VME - CAN2

Intelligent VMEbus / CAN-Bus Interface Board

Software Manual

N O T E

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 Hanover
Germany

Phone: +49-511-372-980
USA: 1-800-504-9856
FAX: +49-511-633-650
E-mail: pm@esd.h.eunet.de

Firmware-Version	cansrv58, cansrv58.cms

Content **Page**

1. Introduction

1.1 Overview

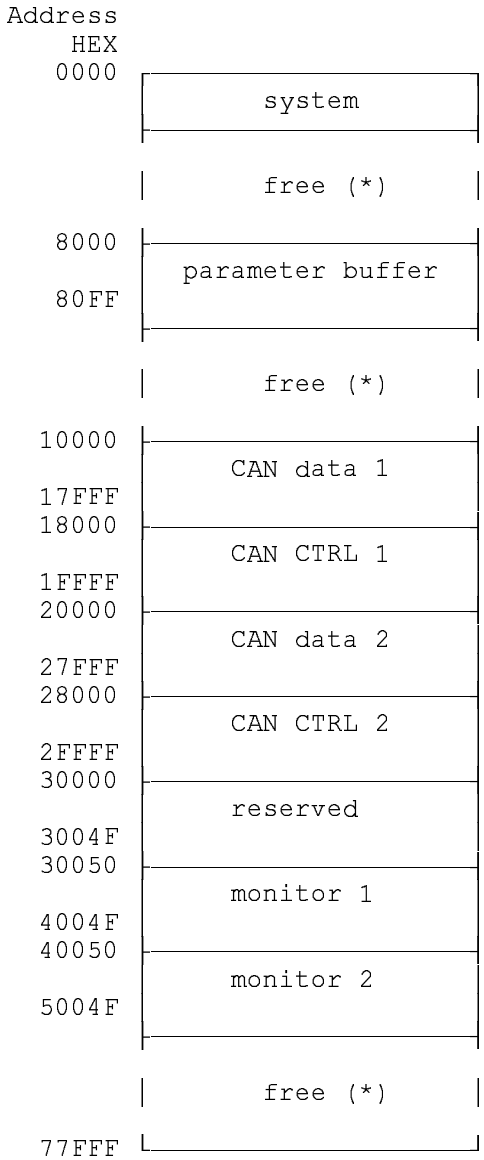
The CAN2 is an intelligent interface board for the VMEbus, which locally manages 2 CAN channels (also called nets).
The management of the CAN channels is taken over by the firmware stored in the local EPROM. It controls the CAN controllers 82C200 or 82527, resp., and manages the battery-backed local SRAM.

The firmware of the CAN2 is designed for operation of the board via the VMEbus! A stand-alone operation and control by the local CPU have not been designed.

The user selects and controls the required operating modes of the CAN identifiers by setting and reading cells of the memory organized in linear. The data to be transmitted are stored in this memory and received data are read here.

1.2 Memory Structure

The memory is divided into different parts by which data, parameters or commands are transmitted.



(*) In the memory parts marked as 'free', the local software installs the Rx-buffers or the CAN-buffers if necessary.

The interface to the CAN consists of two structure fields by which it is possible to receive or transmit data ('**CAN_Data 1**', '**CAN_Data 2**'). The data of a CAN-identifier (CAN-Id.) are stored together with specific parameters in so-called 'structural components'.

In the '**Parameter Buffers**' parameters and commands for the data transfer are transmitted (bitrate, etc.).

A control field ('**CAN_CRTL 1**', '**CAN_CTRL 2**') is assigned to each CAN_data structure, which in first place serves the firmware to store the actual parameters. The user selects the operating mode of each CAN-identifier in the structural components of the CRTL fields.

The '**Monitor Buffers**' serve the sequential storing of received data. Unlike the structure fields CAN data 1 and CAN data 2, in which the messages (in Rx-mode) are always overwritten again, the monitor buffers store the messages in a field in chronological order of their reception.

1.3 Possible Transfer Modes of the Identifiers

One of the following six transfer modes is possible for each identifier:

1. Tx-only
   Data entered into a structural component can only be transmitted.

2. Rx-only
   In this structural component data can only be received.

3. Tx-Auto-Remote
   The data of this structural component are transmitted automatically after the reception of a 'remote request' frame.

4. Monitor
   Received data are not stored in the according structural component, but sequentially with 'time stamp' in the monitor buffer.

5. C-Net (bridge function)
   With this transmission mode a bi-directional interconnection between any two structural components is created, i.e., it is possible, e.g., to link two physically separated CAN nets via the CAN2.

6. CAN-serial
   Received data are not only stored in the structural component, but also in the FIFO, to make it possible for the VMEbus master to process received data by an interrupt cycle in their chronological order.

1.4 Introduction into the Mode of Operation of the Firmware

Below simple operating modes of the Rx-(CAN2 receives data from CAN) and the Tx-transfer (CAN2 transmits messages to the CAN) will be explained for a better understanding of the mode of operation of the firmware. Following this, the possible replies from the CAN2 to the VMEbus master by the FIFO will be explained in short.

The complete explanation of the terms used below can be found in the following descriptions of the memory structures and buffers. Normally, the user will only operate with the data and control structures. The monitor structure and the parameter buffer are needed for continuous function procedures.

An overview of the possible interrupt sources and their processing concludes this manual.

1.4.1 Rx-Transfers

1.4.1.1 Procedure Description

Here, the most simple mode of an Rx-transfer is described: Data are received via CAN and stored in the local memory (without 'time out').

Initiator	Hardware	Hardware	Firmware	Firmware
Cause	Transmission of Tx frames by other CAN participants	Controller receives frame with admissible identifier and triggers Rx IRQ	Rx interrupt	Reception of a valid Rx structural component
Effect	CAN controller receives all transmitted frames and filters frames with the required identifier	Transmitting the received information to the Rx software filter	Selecting the received structures according to the Rx software filter	Actualization of the structural component in the CAN_data structure field

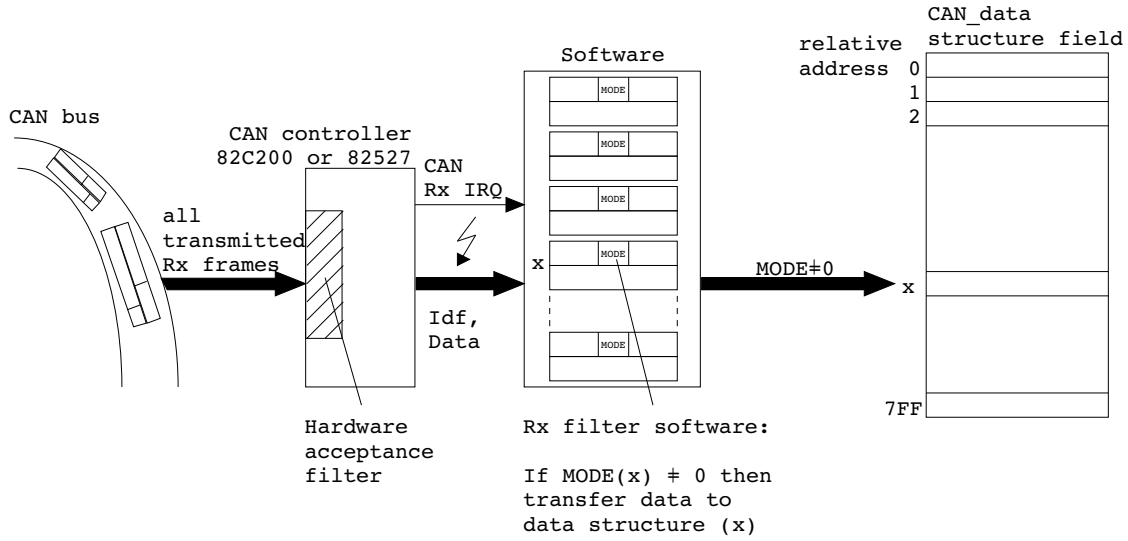


Fig. 1.4.1: Procedure of an Rx-transfer

Description of fig. 1.4.1:

The CAN-controller 82C200 generally receives all messages transmitted on the CAN. If it received valid data, it triggers an interrupt and transmits the data to the local firmware.

The controller has got a hardware filter (acceptance filter) which lets only pass selected identifiers. Because the pass conditions of this filter do not allow to select any identifier combinations, only a pre-selection of the identifiers is made here and a software filter is switched afterwards.

The software filter of the firmware selects the received messages according to the condition '..let pass all received identifiers for whose affiliated structural component an Rx-mode had been selected in the local memory' (MODE P 0).

The messages which comply with all set conditions are stored in the local SRAM ('CAN_data...') under the address which (seen relativley) corresponds to the transmitted CAN-identifier no. Doing this, only the number of data bytes of the corresponding identifier is overwritten which is stated in the message ('LENGTH').

Connections not shown in fig. 1.4.1:

The actual condition of the transmission can be read by the user ('STATUS') in the data-structural field ('CAN_data...').

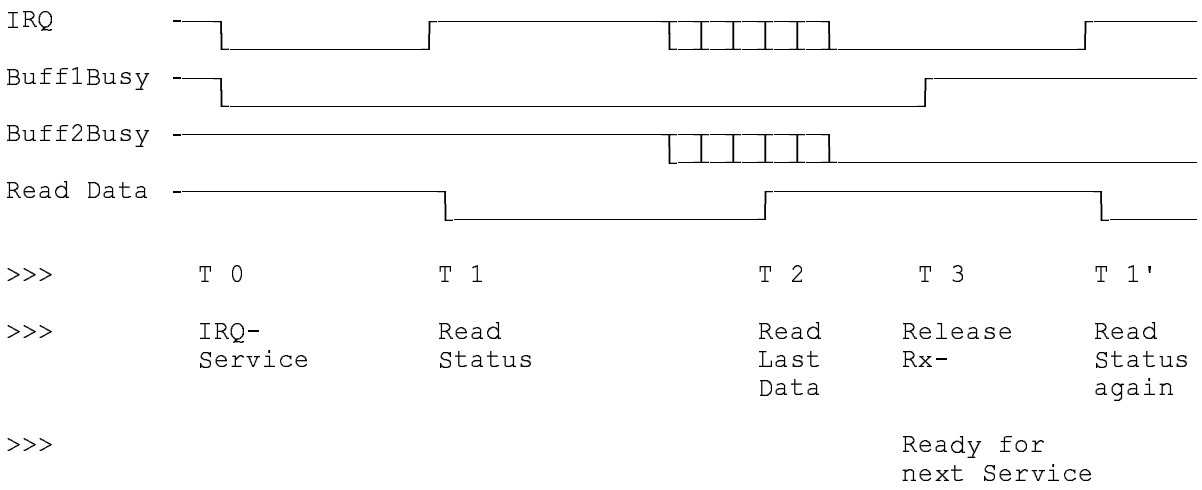
The data stored in a structural component are always overwritten (made topical) when new data arrive on this identifier. To inform the user about the reception of messages on this identifier, it is possible to trigger an interrupt ('EVTRIG') after finishing the Rx-transfer. The user has to define an interrupt routine and assign it to the interrupt.

It is also possible to trigger an interrupt if a transmission error occurred ('STATUS').

The notes given in parentheses ('_') refer to the memory structures or cells which will be described in following chapters.

1.4.1.2 Length of a CAN-Rx-Transfer/Time Chart

The following time chart shows the procedure of an Rx-transfer, starting with the Rx-interrupt (IRQ) of the CAN-controller 82C200/82527. From the shown parameters results the maximum required time which the VME-CAN2 needs to store the received data in its local memory (structural component).



Procedure Description:

<u>Time T 0:</u>
The received data have passed the 'acceptance filter' of the CAN-controller - the controller receives the data. The controller 82C200/82527 has got two data buffers in which it can buffer received data. It es the data in one of its buffers and triggers its Rx-interrupt, which starts a local interrupt routine. The alternative buffer is then still available for the loss-free reception of the next message.

<u>Time T 1:</u>
The interrupt routine reads the status of the controller, recognizes that data have been received and starts reading out the buffer. The time which is needed to read out the buffer depends mainly on the number of received data bytes (LENGTH).

<u>Time T 2:</u>
The last byte of the (first) buffer is read.

<u>Time T 3:</u>
The reading cycle is finished, buffer 1 is released again. At this time an interrupt can already be active again if buffer 2 was free and data have been received.

Time T_1':
The status of the controller is requested by the firmware again and the data are read out of buffer 2.

From this connections following times for reading cycles result:

Full READ-IRQ-Service time: T_0/3 (Worst Case)
(Processing the received Rx-frames occurs one after the other, because each next frame only arrives after complete processing of the one received before.)

Follow-Up-READ-IRQ-Service time: T_1/3' (Overlapping IRQs)
(Processing the received Rx-frames occurs 'parallel', because each next frame arrives during processing the one received before.)

DataLength	T_1 [us]	T_2 [us]	T_3 [us]	T_1' [us]	T_0/3 [us]	T_1/3' [us]
8	16	18	2	10	36,00	30,00
7	16	17,5	3,5	10	37,00	31,00
6	16	17,5	2,5	10	36,00	30,00
5	16	17	2	10	35,00	29,00
4	16	16	2	10	34,00	28,00
3	16	16	2,5	10	34,50	28,50
2	16	15	2	10	33,00	27,00
1	16	14,5	2	10	32,50	26,50
0	16	-	16	10	32,00	26,00
'*disabled*'	16	-	-	8,5	16,00	8,50

'*disabled*'....This CAN-Id. has been accepted by the hardware-acceptance ter, but not by the software ter of the firmware.

Table 1.4.1: Need of time to process the Rx-frames in dependency on the data length

T_0/3 is the complete processing time for interrupts on IDLE- or USER-task level (or any each other interrupt routine with a level which is smaller than IRQ6), to the reading of the last data byte and the release of the Rx-buffer for the next transfer.

T1/3' is the processing time for a complete Rx-reading cycle with overlapping interrupts of one or both CAN-controllers.

1.4.2 Tx-Transfers

In the following figure the transmission of data onto the CAN in two steps is described:

In the first step, the user writes the data to be transmitted ('Data1', 'Data2', etc.) from the VMEbus into the structural component of the local memory (CAN_Data...), which is assigned to the desired identifier.

Initiator	User	User	Hardware	Firmware	Hardware
Cause	Transfer of data to the CAN_data buffer	Setting of the parameter 'LENGTH'	FIFO not empty	FIFO interrupt	CAN Tx free IRQ
Effect	Actualization of the data in the buffer	Writing 'LENGTH' and transmitting the CAN identifier into the FIFO	FIFO interrupt	Sorting the structural components for priority, chaining the structural components by 'NEXT_POINTER'	Transmitting the Tx frame according to the priority to other Tx frames

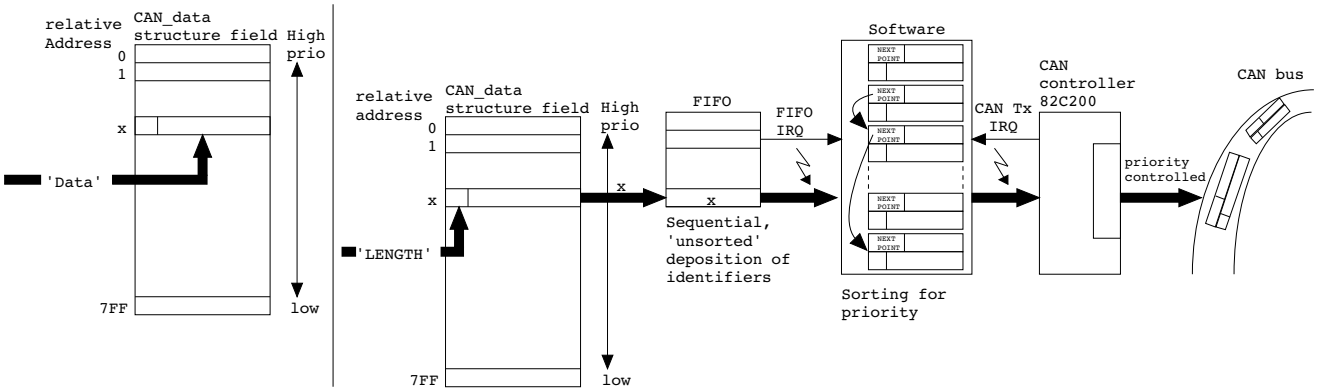


Fig. 1.4.2: Procedure of a Tx-transfer

In the second step the transmission, e.g., by entering the number of bytes to be transmitted ('LENGTH') is started with negative sign.

The FIFO is the buffer zone between the VMEbus and the CAN: In it the pointers are stored sequentially onto the according structural component after triggering the data transfer. The FIFO triggers an interrupt as soon as it is not empty anymore. This causes the local software to take up the data-structural component together with the CTRL-structural component (of the same identifier) into a priority chain and connect the structural components amongst each other by pointers.

A transmission priority is assigned to each element by its identifier (here: x), and therefore to its relative address. The identifier with the lowest address has got the highest priority, the one with the highest address has got the lowest priority.

While the firmware orders the structural components to be transmitted, it waits for the Tx-interrupt of the controller 82C200. This interrupt shows that the controller has finished its last transmission order and is now ready to process another order.

After the interrupt, the message with the highest local transmission priority is loaded into the controller. The controller now tries to transmit the message. The allocation of the transmission priority on the CAN occurs - like the local priority allocation - after the identifier.

Connections not shown in fig. 1.4.2:

The actual condition of the transmission can be read in the data-structural field ('CAN_Data') by the user ('STATUS').

The successful completion of the Tx-transfer or arisen transmission errors are shown in the data-structural component of the identifier ('STATUS') and can result into an entry into the FIFO 'Data to VMEbus' ('EVTRIG'). The FIFO can be read after checking the FIFO status. If a VMEbus interrupt is released, the entry into the FIFO triggers an interrupt. This user-defined interrupt ,must be followed by an interrupt routine to process the interrupt.

If the user should start so many transmission orders from the VMEbus that the FIFO is completely full, an interrupt on the VMEbus will be created. This interrupt tells the user that he has to reduce the data transfer to prevent loss of data ('iovec').

The notes given in parentheses ('_') refer to memory structures or cells which will be described in following chapters.

1.4.3 Status Messages of the CAN2 and Status Request by the VMEbus Master

As already mentioned, it is necessary for various transfer conditions, to give messages from the CAN2 to the VMEbus master. This can be, e.g., status messages about the procedure of the transfer.

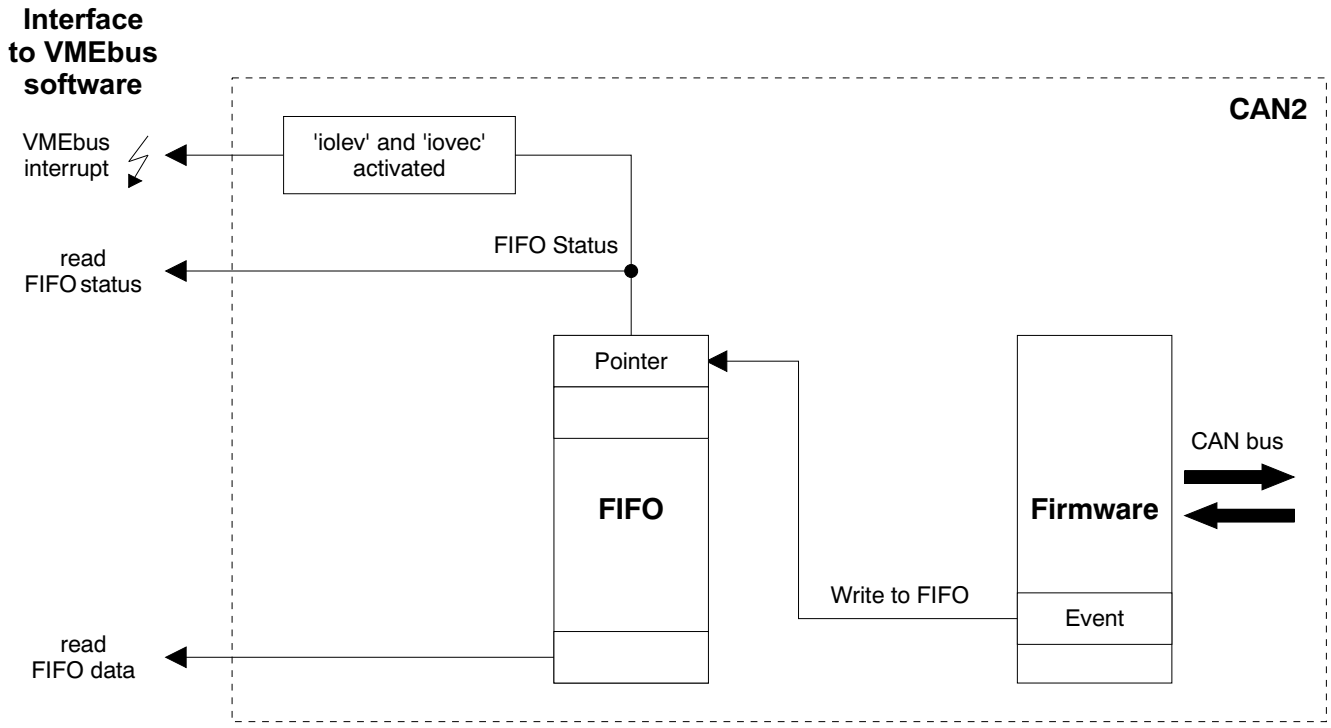


Fig. 1.4.3: Function of the FIFO 'Data to VMEbus'

If something occurs on the CAN, as, e.g., the end of a transfer, it is possible for the local firmware to write a pointer into a FIFO, provided the according parametration exists. This pointer has got the affected identifier and the number of the affected net.

It is possible for the VMEbus master to supervize the status of this FIFO by polling and read out the received pointers.
Furthermore it is possible to let the CAN2 trigger an interrupt if the FIFO is not empty anymore.

The polling and the interrupt handling of the CAN2 will be described in detail in the chapter 'Summary of the Interrupt Options'.

2. Modes of the Data Transmission

This chapter describes the data and control fields of the two CAN-channels. Furthermore the general modes of the data transmission are described with explanations about the mode of operation.

2.1 The Structural Fields CAN_Data 1 and CAN_Data 2

2.1.1 Function and Structure

By the structural fields CAN_Data 1 (net 1) and CAN_Data 2 (net 2) the data of the VMEbus are transmitted onto the CAN.

A structural field is divided into 2048 elements.

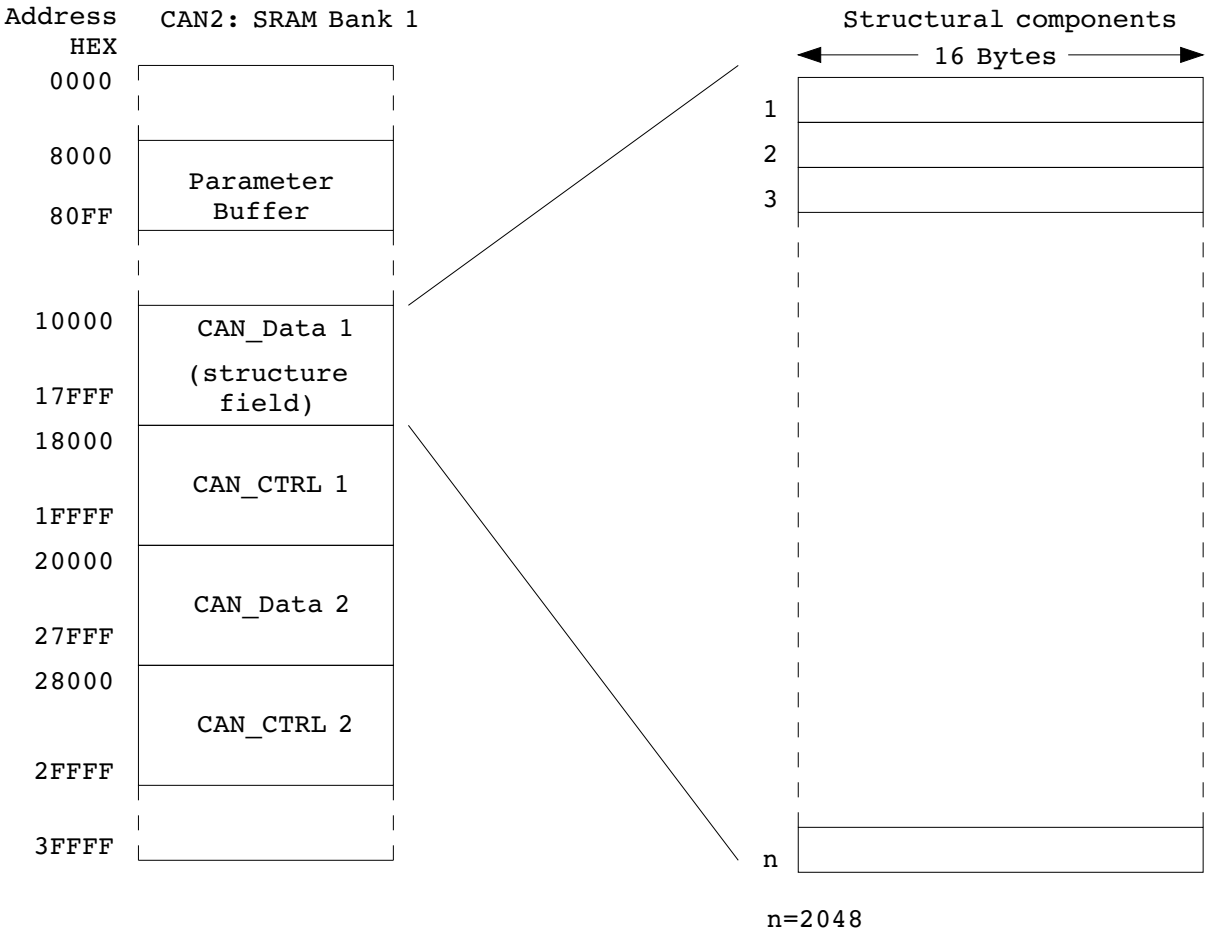


Fig. 2.1.1: Division of a CAN-Data_structural field (example: CAN_Data 1)

To each possible CAN-identifier of a CAN-channel, a structural component is assigned (max. 2048 Ids are possible).

A structural component is 16 bytes long and contains apart from the transmitted data (max 8 byte) 8 byte control and control parameters.

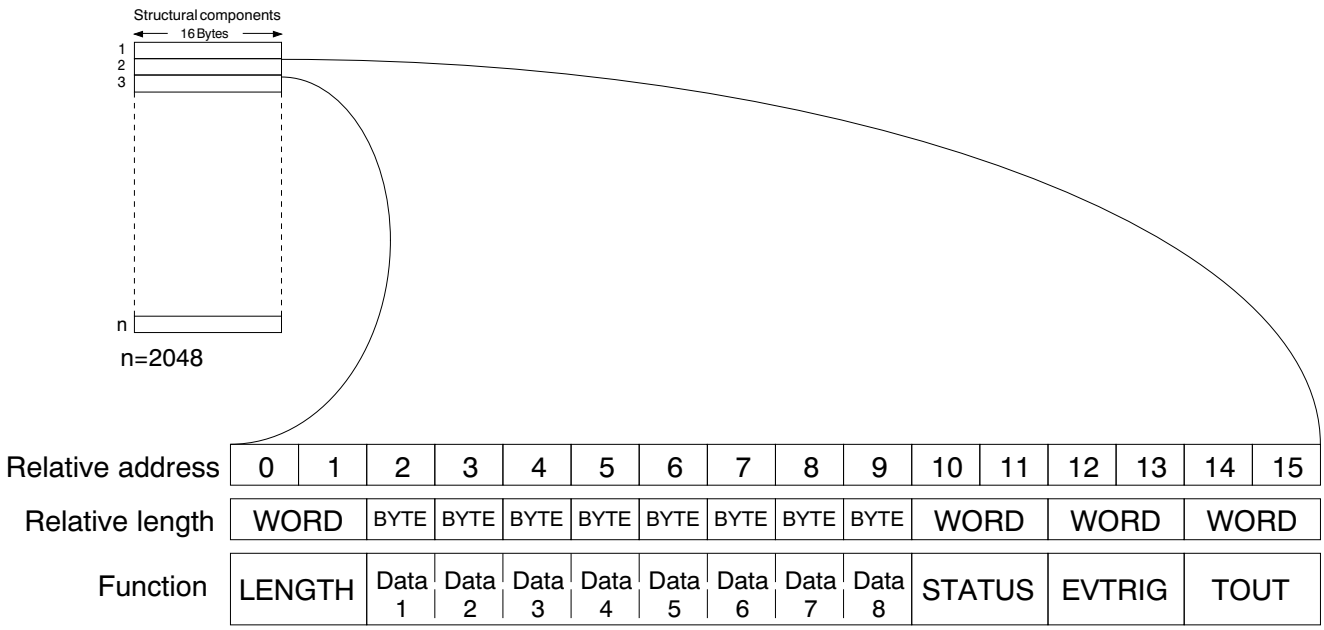


Fig. 2.1.2: Contents of a CAN-Data_structural component

2.1.2 The Bytes of a Data-Structural Component

Below the bytes of a data structural component are described in rising order.

LENGTH..... shows the number of valid data bytes and determines in connection with the cell 'MODE' in the according control-structural component (see chapter 'Parameter and Command Transfer by the Parameter Buffer') the operating mode, fixed for this identifier.

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 start of Tx-transfer)
FFFF...FFF7 (corresponds to negative value of the number of valid bytes and leads to the start of Tx-transfers)

Data1..Data8.... contain the transmitted data.
It is possible to transmit nil to eight bytes.

STATUS..... gives information about the actual condition of the transmission. Following messages have been implemented so far:

Value 'STATUS' (HEX)	Condition of transmission
0000	Transmission successfully completed
0001	Rx-time out (error)
0002	Tx-time out (error)
0004	Tx-error (error)
0005	Controller 'off bus' (error)
0006	Controller 'busy' (error)
0008	Access to non-existing net (error)
0010	CMS: Domain time out
0011	CMS: Wrong acknowledge at CMS-protocol
0012	CMS: Interrupt domain transfer
0020	CMS: Compare error of the CMS watchdog
0101	RTR-frame has been received
FFFE	Wait for Rx-at 'remote request'
FFFF	Transmission not yet completed

Table 2.1.1: Function of the STATUS word

Description of the status messages:

Rx-time out..... (error)
Within the time entered in the cell 'TOUT', no data have been received in this structural component.

The activation of the time-out supervision occurs after the entry of the corresponding operating mode into the CTRL-structural component of the identifier (enter 'MODE' by parameter buffer) by triggering the cell 'LENGTH' of the CRTL-structural component. After the triggering the value entered in the cell 'TOUT' is entered into the 'TIME COUNTER' of the CTRL element, the element integrated into the 'Rx-time out chain' and the 'TIME COUNTER' is counted down.

After passing the time-out time, this error stauts is always set - the FIFO 'Data to VME' can be loaded onto the structural component by a pointer and an interrupt can be triggered (see chapter 'Interrupts').

Tx-time out..... (error)
The structural component could not be transmitted within the time entered in the cell 'TOUT'. The transmission process is interrupted.

The activation of the time-out supervision occurs after the entry of the corresponding operating mode into the CTRL-structural component of the identifier (enter 'MODE' by parameter buffer) by triggering the cell 'LENGTH' of the CTRL-structural component. After the triggering the value entered in the 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 counted down.

After passing the time-out time this error stauts is always set - the FIFO 'Data to VME' can be loaded onto the structural component by the pointer and an interrupt can be triggered (see chapter 'Interrupts').

Tx-error..... (error)
This error arises, if the CAN-controller 82C200 requests new data for transmission by its Tx-interrupt from the firmware, although it had not set its 'Tx-complete' signal so far. The error message can lead to an entry into the FIFO 'Data to VME' which in turn can trigger a VMEbus interrupt.

Controller 'off bus'... (error)
If gross errors arise at transmission attempts of the CAN-controller 82C200 (e.g. bitrate controller ≠ bitrate bus), the controller 'leaves' the bus, because further transmission attempts would be futile.

If this error state arises during running Tx-transmissions, this error message is not only entered into the actual structural component, but also into all other Tx-structures, already taken over by the local priority chain. If the error arises during the first transmission of a Tx-element, the message is only entered into this structural component, this transmission interrupted and the transmission of the next Tx-structural component is started. The error message can lead to an entry into the FIFO 'Data to VME' which in turn can trigger a VMEbus interrupt.

Controller 'busy'..... (error)
This error state arises if the local Tx-priority chain is empty, a Tx-instruction is transmitted by the VMEbus and the firmware gets the message 'busy' at the usual request of the controller about its Tx-status, although the controller should not be busy at that time (-> fatal error!).

The error message can lead to an entry into the FIFO 'Data to VME', if the cell 'EVTRIG' is set. The entry in turn can trigger a VMEbus interrupt.

Access to not..... available net (error)

A not available net has been responded. This error occurs, if, e.g., net 7 is ccessed, although only net 2 and net 3 are available. This error message is important for the CAN-master driver on the VMEbus.
The error messages leads to an entry into the FIFO 'Data to VME', if the cell 'EVTRIG' is set. The entry in turn can trigger a VMEbus interrupt.

The following CMS-error messages always lead to an entry into the FIFO 'Data to VME'. The entry can trigger a VMEbus interrupt:

Domain-time out...

A time-out error occurred during the CMS transfer.

Wrong acknowledge at CMS protocoll...

The CAN2 received a wrong coded acknowledge frame (e.g. wrong command specifier) during the CMS transfer.

Interrupt domain transfer...

The transfer has been interrupted, because the CMS partner transmitted an interrupt domain transfer.

Compare error...

The CMS watchdog discovered an error at the comparison of the desired value to the set point of the toggle bit or the state.

RTR-frame has been.. received

For this structural component an RTR-frame has been received, which prompts the transmission of the data contained in this element. The status message only remains active as long as no transmission occurred.

Wait for Rx-at.... 'remote request'

On this identifier a 'remote request' has been transmitted. The firmware now waits for the arrival of a message for this structural component. The status is reset when a message has been received.

Transmission... not yet finished

The transfer of this structural component is not processed, yet. After finishing the transfer, the status is reset and the structural component is removed from the local priority chain.

EVTRIG.....

determines if an entry into the FIFO 'Data to VME' should occur. This entry can trigger a user-VME interrupt. EVTRIG is only a release condition for the triggering of an interrupt. Additionally the cells 'iolev' and 'iovec' of the command 'CARD-interrupt enable' have to be set in the parameter buffer.

If these release conditions are fullfilled, at the occurrance of the interrupt conditions described below, an interrupt is triggered on the VMEbus with the level of 'iolev' and the vector fixed by 'iovec'.

Value 'EVTRIG' (HEX)	Function
0000	no FIFO entry after cut-in of an at end condition
0001 : 00FF	FIFO entry and possibly triggering of a user IRQ after cut-in of an at end condition

Table 2.1.2: Function of the EVTRIG bytes

Valid at end conditions are successful conditions or conditions ended by the occurrance of an error.

1. Transmission handled successfully
2. Rx-time out (error)
3. Tx-time out (")
4. Tx-rrror (")
5. Controller 'off bus' (")
6. Controller 'busy' (error)

Errors during CMS transfer always lead to the entry of a pointer into the FIFO, independent from the value of the cell EVTRIG!

TOUT..... shows at values bigger nil ($0001...$7FFF) the time-out value in msec (operation without CMS protocoll) or the watchdog-guard time (operation with CMS protocoll).

Operation without CMS protocoll
The activation of the time-out supervision occurs after the entry of the according operating mode into the CTRL-structural component of the identifier (enter 'MODE' by parameter buffer) by triggering the cell 'LENGTH' of the CTRL-structural component. After the triggering the value entered in the cell 'TOUT' is entered into the 'TIME COUNTER' of the CTRL element, the element integrated into a time-out chain and the 'TIME COUNTER' counted down.
The time-out option can be used for the Tx-and also for the Rx-transfers:
In the Tx-mode it triggers a Tx-time out error, if the transmission could not occur within the given time. The Tx-time out is shown in the cell 'STATUS' and can trigger an interrupt. Furthermore in connection with the parameter buffer it is possible, to start Tx-transfers of up to 64 different structural components in periodical time intervals.
In the Rx-mode an Rx-time out is shown in the cell 'STATUS', if no messages had been received on this identifier within the time-out time. The 'TIME COUNTER' is reset again with every reception of a message.

If the command 'periodical Tx-transfer' is set with the corresponding parameters 'net' and 'Idf' in the parameter buffer, the negative values are used by TOUT, to start Tx-transfers in periodical intervals. For the start of the transmissions it is necessary to start a Tx-transfer once sucessfully apart from setting the cells cited above.
The periodical Tx-transfers are finished by resetting the command in the parameter buffer or setting the time-out time to positive values.

> If the above cited cells of the parameter buffer are not set, the time out will be 'disabled' with negative TOUT values. This should be absolutely avoided, because it can happen that the system hangs itself up, if it is not possible to process the Tx-order!

> In all operating modes 'TOUT' should never be set to '0' and never to values smaller than 5 msec, because the resolution of the counter is in this range!

'TOUT' (HEX)	Function
0000 bis 7FFF	depending from 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
8000 bis FFFF	repetition time for 'periodical Tx-transfer' FFFB => 5 msec 8000 =>32,768 sec

Table 2.1.3: Function of the 'TOUT bytes'

Operation with CMS protocol
If the CAN2 has got the CMS implementation, the watchdog-guard time of the CMS watchdog is transmitted with positive values by TOUT. The CMS watchdog will be described in detail in its own chapter.

2.2 The Control Structures CAN_CTRL 1 and CAN_CTRL 2

2.2.1 Function and Structure

The control structures serves the firmware to assign transmission parameters, file status informations and variables and document the order of the active data-structural components.

A CAN-CTRL structural component is assigned to each structural component of both CAN_Data structures. The structural components of the control structures are ed, like the data structures, on relative addresses of the RAM bank1, which are identical to the according CAN-identifier numbers. For both channels are each 2048 controll-structural components available, therefore. A controll-structural component is 16 byte long.

> The direct setting of the parameters of the control structures is largely reserved to the firmware. **The user is only allowed to set the cell 'XTTID' directly!**

The structural components are structured in the following way:

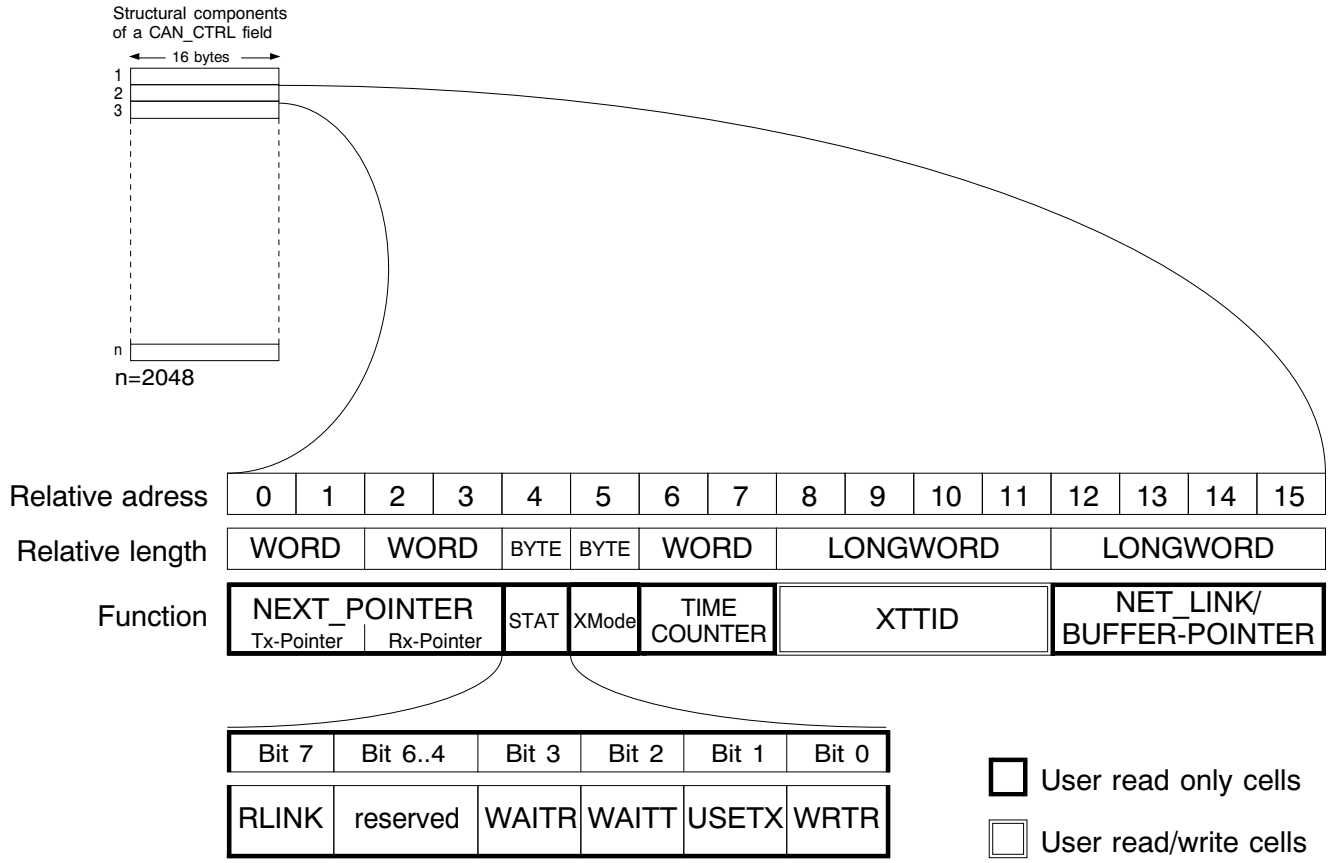


Fig. 2.2.1: Contents of a CAN_CTRL-structural component

2.2.2 The Bytes of a CTRL-Structural Component

Below the bytes of a CTRL-structural component will be described in rising order. Apart from tthe 'XTTID' the cells must only be read by the user.

NEXT_POINTER..... This pointer has got the absolute addresses of CAN_CTRL-structural components.
In byte 1 and 2 the address of the next Tx-structural components (Tx-pointer) is contained. Byte 3 and 4 contain the address of the next Rx-structural component (Rx-pointer):

Tx-pointer
The pointer contains the address of the next active CTRL-structure elemnt with low priority (= bigger identifier == bigger address.
This linkage serves the local order of the structural components with their transmission priority. If the CAN-controller component 82C200 has processed its last transmission order, it requests the next CTRL-structural component with the highest priority, links it with the according data-structural component to a CAN-frame and transmits the message onto the CAN-as soon as the highest priority has been entitled to this frame on the bus.
The last structural component of the chain has got the entry '$FFFF' in the Tx-pointer.

Rx-pointer
The Rx-pointer contains the address of the next active Rx-structural component for which the 'time-out' mode had been selected.

This linkage serves the continuous counting down of the 'time counters' (see below) of the active Rx-elements in the oerder of the activation of the time-out mode. The firmware counts the counters of the elements each one level further.
The last structural component of this time-out chain has got the entry '$FFFF' in the Rx-pointer.

STAT..... The cell 'STAT' divides itself into 8 bit with individual functions. All bits of this cell must only be read by the user:

RLINK (bit 7)
Rx-structure is integrated into Rx-time-out link.

Bit 6...4
These bits are reserved for future applications.

WAITR (bit 3, active '1')
This bit shows that the structural component is in the status 'waiting for Rx-end'. The bit is active as long as the structural component is in Rx-request mode and no valid at end condition occurred. At and conditions are the arrival of the data or an Rx-time out.

WAITT (bit 2, active '1')
This bit shows that the structural component is in the status 'waiting for Tx-end'. The bit is active as long as the Tx-transfer is not finished, and is reset after finishing this order.

USETx-(bit 1, active '1')
This bit is only used by the local firmware.

WRTR-(bit 0, active '1')
This bit shows that the structural component is in the status 'wait for answer of RTR'. On this identifier a 'remote request' has been transmitted and the irmware waits for the reception of messages. After the arrival of a message in this structural field the bit is reset.

XMode..... In the cell 'XMode' the transmission mode is entered by the firmware after it had been set by the user by the parameter buffer (see also chapter 'Parameter and Command Transfer by the Parameter Buffer').

TIME COUNTER..... In these two bytes the actual value of the time-out counter is ed.

If the time-out mode is selected for a structural component in the cell 'MODE', a positive value entered into the cell 'TOUT' and the selected operating mode (Tx-mode, Rx-mode, etc.) is activated, the contents of the cell 'TOUT' are taken over into the TIME COUNTER and this structural component is integrated into a time-out chain. Here the TIME COUNTER of the components are counted down continuously in the order of their chronological entry into the chain.

If the TIME COUNTER has run out before the transmission is finished, in the cell 'STATUS' in the data-structural component the according time-out-error message is set.
If the transmission is finished before the COUNTER ran out, the structural component is removed from the time-out chain.

XTTID..... This cell is not covered by the local firmware. Here the user is able to store own, free selectable parameters onto four bytes. A possible use of this cell is, e.g., the storing of the program counter of a user program into the cell.

In the modes 'Rx-transfers disabled (Tx-only)', 'Rx-transfer with time-out option' and 'transmit RTR' it is possible to use the cell 'XTTID' instead of the cell 'EVTRIG' as necessary condition for the writing into the FIFO 'Data to VME' and therefore use it as interrupt condition: If the cells 'iolev' and 'iovec' of the command 'CARD-interrupt enable' are set in the parameter buffer and if the cell 'XTTID'≠ 0, then a VMEbus interrupt is triggered at the arrival of a valid at end condition.

NET_LINK/
BUFFER-POINTER... At the interconnection of two CAN-IDs by the parameter command 'LINK net...', the local firmware enters the absolute address of the respective data-structural component with which it is connected into four bytes in this cell (see also chapter 'Commands of the Parameter Buffer').
If the identifier of this control-structural component is used for the CMS transfer, it is possible to read the initial address of the CMS buffer in this cell after the buffer initialisation by 'Init domain' (see also command transfer). By this initial address it is possible to calculate the addresses of the relevant buffer cells.

2.3 Rx-Modes

2.3.1 Receiving Rx-Data without Interrupt

Apart from the 'Rx-disable mode' of the selected identifiers, the CAN2 is able to receive the Rx-data transmitted on the CAN in all operating modes.

All received data are stored in the data-structural field and can be read there by the VMEbus. The data-structural field is permanently updated with the received data. The valid data length in bytes can be read in the cell LENGTH of the data-structural component after the reception of the data. Before, during and after the transmission, it is possible to read the actual status of the tansmission in the cell STATUS.

If the data of individual or svereal identifiers should not be ed, this can be achieved by selecting the Rx-disable mode (MODE = 00) for these identifiers. The mode is set by the parameter buffer.

The structure of the components of the data-structural field has already been described in the preceding chapter. The setting of commands of the parameter buffer will be described in the chapter below 'Parameter Buffer'.

2.3.2 Receiving Rx-Data with Interrupt Message on the VMEbus

For the reception of Rx-data with a following interrupt message the same conditions apply as in the preceding chapter.

Additionally following has to be noticed:

1. Generally the cells 'iolev' (**I**/**O**-Interrupt **Lev**el) and 'iovec' (**I**/**O**-Interrupt **Vec**tor) have to be set by the command 'CARD-interrupt enable' in the parameter buffer, to be able to trigger aVMEbus interrupt. You initialize the hardware and fix the VMEbus-interrupt level and the vector. The combination of this cells and the setting of the cells in the parameter buffer will be explained in the chapter below 'Commands of the Parameter Bufferers' in more detail.
2. In the CAN-data-structural component of the desired identifier the cell 'EVTRIG' has to be set. Alternatively it is also possible to set the 'XTTID' in the control-structural component to a value ≠ 0. The cell 'XTTID' can be set as the user likes (apart from '0').
   The assignment of the cells 'EVTRIG' and 'XTTID' has already been described in the precedeing chapters.

By these pre-settings the desired VMEbus interrupt is triggered, if a valid at end condition for the reception occurs. Valid at end conditions are, e.g., 'data successfully received' or 'data have not been received within the Time-out time'. A list of at end conditions can be taken from the description of the cell 'EVTRIG'.

2.3.3 Further Options for the Reception of Rx-Data

If a certain message should arrive on an Rx-identifier within a limited time period, it is possible to start a time-out counter which can trigger an VME interrupt after running out. To select this mode the time-out time (TOUT) has to be entered in the data-structural component of the identifier. The counter is started by setting the cell LENGTH. The entry of the number of data bytes in the lowest-order HEX place of LENGTH is not evaluated by the firmware, here. (The cell MODE in the parameter buffer has 5to be set to '01', like in the preceding Rx-modes.) After the reception of the data the cell LENGTH is assigned with the length of the valid data by the firmware.

2.4 Tx-Modes

2.4.1 Transmitting Tx-Data without Interrupt

For the transmission of Tx-data, in the parameter buffer the mode '00' should be selected for the desired identifier(s).

The data to be tansmitted are entered into the data-structural component of the identifier by the VMEbus. By setting the valid data length into the cell LENGTH, the transmission is started. The values entered into LENGTH have got following meaning, here:

1. Entry of the length as positive number ($0000...$0008):
   The entry of the number of bytes as positive number does not lead to the start of processing the transmission order.
2. Entry of the length as negative number ($FFFF...$FFF7):
   If the number of bytes to be transmitted is entered as negative HEX number, processing the transmission order is started with the entry.
3. Entry of length and the number '6' ($0060...$0068):
   Function identical to 2.

The actual status of the transmission can be read in the cell STATUS of the data-structural component.

2.4.2 Transmitting Tx-Data with Interrupt Message on the VMEbus

For the transmission of Tx-data with a following interrupt message the same conditions are valid as in the preceding chapter.

Additionally the following has to be noticed:

1. Generally, to be able to trigger a VMEbus interrupt, the cells 'iolev' (**I**/**O**-interrupt **lev**el) and 'iovec' (**I**/**O**-interrupt **vec**tor) have to be set in the parameter buffer by the command 'CARD-interrupt enable'.They initialize the hardware and fix the VMEbus-interrupt level and the vector. The combination of this cells and the setting of of the cells in the parameter buffer will be explained in the chapter 'Commands of the Parameter Buffer' in more detail.

2. In the CAN-data-structural component of the desired identifier the cell 'EVTRIG' has to be set. Alternatively it is possible to set the 'XTTID' in the monitoring-structural component to a value ≠ 0. The cell 'XTTID' can be freely set by the user (apart from '0').
   The assignment of the cells 'EVTRIG' and 'XTTID' has already been described in the preceding chapters.

With these preadjustments the desired VMEbus interrupt is triggerd, if a valid at and condition for the transmission order occurs. Valid at and conditions are, e.g., 'data have been transmitted successfully' or 'data could not be transmitted on the bus within the time-out period' (bus error). A list of at end conditions can be taken from the description of the cell 'EVTRIG'.

2.4.3 Transmitting RTR-Frames

To request particular Rx-data, it is possible to transmit an RTR-frame onto the CAN. The cell MODE in the parameter buffer has to be set to '1' for this. The RTR-will be transmitted when the cell LENGTH in the data-structural component of the desired identifier is set accordingly (see description of the cell LENGTH). The last nibble of LENGTH contains the desired data length of the requested message. At this transfer, too, it is possible to trigger a VMEbus interrupt. Valid at ane conditions are, e.g., 'error at RTR-frame transmission' or 'reception of required Rx-data'.

2.4.4 Further Options for the Transmission of Tx-Data

It is possible to supervize the transmission by a time-out monitor. If the transmission does not occur within the determined time, a time-out error is shown in the cell STATUS. The time-out period has to be set in the data-structural component of the desired identifier (see there).

The CAN2 is also able to initiate a transmission if an RTR-frame had been received on an identifier. For this MODE in the parameter buffer has to be set to '02' for the corresponding identifier. At the reception of an RTR-frame, the data-structural component (0...8 byte) is transmitted. Principally the transmission runs like a 'Tx-transfer' in MODE '00'. The same at end conditions apply.

Differences arise at the interrupt creation and the source of data which should be transmitted:

If the cell 'EVTRIG'is set ≠'0', a VMEbus interrupt is triggered after the arrival of the RTR-frame and the data to be transmitted have to be supplied by the VMEbus.

If the cell 'EVTRIG'is set ='0', the data to be transmitted are transferred from the local data-structural component of the CAN2.

3. Special Functions

3.1 The Monitor Buffers

3.1.1 Function and Structure

In the monitor buffers received data can be stored sequentially to record the chronological process and the contents of the transmissions over a longer period.

The selection of the identifier which should be recorded is made in the CTRL-structural field of the identifier by setting the 'monitor mode'.

The recording starts with the first arrival of the specified trigger condition. After this triggering all following messages are recorded and the time of their arrival is stored relative to the trigger time. The trigger condition is transferred by a command of the parameter buffer, which will be explained in more detail later on.

The recordings stop when the monitor buffer is filled.

The following figure shows the principal process of a recording.

Initiator	Hardware	Hardware	Firmware	Firmware
Cause	Transmitting Tx frames of other CAN participants	Controller receives frame with admissible identifier and triggers Rx IRQ	Rx interrupt	Reception of a monitor structural component
Effect	CAN controller receives all transmitted frames and filters frames with the required identifier	Transmitting the received information to Rx software filter	Selecting the received structures according to the Rx software filter	Sequential storage of the messages in the monitor buffer

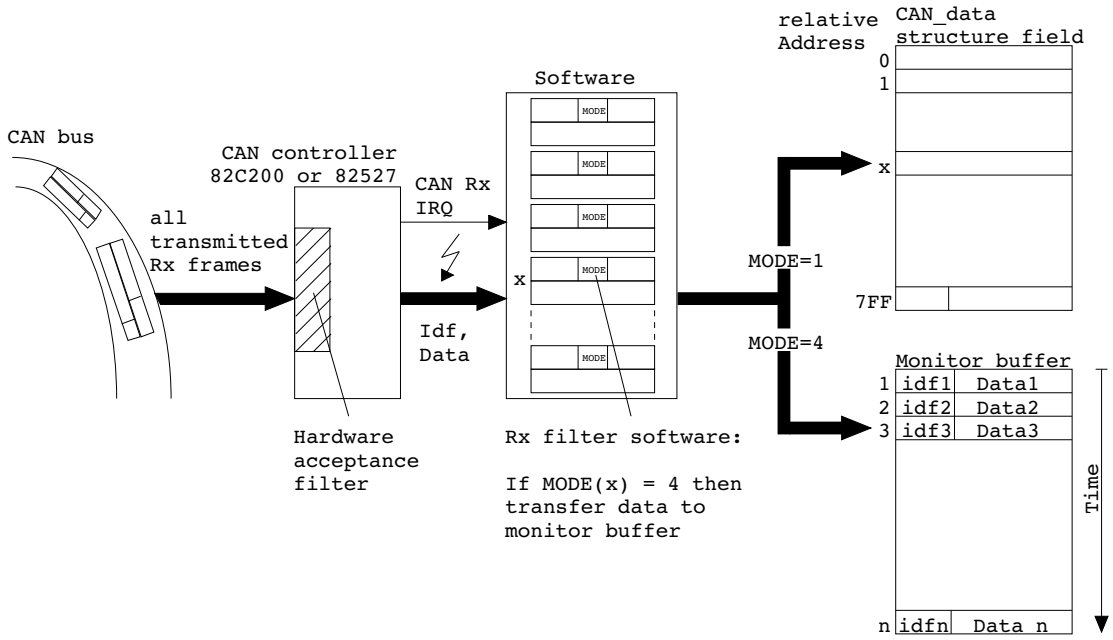


Fig. 3.1.1: Reception and storing of CAN messages in the monitor buffer

A monitor buffer is available to each channel (net) of the CAN2. The position and dimension of the monitor buffers is stored in the parameter buffer.

A structural component of the monitor buffers is always 16 byte long:

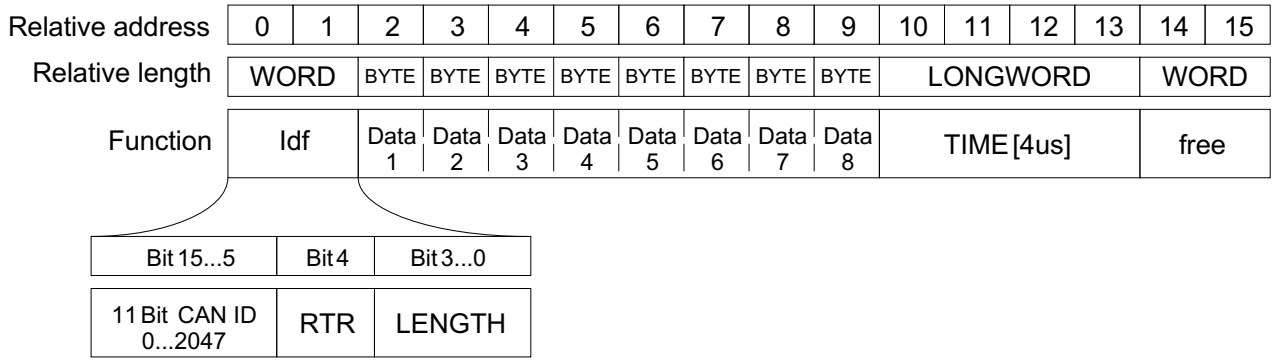


Fig. 3.1.2: Contents of a monitor-structural_component

3.1.2 Explanation of the Bytes of a Monitor-Structural Component

Below the bytes of a monitor-structural component will be described in rising order.

Idf.....contains the 11 bit long CAN-Id, the RTR-bit (remote transmission request) and the number of the transmitted bytes (LENGTH):

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

Data1..Data8....contain the transmitted data.
It is possible to transmit nil to eight byte.

TIME.....time (in [4µs]-steps) which has passed since the first arrival of a transmission in this monitor buffer until the arrival of this data (='time stamp').

3.2 CAN-Serial Mode

3.2.1 Function

In the CAN-serial mode received data are not only stored in the structural field, but also made available to the VMEbus master in the FIFO 'data to VMEbus'.

This data file gives the master the advantage thatit can process sequential received data of an identifier without the risk of data loss in the order of their arrival.

This transfer method is made possible by the use of the pointer FIFO (data to VMEbus) to store data and by triggering an interrupt at the arrival of the first byte of a message. Because already stored data are not permanently made topical like in the data-structural component, there is no risk that data will be overwritten unintentionally.

3.2.2 Structure of the CAN-Serial-Data Blocks

The CAN-serial-data mode is selected for the respective identifier (structural component) by the allocated byte 'MODE' in the monitor-structural field. The byte 'MODE' has to be set by the parameter buffer.

The local software marks CAN-serial-data blocks received in the FIFO for differentiation of the usually stored pointers by setting a mark. The mark contains already the information about the respective CAN net, on which the data have been received. The bytes following the mark contain status messages and the received data. If no further data follow, pointer or a new data block are again stored, beginning with 'mark', in the FIFO after the last data byte.

At the reception of a 'remote request (RTR), the parameter LENGTH is not evaluated by the firmware. No data are stored in the FIFO. But LENGTH can be assigned with any value to, e.g., transmit user-defined information.

The following figure shows as example the position of a message with the maximum length of 8 data byte in the FIFO, which is preceded and followed by pointers of 'normal' transfers.

Only so many bytes are written into the FIFO as data have been received. But only complete words can be stored in the FIFO, that is why one bytwe (D0...D7) always remains unused at the transmission of an odd number of bytes.

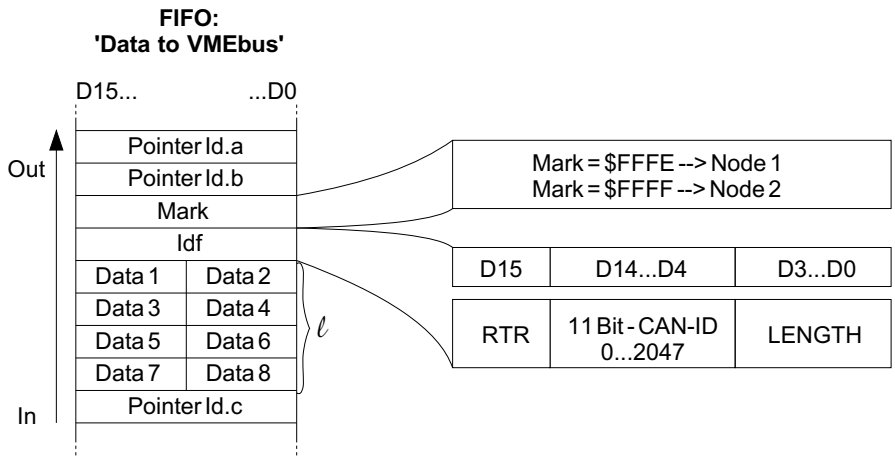


Fig. 3.2.1: Structure of a CAN-serial-data block

3.2.3 Explanation of the Bytes of a CAN-Serial-Data Block

```
CAN-Id..........possible values: 0...2047

RTR-bit.........shows whether a data transmission had been
                requested under this identifier.
                RTR-= '0' ....data transfer
                RTR-= '1' ....remote transmission request
                              (LENGTH will not be evaluated)

LENGTH..........shows the number of transmitted data bytes.
                Possible values: 0...8

Data1..Data8....contain the transmitted data bytes.
                It is possible to transmit nil to eight bytes.
```

3.2.4 CAN-Serial-Data Interrupts

Received CAN-serial-data blocks are shown to the VMEbus master by the CAN-server interrupt. To be able trigger a VME interrupt, the cells 'iolev' and 'iovec' must be set via the command 'CARD-Interrupt-Enable' in the parameter buffer.

In contrast to the other interrupt sources, which can be applied to the VMEbus via the CAN server interrupt, the CAN-serial interrupt is not masked with EVTRIG or XTTID.

The necessary interrupt requirement (apart from setting the cells 'iolev' and 'iovec' by the command 'card-interrupt enable' in the parameter buffer) is the selection of the transmission mode CAN-serial data for the desired identifier. If messages arrive on the identifier, they will be stored in the FIFO and the FIFO triggers the CAN-server interrupt.

The acknowledgement of the VMEbus interrupt will be explained in the chapter 'CAN-Server Interrupt'.

3.3 Rx-Buffers

3.3.1 Function

In the Rx-buffer the data of selected Rx-identifiers can be stored in the chronological order of their arrival.

If the commando 'get Rx-buffer' is called by the parameter channel, the Rx-buffer is arranged by the local firmware. By the command 'set mode' it is possible to select identifiers which should be recorded in this buffer. The commands of the parameter buffer will be explained in an own chapter in more detail.

The data of the identifiers selected for the Rx-buffer are not stored in the data-structural component.

The data memory in the Rx-buffer occurs in a similar way as in the modnitor buffer. Both operating modes differ in some essential points, though:

	Monitor mode	Rx-buffer
Buffer size	4096 messages (à max. 16 byte) (recording ends if buffer is filled)	programmable (max. 4096 messages) (buffer is overwritten always new: 'wrap around')
Trigger	on selected identifier	no trigger, recording starts immediately after selection of the mode
Time stamp	4 µs	1024 µs
VMEbus interrupt	no VME-IRG possible	VME-IRQ possible at the arrival of the first data

Table 3.3.1: Comparison monitor mode - Rx-buffer

3.3.2 Structure of the Rx-Buffer

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

The data area contains apart from the received data further parameters as, e.g., identifiers and net 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 by the command 'Rx-buffer'	read only	data area

Table 3.3.2: Division of the Rx-buffer

Rx-write pointer... This pointer points to the buffer cell (relative to 'BuffStart') which will be recorded at the next reception of an RxId.
Example for the mode of counting for the pointer:
$0000, $0010, $0020,... ..., $0FF0, $0000
After a RESET the pointer is set to '0'.

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

The contents of the Rx-write pointer and the Rx-read pointer are brought in as condition for the writing of the FIFO 'data to VME' and therefore also as condition for the triggering of a VMEbus interrupt: After the arrival of new data in condition 'Rx-read pointer = Rx-write pointer' the FIFO is set.

Rx-mask... This parameter contains the size of the data area of the Rx-buffer. Rx-mask is calculated from the value 'buffer length', which can be given by the user at the buffer initialisation in the parameter buffer (see also description of the parameter buffer): The given value is rounded up by the firmware to values of 2^n and entered into Rx-mask (therefore 2, 4, 8, 16, 32...).

Data area... In the data area the received data are stored. A line with 16 bytes for the storing of status information and data is available for each Rx-identifier.
Following figure shows the structure of a line:

Relative address	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Relative length	WORD		WORD		WORD		BYTE	BYTE	BYTE	BYTE	BYTE	BYTE	BYTE	BYTE	WORD	
Function	TIME STAMP		Net-No.		Idf		Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7	Data 8	free	

Bit 15...5	Bit 4	Bit 3...0
11 Bit CAN ID 0...2047	RTR	LENGTH

Fig. 3.3.1: Structure of the data area of an identifier

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

Net no..... Number of the CAN net on which the RxId has been received (local CAN interface 0/1).

Idf..... contains the 11 bit long CAN-Id, the RTR-bit (remote transmission request) and the number ofd the transmitted bytes (LENGTH):

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

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

3.4 CMS-Implementation

The CMS-implementation can be recognized by the appendix '.cms' at the EPROM designation.

3.4.1 CMS-Domain Transfers

CMS-domain transfers serve the transmission of data blocks by the CAN. An Rx-identifier and a Tx-identifier are necessary for the transmission. The transfer is initiated and controlled by a 'client' and served by a 'server'. It is possible to transmit the data in both directions.
The CAN2 can work at CMS-domain transfers as client and server.

The selection of the net and the identifiers for the domain transfer occurs by the command 'init domain' of the parameter buffer. By this command also the size of the memory area (CMS-buffer) required for the data memory and parameter transfer is defined. A detailed description can be taken from the following chapter 'Commands of the Parameter Buffer'.

Apart from the parameter buffer, the user needs further parameters for the control of the local firmware. In the cell NET_LINK/BUFFER-POINTER, e.g., of the identifiers used for the up- and download, the initial address of the 'CMS-buffer' is stored. The CMS-buffer contains further control paramaters and serves the data memory during the CMS-transfer. It will be explained more detailed in the following chapter.

The data-structural components of the Rx-and Tx-identifiers are partly used as with conventional transmissions. For example, Tx-transfers are started by LENGTH and the success of a transmission can be requested after the transfer in the cell status. Into TOUT the according Rx- and Tx-time-out times have to be entered.

The following figure shows the structural components and (shown briefly) the CMS-buffer.

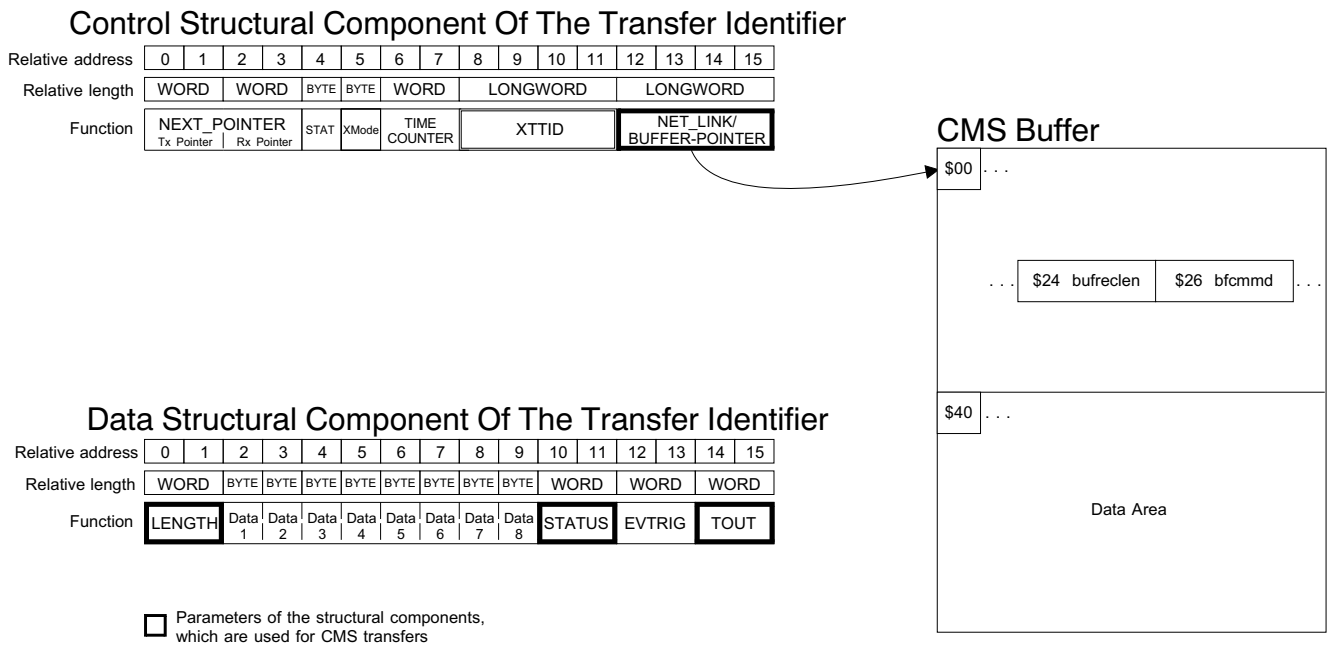


Fig. 3.4.1: Important parameter cells for CMS-transfers

3.4.1.1 CMS-Buffer

The CMS-buffer serves the memory of the data to be transmitted or received and the transmission of commands and status messages.

The following table shows the principal structure of the CMS-buffer. The parameters given in parentheses () are needed by the local firmware and are not important to the user. They must not be changed by the user!

relative address [HEX]	length	access	cell contents	short description
00 : 1A	-	read only	-	These parameters are intended for the local firmware and must not be changed by the user!
1C	LWORD	read only	**buflen**	Maximum length of the data area of this buffer in byte (is given by init domain)
20	LWORD	read only	**bufadr**	Absolute initial address of the data area
24	WORD	read/ write	**bfreclen**	Number of valid data in the buffer Tx -▸ number of data to be transmitted Rx -▸ number of received data
26	WORD	read/ write	**bfcmmd**	Idle : 0 Client download : 1 Client upload : 2 Server mode : 3
28	LWORD	read/ write	**bfmux**	Multiplex value for init domain Up/download ($0000.0000 ... $00FF.FFFF)
2C	WORD	read/ write	**bflast**	Set to transmit at the last data segment CMS-end bit (P 0 -▸ active) or read to request end bit
2E	WORD	read only	**bfsvst**	Server status 1: sv-down 2: sw-up
30	BYTE	read only	(bfstat)	internal status (P0 : initiated)
31	BYTE		(bfxact)	-
32	WORD		(bfcont)	actual counter for data transfer
34	BYTE		(bftbit)	Toggle bit
35	BYTE		(bfssol)	-
36, 37	BYTE		reserved	-
38	LWORD		(bftxid)	Pointer on transmit control structure
3C	LWORD		(bfrxid)	Pointer on receive control structure
40	is given by command 'init domain'	read/ write	data area	Reading received data and writing data which should be transmitted

Table 3.4.1: Structure of the CMS-buffer

3.4.1.2 Transfer Procedure

3.4.1.2.1 To the Following Descriptions

In this chapter the procedure of the up- and downloads will be described. To make the used firmware and the software to be provided by the user easier comprehensible, flow diagrams have been added for all operating modes.

The flow diagrams of the VMEbus-master software, which is not included in the delivery volume, has to be seen as support: The operation of the CAN2 firmware can partly be realized differently.

The flow diagrams of the local CAN2 firmware show the principal procedure of the local programs. For reasons of clearity this description is without a detailed program description.

The described VMEbus interruots of the CAN2 are only triggered, if the cells 'iolev' and 'iovec' have been set before in the parameter buffer, accordingly! The interrupt cause leads always also to the writing of the concerned identifier into the FIFO 'data to VME'.

3.4.1.2.2 Initialisation of the CMS-Transfer Buffer

Independent from the kind of the following transfers, the first steps are always the same:
At first the CMS-buffer has to be initialized. Normally, the buffer is only initialized once after the system start and is maintained until the next RESET:

- Calling the command 'init domain'
  By this command of the parameter buffer, one of the two nets of the CAN2 is selected and the size of the data buffer is given. Furthermore the Rx-identifier and the Tx-identifier for the transfer are selected.

- After the execution of the command, the absolute local initial address of the CMS-buffer can be read in the returned parameter of the parameter buffer 'retpar' or in the cell 'NET_LINK/BUFFER-POINTER' in the monitoring-structural component of the used identifiers.

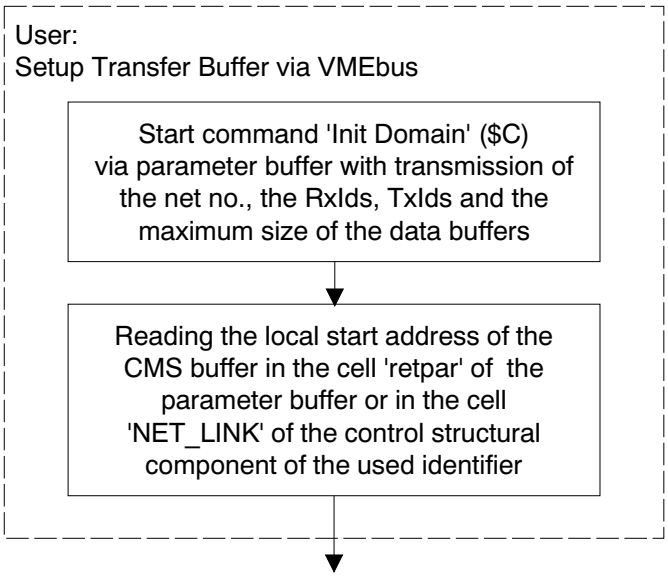


Fig. 3.4.2: Initialisation of the CMS-buffer and reading the buffer-initial address

3.4.1.2.3 Client Download

The following figure shows the principal procedure of a client download.

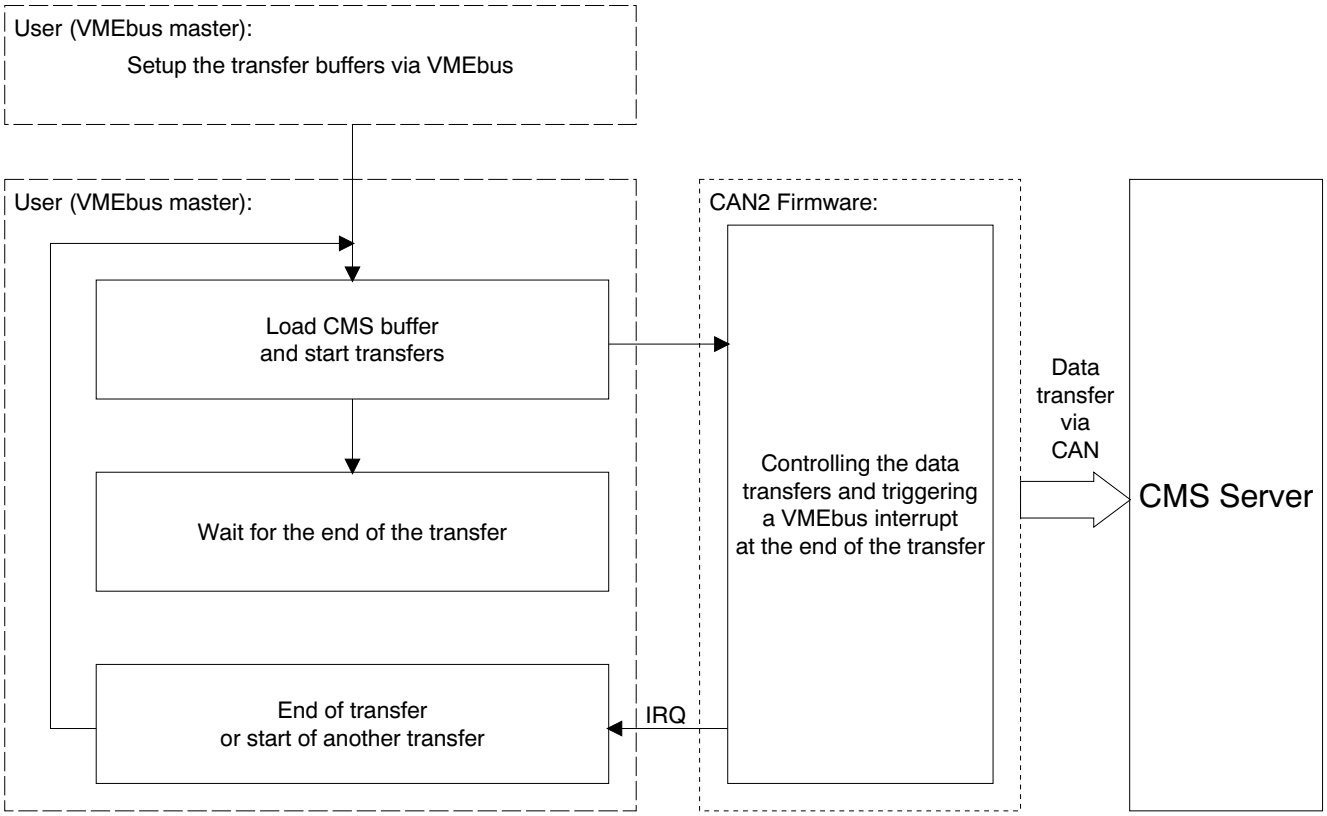


Fig. 3.4.3: Overview of the client download

Requirement for a client download is the initialisation of the CMS-buffer.

The download generally occurs in three steps: Via the VMEbus the data buffer on the CAN2 is loaded and the transfer is started. The CAN2 transmits the data to the CMS-server and triggers a VMEbus interrupt after completing the transfer. The master 'waits' for this interrupt, checks the success of the transfer and starts further transfers, if necessary.

In the following the particular blocks of the figure above will be described more in detail.

The 'initialisation of the transfer buffer' has already been described in the preceding chapter.

The figure shown below exemplary displays the program procedure for the VMEbus master during a client download. This part is not contained in the delivery volume of the CAN2 and has to be realized by the user himself.

AFter the cells 'bfcmmd' and 'bfmux' have been set, the transmission of the data can occur. Because this transmission sequence can also be used for the 'server upload', it is recommended to store this program part as a subroutine. This module is called 'BLOCK-OUT', here. The flow chart to BLOCK-OUT will be explained subsequently to the one of the VMEbus master.

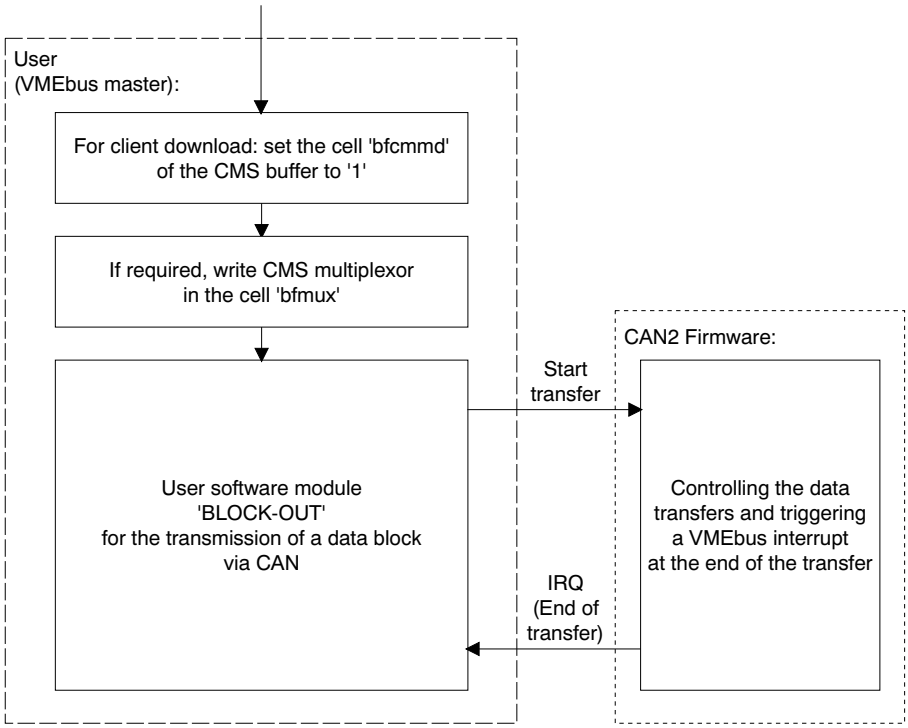


Fig. 3.4.4: VMEbus master at client download

About the user-software module 'BLOCK-OUT' (following figure):

The program part, described here exemplary, carries out following functions:

- Loading the data to be transmitted into the data area of the CMS-buffer and entering the number of bytes into the cell 'bfreclen' in the CMS-buffer.

- If this is already the last data block, the cell 'bflast' has to be set to a value unequal nil. At the transmission of the last segment the CAN2 firmware will set the CMS-end bit accordingly.

- The download is started by the single triggering of a Tx-transfer on the selected Tx-identifier. Corresponding time-out values have to be set before in the data-structural components for the Tx-identifier and the reply on the Rx-identifier.
  Starting the transfer can, e.g., occur by setting the cell 'LENGTH' in the data-structural component to a negative value or a value between $0060 and $0068. The absolute value of LENGTH and the data of this TxId entered in the structural component are meaningless in this case.
  With these preadjustments the local software recognizes that the transfer is a CMS-service and not a standard-Tx-transfer and starts to transmit the contents of the data buffer according to the CMS-protocol.

- If the local firmware of the CAN2 has completed the download of the data of the CMS-buffer, the CAN2 triggers a VMEbus interrupt. The download is completed when all data have been transmitted or when an error occurred during the transmission. Possible error reasons are, e.g., a Tx-error or the missing reply of the CMS-server.

  The VMEbus master has to operate this interrupt and can read the cell 'status' in the data-structural component afterwards to check, whether the transmission has been completed successfully.
  The interrupt is a CAN-server interrupt of the CAN2. In the FIFO the identifier can be found which triggered the interrupt. The resetting of CAN2 interrupts will be described in the following chapter 'Summary of the Interrupt Options'. The conditions of the status cell can be taken from the preceding description of the data-structural component. The conduct at faulty transmission is left to the user, here.

- Because the size of the CMS-data buffer is limited, several transfer cycles might be needed, possibly, to transmit the desired data. In this case it is possible to skip to the start of the module. The new transfer starts again with loading the buffer.

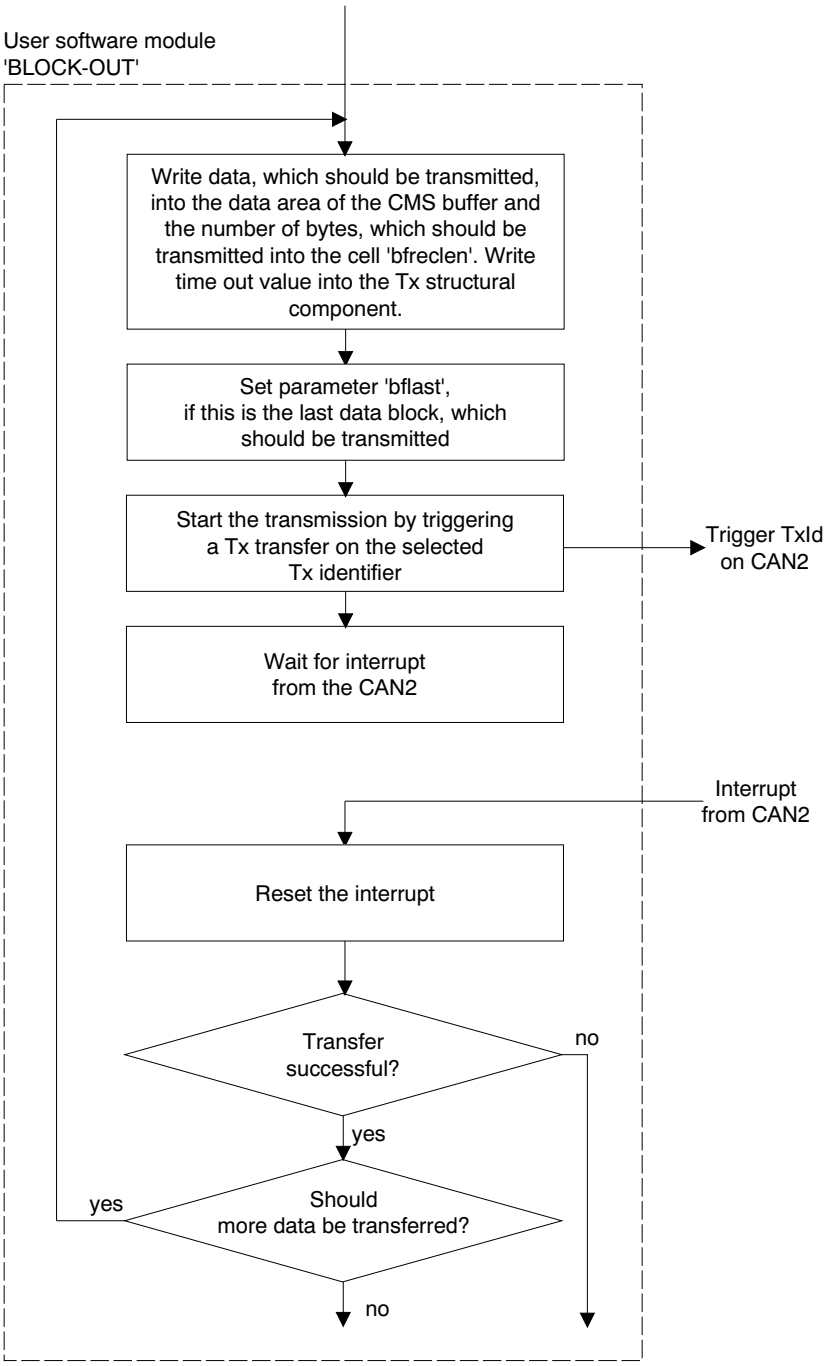


Fig. 3.4.5: User-software module 'BLOCK-OUT'

In the following figure the procedure on the local firmware for a client download is shown. The firmware transmits the necessary init sequence to the connected server and controls the transmission of the data, independently.

If the cell 'bflast' of the CMS-buffer had been set by the user, the end bit will be set at the last transmission.

After the last transmission with the end bit, the firmware resets the cell 'bfcmmd' to '0', which had been set to '1' by the user. The cell 'bfcmmd' is also reset, if the transmission had been interrupted because of an error.

The user is informed about the end of the transmission by an interrupt.

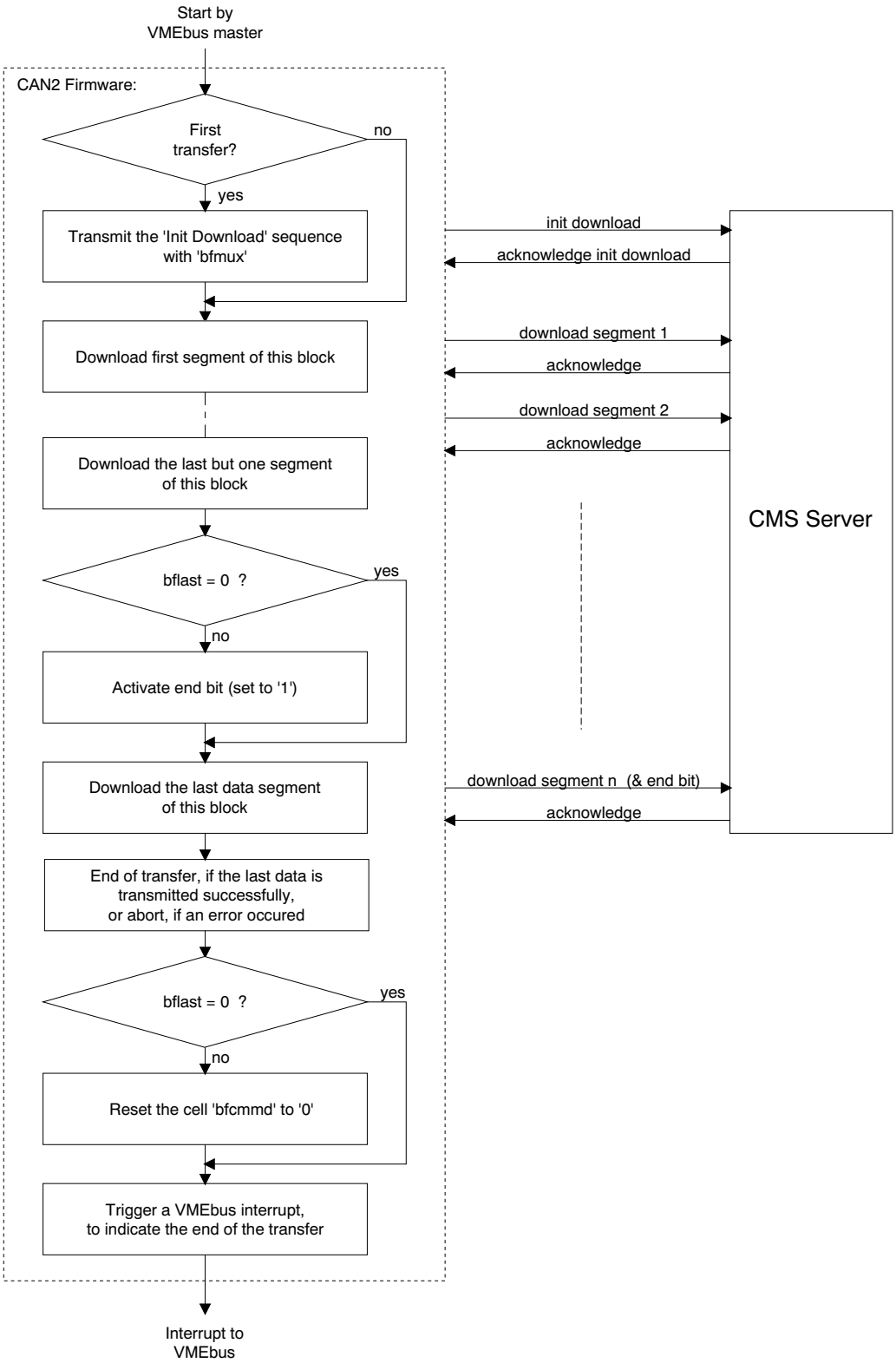


Fig. 3.4.6: Procedure of the local CAN2 firmware for client-download tansfers

3.4.1.2.4 Client Upload

The following figure shows the principal procedure of a client upload.

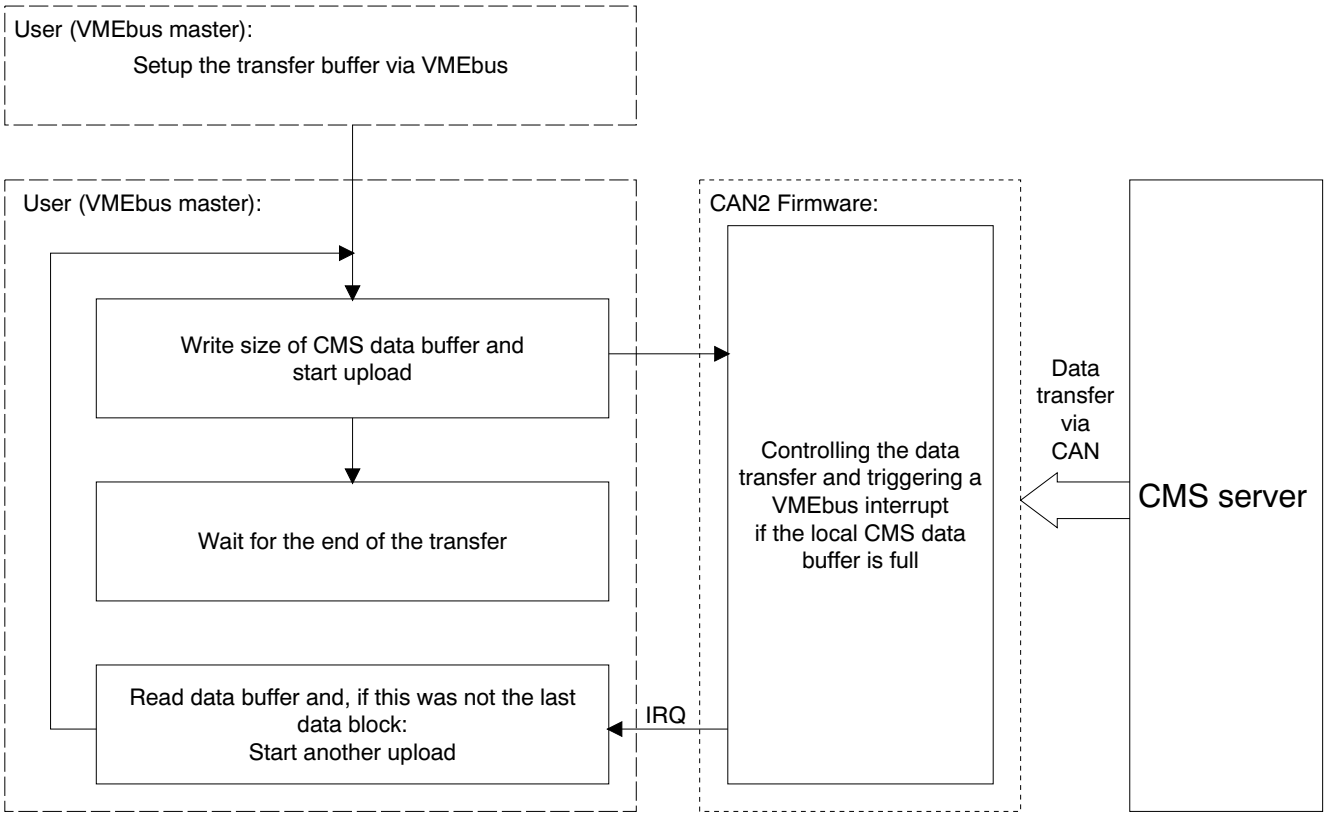


Fig. 3.4.7: Overview of the client upload

Requirement for a client upload is the initialisation of the CMS-buffer.

The upload generally occurs in three steps: By the VMEbus the size of the data buffer on the CAN2 is given and the transfer is started. The CAN2 receives the data from the CMS-server and triggers a VMEbus interrupt when the buffer is full. The master 'waits' for this interrupt, checks the success of the transmission and starts further uploads, if necessary.

In the following the individual blocks of the figure above will be described more in detail.

The 'initialisation of the transfer buffer' has already been described in a preceding chapter.

The figure shown below shows exemplary the program procedure for the VMEbus master during a client upload. This program part is not contained in the scope of delivery of the CAN2 and has to be realized by the user himself.

After the cell 'bfcmmd' has been set, the data transmission can occur. Because this transmission sequence can also be used for the 'server download', it is recommended to store this program part as subroutine. This module is called 'BLOCK-IN', here. The flow diagram to BLOCK-IN will be explained subsequently to the one of the VMEbus master.

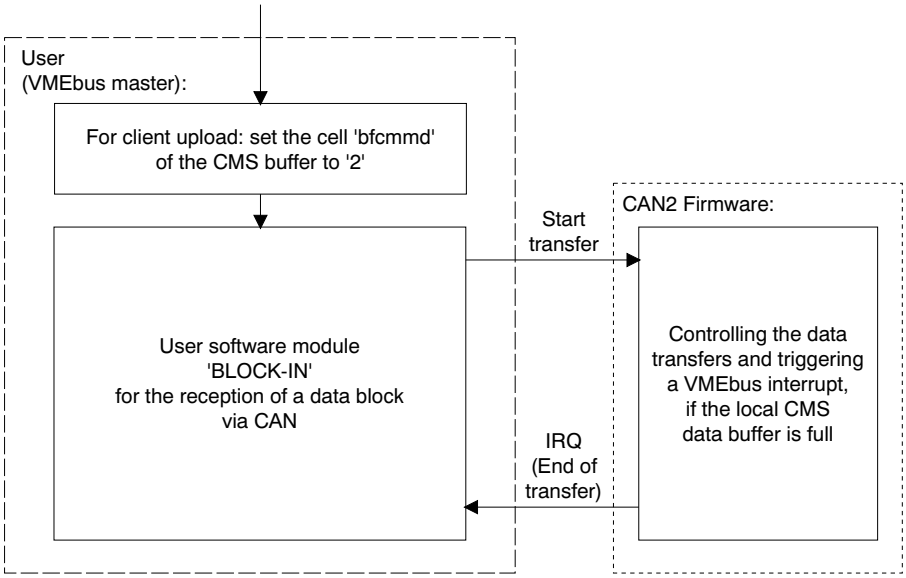


Fig. 3.4.8: VMEbus master at client upload

About the user-software module 'BLOCK-IN' (following figure):

The program part, described here exemplary, carries out following functions:

- At first the maximum number of data to be received are entered into the cell 'bfreclen' of the CMS-buffer. For 'bfreclen' the complete contents of the data area can be used. This value can be read in the cell 'bflen'. When the number of the loaded data reaches 'bfrecl - 6' or the end bit has been received, the transmission is ended. If an error arises during the upload, the transmission will be aborted.

- Starting the upload occurs by triggering a Tx-transfer. The procedure is identical to the one of the software module 'BLOCK-OUT' and can be looked up there.

- After the data buffer of the CAN2 is filled, the board returns with a VMEbus interrupt. The interrupt has to be operated by the VMEbus master. The interrupt handling on the CAN2 will be described more in detail in the following chapter 'Summary of the Interrupt Options'.

- By requesting the cell 'status' in the data-structural component of the Rx-identifier, it is evaluated, if the transmission had been successful. The conditions of the cell status can be taken from the preceding description of the data-structural component. The conduct at faulty transmissions is left to the user, here.

- Now the CMS-buffer is read out.

- Requesting the cell 'bflast' gives information, if more data are to follow or if this was the last data block.

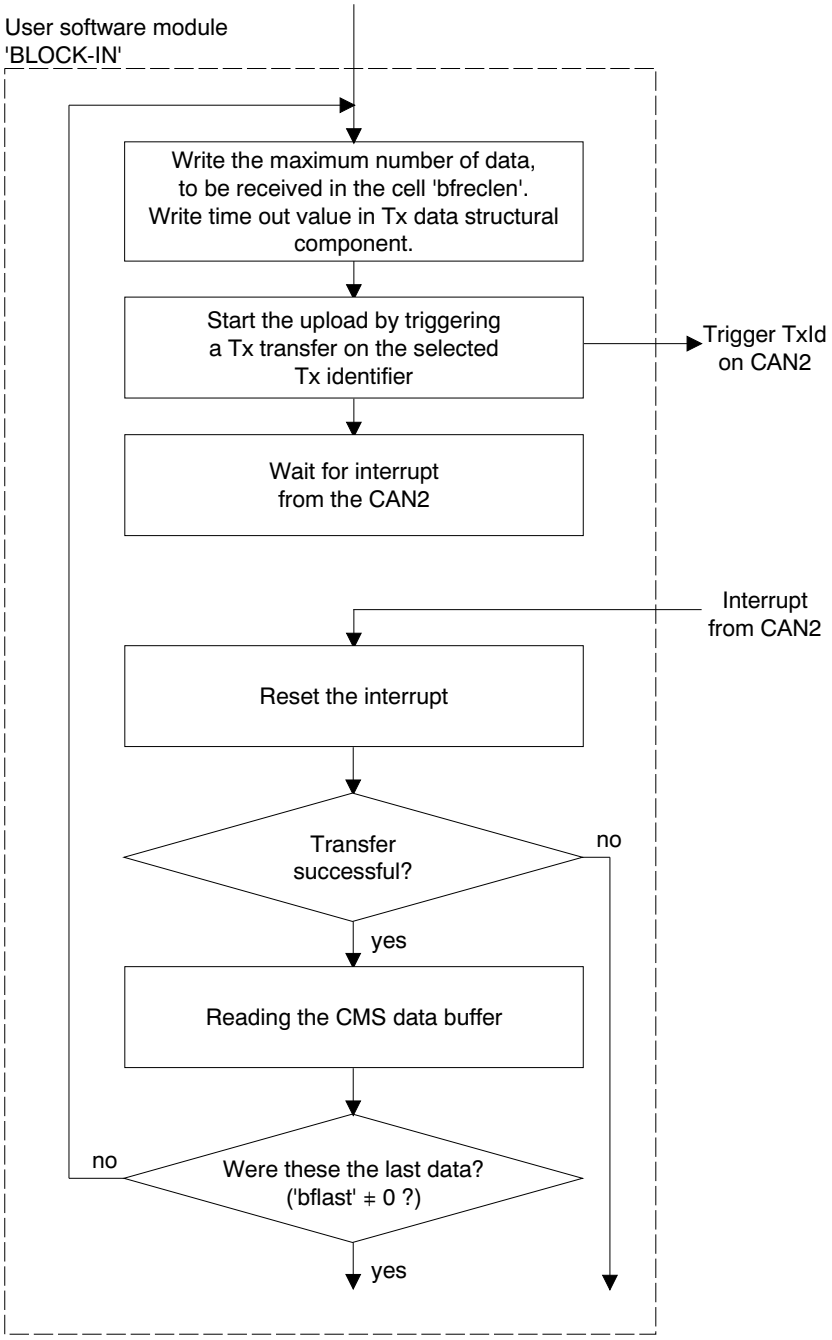


Fig. 3.4.9: User-software module 'BLOCK-IN'

In the following figure the procedure of the local firmware for a client upload is shown. The firmware independently transmits the necessary init sequence to the connected server and controls the transmission of the data.

If an active end bit is received, the local firmware sets the cell 'bflast' to '-1' to inform the VMEbus master about the end of the transmission. Also after the reception of the end bit, the firmware resets the cell 'bfcmmd', which had been set by the user to '2', to '0'. The cell 'bfcmmd' is also reset, if the transmission had been aborted because of an error.

The user is informed about the end of the transmission by an interrupt.

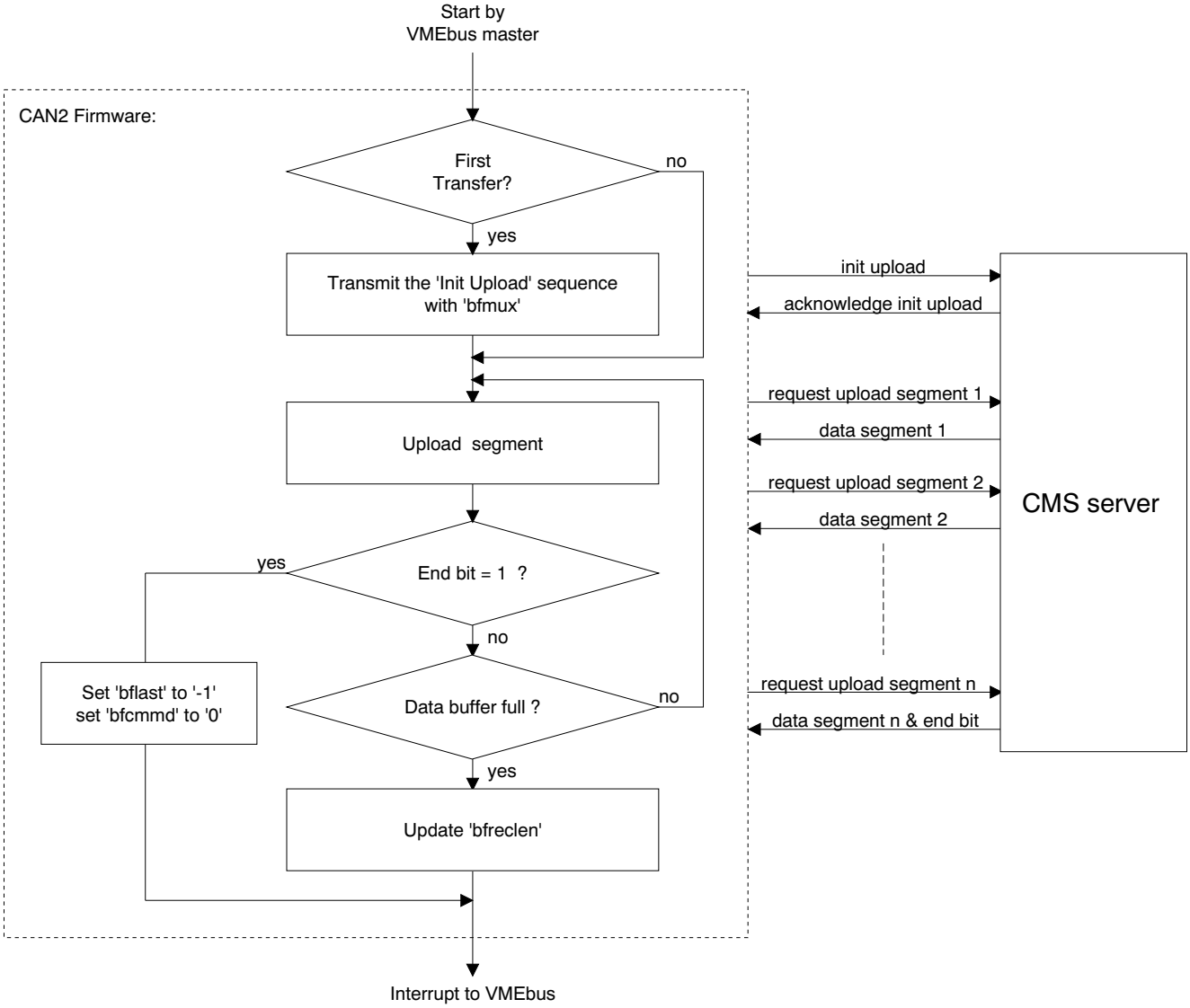


Fig. 3.4.10: Procedure of the local CAN2 firmware for client-upload transfers

3.4.1.2.5 Server Download

The CAN2 is able to handle downloads also as a CMS-server. In this case the transfer is started and controlled by another CAN partner (client).

Like in the already described client transmissions, at first an 'init domain' is carried out on the CAN2 and the CMS-buffer address is determined (see preceding chapter 'Initialisation of the CMS-Transfer Buffer').

Setting the cell 'bfcmmd' in the CMS-buffer to '3' selects the server mode. The mode is activated by triggering a Tx-transfer. The CAN2 is now in server mode 'idle'. The actual server mode can be read in the cell 'bfsvst' of the buffer. The client is now able to request an upload or download from the CAN2. The CAN2 will trigger a VMEbus interrupt then.

The following figure shows simplified the program procedure of the VMEbus master for the initialisation of the CAN2 and the polling of the conditions upload/download. After triggering the Tx-transfer, the VMEbus master is in a Time-out loop. The loop can only be exit by an interrupt of the CAN2.

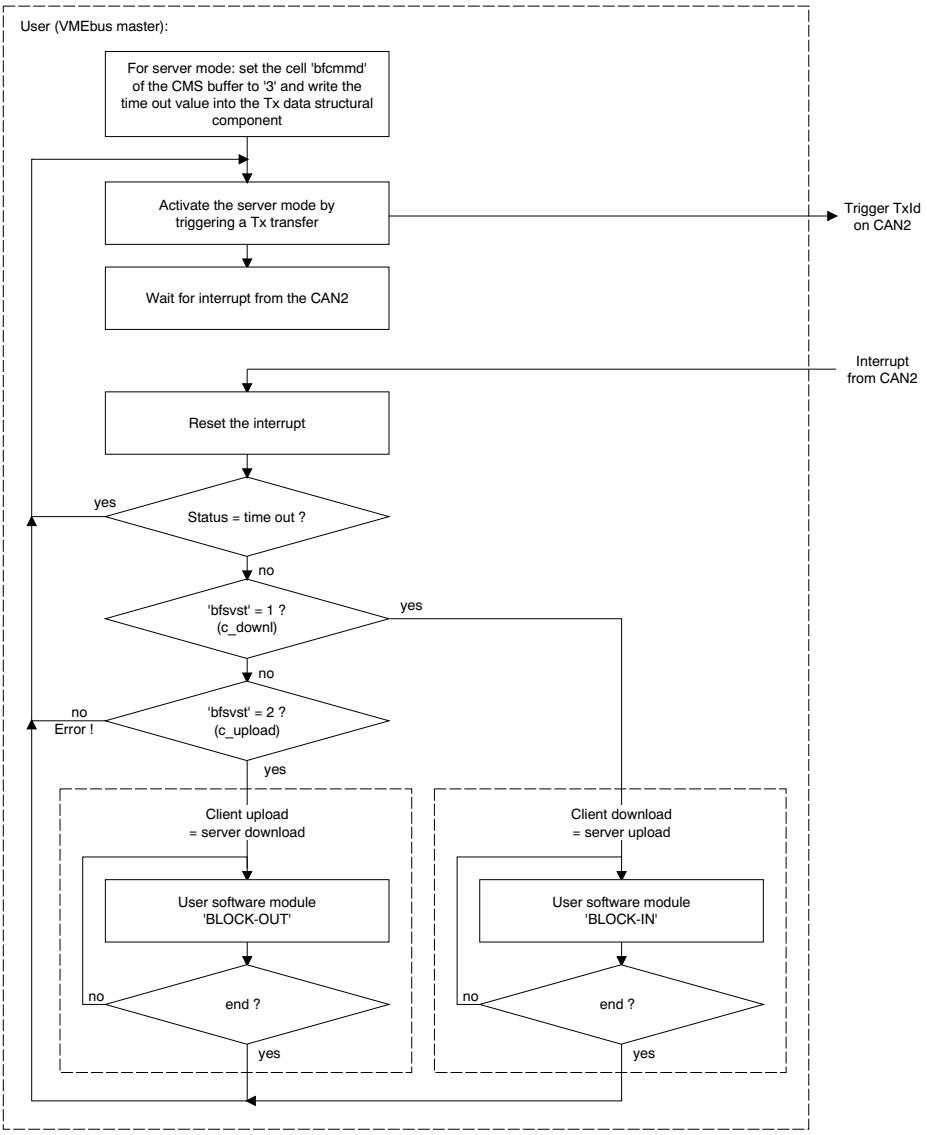


Fig. 3.4.11: Recommended structure for polling the upload/download request

The interaction of VMEbus master, CAN2 and CMS-client is shown in the following figure. Requirement for the transfer is, as with the operating modes described so far, the existence of an initialized CMS-buffer.

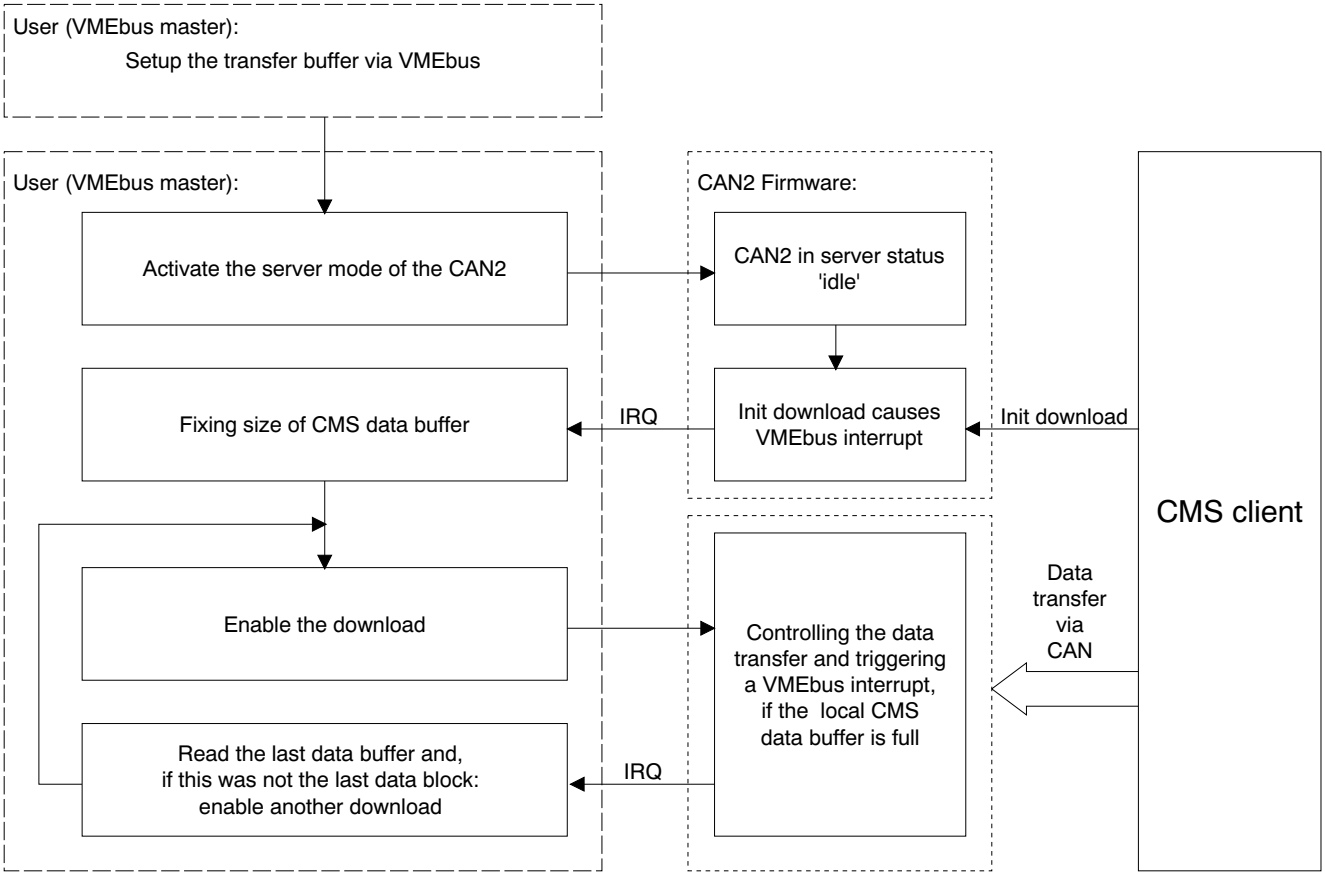


Fig. 3.4.12: Overview of the server download

After the start of the server download, the CAN2 is in server status 'idle'. If the CAN2 receives an 'init download' of the CMS-client, it triggers a VMEbus interrupt. The VMEbus master now has to determine the maximum number of the data to be read in the buffer of the CAN2 and enable the download by a Tx-transfer.
The further download control occurs by the CMS-client. When the data buffer on the CAN2 is full, it triggers an interrupt again. The VMEbus master has to read out the buffer and can enable further downloads, if necessary.

In the following the individual blocks of the figure above will be described more in detail:

The 'initialisation of the transfer buffer' has already been described in a preceding chapter.

The figure shown below shows exemplary the program procedure for the VMEbus master during a server download. This program part is not contained in the scope of delivery of the CAN2 and has to be realized by the user oneself, therefore.
After the cell 'bfcmmd' has been set, the server mode is started by triggering a Tx-transfer. The CAN2 now waits for an init-download or init-upload sequence by the CMS-client. In this case an init download may be assumed.

As already mentioned, the CAN2 triggers a VMEbus interrupt which has to be handled by the VMEbus master. The master reads the cell 'bfsvst' to check, whether it is upload or download and calls corresponding routines.
For a server download this is the module 'BLOCK-IN', which has already been used at the client upload. The module has already been described in detail in the chapter Client Upload.

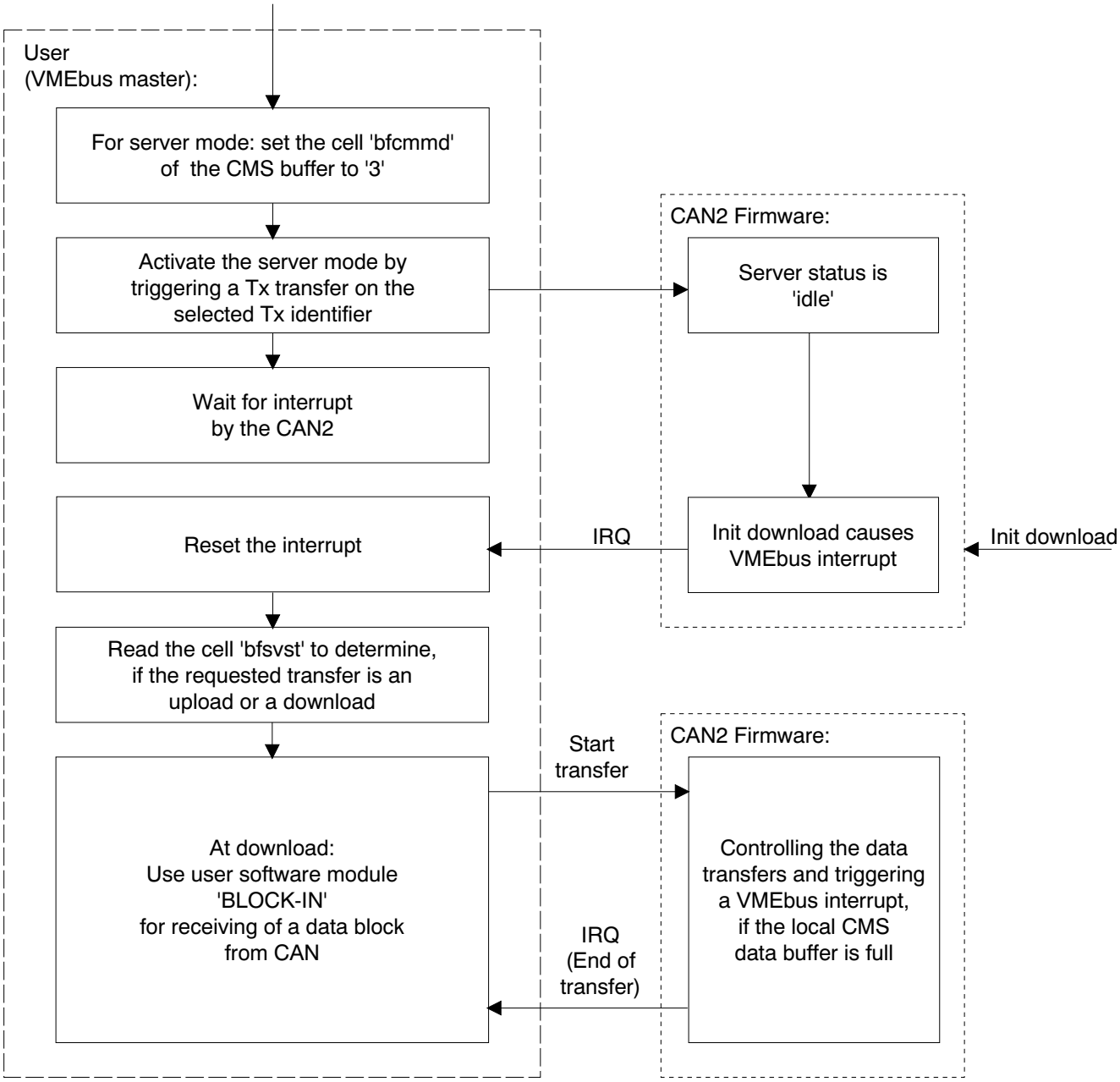


Fig. 3.4.13: VMEbus master during server download

In the following figure the procedure of the local firmware for a server download is shown.

The sequence shown in the first block is only run through once for each server download.

After enable by the VMEbus the firmware starts the download loop. If not all data have been transmitted after filling the data buffer, the download loop can be started again by the VMEbus. The initialization of the local firmware by the VMEbus master occurs by triggering a Tx-transfer on the TxId which had been selected for the CMS-transfers.

If an active end bit is received, the local firmware resets the cell 'bflast' to '-1' to inform the VMEbus master about the end of the transfer. Also after the reception of the end bit, the firmware resets the cell 'bfsvst' to '0' and remains in 'idle' state. The cell 'bfsvst' is also reset, if the transmission had been aborted because of an error.

The user is informed about the end of the transmission by an interrupt.

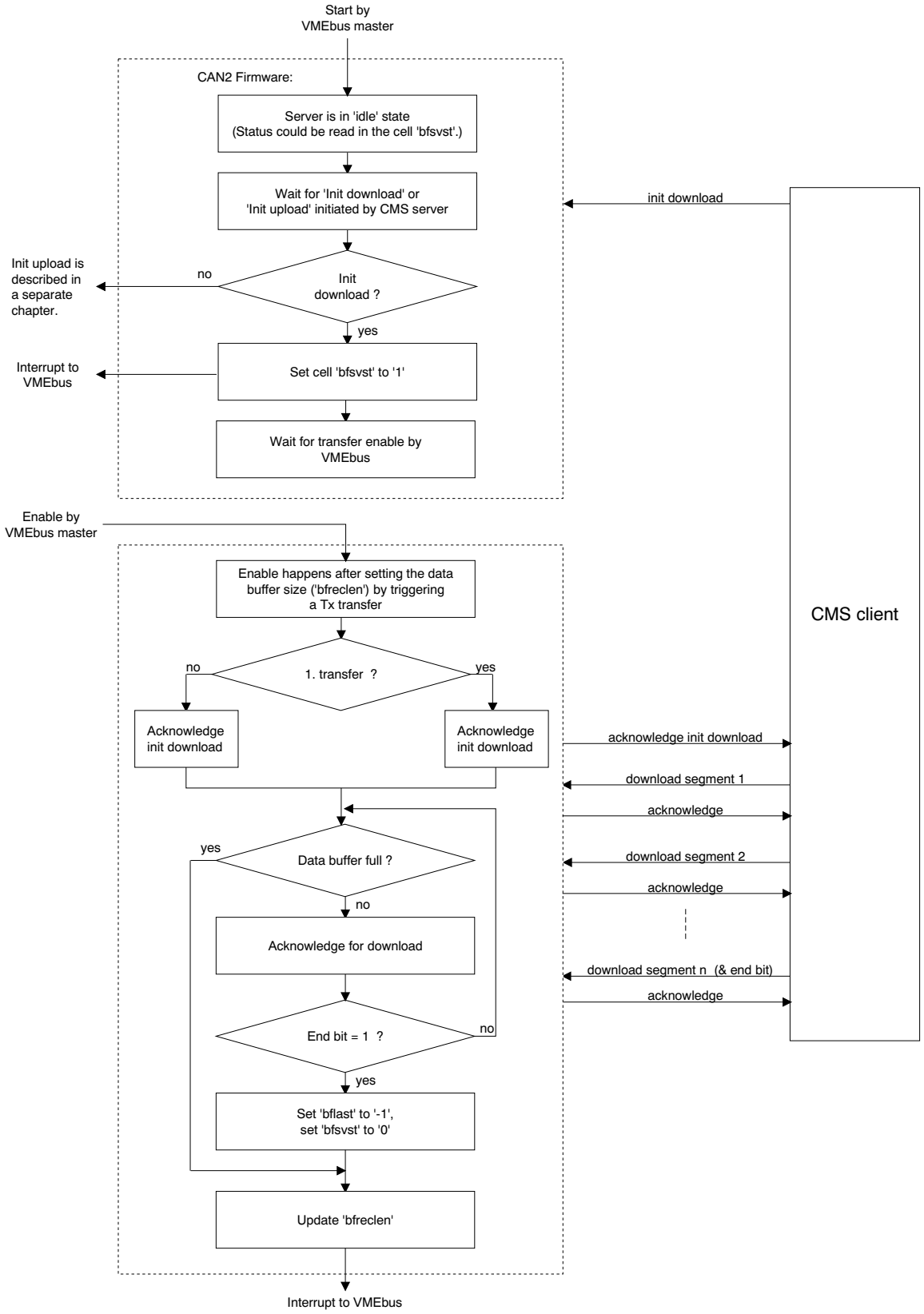


Fig. 3.4.14: Procedure of the local CAN2 firmware for server-download transfers

3.4.1.2.6 Server Upload

The CAN2 is able to handle uploads also as a CMS-server. In this case the transfer is started and controlled by another CAN partner (client).

Like with the already mentioned client transmissions, at first an 'init domain' is carried out on the CAN2 and the CMS-buffer address is determined (see preceding chapter 'Initialisation of the CMS-Transfer Buffer').

Setting the cell 'bfcmmd' in the CMS-buffer to '3' selects the server mode. The mode is activated by triggering a Tx-transfer. The CAN2 now is in the server status 'idle'. The actual server mode can be read in the cell 'bfsvst' of the buffer. The client is now able to request an upload or download by the CAN2. The CAN2 will trigger a VMEbus interrupt then.

In the preceding chapters a simplified program procedure of the VMEbus master, especially for the initialisation of the CAN2 and the polling of the conditions upload/download, has already been shown.

The interaction between VMEbus master, CAN2 and CMS-client is shown in the following figure. Requirement for the transfer is, like with the already mentioned operating modes, the existence of an initialized CMS-buffer.

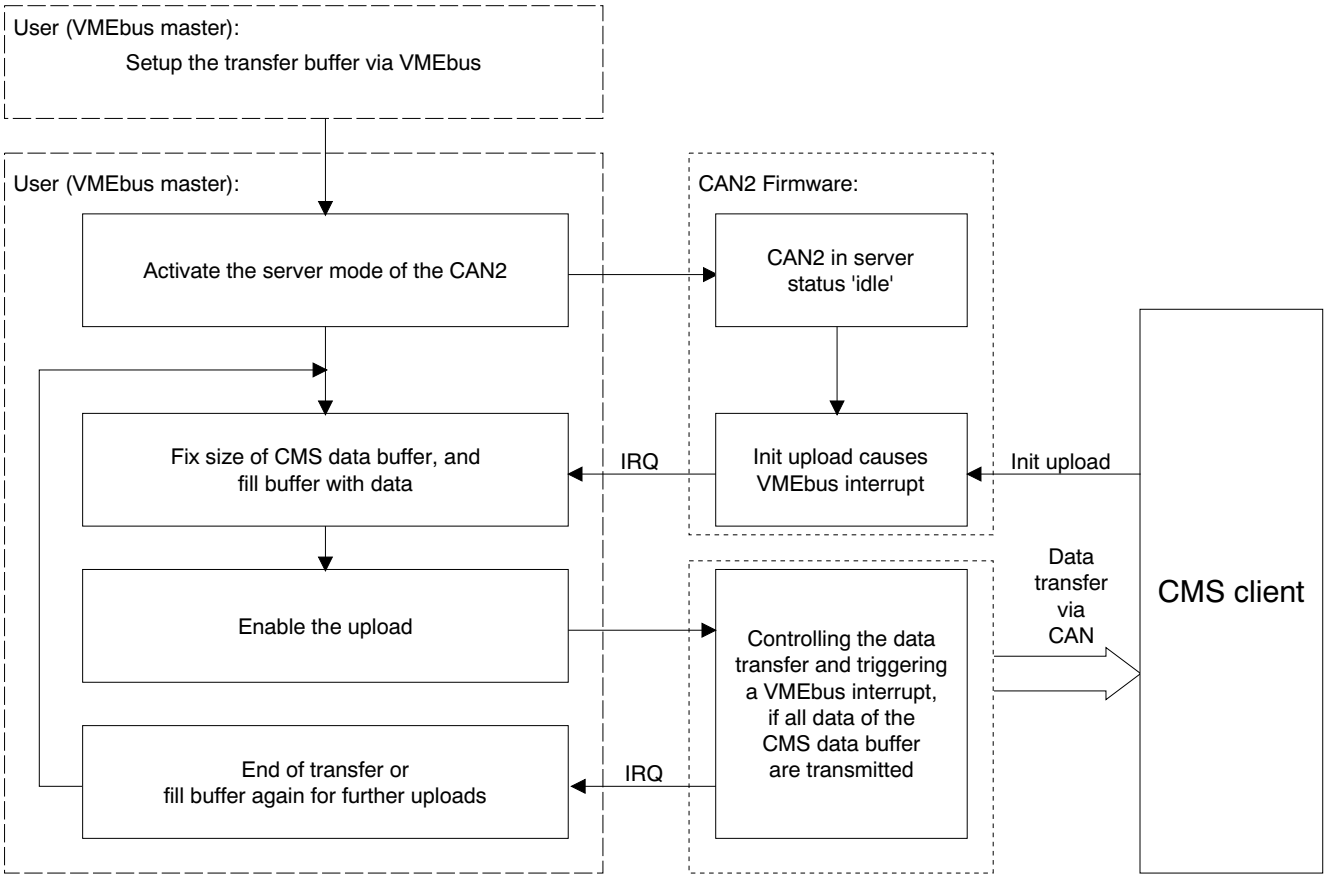


Fig. 3.4.15: Overview of the server upload

After start of the server mode, the CAN2 is in the server mode 'idle'. If the CAN2 receives an 'init upload' of the CMS-client, it triggers a VMEbus interrupt. The VMEbus master now has to determine the size of the data buffer of the CAN2, fill the buffer with data and enable the upload by a Tx-transfer.

The further control of the upload occurs by the CMS-client. If the transfer is completed, the CAN2 triggers an interrupt. The VMEbus master is now able to check the success of the transmission and handle a new transfer, if necessary.

In the following the individual blocks of the figure above will be described more in detail:

The 'Initialisation of the Transfer Buffer' has already been described in a preceding chapter.

The following figure shows exemplary the program procedure for the VMEbus master during a server upload. This program part is not contained in the scope of delivery of the CAN2 and has to be realized by the user oneself, therefore.

After the cell 'bfcmmd' is set, the server mode is started by triggering a Tx-transfer. The CAN2 now waits for an init-download or init-upload sequence by the CMS-client. In this case an init upload may be assumed.

As already mentioned, the CAN2 triggers a VMEbus interrupt which has to be handled by the VMEbus master. The master reads the cell 'bfsvst' to check, whether it is upload or download and calls corresponding routines.
For a server upload this is the module 'BLOCK-OUT', which has already deen used at the client download. The module has already been described in detail in the chapter Client Download.

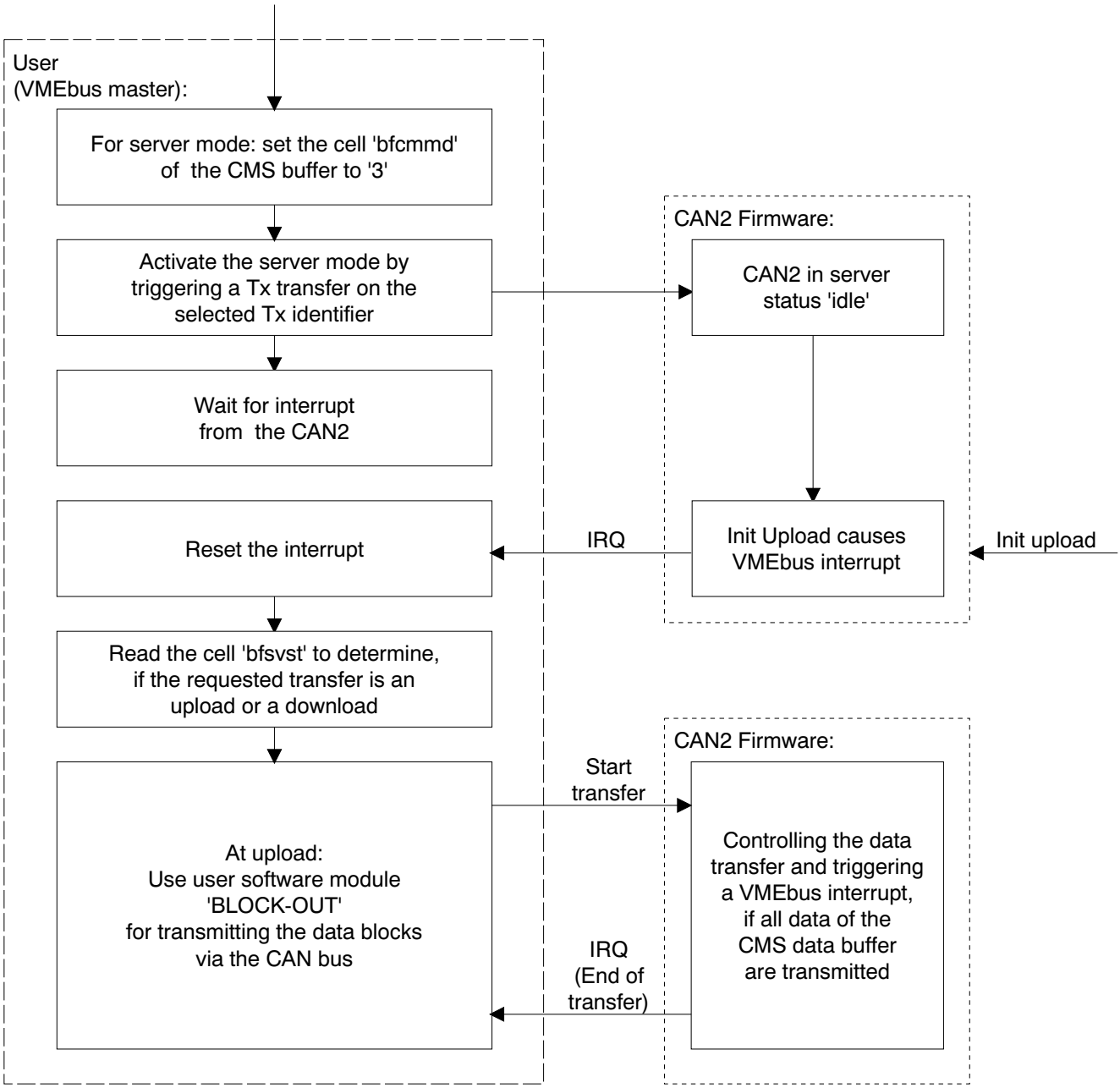


Fig. 3.4.16: VMEbus master during server upload

In the following figure the procedure of the local firmware for a server upload is shown:

The sequence shown in the first block is only run through once for each server upload.

After enable by the VMEbus, the firmware starts the upload loop. During the upload the local firmware counts the received data and triggers a VMEbus interrupt when the buffer is full. The VMEbus master can read out the buffer after this interrupt and then start the upload again.

The initialization of the local firmware by the VMEbus master occurs by triggering a Tx-transfer on the TxId which had been selected for the CMS-transfers.
Before writing the last data block into the CMS-buffer, the cell 'bflast' has to be set to a value unequal to zero. Then the local firmware sets the end bit at the transmission of the last data segment of this block, to inform the CMS-server about the end of the transmission. Furthermore the cell 'bfsvst' is reset to '0' after the last transmission.

The cell 'bfsvst' is also reset, if the transmission has been aborted because of an error.
The user is informed about the end of the transmission via an interrupt.

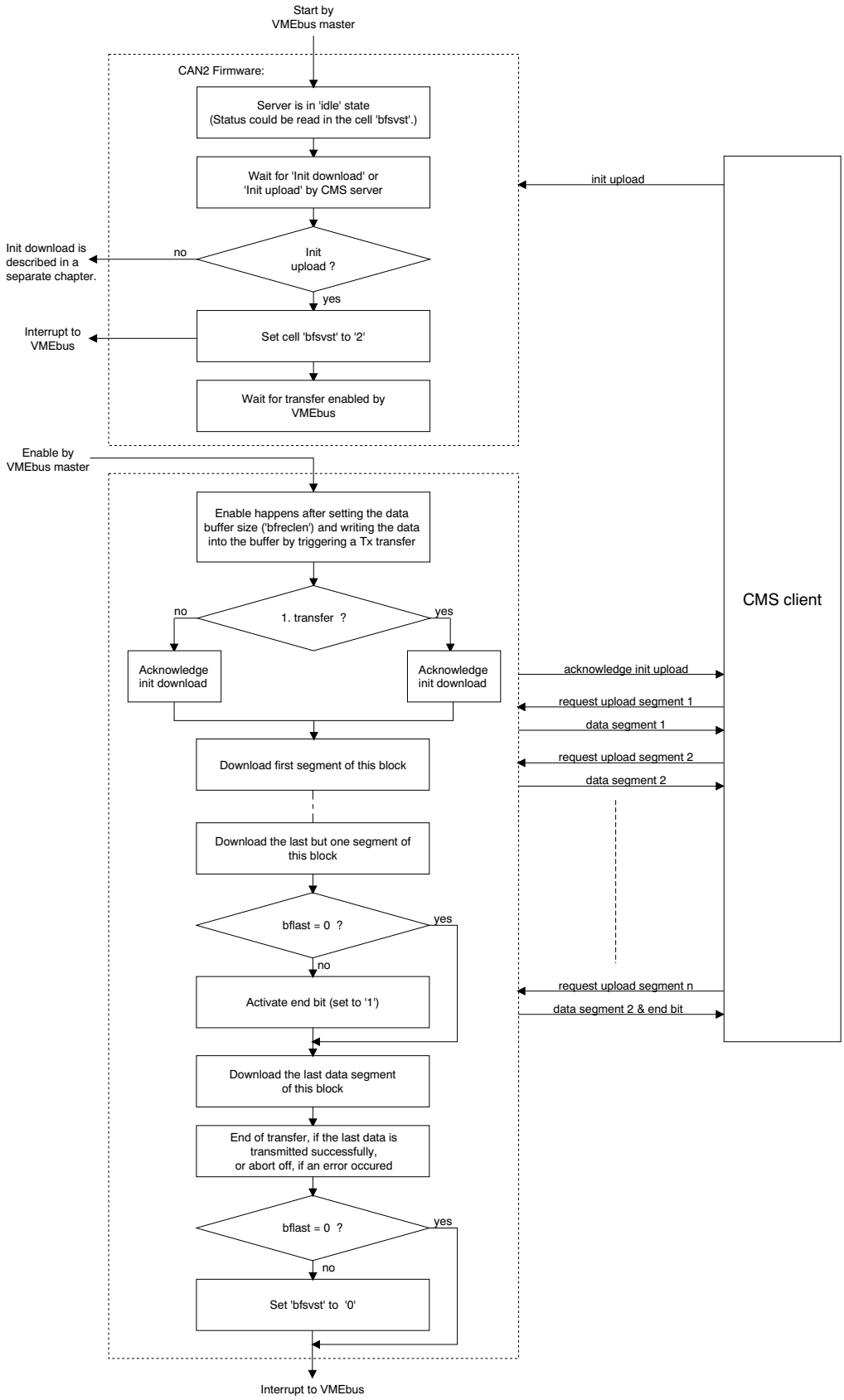


Fig. 3.4.17: Procedure of the local CAN2 firmware for server-upload transfers

3.4.2 CMS-Watchdog

The watchdog Tx-identifier and the desired CAN-net of the CAN2 are selected by the command 'activate periodical Tx-transfers' of the parameter buffer.
The entry will be explained more in detail in the following chapter 'The Paramater Buffer'.

Essential parameter for the CMS-watchdog are stored in the data-structural component of the watchdog-Tx-identifier:

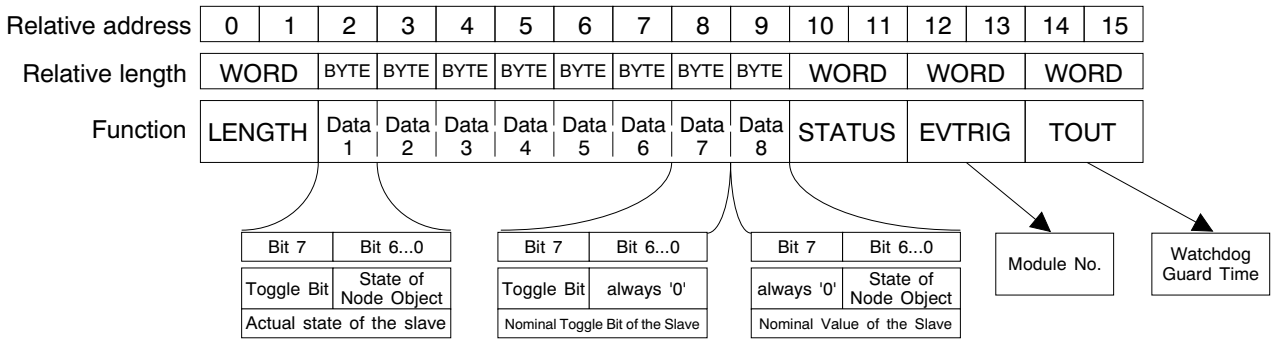


Fig. 3.5.1: Assignment of the data-structural component of the watchdog-Tx-identifier

The procedure of the watchdog function can be obtained from the following figure.

After the start of the watchdog by the VMEbus master, a control of the watchdog by the local firmware occurs. The CAN2 transmits RTR-requests on the watchdog-TxId and evaluates the responses of the accessed module.

If an error arises, a pointer is written into the FIFO 'Data To VME' and by this, if enabled, a VMEbus interrupt is triggered. The interrupt has to be handled by the VMEbus master. The evaluation of the error message is left to the user.

In case of error the watchdog is locally disabled and has to be re-enabled explicitly.

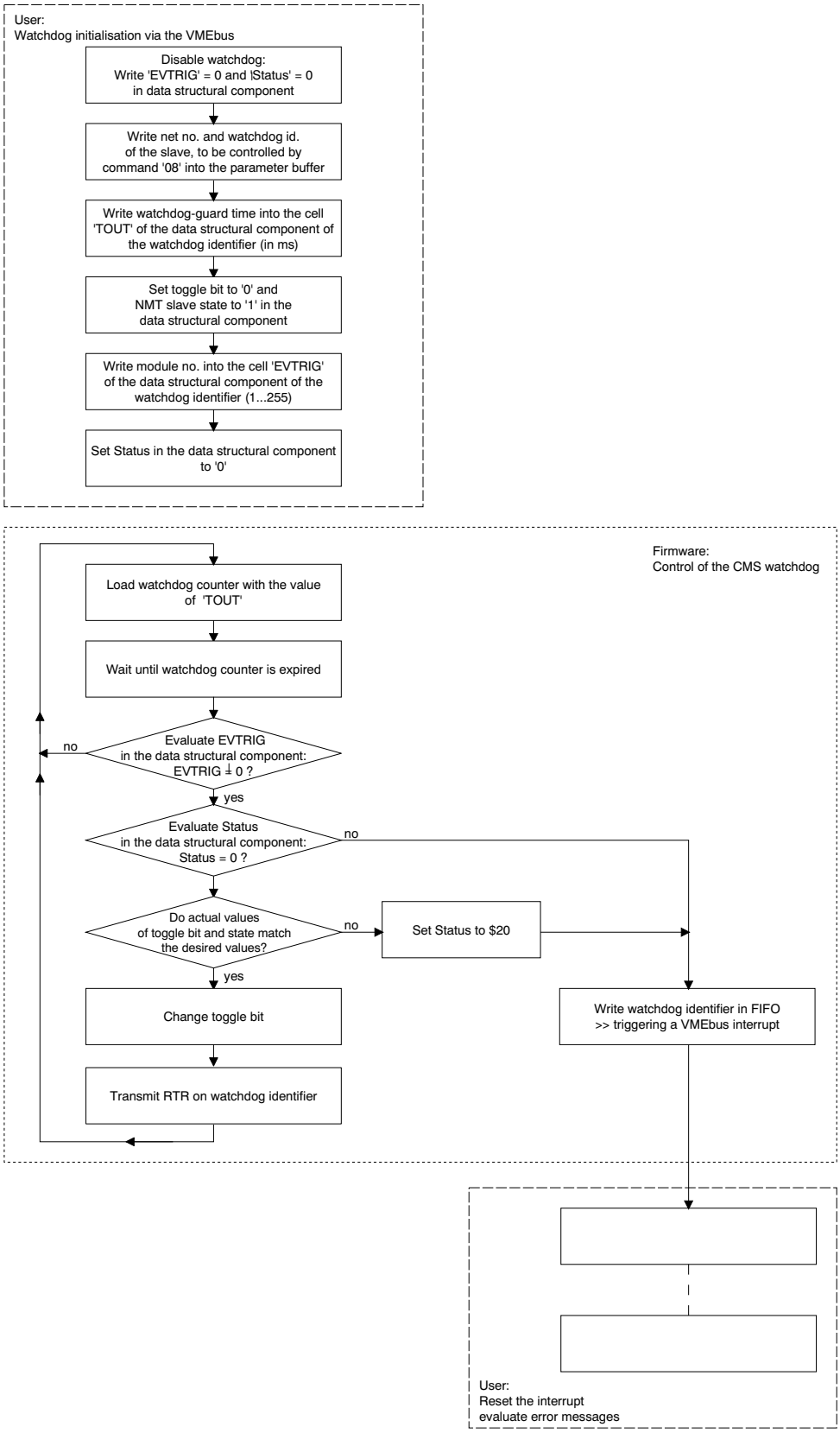


Fig. 3.5.2: Procedure of the watchdog control on the CAN2

4. The Parameter Buffer

4.1 Function and Structure

By the parameter buffer, various commands, the parameters of the monitor buffers and various parameters of the physical CAN interface are transferred.

In the following figure the structure of the parameter buffer with the parameters, relevant to the user is shown.

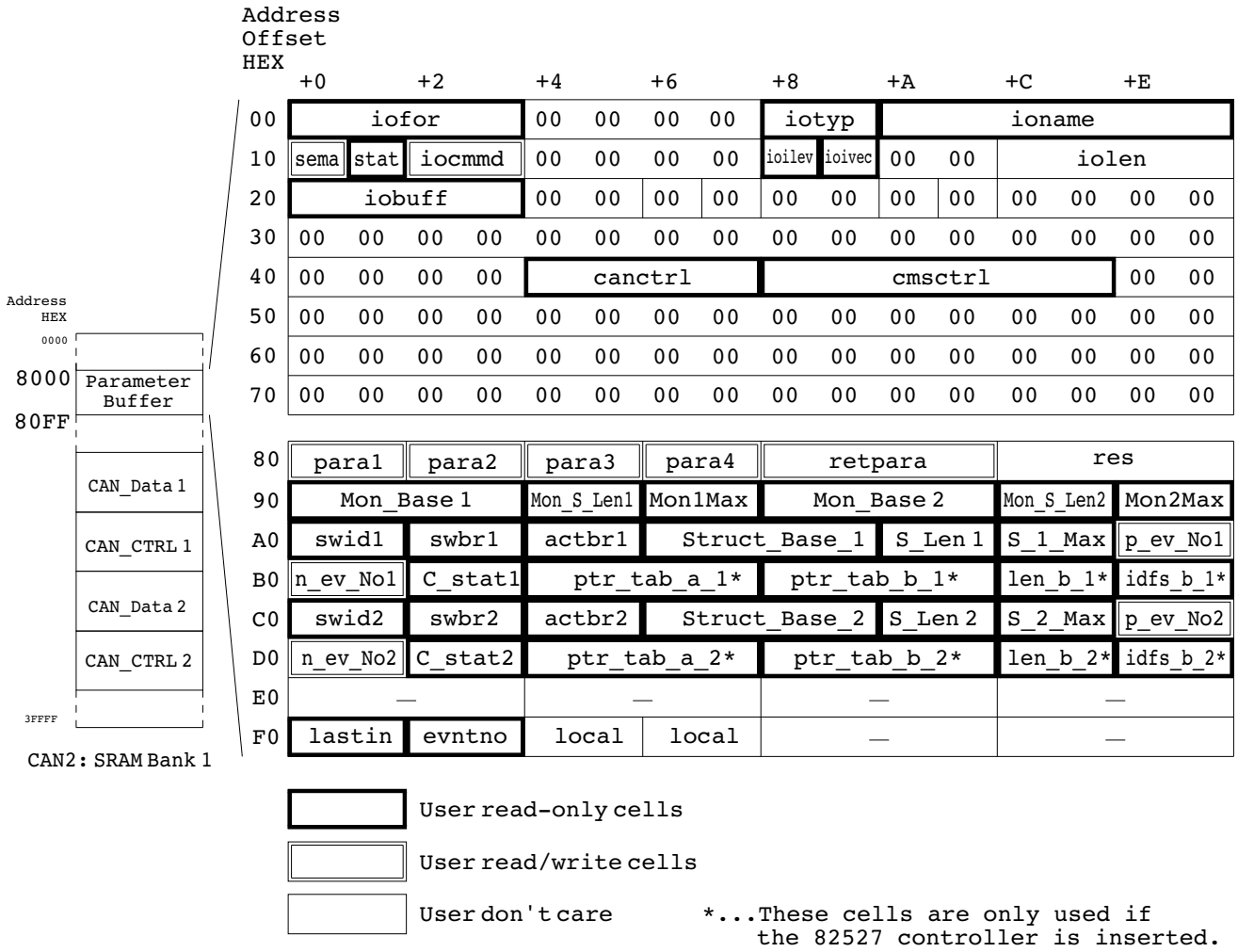


Fig. 4.1.1: Parameter buffer allocation

The figure on the following page shows the parameter buffer with its default settings after a RESET of the CAN2.

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	sema	stat	**FF**	**FF**	00	00	00	00	00	00	00	00	00	00	00	**80**
20	00	00	**80**	**80**	00	00	00	00	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'or'C527'				'___CMS'or'_NoCMS'						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	00	00	00	00
80	00	00	00	00	00	00	00	00	00	00	00	00	res			
90	00	**03**	00	**50**	00	**10**	**10**	00	00	**04**	00	**50**	00	**10**	**10**	00
A0	swid1		swbr1		actbr1		00	**01**	00	00	00	**10**	**08**	00	00	00
B0	00	00	00	**0C**	00	00	00	00	00	00	00	00	00	00	00	00
C0	swid2		swbr2		actbr2		00	**02**	00	00	00	**10**	**08**	00	00	00
D0	00	00	00	**0C**	00	00	00	00	00	00	00	00	00	00	00	00
E0	—				—				—				—			
F0	00	00	00	00	00	00	00	00	—				—			

User read-only cells

User read/write cells

User don't care

Fig. 4.1.2: Parameter buffer allocation

4.2 Transfer of Parameters and Commands

Setting the parameters and commands of a VMEbus-MASTER CPU to the CAN2 has to occur in the following order:

1. - evaluating the cell 'sema'--> is parameter buffer free?
   - if free, then assigning the parameter buffer by setting the cell 'sema' ('TAS'-command)

2. - Setting the parameters and commands by writing into the desired cells.

3. - Transfer of the parameters and commands to the CAN2-slave server by triggering the local 'GIRL'-interrupt (see also chapter 'Firmware-Defined Interrupts').

The GIRL interrupt is triggered with a word access by writing the date $0000 onto the relative I/O-address $00002. The address assignment of the I/O-area and a description of all GIRL interrupts can be obtained from the hardware manual of the CAN2.

4. - Wait for end of command execution: 'Cmmd' = FFFF

5. - Evaluating the cell 'stat':
   00 --> OK
   FF --> ERROR

6. - Enable the cell 'sema'.

4.3 Description of the Parameters

The parameters can be distinguished into 'only readable' and 'write- and readable' parameters. In the figure 4.1.1 above, these various parameter types are marked by different lines.

Apart from the parametration of the CAN2 it is also possible to transfer various commands by the parameter buffer. The command transfer will be described in the chapter 'Command Transfer by the Parameter Channel'.

The description of the parameters occurs in following form below:

'parameter'..... (relative local address, length of the parameter) comments to the parameter

4.3.1 Write- and Readable Parameters

'sema'..... ($10, byte) is assigned with the semaphore:

bit 7 - semaphore : '0' -> channel is free
'1' -> channel is assigned
bit 6..0 - reserved

'iocmmd'..... ($12, word) is only needed at the command transfer (see following chapter 'Commands of the Parameter Buffer'). Default value after RESET: $FFFF (= 'last command executed')

'para1'..... ($80, word)
to
'para4'..... ($86, word) are the parameters needed for the command transfer (see following chapter 'Commands of the Parameter Buffer'). Default value after RESET: $0000

'retpara'..... ($88, long word) contains return parameters after execution of commands. The return parameters are dependend on the selected command.
Exception: At the implementation of the software for the 82527, retpara is also used for the setting of parameters by the user.

'p_ev_No1'.....
'p_ev_No2'..... ($AE, word),
($CE, word) contains information about the evaluation of the changing edges of the 'EMERGENCY-ON' signal (EMERGRNCY-STOP INPUTS):
If the contents of these cells is unequal '0', a VMEbus interrupt is triggered at the 'L' ->'H' - edge of the 'EMERGENCY-ON' signal and the ev-no. (EVENT number) is stored in the cell 'evntno'.

'n_ev_No1'.....
'n_ev_No2'..... ($B0, word),
($D0, word) contains information about the evaluation of the changing edges of the 'EMERGENCY-ON' signal (EMERGENCY-STOP inputs):
If the content of these cells is unequal '0', a VMEbus interrupt is triggered at the 'H' ->'L' - edge of the 'EMERGNECY-ON' signal and the ev-no. (EVENT number) is stored in the cell 'evntno'.

4.3.2 'Read-Only Parameters'

'iofor'..... ($00, long word) pointer to the next similar buffer (here only one buffer, hence 'iofor'=parameter-buffer address [$8000])

'iotyp'..... ($08, word) buffer identifier (always $000C)

'ioname'..... ($0A, 6 bytes ASCII) contains the buffer identifier as ASCII string and a continuous numbering (always 'CANP ').

'stat'..... ($11, byte) contains information about the correct transfer of a command (see also following chapter 'Commands of the Parameter Buffer')

'stat' = $00 --> no error
'stat' P $00 --> error

The command transfer should principally occur in following order:
1. Assign 'sema'
2. Transfer parameters and commands
3. Trigger IRQ (GIRL interrupt)
3. Wait for 'Cmmd' = FFFF
5. Poll 'stat':
   = 00 --> OK
   P 00 --> ERROR
6. Enable 'sema'

'ioilev'..... ($18, byte) contains the value of 'iolev' (para1), set by the command 'card-interrupt enable'

'ioivec'..... ($19, byte) contains the value of 'iovec' (para2), set by the command 'card-interrupt enable'

'iolen'..... ($1C, long word) shows the length of the data area of the parameter buffer.
Default: 'iolen' = $80 == 128 bytes.

'iobuff'..... ($20, long word) is the pointer to the data area allocated to the buffer. As default 'iobuff' points to the cell 'para 1'.

'canctrl'..... ($44, 4 bytes) returns the type of used CAN controller as ASCII code:
'C200' -> 82C200
'C527' -> 82527

'cmsctrl'..... ($48, 6 bytes) shows as ASCII code, whether the CMS-protocol is implemented on the board:
'___CMS' -> CMS-protocol is implemented
'_NoCMS' -> CMS-protocol is not implemented

'Mon_Base1'.... ($90, long word)
'Mon_Base2'.... ($98, long word) contains the local address of the monitor buffers.
Default: 'Mon_Base1' - $30050
'Mon_Base2' - $40050

'Mon_S_Len1'.... ($94, word),
'Mon_S_Len2'.... ($9C, word) contains the length of a monitor-structural component.
Default: 'Mon_S_Len1' - $10 (16 bytes)
'Mon_S_Len2' - $10 (16 bytes)

'Mon1Max'..... ($96, word),
'Mon2Max'..... ($9C, word) contains the maximum number of structural components in the monitor buffer.
Default: 'Mon1Max' - $1000 (DEZ 4096)
'Mon1Max' - $1000 (DEZ 4096)

'swid1'..... ($A0, word),
'swid2'..... ($C0, word) contains the value of the CAN-net no. ($0000...$000F) set by the coding switches (SW1 and SW3) in the front panel after a RESET.
physical channel 1 - coding switch SW1 - 'swid1'
physical channel 2 - coding switch SW3 - 'swid2'

The allocation of the CAN-net no. to the 'physical' channels 1 and 2, named in the hardware description, is freely selectable.

'swbr1'..... ($A2, word),
'swbr2'..... ($C2, word) contains the default-bitrate indices set by the coding switches (SW2 and SW4) on the board after a RESET. The bitrate indices correspond to the contents of the registers BTR0 and BTR1 of the controller 82C200.

Channel 1 - coding switch SW2 - 'swbr1'
Channel 2 - coding switch SW4 - 'swbr2'

It is possible to change the bitrate indices by a command transfer of the parameter channel (see chapter 'Command Transfer by the Parameter Channel'). The actual bitrate indices are stored in the cells 'actbr1' and 'actbr2'.

The table below shows the setting of the bitrate by the coding switches SW2 and SW4 with indication of the register content (bitrate index) of the controller 82C200/82527 and each physically maximum possible line length.

SW2, SW4 [HEX]	Bitrate [kBit/s]	82C200 or 82527 register		typical values of the achievable line length l_{max} [m]	minimum achievable line length l_{min} [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 the limit values of the bit timing of the CAN protocoll, the delay times of the local esd-CAN interface and the delay times of the cable. The cable-running time is assumed with around 5.5 ns/m. Further influences, e.g., the terminal resistances, the specific resistance, the cable geometry or external interferences have not been considered in the specifications!

Table 4.3.1: Setting the bitrate

'actbr1'..... ($A4, word),
'actrb2'..... ($C4, word) contains the actual bitrate index ($0011...$7F7F).
The bitrate index can be modified by a command transfer (see also chapter 'Command Transfer by the Parameter Channel')
Channel 1 - 'actbr1'
Channel 2 - 'actbr2'

'Struct_Base1'.. ($A6, long word),
'Struct_Base2'. ($C6, long word) contains the local base address of the CAN-data structure(CAN_Data1, CAN_Data2). The base address of the monitor structure (CAN_CTRL1, CAN_CTRL2) is always 'Struct_Base'+ $8000.

'S_Len 1'..... ($AA, word),
'S_Len 2'..... ($CA, word) contains the length of a data-structural component:
Default: 'S_Len 1' = $0010 (16byte)
'S_Len 2' = $0010 (16byte)

'S_1_Max'..... ($AC, word),
'S_2_Max'..... ($6C, word) contains the number of data-structural components of the data-strucural fields (CAN_Data1, CAN_Data2).
Default: 'S_1_Max' = $0800 (DEZ 2048)
'S_2_Max' = $0800 (DEZ 2048)

'C_stat1'..... ($B2, word),
'C_stat2'..... ($D2, word) contains in bit 0...7 the status of the CAN-controller component 82C200 or 82527 according to the corresponding manual.

'ptr_tab_a_1*'.. ($B4, long word),
'ptr_tab_a_2*'.. ($D4, long word), this cell is only assigned when the controller 82527 is used.
It contains the pointer to the initial address of the assignment table (table a) between handle and identifier. The table contains 2048 elements (handles). To each handle an identifier and an identifier mode is allocated. The allocation occurs by a command of the parameter buffer ('iocmmd' = $010).

'ptr_tab_b_1*'.. ($B8, long word),
'ptr_tab_b_2*'.. ($D8, long word), this cell is only assigned when the controller 82527 is used.
It contains the pointer to a second table (table b) in which the desired identifiers can be stored in an order by the firmware, that they can be searched as quick as possible at the reception of an Rx-frame, to find out whether this Rx-frame had been determined for the CAN2.0B.

'len_b_1*'.. ($BC, word),
'len_b_2*'.. ($DC, word) this cell is only assigned when the controller 82527 is used.
In this cell the length of the table (table b) described above is entered. The length is calculated in the following way: Length = 2^{n-1} , where the maximum lies at 2047 entries.

'idfs_b_1*'.. ($BE, word),
'idfs_b_2*'.. ($DE, word) this cell is only assigned when the controller 82527 is used.
It contains the number of the identifiers contained in the table (table b) described above.

'Lastin'..... ($F0, word) contains the actual level of the EMERGENCY-ON inputs. Bit 0 represents the level of the EMERGENCY-ON input of channel 1 ('0' or '1') and bit 1 represents the level of channel 2. This cell is evaluated if the cells 'p_ev_No..' or 'n_ev_No...' have been set.

'local'..... ($F4, $F6, word) is only assigned by the firmware.

'evntno'..... ($F2, word) contains the actual EVENT of the 'EMERGENCY-ON' signal.

4.4 Commands of the Parameter Buffer

4.4.1 Overview of the Implemented Commands

The procedure of the command transfer has already been described in the chapter 'Transfer of Parameters and Commands'.

The cell 'stat' in the parameter buffer serves the acknowledgement of the command transfer:

```
'stat' = $00 --> no error at command transfer
'stat' P $00 --> error at transfer
```

Into the cell 'iocmmd' the desired command is entered. The cells 'para1' to 'para4' take up the parameters which are necessary for the corresponding command.

At the moment 14 different commands are implemented (at the use of the controller 82527 the commands 'insert_idf' and 'release_idf' are added):

Cell of parameter buffer					Command
iocmmd	**para1**	**para2**	**para3**	**para4**	
0	bitrate $0000..$000F or $0011..$7F7F	-	-	-	Init CAN Net 1 with bitrate
1	bitrate $0000..$000F or $0011..$7F7F	-	-	-	Init CAN Net 2 with bitrate
4	Net 0,1	M_code1	M_mask1	-	Trigger monitor 1 or monitor 2
6	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
7	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
8	Net 0,1	Idf 0...2047	-	-	Activate periodical Tx-transfers (or CMS Watchdog TxId)
9	Net 0,1	Idf 0...2047	-	-	Stop periodical repeat of periodical Tx-transfers (or CMS RTR)
A	iolev $0001..$0007	iovec $0000..$00FC	-	-	CARD Interrupt enable
B	Net 0,1	Idf_start 0...2047	Idf_end 0...2047	Mode	Set Mode
C	Net 0,1	Client TxId 0...2047	Client RxId 0...2047	Buffer length 1...32 kBytes	Init Domain (only CMS tranfers)
D	Net 0,1	Client TxId 0...2047	Client RxId 0...2047	-	Release CMS buffer
E	Handle-No.	Buffer length	-	-	**retpara** Buffer address Get Rx buffer
F	Handle-No.	-	-	-	Release Rx buffer
10	Net 0,1	Handle 0...2047	ext_mode 0,1	dyn_mode 0,1,2,3	**retpara** idf29/11 0...2947 or $0...2^{29}-1$ only with 82527 inserted: Insert Identifier
11	Net 0,1	Handle 0...2047	-	-	only with 82527 inserted: Release Identifier
FFFF	-	-	-	-	'last command done' (Default setting after RESET)

Table 4.4.2: Commands of the parameter buffer

4.4.2 Description of the Individual Commands

In the following the individual commands and the corresponding parameters will be described in rising order.

4.4.2.1 Setting the Bitrate (Init CAN_x with Bitrate)

The bitrate of the CAN-channel is set again. The CAN-channels work with this new bitrate - the bitrate preset by the coding switches is ignored.
After a RESET the system works with the bitrates of the coding switches again. A table with the allocation of the parameter 'bitrate' to the bitrate can be found at the description of the buffer cells 'actbr1/2'.

The transfer of the bitrate can occur with up to now customary values '0....F', which have to be equated with the positions of the coding switches, or it can occur directly by entering the 16 bit wide value of the registers BTR0 + BTR1 of the CAN-controller.

During running (incompleted) transmissions the modification of the bitrate would lead to errors in all likelihood. This is why the processing of all structural components which are in standby condition are aborted with the corresponding error message.

4.4.2.2 Triggering the Monitor Function (Trigger Monitor x)

This command gives the start condition for the recording of the received data in the monitor buffer. After the trigger condition occurred, all received data for whose identifiers the monitor mode had been selected are stored with their identifier and their 'time of reception' (after the first permitted strucural component) into the monitor buffer.

The trigger condition is the reception of a structural component whose features are within the value range determined by the parameters 'M_codex' and 'M_maskx' (x=1,2).

The selection of the received messages by these parameters takes place by the same bits which are stored in the cell 'Idf' in the monitor-structural component.

Bits of the parameters 'M maskx' and 'M codex' (X=1,2)	15...5	4	3...0
Function	11bit-CAN-Id	RTR	LENGTH

Table 4.4.3: Function of the parameter 'M_codex'

The trigger condition is fulfilled when the bits of the received messages comply with following condition:

```
                                                        !
[Input(CAN-Id., RTR, LENGTH) EXOR M_code] AND M_mask = $0000
```

The parameter 'M_maskx' (x=1,2) determines which bits of the identifier should be examined for their logical condition.

The parameter 'M_codex' (x=1,2) determines which logical state the identifier bits must have got so that the reception of this identifier is accepted as trigger condition for the monitor mode. Only those bits are evaluated which have been selected in the parameter 'M_maskx' for the evaluation.

Applied to a value table this means:

M mask	M code	Input bit	RESULT	Evaluation
0 0	0 0	0 1	0 0	Input bit = don't care
0 0	1 1	0 1	0 0	Input bit = don't care
1 1	**0** 0	**0** 1	0 1	Bit = OK Bit = false
1 1	1 **1**	0 **1**	1 0	Bit = false Bit = OK

Table 4.4.4: Function of the parameter 'M_maskx'

If a bit of the cell 'M_mask' = '0', the corresponding bit of the received identifier is not evaluated, that means when, e.g., all bits of 'M_maskx' (x=1,2) are 'nil', all received messages are evaluated as monitor triggers - no selections occur.

If a bit of the cell 'M_mask' = '1', the bit of the received identifier has to have the same value as the corresponding bit in the cell 'M_codex' to be accepted as trigger condition.

Example:
M_mask1 = 0000 000**1** **1**000 0000
M_code1 = xxxx xxx**1** **0**xxx xxxx
---
permitted = xxxx xxx**1** **0**xxx xxxx
identifiers

x= don't care

The recording stops when the monitor buffer is full. After a new start of the trigger, the old data are overwritten.

4.4.2.3 Interconnection of CAN-Identifiers of Different Nets
(Link Net_A/Idf_A to Net_B/Idf_B)

By this command the CAN-net_A is connected bidirectionally to the CAN-net_B (Idf_B) (or the connection is removed again resp.). I.e., all data received with the identifier 'A' on Net_A are transmitted to Net_B with the identifier 'B', and all data received on Net_B are transmitted to Net_A.

Net_A as well as Net_B can be allocated to the 'physical' CAN-channels 1 or 2:

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

Channel 1 -->'0'
Channel 2 -->'1'

Apart from the single forwarding of CAN-frames from one net to the other it is also possible to transmit and receive message on the nets. Doing this, the two identifiers of the nets are seen as one identifier, i.e., if, e.g., a message is transmitted on identifier A/Net A, it will be transmitted on identifier B/Net B, parallel.

4.4.2.4 **Periodical Transfer of Tx-Messages** (Activate Periodical Tx-Transfers or CMS-Watchdog)

This command serves the periodical repetition of Tx-transfers or the resetting of this mode at the use of the standard EPROMs without CMS implementation.

CMS Implementation
If the used EPROMs are equipped with a CMS implementation, the function 'periodical transmission' becomes inapplicable and instead the CMS-watchdog-Tx-identifier and the net number are transmitted by this command.
The watchdog identifier is transferred in the cell 'para2' and the net number in the cell 'para1'.

Initialization
The initialization of this mode occurs by the transfer of the parameters 'iocmmd', 'Net' and 'Idf' to the parameter buffer.

Activation
Setting the repetition period by the cell 'TOUT' in the data-structural component of the identifier (negative values!; see chapter 'Contents of a Data-Structural Component') activates the mode. The entered value shows the time between the completion of the last Tx-transfer and the start of the new transfer.

The period length of the Tx-frames on the CAN, entered by 'TOUT', has got a temporal insecurity of ±1 ms.

Start
To start the periodical transfer it is necessary to transmit a Tx-transfer of the structural component once successfully. Each further Tx-transfer on this identifier starts the mode again.

End
The periodical Tx-transfers are completed by resetting the command in the parameter buffer or setting the time-out period to positive values.

4.4.2.5 **Enable of the Board Interrupts by iovec and iolev** (CARD-Interrupt Enable)

This command initializes the PIT 68230 to the creation of VMEbus interrupts. The command replaces the setting of individual PIT-interrupt registers by the user, necessary in older firmware versions. (The direct setting of the PIT registers is still possible, but will not be described here, anymore).

If the two cells 'iolev' and 'iovec' are equal to '0', no VMEbus interrupt will be triggered (default value after RESET).

The user can allocate the values 1...7 to parameter 'iolev'. These values determine the corresponding VMEbus-interrupt level 1...7.

The parameter 'iovec' determines the most significant six bits of the interrupt vector which should be applied to the VMEbus (vector base). The least significant two bits (IRQS1, IRQS0) are set by the hardware corresponding to the local interrupt source. They determine the 'interrupt-acknowledge bit' which has to be set in the PIT 68230 to take back the VMEbus interrupt.

Bit	7	6	5	4	3	2	1	0
Function	vector base (user defined)						IRQS1	IRQS0

Table 4.4.5: Function of the parameter 'iovec'

Explanation of the bits in the cell 'iolvec':

Vector base....These bits can be freely programmed by the user.

Bit IRQS1	IRQS0	Interrupt source
0	0	EMY-ON signal (changing edges)
0	1	Command interface (not used)
1	0	FIFO (FIFO 'data from VMEbus' full by VMEbus)
1	1	CAN-server (FIFO 'data to VMEbus' not empty)

Table 4.4.6: Function of the bits IRQS0 and IRQS1

The contents of the cells 'iolev' and 'iovec' is stored by the firmware in the parameter buffer in the cells 'ioilev' and 'ioivec'. The programmed values can be read there directly.

4.4.2.6 Select Operating Mode (Set Mode)

By this command the mode for identifier groups or individual identifiers is set. If the mode should only be set for one identifier, the same value is entered into the cells 'Idf_start' and 'Idf_end'.

The command ‘Set Mode’ is called via the parameter buffer, by setting the cell iocmmd to the value $B. The cells para1...para4 have to be set as described in the previous chapter ‘Overview of the Implemented Commands’. The value for MODE must be set in the cell para4.
The following table gives a short overview of the implemented modes:

MODE	selected transmission mode
0000	Rx-transfers disabled (Tx-only)
0001	depending on the entry in cell LENGTH in the data-structural component: Rx-transfer enabled Transmit RTR
0002	Activate Tx-transfer at RTR-reception
0004	Monitor
0005	CAN-serial
000F	leads to the activation of the previous mode
80xy	Rx-buffer: handle allocation xy = 00...FF

Table 4.4.7: Overview of the implemented modes

Description of the MODEs:

The transmission of Tx-frames is generally, independent from the modes described here, always possible!

Rx-transfer... disabled
(Mode 0)

In this operating mode only data (0...8 bytes) can be transmitted onto the CAN.
In the cell 'TOUT' in the data-structural component, a time-out value can be entered.

The user can be informed about the end of a transmission by an interrupt, if the cells 'iolev' and Iovec' of the command 'CARD-interrupt enable' in the parameter buffer are set and the cell 'EVTRIG' in the data-structural component is ≠ '0' or the cell 'XTTID' in the control-structural component is ≠ '0'.

Valid end conditions which lead to the triggering of the VMEbus interrupt in this case:
- Tx-transfer completed successfully
- Tx-transfer aborted because of error (e.g. time out)

For the entry into the cell 'LENGTH' there are the following alternatives in this mode:

1. Entry of the length as a positive number (0000...0008): The entry of the number of bytes as positive number does not lead to the start of the execution of the transfer instruction.
2. Entry of the length as a negative number (FFFF...FFF7): If the number of the bytes to be transmitted is entered as negative HEX-number, execution of the transfer instruction is started by the entry.
3. Entry of the length and the number '6' (0060...0068): Function identical to 2.

Rx-transfers...
(Mode 1)

The user can read the CAN-data in the data-structural component transparently in this mode.

Valid for the Rx-mode:
The user can be informed about the data reception by the VMEbus by an interrupt, if the cells 'iolev' and 'iovec' of the command 'CARD-interrupt enable' in the parameter buffer have been set and the cell 'EVTRIG' in the data-structural component is ≠ '0' or the cell 'XTTID' in the control-structural component is ≠ '0'.

Valid end conditions which lead to the triggering of the VMEbus interrupt in this case:
- Rx-transfer completed successfully

For the Rx-tranfers it is also possible state a **time-out value.** The time-out value is entered into the cell 'TOUT' in the data-structural component. Within this period a received message is expected on this identifier.

Setting the cell 'LENGTH' integrates the CRTL-structural component of this identifier into a time-out chain in which the 'time counters' of the entered structural components are counted down.
The bit 'WAITR' (wait for Rx) in the status byte of the CTRL-structural component is set to '1' as long as no message has been received or a transmission error occurs.
After the reception of valid data the cell 'LENGTH' is set.

Valid end conditions which lead to the triggering of the VMEbus interrupt in this case:
- Rx-transfer completed successfully
- Rx-transfer aborted because of time out

RTR transmit...
(Mode 1)

In this mode an RTR-frame is transmitted on the corresponding identifier after setting the cell 'LENGTH' in the specified value range. The RTR-frame should initiate the return of data. The entry of the length, which has got no meaning at the CAN-protocol, is also transmitted, here.

The transmission of the RTR-frame occurs similar to the MODE '0'. The selection of the 'time-out mode' and the triggering of an interrupt after a valid end condition are possible.

The end of the RTR-transmission procedure is shown in the 'STATUS' byte in the data-structural component by the value '$FFFE'.

After successful transmission of the RTR-frame, the reception of messages on this identifier is expected. From that moment this mode can be seen as an 'Rx-transfer with time out'.

Valid end conditions which lead to the triggering of a VMEbus interrupt in this case:
- Rx-transfer completed successfully
- Rx-transfer interrupted because of time out

Activate... Tx-transfer at RTR-Rx (Mode 2)

At the reception of an RTR-frame this data-structural component (0...(byte) is transmitted. Principally the transmission occurs like in MODE '0'.

Differences arise from the interrupt generation and the source of the data which should be transmitted:

If the cell 'EVTRIG' is set ≠'0', a VMEbus interrupt is triggered after the reception of the RTR-frame and the data to be transmitted have to be provided by the VMEbus. (For the VMEbus-IRQ the cells 'ioev' and 'iovec' of the command 'CARD-interrupt enable' have to be set in the parameter buffer!)

If the cell 'EVTRIG' is set = '0', the data to be transmitted are transferred from the local data-structural components of the CAN2.

Monitor... (Mode 4)

If messages are received on this identifier in the monitor mode, they will not be stored in the assigned data-structural component, but stored in the order of their chronological reception in the monitor buffer.

This mode serves the supervision of transmissions of selected CAN-identifiers. In the monitor buffer not only data of <u>one</u> identifier are stored, but all transmissions which are configurated for this mode in their structural component and which are received after trigger condition of the monitor buffer.

The recording is completed when the monitor buffer is full. The end of the recording cannot be indicated by a VMEbus interrupt.

Further information about the monitor buffer can be taken from the preceding chapter 'Monitor Buffer' and the description of the command 'trigger monitor 1 or monitor 2'.

CAN-serial... (Mode 5)

Received data are not only stored in the structural component, but also in the FIFO to enable the VMEbus master to continuously process received data in their chronological order by an interrupt cycle.

Further information about the CAN-serial mode can also be taken from the preceding chapter with the same title.

Activating the previous mode... (Mode F)

If the value '$000F' is entered for mode, the previous mode is activated again.
Example: At first the mode '00' had been selected, Rx-transfers disabled, therefore. Then it was switched to monitor mode ('04'). If the mode is set to '$000F' now, the mode '00' would be activated again.

Rx-buffer: handle assignment... (Mode 80xy)	At the use of the Rx-buffer, buffer sizes are assigned to various 'handle numbers'. To the identifiers which should be monitored the handle numbers are allocated by these mode assigments in turn. Further information can be taken from the description of the command 'E' (get Rx-buffer).

4.4.2.7 **CMS-Initialisation** (Init Domain)

For the upload or download via the CMS protocol, the local firmware needs a memory area (CMS buffer) to transfer parameters and store data.

By the command 'Init Domain' the CMS buffer is started and the Rx- and Tx-identifiers by which the transfer should occur are transferred.
The transferred buffer length relates to the size of the CMS-data buffer which should be provided. Values between one byte and 32 kbytes can be selected for the size.

After executing the command the absolute initial address of the CMS-buffer can be read in the return parameter of the parameter buffer 'retpar' or in the cell 'NET_LINK/BUFFER-POINTER' in the control-structural component of the used identifiers.

4.4.2.8 **Releasing the CMS-Buffer** (Release Buffer)

If the CMS-buffer is not needed anymore, the memory area can be released again. As parameters only the identifiers and the net number are transferred, here.

4.4.2.9 **Rx-Buffers**

The fundamental functions of the Rx-buffers have already been described in the preceding chapter with the same title.

Before the assignment of the identifiers the command 'Get Rx-Buffer' has to be requested. The local firmware then starts an Rx-buffer. Each Rx-buffer gets a so-called handle no. at the call of the command 'Get Rx-Buffer'.

Then the identifiers are selected which should be recorded in the buffer. The assignment between identifiers and buffers occurs by the handle numbers: It is possible to assign handle numbers to individual identifiers or identifier blocks via the mode assignment.

Identifiers with the same handle no. are recorded in the a common Rx-buffer, therefore.

Recording of the identifiers in the Rx-buffer starts with the call of this command. After the command call the absolute initial address of the Rx-buffer can be read in the return parameter of the parameter buffer 'retpar' (address of the parameters see previous chapter ‘Function and Structure’.

By the command 'release Rx-buffer' the memory area is released again.

4.4.2.10 Insert/Release Identifier (only with CAN-Controller 82527)

These two commands are only implemented, if the CAN-controller 82527 is inserted. If the 82C200 is used, these commands cannot be executed. Which one of the two controllers is available can be read in the cell 'canctrl' in the parameter buffer (ASCII-coded long word on address $8044).

If the controller 82527 is available, the differing specifications of this controller and the firmware can be taken from the attached pages 'Features of the Controller 82527 Implementation on the CAN2.0B'.

By the command 'insert identifier' it is possible to select a total of 2047 different identifiers for the process by the CAN20B. The identifiers can be 11 bit or 29 bit long. The desired identifier is transferred in the four bytes of the cell of the parameter 'retpara', which is normally only designed for return parameters.

To each identifier a *handle* is assigned (not to be confused with the handle no. of the Rx-buffer!). The handle writes on the memory location in a table in which entered identifiers are stored together with the parameters 'ext_mode' and 'dyn_mode'.

In the parameter *ext_mode* (extended/standard mode) it is determined, if the identifier is 11 bit or 29 bit long:

```
ext_mode = 0            ->    11 bit identifier
ext_mode = 1            ->    29 bit identifier
```

In the parameter *dyn_mode* (dynamic/resident mode) it is determined, if the identifier should be managed dynamically within the 14 objects or be assigned resident to an object. To which object the identifier should be assigned is determined by the firmware in both cases.
To define Rx-identifiers as resident is advisable, if it is known that these identifiers are frequntly accessed, because the local firmware is relieved in this way. It is strictly necessary to define a Tx-identifier as resident, if it should react to an RTR-frame. Dynamic Tx-identifiers do not react to received RTRs!

```
dyn_mode =  0     ->    dynamic Rx-identifier
            1     ->    dynamic Tx-identifier
            2     ->    resident Rx-identifier
            3     ->    resident Tx-identifier
```

Attention! The dyn-mode '0' (dynamic Rx-identifier) makes an exception: All identifiers selected for this mode must have the same length! Each last transferred entry for 'ext_mode' is valid for all identifiers of the dyn_mode '0'.
For all other modes it is possible to select different lengths (11 or 29 bits) for identifiers to which the same mode had been assigned.

5. Summary of the Interrupt Options

5.1 Overview

This chapter should give a summarizing overview of the interrupt generation and management with respect to the two CAN-channels on the CAN2. Interrupts which's function relates to different assemblies of the CAN2 are not stated here.

5.2 User-Defined Interrupts/Interrupt Handling

5.2.1 General/PIT 68230

The user can trigger an interrupt on the VMEbus at various events.

The first condition for the release of an interrupt on the VMEbus is the initialization of the interrupt registers. In these registers the user has to enter the desired interrupt level and interrupt vector. The transfer of the desired register contents is performed by the command 'CARD-Interrupt enable' of the parameter buffer.

Into the cell 'iolev' the level of the desired interrupt is entered and into the cell 'iovec' the most significant six bits of the interrupt vector which should be transferred onto the VMEbus in the interrupt-acknowledge cycle.

The bits 2 to 7 of the interrupt vector of the cell 'iovec' can be freely programmed by the user. The least significant two bits (IRQS0 and IRQS1) contain information about the interrupt source. Mainly it is distinguished between the 'EMY-ON interrupt', the 'FIFO-VME-to-CAN-full-interrupt' and the 'CAN-server interrupt'. The 'CAN-server interrupt' can be triggered at a valid 'end condition' of a CAN-identifier, in the CAN_serial mode or at CMS-transfers (if implemented).

The cells 'iolev' and 'iovec' have already been described in the chapter 'Commands of the Parameter Buffer'.

The initialisation of the PIT registers on the CAN2 has to be carried out after each RESET of the VMEbus master.

The user has to secure that the VMEbus interrupt is intercepted by an interrupt routine, if the user selected the interrupt option!

To clarify the resetting of the interrupt, the connections to the hardware should be mentioned shortly, here:

The VMEbus interrupt is generated by the controller PIT 68230. By means of the PIT, also the interrupt vector is generated. The VMEbus interrupt is taken back, when the corresponding PIT interrupt is reset.

The interrupts of the PIT are reset by setting the respective bit of the PSR (port status register), determined by the interrupt vector, of the PIT to '1'.

iovec IRQS1	IRQS0	Interrupt source	Taking back the interrupt cause
0	0	EMY-ON signal (changing edges)	set bit 0 of the PSR to '1'
0	1	command interface (not used)	-
1	0	FIFO (FIFO 'data from VMEbus' full by VMEbus)	1. wait until bit 6 of thr PSR of the PIT 68230='1'(FIFO free) 2. set bit 2 of the PSR to '1'
1	1	CAN-server (valid end condition occured, CAN-serial or CMS-transfer)	1. set bit 3 of the PSR to '1' 2. read the FIFO 'Data to VMEbus' word-for-word on relative I/O-address $01602

Table 5.2.1: Resetting the VMEbus interrupt by PSR of the PIT 68230

Example: Resetting the FIFO 'VME-to-CAN-full' interrupt

```
---   Interrupt entry   ---
MOVEB=$04,iack+can2base;      only this interrupt!
---   further interrupt routine ---

'iack'.......interrupt-acknowledge address (incl. offset of the I/O-area) of the
             PIT 68230-PSR: $0601B + $78000
'can2base'...basis address of the CAN2 board
```

Setting the PSR takes only back the VMEbus interrupt, but not the local interrupt cause. Taking back the interrupt cause will be described in the following at the individual interrupt sources.

5.2.2 'EMY-ON' Interrupt

The 'EMY-ON interrupt' tells the user that the level of the EMY-On signal has changed. This interrupt is only created when the user set the cells 'iolev' and 'iovec' and wrote its event number into one of the cells 'p_ev_No..' or 'n_ev_No...' of the parameter buffer. All cells cited above are described in the chapter 'The Parameter Buffer'.

The interrupt of the 'EMY-ON' inputs is not triggered by the hardware. The local firmware polls the conditions of these inputs periodically and detects if the level of the inputs has changed. The maximum period between two pollings is 10 msec.

The VMEbus interrupt is taken back as described in section 'General'.

After an 'EMY-ON' interrupt, setting the PIT-PSR bits should occur as soon as possible, because triggering another interrupt is delayed on the CAN2 until the actual interrupt has been acknowledged.

5.2.3 FIFO 'VME-to-CAN-Full' Interrupt

Setting the cells 'iolev' and 'iovec' is sufficient for enabling this interrupt!

The 'FIFO-VME-to-CAN-full-Interrupt' is triggered by the hardware, when the FIFO 'data from VMEbus' has been filled via the VMEbus.

This FIFO serves as buffer between the VMEbus and the CAN during the data transmission (see also chapter 'Tx-Transfers'). If it should have been filled completely via the VMEbus, then this indicates that the local CAN server had not been able to handle the messages in the appropriate time.
This can happen, e.g., when the CAN is permanently busy with Tx-identifiers of higher priority and the CAN-controller gets no transfer admission for this reason.

The interrupt should tell the user that he has to reduce his data flow, if he does not want to risk a data loss.

This interrupt is only reset after a cell of FIFO became free again. The user is informed about this by the level of PSR bit 6 of the PIT 68230. The FIFO is full when bit 6 has got the value '0'. In his interrupt routine the user has to read the bit and wait until it takes over the value '1'. Now the user can take back the PIT-VMEbus interrupt as described in section 'General'. Only then the interrupt routine is left and the transmission of data to the CAN2 is resumed again.

5.2.4 CAN-Server Interrupt

The CAN-server interrupt can be triggered after a valid end condition for each single identifier. Valid end conditions are, e.g., the successful completion of a transmission or a transmission error (e.g. time-out). It can also be triggered at reception of data in the 'CAN-serial mode' or during the CMS services.

To be able to trigger a VMEbus interrupt, apart from the cells 'iovec' and 'iolev' the cell 'EVTRIG' (only for interrupt at end conditions) has to be set in the data-structural component of the identifier which should be monitored. The cell 'EVTRIG' is described in the chapter 'The Structural Fields CAN_Data1 and CAN_Data2'.

In the modes 'Rx-transfer disabled' (Tx-only), 'Rx-transfer with time-out option' and 'transmit RTR' it is possible to set the cell 'XTTID' instead of the cell 'EVTRIG'. The cell 'XTTID' is decribed in chapter 'The Monitor Structures CAN_CTRL1 and CAN_CTRL2'.

Setting the mode is described in chapter 'Parameter- and Command Transfer by the Parameter Buffer'.

The CAN-server interrupt is reset in the order as has already been described in the table above:

The user has to reset the PIT-VMEbus interrupt before reading the FIFO!

This FIFO must <u>each only be read once via the VMEbus and only within the access mode 'word'</u>! By each reading access the internal FIFO counter is incremented, that means with each further access another FIFO cell would be read and the FIFO counter would be incremented in an inadmissible way!

This order is strictly necessary, because further data could already be received during the reading, whose interrupts could not be evaluated then. If the interrupt is reset before that, new data can trigger another interrupt as soon as the master has left the interrupt routine.

By the FIFO that structural component is written on which triggered the interrupt. The end condition which lead to the triggering can be read in the cell STATUS of the data-structural component. In the description of this cell in the chapter 'The Structural Fields CAN_Data1 and CAN_Data2' the valid end conditions are listed.

The contents of the FIFO informs the user which identifier on which channel of the CAN2 triggered the interrupt. If the marks $FFFE or $FFFF are read, the interrupt had been triggered in the CAN-serial mode and the following cells contain a CAN-serial-data block (see chapter 'CAN-Serial-Mode').

Bit FIFO ->	15	14 13 ... 6 5 4	3 2 1 0
Idf-pointer ->	Channel no.	CAN-Id no. (11 bit)	0 0 0 0
Marks for CAN-serial mode	$FFFE -> node 1 $FFFF -> node 2		

Table 5.2.2: Contents of the FIFO 'data to VMEbus'

Explanation of the bits of the FIFO 'data to VMEbus'

Channel no... This bit shows the 'physical' CAN channel on whose identifier the interrupt has been triggered:
channel no. = '0' --> channel 1
channel no. = '1' --> channel 2

CAN-Id no. ... These bits show the identifier (and with this the structural component) which triggered the interrupt (0...2047).

Bit 3 to 0... These bits are always read as '0'.

$FFFE... Mark for the start of a CAN-serial-data block on channel 1 (node 0)
$FFFF... Mark for the start of a CAN-serial-data block on channel 2 (node 1)

5.2.5 Recognizing the End Condition Without Interrupt (Polling)

If no interrupt should be triggered at an end condition, the user, nevertheless, has got the possibility to read the FIFO 'data to VMEbus' to evaluate received CAN messages. By 'polling' bit 7 of the PIT 68230-status register (PSR) it is possible to find out, if data are contained in the FIFO.

The PSR-bit 7 of the PIT 68230 can be read by a byte access on the following address:

PIT 68230 port status register: $0601B

If bit 7 of the PSR has got the value '1', the FIFO does not contain any data, if the bit has got the value '0', the FIFO contains data and can be read out.

For reading the FIFO the same conditions apply which have already been described in the preceding chapter: It must each only be read once after in the PSR was indicated that the FIFO contains valid data!

The contents of the FIFO has already been described in the table above.

5.3 Firmware-Defined Interrupts

Under the designation 'firmware-defined' interrupts a short overview of the IRQs, which are normally processed in the background unnoticed by the user, should be given, here.

The accesses onto these interrupts by the user are limited.

Only the so-called 'GIRL' interrupt (**G**lobal Access -> **I**nte**r**rupt **L**ocal) is set by the user. By it the VMEbus user informs the local firmware that he changed cells in the parameter buffer and that the new parameters should be integrated into the program procedure. The use of this interrupt has already been described in the preceding chapter 'Transfer of Parameters and Commands'.

The interrupt is triggered by writing the date $0000 onto the relative I/O-address $00002 by a word access. The address assignment of the I/O-area and a description of all GIRL interrupts can be taken from the hardware manual of the CAN2 (for the parameter updating the firmware works only with the above mentioned interrupt!).

<u>Onto the following interrupts the user has got no influence</u>:

Other interrupts processed in the background have, e.g., already been mentioned in the chapter 'Mode of Operation of the Firmware at the Example of Single Rx- and Tx-Transfers'. These interrupts cannot be influenced by the user in any way.

The 'FIFO-IRQ' is set by the FIFO as soon as it is not empty anymore, i.e., the interrupt is always active as soon as there are structural-components in the FIFO. It is only reset, when the FIFO is clear again. The FIFO-IRQ informs the firmware that structural-components are ready for transmission.

The 'Tx-IRQ' and the 'Rx-IRQ' of the CAN-controller 82C200 signalize the controller being ready for transmission or reception. They are only evaluated by the firmware. The user is able to read the status of these interrupts indirectly by reading the cell 'STATUS' in the data-structural component and the cell 'STAT' in the control-structural component.

6. Introduction to Configuration and Operation

The programming of the CAN2 can be divided into two sections:

1. Fundamental initialisation
2. Parametration of the individual Ids in the data-structural component
3. Operation of the CAN2 (possibly with further modifications of the initialisation)

Some parameters of the CAN2 have to be set once only, normally, after a RESET of the system. To these belong e.g., the bitrate adjustment or the interrupt initialisation or the MODE allocation for identifiers, too, whose operating mode does not have to be changed during the system run time, anymore.

The following table should offer an introduction to the configuration. The specifications of the table should not be interpreted as fixed rules, but as suggestion.

Fundamental Initialisation	
1.	Set bitrate, if the default bitrate is not kept (-> parameter buffer)
2.	if desired: initialize VMEbus interrupt The interrupt vector and the interrupt level are set with 'iovec' and 'iolev' by the command 'CARD-interrupt enable' in the parameter buffer. Do not forget: The VMEbus interrupt has to be supplied by the user by an according interrupt routine!
2a.	only at implementation of the software for 82527: Entry of the desired identifiers and their parameters by the command of the parameter buffer 'Insert Identifier'
3.	Determining the identifiers operating mode: For the Rx-identifiers of the CAN which should not be received the MODE '0' should be selected by the command 'set mode' of the parameter buffer. These identifiers are not stored into the data-strucural field, in this way. This saves computing capacity and relieves the system. Tx-transfers are still possible on these identifiers. The MODE of the identifiers can also be changed during operation. After a RESET, MODE is set to '01' for all identifiers.
4.	If identifiers of the two CAN nets should be connected, this can occur by the commands of the parameter buffer.
5.	if desired: Request command 'activate periodical Tx-transfer' in the parameter buffer for the corresponding identifier.
6.	if desired: Determine interrupt options for the EMERGENCY OFF input in the parameter buffer.
7-n	if desired: Rx-buffer, Init Domain,...

Parametration of the Ids in the Data-Structural Field	
1.	When required set the cell 'EVTRIG' in the data-structural components of the identifiers which should trigger a VMEbus interrupt at end condition.
2.	Enter the time-out periods into the data-structural components of the desired identifiers. (Can also be changed during operation.)

Operation of the CAN2
After this configuration only the data of the desired Rx-identifiers are updated in the data-structural components. The transmission of Tx-frames is started during the operation by setting the cell LENGTH in the data-structural component. The time-out counter, too, are activated by setting LENGTH. The actual status of the data transfer can each be read in the cell 'STATUS' in the data-structural component. For the selected identifiers the programmed VMEbus interrupt is triggered during operation at interrupt condition events.