TMS470R1x Controller Area Network (CAN) Reference Guide

Literature Number: SPNU197E
July 2005

TEXAS INSTRUMENTS

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265

REVISION HISTORY

REVISION	DATE	NOTES
E	7/05	Page 43, note on wakeup delay from STANDBY due to bus-off condition added. Page 68, SRES bit reset value corrected. Page 70, ABO bit description updated. Page 78, BO bit description updated. Page 95, note on clearing the CCR bit after bus-off condition added

Contents

Figures

Tables

Controller Area Network (CAN)

This reference guide describes the Controller Area Network (CAN). The CAN uses established protocol to communicate serially with other controllers in harsh environments. This chapter describes the TMS470 CAN controllers, the Standard CAN Controller (SCC) and the High-End CAN Controller (HECC). Unless otherwise stated in the text, all information is related to both the SCC and HECC.

Topic **Page**

1 CAN Overview

The TMS470 CAN controller is available in two different implementations that are both fully compliant with the CAN protocol, version 2.0B. The two different CAN controller versions use the same CAN protocol kernel module to perform the basic CAN protocol tasks. Only the message controller differs between the two CAN controller versions. See Table 1.

Key features of the CAN module include:

- ❏ Common CAN protocol kernel (CPK) to perform protocol tasks
- ❏ Standard CAN controller (SCC) for standard CAN applications
  - ■ Sixteen message objects
  - ■ Three receive identifier masks
- ❏ High-end CAN controller (HECC) for complex applications
  - ■ Thirty-two message objects
  - ■ Thirty-two receive identifier masks

Table 1. SCC and HECC Features Overview

Feature	SCC	HECC
Number of message objects	16 Rx/Tx	32 Rx/Tx
Number of receive identifier masks	3	32
CAN, version 2.0B compliant	X	X
Low-power mode	X	X
Programmable wake-up on bus activity	X	X
Programmable interrupt scheme	X	X
Automatic reply to a remote request	X	X
Automatic retransmission in case of error	X	X
Protect against reception of new message	X	X
32-bit time stamp		X
Local network time counter		X
Programmable priority register for each message		X
Programmable transmission and reception time-out		X

1.1 CAN Protocol Processor Features

The CAN protocol kernel (CPK) performs the basic CAN protocol tasks according to CAN protocol specification 2.0B. The CPK is the basic module of all CAN controller implementations.

The CPK provides the following features:

- Full implementation of CAN protocol, version 2.0B
  - Standard and extended identifiers
  - Data and remote frames
- Bus speed of up to 1 Mbps with an 8-MHz system clock
- Programmable prescaler from 1 to 256
- Programmable bit time according to the CAN specification
- Programmable sampling mode
  - One or three samples used to determine the received bit value
- Selectable edge of receive bit flow for synchronization
  - Both edges or falling edge only
- Automatic retransmission of a frame in case of loss of arbitration
- Low-power mode
- Bus failure diagnostic
  - Bus on/off
  - Error passive/active
  - Bus error warning
  - Bus stuck dominant
  - Frame error report: cyclic redundancy check (CRC), stuff, form, bit, and acknowledgment errors
  - Readable error counters
- Self-test mode
  - Operate in a loop-back mode (receiving its own message)
  - Dummy acknowledge

1.2 SCC Features

The TMS470 device family has the following SCC features:

- ❑ All the CPK features
  - ■ Full implementation of CAN protocol, version 2.0B
- ❑ Sixteen message objects, each with the following properties:
  - ■ Configurable as receive or transmit
  - ■ Configurable with standard or extended identifier
  - ■ Protects against reception of new message
  - ■ Supports data and remote frame
  - ■ Composed of 0 to 8 bytes of data
  - ■ Employs a programmable interrupt scheme with two interrupt levels
  - ■ Has a message priority based on object number
- ❑ Two programmable local identifier masks for objects 0–2 and 3–5
  - ■ Configurable as standard or extended message identifier
  - ■ Has an acceptance mask register for identifier extension bit
- ❑ One global programmable identifier mask for objects 6–15
  - ■ Configurable as standard or extended message identifier
  - ■ Has an acceptance mask register for identifier extension bit
- ❑ 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

1.3 HECC Features

The TMS470 device family has the following HECC features:

- ❑ All the CPK features
  - ■ Full implementation of CAN protocol, version 2.0B
- ❑ Thirty-two message objects, 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)
- SCC-compatible mode
  - Upwardly compatible software
  - Software written for the SCC can run without any changes on the HECC
  - SCC-compatible mode is automatically activated after RESET

2 Overview of 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 8 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.

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.

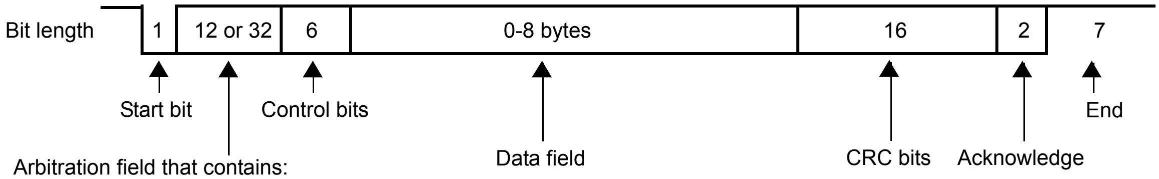
In addition, CAN specification version 2.0B defines two different formats that differ in the length of the identifier field: standard frames with an 11-bit identifier and extended frames with 29-bit identifier.

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, 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. CAN Data Frame



Arbitration field that contains:
– 11-bit identifier + RTR bit for standard frame format
– 29-bit identifier + SRR bit + IDE bit + RTR bit for extended frame format
Where: RTR = Remote Transmission Request
SRR = Substitute Remote Request
IDE = Identifier Extension

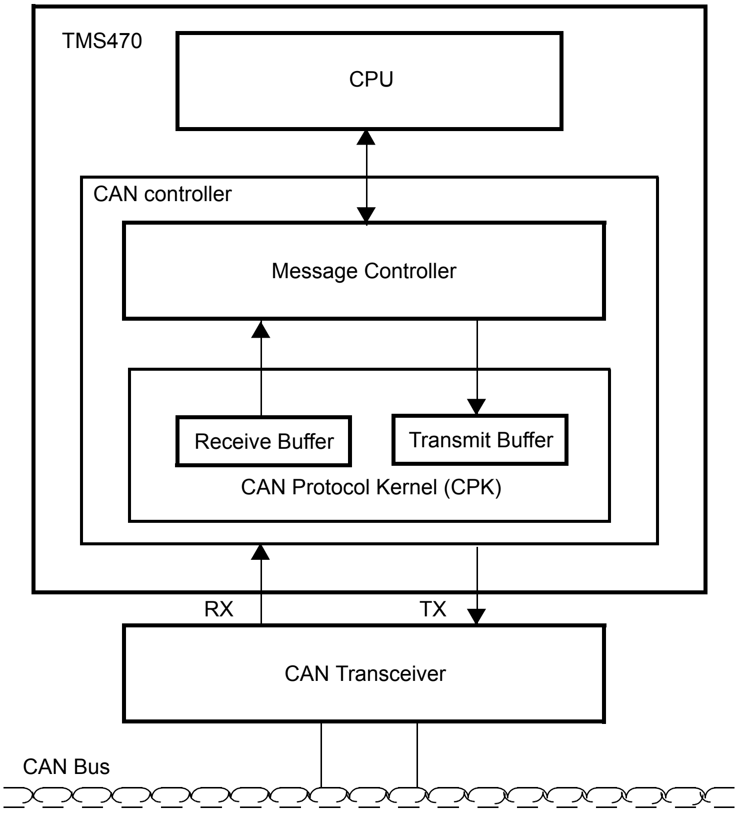
Note: Unless otherwise noted, numbers are amount of bits in field.

2.2 CAN Controller Overview

The TMS470 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 both the SCC and the HECC controllers, shown in Figure 2, is composed of a CAN protocol kernel (CPK) and a message controller.

Figure 2. Architecture of the SCC and HECC CAN Controllers



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.

The SCC and the HECC differ only by their message controller - the CPK always offers the same services. The message management of the message controller is not part of the CAN protocol specification.

3 Standard CAN Controller (SCC) Overview

The SCC is a standard CAN controller with an internal 32-bit architecture, upwardly compatible with HECC.

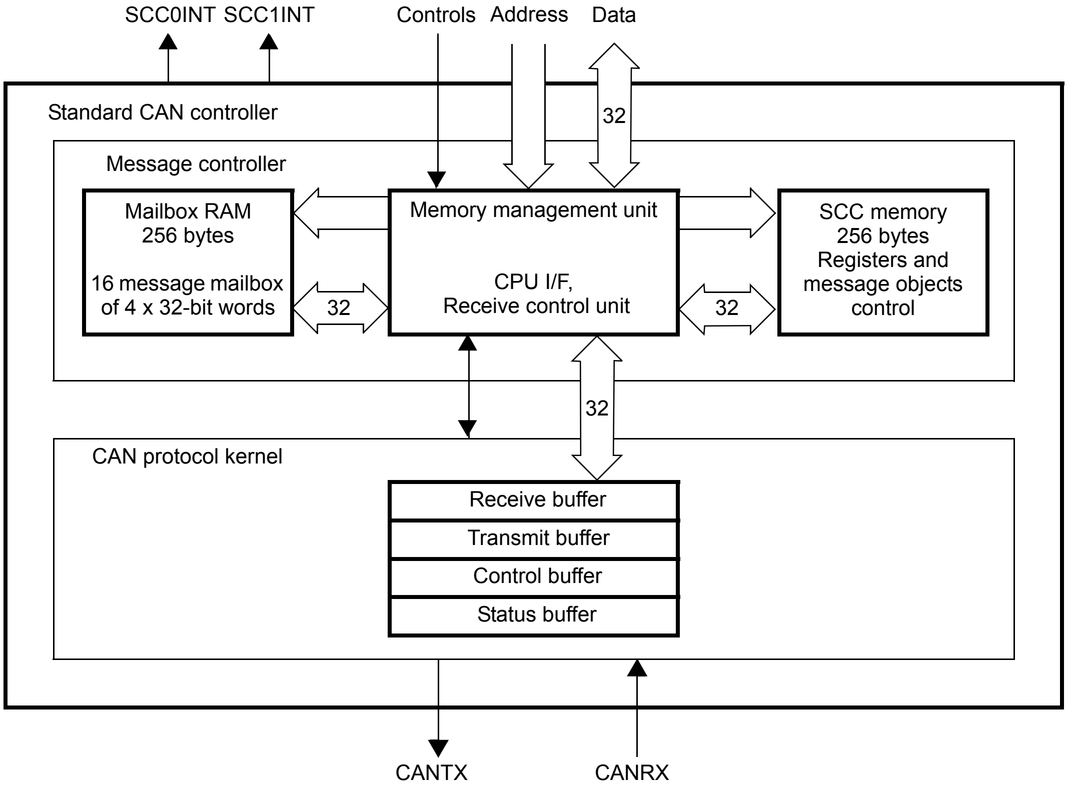
The SCC, shown in Figure 3, consists of the following:

❑ The CAN protocol kernel (CPK)

❑ The message controller comprising the following:

- The memory management unit (MMU), including the CPU interface and the receive control unit (acceptance filtering)
- 256 bytes of mailbox RAM enabling the storage of 16 messages
- 256 bytes of space register containing the global and the local identifier masks.

Figure 3. SCC Functional Block Diagram



After the reception of a valid message by the CPK, the receive control unit of the message controller determines if the message must be stored into one of the 16 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 comprises an identifier of 11 or 29 bits (contained in the arbitration field), 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 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 is transferred into the CPK by the message controller.

The memory registers contain the global identifier mask and the two dedicated identifier masks for the message objects 0–2 and 3–5. After the reception of a valid identifier, the receive control unit checks the state and the identifier of all objects to determine if the received message must be stored into one of the message object’s buffers.

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 is then done 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.

3.1 SCC Memory Map

The SCC module has two different offset addresses mapped in the TMS470 memory. The first offset address, *SCC_Offset*, is used to access the control and status registers, and the acceptance masks of the message objects. This memory range can only be accessed 32-bit wide. The second offset address, *Mailbox_RAM_Offset,* is used to access the message mailboxes. This memory range can be accessed 8-bit, 16-bit and 32-bit wide. Each of these two memory blocks uses 256 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 acceptance filtering, message transmission, and interrupt handling.

The mailbox module in the SCC provides sixteen message mailboxes of 8-byte data length, a 29-bit identifier, and several control bits. Each mailbox can be configured to either transmit or receive data.

Note: Unused Message Mailboxes

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

3.2 SCC Registers

The SCC and HECC registers, listed in Table 2, are used by the CPU to configure and control the CAN controller and the message objects.

Table 2. Combined SCC/HECC Registers Map

Offset Address†	Mnemonic	SCC Name	HECC Name	Page
0x00	CANME	Mailbox enable	Mailbox enable	3-53
0x04	CANMD	Mailbox direction	Mailbox direction	3-54
0x08	CANTRS	Transmission request set	Transmission request set	3-55
0x0C	CANTRR	Transmission request reset	Transmission request reset	3-56
0x10	CANTA	Transmission acknowledge	Transmission acknowledge	3-57
0x14	CANAA	Abort acknowledge	Abort acknowledge	3-58
0x18	CANRMP	Receive message pending	Receive message pending	3-59
0x1C	CANRML	Receive message lost	Receive message lost	3-60
0x20	CANRFP	Remote frame pending	Remote frame pending	3-61
0x24	CANGAM	Global acceptance mask	Reserved	3-62
0x28	CANMC	Master control	Master control	3-63
0x2C	CANBTC	Bit-timing configuration	Bit-timing configuration	3-67
0x30	CANES	Error and status	Error and status	3-71
0x34	CANTEC	Transmit error counter	Transmit error counter	3-74
0x38	CANREC	Receive error counter	Receive error counter	3-74
0x3C	CANGIF0	Global interrupt flag 0	Global interrupt flag 0	3-76
0x40	CANGIM	Global interrupt mask	Global interrupt mask	3-79
0x44	CANGIF1	Global interrupt flag 1	Global interrupt flag 1	3-76
0x48	CANMIM	Mailbox interrupt mask	Mailbox interrupt mask	3-81
0x4C	CANMIL	Mailbox interrupt level	Mailbox interrupt level	3-82
0x50	CANOPC	Overwrite protection control	Overwrite protection control	3-83
0x54	CANTIOC	TX I/O control	TX I/O control	3-84
0x58	CANRIOC	RX I/O control	RX I/O control	3-85
0x5C	CANLNT	Reserved	Local network time	3-44
0x60	CANTOC	Reserved	Time-out control	3-46
0x64	CANTOS	Reserved	Time-out status	3-46

† The actual addresses of the registers are device-specific. See the specific device data sheet to verify the module register addresses. The offset address of the device must be added to the actual register addresses.

4 High-End CAN Controller (HECC) Overview

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

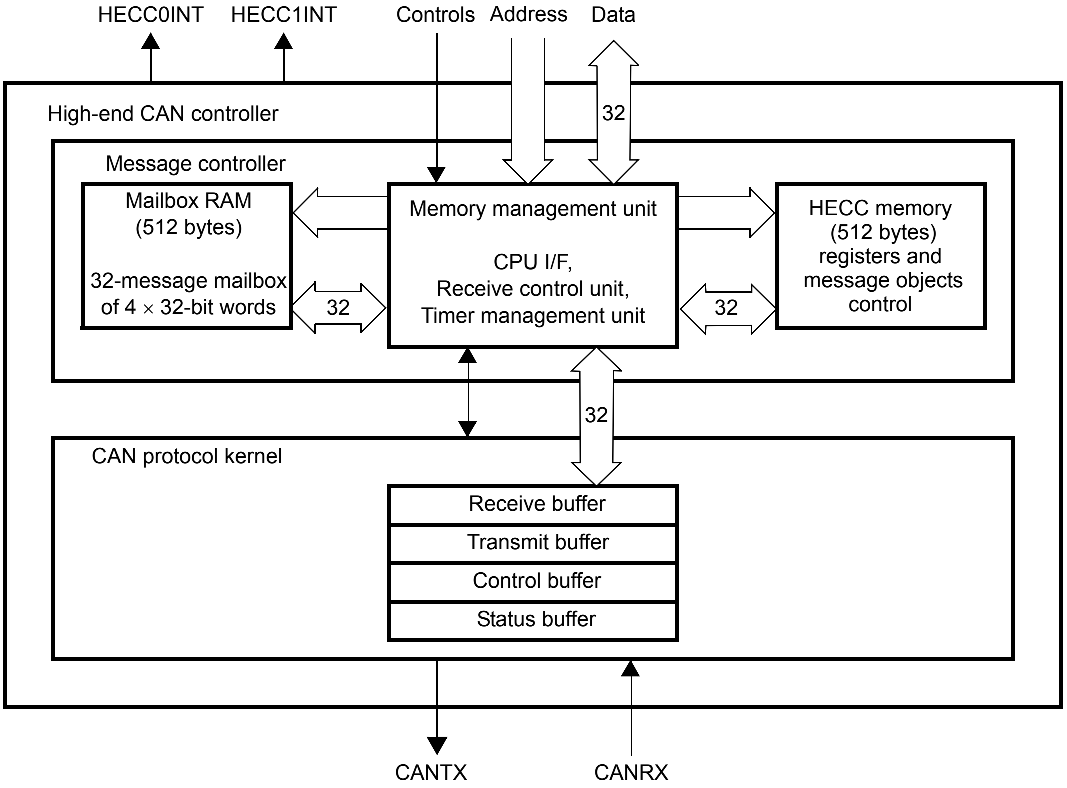
The HECC, shown in Figure 4, consists of the following:

❑ The CAN protocol kernel (CPK)

❑ The message controller comprising the following:

- ■ 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

Figure 4. HECC Functional Block Diagram



After the CPK receives a valid message, 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 cannot find any mailbox to store the received message, the message is discarded.

A message comprises 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 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.

4.1 SCC-Compatible Mode

The HECC can be used in SCC mode. In this mode, all functions specific to the HECC are not available and the HECC behaves exactly as the SCC. This mode is selected by default to allow any application software written for the SCC to run on the HECC without any modification. When using the HECC in SCC-compatible mode, the user must refer to the SCC specification only.

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

Note: HECC Used in SCC Mode

When the HECC is configured in SCC mode, the HECC behaves exactly as the SCC and you should refer to the SCC specification only.

4.2 HECC Memory Map

The HECC module has two different offset addresses mapped in the TMS470 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 5, 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.

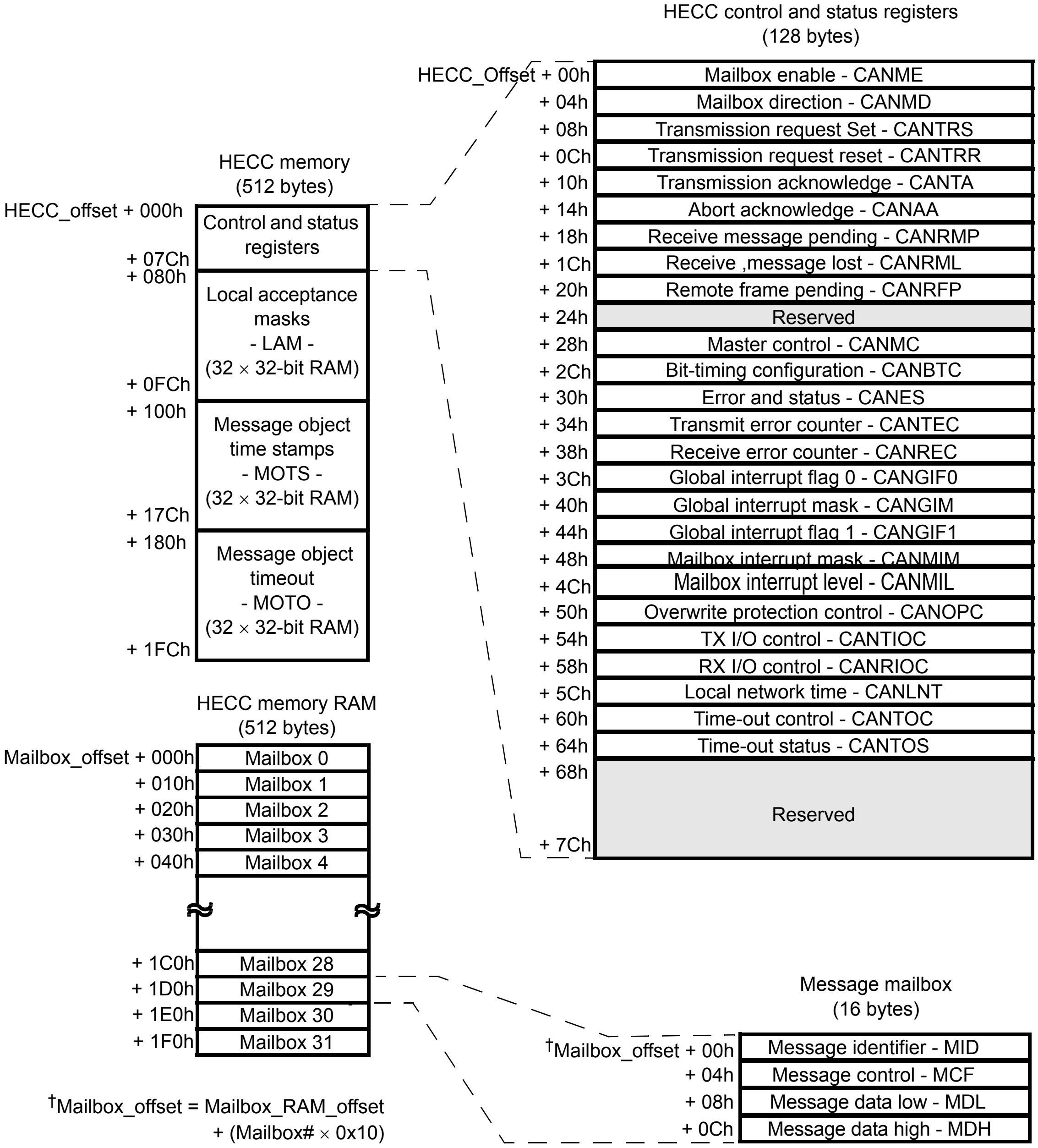
Note: Unused Message Mailboxes

All RAM areas are byte-writable and can be used as normal memory. In this case, you 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.

Note: HECC Used in SCC Mode

When the HECC is used in SCC mode, you should refer to the SCC memory map.

Figure 5. HECC Memory Map



4.3 HECC Registers

The SCC and HECC registers listed in Table 3 are used by the CPU to configure and control the CAN controller and the message objects.

Table 3. Combined SCC/HECC Registers Map

Offset Address[†]	Mnemonic	SCC Name	HECC Name	Page
0x00	CANME	Mailbox enable	Mailbox enable	3-53
0x04	CANMD	Mailbox direction	Mailbox direction	3-54
0x08	CANTRS	Transmit request set	Transmit request set	3-55
0x0C	CANTRR	Transmit request reset	Transmit request reset	3-56
0x10	CANTA	Transmission acknowledge	Transmission acknowledge	3-57
0x14	CANAA	Abort acknowledge	Abort acknowledge	3-58
0x18	CANRMP	Receive message pending	Receive message pending	3-59
0x1C	CANRML	Receive message lost	Receive message lost	3-60
0x20	CANRFP	Remote frame pending	Remote frame pending	3-61
0x24	CANGAM	Global acceptance map	Reserved	3-62
0x28	CANMC	Master control	Master control	3-63
0x2C	CANBTC	Bit-Timing configuration	Bit-timing configuration	3-67
0x30	CANES	Error and status	Error and status	3-71
0x34	CANTEC	Transmit error counter	Transmit error counter	3-74
0x38	CANREC	Receive error counter	Receive error counter	3-74
0x3C	CANGIF0	Global interrupt flag 0	Global interrupt flag 0	3-76
0x40	CANGIM	Global interrupt mask	Global interrupt mask	3-79
0x44	CANGIF1	Global interrupt flag 1	Global interrupt flag 1	3-76
0x48	CANMIM	Mailbox interrupt mask	Mailbox interrupt mask	3-81
0x4C	CANMIL	Mailbox interrupt level	Mailbox interrupt level	3-82
0x50	CANOPC	Overwrite protection control	Overwrite protection control	3-83
0x54	CANTIOC	TX I/O control	TX I/O control	3-84
0x58	CANRIOC	RX I/O control	RX I/O control	3-84
0x5C	CANLNT	Reserved	Local network time	3-44
0x60	CANTOC	Reserved	Time-out control	3-46
0x64	CANTOS	Reserved	Time-out status	3-46

[†] The actual addresses of the registers are device-specific. See the specific device data sheet to verify the module register addresses. The offset address of the device must be added to the actual register addresses.

5 Message Objects

5.1 SCC Message Objects

The message controller of the SCC can handle 16 different message objects.

Each message object can be configured to either transmit or receive. In the SCC, message objects 0–2, 3 to 5, and 6–15 share the same acceptance masks.

A SCC message object consists of 20 bytes of RAM distributed, as shown in Table 4, as:

- ❑ A message mailbox comprising the following:
  - ■ The 29-bit message identifier
  - ■ The message control register
  - ■ 8 bytes of message data
- ❑ A 29-bit acceptance mask

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

Table 4. SCC Message Object Description

Offset / Address[†]	**Mnemonic**	**Name**	**Page**
Mailbox_RAM_Offset + (Object# × 0x10) + 0x00	MID	Message identifier	3-27
Mailbox_RAM_Offset + (Object# × 0x10) + 0x04	MCF	Message control field	3-28
Mailbox_RAM_Offset + (Object# × 0x10) + 0x08	MDL	Message data low word	3-29
Mailbox_RAM_Offset + (Object# × 0x10) + 0x0C	MDH	Message data high word	3-29
SCC_Offset + 0x80 → for objects 0 to 2 or	LAM(0)	Local acceptance mask	3-31
SCC_Offset + 0x8C → for objects 3 to 5 or	LAM(3)	Local acceptance mask	3-31‘
SCC_Offset + 0x24 → for objects 6 to 15	GAM	Global acceptance mask	3-62

[†] The actual addresses of the message objects are device-specific. See the specific device data sheet to verify the SCC module memory offset and RAM offset. Object# specifies the number of the message object.

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.

5.2 HECC Message Objects

The message controller of the HECC can handle 32 different message objects.

Each message object can be configured to either transmit or receive. In the HECC, each message object has its individual acceptance mask.

A HECC message object consists of 28 bytes of RAM distributed, as shown in Table 5, as:

- ❑ A message mailbox comprising the following:
  - ■ 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 HECC registers allow control of the message objects.

Table 5. HECC Message Object Description

Offset / Address[†]	Mnemonic	Name	Page
Mailbox_RAM_Offset + (Object# × 0x10) + 0x00	MID	Message identifier	3-27
Mailbox_RAM_Offset + (Object# × 0x10) + 0x04	MCF	Message control field	3-28
Mailbox_RAM_Offset + (Object# × 0x10) + 0x08	MDL	Message data low word	3-29
Mailbox_RAM_Offset + (Object# × 0x10) + 0x0C	MDH	Message data high word	3-29
HECC_Offset + (Object# × 4) + 0x80	LAM	Local acceptance mask	3-31
HECC_Offset + (Object# × 4) + 0x100	MOTS	Message object time stamp	3-44
HECC_Offset + (Object# × 4) + 0x180	MOTO	Message object time-out	3-45

[†] 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: HECC Used in SCC Mode

When the HECC is used in SCC mode, you must refer to the SCC message objects description (see Table 4).

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.

5.3 CAN 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 HECC (MCF.13:8)

Each of the mailboxes can be configured as one of four message object types (see Table 6). 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 6. 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

5.3.1 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, the priority of the mailbox transmission depends on the mailbox number. The highest mailbox number (=15) comprises the highest transmit priority.

In the HECC, 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.

5.3.2 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.

5.3.3 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 the case of a matching identifier (with the message object configured as send mailbox and bit AAM in this message object is not set) this message is not received.

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 3.11.7 on page 3-59 for information about the CANRMP register.

If the CPU wants to change the data in a message object that is configured as remote frame mailbox (AAM (MID.29) = 1), it has to set the mailbox number MBNR (MC.4-0) and the change data request bit CDR (MC.8) first. The CPU may then perform the access and clear CDR to tell the CAN module that the access is finished. Until CDR is cleared, the transmission of this mailbox is not performed by the CAN module. Since TRS (TRS.31-0) is not affected by CDR, a pending transmission is started after CDR is cleared. Thus, the newest data will be sent.

To change the identifier in that mailbox, the message object first must be disabled (ME[*n*] = 0).

If the CPU wants to request data from another node, it may configure the message object as receive mailbox and set TRS. 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.

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.

5.3.4 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.

5.3.5 Message Identifier Register (MID)

This register can only be written if mailbox *n* is disabled (ME[*n*] (ME.31-0) = 0). Figure 6 and Table 7 describe this register.

Figure 6. Message Identifier Register (MID) Fields

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x00†	IDE	AME	AAM	ID.28:18											ID.17:16	
	RW-x	RW-x	RW-x	RW-x											RW-x	

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	ID.15:0															
	RW-x															

RW = Read any time, write when mailbox is disabled, -*n* = Value after reset, x = indeterminate

† Relative address = Mailbox_RAM_Offset + (Object# × 0x10)

Table 7. Message Identifier Register (MID) Field Descriptions

Bit	Name	Value	Description
31	IDE		Identifier Extension Bit.
		0	Standard identifier (11 bits). The received message or the message to be sent has a standard identifier.
		1	Extended identifier (29 bits). The received message or the message to be sent has an extended 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.
		0	No acceptance mask will be used, all identifier bits must match to receive the message.
		1	The corresponding acceptance mask is used.

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.
		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.
		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.
28–0	ID 28:0		Message Identifier. 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. In extended identifier mode, if the IDE bit (MID.31 = 1), the message identifier is stored in bits ID.28:0.

5.3.6 Message Control Field Register (MCF)

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). Figure 7 and Table 8 describe this register.

Figure 7. Message Control Field Register (MCF)

Bits	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
0x04†	Reserved															

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

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

† Relative address = Mailbox_RAM_Offset + (Object# × 0x10)
‡ HECC only, reserved in the SCC

Table 8. Message Control Field Register (MCF) Field Descriptions

Bit	Name	Value	Description
31–13	Reserved		
13–8	TPL.5:0		Transmit Priority Level. This 6-bit field in the HECC (reserved in the SCC) defines the priority of this mailbox as compared to the other 31 mailboxes. The highest number has the highest priority. In the case that 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		
4	RTR		Remote Transmission Request Bit.
		0	No remote frame is requested.
		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. *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.
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.

5.3.7 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 only be written 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 8 through Figure 11 illustrate these registers.

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

Bits	31 24	23 16	15 8	7 0
0x08[†]	Byte 0	Byte 1	Byte 2	Byte 3
	RW-x		RW-x	

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

Bits	31 24	23 16	15 8	7 0
0x0C[†]	Byte 4	Byte 5	Byte 6	Byte 7
	RW-x		RW-x	

RW = Read at any time, write when the mailbox is disabled or configured in transmission, -*n* = Value after reset, x = indeterminate

[†] Relative address = Mailbox_RAM_Offset + (Object# × 0x10)

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

Bits	31 24	23 16	15 8	7 0
0x08[†]	Byte 3	Byte 2	Byte 1	Byte 0
	RW-x		RW-x	

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

Bits	31 24	23 16	15 8	7 0
0x0C†	Byte 7	Byte 6	Byte 5	Byte 4
	RW-x		RW-x	

RW = Read at any time, write when the mailbox is disabled or configured in transmission, -*n* = Value after reset, x = indeterminate

† Relative address = Mailbox_RAM_Offset + (Object# × 0x10)

Note: Data Field

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

5.4 CAN Acceptance Filter

In the SCC and in the HECC are two different implementations of the handling of the masks for the mailboxes. If the SCC compatibility bit in the CANMC register is set, the HECC behaves like the SCC.

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.

5.4.1 SCC Acceptance Filtering

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.

5.4.2 HECC Acceptance Filtering

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.

5.4.3 Local Acceptance Mask Register (LAM)

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 12 and Table 9 describe this register.

Figure 12. Local Acceptance Mask Register (LAM)

Bits	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
0x80†	LAMI	Reserved		LAM.28:18											LAM.17:16	
	RW-x			RW-x											RW-x	

Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	LAM.15:0															
	RW-x															

RW = Read/Write, *-n* = Value after reset, x = indeterminate for HECC, x = 0 for SCC

† Relative address = + SCC/HECC_Offset + (Object# × 4)

Table 9. Local Acceptance Mask Register (LAM) Field Descriptions

Bit	Name	Value	Description
31	LAMI		Local Acceptance Mask Identifier Extension Bit.
		0	The identifier extension bit stored in the mailbox determines which messages shall be received.
		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.
30–29	Reserved		Reads are undefined and writes have no effect.
28–0	LAM.28:0		Local Acceptance Mask. These bits enable the masking of any identifier bit of an incoming message.
		0	Received identifier bit value must match the corresponding identifier bit of the MID register.
		1	Accept a 0 or a 1 (don’t care) for the corresponding bit of the received identifier.

6 CAN Module Initialization

The CAN module must be initialized before the utilization. Initialization is only possible if the module is in initialization mode. See Figure 13.

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.

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.

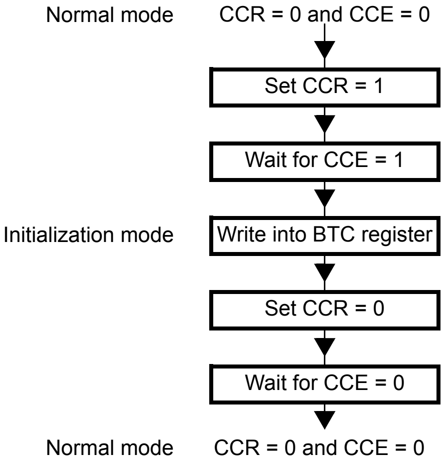
The module is activated again by programming CCR(MC.12) = 0.

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 13. Configuration 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 can't detect a bus-idle condition and therefore is unable to perform a mode transition.

6.1 CAN Bit-Timing Configuration

As shown in Figure 14, the CAN protocol specification partitions the nominal bit time into four different time segments:

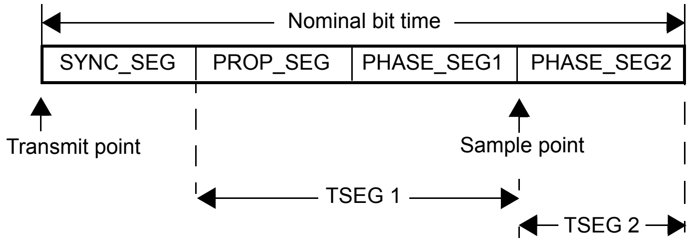
SYNC_SEG: this part of bit time is used to synchronize the various nodes on the bus. An edge is expected to lie within this segment. This segment is always 1 TIME QUANTUM (TQ).

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.

Figure 14. Partition of the Bit Time



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 SCC/HECC modules, 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.)

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 have to be fulfilled when determining the bit segment values:

- $TSEG1_{CALC(min)} \geq TSEG2_{CALC}$
- Information processing time (IPT) $\leq TSEG1_{CALC} \leq$ 16 TQ
- IPT $\leq TSEG2_{CALC} \leq$ 8 TQ
- IPT = 3 / BRP_{CALC} (the resulting IPT has to be rounded up to the next integer value)

> Note: For the special case of baud rate prescaler value BRPCALC = 1, the IPT is equal to 3 time quanta. This parameter is not compliant to the ISO 11898 standard, where the IPT is defined to be less than or equal to 2 time quanta.
>
> Thus, using this mode (BRPCALC = 1) is not allowed.

- 1 TQ $\leq SJW_{CALC} \leq$ min[4 TQ, $TSEG2_{CALC}$] (SJW = Synchronization Jump Width)
- To utilize three-time sampling mode $BRP_{CALC} \geq 5$ has to be selected

6.2 CAN Bit Rate Calculation

Bit rate is calculated as follows (in bits per second):

$$Bitrate = \frac{ICLK}{BRP \times BitTime}$$

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 11.12 on page 72 for more information about the CANBTC register.)

BTC = 0x0000021A

7 CAN 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 15, on page 38 and Figure 16, on page 39.

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 15. SCC Interrupts Scheme

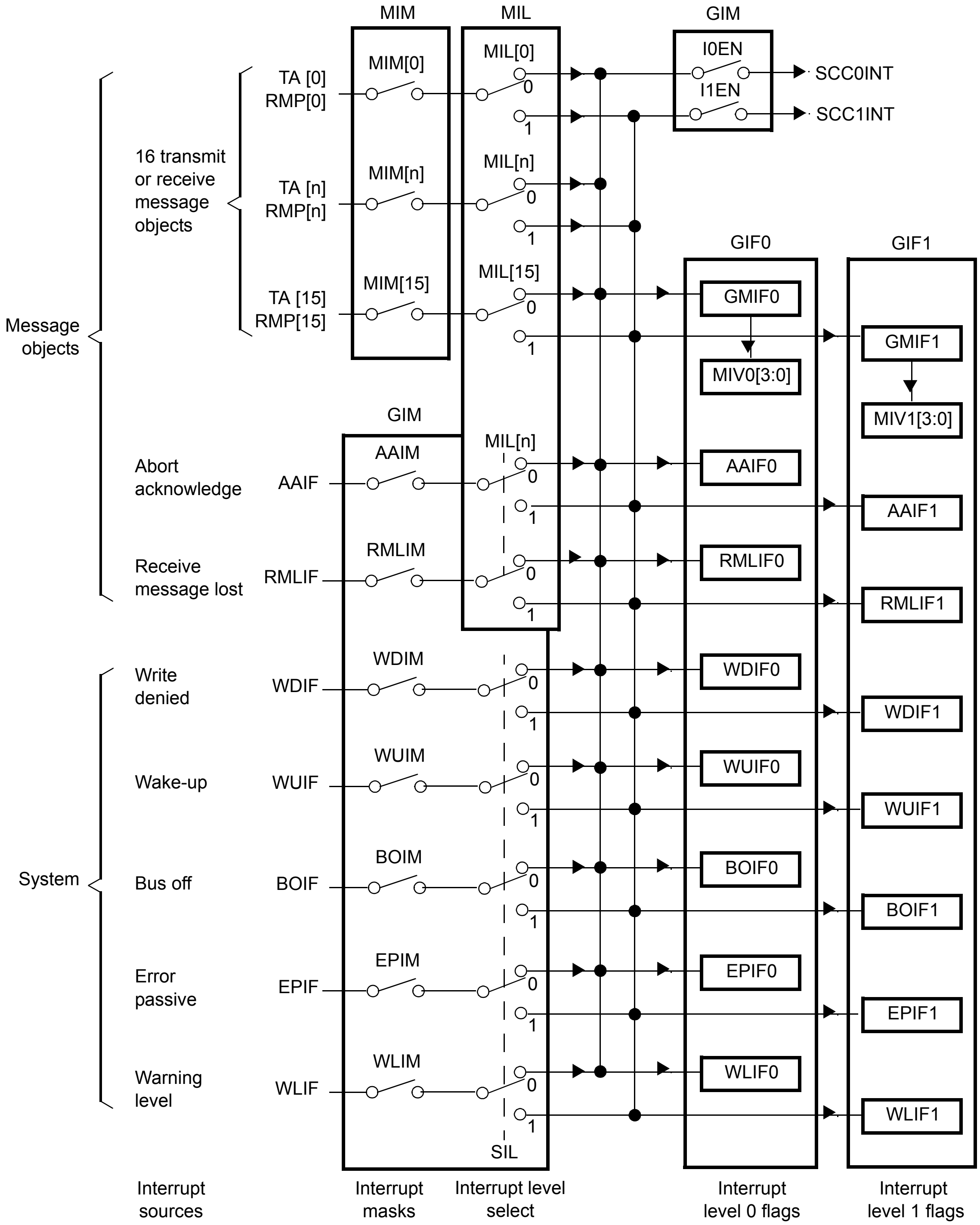
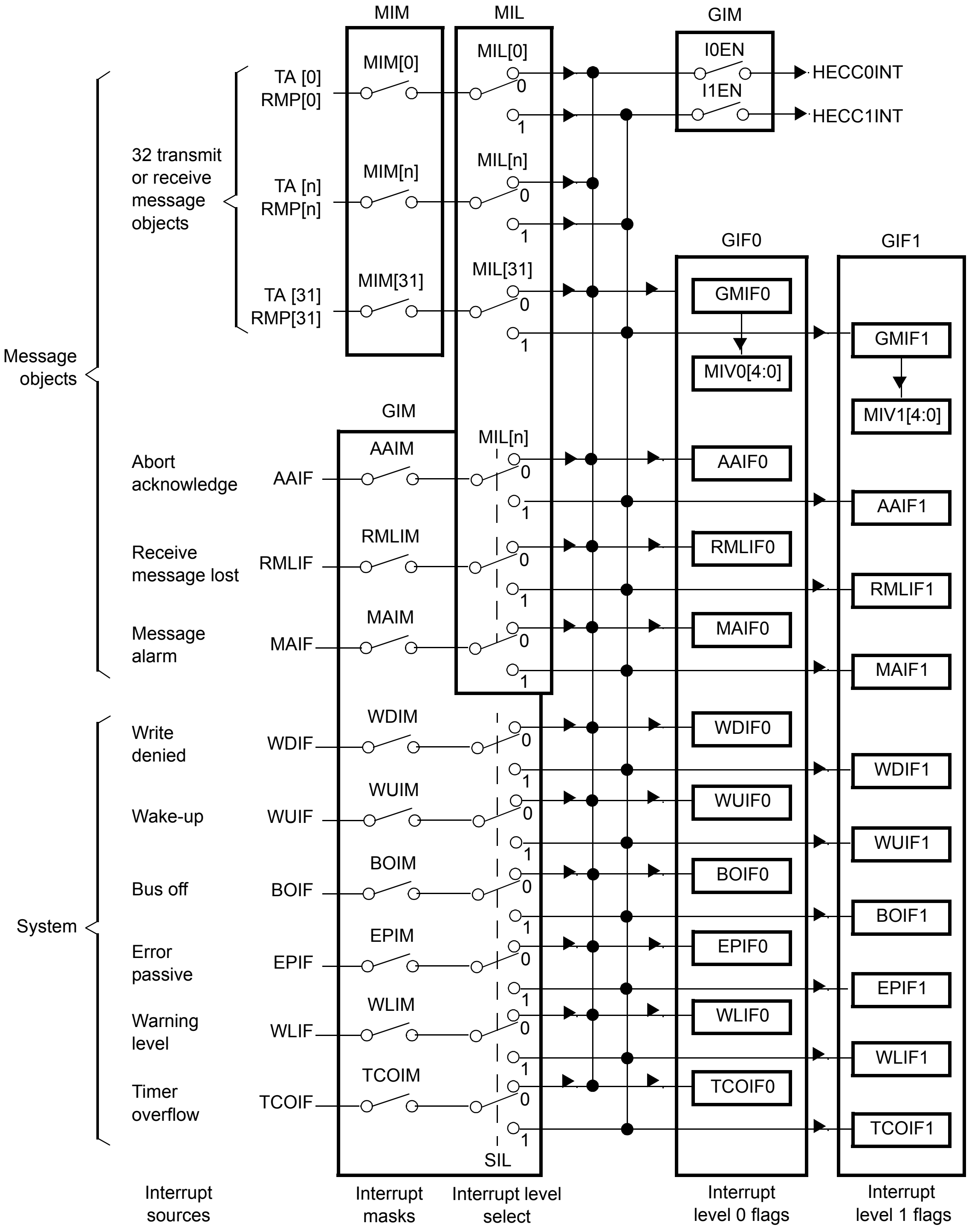


Figure 16. HECC Interrupts Scheme



7.1 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.

7.2 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.

8 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. Therefore, it is possible to have a local power down, where only the clock of the CAN module logic is disabled, or a global power-down mode when the clock for the complete chip is disabled.

8.1 Local Power Down

The local power-down mode is requested by writing a 1 to the PDR (MC.11) bit. When the module enters the local power-down mode, the status bit PDA (ES.3) is set.

During local power-down mode, the clock of the CAN base module is turned off. Only the wake-up logic is still active. 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. Thus, the module causes no error condition on the CAN bus line.

Note: Wakeup Delay From Standby Mode Due to Bus-Off Condition

If CAN goes into Standby mode when the *Bus-Off* condition has occurred, the CCR bit is set automatically upon wake-up. Then, in order for normal operation to resume, the device must wait for 128x11 recessive bits to occur and for the CCE bit to be set before the CPU can clear the CCR bit and wait for the CCE to go to zero.

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.

8.2 Global Power Down

Global power-down mode is requested by the system module (SYS LPM). When the module is able to enter the global power-down mode, all clocks to the CAN module are disabled.

If the module is transmitting a message when SYS LPM is asserted, the wake-up interrupt flag (WUIF0/1 (GIF0.12/ GIF1.12)) is asserted if enabled. The transmission is continued until a successful transmission, a lost arbitration, or an error condition on the CAN bus line occurs. Then, low-power mode is entered. Thus, the module causes no error condition on the CAN bus line.

During global power-down mode, a dominant signal on the CAN bus may generate a wake-up interrupt, thus enabling the CPU to exit global power-down mode. There is no internal filtering for the CAN bus line. The WUIF flag is set asynchronously to enable the assertion of an interrupt without any running clocks.

9 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.

9.1 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. (See Section 3.9.3 on pages 3-44.)

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

Note: Time Stamp Registers

The time stamp registers are available on the HECC only.

9.2 Local Network Time Register (LNT)

Figure 17 and Table 10 illustrate this register.

Figure 17. Local Network Time Register (LNT)

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x5C†	LNT.31:16															
	RWP-0‡															

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	LNT.15:0															
	RWP-0‡															

RWP = Read in all modes, write in privilege mode only, -*n* = Value after reset, x = indeterminate

† Relative address = (+) HECC_Offset
‡ HECC only, reserved in SCC

Table 10. Local Network Time Register (LNT) Field Descriptions

Bit	Name	Value	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.

9.3 Message Object Time Stamp Registers (MOTS)

Figure 18 and Table 11 illustrate this register.

Figure 18. Message Object Time Stamp Registers (MOTS)

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x100†	MOTS.31:16															
	RW-x‡															

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	MOTS.15:0															
	RW-x‡															

RW = Read/Write, -*n* = Value after reset, x = indeterminate

† Relative address = + HECC_Offset + (Object# x 4)
‡ HECC only, reserved in the SCC

Table 11. Message Object Time Stamp Registers (MOTS) Field Descriptions

Bit	Name	Value	Description
31–0	MOTS 31:0		Message Object Time Stamp Register. *HECC only, reserved in the SCC.* Value of the LNT when the message has been actually received or transmitted.

10 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.

Note: Time-Out Registers

These registers are available on the HECC only. The offset addresses are reserved in the SCC.

10.1 Message Object Time-Out Registers (MOTO)

Figure 19 and Table 12 illustrate this register.

Figure 19. Message Object Time-Out Registers (MOTO)

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x180†	MOTO.31:16															
	RW-x															

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	MOTO.15:0															
	RW-x															

RW = Read/Write, -*n* = Value after reset, x = indeterminate

† Relative address = + HECC_Offset + (Object # x 4)

Table 12. Message Object Time-Out Registers (MOTO) Field Descriptions

Bit	Name	Value	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.

10.2 Time Out Control-Register (TOC)

Figure 20 and Table 13 illustrate this register.

Figure 20. Time-Out Control Register (TOC)

Bits	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
0x60†	TOC.31:16															
	RW-0															
Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	TOC.15:0															
	RW-0															

RW = Read/Write, -*n* = Value after reset, x = indeterminate

† Relative address = + HECC_Offset

Table 13. Time-Out Control Register (TOC) Field Descriptions

Bit	Name	Value	Description
31–0	TOC 31:0		Time-Out Control Register.
		0	The time-out function is disabled. The TOS[*n*] flag is never set.
		1	The TOC[*n*] bit has to 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.

10.3 Time Out Status Register (TOS)

Figure 21 and Table 14 illustrate this register.

Figure 21. Time-Out Status Register (TOS)

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x64†	TOS.31:16															
	RC-0															

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	TOS.15:0															
	RC-0															

RC = Read/Clear, -*n* = Value after reset, x = indeterminate

† Relative address = + HECC_Offset

Table 14. Time-Out Status Register (TOS) Field Descriptions

Bit	Name	Value	Description
31–0	TOS 31:0		Time-Out Status Register.
		0	No time-out occurred or it is disabled for that mailbox.
		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.

11 CAN SCC/HECC Control Registers

The 470 SCC and HECC registers listed in Table 15 on page 48 are used by the CPU to configure and control the CAN controller and the message objects. All CAN registers support 8-, 16-, and 32-bit accesses.

Table 15. Combined SCC/HECC Registers

Offset		**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Address †	**Register**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
+ 00h[†]	CANME	ME.31:16															
		ME.15:0															
04h	CANMD	MD.31:16															
		MD.15:0															
08h	CANTRS	TRS.31:16															
		TRS.15:0															
0Ch	CANTRR	TRR.31:16															
		TRR.15:0															
10h	CANTA	TA.31:16															
		TA.15:0															
14h	CANAA	AA.31:16															
		AA.15:0															

Table 15. Combined SCC/HECC Registers (Continued)

<table>
<tr><th>Offset Address †</th><th>Register</th><th>31
15</th><th>30
14</th><th>29
13</th><th>28
12</th><th>27
11</th><th>26
10</th><th>25
9</th><th>24
8</th><th>23
7</th><th>22
6</th><th>21
5</th><th>20
4</th><th>19
3</th><th>18
2</th><th>17
1</th><th>16
0</th></tr>
<tr><td rowspan="2">18h</td><td rowspan="2">CANRMP</td><td colspan="16">RMP.31:16</td></tr>
<tr><td colspan="16">RMP.15:0</td></tr>
<tr><td rowspan="2">1Ch</td><td rowspan="2">CANRML</td><td colspan="16">RML.31:16</td></tr>
<tr><td colspan="16">RML.15:0</td></tr>
<tr><td rowspan="2">20h</td><td rowspan="2">CANRFP</td><td colspan="16">RFP.31:16</td></tr>
<tr><td colspan="16">RFP.15:0</td></tr>
<tr><td rowspan="2">24h</td><td rowspan="2">CANGAM‡</td><td>AMI</td><td colspan="2">Reserved</td><td colspan="11">GAM.28:18</td><td colspan="2">GAM.17:16</td></tr>
<tr><td colspan="16">GAM.15:0</td></tr>
<tr><td rowspan="2">28h</td><td rowspan="2">CANMC</td><td colspan="16">Reserved</td></tr>
<tr><td>LNTC</td><td>LNTM</td><td>SCM</td><td>CCR</td><td>PDR</td><td>DBO</td><td>WUBA</td><td>CDR</td><td>ABO</td><td>STM</td><td>SRES</td><td colspan="5">MBNR</td></tr>
<tr><td rowspan="2">2Ch</td><td rowspan="2">CANBTC</td><td colspan="8">Reserved</td><td colspan="8">BRP</td></tr>
<tr><td colspan="5">Reserved</td><td>ERM</td><td colspan="2">SJW</td><td>SAM</td><td colspan="4">TSEG1</td><td colspan="3">TSEG2</td></tr>
</table>

Table 15. Combined SCC/HECC Registers (Continued)

Offset Address †	Register	31 15	30 14	29 13	28 12	27 11	26 10	25 9	24 8	23 7	22 6	21 5	20 4	19 3	18 2	17 1	16 0
30h	CANES	Reserved							FE	BE	SA1	CRCE	SE	ACKE	BO	EP	EW
		Reserved										SMA	CCE	PDA	Res’rvd	RM	TM
34h	CANTEC	Reserved															
		Reserved								TEC							
38h	CANREC	Reserved															
		Reserved								REC							
3Ch	CANGIF0	Reserved														MAIF0	TCOF0
		GMIF0	AAIF0	WDIF0	WUIF0	RMLIF0	BOIF0	EPIF0	WLIF0	Reserved			MIV0				
40h	CANGIM	Reserved														MAIM	TCOIM
		Reserved	AAIM	WDIM	WUIM	RMLIM	BOIM	EPIM	WLIM	Reserved					SIL	I1EN	I0EN
44h	CANGIF1	Reserved														MAIF1	TCOF1
		GMIF1	AAIF1	WDIF1	WUIF1	RMLIF1	BOIF1	EPIF1	WLIF1	Reserved			MIV1				

Table 15. Combined SCC/HECC Registers (Continued)

Offset Address †	Register	31 15	30 14	29 13	28 12	27 11	26 10	25 9	24 8	23 7	22 6	21 5	20 4	19 3	18 2	17 1	16 0
48h	CANMIM	MIM.31:16															
		MIM.15:0															
4Ch	CANMIL	MIL.31:16															
		MIL.15:0															
50h	CANOPC	OPC.31:16															
		OPC.15:0															
54h	CANTIOC	Reserved															
		Reserved												TX FUNC	TXDIR	TXOUT	TXIN
58h	CANRIOC	Reserved															
		Reserved												RX FUNC	RXDIR	RX OUT	RXIN
5Ch	LNT §	LNT.31:16															
		LNT.15:0															

Table 15. Combined SCC/HECC Registers (Continued)

Offset Address †	Register	31 15	30 14	29 13	28 12	27 11	26 10	25 9	24 8	23 7	22 6	21 5	20 4	19 3	18 2	17 1	16 0
60h	TOC §	TOC.31:16															
		TOC.15:0															
64h	TOS §	TOS.31:16															
		TOS.15:0															
68h To 7Ch		Reserved															

† The actual addresses of the message objects are device-specific. See the specific device data sheet to verify the SCC module memory offset and RAM offset. Object# specifies the number of the message object.

‡ CANGAM register available in SCC module only. Reserved in HECC.

§ LNT, TOC, TOS registers available in HECC module only. Reserved in SCC

11.1 Mailbox Enable Register (CANME)

Each mailbox can be enabled or disabled. Figure 22 and Table 16 illustrate this register.

Figure 22. Mailbox Enable Register (CANME)

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x00[†]	ME.31:16															
	RW-0 ‡															

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	ME.15:0															
	RW-0															

RW = Read/Write, -*n* = Value after reset

[†] Relative address = + SCC/HECC_Offset
[‡] HECC only, reserved in the SCC

Table 16. Mailbox Enable Register (CANME) Field Descriptions

Bit	Name	Value	Description
31–0	ME.31:0		Mailbox Enable Register. After power-up, all bits in CANME are cleared. Disabled mailboxes may be used as additional memory for the CPU.
		0	The corresponding mailbox is disabled.
		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.

11.2 Mailbox Direction Register (CANMD)

Each mailbox can be configured as a transmit or a receive mailbox. Figure 23 and Table 17 illustrate this register.

Figure 23. Mailbox Direction Register (CANMD)

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x04†	MD.31:16															
	RW-0 ‡															

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	MD.15:0															
	RW-0															

RW = Read/Write, -*n* = Value after reset

† Relative address = + SCC/HECC_Offset
‡ HECC only, reserved in the SCC

Table 17. Mailbox Direction Register (CANMD) Field Descriptions

Bit	Name	Value	Description
31–0	MD.31:0		Mailbox Direction Register. After power-up, all bits in CANME are cleared.
		0	The corresponding mailbox is defined as a transmit mailbox.
		1	The corresponding mailbox is defined as a receive mailbox.

11.3 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. It is also set by the CAN module in case of a remote frame request. These bits are reset in case of a successful transmission or an aborted transmission (if requested). 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 CAN 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 24 and Table 18 illustrate this register.

Figure 24. Transmission Request Set Register (CANTRS)

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x08†	TRS.31:16															
	RS-0 ‡															

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	TRS.15:0															
	RS-0															

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

† Relative address = + SCC/HECC_Offset
‡ HECC only, reserved in the SCC

Table 18. Transmission Request Set Register (CANTRS) Field Descriptions

Bit	Name	Value	Description
31–0	TRS.31:0		Transmit Request Set Register. A successful transmission initiates a GMIF0/GMIF1 (GIF0.15/ GIF1.15) interrupt if enabled.
		0	No operation
		1	If the corresponding message object is configured as a transmit mailbox or remote request mailbox, the data or remote message of this mailbox will be transmitted.

11.4 Transmission Request Reset Register (CANTRR)

Setting the TRR[*n*] bit of the message object *n* causes a transmission request to be cancelled 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 in case of a successful transmission (normal operation) or in case of an aborted transmission due to a lost arbitration or an error condition detected on the CAN bus line. In case of an aborted transmission, the corresponding status bit (AA.31-0) is set and in case of a successful transmission, the status bit (TA.31-0) is set. The status of the transmission request reset can be read from the TRS.31-0 bit.

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

If a transmission reset is requested for a transmit mailbox with corresponding TRS[*n*] bit cleared, is requested for a disabled mailbox, or is requested for a receive mailbox, then the CAN clears the TRR[*n*] and sets the abort acknowledge flag AA[*n*] to respond to the request.

The bits in CANTRR are set by writing a 1 from the CPU. Figure 25 and Table 19 illustrate this register.

Figure 25. Transmission Request Reset Register (CANTRR) Field Descriptions

Bits	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
0x0C†	TRR.31:16															
	RS-0 ‡															

Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	TRR.15:0															
	RS-0															

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

† Relative address = + SCC/HECC_Offset

‡ HECC only, reserved in the SCC

Table 19. Transmission Request Reset Register (CANTRR) Field Descriptions

Bit	Name	Value	Description
31–0	TRR.31:0		Transmit Request Reset Register. A successful transmission initiates a GMIF0/GMIF1 (GIF0.15/ GIF1.15) interrupt if enabled.
		0	No operation
		1	Setting TRR[*n*] causes a transmission request to be cancelled if it was initiated by the corresponding TRS[*n*] bit and is not currently being processed. If the corresponding message is currently being processed, the bit is reset in case of a successful transmission (normal operation), or in case of an aborted transmission due to a lost arbitration or an error condition detected on the CAN bus line. If the transmission request reset bit TRR[*n*] is set and no TRS[*n*] bit is set, then the CAN resolves the situation by clearing the TRR[*n*] bit and setting the AA[*n*] flag.

11.5 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 will also clear the interrupt if an interrupt had 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. Figure 26 and Table 20 illustrate this register.

Figure 26. Transmission Acknowledge Register (CANTA) Field Descriptions

Bits	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
0x10†	TA.31:16															
	RC-0 ‡															

Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	TA.15:0															
	RC-0															

RC = Read/Clear, -*n* = Value after reset

† Relative address = + SCC/HECC_Offset
‡ HECC only, reserved in the SCC

Table 20. Transmission Acknowledge Register (CANTA) Field Descriptions

Bit	Name	Value	Description
31–0	TA.31:0		Transmit Acknowledge Register.
		0	The message is not sent.
		1	If the message of mailbox *n* is sent successfully, the bit *n* of this register is set.

11.6 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 zero 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 27 and Table 21 illustrate this register.

Figure 27. Abort Acknowledge Register (CANAA) Field Descriptions

Bits	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
0x14†	AA.31:16															
	RC-0 ‡															
Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	AA.15:0															
	RC-0															

RC = Read/Clear, -*n* = Value after reset

† Relative address = + SCC/HECC_Offset
‡ HECC only, reserved in the SCC

Table 21. Abort Acknowledge Register (CANAA) Field Descriptions

Bit	Name	Value	Description
31–0	TA.31:0		Abort Acknowledge Register.
		0	The transmission is not aborted.
		1	If the transmission of the message in mailbox *n* is aborted, the bit n of this register is set.

11.7 Receive Message Pending Register (CANRMP)

If mailbox *n* contains a received message, the bit RMP[*n*] of this register is set. These bits can only be reset by the CPU and set by the internal logic. A new incoming message will overwrite 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 28 and Table 22 illustrate this register.

Figure 28. Receive Message Pending Register (CANRMP) Field Descriptions

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x18†	RMP.31:16															
	RC-0 ‡															

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	RMP.15:0															
	RC-0															

RC = Read/Clear, -*n* = Value after reset

† Relative address = + SCC/HECC_Offset
‡ HECC only, reserved in the SCC

Table 22. Receive Message Pending Register (CANRMP) Field Descriptions

Bit	Name	Value	Description
31–0	RMP.31:0		Receive Message Pending Register.
		0	The mailbox does not contain a message.
		1	If mailbox *n* contains a received message, bit RMP[*n*] of this register is set.

11.8 Receive Message Lost Register (CANRML)

If an old message has been overwritten by a new one in message object *n*, bit RML[*n*] of this register is set. 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 29 and Table 23 illustrate this register.

Figure 29. Receive Message Lost Register (CANRML) Field Descriptions

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x1C†	RML.31:16															
	RC-0 ‡															

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	RML.15:0															
	RC-0															

R = Read, -*n* = Value after reset

† Relative address = + SCC/HECC_Offset
‡ HECC only, reserved in the SCC

Table 23. Receive Message Lost Register (CANRML) Field Descriptions

Bit	Name	Value	Description
31–0	RML.31:0		Receive Message Lost Register.
		0	No message was lost.
		1	An old unread message has been overwritten by a new one in that mailbox.

11.9 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 CPU has to initiate the reply frame transmission and has to reset the RFP[n] flag. If a mailbox configured in auto-answer mode (AAM=1, MD=0) receives a remote frame, the CAN module clears the RFP[n] flag after the reply frame has been successfully transmitted.

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 30 and Table 24 illustrate this register.

Figure 30. Remote Frame Pending Register (CANRFP) Field Descriptions

Bits	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
0x20†	RFP.31:16															
	RC-0 ‡															

Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	RFP.15:0															
	RC-0															

RC = Read/Clear, -*n* Value after reset

† Relative address = + SCC/HECC_Offset
‡ HECC only, reserved in the SCC

Table 24. Remote Frame Pending Register (CANRFP) Field Descriptions

Bit	Name	Value	Description
31–0	RFP.31:0		Remote Frame Pending Register.
		0	No remote frame request was received.
		1	A remote frame request was received by the module.

11.10 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 31 and Table 25 illustrate this register.

Figure 31. Global Acceptance Mask Register (CANGAM) Field Descriptions

Bits	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
0x24†	AMI	Reserved		GAM.28:18											GAM.17:16	
	RWI-0	R-0		RWI-0												

Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	GAM.15:0															
	RWI-0															

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

† Relative address = + SCC_Offset

Table 25. Global Acceptance Mask Register (CANGAM) Field Descriptions

Bit	Name	Value	Description
31	AMI		Acceptance Mask Identifier extension bit.
		0	The identifier extension bit stored in the mailbox determines which messages shall be received.
		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.
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.
		0	Received identifier bit value must match the corresponding identifier bit of the MID register.
		1	Accept a 0 or a 1 (don’t care) for the corresponding bit of the received identifier.

11.11 Master Control Register (CANMC)

This register is used to control the settings of the CAN module.

Note: Bits Protected in User Mode

Some bits of the CANMC register cannot be changed in user mode.

Figure 32 and Table 26 illustrate this register.

Figure 32. Master Control Register (CANMC) Field Descriptions

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x28†	Reserved															

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	LNTC	LNTM	SCM	CCR	PDR	DBO	WUBA	CDR	ABO	STM	SRES	MBNR				
	RWP0‡	SP-x‡	RWP0‡	RWP1	RWP-0	RWP-0	RWP-0	RW-0	RWP-0	RWP-0	S-0	RW-0‡	RW-0	RW-0	RW-0	RW-0

RWP = Read in all modes, write in privilege mode only, SP = Set in privilege mode only, -*n* = Value after reset, -x = Indeterminate

† Relative address = + SCC/HECC Offset
‡ HECC only, reserved in the SCC

Table 26. Master Control Register (CANMC) Field Descriptions

Bit	Name	Value	Description
31–16	Reserved		Reads are undefined and writes have no effect.
15	LNTC		Local Network Time Clear Bit. *HECC only, reserved in the SCC.* This bit is protected in user mode.
		0	The LNT register is not reset.
		1	The LNT register is reset to zero after a successful transmission or reception of mailbox 16.
14	LNTM		Local Network Time MSB Clear Bit. *HECC only, reserved in the SCC.* This bit is protected in user mode.
		0	The LNT register is not changed.
		1	The MSB (most significant bit) of the LNT register is reset to zero. The LNTM bit is reset after one clock cycle by the internal logic.

Table 26. Master Control Register (CANMC) Field Descriptions (Continued)

Bit	Name	Value	Description
13	SCM		SCC-Compatibility Mode. *HECC only, reserved in the SCC.* This bit is protected in user mode. If this bit is cleared (SCM = 0), the HECC behaves like the SCC, and only mailboxes 15 to 0 can be used. You must refer to the SCC specification to operate the HECC in this mode. Since this mode is selected by default, all software developed for the SCC runs without any change to the HECC module.
		0	The HECC is in SCC-compatibility mode.
		1	The HECC is in normal mode.
12	CCR		Change Configuration Request. This bit is protected in user mode.
		0	The CPU requests normal operation. This can only be done after the configuration register CANBTC is set to the allowed values.
		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 (see Section 3.11.12). This could also mean that the CAN module is in *Bus Off* state (see BO flag of CANES registe).
11	PDR		Power-Down-Mode Request. This bit is protected in user mode.
		0	The CAN module power-down mode is not requested (normal operation).
		1	The CAN module power-down mode is requested.
10	DBO		Data Byte Order. This bit selects the byte order of the message data field (see Section 3.5.3.7).This bit is protected in user mode.
		0	The data is received or transmitted MSB (most significant byte) first.
		1	The data is received or transmitted LSB (least significant byte) first.

Table 26. Master Control Register (CANMC) Field Descriptions (Continued)

Bit	Name	Value	Description
9	WUBA		Wake Up on Bus Activity. This bit is protected in user mode.
		0	The module leaves the power-down mode only after writing a 0 to the PDR bit.
		1	The module leaves the power-down mode after detecting any bus activity.
8	CRD		Change Data Field Request. This bit allows fast data message update.
		0	The CPU requests normal operation.
		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. Update data with CDR mechanism. **Note: When utilizing the CDR bit to update the data field of a mailbox, the SCC and the HECC behave differently. The SCC always considers new data after arbitration loss or bus error. Whereas the HECC tries to send the former data if the message was loaded into the internal transmit buffer before the data update. If the software wants to force the HECC to transmit updated data as soon as possible, it has to perform a dummy write to the TRS register (e.g., TRS=0x00000000) after the data update is finished (CDR bit cleared).**
7	ABO		Auto Bus On. This bit is protected in user mode.
		0	The CCR (MC.12) bit is set when *Bus-Off* state is detected. The CCR bit should be cleared before resuming any CAN communication. *Bus-Off* state is exited after 128 × 11 recessive bits.
		1	After *Bus-Off* state, the module will go back automatically into *Bus-On* state after 128 × 11 recessive bits. This bit is protected in user mode.

Table 26. Master Control Register (CANMC) Field Descriptions (Continued)

Bit	Name	Value	Description
6	STM		Self-Test Mode. This bit is protected in user mode.
		0	The module is in normal 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.
5	SRES		Software Reset. This bit can only be written to and is always read as zero.
		0	No effect
		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.=
4–0	MBNR 4:0		Mailbox Number. The bit MBNR.4 is for HECC only, and is reserved in the SCC. This is the number of the mailbox, for which the CPU requests a write access to the data field. This field is used in conjunction with the CDR bit.

11.12 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 6.1.

Figure 33 and Table 26 illustrate this register.

Figure 33. Bit-Timing Configuration Register (CANBTC)

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x2C†	Reserved								BRP							
	R-x								RWPI-0							

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

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

† Relative address = + SCC/HECC_Offset

Table 27. Bit-Timing Configuration Register (CANBTC) Field Descriptions

Bit	Name	Value	Description
31–24	Reserved		Reads are undefined and writes have no effect.
23–16	BRP.7:0		Time Quantum Prescaler. Bits 7:0 of this field specify the duration of a time quantum (TQ) in CAN module system clock units. The length of one TQ is defined by: $TQ = \frac{BRP}{ICLK}$ 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. $1 \leq BRP_{CALC} \leq 256$ BRP_{BTC} Value to be programmed into the CANBTC register. $0 \leq BRP_{BTC} \leq 255$
15–11	Reserved		Reads are undefined and writes have no effect.
10	ERM		Edge Resynchronization Mode. This bit selects the synchronization mode with the receive data stream on the CAN bus.
		0	The CAN module will resynchronize on the falling edge only.
		1	The CAN module will resynchronize on both rising and falling edges.

Table 27. Bit-Timing Configuration Register (CANBTC) Field Descriptions (Continued)

Bit	Name	Value	Description
9–8	SJW 1:0		Synchronization Jump Width. The parameter SJW indicates by how many units of TQ a bit is allowed to be lengthened or shortened when resynchronizing with the receive data stream on the CAN bus. The synchronization is performed either with the falling edge (ERM = 0) or with both edges (ERM = 1) of the bus signal. The synchronization jump width, SJW, is calculated as follows: $SJW = 1 + SJW.0 + (2 \times SJW.1)$ SJW is programmable from 1 to 4 TQ. The maximum value of SJW is determined by the minimum value of $TSEG2_{CALC}$ and 4TQ: $SJW_{CALC(max)} \leq \min[4\ TQ, TSEG2_{CALC}]$ The value programmed into the CANBTC register has to be decremented by 1, because of the 2-bit size of the binary SJW value (SJW.1:0): $SJW_{BTC} = SJW_{CALC} - 1$ SJW_{CALC} Result of the calculation. $1 \leq SJW_{CALC} \leq 4$ SJW_{BTC} Value to be programmed into the CANBTC register. $0 \leq SJW_{BTC} \leq 3$
7	SAM		Sample point setting. 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.
		0	The CAN module samples only once at the sampling point.
		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 ($BRP_{CALC} > 4$).

Table 27. Bit-Timing Configuration Register (CANBTC) Field Descriptions (Continued)

Bit	Name	Value	Description
6–3	TSEG1 3:0		Time Segment 1. This parameter specifies the length of the TSEG1 segment in TQ units. The value of TSEG1 is calculated as follows: *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. $TSEG1_{CALC}$ value is programmable in the range of 1 TQ to 16 TQ and has to fulfill the following timing rule: **TSEG1 must be greater than or equal to TSEG2 and IPT (for more details about IPT refer to Section 6.1).** 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. $2 \leq TSEG1_{CALC} \leq 16$ $TSEG1_{BTC}$ Value to be programmed into the CANBTC register. $1 \leq TSEG1_{BTC} \leq 15$

Table 27. Bit-Timing Configuration Register (CANBTC) Field Descriptions (Continued)

Bit	Name	Value	Description
2–0	TSEG2 2:0		Time Segment 2. 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 the following timing rule: **TSEG2 must be smaller than or equal to TSEG1 and must be greater or equal to IPT (for more details about IPT refer to Section 6.1).** 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

11.13 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. 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 34 and Table 27 illustrate this register.

Figure 34. Error and Status Register (CANES)

Bits	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
0x30†	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
Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	Reserved										SMA	CCE	PDA	Rsr'vd	RM	TM
	R-0										R-0	R-1	R-0	R-0	R-0	R-0

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

† Relative address = + SCC/HECC_Offset

Table 28. Error and Status Register (CANES) Field Descriptions

Bit	Name	Value	Description
31–25	Reserved		Reads are undefined and writes have no effect.
24	FE		Form Error Flag.
		0	No form error detected.
		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.
23	BE		Bit Error Flag.
		0	No bit error detected.
		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.

Table 28. Error and Status Register (CANES) Field Descriptions (Continued)

Bit	Name	Value	Description
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.
		0	The CAN module detected a recessive bit.
		1	The CAN module never detected a recessive bit.
21	CRCE		CRC Error.
		0	The CAN module never received a wrong CRC.
		1	The CAN module received a wrong CRC.
20	SE		Stuff Error.
		0	No stuff bit error occurred.
		1	A stuff bit error occurred.
19	ACKE		Acknowledge Error.
		0	All messages have been correctly acknowledged.
		1	The CAN module received no acknowledge.
18	BO		Bus-Off State. The CAN module is in *Bus-Off* state.
		0	Normal operation
		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. *Bus Off* state is exited automatically after 128x11 recessive bits. After leaving *Bus Off,* the error counters are cleared. When ABO=0 then the CCR bit should be cleared before resuming any CAN communication. (MC.7) bit.
17	EP		Error Passive State.
		0	The CAN module is in error-active mode.
		1	The CAN module is in error-passive mode.

Table 28. Error and Status Register (CANES) Field Descriptions (Continued)

Bit	Name	Value	Description
16	EW		Warning status.
		0	Both error counters (CANREC and CANTEC) are less than 96.
		1	One of the two error counters (CANREC or CANTEC) has reached the warning level of 96.
15–6	Reserved		Reads are undefined and writes have no effect.
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.
		0	The module is not in suspend mode.
		1	The module has entered suspend mode.
4	CCE		Change Configuration Enable. This bit displays the configuration access right. (See Section 3.6) This bit is set after a latency of one clock cycle up to the length of one frame.
		0	The CPU is denied write access to the configuration registers.
		1	The CPU has write access to the configuration registers.
3	PDA		Power Down Mode Acknowledge.
		0	Normal operation
		1	The CAN module has entered the power-down mode.
2	Reserved		Reads are undefined and writes have no effect.
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.
		0	The CAN protocol kernel is not receiving a message.
		1	The CAN protocol kernel is receiving a message.

Table 28. Error and Status Register (CANES) Field Descriptions (Continued)

Bit	Name	Value	Description
0	TM		Transmit mode. The CPK is in transmit mode. This bit reflects what the CPK is actually doing regardless of mailbox configuration.
		0	The CAN protocol kernel is not transmitting a message.
		1	The CAN protocol kernel is transmitting a message.

11.14 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 35 and Figure 36 illustrate this register.

Figure 35. Transmit Error Counter Register (CANTEC)

Bits	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
0x34†	Reserved															
	R-x															
Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	Reserved								TEC							
	R-x								R-0							

R = Read, *-n* = Value after reset

† Relative address = + SCC/HECC_Offset

Figure 36. Receive Error Counter Register (CANREC)

Bits	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
0x38†	Reserved															
	R-x															
Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	Reserved								REC							
	R-x								R-0							

R = Read, W = Write, S = Set, C = Clear, U = Undefined, *-n:* Value after reset

† Relative address = + SCC/HECC_Offset

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.

11.15 Global Interrupt Flag Registers (CANGIF0/CANGIF1)

These registers allow the CPU to identify the interrupt source. Figure 37, Figure 38, and Table 29 illustrate this register.

Figure 37. Global Interrupt Flag 0 Register (CANGIF0)

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x3C†	Reserved														MAIF0	TCOIF0
	R-x														RC-0‡	RC-0‡

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	GMIF0	AAIF0	WDIF0	WUIF0	RMLIF0	BOIF0	EPIF0	WLIF0	Reserved			MIV0.4	MIV0.3:0			
	R-0	RC-0	RC-0	RC-0	RC-0	RC-0	RC-0	RC-0	R-0			R-0‡	R-0			

RC = Read/Clear, *-n* = Value after reset

Figure 38. Global Interrupt Flag 1 Register (CANGIF1)

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x44†	Reserved														MAIF1	TCOIF1
	R-x														RC-0‡	RC-0‡

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	GMIF1	AAIF1	WDIF1	WUIF1	RMLIF1	BOIF1	EPIF1	WLIF1	Reserved			MIV1.4	MIV1.3:0			
	R-0	RC-0	RC-0	RC-0	RC-0	RC-0	RC-0	RC-0	R-0			R-0‡	R-0			

RC = Read/Clear, *-n* = Value after reset

† Relative address = + SCC/HECC_Offset
‡ HECC only, reserved in the SCC

Table 29. Global Interrupt Flag Registers (CANGIF0/CANGIF1) Field Descriptions

Bit	Name	Value	Description
31–18	Reserved		Reads are undefined and writes have no effect.
17	MAIF0/MAIF1		Message Alarm Flag. *HECC only, reserved in the SCC.*
		0	No time out for the mailboxes occurred.
		1	One of the mailboxes did not transmit or receive a message within the specified time frame.

Table 29. Global Interrupt Flag Registers (CANGIF0/CANGIF1) Field Descriptions

Bit	Name	Value	Description
16	TCOIF0/ TCOIF1		Local Network Time Counter Overflow Flag. *HECC only, reserved in the SCC.*
		0	The MSB of the LNT register is 0.
		1	The MSB of the LNT register is 1.
15	GMIF0/GMIF1		Global Mailbox Interrupt Flag.
		0	No message has been transmitted or received.
		1	One of the mailboxes transmitted or received a message successfully.
14	AAIF0/AAIF1		Abort-Acknowledge Interrupt Flag.
		0	No transmission has been aborted.
		1	A transmission request has been aborted.
13	WDIF0/WDIF1		Write-Denied Interrupt Flag.
		0	The CPU’s write access to the mailbox was successful.
		1	The CPU’s write access to a mailbox was not successful.
12	WUIF0/WUIF1		Wake-Up Interrupt Flag.
		0	The module is still in sleep mode or normal operation
		1	During local power down, this flag indicates that the module has left sleep mode. During global power-down, this flag indicates that there is activity on the CAN bus lines. This flag is also set when a global power- down is requested by the CPU, but the CAN module is not ready to enter power-down mode yet.
11	RMLIF0/ RMLIF1		Receive-Message-Lost Interrupt Flag.
		0	No message has been lost.
		1	At least for one of the receive mailboxes, an overflow condition has occurred.
10	BOIF0/BOIF1		Bus-Off Interrupt Flag.
		0	The CAN module is still in *Bus-On* mode.
		1	The CAN module has entered *Bus-Off* mode.

Table 29. Global Interrupt Flag Registers (CANGIF0/CANGIF1) Field Descriptions

Bit	Name	Value	Description
9	EPIF0/EPIF1		Error-Passive Interrupt Flag.
		0	The CAN module is not in error-passive mode.
		1	The CAN module has entered error-passive mode.
8	WLIF0/WLIF1		Warning Level Interrupt Flag.
		0	None of the error counters has reached the warning level.
		1	At least one 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. This vector indicates the number of the message object that activated the global mailbox interrupt flag (GMIF0/GMIF1 (GIF0.15/ GIF1.15)). It keeps that vector until the appropriate TA[*n*] bit or RMP[*n*] 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 HECC. In the SCC or in SCC-compatible mode, mailbox 15 has the highest priority. When the GMIF0/GMIF1 flag is zero, the corresponding MIV0/ MIV1 vector is undefined. MIV0.4 and MIV1.4 are not available in the SCC. **Note: Bit 4 (MIVO.4 and MIV1.4), as previously described, applies to the HECC only. These bits are reserved and read as 0 in the SCC.**

11.16 Global Interrupt Mask Register (CANGIM)

This register allows the various interrupts to be enabled. If a bit is set, the corresponding interrupt is enabled.

This register is write-protected in user mode. Figure 39 and Table 30 illustrate this register.

Figure 39. Global Interrupt Mask Register (CANGIM)

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x40†	Reserved														MAIM	TCOIM
	R-x														RW-0‡	RW-0‡

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Rsr’vd	AAIM	WDIM	WUIM	RMLIM	BOIM	EPIM	WLIM	Reserved					SIL	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

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

† Relative address = + SCC/HECC_Offset
‡ HECC only, reserved in the SCC

Table 30. Global Interrupt Mask Register (CANGIM) Field Descriptions

Bit	Name	Value	Description
31–18	Reserved		Reads are undefined and writes have no effect.
17	MAIM		Message Alarm Interrupt Mask. *HECC only, reserved in the SCC.*
		0	MAIF interrupt is disabled.
		1	MAIF interrupt is enabled.
16	TCOIM		Local Network Time Counter Overflow Interrupt Mask. *HECC only, reserved in the SCC.*
		0	TCOIF interrupt is disabled.
		1	TCOIF interrupt is enabled.
15	Reserved		Reads are undefined and writes have no effect.

Table 30. Global Interrupt Mask Register (CANGIM) Field Descriptions (Continued)

Bit	Name	Value	Description
14	AAIM		Abort Acknowledge Interrupt Mask.
		0	AAIF interrupt is disabled.
		1	AAIF interrupt is enabled.
13	WDIM		Write Denied Interrupt Mask.
		0	WDIF interrupt is disabled.
		1	WDIF interrupt is enabled.
12	WUIM		Wake-Up Interrupt Mask.
		0	WUIF interrupt is disabled.
		1	WUIF interrupt is enabled.
11	RMLIM		Receive Message Lost Interrupt Mask.
		0	RMLIF interrupt is disabled.
		1	RMLIF interrupt is enabled.
10	BOIM		Bus Off Interrupt Mask.
		0	BOIF interrupt is disabled.
		1	BOIF interrupt is enabled.
9	EPIM		Error Passive Interrupt Mask.
		0	EPIF interrupt is disabled.
		1	EPIF interrupt is enabled.
8	WLIM		Warning Level Interrupt Mask.
		0	WLIF interrupt is disabled.
		1	WLIF interrupt is enabled.
7–3	Reserved		Reads are undefined and writes have no effect.

Table 30. Global Interrupt Mask Register (CANGIM) Field Descriptions (Continued)

Bit	Name	Value	Description
2	SIL		System Interrupt Level. Define the interrupt level for TCOIF, WDIF, WUIF, BOIF, EPIF, and WLIF.
		0	All global interrupts are mapped to the HECC0INT/SCC0INT interrupt line.
		1	All system interrupts are mapped to the HECC1INT/SCC1INT interrupt line.
1	I1EN		Interrupt Line 1 Enable.
		0	The HECC1INT/SCC1INT interrupt line is globally disabled.
		1	This bit globally enables all interrupts for the HECC1INT/SCC1INT interrupt line if the corresponding masks are set.
0	I0EN		Interrupt Line 0 Enable.
		0	The HECC0INT/SCC0INT interrupt line is globally disabled.
		1	This bit globally enables all interrupts for the HECC0INT/SCC0INT interrupt line if the corresponding masks are set.

11.17 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. After power up, all interrupt mask bits are cleared and all interrupts are disabled.

This register is write-protected in user mode. Figure 40 and Table 31 illustrate this register.

Figure 40. Mailbox Interrupt Mask Register (CANMIM).

Bits	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
0x48[†]	MIM.31:16															
	RWP-0‡															
Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	MIM.15:0															
	RWP-0															

RWP = Read in all modes, write in privilege mode only, -*n* = Value after reset

† Relative address = + SCC/HECC_Offset
‡ HECC only, reserved in the SCC

Table 31. Mailbox Interrupt Mask Register (CANMIM) Field Descriptions

Bit	Name	Value	Description
31–0	MIM 31:0		Mailbox Interrupt Mask. These bits allow any mailbox interrupt to be masked individually.
		0	Mailbox interrupt is disabled.
		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).

11.18 Mailbox Interrupt Level Register (CANMIL)

Each of the message objects 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 HECC0INT/SCC0INT (MIL[*n*] = 0) or on line HECC1INT/SCC1INT (MIL[*n*] = 1). This applies also to the AAIF0/ AAIF1 (GIF0.14/GIF1.14) and the RMLIF0/RMLFI1 (GIF0.11/GIF1.11) flags. Figure 41 and Table 32 illustrate this register.

Figure 41. Mailbox Interrupt Level Register (CANMIL)

Bits	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
0x4C†	MIL.31:16															
	RW-0‡															

Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	MIL.15:0															
	RW-0															

RW = Read/Write, -*n* = Value after reset

† Relative address = + SCC/HECC_Offset
‡ HECC only, reserved in the SCC

Table 32. Mailbox Interrupt Level Register (CANMIL) Field Descriptions

Bit	Name	Value	Description
31–0	MIL 31:0		Mailbox Interrupt Level. These bits allow any mailbox interrupt level to be selected individually.
		0	The mailbox interrupt is generated on interrupt line 0.
		1	The mailbox interrupt is generated on interrupt line 1.

11.19 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*]. Figure 42 and Table 33 illustrate this register.

Figure 42. Overwrite Protection Control Register (CANOPC)

Bits	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
0x50†	OPC.31:16															
	RW-0 ‡															
Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	OPC.15:0															
	RW-0															

RW = Read/Write, -*n* = Value after reset

† Relative address = + SCC/HECC_Offset
‡ HECC only, reserved in the SCC

Table 33. Overwrite Protection Control Register (CANOPC) Field Descriptions

Bit	Name	Value	Description
31–0	OPC 31:0		Overwrite Protection Control Register.
		0	If the bit OPC[*n*] is not set, the old message may be overwritten by a new one.
		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.

11.20 CAN I/O Control Registers (CANTIOC, CANRIOC)

The CANTX and CANRX pins may be configured as normal I/O pins if the HECC/SCC is not used. This is done using the CANTIOC and CANRIOC registers. Figure 43 and Table 34 illustrate the CANTIOC register, and Figure 44 and Table 35 illustrate the CANRIOC register.

Figure 43. CAN I/O Control Register (CANTIOC)

Bits	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
0x54†	Reserved															
	R-x															

Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	Reserved								Reserved				TX FUNC	TXDI R	TXOU T	TXIN
	R-x								R-0				RWP-0	RW-0	RW-0	R-x

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

† Relative address = + SCC/HECC_Offset

Table 34. CAN I/O Control Register (CANTIOC) Field Descriptions

Bit	Name	Value	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. **Note: 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.**
		0	The CANTX pin is used as a normal I/O pin; the CAN protocol kernel (CPK) output is unconnected.
		1	The CANTX pin is used for the CAN transmit functions.
2	TXDIR		Transmit Pin Direction Flag.
		0	The CANTX pin is used as an input pin if CANTX is configured as an I/O pin (TXFUNC = 0).
		1	The CANTX pin is used as an output pin if CANTX is configured as an I/O pin (TXFUNC = 0).

Table 34. CAN I/O Control Register (CANTIOC) Field Descriptions

Bit	Name	Value	Description
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 on the CANTX pin.
		0	Logic 0 is present on pin.
		1	Logic 1 is present on pin.

Figure 44. RX I/O Control Register (CANRIOC)

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x58†	Reserved															
	R-x															

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved								Reserved				RXFUNC	RXDIR	RXOUT	RXIN
	R-x								R-0				RWP-0	RW-0	RW-0	R-x

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

† Relative address = + SCC/HECC_Offset

Table 35. RX I/O Control Register (CANRIOC) Field Descriptions

Bit	Name	Value	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. **Note: When the CANRX pin is configured for CAN functions, the flags RXDIR and RXOUT have no meaning and can be set either to 0 or 1. The RXIN flag always shows the actual value of the CANRX pin.**
		0	The CANRX pin is used as a normal I/O pin. The RX signal is still internally connected to the CPK.
		1	The CANRX pin is used for the CAN receive functions.

Table 35. RX I/O Control Register (CANRIOC) Field Descriptions

Bit	Name	Value	Description
2	RXDIR		Receive Pin Direction Flag.
		0	The CANRX pin is used as an input pin if CANRX is configured as an I/O pin (RXFUNC = 0).
		1	The CANRX pin is used as an output 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.
		0	Logic 0 is present on pin
		1	Logic 1 is present on pin.

12 CAN Operation Examples

This section provides examples on how to use the HECC and the SCC in specific applications.

12.1 Configuration of SCC or HECC

The following steps must be performed to configure the SCC/HECC 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.

Note: In the case that the *Bus-Off* condition occurs after the ABO=0 and the CCR=0, the CCR bit is set automatically by the CAN module; after the occurrence of 128x11 recessive bits, the CCE will be set. The CPU must then clear the CCR bit and wait for the CCE to be cleared to resume normal operation.

This completes the setup for the basic functionality.

12.2 Transmit Mailbox

12.2.1 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.)

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.

12.2.2 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.

12.3 Receive Mailbox

12.3.1 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) For HECC only: If the AME bit is set to 1, the corresponding acceptance mask must be programmed.

   For the SCC: The acceptance masks must be programmed during the initialization phase. (See Section 3.12.1 on page 3-86.) For example:

   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.

12.3.2 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.

12.3.3 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.

12.4 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.

12.4.1 Requesting Data From Another Node

To request data from another node, the object is configured as receive mailbox. (See Section 3.12.3.1 on page 3-89.) 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) Now, 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) Now read the received message as explained in Section 3.12.3.2 on page 3-91.

12.4.2 Answering a Remote Request

1) The object needs to be configured as a transmit mailbox, as explained in Section 3.12.2.1 on page 3-88 steps 1 to 4.

2) The auto answer mode (AAM) (MID.29) bit must be set in the MID register before the mailbox is enabled.

   MID(1) = 0x35ac0000

3) The data field must be updated.

   MDL(1) = xxxx0000h

4) Now the mailbox can be enabled by setting the ME flag to 1.

   ME.1 = 1

5) 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.

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

   Wait for TA.1 = 1

12.4.3 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) to enable the object.

   Write MC = 0x00000000

12.5 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.

12.5.1 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.

12.5.2 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.