D_CAN

Controller Area Network

User's Manual

Revision 1.11

20.05.2010

Robert Bosch GmbH
Automotive Electronics

SPECIFICATION REVISION HISTORY

REVISION	DATE	NOTES
0.90	24.03.2006	initial working revision (AE/EIS3 Tl)
0.91	28.03.2006	corrections in layout (AE/EIS3 Tl)
0.92	12.05.2006	removed scan-ports from generic interface, renamed ram-signals at generic interface, added description of Function register and core release register (AE/EIP5 Mo)
0.93	20.07.2006	renaming to user's manual and removing module integration specific chapters. Corrections in layout. (AE/EIP5 Mo)
0.94	31.07.2006	include corrections, added register map, autoinc function and clear bit in IFx-CMR, changes at format (AE/EIP5 Mo)
0.95	22.09.2006	added hws register, changed behavior in case of autoinc is one. (AE/EIP5 Mo)
0.96	09.11.2006	modified the dma-description (AE/EIP5 Mo)
0.97	13.12.2006	removed chapter 3 and 4 from user's manual, correction of some register names reset values, moving chapter 2.3.3 to chapter 1.8 (AE/EIP5 Mo)
0.98	18.12.2006	spelling and layout corrections (AE/EIP5 Mo)
1.00	19.12.2006	only revision change (AE/EIP5 Mo)
1.01	25.10.2007	spelling corrections (AE/EIP5 Mo)
1.10	03.12.2007	revision change and conformance date added (AE/EIP5 Mo)
1.11	12.05.2010	CRR updated, description power down mode enhanced, description of Message Object reconfiguration clarified (AE/EIY2 Ht)

TRACKING OF MAJOR CHANGES

TERMS AND ABBREVIATIONS

This document uses the following terms and abbreviations.

Term	Meaning
CAN	Controller Area Network
BSP	Bit Stream Processor
BTL	Bit Timing Logic
CRC	Cyclic Redundancy Check
DLC	Data Length Code
EML	Error Management Logic
FSE	Frame Synchronization Entity
FSM	Finite State Machine
MO	Message Object

CONVENTIONS

The following conventions are used within this User's Manual.

Helvetica bold	Names of bits and ports
Helvetica italic	States of bits and ports

Table of contents

Chapter 1.

1. Overview

The D_CAN is a CAN IP module that can be integrated as stand-alone device or as part of an SoC or ASIC. It is described in VHDL on RTL level, prepared for synthesis. It consists of the components (see figure 1) Mux, CAN_Core, Message RAM interface, Message Handler, Registers and Message Object (MO) access, Module Interface.

The D_CAN performs CAN protocol communication according to ISO 11898-1 (identical to Bosch CAN protocol specification 2.0 A, B). The bit rate can be programmed to values up to 1 MBit/s depending on the used technology. Additional transceiver hardware is required for the connection to the physical layer (the CAN bus line).

For communication on a CAN network, individual Message Objects are configured. The Message Objects and Identifier Masks are stored in the Message RAM.

All functions concerning the handling of messages are implemented in the Message Handler. Those functions are acceptance filtering, transfer of messages between the CAN_Core and the Message RAM and the handling of transmission requests as well as the generation of the module interrupt.

The register set of the D_CAN can be accessed directly by an external CPU via the module host interface. These registers are used to control/configure the CAN_Core and the Message Handler and to access the message RAM via the IF1 and IF2 register sets.

1.1 Features

* Supports CAN protocol version 2.0 part A, B
* Bit rates up to 1 MBit/s
* Dual clock source, enabling FM-PLL designs
* 16, 32, 64 or 128 Message Objects (configurable during synthesis)
* Each Message Object has its own Identifier Mask
* Programmable FIFO mode for Message Objects
* Programmable loop-back modes for self-test operation
* Parity check mechanism for all RAM modules (optional)
* 2 Interrupt lines
* DMA support with automatic Message Object increment
* Power-down support
* RAM initialization

1.2 Block Diagram

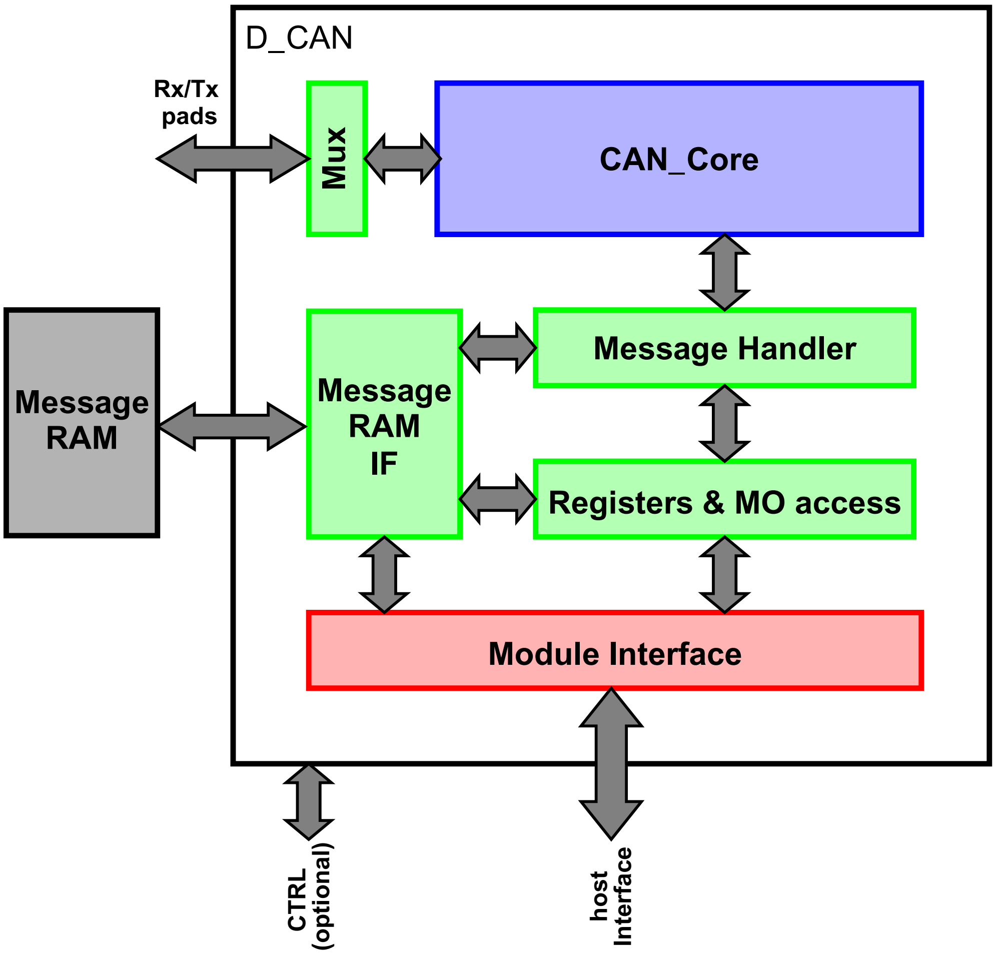

Figure 1 Block Diagram of the D_CAN

CAN_Core

CAN Protocol Controller and Rx/Tx Shift Register, handles all ISO 11898-1 protocol functions.

Mux

This multiplexer controls the functionality of the two CAN ports, that is:
- transmit & receive lines for normal CAN communication
- configurable self test features, when test mode is enabled

Message Handler

State Machine that controls the data transfer between the single ported Message RAM and the CAN Core's Rx/Tx Shift Register. It also handles acceptance filtering and the interrupt setting as programmed in the Control and Configuration Registers.

Message RAM

Single ported RAM, word-length = [CAN message & acceptance filter mask & control bits & status bits] 136 bits + 5 bits parity (optional).

Registers & MO access

Status and configuration registers for module setup and indirect Message Object (MO) access to ensure data consistency; all CPU accesses to the Message RAM are relayed through CPU IFC registers that have the same word-length as the Message RAM.

Module Interface

The D_CAN module is equipped with a generic 32-bit interface. The customer specific interface is a wrapper one hierarchy level higher, implementing a "generic interface" to "host interface" bridge.

1.2.1 Optional Control Ports

The D_CAN Module has some optional control ports which can be accessed by on-chip function registers of other modules. These ports must not be connected in each application.

port	*Direction*	*Description*
CAN_INT_STATUS	Out	Interrupt request, first line, level sensitive, active high (Error-, Status-, MO-Interrupts)
CAN_INT_MO	Out	Interrupt request, second line, level sensitive, active high (MO-Interrupts only)
CAN_IF1DMA	Out	DMA transfer request of IF1, level sensitive, active high.
CAN_IF2DMA	Out	DMA transfer request of IF2, level sensitive, active high.
CAN_UERR	Out	A parity error has detected, pulse, active high.

Table 1 optional control ports

port	*Direction*	*Description*
CAN_CLKSTOP_REQ	In	Requests, that D_CAN clock should be switched off. All pending transfers will be handled and after waiting for 11 Recessive Bits on the CAN-BUS, **Init**-Bit will be set.
CAN_CLKSTOP_ACK	Out	Acknowledge that the D_CAN clock can be switched off.
CAN_RAMINIT_REQ	In	Starts the Message RAM initialization
CAN_RAMINIT_ACK	Out	Acknowledging RAM initialization has finished
CAN_INIT	Out	Indicate that the D_CAN is init-mode, no CAN communication is possible. '0' normal operation mode.

Table 1 optional control ports

1.3 Operating Modes

1.3.1 Software initialization

The software initialization is started by setting the bit **CCTRL.Init**, either by software or by a hardware reset, or by going *Bus_Off*.

While **CCTRL.Init** is set, message transfer from and to the CAN bus is stopped, the status of the CAN bus output **CAN_TXD** is *recessive* (HIGH). The counters of the EML are unchanged. Setting **CCTRL.Init** does not change any configuration register.

To initialize the CAN Controller, the CPU has to set up the Bit Timing Register (**CBT**) and those Message Objects which have to be used for CAN communication. If a Message Object is not needed, it is sufficient to let **MsgVal** bit not valid (LOW), which is the default after RAM initialization. It is mandatory to setup the whole Message Object before setting **MsgVal** to valid.

Access to the Bit Timing Register (**CBT**) for the configuration of the bit timing is only enabled when both bits **CCTRL.Init** and **CCTRL.CCE** are set.

Resetting **CCTRL.Init** finishes the software initialization. Afterwards the Bit Stream Processor BSP (see Application Note 001 "Configuration of Bit Timing") synchronizes itself to the data transfer on the CAN bus by waiting for the occurrence of a sequence of 11 consecutive *recessive* bits (≡ *Bus Idle*) before it can take part in bus activities and starts the message transfer.

The initialization of the Message Objects is independent of **CCTRL.Init** and can be done anytime, but the Message Objects should all be configured to particular identifiers or set to not valid before the BSP starts the message transfer. On power up the RAM has to be initialized (see Chapter 2.2.6). During RAM initialization the parity bits will be generated. Accesses while or before RAM initialization are not allowed and could result in parity errors or other side effects.

1.3.2 CAN Message Transfer

Once the D_CAN is initialized and **CCTRL.Init** is reset to zero, the CAN_Core synchronizes itself to the CAN bus and starts the message transfer.

Received messages are stored into their appropriate Message Objects if they pass the Message Handler's acceptance filtering. The whole message including all arbitration bits, **Xtd**, **Dir**, **DLC**, and eight data bytes, as well as the mask bits and control bits **UMask**, **MXtd**, **MDir**, **EoB**, **MsgLst**, **RxIE**, **TxIE**, **RmtEn** is stored into the Message Object. In consequence, using e.g. the Identifier Mask, the arbitration bits which are masked to "don't care" may change in the Message Object when a received message is stored.

The CPU may read or update each message any time via the Interface Registers. The Message Handler guarantees data consistency in case of concurrent accesses (for reconfiguration see Chapter 1.8.4).

Messages to be transmitted are updated by the CPU. If a permanent Message Object (arbitration and control bits set up during configuration and leaving unchanged for multiple CAN transfers) exists for the message, only the data bytes have to be updated. If several transmit messages are assigned to the same Message Object (when the number of Message Objects is not sufficient), the whole Message Object has to be configured before the transmission of this message is requested.

The transmission of any number of Message Objects may be requested at the same time, they are transmitted subsequently according to their internal priority (The message object numbers are from 1 to configurable up to 128, as lower is the message object number, as higher is the internal priority). Messages may be updated or set to not valid any time, even when their requested transmission is still pending (for reconfiguration see Chapter 1.8.5). The old data will be discarded when a message is updated before its pending transmission has started.

Depending on the configuration of the Message Object, the transmission of a message may be requested autonomously by the reception of a remote frame with a matching identifier.

1.3.3 Disabled Automatic Retransmission

According to the CAN Specification (see ISO11898-1, 6.3.3 Recovery Management), the D_CAN provides means for automatic retransmission of frames that have lost arbitration or that have been disturbed by errors during transmission. The frame transmission service will not be confirmed to the user before the transmission is successfully completed. By default, this means for automatic retransmission is enabled. Further details to DAR mode are provided by Application Note 004 "CAN Operation".

1.3.4 D_CAN power down (Sleep Mode)

The D_CAN module can be set to power down mode controlled by port **CAN_CLKSTOP_REQ** or by application register flag **CFR.ClkStReq**. The D_CAN waits for the completion of all pending transmit requests of the Message Objects. When all requests are completed, D_CAN waits until a bus idle state is recognized. The D_CAN sets then **CCTRL.Init** bit to *one* to prevent any further CAN-transfers, afterwards the D_CAN acknowledges readiness for power down by setting **CAN_CLKSTOP_ACK** to *one* and **CFR.ClkStAck** to *one*. After **CAN_CLKSTOP_ACK** gets *one*, further register accesses can be made by leaving host clock on. A write request to the **CCTRL.Init** bit by application will have no effect. If data transfer between D_CAN and CPU is complete, clock can be switched off. Even if an interrupt was activated due to internal D_CAN events made to get ready for power down, the interrupt service routine can be processed.

To leave power down mode, the application has to turn on the module clocks before resetting signal **CAN_CLKSTOP_REQ** resp. application register flag **CFR.ClkStReq**. The D_CAN will acknowledge this by resetting output signal **CAN_CLKSTOP_ACK** and resetting **CFR.ClkStAck**. Afterwards, the application can restart CAN communication by resetting bit **CCTRL.Init**.

1.3.5 Test Modes

To enable the test mode, bit **CCTRL.Test** has to be set to *one*. This activates the write access to the „Test Register“ (see Chapter 2.2.5).

Note: Test modes should be used for production tests or self test only. It is not recommended to use test modes for main application.

The next paragraphs describe the functionality of the test bits **CTR.Silent**, **CTR.LBack** and **CTR.ExL**, **CTR.Tx0** and **CTR.Tx1** only.

1.3.5.1 Silent Mode

The CAN_Core is set in Silent Mode by programming the bit **CTR.Silent** to *one.*

In Silent Mode, the D_CAN is able to receive valid data frames and valid remote frames, but it sends only *recessive* bits on the CAN bus. If the D_CAN is required to send a *dominant* bit (ACK bit, overload flag, active error flag), the bit is rerouted internally so that the D_CAN monitors this *dominant* bit, although the CAN bus may remain in *recessive* state. The Silent Mode can be used to analyze the traffic on a CAN bus without affecting it by the transmission of *dominant* bits (Acknowledge Bits, Error Frames). Figure 2 shows the connection of signals **CAN_TXD** and **CAN_RXD** to the CAN_Core in Silent Mode.

In ISO 11898-1, the Silent Mode is called the Bus Monitoring Mode.

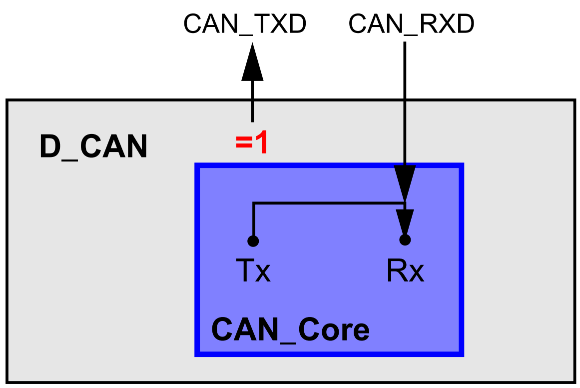

Figure 2 CAN_Core in Silent Mode

1.3.5.2 Loop Back Mode

The CAN_Core can be set in Loop Back Mode by programming the bit **CTR.LBack** to *one*. In Loop Back Mode, the CAN_Core treats its own transmitted messages as received messages and stores them (if they pass acceptance filtering) into a Receive Buffer. Figure 3 shows the connection of signals **CAN_TXD** and **CAN_RXD** to the CAN_Core in Loop Back Mode.

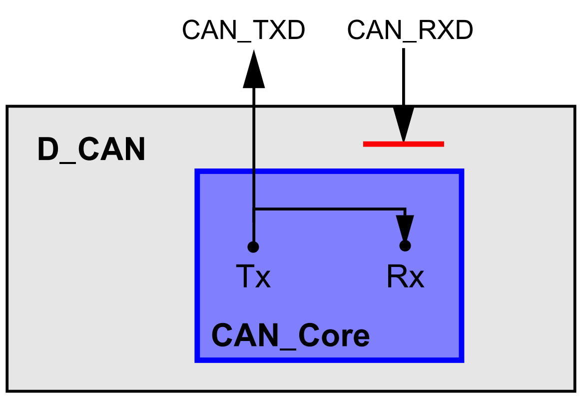

Figure 3 CAN_Core in Loop Back Mode

This mode is provided for hardware self-test functions. To be independent from external stimulation, the CAN_Core ignores acknowledge errors (recessive bit sampled in the acknowledge slot of a data/remote frame) in Loop Back Mode. In this mode the CAN_Core performs an internal feedback from its Tx output to its Rx input. The actual value of the **CAN_RXD** input pin is disregarded by the CAN_Core. The transmitted messages can be monitored at the **CAN_TXD** pin.

1.3.5.3 Loop Back combined with Silent Mode

It is also possible to combine Loop Back Mode and Silent Mode by programming bits **CTR.LBack** and **CTR.Silent** to *one* at the same time. This mode can be used for a “Hot Selftest”, meaning the D_CAN hardware can be tested without affecting a running CAN system connected to the pins **CAN_TXD** and **CAN_RXD**. In this mode the **CAN_RXD** pin is disconnected from the CAN_Core and the **CAN_TXD** pin is held *recessive*. Figure 4 shows the connection of signals **CAN_TXD** and **CAN_RXD** to the CAN_Core in case of the combination of Loop Back Mode with Silent Mode.

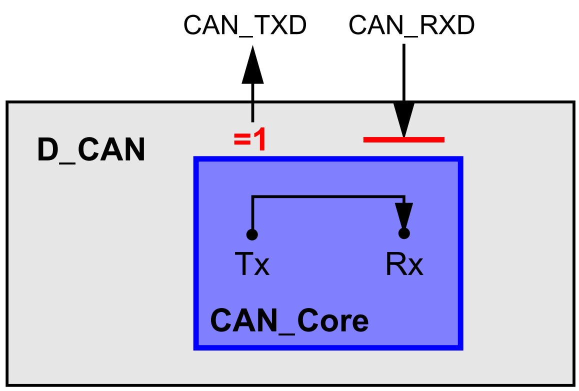

Figure 4 CAN_Core in Loop Back combined with Silent Mode

Note: After message transmission in Loop Back Mode CSTS.TxOK is set, CSTS.RxOK is not set.

1.3.5.4 Software control of Pin CAN_TXD

Four output functions are available for the CAN transmit pin **CAN_TXD**. Additionally to its default function – the serial data output – it can drive the CAN Sample Point signal to monitor the CAN_Core’s bit timing and it can drive constant dominant or recessive values. The last two functions, combined with the readable CAN receive pin **CAN_RXD**, can be used to check the CAN bus’ physical layer.

The output mode of pin **CAN_TXD** is selected by programming the bits **CTR.Tx1** and **CTR.Tx0** as described in Chapter 2.2.5.

Note: The software control for pin CAN_TXD interfere with all CAN protocol functions. CAN_TXD must be left in its default function when CAN message transfer or any of the test modes Loop Back Mode, External Loop Back Mode or Silent Mode should be used.

1.4 Dual Clock Sources

To improve the EMC behavior, a spread spectrum clock can be used for the host clock domain. Due to the high precision clocking requirements of the CAN Core, a separate clock without any modulation has to be provided as **CAN_CLK**. The CAN core should be programmed to have at least 8 clocks per bittime, this is e.g. 1 Mbaud @ **CAN_CLK**>=8 MHz. Even if the host clock (**HOST_CLK**) is very fast, the clock frequency of the CAN core need not to be higher than 8 MHz.

Between the two clock domains within the D_CAN module there is a synchronization mechanism implemented to ensure save data transfer.

Note: In order to archive a stable function of the D_CAN, host clock must always be faster or equal to CAN clock. Also the modulation depth of the spread spectrum clock has to be regarded.

1.5 Dual Interrupts lines

The module provides two interrupt lines. Message Object interrupts can be routed either to the **CAN_INT_STATUS** or to the **CAN_INT_MO** line. The Error and Status interrupts can be observed on **CAN_INT_STATUS** only. By default the Message Object interrupts are routed to the interrupt line **CAN_INT_STATUS**. By setting the **CCTRL.MIL** all message object interrupts (**IntPnd**) are routed to the interrupt line **CAN_INT_MO**.

1.6 Parity Check Mechanism (optional)

To ensure data integrity for Message RAM data, the D_CAN provides a parity check mechanism. One parity bit will be calculated for 32 bits of data. Parity information is stored in the Message RAM on write access and will be checked on read access.

Note: The D_CAN uses "Even Parity Coding", that means an even parity bit is set if the number of ones in the 32 bit word is odd (making the number of ones even).

1.6.1 Behavior on parity error

On any read access to Message RAM, e.g. during a start of a CAN frame transmission, the parity of the Message Object will be checked. If a parity error is detected, an interrupt is generated and **MsgVal** bit of the Message Object will be reset, to avoid transmission of invalid data over CAN bus. Additionally the port **CAN_UERR** signalizes - by generating a high pulse for one **HOST_CLK** clock period - the parity error occurrence for the CPU.

Message Object data can be read by the host CPU, independent of parity error. Thus, the software has to take care, that read data is valid, e.g. by immediately checking the **PEC** register on parity error interrupt.

During RAM initialization also the parity bits are generated. To avoid parity errors after power-on, the RAM has to be initialized using the RAM Initialization function.

1.7 Registers

The D_CAN module allocates an address space of 512 Bytes for the D_CAN registers. Data is accessible by host interface using a data width of 8 bit (byte access), 16 bit (half-word access) and 32 bit (word access).

The two sets of interface registers (IF1 and IF2) providing an indirect read and write access for the host CPU to the Message RAM. They buffer the data to be transferred to and from the RAM, avoiding conflicts between CPU accesses and CAN frame reception/ transmission.

Address	Symbol	Name	Page	Reset	Acc
CAN Status and Configuration Registers					
0x000	CCTRL	CAN Control Register	17	0000 0001	r/w
0x004	CSTS	CAN Status Register	20	0000 0007	r
0x008	CERC	CAN Error Counter Register	22	0000 0000	r
0x00C	CBT	CAN Bit Timing Register	22	0000 2301	r/w
0x010	CIR	CAN Interrupt Register	44	0000 0000	r
0x014	CTR	CAN Test Register	24	0000 0080[1]	r/w
0x018	CFR	CAN Function Register	25	0000 0000	r/w
0x01C	PEC	CAN Parity Error Counter Register	26	0000 UUUU[2]	r
0x020	CRR	CAN Core Release Register	28	111S SSS[3]	r
0x024	HWS	CAN Hardware Configuration Status	29	0000 000S[4]	r
0x028 - 0x080		*reserved for future use*		0000 0000	r
CAN Message Object Status Registers					
0x084	MOTRX	MO Transmission Request X Register	39	0000 0000	r
0x088	MOTRA	MO Transmission Request A Register	38	0000 0000	r
0x08C	MOTRB	MO Transmission Request B Register	38	0000 0000	r
0x090	MOTRC	MO Transmission Request C Register	38	0000 0000	r
0x094	MOTRD	MO Transmission Request D Register	38	0000 0000	r
0x098	MONDX	MO New Data X Register	40	0000 0000	r
0x09C	MONDA	MO New Data A Register	39	0000 0000	r
0x0A0	MONDB	MO New Data B Register	39	0000 0000	r
0x0A4	MONDC	MO New Data C Register	39	0000 0000	r
0x0A8	MONDD	MO New Data D Register	39	0000 0000	r
0x0AC	MOIPX	MO Interrupt Pending X Register	42	0000 0000	r
0x0B0	MOIPA	MO Interrupt Pending A Register	41	0000 0000	r

Table 2 Register Overview

Address	Symbol	Name	Page	Reset	Acc
0x0B4	MOIPB	MO Interrupt Pending B Register	41	0000 0000	r
0x0B8	MOIPC	MO Interrupt Pending C Register	41	0000 0000	r
0x0BC	MOIPD	MO Interrupt Pending D Register	41	0000 0000	r
0x0C0	MOVALX	MO Message Valid X Register	43	0000 0000	r
0x0C4	MOVALA	MO Message Valid A Register	42	0000 0000	r
0x0C8	MOVALB	MO Message Valid B Register	42	0000 0000	r
0x0CC	MOVALC	MO Message Valid C Register	42	0000 0000	r
0x0D0	MOVALD	MO Message Valid D Register	42	0000 0000	r
0x0D4-0x0FC		*reserved for future use*		0000 0000	r
CAN application Interface Registers					
0x100	IF1CMR	IF1 Command Register	31	0000 0001	r/w
0x104	IF1MSK	IF1 Mask Register	35	FFFF FFFF	r/w
0x108	IF1ARB	IF1 Arbitration Register	35	0000 0000	r/w
0x10C	IF1MCTR	IF1 Message Control Register	36	0000 0000	r/w
0x110	IF1DA	IF1 Data A Register	37	0000 0000	r/w
0x114	IF1DB	IF1 Data B Register	37	0000 0000	r/w
0x118 - 0x11C		*reserved for future use*		0000 0000	r
0x120	IF2CMR	IF2 Command Register	31	0000 0001	r/w
0x124	IF2MSK	IF2 Mask Register	35	FFFF FFFF	r/w
0x128	IF2ARB	IF2 Arbitration Register	35	0000 0000	r/w
0x12C	IF2MCTR	IF2 Message Control Register	36	0000 0000	r/w
0x130	IF2DA	IF2 Data A Register	37	0000 0000	r/w
0x134	IF2DB	IF2 Data B Register	37	0000 0000	r/w
0x138 - 0x1FC		*reserved for future use*		0000 0000	r

Table 2 Register Overview

1.) CTR.Rx (Bit7) reset value depends on Rx-Pad.
2.) PEC is not reset by the module-reset.
3.) CRR reset value depends on generic parameter set on D_CAN synthesis.
4.) HWS reset value depends on generic parameter set on D_CAN synthesis.

1.8 Message Object

There are up to 128 Message Objects in the Message RAM. To avoid conflicts between CPU access to the Message RAM and CAN message reception and transmission, the CPU cannot directly access the Message Objects, these accesses are handled via the IFx Interface Registers.

Table 3 gives an overview of the structure of a Message Object. **MsgVal**, **NewDat**, **IntPnd** and **TxRqst** are registers and directly readable like in Chapter 2.4 described.

MsgVal				*NewDat*	*MsgLst*	*IntPnd*		*TxIE*	*RxIE*	*RmtEn*	*TxRqst*	*EoB*
UMask	*Msk28-0*	*MXtd*	*MDir*									
	ID28-0	*Xtd*	*Dir*	*DLC3-0*	*Data 0*	*Data 1*	*Data 2*	*Data 3*	*Data 4*	*Data 5*	*Data 6*	*Data 7*

Table 3 Structure of a Message Object

1.8.1 Message Object Control Flags

MsgVal: Message is valid

0= The Message Object is ignored by the Message Handler.

1= The Message Object is configured and should be considered by the Message Handler.

Note: The CPU must reset the MsgVal bit of all unused Messages Objects during the initialization before it resets bit Init in the CAN Control Register. MsgVal must also be reset if the Messages Object is no longer used in operation. For reconfiguration of Message Objects during normal operation see Chapter 1.8.4 and Chapter 1.8.5.

NewDat: New Data

0= No new data has been written into the data portion of this Message Object by the Message Handler since last time this flag was cleared by the CPU.

1= The Message Handler or the CPU has written new data into the data portion of this Message Object.

MsgLst: Message Lost (only valid for Message Objects with direction = *receive*)

0= No message lost since last time this bit was reset by the CPU.

1= The Message Handler stored a new message into this object when **NewDat** was still set, the CPU has lost a message.

IntPnd: Interrupt Pending

0= This message object is not the source of an interrupt.

1= This message object is the source of an interrupt. The Interrupt Identifier in the Interrupt Register will point to this message object if there is no other interrupt source with higher priority.

TxIE: Transmit Interrupt Enable

0= **IntPnd** will be left unchanged after the successful transmission of a frame.

1= **IntPnd** will be set after a successful transmission of a frame.

RxIE: Receive Interrupt Enable

0= **IntPnd** will be left unchanged after a successful reception of a frame.

1= **IntPnd** will be set after a successful reception of a frame.

RmtEn: Remote Enable

0= At the reception of a Remote Frame, **TxRqst** is left unchanged.

1= At the reception of a Remote Frame, **TxRqst** is set.

TxRqst: Transmit Request

0= This Message Object is not waiting for transmission.

1= The transmission of this Message Object is requested and is not yet done.

EoB: End of Block

0= Message Object belongs to a FIFO Buffer Block and is not the last Message Object of that FIFO Buffer Block.

1= Single Message Object or last Message Object of a FIFO Buffer Block.

Note: ***This bit is used to concatenate two or more Message Objects (up to 128) to build a FIFO Buffer. <u>For single Message Objects (not belonging to a FIFO Buffer) this bit must always be set to one</u>. For details on the concatenation of Message Objects see Application Note 002 "Configuration of the Message Memory".***

1.8.2 Message Object Mask Bits

The Message Object Mask Bits together with the arbitration bits are used for acceptance filtering of incoming messages.

UMask: Use Acceptance Mask

0= Mask ignored. Acceptance formula[1]:
$(RTR_{Rx} == \sim\mathbf{DIR})\ \&\&\ (IDE_{Rx} == \mathbf{IDE})\ \&\&\ (ID_{Rx} == \mathbf{ID})$

1= Use Mask (**Msk28-0**, **MXtd**, and **MDir**) for acceptance filtering, formula:
$((RTR_{Rx}\ \&\ \mathbf{MDIR}) == (\sim\mathbf{DIR}\ \&\ \mathbf{MDIR}))\ \&\&$
$((IDE_{Rx}\ \&\ \mathbf{MXtd}) == (\mathbf{IDE}\ \&\ \mathbf{MXtd}))\ \&\&$
$((ID_{Rx}\ \&\ \mathbf{Msk}) == (\mathbf{ID}\ \&\ \mathbf{Msk}))$

Note: ***If the UMask bit is set to one, the Message Object's mask bits have to be programmed during initialization of the Message Object before MsgVal is set to one.***

Msk28-0: Identifier Mask

0= The corresponding bit in the identifier of the message object cannot inhibit the match in the acceptance filtering.

1= The corresponding identifier bit is used for acceptance filtering.

1. Ansi-C syntax

MXtd: Mask Extended Identifier

0= The extended identifier bit (**IDE**) has no effect on the acceptance filtering.

1= The extended identifier bit (**IDE**) is used for acceptance filtering.

Note: When 11-bit ("standard") Identifiers are used for a Message Object, the identifiers of received Data Frames are written into bits ID28 to ID18. For acceptance filtering, only these bits together with mask bits Msk28 to Msk18 are considered.

MDir: Mask Message Direction

0= The message direction bit (**Dir**) has no effect on the acceptance filtering. Handle with care setting **IFxMSK.MDir** to zero!

1= The message direction bit (**Dir**) is used for acceptance filtering.

1.8.3 CAN Message bits

The Arbitration Registers **ID28-0**, **Xtd**, and **Dir** are used to define the identifier and type of outgoing messages and are used (together with the mask registers **Msk28-0**, **MXtd**, and **MDir**) for acceptance filtering of incoming messages. A received message is stored into the valid Message Object with matching identifier and Direction=*receive* (Data Frame) or Direction=*transmit* (Remote Frame). Extended frames can be stored only in Message Objects with **Xtd** = *one*, standard frames in Message Objects with **Xtd** = *zero*. If a received message (Data Frame or Remote Frame) matches with more than one valid Message Object, it is stored into that with the lowest message number. For further details see Application Note 004 "CAN Operation".

ID28-0: Message Identifier

ID28 - ID0 29-bit Identifier ("Extended Frame").

ID28 - ID18 11-bit Identifier ("Standard Frame").

Xtd: Extended Identifier

0= The 11-bit ("standard") Identifier will be used for this Message Object.

1= The 29-bit ("extended") Identifier will be used for this Message Object.

Dir: Message Direction

0= Direction = *receive*: On **TxRqst**, a Remote Frame with the identifier of this Message Object is transmitted. On reception of a Data Frame with matching identifier, that message is stored in this Message Object.

1= Direction = *transmit*: On **TxRqst**, the respective Message Object is transmitted as a Data Frame. On reception of a Remote Frame with matching identifier, the **TxRqst** bit of this Message Object is set (if **RmtEn** = *one*).

DLC3-0: Data Length Code

0-8 Data Frame has 0-8 data bytes.

9-15 Data Frame has 8 data bytes.

Note: The Data Length Code of a Message Object must be defined the same as in all the corresponding objects with the same identifier at other nodes. When the Message Handler stores a data frame, it will write the DLC to the value given by the received message.

Data 0: 1st data byte of a CAN Data Frame

Data 1: 2nd data byte of a CAN Data Frame

Data 2: 3rd data byte of a CAN Data Frame

Data 3: 4th data byte of a CAN Data Frame

Data 4: 5th data byte of a CAN Data Frame

Data 5: 6th data byte of a CAN Data Frame

Data 6: 7th data byte of a CAN Data Frame

Data 7: 8th data byte of a CAN Data Frame

Note: Byte Data 0 is the first data byte shifted into the shift register of the CAN Core during a reception, byte Data 7 is the last. When the Message Handler stores a Data Frame, it will write all the eight data bytes into a Message Object. If the Data Length Code is less than 8, the remaining bytes of the Message Object will be overwritten by <u>non specified values</u>.

1.8.4 Reconfiguration of Message Objects for the reception of frames

A Message Object with **Dir** = ‘0’ is configured for the reception of data frames, with **Dir** = ‘1’ AND **Umask** = ‘1’ AND **RmtEn** = ‘0’ it is configured for the reception of remote frames.

It is neccessary to reset **MsgVal** to not valid before changing any of the following configuration and control bits:

Id28-0, **Xtd**, **Dir**, **DLC3-0**, **RxIE**, **TxIE**, **RmtEn**, **EoB**, **Umask**, **Msk28-0**, **MXtd**, and **MDir**

These parts of a Message Object may be changed without clearing **MsgVal**:

Data7-0, **TxRqst**, **NewDat**, **MsgLst**, and **IntPnd**

1.8.5 Reconfiguration of Message Objects for the transmission of data frames

A Message Object with **Dir** = ‘1’ AND (**Umask** = ‘0’ OR **RmtEn** = ‘1’) is configured for the transmission of data frames.

It is neccessary to reset **MsgVal** to not valid before changing any of the following configuration and control bits:

Dir, **RxIE**, **TxIE**, **RmtEn**, **EoB**, **Umask**, **Msk28-0**, **MXtd**, and **MDir**

These parts of a message object may be changed without clearing MsgVal:

Id28-0, **Xtd**, **DLC3-0**, **Data7-0**, **TxRqst**, **NewDat**, **MsgLst**, and **IntPnd**

1.8.6 Message Object Bits in the Memory

The following table shows the bit ordering of the Message Object in the Memory. The Message Object Flags **MsgVal, NewDat, IntPnd** and **TxRqst** are implemented as registers and are not stored in the Message Object Memory.

Memory-Bits	Name	Memory-Bits	Name
Control and Mask Flags		CAN Message Bits	
135	MsgLst	98	Xtd
134	UMask	97	Dir
133	TXIE	96:68	ID(28:0)
132	RxIE	67:64	DLC3-0
131	RmtEn	63:56	Data 0
130	EoB	55:48	Data 1
129	MXtd	47:40	Data 2
128	MDir	39:32	Data 3
127:99	Msk(28:0)	31:24	Data 4
		23:16	Data 5
		15:8	Data 6
		7:0	Data 7

Table 4 Memory Ordering of Message Object Bits

Chapter 2.

2. Register Description

2.1 Hardware Reset Description

After hardware reset, the registers of the D_CAN hold the default values described in brackets in Table 32, Table 33 and Table 34.

Additionally the *Bus_Off* state is reset and the output **CAN_TXD** is set to *recessive* (HIGH). The value 0x0001 (**CCTRL.Init** = ‘1’) in the CAN Control Register enables the software initialization. The D_CAN does not influence the CAN bus until the CPU resets **CCTRL.Init** to ‘0’.

The data in the Message RAM is (apart from the **MsgVal**, **NewDat**, **TxRqst** and **IntPnd** bits) not affected by a hardware reset. After power-on, the contents of the Message RAM has be initialized with zeros setting port **CAN_RAMINIT_REQ** to *one* or write of **CFR.RamInit**. The setup of Message Objects is described in chapter 3.2.

2.2 CAN Protocol Related Registers

These registers are related to the CAN protocol controller in the CAN Core. They control the operating modes and the configuration of the CAN bit timing and provide status information.

2.2.1 *CAN Control Register (CCTRL)*

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x000	res												DE2	DE1	MIL	res
	R-0												RW-0	RW-0	RW-0	R-0
Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	res								Test	CCE	DAR	res	EIE	SIE	ILE	Init
	R-0								RW-0	RP-0	RW-0	R-0	RW-0	RW-0	RW-0	RW-1

R = Read, W = Write, P = Protected Write, U = Undefined; -*n* = Value after reset

Table 5 CAN control register (address 0x00)

Bit 19 **DE2:** DMA enable for IF2

0= Disabled - Module DMA output port **CAN_IF2DMA** is always LOW.

1= Enabled - Requesting a message object transfer from IF2 to Message RAM or vice versa with **IF2CMR.DMAactive** enabled the end of the transfer will be marked with setting port **CAN_IF2DMA** to *one*. The port remains *one* until first access to one of the IF2 registers.

Bit 18 **DE1:** DMA enable for IF1

0= Disabled - Module DMA output port **CAN_IF1DMA** is always LOW.

1= Enabled - Requesting a message object transfer from IF1 to Message RAM or vice versa with **IF1CMR.DMAactive** enabled the end of the transfer will be marked with setting port **CAN_IF1DMA** to *one*, the port remains *one* until first access to one of the IF1 registers.

Bit 17 **MIL:** Message object Interrupt Line enable

0= Disabled - Message Object Interrupt **CAN_INT_MO** is always LOW. If **CCTRL.ILE** is enabled all message object interrupts are routed to line **CAN_INT_STATUS** otherwise no message object interrupt will be visible.

1= Enabled - message object interrupts will set **CAN_INT_MO** to *one*, signal remains *one* until all pending interrupts are processed.

Bit 7 **Test:** Test Mode Enable

0= Normal Operation.

1= Test Mode. Enables the write access to Test Register **CTR**.

Bit 6 **CCE:** Configuration Change Enable

0= The CPU has no write access to the configuration registers.

1= The CPU has write access to the Bit Timing Register **CBT** (while **CCTRL.Init** = *one*).

Bit 5 **DAR:** Disable Automatic Retransmission

0= Automatic Retransmission of not successful messages enabled.

1= Automatic Retransmission disabled.

Bit 3 **EIE:** Error Interrupt Enable

0= Disabled - **CSTS.PER, CSTS.BOff** and **CSTS.EWarn** flags will still be updated, but without affecting interrupt line **CAN_INT_STATUS** and Interrupt register **CIR**.

1= Enabled - If **CSTS.PER** flag is one, or **CSTS.BOff** or **CSTS.EWarn** are changed, the interrupt line **CAN_INT_STATUS** gets active (if **ILE**=1) and **CIR.StatusInt** is set.

Bit 2 **SIE:** Status Interrupt Enable

0= Disabled - **CSTS.RxOk**, **CSTS.TxOk** and **CSTS.LEC** will still be updated, but without affecting interrupt line **CAN_INT_STATUS** and Interrupt register **CIR**.

1= Enabled - When a message transfer is successfully completed or a CAN bus error is detected, indicated by flags **CSTS.RxOk**, **CSTS.TxOk** and **CSTS.LEC**, the interrupt line **CAN_INT_STATUS** gets active (if **ILE**=1) and **CIR.StatusInt** is set.

Bit 1 **ILE:** Module Interrupt Line Enable

0= Disabled - Module Interrupt Line **CAN_INT_STATUS** is always LOW.

1= Enabled - error and status interrupts (if **CCTRL.EIE**=1 and **CCTRL.SIE**=1) will set line **CAN_INT_STATUS** to *one*, signal remains *one* until all pending interrupts are processed. If **MIL** is disabled, the message object interrupts will also affect this interrupt line.

Bit 0 **Init:** Initialization

0= Normal Operation.

1= Initialization is started.

Note: ***Due to the synchronization mechanism between the two clock domains, there may be a delay until the value written to*** **CCTRL.Init** ***can be read back. Therefore the programmer has to assure that the previous value written to*** **CCTRL.Init** ***has been accepted by reading*** **CCTRL.Init** ***before setting*** **CCTRL.Init** ***to a new value.***

Note: ***The Bus_Off recovery sequence (see CAN Specification Rev. 2.0) cannot be shortened by setting or resetting*** **CCTRL.Init**. ***If the device goes Bus_Off, it will set*** **CCTRL.Init** ***of its own accord, stopping all bus activities. Once*** **CCTRL.Init** ***has been cleared by the CPU, the device will then wait for 129 occurrences of Bus Idle (129 * 11 consecutive recessive bits) before resuming normal operations. At the end of the Bus_Off recovery sequence, the Error Management Counters will be reset.***

During the waiting time after the resetting of **CCTRL.Init**, ***each time a sequence of 11 recessive bits has been monitored, a Bit0Error code is written to the Status Register, enabling the CPU to readily check up whether the CAN bus is stuck at dominant or continuously disturbed and to monitor the proceeding of the Bus_Off recovery sequence.***

2.2.2 Status Register (CSTS)

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x004	res															
	R-0															

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	res							PER	BOff	EWarn	EPASS	RxOK	TxOK	LEC		
	R-0							RC-0	R-0	R-0	R-0	RC-0	RC-0	RS-111		

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

Table 6 CAN status register (address 0x04)

Bit 8 **PER:** Parity Error detected (optional)

0= No parity error detected since last read access.

1= The Parity Check Mechanism has detected a parity error in the Message RAM, this bit will be reset if Status Register is read.

Bit 7 **BOff:** Bus_Off Status

0= The CAN module is not Bus_Off.

1= The CAN module is in Bus_Off state.

Bit 6 **EWarn:** Warning Status

0= Both error counters are below the error warning limit of 96.

1= At least one of the error counters in the EML has reached the error warning limit of 96.

Bit 5 **EPass:** Error Passive

0= The CAN Core is in the *error active* state. It normally takes part in bus communication and sends an *active error flag* when an error has been detected.

1= The CAN Core is in the *error passive* state as defined in the CAN Specification.

Bit 4 **RxOk:** Received a Message Successfully

0= Since this bit was last read by the CPU, no message has been successfully received. This bit is never reset by D_CAN internal events.

1= Since this bit was last reset by a read access of the CPU, a message has been successfully received (independently of the result of acceptance filtering). This bit will be reset by reading the Status Register.

Bit 3 **TxOk:** Transmitted a Message Successfully

0= Since this bit was read by the CPU, no message has been successfully transmitted. This bit is never reset by D_CAN internal events.

1= Since this bit was last reset by a read access of the CPU, a message has been successfully (error free and acknowledged by at least one other node) transmitted. This bit will be reset by reading the Status Register.

Bits 2-0 **LEC:** Last Error Code (Type of the last error to occur on the CAN bus)

0= **No Error :** Set together with **CSTS.RxOK** or **CSTS.TxOK**.

1= **Stuff Error :** More than 5 equal bits in a sequence have occurred in a part of a received message where this is not allowed.

2= **Form Error :** A fixed format part of a received frame has the wrong format.

3= **AckError :** The message this D_CAN Core transmitted was not acknowledged by another node.

4= **Bit1Error :** During the transmission of a message (with the exception of the arbitration field), the device wanted to send a *recessive* level (bit of logical value ‘1’), but the monitored bus value was *dominant*.

5= **Bit0Error :** During the transmission of a message (or acknowledge bit, or active error flag, or overload flag), the device wanted to send a *dominant* level (data or identifier bit logical value ‘0’), but the monitored bus value was *recessive*. During *Bus_Off* recovery this status is set each time a sequence of 11 *recessive* bits has been monitored. This enables the CPU to monitor the proceeding of the Bus_Off recovery sequence (indicating the bus is not stuck at *dominant* or continuously disturbed).

6= **CRCError :** The CRC check sum was incorrect in the message received, the CRC received for an incoming message does not match with the calculated CRC for the received data.

7= **NoChange :** Any read access to the Status Register re initializes the **LEC** to ‘7’. When the **LEC** shows the value ‘7’, no CAN bus event was detected since the last CPU read access to the Status Register.

The **LEC** field holds a code which indicates the type of the last error to occur on the CAN bus. This field will be cleared to ‘0’ when a message has been transferred (reception or transmission) without error.

2.2.2.1 Status Interrupts

The Interrupt sources **PER**, **BOff** and **EWarn** are grouped as Error Interrupt and will be enabled by bit **CCTRL.EIE**. **RxOk**, **TxOk**, and **LEC** belonging to the Status Interrupt group and could be enabled by **CCTRL.SIE** bit. It is assumed that the corresponding enable bits in the CAN Control Register are set. See also Chapter 2.2.1.

A change of bit **EPass** will never generate a Status Interrupt.

Enabling **CCTRL.SIE**, a Status Interrupt will be generated at each CAN frame, independent of the Message RAM configuration (**IntPnd** Interrupts).

Reading the Status Register will clear the **PER, RxOk**, **TxOk** bits and sets the **LEC to ‘7’**.

2.2.3 Error Counter (CERC)

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x008	res															
	R-0															

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	RP	REC6-0							TEC7-0							
	R-0	R-0							R-0							

R = Read, W = Write, U = Undefined; -*n* = Value after reset

Table 7 Error counter register (address 0x08)

Bit 15 RP: Receive Error Passive

0= The Receive Error Counter is below the *error passive* level.

1= The Receive Error Counter has reached the *error passive* level as defined in the CAN Specification.

Bits 14-8 REC6-0: Receive Error Counter

Actual state of the Receive Error Counter. Values between 0 and 127.

Bits 7-0 TEC7-0: Transmit Error Counter

Actual state of the Transmit Error Counter. Values between 0 and 255.

2.2.4 Bit Timing/ BRP extension Register (CBT)

Note: This register is only writable if bits* CCTRL.CCE *and* CCTRL.Init *are set.

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x00C	res												BRPE			
	R-0												RP-0			

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	res	TSeg2			TSeg1				SJW		BRP					
	R-0	RP-2			RP-0x3				RP-0		RP-0x1					

R = Read, W = Write, P = Protected Write, U = Undefined; -*n* = Value after reset

Table 8 Bit Timing/ BRP extension register (address 0x0C)

Bits 19:16 **BRPE:** Baud Rate Prescaler Extension

0x00-0x0F By programming **BRPE** the Baud Rate Prescaler can be extended to values up to 1023. The actual interpretation by the hardware is that one more than the value programmed by **BRPE** (MSBs) and **BRP** (LSBs) is used.

Bits 14:12 **TSeg2:** The time segment after the sample point

0x0-0x7 valid values for TSeg2 are [0 … 7].

Bits 11:8 **TSeg1:** The time segment before the sample point

0x01-0x0F valid values for TSeg1 are [1 … 15].

Bits 7:6 **SJW:** (Re) Synchronization Jump Width

0x0-0x3 valid programmed values are 0-3. The actual interpretation by the hardware of this value is such that one more than the value programmed here is used.

Bits 5:0 **BRP:** Baud Rate Prescaler

0x00-0x3F The value by which the oscillator frequency is divided for generating the bit time quanta. The bit time is built up from a multiple of this quanta. Valid values for the Baud Rate Prescaler are [0 … 63]. The actual interpretation by the hardware of this value is such that one more than the value programmed here is used.

This register is only writable if bits **CCTRL.CCE** and **CCTRL.Init** are set. The CAN bit time may be programed in the range of [4 … 25] time quanta. The CAN time quantum may be programmed in the range of [1 … 1024] **CAN_CLK** periods. For details see Application Note 001 "Configuration of Bit Timing". The actual interpretation by the hardware of this value is such that one more than the value programmed here is used. **TSeg1** is the sum of Prop_Seg and Phase_Seg1. **TSeg2** is Phase_Seg2. Therefore the length of the bit time is (programmed values) [**TSeg1** + **TSeg2** + 3] t_q or (functional values) [Sync_Seg + Prop_Seg + Phase_Seg1 + Phase_Seg2] t_q.

2.2.5 Test Register (CTR)

The Test Mode is entered by setting bit **CCTRL.Test** to *one*. In Test Mode the bits **EXL, Tx1**, **Tx0**, **LBack** and **Silent** in the Test Register are writable. Bit **Rx** monitors the state of pin **CAN_RXD** and therefore is only readable. All Test Register functions are disabled when bit Test is reset to zero.

Loop Back Mode and **CAN_TXD** Control Mode are hardware test modes, not to be used by application programs.

Note: This register is only writable if bit* CCTRL.Test *is set.

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x014	res															
	R-0															
Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	res								Rx	Tx1-0		LBack	Silent	res		
	R-0								R-U	RP-0		RP-0	RP-0	R-0		

R = Read, W = Write, P = Protected Write, U = Undefined; -*n* = Value after reset

Table 9 Test register (address 0x14)

Bit 7 **Rx:** Receive Pin

Monitors the actual value of the **CAN_RXD** pin

0= The CAN bus is dominant (**CAN_RXD** = ‘0’).

1= The CAN bus is recessive (**CAN_RXD** = ‘1’).

Bits 6:5 **Tx1-0:** Control of **CAN_TXD** pin

00 Reset value, **CAN_TXD** is controlled by the CAN_Core.

01 Sample Point can be monitored at **CAN_TXD** pin.

10 **CAN_TXD** pin drives a dominant (‘0’) value.

11 **CAN_TXD** pin drives a recessive (‘1’) value.

Bit 4 **LBack:** Loop Back Mode

0= Loop Back Mode is disabled.

1= Loop Back Mode is enabled, see Chapter 1.3.5.2.

Bit 3 **Silent:** Silent Mode

0= Normal operation.

1= The module is in Silent Mode, see Chapter 1.3.5.1.

Note: Write access to the Test Register has to be enabled by setting bit Test in the CAN Control Register.
Setting Tx1-0 ≠ “00” disturbs message transfer.

2.2.6 Function Register (CFR)

The Function Register controls the features RAM_Initialisation and Power_Down also by application register.

The D_CAN module can be prepared for Power_Down by setting the port **CAN_CLKSTOP_REQ** to *one* or writing to **CFR.ClkStReq** a *one.* The power down state is left by setting port **CAN_CLKSTOP_REQ** to *zero* or writing to **CFR.ClkStReq** a *zero*, acknowledged by **CAN_CLKSTOP_ACK** is going to *zero* as well as **CFR.ClkStAck**. The **CCTRL.Init** bit is left *one* and has to be written by the application to re-enable CAN-transfers.

***Note: It's recommended to use either the ports* CAN_CLKSTOP_REQ *and* CAN_CLKSTOP_ACK *or the* CCTRL.ClkStReq *and* CFR.ClkStAck. *The application* CFR.ClkStReq *shows also the actual status of the port* CAN_CLKSTOP_REQ.**

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x018	res															
	R-0															

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	res												RAM-Init	res	ClkSt Req	ClkSt Ack
	R-0												RP-0	R-0	RW-0	R-0

R = Read, W = Write, P = Protected Write, U = Undefined; -*n* = Value after reset

Table 10 CAN Function register (address 0x18)

Bit 3 **RAMInit:** Request for automatic RAM Initialization

0= No automatic RAM Initialization is requested, if once a ram initialization is started a write of a zero will be ignored. The Bit is cleared by hardware, after RAM Initialization is completed.

1= Start automatic RAM Initialization. All message objects will be written with zeros and the parity bits will be set. The **RAMInit** Bit will return to zero after the RAM-Initialization process is completed. A RAM Initialization Request is only possible if **CCTRL.Init** is set. The duration of the automatic RAM Initialization is message-buffer-size + 4 host_clock cycles.

Bit 1 **ClkStReq:** ClockStop Request

0= No Clock Stop is requested or a former clock stop request is taken back.

1= Clock Stop request is set, the D_CAN finishes all pending transfer requests and set afterwards the init flag. Afterwards it acknowledges the request by setting **ClkStopAck**. D_CAN can set in power down.

Bit 0 **ClkStAck:** Clock Stop Acknowledge

0= No ClockStop Request is set or still handle some pending transfers.

1= Clock Stop Request has be accepted by D_CAN and the module can set in power down.

Note: Alternatively, the features RAM_Initialisation and Power_Down can be controlled by the optional ports CAN_CLKSTOP_REQ, CAN_CLKSTOP_ACK and CAN_RAMINIT_REQ, CAN_RAMINIT_ACK from outside the D_CAN module.

2.2.7 Parity Error Code (PEC) (optional)

If a parity error is detected, the **CSTS.PER** flag will be set. This bit is not reset by the Parity Check Mechanism - it must be reset by reading the **CSTS** register.

In addition to the **CSTS.PER** flag the memory area where the parity error has been detected will be displayed in the **PEC** register. The register is not reset by the module-reset. Therefore, the content of this register is undefined until the first parity error occurs. After parity error it will contain the last error code as long as power is provided to the module.

If more than one word with a parity error was detected, the highest word number with a parity error will be displayed.

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x01C	res															
	R-0															
Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	res			ParErrWord					Message Number							
	R-0			R-U					R-U							

R = Read, W = Write, U = Undefined; *-n* = Value after reset

Table 11 Parity error code register (addresses 0x1C)

Bits 12:8 **Parity Error in Word :** marks the affected Word

The Message RAM word is split in 5 Words of 32 Bits. Over each 32 Bits Word in the Message RAM a parity calculation is done. The **ParErrWord** marks in which word the latest parity error has occurred.

Memory-Bits	Name	Memory-Bits	Name
ParErrWord[12]		ParErrWord[10]	
159:136	filled with zeros	95:68	ID(27:0)
135	MsgLst	67:64	DLC3-0
134	UMask	ParErrWord[9]	
133	TXIE	63:56	Data 0
132	RxIE	55:48	Data 1
131	RmtEn	47:40	Data 2
130	EoB	39:32	Data 3
129	MXtd		
128	MDir		
ParErrWord[11]		ParErrWord[8]	
127:99	Msk(28:0)	31:24	Data 4
98	Xtd	23:16	Data 5
97	Dir	15:8	Data 6
96	ID(28)	7:0	Data 7

Table 12 Parity Calculation Words

Bits 7:0 Message Number:

0x01-0x80 = Number of latest affected Message Object.

2.2.8 Core Release Register (CRR)

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x020	REL				STEP				SUBSTEP				YEAR			
	R-1				R-1				R-1				R-d			

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	MON								DAY							
	R-d								R-d							

R = Read, W = Write, U = Undefined; -*n* = Value after reset; -d = Value defined at synthesis by generic parameter

Table 13 Core Release Register (addresses 0x20)

Bits 31:28 Core Release:
One digit, BCD-coded.

Bits 27:20 Step of Core Release:
Two digits, BCD-coded.

Bits 19:16 Design Time Stamp, Year:
One digit, BCD-coded. This field is set by generic parameter on D_CAN synthesis.

Bits 15:8 Design Time Stamp, Month:
Two digits, BCD-coded. This field is set by generic parameter on D_CAN synthesis.

Bits 7:0 Design Time Stamp, Day:
Two digits, BCD-coded. This field is set by generic parameter on D_CAN synthesis.

Release	Step	SubStep	Name
0	1	1	Pre-Alpha, first version after adaptation from C_CAN
0	2	0	Beta, FPGA evaluation
0	2	1	Beta, Patch 1
1	0	0	Conformance tested fpga version
1	1	0	Conformance tested asic version (30.11.2007)
1	1	1	Bugfix-01: “Invalid messages stored into Message RAM” (12.05.2010)

Table 14 Coding of Release Versions

2.2.9 Hardware Configuration Status Register (HWS)

Bits	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
0x024	res															
	R-0															

Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	res													paren	mb_w	
	R-0													R-d	R-d	

R = Read, W = Write, U = Undefined; -*n* = Value after reset; -d = Value defined at synthesis by generic parameter

Table 15 HW Configuration Status Register (addresses 0x24)

Bit 2 **paren:** generic parameter is set during synthesis

1= the parity generation is enabled,
0= the parity generation is disabled

Bits 1:0 **mb_w** message buffer count set by synthesis

0= 16 message objects, *1*= 32 message objects,
2= 64 message objects, *3*= 128 message objects.

2.3 Message Interface Register Sets 1 and 2

Address	IF1 Register Set 31 16	15 0	Address	IF2 Register Set 31 16	15 0
0x100	IF1 Command Mask	IF1 Command Request	0x120	IF2 Command Mask	IF2 Command Request
0x104	IF1 Mask 2	IF1 Mask 1	0x124	IF2 Mask 2	IF2 Mask 1
0x108	IF1 Arbitration 2	IF1 Arbitration 1	0x128	IF2 Arbitration 2	IF2 Arbitration 1
0x10C	res.	IF1 Message Control	0x12C	res.	IF2 Message Control
0x110	IF1 Data A 2	IF1 Data A 1	0x130	IF2 Data A 2	IF2 Data A 1
0x114	IF1 Data B 2	IF1 Data B 1	0x134	IF2 Data B 2	IF2 Data B 1

Table 16 IF1 and IF2 Message Interface Register Sets

There are two sets of Interface Registers that control the CPU read and write accesses to the Message RAM. A complete Message Object (see Chapter 1.8) or parts of the Message Object may be transferred between the Message RAM and the IFx Message Buffer Registers (see Chapter 2.3.2) in one single transfer. This transfer, performed in parallel on all selected parts of the Message Object, guarantees the data consistency of the CAN message. Table 16 shows the structure of the two Interface Register sets.

The function of the two Interface Register sets is identical. The second interface register set is provided to serve application programming. Two groups of software drivers may defined, each group is restricted to the use of one of the Interface Register sets. The software drivers of one group may interrupt software drivers of the other group, but not of the same group.

In a simple example, there is one Read_Message task that uses IF1 to get received messages from the Message RAM and there is one Write_Message task that uses IF2 to write messages to be transmitted into the Message RAM. Both tasks may interrupt each other.

Each set of Interface Registers consists controlled by their own Command Registers. The Command Mask Register specifies the direction of the data transfer and which parts of a Message Object will be transferred. The Command Request Register is used to select a Message Object in the Message RAM as target or source for the transfer and to start the action specified in the Command Mask Register.

2.3.1 IF1/2 Command Registers (IF1CMR, IF2CMR)

The control bits of the IF1/2 “Command” Register specify the transfer direction and select which portions of the Message Object should be transferred.

A message transfer is started as soon as the CPU has written the message number to low byte of the Command Request Register and **IFxCMR.AutoInc** is zero. With this write operation, the **IFxCMR.Busy** bit is automatically set to ‘1’ to notify the CPU that a transfer is in progress. After a wait time of 2 to 8 **HOST_CLK** periods, the transfer between the

Interface Register and the Message RAM has been completed and the **IFxCMR.Busy** bit is cleared to ‘0’. The upper limit of the wait time occurs when the message transfer coincides with a CAN message transmission, acceptance filtering, or message storage.

If the CPU writes to both Command Registers consecutively (requests a second transfer while another transfer is already in progress), the second transfer starts when the first one is completed.

Note: While Busy bit of IF1/2 Command Register is one, IF1/2 Register Set is write protected.

Bits	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
0x100 0x120	res		ClrAutoInc	res					WR1RD0	Mask	Arb	Control	ClrIntPnd	TxRqst/NewDat	Data A	Data B
	R-0		P-0	R-0					RP-0	RP-0	RP-0	RP-0	RP-0	RP-0	RP-0	RP-0
Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	Busy	DMA active	Auto-Inc	res					MONum							
	R-0	RP-0	RP-0	R-0					RP-0x1							

R = Read, W = Write, P = Protected Write, U = Undefined; *-n* = Value after reset

Table 17 IFx Command Register (address IF1:0x100, IF2:0x120)

Bit 29 **ClrAutoInc:** Clear the AutoInc bit without starting a transfer

0= **NoClear**: Has no effect to the other Bits of this Register.

1= **Clear**: Clear the AutoInc bit without starting a transfer, all other bits will be ignored.

Bit 23 **WR1RD0:** Write / Read Transfer

0= **Read**: Transfer data from the Message Object addressed by **IFxCMR.MONum** into the selected IFx Message Buffer Registers.

1= **Write**: Transfer data from the selected IFx Message Buffer Registers to the Message Object addressed by **IFxCMR.MONum**.

The other bits of IFx Command Mask Register have different functions depending on the transfer direction :

2.3.1.1 Direction = Write

Bit 22 **Mask:** Access Mask Bits

0= Mask bits unchanged.

1= transfer **Identifier Mask** + **MDir** + **MXtd** to Message Object.

Bit 21 **Arb:** Access Arbitration Bits

0= Arbitration bits unchanged.

1= transfer **Identifier** + **Dir** + **Xtd** + **MsgVal** to Message Object.

Bit 20 **Control:** Access Control Bits

0= Control Bits unchanged.

1= transfer Control Bits to Message Object.

Note:** If* **IFxCMR.TxRqst/NewDat** ***bit is set, bits **IFxMCTR.TxRqst** ***and*** **IFxMCTR.NewDat** ***will be ignored.***

Bit 19 **ClrIntPnd:** Clear Interrupt Pending Bit

Has no influence to Message Object at write transfer.

Note: When writing to a Message Object, this bit is ignored and copying of IntPnd flag from IFx Control Register to Message RAM could only be controlled by **IFxMTR.IntPnd** ***bit.***

Bit 18 **TxRqst/NewDat:** Access Transmission Request Bit and NewDat Bit

0= **TxRqst** and **NewDat** bit will be handled according **IFxMCTR.NewDat** bit and **IFxMCTR.TxRqst** bit.

1= set **TxRqst** and **NewDat** in Message Object to *one*

Note: If a CAN transmission is requested by setting **IFxCMR.TxRqst/NewDat*****, the TxRqst and NewDat bits in the Message Object will be set to one independently of the values in*** **IFxMCTR*****.***

Bit 17 **Data A:** Access Data Bytes 0-3

0= Data Bytes 0-3 unchanged.

1= transfer Data Bytes 0-3 to Message Object.

Bit 16 **Data B:** Access Data Bytes 4-7

0= Data Bytes 4-7 unchanged.

1= transfer Data Bytes 4-7 to Message Object.

2.3.1.2 Direction = Read

Bit 22 **Mask:** Access Mask Bits

0= Mask bits unchanged.

1= transfer **Identifier Mask** + **MDir** + **MXtd** to **IFxMSK** Register.

Bit 21 **Arb:** Access Arbitration Bits

0= Arbitration bits unchanged.

1= transfer **Identifier** + **Dir** + **Xtd** + **MsgVal** to **IFxARB** Register.

Bit 20 **Control:** Access Control Bits

0= Control Bits unchanged.

1= transfer Control Bits to **IFxMCTR** Register.

Bit 19 **ClrIntPnd:** Clear Interrupt Pending Bit

0= **IntPnd** bit remains unchanged.

1= clear **IntPnd** bit in the Message Object.

Bit 18 **TxRqst/NewDat:** Access Transmission Request Bit

0= **NewDat** bit remains unchanged.

1= clear **NewDat** bit in the Message Object.

Note: ***A read access to a Message Object can be combined with the reset of the control bits*** **IntPnd** ***and*** **NewDat*****. The values of these bits transferred to the*** **IFxMCTR** ***always reflect the status before resetting them.***

Bit 17 **Data A:** Access Data Bytes 0-3

0= Data Bytes 0-3 unchanged.

1= transfer Data Bytes 0-3 to **IFxDA**.

Bit 16 **Data B:** Access Data Bytes 4-7

0= Data Bytes 4-7 unchanged.

1= transfer Data Bytes 4-7 to **IFxDB**.

Note: ***The speed of the message transfer does not depend on how many bytes are transferred.***

2.3.1.3 IF1/2 Command Request Registers for read and write access

Bit 15 **Busy:** Busy Flag

0= reset to zero when read/write action has finished.

1= set to one when writing to the **IFxCMR.MONum**. While bit is one, IFx Register Set is write protected.

Bit 14 **DMAactive:** Activation of DMA feature for subsequent internal IFx Register Set update

0= DMA line leaves passive, independent of IFx activities.

1= By writing to the Command Request Register, an internal transfer of Message Object Data between RAM and IFx will be initiated. When this transfer is complete and **DMAactive** bit was set, the CAN_IFxDMA line gets active. The **DMAactive** bit and port **CAN_IFxDMA** are staying active until first read or write access to one of the IFx registers. If **AutoInc** is set **DMAactive** will be left active, otherwise the bit is reset.

Note: ***Due to auto reset feature of DMAactive bit if AutoInc is inactive, this bit has to be set for each subsequent DMA cycle separately. DMA line has to be enabled in CAN Control Register, see Chapter 2.2.1.***

Bits 13 **AutoInc** Automatic Increment of Message Object Number

0= No AutoIncrement of Message Object Number.

1= AutoIncrement of Message Object Number enabled.

The behavior of the Message Object Number increment depends on the Transfer Direction, **IFxCMR.WR1RD0**.

* **Read:** The first transfer will be initiated (Busy Bit will set) at write of **IFxCMR.MONum**. The Message Object Number will be incremented and the next Message Object will be transferred from Message Object RAM to Interface Registers after a read access of Data-Byte 7.
* **Write:** The first as well as each other transfer will be started after write access to Data-Byte7. The Message Object Number will be incremented after successful transfer from the Interface Registers to the Message Object RAM.

Always after successful transfer the Busy Bit will be reset. In combination with **DMAactive** the port **CAN_IFxDMA** is set, too.

Note: If the direction is configured as Read a write access to Data-Byte 7 will not start any transfer, as well as if the direction is configured as Write a read access to Data-Byte 7 will not start any transfer.
At transfer direction Read each read of Data-Byte 7 will start a transfer until IFxCMR.AutoInc is reset. To aware of resetting a NewDat bit of the following message object, the application has to reset IFxCMR.AutoInc before reading the Data-Byte 7 of the last message object which will be read.

Bits 7:0 Message Number:

0x01-0x80	Valid **Message Number**, the Message Object in the Message RAM is selected for data transfer (up to 128 MsgObj).
0x00	Not a valid **Message Number**, interpreted as *0x80*.
0x81-0xFF	Not a valid **Message Number**, interpreted as *0x01-0x7F*.

Note: When an invalid Message Number is written to **IFxCMR.MONum** ***which is higher than the last Message Object number, a modulo addressing will occur. When e.g. accessing Message Object 33 in a D_CAN module with 32 Message Objects only, the Message Object 1 will be accessed instead.***

2.3.2 IF1/2 Message Buffer Registers

The bits of the Message Buffer registers mirror the Message Objects in the Message RAM. The function of the Message Objects bits is described in Chapter 1.8.

Note: While IFxCMR.Busy bit is one, IF1/2 Register Set is write protected.

2.3.2.1 IF1/2 Mask Register (IF1MSK, IF2MSK)

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x104 0x124	MXtd	MDir	res	Msk28-16												
	RP-1	RP-1	R-1	RP-0x1FFF												
Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	Msk15-0															
	RP-0xFFFF															

R = Read, W = Write, P = Protected Write, U = Undefined; -*n* = Value after reset

Table 18 IFx Mask Register (address IF1:0x104, IF2: 0x124)

Bit 31 **MXtd:** Mask Extended Identifier

Bit 30 **MDir:** Mask Message Direction

Bits 28:0 **Msk28-0:** Identifier Mask

2.3.2.2 IF1/2 Arbitration Register (IF1ARB, IF2ARB)

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x108 0x128	MsgVal	Xtd	Dir	ID28-16												
	RP-0	RP-0	RP-0	RP-0x0000												
Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	ID15-0															
	RP-0x0000															

R = Read, W = Write, P = Protected Write, U = Undefined; -*n* = Value after reset

Table 19 IFx Arbitration Register (address IF1: 0x108, IF2: 0x128)

Bit 31 **MsgVal:** Message Valid

Bit 30 **Xtd:** Extended Identifier

Bit 29 **Dir:** Message Direction

Bits 28:0 **ID28-0:** Message Identifier, Bits 28:18 are the standad identifier

2.3.2.3 IF1/2 Message Control Register (IF1MCTR, IF2MCTR)

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x10C 0x12C	res															
	R-0															

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	New Dat	Msg Lst	Int Pnd	U Mask	TxIE	RxIE	Rmt En	Tx Rqst	EoB	reserved			DLC3-0			
	RP-0	RP-0	RP-0	RP-0	RP-0	RP-0	RP-0	RP-0	RP-0	R-0			RP-0			

R = Read, W = Write, P = Protected Write, U = Undefined; -*n* = Value after reset

Table 20 IFx Message Control Register (address IF1: 0x10C, IF2: 0x12C)

Bit 15 **NewDat:** New Data

Bit 14 **MsgLst:** Message Lost (only valid for Message Objects with direction = *receive*)

Bit 13 **IntPnd:** Interrupt Pending

Bit 12 **UMask:** Use Acceptance Mask

Bit 11 **TxIE:** Transmit Interrupt Enable

Bit 10 **RxIE:** Receive Interrupt Enable

Bit 9 **RmtEn:** Remote Enable

Bit 8 **TxRqst:** Transmit Request

Bit 7 **EoB:** End of Block

Bits 3:0 **DLC3-0:** Data Length Code

2.3.2.4 IF1/2 Data A and Data B Registers (IFxDA, IFxDB)

The data bytes of CAN messages are stored in the IF1/2 registers in the following order. In a CAN Data Frame, Data(0) is the first, Data(7) is the last byte to be transmitted or received. In CAN's serial bit stream, the MSB of each byte will be transmitted first.

Bits	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
0x110 0x130	Data(3)								Data(2)							
	RP-0								RP-0							
Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	Data(1)								Data(0)							
	RP-0								RP-0							

R = Read, W = Write, P = Protected Write, U = Undefined; -*n* = Value after reset

Table 21 IFx Data A register (address IF1: 0x110, IF2: 0x130)

Bits	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
0x114 0x134	Data(7)								Data(6)							
	RP-0								RP-0							
Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	Data(5)								Data(4)							
	RP-0								RP-0							

R = Read, W = Write, P = Protected Write, U = Undefined; -*n* = Value after reset

Table 22 IFx Data B register (address IF1: 0x114, IF2: 0x134)

2.4 Message Handler registers

2.4.1 Transmission Request Registers (MOTR<x>)

These registers hold the **TxRqst** bits of the configurable up to 128 Message Objects. By reading the **TxRqst** bits, the CPU can check for which Message Object a Transmission Request is pending. The **TxRqst** bit in a specific Message Object can be set/reset by the CPU via the IFx Message Interface Registers or set by the Message Handler after reception of a Remote Frame or reset by the Message Handler after a successful transmission.

MOTRA (0x88)
MOTRB (0x8C)
MOTRC (0x90)
MOTRD (0x94)

Bits	*31*	*30*	*29*	*28*	*27*	*26*	*25*	*24*	*23*	*22*	*21*	*20*	*19*	*18*	*17*	*16*
	TxRqst<x>															
	R-0															

Bits	*15*	*14*	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
	TxRqst<x>															
	R-0															

R = Read, W = Write, U = Undefined; -*n* = Value after reset

Table 23 Transmission Request registers (addresses 0x88-0x94)

MOTRA Bits 15:0	**TxRqst16-1:**	Transmission Request Bits (valid for all Message Buffer sizes)
MOTRA Bits 31:16	**TxRqst32-17:**	Transmission Request Bits (valid for Message Buffer size of 32, 64 and 128 Message Objects)
MOTRB Bits 31:0	**TxRqst64-33:**	Transmission Request Bits (valid for Message Buffer size of 64 and 128 Message Objects)
MOTRC Bits 31:0	**TxRqst96-65:**	Transmission Request Bits (valid for Message Buffer size of 128 Message Objects)
MOTRD Bits 31:0	**TxRqst128-97:**	Transmission Request Bits (valid for Message Buffer size of 128 Message Objects)

0= This Message Object is not waiting for transmission.

1= The transmission of this Message Object is requested and is not yet done.

2.4.1.1 Transmission Request X Register (MOTRX)

With register „Transmission Request X Register“ the CPU can detect if one bit in the different Transmission Request Registers is set. Each bit of this register is dedicated to one byte of each register whereas the bit value is a logical OR of the value of the corresponding bits in the byte.

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x84	res															
	R-0															

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	TxRqstD				TxRqstC				TxRqstB				TxRqstA			
	R-0				R-0				R-0				R-0			

R = Read, W = Write, U = Undefined; -*n* = Value after reset

Table 24 Transmission request X register (address 0x84)

I.e: bit 0 of „Transmission Request X Register“ is dedicated to bits 7:0 of „Transmission Request A Register (**MOTRA**)“- if one or more bits in this byte are set, bit 0 of **MOTRX** will be set.

2.4.2 New Data Registers (MOND<x>)

These registers hold the **NewDat** bits of the configurable up to 128 Message Objects. By reading the **NewDat** bits, the CPU can check for which Message Object the data portion was updated. The **NewDat** bit of a specific Message Object can be set/reset by the CPU via the IFx “Message Interface” Registers or set by the Message Handler after reception of a Data Frame or reset by the Message Handler at start of a transmission.

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MONDA (0x9C) MONDB (0xA0) MONDC (0xA4) MONDD (0xA8)	NewDat<x>															
	R-0															

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	NewDat<x>															
	R-0															

R = Read, W = Write, U = Undefined; -*n* = Value after reset

Table 25 New data registers (addresses 0x9C-0xA8)

MONDA Bits 15:0	**NewDat16-1:**	New Data Bits (valid for all Message Buffer sizes)
MONDA Bits 31:16	**NewDat32-17:**	New Data Bits (valid for Message Buffer size of 32, 64 and 128 Message Objects)
MONDB Bits 31:0	**NewDat64-33:**	New Data Bits (valid for Message Buffer size of 64 and 128 Message Objects)
MONDC Bits 31:0	**NewDat96-65:**	New Data Bits (valid for Message Buffer size of 128 Message Objects)
MONDD Bits 31:0	**NewDat128-97:**	New Data Bits (valid for Message Buffer size of 128 Message Objects)

0= No new data has been written into the data portion of this Message Object either by the Message Handler nor via Interface Register since last time this flag was cleared by the CPU.

1= The Message Handler or the CPU has written new data into the data portion of this Message Object.

2.4.2.1 New Data X Register (MONDX)

With register „New Data X Register“ the CPU can detect if one bit in the different “New Data” Registers (**MONDA, MONDB, MONDC and MONDD**) is set. Each bit of this register is dedicated to one byte of each register whereas the bit value is a logical OR of the values of the corresponding bits in the byte.

Bits	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
0x98	res															
	R-0															

Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	NewDatD				NewDatC				NewDatB				NewDatA			
	R-0				R-0				R-0				R-0			

R = Read, W = Write, U = Undefined; *-n* = Value after reset

Table 26 New data X register (address 0x98)

I.e : bit 0 of „New Data X Register“ is dedicated to bits 7:0 of „New Data A Register“ (**MONDA**) - if one or more bits in this byte are set, bit 0 of „New Data X Register“ (**MONDX**) will be set.

2.4.3 Interrupt Pending Registers (MOIP<x>)

These registers hold the **IntPnd** bits of the configurable up to 128 Message Objects. By reading the **IntPnd** bits, the CPU can check for which Message Object an interrupt is pending. The **IntPnd** bit of a specific Message Object can be set/reset by the CPU via the IFx “Message Interface” Registers or set by the Message Handler after reception or after a successful transmission of a frame. This will also affect the value of **IntID** in the Interrupt Register.

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MOIPA (0xB0) MOIPB (0xB4) MOIPC (0xB8) MOIPD (0xBC)	IntPnd<x>															
	R-0															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	IntPnd<x>															
	R-0															

R = Read, W = Write, U = Undefined; -*n* = Value after reset

Table 27 Interrupt pending registers (addresses 0xB0-0xBC)

MOIPA Bits 15:0	**IntPnd16-1:**	Interrupt Pending Bits (valid for all Message Buffer sizes)
MOIPA Bits 31:16	**IntPnd32-17:**	Interrupt Pending Bits (valid for Message Buffer size of 32, 64 and 128 Message Objects)
MOIPB Bits 31:0	**IntPnd64-33:**	Interrupt Pending Bits (valid for Message Buffer size of 64 and 128 Message Objects)
MOIPC Bits 31:0	**IntPnd96-65:**	Interrupt Pending Bits (valid for Message Buffer size of 128 Message Objects)
MOIPD Bits 31:0	**IntPnd128-97:**	Interrupt Pending Bits (valid for Message Buffer size of 128 Message Objects)

Bits 31:0 IntPnd128-1: Interrupt Pending Bits (of all Message Objects)

0= This message object is not the source of an interrupt.
1= This message object is the source of an interrupt.

2.4.3.1 Interrupt Pending X Register

With register „Interrupt Pending X Register (**MOIPX**)“ the CPU can detect if one bit in the different Interrupt Pending Registers (**MOIPA, MOIPB, MOIPC, MOIPD**) are set. Each bit of this register is dedicated to one byte of each register whereas the bit value is a logical OR of the value of the corresponding bits in the byte.

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0xAC	res															
	R-0															
Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	IntPndD				IntPndC				IntPndB				IntPndA			
	R-0				R-0				R-0				R-0			

R = Read, W = Write, U = Undefined; -*n* = Value after reset

Table 28 Interrupt pending X register (address 0xAC)

I.e. : bit 0 of „Interrupt Pending X Register“ is dedicated to bits 7:0 of „Interrupt Pending A Register“ (**MOIPA**)- if one or more bits in this byte are set, bit 0 of „Interrupt Pending X Register“ will be set.

2.4.4 Message Valid Registers (MOVAL<x>)

These registers hold the **MsgVal** bits of the configurable up to 128 Message Objects. By reading the **MsgVal** bits, the CPU can check which Message Object is valid. The **MsgVal** bit of a specific Message Object can be set/reset by the CPU via the IFx “Message Interface” Registers.

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MOVALA (0xC4) MOVALB (0xC8) MOVALC (0xCC) MOVALD (0xD0)	MsgVal<x>															
	R-0															
Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	MsgVal<x>															
	R-0															

R = Read, W = Write, U = Undefined; -*n* = Value after reset

Table 29 Message valid registers (addresses 0xC4-0xD0)

MOVALA Bits 15:0	**MsgVal16-1:**	Message Valid Bits (valid for all Message Buffer sizes)
MOVALA Bits 31:16	**MsgVal32-17:**	Message Valid Bits (valid for Message Buffer size of 32, 64 and 128 Message Objects)
MOVALB Bits 31:0	**MsgVal64-33:**	Message Valid Bits (valid for Message Buffer size of 64 and 128 Message Objects)
MOVALC Bits 31:0	**MsgVal96-65:**	Message Valid Bits (valid for Message Buffer size of 128 Message Objects)
MOVALD Bits 31:0	**MsgVal128-97:**	Message Valid Bits (valid for Message Buffer size of 128 Message Objects)

0= This Message Object is ignored by the Message Handler.

1= This Message Object is configured and should be considered by the Message Handler.

2.4.4.1 Message Valid X Register (MOVALX)

With register „Message Valid X Register (**MOVALX**)“ the CPU can detect if one bit in the different “Message Valid (**MOVALA, MOVALB, MOVALC, MOVALD**)” Registers is set. Each bit of this register is dedicated to one byte of each register whereas the bit values is a logical OR of the value of the corresponding bits in the byte.

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0xC0	res															
	R-0															

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	MsgValD				MsgValC				MsgValB				MsgValA			
	R-0				R-0				R-0				R-0			

R = Read, W = Write, U = Undefined; -*n* = Value after reset

Table 30 Message valid X register (address 0xC0)

I.e: bit 0 of „Message Valid X Register“ is dedicated to bits 7:0 of „Message Valid A Register (**MOVALA**)“- if one or more bits in this byte are set, bit 0 of „Message Valid X Register“ will be set.

2.5 Interrupt functionality

2.5.1 Interrupt scheme

The D_CAN provides three groups of interrupt sources, these are:

* Error Interrupts are generated by bits **PER**, **BOff** and **EWarn** monitored in Status Register, see Chapter 2.2.2. This Error Interrupt group will be enabled by setting bit **EIE** which is located in CAN Control Register, see Chapter 2.2.1.
* **RxOk**, **TxOk** and **LEC** (monitored in Status Register, see Chapter 2.2.2) belong to the Status Interrupt group and can be enabled by **SIE** bit which is located in CAN Control Register, see Chapter 2.2.1.
* The Message Object interrupts, which are generated by events concerning the Message Object itself controlled by flags **IntPND**, **TxIE** and **RxIE** which are described in Chapter 2.3.2.

Error and Status interrupts can only be routed to interrupt port **CAN_INT_STATUS** which has to be enabled by setting **CCTRL.ILE**. The Message Object interrupts can be routed to interrupt port **CAN_INT_STATUS** or **CAN_INT_MO** controlled by **CCTRL.MIL**. Setting **CCTRL.MIL** to one, a Message Object Interrupt will set the port **CAN_INT_MO** to one.

2.5.2 Interrupt Register (CIR)

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x10	res															
	R-0															

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	StatusInt	res							IntID7-0							
	R-0	R-0							R-0							

R = Read, W = Write, U = Undefined; *-n* = Value after reset

Table 31 Interrupt register (address 0x10)

Bit 15 **StatusInt:** A Status Interrupt has occurred.

The Status Interrupt is cleared by reading the Status Register.

Bits 7:0 **IntId15-0:** Interrupt Identifier (the number here indicates the source of the interrupt)

0x00	No Message Object interrupt is pending.
0x01-0x80	Number of Message Object which caused the interrupt.[1]
0x81-0xFF	unused.

If several interrupts are pending, the CAN Interrupt Register will point to the pending interrupt with the highest priority, disregarding their chronological order. An interrupt remains pending until the CPU has cleared it. If **IntID** is different from 0x00 and **CCTRL.MIL** is set, the interrupt port **CAN_INT_MO** is active. The interrupt port remains active until **IntID** is back to value 0x00 (the cause of the interrupt is reset) or until **CCTRL.MIL** is reset. If **CCTRL.ILE** is set and **CCTRL.MIL** is reseted the Message Object interrupts will be routed to interrupt port **CAN_INT_STATUS**. The interrupt port remains active until **IntID** is back to value 0x00 (the cause of the interrupt is reset) or until **CCTRL.MIL** is set or **CCTRL.ILE** is reset.

The Message Object's interrupt priority decreases with increasing message number.

A message interrupt is cleared by clearing the Message Object's **IntPnd** bit.

1. Depends on the configured Message Buffer size.

Chapter 3.

3. Memory Map

Address	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Symbol (Reset Value)
0x000													DE2	DE1	MIL										Test	CCE	DAR	reserved	EIE	SIE	ILE	Init	CCTRL (0x0000_0001)
0x004																								PER	BOff	EWarn	EPASS	RxOK	TxOK	LEC			CSTS (0x0000_0007)
0x008																	RP	REC							TEC								CERCr (0x0000_0000)
0x00C													BRPE				reserved	TSeg2			TSeg1				SJW		BRP						CBT (0x0000_2301)
0x010																	StatusInt								IntID_MO								CIR (0x0000_0000)
0x014																									Rx	Tx1-0		LBack	Silent				CTR (0x0000_0080)[1]
0x018																													RAM-Init	reserved	ClkStReq	ClkStAck	CFR (0x0000_0000)
0x01C																				ParErrWord4	ParErrWord3	ParErrWord2	ParErrWord1	ParErrWord0	Message Object Number								PEC (0x0000_UUUU[2])
0x020	REL				STEP				SUBSTEP				YEAR				MON								DAY								CRR (0x111S_SSSS)[3]

Table 32 CAN Status and Configuration Register Map

Address	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Symbol (Reset Value)
0x024																														parityen	mb_w		HWS (0x0000_000S)[4]
0x028-0x080																																	reserved for future use (0x0000_0000)
Address	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Name

Table 32 CAN Status and Configuration Register Map

1.) CTR.Rx (Bit7) depends on Rx-Pad.
2.) PEC is not reset by the module-reset.
3.) CRR.YEAR, CRR.MON and CRR.DAY reset values depend on generic parameter set on D_CAN synthesis.
4.) HWS.parityen and HWS.mb_w reset values depend on generic parameter set on D_CAN synthesis.

Address	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Name (Reset Value)
0x084																	TxRqstD				TxRqstC				TxRqstB				TxRqstA				MOTRX (0x0000_0000)
0x088	TxRqst32							...									TxRqst16							...								TxRqst1	MOTRA (0x0000_0000)
0x08C	TxRqst64															...																TxRqst33	MOTRB (0x0000_0000)
0x090	TxRqst96															...																TxRqst65	MOTRC (0x0000_0000)
0x094	TxRqst128															...																TxRqst97	MOTRD (0x0000_0000)

Table 33 CAN Message Object Status Register Map

Address	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Name (Reset Value)
0x098																	NewDatD				NewDatC				NewDatB				NewDatA				MONDX (0x0000_0000)
0x09C	NewDat32	...															NewDat16	...														NewDat1	MONDA (0x0000_0000)
0x0A0	NewDat64	...																														NewDat33	MONDB (0x0000_0000)
0x0A4	NewDat96	...																														NewDat65	MONDC (0x0000_0000)
0x0A8	NewDat128	...																														NewDat97	MONDD (0x0000_0000)
0x0AC																	IntPndD				IntPndC				IntPndB				IntPndA				MOIPX (0x0000_0000)
0x0B0	IntPnd32	...															IntPnd16	...														IntPnd1	MOIPA (0x0000_0000)
0x0B4	IntPnd64	...																														IntPnd33	MOIPB (0x0000_0000)
0x0B8	IntPnd96	...																														IntPnd65	MOIPC (0x0000_0000)
0x0BC	IntPnd128	...																														IntPnd97	MOIPD (0x0000_0000)

Table 33 CAN Message Object Status Register Map

Address	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Name (Reset Value)
0x0C0																	MsgValD				MsgValC				MsgValB				MsgValA				MOVALX (0x0000_0000)
0x0C4	MsgVal32	...															MsgVal16	...														MsgVal1	MOVALA (0x0000_0000)
0x0C8	MsgVal64	...																														MsgVal33	MOVALB (0x0000_0000)
0x0CC	MsgVal96	...																														MsgVal65	MOVALC (0x0000_0000)
0x0D0	MsgVal128	...																														MsgVal97	MOVALD (0x0000_0000)
0x0D4 - 0x0FC																																	reserved for future use (0x0000_0000)
Address	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Name

Table 33 CAN Message Object Status Register Map

Address	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Name (Reset Value)
0x100			ClrAutoInc						wr1_rd0	update_mask	update_arb	update_control	ClrIntPnd	TxRqst/NewDat	update_dataB	update_dataA	busy	DMAactive	AutoInc						MONum								IF1CMR (0x0000_0001)
0x104	MXtd	MDir	res	standard_msk											ext_msk																		IF1MSK (0xFFFF_FFFF)
0x108	MsgVal	Xtd	Dir	statndard_id											ext_id																		IF1ARB (0x0000_0000)

Table 34 CAN appl. IF Register Map

Address	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Name (Reset Value)
0x10C																	NewDat	MsgLst	IntPnd	UMask	TxIE	RXIE	RmtEn	TxReqst	EoB				DLC				IF1MCTR (0x0000_0000)
0x110	DataByte3								DataByte2								DataByte1								DataByte0								IF1DA (0x0000_0000)
0x114	DataByte7								DataByte6								DataByte5								DataByte4								IF1DB (0x0000_0000)
0x118 - 0x11C																																	reserved (0x0000_0000)
0x120			ClrAutoInc						wr1_rd0	update_mask	update_arb	update_control	ClrIntPnd	TxRqst/NewDat	update_dataB	update_dataA	busy	DMAactive	AutoInc						MONum								IF2CMR (0x0000_0001)
0x124	MXtd	MDir	res	statndard_msk											ext_msk																		IF2MSK (0xFFFF_FFFF)
0x128	MsgVal	Xtd	Dir	standard_id											ext_id																		IF2ARB (0x0000_0000)
0x12C																	NewDat	MsgLst	IntPnd	UMask	TxIE	RXIE	RmtEn	TxReqst	EoB				DLC				IF2MCTR (0x0000_0000)
0x130	DataByte3								DataByte2								DataByte1								DataByte0								IF2DA (0x0000_0000)
Address	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Name

Table 34 CAN appl. IF Register Map

Address	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Name (Reset Value)
0x134	DataByte7								DataByte6								DataByte5								DataByte4								IF2DB (0x0000_0000)
0x138 - 0x1FC																																	reserved for future use (0x0000_0000)
Address	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Name

Table 34 CAN appl. IF Register Map

List of Figures

List of Tables