMP[*n*] (RMP.31-0), is set and a receive interrupt is generated if enabled. If no match is detected, the message is not stored.

When a message is received, the message controller starts looking for a matching mailbox at the mailbox with the highest mailbox number. Mailbox 15 of the SCC and of the HECC in SCC compatible mode has the highest receive priority; mailbox 31 has the highest receive priority of the HECC in HECC mode.

RMP[*n*] (RMP.31-0) has to be reset by the CPU after reading the data. If a second message has been received for this mailbox and the receive-message-pending bit is already set, the corresponding message-lost bit (RML[*n*] (RML.31-0)) is set. In this case, the stored message is overwritten with the new data if the overwrite-protection bit OPC[*n*] (OPC.31-0) is cleared; otherwise, the next mailboxes are checked.

If a mailbox is configured as a receive mailbox and the RTF bit is set for it, the mailbox can send a remote frame. Once the remote frame is sent, the TRS bit of the mailbox is cleared by the CAN module.

1.6 CAN Power-Down Mode

There are two different power-down modes: the global power-down mode, when all clocks are stopped by the CPU, and the local power-down mode, when only the CAN module internal clock is deactivated by the CAN module itself. The global power down is not implemented in the TMS320F2810.

1.6.1 Entering and Exiting Low-Power Mode

Since the CAN module is connected to multiple modes across a network, you must take care before entering and exiting low-power modes. A CAN packet must be received in full by all the nodes; therefore, if transmission is aborted half-way through the process, the aborted packet would violate the CAN protocol resulting in all the nodes generating error frames. The node exiting LPM should do so unobtrusively. For example, if a node exits LPM when there is traffic on the CAN bus it could “see” a truncated packet and disturb the bus with error frames.

The following points must be considered before entering a low-power mode:

1) The CAN module has completed the tansmission of the lst packet requested.

2) The CAN module has signaled to the CPU that it is ready to enter LPM.

1.6.2 Local Power-Down

During local power-down mode, the clock of the CAN module is turned off and only the wake-up logic is still active. The other peripherals continue to operate normallly.

The local power-down mode is requested by writing a 1 to the PDR (MC.11) bit, allowing transmission of any packet in progress to complete. After the transmisssion is completed, the status bit PDA (ES.3) is set. This confirms that the CAN module has entered the power-down mode.

The value read on the CANES register is 0x08 (PDA bit is set). All other register read accesses will deliver the value 0x00.

The module leaves the local power-down mode when the PDR bit is cleared or if any bus activity is detected on the CAN bus line (if the wake-up-on bus activity is enabled).

The automatic wake-up-on bus activity can be enabled or disabled with the configuration bit WUBA of MC register. If there is any activity on the CAN bus line, the module begins its power-up sequence. The module waits until it detects 11 consecutive recessive bits on the CANRX pin and then it goes bus-active.

Note: First Message Received During Power-Down Mode

The first CAN message, which initiates the bus activity, cannot be received. This means that the first message received in power-down and automatic wake-up mode is lost.

After leaving the sleep mode, the PDR and PDA bits are cleared. The CAN error counters remain unchanged.

If the module is transmitting a message when the PDR bit is set, the transmission is continued until a successful transmission, a lost arbitration, or an error condition on the CAN bus line occurs. Then, the PDA bit is activated so the module causes no error condition on the CAN bus line.

To implement the local power-down mode, two separate clocks are used within the CAN module. One clock stays active all the time to ensure power-down operation (i.e., the wake-up logic and the write and read access to the PDA (ES.3) bit). The other clock is enabled depending on the setting of the PDA bit.

1.6.3 Global Power-Down

Global power-down mode is requested by the system module (SYS LPM) and it does so immediately upon rceiving the SYS_LPM request. When the module is able to enter the global power-down mode, all clocks to the CAN module are disabled. A low-power mode for the CAN module (that is, a local power-down mode) must be request before a global LPM request is made.

1.6.4 Enabling/Disabling Clock to the CAN Module

Clock to the CAN module can be enabled or disabled using bit 14 of the PCLKCR register. This bit is useful in applications that do not use the CAN module at all. In such applications, the CAN module clock can be permanently turned off, resulting some power saving. This bit is not intended to put the CAN module in low-power mode and should not be used for the purpose.

1.7 Timer Management Unit

Several functions are implemented in the HECC to control the time when messages are transmitted or should be transmitted. A separate state machine is included in the HECC to handle the time-control functions. This state machine has lower priority when accessing the registers than the CAN state machine has. Therefore, the time-control functions may be delayed by other ongoing actions.

1.7.1 Time Stamp Functions

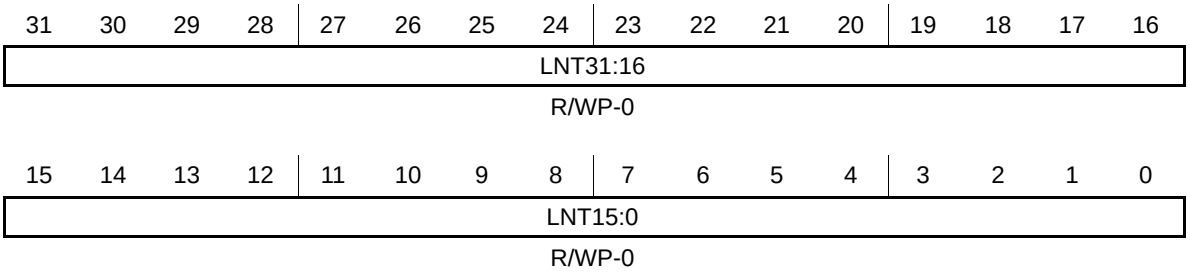
To get an indication of the time of reception or transmission of a message, a free-running 32-bit timer (LNT) is implemented in the module. Its content is written into the time stamp register of the corresponding mailbox (MOTS) when a received message is stored or a message has been transmitted.

The counter is driven from the bit clock of the CAN bus line. The timer is stopped during the initialization mode or if the module is in sleep or suspend mode. After power-up reset, the free-running counter is cleared.

The most significant bit of the LNT register is cleared by writing a 1 to LNTM (MC.14). The LNT register may also be cleared when mailbox 16 transmitted or received (depending on the setting of MD.16 bit) a message successfully. This is enabled by setting the LNTC bit (MC.15). Therefore, it is possible to use mailbox 16 for global time synchronization of the network. The CPU can read and write the counter.

Overflow of the counter is detected by the LNT-counter-overflow-interrupt flag (GIF.16). An overflow occurs when the highest bit of the LNT counter changes to 1. Therefore, the CPU has enough time to handle this situation.

Figure 1–5. Local Network Time Register (LNT)

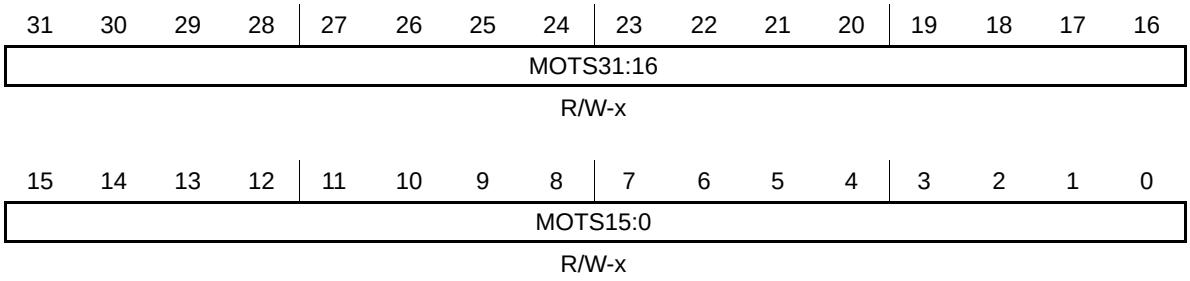


Note: R = Read, WP = Write in privilege mode only, -*n* = Value after reset, x = indeterminate

Note: HECC only, reserved in the SCC

Bit(s)	Name	Description
31:0	LNT.31:0	Local network time register. Value of the local network time counter used for the time-stamp and the time-out functions.

Figure 1–6. Message Object Time Stamp Registers (MOTS)



Legend: R/W = Read/Write, -*n* = Value after reset, x = indeterminate

Bit(s)	Name	Description
31:0	MOTS.31:0	Message object time stamp register. Value of the local network time counter (LNT) when the message has been actually received or transmitted.

1.7.2 Time-Out Functions

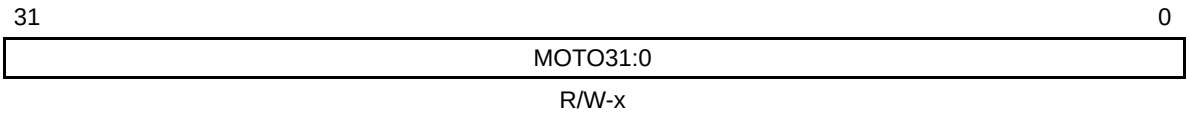
To ensure that all messages are sent or received within a predefined period, each mailbox has its own time-out register. If a message has not been sent or received by the time indicated in the time-out register and the corresponding bit TOC[*n*] is set in the TOC register, a flag is set in the time-out status register (TOS).

For transmit mailboxes the TOS[*n*] flag is cleared when the TOC[*n*] bit is cleared or when the corresponding TRS[*n*] bit is cleared, no matter whether due to successful transmission or abortion of the transmit request. For receive mailboxes, the TOS[*n*] flag is cleared when the corresponding TOC[*n*] bit is cleared.

The CPU may also clear the time-out status register flags by writing a 1 into the time-out status register.

The message object time-out registers (MOTO) are implemented as a RAM. The state machine scans all the MOTO registers and compares them to the LNT counter value. If the value in the LNT register is equal to or greater than the value in the time-out register, and the corresponding TRS bit (applies to transmit mailboxes only) is set, and the TOC[*n*] bit is set, the appropriate bit TOS[*n*] is set. Since all the time-out registers are scanned sequentially, there may be a delay before the TOS[*n*] bit is set.

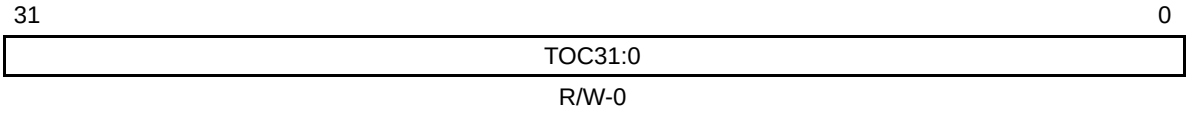
Figure 1–7. Message Object Time-Out Registers (MOTO)



Note: R/W = Read/Write, -*n* = Value after reset, x = indeterminate

Bit(s)	Name	Description
31:0	MOTO.31:0	Message object time-out register. Limit value of the local network time counter (LNT) to actually transmit or receive the message.

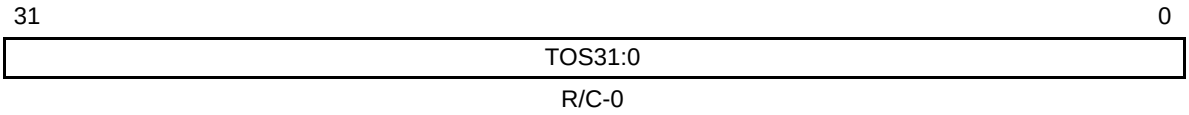
Figure 1–8. Time Out Control-Register (TOC)



Note: R/W = Read/Write, -*n* = Value after reset, x = indeterminate

Bit(s)	Name	Description
31:0	TOC 31:0	Time-out control register 1 The TOC[*n*] bit must be set by the CPU to enable the time-out function for mailbox *n*. Before setting the TOC[*n*] bit, the corresponding MOTO register should be loaded with the time-out value relative to LNT. 0 The time-out function is disabled. The TOS[*n*] flag is never set.

Figure 1–9. Time Out Status Register (TOS)



Note: R/C = Read/Clear, -*n* = Value after reset, x = indeterminate

Bit(s)	Name	Description
31:0	TOS 31:0	Time-out status register 1 The value in the LNT register is larger or equal to the value in the time-out register that corresponds to mailbox *n* and the TOC[*n*] bit is set. 0 No time-out occurred or it is disabled for that mailbox.

Chapter 2

eCAN Configuration

This section explains the process of initialization and describes the procedures to configure the eCAN module.

Topic **Page**

2.1	CAN Module Initialization	2-2
2.2	Steps to Configure eCAN	2-6
2.3	Handling of Remote Frame Mailboxes	2-11
2.4	Interrupts	2-13

2.1 CAN Module Initialization

The CAN module must be initialized before the utilization. Initialization is only possible if the module is in initialization mode. Figure 2–1 is a flow chart showing the process.

Programming CCR (MC.12) = 1 sets the initialization mode. The initialization can be performed only when CCE (ES.4) = 1. Afterwards, the configuration registers may be written.

In order to modify the global acceptance mask register (CANGAM) and the two local acceptance mask registers [LAM(0) and LAM(3)] of the SCC, the CAN module also must be set in the initialization mode.

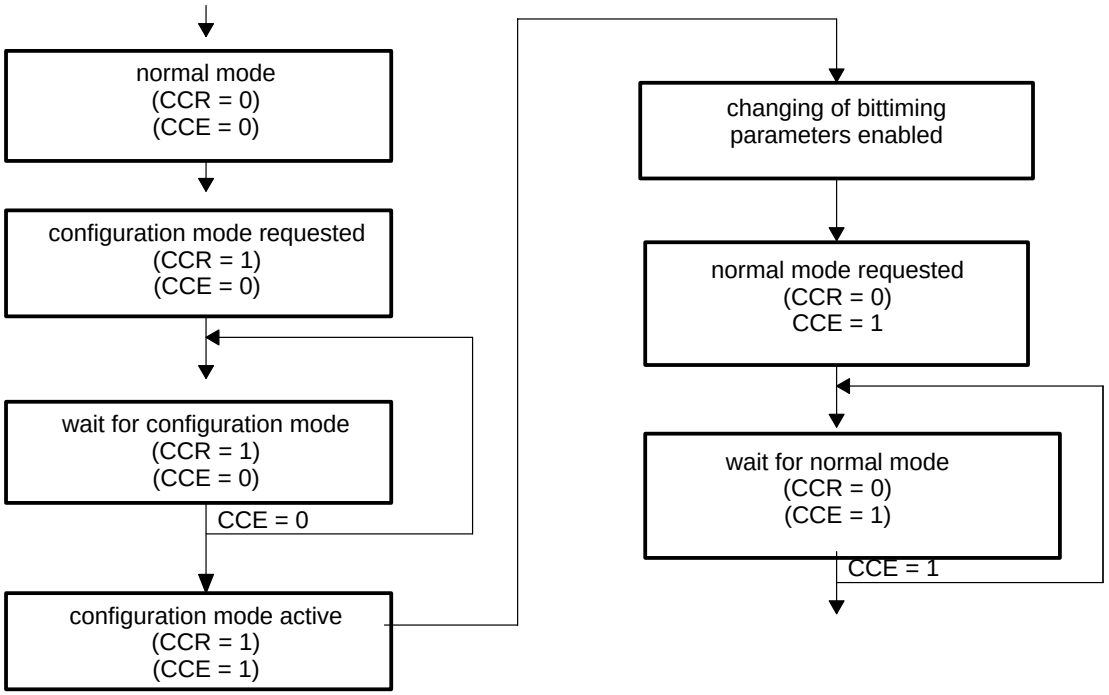
The module is activated again by programming CCR(MC.12) = 0.

After hardware reset, the initialization mode is active.

Note: Bit-Timing Configuration (CANBTC) Register With Zero Value

If the CANBTC register is programmed with a zero value, or left with the initial value, the CAN module will never leave the initialization mode, i.e. CCE (ES.4) bit will remain at 1 when clearing the CCR bit.

Figure 2–1. Initialization Sequence



Note: Enter / Exit Initialization Mode

The transition between initialization mode and normal mode and vice-versa is performed in synchronization with the CAN network. That is, the CAN controller waits until it detects a bus idle sequence (= 11 recessive bits) before it changes the mode. In the event of a stuck-to-dominant bus error, the CAN controller cannot detect a bus-idle condition and therefore is unable to perform a mode transition.

2.1.1 CAN Bit-Timing Configuration

The CAN protocol specification partitions the nominal bit time into four different time segments:

SYNC_SEG: this part of bit time is used to synchronize the various nodes on the bus. An edge is expected to lie within this segment. This segment is always 1 TIME QUANTUM (TQ).

PROP_SEG: this part of the bit time is used to compensate for the physical delay times within the network. It is twice the sum of the signal's propagation time on the bus line, the input comparator delay, and the output driver delay. This segment is programmable from 1 to 8 TIME QUANTA (TQ).

PHASE_SEG1: this phase is used to compensate for positive edge phase error. This segment is programmable from 1 to 8 TIME QUANTA (TQ) and can be lengthened by resynchronization.

PHASE_SEG2: this phase is used to compensate for negative edge phase error. This segment is programmable from 2 to 8 TIME QUANTA (TQ) and can be shortened by resynchronization.

All controllers on a CAN bus must have the same bit rate and bit length. At different clock frequencies of the individual controllers, the bit rate has to be adjusted by the time segments.

In the eCAN module, the length of a bit on the CAN bus is determined by the parameters TSEG1 (BTC.6-3), TSEG2 (BTC.2-0), and BRP (BTC.23.16). (*BRP* is the binary value of BRP.[7:0] + 1.) See Figure 2–2.

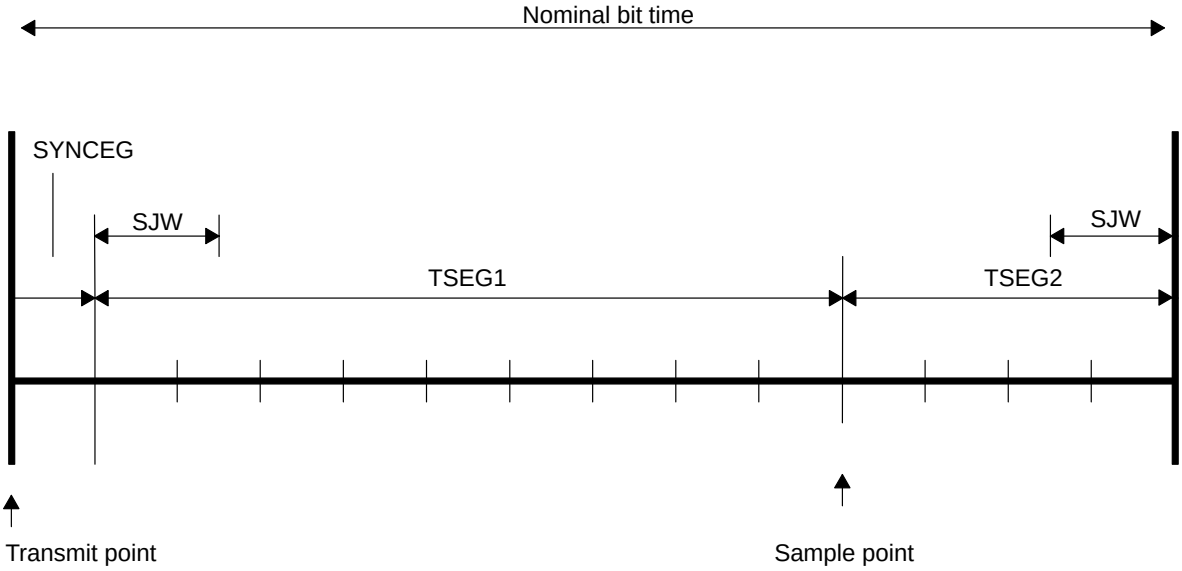
TSEG1 combines the two time segments PROP_SEG and PHASE_SEG1 as defined by the CAN protocol. TSEG2 defines the length of the time segment PHASE_SEG2.

The following bit timing rules must be fulfilled when determining the bit segment values:

- $TSEG1_{CALC(min)} \geq TSEG2_{CALC}$
- $IPT \leq TSEG1_{CALC} \geq 16$ TQ (IPT = Information Processing Time)

- ❏ IPT ≤ $TSEG2_{CALC}$ ≤ 8 TQ
- ❏ IPT = 3 / BRP_{CALC} (the resulting IPT has to be rounded up to the next integer value)
- ❏ 1 TQ ≤ SJW_{CALC} min[4 TQ, $TSEG2_{CALC}$] (SJW = Synchronization Jump Width)
- ❏ To utilize three-time sampling mode, BRP_{CALC} ≥ 5 has to be selected

Figure 2–2. CAN bit timing



2.1.2 CAN Bit Rate Calculation

Bit rate is calculated in bits per second as follows:

$$Bitrate = \frac{ICLK}{BRP \times Bit\ Time}$$

Where *BitTime* is the number of time quanta (TQ) per bit. *ICLK* is the CAN module system clock frequency. *BRP* is the binary value of BRP[7:0] + 1 (BTC.23-16).

BitTime is defined as follows:

BitTime = TSEG1 + TSEG2 + 1

Example:

With ICLK = 8 MHz, if a bit rate of 1 Mbits/s is required, the bit-timing parameters should be programmed as follows:

$$BRP = 1 \rightarrow TQ = \frac{1}{8\ MHz} = 1/8\ \mu s$$

TSEG1 (BTC.6-3) = 4 TQ, TSEG2 (BTC.2-0) = 3 TQ → BitTime = 8 TQ, Bitrate = 1 Mbps

With this setting, a threefold sampling of the bus is not possible; thus SAM (BTC.7) must be set to 0. Since SJW (BTC.9-8) is not allowed to be greater than TSEG2, it is set to 3 TQ (SJW = 3). (See Section 3.4 on page 3-23 for more information about the CANBTC register.)

BTC = 0x0000021A

2.2 Steps to Configure eCAN

The following steps must be performed to configure the eCAN for operation:

Note: Protected Registers Access.

This sequence must be done in TMS470 privilege mode.

1) Set the CANTX and the CANRX pins to CAN functions:

   Write CANTIOC.3:0 = 0x08

   Write CANRIOC.3:0 = 0x08

2) After a reset, bit CCR (MC.12) and bit CCE (ES.4) are set to 1. This allows the user to configure the bit-timing configuration register (CANBTC).

   If the CCE bit is set (ES.4 = 1), proceed to next step; otherwise, set the CCR bit (MC.12 = 1) and wait until CCE bit is set (ES.4 = 1).

3) Program the CANBTC register with the appropriate timing values. Make sure that the values TSEG1 and TSEG2 are not 0. If they are 0, the module does not leave the initialization mode. For example:

   Write BTC = 0x1021A

4) For the SCC, program the acceptance masks now. For example:

   Write LAM(3) = 0x3c0000

5) Program the master control register (CANMC) as follows:

   Clear CCR (MC.12) = 0

   Clear PDR (MC.11) = 0

   Clear DBO (MC.10) = 0

   Clear WUBA (MC.9)= 0

   Clear CDR (MC.8) = 0

   Clear ABO (MC.7) = 0

   Clear STM (MC.6) = 0

   Clear SRES (MC.5) = 0

   Clear MBNR (MC.4-0) = 0

6) Verify the CCE bit is cleared (ES.4 = 0), indicating that the CAN module has been configured.

This completes the setup for the basic functionality.

2.2.1 Configuring a Mailbox for Transmit

To transmit a message, the following steps need to be performed (in this example, for object 1):

1) Clear the appropriate bit in the CANTRS register to 0:

   Clear TRS.1 = 0 (Writing a 0 to TRS has no effect; instead, set TRR.1 and wait until TRS.1 clears.) If the RTR bit is set, the TRS bit can send a remote frame. Once the remote frame is sent, the TRS bit of the mailbox is cleared by the CAN module. The same node can be used to request a data frame from another node.

2) Disable the object by clearing the corresponding bit in the mailbox enable (CANME) register.

   Clear ME.1 = 0

3) Load the message identifier (MID) register of the object. Clear the AME (MID.30) and AAM (MID.29) bits for a normal send mailbox (MID.30 = 0 and MID.29 = 0). This register is usually not modified during operation. It can only be modified when the mailbox is disabled. For example:

   Write MID(1) = 0x15ac0000

   Write the data length into the DLC field of the message control field register (MCF.3:0). The RTR flag is usually cleared (MCF.4 = 0). The CANMCF register is usually not modified during operation and can only be modified when the object is disabled. For example:

   MCF(1) = 2

   Set the mailbox direction by clearing the corresponding bit in the CANMD register.

   Clear MD.1 = 0

4) Set the mailbox enable by setting the corresponding bit in the CANME register

   Set ME.1 = 1

This configures object 1 for transmit mode.

2.2.2 Transmitting a Message

To start a transmission (in this example, for object 1):

1) Write the message data into the mailbox data field.

   Since DBO (MC.10) is set to zero in the configuration section and MCF(1) is set to 2, the data are stored in the 2 MSBytes of MDL(1).

Write MDL(1) = xxxx0000h

2) Set the corresponding flag in the transmit request register (CANTRS.1 = 1) to start the transmission of the message. The CAN module now handles the complete transmission of the CAN message.

3) Wait until the transmit-acknowledge flag of the corresponding mailbox is set (TA.1 = 1). After a successful transmission, this flag is set by the CAN module.

4) The TRS flag is reset to 0 by the module after a successful or aborted transmission (TRS.1 = 0).

5) The transmit acknowledge must be cleared for the next transmission.

   Set TA.1 = 1

   Wait until read TA.1 is 0

6) To transmit another message in the same message object, the mailbox RAM data must be updated. Setting the TRS.1 flag starts the next transmission. Writing to the mailbox RAM can be half-word (16 bits) or full word (32 bits) but the module will always return 32-bit from even boundary. The CPU must accept all the 32 bits or part of it.

2.2.3 Configuring Mailboxes for Receive

To configure a mailbox to receive messages, the following steps must be performed (in this example, mailbox 3):

1) Disable the mailbox by clearing the corresponding bit in the mailbox enable (CANME) register.

   Clear ME.3 = 0

2) Write the selected identifier into the corresponding MID register. The identifier extension bit must be configured to fit the expected identifier. If the acceptance mask is used, the acceptance mask enable (AME) bit must be set (MID.30 = 1). For example:

   Write MID(3) = 0x4f780000

3) If the AME bit is set to 1, the corresponding acceptance mask must be programmed.

   Write LAM(3) = 0x03c0000.

4) Configure the mailbox as a receive mailbox by setting the corresponding flag in the mailbox direction register (MD.3 = 1). Make sure no other bits in this register are affected by this operation.

5) If data in the mailbox is to be protected, the overwrite protection control register (CANOPC) should be programmed now. This protection is useful if no message must be lost. If OPC is set, the software has to make sure that an additional mailbox (buffer mailbox) is configured to store 'overflow' messages. Otherwise messages can be lost without notification.

   Write OPC.3 = 1

6) Enable the mailbox by setting the appropriate flag in the mailbox enable register (CANME). This should be done by reading CANME, and writing back (CANME |= 0x0008) to make sure no other flag has changed accidentally.

The object is now configured for the receive mode. Any incoming message for that object is handled automatically.

2.2.4 Receiving a Message

This example uses message object 3. When a message is received, the corresponding flag in the receive message pending register (CANRMP) is set to 1 and an interrupt may be initiated. The CPU may then read the message from the mailbox RAM. Before the CPU reads the message from the mailbox, it should first clear the RMP bit (RMP.3 = 1). The CPU should also check the receive message lost flag RML.3 = 1. Depending on the application, the CPU has to decide how to handle this situation.

After reading the data, the CPU needs to check that the RMP bit has not been set again by the module. If the RMP bit has been set to 1, the data may have been corrupted. The CPU needs to read the data again because a new message was received while the CPU was reading the old one.

2.2.5 Handling of Overload Situations

If the CPU is not able to handle important messages fast enough, it may be advisable to configure more than one mailbox for that identifier. Here is an example where the objects 3, 4, and 5 have the same identifier and share the same mask. For the SCC, the mask is LAM(3). For the HECC, each object has its own LAM: LAM(3), LAM(4), and LAM(5), all of which need to be programmed with the same value.

To make sure that no message is lost, objects 4 and 5 set the OPC flag, which will prevent unread messages from being overwritten. If the CAN module needs to store a received message, it will first check mailbox 5. If the mailbox is empty, the message is stored there. If the RMP flag of object 5 is set (mailbox occupied), the CAN module will check the condition of mailbox 4. If that mailbox is also busy, the module will check in mailbox 3 and store the message there since the OPC flag is not set for mailbox 3. It also sets the RML flag of object 3, which may initiate an interrupt.

It also may be advisable to have object 4 generate an interrupt telling the CPU to read mailboxes 4 and 5 at once. This technique is also useful for messages that require more than 8 bytes of data (i.e., more than one message). In this case, all data needed for the message can be collected in the mailboxes and be read at once.

2.3 Handling of Remote Frame Mailboxes

There are two functions for remote frame handling. One is a request by the module for data from another node, the other is a request by another node for data that the module needs to answer.

2.3.1 Requesting Data From Another Node

In order to request data from another node, the object is configured as receive mailbox. Using object 3 for this example, the CPU needs to do the following:

1) Set the RTR bit in the message control field register (CANMCF) to 1.

   Write MCF(3) = 0x12

2) Write the correct identifier into the message identifier register (MID).

   Write MID(3) = 0x4f780000

3) Set the TRS flag for that mailbox. Since the mailbox is configured as receive, it will only send a remote request message to the other node.

   Set TRS.3 = 1

4) The module stores the answer in that mailbox and sets the RMP bit when it is received. This action may initiate an interrupt. Also, make sure no other mailbox has the same ID.

   Wait for RMP.3 = 1

5) Read the received message.

2.3.2 Answering a Remote Request

1) Configure the object as as a transmit mailbox.

2) Set the auto answer mode (AAM) (MID.29) bit in the MID register before the mailbox is enabled.

   MID(1) = 0x35ac0000

3) Update the data field.

   MIL(1) = xxxx0000h

4) Enable the mailbox by setting the ME flag to 1.

   ME.1 = 1

   When a remote request is received from another node, the TRS flag is set automatically and the data is transmitted to that node. The identifier of the received message and the transmitted message are the same.

After transmission of the data, the TA flag is set. The CPU may then update the data.

Wait for TA.1 = 1

2.3.3 Updating the Data Field

To update the data of an object that is configured in auto answer mode, the following steps need to be performed. This sequence can also be used to update the data of an object configured in normal transmission with TRS flag set.

1) Set the change data request (CDR) (MC.8) bit and the mailbox number (MBNR) of that object in the master control register (CANMC). This tells the CAN module that the CPU wants to change the data field. For example, for object 1:

   Write MC = 0x0000101

2) Write the message data into the mailbox data register. For example:

   Write MDL(1) = xxxx0000h

3) Clear the CDR bit (MC.8) in order to enable the object.

   Write MC = 0x00000000

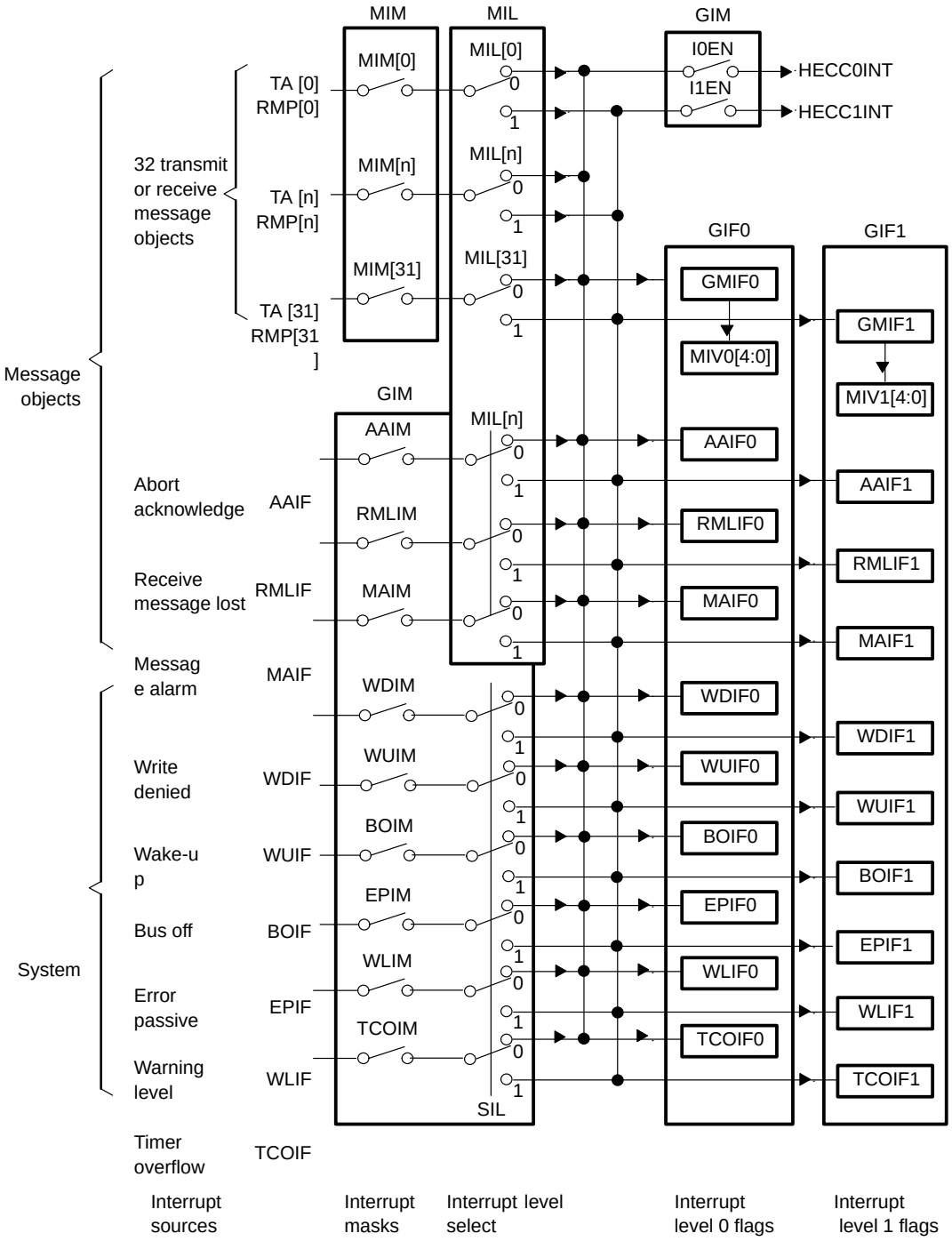
2.4 Interrupts

There are two different types of interrupts. One type of interrupt is a message-object related interrupt, for example, the receive-message-pending interrupt or the abort-acknowledge interrupt. The other type of interrupt is a system interrupt that handles errors or system-related interrupt sources, for example, the error-passive interrupt or the wake-up interrupt. See Figure 2–3.

The following events may initiate one of the two interrupts:

- Message object interrupts
  - Message reception interrupt: a message was received
  - Message transmission interrupt: a message was transmitted successfully
  - Abort-acknowledge interrupt: a sent transmission was aborted
  - Receive-message-lost interrupt: an old message was overwritten by a new one
  - Message alarm interrupt (HECC only): one of the messages was not transmitted or received within a predefined time frame
- System interrupts
  - Write-denied interrupt: the CPU tried to write to a mailbox but was not allowed to
  - Wake-up interrupt: this interrupt is generated after a wake up
  - Bus-off interrupt: the CAN module enters the bus-off state
  - Error-passive interrupt: the CAN module enters the error-passive mode
  - Warning level interrupt: one or both error counters are greater than or equal to 96
  - Time counter overflow interrupt (HECC only): the local network time stamp counter had an overflow

Figure 2–3. Interrupts Scheme



2.4.1 Interrupts Scheme

The interrupt flags are set if the corresponding interrupt condition occurred. The system interrupt flags are set depending on the setting of SIL (GIM.2). If set, the global interrupts will set the bits in the CANGIF1 register, otherwise they will set in the CANGIF0 register. This does not apply to the interrupt flags AAIF (GIF.14) and RMLIF (GIF.11). These bits are set according to the state of the appropriate MIL[*n*] bit (MIL.31-0).

The GMIF0/GMIF1(GIF0.15/GIF1.15) bit is set depending on the setting of the MIL[*n*] bit that corresponds to the message object originating that interrupt. If the MIL[*n*] bit is set, the corresponding message object interrupt flag MIF[*n*] will set the GMIF1 flag in the CANGIF1 register, otherwise, it will set the GMIF0 flag.

If all interrupt flags are cleared and a new interrupt flag is set, the CAN module interrupt output line (SCC0INT/HECC0INT or SCC1INT/HECC1INT) is activated if the corresponding interrupt mask bit is set. The interrupt line stays active until the interrupt flag is cleared by the CPU by writing a 1 to the appropriate bit.

The GMIF0 (GIF0.15) or GMIF1 (GIF0.15) bit must be cleared by writing a 1 to the appropriate bit in the CANTA register or the CANRMP register (depending on mailbox configuration) and cannot be cleared in the CANGIF0/CANGIF1 register.

After clearing one or more interrupt flags, and one or more interrupt flags are still pending, a new interrupt is generated. The interrupt flags are cleared by writing a 1 to the corresponding bit location. If the GMIF0 or GMIF1 bit is set, the mailbox interrupt vector MIV0 (GIF0.4-0) or MIV1 (GIF1.4-0) indicates the mailbox number of the mailbox that caused the setting of the GMIF0/1. It will always display the highest mailbox interrupt vector assigned to that interrupt line.

2.4.2 Message Object Interrupt

Each of the 32 message objects in the HECC or the 16 message objects in the SCC may initiate an interrupt on one of the two interrupt output lines 1 or 0. These interrupts can be receive or transmit interrupts depending on the mailbox configuration.

There is one interrupt mask bit (MIM[*n*]) and one interrupt level bit (MIL[n]) dedicated to each mailbox. To generate a mailbox interrupt upon a receive/transmit event, the MIM bit has to be set. If a CAN message is received (RMP[*n*]=1) in a receive mailbox or transmitted (TA[*n*]=1) from a transmit mailbox, an interrupt is asserted. If a mailbox is configured as remote request mailbox (MD[*n*]=1, MCF.RTR=1), an interrupt occurs upon reception of the reply frame. A remote reply mailbox generates an interrupt upon successful transmission of the reply frame (MD[*n*]=0, MID.AAM=1).

The setting of the RMP[*n*] bit or the TA[*n*] bit also sets the GMIF0/GMIF1 (GIF0.15/GIF1.15) flag in the GIF0/GIF1 register if the corresponding interrupt mask bit is set. The GMIF0/GMIF1 flag then generates an interrupt and the corresponding mailbox vector (= mailbox number) can be read from the bit field MIV0/MIV1 in the GIF0/GIF1 register. If more than one mailbox interrupts are pending, the actual value of MIV0/MIV1 reflects the highest priority interrupt vector. The interrupt generated depends on the setting in the mailbox interrupt level (MIL) register.

The abort acknowledge flag (AA[*n*]) and the abort acknowledge interrupt flag (AAIF) in the GIF0/GIF1 register are set when a transmit message is aborted by setting the TRR[*n*] bit. An interrupt is asserted upon transmission abortion if the mask bit AAIM in the GIM register is set. Clearing the AA[*n*] flag(s) does not reset the AAIF0/AAIF1 flag. The interrupt flag has to be cleared separately. The interrupt line for the abort acknowledge interrupt is selected in accordance with the MIL[*n*] bit of the concerned mailbox.

A lost receive message is notified by setting the receive message lost flag RML[*n*] and the receive message lost interrupt flag RMLIF0/RMLIF1in the GIF0/GIF1 register. If an interrupt shall be generated upon the lost receive message event, the receive message lost interrupt mask bit (RMLIM) in the GIM register has to be set. Clearing the RML[*n*] flag does not reset the RMLIF0/RMLIF1 flag. The interrupt flag has to be cleared separately. The interrupt line for the receive message lost interrupt is selected in accordance with the mailbox interrupt level (MIL[*n*]) of the concerned mailbox.

Each mailbox of the HECC (in HECC mode only) is linked to a message-object, time-out register (MOTO). If a time-out event occurs (TOS[*n*]=1), a message alarm interrupt is asserted to one of the two interrupt lines if the message alarm interrupt mask bit (MAIM) in the GIM register is set. The interrupt line for message alarm interrupt is selected in accordance with the mailbox interrupt level (MIL[*n*]) of the concerned mailbox. Clearing the TOS[*n*] flag does not reset the MAIF0/MAIF1 flag.

2.4.3 Interrupt Handling

The CPU is interrupted by asserting one of the two interrupt lines. After handling the interrupt, which should generally also clear the interrupt source, the interrupt flag must be cleared by the CPU. To do this, the interrupt flag must be cleared in the CANGIF0 or CANGIF1 register by writing a 1 to the interrupt flag. This also releases the interrupt line if no other interrupt is pending.

2.4.4 Configuring for Interrupt Handling

To configure for interrupt handling, the mailbox interrupt level register (CANMIL), the mailbox interrupt mask register (CANMIM), and the global interrupt mask register (CANGIM) need to be configured. The steps to do this are described below:

1) Write the CANMIL register. This defines whether a successful transmission will assert interrupt line 0 or 1. For example, CANMIL = 0xFFFFFFFF will set all mailbox interrupts to level 1.

2) Configure the mailbox interrupt mask register (CANMIM) to mask out the mailboxes that should not cause an interrupt. This register could be set to 0xFFFFFFFF, which enables all mailbox interrupts. Mailboxes that are not used will not cause any interrupts anyhow.

3) Now configure the CANGIM register. The flags AAIM, WDIM, WUIM, BOIM, EPIM, and WLIM (GIM.14-9) should always be set (enabling these interrupts). In addition, the SIL (GIM.2) bit may be set to have the global interrupts on another level than the mailbox interrupts. Both the I1EN (GIM.1) and I0EN (GIM.0) flags should be set to enable both interrupt lines. The flag RMLIM (GIM.11) may also be set depending on the load of the CPU.

This configuration puts all mailbox interrupts on line 1 and all system interrupts on line 0. Thus, the CPU can handle all system interrupts (which are always serious) with high priority, and the mailbox interrupts (on the other line) with a lower priority. All messages with a high priority can also be directed to the interrupt line 0.

2.4.5 Handling Mailbox Interrupts

There are three interrupt flags for mailbox interrupts. These are listed below:

GMIF0/GMIF1: One of the objects has received or transmitted a message. The number of the mailbox is in MIV0/MIV1(GIF0.4-0/GIF1.4-0). The normal handling routine is as follows:

1) Do a half-word read on the GIF register that caused the interrupt. If the value is negative, a mailbox caused the interrupt. Otherwise, check the

AAIF0/AAIF1 (GIF0.14/GIF1.14) bit (abort-acknowledge interrupt flag) or the RMLIF0/RMLIF1 (GIF0.11/GIF1.11) bit (receive-message-lost interrupt flag). Otherwise, a system interrupt has occurred. In this case, each of the system-interrupt flags must be checked.

2) If the RMLIF (GIF0.11) flag caused the interrupt, the message in one of the mailboxes has been overwritten by a new one. This should not happen in normal operation. The CPU needs to clear that flag by writing a 1 to it. The CPU must check the receive-message-lost register (RML) to find out which mailbox caused that interrupt. Depending on the application, the CPU has to decide what to do next. This interrupt comes together with an GMIF0/GMIF1 interrupt.

3) If the AAIF (GIF.14) flag caused the interrupt, a send transmission operation was aborted by the CPU. The CPU should check the abort acknowledge register (AA.31-0) to find out which mailbox caused the interrupt and send that message again if requested. The flag must be cleared by writing a 1 to it.

4) If the GMIF0/GMIF1 (GIF0.15/GIF1.15) flag caused the interrupt, the mailbox number that caused the interrupt can be read from the MIV0/MIV1 (GIF0.4-0/GIF1.4-0) field. This vector may be used to jump to a location where that mailbox is handled. If it is a receive mailbox, the CPU should read the data as described above and clear the RMP.31-0 flag by writing a 1 to it. If it is a send mailbox, no further action is required, unless the CPU needs to send more data. In this case, the normal send procedure as described above is necessary. The CPU needs to clear the transmit acknowledge bit (TA.31-0) by writing a 1 to it.

Chapter 3

eCAN Registers

This chapter contains the registers and bit descriptions.

Topic **Page**

3.1	Mailbox Registers	3-2
3.2	Acceptance Filter	3-15
3.3	Master Control Register (CANMC)	3-19
3.4	Bit-Timing Configuration Register (CANBTC)	3-23
3.5	Error Registers	3-28
3.6	Interrupt Registers	3-33
3.7	eCAN I/O Control Registers (CANTIOC, CANRIOC)	3-42

3.1 Mailbox Registers

Figure 3–1. CAN Local Network Time Register Bits

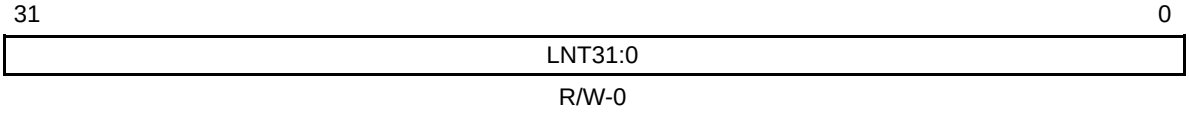
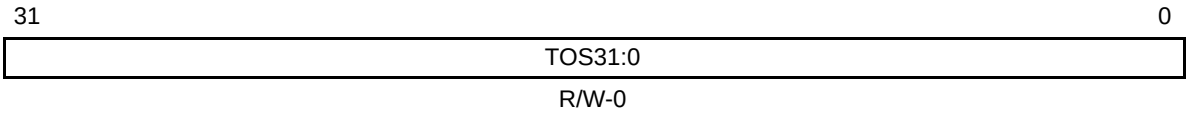


Figure 3–2. TOS Register



Bit(s)	Name	Description
31:0	TOS31:0	The TOSn bit is set when all three of the following conditions are met: 1) The TSC value is greater than or equal to the value in the time-out register. 2) The TOCn bit is set. 3) The TRSn bit is set. The time-out registers are implemented as a RAM. The state machine scans all the time-out Registers and compares them to the time stamp counter value. Since all the Time Out Registers are scanned sequentially, it is possible that even though a transmit mailbox has timed out, the TOSn bit is not set. This can happen when the mailbox succeeded in transmitting and clearing the TRSn bit before the state machine scans the time-out register of that mailbox. This is true for the receive mailbox as well. In this case, the RMPn bit may be set to 1 by the time the state machine scans the time-out register of that mailbox. However, the receive mailbox probably did not receive the message before the time specified in the time-out register.

Figure 3–3. Mailbox Enable Register (CANME)

31	0
ME31:0	
R/W-0	

Legend: R/W = Read/Write, -*n* = Value after reset

Bit(s)	Name	Description
31:0	ME31:0	Mailbox enable. After power-up, all bits in CANME are cleared. Disabled mailboxes may be used as additional memory for the CPU.
		1 The corresponding mailbox is enabled for the CAN module. The mailbox must be disabled before writing to the contents of any identifier field. If the corresponding bit in CANME is set, the write access to the identifier of a message object is discarded.
		0 The corresponding mailbox is disabled.

Figure 3–4. Mailbox Direction Register (CANMD)

31	0
MD31:0	
R/W-0	

Note: HECC only, reserved in the SCC

Bit(s)	Name	Description
31:0	MD31:0	Mailbox direction register. After power-up, all bits are cleared.
		1 The corresponding mailbox is defined as a receive mailbox.
		0 The corresponding mailbox is defined as a transmit mailbox.

3.1.1 Transmission Request Set Register (CANTRS)

When mailbox *n* is ready to be transmitted, the CPU sets the TRS[*n*] bit to 1 to start the transmission.

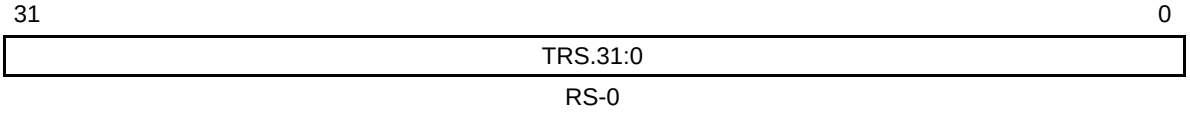
These bits can be set by the CPU and reset by the CAN module logic. The CAN module sets a remote frame request. These bits are reset when a transmission is successful or aborted. If a mailbox is configured as a receive mailbox, the corresponding bit in CANTRS is ignored. If the TRS[n] bit of a remote request mailbox is set, a remote frame is transmitted. If the CPU tries to set a bit while the eCAN module tries to clear it, the bit is set.

Setting CANTRS[*n*] causes the particular message *n* to be transmitted. Several bits can be set simultaneously. Therefore, all messages with the TRS

bit set are transmitted in turn, starting with the mailbox having the highest mailbox number (= highest priority).

The bits in CANTRS are set by writing a 1 from the CPU. Writing a 0 has no effect. After power up, all bits are cleared.

Figure 3–5. Transmit Request Set Register



Legend: RS = Read/Set, -*n* = Value after reset

Bit(s)	Name	Description
31:0	TRS.31:0	Transmit request set register. 1 Setting TRSn transmits the message in that mailbox. Several bits can be set simultaneously with all messages transmitted in turn. 0 No operation

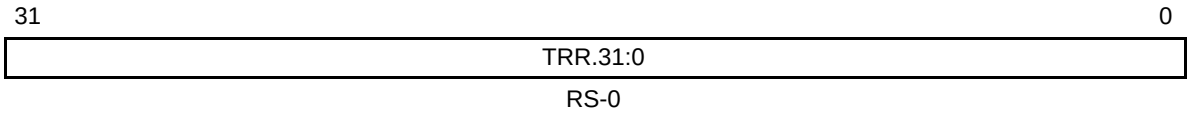
3.1.2 Transmission Request Reset Register (CANTRR)

These bits can only be set by the CPU and reset by the internal logic. These bits are reset when a transmission is successful or is aborted. If the CPU tries to set a bit while the CAN tries to clear it, the bit is set.

Setting the TRR[*n*] bit of the message object *n* cancels a transmission request if it was initiated by the corresponding bit (TRS[*n*]) and is not currently being processed. If the corresponding message is currently being processed, the bit is reset when a transmission is successful (normal operation) or when an aborted transmission due to a lost arbitration or an error condition is detected on the CAN bus line. When a transmission is aborted, the corresponding status bit (AA.31-0) is set. When a transmission is successful, the status bit (TA.31-0) is set. The status of the transmission request reset can be read from the TRS.31-0 bit.

The bits in CANTRR are set by writing a 1 from the CPU.

Figure 3–6. Transmit Request Reset Register Bits



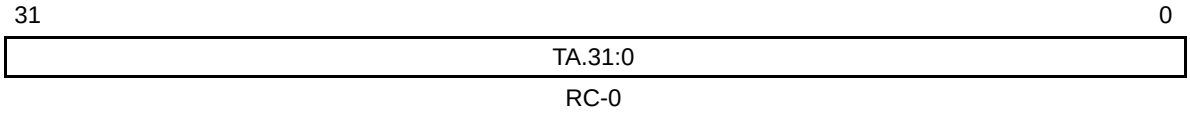
Legend: RS = Read/Set, *-n* = Value after reset

Bit(s)	Name	Description
31:0	TRR 31:0	Transmit request reset register
		1 Setting TRRn cancels a transmission request
		0 No operation

3.1.3 Transmission Acknowledge Register (CANTA)

If the message of mailbox *n* was sent successfully, the bit TA[*n*] is set. This also sets the GMIF0/GMIF1 (GIF0.15/GIF1.15) bit if the corresponding interrupt mask bit in the CANMIM register is set. The GMIF0/GMIF1 bit initiates an interrupt.

The CPU resets the bits in CANTA by writing a 1. This also clears the interrupt if an interrupt has been generated. Writing a 0 has no effect. If the CPU tries to reset the bit while the CAN tries to set it, the bit is set. After power-up, all bits are cleared.



Legend: RS = Read/Clear, *-n* = Value after reset

Bit(s)	Name	Description
31:0	TA.31:0	Transmit acknowledge register
		1 If the message of mailbox *n* is sent successfully, the bit n of this register is set.
		0 The message is not sent.

3.1.4 Abort Acknowledge Register (CANAA)

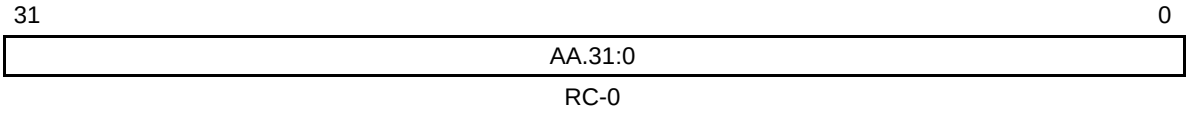
If the transmission of the message in mailbox *n* was aborted, the bit AA[*n*] is set and the AAIF (GIF.14) bit is set, which may generate an interrupt if enabled.

Note: Additional conditions that set the AA[*n*] flag

The AA[*n*] flag is set if a transmission reset is requested and the TRS[*n*] bit of the corresponding transmit mailbox is not set, a receive mailbox or a disabled mailbox is concerned.

The bits in CANAA are reset by writing a 1 from the CPU. Writing a 0 has no effect. If the CPU tries to reset a bit and the CAN tries to set the bit at the same time, the bit is set. After power-up all bits are cleared.

Figure 3–7. Abort Acknowledge Register Bits



Legend: RS = Read/Clear, -*n* = Value after reset

Bit(s)	Name	Description
31:0	AA.31:0	Abort acknowledge register
		1 If the transmission of the message in mailbox n is aborted, the bit n of this register is set.
		0 The transmission is not aborted.

3.1.5 Message Data Registers (MDL, MDH)

Eight bytes of the mailbox are used to store the data field of a CAN message. The setting of DBO (MC.10) determines the ordering of stored data.

The data is transmitted or received from the CAN bus, starting with byte 0.

- When DBO (MC.10) = 1, the data is stored or read starting with the least significant byte of the CANMDL register and ending with the most significant byte of the CANMDH register.
- When DBO (MC.10) = 0, the data is stored or read starting with the most significant byte of the CANMDL register and ending with the least significant byte of the CANMDH register.

The registers MDL(*n*) and MDH(*n*) can be written only if mailbox *n* is configured for transmission (MD[*n*] (MD.31-0)=0) or the mailbox is disabled (ME[*n*] (ME.31-0)=0). If TRS[*n*] (TRS.31-0)=1, the registers MDL(*n*) and MDH(*n*) cannot be written, unless CDR (MC.8)=1, with MBNR (MC.4-0) set to

n. These settings also apply for a message object configured in reply mode (AAM (MID.29)=1).

Figure 3–8. Message Data Low Register with DBO = 0 (MDL)

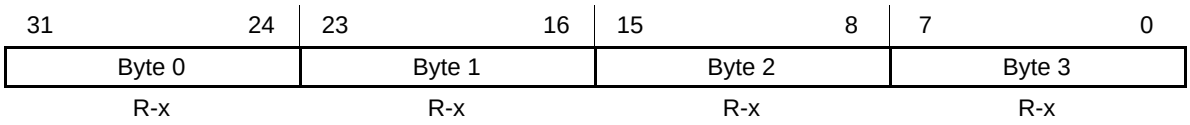


Figure 3–9. Message Data High Register with DBO = 0 (MDH)

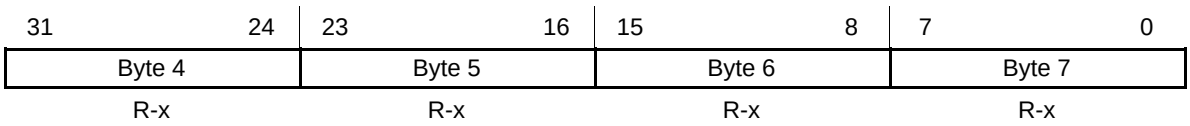


Figure 3–10. Message Data Low Register with DBO = 1 (MDL)

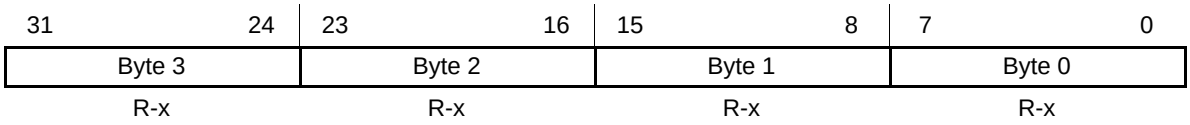
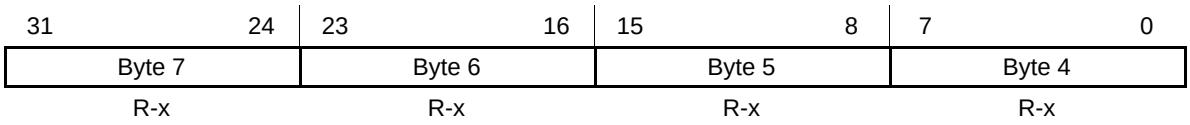


Figure 3–11. Message Data High Register with DBO = 1 (MDH)



Note: Data Field

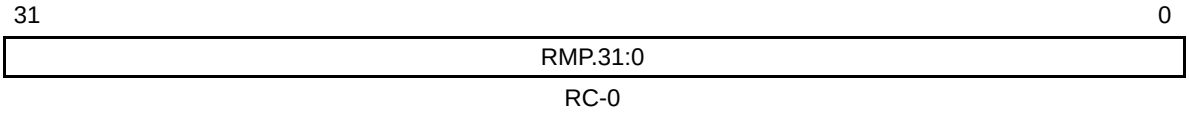
The data field beyond the valid received data is modified by any message reception and is indeterminate.

3.1.6 Receive Message Pending Register (CANRMP)

If mailbox *n* contains a received message, the bit RMP[*n*] of this register is set. These bits can be reset only by the CPU and set by the internal logic. A new incoming message overwrites the stored one if the OPC[*n*](OPC.31-0) bit is cleared, otherwise the next mailboxes are checked for a matching ID. In this case, the corresponding status bit RML[*n*] is set. The bits in the CANRMP and the CANRML registers are cleared by a write access to the base address of the register CANRMP, with a 1 at the corresponding bit location. If the CPU tries to reset a bit and the CAN tries to set the bit at the same time, the bit is set.

The bits in the CANRMP register may set GMIF0/GMIF1 (GIF0.15/GIF1.15) if the corresponding interrupt mask bit in the CANMIM register is set. The GMIF0/GMIF1 bit initiates an interrupt.

Figure 3–12. Receive Message Pending Register Bits



Legend: RS = Read/Clear, -*n* = Value after reset

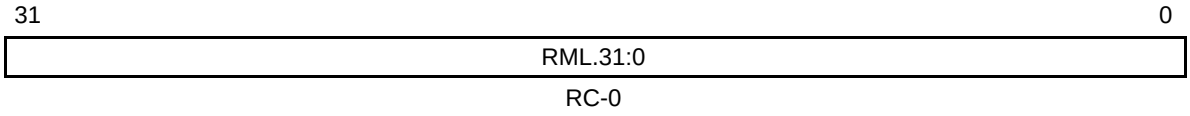
Bit(s)	Name	Description
31:0	RMP.31:0	Receive message pending register
		1 If mailbox *n* contains a received message, bit RMP[*n*] of this register is set.
		0 The mailbox does not contain a message.

3.1.7 Receive Message Lost Register (CANRML)

This register is set if an old message has been overwritten by a new one in message object *n*, bit RML[*n*]. These bits can only be reset by the CPU, and set by the internal logic. The bits can be cleared by a write access to the base address of the register CANRMP with a 1 at the corresponding bit location. If the CPU tries to reset a bit and the CAN tries to set the bit at the same time, the bit is set. The CANRML register is not changed if the OPC[*n*] (OPC.31-0) bit is set.

If one or more of the bits in the CANRML register are set, the RMLIF (GIF0.11/ GIF1.11) bit is also set. This may initiate an interrupt if the RMLIM (GIM.11) bit is set.

Figure 3–13. Receive Message Lost Register Bits



Legend: RS = Read/Clear, *-n* = Value after reset

Bit(s)	Name	Description
31:0	RML.31:0	Receive message lost register
		1 An old unread message has been overwritten by a new one in that mailbox.
		0 No message was lost.

3.1.8 Handling of Remote Frames

If a remote frame is received (the incoming message has RTR (MCF.4) = 1), the CAN module compares the identifier to all identifiers of the mailboxes using the appropriate masks starting at the highest mailbox number in descending order.

In the case of a matching identifier (with the message object configured as send mailbox and AAM (MID.29) in this message object set) this message object is marked as to be sent (TRS[n] is set).

In case of a matching identifier with the mailbox configured as a send mailbox and bit AAM in this mailbox is not set, this message is not received in that mailbox.

After finding a matching identifier in a send mailbox no further compare is done.

In the case of a matching identifier and the message object configured as receive mailbox, this message is handled like a data frame and the

corresponding bit in the receive message pending (CANRMP) register is set. The CPU then has to decide how to handle this situation. See Section 3.1.6 on page 3-8 for information about the CANRMP register.

For the CPU to change the data in a mailbox that is configured as a remote frame mailbox“ (AAM set) it has to set the mailbox number and the change data request bit (CDR [MC.8]) in the MCR first. The CPU may then do the access and clear the CDR bit to tell the eCAN that the access is finished. Until the CDR bit is cleared the transmission of this mailbox is not permitted. Thus the newest data will be sent.

To change the identifier in that mailbox, the mailbox must be disabled first (MEn = 0).

For the CPU to request data from another node it configures the mailbox as a receive mailbox and sets the TRS bit. In this case the module sends a remote frame request and receives the data frame in the same mailbox that sent the request. Therefore only one mailbox is necessary to do a remote request. Note that the CPU must set RTR (MCF.4) to enable a remote frame transmission. Once the remote frame is sent, the TRS bit of the mailbox is cleared by CAN.

The behavior of the message object *n* is configured with MD[*n*] (MD.31-0), the AAM (MID.29), and RTR (MCF.4). It shows how to configure a message object according to the desired behavior.

To summarize, a message object can be configured with four different behaviors:

1) A transmit message object is only able to transmit messages.
2) A receive message object is only able to receive messages.
3) A request message object is able to transmit a remote request frame and to wait for the corresponding data frame.
4) A reply message object is able to transmit a data frame whenever a remote request frame is received for the corresponding identifier.

Note: Remote Transmission Request Bit

When a remote transmission request is successfully transmitted with a message object configured in request mode, the CANTA register is not set and no interrupt is generated. When the remote reply message is received, the behavior of the message object is the same as a message object configured in receive mode.

3.1.9 CPU Message Mailbox Access

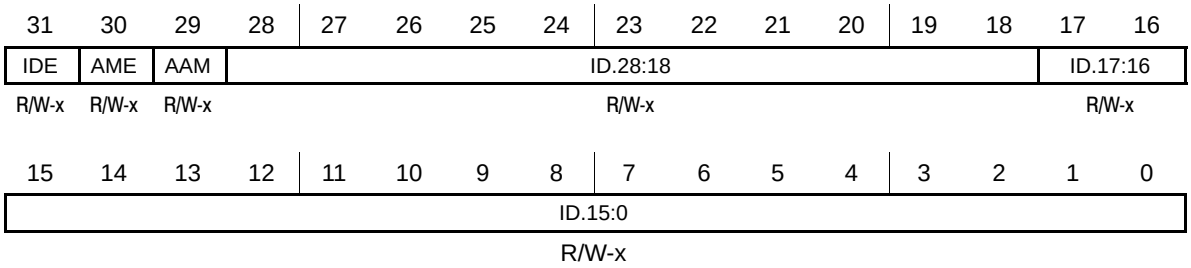
Write accesses to the identifier can only be accomplished when the mailbox is disabled (ME[*n*] (ME.31-0) = 0). During access to the data field, it is critical that

the data does not change while the CAN module is reading it. Hence, a write access to the data field is disabled for a receive mailbox.

For send mailboxes, an access is usually denied if the TRS (TRS.31-0) or the TRR (TRR.31-0) flag is set. In these cases, an interrupt may be asserted. A way to access those mailboxes is to set CDR (MC.8) before accessing the mailbox data.

After the CPU access is finished, the CPU must clear the CDR flag by writing a ‘0 to it. The CAN module checks for that flag before and after reading the mailbox. If the CDR flag is set during those checks, the CAN module does not transmit the message but continues to look for other transmit requests. The setting of the CDR flag also stops the write-denied interrupt (WDI) from being asserted.

Figure 3–14. Message Identifier Register (MID)



Legend: R = Read, W = Write when mailbox is disabled, -*n* = Value after reset, x = indeterminate

Note: This register can be written only when mailbox *n* is disabled (ME[*n*] (ME.31-0) = 0).

Bit(s)	Name	Description
31	IDE	Identifier extension bit. When AMI = 1, the IDE bit of the receive mailbox is a “don’t care” and is overwritten by the IDE bit of the trnasmitted message. The filtering criterion must be satisfied to receive a message. The number of bts to be compared is a function of the vale of the IDE bit of the transmitted message. 1 Extended identifier (29 bits). The received message or the message to be sent has an extended identifier. 0 Standard identifier (11 bits). The received message or the message to be sent has a standard identifier.
30	AME	Acceptance mask enable bit. AME is only used for receiver mailboxes. It must not be set for automatic reply (AAM[*n*]=1, MD[*n*]=0) mailboxes, otherwise the mailbox behavior is undefined. This bit is not modified by a message reception. 1 The corresponding acceptance mask is used. 0 No acceptance mask will be used, all identifier bits must match to receive the message

Bit(s)	Name	Description
29	AAM	Auto answer mode bit. This bit is only valid for message mailboxes configured as transmit. For receive mailboxes, this bit has no effect: the mailbox is always configured for normal receive operation. This bit is not modified by a message reception. 1 Auto answer mode. If a matching remote request is received, the CAN module answers to the remote request by sending the contents of the mailbox. 0 Normal transmit mode. The mailbox does not reply to remote requests. The reception of a remote request frame has no effect on the message mailbox.
28	ID 28:0	Message identifier 1 In standard identifier mode, if the IDE bit (MID.31 = 0), the message identifier is stored in bits ID.28:18. In this case, bits ID.17:0 have no meaning. 0 In extended identifier mode, if the IDE bit (MID.31 = 1), the message identifier is stored in bits ID.28:0.

Figure 3–15. Message Control Field Register (MCF)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			TPL					Reserved			RTR	DLC			
R-0			RW-x					R-0			RW-x	RW-x			

Legend: RW = Read any time, write when mailbox is disabled or configured for transmission. -*n* = Value after reset, x = indeterminate

Note: The register MCF(*n*) can only be written if mailbox *n* is configured for transmission (MD[*n*] (MD.31-0)=0) or if the mailbox is disabled (ME[*n*] (ME.31-0) =0).

Bit(s)	Name	Description
31:13	Reserved	
12:8	TPL.4:0	Transmit priority level. This 5-bit field defines the priority of this mailbox as compared to the other 31 mailboxes. The highest number has the highest priority. When two mailboxes have the same priority, the one with the higher mailbox number is transmitted. TPL applies only for transmit mailboxes. TPL is not used when in SCC-compatibility mode.
7:5	Reserved	

Bit(s)	Name	Description
4	RTR	Remote transmission request bit 1 For receive mailbox: If the TRS flag is set, a remote frame is transmitted and the corresponding data frame will be received in the same mailbox. Once the remote frame is sent, the TRS bit of the mailbox is cleared by CAN. For transmit mailbox: If the TRS flag is set, a remote frame is transmitted, but the corresponding data frame has to be received in another mailbox. 0 No remote frame is requested.
3:0	DLC 3:0	Data length code. The number in these bits determines how many data bytes are sent or received. Valid value range is from 0 to 8. Values from 9 to 15 are not allowed.

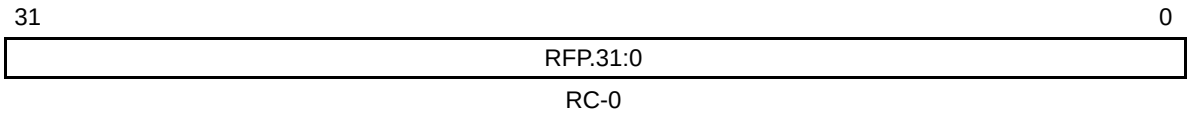
3.1.10 Remote Frame Pending Register (CANRFP)

Whenever a remote frame request is received by the CAN module, the corresponding bit RFP[n] in the remote frame pending register is set. If a remote frame is stored in a receive mailbox (AAM=0, MD=1), the RFPn bit will not be set.

To prevent an auto-answer mailbox from replying to a remote frame request, the CPU has to clear the RFP[n] flag and the TRS[*n*] bit by setting the corresponding transmission request reset bit TRR[*n*]. The AAM bit may also be cleared by the CPU to stop the module from sending the message.

If the CPU tries to reset a bit and the CAN module tries to set the bit at the same time, the bit is not set. The CPU cannot interrupt an ongoing transfer.

Figure 3–16. Remote Frame Pending Register Bits



Legend: RS = Read/Clear, *-n* = Value after reset

Bit(s)	Name	Description
31:0	RFP.31:0	Remote frame pending register. For a receive mailbox, RFPn will be set if a remote frame is received and TRSn will not be affected. For a transmit mailbox, RFPn will be set if a remote frame is received and TRSn will be set if AAM of the mailbox is 1. The ID of the mailbox must match the remote frame ID. 1 A remote frame request was received by the module. 0 No remote frame request was received. The register is cleared by the CPU.

3.2 Acceptance Filter

The identifier of the incoming message is first compared to the message identifier of the mailbox (which is stored in the mailbox). Then, the appropriate acceptance mask is used to mask out the bits of the identifier that should not be compared.

In the SCC (or in the HECC in SCC-compatible mode), the global acceptance mask (GAM) is used for the mailboxes 6 to 15. An incoming message is stored in the highest numbered mailbox with a matching identifier. If there is no matching identifier in message objects 15 to 6, the incoming message is compared to the identifier stored in message objects 5 to 3 and then 2 to 0.

The message objects 5 to 3 use the local acceptance mask LAM(3) of the SCC registers. The message objects 2 to 0 use the local acceptance mask LAM(0) of the SCC registers. See Section 3.5.4.3, *"Local Acceptance Mask Register (LAM)"* , on page 3-31 for specifics uses.

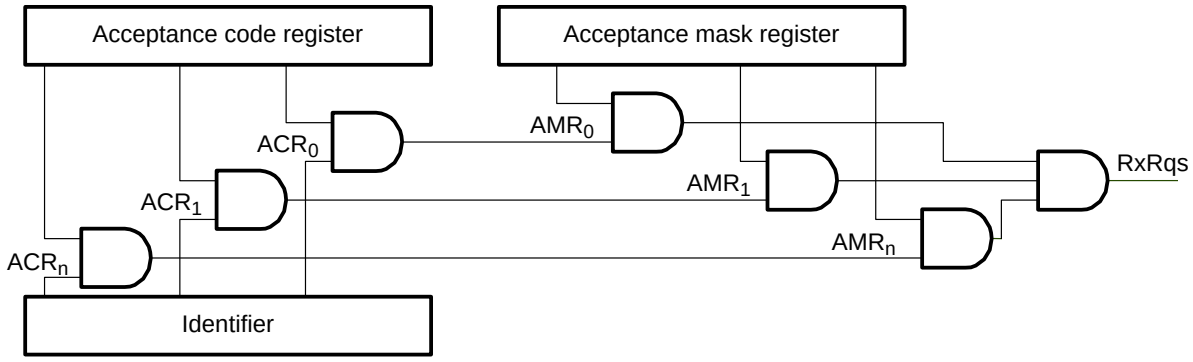
To modify the global acceptance mask register (CANGAM) and the two local-acceptance-mask registers of the SCC, the CAN module must be set in the initialization mode. See Section 3.6, *"CAN Module Initialization",* on page 3-33.

As can be seen in Figure 3–17, the identifier of the incoming message is first compared to the Acceptance Code Register (which is stored in the mailbox) and then the appropriate acceptance mask is used to mask out those bits of the identifier that should not be compared. The acceptance code consists of the contents of the identifier words of the current mailbox.

Each of the 32 mailboxes of the HECC has its own local acceptance mask LAM(0) to LAM(31). There is no global acceptance mask in the HECC.

The selection of the mask be used for the comparison depends on the module.

Figure 3–17. Acceptance Filter



3.2.1 Local Acceptance Masks (CANLAM)

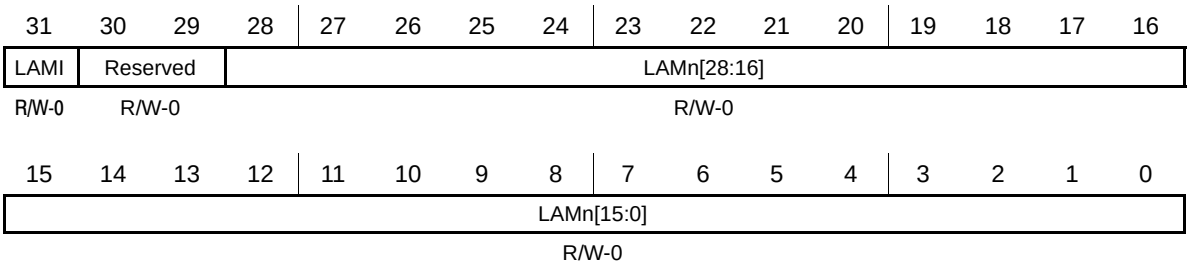
The local acceptance filtering allows the user to locally mask (don't care) any identifier bits of the incoming message.

In the SCC, the local-acceptance-mask register LAM(0) is used for mailboxes 2 to 0. The local-acceptance-mask register LAM(3) is used for mailboxes 5 to 3. For the mailboxes 6 to 15, the global-acceptance-mask (CANGAM) register is used. The LAM(0) register is located at address 0x80 of the SCC register memory. The LAM(3) register is located at address 0x8C of SCC register memory.

After a hardware or a software reset of the SCC module, the LAM(0) and the LAM(3) registers are reset to zero. After a reset of the HECC, the LAM registers are not modified.

In the HECC, each mailbox (0 to 31) has its own mask register, LAM(0) to LAM(31). An incoming message is stored in the highest numbered mailbox with a matching identifier.

Figure 3–18. Local acceptance mask register (LAMn)



Bit(s)	Name	Description
31	LAMI	Local acceptance mask identifier extension bit 1 Standard and extended frames can be received. In case of an extended frame, all 29 bits of the identifier are stored in the mailbox and all 29 bits of the local acceptance mask register are used for the filter. In case of a standard frame, only the first eleven bits (bits 28 to 18) of the identifier and the local acceptance mask are used. 0 The identifier extension bit stored in the mailbox determines which messages shall be received.
30:29	Reserved	Reads are undefined and writes have no effect.
28:0	LAM.28:0	These bits enable the masking of any identifier bit of an incoming message. 1 Accept a 0 or a 1 (don’t care) for the corresponding bit of the received identifier. 0 Received identifier bit value must match the corresponding identifier bit of the MID register.

You can locally mask any identifier bits of the incoming message. A 1 value means “don’t care“ or accept either a 0 or 1 for that bit position. A 0 value

means that the incoming bit value must match identically to the corresponding bit in the message identifier.

If the local acceptance mask identifier extension bit is set (LAMI = 1 => don't care) standard and extended frames can be received. An extended frame uses all 29 bits of the identifier stored in the mailbox and all 29 bits of local acceptance mask register for the filter. For a standard frame only the first eleven bits (bit 28 to 18) of the identifier and the local acceptance mask are used.

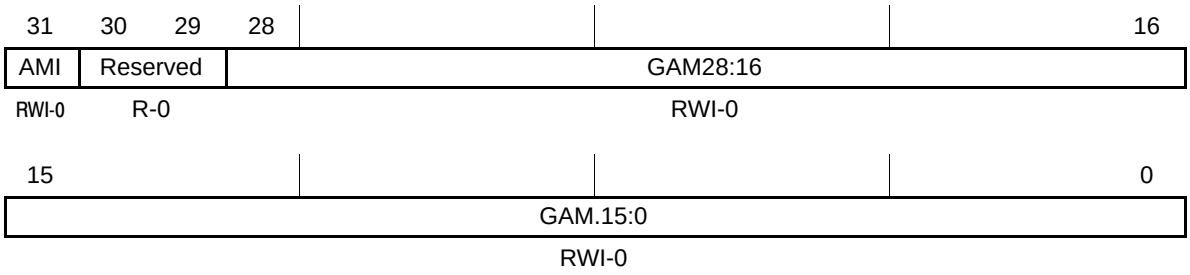
If the local acceptance mask identifier extension bit is reset (LAMI = 0), the identifier extension bit stored in the mailbox determines the messages that are received.

3.2.2 Global Acceptance Mask Register (CANGAM)

The global acceptance mask is used by the SCC or by the HECC in SCC-compatible mode. The global acceptance mask is used for the mailboxes 6 to 15 if the AME bit (MID.30) of the corresponding mailbox is set. A received message will only be stored in the first mailbox with a matching identifier.

The global acceptance mask is used for the mailboxes 6 to 15 of the SCC.

Figure 3–19. CANGAM Register Bits (0x24h)



Legend: RWI = Read at any time, write during initialization mode only, *-n* = Value after reset

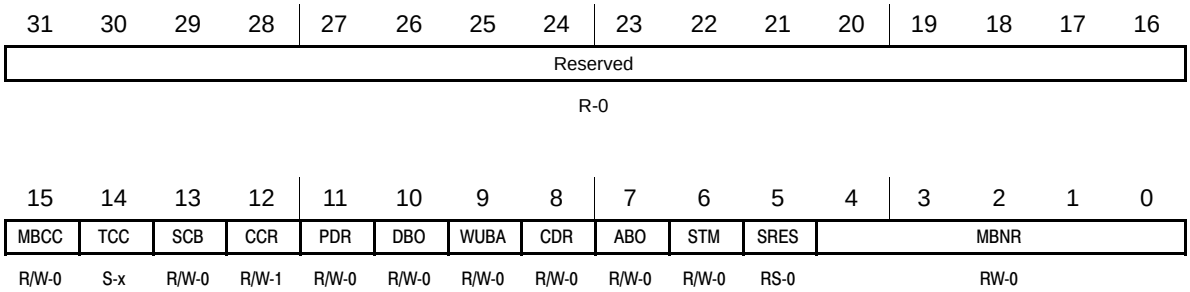
Bit(s)	Name	Description
31	AMI	Acceptance mask indentifier extension bit
		1 Standard and extended frames can be received. In case of an extended frame, all 29 bits of the identifier are stored in the mailbox and all 29 bits of global acceptance mask register are used for the filter. In case of a standard frame, only the first eleven bits (bit 28 to 18) of the identifier and the global acceptance mask are used. The IDE bit of the receive mailbox is a "don't care" and is overwritten by the IDE bit of the transmitted message. The filtering criterion must be satisfied in order to receive a message. The number of bits to be compared is a function of the value of the IDE bit of the transmitted message.

Bit(s)	Name	Description
		0 The identifier extension bit stored in the mailbox determines which messages shall be received. The IDE bit of the receive mailbox determines the number of bits to be compared. Filtering is not applicable. The MSGIDs must match bit-for-bit in order to receive a message.
30:29	Reserved	Reads are undefined and writes have no effect.
28:0	GAM 28:0	Global acceptance mask. These bits allow any identifier bits of an incoming message to be masked. Accept a 0 or a 1 (don't care) for the corresponding bit of the received identifier.Received identifier bit value must match the corresponding identifier bit of the MID register.

3.3 Master Control Register (CANMC)

This register is used to control the settings of the CAN module. Some bits of the CANMC register cannot be changed in user mode. For read/write operations , only 32-bit access is supported.

Figure 3–20. CANMC Register Bits



Legend: R = Read, WP = Write in privilege mode only, S = Set in privilege mode only, -n = Value after reset, x = Indeterminate

Note: HECC only, reserved in the SCC

Bit(s)	Name	Description
31:16	Reserved	Reads are undefined and writes have no effect.
15	MBCC	Mailbox timestamp counter clear bit. This bit is reserved in SCC mode and it is protected in user mode. 1 The time stamp counter is reset to 0 after a successful transmission or reception of mailbox 16. 0 The time stamp counter is not reset.
14	TCC	Time stamp counter MSB clear bit. this bit is reserved in SCC mode and it is protected in user mode. 1 The MSB of the time stamp counter is reset to 0. The TCC bit is reset after one clock cycle by the internal logic. 0 The time stamp counter is not changed.
13	SCB	SCC compatibility bit. This bit is reserved in SCC mode and it is protected in user mode. 1 Select HECC mode. 0 The HECC is in SCC compatibility mode. In this mode, it behaves lie the SCC so all the software developed for the SCC runs without changes. Only mailboxes 15 to 0 can be used.

Bit(s)	Name	Description
12	CCR	Change configuration request. This bit is protected in user mode. 1 The CPU requests write access to the configuration register CANBTC and the acceptance mask registers (CANGAM, LAM[0], and LAM[3]) of the SCC. After setting this bit, the CPU must wait until the CCE flag of CANES register is at 1 before proceeding to the CANBTC register. 0 The CPU requests normal operation. This can be done only after the configuration register CANBTC was set to the allowed values.
11	PDR	Power down mode request. This bit is automatically cleared by the eCAN module upon wakeup from low-power mode. This bit is protected in user mode. 1 The local power-down mode is requested. 0 The local power-down mode is not requested (normal operation).
10	DBO	Data byte order. This bit selects the byte order of the message data field. This bit is protected in user mode. 1 The data is received or transmitted least significant byte first. 0 The data is received or transmitted most significant byte first.
9	WUBA	Wake up on bus activity. This bit is protected in user mode. 1 The module leaves the power-down mode after detecting any bus activity. 0 The module leaves the power-down mode only after writing a 0 to the PDR bit.
8	CDR	Change data field request. This bit allows fast data message update. 1 The CPU requests write access to the data field of the mailbox specified by the MBNR.4:0 field (MC.4-0). The CPU must clear the CDR bit after accessing the mailbox. The module does not transmit that mailbox content while the CDR is set. This is checked by the state machine before and after it reads the data from the mailbox to store it in the transmit buffer. **Note:** Once the TRS bit is set for a mailbox and then data is changed in the mailbox using the CDR bit, the CAN module fails to transmit the new data and transmits the old data instead. To avoid this, reset transmission in that mailbox using the TRRn bit and set the TRSn bit again. The new data will then be transmitted. 0 The CPU requests normal operation.
7	ABO	Auto bus on. This bit is protected in user mode. 1 After bus-off state, the module will go back automatically into bus-on state after 128 ∗ 11 recessive bits have been received.

Bit(s)	Name	Description
		0 Bus-off state is not exited after 128*11 recessive bits.
6	STM	Self test mode. This bit is protected in user mode.
		1 The module is in self-test mode. In this mode, the CAN module generates its own acknowledge (ACK) signal, thus enabling operation without a bus connected to the module. The message is not sent, but read back and stored in the appropriate mailbox.
		0 The module is in normal mode.
5	SRES	This bit can only be written and is always read as zero.
		1 A write access to this register causes a software reset of the module (all parameters, except the protected registers, are reset to their default values). The mailbox contents and the error counters are not modified. Pending and ongoing transmissions are canceled without perturbing the communication.
		0 No effect
4:0	MBNR 4:0	Mailbox number
		1 The bit MBNR.4 is for HECC only, and is reserved in the SCC.
		0 Number of mailbox, for which the CPU requests a write access to the data field. This field is used in conjunction of the CDR bit.

3.3.1 CAN Module Action in SOFT Mode

1) If there is no traffic on the CAN bus and SUSPEND mode is requested, the node goes into SUSPEND mode.
2) If there is traffic on the CAN bus and SUSPEND mode is requested, the node goes into SUSPEND mode when the ongoing frame is over.
3) If the node was transmitting, when SUSPEND is requested, it will go to SUSPEND state after it gets the acknowledgement. If it does not get an acknowledgement or if there are some other errors, it will transmit an error frame and then go to SUSPEND state. The TEC is modified accordingly. In the second case i.e. it is suspended after transmitting an error frame, the node will re–transmit the original frame after coming out of suspended state. The TEC will be modified after transmission of the frame accordingly.
4) If the node was receiving, when SUSPEND is requested, it will go to SUSPEND state after transmitting the acknowledgement bit. If there is any error, the node will send an error frame and go to SUSPEND state. The REC will be modified accordingly before going to SUSPEND state.

5) If there is no traffic on the CAN bus and SUSPEND removal is requested, the node comes out of SUSPEND state.

6) If there is traffic on the CAN bus and SUSPEND removal is requested, the node comes out after the bus goes to idle. Therefore, a node does not receive any “partial” frame, which could lead to generation of error frames.

7) When the node is suspended, it will not participate in transmitting or receiving any data. Thus neither acknowledgement bit nor any error frame is sent. TEC and REC are not modified during SUSPEND state.

3.4 Bit-Timing Configuration Register (CANBTC)

The CANBTC register is used to configure the CAN node with the appropriate network-timing parameters. This register must be programmed before using the CAN module.

This register is write-protected in user mode and can only be written in initialization mode. (See Section 3.6.1 on page 3-34.)

Note: Forbidden Configuration Values

To avoid unpredictable behavior of the CAN module, the CANBTC register should never be programmed with values not allowed by the CAN protocol specification and by the bit timing rules listed in Section 2.1.1.

Figure 3–21. CANBTC Register Bits

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								BRP.7	BRP.6	BRP.5	BRP.4	BRP.3	BRP.2	BRP.1	BRP.0
R-x								RWPI-0							

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					SBG	SJW		SAM	TSEG1				TSEG2		
R-0					RWPI-0	RWPI-0		RWPI-0	RWPI-0				RWPI-0		

Legend: RWPI = Read in all modes, write in privilege mode during initialization mode only, -*n* = Value after reset

Bit(s)	Name	Description
31:24	Reserved	Reads are undefined and writes have no effect.
23:16	BRP.7:0	Baud rate prescaler. This register sets the prescaler for the baud rate settings.

The length of one TQ is defined by:

$$TQ = \frac{BRP+1}{ICLK}$$

where ICLK is the frequency of the CAN module system clock and BRP is the binary value of (BRP.7:0 + 1), calculated as follows:

BRP = 1 + BRP.0 + (2 x BRP.1) + (4 x BRP.2) + (8 x BRP.3) + (16 x BRP.4) + (32 x BRP.5) + (64 x BRP.6) + (128 x BRP.7)

BRP is programmable from 1 to 256.

The value programmed into the CANBTC register has to be decremented by 1, because of the 8-bit size of the binary BRP value (BRP.7:0):

$BRP_{BTC} = BRP_{CALC} - 1$

BRP_{CALC} Result of bit-timing calculation.
1 BRP_{CALC} 256

BRP_{BTC} Value to be programmed into the CANBTC register.
0 BRP_{BTC} 255

Bit(s)	Name	Description
15:11	Reserved	
10	SBG	
		1 The eCAN module will resynchronize on both rising and falling edges.
		0 The eCAN module will resynchronise on the falling edge only.
9:8	SJW 1:0	Synchronization jump width. This determines the amount of time the synchronization adapts. This value is enhanced by one when the CAN module accesses these parameters.
7	SAM	This parameter sets the number of samples used by the CAN module to determine the actual level of the CAN bus. When the SAM bit is set, the level determined by the CAN bus corresponds to the result from the majority decision of the last three values. The sample points are at the sample point and twice before with a distance of 1/2 TQ.
		1 The CAN module will sample three times and make a majority decision. The triple sample mode shall be selected only for bit rate prescale values greater than 4 ($BRP_{CALC} > 4$).
		0 The CAN module samples only once at the sampling point.

Bit(s)	Name	Description
6:3	TSEG1 3:0	Time segment 1. TSEG1 is enhanced by one when the module accesses this parameter. The length of a bit on the CAN bus is determined by the parameters TSEG1, TSEG2, and BRP. All controllers on the CAN bus must have the same baud rate and bit length. At different clock frequencies of the individual controllers, the baud rate has to be adjusted by the said parameters. In the bit timing logic the conversion of the parameters to the required bit timing is realized. TSEG2 must be larger than or equal to 2. If a resynchronisation on falling AND rising edge is required it should be larger than or equal to 3. This parameter specifies the length of the TSEG1 segment in TQ units. The value of TSEG1 is calculated as follow: *TSEG1 = 1 + TSEG1.0 + (2 *TSEG1.1) + (4 *TSEG1.2) + (8 * TSEG1.3)* TSEG1 combines PROP_SEG and PHASE_SEG1 segments: *TSEG1 = PROP_SEG + PHASE_SEG1* where PROP_SEG and PHASE_SEG1 are the length of these two segments in TQ units. The value programmed into the CANBTC register has to be decremented by 1, because of the 4-bit size of the binary TSEG1 value (TSEG1.3:0): $TSEG1_{BTC} = TSEG1_{CALC} - 1$ $TSEG1_{CALC}$ Result of the calculation. 2 $TSEG1_{CALC}$ 16 $TSEG1_{BTC}$ Value to be programmed into the CANBTC register. 1 $TSEG1_{BTC}$ 15
2:0	TSEG2 3:0	Time Segment 2. TSEG2 is enhanced by one when the CAN module accesses this parameter. TSEG2 defines the length of PHASE_SEG2 segment in TQ units and is calculated as follow: *TSEG2 = 1 + TSEG2.0 + (2 TSEG2.1) + (4 TSEG2.2)* $TSEG2_{CALC}$ is programmable in the range of 1 TQ to 8 TQ and has to fulfill following timing rule: TSEG2 must be smaller than or equal to TSEG1 and must be greater than or equal to IPT. The value programmed into the CANBTC register has to be decremented by 1, because of the 3-bit size of the binary TSEG2 value (TSEG2.3:0): $TSEG2_{BTC} = TSEG2_{CALC} - 1$ $TSEG2_{CALC}$ Result of the calculation. $TSEG2_{BTC}$ Value to be programmed into the CANBTC

With the corresponding bit timing it is possible to perform multiple samplings of the bus line at the sample point. The level determined by the CAN bus corresponds to the result from a majority decision of three sample values.

The length of T_{SCL} is defined by:

$$TSCL = \frac{(BRP + 1)}{ICLK}$$

The register contents are zero after a reset.

TSCL is calculated according to the above formula. The synchronization segment SYNCSEG has always the length of 1 * TSCL.

IPT (information processing time) corresponds to the time necessary for the processing of the bit read. IPT corresponds to two units of TSCL.

The parameter SJW (2 bits) indicates, by how many units of TSCL a bit is allowed to be lengthened or shortened when resynchronizing. Values between 1 (SJW = 00b) and 4 (SJW = 11b) are adjustable. The synchronization is performed either with the falling edge (SBG = 0) or with both edges (SBG = 1) of the bus signal.

With the corresponding bit timing it is possible to reach a multiple sampling of the bus line at the sample point by setting SAM. The level determined by the CAN bus then corresponds to the result from the majority decision of the last three values. The sample points are at the sample point and twice before with a distance of 1/2 * TSCL.

Example:

A transmission rate of 1 MBps is adjusted, a bit has a length of 1 μs.

The baud rate prescaler is reset to 0. That means a bit for this data transmission rate has to be programmed with a length of eight * TSCL at 8 MHz. According to the above formula the values to be set are always by one smaller than the calculated values.

For example
TSEG1 = 4 / TSEG2 = 3 / SYNCSEG = 1.

With this setting a threefold sampling of the bus is not possible, thus SAM = 0 should be set. SJW is not allowed to be greater than TSEG2, so it is set to 2 units (SJW = 1).

3.5 Error Registers

The status of the CAN module is shown by the Errora and Status Register (CANES) and the error counter registers, which are described in this section.

3.5.1 Error and Status Register (CANES)

The error and status register comprises information about the actual status of the CAN module and displays bus error flags as well as error status flags. The way of storing the bus error flags (FE, BE, CRCE, SE, ACKE) and the error status flags (BO, EP, EW) in the ES register is subject to a special mechanism. If one of these error flags is set, then the current state of all other error flags is frozen. In order to update the ES register to the actual values of the error flags, the error flag which is set has to be acknowledged by writing a 1 to it. This mechanism allows the software to distinguish between the first error and all following.

Figure 3–22. CANES Register Bits

31	24	23	22	21	20	19	18	17	16
Reserved	FE	BE	SA1	CRCE	SE	ACKE	BO	EP	EW
R-0	RC-0	RC-0	R-1	RC-0	RC-0	RC-0	RC-0	RC-0	RC-0

15	6	5	4	3	2	1	0
Reserved		SMA	CCE	PDA	Res.	RM	TM
R-0		R-0	R-1	R-0	R-0	R-0	R-0

Legend: R = Read, C = Clear, *-n* = Value after reset

Bit(s)	Name	Description
31:25	Reserved	Reads are undefined and writes have no effect.
24	FE	Form error flag. 1 A form error occurred on the bus. This means that one or more of the fixed-form bit fields had the wrong level on the bus. 0 No form error detected; the CAN module was able to send and receive correctly.
23	BE	Bit error flag. 1 The received bit does not match the transmitted bit outside of the arbitration field or during transmission of the arbitration field, a dominant bit was sent but a recessive bit was received. 0 No bit error detected.
22	SA1	Stuck at dominant error. The SA1 bit is always at 1 after a hardware reset, a software reset, or a *Bus-Off* condition. This bit is cleared when a recessive bit is detected on the bus.

Bit(s)	Name	Description
		1 The CAN module never detected a recessive bit.
		0 The CAN module detected a recessive bit.
21	CRCE	CRC error.
		1 The CAN module received a wrong CRC.
		0 The CAN module never received a wrong CRC.
20	SE	Stuff error.
		1 A stuff bit error occurred.
		0 No stuff bit error occurred.
19	ACKE	Acknowledge error.
		1 The CAN module received no acknowledge.
		0 All messages have been correctly acknowledged.
18	BO	Bus-off state. The CAN module is in bus-off state.
		1 There is an abnormal rate of errors on the CAN bus. This condition occurs when the transmit error counter (CANTEC) has reached the limit of 256. During *Bus Off,* no messages can be received or transmitted. The only ways to exit this state by clearing the CCR (MC.12) bit or by setting the Auto Bus On (ABO) (MC.7) bit. After leaving *Bus Off*, the error counters are cleared.
		0 Normal operation
17	EP	Error-passive state
		1 The CAN module is in error-passive mode.
		0 The CAN module is in error-active mode.
16	EW	Warning status
		1 One of the two error counters (CANREC or CANTEC) has reached the warning level of 96.
		0 Values of both error counters (CANREC and CANTEC) are less than 96.
15:6	Reserved	Reads are undefined and writes have no effect.

Bit(s)	Name	Description
5	SMA	Suspend mode acknowledge. This bit is set after a latency of one clock cycle—up to the length of one frame—after the suspend mode was activated. The suspend mode is activated with the debugger tool when the circuit is not in run mode. During the suspend mode, the CAN module is frozen and cannot receive or transmit any frame. However, if the CAN module is transmitting or receiving a frame when the suspend mode is activated, the module will enter suspend mode only at the end of the frame. 1 The module has entered suspend mode. 0 The module is not in suspend mode.
4	CCE	Change configuration enable. This bit displays the configuration access right. This bit is set after a latency of one clock cycle up to the length of one frame. 1 The CPU has write access to the configuration registers. 0 The CPU is denied write access to the configuration registers.
3	PDA	Power-down mode acknowledge 1 The CAN module has entered the power-down mode. 0 Normal operation
2	Reserved	Reads are undefined and writes have no effect 1 The CAN module has entered the power-down mode. 0 Normal operation
1	RM	Receive mode. The CAN protocol kernel (CPK) is in receive mode. This bit reflects what the CPK is actually doing regardless of mailbox configuration. 1 The CAN protocol kernel is receiving a message. 0 The CAN protocol kernel is not receiving a message.
0	TM	Transmit mode. The CPK is in transmit mode. This bit reflects what the CPK is actually doing regardless of mailbox configuration. 1 The CAN protocol kernel is transmitting a message. 0 The CAN protocol kernel is not transmitting a message.

3.5.2 CAN Error Counter Registers (CANTEC/CANREC)

The CAN module contains two error counters: the receive error counter (CANREC) and the transmit error counter (CANTEC). The values of both counters can be read via the CPU interface. These counters are incremented or decremented according to the CAN protocol specification version 2.0.

Figure 3–23. Transmit Error Counter Register - CANTEC

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-x															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								TEC							
R-x								R-0							

Figure 3–24. Receive Error Counter Register - CANREC

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-x															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								REC							
R-x								R-0							

After reaching or exceeding the error passive limit (128), the receive error counter will not be increased anymore. When a message was received correctly, the counter is set again to a value between 119 and 127 (compare with CAN specification). After reaching the bus-off state, the transmit error counter is undefined while the receive error counter changes its function.

After reaching the bus-off state, the receive error counter is cleared. It will then be incremented after every 11 consecutive recessive bits on the bus. These 11 bits correspond to the gap between two frames on the bus. If the counter reaches 128, the module automatically changes back to the bus-on status if this feature is enabled (Auto Bus On bit (ABO) (MC.7) set). All internal flags are reset and the error counters are cleared. After leaving initialization mode, the error counters are cleared.

3.6 Interrupt Registers

Interrupts are controlled by the interrupt flag registers, interrupt mask registers, mailbox interrupt level registers, and the overwrite protection register. These registers are described in the following subsections.

3.6.1 Global Interrupt Flag Registers (CANGIF0 / CANGIF1)

These registers allow the CPU to identify the interrupt source.

The interrupt flag bits are set if the corresponding interrupt condition did occur. The global interrupt flags are set depending on the setting of the GIL bit in the GIM register. If that bit is set the global interrupts will set the bits in the GIF1 register otherwise in the GIF0 register. This also applies to the Interrupt Flags AAIF and RMLIF. These bits are set according to the setting of the appropriate GIL bit in the GIM register. .

The following bits are set regardless of the corresponding interrupt mask bits in the CANGIM register:

MTOFn	WDIFn	BOIFn
TCOIFn	WUIFn	EPIFn
AAIFn	RMLIFn	WLIFn

If all interrupt flags are cleared and a new interrupt flag is set the interrupt output line is activated when the corresponding interrupt mask bit is set. The interrupt line stays active until the interrupt flag is cleared by the CPU by writing a 1 to the appropriate bit. The GMIFx flags must be cleared by writing a 1 to the appropriate bit in the TA register or the RMP register (depending on mailbox configuration) and can not be cleared in the GIFx register. After clearing one or more interrupt flags and one or more interrupt flags still set, a new interrupt is generated. The interrupt flags are cleared by writing a 1 to the corresponding bit location. If the GMIFx is set the Mailbox Interrupt Vector MIVx indicates the mailbox number of the mailbox that caused the setting of the GMIFx. In case more than one mailbox interrupt is pending, it will always display the highest mailbox interrupt vector assigned to that interrupt line.

Figure 3–25. Global Interrupt Flag 0 Register - CANGIF0

31												18	17	16
Reserved													MTOF0	TCOF0
R-x													R/C-0	R/C-0

15	14	13	12	11	10	9	8	7	5	4	3	2	1	0
GMIF0	AAIF0	WDIF0	WUIF0	RMLIF0	BOIF0	EPIF0	WLIF0	Reserved		MIV0.4	MIV0.3	MIV0.2	MIV0.1	MIV0.0
R-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R-0		R-0	R-0	R-0	R-0	R-0

Figure 3–26. Global Interrupt Flag 1 Register - CANGIF1

31												18	17	16
Reserved													MTOF1	TCOF1
R-x													R/C-0	R/C-0

15	14	13	12	11	10	9	8	7	5	4	3	2	1	0
GMIF1	AAIF1	WDIF1	WUIF1	RMLIF1	BOIF1	EPIF1	WLIF1	Reserved		MIV1.4	MIV1.3	MIV1.2	MIV1.1	MIV1.0
R-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R-0		R-0	R-0	R-0	R-0	R-0

Legend: R = Read, C = Clear, *-n* = Value after reset

Note: HECC only, reserved in the SCC

Bit(s)	Name	Description
31:18	Reserved	Reserved. Reads are undefined and writes have no effect.
17	MTOF0	Mailbox time-out flag. This bit is not available in the SCC mode.
		1 One of the mailboxes did not transmit or receive a message within the specified time frame and the corresponding MILn bit is cleared.
		0 No time out for the mailboxes occurred.
17	MTOF1	Mailbox time-out flag. This bit is not available in the SCC mode.
		1 One of the mailboxes did not transmit or receive a message within the specified time frame and the corresponding MILn bit is set.
		0 No time out for the mailboxes occurred.
16	TCOF0/1	Time stamp counter overflow flag.
		1 The MSB of the time stamp counter has changed from 0 to 1.
		0 The MSB of the time stamp counter is 0. That is, it has not changed from 0 to 1.
15	GMIF0	Global mailbox interrupt flag. This bit is set only when the corresponding mailbox interrupt mask bit in the CANMIM register is set.

Bit(s)	Name	Description
		1 One of the mailboxes transmitted or received a message successfully and the corresponding MILn bit is cleared.
		0 No message has been transmitted or received.
15	GMIF1	Global mailbox interrupt flag. This bit is set only when the corresponding mailbox interrupt mask bit in the CANMIM register is set.
		1 One of the mailboxes transmitted or received a message successfully and the corresponding MILn bit is set.
		0 No message has been transmitted or received.
14	AAIF0	Abort-acknowledge interrupt flag
		1 A send transmission request has been aborted and the MILn bit corresponding to that mailbox is cleared.
		0 No transmission has been aborted.
14	AAIF1	Abort-acknowledge interrupt flag
		1 A transmission request has been aborted and the MILn bit corresponding to that mailbox is set.
		0 No transmission has been aborted.
13	WDIF0/WDIF1	Write-denied interrupt flag
		1 The CPU write access to a mailbox was not successful.
		0 The CPU write access to the mailbox was successful.
12	WUIF0/WUIF1	Wake-up interrupt flag
		1 During local powerdown, this flag indicates that the module has left sleep mode. During global powerdown, this flag indicates that there is activity on the CAN bus lines. This flag is also set when a global powerdown is requested by the CPU, but the eCAN module is not ready to enter powerdown mode yet.
		0 The module is still in sleep mode or normal operation
11	RMLIF0	Receive-message-lost interrupt flag
		1 At least for one of the receive mailboxes, an overflow condition has occurred and the corresponding bit in the MILn register is cleared.
		0 No message has been lost.
11	RMLIF1	Receive-message-lost interrupt flag

Bit(s)	Name	Description
		1 At least for one of the receive mailboxes, an overflow condition has occurred and the corresponding bit in the MILn register is set.
		0 No message has been lost.
10	BOIF0/BOIF1	Bus off interrupt flag
		1 The CAN module has entered bus-off mode.
		0 The CAN module is still in bus-on mode.
9	EPIF0/EPIF1	Error passive interrupt flag
		1 The CAN module has entered error-passive mode.
		0 The CAN module is not in error-passive mode.
8	WLIF0/WLIF1	Warning level interrupt flag
		1 At least one of the error counters has reached the warning level.
		0 None of the error counters has reached the warning level.
7:5	Reserved	Reads are undefined and writes have no effect.
4:0	MIV0.4:0/ MIV1.4:0	Message object interrupt vector. Only bits 3:0 are available in SCC mode. This vector indicates the number of the message object that set the global mailbox interrupt flag. It keeps that vector until the appropriate MIFn bit is cleared or when a higher priority mailbox interrupt occurred. Then the highest interrupt vector is displayed, with mailbox 31 having the highest priority. In the SCC or SCC-compatible mode, mailbox 15 has the highest priority. Mailboxes 16 to 31 are not recognized. If no flag is set in the TA/RMP register and GMIF1 or GMIF0 also cleared, this value is undefined.

3.6.2 Global Interrupt Mask Register (CANGIM)

The set up for the interrupt mask register is the same as for the interrupt flag register. If a bit is set, the corresponding interrupt is enabled. This register is write-protected in user mode.

Figure 3–27. Global Interrupt Mask Register (CANGIM) Bits

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved														MTOM	TCOM
R-0														RW-0	RW-0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	AAM	WDIM	WUIM	RMLIM	BOIM	EPIM	WLIM	Reserved					GIL	I1EN	I0EN
R-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	R-0					RWP-0	RWP-0	RWP-0

Legend: R = Read, W = Write, WP = Write in privilege mode only, *-n* = Value after reset

Bit(s)	Name	Description
31:18	Reserved	Reads are undefined and writes have no effect.
17	MTOM	
		1
		0
16	TCOM	
		1
		0
15	Reserved	Reads are undefined and writes have no effect.
14	AAIM	Abort Acknowledge Interrupt Mask.
		1 AAIF interrupt is enabled.
13	WDIM	Write denied interrupt mask
		1 WDIF interrupt is enabled.
		0 WDIF interrupt is disabled.
12	WUIM	Wake-up interrupt mask
		1 WUIF interrupt is enabled.
		0 WUIF interrupt is disabled.
11	RMLIM	Receive message lost interrupt mask

Bit(s)	Name	Description
		1 RMLIF interrupt is enabled.
		0 RMLIF interrupt is disabled.
10	BOIM	Bus off interrupt mask
		1 BOIF interrupt is enabled.
		0 BOIF interrupt is disabled.
9	EPIM	Error-passive interrupt mask
		1 EPIF interrupt is enabled.
		0 EPIF interrupt is disabled.
8	WLIM	Warning level interrupt mask
		1 WLIF interrupt is enabled.
		0 WLIF interrupt is disabled.
7:3	Reserved	Reads are undefined and writes have no effect.
2	GIL	Global interrupt level for the interrupts TCOF, WDIF, WUIF, BOIF, EPIF, and WLIF.
		1 All global interrupts are mapped to the HECC_INT_REQ[1] interrupt line.
		0 All global interrupts are mapped to the HECC_INT_REQ[0] interrupt line.
1	I1EN	Interrupt 1 enable
		1 This bit globally enables all interrupts for the HECC_INT_REQ[1] line if the corresponding masks are set.
		0 AAIF interrupt is disabled.
		0 The HECC_INT_REQ[1] interrupt line is disabled.
1	I0EN	Interrupt 0 enable
		1 This bit globally enables all interrupts for the HECC_INT_REQ[0] line if the corresponding masks are set.
		0 The HECC_INT_REQ[0] interrupt line is disabled.

The GMIF has no corresponding bit n the GIM because it originates from the MIF register and must be cleared there.

3.6.3 Mailbox Interrupt Mask Register (CANMIM)

There is one interrupt flag available for each mailbox. This may be a receive or a transmit interrupt depending on the configuration register. This register is write-protected in user mode.

31	0
MIM.31:0	
R/W-0	

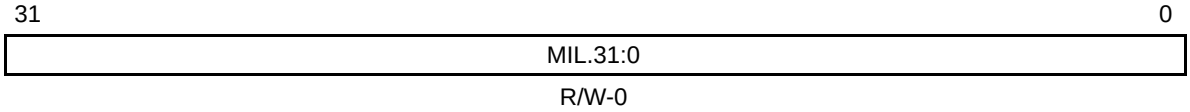
Note: R = Read, W = Write, -*n* = Value after reset

Bit(s)	Name	Description
31:0	MIM 31:0	Mailbox interrupt mask. After powerup all interrupt mask bits are cleared and the interrupts are disabled. These bits allow any mailbox interrupt to be masked individually. 1 Mailbox interrupt is enabled. An interrupt is generated if a message has been transmitted successfully (in case of a transmit mailbox) or if a message has been received without any error (in case of a receive mailbox). 0 Mailbox interrupt is disabled.

3.6.4 Mailbox Interrupt Level Register (CANMIL)

Each of the 32 mailboxes may initiate an interrupt on one of the two interrupt lines. Depending on the setting in the mailbox interrupt level register (CANMIL), the interrupt is generated on HECC_INT_REQ[0] (MILn = 0) or on line HECC_INT_REQ[1] (MIL[*n*] = 1). This applies also to the AAIFx and the RMLIFx flags.

Figure 3–28. Mailbox Interrupt Level Register Bits



Note: R = Read, W = Write, *-n* = Value after reset

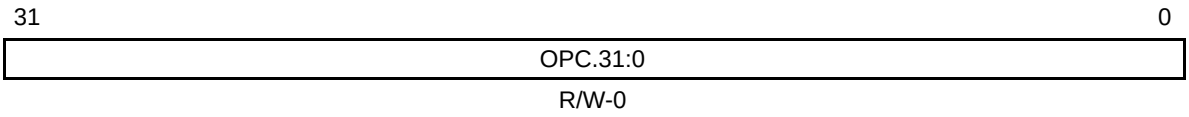
Bit(s)	Name	Description
31:0	MIL 31:0	Mailbox interrupt level. These bits allow any mailbox interrupt level to be selected individually. 1 The mailbox interrupt is generated on interrupt line 1. 0 The mailbox interrupt is generated on interrupt line 0.

3.6.5 Overwrite Protection Control Register (CANOPC)

If there is an overflow condition for mailbox *n* (RMP[*n*] is set to 1 and a new receive message would fit for mailbox *n*), the new message is stored depending on the settings in the CANOPC register. If the corresponding bit OPC[*n*] is set to 1, the old message is protected against being overwritten by the new message; thus, the next mailboxes are checked for a matching ID. If no other mailbox is found, the message is lost without further notification. If the bit OPC[*n*] is cleared to 0, the old message is overwritten by the new one. This will be notified by setting the receive message lost bit RML[*n*].

For read/write operations, only 32-bit access is supported.

Figure 3–29. Overwrite Protection Control Register Bits



Note: R = Read, W = Write, -*n* = Value after reset

Bit(s)	Name	Description
31:0	OPC 31:0	Overwrite protection control register
		1 If the bit OPC[*n*] is set to 1, an old message stored in that mailbox is protected against being overwritten by the new message.
		0 If the bit OPC[*n*] is not set, the old message may be overwritten by a new one.

3.7 eCAN I/O Control Registers (CANTIOC, CANRIOC)

The CANTX and CANRX pins may be configured as normal I/O pins if the CAN module is not used. This is done using the CANTIOC and CANRIOC registers. For read/write operations, only 32-bit access is supported.

Figure 3–30. TX I/O Control Register - CANTIOC

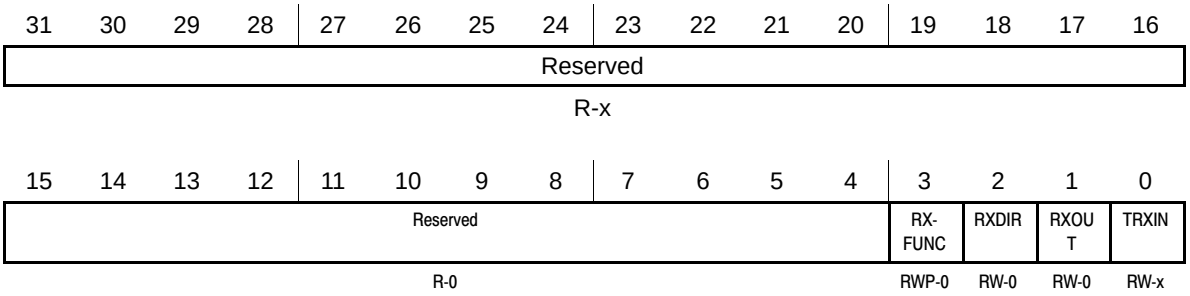
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved												TX-FUNC	TXDIR	TX-OUT	TXIN
R-0												RWP-0	RW-0	RW-0	RW-x

Legend: RW = Read/Write, RWP = Read in all modes, write in privilege mode only, -*n* = Value after reset, x = indeterminate

Bit(s)	Name	Description
31:4	Reserved	Reads are undefined and writes have no effect.
3	TXFUNC	External pin I/O enable flag. This bit is protected in user mode. When the CANTX pin is configured for CAN functions, the flags TXDIR and TXOUT have no meaning and can be set either to 0 or 1. The TXIN flag always shows the actual value of the CANTX pin. 1 The CANTX pin is used for the CAN transmit functions. 0 The CANTX pin is used as a normal I/O pin, the CAN protocol kernel (CPK) output is unconnected.
2	TXDIR	Transmit Pin Direction Flag. 1 The CANTX pin is used as an output pin if CANTX is configured as an I/O pin (TXFUNC = 0). 0 The CANTX pin is used as an input pin if CANTX is configured as an I/O pin (TXFUNC = 0).
1	TXOUT	Output value for CANTX pin. This value is output on the CANTX pin if CANTX is configured as an I/O pin (TXFUNC = 0) and as an output (TXDIR = 1).
0	TXIN	Reflects the value for the CANTX pin 1 Logic 1 present on pin 0 Logic 0 present on pin

Figure 3–31. RX I/O Control Register - CANRIOC



Legend: RW = Read/Write, RWP = Read in all modes, write in privilege mode only, -*n* = Value after reset, x = indeterminate

Bit(s)	Name	Description
31:4	Reserved	Reads are undefined and writes have no effect.
3	RXFUNC	External pin I/O enable flag. This bit is protected in user mode. 1 The CANRX pin is used for the CAN receive functions. 0 The CANRX pin is used as a normal I/O pin. The RX signal is still internally connected to the CPK.
2	RXDIR	Receive pin direction flag 1 The CANRX pin is used as an output pin if CANRX is configured as an I/O pin (RXFUNC = 0). 0 The CANRX pin is used as an input pin if CANRX is configured as an I/O pin (RXFUNC = 0).
1	RXOUT	Output value for CANRX pin. This value is output on the CANRX pin if CANRX is configured as an I/O pin (RXFUNC = 0) and as an output (RXDIR = 1).
0	RXIN	Reflects the value on the CANRX pin. 1 Logic 1 present on pin 0 Logic 0 present on pin