TM

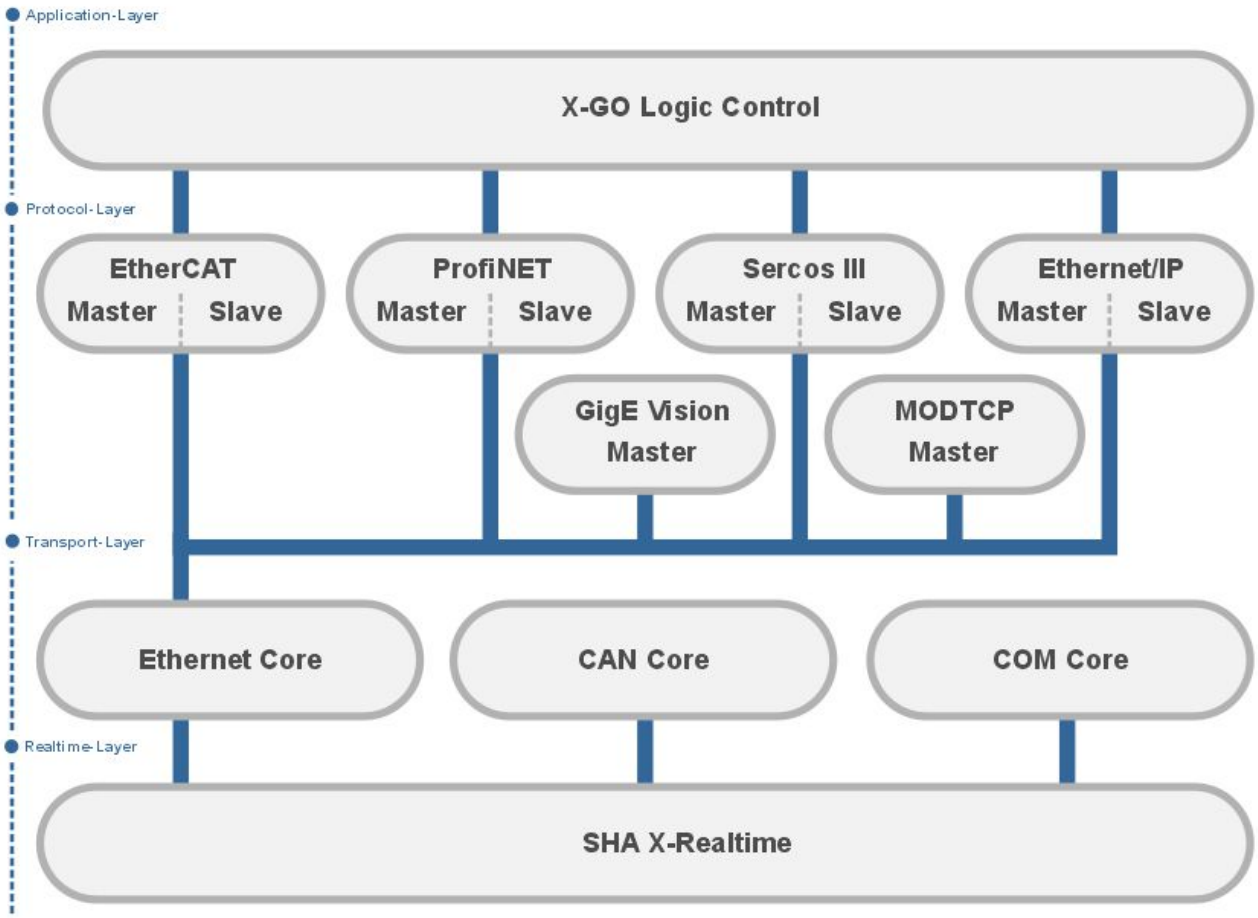
Ethernet Realtime Core Library (64 Bit) Documentation

Date: April, 14.2021

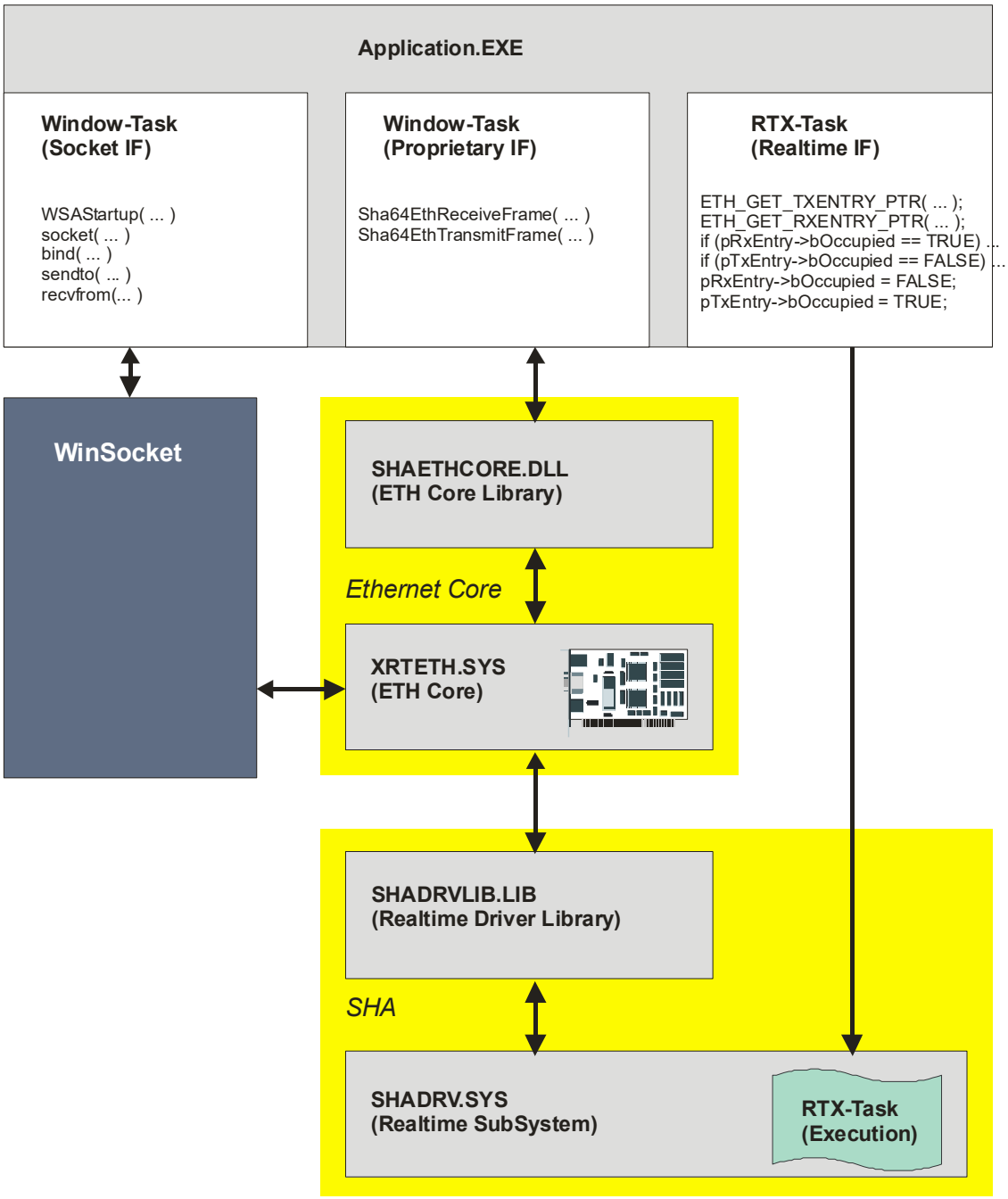
Ethernet Realtime Core Library Documentation

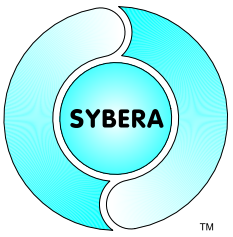
1 Introduction

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 Transport Stack Libraries, so called *Realtime Cores*. All *Realtime Cores* are based on the SHA X-Realtime system.



The buffer management of the ethernet real-time core is connected to several programming interfaces. One is for cyclic frame handling inside the real-time application task, one is for acyclic frame transmission of WinSocket applications and another one is for acyclic frame transmission by a proprietary interface. The basic of all these interfaces is a priority controlled ring buffer system, for TX (sending frames) and RX (receiving frames. Therefore 3 real-time tasks are involved:





The ethernet core executes 3 task (4 with ERR-Task) in a defined priority order (see parameters). The given period therefor defines one cycle (the execution of TX, RX, (ERR) and APP task).

TX task

- transmitting ethernet frames from buffer
- placed inside ethernet core
- typically highest priority

RX task

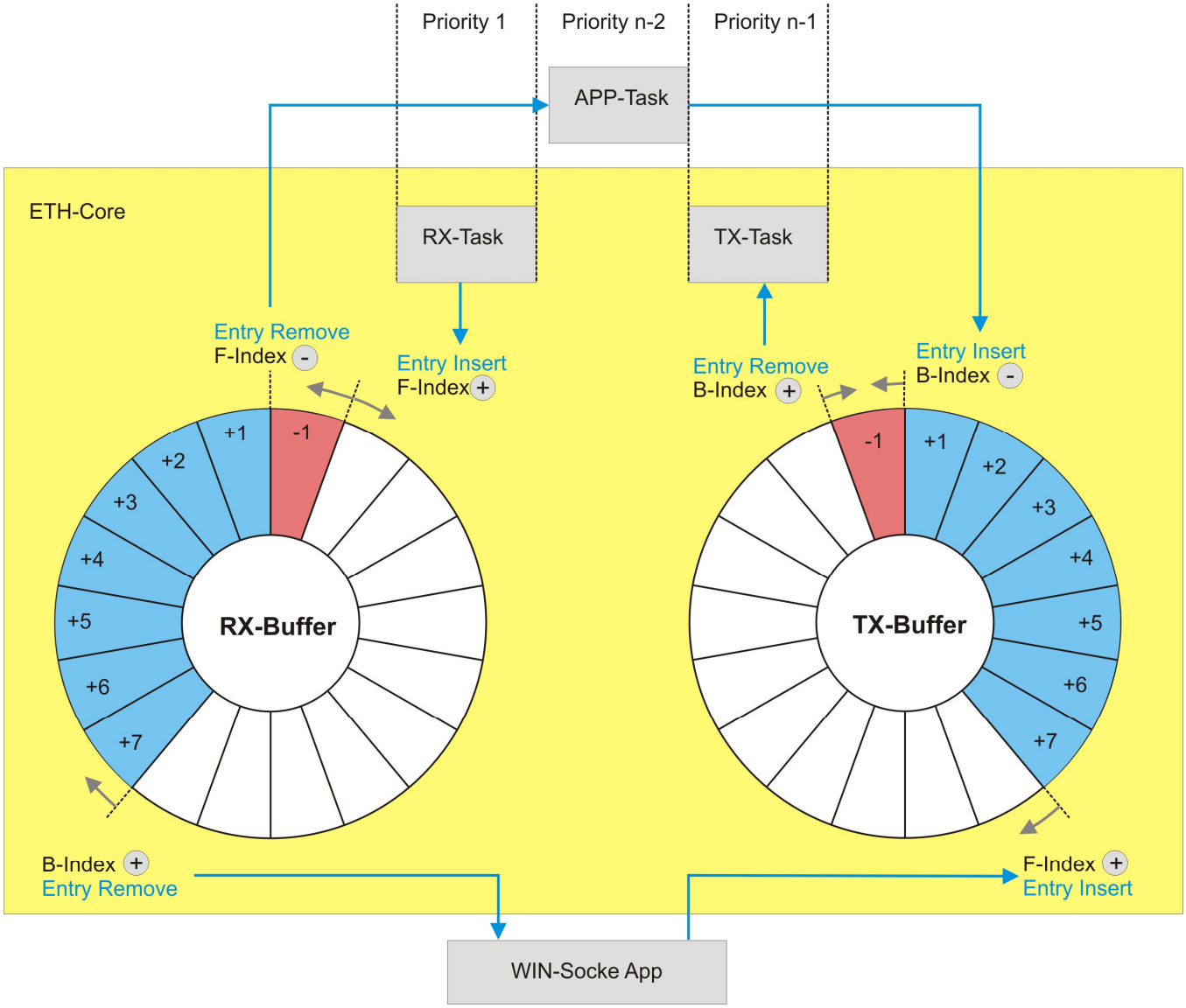
- receiving ethernet frames into buffer
- placed inside ethernet core
- typically highest priority - 1

APP task

- logical data operation
- placed inside rea-time application
- typically low priority - 2

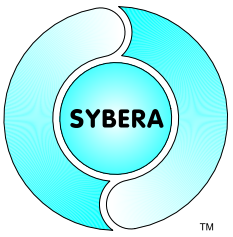
The ring buffer management consists of 2 table structures (RX and TX), containing elements of type of ETH_ENTRY. The ring buffer itself is controlled by a forward index (findex) and a backward index (bindex).

```
typedef struct _ETH_ENTRY
{
      PETH_FRAME  pFrame;          //Pointer to Frame
      USHORT      FrameSize;       //Frame Size
      BOOLEAN     bOccupied;       //Occupied flag
      __int64     TscCnt;          //Time Stamp Counter
      UCHAR       FramePhase;      //Frame Phase
      UCHAR       ClockSet;        //Clock Setting
      ULONG64     TimeStamp;       //Timestamp

} ETH_ENTRY, *PETH_ENTRY;
```



On receiving frames (RX) the integrated RX task first puts a new frame at the current forward index position, then increments the index. The following real-time APP task removes the entry by the forward index and then decrements the index. Asynchronously the WinSocket (or proprietary) interface removes an pending entry (due to the filter settings) by the current backward index, and increments the index.

On sending frames (TX) the real-time APP task inserts a new entry and then decrements the backward index. Afterwards the integrated TX task removes the entry by the backward index, then increments the index. Asynchronously the WinSocket (or proprietary) interface inserts new frames (due to the filter settings) by the forward index, then increments the index.



1.1 Features

- Multiport Ethernet Support
- Direct Ethernet frame data access in Windows application
- Core interface for Ethernet communication (Level1)
- Core interface for Ethernet communication (Level2)
- Socket interface for Ethernet communication (Level1)
- Mixed Ethernet communication (Level1 and Level2)
- Priority controlled frame buffer management
- True 64 bit mode for application and realtime task
- Support of Large Frames upto 8K
- Optimized DMA caching

1.2 Supported Development Environments

SHA was build to support serveral development platforms. Currently following platforms are supported:

- Visual C++ (from Version 8)
- CVI LabWindows

1.3 Supported OS

- Windows 7
- Windows 8
- Windows 10

1.4 Supported Hardware

The "Ethernet Realtime Core" supports currently following Hardware:

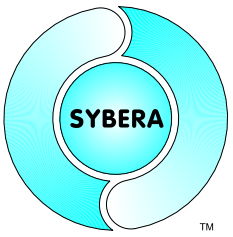
Intel (100MBit)

PCI\VEN_8086&DEV_1031 ; Intel ICH2
PCI\VEN_8086&DEV_1032 ;
PCI\VEN_8086&DEV_1033 ;
PCI\VEN_8086&DEV_1034 ;
PCI\VEN_8086&DEV_1039 ; Intel ICH2
PCI\VEN_8086&DEV_103A ; Intel ICH4
PCI\VEN_8086&DEV_103B ;
PCI\VEN_8086&DEV_103C ;
PCI\VEN_8086&DEV_103D ; Intel PRO/100 VE
PCI\VEN_8086&DEV_103E ;
PCI\VEN_8086&DEV_1050 ; Intel ICH5
PCI\VEN_8086&DEV_1051 ;
PCI\VEN_8086&DEV_1052 ;
PCI\VEN_8086&DEV_1053 ;
PCI\VEN_8086&DEV_1054 ;
PCI\VEN_8086&DEV_1055 ;
PCI\VEN_8086&DEV_1059 ; Intel PRO/100 M
PCI\VEN_8086&DEV_1092 ; Intel PRO/100 VE
PCI\VEN_8086&DEV_10FE ; Intel 8255x
PCI\VEN_8086&DEV_1209 ; Intel 8255x
PCI\VEN_8086&DEV_1229 ; Intel PRO/100
PCI\VEN_8086&DEV_2449 ; Intel(R) PRO/100E Adapter

Intel (1GBit)

PCI\VEN_8086&DEV_1000 ; Intel 82542
PCI\VEN_8086&DEV_1001 ; Intel 82543GC
PCI\VEN_8086&DEV_1008 ; Intel 82544EI
PCI\VEN_8086&DEV_1004 ; Intel 82543GC
PCI\VEN_8086&DEV_1009 ; Intel 82544EI
PCI\VEN_8086&DEV_100C ; Intel 82543EI
PCI\VEN_8086&DEV_100D ; Intel 82544GC
PCI\VEN_8086&DEV_100E ; Intel 82540EM
PCI\VEN_8086&DEV_100F ; Intel 82545EM
PCI\VEN_8086&DEV_1010 ; Intel 82546EB
PCI\VEN_8086&DEV_1011 ; Intel 82545EM
PCI\VEN_8086&DEV_1012 ; Intel 82546EB
PCI\VEN_8086&DEV_1015 ; Intel 82540EM
PCI\VEN_8086&DEV_101D ; Intel 82546EB
PCI\VEN_8086&DEV_1013 ; Intel 82541EI
PCI\VEN_8086&DEV_1016 ; Intel 82540EP
PCI\VEN_8086&DEV_1017 ; Intel 82540EP
PCI\VEN_8086&DEV_1019 ; Intel 82547EI
PCI\VEN_8086&DEV_101E ; Intel 82540EP
PCI\VEN_8086&DEV_1018 ; Intel 82541EI
PCI\VEN_8086&DEV_101A ; Intel 82547EI
PCI\VEN_8086&DEV_1014 ; Intel 82541ER

PCI\VEN_8086&DEV_1026 ; Intel 82545GM
PCI\VEN_8086&DEV_1027 ; Intel 82545GM
PCI\VEN_8086&DEV_1028 ; Intel 82545GM
PCI\VEN_8086&DEV_1049 ; Intel 82566MM
PCI\VEN_8086&DEV_104A ; Intel 82566DM
PCI\VEN_8086&DEV_104B ; Intel 82566DC
PCI\VEN_8086&DEV_104D ; Intel 82566MC
PCI\VEN_8086&DEV_105E ; Intel 82571EB
PCI\VEN_8086&DEV_105F ; Intel 82571EB
PCI\VEN_8086&DEV_1060 ; Intel 82571EB
PCI\VEN_8086&DEV_1075 ; Intel 82547EI
PCI\VEN_8086&DEV_1076 ; Intel 82545GM
PCI\VEN_8086&DEV_1077 ; Intel 82547EI
PCI\VEN_8086&DEV_1078 ; Intel 82547EI
PCI\VEN_8086&DEV_1079 ; Intel 82546EB
PCI\VEN_8086&DEV_107A ; Intel 82546EB
PCI\VEN_8086&DEV_107B ; Intel 82546EB
PCI\VEN_8086&DEV_107C ; Intel 82547EB
PCI\VEN_8086&DEV_107D ; Intel 82572EI
PCI\VEN_8086&DEV_107E ; Intel 82572EI
PCI\VEN_8086&DEV_107F ; Intel 82572EI
PCI\VEN_8086&DEV_108B ; Intel 82573V
PCI\VEN_8086&DEV_108C ; Intel 82573E
PCI\VEN_8086&DEV_1096 ; Intel 80003ES2LAN
PCI\VEN_8086&DEV_1098 ; Intel 80003ES2LAN
PCI\VEN_8086&DEV_109A ; Intel 82573L
PCI\VEN_8086&DEV_10A4 ; Intel 82571EB
PCI\VEN_8086&DEV_10A5 ; Intel 82571EB
PCI\VEN_8086&DEV_10B9 ; Intel 82572EI
PCI\VEN_8086&DEV_10BA ; Intel 80003ES2LAN
PCI\VEN_8086&DEV_10BB ; Intel 80003ES2LAN
PCI\VEN_8086&DEV_10BC ; Intel 82571EB
PCI\VEN_8086&DEV_10BD ; Intel 82566DM-2
PCI\VEN_8086&DEV_10BF ; Intel 82567LF
PCI\VEN_8086&DEV_10CB ; Intel 82567V
PCI\VEN_8086&DEV_10CC ; Intel 82567LM-2
PCI\VEN_8086&DEV_10CD ; Intel 82567LF-2
PCI\VEN_8086&DEV_10CE ; Intel 82567V-2
PCI\VEN_8086&DEV_10D3 ; Intel 82574L
PCI\VEN_8086&DEV_10DE ; Intel 82567LM-3
PCI\VEN_8086&DEV_10DF ; Intel 82567LF-3
PCI\VEN_8086&DEV_10E5 ; Intel 82567LM-4
PCI\VEN_8086&DEV_10EA ; Intel 82577LM
PCI\VEN_8086&DEV_10EB ; Intel 82577LC
PCI\VEN_8086&DEV_10EF ; Intel 82578DM
PCI\VEN_8086&DEV_10F0 ; Intel 82578DC
PCI\VEN_8086&DEV_10F5 ; Intel 82567LM
PCI\VEN_8086&DEV_10F6 ; Intel 82574L
PCI\VEN_8086&DEV_1501 ; Intel 82567V-3
PCI\VEN_8086&DEV_1502 ; Intel 82579LM
PCI\VEN_8086&DEV_1503 ; Intel 82579V
PCI\VEN_8086&DEV_150C ; Intel 82583V
PCI\VEN_8086&DEV_1520 ; I350 Ethernet Controller Virtual Function
PCI\VEN_8086&DEV_1521 ; I350 Gigabit Network Connection
PCI\VEN_8086&DEV_1522 ; I350 Gigabit Fiber Network Connection
PCI\VEN_8086&DEV_1523 ; I350 Gigabit Backplane Connection
PCI\VEN_8086&DEV_1524 ; I350 Gigabit Connection



PCI\VEN_8086&DEV_1526 ; 82576 Gigabit Network Connection
PCI\VEN_8086&DEV_1533 ; Intel I210
PCI\VEN_8086&DEV_1536 ; I210 Gigabit Fiber Network Connection
PCI\VEN_8086&DEV_1537 ; I210 Gigabit Backplane Connection
PCI\VEN_8086&DEV_1538 ; I210 Gigabit Network Connection
PCI\VEN_8086&DEV_1539 ; Intel I211
PCI\VEN_8086&DEV_153A ; Intel I217-LM
PCI\VEN_8086&DEV_153B ; Intel I217-V
PCI\VEN_8086&DEV_1559 ; Intel I218-V
PCI\VEN_8086&DEV_155A ; Intel I218-LM
PCI\VEN_8086&DEV_156F ; Intel I219-LM
PCI\VEN_8086&DEV_1570 ; Intel I219-V
PCI\VEN_8086&DEV_157B ; Intel I210
PCI\VEN_8086&DEV_157C ; I210 Gigabit Backplane Connection
PCI\VEN_8086&DEV_15A0 ; Intel I218-LM2
PCI\VEN_8086&DEV_15A1 ; Intel I218-V2
PCI\VEN_8086&DEV_15A2 ; Intel I218-LM
PCI\VEN_8086&DEV_15A3 ; Intel I218-V2
PCI\VEN_8086&DEV_15B7 ; Intel (2) I219-LM
PCI\VEN_8086&DEV_15B8 ; Intel (2) I219-V
PCI\VEN_8086&DEV_15B9 ; Intel (3) I219-LM
PCI\VEN_8086&DEV_15BB ; Intel (6) I219-LM
PCI\VEN_8086&DEV_15BC ; Ethernet Connection (7) I219-V
PCI\VEN_8086&DEV_15BD ; Ethernet Connection (6) I219-LM
PCI\VEN_8086&DEV_15BE ; Ethernet Connection (6) I219-V
PCI\VEN_8086&DEV_15D6 ; Intel (5) I219-V
PCI\VEN_8086&DEV_15D7 ; Intel (4) I219-LM
PCI\VEN_8086&DEV_15D8 ; Intel (4) I219-V
PCI\VEN_8086&DEV_15DF ; Ethernet Connection (8) I219-LM
PCI\VEN_8086&DEV_15E0 ; Ethernet Connection (8) I219-V
PCI\VEN_8086&DEV_15E1 ; Ethernet Connection (9) I219-LM
PCI\VEN_8086&DEV_15E2 ; Ethernet Connection (9) I219-V
PCI\VEN_8086&DEV_15E3 ; Intel (5) I219-LM
PCI\VEN_8086&DEV_15F4 ; Ethernet Connection (15) I219-LM
PCI\VEN_8086&DEV_15F5 ; Ethernet Connection (15) I219-V
PCI\VEN_8086&DEV_15F6 ; I210 Gigabit Ethernet Connection
PCI\VEN_8086&DEV_15F9 ; Ethernet Connection (14) I219-LM
PCI\VEN_8086&DEV_15FA ; Ethernet Connection (14) I219-V
PCI\VEN_8086&DEV_15FB ; Ethernet Connection (13) I219-LM
PCI\VEN_8086&DEV_15FC ; Ethernet Connection (13) I219-V
PCI\VEN_8086&DEV_294C ; Intel 82566DC-2
PCI\VEN_8086&DEV_0D4C ; Ethernet Connection (11) I219-LM
PCI\VEN_8086&DEV_0D4D ; Ethernet Connection (11) I219-V
PCI\VEN_8086&DEV_0D4E ; Ethernet Connection (10) I219-LM
PCI\VEN_8086&DEV_0D4F ; Ethernet Connection (10) I219-V
PCI\VEN_8086&DEV_0D53 ; Ethernet Connection (12) I219-LM
PCI\VEN_8086&DEV_0D55 ; Ethernet Connection (12) I219-V

Realtek (100MBit)

PCI\VEN_10EC&DEV_8139 ; Realtek
PCI\VEN_1186&DEV_1300 ; D-Link DFE-530TX+ PCI-Adapter
PCI\VEN_1113&DEV_1211 ; Accton SMC Series Fast Ethernet
PCI\VEN_1516&DEV_0800 ; Asound 10/100M Based Fast Ethernet-Karte
PCI\VEN_1516&DEV_0803 ; Asound 10/100M Based Fast Ethernet-Karte
PCI\VEN_1516&DEV_0891 ; Asound 10/100M Based Fast Ethernet-Karte
PCI\VEN_021b&DEV_8139 ; Compaq 10/100 Ethernet-PC-Karte
PCI\VEN_021b&DEV_8138 ; Compaq 10/100 Ethernet-PC-Karte

Realtek (1GBit)

PCI\VEN_10EC&DEV_8136 ; Realtek
PCI\VEN_10EC&DEV_8137 ; Realtek
PCI\VEN_10EC&DEV_8161 ; Realtek
PCI\VEN_10EC&DEV_8167 ; Realtek
PCI\VEN_10EC&DEV_8168 ; Realtek
PCI\VEN_10EC&DEV_8169 ; Realtek
PCI\VEN_173B&DEV_03EA ; BUFFALO LGY-PCI32-GT Gigabit Ethernet Adapter
PCI\VEN_1259&DEV_C107 ; Corega CG-LAPCIGT
PCI\VEN_1186&DEV_4300 ; D-Link DFE-5280T

2 Installation

2.1 Check third party software

First make shure that no other realtime software, or subsystem (e.g. BECKHOFF Twincat, NUMEGA SoftICE, ..) is running on the system.

2.2 Installing the Ethernet Core Software

For installation following steps are required:

- Make shure Library SHA64 has been installed before.
- Run SYSETUP64.EXE of Ethernet Core (as Administrator)
- Next Update existing network driver
- Reboot the system
- Build your program with the library interface
- Run the program

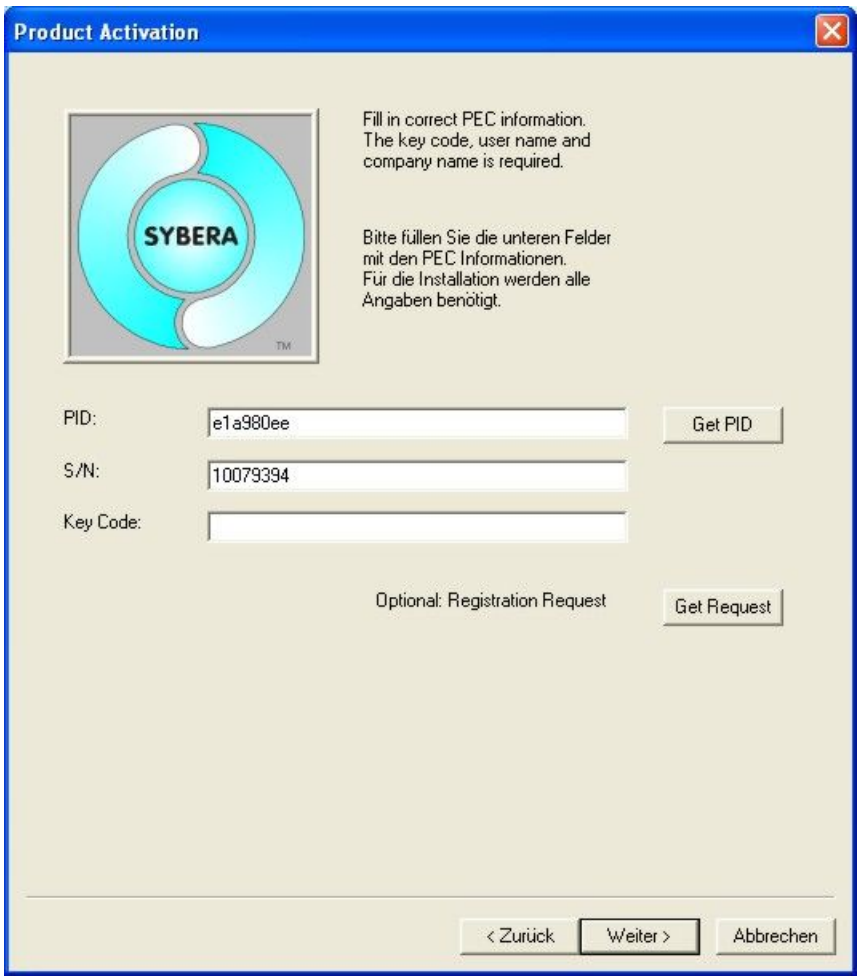
2.2.1 Product Activation

For product activation a PEC-Code (Product Enabling Code) is required, consisting of 3 items:

- PID: Platform Identifier (received by pressing the button)
- S/N: Serial number (received from license label or “12345678” for evaluation)
- KeyCode: Installation Code (received from license server or “12345678-12345678” for evaluation)

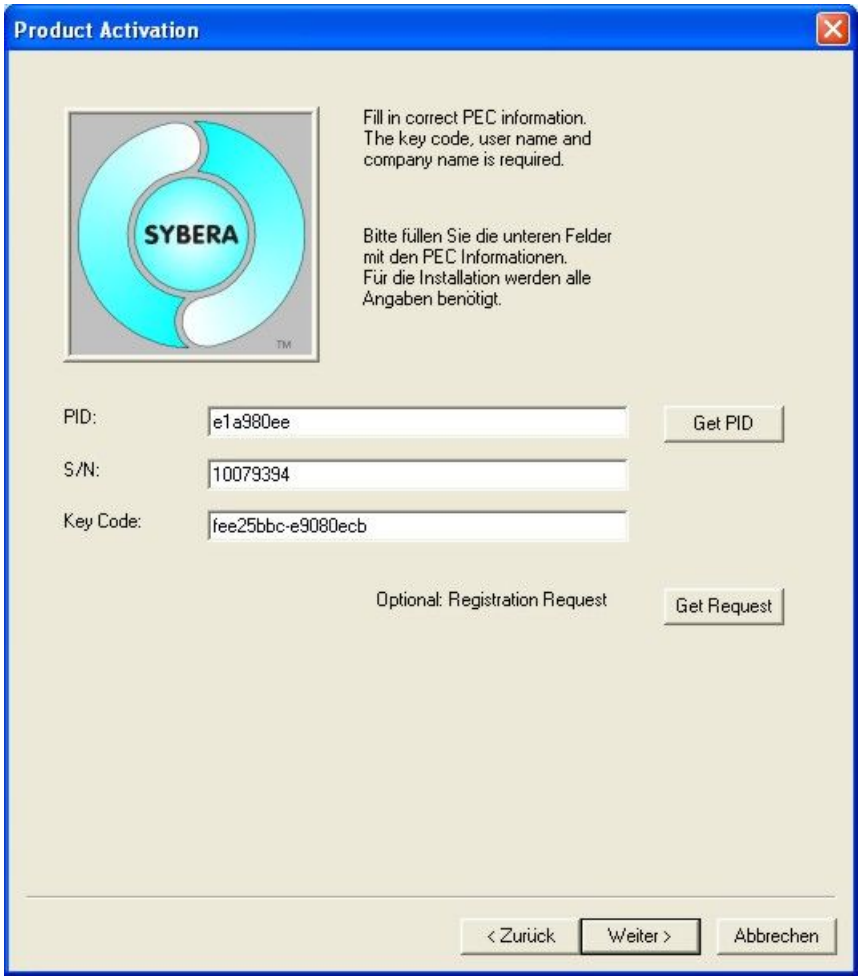
Each License is bound to a single PC-Platform by a corresponding PID-Code. This PID-Code is to be read out at Installation with the SYSETUP-Software, and sent to the SYBERA registration server for generating the valid PEC-Code (not required for Open-Volume-Licenses).

Get the PID code and enter the serial number



Depending if the PC is online, the KeyCode may be requested direct or indirect from the SYBERA registration server (see the registration manual)

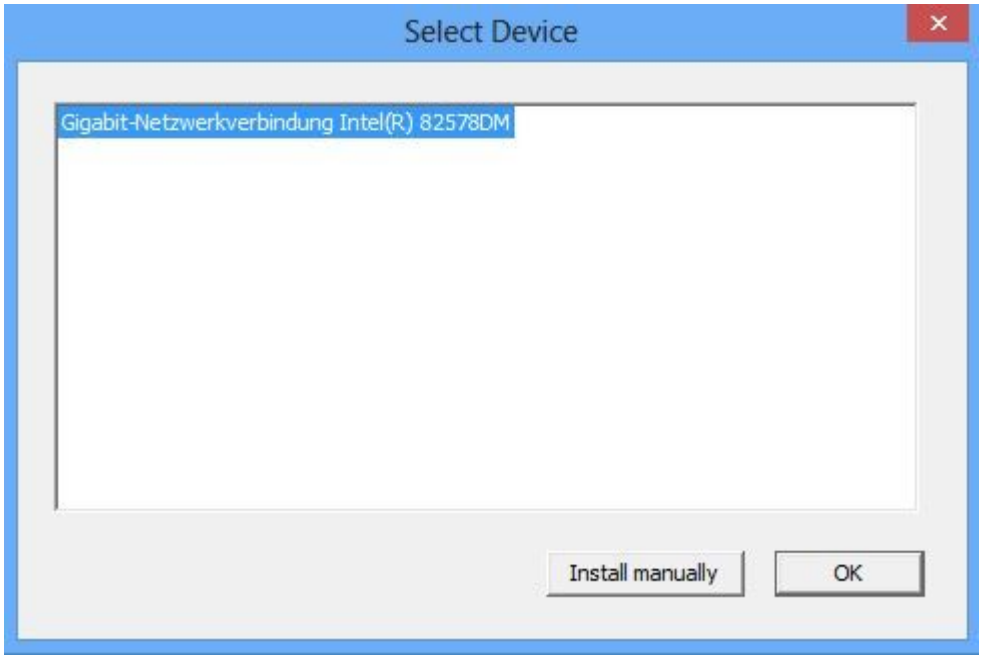
The SYBERA registration server will now provide the required PEC information for installation



2.2.2 Automatic Driver Install

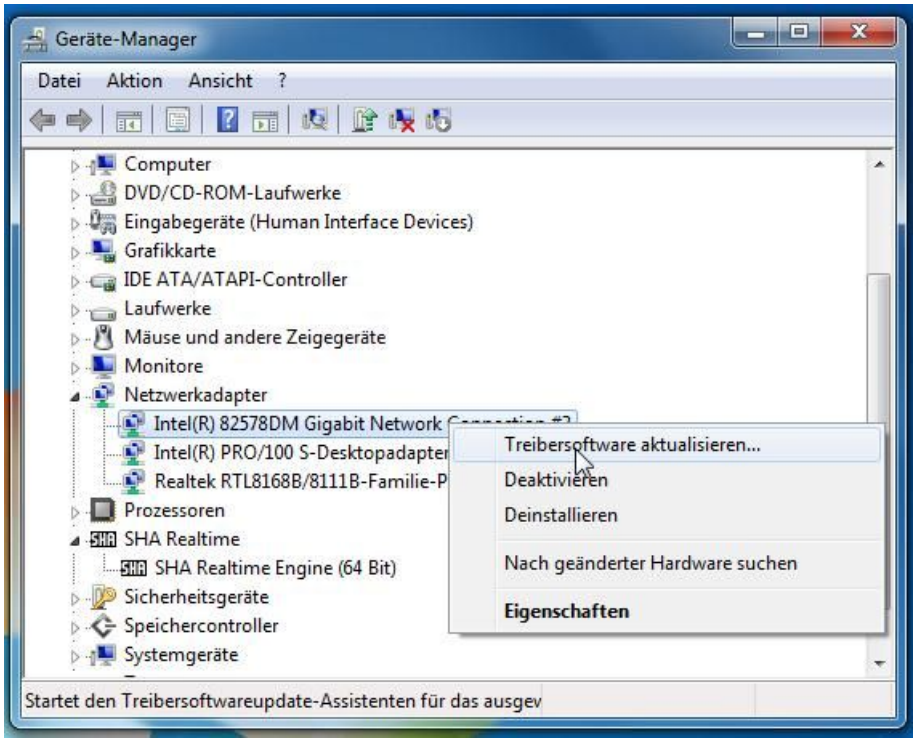
The Setup Software of the Ethernet Realtime Core let you choose an existing ethernet adapter to update it to a realtime driver.

Sample:



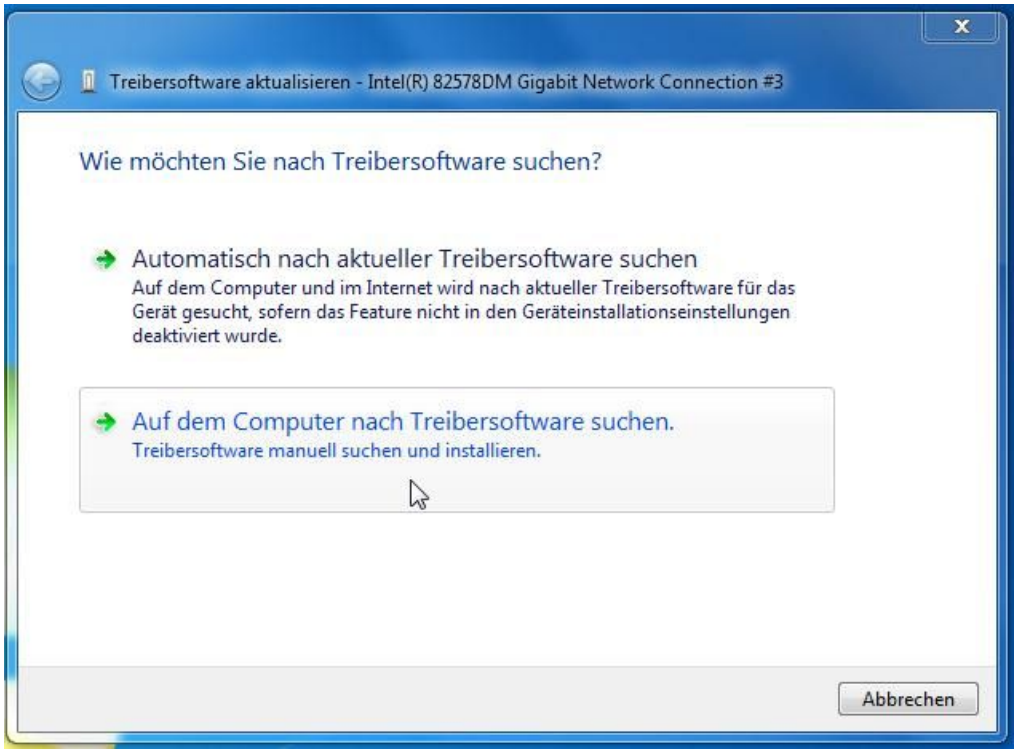
2.2.3 Manual Ethernet Realtime Driver Installation (optional)

The "Ethernet Realtime Core" is based on the INTEL or Realtek PCI(e)-Adapters (also PCMCIA and PCI express card) and will be installed as a NDIS NIC driver. When installing the Ethernet adapter, Windows won't ask for a PNP device driver and installs automatically its own NIC driver, since INTEL / REALTEK Adapters are a known hardware to the system. Therefore this driver must be updated with the Realtime Ethernet Driver (XRTETH64.SYS) inside the Windows Device Manager.

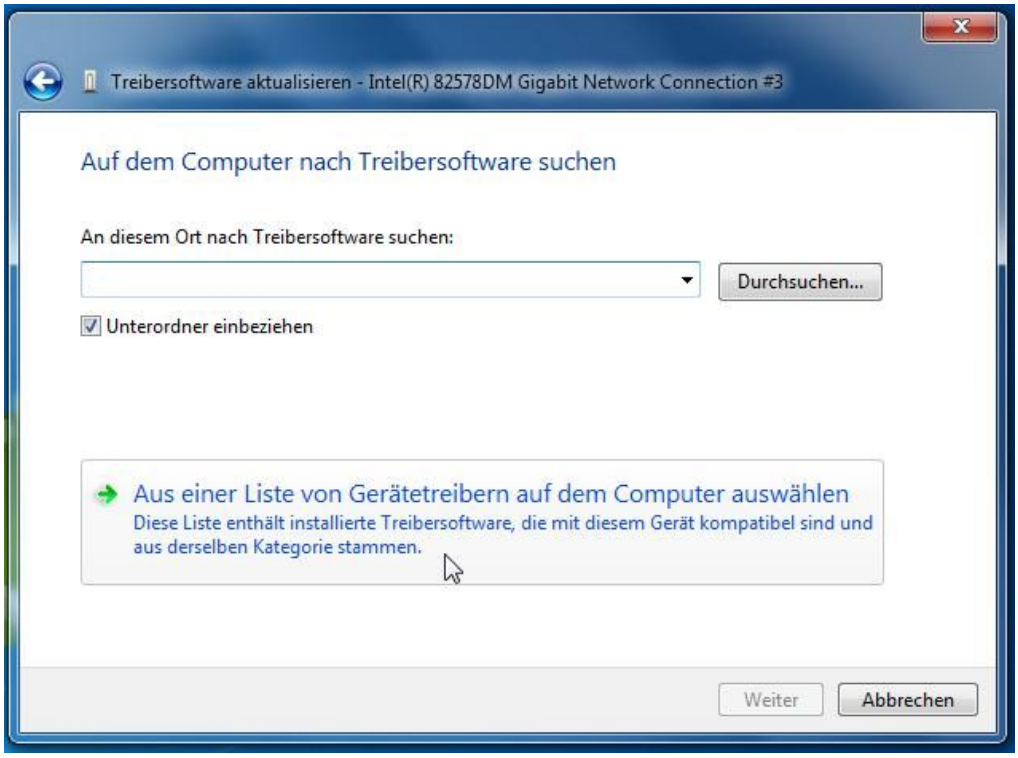


Note: The realtime ethernet driver is found at `ETHxxx\DRV\XRTETH64.SYS`

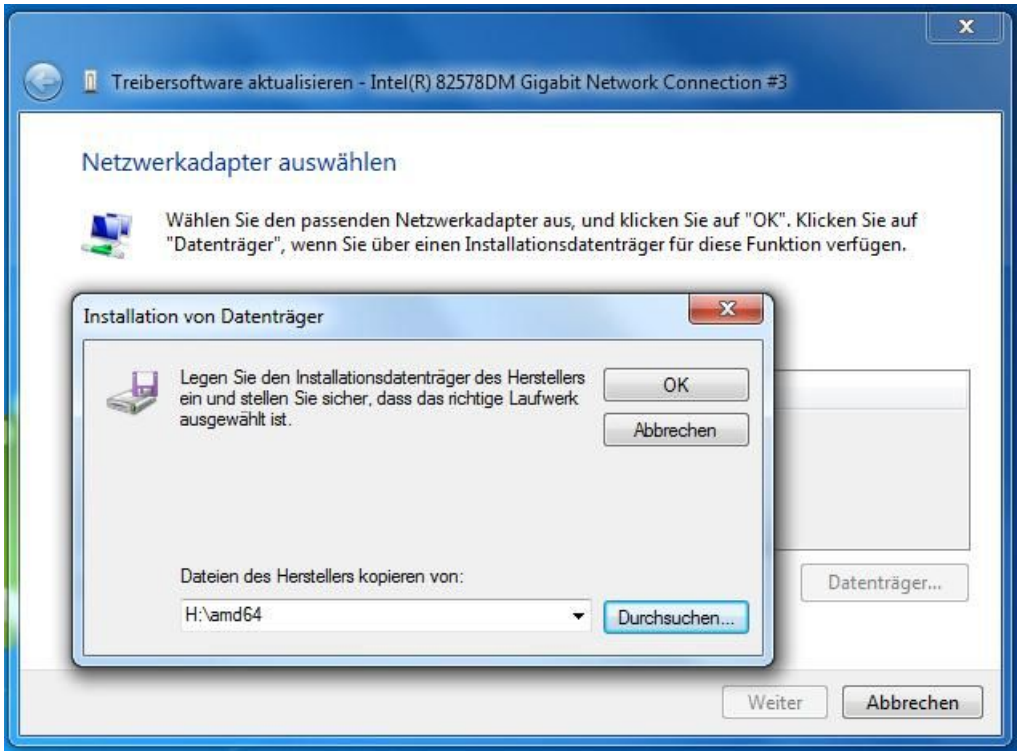
The new driver must be installed manually



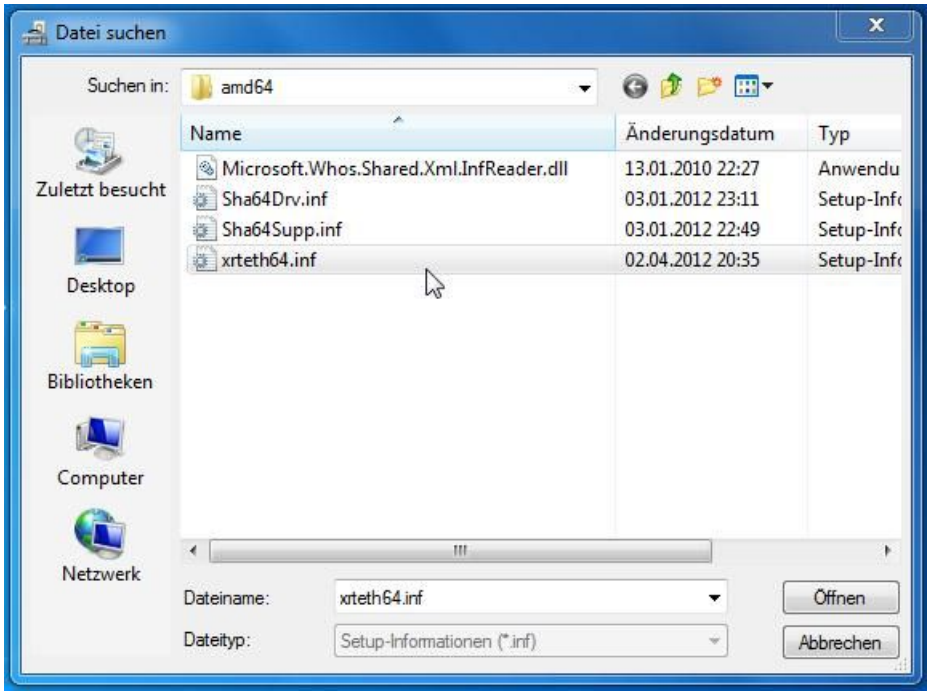
Choose the driver from a list



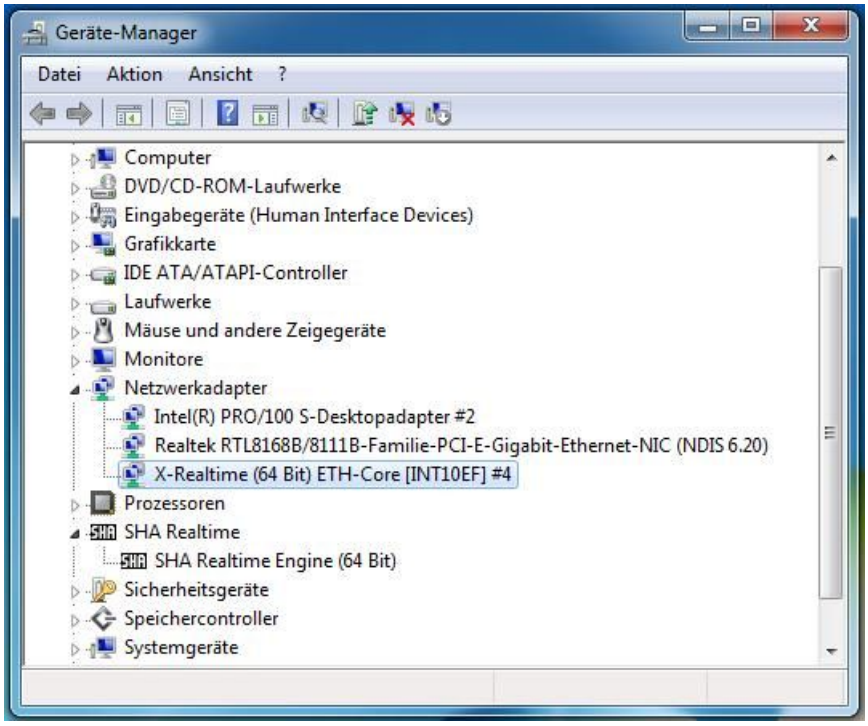
Therefore the path to the drivers XRTETH64.INF file must be given



Select the XRTETH64.INF file



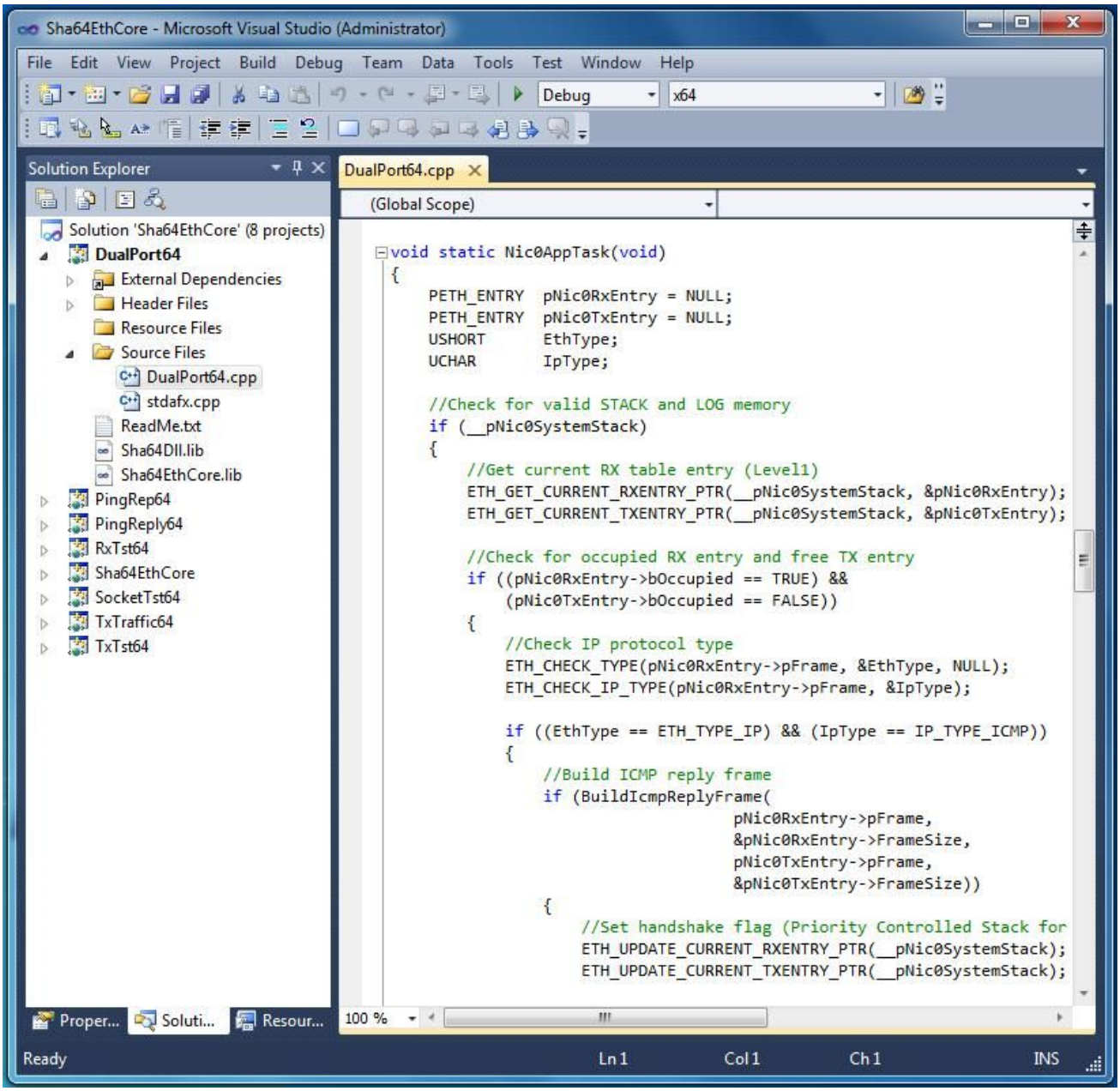
Check the new driver has been installed correctly



3 Features

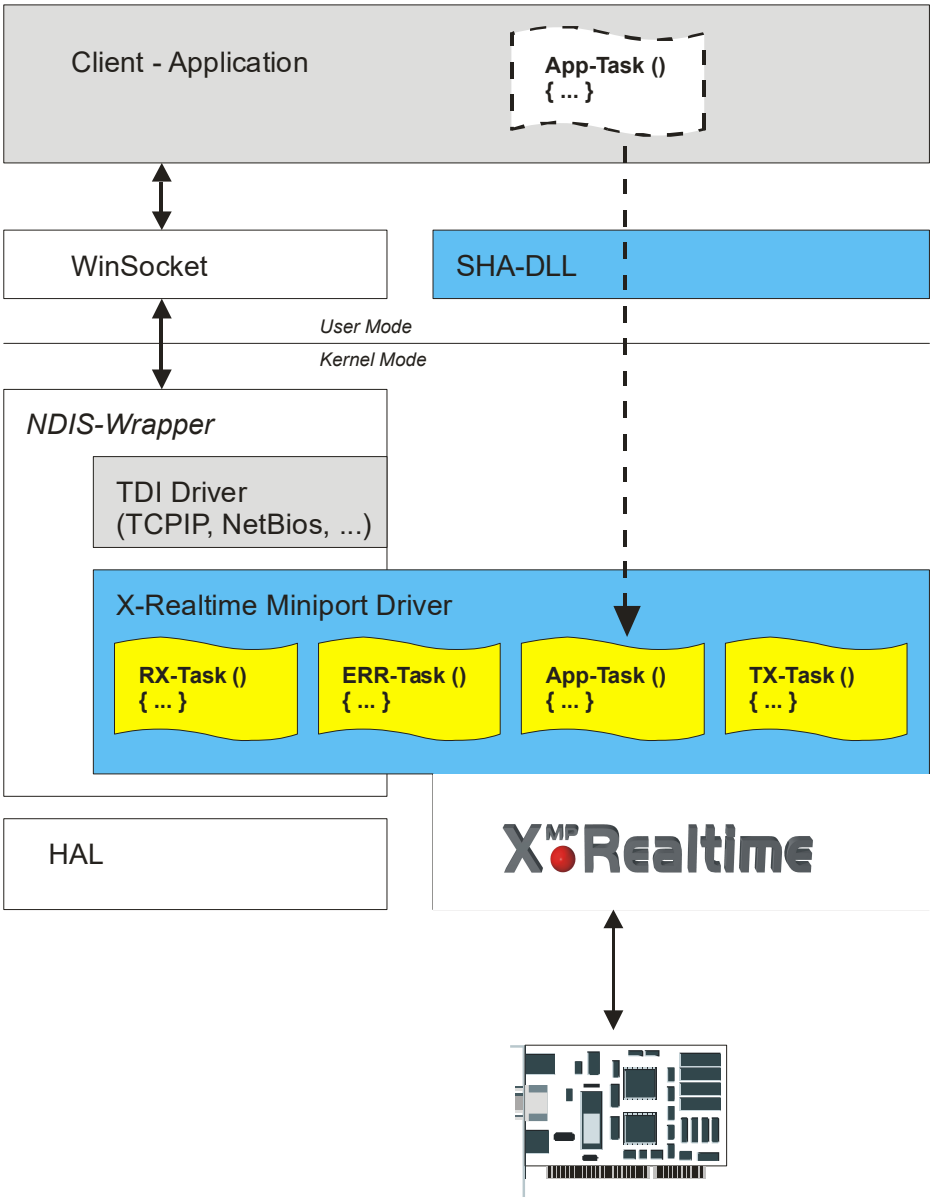
The realtime library "Ethernet Realtime Core" allows sending and receiving of ethernet frames, as well as the functional operation of data at realtime. The realtime library allows handling of ethernet frames at realtime level 1 (ring buffered ethernet frames) or at realtime level 2 (functional operation within realtime task). Therefore the ethernet frames can be accessed in RAW format. RAW means, that the complete ethernet frame data is available for operation.

Sample Project:



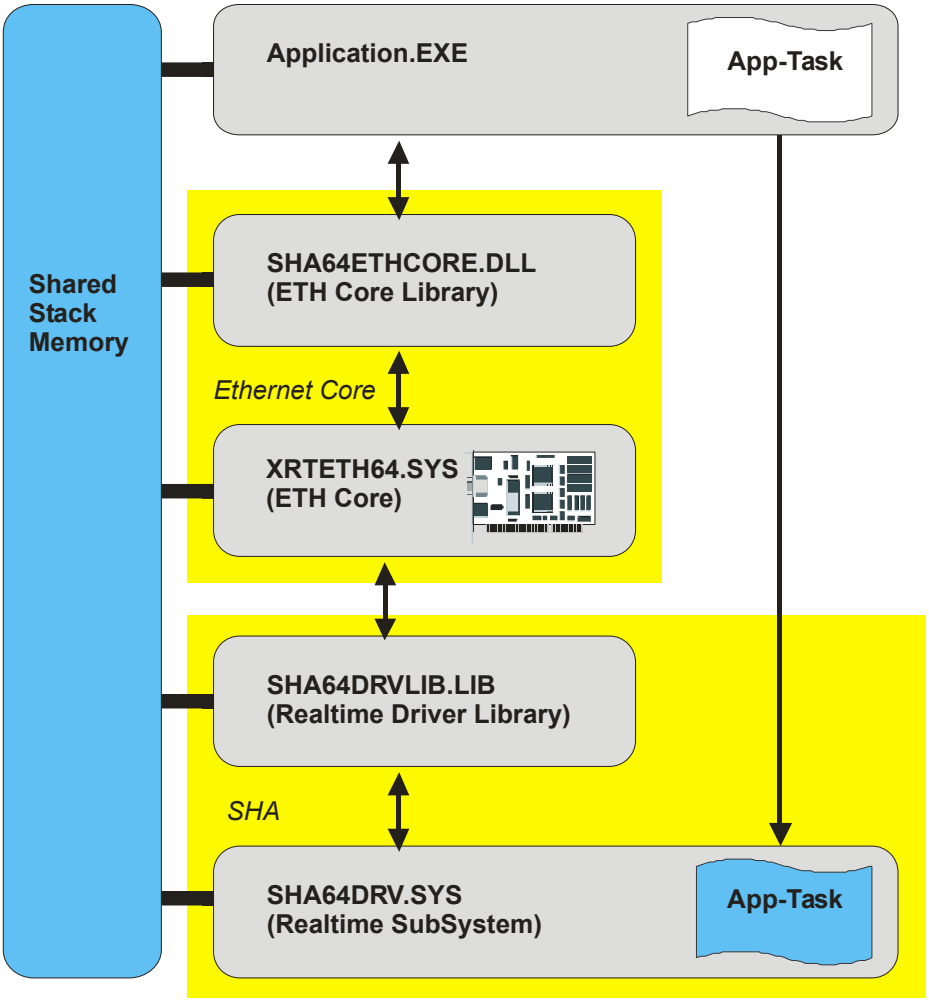
3.1 Ethernet Realtime Interface

A proprietary core interface, as well as the standard SOCKET interface is available for developing realtime applications. A filter management allows frames to be directed to the core interface, the proprietary interface or the windows socket interface.



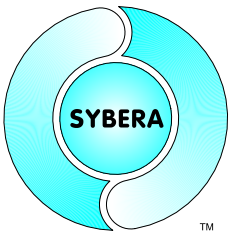
3.2 Ethernet Realtime Stack

The ethernet stack allows direct handling of ethernet frames at realtime. There is no need to access the hardware directly. The developer can focus on handling the raw frame data.



The core allow both, real-time Level1 (collecting and buffering of data at real-time, without loss of data), and Real-Time Level 2 (the cyclic functional operation at Real-Time, with one fixed buffer location). Additionally Level1 and Level2 may be combined. The buffer management of the ethernet real-time core is controlled by the flag lev2_flag of the core stack.

```
//Set core control elements
ETH_STACK_CONTROL(__pNicUserStack, 0, 0);       //Reset RX/TX buffer indexes
ETH_LEVEL2_CONTROL(__pNicUserStack, FALSE);     //Disable fixed buffer location
ETH_TASK_CONTROL(__pNicUserStack, TRUE);        //Enable RX and TX tasks
```



3.3 Multiple NIC Adapter Support

The Ethernet Core is designed to support multiple adapters at a time. For each adapter a Core Stack will be created with a indexing device number (**EthParams.DevNum** starting from 0).

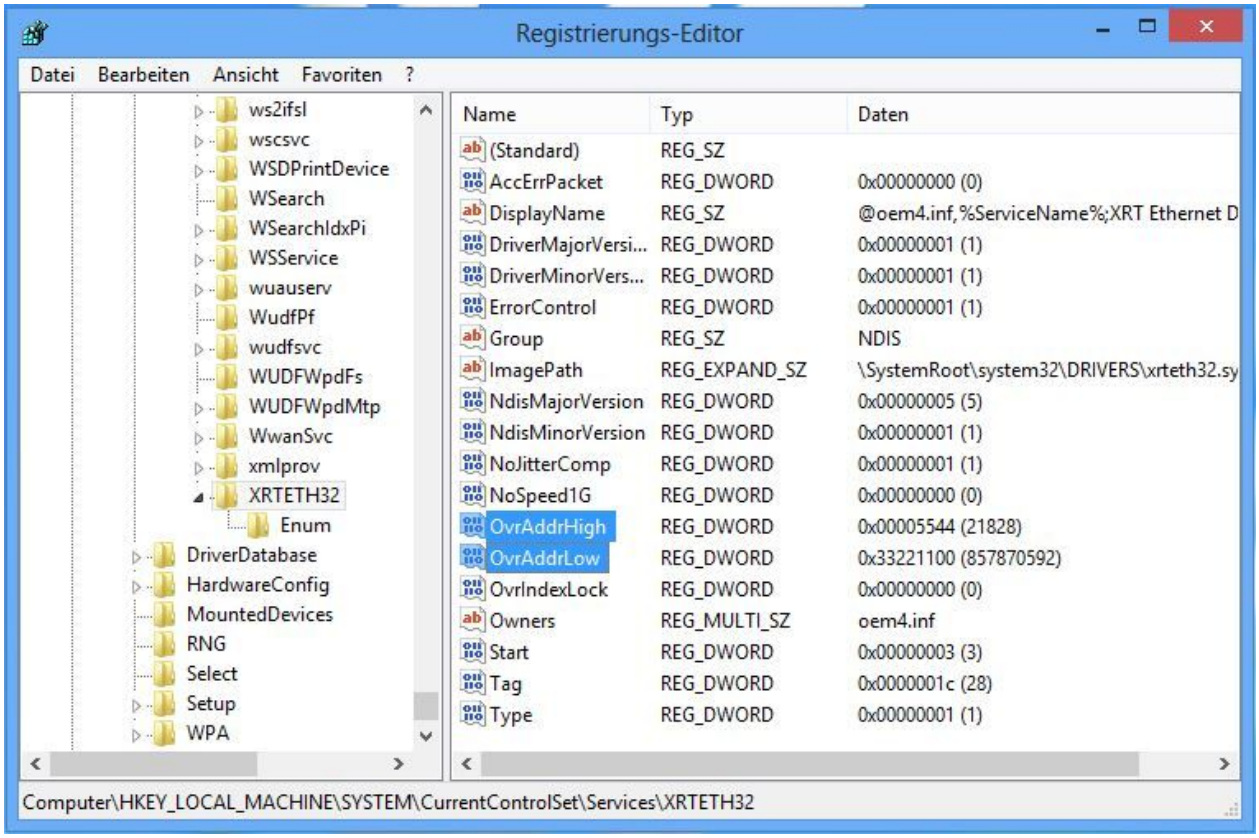
Sample

```
//*** Required ETH core parameters for NIC0 ***
memset(&ParamsNic0, 0, sizeof(ETH_PARAMS));
ParamsNic0.dev_num = 0;                    //Device number
ParamsNic0.period = 100;                   //Realtime period
ParamsNic0.eth_type = ETH_TYPE_IP;         //Set ethernet frame type filter for
                                           //the selected interface
ParamsNic0.eth_if = ETH_IF_CORE;           //Route all IP frames to the
                                           //CORE interface
ParamsNic0.fpAppTask = Nic0AppTask;        //Realtime level2 task for NIC0

//*** Required ETH core parameters for NIC1 ***
memset(&ParamsNic1, 0, sizeof(ETH_PARAMS));
ParamsNic1.dev_num = 1;                    //Device number
ParamsNic1.period = 100;                   //Realtime period
ParamsNic1.eth_type = ETH_TYPE_IP;         //Set ethernet frame type filter for
                                           //the selected interface
ParamsNic1.eth_if = ETH_IF_CORE;           //Route all IP frames to the
                                           //CORE interface
ParamsNic1.fpAppTask = Nic1AppTask;        //Realtime level2 task for NIC1

//Create ethernet core
Error = Sha64EthCreate(&ParamsNic0);
if (ERROR_SUCCESS == Error)
{

      //Create ethernet core
      Error = Sha64EthCreate(&ParamsNic1);
      if (ERROR_SUCCESS == Error)
      {
      …
      }
}
```

3.4 Override Address

The unique NIC adapter MAC address is not used by the Ethernet Realtime Core, instead a override address is used and may be changed. The default override address is:

OVERRIDE_ADDRESS = 00-11-22-33-44-55

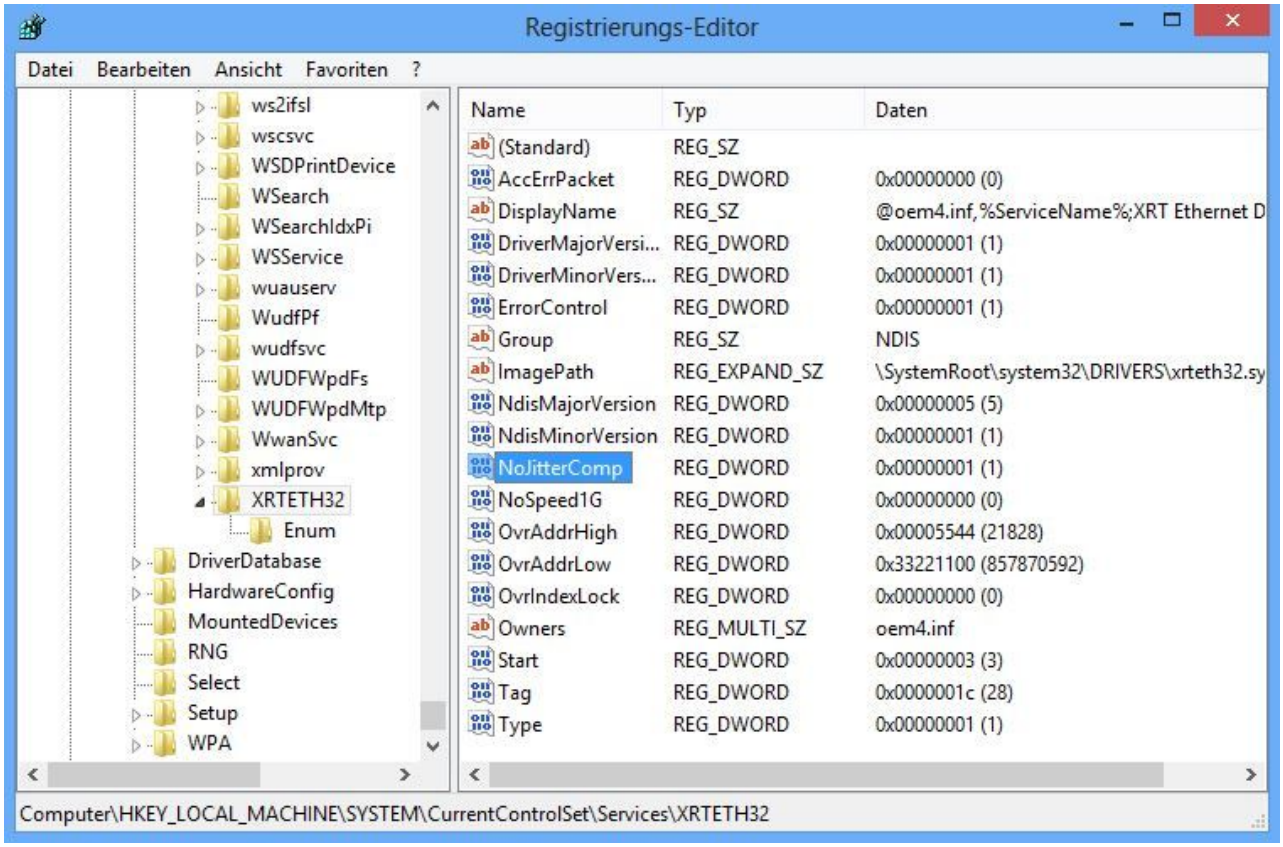


If more NIC adapters are installed, the MSB of the OVERRIDE_ADDRESS is added by the DevNum (e.g. DevNum = 1 -> . OVERRIDE_ADDRESS = 00-11-22-33-44-56)

For delay measurement applications (e.g. loopback or switch delay) it may be required, that all installed NIC adapters use the same NIC address. Therefor the registry entry “OvrIndexLock” = 1 is required:

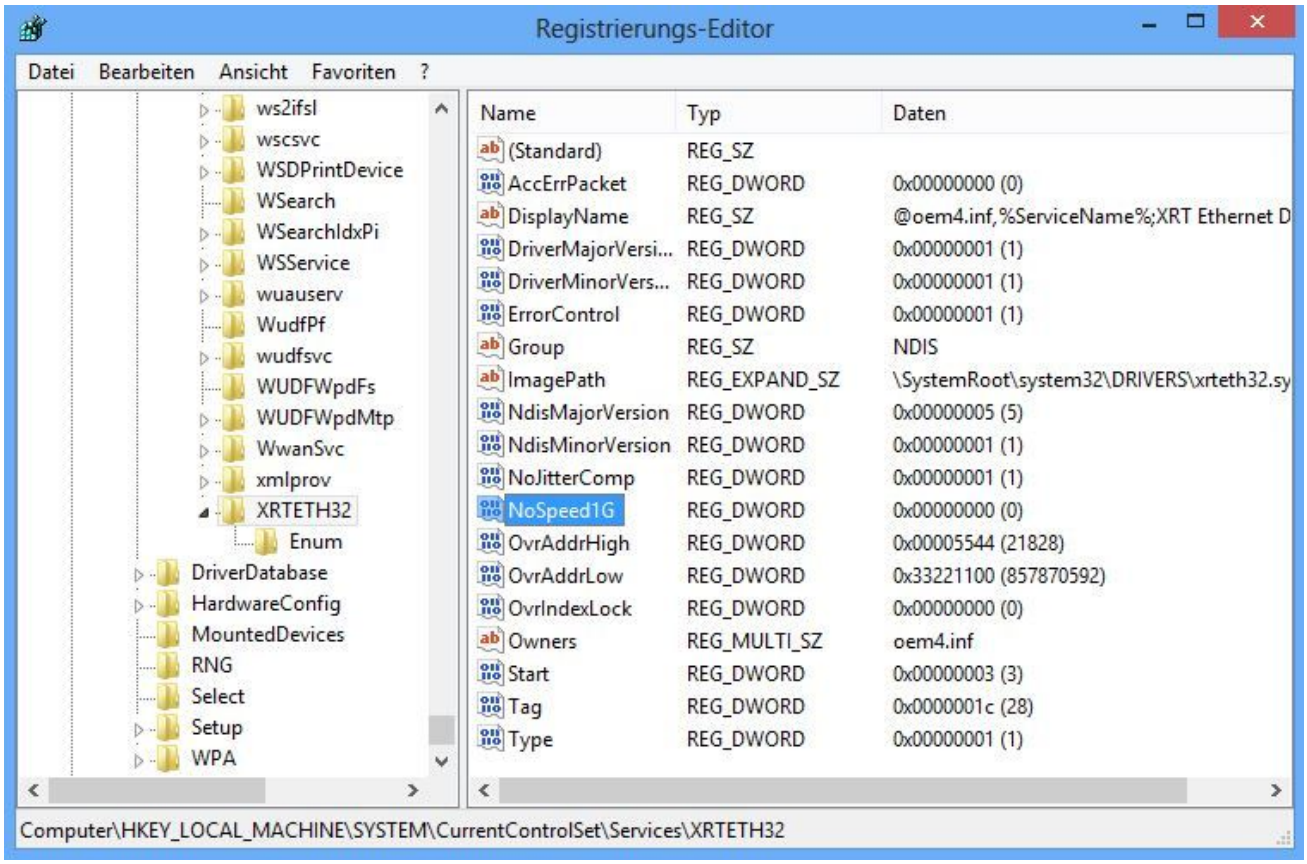
3.5 Dynamic Jitter Compensation

SYBERA uses the procedure "Dynamic Jitter Compensation" with active and passive feedback compensation within the realtime engine. Although the X-Real time engine of SYBERA allows a native maximum Jitter of approx. 15 µ sec (according to hardware platform), this behaviour may be reduced below 3 µsec by the dynamic jitter compensation.
For compatibility reason on some platforms it may be required to disable the dynamic jitter compensation. Therefore the registry entry “NoJitterComp” has to be set to 1:



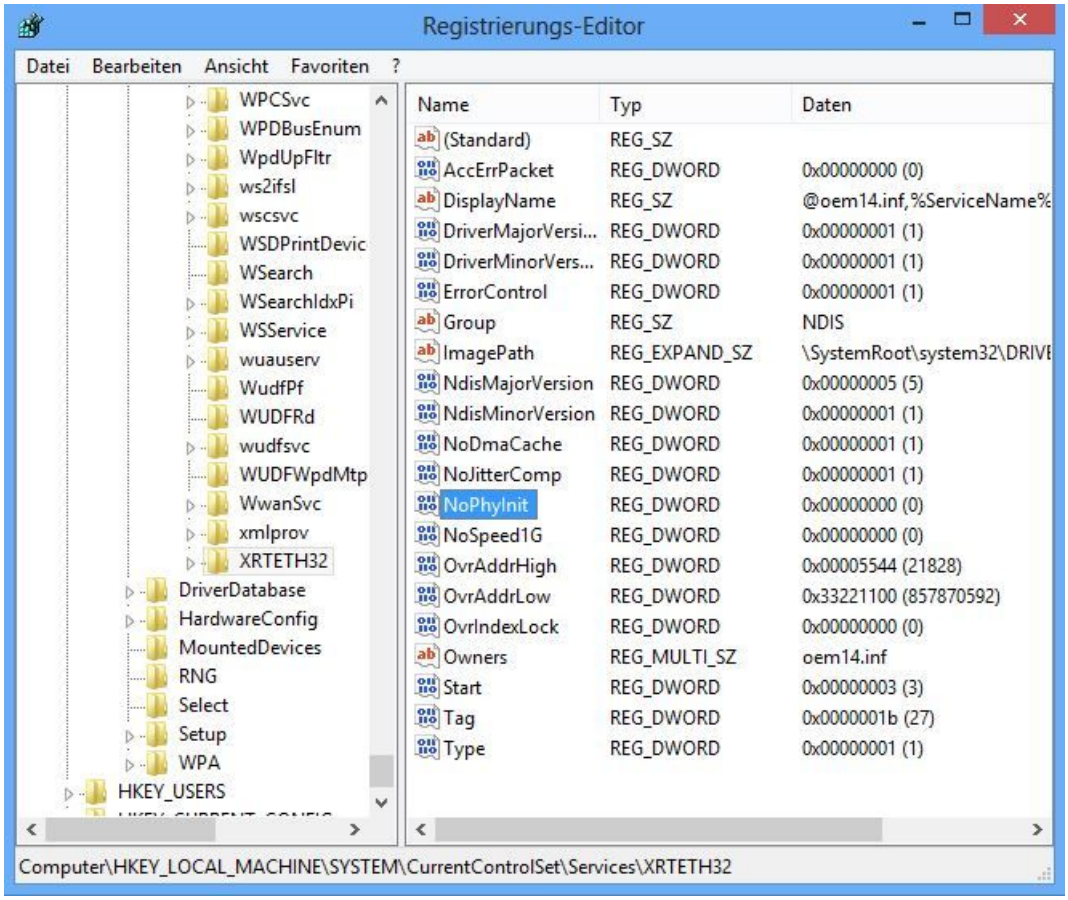
3.6 Speed Limitation

The Ethernet realtime core allows to reduce the frame speed below 1G by setting the registry entry “NoSpeed1G” to 1:



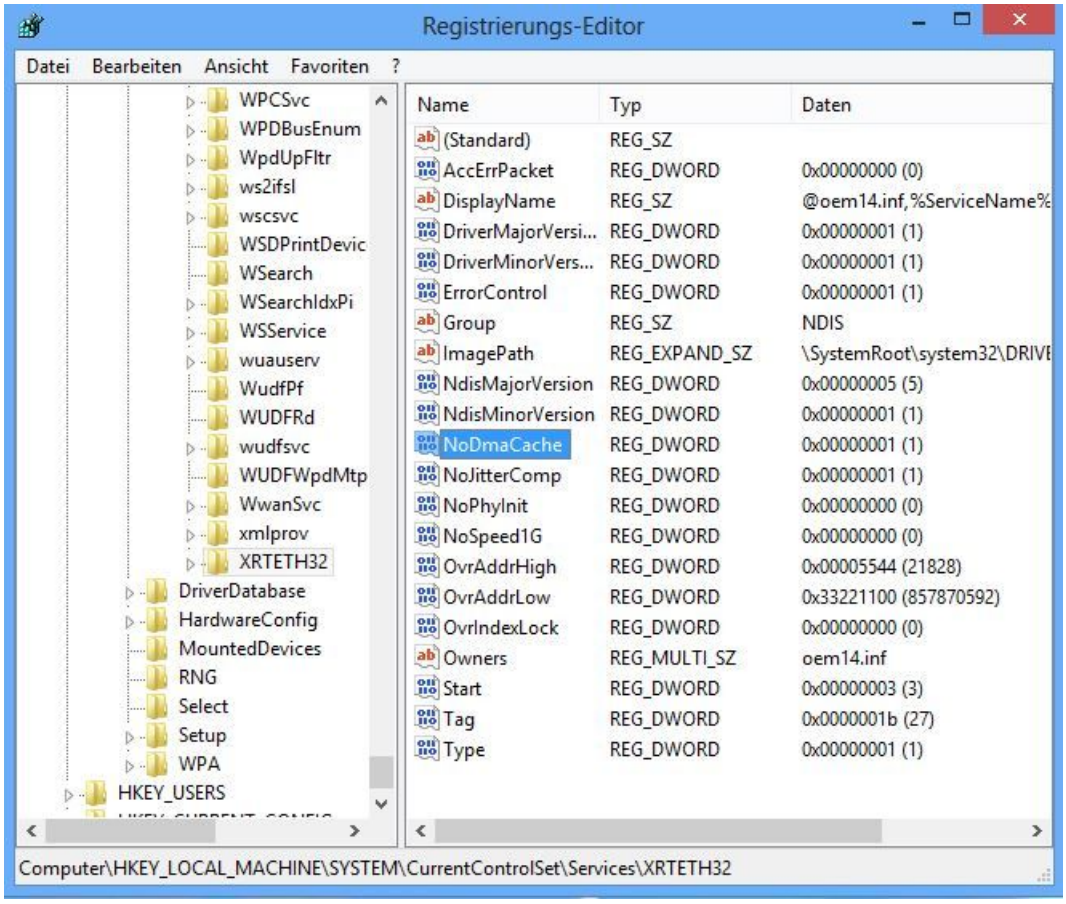
3.7 Dynamic PHY Initialization

Some ethernet adapters don't allow a dynamic speed negotiation. In this case set NoPhyInit to 0. After change the system needs to be rebooted.



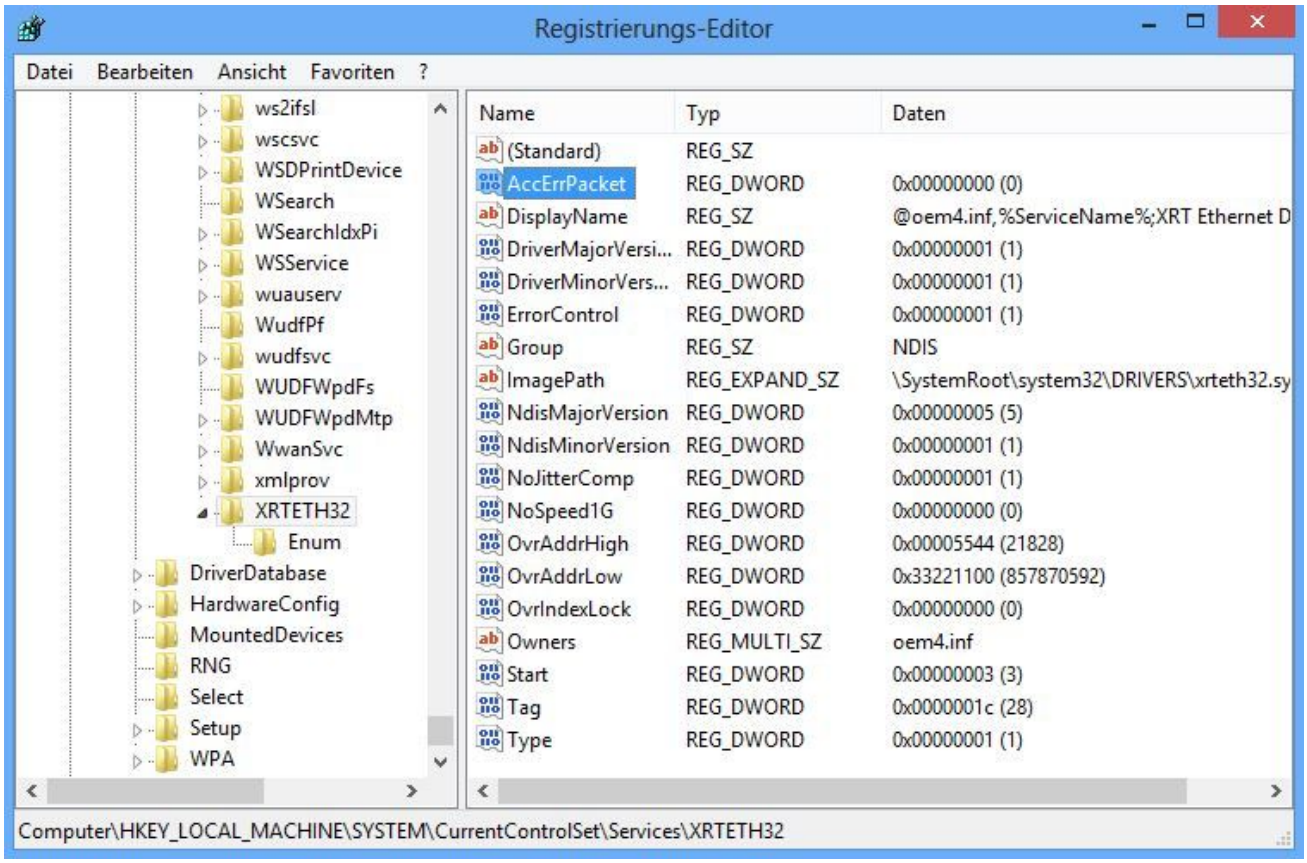
3.8 DMA memory cache

Some PCI buses haven`t implement a cache snoop mechanism. In this case set NoDmaCache to 0. If DMA memory cache has been disabled, the performance is reduced. After change the system needs to be rebooted.



3.9 Accepting Error Packet

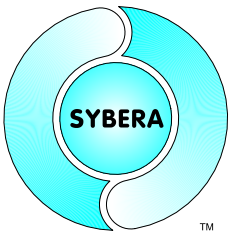
For diagnosic reasons it may be required to accept frame error packets. Therfore the registry entry “AccErrPacket” has to be set to 1:



4 Programming

4.1 Project files for VisualC++ and LabWindows

LIB\SHA64ETHCORE.LIB	Project Import Library
LIB\SHA64ETHCORE.DLL	Dynamic Link Library (copied to \WINDOWS\SYSTEM32)
INC\SHA64ETHCORE.H	Exported function prototypes
INC\ETH64MACROS.H	Exported ETH Macros
INC\ETH64COREDEF.H	ETH core definitions



4.2 Header File ETH64COREDEF.H

This header file defines all structures required for handling the core interface and realtime stack data. It also defines the ethernet protocols, like TCP, UDP, ARP, ...The structure elements are combined hierachically.

4.2.1 Structure ETH_PARAMS

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

```
typedef struct _ETH_PARAMS
{
      //Input parameters
      ULONG             dev_num;          //device number
      ULONG             period;           //Realtime scheduling period
      USHORT            eth_type;         //Ethernet filter type
      UCHAR             eth_if;           //Ethernet filter interface
      UCHAR             reserved;         //Reserved
      ETH_RX_FILTER     rx_filter;        //RX frame filter

      //Output parameters
      ULONG             time_stamp[2];    //RX Frame timestamp
      ULONG             core_drv_ver;     //Core driver version
      ULONG             sha_drv_ver;      //SHA driver version
      ULONG             sha_lib_ver;      //SHA library version
      ETH_ERR_CNTS      err_cnts;         //Error counters

      //Input - Output parameters
      ETH_FRAME         frame;            //ETH frame
      USHORT            frame_size;       //Frame size
      UCHAR             frame_phase;      //Frame phase
      UCHAR             clock_set;        //Clock setting (at TX)

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

      //Realtime level2 output parameters
      PETH_STACK        pSystemStack;     //ETH_STACK structure for realtime
                                          //application task
      PETH_STACK        pUserStack;       //ETH_STACK structure for windows
                                          //application task

} ETH_PARAMS, *PETH_PARAMS;
```

4.2.2 Structure ETH_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 _ETH_STACK
{
ETH_STACK_HDR     hdr;              //Stack header
ETH_TABLE         tx_table;         //TX ringbufer table
ETH_TABLE         rx_table;         //RX ringbufer table

} ETH_STACK, *PETH_STACK;


typedef struct _ETH_STACK_HDR
{
BOOLEAN  err_flag;                  //Frame Error Flag
BOOLEAN  run_flag;                  //Stack enabling Flag
BOOLEAN  lev2_flag;                 //Realtime Level2 Flag
BOOLEAN  internal_flag;             //Internal task flag
                                    //(only TX, RX and ERR task running)
USHORT   dev_num;                   //NIC device index
USHORT   cycle_corr;                //Cycle correction value
ULONG    tx_cnt;                    //TX counter
ULONG    rx_cnt;                    //RX counter
ULONG    phase_cnt;                 //Phase counter
ULONG    phase_num;                 //Number of phases
ULONG    phase_lock;                //Phase change lock
ULONG64  tx_timestamp_latched;      //Latched TX timestamp
ULONG64  rx_timestamp_latched;      //Latched RX timestamp

ULONG    clock_adj;         //NIC clock adjustment
                            //(set ETH_ENTRY->ClockSet |= ETH_CLOCK_SET_ADJ)

ULONG    clock_offs;        //NIC clock offset
                            //(set ETH_ENTRY->ClockSet |= ETH_CLOCK_SET_OFFS)

USHORT   clock_frame_offs; //Timestamp frame offset
                            //(set ETH_ENTRY->ClockSet |= ETH_CLOCK_SET_FRAME_OFFS)

USHORT   clock_time_ext;   //Timestamp extension
                            //(set ETH_ENTRY->ClockSet |= ETH_CLOCK_SET_TIME_EXT)

ULONG64 clock_target_time;//NIC TX target time
                            //(set ETH_ENTRY->ClockSet |= ETH_CLOCK_SET_TARGET_TIME)

ULONG64 pci_reg_va;        //PCI register virtual address

} ETH_STACK_HDR, *PETH_STACK_HDR;
```


```
typedef struct _ETH_TABLE
{
     UCHAR  buffer[ETH_BUFFER_SIZE];      //Ethernet frame buffer
     ULONG  findex;                       //Forward index
     ULONG  bindex;                       //Back Index


     struct
     {
      ETH_ENTRY entry;

     } list[MAX_ENTRIES];

} ETH_TABLE, *PETH_TABLE;



typedef struct _ETH_ENTRY
{
     PETH_FRAME   pFrame;           //Pointer to Frame
     USHORT       FrameSize;        //Frame Size
     BOOLEAN      bOccupied;        //Occupied flag
     BOOLEAN      bReserved[3];     //Reserved
     UCHAR        FramePhase;       //Frame Phase
     UCHAR        ClockSet;         //Clock Setting
     ULONG64      TimeStamp;        //Timestamp

} ETH_ENTRY, *PETH_ENTRY;
```


```
 //Ethernet 14-byte Header
 typedef     struct _ETH_HEADER
 {
      UCHAR target[ETH_ADDRESS_LEN];
      UCHAR source[ETH_ADDRESS_LEN];
      UCHAR type_length[2];

 } ETH_HEADER, *PETH_HEADER;


//Ethernet 18-byte VLAN Header
typedefstruct _ETH_VLAN_HEADER
{
      UCHAR target[ETH_ADDRESS_LEN];
      UCHAR source[ETH_ADDRESS_LEN];

      struct
      {
       ULONG type_high   : 8;
       ULONG type_low    : 8;
       ULONG priority    : 3;
       ULONG reserved    : 1;
       ULONG id          : 12;
      } tag;

      UCHAR type_length[2];

} ETH_VLAN_HEADER, *PETH_VLAN_HEADER;


 //Ethernet Buffer (Including Ethernet Header) for Transmits
 typedef union _ETH_FRAME
 {
   UCHAR bytes[MAX_ETH_PACKET_SIZE];

   //Ethernet frame with standard Header
   struct
   {
       ETH_HEADER  hdr;
       UCHAR       pProtocol[  MAX_ETH_PACKET_SIZE - sizeof(ETH_HEADER)];

 } s;

   //Ethernet frame with VLAN Header
   struct
   {
     ETH_VLAN_HEADER    hdr;
     UCHAR              pProtocol[MAX_ETH_PACKET_SIZE - sizeof(ETH_VLAN_HEADER)];

   } vlan;

 } ETH_FRAME, *PETH_FRAME;
```

All structures are fully accessible inside the real-time application task, as well as inside the windows application. The stack pointer is returned on the function Sha64EthCreate(), which includes all elements for accessing the TX table inside the real-time application task (the include file ETH64Macros.h provides additional macros for table handling)

Sample:

```
//Get current TX table entry
PETH_TABLE pTable = (PETH_TABLE)&__pStack->tx_table;
PULONG     bindex = (pTable->bindex == 0) ? (MAX_ETH_ENTRIES - 1)
                                          : (pTable->bindex - 1);

//Check for free TX entry
if (pNicTxEntry->bOccupied == FALSE)
{
      PETH_ENTRY pEntry = (PETH_ENTRY)&_pTable->list[bindex];

      //Save Information
      memcpy(pEntry->pFrame->s.pProtocol, ProtocolData, FRAME_SIZE);
      pNicTxEntry->FrameSize = FRAME_SIZE;

      //Update entry location and set handshake flage
      pTable->bindex = bindex;
}
```

4.3 Header File ETH64MACROS.H

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

This Macro is to start or stop realtime tasks:

```
ETH_TASK_CONTROL(__pStack, __bRun)

__pStack          Type: PETH_STACK             //Ethernet Stack
__bRun            Type: BOOLEAN                //RUN Flag
```

This Macro is to enable or disable realtime level2:

```
ETH_LEVEL2_CONTROL(__pStack, __bCond)

__pStack          Type: PETH_STACK             //Ethernet Stack
__bCond           Type: BOOLEAN                //Level2 Flag
```

This Macro is to control the TX and RX stack index:

```
ETH_STACK_CONTROL(__pStack, __TxIndex, __RxIndex)

__pStack          Type: PETH_STACK             //Ethernet Stack
__TxIndex         Type: ULONG                  //TX stack index
__RxIndex         Type: ULONG                  //RX stack index
```

This Macro is to check for a idle stack condition:

```
ETH_CHECK_STACK_IDLE(__pStack, __pbIdle)

__pStack          Type: PETH_STACK             //Ethernet Stack
__pbIdle          Type: PBOOLEAN               //Pointer to IDLE flag
```

This Macro is to get the TX entry pointer:

```
ETH_GET_TXENTRY_PTR(__pStack, __Index, __ppEntry)

__pStack          Type: PETH_STACK             //Ethernet Stack
__Index           Type: ULONG                  //Table Index
__ppEntry         Type: PENTRY*                //Pointer to Stack Entry
```

This Macro is to get the RX entry pointer:

```
ETH_GET_RXENTRY_PTR(__pStack, __ppEntry)

__pStack          Type: PETH_STACK             //Ethernet Stack
__Index           Type: ULONG                  //Table Index
__ppEntry         Type: PENTRY*                //Pointer to Stack Entry
```

This Macro is to get the current Level2 TX entry pointer, if Level2 is combined with Level1:

```
ETH_GET_CURRENT_ TXENTRY_PTR(__pStack, __Index, __ppEntry)

__pStack          Type: PETH_STACK             //Ethernet Stack
__ppEntry         Type: PENTRY*                //Pointer to Stack Entry
```

This Macro is to get the current Level2 RX entry pointer, if Level2 is combined with Level1:

```
ETH_GET_CURRENT_RXENTRY_PTR(__pStack, __ppEntry)

__pStack          Type: PETH_STACK             //Ethernet Stack
__ppEntry         Type: PENTRY*                //Pointer to Stack Entry
```

This Macro is to update the handshake flag of the current Level2 TX entry pointer, if Level2 is combined with Level1:

```
ETH_UPDATE_CURRENT_TXENTRY_PTR(__pStack)

__pStack          Type: PETH_STACK             //Ethernet Stack
```

This Macro is to update the handshake flag of the current Level2 RX entry pointer, if Level2 is combined with Level1:

```
ETH_UPDATE_CURRENT_TXENTRY_PTR(__pStack)

__pStack          Type: PETH_STACK             //Ethernet Stack
```

This Macro is to get the current Level2 TX Frame pointer, if Level2 is combined with Level1:

```
ETH_GET_CURRENT_USER_TXENTRY_PTR(__pUserStack, __pSystemStack, __ppFrame)

__pUserStack      Type: PETH_STACK             //User Ethernet Stack
__pSystemStack    Type: PETH_STACK             //System Ethernet Stack
__ppFrame         Type: PETH_FRAME*            //Pointer to Frame in Userspace
```

This Macro is to get the current Level2 RX Frame pointer, if Level2 is combined with Level1:

```
ETH_GET_CURRENT_USER_TXENTRY_PTR(__pUserStack, __pSystemStack, __ppFrame)

__pUserStack      Type: PETH_STACK             //User Ethernet Stack
__pSystemStack    Type: PETH_STACK             //System Ethernet Stack
__ppFrame         Type: PETH_FRAME*            //Pointer to Frame in Userspace
```

This Macro is to check the error condition:

```
ETH_CHECK_ERROR(__pStack, __pbError)

__pStack          Type: PETH_STACK             //Ethernet Stack
__pbError         Type: PBOOLEAN               //Pointer to ERROR flag
```

This Macro is to set the error condition:

```
ETH_SET_ERROR(__pStack)

__pStack          Type: PETH_STACK             //Ethernet Stack
```

This Macro to check the ethernet frame type:

```
ETH_CHECK_TYPE(__pFrame, __pType, __bVLan)

__pStack          Type: PETH_STACK             //Ethernet Stack
__pType           Type: PUCHAR                 //Pointer to Ethernet type
__bVLan           Type: PBOOLEAN               //VLAN flag (optional)
```

This Macro is to check the IP type:

```
ETH_CHECK_IP_TYPE(__pFrame, __pIpType)

__pStack          Type: PETH_STACK             //Ethernet Stack
__pIpType         Type: PUCHAR                 //Pointer to IP type
```

This Macro is to check the UDP type:

```
ETH_CHECK_UDP_TYPE(__pFrame, __pUdpSrcPort, __pUdpDstPort)

__pStack          Type: PETH_STACK             //Ethernet Stack
__pUdpSrcPort     Type: PUCHAR                 //Pointer to UDP Source Port
__pUdpDstPort     Type: PUCHAR                 //Pointer to UDP Destination Port
```

4.4 Ethernet Core Interface

The header file SHA64ETHCORE.H defines all required prototypes and parameters of the Ethernet Core Library. The header file is based on the file ETH64COREDEF.H. In the following all function prototypes will be discussed by samples.

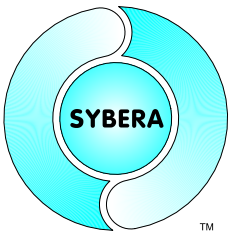
4.4.1 Sha64EthCreate

This function opens the Ethernet interface and installs the realtime ringbuffer system. On success the returning value is ERROR_SUCCESS, otherwise the returning value corresponds to that from GetLastError().

VC
```
ULONG Sha64EthCreate(PETH_PARAMS);
```

Sample:

```
//Reset parameters
memset(&EthParams, 0, sizeof(ETH_PARAMS));

//Required Ethernet CORE parameters
EthParams.period = 100;                  //Realtime scheduling period
EthParams.fpAppTask = NULL;              //No realtime level2 task
EthParams.eth_type = ETH_TYPE_IP;        //Set ethernet filter type
EthParams.eth_if = ETH_IF_SOCKET;        //Set ethernet filter interface

//Enable Ethernet CORE
Error = Sha64EthCreate(&EthParams);
if (ERROR_SUCCESS == Error)
{
      //Init global elements
      __pNic0SystemStack = ParamsNic0.pSystemStack;
      __pNic0UserStack   = ParamsNic0.pUserStack;

      //Set control elements
      ETH_LEVEL2_CONTROL(pNic0UserStack, FALSE);
      ETH_TASK_CONTROL(__pNic0UserStack, TRUE);
      …
}
```



4.4.1.1 Ethernet Core Filter

The Ethernet realtime core contains an Y-Multiplexer between acyclic frames (typically Win Socket or proprietary Interface) and cyclic frames (logic frame operation inside real-time task) with a Cross Filter for Ethernet frames. The elements *eth_type* and *eth_if* allow to define a filter condition for the selected interface. This means:

ETH_IF_SOCKET_RX	All RX frames of specified filter type will be directed to the socket interface
ETH_IF_SOCKET_TX	All TX frames of specified filter type will be directed to the socket interface
ETH_IF_CORE_RX	All RX frames of specified filter type will be directed to the core interface
ETH_IF_CORE_TX	All TX frames of specified filter type will be directed to the core interface

The ethernet interface filter is a bit mask, combining desired interface pathes. The following sample describes all frames (RX/TX) of specified filter type will be directed to the socket interface:

FilterIF = ETH_IF_SOCKET_RX | ETH_IF_SOCKET_TX

FilterType = 0	No pass through to the selected interface
FilterType = ETH_TYPE_XXX	Access only frames of type XXX for the selected interface. All other frames are directed to the other interface

FilterIF = 0 FilterType = 0 No filter is set. All frames will be directed to all interfaces


```
//Define interface types for ethernet filter
#define   ETH_IF_SOCKET_RX          0x10
#define   ETH_IF_SOCKET_TX          0x20
#define   ETH_IF_CORE_RX            0x40
#define   ETH_IF_CORE_TX            0x80

//Define data filter types
#define ETH_FILTER_NONE             0
#define ETH_FILTER_MATCH            1
#define ETH_FILTER_MISMATCH         2
```

Control Elements:

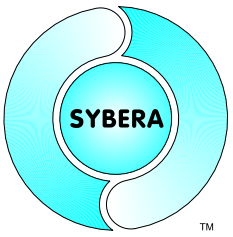
```
ETH_PARAMS.eth_type
ETH_PARAMS.eth_if
ETH_PARAMS.rx_filter
```

Implemented Interface Filter Logic (RX / TX):

```
if ((eth_type == internal_type) && ((internal_if & eth_if) == eth_if))   ||
    (eth_type != internal_type) && ((internal_if & eth_if) != eth_if)))
{
      /*Receive Frame from specified interface*/
}
```

Implemented Data Filter Logic (RX):

```
//Check filter type
if (rx_filter.Type == ETH_FILTER_NONE)
    return TRUE;

//Check filter data
for (i=0; i< rx_filter.Size; i++)
    if (pFrame->bytes[rx_filter.Offs + i] != rx_filter.Data[i])
        if (rx_filter.Type == ETH_FILTER_MATCH)
{
      /*Receive Frame from specified interface*/
}
```



4.4.2 Sha64EthDestroy

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

VC
```
ULONG Sha64EthDestroy(PETH_PARAMS);
```

4.4.3 Sha64EthGetVersion

This function retrieves the version information strings of the Ethernet Core Library, the Ethernet Core Driver, the SHA Dll, the SHA Library and the SHA Driver. The memory for the information strings must be allocated first.

VC
```
ULONG Sha64EthGetVersion (PETH_PARAMS);
```

Sample:

```
ETH_PARAMS EthParams;
char  szEthLibVer[MAX_PATH];
char  szEthDrvVer[MAX_PATH];
char  szShaLibVer[MAX_PATH];
char  szShaDrvVer[MAX_PATH];

//Get version information
Sha64EthGetVersion(&EthParams);

sprintf(m_szEthLibVer, "%.2f", EthParams.core_dll_ver / (double)100);
sprintf(m_szEthDrvVer, "%.2f", EthParams.core_drv_ver / (double)100);
sprintf(m_szShaLibVer, "%.2f", EthParams.sha_lib_ver / (double)100);
sprintf(m_szShaDrvVer, "%.2f", EthParams.sha_drv_ver / (double)100);

printf("CORE-DLL Version: %s\n
           CORE-DRV Version: %s\n
           SHA-LIB Version: %s\n
           SHA-DRV Version: %s\n",
           szCoreDllVersion,
           szCoreDrvVersion,
           szShaLibVersion,
           szShaDrvVersion);
```

4.4.4 Sha64EthResetTransmission

This function resets the ethernet core and releases all resources for communication.

VC
```
ULONG Sha64EthResetTransmission(PETH_PARAMS);
```

4.4.5 Sha64EthCheckStatus

This function gets the status of the ethernet core.

VC
```
ULONG Sha64EthCheckStatus(PETH_PARAMS);
```

4.4.6 Sha64EthTransmitFrame

This function transmits an available ethernet frame from top of stack

VC
```
ULONG Sha64EthTransmitFrame(PETH_PARAMS);
```

4.4.7 Sha64EthReceiveFrame

This function gets an available ethnet frame from top of stack

VC
```
BOOLEAN Sha64EthReceiveFrame(PETH_PARAMS);
```

4.4.8 Sha64EthSetMode

This function controls the stack behaviour

VC
```
BOOLEAN Sha64EthSetMode(PETH_PARAMS_MODE);
```

4.4.9 Sha64EthSetMode

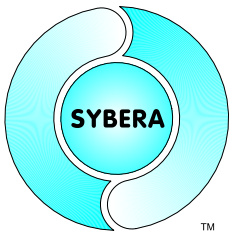
This function controls the time stamping of the I210 clock

VC
```
BOOLEAN Sha64EthSetClock(PETH_PARAMS_CLOCK);
```

Sample: Socket Cmmunication

```
#include <stdio.h>
#include <conio.h>
#include <windows.h>
#include "c:\eth\Sha64EthCore.h"
#include "c:\sha\shaexp.h"


void main(void)
{
	char szCoreDllVersion[MAX_PATH];
	char szCoreDrvVersion[MAX_PATH];
	char szShaLibVersion[MAX_PATH];
	char szShaDrvVersion[MAX_PATH];
	ETH_PARAMS EthParams;
	DWORD Error;
	UCHAR i = 0;
	UCHAR c = '\\';

	printf("\n*** Ethernet CORE Socket Test ***\n\n");

	//Reset parameters
	memset(&EthParams, 0, sizeof(ETH_PARAMS));

	//****************************************
	//*** Required Ethernet CORE parameters ***
	//****************************************
	EthParams.period = 100;			//Realtime scheduling period
	EthParams.fpAppTask = NULL;		//No realtime level2 task
	EthParams.eth_type = ETH_TYPE_IP;	//Set ethernet frame type
						//Send all other frames to core
						//interface//interface
	EthParams.eth_if = ETH_IF_SOCKET_RX | ETH_IF_SOCKET_TX;

	//Enable Ethernet CORE
	Error = Sha64EthCreate(&EthParams);
	if (ERROR_SUCCESS == Error)
	{
		//*****************************
		//*** Open socket interface ***
		//*****************************
		// Initialize Winsock
		WSADATA     wsaData;
		int         wsaError = WSAStartup(MAKEWORD(2,2), &wsaData);

		if (wsaError == NO_ERROR)
		{
			//Create a CLIENT SOCKET for UDP datagrams
			SOCKET ClientSocket = socket(AF_INET, SOCK_DGRAM,
							IPPROTO_UDP);
			if (ClientSocket != INVALID_SOCKET)
			{
				//Get IP address of the server by name
				hostent* pSrvAddr;
				pSrvAddr = gethostbyname("mypc");
				if (pSrvAddr)
```


```
                {
                      //Print server IP address
                      PUCHAR pAddr = (PUCHAR)pSrvAddr->h_addr;
                      printf("Server Addr: %i.%i.%i.%i\n",
                            pAddr[0], pAddr[1], pAddr[2], pAddr[3]);

                      sockaddr_in ServerAddr;
                      sockaddr_in ClientAddr;
                      int ServerAddrLen = sizeof(ServerAddr);

                      //IP Address of the server
                      ServerAddr.sin_family = AF_INET;
                      ServerAddr.sin_addr =
                                  *((struct in_addr*)pSrvAddr->h_addr);
//                    ServerAddr.sin_addr.s_addr =
                                  inet_addr("192.168.1.78");
                      ServerAddr.sin_port = htons(0x8892);

                      //IP Address of this station
                      ClientAddr.sin_family = AF_INET;
                      ClientAddr.sin_addr.s_addr =
                                  inet_addr("192.168.1.22");
//                    ClientAddr.sin_addr.s_addr =
                                  inet_addr("192.168.1.23");
                      ClientAddr.sin_port = htons(0x8892);

                      //Bind the socket to the NIC adabter
                      if (SOCKET_ERROR != bind(
                                        ClientSocket,
                                        (SOCKADDR*)&ClientAddr,
                                        sizeof(ClientAddr)))
                      {
                            //Declare and initialize variables
                            int LoopCnt = 0;
                            int BytesSnt;
                            int BytesRcv;
                            int RcvBufferSize = 512;
                            char SndBuffer[512];
                            char RcvBuffer[512];

                            printf("Press any key ...\n");
                            while (!kbhit())
                            {
                                  //Setup send buffer
                                  sprintf(SndBuffer,
                                  "Client Datagram %i", LoopCnt++);

                                  //Send UDP data
                                  BytesSnt = sendto(
                                        ClientSocket,
                                        SndBuffer,
                                        strlen(SndBuffer),
                                        0,
                                        (struct sockaddr*)&ServerAddr,
                                        sizeof(ServerAddr));

                                  //Do some delay
                                  Sleep(100);
```




```
                                    //Receive UDP data
                                    BytesRcv = recvfrom(
                                          ClientSocket,
                                          RcvBuffer,
                                          RcvBufferSize,
                                          0,
                                          (struct sockaddr*)&ServerAddr,
                                          &ServerAddrLen);

                                    if (BytesRcv)
                                    {
                                          //Print data
                                          printf("RCV: \n");
                              for (int i=0; i<20; i++)
                              { printf("%c",   RcvBuffer[i]); }
                              printf("\n");
                              for (int j=0; j<20; j++) { printf("%02x",
                                          RcvBuffer[j]); }  printf("\n");
                                    }
                                 }
                              }
                        }
                        //Close the socket
                        closesocket(ClientSocket);
                  }
                  //Clean up WSA
                  wsaError = WSAGetLastError();
                  WSACleanup();
            }
            //Cleanup ethernet core
            Sha64EthDestroy(&EthParams);
      }

      printf("Press any key ...\n");
      while (!kbhit()) { Sleep(100); }
}
```

Sample:

Proprietary interface to send frame at precizce target time including the timestamp within frame:

```
ETH_PARAMS TxParams = { 0 }

//Copy frame data
memcpy(TxParams.frame.bytes, FRAME_DATA, FRAME_SIZE);
TxParams. FrameSize = FRAME_SIZE;

//Preset stack elements
pStack->clock_target_time = TargetTimeNsec
pStack->clock_frame_offs  = CLOCK_FRAME_OFFSET;
pStack->clock_time_ext    = CLOCK_TIME_EXT;

//Set clock for TX time stamping
TxParams.clock_set = ETH_CLOCK_SET_MODE_1STEP |
                     ETH_CLOCK_SET_FRAME_OFFS |
                     ETH_CLOCK_SET_TIME_EXT   |
                     ETH_CLOCK_SET_TARGET_TIME;

Sha64EthTransmitFrame(&TxParams);
```

5 Setting the Ethernet-Mode

The ethernet core can be operated in different ways with the help of parameters (similar to the registry parameters) and thus adapt the stack to the respective fieldbus system optimally. The task sequence, task synchronization, jitter compensation and PHY control can be individually adapted to the DMA operation of the Ethernet adapter. The following parameters are given tot he function Sha64EthSetMode.

Prototype: **ULONG Sha64EthSetMode(&ModeParams);**

DevNum Index of the realtime NIC adapter tob e used (typically 0 when using just 1 port)

OvrAddrLow MAC Override Address (lower 4 Bytes of new MAC address)
OvrAddrHigh MAC Override Address (upper 2 Bytes of new MAC address). The MAC address of the adapter will be overrided.
The default override address is 00-11-22-33-44-55

OvrIndexLock MAC Override Address Auto-Icrement Lock
Each ehternet port instance increments the override address by one.
If this parameter is set, the base override address will not be incremented.

AccErrPacket Accept error packets on receiving

NoSpeed1G No 1GBit speed negotiation

NoPhyInit No PHY initialization on ethernet core start

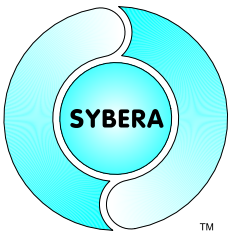
NoPhyReinit Only one time PHY initialization on stack startup

NoJitterComp No Jitter compensation of first task
If this parameter is set to 0, a incremental jitter compensation is done.
This will increase the accuracy, be decrease the bandwith by 20% of a cycle

NoCycleCorr No dynamic cycle correction
If this parameter is set to 0, the period may be adjusted directly within the realtime Task. Therefor the stack parameter cycle_corr needs to be set.

NoDmaCache : No DMA caching

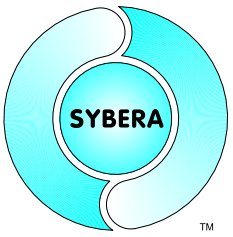
NoSocket No socket interface communication, even if the fuilter to the socket is open. (see also ethernet core filter management)



NoSerialize	Wait for task completion (Number of TX frames must be equal to RX frames). If this parameter is set, the RX task waits until the number of RX frames is Equal to the number of TX frames, or The GatherWait time expires. This parameter has to be used in combination with GahterWait.
TaskOrder	Priority value of internal core tasks (TX, RX, ERR) and the APP task. TX (Byte0) - RX (Byte1) - ERR (Byte2) - APP (Byte3)
FlushTransmit	: Sending all TX frames on the stack within one cycle
SwapTransmit	Changing the TX priority (acyclic frames first) Setting this flag send the frames of the proprietary interface first, before sending the frames of the realtime task
RetryTX	Retry Loop Time for TX Task to transmit a single frame This parameter sets the retry time [usec] for sending a single TX frames
RetryRX	Retry Loop Time for RX Task to receive a single frame This parameter sets the retry time [usec] for receiving a single RX frames
PhaseNum	Dividing a realtime Cycle into phases This parameter divides a core cycle into phases. This means, that the given period is multiplied by the phase num. This allows each frame to be assigned to a specified phase. There for the TX entry has the element FramePhase, which assignes the phase the frame has to be sent (GatherWait is only valid for the first phase of the cycle).
GatherWait	Wait time to collect RX frames If this parameter is given, it defines the wait time [usec] for collecting RX frames within one cycle. This is helpful on waiting for protocol responses, but reduces the bandwith.
MixedIfCorr	Correction of a possible race condition when using realtime task and proprietary interface together (acyclic and cyclic communication together). This flag is useful, when not using cycle phases, but reduces the bandwith.
DbgControl	Debug Sequencing Control (see realtime sequencing)
DbgFilter	Debug Sequencing Filter (see realtime sequencing)

Sample:

```
ETH_PARAMS_MODE ModeParams = { 0 };
ModeParams.mode.dev_num       = DevNum;
ModeParams.mode.NoCycleCorr   = bDc ? 0 : 1;
ModeParams.mode.NoJitterComp  = 1;
ModeParams.mode.NoSocket      = 0;
ModeParams.mode.NoSerialize   = 1;
ModeParams.mode.MixedIfCorr   = 1;
ModeParams.mode.FlushTransmit = 1;
ModeParams.mode.SwapTransmit  = 0;
ModeParams.mode.TaskOrder     = 0x0F030201;
ModeParams.mode.TxRetry       = Period / 10;
ModeParams.mode.RxRetry       = Period / 10;
ModeParams.mode.PhaseNum      = 0;
ModeParams.mode.AccErrPacket  = 0;
ModeParams.mode.NoSpeed1G     = 1;
ModeParams.mode.NoPhyInit     = 0;
ModeParams.mode.NoPhyReinit   = 0;
ModeParams.mode.NoDmaCache    = 0;
ModeParams.mode.GatherWait    = 0;

//Set ethernet mode parameters
ULONG Result = Sha64EthSetMode(&ModeParams);
```



5.1 TaskOrder

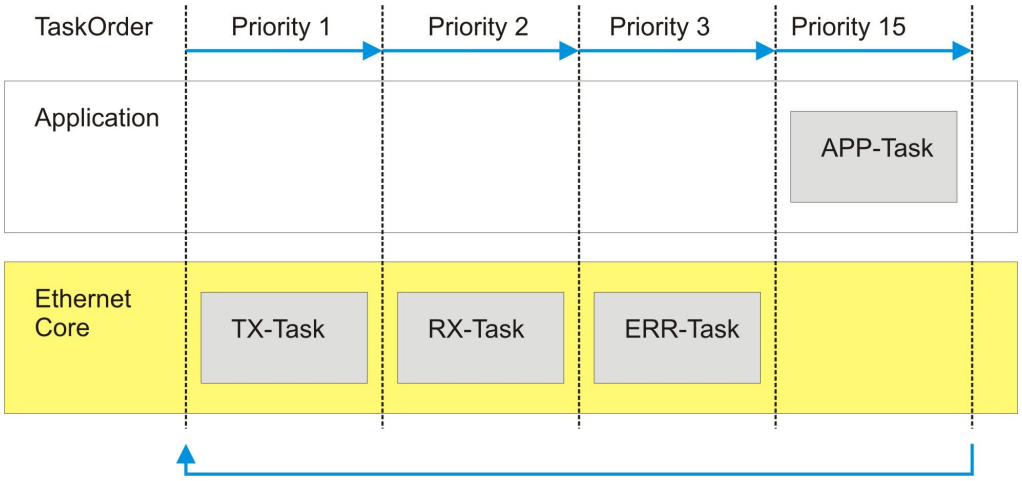
The ethernet core is based on 4 realtime tasks, TX task (sending frames), RX task (receiving frames), ERR task (ethernet error handling) and APP task (data handling). The TaskOrder Parameter determines the sequence of the task execution. E.g. a TaskOrder Parameter with the value 0x0F030201 means following

TX Task : Priority 0x01 (Byte 0 of TaskOrder)
RX Task : Priority 0x02 (Byte 1 of TaskOrder)
ERR Task : Priority 0x03 (Byte 2 of TaskOrder)
APP Task : Priority 0x0F (Byte 3 of TaskOrder)

Since the first task has highest accuracy (lowest jitter) at execution, it's important to select this task with the most influence on the fielbus behavior at first. Also the jitter of the first task is being compensated, due to the registry parameter *NoJitterComp* (compensation activated by value = 0).

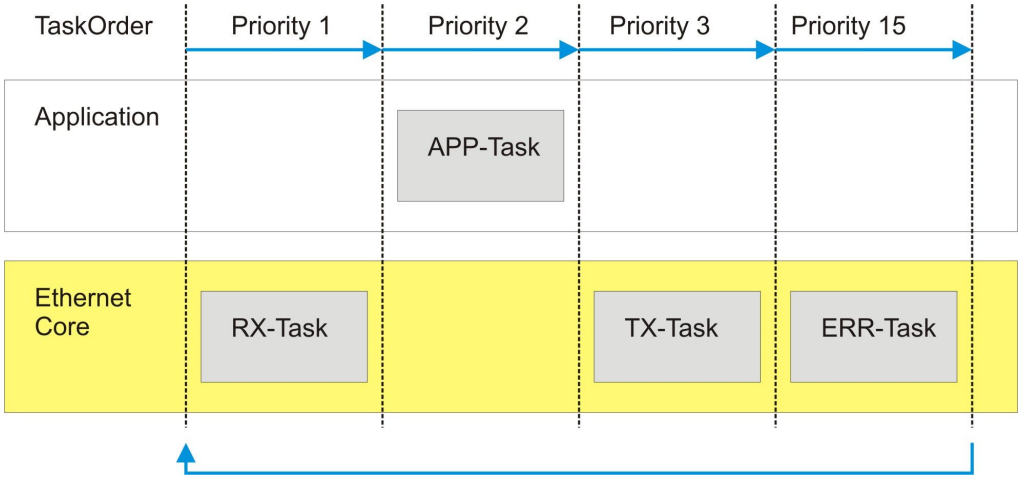
For master operation, the TX task has the most influence, since this task is controlling the bus. In that case the TX task must be first in the task Order.

TaskOrder (0F030201) for master operation : TX -> RX -> ERR -> APP



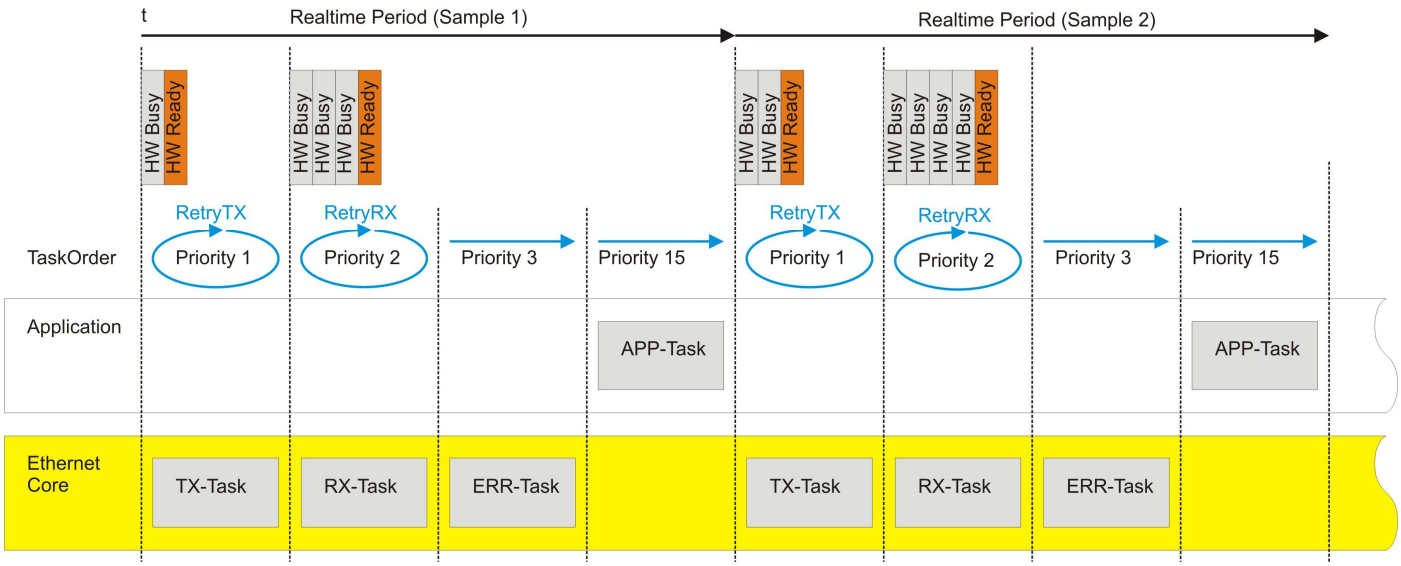
For slave operation, the RX task has the most influence on the fieldbus behavior. Therefore RX task should be the first task.

TaskOrder (020F0103) for slave operation : RX -> APP -> TX -> ERR



5.2 Task Retry

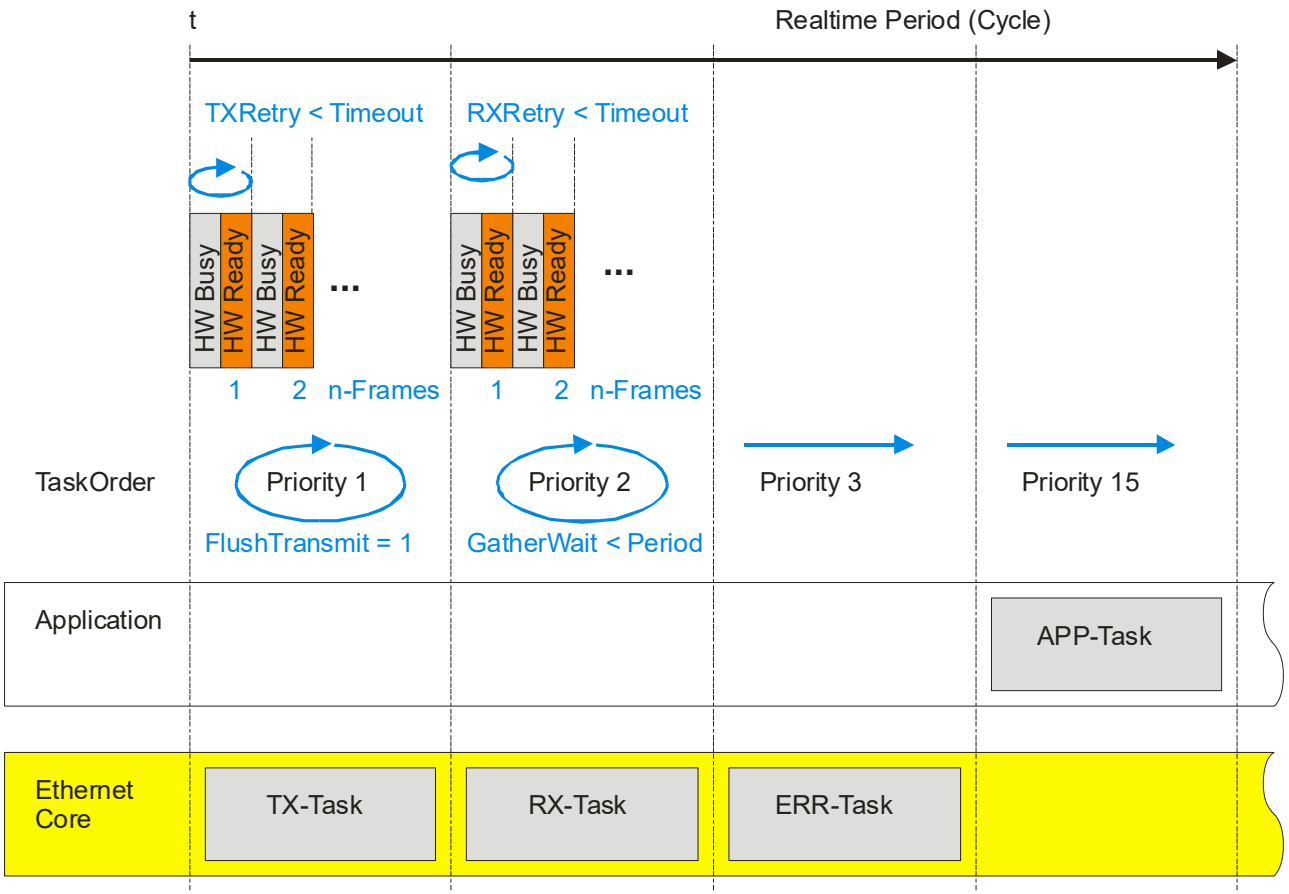
The retry parameters (time in [usec] attempt to allow reception or transmission of a frame within the sample once again in the Adapter-Busy state, which in turn leads to a higher bus load. The task retry allows a finer graduation of the realtime sampling operation. If the ethernet adapter is busy at sampling time, the retry time allows to proceed the ethernet data within the same realtime period. Unfortunately the retry time reduces the bandwith of the realtime period. Thus, when working with retries, the realtime period should be higher with less realtime samples.



Typical Values:

RetryTX = Realtime Period / 10
RetryRX = Realtime Period / 10

5.3 Serialize, FlushTransmit and GatherWait

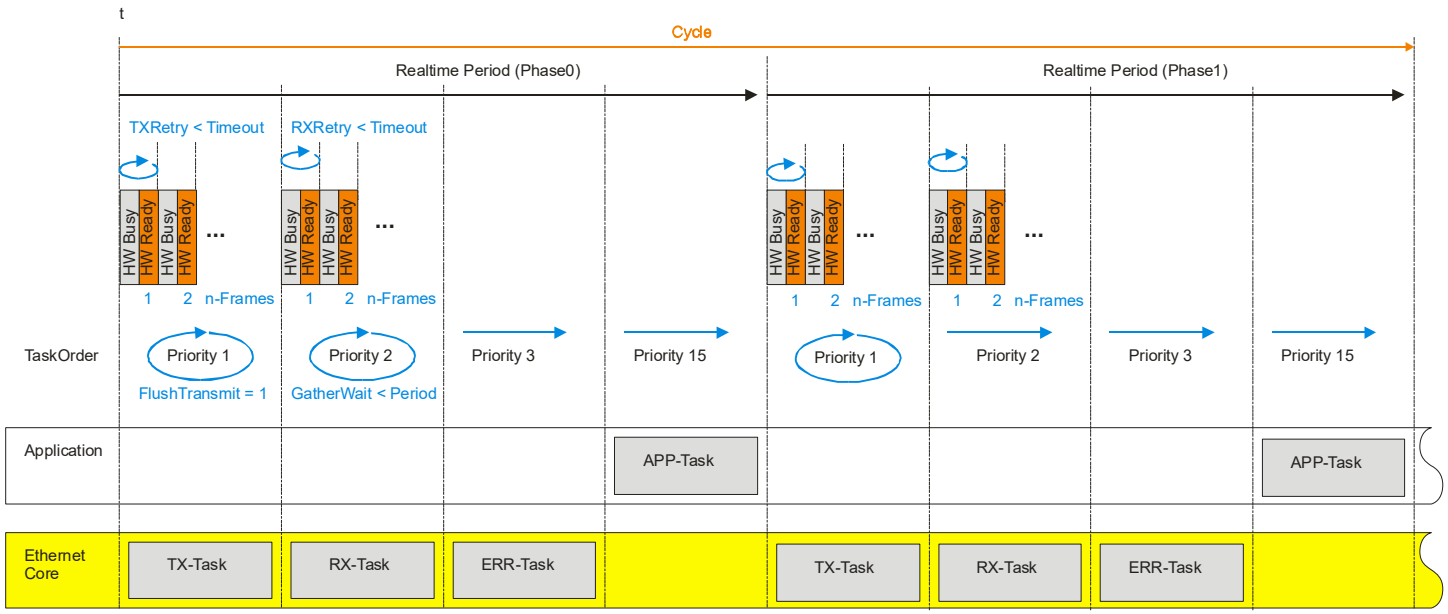


NoSerialize = 0 : (n-RX == n-TX) ? (RxTime < GatherWait) : Time = GatherWait
NoSerialize = 1 : RxTime = GatherWait

Typical Values:

RetryTX = Realtime Period / 10
RetryRX = Realtime Period / 10
GatherWait < Realtime Period / 2
FlushTransmit = 1

5.4 Phase Management

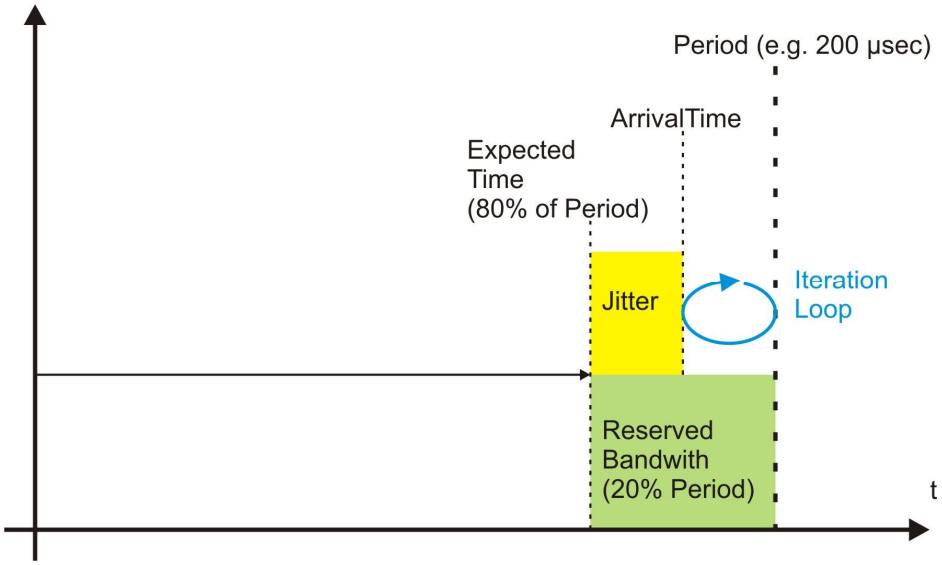


NoSerialize = 0 : (n-RX == n-TX) ? (RxTime < GatherWait) : Time = GatherWait
NoSerialize = 1 : RxTime = GatherWait
GatherWait is valid only in Phase0

5.5 Jitter Compensation

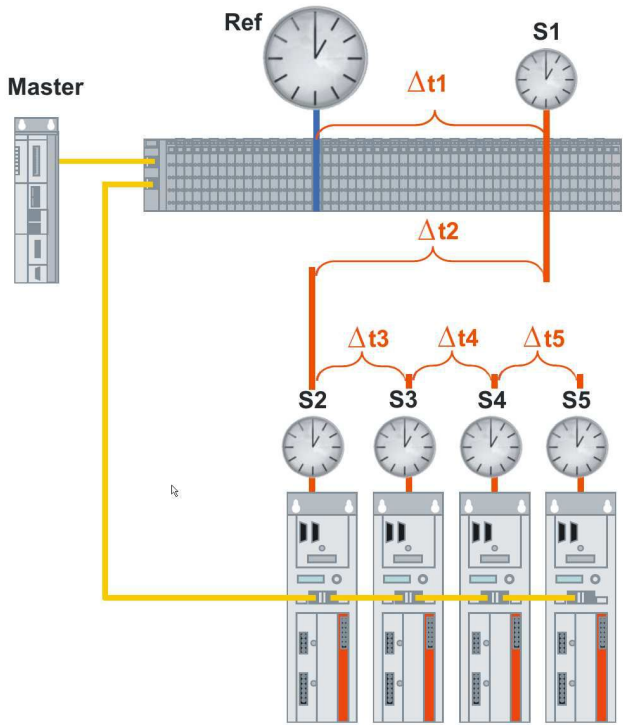
The first task of the task sequence may be compensated by a dynamic approachment to the correct period time. This is be done by reducing the bandwith by 20% for the dynamic iteration. The jitter compensation is enabled by the following registry parameter in ETH:

NoJitterComp = 0



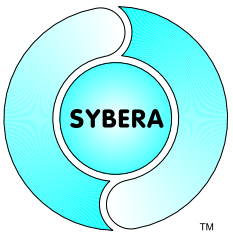
5.6 Cycle Correction

Some fieldbus systems require dynamic tracking of the MasterClock depending on an external clock (reference clock). The period of the master clock is dynamically adjusted here. With active cycle correction (NoCycleCorr = 0), the bandwidth is therefore reduced by approx. 20%. This method is usually only used with synchronous motion control.

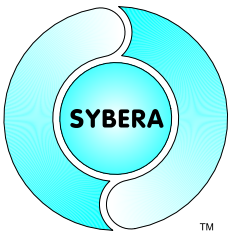


Implemented Stack Logic:

```
__inline BOOLEAN __SelectNextEntry(PDEVICE_OBJECT pDevice)
{
      PDEV_INFO pDevInfo  = (PDEV_INFO)pDevice->DeviceExtension;
      PETH_STACK pStack   = (PETH_STACK)pDevInfo->pStackVA;
      PETH_STACK_HDR pHdr = (PETH_STACK_HDR)&pStack->hdr;
      PETH_TABLE pTable   = (PETH_TABLE)&pStack->tx_table;
      PULONG pBIndex      = (PULONG)&pTable->bindex;
      PULONG pFIndex      = (PULONG)&pTable->findex;
      PETH_ENTRY pEntry;
      int i;

      //Loop through all entries
      for (i=0; i<MAX_ETH_ENTRIES; i++)
      {
            //Check SWAPTRANSMIT flag and get entry
            if (pDevInfo->SwapTransmit)
                  { pEntry = (PETH_ENTRY)&pTable->list[*pFIndex]; }
            else  { pEntry = (PETH_ENTRY)&pTable->list[*pBIndex]; }

            //Check entry
            if (pEntry->bOccupied  == TRUE)
            if (pEntry->FramePhase == pDevInfo->PhaseCnt)
                  return TRUE;

            //Decrease/decrease index
            if (pDevInfo->SwapTransmit)
                  { (*pFIndex) = (--(*pFIndex)) % MAX_ETH_ENTRIES; }
            else  { (*pBIndex) = (++(*pBIndex)) % MAX_ETH_ENTRIES; }
      }

      //No entry found
      return FALSE;
}


VOID static TxHandler(PDEVICE_OBJECT pDevice)
{
      PDEV_INFO pDevInfo  = (PDEV_INFO)pDevice->DeviceExtension;

      //Reset serialize counter
      pDevInfo->SerializeCnt = 0;

      //Check next TX entry
      while (__SelectNextEntry(pDevice))
      {
            //Transmit single frame
            if (__TxHandler(pDevice)) { pDevInfo->SerializeCnt++; }

            //Check FLUSHTRANSMIT flag
            if (pDevInfo->FlushTransmit == 0)
                  return;
      }
}
```




```
VOID static RxHandler(PDEVICE_OBJECT pDevice)
{
      PDEV_INFO  pDevInfo = (PDEV_INFO)pDevice->DeviceExtension;
      ULONG64    WaitTsc  = __rdtsc() +
                            __UsecToTsc(pDevInfo->GatherWait, pDevInfo->CpuFreq);

      //Receive loop
      do
      {
            //Receive a single frame
            if (__RxHandler(pDevice)) { pDevInfo->SerializeCnt--; }

            //Check serialize flag and count
            if (pDevInfo->NoSerialize  == 0)
            if (pDevInfo->SerializeCnt == 0)
                  return;

            //No wait loop on active phase
            if (pDevInfo->PhaseNum)
            if (pDevInfo->PhaseCnt)
                  return;
      }
      while (__rdtsc() < WaitTsc);
}
```



6 Intel I210 Clock - Time Stamping

The Realtime Ethernet Core offers the ability to make use oft he internal clock oft he Intel I210 ethernet chip. This allows a high precision protocol management. The ehternet core offers a lot of facilities to control the frame handlich by the internal clock management.

Supported Modes:

TX 1STEP Mode (timestamp is placed within frame, controlled by offset)
TX/RX 2STEP Mode (timestamp is latched into registers on sending/receiving frames)

```
//Define clock modes
#define CLOCK_MODE_1STEP                1
#define CLOCK_MODE_2STEP                2
```

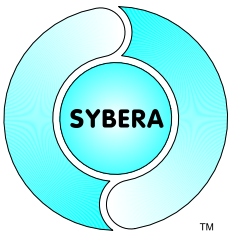
The timestamp handling is available fort he poprietary core interface, as well as for the realtime frame handling.

Control Element (ETH_ENTRY):

```
ETH_ENTRY.ClockSet
```

ClockSet in combination is a combination of following flags, on setting the corresponding stack elements:

```
ETH_CLOCK_SET_MODE_1STEP       Set 1Step Clock Mode
ETH_CLOCK_SET_MODE_2STEP       Set 2Step Clock Mode
ETH_CLOCK_SET_ADJ              Set Clock Adjustment
ETH_CLOCK_SET_OFFS             Set Clock Offset
ETH_CLOCK_SET_FRAME_OFFS       Set Frame Offset for TimeStamp
ETH_CLOCK_SET_TIME_EXT         Set Time Extension (2Bytes)
ETH_CLOCK_SET_LATCH            Latches the TimeStamp
ETH_CLOCK_SET_TARGET_TIME      Set TX Target Time
```



Stack Elements (ETH_STACK_HDR)

```
ULONG64   tx_timestamp_latched:    //Latched TX timestamp          (read)
ULONG64   rx_timestamp_latched;    //Latched RX timestamp          (read)
ULONG     clock_adj;               //NIC clock adjustment          (write/read)
ULONG     clock_offs;              //NIC clock offset              (write/read)
USHORT    clock_frame_offs;        //Timestamp frame offset        (write/read)
USHORT    clock_time_ext;          //Timestamp extension           (write/read)
ULONG64   clock_target_time;       //NIC TX target time            (write/read)
ULONG64   pci_reg_va;              //PCI register virtual address  (read)
```

Sample:
TX realtime task to send frame at precize target time including the timestamp within frame:

```
//Get current TX entry
PETH_ENTRY pEntry = NULL;
ETH_GET_CURRENT_TXENTRY_PTR(pStack, &pEntry);

//Check if entry is free
if (pEntry->bOccupied == FALSE)
{
      //Preset stack elements
      pStack->clock_target_time = TargetTimeNsec
      pStack->clock_frame_offs  = CLOCK_FRAME_OFFSET;
      pStack->clock_time_ext    = CLOCK_TIME_EXT;

      //Copy frame data
      memcpy(pEntry->pFrame->bytes, FRAME_DATA, FRAME_SIZE);
      pEntry->FrameSize = FRAME_SIZE;
      pEntry.ClockSet = ETH_CLOCK_SET_MODE_1STEP |
                        ETH_CLOCK_SET_FRAME_OFFS |
                        ETH_CLOCK_SET_TIME_EXT   |
                        ETH_CLOCK_SET_TARGET_TIME;
      pEntry.bOccupied = TRUE;
}
```