HCAN2

HCAN2-SW

Software Documentation of HCAN2 driver
for Windows embedded CE™ 6.0 R2

Copyright

72.53.0000.0

Manual Revision No.	Changes	Date
1	First edition adopted from the CE500 manual	03.04.2008/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

April 2008-004

Index

1. Introduction

The Hitachi Controller Area Network 2 (HCAN2) is a module that controls the Controller Area Network (CAN) provided for realtime communication in automobiles or industrial equipment systems. For details on the CAN specification, refer to the CAN Specification Version 2.0, Robert Bosch GmbH, 1991.

This module is implemented in the Renesas processor SH7760. The following document describes how the features of this module can be used within a driver supplied with the Windows embedded CE 6.0 R2 starterkit for the board HiCO.SH7760.

The driver realizes a subset of the rich functionality of HCAN2. The complete set of mailboxes and the different triggering and filtering capabilities are NOT used. The driver is as far as possible compatible to the drivers of the well known HiCOCAN-boards from emtrion. So it is easy to reuse all the applications already written for these boards.

This driver is a layer 2 implementation of the CAN protocol stack and supports basic CAN telegrams and extended CAN telegrams.

1.1. Software Package Contents

1.1.1. Windows embedded CE 6.0 R2 Starterkit

The DLLs are already integrated in the kernel which is delivered with the starter kit Windows embedded CE 6.0 R2 for HiCO.SH7760. The library and the header file are part of the SDK.

The components in detail:

- “7760CAN.lib” library for creating a Windows CE application
- “HiCOCAN.h” C header file
- Sample application (with sources)
- This Documentation as a PDF file
- The sources of the driver are available only with the advanced starter kits. These sources are integrated into the directory tree of the Emtrion starter kit.

1.2. Overview of the API Functions

The table below provides an overview of the driver and its API functions. In comparison to the API of the CAN interface cards from emtrion, there are some functions which are no longer supported.

If you require an API function in a driver that does not yet support this functionality, please contact us (see p. 2).

Function		
	supported	unsupported
General Functions		
HiCOCANOpenDriver	X	
HiCOCANCloseDriver	X	
HiCOCANResetDriver	X	
HiCOCANGetErrorString	X	
HiCOCANGetExtendedErrorString	X	
HiCOCANSetResource		X
HiCOCANGetDriverInformation	X	
Functions for Controlling the CAN Nodes		
HiCOCANOpen	X	
HiCOCANStart	X	
HiCOCANStop	X	
HiCOCANReset		X
HiCOCANResetContr	X	
HiCOCANClrOverrun	X	
HiCOCANAbortTransmit	X	
HiCCCANRegisterEvent	X	

Timestamp			
	HiCOCANSetTimestamp	X	
Read/Write Functions			
	HiCOCANWrite	X	
	HiCOCANRead	X	
	HiCOCANReadEx	X	
Status Request			
	HiCOCANState		X
	HiCOCANStateContr	X	
	HiCOCANTraQState	X	
	HiCOCANRecQState	X	
Modifing the Communcation Parameters			
	HiCOCANSetAcceptMask	X	
	HiCOCANSetBaud	X	
	HiCOCANSetTimingReg		X
	HiCOCANParameter		X

2. Installation

2.1. Installation for Emtrion starter kits

In the latest versions of the starter kits the driver and the basic registry entries are already integrated into the kernel. Additionally all necessary entries and services are added to the OAL.

The kernel mode driver obtains the resources (memory addresses, interrupts) used by the module HCAN2 of HiCO.SH7760 from the registry. These entries are available with the delivered kernel.

NOTE: Usually, there is no need to modify them.

The entries may be changed in the registry as follows:

- Manually by using WindowCEs' remote registry editor.
- Writing an own application.

A description of the contents can be found in the following sections.

2.2. Necessary Registry Entries

For the HCAN2 module, following entries must be made under this key (where X refers to the module number 0 to 1):

[HKEY_LOCAL_MACHINE\drivers\BuiltIn\HCAN7760_x]

Entry	Type	Comments
Prefix	String	Name for the device manager
Dll	String	Name of the low level driver which runs in kernel mode
Order	DWORD	Mouting order of this driver
Index	DWORD	Index of the driver
MemBase	DWORD	Specifies the hardware base address of the HCAN2 module in the CPU core
MemSize	DWORD	Size of the register area

Entry	Type	Comments
SysIntr	DWORD	Specifies the logical interrupt number which is used in the WindowsCE kernel
Priority256	DWORD	Proirity of the IST
BaudTable	DWORD	Specifies a table of typical baud rates which can be selected by the application. The last entry will be chosen as a start-up baud rate.

There are some more registry entries which are needed for the device manager of the operating system.

Usually, there is no need to modify any of these entries. Pay attention, changing the memory and interrupt entries leads to malfunction of the driver. It is possible to reset the entries to the default values by means of deleting the persistent registry.

If you have made changes, it is necessary to call “writereg.exe” to make the changes persistent. Please have a look into the manual of the WindowsCE starterkit to find more details about the persistent registry.

A typical entry might be:

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\HCAN7760_0]
      "Prefix"="HCN"
      "Dll"="HCAN7760.dll"
      "Order"=dword:0
      "Index"=dword:1
;-------------------------------------------------------------------
; interrupt information
;
      "SysIntr"=dword:28
      "Priority256"=dword:8c
;-------------------------------------------------------------------
; memory information
;
      "MemBase"=dword:FE380000
      "MemLen"=dword:500
;-------------------------------------------------------------------
; Table for baud rates, pay attention, the entries depend on PCLK
; peripheral clock of the CPU
;
;     name_low, name_high, tseg1, tseg2, brp
;
; entry in line 10 is default entry for system, it'name_low and
name_high value is zero
;
       "BaudTable"=hex:\
      0a,00,09,04,c7,\
      14,00,04,01,c7,\
      32,00,04,01,4f,\
      64,00,04,01,27,\
      7d,00,04,01,1f,\
      fa,00,04,01,0f,\
      f4,01,04,01,07,\
      20,03,04,01,04,\
      e8,03,04,01,03,\
      00,00,04,01,c7
```

“Name_low” and “name_high” are references to the number for selecting the baudrate with the function HiCOCANSetBaud(). The last three values of a line are the values which have to be programmed into the timing registers of the HCAN2 module. The last line of the table gives the default baudrate after starting up the driver. Here 20KBaud is chosen.

3. Multitasking

The supplied driver is capable of multitasking, i.e., it is able to support more than one process at the same time. As a rule, all driver functions are available to all processes. However, there are some restrictions:

1. Each of the maximally 2 available CAN nodes may be used by one process only.

2. The HiCOCANReset function should not be used. This function resets all temporary parameter settings and deletes both the receive- and transmit queue. This will cause problems when the two CAN nodes are used by several processes.
3. The HiCOCANResetDriver function usually re-enables all resources used by the driver. This is independent of the calling process, which means that a CAN node might not be accessed any more without restarting the DLL.

4. Application Interface

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

4.1. Basic Structure of an Application

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

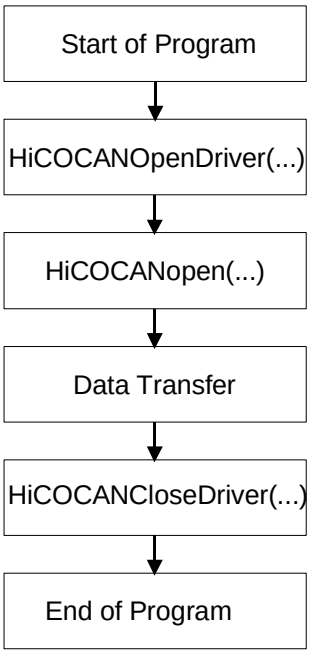

Fig. 1: Basic structure of an application

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.

The **HiCOCANStart** function must be called before the data traffic with the CAN net is started.

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.

4.1.1. 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.

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 1µs */
}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.

4.1.2. 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_BUSOFF	No error has occurred while the function is running; the accessed CAN node is in Bus-Off state.
HTX_OVERRUN	No error has occurred while the function is running; the accessed CAN node is in Overrun state.
HTX_CANERROR	No error has occurred while the function is running; however, the error counters of the accessed CAN node are unequal to 0.
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_APPLICATION	Invalid order of function calls through the application (A driver function was called without calling HiCOCANOpenDriver.)
HTX_ERROR_CANNOT_SET_IRQ	Desired interrupt not available
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	Transmit queue of the specified CAN node

Return value	Significance
HTX_TRANSMITQUEUE_FULL	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 module number)
HTX_ERROR_NOT_SUPPORTED	Function not supported by the module's firmware.
HTX_ERROR_REGISTRY	An error occurred while accessing the data in the Windows registry (the registry could not be read).
HTX_ERROR_RESOURCE	Error while accessing configuration data; no resources were allocated to the module (addresses and interrupts)
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_TRIGGERLEVELRESET_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_NODE	The software was unable to find the specified CAN node (with the HiCOCANOpen function).

Note

In order to have as much information as possible, many functions also return the state of the CAN controller like “overrun” or “busoff”. This is very important to know for the communication functions (HiCOCANRead, HiCOCANWrite, ….). Only in the case that no telegram transfer (reading or writing) was possible due to an empty of full queue, the functions return the appropriate value. In all the other cases, the functions return the state of the CAN controller as well.

4.2. Communication with HiCOCAN

4.2.1. Message Transfer

Using the HiCOCANRead or the HiCOCANReadEx function, you are able to read the received messages from the receive queue of the HCAN2 module. You can write the messages to be transmitted to the transmit queue of the module with the HiCOCANWrite function.

All three functions have a timeout time. This timeout time takes effect when the receive queue is empty or the transmit queue is full, respectively. In this case, the functions HiCOCANRead, HiCOCANReadEx and HiCOCANWrite are waiting for a free entry in the transmit queue until the timeout time has elapsed at the latest, before returning an error. This is done via driver-internal events by means of which the corresponding functions are blocked until the event occurs or the timeout time has elapsed. The use of those events prevents the functions from wasting runtime.

The events are set to the "Signalled state" via the interrupt service routine. The routine first processes the following causes of an interrupt:

- The new message was entered in the receive queue.
- The HCAN2 module has taken a message from the transmit queue.
- At least one message could not be received by the HCAN2 module, because the receive queue and the CAN controller's receive buffer were overrun.
- CAN controller entered bus-off state.

With the first two causes of an interrupt the respective event is set to the "Signalled state". As explained above, the functions HiCOCANRead, HiCOCANReadEX and HiCOCANWrite continue to process, reading the message from or writing it to the queues.

4.3. General Functions

4.3.1. HiCOCANOpenDriver()

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

DWORD HiCOCANOpenDriver(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.

4.3.2. 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

Note
After using the HiCOCANCloseDriver function, the HiCOCANOpenDriver function needs to be executed first, before you are able to use the driver's other functions!

Hint
Before calling this function, it is recommended that you halt all CAN nodes used by the calling process (application) via the HiCOCANStop function.
If this function was not used while debugging an application, for example, use the HiCOCANResetDriver function. This is necessary in order to free the resources used again.

4.3.3. HiCOCANResetDriver()

The function

DWORD HiCOCANResetDriver(void)

resets the driver, enabling all resources such as memory locations and interrupts.

Return values:

HTX_SUCCESS

HTX_ERROR_DRIVER

HTX_ERROR_APPLICATION

Important note
The HiCOCANResetDriver function should only be used during the development if the application was closed without successfully executing the HiCOCANCloseDriver function.
This function has an effect on all processes (applications) that make use of this driver! Therefore, make sure you have a look at section "Multitasking“ on p. 13 when using this function!
After using the HiCOCANResetDriver function, the HiCOCANOpenDriver function must be executed again!

4.3.4. 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.

This function is available for ANSI- and for Unicode. You may switch between the codes by defining the UNICODE symbol before linking the HICOCAN.h header file.

4.3.5. HiCOCANGetExtendedErrorString()

If a function returns the value HTX_ERROR_DRIVER, an extended error code is available using the Windows system function GetLastError. This extended error code can be converted into an error text using the following function:

DWORD HiCOCANGetExtendedErrorString (DWORD ErrorCode, TCHAR *Puffer)

Error text:

ErrorCode	Error code made available with GetLastError() for which the error text is to be created.
*Puffer	Pointer to the beginning of a buffer as large as 100 characters, where the error text is to be entered.

Return values:

HTX_SUCCESS

HTX_ERROR

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

This function is available for both ANSI- and Unicode. You may switch between the codes by defining the UNICODE symbol before linking the HICOCAN.h header file.

4.3.6. HiCOCANGetDriverInformation()

This function is used to query the version information.

It has the following prototype:

```
DWORD HiCOCANGetDriverInformation( LPBYTE lpbVersionNumber,
                                  void* lpExtendedInformation );
```

Parameters:

lpbVersionNumber	Pointer to a variable in which the version number of the driver is to be entered.
* lpExtendedInformation	Pointer to a structure of type sHiCOCANDriverInfo.

Return values:

HTX_SUCCESS

HTX_ERROR_APPLICATION

HTX_ERROR

The function fills the specified variables with the corresponding information. The returned version number is coded as follows: The upper four bits contain the major version, and the lower four bits the minor version. This means that for driver version 4.5 the value 45h is returned.

With the lpExtendedInformation parameter the following has to be considered. The bStructVersion element needs to be initialized by the application with a valid value (for valid values, see the description of the sHiCOCANDriverInfo structure). If, after calling the function, this element is bStructVersion 0, then all other data in this structure are invalid.

4.3.7. HiCOCANSetResource()

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

no longer supported with HiCO.SH7760

4.4. Functions for Controlling the CAN Nodes

4.4.1. HiCOCANOpen()

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

```
DWORD HiCOCANOpen( BYTE can );
```

Can	Number of the CAN node for which the driver is to be opened.

Reurn values:

HTX_SUCCESS
HTX_ERROR_APPLICATION
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
HTX_ERROR_NOCONFIG

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.

4.4.2. HiCOCANClose()

The following function enables a CAN node which can then be used by another application:

DWORD HiCOCANClose (BYTE can)

Can	Number of the CAN node (0...1)

Return values:

HTX_SUCCESS

HTX_ERROR_APPLICATION

HTX_ERROR_UNKNOWN_NODE

Note

- After using the HiCOCANCloseDriver function, the HiCOCANOpenDriver function needs to be executed first, before you are able to use the driver's other functions!
- HiCOCANClose is also called by the HiCOCANCloseDriver() function.

4.4.3. 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...1)

Return values:

HTX_SUCCESS

HTX_OVERRUN

HTX_BUSOFF

HTX_CANERROR

HTX_ERROR_APPLICATION

HTX_ERROR_UNKNOWN_NODE

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

4.4.4. HiCOCANStop()

The opposite function of HiCOCANStart is this function:

DWORD HiCOCANStop (BYTE can)

Can	Number of the CAN node to be stopped (0...1)

Return values:

HTX_SUCCESS

HTX_OVERRUN

HTX_BUSOFF

HTX_CANERROR

HTX_ERROR_APPLICATION

HTX_ERROR_UNKNOWN_NODE

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 are deleted. In operating modes other than RUN HiCOCANStop has no effect.

4.4.5. HiCOCANReset()

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

no longer supported with HiCO.SH7760

4.4.6. 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...1)
newInit	If TRUE, the specified CAN node is initialized again with the configuration data from the flash.

Return values:

HTX_SUCCESS

HTX_OVERRUN

HTX_BUSOFF

HTX_CANERROR

HTX_ERROR_APPLICATION

HTX_ERROR_UNKNOWN_NODE

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.

4.4.7. 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_OVERRUN
HTX_BUSOFF
HTX_CANERROR
HTX_ERROR_APPLICATION
HTX_ERROR_UNKNOWN_NODE

4.4.8. 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_OVERRUN
HTX_BUSOFF
HTX_CANERROR
HTX_ERROR_APPLICATION
HTX_ERROR_UNKNOWN_NODE

4.4.9. HiCOCANRegisterEvent()

Using the HiCOCANRegisterEvent function, a Win32 event can be registered, which is set when the specified level of the receive queue is reached or exceeded.

The function has this prototype:

```
DWORD HiCOCANRegisterEvent( BYTE can, HANDLE *lphEvent,
                            DWORD dwTriggerLevelSet,
                            DWORD dwTriggerLevelReset );
```

Parameters:

can	Number of the CAN node for which the event is to be registered.
*lphEvent	Pointer to the handle of the Win32 event.
dwTriggerLevelSet	Trigger level at which the Win32 event is set.
dwTriggerLevelReset	Trigger level below which the Win32 event is reset.

Return values:

HTX_SUCCESS

HTX_OVERRUN

HTX_BUSOFF

HTX_CANERROR

HTX_ERROR_APPLICATION

HTX_ERROR_UNKNOWN_NODE

HTX_ERROR_TRIGGERLEVELSET_INVALID

HTX_ERROR_TRIGGERLEVELRESET_INVALID

The application creates a Win32 event and passes the pointer to the handle of this function. The driver sets the specified event by means of the OS function SetEvent if the level of the receive queue reaches and/or exceeds the specified trigger level dwTriggerLevelSet (number of messages in the receive queue >= dwTriggerLevelSet). The driver resets the specified event when the specified trigger level dwTriggerLevelReset is not reached (number of messages in the receive queue < dwTriggerLevelReset).

Passing the value NULL in the lphEvent parameter to the HiCOCANRegisterEvent function will again deactivate this functionality.

Note
The driver expects the event to be passed to be valid. This means that the calling application cancels the event for the driver and then returns the handle to the operating system via CloseEvent.

Example using the function HiCOCANRegisterEvent :

This example demonstrates the use of the function HiCOCANRegisterEvent for can node 0. In this example, the application wants the event to be set when 20 or more messages in the receive queue of the can node 0.

The application also tells the driver that the event is to be reset when the application has read the messages and there are less than five messages in the receive queue of can node 0.

```
HANDLE          hEvent;
DWORD           dwRetVal = HTX_SUCESS;

// Create Win32 event. IMPORTANT : You MUST create a manual reset event,
// which is not signaled !!!
hEvent = CreateEvent( NULL, FALSE, FALSE, NULL );
if (hEvent == NULL)
{
 // ERROR : You MUST NOT call HiCOCANRegisterEvent, because the
    // event is not valid !
    // error handling
    return;
}
dwRetVal = HiCOCANRegisterEvent( 0, &hEvent, 20, 5 );
if ((dwRetVal != HTX_SUCCESS) &&
   (dwRetVal != HTX_OVERRUN) &&
   (dwRetVal != HTX_BUSOFF) &&
   (dwRetVal != HTX_CANERROR) )
{
    // ERROR : The event will NOT handled by the HiCOCAN driver. Check
    // the error code for detail information.
    // error handling
    return;
```

```
}
// IMPORTANT : You MUST NOT close the event handle before you have the
// event deregistered by the driver (see below)

// begin a loop
while ( fEndApplication == FALSE )
{
    // Wait until the event is signaled
    dwRetVal = WaitForSingleObject( hEvent, INFINITE );
    if ( dwRetVal == WAIT_OBJECT_0 )
    {
        // event signal -> do something e.g. read telegrams by calling
        // HiCOCANRead or HiCOCANReadEx
    }
    else
    {
        // error handling
    }
}

// We want that the application should exit -> deregister event
dwRetVal = HiCOCANRegisterEvent( 0, NULL, 0, 0 );
if ((dwRetVal != HTX_SUCCESS) &&
  (dwRetVal != HTX_OVERRUN) &&
  (dwRetVal != HTX_BUSOFF) &&
  (dwRetVal != HTX_CANERROR) )
{
    // ERROR : The event will NOT handled by the HiCOCAN driver.
    // Check the error code for detail information.
    // error handling
    return;
}
// This is the earliest-possible moment to close the event handle !!!.
CloseHandle( hEvent );
```

4.5. Timestamp

HiCOCAN's firmware provides each message with a timestamp indicating the time (with 1-µs resolution) when the message was received. The timer installed on the board starts with power-on or every time the HCAN2 module is reset at 0 days, 00:00:00.000.000 hours.

4.5.1. 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 can, psTS timestamp)

board	Number of the CAN node for which the event is to be registered.
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.

A timestamp generator is available, but at the moment it is not possible to initialise the counter.

Return values:

HTX_SUCCESS

HTX_ERROR_APPLICATION

4.6. Read/Write Functions

4.6.1. 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; see chapter "Data Structures" on p. 16.

Prototype of the function with the driver for Windows:

DWORD HiCOCANWrite(BYTE Can, psPCCanMsg msg, DWORD Timeout)

Can	Number of the node which is to transmit the message.
Msg	Pointer to message data
Timeout	Specifies the time HiCOCANWrite is waiting for a free entry in the transmit queue when the transmit queue is full. The parameter is entered in 100-ns units. The value 0 specifies an infinite waiting period.

Return values:

HTX_SUCCESS

HTX_OVERRUN

HTX_BUSOFF

HTX_CANERROR

HTX_ERROR_APPLICATION

HTX_ERROR_UNKNOWN_NODE

HTX_ERROR_FULL_QUEUE

HTX_ERROR_ILLEGAL_ID

HTX_ERROR_ILLEGAL_LENGTH

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. With the Windows version the application is waiting until the timeout time has elapsed. If there

is no free entry in the transmit queue until the timeout time has elapsed, the Windows version will also return the HTX_ERROR_FULL_QUEUE error code.

Hint
The timestamp data are taken into consideration.

4.6.2. 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. 16.

Prototype of the function with the driver for Windows:

DWORD HiCOCANRead(BYTE can, psPCCanMsg msg, DWORD Timeout)

can	Number of the desired Can node
msg	Pointer to a structure of the type sPCCanMsg provided by the application.
Timeout	Specified timeout time HiCOCANRead is waiting for the entry of new messages when the receive queue is empty. The parameter is entered in 100-ns units. The zero value stands for an infinite waiting period.

Return values:

HTX_SUCCESS

HTX_OVERRUN

HTX_BUSOFF

HTX_CANERROR

HTX_ERROR_APPLICATION

HTX_ERROR_UNKNOWN_NODE

HTX_ERROR_EMPTY_QUEUE

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 "Timestamp" on p. 36.

If no message is available in the corresponding receive queue the driver waits until the timeout time has elapased. If no message is received during the timeout time, the error code HTX_ERROR_EMPTY_QUEUE is returned; otherwise the received message is returned.

4.6.3. HiCOCANReadEx()

Using the driver's HiCOCANReadEx function, a list of CAN message can be read from the receive queue of a specific CAN node. HiCOCANReadEx enters the list of message data in a structure of the type sPCCanMsg. A corresponding structure needs to be provided by the application (allocated memory), and HiCOCANReadEx 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. 16. ListSize is a pointer to the size of the list. When the function returns, it contains the actual number of CAN telegrams read.

Prototype of the function with the driver for Windows:

DWORD HiCOCANReadEx(BYTE can, psPCCanMsg MsgList, DWORD Timeout, DWORD *ListSize)

can	Number of the desired Can node
MsgList	Pointer to a structure of the type sPCCanMsg provided by the application.
Timeout	Specified timeout time HiCOCANRead is waiting for the entry of new messages when the receive queue is empty. The parameter is entered in 100-ns units. The zero value stands for an infinite waiting period.
ListSize	Size of the list, provided by the application; number of actual entries

Return values:

HTX_SUCCESS

HTX_OVERRUN

HTX_BUSOFF

HTX_CANERROR

HTX_ERROR_APPLICATION

HTX_ERROR_UNKNOWN_NODE

HTX_ERROR_EMPTY_QUEUE

HiCOCANReadEx first checks whether there are messages available in the receive queue of the desired CAN node. If this is the case, the messages are transferred to the list by the application to which the MsgList pointer points. The supplied timestamp specifies the time that the CAN controller received this message. For more detailed information, please refer to section "Timestamp" on p. 36.

The Windows version of the driver waits until the timeout time has elapased. If no message is received during the timeout time, the error code HTX_ERROR_EMPTY_QUEUE is returned; otherwise the received message is returned.

4.7. Status Request

4.7.1. HiCOCANState()

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

no longer supported with HiCO.SH7760

Return values:

HTX_SUCCESS

4.7.2. 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 1 displays the significance of the returned value.
*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.
*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.

Return values:

HTX_SUCCESS

HTX_OVERRUN

HTX_BUSOFF

HTX_CANERROR

HTX_ERROR_APPLICATION

HTX_ERROR_UNKNOWN_NODE

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	No difference to Bit position 0
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 1: Significance of the returned value

4.7.3. 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_OVERRUN

HTX_BUSOFF

HTX_CANERROR

HTX_ERROR_APPLICATION

HTX_TRANSMITQUEUE_FULL

HTX_RECEIVEQUEUE_EMPTY

Number of the messages entered

4.8. 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.

4.8.1. 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 [1].

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	unsupported
Code	Value of the acceptance code register; For details please have a look at [1].
MaskReg	Value of the acceptance mask register; For details please have a look at [1].

Return values

HTX_SUCCESS

HTX_OVERRUN

HTX_BUSOFF

HTX_CANERROR

HTX_ERROR_APPLICATION

HTX_ERROR_UNKNOWN_NODE

4.8.2. HiCOCANSetBaud()

DWORD HiCOCANSetBaud(BYTE can, BYTE rate)

takes the values for the bus timing register from the registry. 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_BAUD800K 80 #define HiCOCAN_BAUD1M 100

Return values:

HTX_SUCCESS

HTX_OVERRUN

HTX_BUSOFF

HTX_CANERROR

HTX_ERROR_APPLICATION

HTX_ERROR_UNKNOWN_NODE

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

4.8.3. HiCOCANSetTimingReg()

This function allows you to set the timing registers of a CAN controller manually.

No longer supported with HiCO.SH7760

since these entries are now controlled by the registry and the function HiCOCANSetBaud().

Return values:

HTX_ERROR_NOT_SUPPORTED

4.8.4. HiCOCANParameter()

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

No longer supported with HiCO.SH7760

Return values:

HTX_ERROR_NOT_SUPPORTED

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.
This application cannot be compiled within the platform builder environment. It's a separate VS2005 based C++ application which includes statically the MFC libraries. Before compiling, it is necessary to install the SDK for the used kernel.

5.1. Detailed Description

After starting the application, the required settings are taken for using the two CAN nodes:

1. Open the driver
2. Open the nodes
3. Start the nodes

After this, three working threads will be created. Two of these threads each serve one CAN node. The message will then be sent between these CAN nodes in a "ping-pong" manner.

For this, the first thread ("PingThread") initiates the transfer by creating a message and sending it via the first node.

The second thread ("PongThread") receives this message via the second CAN node, modifies the identifier and retransmits the message via the second CAN node.

The PingThread receives the message from the first node, and updates the output items of the dialog.

The PingThread retransmits the message via the first CAN node to the PongThread, which is already waiting for an incoming message.

The third Thread cyclically polls the status of the two nodes and displays the information obtained in the application window.

The program will be executed until the user closes the window. A set flag causes the threads to close themselves. In addition, the driver is closed and the program left.

6. Troubleshooting

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] Hardware manual, Hitachi SuperHTM RISC engine SH7760 HD6417760BP200D.
[2] Starterkit WindowsCE 5.0 for HiCO.SH7760.
[3] CAN Monitor User Guide V2.1, Emtrion GmbH (former Hitex), 2000