TM

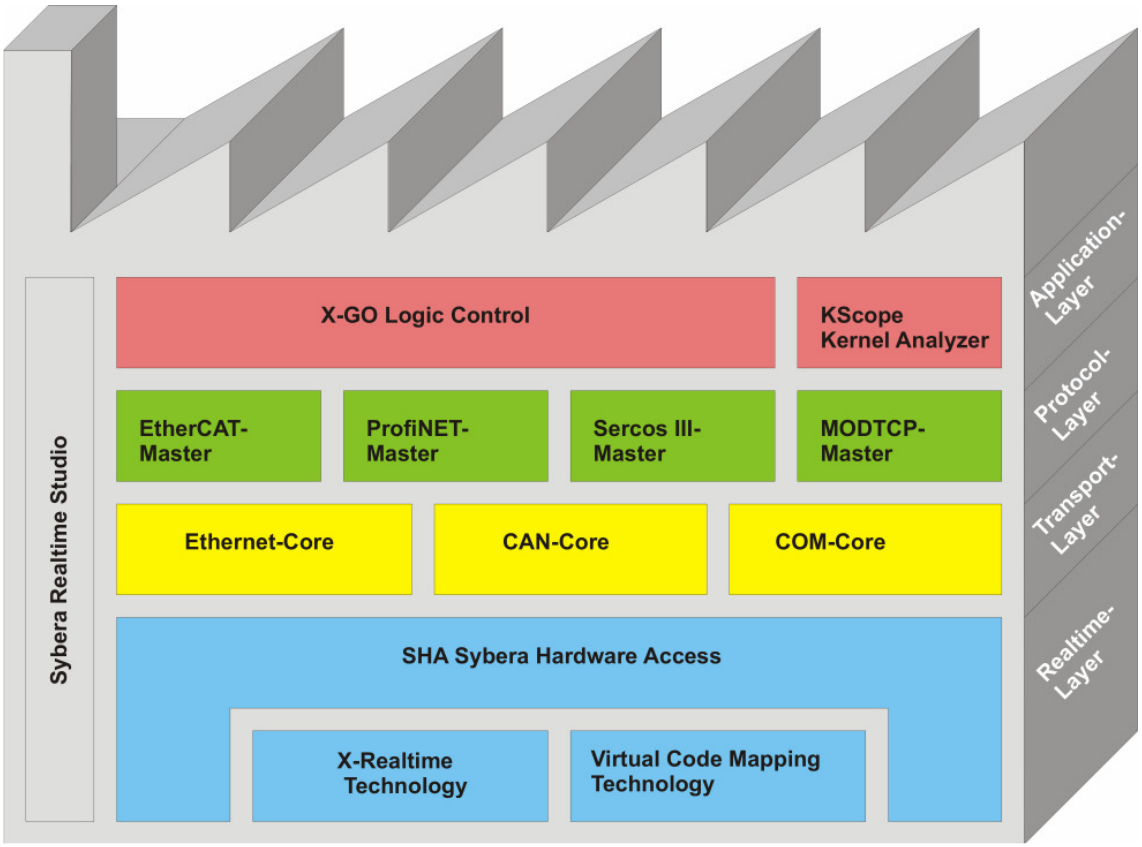
MODBus-TCP Realtime Master Library Documentation

Date: Sept, 17.2012

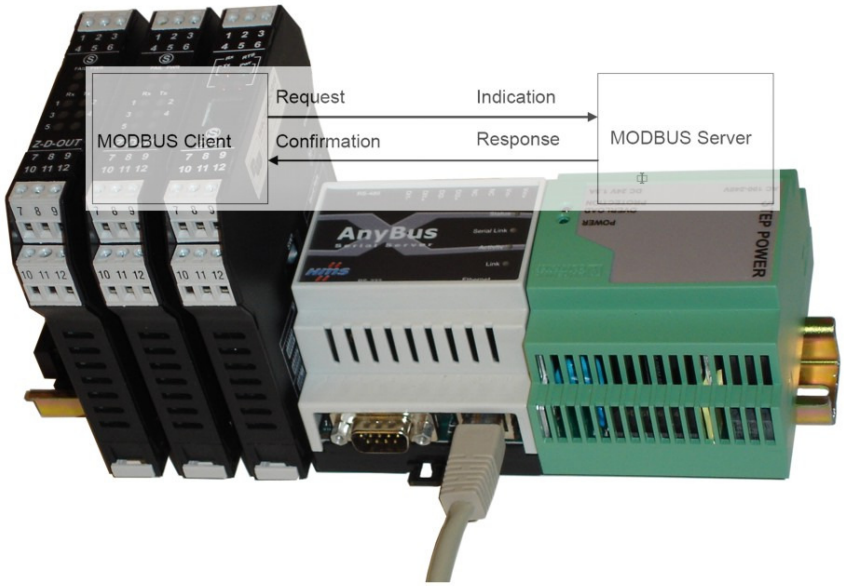
MODBus-TCP Realtime Master Library Documentation

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 RealtimeCores 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.



Based on the increasing requirements of networking within the industrial area, the ethernet communication plays a more and more important role for the control of devices. Different ethernet standards are available for the bus related communication, like SERCOS, Powerlink, EtherCAT, PROFINET and MODBus TCP. The communication principle rested characteristically on a deterministic process data-exchange and requires therefore for control and devices deterministic Soft- and hardware.
The MODBus TCP Realtime Master allows the deterministic control of MODBus telegrams over a closed TCP/IP communication network. This has been realized by a deterministic controlled Master/Slave handshaking. All TCP messages are controlled by a special developed TCP realtime state machine. Thereby the TCP connect, as well as the realtime controlled TCP disconnect is realized in realtime. The realtime master allows a star based network connection of MODBus participants, whereby the MAC- and IP-Adress of the buscoupplers are used for separation.



The direct control of the field bus devices with a PC and the operating system windows became relevant with the introduction of so-called realtime extensions. An example therefor is the realtime extension of SYBERA that enables ethernet update cycles upto 50 µsec. The Ethernet based bus-communication differs not only through a different protocol-specification, but rather also through the bus-topology. While EtherCAT realizes a ring-topology, MODBus TCP realizes a star topology. Similar to EtherCAT, MODBus TCP enables the typical Master/Slave-principle between control and devices. The MODBus TCP realtime communication prefaces a defined Master/Slave-relation.With the new realtime MODBus TCP master the necessity of a separate controller-hardware falls. The MODBus TCP control can be realized simply by the PC by standard Ethernet adapters

1.1 Supported Platforms

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

1.2 Supported OS

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

2 MODBus TCP Library Installation

For installation following steps are required:

Preparation

1. Provide a PC with INTEL or REALTEK Ethernet adapter and Windows operating system with administrator rights
2. Check the installed Ethernet adapter has given a correct IP address

Installation

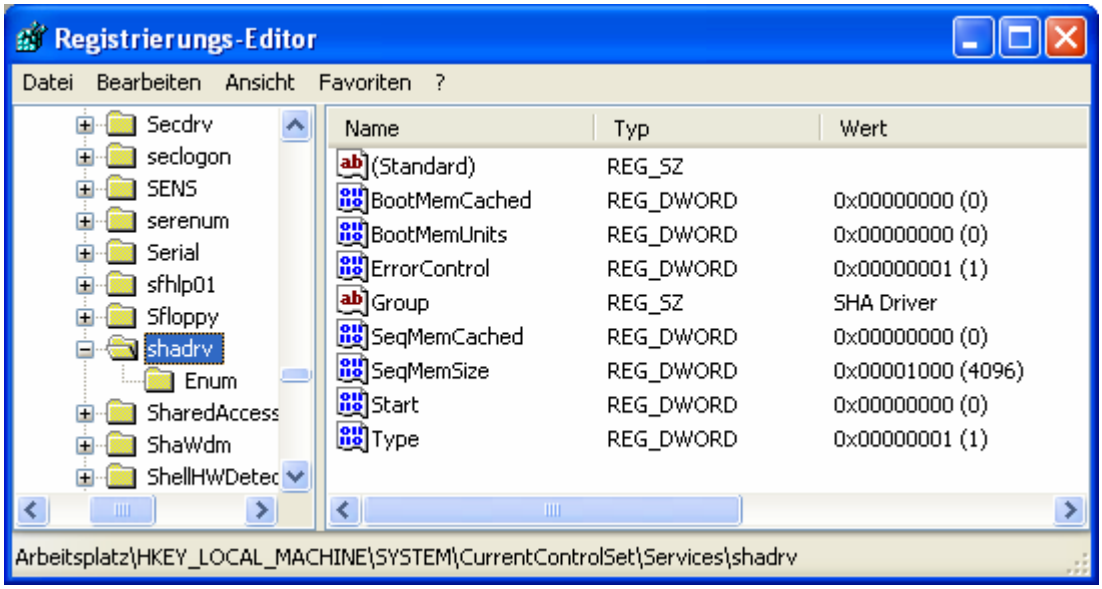
3. Install SHA realtime system (separate software package)
4. Install ETH transport library (separate software package)
5. Install the Ethernet X-Realtime NDIS Driver(s) (see manual)
6. Run the program SYSETUP of the MODBus Tcp library

   On Installation the PEC information (PID, SERNUM and KEYCODE) must be entered. The KEYCODE for the evaluation version is: 00001111-22223333
7. Optional: Check license with SYLICENCECHECK.EXE

Operation

8. Build the program with the library interface
9. Run the program

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. The registry key shadrv must be permanent:



Note: In order to operate SYBERA software under VISTA, this must be carried out as an ADMINISTRATOR. It's not enough to be logged on as administrator, since VISTA does not share the rights with the processes automatically.

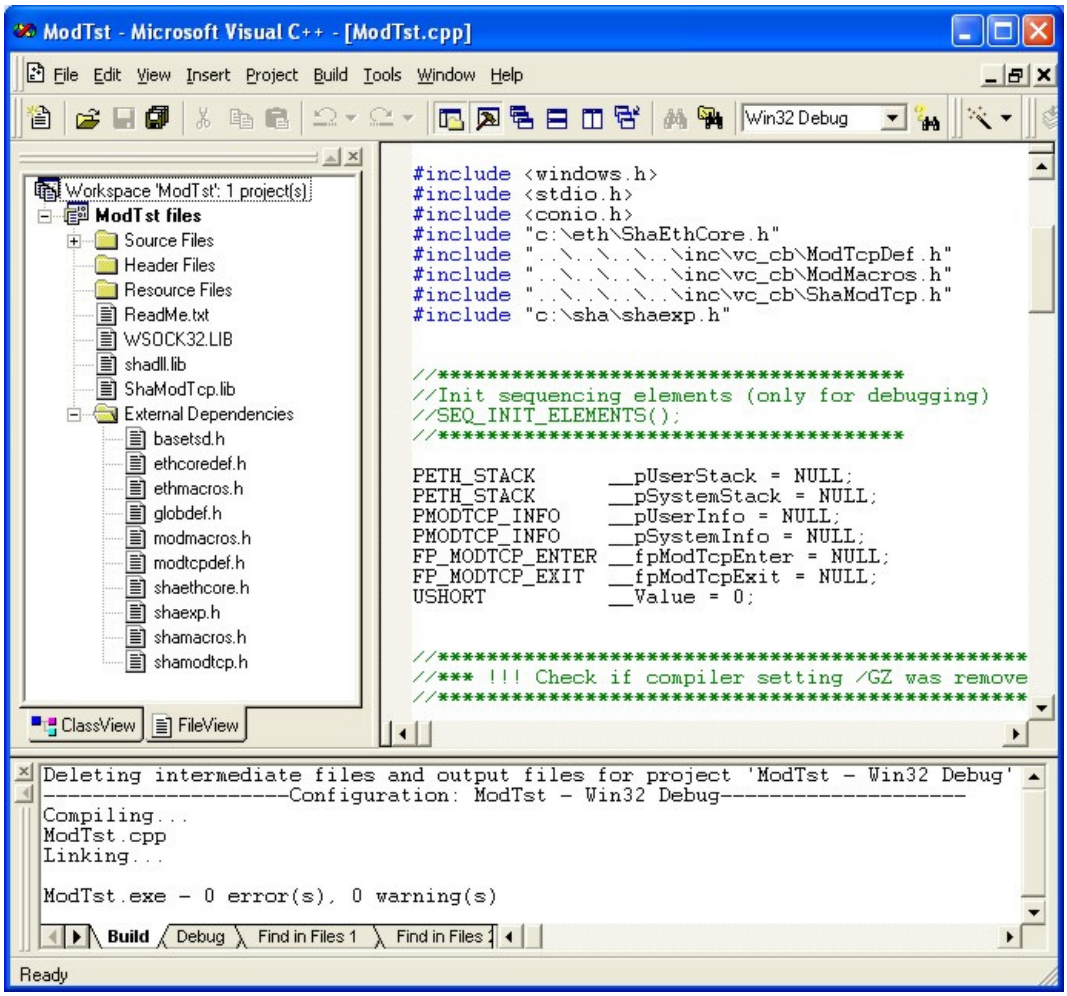
Note: For proper operation, make shure the *INTEL Speedstep Technologie* is turned OFF within the BIOS.

3 MODBus TCP Realtime Master Library

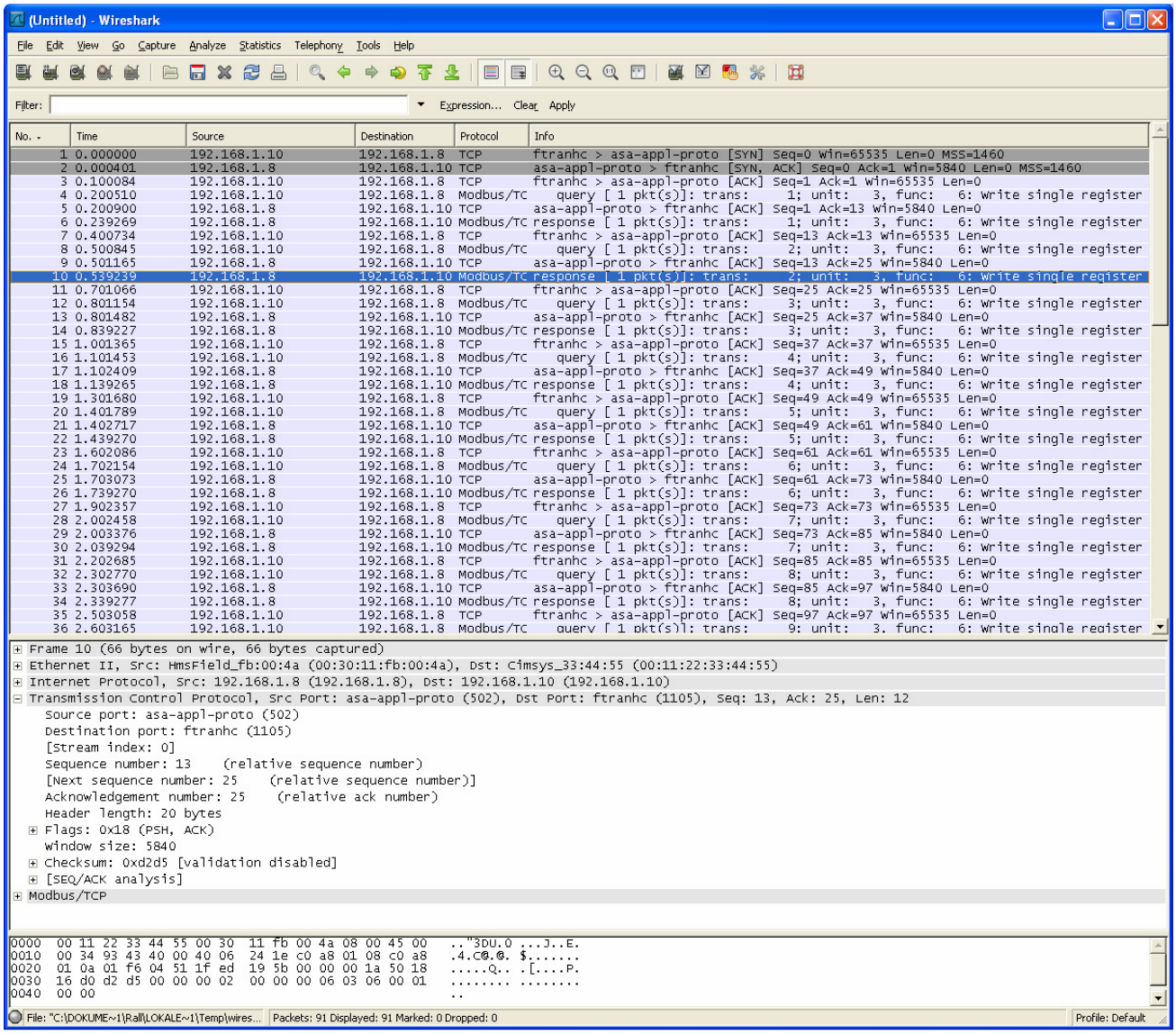
The interface functions of the MODBus TCP Realtime Master Library are exported by a dynamic link library. Following include files and libraries are required:

SHAMODTCP.DLL	MODBus TCP Master DLL (VISUAL C++)
SHAMODTCP.LIB	MODBus TCP Master LIB (VISUAL C++)
SHAMODTCPOML.DLL	MODBus TCP Master DLL (BORLAND C++ / Delphi)
SHAMODTCPOML.LIB	MODBus TCP Master LIB (BORLAND C++ / Delphi)
SHAMODTCP.H	Exported Function Prototypes
MODTCPDEF.H	MODBus TCP Basic Definitions
MODMACROS.H	MODBus TCP Macro Definitions

Sample Project



Sample Startup Protocoll:



3.1 Include File MODTCPDEF.H

The include file MODTCPDEF.H declares all required structures when handling MODBus TCP interface functions or handling the Core Realtime Stack directly (Realtime Level2).

3.1.1 Structure MOD_PARAMS

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

```
typedef struct _MODTCP_PARAMS
{
      //Input parameters
      TYPE32            HostIp;                 //HOST IP Address

      //Output parameters
      ULONG             core_dll_ver;           //Core DLL version
      ULONG             core_drv_ver;           //Core driver version
      FP_MODTCP_ENTER   fpModTcpEnter;          //Function       Pointer       to
ModTcpEnter()
      FP_MODTCP_EXIT    fpModTcpExit;           //Function       Pointer       to
ModTcpExit()

      //Input - Output parameters
      ETH_PARAMS        EthParams;              //Ethernet Core Parameters

} MODTCP_PARAMS, *PMODTCP_PARAMS;
```

Note:

The structure ETH_PARAMS is part of the Ethernet Core Library and described in the the documentation of this core libraray. Thus the Ethernet Core library must be installed first. The required elements of the structure ETH_PARAMS must be used in the same way as using the Ethernet realtime core.

3.1.2 Structure MODTCP_INFO

This structure keeps all information of each MODBus TCP modul and may be required for further interface functions.

```
typedef struct _MODTCP_INFO
{
      UCHAR             Mac[ETH_ADDRESS_LEN];   //MAC Address
      TYPE32            Ip;                     //IP Address
      USHORT            Port;                   //Port
      MODTCP_FRAME      Frame;                  //MODTCP Frame Data
      USHORT            Len;                    //MODTCP Frame Length
      BOOLEAN           bConnect;               //Connection Request Flag
      BOOLEAN           bTimeout;               //Timeout Flag
      MODTCP_MSG        Msg;                    //Messaging Elements
      __int64           Tsc;                    //Messaging TSC
      UCHAR             State;                  //Messaging State

} MODTCP_INFO, *PMODTCP_INFO;


     typedef struct _MODTCP_MSG
     {
          ULONG          QuerySeqNum;      //Query Sequence Number
          ULONG          QueryAckNum;      //Query Acknowledgement Number
          USHORT         QueryLen;         //Query Segment Length
          ULONG          RespSeqNum;       //Response Sequence Number
          ULONG          RespAckNum;       //Response Acknowledgement Number
          USHORT   RespLen;                //Response Segment Length

     } MODTCP_MSG, *PMODTCP_MSG;


     typedef struct _MODTCP_FRAME
     {
          MODTCP_HDR     Hdr;              //MODTCP Protocol Header
          UCHAR          Data[MAX_TCPDATA_SIZE - sizeof(MODTCP_HDR)];
                                           //MODBus Data
     } MODTCP_FRAME, *PMODTCP_FRAME;


            typedef struct _MODTCP_HDR
            {
                  USHORT      TransID;          //Transaction ID
                  USHORT      ProtID;           //Protocol ID
                  USHORT      Len;              //Length

            } MODTCP_HDR, *PMODTCP_HDR;
```

Note:

The elements *MAC*, *IP*, *PORT*, *bConnect*, *Frame.Data* and *Len* must be initialized before running.

Sample:

```
//Set MODTCP information
memset(__pUserInfo, 0, InfoSize);
memcpy(__pUserInfo->Mac, "\x00\x30\x11\xFB\x00\x4A", ETH_ADDRESS_LEN);
__pUserInfo->Ip.bit32 = inet_addr("192.168.1.8");
__pUserInfo->Port = 502;
__pUserInfo->bConnect = TRUE;
__pUserInfo->bTimeout = FALSE;

//Set initial output frame
MOD_REQ_WRITEREG(__pUserInfo->Frame.Data, 0x03, 0x0001, 0x0005);
__pUserInfo->Len = 6 + sizeof(MODTCP_HDR);
```

3.2 Include file MODMACROS.H

This include file defines all macros required for handling the realtime Task

//Macro to set request data to MODBus ADU
```
MOD_SET_REQUEST(__pAdu, __UnitID, __Func, __Addr, __DataSize, __pData)
```

//Macro to get response data from MODBus ADU
```
MOD_GET_RESONSE(__pAdu, __pUnitID, __pFunc, __pDataSize, __pData)
```

//Macro to set READ COIL request data to MODBus ADU
```
MOD_REQ_READCOIL(__pAdu, __UnitID, __Addr, __Quant)
```

//Macro to get READ COIL response from MODBus ADU
```
MOD_RESP_READCOIL(__pAdu, __pUnitID, __pByteCnt, __pData)
```

//Macro to set READ DISCRETE request data to MODBus ADU
```
MOD_REQ_READDISCRETE(__pAdu, __UnitID, __Addr, __Quant)
```

//Macro to get READ DISCRETE response from MODBus ADU
```
MOD_RESP_READDISCRETE(__pAdu, __pUnitID, __pByteCnt, __pData)
```

//Macro to set READ REGISTER request data to MODBus ADU
```
MOD_REQ_READREG(__pAdu, __UnitID, __Addr, __Quant)
```

//Macro to get READ REGISTER response from MODBus ADU
```
MOD_RESP_READREG(__pAdu, __pUnitID, __pByteCnt, __pData)
```

//Macro to set READ INPUT request data to MODBus ADU
```
MOD_REQ_READINPUT(__pAdu, __UnitID, __Addr, __Quant)
```

//Macro to get READ INPUT response from MODBus ADU
```
MOD_RESP_READINPUT(__pAdu, __pUnitID, __pByteCnt, __pData)
```

//Macro to set WRITE COIL request data to MODBus ADU
```
MOD_REQ_WRITECOIL(__pAdu, __UnitID, __Addr, __Value)
```

//Macro to get WRITE COIL response from MODBus ADU
```
MOD_RESP_WRITECOIL(__pAdu, __pUnitID, __pAddr, __pValue)
```

//Macro to set WRITE REGISTER request data to MODBus ADU
```
MOD_REQ_WRITEREG(__pAdu, __UnitID, __Addr, __Value)
```

//Macro to get WRITE REGISTER response from MODBus ADU
```
MOD_RESP_WRITEREG(__pAdu, __pUnitID, __pAddr, __pValue)
```

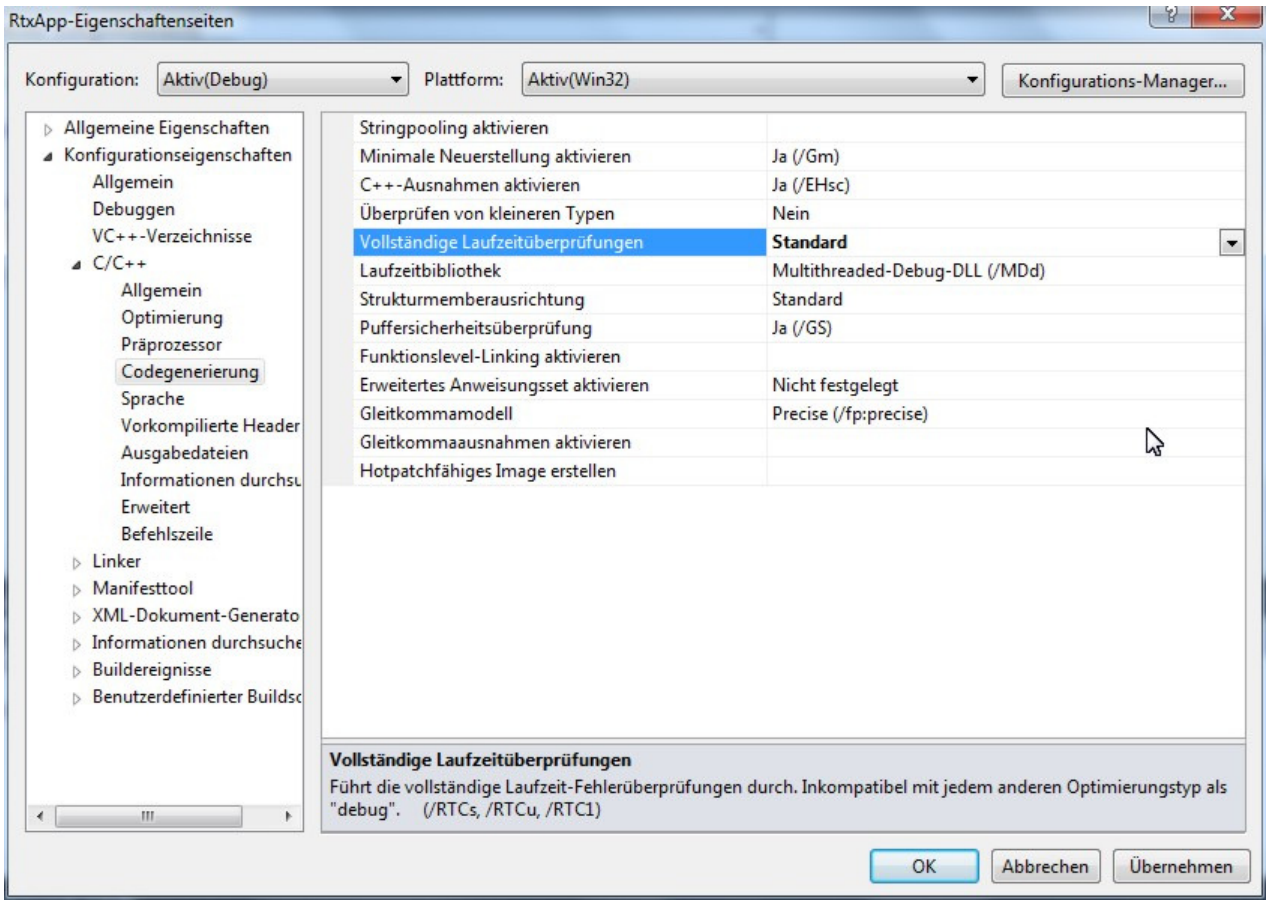
4 MODBus TCP Core Interface

The include file SHAMODTCP.H defines all required prototypes and parameters of the Ethernet Core Library. The include file is based on the files RAWCOREDEF.H and ETHCOREDEF.H. 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++, Borland C++ Builder and CVI Lab Windows
DP : Borland Delphi

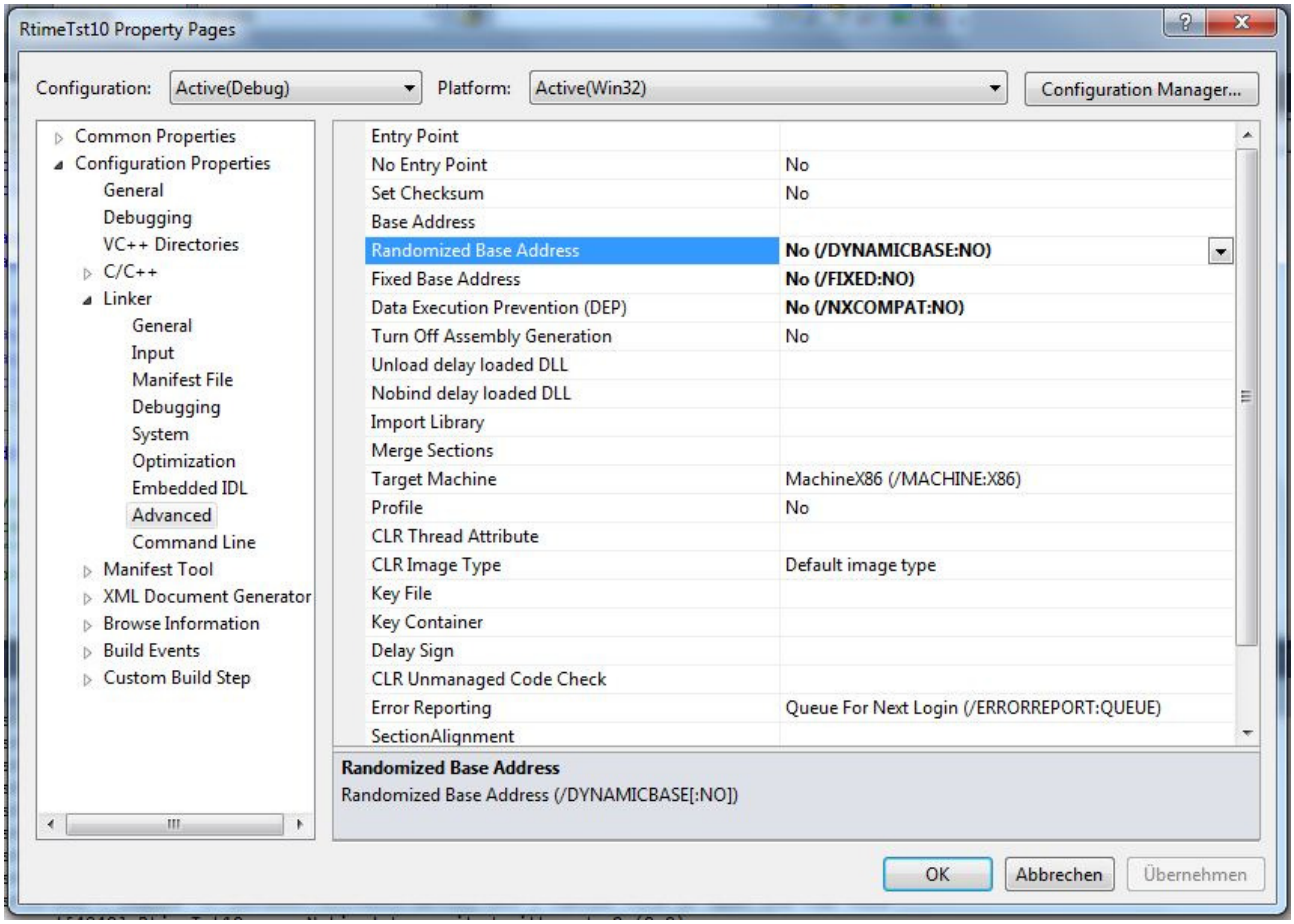
4.1.1 Visual Studio 2010 Compiler Settings

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:



4.1.3 ShaModTcpCreate

This function initializes the MODBus TCP module states. On success the returning value is ERROR_SUCCESS, otherwise the returning value corresponds to that with GetLastError().

VC
```
ULONG ShaModTcpCreate (PMOD_PARAMS);
```

4.1.4 ShaModTcpDestroy

This function closes the MODBus TCP communication.

VC
```
ULONG ShaModTcpDestroy(PMOD_PARAMS);
```

4.1.5 ShaModTcpGetVersion

This function retrieves the version information of the MODBus TCP Master Library, the Ethernet Core Library, the Ethernet Core Driver, the SHA Dll, the SHA Library and the SHA Driver.

VC
```
ULONG ShaModTcpGetVersion(PMOD_PARAMS);
```

Sample:

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "c:\eth\ShaEthCore.h"
#include "..\..\..\..\inc\vc_cb\ModTcpDef.h"
#include "..\..\..\..\inc\vc_cb\ModMacros.h"
#include "..\..\..\..\inc\vc_cb\ShaModTcp.h"
#include "c:\sha\shaexp.h"


PETH_STACK          __pUserStack = NULL;
PETH_STACK          __pSystemStack = NULL;
PMODTCP_INFO        __pUserInfo = NULL;
PMODTCP_INFO        __pSystemInfo = NULL;
FP_MODTCP_ENTER     __fpModTcpEnter = NULL;
FP_MODTCP_EXIT      __fpModTcpExit = NULL;
USHORT              __Value = 0;


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

void static AppTask(void)
{
 //Check if system memory is present
 if    ((!__pSystemStack)      ||
       (!__fpModTcpEnter)      ||
       (!__fpModTcpExit))
       return;

 //*************************************
 //Call MODTCP enter function
 if (__fpModTcpEnter(__pSystemStack, __pSystemInfo))
 {
       //Toggle QUERY value
       MOD_RESP_WRITEREG(__pSystemInfo->Frame.Data, NULL, NULL, &__Value);
       if (__Value == 0)
             MOD_REQ_WRITEREG(__pSystemInfo->Frame.Data, 0x03, 0x0001,
       0x0005);
       else
             MOD_REQ_WRITEREG(__pSystemInfo->Frame.Data, 0x03, 0x0001,
 0x0000);
 }
 //*************************************

 //*************************************
 //Call MODTCP exit function
 __fpModTcpExit(__pSystemInfo);
 //*************************************
}
```

```
void main(void)
{
 long InfoSize = sizeof(MODTCP_INFO);
 HANDLE hInfo = NULL;

 printf("\n*** MODBus Core Realtime Level2 Test ***\n\n");

 //Allocate memory for MODTCP information
 if (ERROR_SUCCESS == ShaAllocMemWithTag(
                          0,                //Memory TAG for attachments
                          IFT_PCIBUS,       //Interface type
                          TRUE,             //Busmaster memory
                          FALSE,            //Noncached physical memory
                          0,                //(Optional) Required channel for
                                            //system DMA
                          NULL,             //Physical memory address
                          &InfoSize,        //Memory size
                          (PVOID*)&__pUserInfo,   //Pointer to memory in user
                                                  //address space
                          (PVOID*)&__pSystemInfo, //Pointer to memory in
 system
                                                  //address space
                          &hInfo))                //Handle to memory device
 {
       //Set MODTCP information
       memset(__pUserInfo, 0, InfoSize);
       memcpy(__pUserInfo->Mac, "\x00\x30\x11\xFB\x00\x4A",
 ETH_ADDRESS_LEN);
       __pUserInfo->Ip.bit32 = inet_addr("192.168.1.8");
       __pUserInfo->Port = 502;
       __pUserInfo->bConnect = TRUE;
       __pUserInfo->bTimeout = FALSE;

       //Set initial output frame
       MOD_REQ_WRITEREG(__pUserInfo->Frame.Data, 0x03, 0x0001, 0x0005);
       __pUserInfo->Len = 6 + sizeof(MODTCP_HDR);

       //Required MOD parameters
       MODTCP_PARAMS ModTcpParams;
       ModTcpParams.EthParams.dev_num = 0;
       ModTcpParams.EthParams.period = 100000; //[µsec]
       ModTcpParams.EthParams.sched_cnt = 1;
       ModTcpParams.EthParams.fpAppTask = AppTask;

       //Enable MODTCP core
       if (ERROR_SUCCESS == ShaModTcpCreate(&ModTcpParams))
       {
             //Init realtime elements
             __fpModTcpEnter   = ModTcpParams.fpModTcpEnter;
             __fpModTcpExit    = ModTcpParams.fpModTcpExit;
             __pUserStack      = ModTcpParams.EthParams.pUserStack;
             __pSystemStack    = ModTcpParams.EthParams.pSystemStack;

             //Get version information
             ShaModTcpGetVersion(&ModTcpParams);
```

```
            printf("MODTCP-DLL  : %.2f\nMODTCP-DRV  : %.2f\n",
                  ModTcpParams.core_dll_ver / (double)100,
                  ModTcpParams.core_drv_ver / (double)100);
            printf("ETHCORE-DLL : %.2f\nETHCORE-DRV : %.2f\n",
                  ModTcpParams.EthParams.core_dll_ver / (double)100,
                  ModTcpParams.EthParams.core_drv_ver / (double)100);
            printf("SHA-LIB     : %.2f\nSHA-DRV     : %.2f\n",
                  ModTcpParams.EthParams.sha_lib_ver / (double)100,
                  ModTcpParams.EthParams.sha_drv_ver / (double)100);
            printf("\n");

            //Do a check loop
            printf("Press any key ...\n");
            while (!kbhit())
            {
                  if (__pUserInfo->bTimeout)
                  {
                        //Reset TCP timeout flag
                        __pUserInfo->bTimeout = FALSE;

                        //Press key to continue
                        printf("TCP Timeout - Press [ENTER] ...\n");
                        getchar();
                  }

                  //Do some delay
                  Sleep(100);
            }

            //Disconnect PORT
            __pUserInfo->bConnect = FALSE;
            Sleep(1000);

            //Detach memory
            ShaModTcpDestroy(&ModTcpParams);
      }
      //Free info memory
      ShaFreeMem(hInfo);
  }
}
```

5 Realtime Operation

After enabling the MODBus TCP system (*ShaModTcpCreate*), the realtime tasks become active. The application realtime task is decorated by Realtime Wrapper functions:

```
//Call MODTCP enter function
BOOLEAN __fpModTcpEnter(
                  __pSystemStack,   //In:  Ethernet Stack Pointer
                  __pSystemInfo);   //Out: MODTCP Information Pointer


typedef BOOLEAN   (__cdecl *FP_MODTCP_ENTER)(PETH_STACK, PMODTCP_INFO);
typedef VOID      (__cdecl *FP_MODTCP_EXIT) (PMODTCP_INFO);
```

These wrapper functions are used to manage the realtime ProfiNET protocol management, like ethernet frame update, error handling, stack management,…

Sample

```
void static AppTask(void)
{
  //Check if system memory is present
  if ((!__pSystemStack) ||
      (!__fpModTcpEnter)      ||
      (!__fpModTcpExit))
      return;

  //Call MODTCP enter function
  if (__fpModTcpEnter(__pSystemStack, __pSystemInfo))
  {
    //Toggle QUERY value
    MOD_RESP_WRITEREG(__pSystemInfo->Frame.Data, NULL, NULL, &__Value);
    if (__Value == 0)
         { MOD_REQ_WRITEREG(__pSystemInfo->Frame.Data, 0x03, 0x0001, 0x0005); }
    else { MOD_REQ_WRITEREG(__pSystemInfo->Frame.Data, 0x03, 0x0001, 0x0000); }
  }

  //Call MODTCP exit function
  __fpModTcpExit(__pSystemInfo);
}
```

6 Error Handling

The master library provides an error handling and tracing mechanism.

6.1 Event File

On execution the master library logs error event to the Windows Event Manager. The master library logs Application and System events. These events can be exported to a file and provided for support purposes.