

VME - CAN4

CAN Controller for 4 CAN Networks

Manual of the local Firmware

NOTE

The information in this document has been carefully checked and is believed to be entirely reliable. **esd** makes no warranty of any kind with regard to the material in this document, and assumes no responsibility for any errors that may appear in this document. **esd** reserves the right to make changes without notice to this, or any of its products, to improve reliability, performance or design.

esd assumes no responsibility for the use of any circuitry other than circuitry which is part of a product of **esd gmbh**.

esd does not convey to the purchaser of the product described herein any license under the patent rights of **esd gmbh** nor the rights of others.

esd electronic system design gmbh

Vahrenwalder Str. 205
D-30165 Hannover
Germany

Tel: +49-511-37298-0
Fax: +49-511-37298-68
Email: info@esd-electronics.com
Internet: <http://www.esd-electronics.com>

Manual file	I:\TEXTE\DOKU\MANUALS\VME\CAN4\CAN4-12S.EN6
Date of print	04.10.1999

Software version described	
-----------------------------------	--

Changes in the chapters

The changes in the users' manual listed below affect changes in the **software** as well as changes in the **description** of facts only.

Chapter	Changes versus previous version
-	First English version.

Technical details are subject to change without further notice.

Content

1. Overview	1
1.1 About this Manual	1
2. Initializing the CAN4	3
3. Memory Structure of CAN4	5
3.1 Shared RAM Assignment	5
3.2 Local Link Structure	6
4. Description of the Various Memory Ranges	9
4.1 Data Structures CAN-Data 1...CAN-Data 4	9
4.1.1 Function and Structure	9
4.1.2 Bytes of a Data Structure Element	10
4.2 Control Structures CAN_CTRL 1... CAN_CTRL 4	16
4.2.1 Function and Structures	16
4.2.2 Bytes of a CTRL-Structure Element	17
5. Special Functions	19
5.1 Rx-Buffer	19
5.1.1 Function	19
5.1.2 Structure of Rx-Buffer	20
6. The Parameter Buffer	23
6.1 Function and Structure	23
6.2 Specifying Parameters and Commands	25
6.3 Description of Parameters	26
6.3.1 Writeable and Readable Paramete	26
6.3.2 'Only Readable' Parameters	27
6.4 Commands of Parameter Buffer	31
6.4.1 Overview of Implemented Commands	31
6.4.2 Description of Commands	34
7. User System Clock	43
7.1 Overview	43
7.2 Configuring and Activating the System Clock	43

1. Overview

1.1 About this Manual

This manual describes the local firmware of the CAN4. The firmware has been stored in the Flash EPROM.

2. Initializing the CAN4

During initialization of the CAN4 by the VMEbus master the VMEbus address A24 or A32 is assigned to the board. **The registers for setting the VMEbus basis addresses are described in the hardware manual.**

Initialization sequence (example):

1. Searching for CAN4 in A16 address range

2. Evaluating VMEbus structure

2.1 Initializing the basis address of the board in A32/D32 range:

- check, whether 32-bits data transfer (aligned) is successful
- if yes -> OK, exit
- if no -> P2 not is not equipped -> BUSERROR caused by missing address A24...A32
-> Read/Write check faulty -> 2.2

2.2 Initializing basis address of board in A24/D16 range:

- check, whether Read/Write OK
- if yes -> OK, exit
- if no -> hardware error?

3. Determine basis address of board

Further information on determining the address can be taken from the description of registers in the hardware manual.

3. Memory Structure of CAN4

3.1 Shared RAM Assignment

The following table shows the assignment of the Shared RAM range accessible via VMEbus. The address offset is shown in relative to the programmed VMEbus basis address.

All addresses (except for two cells of the link structure) are assigned dynamically. The following table is as at 19.03.97. The absolute values of the addresses may change in future upgrades! When programming you have to use the according pointer cells instead of the absolute address values, therefore!

The link structure of the CAN4 will be described in the following chapter.

Address range from... [HEX]	Assignment	Notes
000. 000 ...	Link structure read only	This memory area must only be read accessed by the user.
001. 000 ... 001. 100 ... 001. 200 ... 001. 300 ...	iobuff 1 iobuff 2 iobuff 3 iobuff 4	Command buffer of the 4 CAN channels
001. 400 ...	free memory	free memory under buffer management (direct access by user is not permissible)
020. 000 ... 030. 000 ... 040. 000 ... 050. 000 ...	CAN #1 CAN #2 CAN #3 CAN #4	data and control buffer of the 4 CAN channels
060. 000 ...	free memory	free memory under buffer management (direct access by user is not permissible)
07F. F00 ...	Irq_Trigger_Cells	interrupt trigger cells
080. 000 1FF. FFE	free memory	free memory under buffer management (direct access by user is not permissible) (only available, if option 4x 512 kbyte SRAM is equipped)
1FF. FF0	Read_Area_Clock	register of user-defined cycle handling

Table 3.1.1: Assignment of Shared RAM 1

3.2 Local Link Structure

Except for cells **c4i off** and **c4i olen**, which are on relative addresses +\$0 and +\$2, and the three long cells for identification, all addresses are managed dynamically. The address values are stored into memory cells.

All cells can only be read by the user and cannot be changed!

The following table shows the basis link structure. The first cell **c4i off** shows the relative address of the link structure of the first CAN channel. Structure and length of link structures are the same for all four channels. The length is specified in **c4i olen**.

Address offset [HEX]	Data width	Contents	Notes	Default value after RESET [HEX] (as at 20.03.97)
+0	word	ctoff	offset address of link structure of first CAN channel	10
+2	word	ctlen	length of link structures of CAN channels (in bytes)	10
+4	long	c4id	ASCII-ID	'CAN4'
+8	long	swrev	ASCII-ID	'5X__'
+C	long	swcms	ASCII-ID	'_CMS'

Table 3.2.1: Cells with pointers

The following table shows the structure of link structures **by means of the first CAN channel**. The structure starts from the address which has been specified in **c4i off**. The link structures of the other three CAN channels directly follow this structure. From the length of the structure (**i ol en**), therefore, the addresses of the link structures of the other channels can be determined.

Address offset c4i off + [HEX]	Data width	Contents	Notes	Default value after RESET [HEX] (as at 20.03.97)
+0	long	c4cdat	offset address of first CAN-data structure	20000
+4	long	c4i obf	offset address of first CAN-command channel	1000
+8	long	c4ctrq	interrupt trigger address for local interrupt (CAN1)	0007. FF00
+C	long	c4mtrq	interrupt trigger address for interrupt to VMEbus (CAN1)	0007. FF20

Table 3.3.1: Structure of link structures by means of CAN channel 1

With the current default settings the following link structures were to be found under the following addresses, therefore:

CAN-link structure 2: \$20

CAN-link structure 3: \$30

CAN-link structure 4: \$40

4. Description of the Various Memory Ranges

4.1 Data Structures CAN-Data 1...CAN-Data 4

4.1.1 Function and Structure

Data is exchanged between VMEbus and CAN via data-structure fields.

A structure field is divided into 2048 elements.

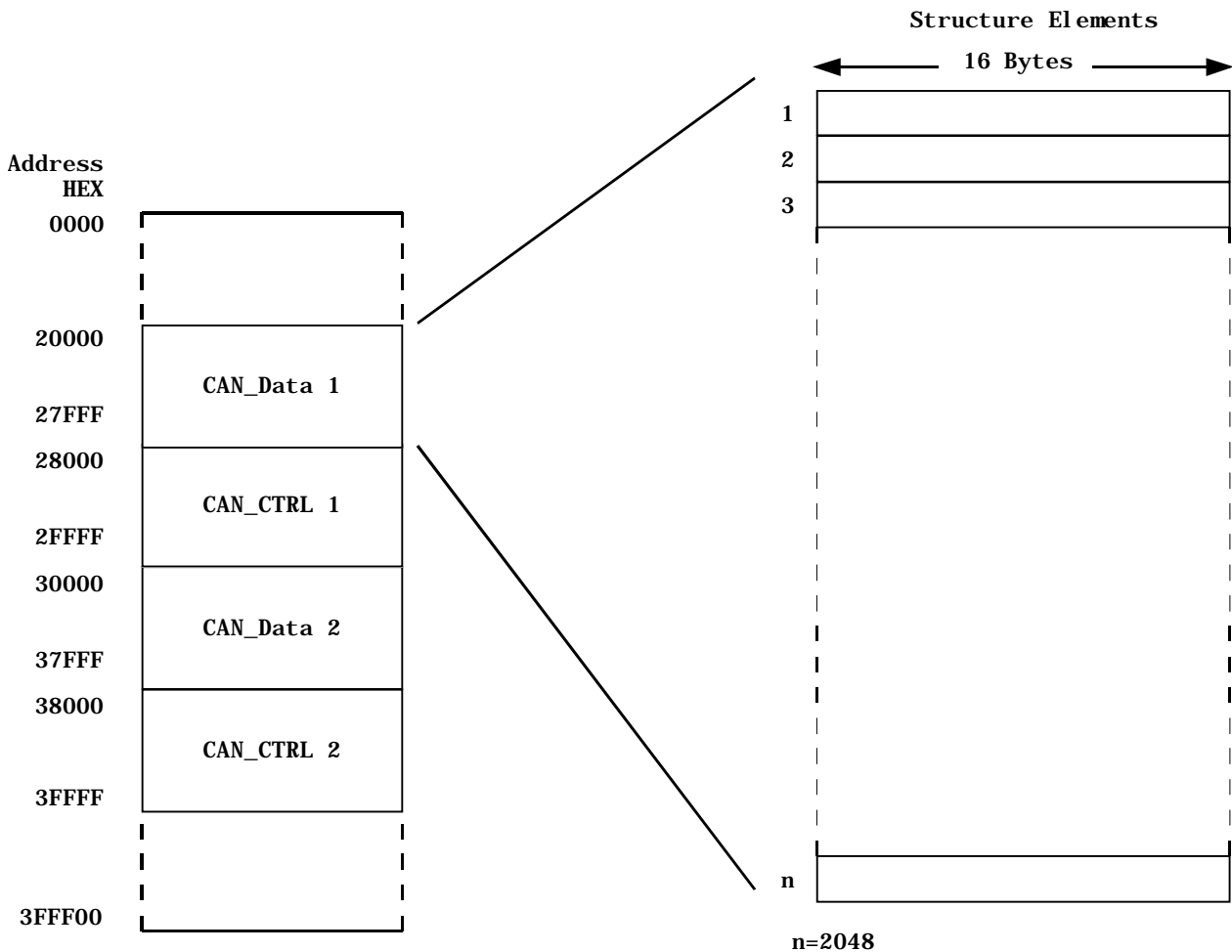


Fig. 4.1.1: Structure of a CAN_Data_Structure_Field (example: CAN-data 1, CAN-data 2)

A structure element is assigned to each possible CAN identifier of a CAN channel (max. 2048 Ids are possible).

A structure element is 16 bytes long and contains 8 bytes monitoring and control parameters apart from the data transmitted (max. 8 bytes).

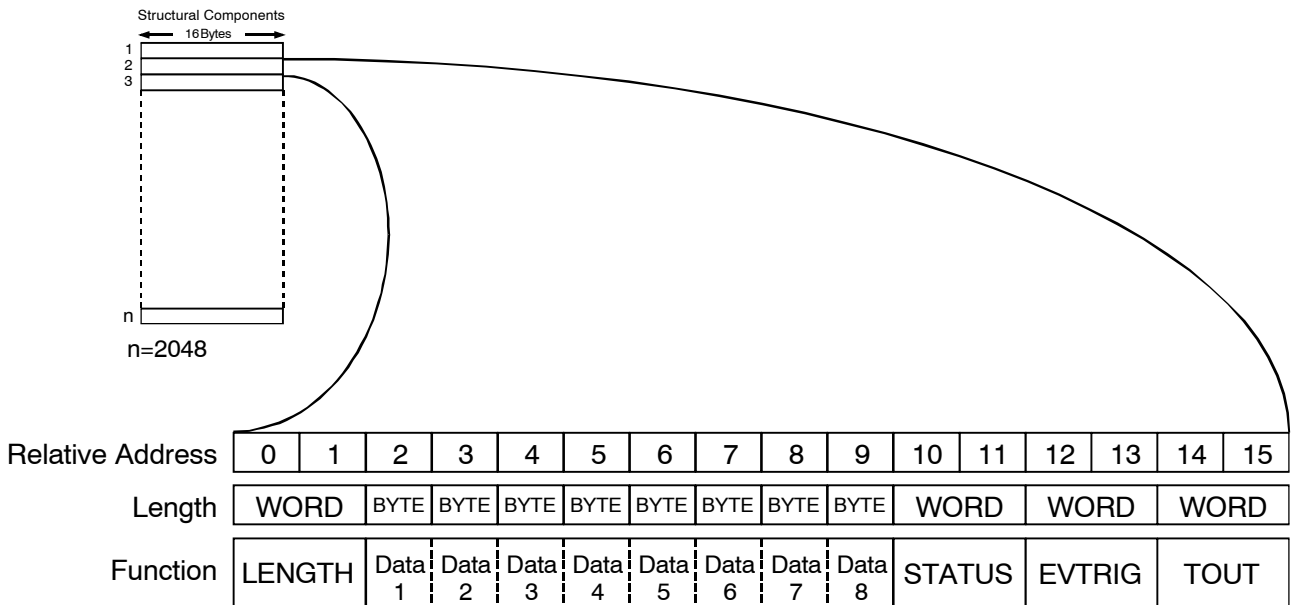


Fig. 4.1.2: Contents of a CAN_Data_Structure_Element

4.1.2 Bytes of a Data Structure Element

Below, the bytes of a data structure element is described in ascending order.

LENGTH... shows the number of valid data bytes and by means of cell 'MODE' determines the operating mode for this identifier in the according control-structure element (see chapter 'Parameter und Command Specification via Parameter Buffer').

Possible values [HEX]:

- 0000...0008 (only number of valid bytes)
- 0020...0028 (LENGTH in mode RTR-transmit)
- 0040...0048 (LENGTH in mode Rx-transfer with time out)
- 0060...0068 (number of valid bytes and starting Tx-transfer)

Data1..Data8... contain the data transmitted.
You can only transmit 0 to eight bytes.

STATUS... has information about the current status of transmission.
 Values of STATUS are generally:

- STATUS = 0 everything OK
- STATUS < 0 some kind of 'Busy', data invalid, transfer not finished
- 0 < STATUS < \$100 crucial error during transfer
- STATUS ≥ \$100 special status messages: RTR, ...

Especially with values under 0 you must not request via 'EQUAL \$FFxx' in program sequence, but only via negative values to make sure that new error-status values, which still have to be defined, will be recognized!

STATUS messages \$0013...\$0019 trigger the transmission of an Abort-Domain-Frame.

The following message have been implemented so far:

STATUS [HEX]	Label of local firmware	Note
FFFF FFFE FFFD FFFC FFFB	ERMBSY ERMWRX ERMTXQ ERMBS1 ERMTLK	Tx is busy, active (transmission not finished) waiting for Rx after remote request Tx is busy, in queue Tx is busy, from queue to active Tx is busy by node link
0000		OK, transmission successfully done
0001 0002 0003 0004 0005 0006 0007 0008	ERMRTO ERMTTO ERMres ERMTER ERMOFB ERMBUS ERMTUL	Rx-timeout Tx-timeout of active element released due to 'off-bus' (similar to \$005) Tx-error controller 'off-bus' controller 'busy' release due to INIBIT Tx-timeout of element in Tx-chain
0010 0013 0014 0015 0016 0017 0018 0019 0020	ERMbfe ERMcd1 ERMcd2 ERMcu1 ERMcu2 ERMton ERMsws ERMWST	CMS: domain timeout forced error (VME sets bfcmmnd=0, while domain -stat≠0) client: wrong ack at 'init_download' client: wrong ack at 'download_segment' client: wrong ack at 'init_upload' client: wrong ack at 'upload_segment' client/server: Rx-timeout server: 'wrong_state' CMS: compare error of CMS-watchdog
001C 001D 001E 001F	ERMtto ERMabx ERMwst ERMcmd	Tx-timeout on 'domain_transmit' 'abort_domain' received no server or client active wrong VME-command

STATUS [HEX]	Label of local firmware	Note
0101	RTRRX	RTR-frame received
0104	ERMTR	error on transmit RTR

Table 4.1.1: Meaning of values of STATUS word

Description of status messages:

Rx-timeout...
(error) No data has been received in this structure element within the time specified in cell 'TOUT'.

Time-out monitoring is activated after entering the according operating mode into the CTRL structure element of the identifier (enter 'MODE' via parameter buffer) by triggering cell 'LENGTH' of the CTRL structure element. After triggering the value specified in cell 'TOUT' is entered into the 'TIME COUNTER' of the CTRL element, the element is integrated into the 'Rx-Time Out chain' and the 'TIME COUNTER' is count down.

After the time out has expired this error status is always set - FIFO 'Data To VME' can be loaded by a pointer to the structure element and an interrupt can be triggered (see chapter 'Interrupts').

Tx-timeout...
(error) The structure element could not be transmitted within the time specified in cell 'TOUT'. The transmission attempt is aborted.

Time-out monitoring is activated after entering the according operating mode into the CTRL structure element of the identifier (enter 'MODE' via parameter buffer) by triggering cell 'LENGTH' of the CTRL structure element. After triggering the value specified in cell 'TOUT' is entered into the 'TIME COUNTER' of the CTRL element, the element is integrated into the 'Tx-Time Out chain' and the 'TIME COUNTER' is count down.

After the time out has expired this error status is always set - FIFO 'Data To VME' can be loaded by a pointer to the structure element and an interrupt can be triggered (see chapter 'Interrupts').

Tx-error... This error applies, if CAN controller 82C200 requests new data for transmission from the firmware via its Tx-interrupt, even though it had not yet set its 'Tx-complete' signal. The error message can trigger an entry into FIFO 'Data To VME', which can trigger a VMEbus interrupt.

controller 'off bus'... (error) If crucial errors occur during transmission attempts of CAN controller 82C200 (such as bit rate controller \neq bit rate bus), the controller 'exits' the bus, because further transmission attempts were useless.

If this error status occurs during active Tx-transmissions, this error message is not only entered into the current structure element, but also in all other Tx-structures already taken into the local priority chain. If the error occurs during the first transmission of a Tx-element, the message is only entered into this structure element, the transmission is aborted and the transmission of the following Tx-structure element is started. The error message can trigger an entry into FIFO 'Data To VME', which can trigger a VMEbus interrupt.

controller 'busy'... (error) This error occurs, if the local Tx-priority chain is empty, a Tx-command is transmitted via VMEbus, and the firmware receives the message 'Busy' for the usual request of the controller of its Tx-status, even though the controller should not be busy at this point (-> crucial error!).

The error message might trigger an entry into FIFO 'Data to VME', if cell 'EVTRIG' is set. This entry might trigger a VMEbus interrupt.

domain-timeout... A time out error has occurred in CMS domain transfer.

compare error... CMS watchdog has found an error while comparing the set point value with the actual value of toggle bit or status.

RTR-frame... received An RTR-frame has been received for this structure element. This frame requests the transmission of data contained in this element. The status message only remains active as long as no transmission has been made.

waiting for Rx... A 'Remote Request' has been transmitted on this identifier. The firmware now waits for the reception of a message for this structure element. The status is reset after a message has been received.

Tx is busy, active... The transfer of this structure element has not yet been finished. After it has been finished, the status is reset and removed from the local priority chain.

.....

EVTRIG... determines, whether an entry into FIFO 'Data to VME' is to be made. This entry might trigger a user-VME interrupt. EVTRIG is only an enable status for triggering an interrupt. The interrupt has to be enabled by the hardware, i.e. the interrupt vector register must have been set (See hardware manual, chapter 'Interrupt-Vector Register' for register description).

If these enable statuses have been complied with, an interrupt is triggered on the VMEbus with the level 'iolev' and the vector specified via 'iovec', when the interrupt status described below occurs.

Value 'EVTRIG' [HEX]	Function
0	no FIFO entry after an end status has occurred
0001 : 00FF	FIFO entry and possible triggering of a user IRQ after end status has occurred

Table 4.1.2: Function of EVTRIG cell

Valid end statuses are successful transmissions or transmission which have been terminated by an error:

1. transmission successful
2. Rx-time out (error)
3. Tx-time out (")
4. Tx-error (")
5. Controller 'Off Bus' (")
6. Controller 'Busy' (error)

Errors in CMS transfer always trigger the entry of a pointer into the FIFO, regardless of the value of cell EVTRIG!

TOUT... specifies the time out value in msec (operation without CMS protocol) or the watchdog guard time (operation with CMS protocol) for values greater than 0 (\$0001...\$7FFF).

Operation without CMS protocol

Time out monitoring is activated after the according operation mode has been specified in the CTRL structure element of the identifier (enter 'MODE' via parameter buffer) by triggering cell 'LENGTH' of the CTRL structure element. After triggering the value specified in cell 'TOUT' is entered into 'TIME COUNTER' of the CTRL element, the element is integrated into a time out chain and 'TIME COUNTER' is count down.

The time out option can be used for Tx- as well as Rx-transfers:

In Tx-mode it triggers a Tx-time out error, if no transmission had been made within the time specified. The Tx-time out is shown in cell 'STATUS' and might trigger an interrupt. Furthermore it is possible to start Tx-transfers of up to 64 different structure elements in periodical intervals via the parameter buffer.

In Rx-mode an Rx-time out is shown in cell 'STATUS', if no messages have been received on this identifier within the time out time. 'TIME COUNTER' is reset with each reception of a message.

If the parameter buffer cells mentioned above have not been set, the time out is disabled by negative TOUT values.
This is to be strictly avoided, because the system might crash, if the Tx-command cannot be handled!

'TOUT' is never to be set to '0' and values smaller than 5 msec in any operating mode, because the resolution of the timer is within this range!

'TOUT' [HEX]	Function
0000...7FFF	depending on implementation: without CMS: time out in msec 0005 => 5 msec (*) 7FFF => 32.767 sec with CMS: watchdog guard time 0005 => 5 msec 7FFE =>32.767 sec

Table 4.1.3: Function of 'TOUT' cell

Operation with CMS protocol

If the CAN4 is equipped with CMS implementation, positive TOUT values specify the CMS watchdog. The CMS watchdog will be described in more detail in its own chapter.

4.2 Control Structures CAN_CTRL 1... CAN_CTRL 4

4.2.1 Function and Structures

Control structures are used by the firmware to assign transmission parameters, store status information and variables and document the order of active data structure elements.

Each CAN_Data structure has a CAN-CTRL structure. A control structure element contains 16 bytes.

The direct setting of parameters of control structures is mainly limited to the firmware. **The user is only permitted to set cell XTTID directly!**

Structure elements are structured as follows:

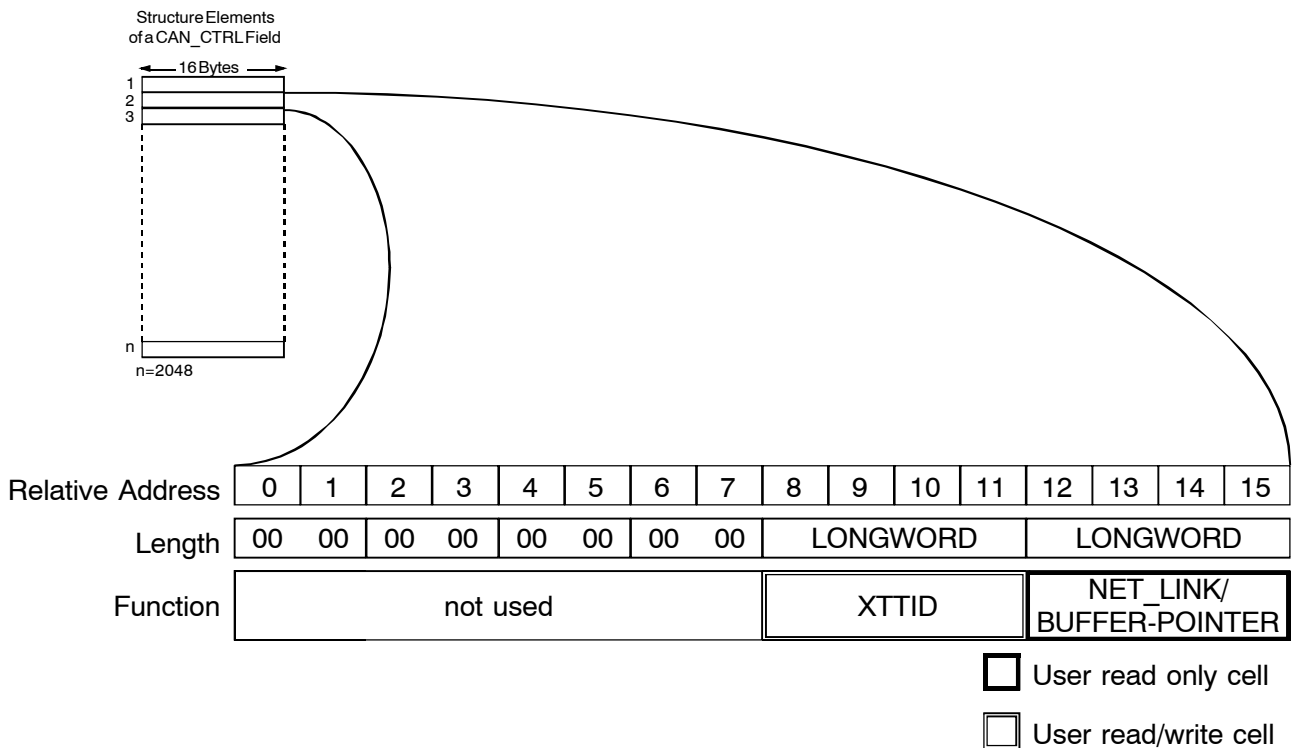


Fig. 4.2.1: Contents of a CAN_CTRL structure element

4.2.2 Bytes of a CTRL-Structure Element

Below, the bytes of a CTRL-structure element will be described in ascending order.

XTTID... This cell is not assigned by the local firmware. Users can store their own, freely selectable, parameters in four bytes. The cell can be used, for example, for storing the program counter of a user program into the cell.

In modes 'Rx-transfers disabled (Tx-only)', 'Rx-transfer with time-out option' and 'Transmit RTR', cell 'XTTID' can be used instead of cell 'EVTRIG' as required status for writing into FIFO 'Data to VME' and can therefore be used as interrupt status: If cells 'iolev' and 'iovec' of the 'CARD-Interrupt Enable' command are set in the parameter buffer, and cell 'XTTID' ≠ 0, a VMEbus interrupt is triggered as soon as a valid end status occurs.

NET_LINK/ BUFFER-POINTER...

If the identifier of this control structure element is used for CMS-transfer, the initial address of the CMS-buffer can be read in this cell after the buffer has been initialized by 'Init Domain' (see also command specification). By means of this initial address the addresses of relevant buffer cells can be determined.

5. Special Functions

5.1 Rx-Buffer

5.1.1 Function

Data of selected Rx-identifiers can be stored into Rx-buffers in order of their chronological reception.

If the command 'Get Rx-Buffer' is called via parameter channel, the Rx-buffer is installed by the local firmware. By means of command 'Set Mode' identifiers can be selected which are to be stored in this buffer. The commands of the parameter buffer will be explained in detail in a special chapter.

The data of identifiers selected for the Rx-buffer are not stored in the data-structure element, but in a memory area whose basis address will be returned as return parameter when executing the command.

	Rx-buffer
Buffer capacity	programmable (max. 4096 messages) (buffer is always overwritten again: 'Wrap Around')
Trigger for storing	no trigger, storing starts immediately after mode has been selected
Time stamp	1024 μ s
VMEbus interrupt	VME-IRQ possible when first data arrive

Table 5.1.1: Summary of features of Rx-buffer

5.1.2 Structure of Rx-Buffer

The Rx-buffer can be divided into a header and a data area. The header contains the pointers and the number of valid elements in the buffer.

The data area contains, apart from received data, further parameters such as identifiers and network No..

Relative address [HEX]	Length	Access	Cell contents
0	WORD	read only	Rx-Write-Pointer (WRP)
2	WORD	read/write	Rx-Read-Pointer (RDP)
4	WORD	read only	Rx-Mask (number of elements contained)
6 : E	-	-	reserved
10 (BuffStart) : ??	is programmed via command 'Get Rx-Buffer' in parameter buffer	read only	data area

Table 5.1.2: Structure of Rx-buffer

Rx-Write-Pointer... This pointer points to the buffer cell (relative to 'BuffStart') which will be overwritten after the next RxId has been received.
 Example for counting mode of pointers:
 \$0000, \$0010, \$0020,... ..., \$0FF0, \$0000
 After a RESET the pointer is set to '0'.

Rx-Read-Pointer... The Read-Pointer points to the next cell which will be read out. After a RESET the pointer is set to '0'.

The contents of Rx-Write-Pointer and Rx-Read-Pointer are used as status for writing FIFO 'Data To VME' and therefore as status for triggering a VMEbus interrupt as well: After new data has been received in status 'Rx-Read-Pointer=Rx-Write-Pointer', the FIFO will be set.

Rx-Mask... This parameter contains the capacity of the data area of the Rx-buffer. Rx-Mask is determined from value 'Buffer Length', which can be specified in the parameter buffer by the user during buffer initialization (see also description of parameter buffer): The specified value is rounded by the firmware to values of 2^n and is entered into Rx-Mask (this means 2, 4, 8, 16, 32...).

Data area... In the data area the data which has been received is stored. Each Rx-identifier can use a line with 16 bytes to store status information and data.

The following figure represents the structure of a line:

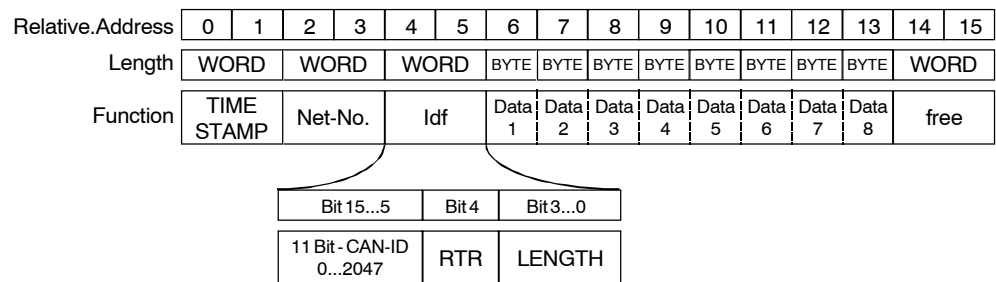


Fig. 5.1.1: Structure of the data area of an identifier

TIME STAMP... time (in [1024 μ s] steps) which has passed since the last system start (RESET or power-on).

Net-No. ... number of CAN network on which the RxId has been received (local CAN interface 0, 1, 2, 3).

Idf ... contains the 11-bit long CAN-Id, the RTR-bit (Remote Transmission Request) and the number of bytes transmitted (LENGTH):

- CAN-Id... possible values: 0...2047
- RTR-Bit... indicates, whether a data transfer has been requested under this identifier
- RTR = '0'... data transfer
- RTR = '1'... remote transmission request
- LENGTH... indicates the number of data bytes transmitted. Possible values: 0...8

Data1..Data8... contain the data received.

6. The Parameter Buffer

6.1 Function and Structure

By means of the parameter buffer various commands and parameters of CAN interfaces are specified. Each CAN channel has got its own parameter buffer.

The following figure represents the structure of the parameter buffer and the parameters relevant to the user.

Address
Offset
HEX

	+0	+2	+4	+6	+8	+A	+C	+E
00	i ofor		i oback		i otyp		i oname	
10	sema	stat	i ocmmmd		i onext		—	i obnum i olen
20	i obuff		—	—	ioirq	—		—
30	—		—		—		—	
40	—		Bi otyp		Bi ocms			
50	—		—		—		—	
60	—		—		—		—	
70	—		—		—		i ottic	i olcmd
80	para1	para2	para3	para4	par32		irtrig	
90	Mon_Base		Mn_S_Len	MonMax	—		—	
A0	swid	swbr	actbr	Struct_Base		S_Len	S_Max	—
B0	—	C_stat	ptr_tab_a*		ptr_tab_b*		len_b*	ids_b*
C0	—		—		—		—	
D0	—		—		—		—	
E0	i owdgm	—	—		—		i oclnk	
F0	—		local	irqmsk	ackirq		ownack	

	User read- only cells
	User read/write cells
	User don' t care

Fig. 6.1.1: Structure of parameter buffer

The following figure shows the parameter buffer with its default settings after a RESET on CAN4.

Address Offset HEX	+0	+2	+4	+6	+8	+A	+C	+E
00	00 00	80 00	00 00	00 00	00 0C	' CANP' z. B. ' 5' ' 8'		
10	sem stat	FF FF	00 00	yy yy	00 00	00 00	00 uu	00 80
20	00 00	vv 80	00 00	00 FF	00 00	00 00	00 00	00 00
30	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00
40	00 00	00 00	' C200'		' ___CMS'			00 00
50	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00
60	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00
70	00 00	00 00	00 00	00 00	00 00	00 00	02 FB	xx xx
80	00 00	00 00	00 00	00 00	00 00	00 00	00 07	FF ww
90	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00
A0	00 uu	00 0F	00 11	Struct_Base		00 10	08 00	00 00
B0	00 00	00 0C	00 00	00 00	00 00	00 00	00 00	00 00
C0	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00
D0	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00
E0	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00
F0	00 00	00 00	00 00	xx xx	00 47	FF rr	00 47	FF ww

- | |
|--|
| |
|--|

 User read-only cells
- | |
|--|
| |
|--|

 User read/write cells
- | |
|--|
| |
|--|

 User don't care

CAN channel	Default setting of channel-dependent variables (as at 19.03.97)					
	rr	uu	vv	ww	xx xx	yy yy
1	80	00	01	00	81 01	01 00
2	90	01	02	08	82 02	02 00
3	A0	02	03	10	84 04	03 00
4	B0	03	04	18	88 08	00 00

Fig. 6.1.2: Default setting of parameter buffer after a RESET

6.2 Specifying Parameters and Commands

Parameters and commands of a VMEbus MASTER-CPU of CAN4 have to be set in the following order:

1. - Evaluating cell '**sema**'--> is parameter buffer available?
(via TAS iosema (multiprocessor) or BSET=7, iosema)
 - If available, assigning parameter buffer by setting cell '**sema**' ('TAS' command)
2. - Setting parameters and commands by writing into desired cells.
3. - Transmitting parameters and commands to CAN4 slave server by triggering command IRQ (pointer in 'irtrig')
3. - Waiting for end of command processing: 'Cmmd' = FFFF
5. - Evaluating cell '**stat**':
 - 00 --> OK
 - FF --> ERROR
6. - Enabling cell '**sema**'.

6.3.2 Only Readable Parameters

i of or, i oback...	(\$00, \$04, long words) internal management pointers
i otyp...	(\$08, word) buffer identifier (always \$000C)
i oname...	(\$0A, 6 byte ASCII) contains buffer identifier as ASCII string and is serially numbered (always 'CANP').
stat...	(\$11, byte) contains information about the correct transmission of a command (see also the following chapter 'Commands of Parameter Buffer'). 'stat' = \$00 --> no error 'stat' P = \$00 --> error 'stat' = \$FF --> Cmmd-Error 'stat' = \$FE --> Para-Error Commands should generally be transmitted in the following order: 1. assign ' sema ' 2. transmit parameters and commands 3. trigger CMD-IRQ 3. wait for 'Cmmd' = FFFF or enable interrupt (iosema_0='1') 5. request ' stat ': = 00 --> OK P 00 --> ERROR 6. enable ' sema '
i onext...	(\$14, long word) contains offset address of the parameter buffer and the following CAN channel. i onext of the last buffer points to the first buffer again.
i ol ev...	(\$18, byte) is not assigned by CAN4 firmware
i ovec...	(\$19, byte) is not assigned by CAN4 firmware
i obnum...	(\$1C, word) contains buffer No. CAN channel 1 -> 0 CAN channel 2 -> 1 CAN channel 3 -> 2 CAN channel 4 -> 3
i ol en...	(\$1E, word) shows length of the data area of the parameter buffer. Default: ' i ol en ' = \$80 == 128 bytes.

i obuff...	(\$20, long word) pointer to the data area assigned to the buffer. In default setting ' i obuff ' points to cell ' para 1 '.
i oi rq...	(\$27, byte) = '00' -> 'Cmd_to_VME_IRQ' is NOT supported ≠ '00' -> 'Cmd_to_VME_IRQ' is supported
Bi otyp...	(\$44, 4 bytes) returns the type of CAN controller used in ASCII code: 'C200' -> 82C200
Bi ocms...	(\$48, 6 bytes) shows in ASCII code, whether the CMS protocol is implemented on board: '__CMS' -> CMS protocol implemented
i otti c...	(\$7C, word) determines resolution interval of time stamp. The cell contains the smallest unit of time in ns. Default: ' timetic ' = \$02FB (DEC 763 ns) -> hardware/CPU-clock-dependent = \$0000 -> not supplied = = 4000 nsec (CAN2)
i ol cmd...	(\$7E, word) informs the programmer and contains the command which has been handled last.
Mon_Base...	(\$90, long word) contains the local address of the monitor buffer. Default: ' Mon_Base ' - \$0 (no monitor)
Mon_S_Len...	(\$94, word) contains the length of a monitor structure element. Default: ' Mon_S_Len ' - \$0
MonMax...	(\$96, word) contains the maximum number of structure elements in the monitor buffer. Default: ' MonMax ' - \$0
swi d...	(\$A0, word)
swbr...	(\$A2, word) always \$0F
act br...	(\$A4, word) contains the topical bit rate index. The bit rate index can be modified via a command entry (see also chapter 'Transmission of Commands via Parameter Channel')

The following table shows the values of the bit rate depending on the register contents (bit rate index) of the controller, and each physically maximum attainable line length.

Index [HEX]	Bit rate [kbit/s]	actbr 82C200 or SJA1000 register		typical values of attainable line length l_{ax} [m]	minimum attainable line length l_{ein} [m]
		BTR0 [HEX]	BTR1 [HEX]		
0	1000	00	14	37	20
1	666.6	00	18	80	65
2	500	00	1C	130	110
3	333.3	01	18	180	160
4	250	01	1C	270	250
5	166	02	1C	420	400
6	125	03	1C	570	550
7	100	04	1C	710	700
8	66.6	45	2F	1000	980
9	50	09	1C	1400	1400
A	33.3	4B	2F	2000	2000
B	20	18	1C	3600	3600
C	12.5	5F	2F	5400	5400
D	10	31	1C	7300	7300
E	800	00	16	59	42

The specifications in the table are based on limit values of the bit timing of the CAN protocol, the delays of the local CAN interface and the delays of the cable. The delay of the cable is assumed at about 5.5 ns/m. Further influences, such as by terminating impedances, resistivity, cable geometry or external disturbances have not been considered in these specifications!

Table 6.3.1: Setting of bit rate

Struct_Base...	(\$A6, long word) contains local basis address of CAN data structure (CAN_Data). The basis address of the control structure (CAN_CTRL) is always ' Struct_Base ' + \$8000.
S_Len...	(\$AA, word) contains the length of a data structure element: Default: ' S_Len ' = \$0010 (16 bytes)
S_Max...	(\$AC, word) contains the number of data structure elements of the data structure field (CAN_Data) Default: ' S_Max ' = \$0800 (DEC 2048)
C_stat...	(\$B2, word) contains status of CAN controller components 82C200 or SJA1000 in bit 0...7, according to the corresponding manual.
ptr_tab_a*...	(\$B4, long word) reserved for future applications
ptr_tab_b*...	(\$B8, long word) reserved for future applications
len_b*...	(\$BC, word) reserved for future applications
ids_b*...	(\$BE, word) reserved for future applications
owdgm...	(\$E0, word) CAN4 special: Mode 'node_guarding' when VME watchdog fails: \$0000 - (default) - ignore watchdog \$0001 - abort 'node_guarding'
oclnk...	(\$EC, long word) local link pointer for clock server
local ...	(\$F4, word) is only assigned by firmware
irqmsk...	(\$F6, word) reserved for future applications
ackirq...	(\$F8, long word) contains local address for triggering 'Cmd_to_VME_IRQ'
ownack...	(\$FC, long word) contains local address for resetting 'Cmd_IRQ' (points to same cell as 'irtrig')

6.4 Commands of Parameter Buffer

6.4.1 Overview of Implemented Commands

The sequence of command transmissions has already been explained in chapter 'Specifying Parameters and Commands'.

Cell '**stat**' in the parameter buffer acknowledges the transmission of commands:

' stat '	=	\$00	-->	no error in command transmission
' stat '	P	\$00	-->	error in transmission
' stat '	=	\$FF	-->	Cmmd-Error
' stat '	=	\$FE	-->	Para-Error

The desired command is entered in cell '**icmmd**'. Cells '**para1**' to '**para4**' and '**par32**' store the parameters required for the according command.

Cells of parameter buffer							Command
i ocmmmd [HEX]	Program Label	para1	para2	para3	para4	par32	
0 or 1 (equal)	svini0	bit rate \$0000.. \$000F or \$0011.. \$7F7F	-	-	-	-	init CAN net 1 with bit rate
(4) or (5) (equal)	svmon0	-	M_code1	M_mask1	-	-	trigger monitor 1 or monitor 2 (without function at the moment)
6	nodlk0	Net_A 0,1	Idf_A 0...2047	Net_B 0,1	Idf_B 0...2047	-	link Net_A/Idf_A to Net_B/Idf_B (not to be used for new applications!)
7	nodlk1	Net_A 0,1	Idf_A 0...2047	Net_B 0,1	Idf_B 0...2047	-	unlink Net_A/Idf_A from Net_B/Idf_B (not to be used for new applications!)
8	autlk0	-	Idf 0...2047	-	-	-	activate periodical Tx-transfers (or CMS Watchdog TxId)
9	autlk1	-	Idf 0...2047	-	-	-	stop periodical repeat of periodical Tx-transfers (or CMS RTR)
(A)	iniirq	iolev \$0001... \$0007	iovec \$0000..\$00FC	-	-	-	CARD interrupt enable (not suitable for new applications - command triggers error message)
B	svirq	0	Idf_start 0...2047	Idf_end 0...2047	MODE	-	set MODE
C	domwsp	-	client TxId 0...2047	client RxId 0...2047	buffer length 1...32 kbytes	<i>returns</i> buffer address	init domain (only CMS transfers)
D	domrbf	-	client TxId 0...2047	client RxId 0...2047	-	<i>returns</i> buffer address	release CMS buffer
E	grxbuf	handle-no.	buffer length	-	-	<i>returns</i> buffer address	get Rx buffer
F	rrxbuf	handle-no.	-	-	-	-	release Rx buffer
-10	idfins	-	handle 0...2047	ext_mode 0,1	dyn_mode 0,1,2,3	idf 29/11 0...2947 or 0...2 ²⁹ -1	only if 29-bit identifier required: Insert Identifier (is not yet being supported)
11	idfdel	-	handle 0...2047	-	-	-	only if 29-bit identifier required: Release Identifier (is not yet being supported)
12	gdlbuf	buffer length	-	-	-	<i>returns</i> download buffer address	get code download buffer (is not yet being supported)
13	coddld	-	-	-	-	<i>set</i> buffer address	copy downloaded program in Flash- EPROM and RESET board (is not yet being supported)
14	gtxbuf	handle-no.	buffer length	-	-	<i>returns</i> buffer address	get Tx buffer
15	rtxbuf	handle-no.	-	-	-	-	release Tx buffer

Cells of parameter buffer							Command
i ocmmmd [HEX]	Program Label	para1	para2	para3	para4	par32	
16	ndlink	idfnet_1	idfnet_2	idfnet_3	idfnet_4	-	4-channel net link
17	ndulnk	idfnet_1	idfnet_2	idfnet_3	idfnet_4	-	4-channel net unlink
FFFF		-	-	-	-	-	'last command done' (Default setting after RESET)

Parameter values marked by '-' are not evaluated.

Table 6.4.1: General commands of parameter buffer

6.4.2 Description of Commands

In this chapter the individual commands and the corresponding parameters will be described in ascending order.

6.4.2.1 Setting the Bit Rate (iocmmd = 0)

The bit rate of the CAN channel is set again. A table with the assignment of parameter 'bit rate' to the bit rate can be found under the description of buffer cell '**actbr**'.

The bit rate is specified by directly entering it into the 16-bit wide register BTR0 + BTR1 of the CAN controller.

In active (unfinished) transmissions changing the bit rate would probably cause errors. Therefore, the handling of all structure elements which are in waiting status is aborted with the according error message.

6.4.2.2 Triggering the Monitor Function (iocmmd = 4)

This function is not being supported at the moment.

6.4.2.3 Linking CAN Identifiers of Different Networks (iocmmd = 6, 7)

This command is not to be used for new applications! It is only supported for reasons of compatibility to VME-CAN2.

By means of this command the CAN-Net_A-Identifier Idf_A is bidirectionally linked to CAN-Net_B (Idf_B) (or the link is removed again). This means that all data with identifier 'A' received in Net_A are transmitted to Net_B with Idf. 'B', and all data received in Net_B are transmitted to Net_A.

For reasons of compatibility to VME-CAN2 the networks can only be linked between channels 1 and 2 or 3 and 4.

Parameters 'Net_A' = 0/1 and
 'Net_B' = 0/1

 channels 1, 3 -->'0'
 channels 2, 4 -->'1'

In addition to simply pass on CAN frames from one network to another it is possible to transmit and receive message in the networks. For this the two identifiers of the networks are seen as one identifier, that means if, e.g., a message is transmitted via identifier A/Net_A, it is simultaneously also transmitted on identifier B/Net_B.

6.4.2.4 Periodical Transmission of Tx-Messages (iocmmd = 8, 9)

This command repeats Tx-transfers periodically or resets this mode when using standard EPROMs without CMS-implementation.

CMS-implementation

If CMS has been implemented into the EPROMs used, the function 'Periodical Transmission' does not apply. Instead, CMS-watchdog Tx-identifier and network number are transmitted via this command. The watchdog identifier is specified in cell 'para2' and the network number in cell 'para1'.

Initialization

This mode is initialized by transmitting parameters 'iocmmd', 'Net' and 'Idf' to the parameter buffer.

Activation

The operating mode becomes active immediately after initialization.

The period time of Tx-frames on the CAN, as specified in 'TOUT', might be incorrect in the range of ± 4 ms.

End

The periodical Tx-transfers are ended by resetting the command in the parameter buffer or setting the time out to negative values.

6.4.2.5 Select Operating Mode (MODE) (iocmmd = \$B)

By means of this command operating modes can be selected for identifier groups or individual identifiers. If the operating mode is to be selected for only one identifier, the same value has to be entered in cells 'Idf_start' and 'Idf_end'.

The command 'Set Mode' is called via the parameter buffer by entering the value \$B in cell iocmmd. Cells para1...para4 have to be set as in the previous chapter 'Overview of Commands Implemented'. In cell para4 the value for MODE is entered.

In the operating modes selected various transfers can be made via the identifiers. Sometimes further action is required, as for instance setting cell 'length' in the assigned data structure element when transmitting. Chapter 7 describes the correct way of executing all possible transfers.

The following table gives an overview of the possible transfers in the operating modes selected:

Entry for MODE	Rx-transfers possible	Tx-transfers possible	Automatic Tx on RTR possible	Notes
0	no	yes	no	basic Rx/Tx-transfers
1	yes	yes	no	
2	yes	yes	yes	
4	(yes)	no	no	special case: monitor MODE (not yet implemented)
5	(yes)	no	no	special case: CAN serial MODE (not yet implemented)
000F	This entry resets the identifier/s into the previous operating MODE (no operating MODE).			
80xy	?	?	?	special case: handle assignment for Rx-buffer (not yet implemented)
81xy	?	?	?	special case: handle assignment for Tx-buffer (not yet implemented)

Table 6.4.2: Overview of implemented modes

6.4.2.6 CMS-Initialization (iocmmd = \$C, \$D)

For the upload or download via CMS-protocol the local firmware requires a memory area (CMS-buffer) in order to be able to transmit parameters and store data.

By means of command 'init domain' the CMS-buffer is installed and the Rx- and Tx-identifiers which have been selected for transfers are specified.

The buffer length specified regards to the length of the CMS-data buffer which is to be made available. Values of up to 32kbytes can be selected for the length.

After the command has been executed, the absolute local initial address of the CMS-buffer can be read in parameter 'par32'.

If the CMS-buffer is not required anymore, the memory area can be released again (iocmmd = \$D). Only the identifiers and the network number are transmitted as parameters.

6.4.2.7 Rx-Buffer (iocmmd = \$E, \$F)

Only when using the Rx-buffer functionality:

Before assigning the identifiers you have to execute command 'Get Rx-buffer'. The local firmware then installs an Rx-buffer. Each Rx-buffer receives a so-called handle No. when the command 'Get Rx-buffer' is executed.

Then the identifiers are selected which are to be stored in the buffer. Identifiers and buffers are assigned by means of the handle numbers: handle numbers can be assigned to individual identifiers or identifier blocks by means of the MODE assignment.

Identifiers with the same handle No. are therefore stored into the same Rx-buffer.

The identifiers are stored in the Rx-buffer by executing this command. After the command has been executed the absolute initial address of the Rx-buffer can be read in parameter 'par32'.

By means of command 'Release Rx-buffer' (iocmmd = \$F) the memory area is released again.

6.4.2.8 Insert/Release Identifier (only for 29-bit Identifiers) (iocmmd = \$10, \$11)

The 29-bit identifier assignment is not yet being supported.

6.4.2.9 Handle Code Download Buffer (iocmmd = \$12, \$13)

This function is used to load program updates. It is not yet being supported.

6.4.2.10 Tx-Buffer (iocmmd = \$14, \$15)

The documentation of the Tx-buffer has yet to be worked out.

6.4.2.11 Net Link Function for up to Four Networks (iocmmd = \$16, \$17)

By means of this command identifiers of different networks can be linked. If two identifiers are to be linked in two different CAN networks, they have to be entered in the parameter buffers of *both* networks. The same applies for four networks. The parameter structure is as follows:

Parameters of parameter buffer:	para1	para2	para3	para4
Net link identifiers:	idfnet_1	idfnet_1	idfnet_1	idfnet_1
Link with CAN channel...	network 1	network 2	network 3	network 4

Examples:

1. Identifier A in CAN network 2 is to be linked bidirectionally to identifier B in CAN network 4.

Entry in parameter buffer of network 2:			
para1	para2	para3	para4
idfnet_1	idfnet_2	idfnet_3	idfnet_4
0	identifier A	0	identifier B

Entry in parameter buffer of network 4:			
para1	para2	para3	para4
idfnet_1	idfnet_2	idfnet_3	idfnet_4
0	identifier A	0	identifier B

2. Identifier A in CAN network 1 is to be linked bidirectionally to identifier B in CAN network 2, identifier C in CAN network 3 and identifier D in CAN network 4. The identifiers of networks 2, 3 and 4, however, are *not* to be linked with each other.

Entry in parameter buffer of network 1:			
para1	para2	para3	para4
idfnet 1	idfnet_2	idfnet_3	idfnet_4
identifier A	identifier B	identifier C	identifier D

Entry in parameter buffer of network 2:			
para1	para2	para3	para4
idfnet_1	idfnet 2	idfnet_3	idfnet_4
identifier A	identifier B	0	0

Entry in parameter buffer of network 3:			
para1	para2	para3	para4
idfnet_1	idfnet_2	idfnet 3	idfnet_4
identifier A	0	identifier C	0

Entry in parameter buffer of network 4:			
para1	para2	para3	para4
idfnet_1	idfnet_2	idfnet_3	idfnet 4
identifier A	0	0	identifier D

If an RTR were received on identifier D / network 4 in this constellation, the RTR is passed on to identifier A /network 1. Identifier A / network 1 then responds with a Tx-frame which is again transmitted via identifier D / network 4, but also via identifier B / network 2 and identifier C / network 3.

7. User System Clock

7.1 Overview

The user clock is a custom development for synchronizing several CAN4 boards. It is available as an option only. The CAN4 can either be operated as system clock master or as system clock slave. The cycle line can be connected via special signals on the CAN connector or via P2.

A detailed description of the system clock functionality can be taken from the hardware manual.

The system clock is configured via the VMEbus master.

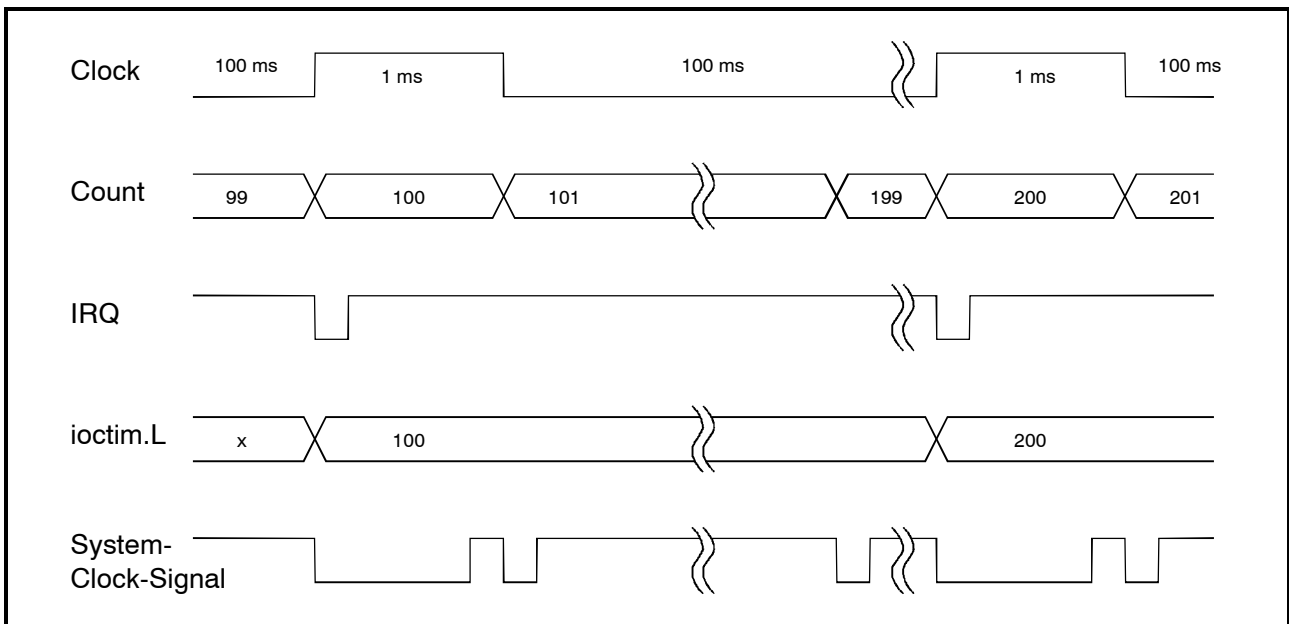


Fig. 7.1.1: Generating system clock

7.2 Configuring and Activating the System Clock

The system clock is configured and activated by means of the parameter buffer of the first CAN channel.

Cells of parameter buffer							Command
i ocmmmd [HEX]	Program Label	para1	para2	para3	para4	par32	
1F	-	clock_MODE	count_mode	ref_mode	irq_mode	time	configure system clock

Table 7.2.1: System clock commands in the parameter buffer of the first CAN channel

Description of Command Parameters:

- clk_mode... 0 - disable all (all other parameters will not be evaluated in this case)
1 - master MODE
2 - slave MODE
- count_mode... 0 - count binary
1 - count msec/day
- ref_mode... 0 - free run
1 - reference is local RTC (Real Time Clock)
- irq_mode... 0 - no interrupt
1 - interrupt on 100 ms pulse
- time... <0 - clear msec counter and start counting
≥ 0 - load counter and start synchronisation to reference

Status Cells:

The status cells are in the parameter buffer of the first CAN channel.

Name of cell	Address <small>(relative to basis address of 1. parameter buffer - at the moment \$01.000)</small>	Access	Description
iocirs	+\$E4	word	interrupt status, here: always '0'
iocmod	+\$E6	word	interrupt MODE = parameter 4 of command, such as 001
ioctim	+\$E8	long	here: msec count with 100 ms edge

Table 7.2.2: Status cells of system clock