Programming reference guide

cifX/netX Application Programmer's Guide

Hilscher Gesellschaft für Systemautomation mbH

www.hilscher.com

DOC130709PRG01EN | Revision 1 | English | 2018-08 | Released | Public

Table of contents

1 Introduction

1.1 About this document

This guide is an introduction to the programming of Hilscher netX-based device. It explains the fundamentals necessary to understand the functions and the handling of Hilscher netX hardware and will show the programming principals needed for correct device handling.

It should also help to choose the level where to start with developments, describes where to find the necessary information.

Furthermore it explains the available information in regards to the programming level, describes necessary synchronization mechanism for the data transfer between a host and the netX firmware and covers all aspects from a simple memory connection to a netX (dual-port-memory access), the use of pre-created low level hardware functions, the netX C toolkit, the implementation of own drivers based the toolkit until the use of Hilscher device drivers for different operating systems.

1.2 List of revisions

Rev	Date	Name	Chapter	Revision
1	2018-08-30	RMA	all	Document created.

Table 1: List of revisions

1.3 Terms, abbreviations and definitions

Term	Description
cifX	**C**ommunication **I**nter**f**ace based on netX
comX	**Co**mmunication **M**odule based on netX
PCI	**P**eripheral **C**omponent **I**nterconnect
DPM	**D**ual-**P**ort **M**emory Physical interface to all communication board (DPM is also used for PROFIBUS-**DP M**aster).
API	**A**pplication **P**rogramming **I**nterface

Table 2: Terms, abbreviations and definitions

2 Programming resources

Hilscher offers several resources for the programming. The following list contains the most important manuals necessary.

Hilscher Website: https:\\www.hilscher.com

2.1 Manuals and documents

- **netX Dual-Port Memory Interface DPM xx EN.pdf**
  netX Dual-Port Memory packet-based services API xx EN.pdf
  netX Dual-Port Memory - Programming reference guide PRG xx EN.pdf

  These manuals describe the memory layout of the dual-port memory (DPM) and explain the information which are located in the DPM. The dual-port memory is the communication basis for a host to the netX device.

- **cifX netX Toolkit - DPM TK xx EN.pdf**

  The toolkit manual describes the C toolkit, the usage, implementation and the adaptation to own target systems.

- **cifX API PR xx EN.pdf**

  The CIFX API manual describes the Hilscher standard application programming interface. This API offers all necessary functions and information needed to handle netX-based device.

  With the knowledge from the DPM layout, a correlation between the information in the DPM and the API functions can be seen (e.g. what is a channel and how to exchange asynchronous command and cyclic I/O data).

- **cifX Device Driver - <operating system> DRV xx EN.pdf**

  Each operating system specific Hilscher device driver comes with a separate manual that describes the specific information and behavior.

- **PC Cards CIFX <card type> UM xx EN.pdf** and
  PC Cards cifX Software Installation UM xx EN.pdf

  Hilscher offers netX devices in different form factors (PC boards, PCI104, Compact PCI, etc.) for the most common hardware connections (DPM, ISA, PC104/PC104+, PCI/PCIe, serial interface, USB, Ethernet).

  Each device (card type) comes with a specific hardware installation, operation, and description manual, including technical data, connector details, jumper settings, LED state definitions etc. And a software installation and configuration guide.

2.2 Product CDs/DVDs and content

- **Communication Solutions DVD**

  Device Driver CD for Windows (NXDRV-WIN)

  Driver Toolkit CD (NXDRV-TKIT)

- **Device Driver CDs** (NXDRV-WIN)
  (NXDRV-CE)
  (NXDRV-Linux)
  (NXDRV-QNX)
  (NXDRV-VxWorks)
  (NXDRV-INtime)
  (NXDRV-RTX)

  Driver documentation and installation (operating system dependent)

  C development resources (header and libraries) (operating system dependent)

  Sources and Examples (operating system dependent)

  Additional manual: *CIFX API* manual

  Additional manual: *netX Dual-Port Memory Interface DPM* manual

- **cifX netX Toolkit CD** (NXDRV-TKIT)

  Toolkit documentation and source code

  Toolkit example implementation (Win32 / nonOS / MQX / rcX)

  Toolkit hardware functions example implementation (Win32 / WinCE / nonOS)

  Additional manual: *CIFX API* manual

  Additional manual: *netX Dual-Port Memory Interface DPM* manual

  Additional manual: *Serial DPM Interface with netX* manual

  Additional manual: *SPI Slave DPM netX 100 500 HAL* manual

  Additional manual: *Second Stage Bootloader netX* manual

2.3 Choose a programming level

Hilscher offers support to different programming levels. Programming levels are starting at the plain dual-port memory until up to the device driver level for the most common operating systems.

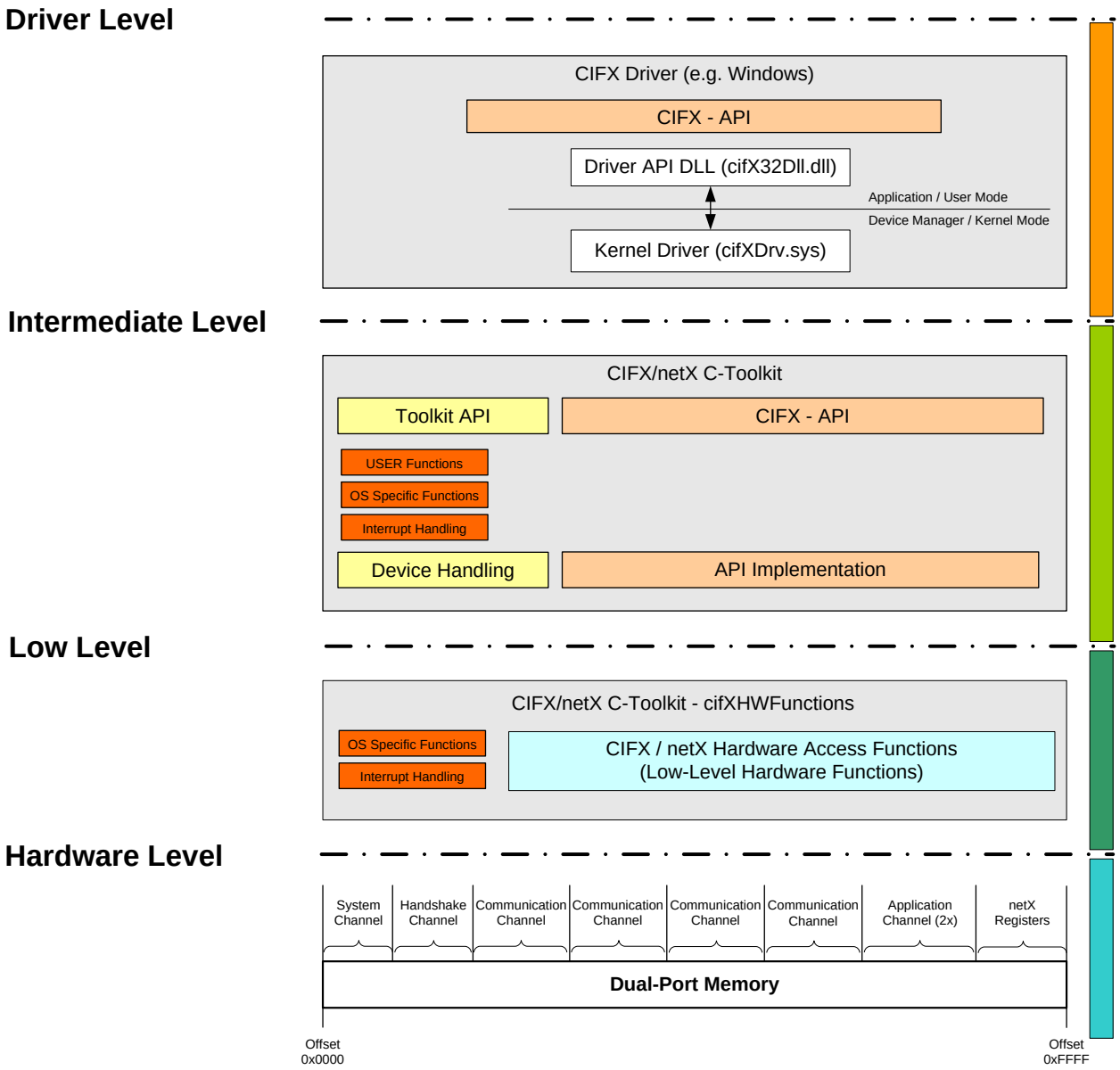

Figure 1: Overview: Programming levels

Figure 2 gives an overview of the possible development environments and the migration to embedded systems. This helps to choose a programming level fit best to the application needs.

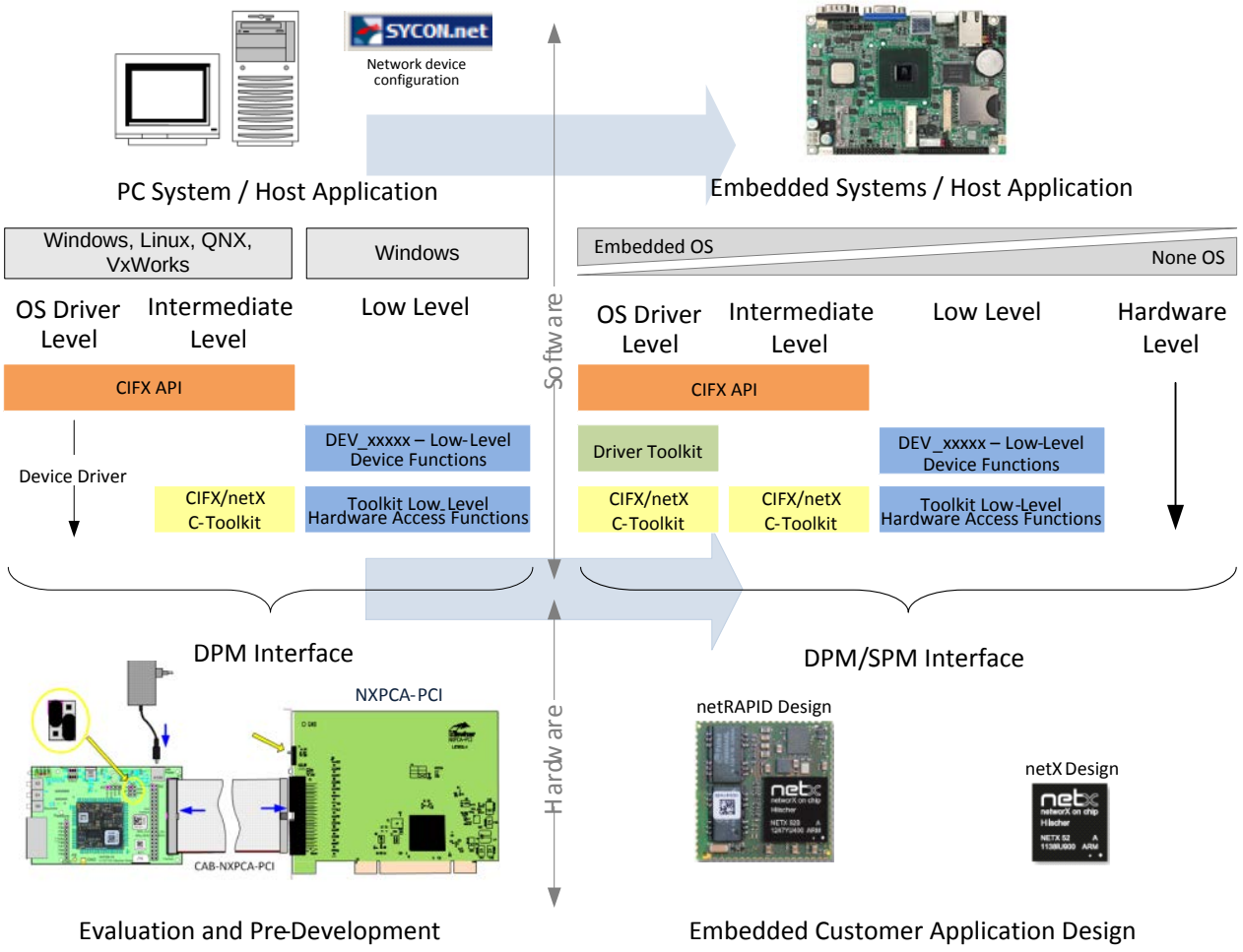

Figure 2: Select a programming level

2.3.1 Hardware level

The hardware level is the direct access to the physical DPM of a device. Programming is supported by offering C header files describing the DPM structures, content, states, flags, bit masks etc.

Usage	One or more netX chip(s) or COMX modules connected to microcontrollers - Very limited system resources - Highly optimized hardware access - Dedicated function to access the hardware - No C-based development environment
Manuals	netX Dual-Port Memory Interface DPM xx EN.pdf (Dual Port Memory layout, structures and functionalities)
CD / DVD	NXDRV-TKIT or Communication Solution DVD
Header files	rcX_User.h Dual Port Memory structures rcX_Public.h Asynchronous packet definitions

2.3.2 Low level

Access to the DPM via pre-created hardware functions which are part of the "*cifX netX Toolkit*" (see "*Toolkit Low-Level Hardware Access Functions*").

Usage	One or more netX Chip(s), COMX modules or CIFX cards connected to microcontrollers - DPM or ISA connection to the netX hardware - with or without an operating system - limited system resources (RAM/FLASH)

Programming resources

Manuals	netX Dual-Port Memory Interface DPM xx EN.pdf (Dual-port memory layout, structures and functionalities) CIFX netX Toolkit DPM TK xx EN.pdf
CD /DVD	NXDRV-TKIT or Communication Solution DVD
Source files	Toolkit .\Examples\cifXTKitHWFunctions directory cifXUser.h cifXErrors.h netX_RegDefs.h cifXHWFunctions.h.c / cifXHWFunctions.h cifXEndianess.c / cifXEndianess.h cifXInterrupt.c / cifXInterrupt.h

2.3.3 Intermediate level

Using the "*cifX netX Toolkit*" and CIFX API functions offered by the toolkit. Porting the toolkit to own hardware platforms or write own device drivers.

Usage	One or more netX chip(s), COMX modules, CIFX cards connected to a host PC system - Support for standard Hilscher devices (complete function support) - C / C++ development environment - With or without an operating system - Writing own drivers

Programming resources

Manuals	CIFX netX Toolkit DPM TK xx EN.pdf CIFX API PR xx EN.pdf
CD /DVD	NXDRV-TKIT or Communication Solution DVD
Source files	Toolkit .\cifXToolkit directory

2.3.4 Driver level

Using the CIFX API offered by a operating system drivers created by Hilscher to write own user applications on top of a Hilscher netX based communication device.

Usage	Host PC system with an operating system or real time extension supported by Hilscher - COMX modules, CIFX cards connected to a host PC - ISA / PC104 / PCI or PCIexpress bus system - Support for standard Hilscher devices (complete function support) - Creation own high level user applications

Programming resources

Manuals	CIFX API PR xx EN.pdf
CD /DVD	NXDRV-xxx or Communication Solution DVD
Source files	Delivered with the driver cifXUser.h cifXErrors.h

3 Fundamentals - DPM layout and content

The fundamental of the netX hardware is the so call dual-ported memory (DPM) of netX-based hardware. Hilscher has defined a memory structure (netX DPM Interface structure) and corresponding definitions suitable for the handling of communication devices, offering all information needed by applications to manage the hardware and the underlying Hilscher communication firmware.

It covers also system state information, the functionalities to exchange data with netX based hardware and to synchronize data access between a host and the asynchronous working netX firmware.

All Hilscher netX-based devices processing a Hilscher standard firmware performing in the same way.

3.1 The DPM layout

The DPM structure is a representation of the hardware functions and places the functions and corresponding information into separate, independent, areas.

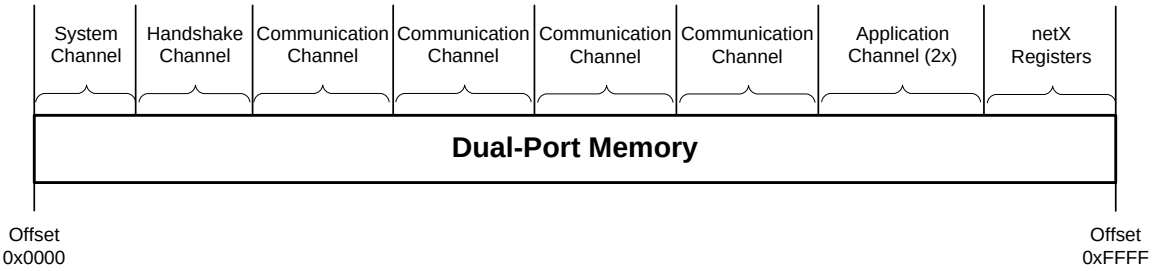

Figure 3: Dual-port memory layout

- Maximum DPM size = 64Kbyte
- COMX modules can have a different size (e.g. 8 KByte)

3.1.1 DPM area definition

System channel / System device

- Global system information and system services like firmware download and hardware reset etc.
- Global system state information
- General channel information

Handshake channel

- Special hardware registers containing synchronization flags and channel states
- Independent registers pairs for each channel
- Register pairs consists of a host dedicated and netX dedicated register
- Registers are able to create interrupts on the corresponding (host register on netX, netX register on host side)

Communication channel (max. 4 times)

- Dedicated channel to access a protocol stack
- Dedicated handshake registers in the "Handshake-Channel"
- Independent form other channels on the hardware
- General communication control and state information
- Protocol specific control, state and data areas
- Default layout, default size 15616 byte

Application channel

- Dedicated channel to access a customer application running on netX
- Optional and not defined yet

netX Global Register Block

- Direct access to netX chip registers
- Defined in the "netX Program Reference Guide"
- Functions like netX chip state / Reset / Interrupt / DMA etc.
- Needed for RAM-based devices, not needed for Flash-based devices

3.1.2 DPM definitions, structures and header files

The whole DPM structures and definitions are provided by two C header files.

- rcx_User.h

  Definition of the DPM structure, data blocks and all global definitions offered by the DPM. The C-header file contains all parts necessary to work with the DPM by using symbolic names.

- rcx_Public.h

  Definition of "*rcX Public Packet*" functions.

  Packets are asynchronous commands which can be sent to the hardware. A packet is the combination of a "Packet-Header", containing global command, routing and handling information for the packet, and a following "User-Data Area" containing command specific data.

Note: All structures and definition used in the following chapters can be found in these two header files

3.2 System Channel structure

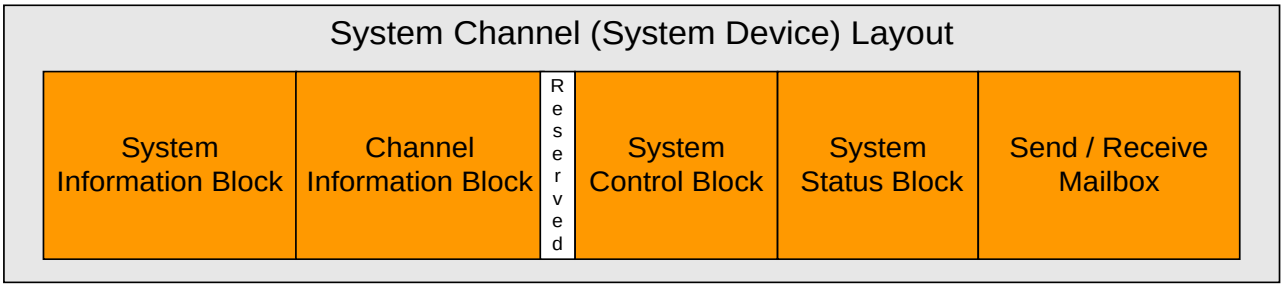

Figure 4: System Channel / System Device structure

System Channel structure definition:

```
/*****************************************************************************/
/*! Structure of the whole system channel (DPM) (Size 512 Byte)              */
/*****************************************************************************/
typedef __RCX_PACKED_PRE struct NETX_SYSTEM_CHANNELtag
{
  NETX_SYSTEM_INFO_BLOCK      tSystemInfo;                        /*!< 0x000:0x02F System information block */
  NETX_CHANNEL_INFO_BLOCK     atChannelInfo[NETX_MAX_SUPPORTED_CHANNELS];
                                                                  /*!< 0x030:0x0AF Channel information block */
  NETX_HANDSHAKE_CELL         tSysHandshake;                      /*!< 0x0B0:0x0B3 Handshake cells used, if not
                                                                                    in Handshake block */
  uint8_t                     abReserved[4];                      /*!< 0x0B4:0x0B7 unused/reserved */
  NETX_SYSTEM_CONTROL_BLOCK   tSystemControl;                     /*!< 0x0B8:0x0BF System control block */
  NETX_SYSTEM_STATUS_BLOCK    tSystemState;                       /*!< 0x0C0:0x0FF System state block */
  NETX_SYSTEM_SEND_MAILBOX    tSystemSendMailbox;                 /*!< 0x100:0x17F Send mailbox */
  NETX_SYSTEM_RECV_MAILBOX    tSystemRecvMailbox;                 /*!< 0x180:0x1FF Receive mailbox */
} __RCX_PACKED_POST NETX_SYSTEM_CHANNEL;
```

- **System Information Block:** (NETX_SYSTEM_INFO_BLOCK)

  General system information for the complete device

- **Channel Information Block:** (NETX_CHANNEL_INFO_BLOCK)

  Information about available communication channels, general channel information and channel layout.

- **System Control Block:** (NETX_SYSTEM_CONTROL_BLOCK)

  General system control functions, currently not used

- **System Status Block:** (NETX_SYSTEM_STATUS_BLOCK)

  General system and device information

- **System Send/Receive Mailbox:** (NETX_SYSTEM_SEND /RECEIVE_MAILBOX)

  System channel mailbox for asynchronous packet based commands

3.2.1 System information

General system information can be obtained from the System Information Block NETX_SYSTEM_INFO_BLOCK (see below).

```
/*****************************************************************************/
/*! System information block (Size = 48 Byte)                                */
/*****************************************************************************/
typedef __RCX_PACKED_PRE struct NETX_SYSTEM_INFO_BLOCKtag
{
  uint8_t   abCookie[4];                                  /*!< 0x00 "netX" cookie */
  uint32_t  ulDpmTotalSize;                               /*!< 0x04 Total Size of the whole dual-port
memory in bytes */
  uint32_t  ulDeviceNumber;                               /*!< 0x08 Device number */
  uint32_t  ulSerialNumber;                               /*!< 0x0C Serial number */
  uint16_t  ausHwOptions[4];                              /*!< 0x10 Hardware options, xC port 0..3 */
  uint16_t  usManufacturer;                               /*!< 0x18 Manufacturer Location */
  uint16_t  usProductionDate;                             /*!< 0x1A Date of production */
  uint32_t  ulLicenseFlags1;                              /*!< 0x1C License code flags 1 */
  uint32_t  ulLicenseFlags2;                              /*!< 0x20 License code flags 2 */
  uint16_t  usNetxLicenseID;                              /*!< 0x24 netX license identification */
  uint16_t  usNetxLicenseFlags;                           /*!< 0x26 netX license flags */
  uint16_t  usDeviceClass;                                /*!< 0x28 netX device class */
  uint8_t   bHwRevision;                                  /*!< 0x2A Hardware revision index */
  uint8_t   bHwCompatibility;                             /*!< 0x2B Hardware compatibility index */
  uint8_t   bDevIdNumber;                                 /*!< 0x2C Device identification number (rotary
switch) */
  uint8_t   bReserved;                                    /*!< 0x2D Reserved byte */
  uint16_t  usReserved;                                   /*!< 0x2E:0x2F Reserved */
} __RCX_PACKED_POST NETX_SYSTEM_INFO_BLOCK;
```

3.2.2 System status

The System Status Block contains global system information and the corresponding data structure is defined as NETX_SYSTEM_STATUS_BLOCK.

```
/*****************************************************************************/
/*! System status block (Size = 64 Byte)                                     */
/*****************************************************************************/
typedef __RCX_PACKED_PRE struct NETX_SYSTEM_STATUS_BLOCKtag
{
  uint32_t  ulSystemCOS;                                  /*!< 0x00 System channel change of state
acknowledge */
  uint32_t  ulSystemStatus;                               /*!< 0x04 Actual system state */
  uint32_t  ulSystemError;                                /*!< 0x08 Actual system error */
  uint32_t  ulBootError;                                  /*!< 0x0C Bootup error (only set by 2nd Stage
Bootloader) */
  uint32_t  ulTimeSinceStart;                             /*!< 0x10 time since start in seconds */
  uint16_t  usCpuLoad;                                    /*!< 0x14 cpu load in 0,01% units (10000 => 100%)
*/
  uint16_t  usReserved;                                   /*!< 0x16 Reserved */
  uint32_t  ulHWFeatures;                                 /*!< 0x18 Hardware features */
  uint8_t   abReserved[36];                               /*!< 0x1C:3F Reserved */
} __RCX_PACKED_POST NETX_SYSTEM_STATUS_BLOCK;
```

3.3 Handshake Channel

The Handshake Channel contains the "Channel Handshake Registers" with their channel states and synchronization flags.

Synchronized access to data areas is always necessary if data consistency must be assured between the host and the netX firmware, especially if a data area consists of more than one byte.

Byte consistency is guaranteed by the DPM hardware preventing concurrent access to a single byte while multi byte areas requiring other mechanism realized by the handshake flags (software) and by defining access rules (read only / write only areas).

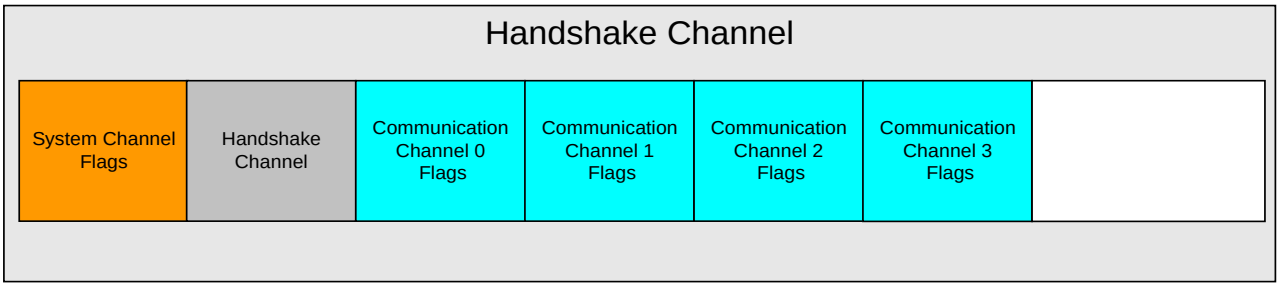

Figure 5: Handshake Channel

Handshake channel structure definition

```
/*****************************************************************************/
/*! Handshake channel definition                                             */
/*****************************************************************************/
typedef struct NETX_HANDSHAKE_CHANNELtag
{
  NETX_HANDSHAKE_CELL tSysFlags;                                  /*!< 0x00 System handshake flags */
  NETX_HANDSHAKE_CELL tHskFlags;                                  /*!< 0x04 not used */
  NETX_HANDSHAKE_CELL tCommFlags0;                                /*!< 0x08 channel 0 handshake flags */
  NETX_HANDSHAKE_CELL tCommFlags1;                                /*!< 0x0C channel 1 handshake flags */
  NETX_HANDSHAKE_CELL tCommFlags2;                                /*!< 0x10 channel 2 handshake flags */
  NETX_HANDSHAKE_CELL tCommFlags3;                                /*!< 0x14 channel 3 handshake flags */
  NETX_HANDSHAKE_CELL tAppFlags0;                                 /*!< 0x18 not supported yet         */
  NETX_HANDSHAKE_CELL tAppFlags1;                                 /*!< 0x1C not supported yet         */
  uint32_t            aulReserved[ 56 ];                          /*!< 0x20 - 0xFF */
} NETX_HANDSHAKE_CHANNEL;
```

Note: *Handshake Registers* are able to generate interrupts.
On the host side if the firmware writes to the *netX flags* and on the firmware side if the host writes to the *host flags*.

3.3.1 Handshake Register functionality

Each channel has an own handshaking register, except the *Handshake Channel* itself which does not need a synchronization mechanism.

A handshake register is defined as a 32-bit value, divided into two parts. The "*NETX Flags* " and the "*HOST Flags*" and a definition where only the owner of a register is allowed to read and write it, while the opposite user is only allowed to read the register. And handshake registers are able to generate interrupt requests.

System Channel Handshake Register structure

The system channel handshake register defines 8 bits for the netX firmware (netX Flags = bNetxFlags) and 8 bits for the host application (Host Flags = bHostFlags), because it need less synchronization flags than a communication channel (see t8Bit structure below).

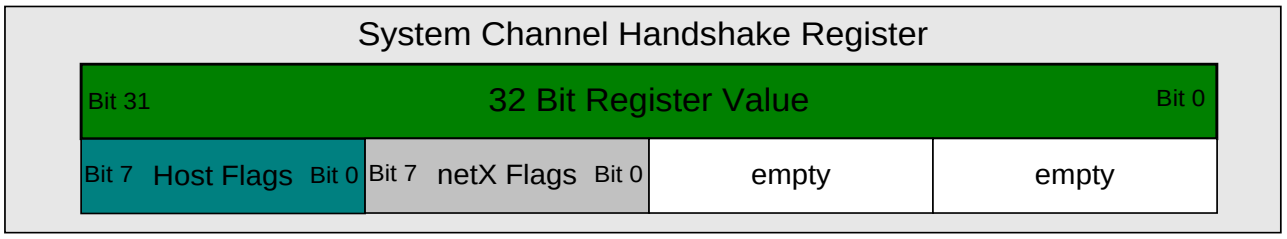

Figure 6: System Channel Handshake Register

Communication Channel Handshake Register structure

The communication channel handshake register defines 16 bits for the netX firmware (netX Flags = usNetxFlags) and 16 bits for the host application (Host Flags = usHostFlags) (see t16Bit structure below).

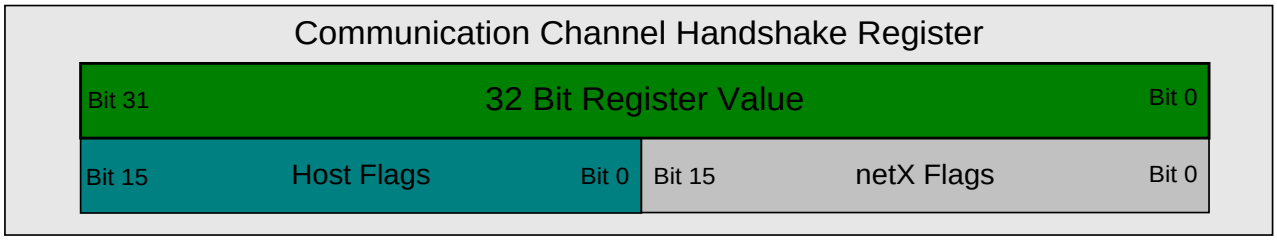

Figure 7: Communication Channel Handshake Register

Handshake register pairs are defined in the structure NETX_HANDSHAKE_CELL.

```
/*****************************************************************************/
/*! Handshake cell definition                                                */
/*****************************************************************************/
typedef __RCX_PACKED_PRE union NETX_HANDSHAKE_CELLtag
{
  __RCX_PACKED_PRE struct
  {
    volatile uint8_t abData[2];        /*!< Data value, not belonging to handshake */
    volatile uint8_t bNetxFlags;       /*!< Device status flags (8Bit Mode) */
    volatile uint8_t bHostFlags;       /*!< Device command flags (8Bit Mode) */
  } __RCX_PACKED_POST t8Bit;

  __RCX_PACKED_PRE struct
  {
    volatile uint16_t usNetxFlags;     /*!< Device status flags (16Bit Mode) */
    volatile uint16_t usHostFlags;     /*!< Device command flags (16Bit Mode)*/
  } __RCX_PACKED_POST t16Bit;
  volatile uint32_t ulValue;            /*!< Handshake cell value */
} __RCX_PACKED_POST NETX_HANDSHAKE_CELL;
```

3.4 Communication Channel structure

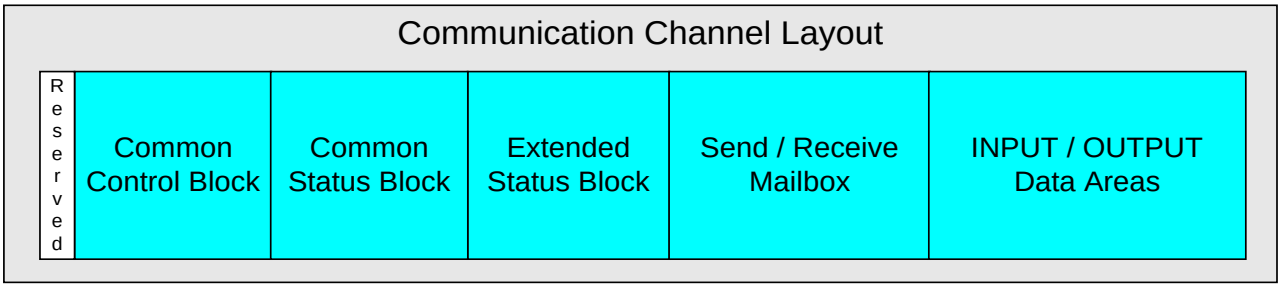

Figure 8: Communication Channel structure

Communication Channel structure definition (NETX_DEFAULT_COMM_CHANNEL)

```
/*****************************************************************************/
/*! Structure of the DEFAULT communication channel (Size 15616 Byte)         */
/*****************************************************************************/
typedef __RCX_PACKED_PRE struct NETX_DEFAULT_COMM_CHANNELtag
{
  NETX_HANDSHAKE_BLOCK        tReserved;                        /*!< 0x000:0x007 Reserved for later use */
  NETX_CONTROL_BLOCK          tControl;                         /*!< 0x008:0x00F Control block */
  NETX_COMMON_STATUS_BLOCK    tCommonStatus;                    /*!< 0x010:0x04F Common status block */
  NETX_EXTENDED_STATUS_BLOCK  tExtendedStatus;                  /*!< 0x050:0x1FF Extended status block */
  NETX_SEND_MAILBOX_BLOCK     tSendMbx;                         /*!< 0x200:0x83F Send mailbox block */
  NETX_RECV_MAILBOX_BLOCK     tRecvMbx;                         /*!< 0x840:0xE7F Recveice mailbox block */
  uint8_t                     abPd1Output[NETX_HP_IO_DATA_SIZE];  /*!< 0xE80:0xEBF Process data 1 output area */
  uint8_t                     abPd1Input[NETX_HP_IO_DATA_SIZE];   /*!< 0xEC0:0xEFF Process data 1 input area */
  uint8_t                     abReserved1[256];                 /*!< 0xF00:0xFFF Reserved */
  uint8_t                     abPd0Output[NETX_IO_DATA_SIZE];     /*!< 0x1000:0x267F Process data 0 output area */
  uint8_t                     abPd0Input[NETX_IO_DATA_SIZE];      /*!< 0x2680:0x3CFF Process data 0 input area */
} __RCX_PACKED_POST NETX_DEFAULT_COMM_CHANNEL;
```

- **Common Control Block:** (NETX_CONTROL_BLOCK)

  Common communication channel and protocol stack control functions
- **Common Status Block:** (NETX_COMMON_STATUS_BLOCK)

  Common communication channel and protocol stack status information
- **Extended Status Block:** (NETX_EXTENDED_STATUS_BLOCK)

  Protocol stack specific state information
- **Send/Receive Mailbox:** (NETX_SEND_MAILBOX_BLOCK / NETX_RECEIVE_MAILBOX_BLOCK)

  Communication channel mailbox system for asynchronous packet based commands.
- **INPUT/OUTPUT Areas:** (abPd(x)Output / abPd(x)Input)

  Communication channel cyclic process data images (Input / Output data areas)

Note: An 8 Kbyte *Communication Channel Structure* is also available, see NETX_8K_DPM_COMM_CHANNEL.

3.4.1 Channel information

General communication channel information about all available channels is located in the "*System Channel -> Channel Information Block*".

3.4.2 Channel state

Communication channel general state and protocol stack specific states are offered by the following structures:

- Common Status Block => NETX_COMMON_STATUS_BLOCK
  General communication channel information.
- Extended Status Block => NETX_EXTENDED_STATUS_BLOCK
  Protocol stack specific channel information.

3.5 Data transfer methods and areas

The netX firmware offers two general methods to exchange data with it.

- **Non-Cyclic data via *Packets* and a *Mailbox System***

  Non-cyclic data are binary data streams named "*Packets*". A packet is a structure which consists of a header with general administration information (command / length / source / destination etc) and a data area. The mailbox system contains two memory areas used to exchange packets between the host and the netX device.

- **Cyclic data via Input/Output *Data Areas***

  Cyclic data are the field bus protocol stack input and output data which are cyclical exchanged between members of a field bus network.

3.5.1 Packet definition and transfer via a mailbox system

A packet is a memory area including command and data areas which should be transferred to and back from the hardware.

Packets are exchanged with the firmware by using a "*Mailbox System*".

A "*Mailbox System*" defines one memory area to send packets to the firmware and one area to receive packets from the firmware.

Access synchronisation to these areas is done by "*Handshake-Register-Flags" (see Handshake Registers*), signalling the state of the "*Mailbox Memory Area*" (empty or full).

To simplify the send and receive handling the mailbox system and their state flags are divided into a send direction and receive direction with separate handshake flags.

General packet definition

Part of packet	Variable	Data type	Description
Packet Header-> tHeader	ulDest	UINT32	Destination Queue Handle
	ulSrc	UINT32	Source Queue Handle
	ulDestId	UINT32	Destination Queue Reference
	ulSrcId	UINT32	Source Queue Reference
	ulLen	UINT32	Packet Data Length (in Bytes)
	ulId	UINT32	Packet Identification As Unique Number
	ulState	UINT32	Status / Error Code
	ulCmd	UINT32	Command / Response
	ulExt	UINT32	Reserved
	ulRout	UINT32	Routing Information
Packet Data-> tData	…	…	User Data => Specific to ulCmd

Table 3: General packet structure

Default Packet structure from rcX_User.h

```
/*****************************************************************************/
/*! Default RCX packet header structure                                      */
/*****************************************************************************/
typedef __RCX_PACKED_PRE struct RCX_PACKET_HEADERtag
{
  uint32_t   ulDest;                     /*!< 00:04, Destination of packet, process queue */
  uint32_t   ulSrc;                      /*!< 04:04, Source of packet, process queue */
  uint32_t   ulDestId;                   /*!< 08:04, Destination reference of packet*/
  uint32_t   ulSrcId;                    /*!< 12:04, Source reference of packet */
  uint32_t   ulLen;                      /*!< 16:04, Length of packet data without header */
  uint32_t   ulId;                       /*!< 20:04, Identification handle of sender */
  uint32_t   ulState;                    /*!< 24:04, Status code of operation */
  uint32_t   ulCmd;                      /*!< 28:04, Packet command */
  uint32_t   ulExt;                      /*!< 32:04, Extension */
  uint32_t   ulRout;                     /*!< 36:04, Router (internal use only) */
} __RCX_PACKED_POST RCX_PACKET_HEADER;

/*****************************************************************************/
/*! Default RCX packet structure, including user data                        */
/*****************************************************************************/
typedef __RCX_PACKED_PRE struct RCX_PACKETtag
{
  RCX_PACKET_HEADER tHeader;                    /*!< Packet header */
  uint8_t     abData[RCX_MAX_DATA_SIZE];        /*!< Packet data */
} __RCX_PACKED_POST RCX_PACKET;
```

Mailbox System

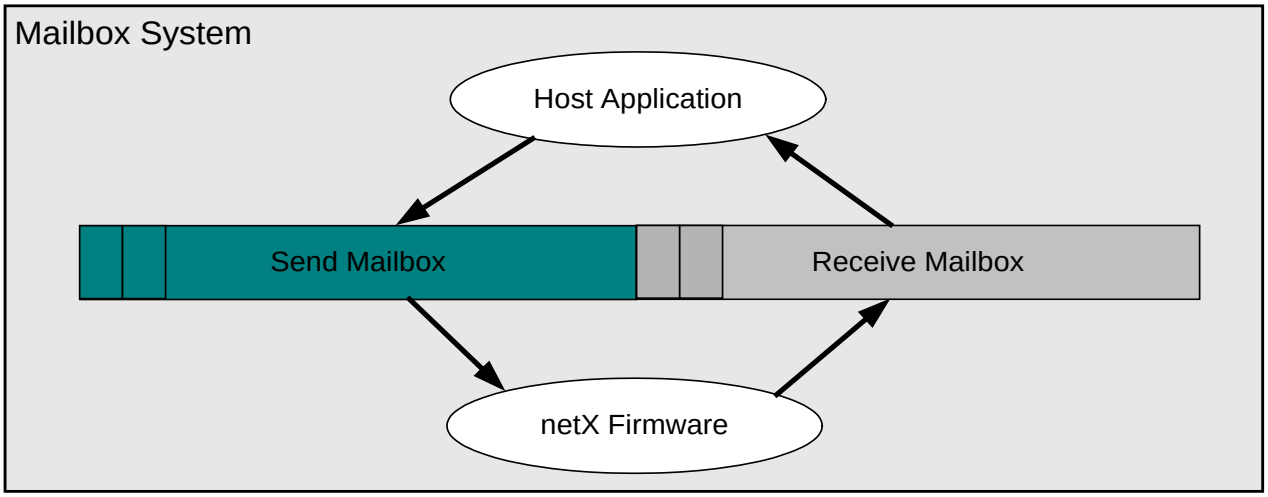

Figure 9: Mailbox System

The netX firmware defines 2 mailbox structures:

- System Channel Mailbox

  NETX_SYSTEM_SEND_MAILBOX / NETX_SYSTEM_RECEIVE_MAILBOX

- Communication Channel Mailbox

  NETX_SEND_MAILBOX_BLOCK / NETX_RECV_MAILBOX_BLOCK

Example: System Channel Mailbox structure

```
/*****************************************************************************/
/*! System send packet mailbox (Size 128 Byte)                               */
/*****************************************************************************/
typedef __RCX_PACKED_PRE struct NETX_SYSTEM_SEND_MAILBOXtag
{
  uint16_t  usPackagesAccepted;                    /*!< Number of packages that can be accepted */
  uint16_t  usReserved;                              /*!< Reserved */
  uint8_t   abSendMbx[NETX_SYSTEM_MAILBOX_MIN_SIZE];  /*!< Send mailbox packet buffer */
} __RCX_PACKED_POST NETX_SYSTEM_SEND_MAILBOX;

/*****************************************************************************/
/*! System receive packet mailbox (Size 128 Byte)                            */
/*****************************************************************************/
typedef __RCX_PACKED_PRE struct NETX_SYSTEM_RECV_MAILBOXtag
{
  uint16_t  usWaitingPackages;                   /*!< Number of packages waiting to be processed */
  uint16_t  usReserved;                              /*!< Reserved */
  uint8_t   abRecvMbx[NETX_SYSTEM_MAILBOX_MIN_SIZE];  /*!< Receive mailbox packet buffer */
} __RCX_PACKED_POST NETX_SYSTEM_RECV_MAILBOX;
```

3.5.2 I/O data transfer

I/O Data Areas containing the cyclic process data of a fieldbus protocol stack. These areas are also divided into output and input areas with dedicated synchronization flags for each area and transfer direction.

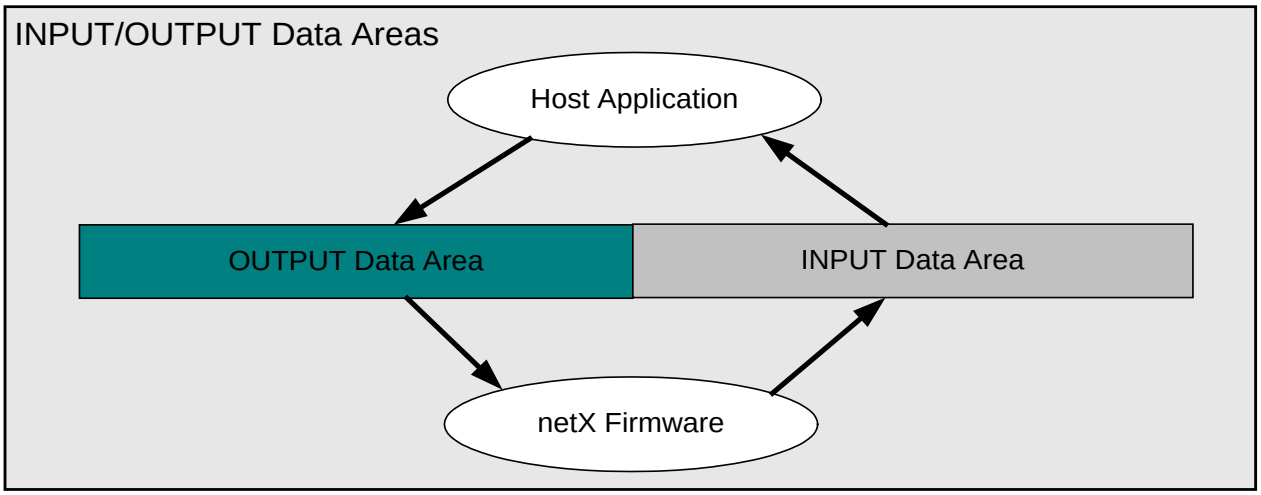

Figure 10: Input / Output Data Area

3.6 Communication mechanism and synchronization

Handshake registers are used to signal general channel information, activate commands and for synchronized data access to the mailbox system, I/O data areas and to handle change of state information (COS).

3.6.1 System Channel Handshake Register

- **netX System Flags (NSF)**

  => read and written by the netX firmware, host is only allowed to read

- **Host System Flags (HSF)**

  => read and written by the host, netX firmware is only allowed to read

3.6.2 Communication Channel Handshake Register

- **netX Communication Flags (NCF)**

  => read and written by the netX firmware, host is only allowed to read

- **Host Communication Flags (HCF)**

  => read and written by the host, netX firmware is only allowed to read

3.6.3 Working with Handshake Registers and Flags

Handshake flags are located in the handshake registers and either used to signal a specific state, activate a specific command or, in conjunction with another flag, to create a synchronized data access.

3.6.3.1 Simple State and Command Handshake Flags

These flags just having two states (signaled = 1 / none signaled = 0).

- 0 State / Command is inactive
- 1 State / Command is active

System Channel State / Command Flags

HSF_RESET	1 = Activate a RESET on the device / 0 = do nothing
HSF_BOOTSTART	1 = Activate the Bootloader of the device / 0 = do nothing
NSF_READY	1 = System Channel is READY / 0 = System Channel is not READY
NSF_ERROR	1 = System Channel error / 0 = no error

Communication Channel State / Command Flags

NCF_COMMUNICATING	1 = Communicating / 0 = not Communicating
NCF_ERROR	1 = Error / 0 = no error

3.6.3.2 Synchronization Handshake Flags

A netX system defines 3 different synchronization mechanisms realized by handshake flags. Therefore pairs of handshake bits are defined. One bit form the host handshake flags and one bit from the netX handshake flags. Both together are used to handle the synchronization.

Following functions are synchronized:

- Packet Data Transfer via the mailbox system
- I/O data transfer mechanism
- COS - Change Of State mechanism

3.6.3.3 Packet data transfer - Handshake Flag synchronization

The following flags pairs are used for packet data transfer (mailbox handling).The flags are found in the system channel handshake register and the communication channel handshake register (ulNetxFlags / ulHostFlags):

- System Channel Flags

  HSF_SEND_MBX_CMD / NSF_SEND_MBX_ACK

  NSF_RECV_MBX_CMD / HSF_RECV_MBX_ACK

- Communication Channel Flags

  HCF_SEND_MBX_CMD / NCF_SEND_MBX_ACK

  NCF_RECV_MBX_CMD / HCF_RECV_MBX_ACK

3.6.3.4 I/O data transfer - Handshake Flag synchronization

The following flags pairs are used for I/O data transfer synchronization, the flags are found in the communication channel handshake register (ulNetxFlags / ulHostFlags):

- I/O Data Exchange

  HCF_PDO_OUT_CMD / NCF_PD0_OUT_ACK

  HCF_PDO_IN_CMD / NCF_PD0_IN_ACK

The I/O data transfer uses an additional definition (*Handshake Mode*) which defines the initiator of a data transfer.

Handshake Mode definition

- Buffered Host Controlled Mode Host initiates the INPUT/OUTPUT data update
- Buffered Device Controlled Mode Device initiates the INPUT/OUTPUT data update

The combination of the flag state and the handshake mode is used to handle the I/O data transfer.

3.6.3.5 Change of State Mechanism (COS)

A communication channel has more options and commands than bits in the handshake flag register.

Therefore a so called "*Change of State (COS)*" mechanism is defined which extends the direct usable handshake flags by another 32-bit value with additional states.

Furthermore the COS mechanism expects an acknowledgement if a state change was signaled, before another state change will be signaled.

This mechanism is direction oriented and distinguishes between state changes from the host application and form the device:

- Device state changes are named "*Communication COS Handling*"
- Application state changes are named "*Application COS Handling*

3.7 Programming interface - CIFX-API

The CIFX API is the general application programming interface (API) which offers all necessary function to handle a netX device.

The API offers general handling functions and functions corresponding to the DPM components described above.

Example API functions

Asynchronous services (Packets)	
API function	**Description**
xChannelGetMBXState	Retrieve the channels mailbox state
xChannelGetPacket	Read a packet from the channel receive mailbox
xChannelPutPacket	Send a packet via the channel send mailbox
…	
Cyclic data services (I/O's)	
API function	**Description**
xChannelIORead	Instructs the device to place the latest data into the DPM input data area and passes them to the user
xChannelIOWrite	Copies the data to the DPM send data area and waits for the firmware to retrieve them
…	

Table 4: Asynchronous services / Cyclic data services (Example API functions)

For a complete API description, see "***cifX API PR xx EN.pdf"*** manual.

4 Additional information

4.1 Hardware and protocol stack identification

The DPM contains information to determine the hardware type and the protocol stack type. And if the protocol is a master or slave stack.

- Hardware Identification

  This information can be found in the "**System Information Block**"

  Device Class => DPM Offset 0x0028 usDeviceClass

- Hardware Assembly Option

  Defines which communication hardware is offered by the device

  This information can be found in the "**System Information Block**"

  Hardware Option => DPM Offset 0x0010 ausHwOptions

- Protocol Stack Identification

  The protocol stack identification per communication channel can be found in the Channel I "**Channel Information Block**"

  Therefore evaluate the channel type (bChannelType = COMMUNICATION) and read the communication class (usCommunicationClass) and protocol class (usProtocolClass).

For more information, see "**netX Dual Port Memory Interface DPM...pdf**" manual.

4.2 Additional functions

- System and Channel Information

  Can be read via *xSysdeviceInfo*()

- Handshake States

  Can be read via a *xChannelInfo*() function

- Transfer Packet

  If a synchronous packet transfer must be realized, where the user function has to wait until the answer to the previously command packet is received, a transfer packet functions must be implemented by using *xChannelPutPacket*() and *xChannelGetPacket*().

- Interrupt mode / Event handling

  Event handling can be used if the hardware is handled in interrupt mode. Than the toolkit / drivers are able to create events for an application to signal state changes of the hardware. The user application can use the *xChannelRegisterEvent*() function to use hardware events.

- Watchdog

  The CIFX API offers a watchdog function per channel, where the protocol stack is able to supervise an user application and if the user application fails, the bus communication can be stopped. See *xChannelWatchdog*() function in the CIFX API.

- DMA

  PCI bus based netX hardware is able to work as a bus master DMA device. This means the hardware can active read and write data form the memory of a host system. The netX DMA mode is limited to the Input/Output data from the netX hardware. The CIFX API offers a *xChannelDMAState*() functions to switch the I/O data exchange with the hardware to DMA mode assumed, the hardware is PCI bus based and runs in interrupt mode.

4.3 Hardware and driver installation guides

Hilscher devices are delivered with a hardware and driver installation guide describing the step by step handling to get a working netX device into a PC based host system

A description on how to install and pre-configure the hardware you will found under:

.\documentation\Installation - "cifX Device Driver Installation for Windows"

4.4 Load a firmware to a CIFX card

Hilscher offers different communication devices. PC based CIFX cards e.g. CIFX 50 / CIFX 70 / CIFX 90 devices, which are PCI bus based so not store the firmware and configuration file on the hardware.

Such devices must be pre-loaded with a firmware to be usable by a configuration tool or by any application which want a field bus protocol firmware.

Actual firmware can be taken from the "Communication Solution DVD" (either shipped with the device or downloadable Hilscher web site the following link:

https://www.hilscher.com, Support > Downloads

Go to the "Firmware" subdirectory and choose the appropriate firmware for your hardware as described in the "***cifX Device Driver Installation for Windows***" manual.

On COMX module the firmware is stored on the hardware and a firmware download during start-up is not necessary.

4.5 Start with an example program

After loading the hardware with the necessary firmware (in case of PCI based devices) it is possible to use it and to start with example programs (e.g. "cifxTest_console example" from the CIFX Driver CD, see .\examples\cifXTest_console).

This example contains most of the CIFX API functions in separate source modules an mostly contains a corresponding project file (e.g. Microsoft Visual Studio project under Windows) which can be used to compile and run the example.

The main function of the example can be found in the source module cifXTest_console.cpp which than calls the function RunCifXConsoleTest() in the source module cifXConsole_Main.cpp for further processing.

4.6 Protocol stack handling via CIFX API

An overview of the minimal necessary API functions and the order how they should be called can be found in the CIFX-API manual Chapter 5 and 6.

5 Additional resources

5.1 Protocol API manuals

Each fieldbus protocol stack has it own documentation, describing the protocol specific functions and commands.

Protocol API manuals can be found on the "**Communication Solution DVD**" in the **.\Documentation\Programming Manuals** directory and the manual file names are containing the protocol stack name, e.g. "*PROFIBUS Master Protocol API....pdf*"

6 Appendix

6.1 List of tables

6.2 List of figures

6.3 Contacts

Headquarters

Germany
Hilscher Gesellschaft für
Systemautomation mbH
Rheinstrasse 15
65795 Hattersheim
Phone: +49 (0) 6190 9907-0
Fax: +49 (0) 6190 9907-50
E-Mail: info@hilscher.com

Support
Phone: +49 (0) 6190 9907-99
E-Mail: de.support@hilscher.com

Subsidiaries

China
Hilscher Systemautomation (Shanghai) Co. Ltd.
200010 Shanghai
Phone: +86 (0) 21-6355-5161
E-Mail: info@hilscher.cn

Support
Phone: +86 (0) 21-6355-5161
E-Mail: cn.support@hilscher.com

France
Hilscher France S.a.r.l.
69500 Bron
Phone: +33 (0) 4 72 37 98 40
E-Mail: info@hilscher.fr

Support
Phone: +33 (0) 4 72 37 98 40
E-Mail: fr.support@hilscher.com

India
Hilscher India Pvt. Ltd.
Pune, Delhi, Mumbai
Phone: +91 8888 750 777
E-Mail: info@hilscher.in

Italy
Hilscher Italia S.r.l.
20090 Vimodrone (MI)
Phone: +39 02 25007068
E-Mail: info@hilscher.it

Support
Phone: +39 02 25007068
E-Mail: it.support@hilscher.com

Japan
Hilscher Japan KK
Tokyo, 160-0022
Phone: +81 (0) 3-5362-0521
E-Mail: info@hilscher.jp

Support
Phone: +81 (0) 3-5362-0521
E-Mail: jp.support@hilscher.com

Korea
Hilscher Korea Inc.
Seongnam, Gyeonggi, 463-400
Phone: +82 (0) 31-789-3715
E-Mail: info@hilscher.kr

Switzerland
Hilscher Swiss GmbH
4500 Solothurn
Phone: +41 (0) 32 623 6633
E-Mail: info@hilscher.ch

Support
Phone: +49 (0) 6190 9907-99
E-Mail: ch.support@hilscher.com

USA
Hilscher North America, Inc.
Lisle, IL 60532
Phone: +1 630-505-5301
E-Mail: info@hilscher.us

Support
Phone: +1 630-505-5301
E-Mail: us.support@hilscher.com