TMS320DM643x DMP High-End CAN Controller

User's Guide

Literature Number: SPRU981A
May 2007

Texas Instruments

Contents

List of Figures

List of Tables

Read This First

About This Manual

This document describes the high-end controller area network (HECC) peripheral in the TMS320DM643x Digital Media Processor (DMP).

Notational Conventions

This document uses the following conventions.

- Hexadecimal numbers are shown with the suffix h. For example, the following number is 40 hexadecimal (decimal 64): 40h.
- Registers in this document are shown in figures and described in tables.
  - Each register figure shows a rectangle divided into fields that represent the fields of the register. Each field is labeled with its bit name, its beginning and ending bit numbers above, and its read/write properties below. A legend explains the notation used for the properties.
  - Reserved bits in a register figure designate a bit that is used for future device expansion.

Related Documentation From Texas Instruments

The following documents describe the TMS320DM643x Digital Media Processor (DMP). Copies of these documents are available on the Internet at www.ti.com. *Tip:* Enter the literature number in the search box provided at www.ti.com.

The current documentation that describes the DM643x DMP, related peripherals, and other technical collateral, is available in the C6000 DSP product folder at: www.ti.com/c6000.

SPRU978 — ***TMS320DM643x DMP DSP Subsystem Reference Guide.*** Describes the digital signal processor (DSP) subsystem in the TMS320DM643x Digital Media Processor (DMP).

SPRU983 — ***TMS320DM643x DMP Peripherals Overview Reference Guide.*** Provides an overview and briefly describes the peripherals available on the TMS320DM643x Digital Media Processor (DMP).

SPRAA84 — ***TMS320C64x to TMS320C64x+ CPU Migration Guide.*** Describes migrating from the Texas Instruments TMS320C64x digital signal processor (DSP) to the TMS320C64x+ DSP. The objective of this document is to indicate differences between the two cores. Functionality in the devices that is identical is not included.

SPRU732 — ***TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide.*** Describes the CPU architecture, pipeline, instruction set, and interrupts for the TMS320C64x and TMS320C64x+ digital signal processors (DSPs) of the TMS320C6000 DSP family. The C64x/C64x+ DSP generation comprises fixed-point devices in the C6000 DSP platform. The C64x+ DSP is an enhancement of the C64x DSP with added functionality and an expanded instruction set.

SPRU871 — ***TMS320C64x+ DSP Megamodule Reference Guide.*** Describes the TMS320C64x+ digital signal processor (DSP) megamodule. Included is a discussion on the internal direct memory access (IDMA) controller, the interrupt controller, the power-down controller, memory protection, bandwidth management, and the memory and cache.

High-End CAN Controller

1 Introduction

This document describes the high-end controller area network (HECC) peripheral in the TMS320DM643x Digital Media Processor (DMP).

1.1 Overview

The controller area network (CAN) 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. The CAN controller uses established protocol to communicate serially with other controllers in harsh environments. This document describes the CAN controllers: the standard CAN controller (SCC) and the high-end CAN controller (HECC). Unless otherwise stated, all information is related to both the SCC and HECC.

1.2 Features

The CAN controller consists of the following features (see Table 1).

- Common CAN protocol kernel (CPK) to perform protocol tasks
- Standard CAN controller (SCC) for standard CAN applications:
  - 16 message objects
  - 3 receive identifier masks
- High-end CAN controller (HECC) for complex applications:
  - 32 message objects
  - 32 receive identifier masks

Table 1. SCC Features vs. HECC Features

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

1.2.1 CAN Protocol Processor Features

The CAN protocol kernel (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
- 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 error
  - Readable error counters
- Self-test mode
  - Operate in a loop-back mode (receiving its own message)
  - Dummy acknowledge

1.2.2 SCC Features

The SCC features are:

- All the CAN protocol kernel (CPK) features: Full implementation of CAN protocol, version 2.0B
- 16 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
- Dual clock support to avoid CAN bit-timing jitter

1.2.3 HECC Features

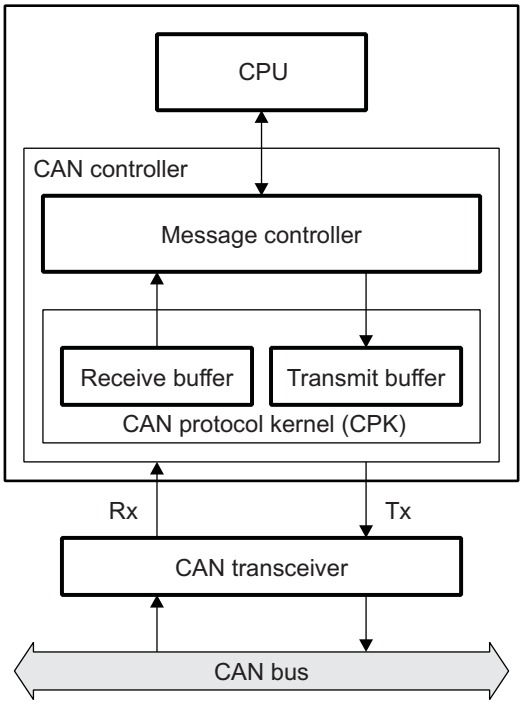
The HECC features are:

- All the CAN protocol kernel (CPK) features: Full implementation of CAN protocol, version 2.0B
- 32 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
- Dual clock support to avoid CAN bit-timing jitter

1.3 Functional Block Diagram

A block diagram of the HECC is shown in Figure 1.

Figure 1. HECC Block Diagram



1.4 Industry Standard Compatibility Statement

The CAN protocol kernel (CPK) performs the basic CAN protocol specification 2.0B.

1.5 Terminology Used in This Document

The following is a brief explanation of some terms used in this document:

Term	Meaning
CAN	controller area network
HECC	high-end CAN controller
SCC	standard CAN controller

2 Architecture

This section describes the architecture of the controller area network (CAN). The 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.

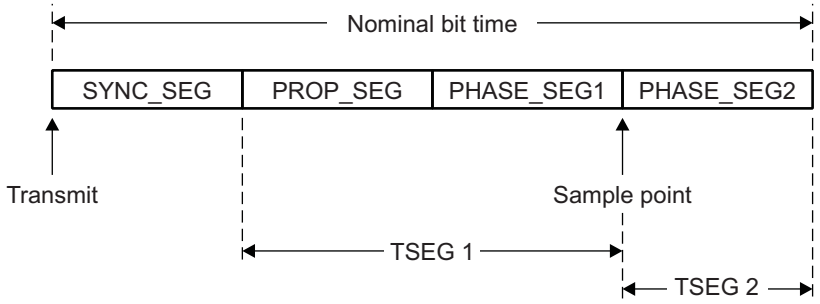
2.1 Clock Control

2.1.1 CAN Bit-Timing Configuration

As shown in Figure 2, 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 1-8 time quanta (TQ).
- **PHASE_SEG1:** this phase is used to compensate for positive edge phase error. This segment is programmable from 1-8 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 2-8 time quanta (TQ) and can be shortened by resynchronization.

Figure 2. 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, TSEG2, and BRP in the bit-timing configuration register (CANBTC). BRP is the binary value of BRP + 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: Special Case of Baud Rate Prescaler Value $BRP_{CALC} = 1$

For the special case of a baud rate prescaler value of $BRP_{CALC} = 1$, 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 ($BRP_{CALC} = 1$) is not allowed.

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

2.1.2 CAN Bit Rate Calculation

The CAN module input clock is sourced by the bypass clock of the device. See the device-specific data manual for more information. The bit rate is calculated as follows (in bits per second):

$$\text{Bitrate} = \frac{\text{CAN module system clock}}{\text{BRP x BitTime}}$$

Where *BitTime* is the number of time quanta (TQ) per bit. *BitTime* is defined as:

BitTime = TSEG1 + TSEG2 + 1

CAN module system clock is the CAN module system clock frequency.

BRP is the binary value of BRP + 1.

With CAN module system clock = 16 MHZ, if a bit rate of 1 Mbps is required, the bit-timing parameters should be programmed as:

$$\text{BRP} = 2 \rightarrow \text{TQ} = \frac{2}{16\ \text{MHz}} = 1/8\ \mu\text{s}$$

TSEG1 = 4 TQ, TSEG2 = 3 TQ $\rightarrow$ BitTime = 8 TQ, Bitrate = 1 Mbps

With this setting, a three-fold sampling of the bus is not possible; the SAM bit must be cleared to 0. Since the SJW bit is not allowed to be greater than TSEG2, it is set to 3 TQ (SJW = 3).

Bit-timing configuration register (CANBTC) = 0001 021Ah

2.2 Memory Map

This section discusses the memory map of the HECC and the HECC when used in SCC mode (in this document, referred to as the SCC memory-map). Access to the reserved locations on the SCC and HECC may hang the device. Software must not access the reserved locations of the SCC and HECC.

2.2.1 SCC Memory-Map

The SCC module has two different offset addresses mapped in the 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 be accessed 8-bit, 16-bit, and 32-bit wide.
- The second offset address, Mailbox_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.

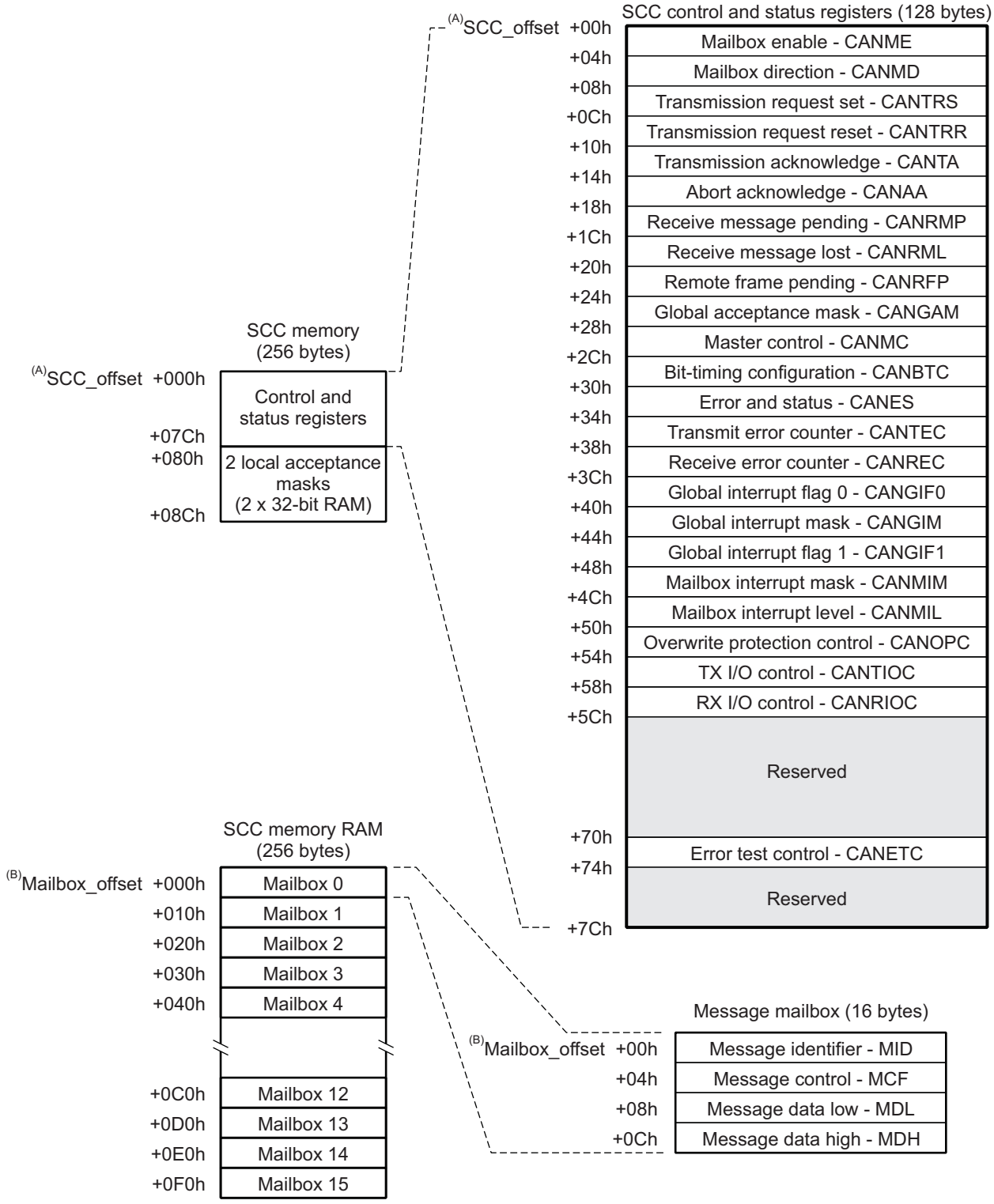
The message storage is implemented by a RAM that can be addressed by the CAN controller or the CPU. The CPU controls the CAN controller by modifying the various mailboxes in the RAM or the additional registers. The contents of the various storage elements are used to perform acceptance filtering, message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

The mailbox module in the SCC provides 16 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. Figure 3 shows the SCC memory-map.

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.

Figure 3. SCC Memory-Map



A The SCC_offset address location is referenced in the configuration memory-map summary of the data manual as the HECC Control Region.

B The Mailbox_offset address location is referenced in the configuration memory-map summary of the data manual as the HECC RAM Region.
Mailbox_offset = Mailbox_offset + (Mailbox# × 10h).

2.2.2 HECC Memory-Map

The HECC module has two different offset addresses mapped in the memory:

- The first offset address, HECC_offset, is used to access the control register and the status register. The access to the control and status registers (memory range: HECC_offset ... HECC_offset + 07Ch) is 8-bit, 16-bit, and 32-bit wide. The HECC control and status registers, shown in Figure 4, uses 128 bytes of address space.
- The second offset address, Mailbox_offset, is used to access the mailboxes. This memory range can be accessed 8-bit, 16-bit, and 32-bit wide. The acceptance mask, time-stamp, and time-out of the message objects are implemented in a control RAM block, which has a base address of Mailbox_offset + 1000h. This memory range can also be accessed 8-bit, 16-bit, and 32-bit wide.

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.

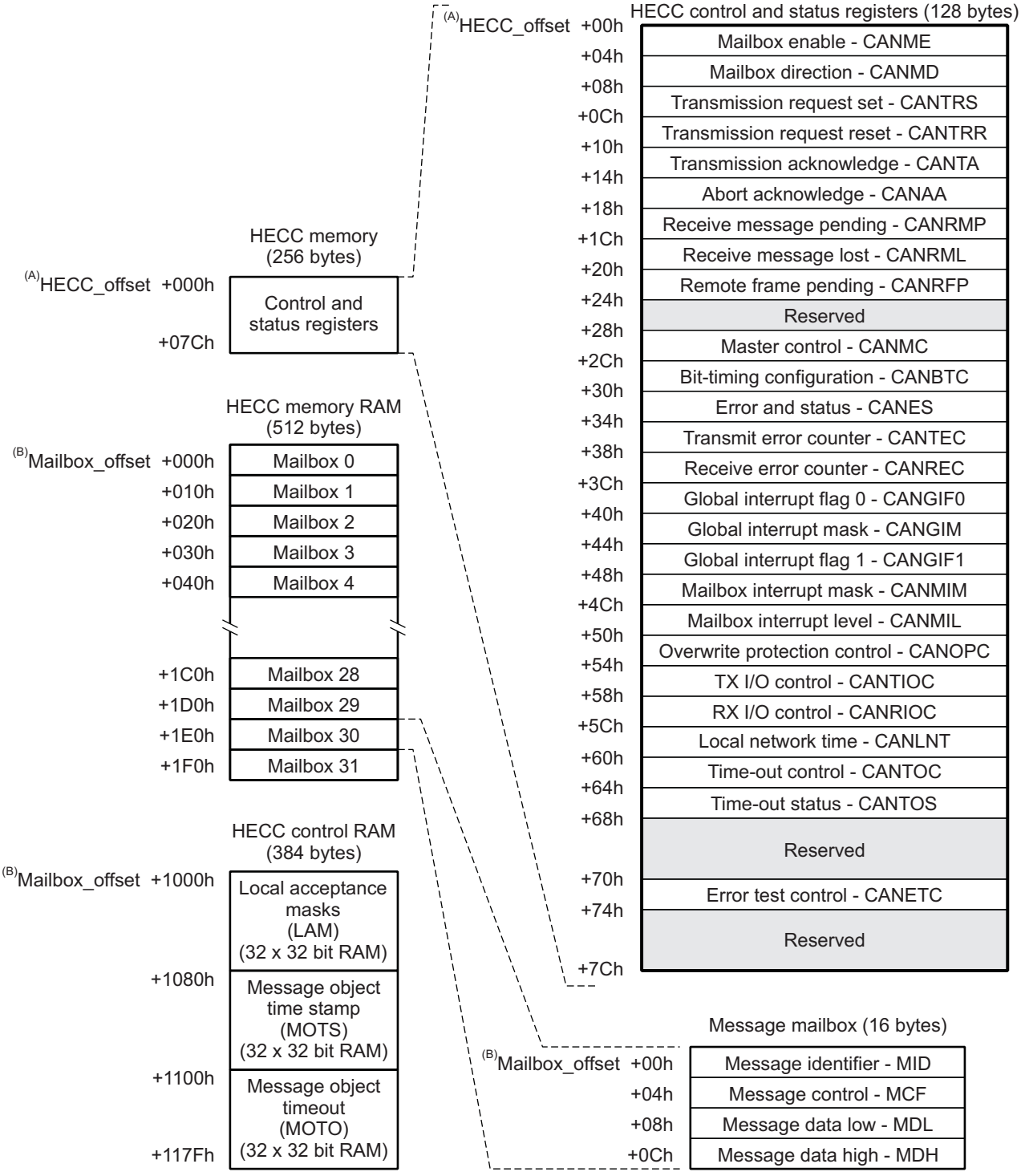
The mailbox module in the HECC provides 32 message mailboxes of 8-byte data length, a 29-bit identifier, and several control bits. Each mailbox can be configured as either transmit or receive. In the HECC, each mailbox has an individual acceptance mask.

When the HECC is used in SCC mode, refer to the SCC memory map in Figure 3.

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 obtained by disabling the corresponding mailbox or by disabling the corresponding functions.

Figure 4. HECC Memory-Map



A The HECC_offset address location is referenced in the configuration memory-map summary of the data manual as the HECC Control Region.

B The Mailbox_offset address location is referenced in the configuration memory-map summary of the data manual as the HECC RAM Region.
Mailbox_offset = Mailbox_offset + (Mailbox# × 10h).

2.3 Signal Descriptions

The CANTX and CANRX pins may be configured to be functional HECC/SCC pins. This is done by programming the transmit I/O control register (CANTIOC) and the receive I/O control register (CANRIOC), accordingly. The HECC signals are described in Table 2.

Table 2. HECC Signal Descriptions

Pin	Type	Description
CANRX	I/O	CAN receive signal
CANTX	I/O	CAN transmit signal

2.4 Pin Multiplexing

On the DM643x DMP extensive pin multiplexing is used to accommodate the largest number of peripheral functions in the smallest possible package. Pin multiplexing is controlled using a combination of hardware configuration at device reset and software programmable register settings. Refer to the device-specific data manual to determine how pin multiplexing affects the SCC/HECC.

2.5 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:

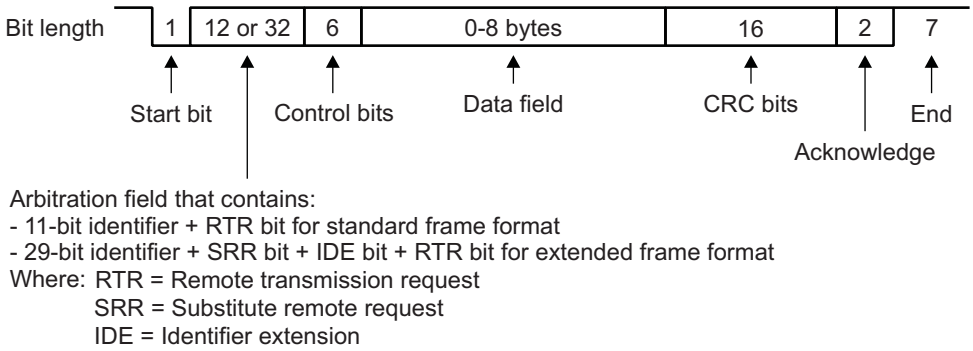
- standard frames with an 11-bit identifier
- extended frames with a 29-bit identifier

CAN standard data frames contain 44-108 bits and CAN extended data frames contain 64-28 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 131 bits for a standard frame and 156 bits for an extended frame.

Bit fields within the data frame, shown in Figure 5, 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 5. CAN Data Frame



2.6 CAN Controller Overview

The CAN controllers provide the CPU with full functionality of the CAN protocol, version 2.0B. The CAN controller minimizes the CPU 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 1, is composed of a CAN protocol kernel (CPK) and a message controller.

- 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.

2.6.1 Standard CAN Controller (SCC) Overview

The SCC (Figure 6) is a standard CAN controller with an internal 32-bit architecture, upwardly-compatible with the HECC. The SCC consists of the:

- the CAN protocol kernel (CPK)
- the message controller:
  - 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

After the CPK receives a valid message, 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 11-bit or 29-bit identifier (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.

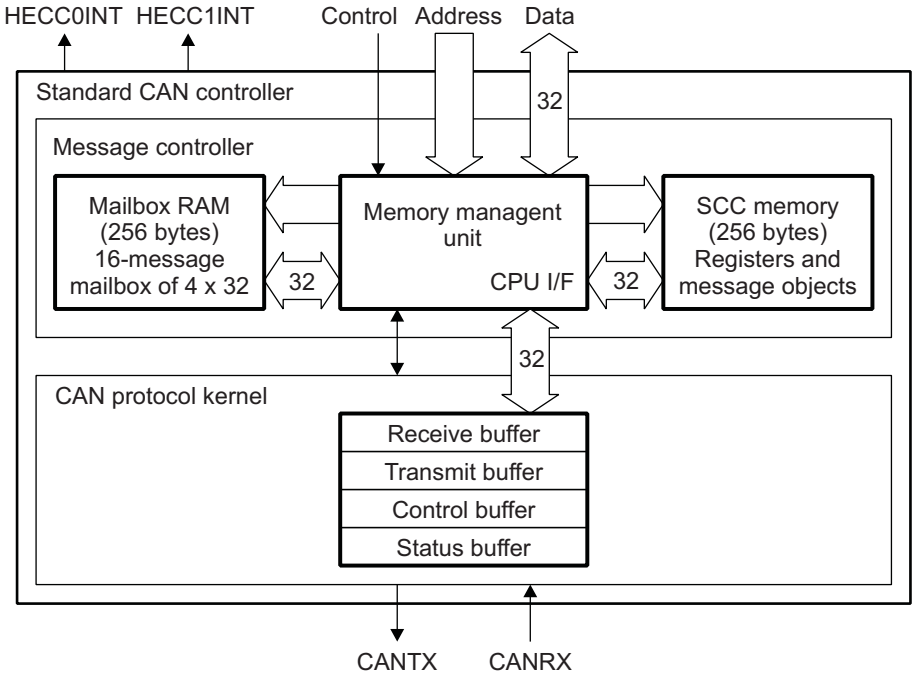
The memory registers contain the global identifier mask and the two dedicated identifier masks for the message objects 0-2 and 3-5. After the reception of a valid identifier, the receive control unit checks the state and the identifier of all objects to determine if the received message must be stored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

To avoid jitter on the CAN bit timing caused by using a frequency-modulated PLL to generate the peripheral clock for the CAN module, the CPK clock can be separated from the rest of the peripheral clock domain and can be supplied with a jitter-free clock source.

The separation of the CPK clock from the peripheral clock is controlled by a dedicated bit. For details about the separation of the CPK clock, refer to the device-specific data manual.

Figure 6. SCC Functional Block Diagram



2.6.2 High-End CAN Controller (HECC) Overview

The HECC (Figure 7) is a new-generation, advanced CAN controller with an internal 32-bit architecture. The HECC consists of the:

- the CAN protocol kernel (CPK)
- the message controller:
  - 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
  - 128 bytes of memory comprising the registers and the message objects controls
  - 384 bytes of control RAM comprising the message objects control registers

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-bit 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.

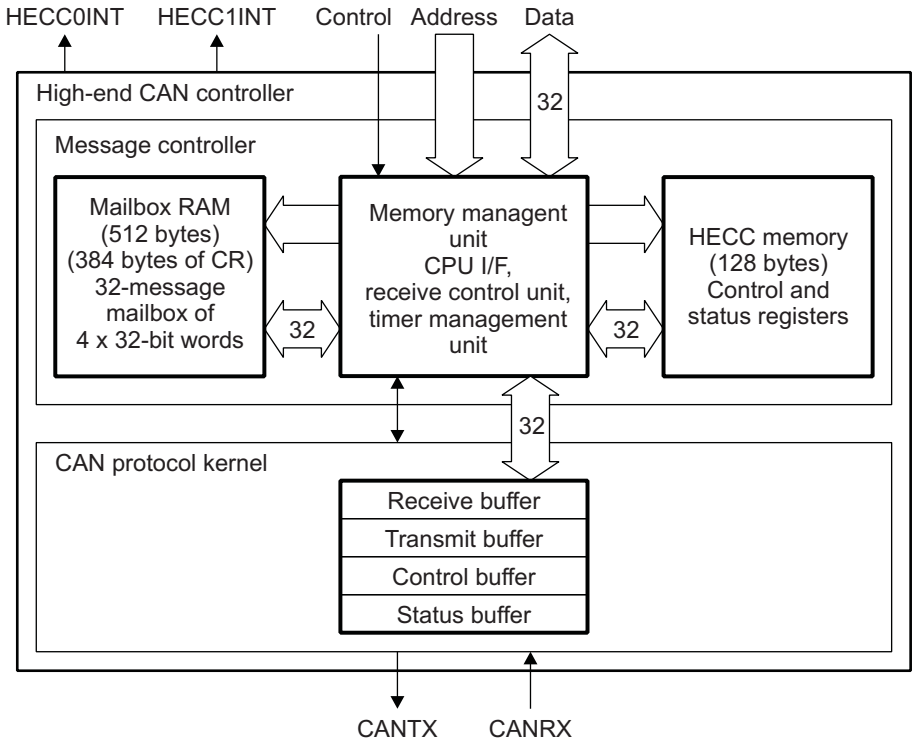
The timer management unit comprises a local network time counter and affixes a time stamp to all messages received or transmitted. The timer management unit controls all message reception and transmission, and generates an alarm when a message has not been received or transmitted during an allowed period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

To avoid jitter on the CAN bit timing caused by using a frequency-modulated PLL to generate the peripheral clock for the CAN module, the CPK clock can be separated from the rest of the peripheral clock domain and can be supplied with a jitter-free clock source.

The separation of the CPK clock from the peripheral clock is controlled by a dedicated bit. For details about the separation of the CPK clock, refer to the device-specific data manual.

Figure 7. HECC Functional Block Diagram



2.6.2.1 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. The SCC-compatible mode is selected with the SCM bit in the master control register (CANMC).

Note: HECC Used in SCC-Compatible Mode

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

2.7 Message Objects

The following sections describe the SCC message objects, the HECC message objects, and the CAN message mailbox.

2.7.1 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, 3-5, and 6-15 share the same acceptance masks. A SCC message object consists of 20 bytes of RAM distributed.

- 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.

Note: Unused Message Mailboxes

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

2.7.2 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 an individual acceptance mask. A HECC message object consists of 28 bytes of RAM distributed.

- 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.

When the HECC is used in SCC mode, you must refer to the SCC message objects description (see Section 2.7.1).

Note: Unused Message Mailboxes

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

2.7.3 CAN Message Mailbox

The message mailboxes are the RAM area where the CAN messages are actually stored after received or before 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 (IDE) bit in the message identifier register (MID)
- The acceptance mask enable (AME) bit in MID
- The auto answer mode (AAM) bit in MID
- The remote transmission request (RTR) bit in the message control field register (MCF)
- The data length code (DLC) bits in MCF
- Up to eight bytes for the data field
- The transmit priority level (TPL) bits, in a register on the HECC, in MCF

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

Table 3. Message Object Behavior Configuration

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

2.7.3.1 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 in the transmission request set register (CANTRS) has been set.

If more than one mailbox is configured as a transmit mailbox and more than one corresponding TRS[*n*] bit 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) has the highest transmit priority.

In the HECC, the priority of the mailbox transmission depends on the setting of the TPL bits in the message control field register (CANMCF). The mailbox with the highest value in the TPL bits is transmitted first. Only when two mailboxes have the same priority value is the higher numbered mailbox transmitted first.

If a transmission fails because of 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.

2.7.3.2 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 RMP[*n*] bit in the receive message pending register (CANRMP) 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 the HECC (in SCC-compatible mode) has the highest receive priority; mailbox 31 of the HECC in HECC mode has the highest receive priority.

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

Note: Operating in Loop-back Mode

While operating in loop-back mode:

- mailbox 0 receives data even if it is not programmed to receive data or
- the last used reception mailbox, whether it was in loop-back mode or normal operation mode, receives data even if none of the mailboxes is programmed to receive data

2.7.3.3 Handling of Remote Frames

Note: Remote Transmission Request (RTR) Bit

When a remote transmission request is successfully transmitted with a message object configured in request mode, the transmission acknowledge register (CANTA) 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.

If a remote frame is received, the incoming message has the remote transmission request (RTR) bit in the message control field register (MCF) set to 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 a send mailbox and the AAM bit in the message identifier register (MID) in this message object is set), this message object is marked as to be sent (TRS[*n*] bit in the transmission request set register (CANTRS) is set). In the case of a matching identifier (with the message object configured as a send mailbox and the AAM bit in this message object is not set), this message object is not received. After finding a matching identifier in a send mailbox, no further comparison is done.

In the case of a matching identifier and the message object configured as a receive mailbox, this message object is handled like a data frame and the corresponding RMP[*n*] bit in the receive message pending register (CANRMP) is set. The CPU then has to decide how to handle this situation.

If the CPU wants to change the data in a message object that is configured as remote frame mailbox (AAM = 1 in MID), the CPU first has to set the mailbox number (MBNR) bits and the change data request (CDR) bit in the master control register (CANMC). The CPU may then perform the access and clear the CDR bit to indicate to the CAN module that the access is finished. Until the CDR bit is cleared, the transmission of this mailbox is not performed by the CAN module. Since the TRS[*n*] bit in CANTRS is not affected by the CDR bit, a pending transmission is started after the CDR bit 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) in the mailbox enable register (CANME).

If the CPU wants to request data from another node, it may configure the message object as receive mailbox and set the TRS[*n*] bit. In this case, the module sends a remote frame request and receives the data frame in the same mailbox that sent the request. Therefore, only one mailbox is necessary to do a remote request. Note that the CPU must set the RTR bit in MCF to enable a remote frame transmission.

The behavior of the message object *n* is configured with the corresponding MD[*n*] bit in the mailbox direction register (CANMD), the AAM bit in MID, and the RTR bit in MCF.

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

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

2.7.3.4 CPU Message Mailbox Access

Write accesses to the identifier can only be accomplished when the mailbox is disabled (ME[*n*] = 0) in the mailbox enable register (CANME). 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[*n*] bit in the transmission request set register (CANTRS) or the TRR[*n*] bit in the transmission request reset register (CANTRR) is set. In these cases, an interrupt may be asserted. A way to access those mailboxes is to set the CDR bit in the master control register (CANMC) before accessing the mailbox data.

After the CPU access is finished, the CPU must clear the CDR bit by writing a 0 to the bit. The CAN module checks for that flag before and after reading the mailbox. If the CDR bit 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 bit also stops the write-denied interrupt (WDI) from being asserted.

2.8 HECC 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 on-going actions.

2.8.1 Time-Stamp Functions

Note: The message object time-stamp registers (MOTS0-MOTS31) are available on the HECC only. The offset addresses are reserved in the SCC mode.

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

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

The most significant bit of LNT is cleared by writing a 1 to the LNTM bit in the master control register (CANMC). The LNT may also be cleared (when enabled by setting the LNTC bit in CANMC to 1) when mailbox 16 transmitted or received (depending on the setting of the MD bit in the mailbox direction register (CANMD)) a message successfully. 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 (TCOIF*n*) bit in the global interrupt flag register (CANGIF*n*). An overflow occurs when the highest bit of the LNT counter changes to 1. Thus, the CPU has enough time to handle this situation.

2.8.2 Time-Out Functions

Note: The message object time-out registers (MOTO0-MOTO31) are available on the HECC only. The offset addresses are reserved in the SCC mode.

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

For transmit mailboxes, the TOS[*n*] bit is cleared when the corresponding TOC[*n*] bit is cleared or when the corresponding TRS[*n*] bit in the transmission request set register (CANTRS) is cleared, no matter either because of successful transmission or because of abortion of the transmit request. For receive mailboxes, the TOS[*n*] bit is cleared when the corresponding TOC[*n*] bit is cleared.

The CPU may also clear the CANTOS 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 local network time register (LNT) counter value. If the value in LNT is equal to or greater than the value in the time-out register, the corresponding TRS[*n*] bit (applies to transmit mailboxes only) is set, and the TOC[*n*] bit is set, then the appropriate TOS[*n*] bit is set. Because all the time-out registers are scanned sequentially, there may be a delay before the TOS[*n*] bit is set.

2.9 Reset Considerations

The HECC has a software reset. A write of 1 to the software reset (SRES) bit in the master control register (CANMC) causes a software reset of the module. All parameters, except the following bits, are reset to their default values:

- BRP, ERM, SJW, SAM, TSEG1, and TSEG2 in the bit-timing configuration register (CANBTC)
- LNTC, LNTM, SCM, CCR, PDR, DBO, WUBA, ABO, and STM in the master control register (CANMC)
- TXFUNC in the transmit I/O control register (CANTIOC)
- RXFUNC in the receive I/O control register (CANRIOC)

The mailbox contents and the error counters are not modified. Pending and on-going transmissions are canceled without disturbing the communication.

2.10 CAN Module Initialization

The CAN module must be initialized before the utilization. To initialize the CAN module:

1. Perform the necessary device pin multiplexing setup (see the device-specific data manual).
2. Program the VDD3P3V_PWDN register to power up the IO pins for the CAN module (see the device-specific data manual).
3. Enable the CANTX pin by setting the TXFUNC bit in the transmit I/O control register (CANTIOC) to 1.
4. Enable the CANRX pin by setting the RXFUNC bit in the receive I/O control register (CANRIOC) to 1.
5. Perform the procedure in Figure 8.

Programming the CCR bit in the master control register (CANMC) to 1 sets the initialization mode. The initialization can be performed only when the CCE bit in the error and status register (CANES) is set to 1 to enable the configuration registers to be written.

To modify the global acceptance mask register (CANGAM) and the two SCC local acceptance mask registers (SCCLAM0 and SCCLAM3) of the SCC, the CAN module also must be set in the initialization mode.

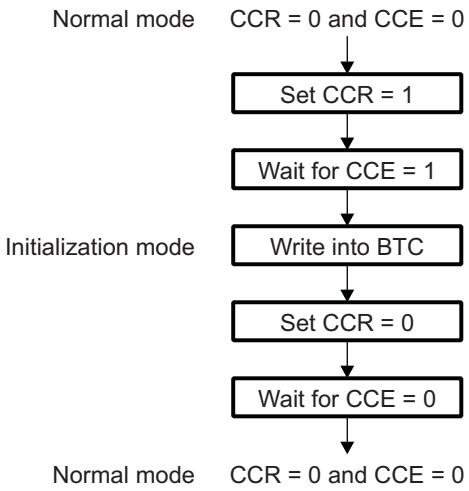
The module is activated again by programming the CCR bit in CANMC to 0. After hardware reset, the initialization mode is active.

If the bit-timing configuration register (CANBTC) is programmed to 0 or with the initial value, the CAN module never leaves the initialization mode; that is, the CCE bit in CANES remains at 1 when clearing the CCR bit in CANMC.

Note: Enter/Exit Initialization Mode

The transition between initialization mode and normal mode and conversely 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. If a stuck-to-dominant bus error occurs, the CAN controller can not detect a bus-idle condition and, therefore, is unable to perform a mode transition.

Figure 8. Configuration Sequence



2.11 Interrupt Support

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 9 and Figure 10.

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

2.11.1 Interrupts Scheme

The interrupt flags are set if the corresponding interrupt condition occurred. The system interrupt flags are set depending on the setting of the SIL bit in the global interrupt mask register (CANGIM). If set to 1, the global interrupts set the bits in the global interrupt flag 1 register (CANGIF1); if cleared to 0, the global interrupts set the bits in CANGIF0.

The GMIF*n* bit in CANGIF*n* is set depending on the setting of the MIL[*n*] bit in CANMIL that corresponds to the message object originating that interrupt. If the MIL[*n*] bit is set, the corresponding message object interrupt flag (GMIF*n*) sets the GMIF1 bit in CANGIF1 or sets the GMIF0 bit in CANGIF0.

If all interrupt flags are cleared and a new interrupt flag is set, the CAN module interrupt output line (HECC0INT or 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 GMIF*n* bit in CANGIF*n* must be cleared by writing a 1 to the appropriate bit in the transmission acknowledge register (CANTA) or the receive message pending register (CANRMP), depending on mailbox configuration. These bits cannot be cleared in CANGIF*n*.

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 GMIF*n* bit is set, the interrupt vector (MIV*n*) bits in CANGIF*n* indicate the mailbox number of the mailbox that caused the setting of the GMIF*n* bit. It will always display the highest mailbox interrupt vector assigned to that interrupt line.

Figure 9. SCC Interrupts Scheme

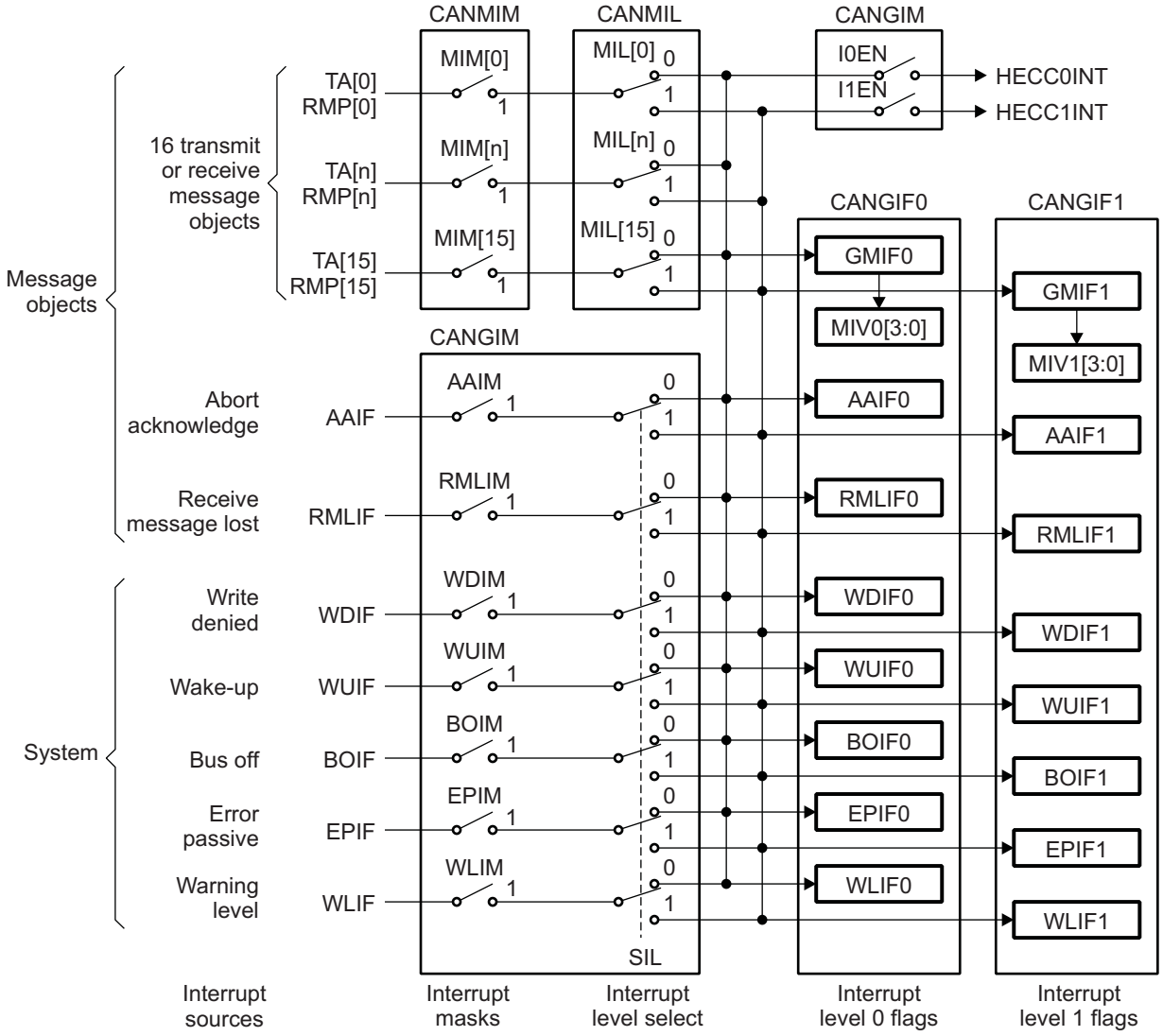
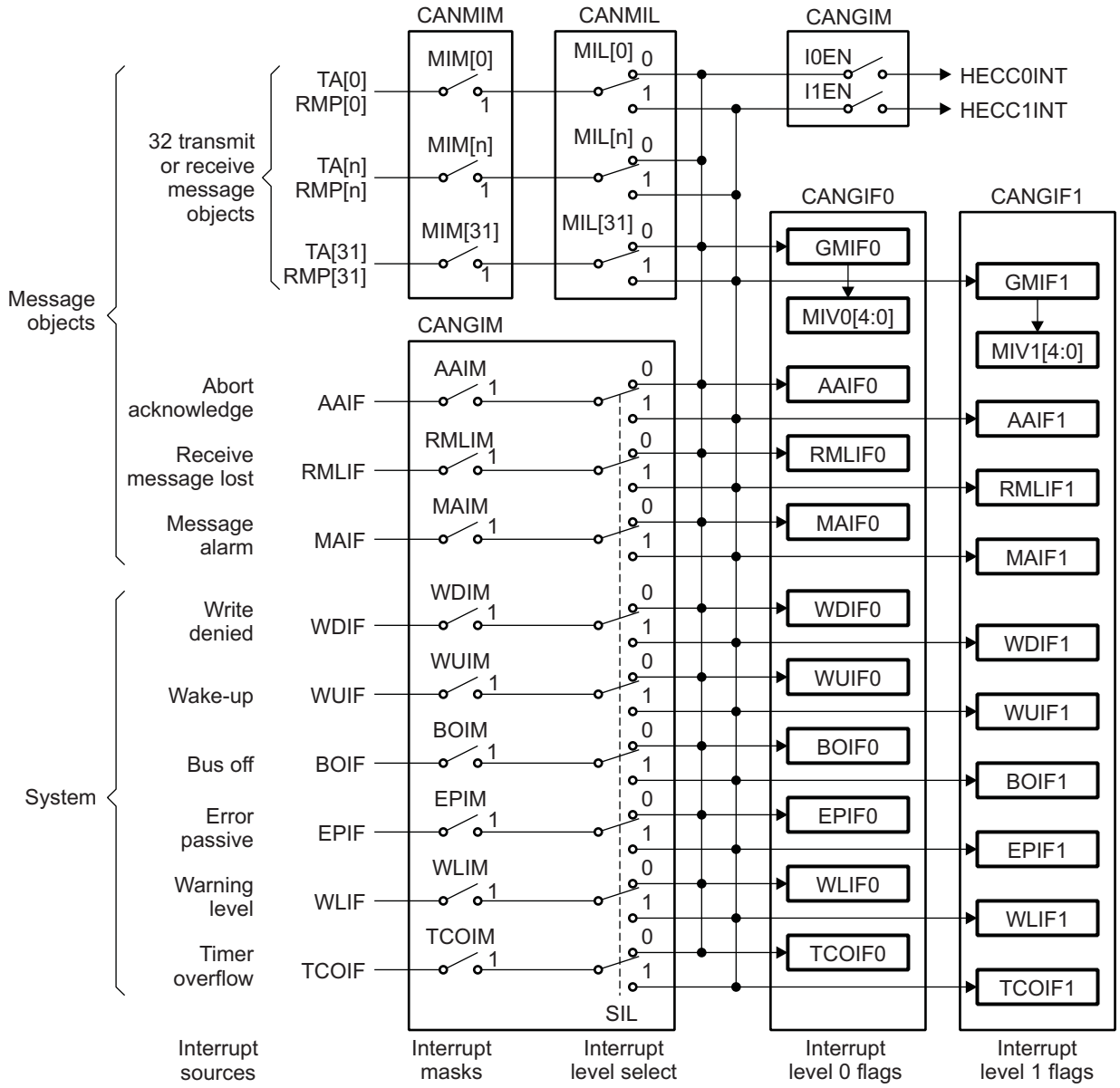


Figure 10. HECC Interrupts Scheme



2.11.2 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 (MIM[*n*]) bit in the mailbox interrupt mask register (CANMIM) and one interrupt level (MIL[*n*]) bit in the mailbox interrupt level register (CANMIL) dedicated to each mailbox. To generate a mailbox interrupt upon a receive/transmit event, the MIM[*n*]) bit has to be set. If a CAN message is received in a receive mailbox (RMP[*n*] = 1 in the receive message pending register (CANRMP)) or is transmitted from a transmit mailbox (TA[*n*] = 1 in the transmission acknowledge register (CANTA)), an interrupt is asserted. If a mailbox is configured as remote request mailbox (MD[*n*] = 1 in the mailbox direction register (CANMD) and RTR = 1 in the message control field register (MCF)), 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 in CANMD and AAM = 1 in the message identifier register (MID)).

The setting of the RMP[*n*] bit in CANRMP or the TA[*n*] bit in CANTA also sets the GMIF*n* flag in the global interrupt flag *n* register (CANGIF*n*), if the corresponding interrupt mask bit is set. The GMIF*n* flag then generates an interrupt and the corresponding mailbox vector (mailbox number) can be read from the MIV*n* bits in CANGIF*n*. If more than one mailbox interrupts are pending, the actual value of the MIV*n* bits reflects the highest priority interrupt vector. The interrupt generated depends on the setting in CANMIL.

The abort acknowledge flag (AA[*n*]) bit in the abort acknowledge register (CANAA) and the abort acknowledge interrupt flag (AAIF*n*) bit in CANGIF*n* are set when a transmit message is aborted by setting the TRR[*n*] bit in the transmission request reset register (CANTRR). An interrupt is asserted upon transmission abortion, if the AAIM bit in the global interrupt mask register (CANGIM) is set. Clearing the AA[*n*] flag(s) in CANAA does not reset the AAIF*n* bit in CANGIF*n*. The interrupt flag has to be cleared separately. The interrupt line for the abort acknowledge interrupt is selected with the SIL bit in CANGIM.

A lost receive message is notified by setting the receive message lost flag (RML[*n*]) bit in the receive message lost register (CANRML) and the receive message lost interrupt flag (RMLIF*n*) bit in CANGIF*n*. If an interrupt is generated upon the lost receive message event, the receive message lost interrupt mask (RMLIM) bit in CANGIM has to be set. Clearing the RML[*n*] flag in CANRML does not reset the RMLIF*n* bit in CANGIF*n*. The interrupt flag has to be cleared separately. The interrupt line for the receive message lost interrupt is selected with the SIL bit in CANGIM.

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 in the time-out status register (TOS)), the message alarm interrupt flag (MAIF*n*) bit in CANGIF*n* is set and a message alarm interrupt is asserted to one of the two interrupt lines if the message alarm interrupt mask (MAIM) bit in CANGIM is set. The interrupt line for the message alarm interrupt is selected with the SIL bit in CANGIM. Clearing the TOS[*n*] flag does not reset the MAIF*n* bit in CANGIF*n*.

2.11.3 Interrupt Events

The HECC/SCC interrupt events to the CPU are listed in Table 4.

Table 4. HECC Interrupts

Event	Acronym	Source
57	HECC0INT	HECC
58	HECC1INT	HECC

2.11.4 Interrupt Multiplexing

The interrupts generated by the HECC/SCC peripheral to the CPU are multiplexed with other interrupt sources. For more details on how to manage multiplexing of CPU interrupts, see the *TMS320DM643x DMP DSP Subsystem Reference Guide* (SPRU978).

2.11.5 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 global interrupt flag register *n* (CANGIF0 or CANGIF1) by writing a 1 to the interrupt flag. This also releases the interrupt line if no other interrupt is pending.

2.11.5.1 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:

1. Write to the MIL[*n*] bits in CANMIL. This defines whether a successful transmission asserts interrupt line 0 (MIL[*n*] = 0) or line 1 (MIL[*n*] = 1). For example, CANMIL = FFFF FFFFh sets all mailbox interrupts to level 1.
2. Configure CANMIM to mask out the mailboxes that should not cause an interrupt. For example, CANMIM could be set to FFFF FFFFh, which enables all mailbox interrupts. Mailboxes that are not used do not cause any interrupts.
3. Now configure CANGIM. The AAIM, WDIM, WUIM, BOIM, EPIM, and WLIM bits should always be set (enabling these interrupts). In addition, the SIL bit may be set to have the global interrupts on another level than the mailbox interrupts. Both the I1EN bit and the I0EN bit should be set to enable both interrupt lines. The RMLIM bit may also be set, depending on the load of the CPU.

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

2.11.5.2 Handling Mailbox Interrupts

There are three interrupt flags for mailbox interrupts. Do a halfword read on the global interrupt flag register (CANGIF*n*) that caused the interrupt. If the value is negative, a mailbox caused the interrupt (GMIF*n* bit in CANGIF*n*); or, check the AAIF*n* bit or the RMLIF*n* bit in CANGIF*n*; or, a system interrupt has occurred and each of the system-interrupt flags must be checked.

1. If the RMLIF*n* bit 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 RML[*n*] bits in the receive message lost register (RML) to find out which mailbox *n* caused that interrupt. Depending on the application, the CPU has to decide what to do next. This interrupt comes together with a GMIF0/GMIF1 interrupt.
2. If the AAIF*n* bit caused the interrupt, a send transmission operation was aborted by the CPU. The CPU should check the AA[*n*] bits in the abort acknowledge register (CANAA) to find out which mailbox *n* caused the interrupt and send that message again, if requested. The flag must be cleared by writing a 1 to it.
3. If the GMIF*n* bit caused the interrupt, the mailbox number that caused the interrupt can be read from the MIV*n* bits. 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 and clear the RMP[*n*]) flag in the receive message pending register (CANRMP) 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 is necessary. The CPU needs to clear the TA[*n*] flag in the transmission acknowledge register (CANTA) by writing a 1 to it.

2.12 EDMA Event Support

This HECC does not support any EDMA event outputs.

2.13 Power Management

The HECC peripheral can be placed in reduced-power modes to conserve power during periods of low activity. The power management of the HECC peripheral is controlled by the processor Power and Sleep Controller (PSC). The PSC acts as a master controller for power management for all of the peripherals on the device. For detailed information on power management procedures using the PSC, see the *TMS320DM643x DMP DSP Subsystem Reference Guide* (SPRU978).

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 where the clock for the complete chip is disabled.

2.13.1 Local Power-Down Mode

The local power-down mode is requested by writing a 1 to the PDR bit in the master control register (CANMC). When the module enters the local power-down mode, the PDA bit in the error and status register (CANES) 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. Any register access is possible as the clock is enabled. The actual contents of any register can be read back, even during power down. The module leaves the local power-down mode when the PDR bit is cleared to 0 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 is enabled or disabled with the WUBA bit in the master control register (CANMC). 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 bit in CANMC and the PDA bit in CANES 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: If the CAN module goes into standby mode when the bus-off condition has occurred, the CCR bit in CANMC is set automatically upon wake-up. Then, for normal operation to resume, the device must wait for 128×11 recessive bits to occur and for the CCE bit in CANES to be set, before the CPU can clear the CCR bit and wait for the CCE bit to go to 0.

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 (the wake-up logic). The other clock is enabled depending on the setting of the PDA bit in CANES.

2.13.2 Global Power-Down Mode

The global power-down mode is requested by the processor Power and Sleep Controller (PSC). When the PSC 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 the PSC is asserted, the wake-up interrupt flag (WUIF*n*) bit in the global interrupt flag *n* register (CANGIF*n*) 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*n* flag is set asynchronously to enable the assertion of an interrupt without any running clocks.

2.14 Emulation Considerations

The suspend mode acknowledge (SMA) bit in the error and status register (CANES) is set after a latency of 1 CAN module system clock cycle—up to the length of one frame—after the suspend mode is 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 enters suspend mode only at the end of the frame.

3 CAN Operation Examples

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

3.1 Configuration of SCC or HECC

The following steps must be performed to configure the SCC/HECC for operation:

1. Set the CANTX and the CANRX pins to CAN functions:
   - Write to the transmit I/O control register (CANTIOC) = 08h
   - Write to the receive I/O control register (CANRIOC) = 08h
2. After a reset, the CCR bit in the master control register (CANMC) and the CCE bit in the error and status register (CANES) are set to 1. This allows you to configure the bit-timing configuration register (CANBTC).

   If the CCE bit is set to 1, proceed to the next step; otherwise, set the CCR bit in CANMC to 1 and wait until the CCE bit in CANES is set to 1.
3. Program CANBTC with the appropriate timing values. Make sure that the TSEG1 and TSEG2 bit values are not 0. If they are 0, the module does not leave the initialization mode. For example: write CANBTC = 1021Ah.
4. For the SCC, program the acceptance mask registers. For example: write SCCLAM3 = 3C 0000h.
5. Program the master control register (CANMC):
   - Clear the CCR bit to 0
   - Clear the PDR bit to 0
   - Clear the DBO bit to 0
   - Clear the WUBA bit to 0
   - Clear the CDR bit to 0
   - Clear the ABO bit to 0
   - Clear the STM bit to 0
   - Clear the SRES bit to 0
   - Clear the MBNR bits to 0
6. Verify the CCE bit in CANES is cleared to 0, indicating that the CAN module has been configured.

3.2 Transmit Mailbox

3.2.1 Configuring a Mailbox for Transmit

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

1. Clear the appropriate TRS[*n*] bit in the transmission request set register (CANTRS) to 0. Writing a 0 to the TRS[*n*] bit has no effect; instead, set the TRR[*n*] bit in the transmission request reset register (CANTRR) to 1 and wait until the TRS[*n*] bit is cleared.
   - Clear TRS[1] = 0 (set the TRR[1] bit in CANTRR to 1 and wait until the TRS[1] bit is cleared.)
2. Disable the object by clearing the corresponding ME[*n*] bit in the mailbox enable register (CANME) to 0.
   - Clear CANME[1] = 0
3. Load the message identifier register *n* (MID*n*) of the object. Clear the AME bit and the AAM bit to 0 for a normal send mailbox. This register is usually not modified during operation and can only be modified when the mailbox is disabled.
   - For example: write MID1 = 15AC 0000h
4. Write the data length into the DLC bits in the message control field register *n* (MCF*n*). The RTR bit is usually cleared to 0. This register is usually not modified during operation and can only be modified when the object is disabled.
   - For example: MCF1 = 2
5. Configure the mailbox direction as a transmit mailbox by clearing the corresponding MD[*n*] bit in the mailbox direction register (CANMD) to 0.
   - Clear MD[1] = 0
6. Configure the mailbox enable by setting the corresponding ME[*n*] bit in CANME to 1.
   - Set ME[1] = 1

3.2.2 Transmitting a Message

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

1. Write the message data into the mailbox data field. Since the DBO bit in the master control register (CANMC) is cleared to 0, in the configuration section, and the message control field register 1 (MCF1) is set to 2, the data are stored in the 2 MSBytes of the message data low register 1 (MDL1).
   - Write MDL1 = xxxx 0000h
2. Set the corresponding TRS[*n*] bit in the transmission request set register (CANTRS) to 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 (TA[*n*]) bit in the transmission acknowledge register (CANTA) of the corresponding mailbox is set to 1. After a successful transmission, this flag is set by the CAN module.
   - TA[1] = 1
4. The TRS[*n*] bit in CANTRS 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 the TA[*n*] bit in CANTA to 1. Wait until read, TA[*n*] bit is 0.
   - Set TA[1] = 1. Wait until read, TA[1] = 0.
6. To transmit another message in the same message object, the mailbox RAM data must be updated. Set the TRS[*n*] bit in CANTRS to 1, to start the next transmission.
   - Set TRS[1] = 1

3.3 Receive Mailbox

3.3.1 Configuring Mailboxes for Receive

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

1. Disable mailbox *n* by clearing the corresponding ME[*n*] bit in the mailbox enable register (CANME) to 0.
   - Clear ME[3] = 0
2. Write the selected identifier into the corresponding message identifier register *n* (MID*n*). The identifier extension (IDE) bit must be configured to fit the expected identifier. If the acceptance mask is used, the AME bit must be set to 1.
   - For example: write MID3 = 4F78 0000h
3. For the HECC: 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 3.1).
   - For example: write SCCLAM3 = 03C 0000h
4. Configure the mailbox direction as a receive mailbox by setting the corresponding MD[*n*] bit in the mailbox direction register (CANMD) to 1. Make sure no other bits in this register are affected by this operation.
   - Set MD[3] = 1
5. If data in the mailbox is to be protected, the overwrite protection control register (CANOPC) should be programmed. This protection is useful if no message must be lost. If OPC[*n*] 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 mailbox *n* by setting the corresponding ME[*n*] bit in CANME. This should be done by reading CANME and writing back (CANME |= 0008h) 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.

3.3.2 Receiving a Message

This example uses message object 3. When a message is received, the corresponding (RMP[*n*]) 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) in the receive message lost register (CANRML). 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[*n*] bit in CANRMP has not been set again by the module. If the RMP[*n*] 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 message.

3.3.3 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 SCCLAM3. For the HECC, each object has its own LAM: LAM3, LAM4, and LAM5, 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[*n*] 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[*n*] 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[*n*] flag is not set for mailbox 3. It also sets the RML[*n*] 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 (that is, more than one message). In this case, all data needed for the message can be collected in the mailboxes and be read at once.

3.4 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.

3.4.1 Requesting Data From Another Node

To request data from another node, the object is configured as receive mailbox (see Section 3.3.1). Using object 3 for this example, the CPU needs to do the following:

1. Set the RTR bit in the message control field register *n* (MCF*n*) to 1: write MCF3 = 12h.
2. Write the correct identifier into the message identifier register *n* (MID*n*): write MID3 = 4F78 0000h.
3. Set the TRS[*n*] bit in the transmission request set register (CANTRS) 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[*n*]) flag in the receive message pending register (CANRMP) 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 3.3.2.

3.4.2 Answering a Remote Request

To answer a remote request, the object needs to be configured as a transmit mailbox (see Section 3.2.1).

1. The AAM bit in the message identifier register *n* (MID*n*) must be set before mailbox *n* is enabled: MID1 = 35AC 0000h.
2. The data field in the message data low-word register *n* (MDL*n*) must be updated: MDL1 = xxxx 0000h.
3. Now mailbox *n* is enabled by setting the ME[*n*] bit in the mailbox enable register (CANME) to 1: ME[1] = 1.
4. When a remote request is received from another node, the TRS[*n*] flag in the transmission request set register (CANTRS) is set automatically and the data is transmitted to that node. The identifier of the received message and the transmitted message are the same.
5. After transmission of the data, the TA[*n*] bit in the transmission acknowledge register (CANTA) is set. The CPU may then update the data. Wait for TA[1] = 1.

3.4.3 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 the TRS[*n*] flag in the transmission request set register (CANTRS) set.

1. Set the change data request (CDR) bit and the mailbox number (MBNR) bits of that object in the master control register (CANMC). This lets the CAN module know that the CPU wants to change the data field. For example, for object 1: write CANMC = 0000 0101h.
2. Write the message data into message data low-word register *n* (MDL*n*).
   For example: write MDL1 = xxxx 0000h.
3. Clear the CDR bit in CANMC to enable the object: Write CANMC = 0000 0000h.

4 Registers

This section describes the registers in the HECC/SCC. Access to the reserved locations on the SCC and HECC may hang the device. Software must not access the reserved locations of the SCC and HECC.

4.1 Control Registers

The HECC/SCC control registers listed in Table 5 are used by the CPU to configure and control the CAN controller and the message objects. All CAN registers support 8-bit, 16-bit, and 32-bit accesses. See the device-specific data manual for the memory address of these registers.

Table 5. HECC/SCC Control Registers

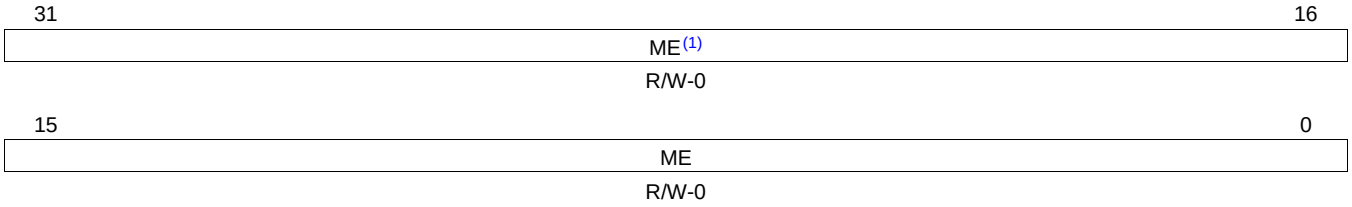
Offset[1]	Acronym	Register Description	Section
00h	CANME	Mailbox enable register	Section 4.1.1
04h	CANMD	Mailbox direction register	Section 4.1.2
08h	CANTRS	Transmission request set register	Section 4.1.3
0Ch	CANTRR	Transmission request reset register	Section 4.1.4
10h	CANTA	Transmission acknowledge register	Section 4.1.5
14h	CANAA	Abort acknowledge register	Section 4.1.6
18h	CANRMP	Receive message pending register	Section 4.1.7
1Ch	CANRML	Receive message lost register	Section 4.1.8
20h	CANRFP	Remote frame pending register	Section 4.1.9
24h	CANGAM	Global acceptance mask register (SCC only)	Section 4.2.2
28h	CANMC	Master control register	Section 4.1.10
2Ch	CANBTC	Bit-timing configuration register	Section 4.1.11
30h	CANES	Error and status register	Section 4.1.12
34h	CANTEC	Transmit error counter register	Section 4.1.13
38h	CANREC	Receive error counter register	Section 4.1.14
3Ch	CANGIF0	Global interrupt flag 0 register	Section 4.1.15
40h	CANGIM	Global interrupt mask register	Section 4.1.16
44h	CANGIF1	Global interrupt flag 1 register	Section 4.1.15
48h	CANMIM	Mailbox interrupt mask register	Section 4.1.17
4Ch	CANMIL	Mailbox interrupt level register	Section 4.1.18
50h	CANOPC	Overwrite protection control register	Section 4.1.19
54h	CANTIOC	Transmit I/O control register	Section 4.1.20
58h	CANRIOC	Receive I/O control register	Section 4.1.21
5Ch	CANLNT	Local network time register (HECC only)	Section 4.1.22
60h	CANTOC	Time-out control register (HECC only)	Section 4.1.23
64h	CANTOS	Time-out status register (HECC only)	Section 4.1.24
70h	CANETC	Error test control register	Section 4.1.25

(1) The HECC/SCC Offset address location is referenced in the configuration memory-map summary of the data manual as the HECC Control Region.

4.1.1 Mailbox Enable Register (CANME)

The mailbox enable register (CANME) is used to enable or disable each mailbox. The CANME is shown in Figure 11 and described in Table 6.

Figure 11. Mailbox Enable Register (CANME)



LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

(1) Bits 31-16 on the HECC only; reserved on the SCC.

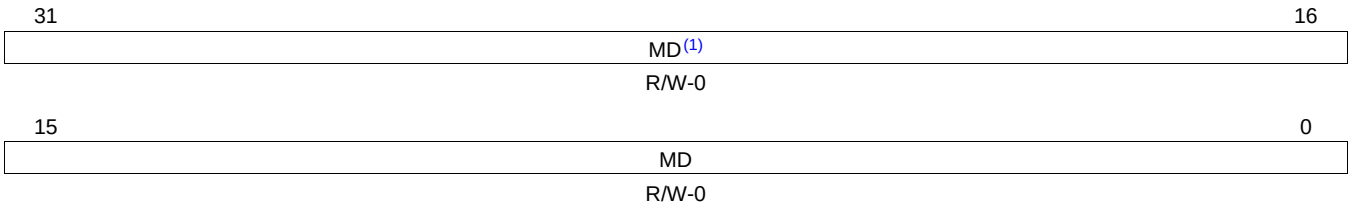
Table 6. Mailbox Enable Register (CANME) Field Descriptions

Bit	Field	Value	Description
31-0	ME[*n*]		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 mailbox is disabled.
		1	The mailbox is enabled in 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.

4.1.2 Mailbox Direction Register (CANMD)

The mailbox direction register (CANMD) configures each mailbox as a transmit or a receive mailbox. The CANMD is shown in Figure 12 and described in Table 7.

Figure 12. Mailbox Direction Register (CANMD)



LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

(1) Bits 31-16 on the HECC only; reserved on the SCC.

Table 7. Mailbox Direction Register (CANMD) Field Descriptions

Bit	Field	Value	Description
31-0	MD[*n*]		Mailbox direction. After power-up, all bits in CANMD are cleared.
		0	The mailbox is defined as a transmit mailbox.
		1	The mailbox is defined as a receive mailbox.

4.1.3 Transmission Request Set Register (CANTRS)

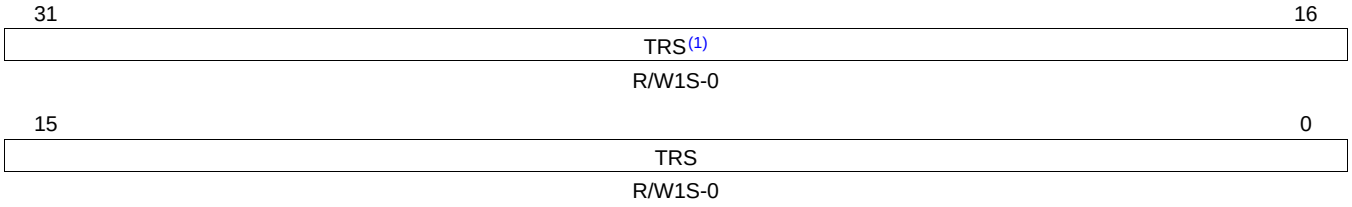
When mailbox *n* is ready to be transmitted, the CPU sets the TRS[*n*] bit in the transmission request set register (CANTRS) to 1 to start the transmission.

The CANTRS bits are set by the CPU and reset by the CAN module logic. It is also set by the CAN module in the case of a remote frame request. These bits are reset in the 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 the TRS[*n*] bit causes the particular message *n* to be transmitted. Several bits can be set simultaneously; therefore, all messages with the TRS[*n*] 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. The CANTRS is shown in Figure 13 and described in Table 8.

Figure 13. Transmission Request Set Register (CANTRS)



LEGEND: R/W = Read/Write; W1S = Write 1 to set; -*n* = value after reset

(1) Bits 31-16 on the HECC only; reserved on the SCC.

Table 8. Transmission Request Set Register (CANTRS) Field Descriptions

Bit	Field	Value	Description
31-0	TRS[*n*]		Transmit request set. A successful transmission initiates a GMIF0/GMIF1 interrupt (bit 15 in CANGIF0/CANGIF1) if enabled. After power-up, all bits in CANTRS are cleared.
		0	No operation occurs.
		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.

4.1.4 Transmission Request Reset Register (CANTRR)

Setting the TRR[*n*] bit in the transmission request reset register (CANTRR) of the message object *n* causes a transmission request to be cancelled, if it was initiated by the corresponding TRS[*n*] bit in the transmission request set register (CANTRS) and is not currently being processed. If the corresponding message is currently being processed, the TRR[*n*] bit is reset in the case of a successful transmission (normal operation) or in the case of an aborted transmission due to a lost arbitration or an error condition detected on the CAN bus line. In the case of an aborted transmission, the corresponding AA[*n*] bit in the abort acknowledge register (CANAA) is set. In the case of a successful transmission, the corresponding TA[*n*] bit in the transmission acknowledge register (CANTA) is set. The status of the transmission request reset can be read from the TRS[*n*] bit in CANTRS.

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

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

The bits in CANTRR are set by writing a 1 from the CPU; writing a 0 has no effect. The transmission request reset register (CANTRR) is shown in Figure 14 and described in Table 9.

Figure 14. Transmission Request Reset Register (CANTRR)

31															16
TRR[(1)]															
R/W1S-0															
15															0
TRR															
R/W1S-0															

LEGEND: R/W = Read/Write; W1S = Write 1 to set; -*n* = value after reset

(1) Bits 31-16 on the HECC only, reserved on the SCC

Table 9. Transmission Request Reset Register (CANTRR) Field Descriptions

Bit	Field	Value	Description
31-0	TRR[*n*]		Transmit request reset. A successful transmission initiates a GMIF0/GMIF1 interrupt (bit 15 in CANGIF0/CANGIF1) if enabled.
		0	No operation occurs.
		1	Setting TRR[*n*] = 1 causes a transmission request to be cancelled if it was initiated by the corresponding TRS[*n*] bit in CANTRS and is not currently being processed.

4.1.5 Transmission Acknowledge Register (CANTA)

Note: Remote Transmission Request (RTR) Bit
When a remote transmission request is successfully transmitted with a message object configured in request mode, the transmission acknowledge register (CANTA) 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.

If the message of mailbox *n* was sent successfully, the TA[*n*] bit in the transmission acknowledge register (CANTA) is set. This also sets the GMIF*n* bit in the global interrupt flag *n* register (CANGIF*n*), if the corresponding interrupt mask (MIM[*n*]) bit in the mailbox interrupt mask register (CANMIM) is set. The GMIF*n* 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 module tries to set it, the bit is set. After power-up, all bits are cleared. The CANTA is shown in Figure 15 and described in Table 10.

Figure 15. Transmission Acknowledge Register (CANTA)

31															16
TA[(1)]															
R/W1C-0															
15															0
TA															
R/W1C-0															

LEGEND: R/W = Read/Write; R = Read only; W1C = Write 1 to clear; -*n* = value after reset

(1) Bits 31-16 on the HECC only; reserved on the SCC.

Table 10. Transmission Acknowledge Register (CANTA) Field Descriptions

Bit	Field	Value	Description
31-0	TA[*n*]		Transmit acknowledge.
		0	The message is not sent.
		1	The message of mailbox *n* was sent successfully.

4.1.6 Abort Acknowledge Register (CANAA)

If the transmission of the message in mailbox *n* was aborted, the AA[*n*] bit in the abort acknowledge register (CANAA) is set and the AAIF*n* bit in the global interrupt flag *n* register (CANGIF*n*) is set, which may generate an interrupt if enabled.

Note: Additional Conditions that Set the Abort Acknowledge (AA) Bit

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

The CANAA bits are reset by writing a 1 from the CPU. Writing a 0 has no effect. If the CPU tries to reset the bit while the CAN module tries to set it, the bit is set. After power-up, all bits are cleared. The CANAA is shown in Figure 16 and described in Table 11.

Figure 16. Abort Acknowledge Register (CANAA)

31															16
AA[(1)]															
R/W1C-0															
15															0
AA															
R/W1C-0															

LEGEND: R/W = Read/Write; R = Read only; W1C = Write 1 to clear; -*n* = value after reset

(1) Bits 31-16 on the HECC only; reserved on the SCC.

Table 11. Abort Acknowledge Register (CANAA) Field Descriptions

Bit	Field	Value	Description
31-0	AA[*n*]		Transmission abort acknowledge.
		0	The transmission is not aborted.
		1	The transmission of the message in mailbox *n* was aborted.

4.1.7 Receive Message Pending Register (CANRMP)

If mailbox *n* contains a received message, the RMP[*n*] bit in the receive message pending register (CANRMP) is set. These bits can only be reset by the CPU and set by the internal logic. A new incoming message overwrites the stored one, if the OPC[*n*] bit in the overwrite protection control register (CANOPC) is cleared; otherwise, the next mailboxes are checked for a matching ID. In this case, the corresponding message lost (RML[*n*]) bit in the receive message lost register (CANRML) is set. The bits in CANRMP and CANRML are cleared by a write access to the base address of CANRMP of 1 to the corresponding bit location. If the CPU tries to reset the bit while the CAN module tries to set it, the bit is set.

CANRMP may also set the GMIF*n* bit in the global interrupt flag *n* register (CANGIF*n*), if the corresponding interrupt mask (MIM[*n*]) bit in the mailbox interrupt mask register (CANMIM) is set. The GMIF*n* bit initiates an interrupt. The CANRMP is shown in Figure 17 and described in Table 12.

Figure 17. Receive Message Pending Register (CANRMP)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RMP[(1)]															
R/W1C-0															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RMP															
R/W1C-0															

LEGEND: R/W = Read/Write; R = Read only; W1C = Write 1 to clear; -*n* = value after reset

(1) Bits 31-16 on the HECC only; reserved on the SCC.

Table 12. Receive Message Pending Register (CANRMP) Field Descriptions

Bit	Field	Value	Description
31-0	RMP[*n*]		Receive message pending.
		0	Mailbox *n* does not contain a message.
		1	Mailbox *n* contains a received message.

4.1.8 Receive Message Lost Register (CANRML)

If an old message has been overwritten by a new message in message object *n*, the RML[*n*] bit in the receive message lost register (CANRML) is set. These bits can only be reset by the CPU and set by the internal logic. The bits are cleared by a write access to the base address of the receive message pending register (CANRMP) with a 1 to the corresponding bit location. If the CPU tries to reset the bit while the CAN module tries to set it, the bit is set. CANRML is not changed if the OPC[*n*] bit in the overwrite protection control register (CANOPC) is set.

If one or more bits in CANRML are set, the RMLIF*n* bit in the global interrupt flag *n* register (CANGIF*n*) is also set. This may initiate an interrupt, if the RMLIM bit in the global interrupt mask register (CANGIM) is set. The CANRML is shown in Figure 18 and described in Table 13.

Figure 18. Receive Message Lost Register (CANRML)

31		16
RML[(1)]		
R/W1C-0		

15		0
RML		
R/W1C-0		

LEGEND: R/W = Read/Write; R = Read only; W1C = Write 1 to clear; -*n* = value after reset

(1) Bits 31-16 on the HECC only; reserved on the SCC.

Table 13. Receive Message Lost Register (CANRML) Field Descriptions

Bit	Field	Value	Description
31-0	RML[*n*]		Receive message lost.
		0	No message was lost.
		1	An old unread message has been overwritten by a new message.

4.1.9 Remote Frame Pending Register (CANRFP)

When a remote frame request is received by the CAN module, the RFP[*n*] bit in the remote frame pending register (CANRFP) is set. If a remote frame is stored in a receive mailbox (AAM = 0 in MID and MD = 1 in CANMD), the CPU has to initiate the reply frame transmission and has to reset the RFP[*n*] bit. If a mailbox configured in auto-answer mode (AAM = 1 in MID and MD = 0 in CANMD) receives a remote frame, the CAN module clears the RFP[*n*] bit 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*] bit and clear the TRS[*n*] bit in the transmission request set register (CANTRS) by setting the corresponding transmission request reset (TRR[*n*]) bit in the transmission request reset register (CANTRR). The AAM bit in the message identifier register (MID) may also be cleared by the CPU to stop the module from sending the message.

If the CPU tries to reset the bit while the CAN module tries to set it, the bit is not set. The CPU cannot interrupt an ongoing transfer. The CANRFP is shown in Figure 19 and described in Table 14.

Figure 19. Remote Frame Pending Register (CANRFP)

31															16
RFP[(1)]															
R/W1C-0															
15															0
RFP															
R/W1C-0															

LEGEND: R/W = Read/Write; R = Read only; W1C = Write 1 to clear; -*n* = value after reset

(1) Bits 31-16 on the HECC only; reserved on the SCC.

Table 14. Remote Frame Pending Register (CANRFP) Field Descriptions

Bit	Field	Value	Description
31-0	RFP[*n*]		Remote frame pending.
		0	No remote frame request was received.
		1	A remote frame request was received by the module.

4.1.10 Master Control Register (CANMC)

Note: Bits Protected in User Mode

Some bits of the master control register (CANMC) cannot be changed in user mode.

The master control register (CANMC) is used to control the settings of the CAN module. The CANMC is shown in Figure 20 and described in Table 15.

When using the CDR bit in CANMC to update the data field of a mailbox, the SCC and the HECC behave differently. The SCC always considers new data after an 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 transmission request set register (CANTRS) (for example, CANTRS = 0000 0000h) after the data update is finished (the CDR bit is cleared).

Figure 20. Master Control Register (CANMC)

31							16
Reserved							
R-0							
15	14	13	12	11	10	9	8
LNTC(1)	LNTM(1)	SCM(1)	CCR	PDR	DBO	WUBA	CDR
W-0	W-x	W-0	R/W-1	R/W-0	R/W-0	R/W-0	R/W-0
7	6	5	4				0
ABO	STM	SRES	MBNR				
R/W-0	R/W-0	R/W-0	R/W-0(1)		R/W-0		

LEGEND: R/W = Read/Write; R = Read only; W = Write only; -*n* = value after reset; x = value is indeterminate

(1) HECC only; reserved on the SCC.

Table 15. Master Control Register (CANMC) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	Reserved. Reads are undefined and writes have no effect.
15	LNTC		Local network time clear. HECC only; reserved in the SCC. This bit is protected in user mode.
		0	The local network time register (CANLNT) is not reset.
		1	The local network time register (CANLNT) is reset to 0 after a successful transmission or reception of mailbox 16.
14	LNTM		Local network time MSB clear. HECC only; reserved in the SCC. This bit is protected in user mode.
		0	The local network time register (CANLNT) is not changed.
		1	The most-significant bit (MSB) of the local network time register (CANLNT) is reset to 0. The LNTM bit is reset after one clock cycle by the internal logic.
13	SCM		SCC-compatibility mode. HECC only; reserved in the SCC. This bit is protected in user mode.
			The HECC behaves like the SCC and only mailboxes 15-0 are used. See 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.

Table 15. Master Control Register (CANMC) Field Descriptions (continued)

Bit	Field	Value	Description
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 bit-timing configuration register (CANBTC) is set to the allowed values.
		1	The CPU requests write access to the bit-timing configuration register (CANBTC) and the acceptance mask registers (CANGAM, SCCLAM0, and SCCLAM3) of the SCC. After setting this bit, the CPU must wait until the CCE bit in the error and status register (CANES) is set to 1 before proceeding to CANBTC. This could also mean that the CAN module is in bus-off state (the BO bit in CANES). When the CCR bit is set, no new transmit requests can be issued.
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 4.2.5).This bit is protected in user mode.
		0	The data is received or transmitted most-significant byte first.
		1	The data is received or transmitted least-significant byte first.
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	CDR		Change data field request. This bit allows fast data message updates.
		0	The CPU requests normal operation.
		1	The CPU requests write access to the data field of the mailbox specified by the MBNR field. The CPU must clear the CDR bit after accessing the mailbox. The CAN module does not transmit that mailbox content while the CDR bit is set. This bit 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.
7	ABO		Auto bus on. This bit is protected in user mode.
		0	The CCR bit is set when the bus-off state is detected. The CCR bit should be cleared before resuming any CAN communication. The bus-off state is exited after 128 $\times$ 11 recessive bits.
		1	After the bus-off state, the CAN module goes back automatically into the bus-on state after 128 $\times$ 11 recessive bits.
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 is read back and stored in the appropriate mailbox.
5	SRES		Software reset. This bit can only be written to and is always read as 0.
		0	No effect occurs.
		1	A write access to this register causes a software reset of the module, that is, all parameters, except the following bits, are reset to their default values. • BRP, ERM, SJW, SAM, TSEG1, and TSEG2 in CANBTC • LNTC, LNTM, SCM, CCR, PDR, DBO, WUBA, ABO, and STM in CANMC • TXFUNC in CANTIOC • RXFUNC in CANRIOC The mailbox contents and the error counters are not modified. Pending and ongoing transmissions are canceled without disturbing the communication.
4-0	MBNR	0–1Fh	Mailbox number. The MBNR bit 4 is for the HECC only and is reserved on 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.

4.1.11 Bit-Timing Configuration Register (CANBTC)

Note: Invalid Configuration Values

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

The bit-timing configuration register (CANBTC) is used to configure the CAN module with the appropriate network-timing parameters. CANBTC must be programmed before using the CAN module. CANBTC is write-protected in user mode and can only be written in initialization mode. (See Section 2.1.1).

The CANBTC is shown in Figure 21 and described in Table 16.

Figure 21. Bit-Timing Configuration Register (CANBTC)

31	24	23	16
Reserved		BRP	
R-x		R/WI-0	

15	11	10	9	8	7	6	3	2	0
Reserved		ERM	SJW		SAM	TSEG1		TSEG2	
R-0		R/WI-0	R/WI-0		R/WI-0	R/WI-0		R/WI-0	

LEGEND: R/W = Read/Write; R = Read only; WI = Write during initialization mode only; -*n* = value after reset; x = value is indeterminate

Table 16. Bit-Timing Configuration Register (CANBTC) Field Descriptions

Bit	Field	Value	Description
31-24	Reserved	0	Reserved. Reads are undefined and writes have no effect.
23-16	BRP	0-FFh	Time quantum prescaler. Specifies the duration of a time quantum (TQ) in CAN module system clock units. The length of one TQ is defined by: $TQ = \frac{BRP}{\text{CAN module system clock}}$ where *CAN module system clock* is the frequency of the CAN module system clock and *BRP* is the binary value of (BRP + 1), calculated as follows: BRP = 1 + BRP[0] + (2 × BRP[1]) + (4 × BRP[2]) + (8 x BRP[3]) + (16 × BRP[4]) + (32 × BRP[5]) + (64 × BRP[6]) + (128 × BRP[7]) **Note:** The CPK clock frequency is the CAN module system clock, if the dual clock option is disabled (default). If the dual clock option is enabled, the CPK clock is supplied with a separate clock source. The value programmed in CANBTC must be decremented by 1, because of the 8-bit size of the binary BRP value: $BRP_{CANBTC} = BRP_{CALC} - 1$ BRP_{CALC} = result of the bit-timing calculation $1 \le BRP_{CALC} \le 256$ BRP_{CANBTC} = value to be programmed in CANBTC $0 \le BRP_{CANBTC} \le 255$
15-11	Reserved	0	Reserved. Reads are undefined and writes have no effect.

Table 16. Bit-Timing Configuration Register (CANBTC) Field Descriptions (continued)

Bit	Field	Value	Description
10	ERM		Edge resynchronization mode. This bit selects the synchronization mode with the receive data stream on the CAN bus.
		0	The CAN module resynchronizes on the falling edge only.
		1	The CAN module resynchronizes on both rising and falling edges.
9-8	SJW	0-3h	Synchronization jump width. Indicates 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 × SJW[1]) SJW is programmable from 1–4 TQ. The maximum value of SJW is determined by the minimum value of $TSEG2_{CALC}$ or 4TQ: $SJW_{CALC(max)} \leq$ min [4 TQ or $TSEG2_{CALC}$] The value programmed in CANBTC must be decremented by 1, because of the 2-bit size of the binary SJW value: $SJW_{CANBTC} = SJW_{CALC} - 1$ SJW_{CALC} = result of the synchronization jump width calculation $1 \leq SJW_{CALC} \leq 4$ SJW_{CANBTC} = value to be programmed in CANBTC $0 \leq SJW_{CANBTC} \leq 3$
7	SAM		Sample point setting. 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 samples three times and makes a majority decision. The triple sample mode shall be selected only for bit rate prescale values greater than 4 ($BRP_{CALC} > 4$).
6-3	TSEG1	0-Fh	Time segment 1. 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, see Section 2.1.1). The value programmed in CANBTC must be decremented by 1, because of the 4-bit size of the binary TSEG1 value: $TSEG1_{CANBTC} = TSEG1_{CALC} - 1$ $TSEG1_{CALC}$ = result of the calculation $2 \leq TSEG1_{CALC} \leq 16$ $TSEG1_{CANBTC}$ = value to be programmed in CANBTC $1 \leq TSEG1_{CANBTC} \leq 15$

Table 16. Bit-Timing Configuration Register (CANBTC) Field Descriptions (continued)

Bit	Field	Value	Description
2-0	TSEG2	0-7h	Time segment 2. Specifies the length of the PHASE_SEG2 segment in TQ units and is calculated as follow: TSEG2 = 1 + TSEG2[0] + (2 × TSEG2[1]) + (4 × TSEG2[2]) $TSEG2_{CALC}$ value 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, see Section 2.1.1). The value programmed in CANBTC must be decremented by 1, because of the 3-bit size of the binary TSEG2 value: $TSEG2_{CANBTC} = TSEG2_{CALC} - 1$ $TSEG2_{CALC}$ = result of the calculation. $TSEG2_{CANBTC}$ = value to be programmed in CANBTC

4.1.12 Error and Status Register (CANES)

The error and status register (CANES) contains information about the actual status of the CAN module and displays bus error flags and 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 CANES 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 CANES to the actual values of the error flags, the error flag that 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 errors. The CANES is shown in Figure 22 and described in Table 17.

Figure 22. Error and Status Register (CANES)

31	25	24	23	22	21	20	19	18	17	16
Reserved		FE	BE	SA1	CRCE	SE	ACKE	BO	EP	EW
R-0		R/W1C-0	R/W1C-0	R/W1C-1	R/W1C-0	R/W1C-0	R/W1C-0	R/W1C-0	R/W1C-0	R/W1C-0

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

LEGEND: R/W = Read/Write; R = Read only; W1C = Write 1 to clear; -*n* = value after reset

Table 17. Error and Status Register (CANES) Field Descriptions

Bit	Field	Value	Description
31-25	Reserved	0	Reserved. Reads are undefined and writes have no effect.
24	FE		Form error flag.
		0	No form error has been 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 has been 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.
22	SA1		Stuck at dominant error. The SA1 bit is always a 1 after a hardware reset, a software reset, or a bus-off condition. The SA1 bit is cleared when a recessive bit is detected on the bus or when a 1 is written to the bit.
		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 register (CANTEC) has reached the limit of 256. During a bus-off state, no messages are received or transmitted. The bus-off state is exited automatically after 128 x 11 recessive bits. After leaving a bus-off state, the error counters are cleared. When ABO = 0 in the master control register (CANMC), then the CCR bit in CANMC should be cleared before resuming any CAN communication.

Table 17. Error and Status Register (CANES) Field Descriptions (continued)

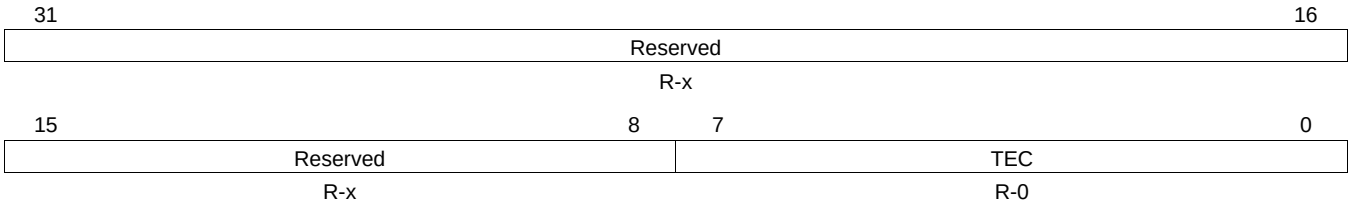
Bit	Field	Value	Description
17	EP		Error passive state.
		0	The CAN module is in error-active mode.
		1	The CAN module is in error-passive mode.
16	EW		Warning status.
		0	Both error counter registers (CANREC and CANTEC) are less than 96.
		1	One of the two error counter registers (CANREC or CANTEC) has reached the warning level of 96.
15-6	Reserved	0	Reserved. Reads are undefined and writes have no effect.
5	SMA		Suspend mode acknowledge. The SMA bit is set after a latency of 1 CAN module system 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 (see Section 2.10). After setting the CCR bit in the master control register (CANMC), the CCE bit is set after finishing the current bus activity, which means that the CCE bit is set after a latency of 1 clock cycle up to the length of one frame.
			If the CCR bit in CANMC is set when in bus-off state, the CCE bit is not set until 128 groups of 11 recessive bits occur.
		0	The CPU is denied write access to the configuration registers.
		1	The CPU has write access to the configuration registers. The CAN transmission is turned off.
3	PDA		Power-down mode acknowledge.
		0	Normal operation
		1	The CAN module has entered the power-down mode.
2	Reserved	0	Reserved. Reads are undefined and writes have no effect.
1	RM		Receive mode. The CAN protocol kernel (CPK) is in receive mode. The RM 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.
0	TM		Transmit mode. The CAN protocol kernel (CPK) is in transmit mode. The TM 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.

4.1.13 Transmit Error Counter Register (CANTEC)

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

After reaching the bus-off state, the transmit error counter register (CANTEC) is undefined while the receive error counter register (CANREC) changes its function. After leaving the initialization mode, CANTEC is cleared. The CANTEC is shown in Figure 23.

Figure 23. Transmit Error Counter Register (CANTEC)



LEGEND: R = Read only; -*n* = value after reset; x = value is indeterminate

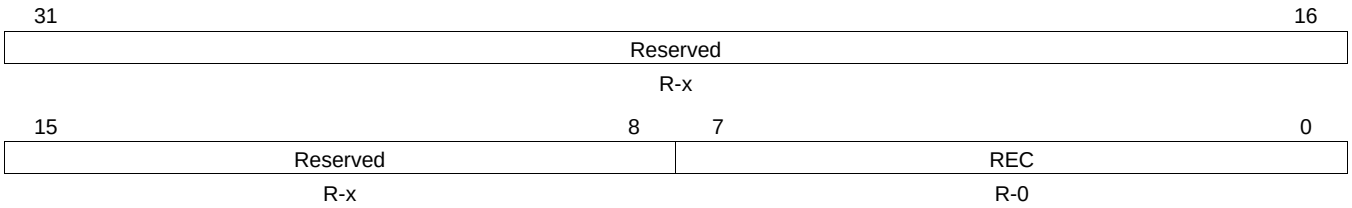
4.1.14 Receive Error Counter Register (CANREC)

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

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

After reaching the bus-off state, CANREC is cleared. CANREC then is 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 (ABO = 1 in CANMC). All internal flags are reset and the error counters are cleared. After leaving the initialization mode, CANREC is cleared. The CANREC is shown in Figure 24.

Figure 24. Receive Error Counter Register (CANREC)



LEGEND: R = Read only; -*n* = value after reset; x = value is indeterminate

4.1.15 Global Interrupt Flag *n* Register (CANGIF0 and CANGIF1)

The global interrupt flag *n* register (CANGIF0 and CANGIF1) allow the CPU to identify the interrupt source. The CANGIF*n* is shown in Figure 25 and described in Table 18.

Figure 25. Global Interrupt Flag *n* Register (CANGIF*n*)

31						18	17	16
Reserved							MAIF*n*(1)	TCOIF*n*(1)
R-x							R/W1C-0	R/W1C-0
15	14	13	12	11	10	9	8	
GMIF*n*	AAIF*n*	WDIF*n*	WUIF*n*	RMLIF*n*	BOIF*n*	EPIF*n*	WLIF*n*	
R-0	R/W1C-0	R/W1C-0	R/W1C-0	R/W1C-0	R/W1C-0	R/W1C-0	R/W1C-0	
7		5	4				0	
Reserved			MIV*n*					
R-0			R-0(1)		R-0			

LEGEND: R/W = Read/Write; R = Read only; W1C = Write 1 to clear; -*n* = value after reset; x = value is indeterminate

(1) HECC only; reserved on the SCC.

Table 18. Global Interrupt Flag *n* Register (CANGIF*n*) Field Descriptions

Bit	Field	Value	Description
31-18	Reserved	0	Reserved. Reads are undefined and writes have no effect.
17	MAIF*n*		Message alarm flag. HECC only, reserved on 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.
16	TCOIF*n*		Local network time counter overflow flag. HECC only, reserved on the SCC.
		0	The MSB of the local network time register (CANLNT) is 0.
		1	The MSB of the local network time register (CANLNT) is 1.
15	GMIF*n*		Global mailbox interrupt flag.
		0	No message has been transmitted or received.
		1	One of the mailboxes transmitted or received a message successfully.
14	AAIF*n*		Abort-acknowledge interrupt flag.
		0	No transmission has been aborted.
		1	A transmission request has been aborted.
13	WDIF*n*		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	WUIF*n*		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	RMLIF*n*		Receive-message-lost interrupt flag.
		0	No message has been lost.
		1	For at least one of the receive mailboxes, an overflow condition has occurred.
10	BOIF*n*		Bus-off interrupt flag.
		0	The CAN module is still in bus-on mode.
		1	The CAN module has entered bus-off mode.

Table 18. Global Interrupt Flag *n* Register (CANGIF*n*) Field Descriptions (continued)

Bit	Field	Value	Description
9	EPIF*n*		Error-passive interrupt flag.
		0	The CAN module is not in error-passive mode.
		1	The CAN module has entered error-passive mode.
8	WLIF*n*		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	0	Reserved. Reads are undefined and writes have no effect.
4-0	MIV*n*	0-Fh	Message object interrupt vector. The MIV*n* bit 4 is for the HECC only and is reserved on the SCC. These bits are reserved and read as 0 in the SCC. Indicates the number of the message object that activated the global mailbox interrupt(GMIF*n*) flag. This field keeps the 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 GMIF*n* flag is 0, the corresponding MIV*n* vector is undefined.

4.1.16 Global Interrupt Mask Register (CANGIM)

The global interrupt mask register (CANGIM) allows the various interrupts to be enabled. If a bit is set in CANGIM, the corresponding interrupt is enabled. CANGIM is write-protected in user mode. The CANGIM is shown in Figure 26 and described in Table 19.

Figure 26. Global Interrupt Mask Register (CANGIM)

31						18	17	16
Reserved							MAIM	TCOIM
R-x							R/W-0	R/W-0
15	14	13	12	11	10	9	8	
Reserved	AAIM	WDIM	WUIM	RMLIM	BOIM	EPIM	WLIM	
R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	
7					3	2	1	0
Reserved						SIL	I1EN	I0EN
R-0						R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset; x = value is indeterminate

Table 19. Global Interrupt Mask Register (CANGIM) Field Descriptions

Bit	Field	Value	Description
31-18	Reserved	0	Reserved. Reads are undefined and writes have no effect.
17	MAIM		Message alarm interrupt mask. HECC only, reserved in the SCC.
		0	The MAIF interrupt is disabled.
		1	The MAIF interrupt is enabled.
16	TCOIM		Local network time counter overflow interrupt mask. HECC only, reserved in the SCC.
		0	The TCOIF interrupt is disabled.
		1	The TCOIF interrupt is enabled.
15	Reserved	0	Reserved. Reads are undefined and writes have no effect.
14	AAIM		Abort acknowledge interrupt mask.
		0	The AAIF interrupt is disabled.
		1	The AAIF interrupt is enabled.
13	WDIM		Write denied interrupt mask.
		0	The WDIF interrupt is disabled.
		1	The WDIF interrupt is enabled.
12	WUIM		Wake-up interrupt mask.
		0	The WUIF interrupt is disabled.
		1	The WUIF interrupt is enabled.
11	RMLIM		Receive message lost interrupt mask.
		0	The RMLIF interrupt is disabled.
		1	The RMLIF interrupt is enabled.
10	BOIM		Bus-off interrupt mask.
		0	The BOIF interrupt is disabled.
		1	The BOIF interrupt is enabled.
9	EPIM		Error passive interrupt mask.
		0	The EPIF interrupt is disabled.
		1	The EPIF interrupt is enabled.
8	WLIM		Warning level interrupt mask.
		0	The WLIF interrupt is disabled.
		1	The WLIF interrupt is enabled.

Table 19. Global Interrupt Mask Register (CANGIM) Field Descriptions (continued)

Bit	Field	Value	Description
7-3	Reserved	0	Reserved. Reads are undefined and writes have no effect.
2	SIL		System interrupt level. Defines the interrupt level for TCOIF, WDIF, WUIF, BOIF, EPIF, AAIF, MAIF, RMLIF, and WLIF.
		0	All global interrupts are mapped to the HECC0INT interrupt line.
		1	All system interrupts are mapped to the HECC1INT interrupt line.
1	I1EN		Interrupt line 1 enable.
		0	The HECC1INT interrupt line is globally disabled.
		1	All interrupts for the HECC1INT interrupt line are globally enabled if the corresponding masks are set.
0	I0EN		Interrupt line 0 enable.
		0	The HECC0INT interrupt line is globally disabled.
		1	All interrupts for the HECC0INT interrupt line are globally enabled if the corresponding masks are set.

4.1.17 Mailbox Interrupt Mask Register (CANMIM)

The mailbox interrupt mask register (CANMIM) is used to enable or disable each mailbox interrupt. 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. CANMIM is write-protected in user mode. The CANMIM is shown in Figure 27 and described in Table 20.

Figure 27. Mailbox Interrupt Mask Register (CANMIM)

31															16
MIM[(1)]															
R/W-0															

15															0
MIM															
R/W-0															

LEGEND: R/W = Read/Write; -*n* = value after reset

(1) Bits 31-16 on the HECC only; reserved on the SCC.

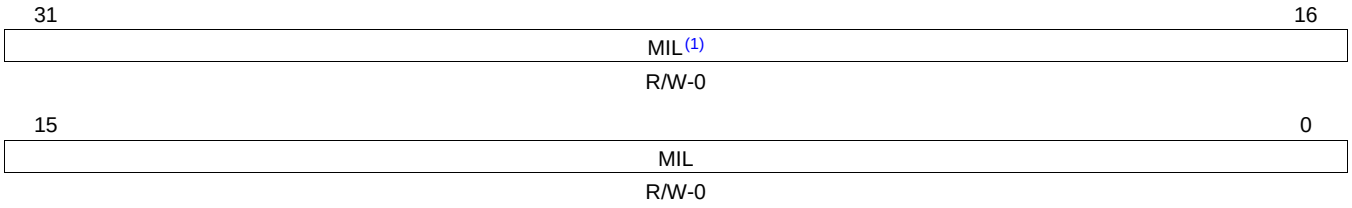
Table 20. Mailbox Interrupt Mask Register (CANMIM) Field Descriptions

Bit	Field	Value	Description
31-0	MIM[*n*]		Mailbox interrupt mask. Selects any mailbox interrupt to be masked individually.
		0	The mailbox interrupt is disabled.
		1	The 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).

4.1.18 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 (MIL[*n*] = 0) or on HECC1INT (MIL[*n*] = 1). The CANMIL is shown in Figure 28 and described in Table 21.

Figure 28. Mailbox Interrupt Level Register (CANMIL)



LEGEND: R/W = Read/Write; -*n* = value after reset

(1) Bits 31-16 on the HECC only; reserved on the SCC.

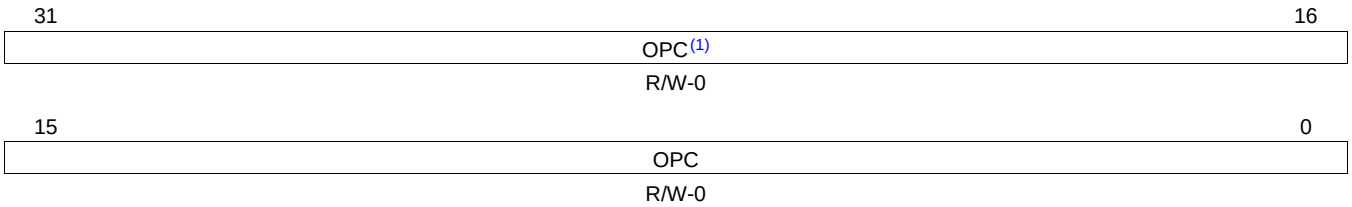
Table 21. Mailbox Interrupt Level Register (CANMIL) Field Descriptions

Bit	Field	Value	Description
31-0	MIL[*n*]		Mailbox interrupt level.
		0	The mailbox interrupt is generated on interrupt line 0 (HECC0INT).
		1	The mailbox interrupt is generated on interrupt line 1 (HECC1INT).

4.1.19 Overwrite Protection Control Register (CANOPC)

If there is an overflow condition for mailbox *n* (RMP[*n*] = 1 in CANRMP and a new receive message would fit for mailbox *n*), the new message is stored depending on the settings in the overwrite protection control register (CANOPC). If the corresponding OPC[*n*] bit 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 OPC[*n*] bit is cleared to 0, the old message is overwritten by the new message. This will be notified by setting the receive message lost (RML[*n*]) bit in CANRML. The CANOPC is shown in Figure 29 and described in Table 22.

Figure 29. Overwrite Protection Control Register (CANOPC)



LEGEND: R/W = Read/Write; -*n* = value after reset

(1) Bits 31-16 on the HECC only; reserved on the SCC.

Table 22. Overwrite Protection Control Register (CANOPC) Field Descriptions

Bit	Field	Value	Description
31-0	OPC[*n*]		Overwrite protection control.
		0	The old message may be overwritten by a new message.
		1	An old message stored in the mailbox is protected against being over-written by a new message.

4.1.20 Transmit I/O Control Registers (CANTIOC)

The transmit I/O control register (CANTIOC) is shown in Figure 30 and described in Table 23.

Note: When the CANTX pin is configured for CAN functions (TXFUNC = 1), the TXIN bit always shows the actual value of the CANTX pin.

Figure 30. Transmit I/O Control Register (CANTIOC)

31	16
Reserved	
R-x	

15	4	3	2	1	0
Reserved		TXFUNC	Reserved		TXIN
R-x		R/W-0	R-x		R-x

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset; x = value is indeterminate

Table 23. Transmit I/O Control Register (CANTIOC) Field Descriptions

Bit	Field	Value	Description
31-4	Reserved	0	Reserved. Reads are undefined and writes have no effect.
3	TXFUNC		External pin I/O enable flag. This bit is protected in user mode.
		0	The CANTX pin is disabled.
		1	The CANTX pin is used for the CAN transmit functions.
2-1	Reserved	0	Reserved. Reads are undefined and writes cause undefined behavior.
0	TXIN		Transmit in. This bit reflects the value on the CANTX pin.
		0	The pin is at logic low (0).
		1	The pin is at logic high (1).

4.1.21 Receive I/O Control Registers (CANRIOC)

The receive I/O control register (CANRIOC) is shown in Figure 31 and described in Table 24.

Note: When the CANRX pin is configured for CAN functions (RXFUNC = 1), the RXIN bit always shows the actual value of the CANRX pin.

Figure 31. Receive I/O Control Register (CANRIOC)

31	16
Reserved	
R-x	

15	4	3	2	1	0
Reserved		RXFUNC	Reserved		RXIN
R-x		R/W-0	R-x		R-x

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset; x = value is indeterminate

Table 24. Receive I/O Control Register (CANRIOC) Field Descriptions

Bit	Field	Value	Description
31-4	Reserved	0	Reserved. Reads are undefined and writes have no effect.
3	RXFUNC		External pin I/O enable flag. This bit is protected in user mode.
		0	The CANRX pin is disabled.
		1	The CANRX pin is used for the CAN receive functions.
2-1	Reserved	0	Reserved. Reads are undefined and writes cause undefined behavior.
0	RXIN		Input value for the CANRX pin. This bit reflects the value on the CANRX pin.
		0	The pin is at logic low (0).
		1	The pin is at logic high (1).

4.1.22 Local Network Time Register (CANLNT)

Note: The local network time register (CANLNT) is available on the HECC only. The offset addresses are reserved in the SCC mode.

The local network time register (CANLNT) is shown in Figure 32 and described in Table 25.

Figure 32. Local Network Time Register (CANLNT)

31	16
LNT	
R/W-x	

15	0
LNT	
R/W-x	

LEGEND: R/W = Read/Write; -*n* = value after reset; x = value is indeterminate

Table 25. Local Network Time Register (CANLNT) Field Descriptions

Bit	Field	Value	Description
31-0	LNT	0–FFFF FFFF	Local network time register. The value of the local network time counter is used for the time-stamp and the time-out functions.

4.1.23 Time-Out Control Register (CANTOC)

Note: The time-out control register (CANTOC) is available on the HECC only. The offset addresses are reserved in the SCC mode.

The TOC[*n*] bit in the time-out control register (CANTOC) has to be set by the CPU to enable the time-out function for mailbox *n*. Before setting the TOC[*n*] bit, the corresponding message object time-out register *n* (MOTO*n*) should be loaded with the time-out value relative to the local network time register (CANLNT). The CANTOC is shown in Figure 33 and described in Table 26.

Figure 33. Time-Out Control Register (CANTOC)

31															16
TOC															
R/W-0															
15															0
TOC															
R/W-0															

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 26. Time-Out Control-Register (CANTOC) Field Descriptions

Bit	Field	Value	Description
31-0	TOC[*n*]		Time-out control register.
		0	The time-out function is disabled. The TOS[*n*] bit in the time-out status register (CANTOS) is never set.
		1	The time-out function is enabled.

4.1.24 Time-Out Status Register (CANTOS)

Note: The time-out status register (CANTOS) is available on the HECC only. The offset addresses are reserved in the SCC mode.

The time-out status register (CANTOS) is shown in Figure 34 and described in Table 27.

Figure 34. Time-Out Status Register (CANTOS)

31															16
TOS															
R/W1C-0															
15															0
TOS															
R/W1C-0															

LEGEND: R/W = Read/Write; W1C = Write 1 to clear; -*n* = value after reset

Table 27. Time-Out Status Register (CANTOS) Field Descriptions

Bit	Field	Value	Description
31-0	TOS[*n*]		Time-out status register.
		0	No time-out occurred or the time-out function is disabled for that mailbox.
		1	The value in the local network time register (CANLNT) is larger or equal to the value in the time-out register for the corresponding mailbox *n* and the TOC[*n*] bit in the time-out control register (CANTOC) is set.

4.1.25 Error Test Control Register (CANETC)

All the bits of the error test control register (CANETC) are functional in error test mode only. When error test mode is enabled, the CAN module is configured in loopback mode automatically, except when the BEN bit is set and the CAN module is in receive mode or when the CRCEN bit is set. In these cases, loopback mode is disabled automatically and you should set the CAN module in receive mode. The CANETC is shown in Figure 35 and described in Table 28.

Figure 35. Error Test Control Register (CANETC)

31	25	24	23	22	21	20	19	16
Reserved		FEN	BEN	SA1E	CRCEN	SEN	Reserved	
R-x		R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-x	

15	12	11	8	7	4	3	0
Reserved		CANETCEN		FIELD_SEL		Reserved	
R-x		R/W-5h		R/W-0		R-x	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset; x = value is indeterminate

Table 28. Error Test Control Register (CANETC) Field Descriptions

Bit	Field	Value	Description
31-25	Reserved	0	Reserved. Reads are undefined and writes have no effect.
24	FEN		Form error enable.
		0	No form error will be generated.
		1	A form error will be created. The bit received during the fixed format field is toggled and passed to the error checking circuitry. Using the FIELD_SEL bits, a form error can be verified in the following fields: • CRC delimiter field • Acknowledge delimiter field • End-of-frame field
23	BEN		Bit error enable.
		0	No bit error will be generated.
		1	A bit error will be created. The received bit is XORed with 1 and passed to the bit monitor circuitry. Using the FIELD_SEL bits, a bit error (during transmit mode) can be verified in the following fields: • CRC field • Data field • Data length control field During receive mode, bit error is verified in ACK slot field. In this mode, the CAN module will not be in loopback mode. It should be set in receive mode by the user.
22	SA1E		Stuck at dominant error enable.
		0	No stuck at dominant (SA1) error will be generated.
		1	The received bit is ANDed with 0 to generate a stuck at dominant (SA1) error.
21	CRCEN		CRC error enable.
		0	No CRC error will be generated.
		1	The CRC bits received as well as the fields used for CRC calculation are masked and passed to the CRC check circuitry. Using the FIELD_SEL bits, a CRC error can be verified in the following fields: • STDID field • EXTID field • Data field • CRC field For verifying CRC errors, the CAN module will not be in loopback mode and should be set in receive mode.

Table 28. Error Test Control Register (CANETC) Field Descriptions (continued)

Bit	Field	Value	Description
20	SEN		Stuff error enable.
		0	No stuff error will be generated.
		1	The stuff bits received are toggled and passed to the error monitor circuitry. Using the FIELD_SEL bits, a stuff error can be verified in the following fields: • STDID field • EXTID field • Data field
19-12	Reserved	0	Reserved. Reads are undefined and writes have no effect.
11-8	CANETCENA	0-Fh	Test register enable key.
		0-4h	CANETC is disabled.
		5h	CANETC is enabled.
		6h-Fh	CANETC is disabled.
7-4	FIELD_SEL	0-Fh	CAN field selects.
		0	No field is selected.
		1	The STDID field is selected.
		2h	The DATALENGTHCODE field is selected.
		3h	The DATA field is selected.
		4h	The CRC field is selected.
		5h	The CRC delimiter field is selected.
		6h	The ACKSLOT field is selected.
		7h	The ACKDEL field is selected.
		8h	The ENDOFFRAME field is selected.
		9h	The EXTID field is selected.
		Ah-Fh	Reserved
3-0	Reserved	0	Reserved. Reads are undefined and writes have no effect.

4.2 Message Object Registers

Table 29 lists the HECC/SCC message object registers. See Section 2.7 for details on message objects and mailboxes.

Table 29. HECC/SCC Message Object Registers

Offset[1]	Acronym	Register Description	Section
SCC_Offset + 80h (for objects 0 to 2)	SCCLAM0	SCC local acceptance mask register 0	Section 4.2.1
SCC_Offset + 8Ch (for objects 3 to 5)	SCCLAM3	SCC local acceptance mask register 3	Section 4.2.1
SCC_Offset + 24h (for objects 6 to 15)	CANGAM	Global acceptance mask register	Section 4.2.2
Mailbox_Offset + (Object# × 10h) + 00h	MID*n*	Message identifier registers	Section 4.2.3
Mailbox_Offset + (Object# × 10h) + 04h	MCF*n*	Message control field registers	Section 4.2.4
Mailbox_Offset + (Object# × 10h) + 08h	MDL*n*	Message data low-word registers	Section 4.2.5.1
Mailbox_Offset + (Object# × 10h) + 0Ch	MDH*n*	Message data high-word registers	Section 4.2.5.2
Mailbox_Offset + (Object# × 4) + 1000h	LAM*n*	HECC local acceptance mask registers	Section 4.2.6
Mailbox_Offset + (Object# × 4) + 1080h	MOTS*n*	Message object time-stamp registers	Section 4.2.7
Mailbox_Offset + (Object# × 4) + 1100h	MOTO*n*	Message object time-out registers	Section 4.2.8

(1) The SCC_Offset address location is referenced in the configuration memory-map summary of the data manual as the HECC Control Region.
The Mailbox_Offset address location is referenced in the configuration memory-map summary of the data manual as the HECC RAM Region.

4.2.1 SCC Local Acceptance Mask Register *n* (SCCLAM0 and SCCLAM3)

The SCC local acceptance mask registers (SCCLAM0 and SCCLAM3) are used by the HECC in SCC-compatible mode. The SCC local acceptance filtering allows you to locally mask any identifier bits of the incoming message. In the SCC, the local acceptance mask register 0 (SCCLAM0) is used for mailboxes 2-0 and the local acceptance mask register 3 (SCCLAM3) is used for mailboxes 5-3. For mailboxes 15-6, the global acceptance mask register (CANGAM) is used (see Section 4.2.2). The SCCLAM*n* is shown in Figure 36 and described in Table 30.

After a hardware or a software reset of the SCC, SCCLAM0 and SCCLAM3 are reset to 0.

Figure 36. SCC Local Acceptance Mask Register *n* (SCCLAM*n*)

31	30	29	28	16
LAMI	Reserved		LAM	
R/W-0	R-x		R/W-0	

15	0
LAM	
R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset; x = value is indeterminate

Table 30. SCC Local Acceptance Mask Register *n* (SCCLAM*n*) Field Descriptions

Bit	Field	Value	Description
31	LAMI		Local acceptance mask identifier extension.
		0	The identifier extension bit stored in the mailbox determines which messages shall be received.
		1	Standard and extended frames can be received. In the 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 the case of a standard frame, only the first 11 bits (bits 28-18) of the identifier and the local acceptance mask are used.
30-29	Reserved	0	Reserved. Reads are undefined and writes have no effect.
28-0	LAM[*n*]		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 in the message identifier register (MID).
		1	Accept a 0 or a 1 (don''t care) for the corresponding bit of the received identifier.

4.2.2 Global Acceptance Mask Register (CANGAM)

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

The CANGAM is shown in Figure 37 and described in Table 31.

Figure 37. Global Acceptance Mask Register (CANGAM)

31	30	29	28	16
AMI	Reserved		GAM	
R/W-0	R-0		R/W-0	

15	0
GAM	
R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

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

Bit	Field	Value	Description
31	AMI		Acceptance mask identifier extension.
		0	The identifier extension bit stored in the mailbox determines which messages shall be received.
		1	Standard and extended frames can be received. In the case of an extended frame, all 29 bits of the identifier are stored in the mailbox and all 29 GAM bits are used for the filter. In the case of a standard frame, only the first 11 bits (bits 28-18) of the identifier and the GAM bits are used.
30-29	Reserved	0	Reserved. Reads are undefined and writes have no effect.
28-0	GAM[*n*]		Global acceptance mask. These bits allow any identifier bits of an incoming message to be masked.
		0	The received identifier bit value must match the corresponding identifier bit in the message identifier register (MID).
		1	A 0 or a 1 (don't care) will be accepted for the corresponding bit of the received identifier.

4.2.3 Message Identifier Register *n* (MID0-MID31)

Note: The message identifier registers (MID0-MID15) are available in both the HECC and SCC modes; however, MID16-MID31 are only available in HECC mode and the offset addresses are reserved in the SCC mode.

The message identifier register *n* (MID*n*) can only be written if mailbox *n* is disabled (ME[*n*] = 0 in the mailbox enable register (CANME)). MID*n* is shown in Figure 38 and described in Table 32.

Figure 38. Message Identifier Register *n* (MID*n*)

31	30	29	28	16
IDE	AME	AAM	ID	
R/W-x	R/W-x	R/W-x	R/W-x	

15	0
ID	
R/W-x	

LEGEND: R/W = Read/Write; -*n* = value after reset; x = value is indeterminate

Table 32. Message Identifier Register *n* (MID*n*) Field Descriptions

Bit	Field	Value	Description
31	IDE		Identifier extension
		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. AME is only used for receive mailboxes. It must not be set for automatic reply (AAM[*n*] = 1, MD[*n*] = 0 in the mailbox direction register); otherwise, the mailbox behavior is undefined. AME is not modified by a message reception.
		0	No acceptance mask is used, all identifier bits must match to receive the message.
		1	The corresponding acceptance mask is used.
29	AAM		Auto answer mode. AAM is only valid for message mailboxes configured as transmit; for receive mailboxes, AAM has no effect (the mailbox is always configured for normal receive operation). AAM 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	0-1FFF FFFFh	Message identifier. In standard identifier mode (IDE = 0), the message identifier is stored in ID bits 28-18 and ID bits 17-0 have no meaning. In extended identifier mode (IDE = 1), the message identifier is stored in ID bits 28-0.

4.2.4 Message Control Field Register *n* (MCF0-MCF31)

Note: The message control field registers (MCF0-MCF15) are available in both the HECC and SCC modes; however, MCF16-MCF31 are only available in HECC mode and the offset addresses are reserved in the SCC mode.

The message control field register *n* (MCF*n*) can only be written if mailbox *n* is configured for transmission (MD[*n*] = 0 in the mailbox direction register (CANMD)) or if mailbox *n* is disabled (ME[*n*] = 0 in the mailbox enable register (CANME)). MCF*n* is shown in Figure 39 and described in Table 33.

Figure 39. Message Control Field Register *n* (MCF*n*)

31	16
Reserved	
R-0	

15	14	13	8	7	5	4	3	0
Reserved		TPL[1]		Reserved		RTR	DLC	
R/W-x		R/W-x		R-0		R/W-x	R/W-x	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset; x = value is indeterminate

(1) HECC only; reserved on the SCC.

Table 33. Message Control Field Register *n* (MCF*n*) Field Descriptions

Bit	Field	Value	Description
31-14	Reserved	0	Reserved. Reads return 0 and writes have no effect.
13-8	TPL	0-3Fh	Transmit priority level. This 6-bit field in the HECC (reserved on 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	0	Reserved. Reads return 0 and writes have no effect.
4	RTR		Remote transmission request.
		0	No remote frame is requested.
		1	For a receive mailbox, if TRS[*n*] = 1 in the transmission request set register (CANTRS), a remote frame is transmitted and the corresponding data frame will be received in the same mailbox. For a transmit mailbox, if TRS[*n*] = 1, a remote frame is transmitted but the corresponding data frame has to be received in another mailbox.
3-0	DLC	0-Fh	Data length code. This 4-bit field determines the number of data bytes sent or received. Valid values are from 0-8; values from 9-15 are not allowed.

4.2.5 Message Data Registers (MDL*n* and MDH*n*)

Eight bytes of the mailbox are used to store the data field of a CAN message. The setting of the DBO bit in the master control register (CANMC) determines the ordering of stored data. The data is transmitted or received from the CAN bus, starting with byte 0.

- When DBO = 1, the data is stored or read starting with the least-significant byte of the message data low-word register *n* (MDL*n*) and ending with the most-significant byte of the message data high-word register *n* (MDH*n*).
- When DBO = 0, the data is stored or read starting with the most-significant byte of the message data low-word register *n* (MDL*n*) and ending with the least-significant byte of the message data high-word register *n* (MDH*n*).

The MDL*n* and MDH*n* can only be written if mailbox *n* is configured for transmission (MD[*n*] = 0 in the mailbox direction register (CANMD)) or if mailbox *n* is disabled (ME[*n*] = 0 in the mailbox enable register (CANME)).

If TRS[*n*] = 1 in the transmission request set register (CANTRS), MDL*n* and MDH*n* cannot be written unless the CDR bit in CANMC is set to 1 with the MBNR bits in CANMC set to *n*. These settings also apply for a message object configured in reply mode (AAM = 1 in the message identifier register (MID)).

Note: Data Field

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

4.2.5.1 Message Data Low-Word Register n (MDL0-MDL31)

Note: The message data low-word registers (MDL0-MDL15) are available in both the HECC and SCC modes; however, MDL16-MDL31 are only available in HECC mode and the offset addresses are reserved in the SCC mode.

The message data low-word register *n* (MDL*n*) is shown in Figure 40 for DBO = 0 in CANMC and is shown in Figure 41 for DBO = 1 in CANMC.

Figure 40. Message Data Low-Word Register *n* (MDL*n*) with DBO = 0

31	24	16
Byte 0	Byte 1	
R/W-x	R/W-x	

15	8	0
Byte 2	Byte 3	
R/W-x	R/W-x	

LEGEND: R/W = Read/Write; -*n* = value after reset; x = value is indeterminate

Figure 41. Message Data Low-Word Register *n* (MDL*n*) with DBO = 1

31	24	16
Byte 3	Byte 2	
R/W-x	R/W-x	

15	8	0
Byte 1	Byte 0	
R/W-x	R/W-x	

LEGEND: R/W = Read/Write; -*n* = value after reset; x = value is indeterminate

4.2.5.2 Message Data High-Word Register n (MDH0-MDH31)

Note: The message data high-word registers (MDH0-MDH15) are available in both the HECC and SCC modes; however, MDH16-MDH31 are only available in HECC mode and the offset addresses are reserved in the SCC mode.

The message data high-word register *n* (MDH*n*) is shown in Figure 42 for DBO = 0 in CANMC and is shown in Figure 43 for DBO = 1 in CANMC.

Figure 42. Message Data High-Word Register *n* (MDH*n*) with DBO = 0

31	24	23	16
Byte 4		Byte 5	
R/W-x		R/W-x	

15	8	7	0
Byte 6		Byte 7	
R/W-x		R/W-x	

LEGEND: R/W = Read/Write; -*n* = value after reset; x = value is indeterminate

Figure 43. Message Data High-Word Register *n* (MDH*n*) with DBO = 1

31	24	23	16
Byte 7		Byte 6	
R/W-x		R/W-x	

15	8	7	0
Byte 5		Byte 4	
R/W-x		R/W-x	

LEGEND: R/W = Read/Write; -*n* = value after reset; x = value is indeterminate

4.2.6 HECC Local Acceptance Mask Register *n* (LAM0-LAM31)

Note: The HECC local acceptance mask registers (LAM0-LAM31) are available on the HECC only. The offset addresses are reserved in the SCC mode.

The HECC local acceptance filtering allows you to locally mask any identifier bits of the incoming message. In the HECC, each of the 32 mailboxes (0-31) has its own local acceptance mask register *n* (LAM*n*). There is no global acceptance mask register in the HECC. An incoming message is stored in the highest numbered mailbox with a matching identifier. The HECC local acceptance mask register *n* (LAM*n*) is shown in Figure 44 and described in Table 34.

After a hardware or a software reset of the HECC, LAM*n* is not modified.

Figure 44. HECC Local Acceptance Mask Register *n* (LAM*n*)

31	30 29	28 16
LAMI	Reserved	LAM
R/W-0	R-x	R/W-0

15 0
LAM
R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset; x = value is indeterminate

Table 34. HECC Local Acceptance Mask Register *n* (LAM*n*) Field Descriptions

Bit	Field	Value	Description
31	LAMI		Local acceptance mask identifier extension.
		0	The identifier extension bit stored in the mailbox determines which messages shall be received.
		1	Standard and extended frames can be received. In the 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 the case of a standard frame, only the first 11 bits (bits 28-18) of the identifier and the local acceptance mask are used.
30-29	Reserved	0	Reserved. Reads are undefined and writes have no effect.
28-0	LAM[*n*]		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 in the message identifier register (MID).
		1	Accept a 0 or a 1 (don''t care) for the corresponding bit of the received identifier.

4.2.7 Message Object Time-Stamp Register *n* (MOTS0-MOTS31)

Note: The message object time-stamp registers (MOTS0-MOTS31) are available on the HECC only. The offset addresses are reserved in the SCC mode.

The message object time-stamp register *n* (MOTS*n*) contains the value of the local network time register (CANLNT) when the message has been actually received or transmitted. MOTS*n* is shown in Figure 45 and described in Table 35.

Figure 45. Message Object Time-Stamp Register *n* (MOTS*n*)

31	16
MOTS	
R/W-x	

15	0
MOTS	
R/W-x	

LEGEND: R/W = Read/Write; -*n* = value after reset; x = value is indeterminate

Table 35. Message Object Time-Stamp Register *n* (MOTS*n*) Field Descriptions

Bit	Field	Value	Description
31-0	MOTS	0-FFFF FFFFh	Message object time-stamp register. HECC only. Value of the local network time register (CANLNT) when the message has been actually received or transmitted.

4.2.8 Message Object Time-Out Register *n* (MOTO0-MOTO31)

Note: The message object time-out registers (MOTO0-MOTO31) are available on the HECC only. The offset addresses are reserved in the SCC mode.

The message object time-out register *n* (MOTO*n*) contains the limit value of the local network time register (CANLNT) to receive or transmit the message. MOTO*n* is shown in Figure 46 and described in Table 36.

Figure 46. Message Object Time-Out Register *n* (MOTO*n*)

31	16
MOTO	
R/W-x	

15	0
MOTO	
R/W-x	

LEGEND: R/W = Read/Write; -*n* = value after reset; x = value is indeterminate

Table 36. Message Object Time-Out Register (MOTO) Field Descriptions

Bit	Field	Value	Description
31-0	MOTO	0-FFFF FFFFh	Message object time-out register. Limit value of the local network time register (CANLNT) to actually transmit or receive the message.

Appendix A Revision History

Table A-1 lists the changes made since the previous version of this document.

Table A-1. Document Revision History

Reference	Additions/Modifications/Deletions
Section 2.7.3.2	Added Note.
Section 2.11.1	Deleted last sentence from first paragraph.
Figure 9	Changed figure.
Figure 10	Changed figure.
Section 2.11.2	Changed last sentence in fourth paragraph.
	Changed last sentence in fifth paragraph.
	Changed third sentence in last paragraph.
Table 19	Changed SIL bit 2 Description.

IMPORTANT NOTICE

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

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

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

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

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

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

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

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
RFID	www.ti-rfid.com	Telephony	www.ti.com/telephony
Low Power Wireless	www.ti.com/lpw	Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

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