HiCOCAN

HiCOCAN-SW

Software Documentation for DOS

72.53.0000.0

Revision No.	**Changes**	**Date**
1	First edition for DOS version of manual	24.04.2008 / Rr

This document is published by:

Emtrion GmbH

Greschbachstr. 12

D-76229 Karlsruhe

Tel:	+49 (721) 62725-0
Fax:	+49 (721) 62725-19
E-mail:	mail@emtrion.de
Internet:	http://www.emtrion.de

May 2007-011

Index

1 Introduction

HiCOCAN is the perfect solution for connecting your PC to a CAN net. No matter what applications you wish to develop, HiCOCAN provides the performance required for your specific design needs, at the highest bus rates.

The following HiCOCAN boards are available:

HiCOCAN-PCI-1		Standard PC card for the PCI bus of a standard PC with one CAN interface
HiCOCAN-PCI-2		Standard PC card for the PCI bus of a standard PC with two CAN interfaces
HiCOCAN-CPCI		Compact PCI card with two CAN interfaces
HiCOCAN-MiniPCI		miniPCI type IIIA board with two CAN interfaces (no DOS support!)
HiCOCAN-104-2H	Second generation	PC/104 variant with two CAN interface, high-speed (jumpers)
HiCOCAN-104-2L	Second generation	PC/104 variant with two CAN interface, low-speed, fault-tolerant (jumpers)

All boards are shipped with a preinstalled firmware, which handles Layer 2 of the ISO/OSI reference model. The PC's function libraries are used for communications with the firmware and so provide optimum access to the CAN net.

The main intention of this manual is to give a good starting point for writing a driver for special operating systems which are not supported out of the box by emtrion for these boards.

In the following chapter you will find a description of the different API functions for DOS. Anyway we recommend studying carefully the source code to learn more about the behaviour of the driver.

1.1 Software Package Contents

The delivery of HiCOCAN-DOS-SW contains the following:

HiCOCAN-	**PCI**	**CPCI**	**MiniPCI**	**104-2H**	**104-2L**
DOS support	X	X	no	X	X

- C-source code for HiCOCAN-PCI/CPCI or HiCOCAN-104 for DOS driver
- HICOCAN.h C header file
- Sample application
- CONFIG.EXE configuration tool

The installation will create the following directory tree:

DOS	Driver	Isa		C sources and example for DOS applications with HiCOCAN-ISA
		Pci		C sources and example for DOS applications with HiCOCAN-PCI/CPCI
	Tools	Isa		Configuration tool for HiCOCAN-ISA
		Pci		Configuration tool for HiCOCAN-PCI/CPCI

1.2 Overview of the API Functions

The table below provides an overview of the API functions.

If you require an API function in a driver that does not yet support this functionality, please contact us.

Function		
	PC104	PCI/CPCI
General Functions		
HiCOCANCloseDriver	X	X
HiCOCANGetErrorString	X	X
HiCOCANSetResource	X	
Functions for Controlling the CAN Nodes		
HiCOCANopen	X	X
HiCOCANStart	X	X
HiCOCANStop	X	X
HiCOCANReset	X	X
HiCOCANResetContr	X	X
HiCOCANClrOverrun	X	X
HiCOCANAbortTransmit	X	X
Timestamp		
HiCOCANSetTimestamp	X	X
Read/Write Functions		
HiCOCANWrite	X	X
HiCOCANRead	X	X
Status Request		
HiCOCANState	X	X
HiCOCANStateContr	X	X

	HiCOCANTraQState	X	X
	HiCOCANRecQState	X	X
Modifing the Communcation Parameters			
	HiCOCANSetAcceptMask	X	X
	HiCOCANSetBaud	X	X
	HiCOCANSetTimingReg	X	X
	HiCOCANParameter	X	X

2 Special Remarks on DOS

The application interface of the driver for DOS essentially corresponds to the application interface of the driver for Windows. The minor differences between the two versions are indicated in the documentation of the application interface.

When designing DOS applications note that the driver for DOS requires the use of the memory models Compact, Large or Huge.

Ideally, the compiler is set in such a way that FAR pointers are automatically used.

In addition, the following must be considered, which also applies to completely developed applications.

To ensure an optimum operation under DOS, we recommend that you reserve the memory area used by the DPM with the X= switch of EMM386.EXE.

Example:
HiCOCAN's DPM begins at address D800:0000. In this case, the following must be entered in the CONFIG.SYS file:

```
device=C:\DOS\emm386.exe X=D800-D880
```

The DPM's base address can be determined with the configuration tool. In doing so, note that the information displayed on the module's firmware and the state of configuration might not be correct.

3 Application Interface

The transfer constants and return values mentioned in this chapter are defined in the supplied header file HiCOCAN.h.

3.1 Basic Structure of an Application

The basic structure of an application is shown by Fig. 1:

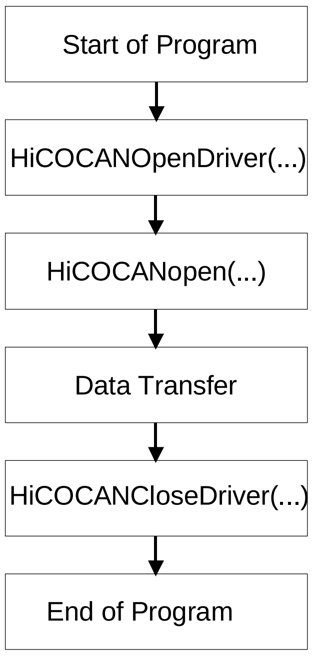

Fig. 1: Basic structure of an application

The first function **HiCOCANOpenDriver** executes all the necessary initializations within the software.

The **HiCOCANopen** function must be executed for each CAN node. After this, the application is allowed to make use of all other software functions (DLL) without any restrictions.

If a CAN node was programmed with StartMode 0 for using the configuration tool ,the **HiCOCANStart** function must be called before the data traffic with the CAN net is started. For HiCOCAN-MiniPCI this command must be called anyway.

The application is closed with **HiCOCANCloseDriver**. The resources used by the software are then freed again. HiCOCANClose does not have to be called for each individual node, since this is done by the HiCOCANCloseDriver function.

3.1.1 Data Structures

In the **HiCOCAN.h** C header file two data structures are defined which are required for both reads and writes and for reading the timestamps:

```
typedef struct
{
       BYTE   ff;         /* frame format:  0 = basic CAN,        */
                          /*                1 = extended CAN      */
       BYTE   rtr;        /* 0 = normal frame, 1 = remote frame    */
       BYTE   dlc;        /* data length 0..8                      */
       DWORD  id;         /* telegram ID                           */
       BYTE   data[8];   /* data                                   */
       sTS    timestamp; /* time stamp*/
}sPCCanMsg, *psPCCanMsg;
```

ff specifies whether the CAN message is a Basic CAN message with 11 identifier bits (HiCOCAN_FORMAT_BASIC) or a CAN message in the Extended CAN format with 29 identifier bits (HiCOCAN_FORMAT_EXTENDED).

rtr indicates the frame type; HiCOCAN_REMOTE_FRAME stands for remote messages and HiCOCAN_NORMAL_FRAME for normal messages

For HiCOCAN-MiniPCI there is an extension available. If bit 7 is set, this telegram has been received during an overrun condition of the CAN network. This means that some telegrams are possibly missing. If bit 6 is set, this telegram has been received with a fault tolerant error condition on the CAN network. This bit makes only sense if there is a fault tolerant transceiver available.

The number of data bytes is entered with **dlc**.

The ID of a message is always specified with the DWORD **id**; no matter if it is a Basic CAN message or an Extended CAN message.

An array of bytes **data[8]** is reserved for the message data.

Each received message is provided with a timestamp:

```
typedef struct
{
       WORD   day;
       BYTE   hour;
       BYTE   min;
       BYTE   sec;
       WORD   ms;
       WORD   us;        /* micro seconds; max. resolution 10µs */
}sTS, *psTS;
```

In addition, the C header file contains the sHiCOCANDriverInfo data structure which is required for the HiCOCANGetDriverInformation function:

```
typedef struct
{
  BYTE   bStructVersion;
  LPBYTE lpbVersionStringASCII;
  LPWORD lpwVersionStringUnicode;
  DWORD  dwSubVersion;
  WORD   wSizeOfPCRecQ;
} sHiCOCANDriverInfo, *psHiCOCANDriverInfo;
```

The bStructVersion element specifies the structure's version number. It has to be initialized by the application (currently, to 1).

The lpbVersionStringASCII element contains a pointer to a buffer which is at least 30 bytes in size. In this buffer, an ASCII string is stored that contains the revision number as ASCII text. For driver version 4.5, it will look as follows: $ProjectRevision : 4.5$

The element lpwVersionStringUnicode contains a pointer to a buffer which is at least 60 bytes in size. In this buffer, a string is stored that contains the revision number as unicode text. It will look like the ASCII string.

The dwSubVersion element stores the version number of the kernel driver.

The wSizeOfPCRecQ element stores the size of the driver-internal receive queue.

3.1.2 Return Values of the Functions

All software functions return a 32-bit code as return value. The following is a list of all return values available:

Return value	Significance
HTX_SUCCESS	No error detected
HTX_ERROR	General error
HTX_ERROR_ALREADY_OPENED	CAN node already open
HTX_ERROR_ALREADY_USED	CAN node already used by another process
HTX_ERROR_BOARD_NOT_ RUNNING	Error in the bootstrap loader or interrupt service routine cannot be installed.
HTX_ERROR_CANNOT_SET_IRQ	Desired interrupt not available
HTX_ERROR_CHECKSUM_ BOOTSTRAPPER	Checksum error in the bootstrap loader area
HTX_ERROR_CHECKSUM_ FIRMWARE	Checksum error in the firmware area
HTX_ERROR_DRIVER	An error occurred in the driver. Possible causes: ● Driver has not been installed properly. ● The HiCOCANOpenDriver function was not properly called by the application.
HTX_ERROR_EMPTY_QUEUE HTX_RECEIVEQUEUE_EMPTY	Receive queue of the specified CAN node empty
HTX_ERROR_FULL_QUEUE HTX_TRANSMITQUEUE_FULL	Transmit queue of the specified CAN node full
HTX_ERROR_ILLEGAL_ID	Illegal message ID
HTX_ERROR_ILLEGAL_LENGTH	Illegal message length
HTX_ERROR_MULTIPLE_NODE	Multiple CAN node in the system (e.g. if there are two modules with the same

Return value	Significance
	module number)
HTX_ERROR_NOCONFIG	The flash does not contain valid configuration data for the specified CAN node.
HTX_ERROR_NOT_SUPPORTED	Function not supported by the module's firmware.
HTX_ERROR_REGISTRY	An error occurred while accessing the Plug and Play configuration data in the Windows registry (the registry could not be read).
HTX_ERROR_SYSTEM	The firmware reported a system error. Bits 16 through 23 of the return value contain an error number. If the error number equals 1, the HiCOCAN module is busy. Other error numbers are internal firmware errors. For further information, please contact the emtrion support team.
HTX_ERROR_TRIGGERLEVELSET_ INVALID	The value specified for the trigger level dwTriggerLevelSet is not valid, because the receive queue does not have this size.
HTX_ERROR_TRIGGERLEVELRES ET_INVALID	The value specified for the trigger level dwTriggerLevelReset is not valid, because it is greater or equal to trigger level dwTriggerLevelSet.
HTX_ERROR_UNKNOWN_BOARD	The specified board could not be found. This return value is also output if the software was not informed about the CAN nodes present on the board with the HiCOCANOpen function.
HTX_ERROR_UNKNOWN_NODE	The software was unable to find the specified CAN node (with the HiCOCANOpen function). The error does not occur if CAN node 2 is to be accessed on a board where only one CAN node is

Return value	Significance
	installed. With all other software functions this return value means that the software (DLL) was not informed about the CAN node via the HiCOCANOpen function.

3.2 Numbering the CAN Nodes

It is possible to operate up to 4 HiCOCAN boards (not HiCOCAN-MiniPCI) in a system; the boards must be given different board numbers

The CAN node is selected via the node number (can) according to the following formula:

can = 2 * board number + number of the CAN controller - 1

"Number of the CAN controller" may have a value of 1 or 2, which results in the following values for can:

Number of CAN node	Location of the CAN node
0	Module 0, CAN controller 1
1	Module 0, CAN controller 2
2	Module 1, CAN controller 1
3	Module 1, CAN controller 2
4	Module 2, CAN controller 1
5	Module 2, CAN controller 2
6	Module 3, CAN controller 1
7	Module 3, CAN controller 2

Table 1: Assigning the CAN node number to the CAN controllers on max. 4 boards

Note
The numbering of the CAN nodes depends on the number of the CAN nodes actually available on the board.
The number of physically available CAN nodes, however, will not be checked!

3.3 Communication with HiCOCAN

3.3.1 Commands to the Board

The board and the installed CAN nodes are controlled via entries in specific command cells of the DPM. The firmware analyzes these cells as follows:

- First it is checked whether the command cell is still in use by a previous command.
- If the cell is free, the required data can be entered in the DPM.
- The information about the command to be executed by the board is entered in the command cell.
- The firmware executes the command and then resets the command cell.

3.3.2 Determining the Status Information

The status information contained in the communication areas in the DPM is updated each time the firmware is modified. It is thus sufficient to read out the corresponding state via the functions provided.

3.3.3 Message Transfer with the DOS Driver

3.3.3.1 Message Transfer in the Polling Mode

Sending a message

- The software checks the filling of the transmit queue.
- If at least one entry is free, the software writes the message to the transmit queue.
- The write pointer of the transmit queue is set to the next free entry in the transmit queue.
- The firmware of the board takes the message from the transmit queue and transmits it.
- The read pointer of the transmit queue is set to the message to be transmitted next.

Receiving a message

- When the board has received a message, the firmware enters it in the receive queue, if space is available.
- The software cyclically checks whether or not a message was received. If so it is read out.
- The receive queue's read pointer is set to the message to be read next.

In order to determine whether a queue is empty, the read and write pointer are compared with each other. If they are idential, the queue is empty.

3.3.3.2 Message Transfer in the Interrupt Mode

Receiving and transmitting a message is also possible in the interrupt mode. An interrupt is generated with the following events:

- A new message was entered in the receive queue.
- The firmware has taken a message from the transmit queue.
- At least one message could not be received by the CAN controller, because the receive queue in the DPM and the CAN controller's receive buffer were overrun.
- The CAN controller entered bus-off state.

The software reads the cause and acknowledges an interrupt in this sequence:

1. Use the semaphore in the communications area of the board;
2. Determine the cause of the interrupt;
3. Activate the used semaphore;
4. Read the DPM's interrupt cell in order to acknowledge the interrupt;
5. Analyze the cause of interrupt and execute the CALLBACK function provided by the application. If no CALLBACK function is available, no function will be executed.

3.4 Interrupt Handling

3.4.1 Interrupt-Driven Reads

The read interrupt is enabled each time a new message was received by the receive queue. The interrupt executes the CALLBACK function that was specified when the node was opened. The CALLBACK function reads out the receive queue, for example.

```
void far RecHandlerfunc( BYTE node, WORD count );
```

Note
If no interrupt-driven reads are to be performed for the CAN node, **NULL** must be specified for the RecHandler parameter when using the HiCOCANOpen function.

Since the PC cannot ensure that no interrupts will get lost, this CALLBACK function obtains the number of messages contained in the receive queue as parameter.

The CALLBACK function defined in the application should be structured in such a way that the read function HiCOCANRead (see below) is called in a 'For loop' with "count" as end value. The "count" parameter contains the number of messages available in the receive queue of the corresponding CAN node. 'For loop' ensures that the receive buffer is emptied even when interrupts are lost.

Example of a CALLBACK function in the application:

```
void RecHandlerfunc( BYTE node, WORD count )
{
        WORD i;
        :
        :
        for(i = 0; i < count; i++)
        {
                :
                :
                HiCOCANRead( ... );
                :
                :
        }
        :
        :
}
```

3.4.2 Interrupt-Driven Writes

With interrupt-driven writes an interrupt is released each time a message is taken from the transmit queue. The CALLBACK function invoked by the interrupt can be used to fill the transmit queue by means of the HiCOCANWrite function.

```
void far TraHandlerfunc( BYTE node );
```

Note
If no interrupt-driven writes are to be performed for the CAN node, **NULL** must be specified for the parameter TraHandler **NULL** when using the HiCOCANopen function.

3.4.3 Error Interrupts

An interrupt occurs when a CAN node switches to the Bus-Off state or when it is no longer able to receive messages due to a filled-up receive queue (Overrun). The driver then activates the corresponding CALLBACK function.

Prototype of the CALLBACK function for Overrun and Bus-Off error:

```
void far Ov_ErrHandlerfunc( BYTE node );
```

```
void far BO_ErrHandlerfunc( BYTE node );
```

Note
If no CALLBACK function is to be implemented with a CAN node, **NULL** must be specified for the corresponding parameter with the HiCOCANopen function.

3.5 General Functions

It is possible to have up to four boards in a system and they must differ in their board numbers (the relevant jumpers are described in later chapters) and their resources.
HiCOCANInitDriver()

Before calling a driver's function for the first time, the driver has to be initialized. For this, call:

DWORD HiCOCANInitDriver(void)

Return values:

HTX_SUCCESS

HTX_ERROR

Note
If this function yields HTX_ERROR, no further functions of the driver DLL may be invoked. Check whether the driver was installed properly.

3.5.1 HiCOCANCloseDriver()

Before closing the application call:

```
DWORD HiCOCANCloseDriver( void )
```

in order to re-enable all resources used by the DLL.

Return value:

HTX_SUCCESS

HTX_ERROR_APPLICATION

HTX_ERROR_SYSTEM

3.5.2 HiCOCANGetErrorString()

Determines the symbolic name HTX_...... of the specified value:

```
DWORD HiCOCANGetErrorString (DWORD ErrorCode, TCHAR *Puffer)
```

ErrorCode	Error code for which the error text is to be determined.
*Puffer	Pointer to the beginning of a buffer with a size of 100 characters, where the error code is to be entered.

Return values:

HTX_SUCCESS

HTX_ERROR

If the return value is HTX_ERROR, "UNKNOWN ERRORCODE" is returned as text in the array buffer.

3.5.3 HiCOCANSetResource()

In systems that do not support Plug and Play, the function:

```
void HiCOCANSetResource( BYTE Irq, DWORD MemAddress,
                         WORD IOAddress )
```

can be used to manually assign resources to the HiCOCAN modules.

Irq	Number of the interrupt to be used. Valid values are 3, 4, 5, 9, 10, 11, 12, 15
MemAddress	Start address for the DPM: The value is to be entered as a linear address. Example: If the DPM is to be found from address D800:0000 in the memory, the value D8000 must be entered for this parameter. Valid value range: D800:0000 through EFFF:0000
IOAddress	I/O address to be used for the module. Valid value range: 200h through 3FFh.

Return value: none

Calling the HiCOCANSetResources function will search for HiCOCAN modules that were not assigned valid resources. The first HiCOCAN module found without valid resources is assigned the resources that were specified for the function.

Note
The specified resources should not be used by another module within the system!

3.6 Functions for Controlling the CAN Nodes

3.6.1 HiCOCANopen()

This function is used to initialize the driver for a specific CAN node:

**DWORD HiCOCANopen(BYTE can, BYTE irqnr, (void*) RecHandler,
(void*) TraHandler, (void*) BO_ErrHandler,
(void*) Ov_ErrHandler);**

Can	Number of the CAN node for which the driver is to be opened.
Irqnr	Number of the desired Irq; if FFh the current interrupt is to be kept
RecHandler	ISR (CALLBACK) for receive interrupts. the count parameter specifies the number of messages received. Prototype of the CALLBACK function with the DOS driver: **void far RecHandlerfunc (BYTE node, WORD count)**
TraHandler	ISR (CALLBACK) for transmit interrupts. Prototype of the CALLBACK function with the DOS driver: **void far TraHandlerfunc (BYTE node)**
BO_ErrHandler	ISR (CALLBACK) for Bus-Off error interrupts. Prototype of the CALLBACK function with the DOS driver: **void far BO_ErrHandlerfunc (BYTE node)**
Ov_ErrHandler	ISR (CALLBACK) for Overrun error interrupts. Prototype of the CALLBACK function with the DOS driver: **void far Ov_ErrHandlerfunc (BYTE node)**

Return values:

HTX_SUCCESS
HTX_ERROR_DRIVER
HTX_ERROR_UNKNOWN_NODE
HTX_ERROR_MULTIPLE_NODE
HTX_ERROR_CANNOT_SET_IRQ
HTX_ERROR_REGISTRY
HTX_ERROR_ALREADY_OPENED
HTX_ERROR_ALREADY_USED
HTX_ERROR_BOARD_NOT_RUNNING
HTX_ERROR_CHECKSUM_BOOTSTRAPPER
HTX_ERROR_CHECKSUM_FIRMWARE
HTX_ERROR
HTX_ERROR_NOCONFIG
HTX_ERROR_SYSTEM

Executing the HiCOCANopen function informs the driver about the specified CAN node and performs the necessary initializations. After that, the CAN node may be accessed by all other functions.

Note
If no interrupts are to be used, **NULL** pointers must be specified for the CALLBACK functions. As a result, the interrupt is only acknowledged by the interrupt service routine, but no CALLBACK function is activated.

Important note
The board provides one interrupt line so that only one interrupt can be enabled. Therefore, the same interrupt number must be specified for both CAN nodes of a single board. Note that the number of physically available CAN nodes is not checked!

3.6.2 HiCOCANStart()

If HiCOCAN is at STOP state it can instantly re-enter RUN state with this option only:

DWORD HiCOCANStart (BYTE can)

Can	Number of the CAN node (0...7)

Return values:

HTX_SUCCESS

HTX_ERROR_UNKNOWN_NODE

HTX_ERROR_SYSTEM

This function instructs HiCOCAN to activate the specified CAN controller, in order to establish the CAN message transfer. Available CAN messages in the DPM's message queues will be kept. In operating modes other than STOP the HiCOCANStart function has no effect.

3.6.3 HiCOCANStop()

The opposite function of HiCOCANStart is this function:

DWORD HiCOCANStop (BYTE can)

Can	Number of the CAN node to be stopped (0...7)

Return values:

HTX_SUCCESS

HTX_ERROR_UNKNOWN_NODE

HTX_ERROR_SYSTEM

HiCOCANStop instructs the HiCOCAN firmware to set the specified CAN node to the reset mode, in order to stop the message transfer. Available CAN messages in the message queue of the DPM are deleted. In operating modes other than RUN HiCOCANStop has no effect.

3.6.4 HiCOCANReset()

The following function serves to restart the firmware on a HiCOCAN board:

DWORD HiCOCANReset (BYTE boardnr)

boardnr	Entry specifying the HiCOCAN board by using one of the following definitions of the supplied HICOCAN.H header file: #define HiCOCAN0 0 #define HiCOCAN1 2 #define HiCOCAN2 4 #define HiCOCAN3 6

Return values:

HTX_SUCCESS

HTX_ERROR_UNKNOWN_BOARD

HTX_ERROR_BOARD_NOT_RUNNING

HTX_ERROR_CHECKSUM_BOOTSTRAPPER

HTX_ERROR_CHECKSUM_FIRMWARE

HTX_ERROR_SYSTEM

3.6.5 HiCOCANResetContr()

The following function can be used to reset specific CAN nodes:

DWORD HiCOCANResetContr(BYTE can, bool newInit)

can	Number of the CAN node to be reset (0...7)
newInit	If TRUE, the specified CAN node is initialized again with the configuration data from the flash.

Return values:

HTX_SUCCESS

HTX_ERROR_UNKNOWN_NODE

HTX_ERROR_SYSTEM

If the CAN node is not initialized again, it is restarted in any case. If the CAN node is initialized again, it will be started only if the corresponding configuration parameter StartMode is unequal to null. This information is obtained from the relevant configuration data contained in the flash.

3.6.6 HiCOCANClrOverrun()

The Overrun bit indicates that at least one message could not be received (and got lost) due to a full receive queue. The Overrun bit of a specific CAN node is reset with this function:

DWORD HiCOCANClrOverrun(BYTE can)

can	Number of the CAN node whose overrun bit is to be reset

Return values:

HTX_SUCCESS

HTX_ERROR_UNKNOWN_NODE

HTX_ERROR_SYSTEM

3.6.7 HiCOCANAbortTransmit()

Using the following function, a message that is currently being transferred by the CAN controller can be aborted:

DWORD HiCOCANAbortTransmit(BYTE can)

Can	Number of the CAN node whose transmit buffer is to be deleted.

Return values:

HTX_SUCCESS

HTX_ERROR_UNKNOWN_NODE

HTX_ERROR_SYSTEM

3.7 Timestamp

HiCOCAN's firmware provides each message with a timestamp indicating the time (with 10-µs resolution) when the message was received. The timer installed on the board starts with power-on or every time the board is reset at 0 days, 00:00:00.000.000 hours.

3.7.1 HiCOCANSetTimestamp()

This function enables the application to set a timestamp at a specific time: The HiCOCANSetTimestamp function has been implemented to enable the application to place timestamps at a particular time. It provides the firmware with the specified timestamp information. The firmware will then accordingly set the timer on the HiCOCAN module.

DWORD HiCOCANSetTimestamp(BYTE board, psTS timestamp)

board	Entry specifying the HiCOCAN board by using one of the following definitions of the supplied HICOCAN.H header file: #define HiCOCAN0 0 #define HiCOCAN1 2 #define HiCOCAN2 4 #define HiCOCAN3 6
timestamp	Pointer to a variable of the type sTS containing the relevant time information. The sTS data type is defined in the supplied C-header file HiCOCAN.h; see chapter "Data Structures" on p. 12. The driver for DOS is a far pointer.

Return values:

HTX_SUCCESS

HTX_ERROR_UNKNOWN_BOARD

HTX_ERROR_SYSTEM

3.8 Read/Write Functions

3.8.1 HiCOCANWrite()

HiCOCANWrite instructs the driver layer to write a CAN message to the transmit queue of the corresponding CAN node. The data for the structure of the CAN message must be reported in form of a pointer to a structure of the type sPCCanMsg. This type is defined in the supplied HiCOCAN.h C-header file.

DWORD HiCOCANWrite(BYTE Can, psPCCanMsg msg)

Can	Number of the node which is to transmit the message.
Msg	Pointer to message data The driver for DOS is a far pointer.

Return values:

HTX_SUCCESS

HTX_ERROR_UNKNOWN_NODE

HTX_ERROR_FULL_QUEUE

HTX_ERROR_ILLEGAL_ID

HTX_ERROR_ILLEGAL_LENGTH

HTX_ERROR_SYSTEM

HiCOCANWrite first checks whether there is sufficient space in the transmit queue of the desired CAN node. If sufficient space is available, the CAN message is created from the specified message data and written to the transmit queue. If the space is unavailable, the HTX_ERROR_FULL_QUEUE error code is immediately returned with the DOS version.

Hint
The timestamp data are taken into consideration.

3.8.2 HiCOCANRead()

Using the driver's HiCOCANRead function, a CAN message can be read from the receive queue of a specific CAN node. HiCOCANRead enters the message data in a structure of the type sPCCanMsg. A corresponding structure needs to be provided by the application (allocated memory), and HiCOCANRead must have a pointer to this structure. The sPCCanMsg type is defined in the supplied HiCOCAN.h C-header file; see chapter "Data Structures" on p. 12.

DWORD HiCOCANRead(BYTE can, psPCCanMSg msg)

can	Number of the desired Can node
msg	Pointer to a structure of the type sPCCanMsg provided by the application. The driver for DOS is a far pointer.

Return values:

HTX_SUCCESS

HTX_ERROR_UNKNOWN_NODE

HTX_ERROR_EMPTY_QUEUE

HTX_ERROR_SYSTEM

HiCOCANRead first checks whether there is a message available in the receive queue of the desired CAN node. If this is the case, the message is transferred to the structure provided by the application to which the msg pointer points. The supplied timestamp specifies the time that the CAN controller received this message. For more detailed information, please refer to section "HiCOCANSetTimestamp()" on p. 30.

If no message is available in the corresponding receive queue in the DOS driver, HTX_ERROR_EMPTY_QUEUE is returned.

3.9 Status Request

3.9.1 HiCOCANState()

The state of a HiCOCAN board can be requested with this function:

DWORD HiCOCANState(BYTE board, BYTE* State)

board	Entry specifying the HiCOCAN board by using one of the following definitions: #define HiCOCAN0 0 #define HiCOCAN1 2 #define HiCOCAN2 4 #define HiCOCAN3 6
*State	Pointer to a variable of type BYTE defined by the calling function. The state of the desired board is written to the specified variable. Table 2 displays the significance of the returned value. Please note that the HiCOCAN board is at the "ready" state only, if all status bits are 0 except for status bit 6. The driver for DOS is a far pointer.

Return values:

HTX_SUCCESS

HTX_ERROR_UNKNOWN_BOARD

HTX_ERROR_SYSTEM

Hint
The returned state is valid only if the function returns HTX_SUCCESS.

Bit pos.	Function	Bit Value	Significance
7	ERROR	1 = yes	General error
6	RUN	1 = yes	Firmware running
5	SYS_FAIL	1 = yes	Hardware error
4	FW_CHK	1 = yes	Firmware error (checksum)
3	BOOTCHK	1 = yes	Bootstrapper error (checksum)
2	---		Reserved
1	CFGCHK	1 = yes	No configuration data available for the CAN nodes
0	---		Reserved

Table 2: Significance of the returned value

3.9.2 HiCOCANStateContr()

The HiCOCANState function provides information on the state of the CAN controller:

DWORD HiCOCANStateContr(BYTE can, BYTE* State, BYTE* NumOfRecErrors, BYTE* NumOfTraErrors)

can	Number of the desired CAN node
*State	Pointer to a variable of type BYTE defined by the calling function. The state of the desired CAN controller is written to the specified variable. Table 3 displays the significance of the returned value. The DOS driver is a far pointer.
*NumOfRecErrors	Pointer to a variable of type BYTE defined by the calling function. The number of the receive errors is written to the specified variable. The DOS driver is a far pointer.
*NumOfTraErrors	Pointer to a variable of type BYTE defined by the calling function. The number of the transmit errors is written to the specified variable. The DOS driver is a far pointer.

Return values:

HTX_SUCCESS

HTX_ERROR_UNKNOWN_NODE

HTX_ERROR_SYSTEM

Hint
The returned state and the numbers of errors are valid only if the function returns HTX_SUCCESS.

Bit pos.	Function	Bit value	Significance
7	Bus-Off	1 = yes	CAN controller is in the Bus off state
6	Error Passive	1 = yes	CAN controller is in the Error-Passive state
5	Controller transmits a message	1 = yes	
4	Controller receives a message	1 = yes	
3	Last request for transmit message successfully terminated	1 = yes	
2	Transmit buffer available	1 = yes	
1	Overrun CANRxFifo	1 = yes	CAN controller has set the Overrun bit, i.e., at least one message could not be received due to a full receive queue.
0	Messages are available	1 = yes	A message is available in the receive buffer of the controller.

Table 3: Significance of the returned value

Note
If the status bits 4 and 5 are simultaneously set, the controller is in the Stop state. For a more detailed description of the significance, please refer to the CAN controller's data sheet [12] or the Application Note [13].

3.9.3 HiCOCANTraQState(), HiCOCANRecQState()

Using the functions

DWORD HiCOCANTraQState(BYTE can)

DWORD HiCOCANRecQState(BYTE can)

the state of the transmit or receive queue of each CAN node can be examined. If no error occurred, the number of entries is returned.

can	Number of the desired Can node

Return values:

HTX_ERROR_UNKNOWN_NODE

HTX_ERROR_SYSTEM

HTX_TRANSMITQUEUE_FULL (only for function HiCOCANTraQState)

HTX_RECEIVEQUEUE_EMPTY (only for function HiCOCANRecQState)

Number of the messages entered

3.10 Modifying the Communications Parameters

The functions described in this chapter allow you to temporarily alter communications parameters such as the baud rate or acceptance filter.

3.10.1 HiCOCANSetAcceptMask()

The CAN controller allows for filtering the messages by means of the hardware. For this purpose, several registers are provided whose functions are described in the data sheet [12] or the corresponding Application Note [13], respectively.

The acceptance mask is set with this function:

```
DWORD HiCOCANSetAcceptMask(BYTE can, BYTE FilterMode,
                           DWORD Code,DWORD MaskReg)
```

Can	Number of the desired Can node
FilterMode	Value of the acceptance filter mode; As for the significance of this value, please refer to the data sheet [12] or the Application Note [13]. The constants HiCOCAN_FILTERMODE_DUAL and HiCOCAN_FILTERMODE_SINGLE defined in the HICOCAN.h header file can be used to set this value.
Code	Value of the acceptance code register; The most significant byte with the register ACR3 and the least significant byte with register ACR0 correspond with each other. As for the significance of this value, please refer to the data sheet [12] or the Application Note [13].
MaskReg	Value of the acceptance mask register; The most significant byte with the register ACR3 and the least significant byte with register ACR0 correspond with each other. As for the significance of this value, please refer to the data sheet [12] or the Application Note [13].

Return values

HTX_SUCCESS

HTX_ERROR_UNKNOWN_NODE

HTX_ERROR_SYSTEM

3.10.2 HiCOCANSetBaud(), HiCOCANSetTimingReg

The baud rate at which the CAN node operates is determined by the two 8-bit-wide bus timing registers BTR0 and BTR1. As for the significance of the contained bits, please refer to the data sheet [12] or the Application Note [13] of the CAN controller used. Note that the CAN controllers are operated with 20 MHz.

Two functions are provided for setting these registers and thus the baud rate. The first function writes the specified values for the Bus timing registers of the specified CAN node directly to these registers:

DWORD HiCOCANSetTimingreg(BYTE can, BYTE reg0, BYTE reg1)

can	Number of the desired Can node
reg0	Value for timing register 0
reg1	Value for timing register 1

Return values:

HTX_SUCCESS

HTX_ERROR_UNKNOWN_NODE

HTX_ERROR_SYSTEM

The second function:

DWORD HiCOCANSetBaud(BYTE can, BYTE rate)

takes the values for the bus timing register from the Board's flash device. These values are stored by means of the configuration tool. The parameters to be specified are the following:

can	Number of the desired Can node
rate	Entry specifying the desired baud rate by using the following definitions (defined in the HiCOCAN.h header file): #define HiCOCAN_BAUD10K 1 #define HiCOCAN_BAUD20K 2 #define HiCOCAN_BAUD50K 5 #define HiCOCAN_BAUD100K 10 #define HiCOCAN_BAUD125K 12 #define HiCOCAN_BAUD250K 25 #define HiCOCAN_BAUD500K 50 #define HiCOCAN_BAUD1M 100

Return values:

HTX_SUCCESS

HTX_ERROR_UNKNOWN_NODE

HTX_ERROR_SYSTEM

Important note
A message that might be transmitted while the baud rate is being changed will get lost with both functions!

3.10.3 HiCOCANParameter()

The HiCOCANParameter function allows you to determine the communications parameters currently set in the CAN controller:

DWORD HiCOCANParameter(BYTE can, BYTE *TimingReg0, BYTE *TimingReg1, WORD *Baudrate, BYTE *FilterMode, DWORD *Code, DWORD *MaskReg)

Can	Number of the desired CAN node
*TimingReg0, *TimingReg1	Pointer to two variables to which the values of the bus timing registers 0 and 1 are copied. The DOS driver is a far pointer.
*Baudrate	Pointer to a variable which is provided with a value (in kBaud) for the baud rate. The DOS driver is a far pointer.
*FilterMode, *Code, *MaskReg	Pointer to variables where the values for acceptance filter mode, the acceptance code register and the acceptance mask register are to be entered. The DOS driver is a far pointer.

Return values:

HTX_SUCCESS

HTX_ERROR_UNKNOWN_NODE

HTX_ERROR_NOT_SUPPORTED

HTX_ERROR_SYSTEM

Hint
The parameters' returned values are valid only if the function returns HTX_SUCCESS.

4 Configuration Tool

The configuration tool serves to permanently configure the board. The DOS version of this tool reads or writes configuration data to a HiCOCAN board, or displays information about a specific board.

The **config.exe** configuration tool is called as follows:

```
config number operation <filename>
```

number	Board number Possible values: 0, 1, 2 or 3
operation	Switch for the desired function. The following switches may be used: /r or /R : The configuration data are read from the board and written to file <filename>. /w or /W : The configuration data are read from the file <filename> and transmitted to the board. /i or /I : Determines the board information and outputs the information to the screen. The <filename> parameter is not required with this switch. The information determined is in accordance with the information obtained from the Windows version of this configuration tool (see menu**Fehler! Verweisquelle konnte nicht gefunden werden.**).
<filename>	Name of the file containing the configuration data. The file format is the same as of the files of the Windows version; it is described in section "Format of the Configuration Files" on p. **Fehler! Textmarke nicht definiert.**.

Hint
To ensure that the program is properly running under DOS, see the notes on how to use EMM386.EXE in section "Remarks on the DOS Driver”.

5 Demo Application

In order to show you how to use the application interface, the software package includes a demo application. This application transmits CAN messages between the two CAN nodes of a board, which are connected via a cable for this purpose. The delivery includes one version each for DOS and Windows.

First the main program is prepared for using the two CAN nodes by creating CALLBACK functions for the receive interrupts.

For the receive interrupt of node 1 a CALLBACK function is installed which is to modify and send back the data of a received message.

For the receive interrupt of node 0 a CALLBACK function is established which is to read receive messages and report them to the main program by setting a flag (MessageFlag).

After having prepared the CAN nodes, a CAN message is created in the main program; this message is a normal (not a remote) BASIC-CAN message with eight data bytes. The program interprets these eight data bytes of the message as two DWORD variables. Exactly one bit is set in each of the variables.

The MessageFlag is checked within a while loop of the main program. If it is set, the message is written to the screen and retransmitted to node 1. If a DWORD with value 0 has been received, the message data are reinitialized, resulting in some sort of running light.

This is repeated until any key is pressed. After that, the aforementioned while loop is terminated, the driver closed and the program left.

6 Troubleshooting (HiCOCAN-xxx)

6.1 Support

This product has been thoroughly tested over the development period. Due to its complexity, however, no guarantee can be given that the boards operate seamlessly under any circumstances. We are therefore grateful for any feedback regarding an improper operation of the boards.

If any problems occur, have a look at the FAQ section of this manual first. Or visit our website at http://www.emtrion.com/support/index.html for the latest FAQ.

If you cannot find the necessary information, contact our Support Team via e-mail, fax or phone. Your support question will be answered as soon as possible.

To accelerate the process, please fill out the supplied form, which can be found in the Support directory of the CD or on the internet at http://www.emtrion.com/support/index.html.

Please fill in the form and send, fax or email it to:

Emtrion GmbH
Greschbachstr. 12
D-76229 Karlsruhe
Tel: 0721 / 62725 – 0
Fax: 0721 / 62725 – 19
E-mail : mail@emtrion.de

7 Reference

[1] PC/104 Specification
Version 2.3, June 1996
http://www.controlled.com/pc104/techp1.html

[3] Am29F400B, CMOS 5.0V only, sector erase Flash Memory
AMD

[4] CY7C132, 2Kx8 Dual-Port Static RAM
Cypress Semiconductor Corporation

[10] Plug-and-Play BIOS Specification
Version 10.A, May 5, 1994
Compac Computer Corporation, Phoenix Technologies Ltd., Intel Corporation

[11] Extended System Configuration Data Specification
Version 1.0A, May 31, 1994
Compac Computer Corporation, Phoenix Technologies Ltd., Intel Corporation

[12] SJA1000 Stand-alone CAN controller
Data Sheet
Preliminary specification, 1997 Nov 04
Philips Semiconductors

[13] Application Note SJA1000 Stand-alone CAN controller
AN97076
Authors : Hank, Peter and Jöhnk Egon
1997 Dec 15
Philips Semiconductors