

HART Slave

System Interface for the Hardware Environment and the PC-Simulation

Borst Automation

Walter Borst

Im Wingert 4

D-65626 Fachingen

GERMANY

Voice: +49 (0) 6432 989176

Fax: +49 (0) 6432 989129

Email: info@borst-automation.com

Home: <http://borst-automation.com>

3.9.2007

Revision History:

| | | |
|----------|-----------|-----------------------------------|
| W. Borst | 6.9.1999 | First Version |
| W. Borst | 29.7.2000 | Additions |
| W. Borst | 29.1.2006 | Minor Changes / English Version |
| W. Borst | 3.9.2007 | More Details of the PC-Simulation |

Content

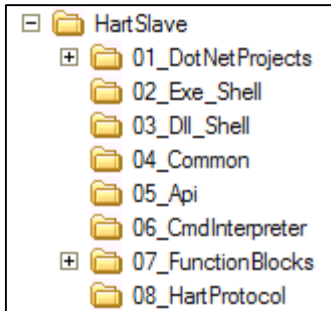
| | | |
|----------|---|---|
| 1 | OVERVIEW | 3 |
| 2 | PC-SIMULATION | 4 |
| 2.1 | SET BAUDRATE AND COMPORT..... | 4 |
| 2.2 | SET SLAVE ADDRESS..... | 4 |
| 2.3 | RUN THE SIMULATION..... | 4 |
| 2.4 | PRINCIPLE OF OPERATION..... | FEHLER! TEXTMARKE NICHT DEFINIERT. |
| 2.5 | PROPOSED MODIFICATIONS FOR TEST PURPOSES..... | 4 |
| 2.5.1 | <i>Number of Preambles</i> | 4 |
| 2.5.2 | <i>Device Busy</i> | 4 |
| 2.5.3 | <i>Unique Identifier</i> | 4 |
| 3 | THE HART DRIVER (EMBEDDED AND PC) | 5 |
| 3.1 | EXPOSED FUNCTIONS TO BE SERVED BY THE ENVIRONMENT:..... | 5 |
| 3.2 | MACROS REQUIRED BY THE DRIVER(PROTOCOL LAYER)..... | 6 |
| 3.3 | MACROS USED BY THE FUNCTION BLOCK MODEL..... | 7 |
| 4 | GLOBAL VARIABLES | 9 |
| 4.1 | STRUCTURES..... | 9 |
| 4.1.1 | <i>strHrtComSM</i> | 9 |
| 5 | FUNCTIONS | 10 |
| 5.1 | HRT5MSCYCLE()..... | 10 |
| 5.2 | HRTGETCHAR()..... | 10 |
| 5.3 | HRTTRANSMDONE()..... | 10 |
| 5.4 | HRTINIT()..... | 11 |
| 6 | LIST OF FILES | 11 |
| 7 | CONFORMANCE TESTING | 12 |
| 7.1 | DATA LINK LAYER..... | 12 |
| 7.2 | APPLICATION LAYER..... | 13 |
| 8 | HART 6.0 | 13 |

1 Overview

The implementation of the HART driver is still based on HART 5.x, which was the established standard through a lot of years. Anyhow, see chapter for further details.

The HART slave serves two purposes. It can be used as a kick off to develop HART communication software for an embedded system and it could be used as a test platform.

For the usage as a test platform a Visual Studio (.Net 2003) solution is provided which is structured as it follows.



The folder 01_Exe_Shell contains the sources required to build an executable (HartDevSimExe), which is a simple console application.

The console application is loading a DLL built by the second project (HartDevSimDll).

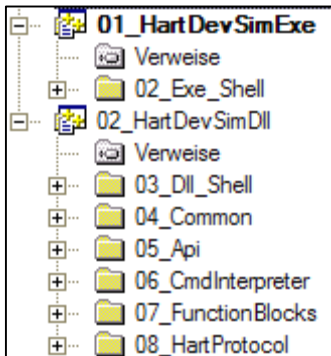
01_Dll_Shell contains the sources of the framework of the windows dll. In 02_Common files will be found, which are shared by some of the modules.

03_Api collects the files implementing the internal interfaces. In 04_CmdInterpreter the sources for the HART command interpreter are summarized.

The sources in 05_FunctionBlocks are the application itself, which is served by the command interpreter.

The lower layer stuff - the HART protocol - is finally stored in 06_HartProtocol.

The directory is structured to meet the requirements for testing in different environments.



There are three main directories. #TestBench is used for the testing itself. All necessary executables and dlls are placed in this folder. Because these programs has to work together it is of advantage to have them all in one directory. To achieve this the VS projects contains a custom build which copies the files in charge to the #TestBench.

Documetation is containing this file and eventual help files and other documents.

The subdirectories in the folder Implementation reflects the structure of the Visual Studio projects, at least as far as possible.

When this package was developed in the years 1999 and 2000 the task was to find out, if the source code for a HART command interpreter together with a function block shell could be generated by a Microsoft Access data base. The result of the project was that it works. But it also turned out that generating source code in such a depth is not worth the effort, which has to be spent.

Therefore the structure of the files is very formal and constant. But this is no longer needed. Please ignore the advise 'DO NEVER MAKE CHANGES IN THIS FILE MANUALLY!' in some of the headers.

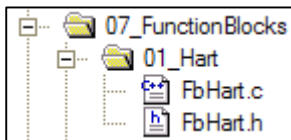
2 PC-Simulation

2.1 Set Baudrate and ComPort

To set the right baud rate and the com port, changes have to be made in the file HartApi.h in the project folder 03_Api, line 38 ff:

```
//User definitions
#define COMPORT      1
#define BAUDRATE    1200
```

2.2 Set Slave Address



The slave address is part of the data of the HART Function Block. The value is initialized in the C-source implementing this function block. The file is found in the subdirectory 01_Hart in the folder 05_FunctionBlocks.

The relevant source code in line 169 ff reads:

```
/*
 * Short address for HART communication
 */
EEPROM  UINT8 ee_ui8FbHartAddress;
RAM     UINT8 re_ui8FbHartAddress;
RAM     UINT8 rl_ui8FbHartAddress = 0;
```

2.3 Run the Simulation

Start HartDevSim.exe in the application directory.

2.4 Proposed Modifications for Test Purposes

2.4.1 Number of Preambles

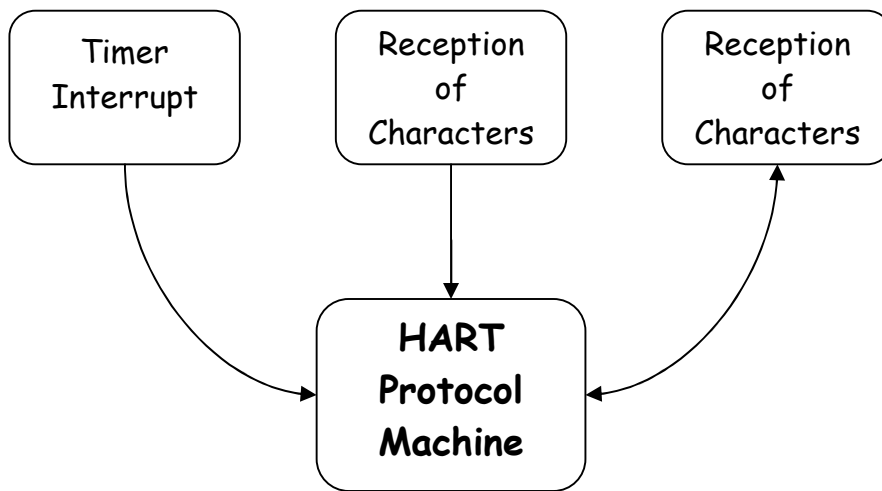
2.4.2 Device Busy

2.4.3 Unique Identifier

3 The HART Driver (Embedded and PC)

The driver needs basically two functionalities from the embedded environment. Because it is based on a time oriented state machine it is providing a function which has to be called every 5 ms. This can be done from a timer interrupt routine.

The connection to the communications can be done in two ways. The first way is to call the driver with the reception of each single character. Alternatively the driver may also be called after the reception of a whole frame. The transmission of the response is handled by a common Tx function.



3.1 Exposed Functions to be Served by the Environment:

| Name | Function | Comment |
|------------------------------|--|--|
| <code>hrt5msCycle()</code> | Anchor function of the state machine of the protocol stack. Has to be called in a 5 ms cycle. | |
| <code>hrtGetChar()</code> | Has to be called on reception of a character. | |
| <code>hrtTransmDone()</code> | Has to be called if the last stop bit was transmitted. | This function call is not needed if disable and enable reception and transmission is handled elsewhere. |
| <code>hrtInit()</code> | Initializes the driver. Has to be called on reset after the initialization of the embedded hardware. | |

3.2 Macros Required by the Driver(Protocol Layer)

| Name | Function | Returns |
|---|--|--|
| M_HRT_START_SEND (BYTE* byBuf, BYTE byLen) | Starts sending of a data frame. | VOID |
| M_HRT_PUSH_REGS | Saves relevant processor registers. | VOID |
| M_HRT_POP_REGS | Restores relevant processor registers | VOID |
| M_HRT_DISABLE_RCV | Disable receiver. | VOID |
| M_HRT_ENABLE_RCV | Enable receiver. | VOID |
| M_HRT_INIT_COM | Initializes the communications hardware, is called in hrtInit(). | VOID |
| M_HRT_IS_CARRIER | Detects if a carrier is on the line. | TRUE, FALSE |
| M_HRT_GET_RCV_ERROR | Returns error details on reception of a character. | ENUM 0: No Error Bit 6: Parity Bit 5: Overrun Bit 4: Framing |
| M_HRT_GET_RCV_CHAR | Returns the received character. | UCHR |
| M_HRT_ENABLE_INTERRUPT | Enable interrupts. | VOID |
| M_HRT_DISABLE_INTERRUPT | Disable interrupts | VOID |

3.3 Macros Used by the Function Block Model

| Name | Function | Returns |
|--|---|-------------------------|
| M_API_IS_EE_VALID | Checks if EEPROM is valid (Checksum). | TRUE, FALSE |
| M_API_EE_WRITE (EEPROM void* pDst, const void* pSrc, BYTE byLen) | Writes data to the EEPROM. | VOID |
| M_API_EE_READ (void* pDst, EEPROM void* pSrc, BYTE byLen) | Reads data from the EEPROM. | VOID |
| M_API_TRIGGER_WATCHDOG | Triggers the watchdog. | VOID |
| M_API_IS_WR_PROTECTED | Checks if hardware write protect is set. | TRUE, FALSE |
| M_API_PUT_UINT16 (void* pDst, UINT16 uiVal) | Encodes data for the communication, note: this depends on the endian for the target code. | VOID |
| M_API_PUT_UINT24 (void* pDst, UINT24 ui24Val) | | VOID |
| M_API_PUT_UINT32 (void* pDst, UINT32 uiVal) | | VOID |
| M_API_PUT_FLOAT (void* pDst, FLOAT flVal) | | VOID |
| M_API_GET_UINT16 (void* pSrc) | Decodes data from a buffer, note: this depends on the endian for the target code. | UINT16 (*) ¹ |
| M_API_GET_UINT24 (void* pSrc) | | UINT24 (*) |
| M_API_GET_UINT32 (void* pSrc) | | UINT32 (*) |
| M_API_GET_FLOAT (void* pSrc) | | FLOAT (*) |

¹ (*) = Betrifft nur Applikation, derzeit nicht implementiert.

| Name | Funktion | Returns |
|--|--|----------|
| <pre>M_API_MEMCPY (void* pDst const void* pSrc BYTE byLen) </pre> | Copy a stream of octets. | VOID |
| <pre>M_API_SWAP2 (UCHR* pSrcDst) </pre> | Swap 2 byte | VOID |
| <pre>M_API_SWAP3 (UCHR* pSrcDst) </pre> | Swap 3 byte | VOID |
| <pre>M_API_SWAP4 (UCHR* pSrcDst) </pre> | Swap 4 byte | VOID |
| <pre>M_API_IEEEtoMSfloat (void* pSrcDst) </pre> | Converts from IEEE 754 to Microsoft float format | VOID |
| <pre>M_API_MSfloatToIEEE (void* pSrcDst) </pre> | Converts from Microsoft float format to IEEE 754 | VOID |
| <pre>M_API_PACK_ASCII (UCHR* pucDst, BYTE* pbyDstLen, UCHR* pucSrc, BYTE bySrcLen,) </pre> | Converts from visible string to Packed ASCII | VOID (*) |
| <pre>M_API_UNPACK_ASCII (UCHR* pucDst, BYTE* pbyDstLen, UCHR* pucSrc, BYTE bySrcLen,) </pre> | Converts from Packed ASCII to visible string | VOID (*) |

4 Global Variables

4.1 Structures

4.1.1 strHrtComSM

```
typedef struct tagComSM
{
    BYTE byState;
    BYTE byCount;
    BYTE byMasterType;
    BYTE byPollCount;
    BYTE byCmd;
    BOOL bAddrErr;
    BYTE byErr;
    BYTE byPosRspl;
    BYTE byPAtoSnd;
    BYTE byPAcount;
    BYTE byData[HRT_MAX_IO_BUFFER];
} T_COM_SM;

T_COM_SM      strHrtComSM;          /* IO state machine */
```

| Name | Wertebereich | Bedeutung |
|--------------|--|--|
| byState | HRT_IO_IDLE HRT_WAIT_NEXT_PREAMBLE HRT_RCV_WAIT_DEL HRT_RCV_READING HRT_SEND_PENDING HRT_SENDING HRT_IO_LOCKED | States of the low level machine |
| byCount | 0..(MAX_IO_BUFFER-1) | Number of characters received or number of characters to be sent. |
| byMasterType | HRT_PRIM_MASTER HRT_SEC_MASTER | HART supports two masters. |
| byPollCount | 0..HRT_MAX_POLL_COUNT | Number of 50 ms cycles the low level is waiting for the application until a busy response is sent. |
| byPAcount | 0..20 | Number of already sent preambles. |
| byData[] | 0..255 | Recive and transmit buffer. |

| Name | Wertebereich | Bedeutung |
|-------------|--|---|
| ByErr | HRT_ERR_BUFFER_OVERFLOW HRT_ERR_LONG_PARITY HRT_ERR_FRAMING HRT_ERR_OVERRUN HRT_ERR_VERT_PARITY HRT_ERR_SUMMARY HRT_ERR_NONE | Error flags of the receiving procedure. |
| byPosRspl | HRT_SHORTF_RSPCODE1_POS HRT_LONGF_RSPCODE1_POS | Position of response code 1 in the data frame. |
| ByPAtoSnd | 5..20 | Number of preamble to be sent. |
| ByPollCount | 0..HRT_MAX_POLL_COUNT | Number of 50 ms cycles the protocol is waiting for the application. |
| ByCmd | 0..255 | Command of the currently active service. |
| BAddrErr | TRUE..FALSE | Flag to signal an error within the address area in the frame. |

5 Functions

5.1 hrt5msCycle()

```
VOID hrt5msCycle(VOID);
```

This function is called in a 5 ms cycle by a timer interrupt. This function is driving the state machine of the HART protocol.

Depending on the implementation this function is using sysPushRegs and sysPopRegs to store and restore the actual context.

5.2 hrtGetChar()

```
VOID hrtGetChar(VOID);
```

Processes a received character and eventual pending errors. For more run time efficiency this function may be placed inline in the UART interrupt.

5.3 hrtTransmDone()

```
VOID hrtTransmDone(VOID);
```

Has to be called when the stop bit of the last character was sent.

5.4 hrtInit()

```
VOID hrtInit(VOID);
```

This function initializes the driver. It is called upon reset.

6 List of Files

| Verzeichnis | Name | Auto ² | Bedeutung |
|-------------|-----------------|-------------------|---|
| Api | UserApi.c/.h | No | Target system interface: PC or Target. |
| | HartApi.c/.h | No | Target system interface for the HART communication. |
| Common | Convert.c/.h | No | Miscellaneous conversions. |
| | Hart.h | No | HART specific declarations |
| | MemDecls.h | No | Definition of memory classes. |
| | Types.h | No | Basic types |
| | PcComPort.c/.h | No | Implementation for a die PC Simulation (via UserApi) ³ |
| | FbInit.c/.h | Yes | Example: Initialization of the function blocks. |
| FbHart | FbHart.c/.h | Yes | Example: Hart function block |
| HartTest | HartTest.cpp/.h | No | Console application for a PC simulation. |
| | TestDll.c/.h | No | Windows DLL for a PC simulation ⁴ |
| HrtSrvIn | HrtSrvIn.c/.h | Yes | Example: The command interpreter. |
| Iout | Iout.c/.h | Yes | Example: Function block current out. |
| Meas | Meas.c/.h | Yes | Example: Integration of the measurement. |
| Protocol | HartLoL.c/.h | No | HART Data Link Layer Protocol |

² Auto = Generated from Access script

³ Not required for the target project

⁴ Not required for the target project

7 Conformance Testing

7.1 Data Link Layer

The HART conformance test 5.0 conducted with the PC simulation showed the following test results:

```
device passed HART CONFORMANCE test DT00001
device passed HART CONFORMANCE test DT00002
device passed HART CONFORMANCE test DT00004
device passed HART CONFORMANCE test DT00005
device passed HART CONFORMANCE test DT00006
device passed HART CONFORMANCE test DT00007
device passed HART CONFORMANCE test DT00008
device passed HART CONFORMANCE test DT00009
```

```
device passed HART CONFORMANCE test DT00010
device passed HART CONFORMANCE test DT00012
device passed HART CONFORMANCE test DT00013
```

```
device passed HART CONFORMANCE test DT00014
device passed HART CONFORMANCE test DT00015
device passed HART CONFORMANCE test DT00017
device passed HART CONFORMANCE test DT00018
```

A 0xFF was seen immediately following an ACK at byte number 78.

It appears to have been sent by the slave. This is not recommended but does not cause a fail.⁵

A 0xFF was seen immediately following an ACK at byte number 119.

It appears to have been sent by the slave. This is not recommended but does not cause a fail.

A 0xFF was seen immediately following an ACK at byte number 278.

It appears to have been sent by the slave. This is not recommended but does not cause a fail.

```
device passed HART CONFORMANCE test DT00020
device passed HART CONFORMANCE test DT00024
device passed HART CONFORMANCE test DT00032
device passed HART CONFORMANCE test DT00033
device passed HART CONFORMANCE test DT00034
```

The slave device indicated that it does not support command 59 in frame number 4.

HART CONFORMANCE Test DT00035 appears to not apply to the slave device

⁵ The character 0xff appears due to problems of the modem switching off the carrier.

7.2 Application Layer

This test is not required for the delivery of the low level machine. We could do this test in your project if this is required. Please ask for a separate quote.

8 HART 7.0

The driver was developed for HART 5.0. It is compatible with HART 7.0 in that way that the correct information about the version is transmitted to the host.

If you need an implementation of HART 7.0 please contact:

Borst Automation

Im Wingert 4

65626 Fachingen

Tel.: 06432 989176

Fax: 06432 989129

Email: weehborst@aol.com

Home: <http://www.hart-profibus.com>