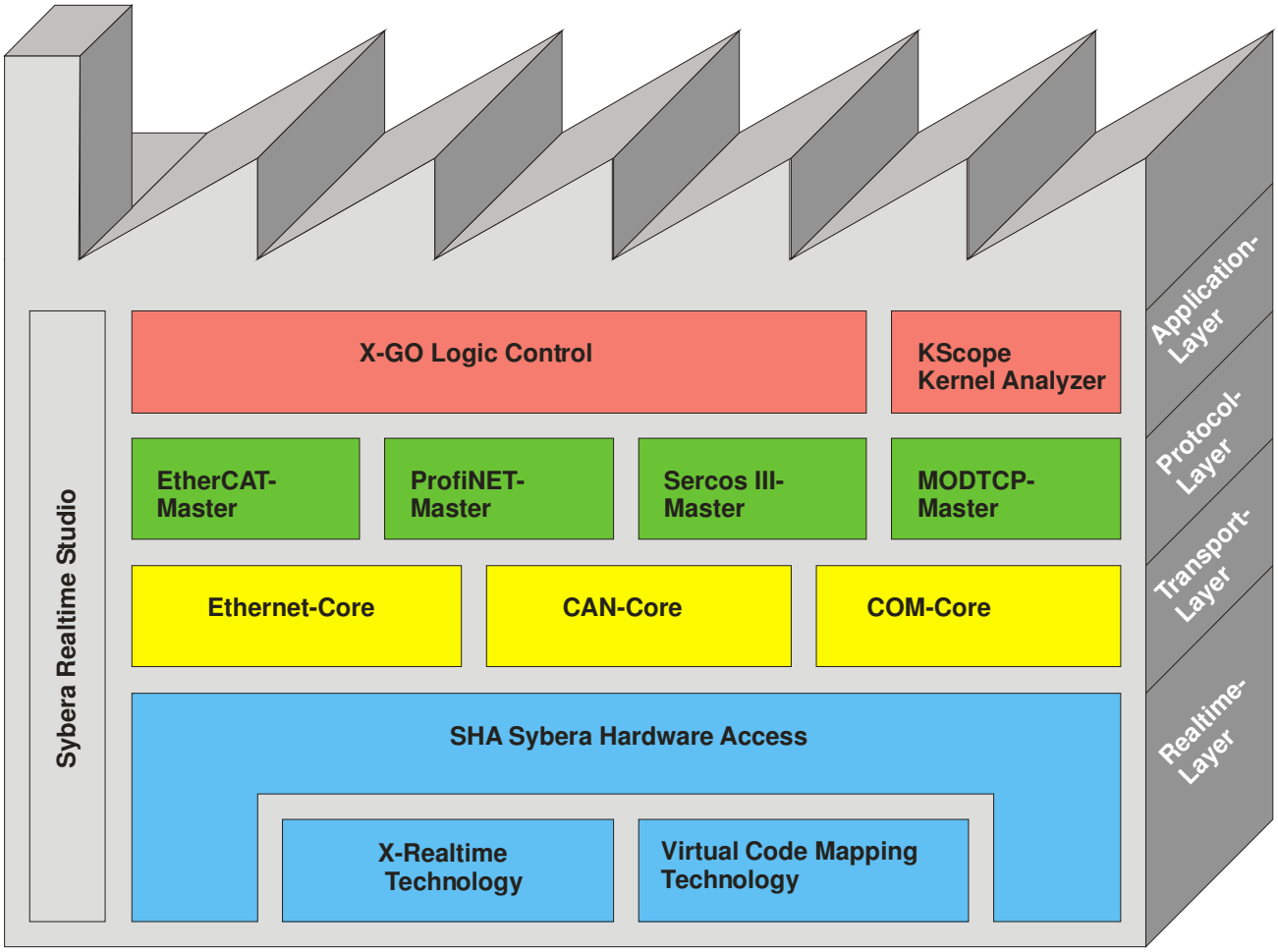

TM

# *Serial Realtime Core Documentation*

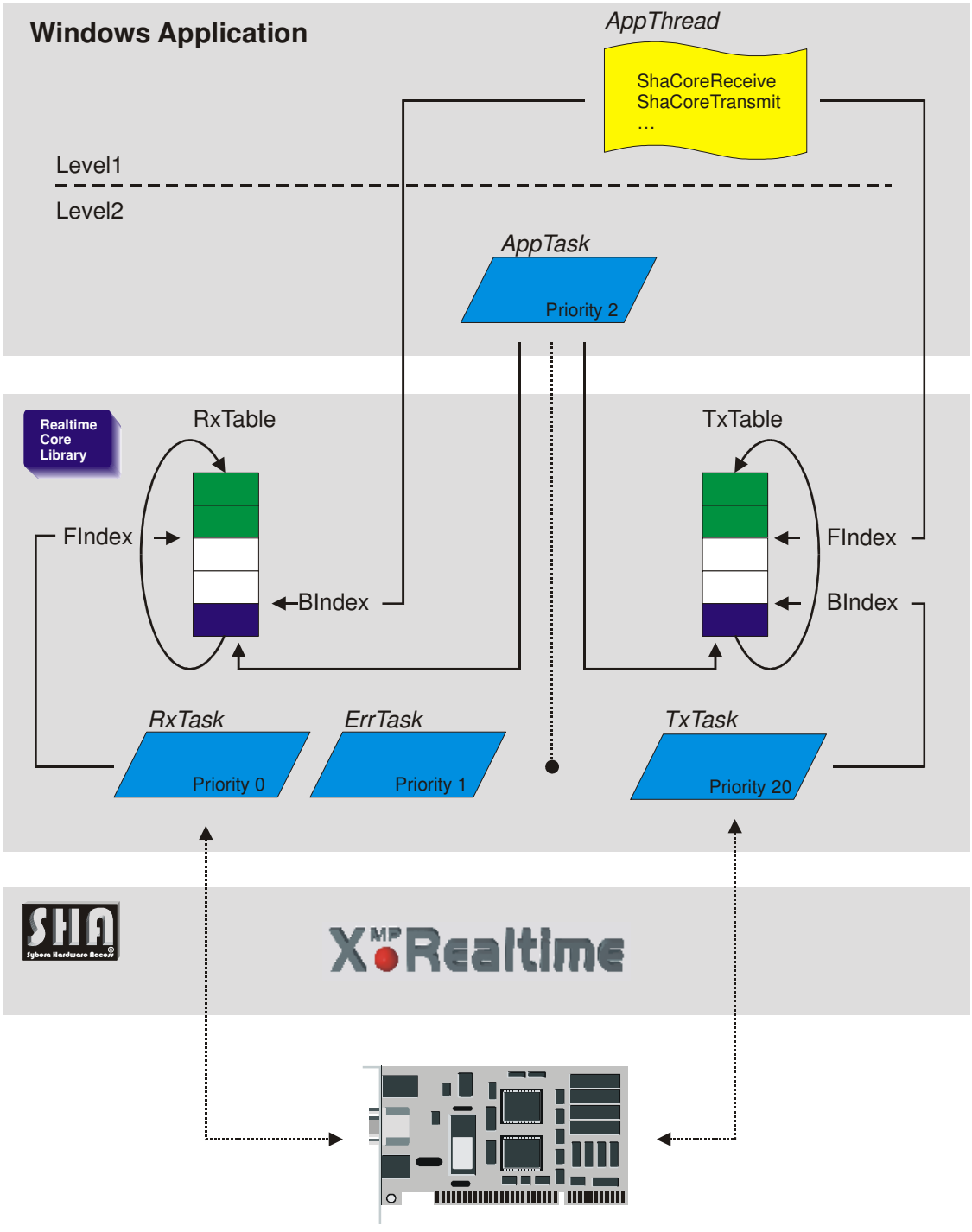
Date: Sept, 17.2012

# 1 Introduction

The idea of further interface abstruction of the SHA X-Realtime for several communication channels and bus systems, like serial communication, CANBUS, Ethernet (TCP/IP), … is realized by the SYBERA AddOn Software Moduls, so called RealtimeCores. All Realtime-Cores are based on the SHA X-Realtime system. The RealtimeCores are intended to fullfill Realtime-Level-1, which means collecting and buffering data in realtime without loss of data, as well as Realtime-Level-2, which means functional operation at realtime. Thus the RealtimeCores usually require simple passive harware. One of the great benefits is the adjustable scheduling time of incoming and outgoing data.



The AddOn COM Realtime library allows handling of serial data in realtime. At realtime level 1 incoming and outgoing data will be buffered inside RX and TX ringbuffers, controlled by forward and backward indexing. A simple proprietary core interface, as well as the standard COM interface is available for communication with the windows application. Additional functional operation is possible at realtime level 2. Therefore a realtime task can be setup inside the application. The data exchanges is handled via shared memory area.



## 1.1 Supported Platforms

- Visual C++ (from Version 6.0 with ServicePack 5)
- Borland C++Builder
- CVI LabWindows
- Borland Delphi

## 1.2 Supported Hardware

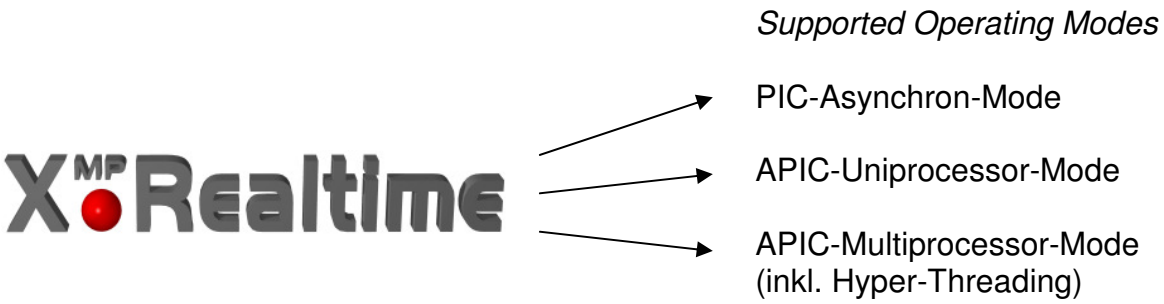
- PC Serial Internal
- OXFORD PCI (not all types)
- OXFORD PCMCIA (not all types)
- QUTECH PCI (not all types)
- QUATECH PCMCIA (not all types)

## 1.3 Supported OS

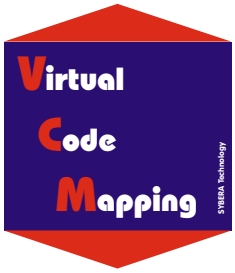
Windows 2000, XP, VISTA, 7 (32 Bit)

## 2 X-Realtime Technology

Basic of the SHA software is the realtime subsystem, called X$^{MP}$-Realtime-Engine. With the new X$^{MP}$-Realtime-Engine, SYBERA opens a new dimension to the realtime control under Windows. With support of multiprocessor-platforms, the realtime behaviour is clearly improved and the overall-performance was increased. On this occasion, the new X$^{MP}$-Realtime-Engine exclusivly reserves a physical or logical processor for the realtime operation. Besides pure multiprocessor platfroms also the INTEL hyperthreading technology of the PentiumIV processor is fully supported.

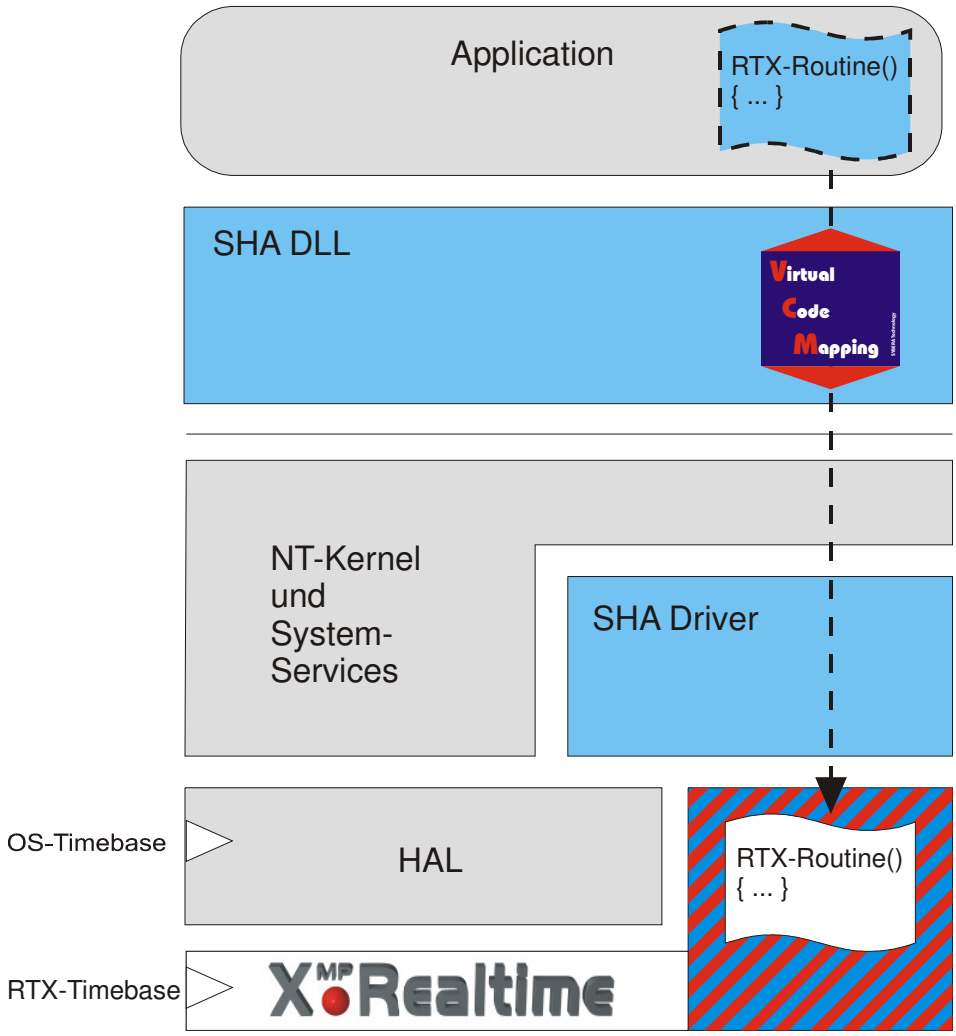



The subsystem is asynchronously coupled with ist on scheduler clock, so that both systems (SubSystem and OS) are working almost independently. The lock mechanism of multiprocessor control is administered internally, so that the existing SHA-Interface remains unchanged. Additionally the system supports APIC interrupt control and switches automatically into the right operating mode.




A further implemented mechanis is called "Virtual Code Mapping". This mechanism allows placing a realtime routine or a interrupt service routine inside any application-project. These routines will be decoded and mapped to the SHA subsystem at runtime.

X-Realtime allows non-preemptive realtime multitasking with multiprocessor support. The system automatically recognizes which platform is present and switches to the correct operating mode. When Hyperthreading is present, a logical processor will be claimed for the realtime control.




Conventional realtime-subsystems usually work with a synchronized scheduling mechanism for the realtime subsystem and the OS, which usually shows a bad jitter behaviour at high OS load. The X-Realtime Engine works asynchronously with separated clock sources that clearly leads to a better jitter behavior and thereby realizes a complete decoupling of realtime-task to the existing operating system.

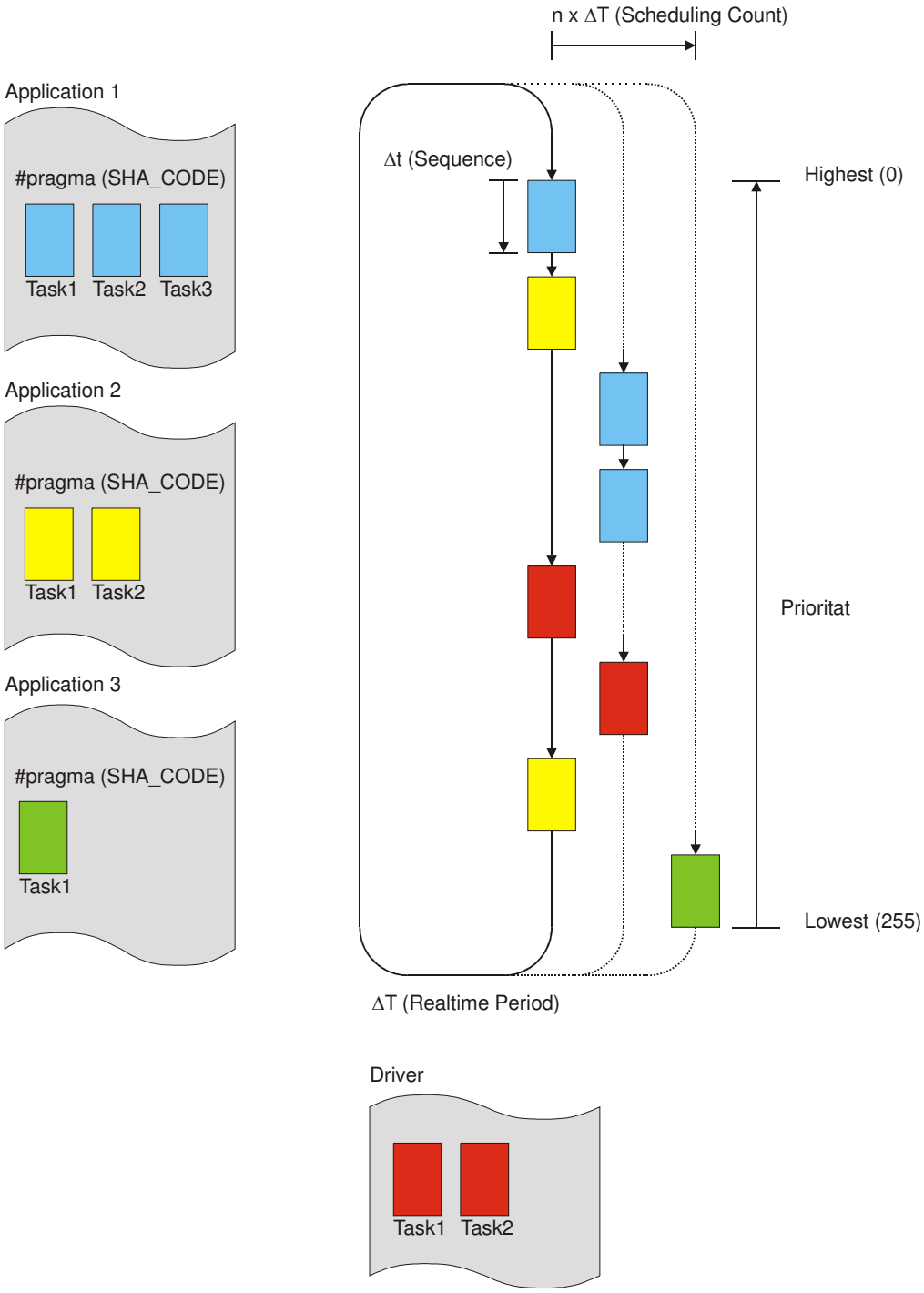
With the X-Realtime Engine, realtime task cycles are realizable upto 10 µsec (100 KHz) sampling rate. An integrated watchdog-system controls the realtime task and determines the remaining task-time. The SHA X-Failsafe-System offers additionally the possibility to keep a rescue task busy or to proceed a controlled shutdown, even on heavy exception errors (for example Blue-Screen). With the X-Failsafe-System, for example a robot-arm can be driven out from a hazard zone and an alarm signal is caused.

The realtime routine has to be equal to a RING0 EXECUTION routine for interrupt control (see Interrupt Access Module), however without a return value and it‘s not depending on the system load. With the X-Realtime routine the same programming methods and restric tions are valid like on each other RING0 EXECUTION routine.

With the X-Realtime system several tasks can be programmed within an application or within a device driver and will be automatically mapped to the X-Realtime system layer at runtime. Every task can be setup with its own scheduling cycle which interacts independ-ently to any other task cycles. Additionaly each task can given and changed its own priority dynamically. So several applications with their own realtime tasks can run at once. To-gether with application task also device drivers can setup their own realtime tasks to run within the X-Realtime system.

*Note:*
X-Realtime may not run with Firewall-Protection or Virus-Scan Software. Please disable / deinstall such protection programms.

## 2.1 X-Realtime Multitasking



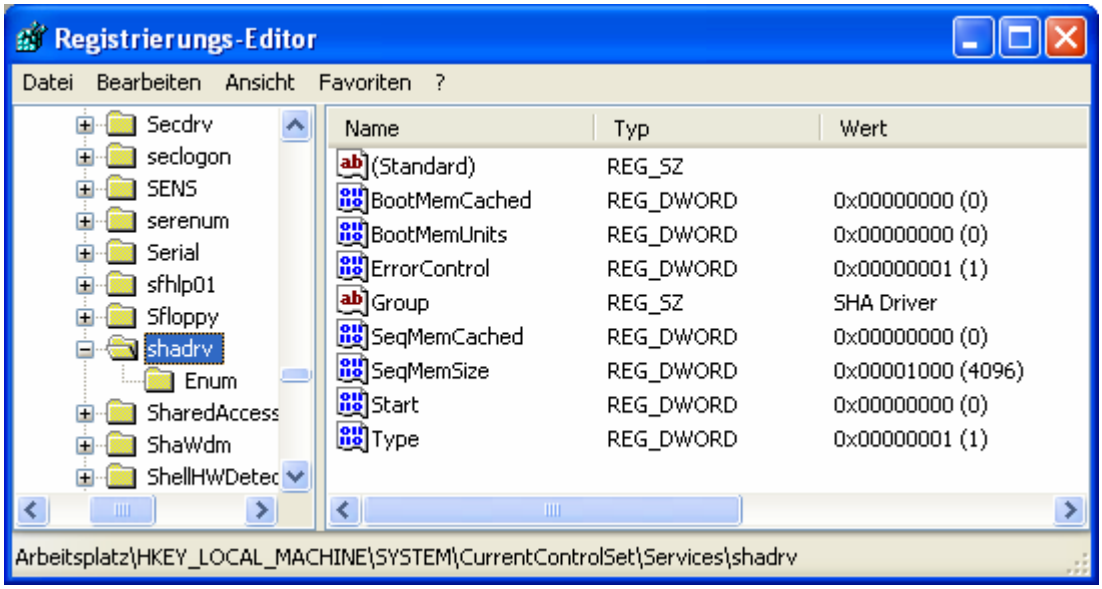
# 3 Core Installation

For installation following steps are required:

- First install the Library SHA Ver. 11.20 (or higher)
  (make sure the directory path has no space characters)
- Next install the RealtimeCore PORT-Driver
- Next run the program SYSETUP of the COM library
  (make sure the directory path has no space characters)
- Optional: Check license with SYLICENCECHECK.EXE
- Build your program with the library interface
- Run the program

After start of SYSETUP the PEC information (user name, company and keycode) must be entered. The keycode for the demo version is: 00001111-22223333

*Note:*
After finishing installation, you must reboot your PC before starting the compiler !!!. The Ethernet Realtime Core is not running with the SHA DynamicLoad module. Depending on the license type, a new PEC code is required. For PEC information please contact SYBERA. The registry key shadrv must be permanent:

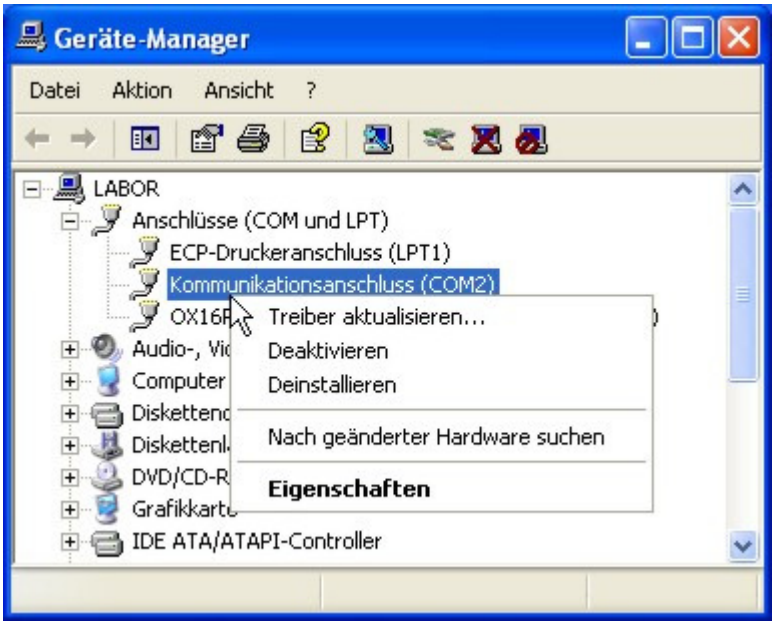


## 3.1 COM RealtimeCore Driver Installation

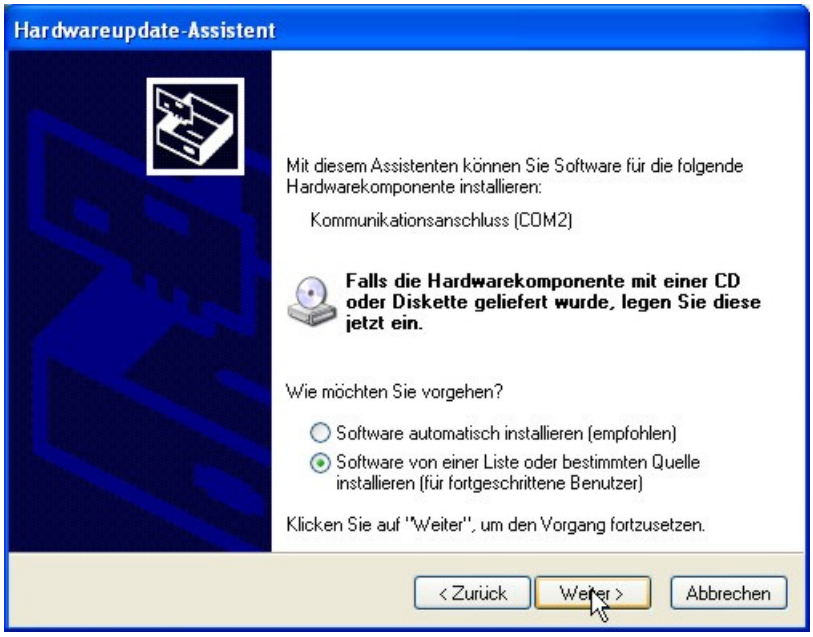
The "COM RealtimeCore" is based on the PC Serial Ports, as well as on QUATECH and OXFORD PCI Adapters (also PCMCIA) and will be installed as a PORT driver.

### 3.1.1 Driver installation for PC Serial Ports

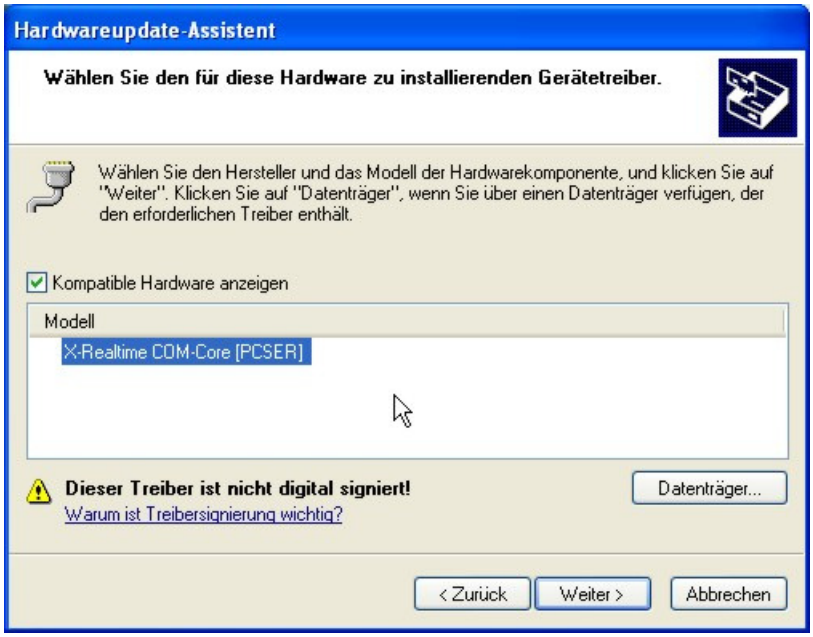
For an existing standard serial port, the driver must be updated with the SHA Realtime COM Realtime Core Driver inside the Windows Device Manager.



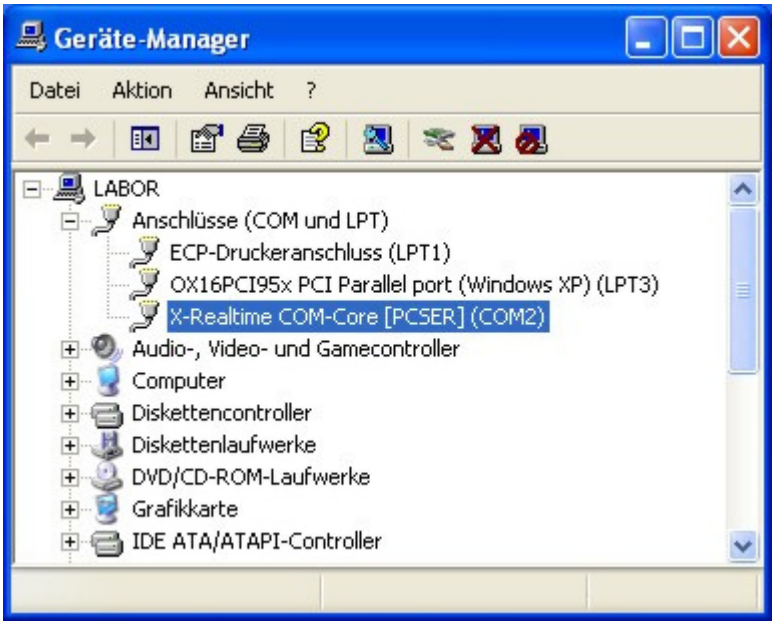
The new driver must be installed manually




Therefor the path to the drivers XRTCOM.INF file must be given




Ignore the warning of the missing digital sign

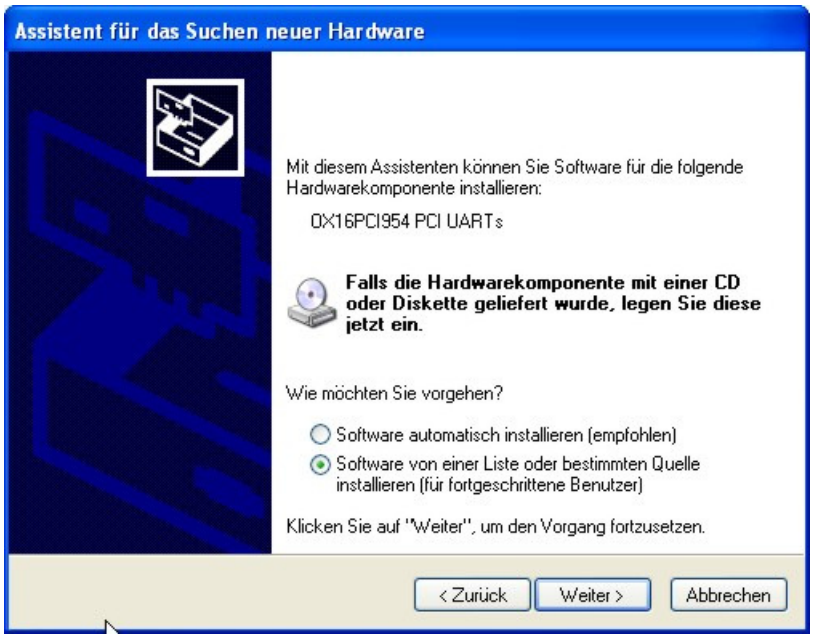



Check, if the new driver was installed correctly. If the driver is not running, reboot the system and check again.

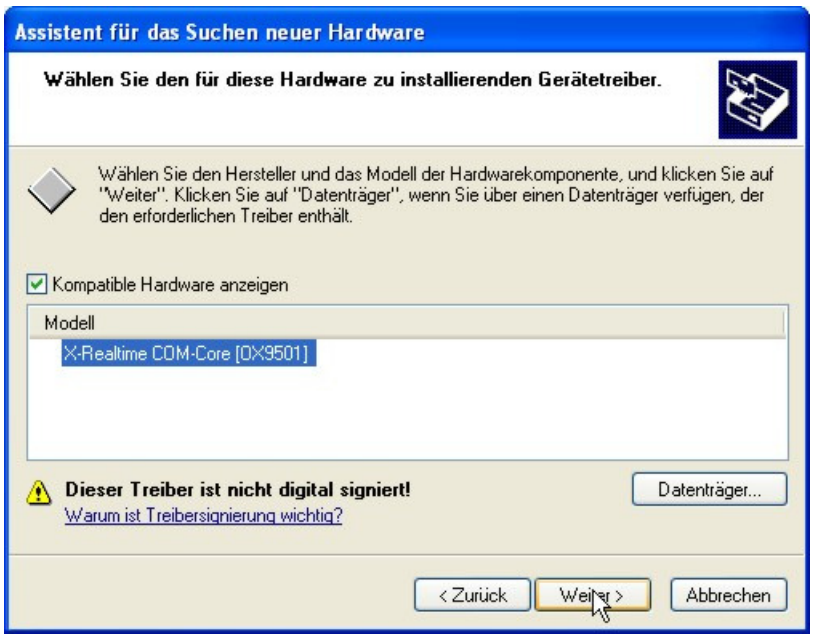
### 3.1.2 Driver installation for PCI Adapters

When installing the PCI-Adapter, the Windows Hardware Wizard asks for a PNP device driver. Therefor the COM Realtime Core Driver has to be installed

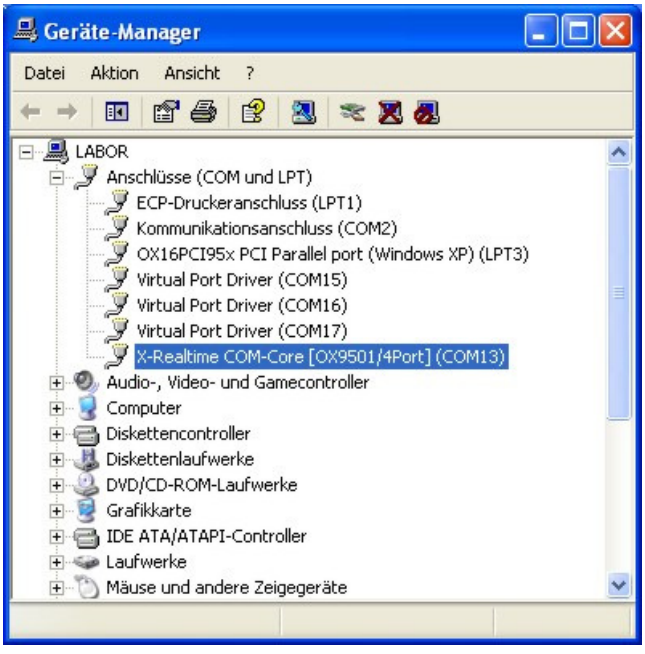
The new driver must be installed manually




Therefor the path to the drivers XRTCOM.INF file must be given

Since some adapter have more than one COM port, for each additional port a Virtual Port Driver has to be installed. Ignore the warning of the missing digital sign.

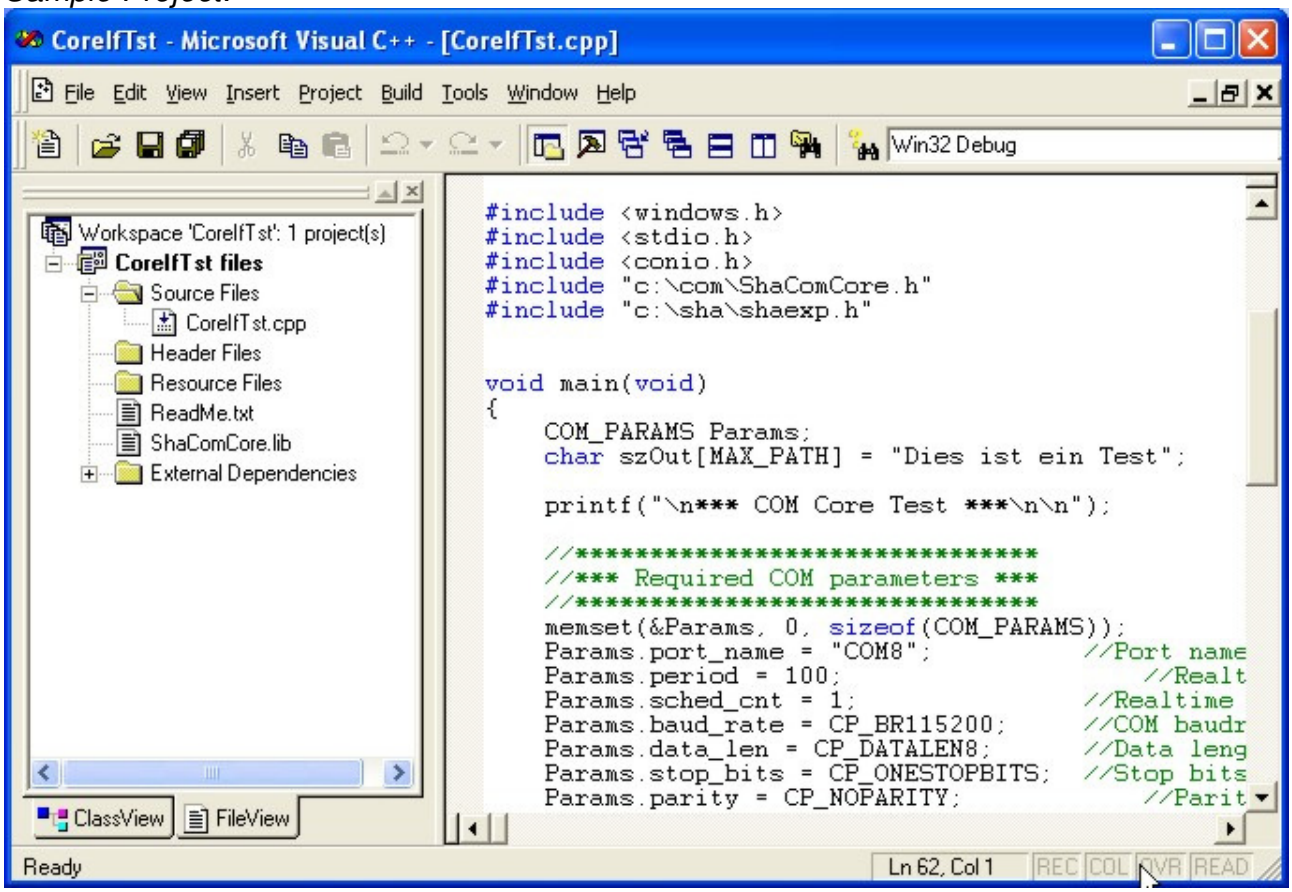




Check, if the new driver was installed correctly. If the driver is not running, reboot the system and check again.

# 4 COM Realtime Core Library

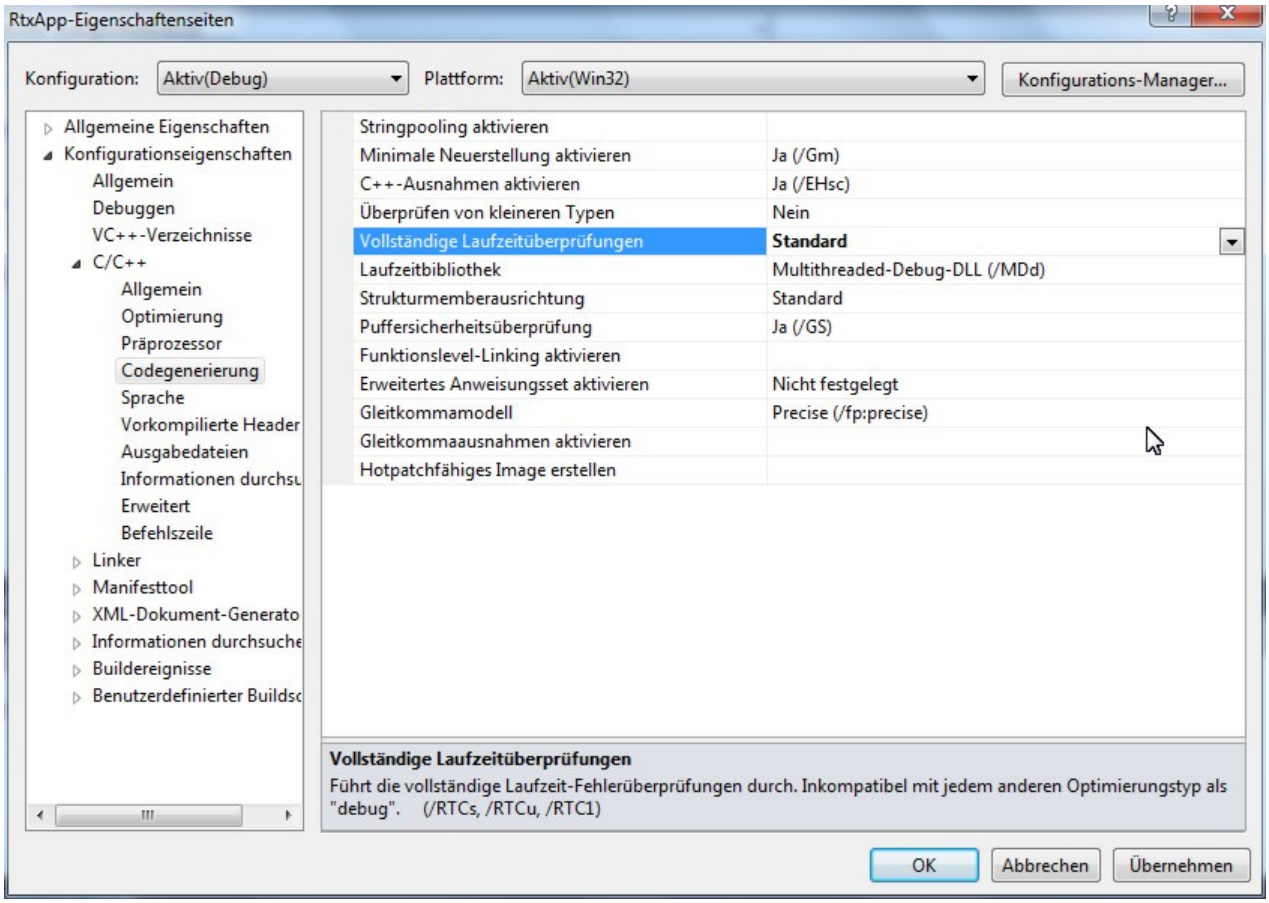
The AddOn realtime library "COM Realtime Core" allows sending and receiving of serial data, as well as the functional operation of data in realtime. The COM Realtime Core library allows handling of serial data in realtime level 1 (ring buffered serial bytes) or realtime level 2 (functional handling within realtime task). The COM realtime core allows making use of the standard COM interface of Windows, without changing the software, or programming via a simple proprietary interface.

*Sample Project:*



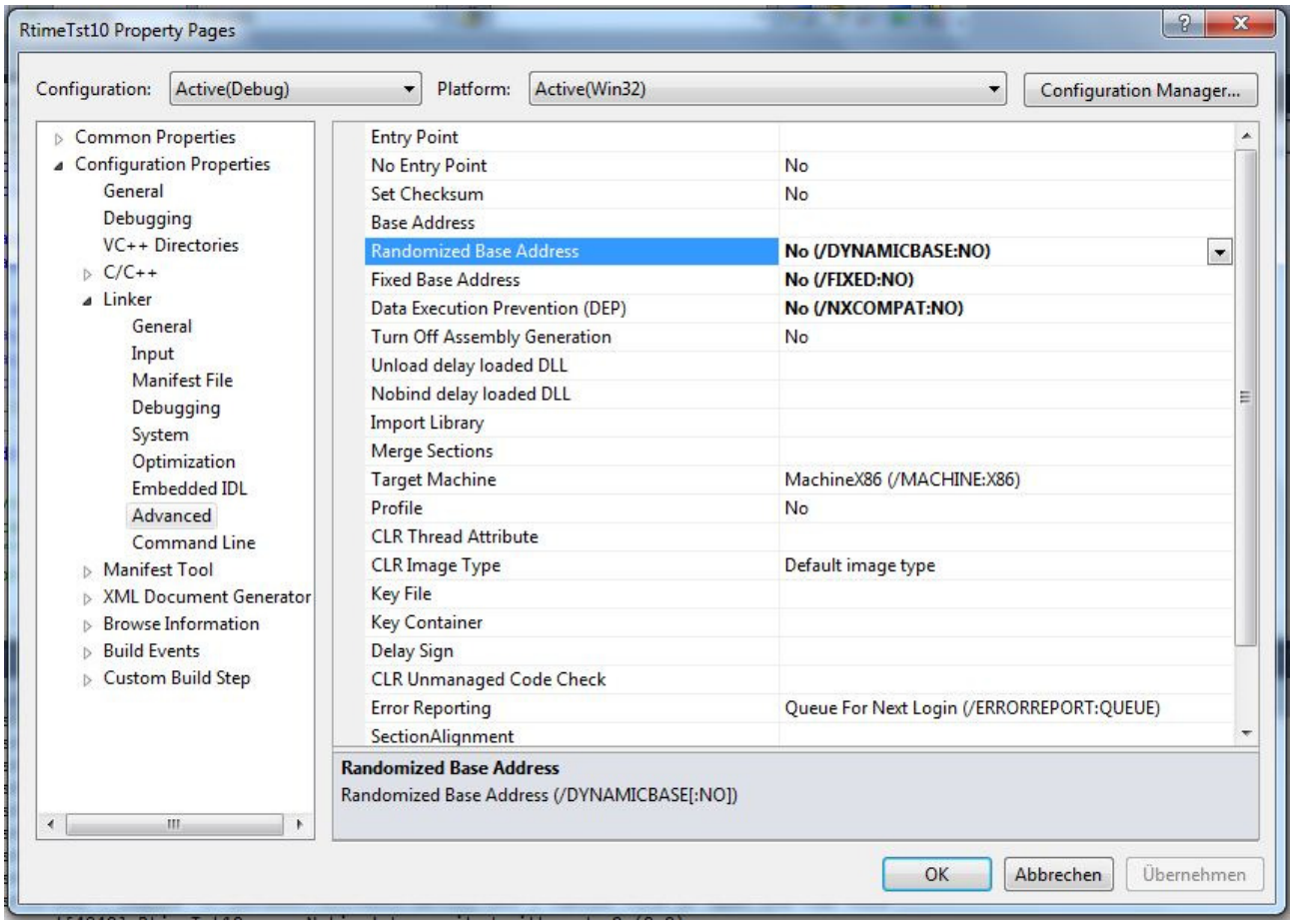
### 4.1.1 Visual Studio 2010 Compiler Settings

With Visual Studio 2010 a change in the COMPILER settings was introduced. To make the Virtual Code Mapping (VCM) working correctly, the settings must be changed:

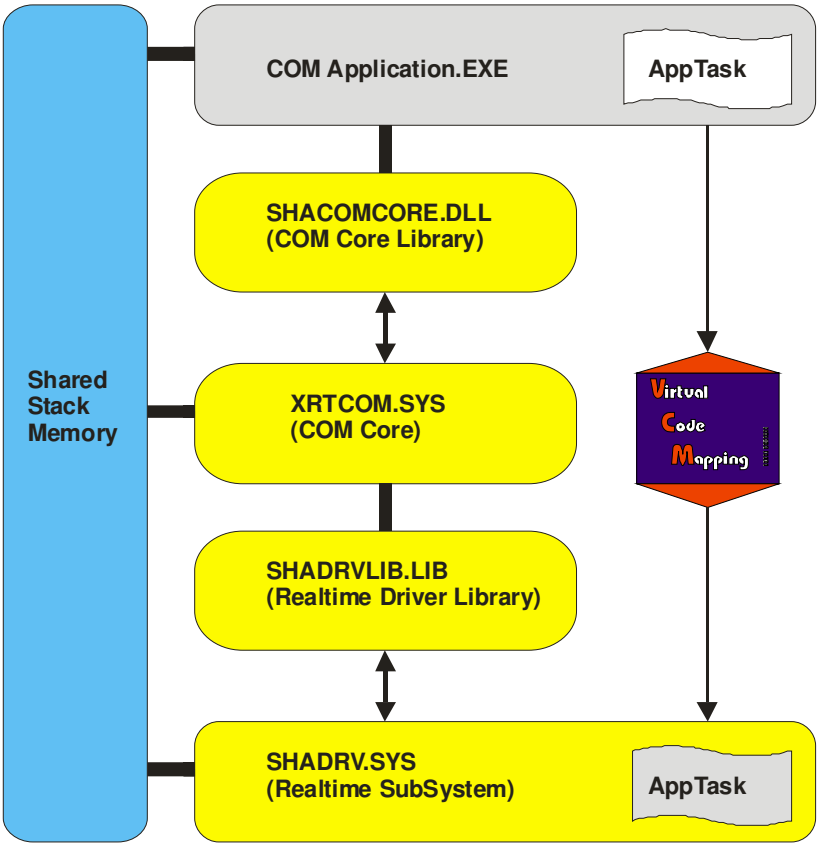


### 4.1.2 Visual Studio 2010 Linker Settings

With Visual Studio 2010 a change in the LINKER settings was introduced. To make the Virtual Code Mapping (VCM) working correctly, the settings must be changed:



## 4.2 Realtime Driver Stack



In the following all function prototypes will be discussed by samples. Since all platforms have their own syntax and dependencies, therefore the topics for the different platforms are marked as follow:

VC : Visual C++, eMbedded VC, Borland C++ Builder and LabWindows CVI

DP : Borland Delphi

### 4.2.1 VisualC++ and LabWindows CVI

Project files for VisualC++ and LabWindows:

| | |
|---|---|
| LIB\SHACOMCORE.LIB | Project Import Library |
| LIB\SHACOMCORE.DLL | Dynamic Link Library (copied to \WINNT\SYSTEM32) |
| INC\SHACOMCORE.H | Exported function prototypes |
| INC\COMMACROS.H | COM core macros |
| INC\COMCOREDEF.H | COM core definitions |
| INC\UARTDEF.H | UART definitions |

### 4.2.2 Borland C++Builder

Project files for Borland C++ Builder:

| | |
|---|---|
| LIB\SHACOMCOREOML.LIB | Project Import Library |
| LIB\SHACOMCOREOML.DLL | Dynamic Link Library (copied to \WINNT\SYSTEM32) |
| INC\SHACOMCORE.H | Exported function prototypes |
| INC\COMMACROS.H | COM core macros |
| INC\COMCOREDEF.H | COM core definitions |
| INC\UARTDEF.H | UART definitions |

### 4.2.3 Borland Delphi

Project files for Borland Delphi:

| | |
|---|---|
| LIB\SHACOMCOREOML.LIB | Project Import Library |
| LIB\SHACOMCOREOML.DLL | Dynamic Link Library (copied to \WINNT\SYSTEM32) |
| INC\SHACOMCORE.PAS | Exported function prototypes |
| INC\COMMACROS.PAS | Exported COM functions |
| INC\COMCOREDEF.PAS | COM core definitions |

## 4.3 Header File COMCOREDEF.H

This header file defines all structures required for handling the core interface and realtime stack data. The structure elements are combined hierachically.

### 4.3.1 Structure COM_PARAMS

This structure is required by all core interface functions, and contains all required and optional input and output data members.

```
typedef struct _COM_PARAMS
{
      //Input parameters
      PCHAR             port_name;        //Pointer to Port name (e.g. COM1)
      ULONG             period;           //Realtime scheduling period
      ULONG             sched_cnt;        //Application scheduling count
      ULONG             timeout;          //Timeout constant in msec
      ULONG             baud_rate;        //Baudrate
      UCHAR             data_len;         //Data length
      UCHAR             stop_bits;        //Stop bits
      UCHAR             parity;           //Parity
      BOOLEAN           flow_ctrl;        //Flow control
      BOOLEAN           dtr_ctrl;         //DTR control
      BOOLEAN           rts_ctrl;         //RTS control
      UCHAR             xon_char;         //XON char
      UCHAR             xoff_char;        //XOFF char
      ULONG             rx_thres          //RX Realtime Cycle Threshold

      //Output parameters
      USHORT            port_pa;          //Base Port Address
      ULONG             port_id;          //Port identifier
      ULONG             memory_tag;       //Memory TAG
      ULONG             remain_time;      //Remaining realtime
      ULONG             core_dll_ver;     //Core DLL version
      ULONG             core_drv_ver;     //Core driver version
      ULONG             sha_drv_ver;      //SHA driver version
      ULONG             sha_lib_ver;      //SHA library version
      COM_ERR_CNTS      err_cnts;         //COM error counters

      //Input - Output parameters
      PUCHAR            pBuffer;          //Pointer to data buffer
      ULONG             len;              //Read/Write length

      //Realtime level2 input parameters
      FP_RING0          fpAppTask;        //Function pointer to realtime task

      //Realtime level2 output parameters
      PCOM_STACK        pSystemStack;     //COM_STACK structure for realtime task
      PCOM_STACK        pUserStack;       //COM_STACK structure for windows task

} COM_PARAMS, *PCOM_PARAMS;
```

### 4.3.2 Structure COM_STACK

This structure is required when accessing the ethernet data of the Realtime Core directly (Realtime Level2). The structure elements are combined hierachically:

```
typedef struct _COM_STACK
{
      COM_STACK_HDR     hdr;
      COM_TABLE         tx_table;
      COM_TABLE         rx_table;

} COM_STACK, *PCOM_STACK;


      typedef struct _COM_STACK_HDR
      {
            BOOLEAN           run_flag;
            BOOLEAN           err_flag;
            BOOLEAN           lev2_flag;

      } COM_STACK_HDR, *PCOM_STACK_HDR;


      typedef struct _COM_TABLE
      {
            ULONG       findex;           //Forward index
            ULONG       bindex;           //Back Index
            COM_ENTRY   list[MAX_COM_ENTRIES];

      } COM_TABLE, *PCOM_TABLE;


            typedef struct _COM_ENTRY
            {
                  COM_PACKET  packet;           //COM packet
                  BOOLEAN     bOccupied;        //table flag
                  __int64     TscCnt;           //Time Scale Counter

            } COM_ENTRY, *PCOM_ENTRY;


                  typedef struct _COM_PACKET
                  {
                       UCHAR       buffer[MAX_COM_PACKET_SIZE];
                       ULONG       len;

                  } COM_PACKET, *PCOM_PACKET;
```

### 4.4 Header File COMMACROS.H

This header file defines all macros required for handling realtime level 2.

//Macro to write COM port byte

```
#define COM_SYSTEM_WRITE_BYTE(__Addr, __Val)    \
{                                               \
      UCHAR Val = (UCHAR)(__Val);               \
      USHORT Addr = (USHORT)(__Addr);           \
      __asm mov dx, Addr                        \
      __asm mov al, Val                         \
      __asm out dx, al                          \
}                                               \
```

//Macro to read COM port byte

```
#define COM_SYSTEM_READ_BYTE(__Addr, __pVal)    \
{                                               \
      PUCHAR pVal = (PUCHAR)(__pVal);           \
      USHORT Addr = (USHORT)(__Addr);           \
      __asm mov dx, Addr                        \
      __asm in al, dx                           \
      __asm mov ecx,dword ptr [pVal]            \
      __asm mov byte ptr [ecx], al              \
}                                               \
```

//Macro to set COM port bits

```
#define COM_SYSTEM_SET_BITS(__Addr, __Val)      \
{                                               \
      UCHAR Val = (UCHAR)(__Val);               \
      USHORT Addr = (USHORT)(__Addr);           \
      __asm mov dx, Addr                        \
      __asm in al, dx                           \
      __asm or al, Val                          \
      __asm out dx, al                          \
}                                               \
```

//Macro to clear COM port bits

```
#define COM_SYSTEM_CLR_BITS(__Addr, __Val)      \
{                                               \
      UCHAR NegVal = (UCHAR)(~__Val);           \
      USHORT Addr = (USHORT)(__Addr);           \
      __asm mov dx, Addr                        \
      __asm in al, dx                           \
      __asm and al, NegVal                      \
      __asm out dx, al                          \
}                                               \
```

//Macro to start or stop realtime tasks

```
#define COM_TASK_CONTROL(__pStack, __bRun)                          \
{                                                                   \
      PCOM_STACK        _pStack = (PCOM_STACK)__pStack;             \
      PCOM_STACK_HDR    _pHdr = (PCOM_STACK_HDR)&_pStack->hdr;      \
      _pHdr->err_flag = FALSE;                                      \
      _pHdr->run_flag = __bRun;                                     \
}                                                                   \
```

//Macro to enable or disable realtime level2 and save condition

```
#define COM_LEVEL2_CONTROL(__pStack, __bCond)                       \
{                                                                   \
      PCOM_STACK        _pStack = (PCOM_STACK)__pStack;             \
      PCOM_STACK_HDR    _pHdr = (PCOM_STACK_HDR)&_pStack->hdr;      \
      _pHdr->lev2_flag = __bCond;                                   \
}                                                                   \
```

//Macro to control TX and RX stack index

```
#define COM_STACK_CONTROL(__pStack, __TxIndex, __RxIndex)                   \
{                                                                           \
      PCOM_STACK        _pStack = (PCOM_STACK)__pStack;                     \
      PCOM_STACK_HDR    _pHdr = (PCOM_STACK_HDR)&_pStack->hdr;              \
      PCOM_TABLE        _pTxTable = (PCOM_TABLE)&_pStack->tx_table;         \
      PCOM_TABLE        _pRxTable = (PCOM_TABLE)&_pStack->rx_table;         \
      PCOM_ENTRY        _pTxEntry = (PCOM_ENTRY)&_pTxTable->list[__TxIndex]; \
      PCOM_ENTRY        _pRxEntry = (PCOM_ENTRY)&_pRxTable->list[__RxIndex]; \
      BOOLEAN           _bRun = _pHdr->run_flag;                            \
      _pHdr->run_flag = FALSE;                                              \
      _pTxEntry->bOccupied = FALSE;                                         \
      _pRxEntry->bOccupied = FALSE;                                         \
      _pTxTable->bindex = __TxIndex;                                        \
      _pRxTable->bindex = __RxIndex;                                        \
      _pTxTable->findex = __TxIndex;                                        \
      _pRxTable->findex = __RxIndex;                                        \
      _pHdr->run_flag = _bRun;                                              \
}                                                                           \
```

//Macro to check for idle stack

```
#define COM_CHECK_STACK_IDLE(__pStack, __pbIdle)                    \
{                                                                   \
      PCOM_STACK  _pStack = (PCOM_STACK)__pStack;                   \
      PCOM_TABLE  _pTxTable = (PCOM_TABLE)&_pStack->tx_table;       \
      PCOM_TABLE  _pRxTable = (PCOM_TABLE)&_pStack->rx_table;       \
      BOOLEAN               _bRun = _pHdr->run_flag;                \
      *__pbIdle = FALSE;                                            \
      if ((_pTxTable->findex == _pTxTable->bindex) &&               \
            (_pRxTable->findex == _pRxTable->bindex))               \
            *__pbIdle = TRUE;                                       \
}                                                                   \
```

//Macro to get current TX entry pointer

```
#define COM_GET_TXENTRY_PTR(__pStack, __ppEntry)                    \
{                                                                   \
      PCOM_STACK  _pStack = (PCOM_STACK)__pStack;                   \
      PCOM_TABLE  _pTable = (PCOM_TABLE)&_pStack->tx_table;         \
      ULONG       _Index      = (ULONG)_pTable->bindex;             \
      PCOM_ENTRY  _pEntry = (PCOM_ENTRY)&_pTable->list[_Index];     \
      *__ppEntry = _pEntry;                                         \
}                                                                   \
```

//Macro to get current RX entry pointer

```
#define COM_GET_RXENTRY_PTR(__pStack, __ppEntry)                    \
{                                                                   \
      PCOM_STACK  _pStack = (PCOM_STACK)__pStack;                   \
      PCOM_TABLE  _pTable = (PCOM_TABLE)&_pStack->rx_table;         \
      ULONG       _Index      = (ULONG)_pTable->findex;             \
      PCOM_ENTRY  _pEntry = (PCOM_ENTRY)&_pTable->list[_Index];     \
      *__ppEntry = _pEntry;                                         \
}                                                                   \
```

//Macro to check error flag

```
#define COM_CHECK_ERROR(__pStack, __pbError)                        \
{                                                                   \
      PCOM_STACK        _pStack = (PCOM_STACK)__pStack;             \
      PCOM_STACK_HDR    _pHdr = (PCOM_STACK_HDR)&_pStack->hdr;      \
      *__pbError = _pHdr->err_flag;                                 \
}                                                                   \
```

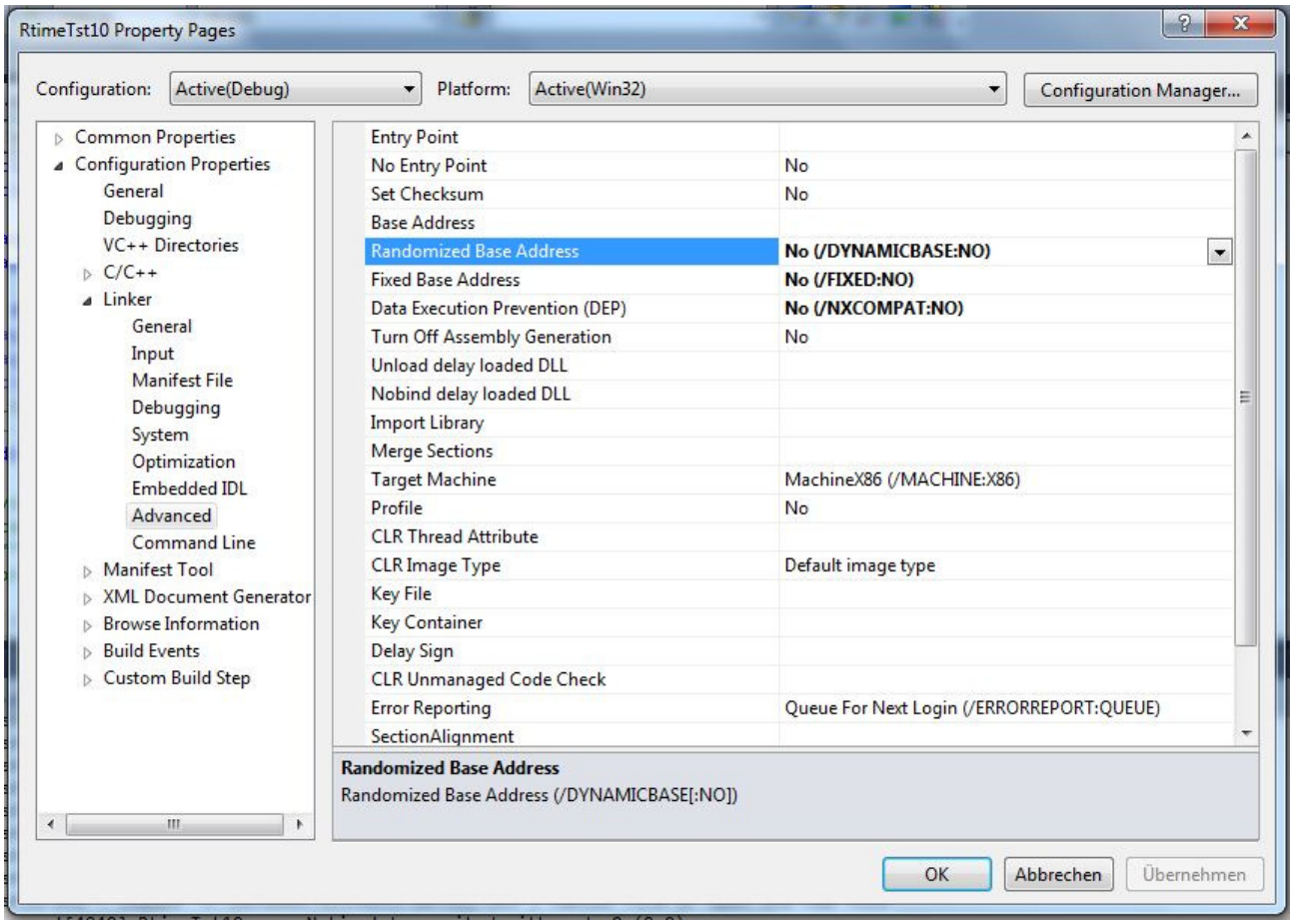
//Macro to set error condition

```
#define COM_SET_ERROR(__pStack)                                     \
{                                                                   \
      PCOM_STACK        _pStack = (PCOM_STACK)__pStack;             \
      PCOM_STACK_HDR    _pHdr = (PCOM_STACK_HDR)&_pStack->hdr;      \
      _pHdr->err_flag = TRUE;                                       \
      _pHdr->run_flag = FALSE;                                      \
}                                                                   \
```

## 4.5 COM Core Interface

The header file SHACOMCORE.H defines all required prototypes and parameters of the COM Core Library. In the following all function prototypes will be discussed by samples.

Note:
With Visual Studio 2010 a change in the LINKER settings was introduced. To make the Virtual Code Mapping (VCM) working correctly, the settings must be changed:



### 4.5.1 ShaComCreate

This function opens the communication port and installs the realtime ringbuffer system. On success the returning value is ERROR_SUCCESS, otherwise the returning value corresponds to that from GetLastError(). The usable interface types are defined in the header file "GLOBDEF.H".

```
VC    ULONG ShaCommCreate(PCOM_PARAMS pComParams);   //Initial parameters
```

Note:
COM1 through COM9 can be directly referenced just like a filename from programs and from the command line. However, COM10 and above must be referenced with the following syntax:

\\.\COM10 Stringformat: "\\\\.\\COM10"

Sample:

```
memset(&Params, 0, sizeof(COM_PARAMS));
Params.port_name = "COM8";                //Port name
Params.period = 100;                      //Realtime scheduling period
Params.sched_cnt = 1;                     //Realtime scheduling count
Params.baud_rate = CP_BR115200;           //COM baudrate
Params.data_len = CP_DATALEN8;            //Data length
Params.stop_bits = CP_ONESTOPBITS;        //Stop bits
Params.parity = CP_NOPARITY;              //Parity
Params.flow_ctrl = FALSE;                 //Flow control
Params.dtr_ ctrl = FALSE;                 //DTR control
Params.rts_ ctrl = FALSE;                 //RTS control
Params.xon_char = FALSE;                  //XON char
Params.xoff_char = FALSE;                 //XOFF char
Params.timeout = 3000;                    //Read timeout constant in msec
Params.rx_thres = 2;                      //RX Realtime Cycle Threshold
Params.fpAppTask = NULL;                  //No realtime level2 task

//Init COM port
if (ERROR_SUCCESS == ShaComCreate(&Params))
{ …
}
```

### 4.5.2 ShaComDestroy

This function closes the communication port and releases all resources for communication.

VC void ShaCommDestroy(PCOM_PARAMS pComParams);

### 4.5.3 ShaComReset

This function resets the communication port and releases all resources for communication.

VC `BOOLEAN ShaComReset(PCOM_PARAMS pComParams);`

### 4.5.4 ShaComCheckStatus

This function gets the status of the comminication port.

VC `void ShaComCheckStatus(PCOM_PARAMS pComParams);`

### 4.5.5 ShaComTransmitFrame

This function read data from the Realtime RingBuffer. The function requires to allocate a buffer for the sending data.

VC `void ShaComTransmitFrame(PCOM_PARAMS pComParams);`

### 4.5.6 ShaComReceiceFrame

This function read data from the Realtime RingBuffer. The function requires to allocate a buffer for the receive data.

VC `void ShaComReceiveFrame(PCOM_PARAMS pComParams);`

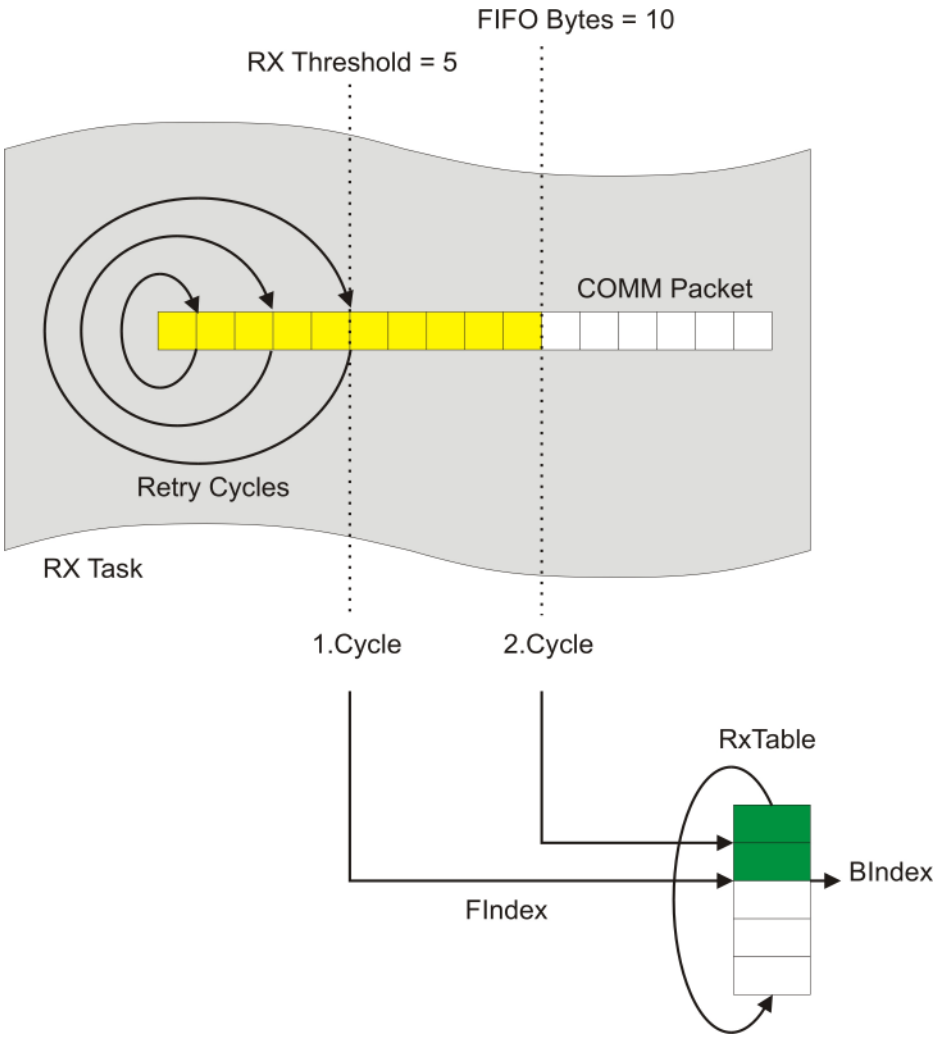
### 4.6 Packet Traffic

The COMM Realtime Core is designed to receive or transmit packets of data per realtime cycle. Therefore each table entry keeps space for upto MAX_COM_PACKET_SIZE of data bytes. Internally the COMM core uses a conditional retry unit to check the ports for receive or transmit conditions per realtime cycle. How many data bytes actually can be received or transmitted per one realtime cycle depends on following topics:

- Baudrate — As higher the Baudrate, as more bytes will be received per Realtime Cycle.
- Realtime Period — As lower the realtime period is set, as more bytes will be received or Transmitted per realtime cycle
- FIFO Size — Most of UART based Receiver/Transmitter chips keeps 16 or more Bytes as FIFO buffer (FCR register Bit0).
- RX Threshold — The RX Threshold will force the COMM core to wait as many realtime cycles as required, to get the amount of threshold bytes, until the COMM packet will delivered to the stack.
- Retry Unit — The internal retry unit checks changes of the UART FIFO in a conditional loop. This assures not to disturb the realtime cycle (in contrary to wait delays). The retry unit works within one realtime cycle.

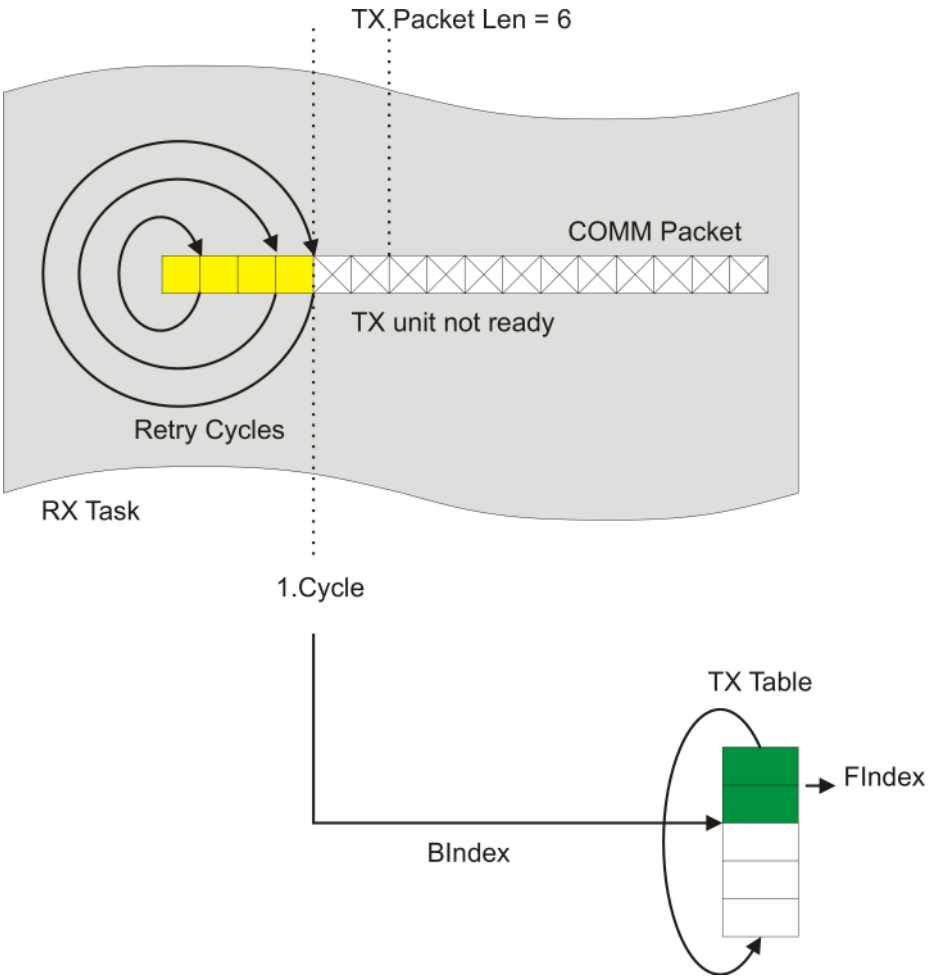
### 4.6.1 RX Traffic

How many bytes are received per cycle depends on the RX threshold value, the baudrate and the retry cycles. A RX threshold value = 1 results safest realtime condition, but less performance.



### 4.6.2 TX Traffic

Set TX output data. How many bytes may transmitted within one cycle depends on the baudrate and the UART chip. Usually a one byte transmit (one byte per cycle) results safest realtime condition, but less performance.



## 4.7 Sample program (Realtime Level 2):

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "c:\com\ShaComCore.h"
#include "c:\sha\shaexp.h"


PCOM_STACK        pUserStack = NULL;
PCOM_STACK        pSystemStack = NULL;
char              RxData[MAX_PATH] = "";
UCHAR             TxData[MAX_PATH] = "Dies ist ein Test";
ULONG             TxSize = strlen((char*)TxData);
ULONG             RxSize = strlen((char*)TxData);
ULONG             TxIndex = 0;
ULONG             RxIndex = 0;

//*****************************************************************
//*** !!! Check if compiler setting /GZ was removed !!!         ***
//*****************************************************************

void static AppTask(void)
{
      PCOM_ENTRY pTxEntry = NULL;
      PCOM_ENTRY pRxEntry = NULL;

//    __asm int 3

      //Get ethernet stack frame TX and RX entries of stack location 0
      COM_GET_TXENTRY_PTR(pSystemStack, &pTxEntry);
      COM_GET_RXENTRY_PTR(pSystemStack, &pRxEntry);

      //*********************************************
      //*** Do the transmit stuff
      //*********************************************

      //Check if TX entry is emptry
      if ((pTxEntry->bOccupied == FALSE)
      {
            //Set TX output data. How many bytes may transmitted
            //within one cycle depends on the baudrate and the UART chip.
            //Usually a one byte transmit (one byte per cycle) results safest
            //realtime condition, but less performance.
            if (TxIndex < TxSize)
            {
                  //Set TX data
                  MEM_CPY_SAFE(pTxEntry->packet.buffer, &TxData[TxIndex], 1);
                  pTxEntry->packet.len = 1;

                  //Increase TX index
                  TxIndex++;
            }

            //Occupy TX entry
            pTxEntry->bOccupied = TRUE;
```

```
	}

	//*********************************************
	//*** Do the receive stuff
	//*********************************************

	//Check if TX entry is emptry
	if (pRxEntry->bOccupied == TRUE)
	{
		//Check packet length. How many bytes are received per cycle
		//depends on the RX threshold value and the baudrate. A RX
		//threshold value = 1 results safest realtime condition, but
		//less performance.
		if (RxIndex + pRxEntry->packet.len < RxSize)
		{
			//Get RX data
			MEM_CPY_SAFE(
				&RxData[RxIndex],
				pRxEntry->packet.buffer,
				pRxEntry->packet.len);

			//Incease RX index
			RxIndex += pRxEntry->packet.len;
		}

		//Free RX entry
		pRxEntry->bOccupied = FALSE;
	}
}



void main(void)
{
	COM_PARAMS Params;
	char szOut[MAX_PATH] = "Dies ist ein Test";

	printf("\n*** COM Core Test ***\n\n");

	//*******************************
	//*** Required COM parameters ***
	//*******************************
	memset(&Params, 0, sizeof(COM_PARAMS));
	Params.port_name = "COM8";				//Port name
	Params.period = 10;					//Realtime scheduling period
	Params.sched_cnt = 1;					//Realtime scheduling count
	Params.baud_rate = CP_BR115200;			//COM baudrate
	Params.data_len = CP_DATALEN8;			//Data length
	Params.stop_bits = CP_ONESTOPBITS;		//Stop bits
	Params.parity = CP_NOPARITY;			//Parity
	Params.flow_ctrl = FALSE;				//Flow control
	Params.dtr_ ctrl = FALSE;				//DTR control
	Params.rts_ ctrl = FALSE;				//RTS control
	Params.xon_char = FALSE;				//XON char
	Params.xoff_char = FALSE;				//XOFF char
	Params.timeout = 3000;					//Read timeout constant in msec
	Params.rx_thres = 2;					//RX Realtime Cycle Threshold
	Params.fpAppTask = AppTask;				//No realtime level2 task
```

```
//Init COM port
if (ERROR_SUCCESS == ShaComCreate(&Params))
{
      //Init stack pointers
      pSystemStack = Params.pSystemStack;
      pUserStack   = Params.pUserStack;

      //Set level2 condition
      //Set TX and RX stack location to 0
      //Enable task scheduler
      COM_LEVEL2_CONTROL(pUserStack, TRUE);
      COM_STACK_CONTROL(pUserStack, 0, 0);
      COM_TASK_CONTROL(pUserStack, TRUE);

      //Print SHA driver, DLL and CORE versions
      ShaComGetVersion(&Params);
      printf("CORE-DLL Version: %.2f\nCORE-DRV Version: %.2f\n
              SHA-LIB Version: %.2f\nSHA-DRV Version: %.2f\n",
            Params.core_dll_ver / (double)100,
            Params.core_drv_ver / (double)100,
            Params.sha_lib_ver / (double)100,
            Params.sha_drv_ver / (double)100);

      while (!kbhit())
      {
            //Check if error condition occured
            if (pUserStack->hdr.run_flag == FALSE)
            {
                  //Get last status
                  ShaComCheckStatus(&Params);
                  printf("DERR:   %i\n", Params.err_cnts.derr);
                  printf("FRERR:  %i\n", Params.err_cnts.frerr);
                  printf("PARERR: %i\n", Params.err_cnts.parerr);
                  printf("OVERR:  %i\n", Params.err_cnts.overr);
                  break;
            }

            //Reset TX
            if (TxIndex == TxSize)
            {
                  printf("TX: %s\n", TxData);
                  TxIndex = 0;
            }

            //Reset TX
            if (RxIndex == RxSize)
            {
                  printf("RX: %s\n", RxData);
                  strcpy(RxData, "");
                  RxIndex = 0;
            };

            //Do some delay
            Sleep(100);
      }

      //Disable task scheduler
      COM_TASK_CONTROL(pUserStack, FALSE);
```

```
            //Cleanup COM port
            ShaComDestroy(&Params);
        }
}
```

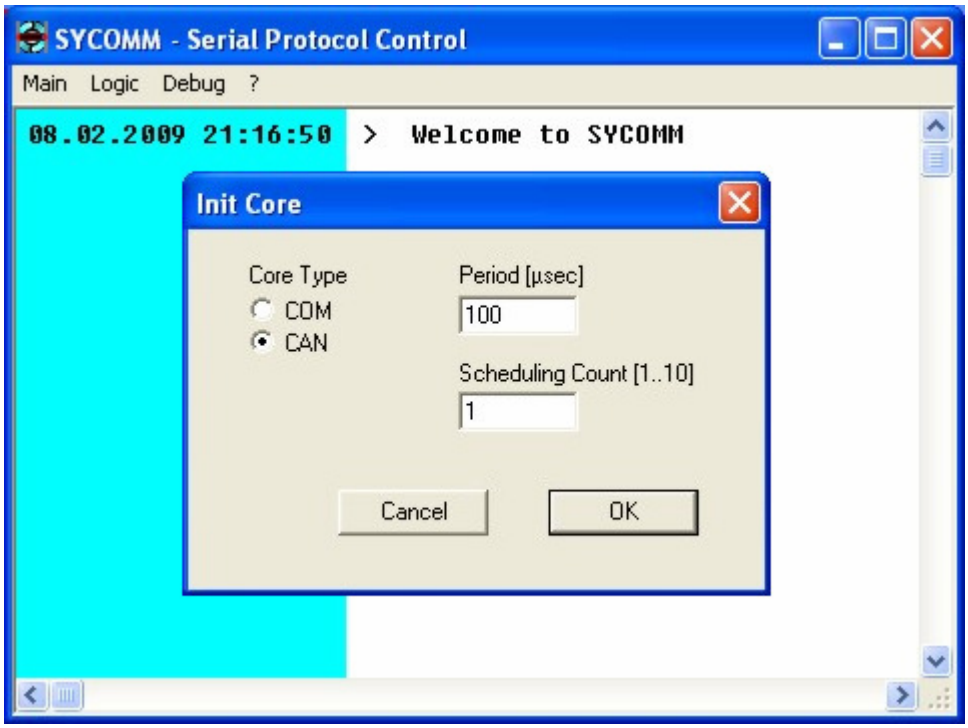
## 4.8 Sample program (Standard COM Interface):

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>

HANDLE hComm = NULL;

BOOLEAN CommSet(void)
{
      DCB   Dcb;
      COMMTIMEOUTS Cto;

      //Init DBC structure
      Dcb.fBinary = TRUE;

      //Set COM parameters
      Dcb.BaudRate      = CBR_115200;
      Dcb.StopBits      = ONESTOPBIT;
      Dcb.ByteSize      = 8;
      Dcb.Parity  = NOPARITY;
      Dcb.fParity = (Dcb.Parity == NOPARITY) ? FALSE : TRUE;

      //Set hardware flow control
      Dcb.fDtrControl   = DTR_CONTROL_HANDSHAKE; //DTR_CONTROL_ENABLE;
      Dcb.fRtsControl   = RTS_CONTROL_HANDSHAKE; //RTS_CONTROL_ENABLE;
      Dcb.fOutxDsrFlow  = FALSE;
      Dcb.fOutxCtsFlow  = FALSE;

      //Set software flow control
      Dcb.fInX    = FALSE;
      Dcb.fOutX   = FALSE;

      Dcb.XonChar  = 0x11;
      Dcb.XoffChar = 0x17;

      //Set DCB settings
      if (!(SetCommState(hComm, &Dcb)))
            return FALSE;

      //Set timeout behaviour for Overlapping
      Cto.ReadIntervalTimeout = -1;
      Cto.ReadTotalTimeoutMultiplier = 0;
      Cto.ReadTotalTimeoutConstant = 1000;
      Cto.WriteTotalTimeoutMultiplier = 0;
      Cto.WriteTotalTimeoutConstant = 0;

      //Set TIMEOUT settings
      if (!(SetCommTimeouts(hComm, &Cto)))
            return FALSE;

      //Everything is OK
      return TRUE;
}

#define OUT_STR   "Dies ist ein Test"
```

```
void main(void)
{
	char OutBuffer[MAX_PATH];
	char InBuffer[MAX_PATH];
	ULONG BytesWritten;
	ULONG BytesRead;

	printf("*** COM Core Test for file interface ***\n\n");

	//open COM file
	//(Note: Names of COM ports > 10 need to be prefixed by "\\\\.\\")
	hComm = CreateFile(
			"\\\\.\\COM10",
			GENERIC_WRITE | GENERIC_READ,
			0, NULL,
			OPEN_EXISTING,
			0, NULL);

	if (hComm != INVALID_HANDLE_VALUE)
	{
		//Set COM parameters
		if (CommSet())
		{
			//Purge all communication buffers
			if (PurgeComm(hComm, PURGE_TXCLEAR | PURGE_RXCLEAR))
			{
				printf("Press any key to exit ...\n");
				while (!kbhit())
				{
					//Set buffers
					strcpy(OutBuffer, OUT_STR);
					memset(InBuffer, 0, MAX_PATH);
					int OutSize = strlen(OUT_STR);
					int InSize = strlen(OUT_STR);

					//Write out data packet
					if (!(WriteFile(hComm, OutBuffer, OutSize,
							&BytesWritten, NULL)))
						break;

					//Read in data packet
					if (!(ReadFile(hComm, InBuffer, InSize,
							&BytesRead, NULL)))
						break;
				}
			}
		}
		//Close COM file
		CloseHandle(hComm);
	}
}
```

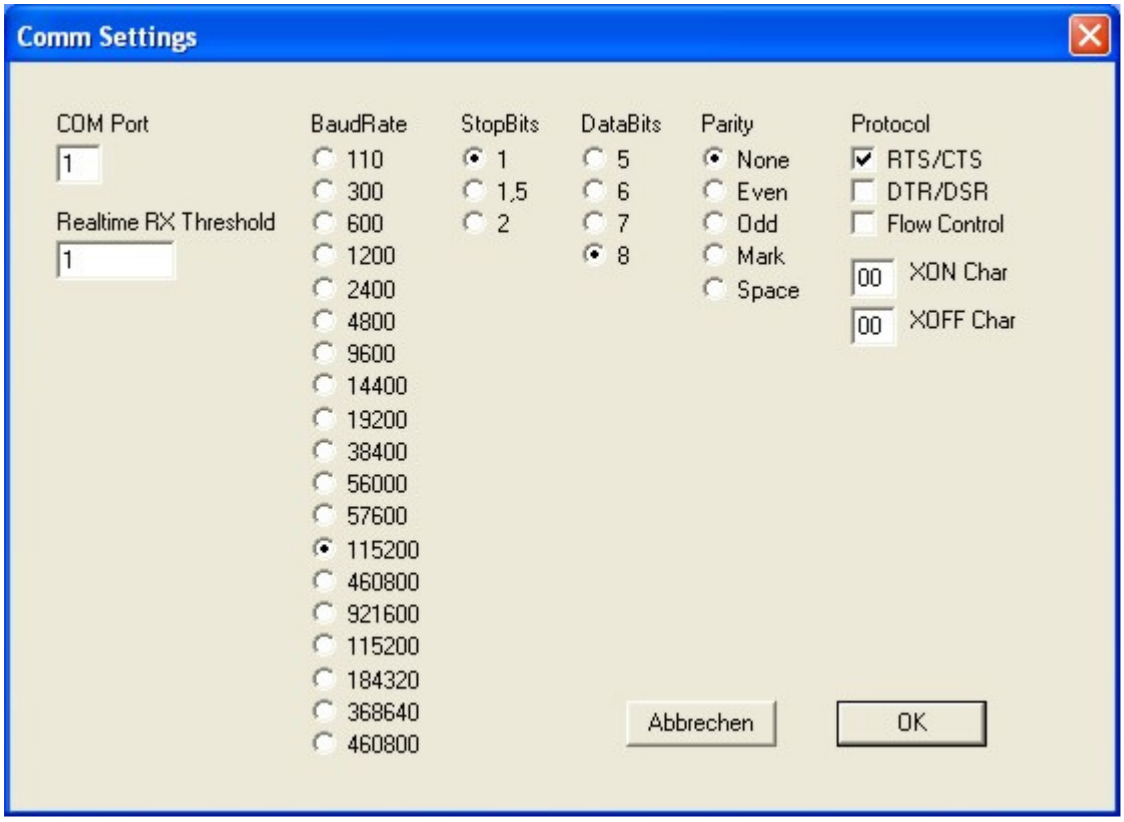
# 5 SYCOMM Protocol Control

The SYCOMM Software is used for contolling and analyzing serial protocols under real-time conditions. Thereby all protocols can be managed by an easy-to-use textbased SCRIPT language which is parsed at runtime. Additionally a singlestep mode allows comfortable protocol debugging.



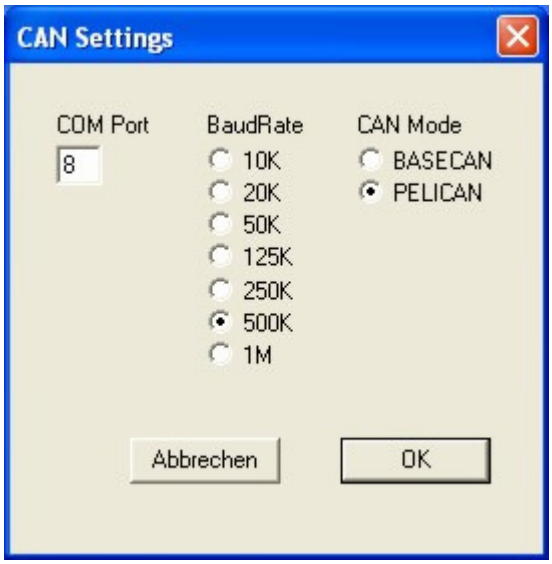
## 5.1 COMM Settings

When running SYCOMM a dialogbox for the serial communication settings appears. Withit all required settings for serial devices are handled. All parameters will be stored within the file COMM.PAR.



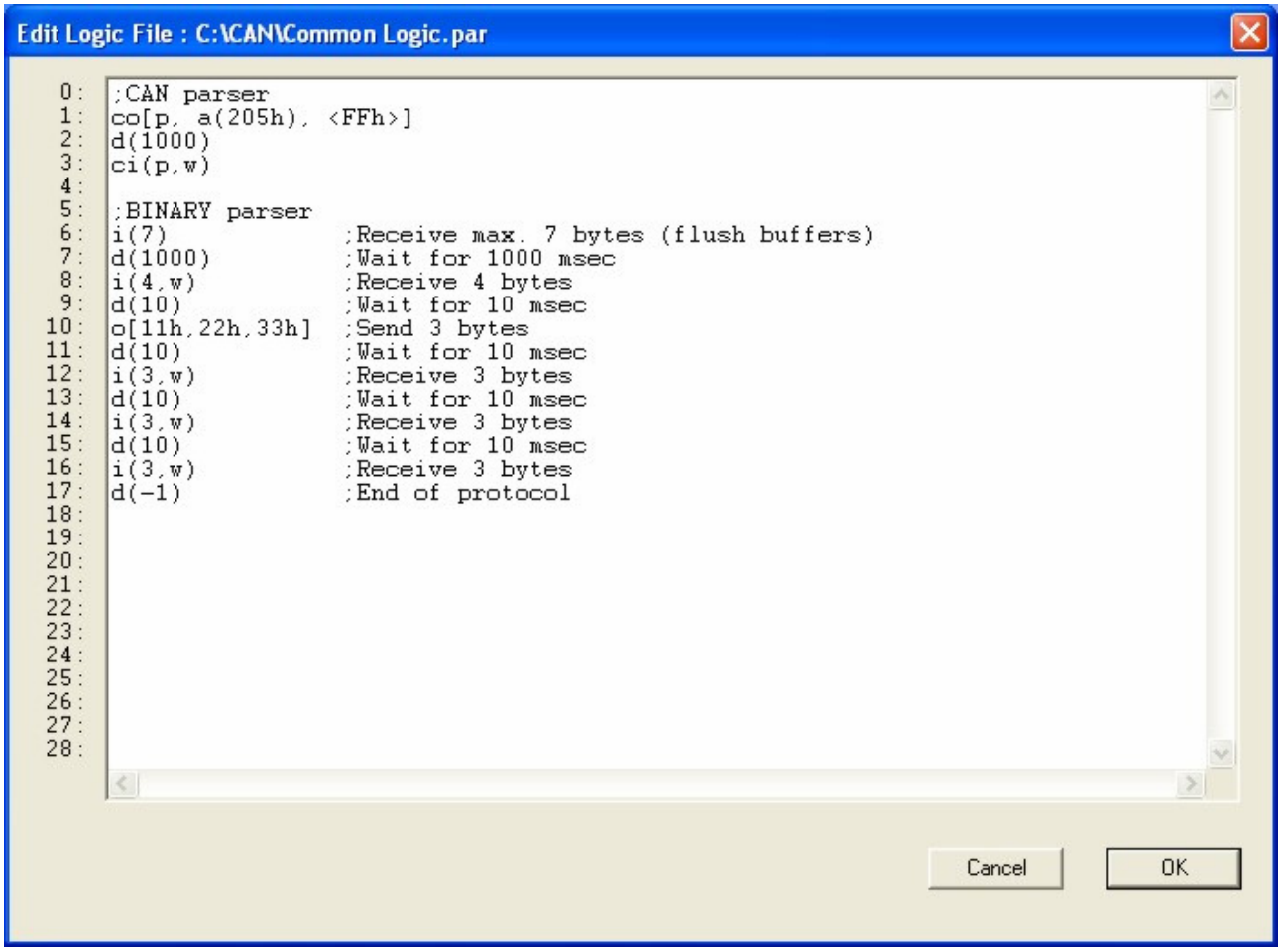
## 5.2 CAN Settings

When running SYCOMM a dialogbox for the serial communication settings appears. Withit all required settings for serial devices are handled. All parameters will be stored within the file CAN.PAR.



## 5.3 Protokoll SCRIPT

The control of the serial protocol is managed by the a simple SCRIPT language. The SCRIPT language allows control serial HOST and SLAVE protocols.

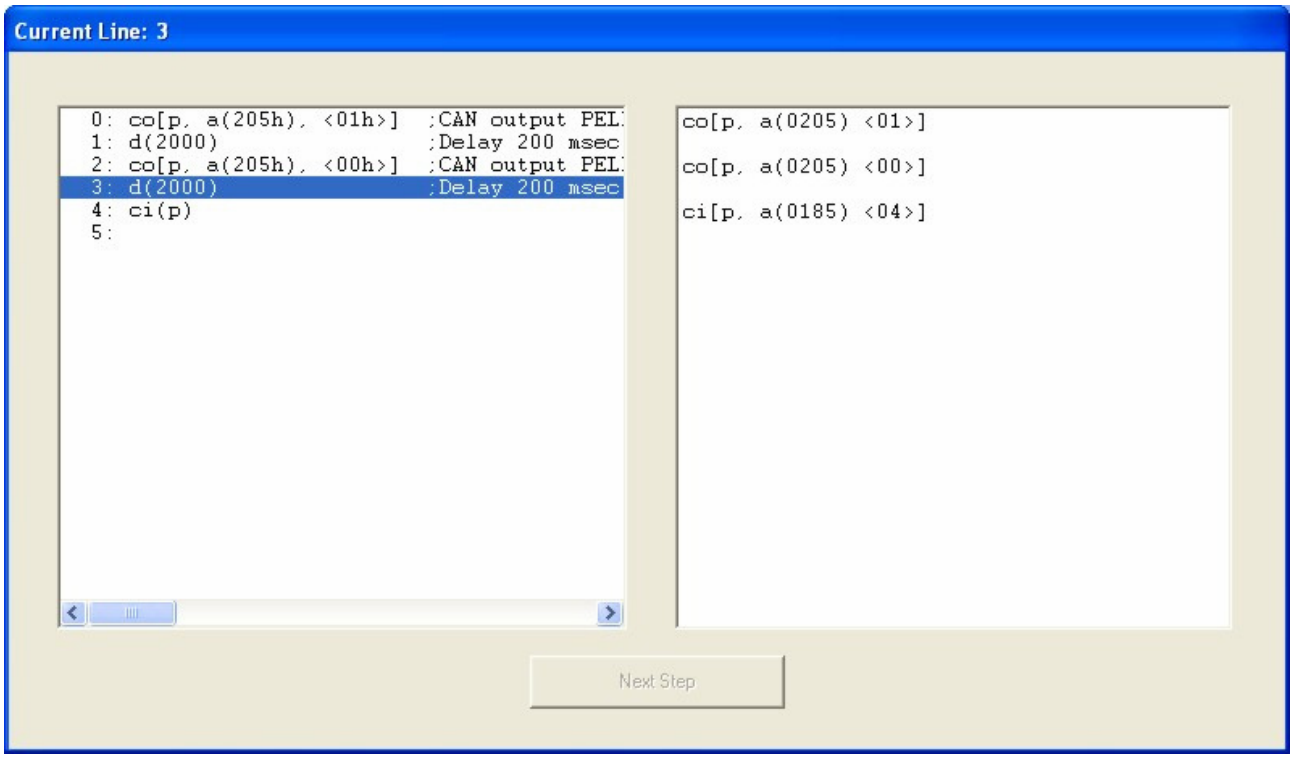


Comments are separated by [;] of the commands, but at each line only one command is allowed. Following script commands are valid:

| | |
|---|---|
| **i(x)** | : Receiving max.x bytes (dec) |
| **i(xh)** | : Receiving max. x bytes (hex) |
| **i(x,w)** | : Receiving x bytes and wait until completion |
| **i(x,o)** | : Receiving x bytes (only once) |
| **o[x,y,z,…]** | : Sending output bytes |
| **o[x,y,z,…, o]** | : Sending output bytes (only once) |
| **d(x)** | : Delay of x Cycle Counts (e.g. Period = 100µsec -> x * 1000 msec) |
| **d(x,o)** | : Delay of x Cycle Counts (only once) |
| **d(-1)** | : Stop the protocol |
| **co[p, a(xh), <x1, x2, x3, …, x8>]** | : Output PELICAN mode<br>Address x (hex)<br>SDO data (upto 8 Bytes) |
| **co[b, a(x), <x1, x2, x3, …, x8>]** | : Output BASECAN mode<br>Address x (dec)<br>SDO data (upto 8 Bytes) |
| **co[b, a(x), <x1, x2, x3, …, x8>, o]** | : Output BASECAN mode<br>Address x (dec)<br>SDO data (upto 8 Bytes)<br>Only once |
| **ci(p)** | : Input PELICAN mode |
| **ci(p,w)** | : Input PELICAN mode and wait until receiption |
| **ci(p,w,o)** | : Input PELICAN mode and wait until receiption (only once) |
| **ci(b,w)** | : Input BASECAN mode |

## 5.4 Step Mode Monitor

The step mode monitor (left window) displays the current line to be executed. To keep track when debugging asynchronous protocols it may be required to step through the protocol. Therefor the SingleStep mode can be activated in the menu [Protocol]. Each step will proceed by any key press. In SingleStep mode a line may be selected to be executed. This will is done by simple clicking on the corresponding line.




The Protocol Data Monitor (right window) allows to check the CAN protocol data sequence. Each line displays its current content.