Programming reference guide

netX Dual-Port Memory

Hilscher Gesellschaft für Systemautomation mbH

www.hilscher.com

DOC160904PRG02EN | Revision 2 | English | 2019-09 | Released | Public

Table of contents

1 Introduction

1.1 About this document

This document is an extension to the netX DPM Interface Manual for netX-based products and offers further detailed descriptions and background information on certain functionalities.

This also includes a memory dump and short example sources to provide explanations.

1.2 List of revisions

Rev	Date	Name	Revisions
1	2017-11-27	RMA	Created.
2	2019-09-05	RMA	Section *Update Start* added.
			C-definitions have prefix HIL_

Table 1: List of revisions

1.3 Terms, abbreviations, and definitions

Term	Description
CMD	Command
COS	Change of State
DPM	Dual-Port Memory
FW	Firmware
FIFO	“First in, first out”, storage mechanism
IO	Input/Output Data
MBX	Mailbox
MFW	Maintenance Firmware
PIO	Programmable Input/Output pins
SSBL	Second Stage Bootloader

Table 2: Terms, abbreviations, and definitions

All variables, parameters, and data of this manual use the LSB/MSB (“Intel”) data representation.

The terms host, host system, application, host application, and driver are used interchangeably to identify a process interfacing the netX via its DPM in a dual-processor system.

1.4 References to documents

[1] Hilscher Gesellschaft für Systemautomation mbH: Dual-Port Memory Interface Manual, netX Dual-Port Memory Interface, Revision 16, English.

[2] Hilscher Gesellschaft für Systemautomation mbH: Packet API, netX Dual-Port Memory packet-based services, Revision 3, English.

[3] Hilscher Gesellschaft für Systemautomation mbH: User Manual netX Second Stage Boot Loader, Revision 8, English.

[4] Hilscher Gesellschaft für Systemautomation mbH: netX LED Description, Revision 16, English.

2 General netX start-up

The netX supports different start-up scenarios depending on the hardware design. A netX design can be very flexible and the documentation focuses designs with a DPM and systems based on FLASH or RAM.

In a FLASH-based system, all executable code is stored in the FLASH while RAM-based devices must be loaded by an application. A netX firmware consists of two parts, an SSBL (Second Stage Bootloader) and a firmware file. The boot procedure is divided into 3 stages.

First stage	ROM loader start-up
Second stage	The ROM loader starts the SSBL and creates the system channel *(*see default DPM layout)
Third stage	The SSBL starts the firmware which creates the communication channels

The current stage is also signaled by the system LED recommended for all netX hardware designs.

2.1 Boot procedure

Step 1: Power-On Reset

The first component to start is always the chip-internal ROM loader. Its task is to decide on which executable code to start. Hardware settings can influence the behavior of the ROM loader allowing it to select whether to start an available firmware or to wait for a firmware to be loaded via the DPM. The ROM loader also initializes the netX controller and enables generic access to the netX (see Bootwizard Tool).

Step 2: Starting the SSBL

If no boot code is found, the ROM loader waits until an SSBL is downloaded. An SSBL is a piece of software offering additional options (beyond the scope of the ROM loader) which creates the so-called system device (system channel) and thus the DPM layout as described in reference [1]. From this point on, the DPM functionalities (e.g. mailbox system) and system services can be used to communicate with the hardware.

Step 3: Starting the final firmware

Step 3 is the start of the final firmware which contains the complete netX application also known as protocol stack software.

If a firmware is stored in the FLASH of the device, it will automatically be started by the SSBL, otherwise the SSBL will wait for a firmware to be loaded via the system channel mailbox.

Note: The ROM loader of step 1 is a pure hardware function of the netX chip and executed automatically, while steps 2 and 3 are software-driven and depend on the target hardware. If the target hardware supports non-volatile boot devices, downloading the SSBL and firmware is not necessary after power-on reset because the ROM loader will find the SSBL or an executable firmware during step 1. Without a non-volatile boot device, steps 2 and 3 must always be processed after each power-on reset.

Bootable code

A bootable code needs a boot header to be recognized by the ROM loader. This boot header contains some important information.

```
/*****************************************************************************/
/* File Header Substructures for Hilscher Downloadable Files                 */
/*****************************************************************************/

/* BOOT header (64 bytes, used for NXF) */
typedef struct HIL_FILE_BOOT_HEADER_V1_0tag
{
  /* boot block identification and bus width (8/16/32 bits) in case of a parallel flash source
device  */
  uint32_t     ulMagicCookie;              /**< see HIL_FILE_HEADER_FIRMWARE_xxx_COOKIE          */

  /* boot image source device configuration value (either parallel or serial flash) */
  union
  { uint32_t   ulSramBusTiming;            /**< parallel flash on SRAM bus: bus timing value     */
    uint32_t   ulSpiClockSpeed;            /**< serial flash on SPI:        clock speed value    */
  } unSrcMemCtrl;

  /* application data description values
*/
  uint32_t     ulAppEntryPoint;            /**< app. entry point, netX code execution starts here */
  uint32_t     ulAppChecksum;              /**< app. checksum starting from byte offset 64       */
  uint32_t     ulAppFileSize;              /**< app. size in DWORDs starting from byte offset 64  */
  uint32_t     ulAppStartAddress;          /**< app. relocation start address for binary image   */
  uint32_t     ulSignature;                /**< app. signature, always 0x5854454E = "NETX"        */

  /* destination device control values */
  union
  { /* SDRAM */
    struct
    { uint32_t ulSdramGeneralCtrl;         /**< value for SDRAM General Control register         */
      uint32_t ulSdramTimingCtrl;          /**< value for SDRAM Timing register                  */
      uint32_t aulReserved[3];
    } tSDRAMCtrl;
    /* Extension Bus */
    struct
    { uint32_t ulExtConfigCS0;             /**< value for EXT_CONFIG_CS0 register                */
      uint32_t ulIoRegMode0;               /**< value for DPMAS_IO_MODE0 register                */
      uint32_t ulIoRegMode1;               /**< value for DPMAS_IO_MODE1 register                */
      uint32_t ulIfConf0;                  /**< value for DPMAS_IF_CONF0 register                */
      uint32_t ulIfConf1;                  /**< value for DPMAS_IF_CONF1 register                */
    } tExtBusCtrl;
    /* SRAM */
    struct
    { uint32_t ulExtConfigSRAMn;           /**< value for EXT_SRAMn_CTRL register                */
      uint32_t aulReserved[4];
    } tSRAMCtrl;
  } unDstMemCtrl;
  uint32_t     ulMiscAsicCtrl;             /**< internal ASIC control register value (set to 1)   */
  uint32_t     ulSerialNumber;             /**< serial no. or user param. (ignored by ROM loader) */
  uint32_t     ulSrcDeviceType;            /**< HIL_SRC_DEVICE_TYPE_xxx                          */
  uint32_t     ulBootHeaderChecksum;       /**< sums up all 16 DWORDs and multiplies result by -1 */
} HIL_FILE_BOOT_HEADER_V1_0, *PHIL_FILE_BOOT_HEADER_V1_0;
```

2.2 Chip detection (ROM loader only)

If only the ROM loader of a netX chip is active (no bootloader or firmware is running), the netX chip can be detected by some specific values inside the DPM.

DPM content if ROM loader is active:

DPM unavailable	0xFFFF or 0x0BAD		DPM not active / connected
Chip type	**DPM address**	**Content**	**Additional information**
netX 500/100			
	none	none	There is no possibility to distinguish between netX 100/500
netX 50			
	0x0000	0x4C42584E ("NXBL")	netX 50 boot ID
netX 10			
	0x00FC	0x00000F1	
	0x0100	0x4C42584E ("NXBL")	
netX 51/52			
	0x00D9	0xF2	
	0x00CC	0xF2	
	0x0030	0x00000000	
	0x0034	0x00000000	
	0x00FC	0x00000400 0x00000800	netX 51 netX 52
	0x0100	0x4C42584E ("NXBL")	

Should a host application handle all netX chips, the detection functions should check for chip types in the following order

- netX 50
- netX 10
- netX 51/52
- default = netX 500/100

Note: For a detailed description of the chip-depending information, see the corresponding manual netX *xxx Programming Reference Guide*

2.3 Boot stage detection

The detection of the current boot stage depends on the system layout of the host CPU and the netX device implementation. FLASH-based devices always have an SSBL which should always be started after powering the device while RAM-based devices just offer the DPM created by the ROM loader.

Moreover, the netX chip type is usually known because it is firmly connected to the host CPU (e.g. data / address bus) and the operation of the netX (FLASH/RAM-based) is defined by the system layout.

Therefore, the application only has to check if a firmware must be loaded for RAM-based devices or if the SSBL or a standard firmware is already running.

Stage	DPM offset	FLASH-based device	RAM-based device	
DPM not active / connected	0x00000000	0xFFFF or 0x0BAD		netX cookie
SSBL is running	0x00000000	BOOT	BOOT	
Firmware is running	0x00000000	netX	netX	

2.3.1 System LED

Apart from the information in the DPM, each netX device is usually equipped with a system status LED (SYS LED) indicating the current state of the system. The following blink patterns are defined:

Color	State	Meaning
Yellow	Flashing cyclically at 1 Hz	netX is in bootloader mode waiting for firmware download
	Solid	netX is in bootloader mode, but an error occurred
Yellow / Green	Flashing alternatingly	The SSBL is active
Green	Solid	netX operating system is running and a firmware is started
Off	N/A	netX has no power supply or a hardware defect is detected

Table 3: SYS LED

2.4 Check firmware initialization

Irrespective of the chip detection, the main task (before an application can use the DPM to communicate with a netX system) is to check whether a firmware is running and whether the system and communication channels are available.

The netX firmware defines that the last element to be initialized in the DPM is the DPM cookie value located at offset 0x0000. If the DPM cookie is available, any other initialization of a firmware, including the DPM content, is finished and ready for evaluation by the host application.

Checking procedure:

- Check the DPM cookie until valid
- Check for system channel "READY" in *bNetXFlags*
  - Check the system error flag NSF_ERROR in *bNetXFlags*
    If set, a corresponding error code can be found in *ulSystemError.*
- Check for communication channel "READY" in *usNetXFlags*
  - Check if the communication channel error flag NCF_ERROR in *usNetXFlags* is set.
    If set, a corresponding error code can be found in *ulCommunicationError.*

DPM content example (DPM start address = 0x0026000):

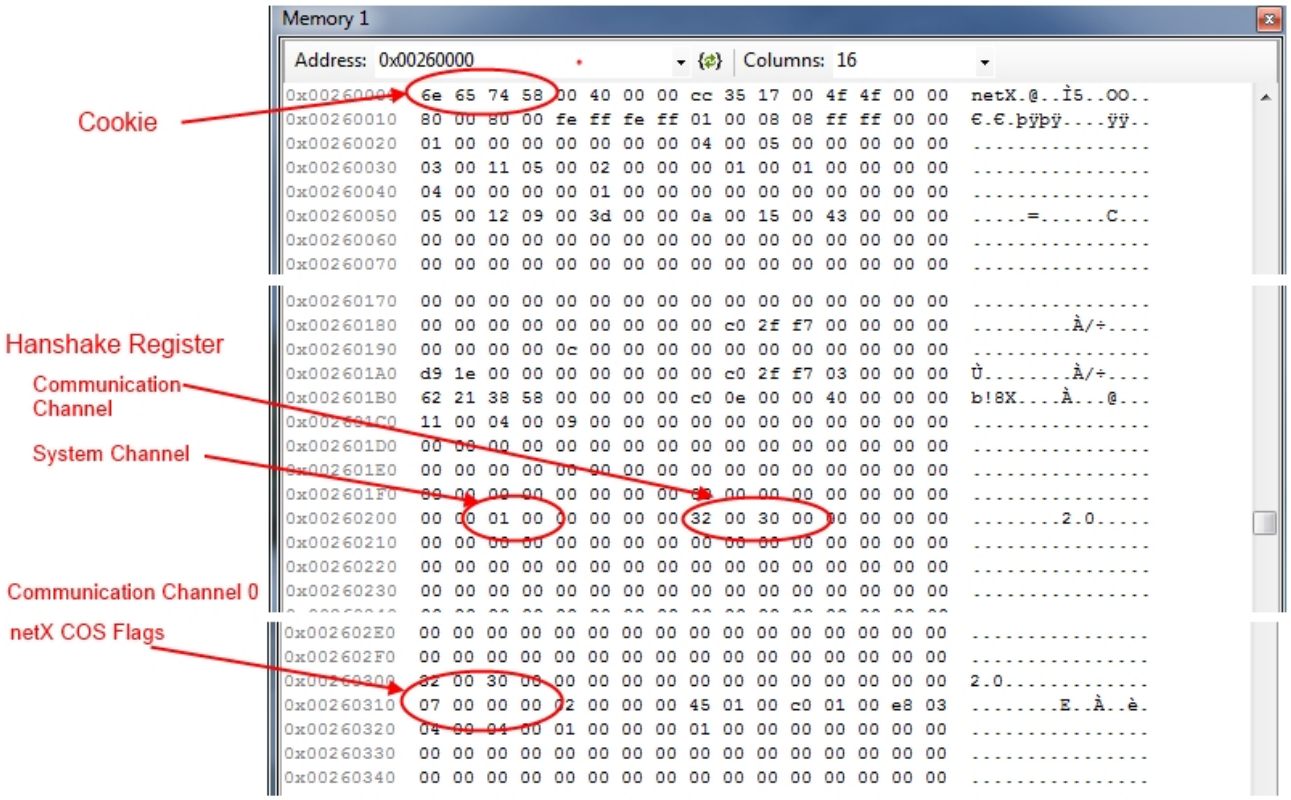

Figure 1: DPM content example

DPM Offset	Identifier	Content	Description
0x0000	DPM cookie	"netX"	Firmware is running
System Channel Information			
0x0202	bNetXFlags	0x01	netX System Flags (NSF) Bit 0 = READY
0x0203	bHostFlags	0x00	Host System Flags (HSF)
Communication Channel 0 Information			
0x0310	ulCommunicationCOS	0x0007	Communication Change of State Register Bit 0 = READY Bit 1 = RUNNING Bit 2 = BUS ON
0x0208	usNetXFlags	0x0032	netX Communication Flags (NCF) Bit 1 = NCF_ERROR Bit 4 = NCF_SEND_MBX_ACK Bit 5 = NCF_RECV_MBX_CMD
0x020A	usHostFlags	0x0030	Host Communication Flags (HCF) Bit 4 = HCF_SEND_MBX_CMD Bit 5 = HCF_RECV_MBX_ACK
0x0318	ulCommunicationError	0xC0000145	ERR_HIL_CABLE_DISCONNECT cable disconnected

In this example:

- A firmware is started
- System channel and communication channel 0 are READY
- Communication channel 0 is also configured (RUNNING)
- Communication channel 0 signals an error (NCF_ERROR)
- Communication channel 0 has no packets waiting in the send and receive mailbox

Note: The firmware will update *ulCommunicationCOS* only if the application acknowledges prior changes (indicated by firmware bit NCF_NETX_COS_CMD) via bit HCF_NETX_COS_ACK. NCF_NETX_COS_CMD / HCF_NETX_COS_ACK should therefore be handled whenever the handshake registers are processed.

2.4.1 Communication channel LEDs

Each netX device offers up to 4 LEDs per fieldbus communication connection. These so-called communication LEDs (COM LED) indicate the current state of a running fieldbus firmware and its network connection.

These LEDs are firmware-specific and described in a separate manual.

3 Chip functions

3.1 Reset operations

Several reset operations are defined for the netX hardware and firmware.

Reset overview

- **Hardware (chip) reset** to reset the netX chip and its components
- **System reset** to restart a netX device handled by the firmware
- **Boot Start** to restart a netX device and stopping in maintenance mode (e.g. SSBL or MFW)
- **Update Start** is only supported by netX90/4x00-based devices, to install / exchange a firmware.
- **Channel initialization** used to restart a communication channel

3.1.1 Hardware reset

A hardware (chip) reset can be processed only via the netX register block and depends on the netX chip type. A chip reset is usually used on RAM-based devices to bring the hardware into a defined state, e.g. after a power cycle.

Note: The accessible DPM address range must be 64 Kbytes, otherwise the register block is not accessible. For a description of the netX register block, see the corresponding manual netX *xxxx Programming Reference Guide*.

- netX 500/100/50
  The register block is located at the end of the DPM
- netX 51/52
  After power-up without any firmware (ROM loader only), the register block is mapped to the beginning of the DPM. An SSBL / a firmware will relocate the register block at the end of the DPM.

Reset pattern

To execute a netX chip reset, a reset pattern has to be written to the netX reset register.

```
Reset pattern:
0x00000000, 0x00000001,0x00000003, 0x00000007, 0x0000000F, 0x0000001F,0x0000003F,
0x0000007F, 0x000000FF
```

This pattern must be written to the netX register DPM_HOST_RESET_REQ. By writing the last value, the chip reset will be activated automatically.

Write Reset pattern

```
DWORD tResetPatttern[9] = { 0x00000000, 0x00000001, 0x00000003, 0x00000007, 0x0000000F,
                            0x0000001F, 0x0000003F, 0x0000007F, 0x000000FF}

DWORD* pDPM_HOST_RESET_REQ = DPM_HOST_RESET_REQ

for (ulIdx = 0; ulIdx < sizeof(tResetPattern); ++ulIdx)
{
  *pDPM_HOST_RESET_REQ = tResetPatttern[iIdx];
}
```

DPM_HOST_SYS_STA can be used to detect the end of the chip reset.

Wait Reset done

```
DWORD* pDPM_HOST_SYS_STA = DPM_HOST_SYS_STA

while(1)
{
  /* Check if state */
  if( (0xFFFFFFFF  != *pDPM_HOST_SYS_STA) &&
      (0x0BAD0BAD  != *pDPM_HOST_SYS_STA))
  {
    /* Reset done */
    break;
  }
}
```

3.1.2 System reset

A system reset is executed by an SSBL, the MFW or protocol firmware.

The reset will be initiated if the application writes a 0x55AA55AA pattern (system reset cookie) to the *ulSystemCommandCOS* variable in the system control block and sets flag HSF_RESET in *bHostFlags.*

If the firmware recognizes the reset, flag NSF_READY in *bNetxFlags* will be cleared, indicating that a systemwide reset will be processed and the entire DPM will be initialized with zeros.

An application has to use the handling To detect the end of the reset, see section *Check firmware initialization* on page 9.

Value	Definition / Description
0x55AA55AA	HIL_SYS_RESET_COOKIE

Table 4: System reset cookie

System reset - flowchart

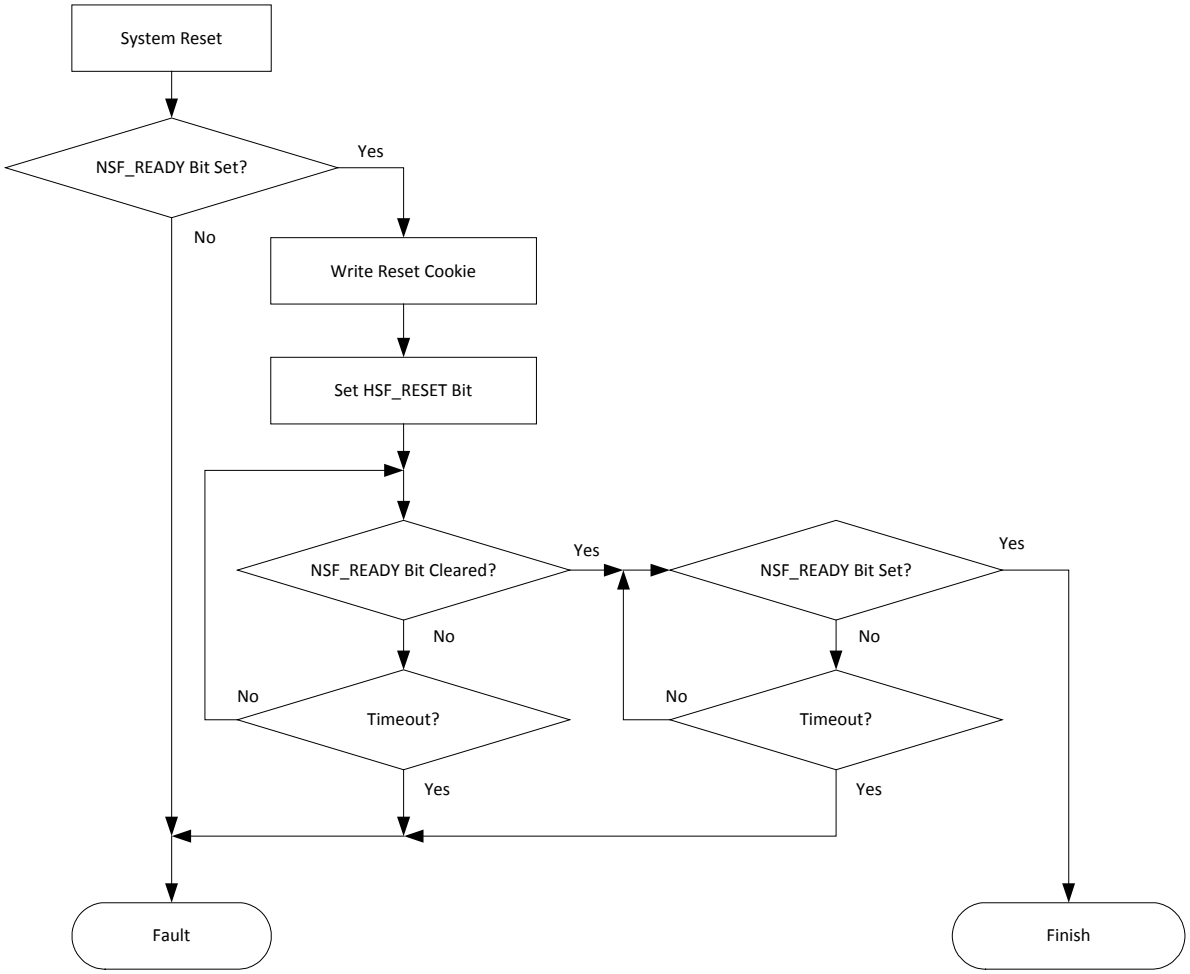

Figure 2: System reset - flowchart

System reset procedure

- Check if flag NSF_READY is set in *bNetXFlags*.
- Write the reset cookie 0x55AA55AA (HIL_SYS_RESET_COOKIE) to the *ulSystemCommand*COS variable in the system control block (offset 0x00B8)
- Set flag HSF_RESET in *bHostFlags*.
- Wait until flag NSF_READY in *bNetXFlags* is cleared (reset operation in progress).
- Process the handling. Check firmware initialization.

Timing consideration

The duration of the reset outlined above, depends on the firmware. Typically, flag NSF_READY will be cleared within 100 ms to 500 ms after setting flag HSF_RESET.

When cleared, flag NSF_READY will be set again after about 0.5 – 10 s.

3.1.3 Boot Start

If the SSBL / MFW does not start any available firmware, the Boot Start option can be used by setting flag HSF_BOOTSTART flag together with flag HSF_RESET.

The SSBL / MFW will ignore an existing firmware and stay in the so-called maintenance mode.

Note: Boot Start works only on FLASH based devices, where all files are stored in a FLASH. On a netX target without FLASH (RAM-based), the firmware starts over without activating the maintenance mode.

Boot Start - flowchart

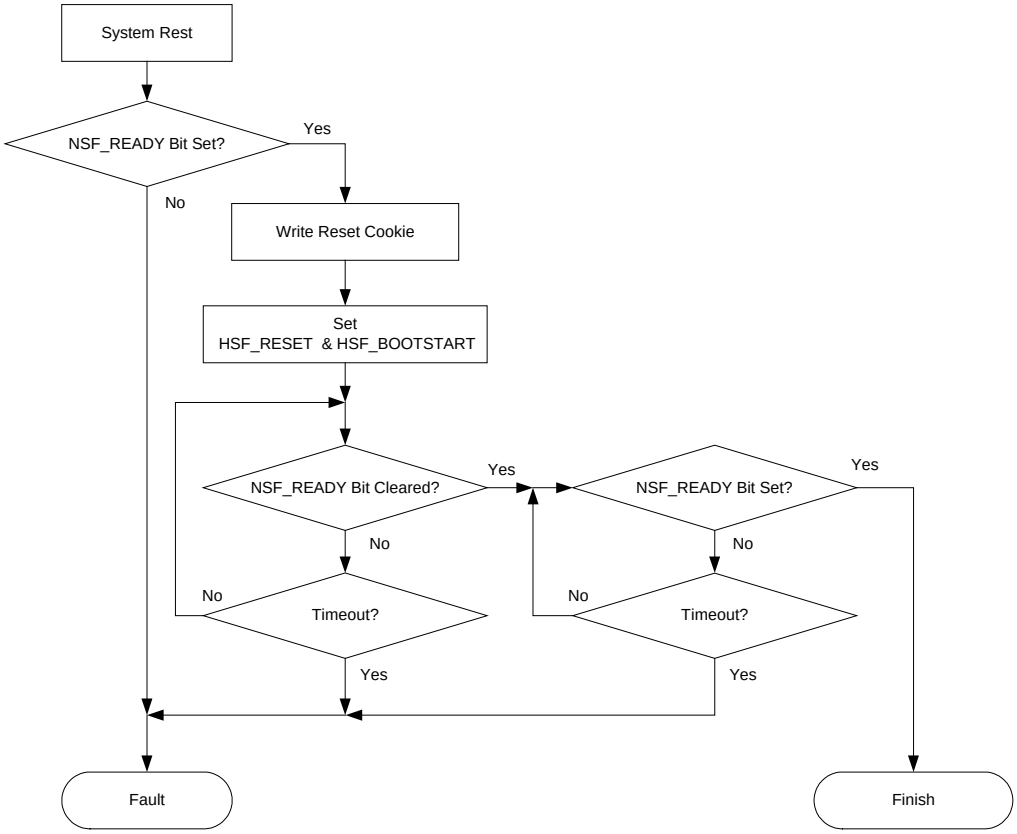

Figure 3: Boot Start - Flowchart

Boot Start procedure

- Check if flag NSF_READY is set in *bNetXFlags*.
- Write the reset cookie 0x55AA55AA (HIL_SYS_RESET_COOKIE) to the *ulSystemCommand*COS variable in the system control block (offset 0x00B8)
- Set flag HSF_RESET in *bHostFlags*.
- Wait until flag NSF_READY in *bNetXFlags* is cleared (reset operation in progress).
- Process the handling, see section *Check firmware initialization* on page 9.

.

3.1.4 Update Start

Update Start is a special option for netX 90/4x00-based systems because these are Flash-based and an update is possible only by starting an integrated Maintenance Firmware (MFW).

The MFW is responsible for checking the available files. In case of a changed firmware or different boot parameters, it will install the corresponding firmware and and restart the netX system.

Update Start is activated by writing additional boot options to the system control block (offset 0x00B8) variable *ulSystemControl* followed by a standard System reset.

ulSystemControl is located at DPM offset 0x00BC and the options are defined in the *netX DPM Interface Manual*.

Note: Update Start only works on Flash-based systems where the firmware is stored in a Flash. On a netX target without a Flash (i.e. RAM-based systems), the firmware starts over without activating the MFW.

3.2 Communication channel

3.2.1 Channel initialization

Channel initialization is used to restart a protocol stack running behind a communication channel, without resetting the whole device. The communication channel will immediately interrupt all open network connections and close them regardless of their current state.

Channel initialization - flowchart

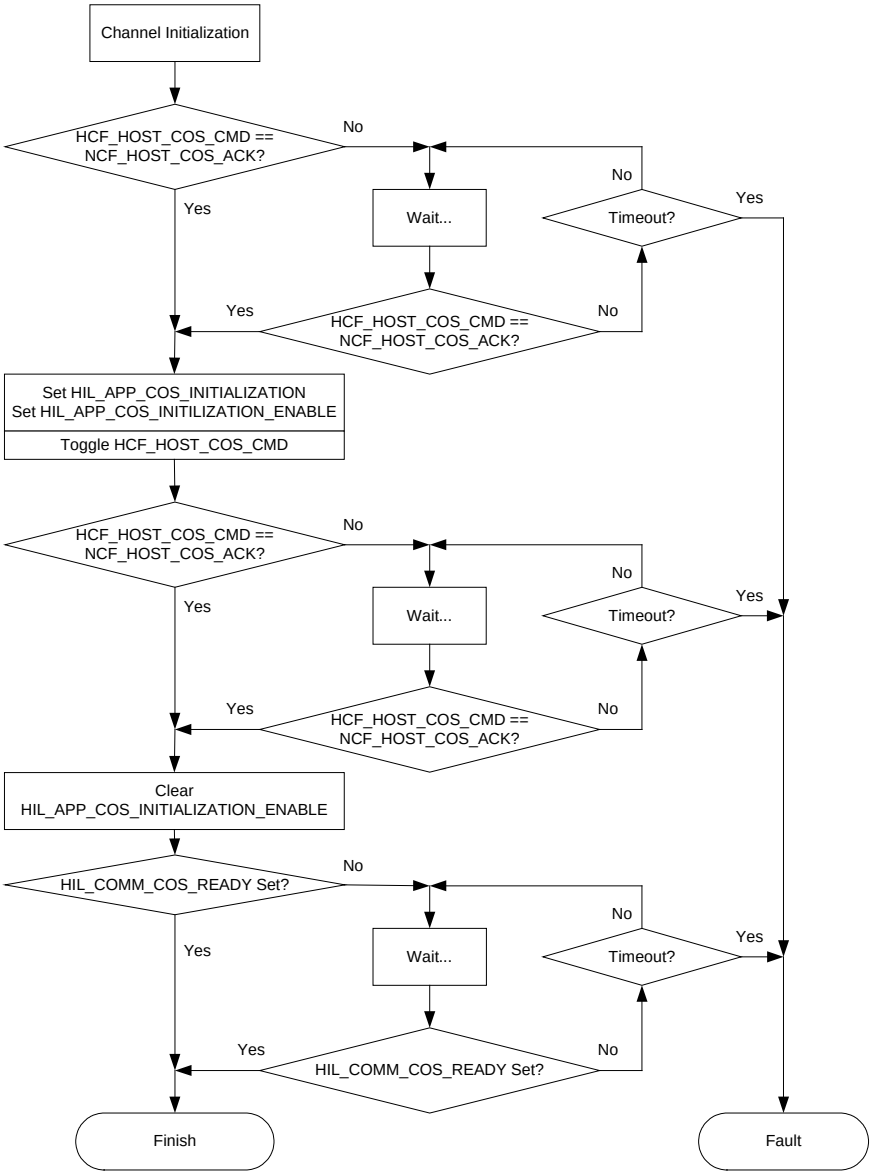

Figure 4: Channel iInitialization - flowchart

Channel initialization procedure

Note: If the protocol stack sets config lock flag HIL_COMM_COS_CONFIG_LOCKED (see *ulCommunicationCOS*), a reinitialization of a communication channel is not allowed and will not be processed by the protocol stack.

- Check if write access to the host COS flags is available by comparing the handshake flags HCF_HOST_COS_CMD == NCF_HOST_COS_ACK (see *usHostFlags* / *usNetXFlags*) in the communication flags of the channel. If both flags are equal, write access is allowed.
- Set flags HIL_APP_COS_INITIALIZATION and HIL_APP_COS_INITIALIZATION_ENABLE in register *ulApplicationCOS* of the common control block.
- Signal the new COS state to the communication channel by toggling flag HCF_HOST_COS_CMD in *usHostFlags*.
- Wait until flags HIL_COMM_COS_READY and HIL_COMM_COS_RUN in register *usCommunicationCOS* of the common status block are cleared.
  HIL_COMM_COS_READY will stay cleared for about 20 ms.
- Check if the protocol stack signals new COS state changes by comparing the handshake flags NCF_NETX_COS_CMD != HCF_NETX_COS_ACK in *usNetXFlags* / *usHostFlags* and acknowledge each new COS change of the communication channel to make sure that the channel can proceed with the initialization and is able to signal state changes.
- If flag HIL_COMM_COS_READY is set, the initialization was successful.
- Proceed with further application settings just like you started the first time (e.g. BUS On / Configuration Lock / DMA on, etc.)

Note: To prevent the same command from being processed again, make sure to clear flag HIL_APP_COS_INIT_ENABLE in *ulApplicationCOS* before signaling further application settings.

3.3 Start / stop communication

A communication channel (protocol stack) offers a function to start and stop the bus communication by an application.

The fieldbus configuration determines the general protocol stack behavior and defines whether the stack is allowed to start the bus communication automatically (as soon as the configuration is loaded) or has to wait for the application to release the communication bus.

Possible start-up configuration settings (see SYCON.net):

Start of bus communication	Description
Automatically by device	The fieldbus communication starts as soon as the protocol stack has loaded and evaluated the configuration.
Controlled by application	The protocol stack loads and evaluates the configuration bus, and waits for the application to start it.

The application developer decides on which method to use. In general, the controlled start method allows a better control of the network communication irrespective of whether a master or slave protocol is used.

The protocol stack offers the HIL_COMM_COS_BUS_ON flag in the *ulCommunicationCOS* field to indicate if the bus communication is enabled or not.

State flag	Value	State
HIL_COMM_COS_BUS_ON	0	Bus communication is disabled
	1	Bus communication is enabled

3.3.1 (Re-)start communication

To allow the protocol stack to open network connections, if configured with "Controlled by application", the application has to signal the Bus On flag (HIL_COMM_COS_BUS_ON = 1) in register *ulApplicationCOS* located in the common status block of the channel. This also implies the COS handling defined for application commands.

If the protocol stack has established a cyclic connection to at least one network device, the general communication flag NCF_COMMUNICATING (in the netX Communication Flags register *usNetXFlags*) will signal this status.

3.3.2 Stop communication

To force a protocol stack to disable all network connections, the host application can send a Bus Off command to the communication channel by setting HIL_COMM_COS_BUS_ON = 0 in register *ulApplicationCOS* and executing the COS handling defined for application commands.

The firmware then closes all open network connections and clears flag NCF_COMMUNICATING. All further attempts to re-open a connection will be rejected until Bus On is set again (sending HIL_COMM_COS_BUS_ON = 1).

Start communication procedure

- Check if write access to the host COS flags is available by comparing the handshake flags HCF_HOST_COS_CMD == NCF_HOST_COS_ACK in the communication flags of the channel. If both flags are equal, write access is allowed.
- In the *ulApplicationCOS* field, the application sets HIL_APP_COS_BUS_ON (= 1) to start the bus communication. In addition, flag HIL_APP_COS_BUS_ON_ENABLE must be set to signal that the command is active.
- The new host state (command) is signaled to the communication channel by toggling flag HCF_HOST_COS_CMD in `usHostFlags`.
- The firmware updates flag HIL_COMM_COS_BUS_ON in the `ulCommunicationCOS` field, acknowledges the command by toggling flag NCF_HOST_COS_ACK, and starts the bus communication. If the protocol stack is able to establish a cyclic data connection to at least one device of the network, the firmware also sets flag NCF_COMMUNICATING in `usNetXFlags`.
- If the application recognizes the acknowledgement flag NCF_HOST_COS_ACK equal to HCF_HOST_COS_CMD, it should reset flag HIL_APP_COS_BUS_ON_ENABLE in the `ulApplicationCOS` field to prevent the command from being signaled repeatedly when another state is sent to the communication channel.

Stop communication procedure

The sequence to stop the bus communication is similar to the start sequence except that flag HIL_APP_COS_BUS_ON is set to 0.

- In the *ulApplicationCOS* field, the application clears HIL_APP_COS_BUS_ON (= 0) to stop the bus communication. In addition, flag HIL_APP_COS_BUS_ON_ENABLE must be set to signal that the command is active.

The firmware updates flag HIL_COMM_COS_BUS_ON, closes all open network connections, and clears flag NCF_COMMUNICATING in the netX communication flags as soon as all connections are closed.

A slave protocol stack rejects attempts to re-open a connection until the bus is switched on again.

3.4 Lock/unlock configuration

The ‘lock/unlock configuration’ mechanism prevents the configuration settings from being deleted, altered, overwritten or changed during run-time. Any configuration tool has to reject such attempts when the ‘configuration locked’ flag is set.

The configuration of a channel firmware can be locked and unlocked via direct access to the DPM or by passing a packet through the channel mailbox.

Lock/unlock configuration - flowchart

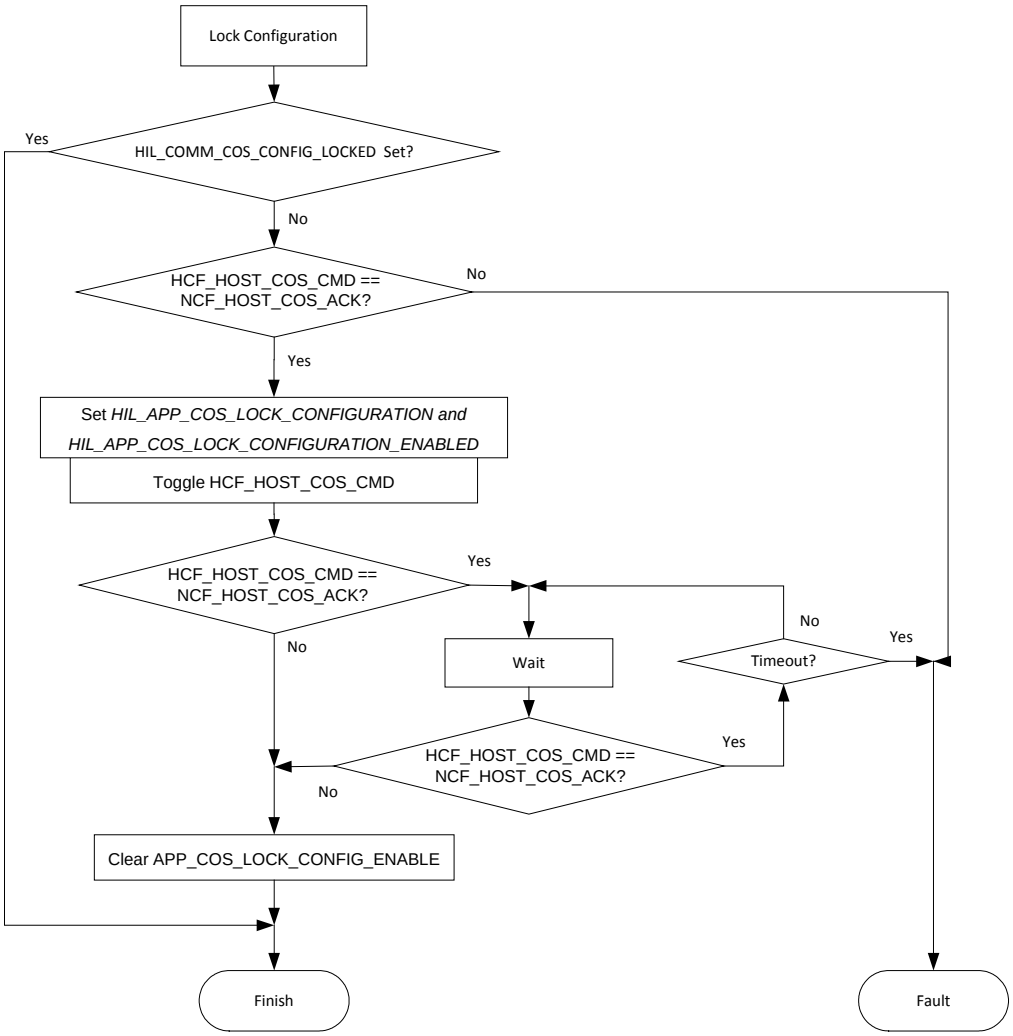

Figure 5: Lock configuration - flowchart

Lock configuration procedure

- Check if write access to the host COS flags in *ulApplicationCOS* is available by comparing the handshake flags HCF_HOST_COS_CMD and NCF_HOST_COS_ACK in the communication flags (*usHostFlags* / *usNetXFlags*) of the channel.
  If both flags are equal, write access is allowed.
- Set flag HIL_APP_COS_LOCK_CONFIGURATION together with flags HIL_APP_COS_LOCK_CONFIGURATION_ENABLE in register *ulApplicationCOS*.
- Process an application COS handling by toggling flag HCF_HOST_COS_CMD in the host communication flags *usHostFlags*. This signals a communication channel that a new COS state/command is available.
- The communication channel updates flag HIL_COMM_COS_CONFIG_LOCKED in field *ulCommunicationCOS* and indicates that the configuration settings are locked now.
- Clear flag HIL_APP_COS_LOCK_CONFIGURATION_ENABLE in field *ulApplicationCOS* to reduce possible errors while enabling other commands.

Unlock configuration procedure

The sequence to unlock the configuration is similar to the lock configuration sequence. The only difference is in clearing flag HIL_APP_COS_LOCK_CONFIGURATION instead of setting it.

- Clear flag HIL_APP_COS_LOCK_CONFIGURATION and set flag HIL_APP_COS_LOCK_CONFIGURATION_ENABLE in register *ulApplicationCOS*.

The firmware will update flag HIL_COMM_COS_CONFIG_LOCKED to indicate that configuration changes are allowed.

3.5 Watchdog handling

The watchdog detects malfunctions of the host application and/or the communication channel firmware.

The watchdog can be enabled independently for each communication channel and is based on two fields:

- *ulDeviceWatchdog* located in the common control block
- *ulHostWatchdog* located in the common status block

The protocol stack configuration defines the timeout value for the watchdog. The current value can be read from the *usWatchdogTime* field located in the common status block.

Min. configurable timeout: 20 ms. Default value set by the configuration software: 1000 ms.

From the perspective of the application, watchdog handling is a simple copy function. The content of field *ulDeviceWatchdog* must be copied to field *ulHostWatchdog*.

The watchdog timeout is configurable in SYCON.net or during a protocol stack configuration via packet services.

There is no watchdog function for the system channel or for the handshake channel.

Watchdog processing by a host application:

- To activate and trigger the watchdog, copy the value *ulHostWatchdog* to the field *ulDeviceWatchdog*.
- To deactivate the watchdog supervision by the firmware, write 0 to *ulDeviceWatchdog.* The firmware reinitializes *ulHostWatchdog* to *0x00000001.*

The host application is able to supervise the communication channel by checking the value of *ulHostWatchdog* because the netX firmware will always increment the content after the application has processed the data copy and the firmware has detected that *ulDeviceWatchdog* is equal to *ulHostWatchdog*.

3.6 I/O data exchange via DMA

The feature Direct Memory Access (DMA) is available on PCI-based hardware only and allows a netX-based PCI hardware to act as a bus master DMA controller.

Note: Only a PCI-based hardware equipped with netX 100/500 controllers supports the DMA. A netX firmware must support this feature. By default, I/O data transfer is possible only via DMA.

Note: DMA can reduce the workload of the host CPU and speed up data transfer between host and netX hardware. However, this depends on the amount of data transferred and is not recommended for data sizes < 256 Bytes per transfer.

In DMA mode, the netX hardware handles the I/O data transfers actively reading/writing the data from/to the memory of the host system. For this purpose, the host system has to provide DMA buffers and has to program the netX DMA interface with the address and size of each buffer.

Once the netX DMA interface is programmed, the host application can switch to DMA mode during runtime. The *Application COS* flags HIL_APP_COS_DMA/HIL_APP_COS_DMA_ENABLE control the DMA mode switching.

The communication handshake flags PDx_IN_CMD/ACK, PDx_OUT_CMD/ACK (described in the netX DPM Interface Manual) still handle the data exchange synchronization. An explicit data copy from and to the DPM I/O data areas is unnecessary because it is copied by the netX hardware.

The netX supports 8 DMA channels which have a fixed assignment to the input/output areas of the communication channels, see Table 5.

Communication channel	DMA channel	Data image
0	0	Input data
	1	Output data
1	2	Input data
	3	Output data
2	4	Input data
	5	Output data
3	6	Input data
	7	Output data

Table 5: DMA Channel Assignment

Note: DMA handling is not offered as a general netX feature and has several restrictions (e.g. netX chip type / hardware connection to the host / data transfer only as a whole / creation of the host DMA buffers / netX chip initialization).

Contact Hilscher if you want to use the DMA feature.

4 Examples

4.1 System identification and start-up handling

Figure 6 shows the start-up handling to identify a running netX boot loader or firmware.

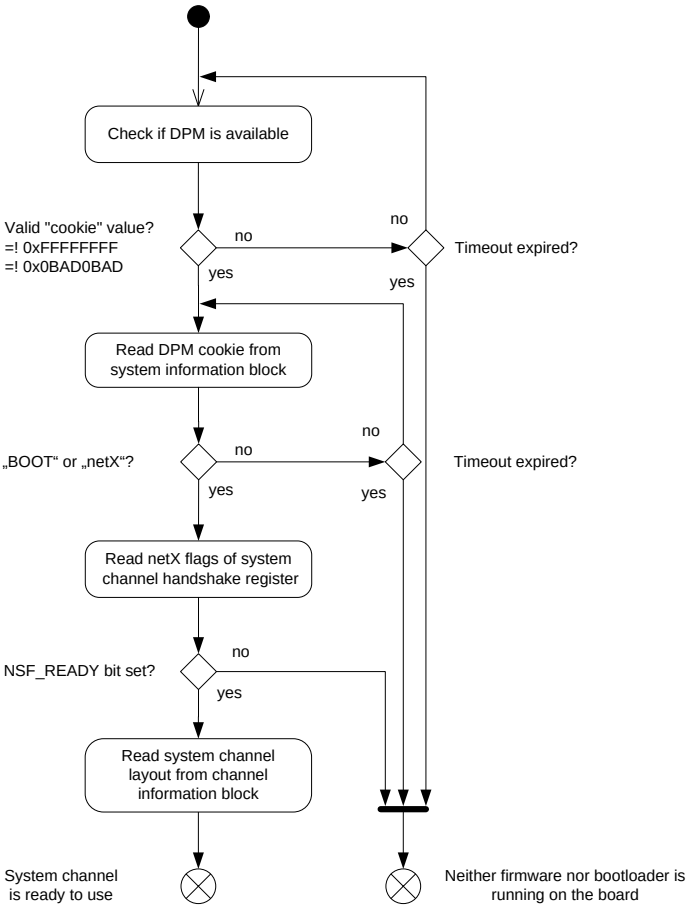

Figure 6: System identification and start-up handling

4.2 Channel identification and start-up handling

After a successful identification of the system start-up, the application can use a static layout to access the DPM. However, it is also possible to read the DPM layout and to adapt the application to a dynamic DPM layout (optional).

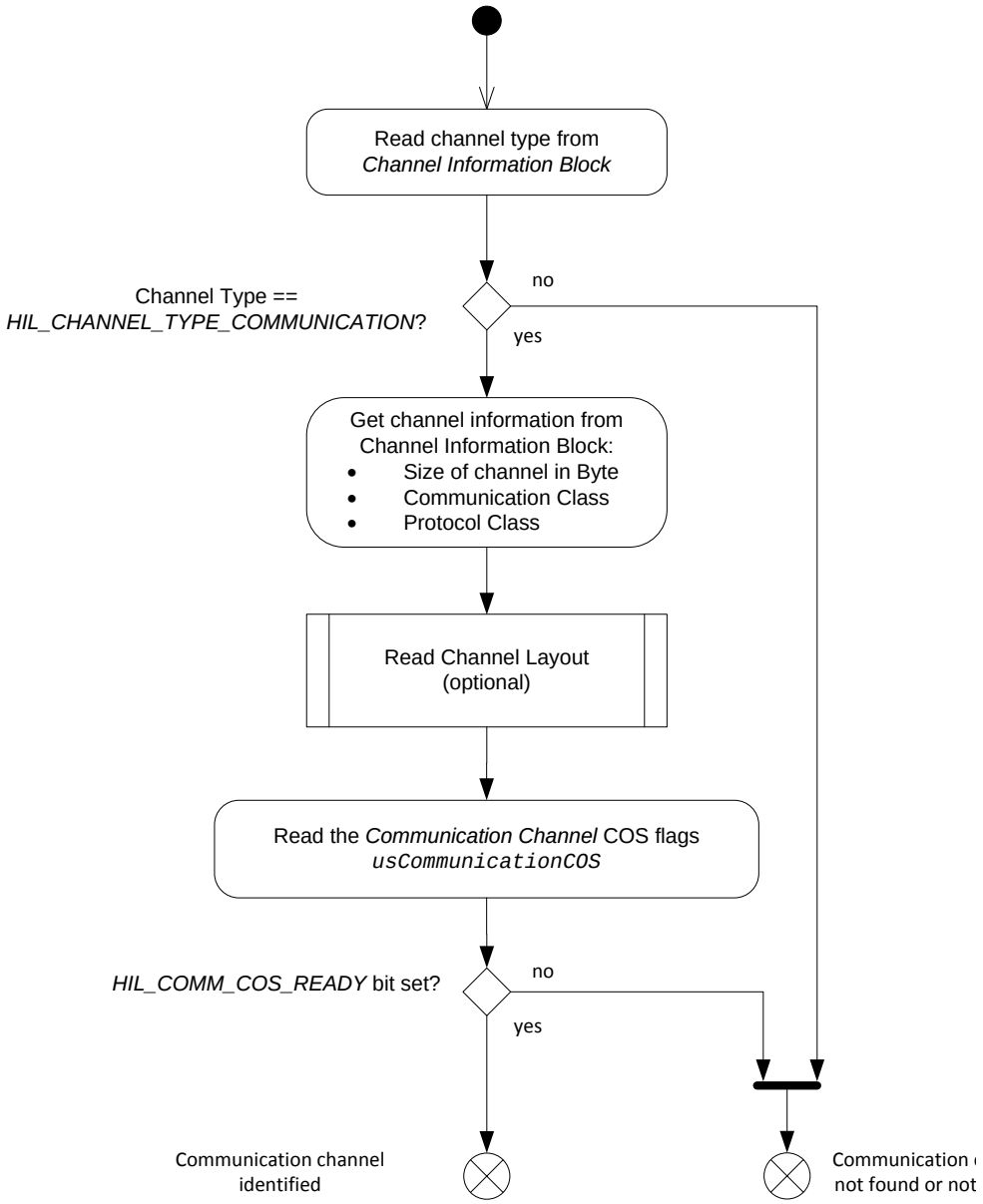

Figure 7: Channel identification and start-up handling

Note:	Reading the COS flags also requires a COS flag handling to enable the communication channel to update the current COS states.

4.3 Packet transfer

4.3.1 Sending a packet to the device

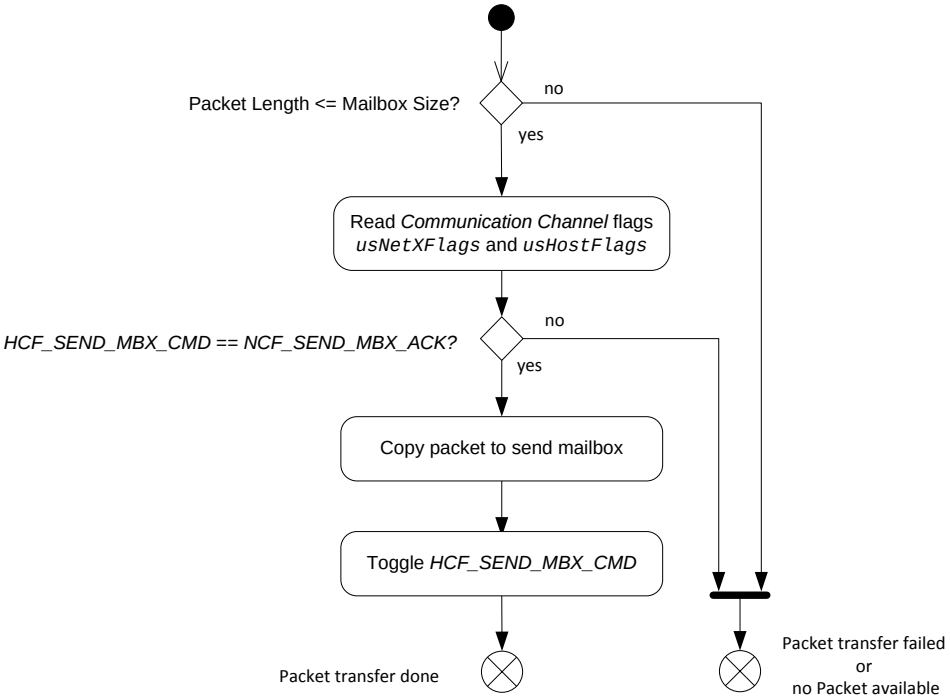

Figure 8: Sending a packet to the device

4.3.2 Reading a packet from the device

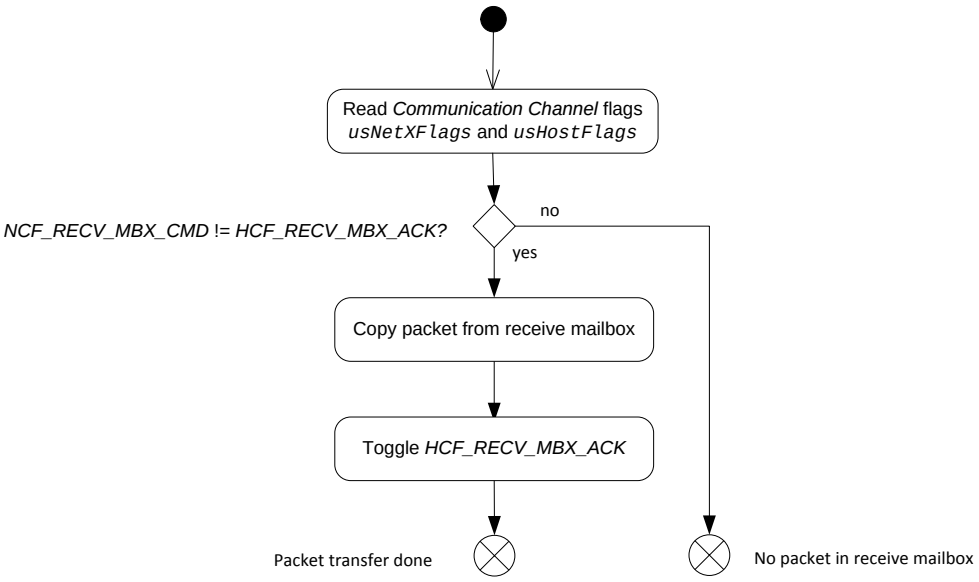

Figure 9: Reading a packet from the device

4.4 I/O data transfer

4.4.1 Writing output data to the device

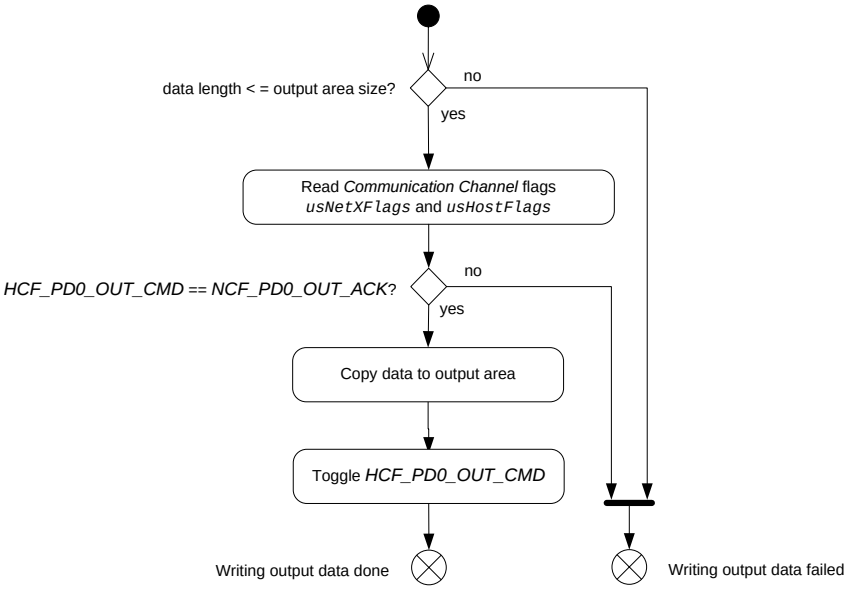

Figure 10: Writing output data to the device

Note:	This example shows the handling if the communication channel is configured in the so-called "buffered host-controlled" I/O data transfer mode. The logic of checking whether access to the output area is allowed, will change if another I/O data transfer mode is used.

The currently configured I/O data transfer mode can be read from the common status block *bPDOutHskMode* field.

4.4.2 Reading input data from the device

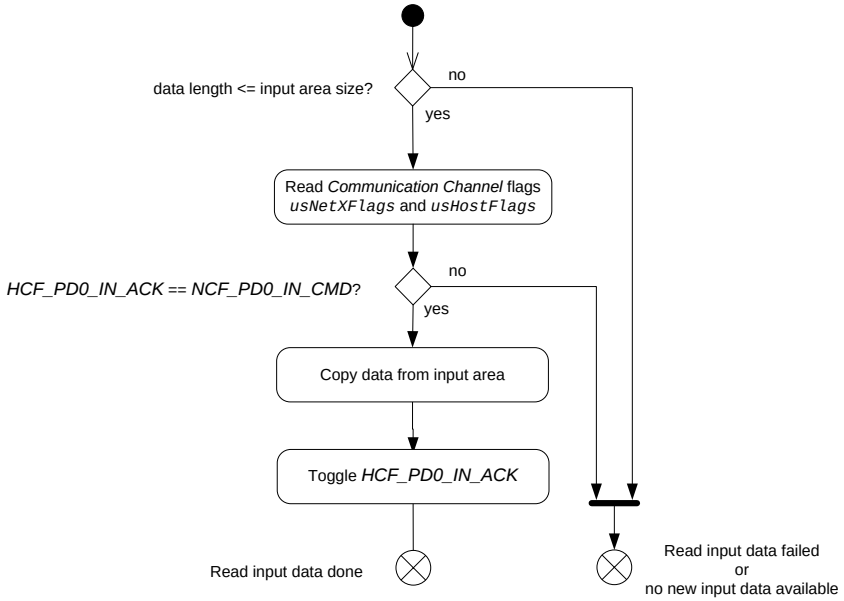

Figure 11: Reading input data from the device

Note:	This example shows the handling if the communication channel is configured in the so-called "buffered host-controlled" I/O data transfer mode. The logic of checking whether access to the output area is allowed, will change if another I/O data transfer mode is used.

The currently configured I/O data transfer mode can be read from the common status block *bPDInHskMode* field.

4.5 Change of State (COS) handling

4.5.1 Host COS handling

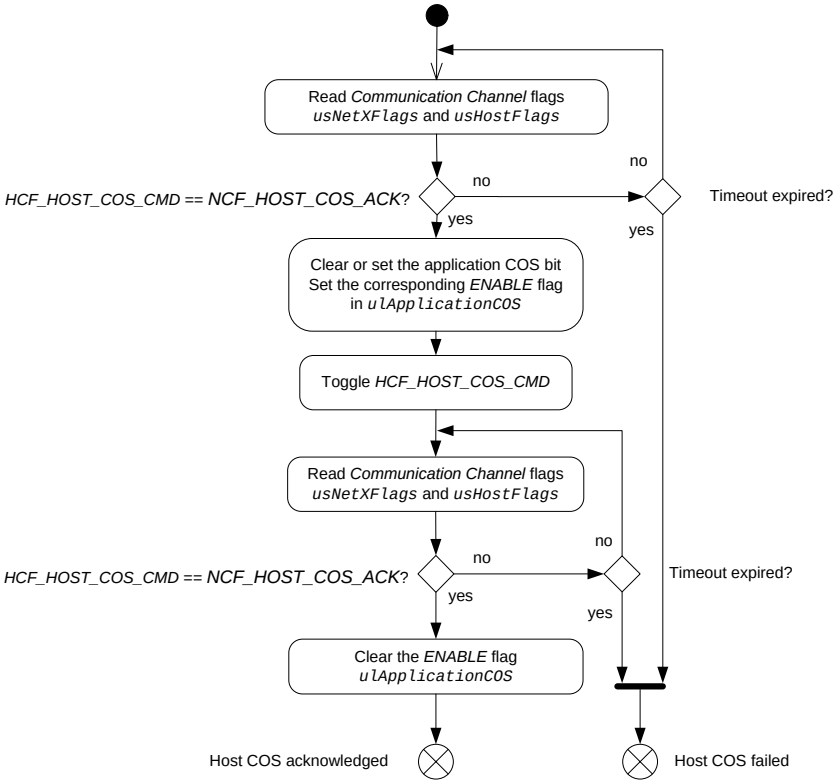

Figure 12: Host COS handling

4.5.2 Communication COS handling

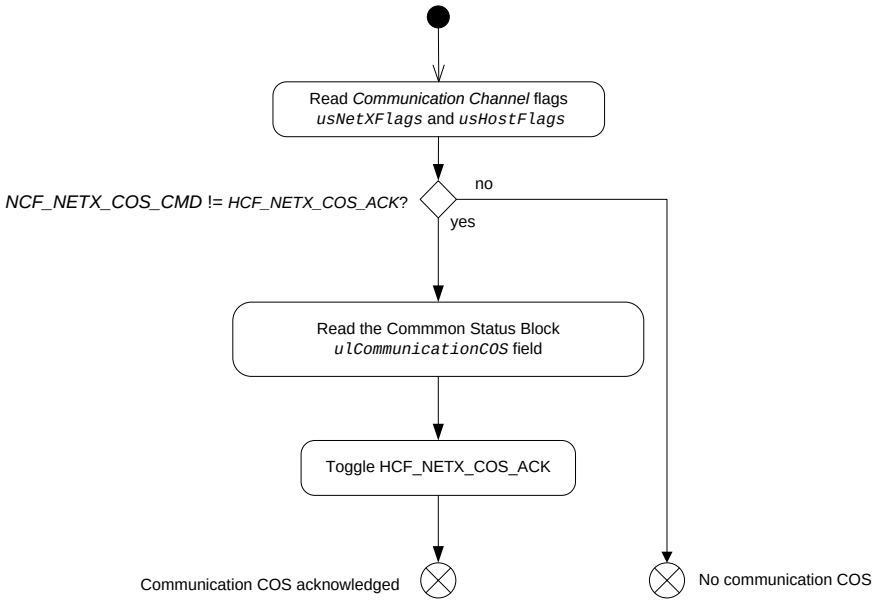

Figure 13: Communication COS handling

4.6 Hardware and protocol stack identification

4.6.1 General system information

The general system information can be read from the system information block.

- General system information
  System Information block => `ulDeviceNumber`
  => `ulSerialNumber`
  => `usManufacturer`
  => `usProductionDate`
  => `ulLicenseFlags1`
  => `ulLicenseFlags2`
  => `bHwRevision`

This information is useful for device-specific questions and support requests.

4.6.2 Hardware identification and options

For information on hardware description and hardware options (hardware-supported protocols), see the system information block.

- Hardware identification
  defines the type of netX chip / Hilscher device
  Device class => DPM Offset 0x0028
  System information block => *`usDeviceClass`* field
- Hardware assembly option
  defines the communication hardware to be offered by the device
  Hardware option => DPM Offset 0x0010
  System information block => *ausHwOptions[4]* field

4.6.3 Protocol stack identification

- Protocol stack identification
  The protocol stack identification per communication channel can be found in:
  Channel information block => *bChannelType* => *usCommunicationClass*
  Check if *bChannelType* channel type == *COMMUNICATION*
  Read the communication class *`usCommunicationClass`* and *`usProtocolClass`*

The protocol stack information may become necessary if the host application has

- to support multiple protocol stacks and
- to check which protocol stack is currently active.

5 Additional resources

5.1 cifX/netX toolkit

Hilscher offers a C-based toolkit offering an abstract API (called CIFX-API) which handles the netX DPM in a standard way. The toolkit code is open source and can be used as an example or as the basis for a user application.

Link to the CIFX-API: https://kb.hilscher.com/pages/viewpage.action?pageId=22808198

Link to the cifX/netX toolkit: https://kb.hilscher.com/pages/viewpage.action?pageId=22808198

5.2 Protocol API manuals

Each fieldbus protocol stack has its own documentation describing protocol-specific functions and commands.

You will find Protocol API manuals on the Hilscher "**Communication Solution DVD**" (see **.\Documentation\Programming Manuals** directory), delivered with most of the Hilscher devices or in the Hilscher knowledgebase.

Link to the Protocol API manuals: https://kb.hilscher.com/display/HILKB/Technologies

6 Appendix

6.1 List of figures

6.2 List of tables

6.3 Legal notes

Copyright

Important notes

Utmost care was/is given in the preparation of the documentation at hand consisting of a user's manual, operating manual and any other document type and accompanying texts. However, errors cannot be ruled out. Therefore, we cannot assume any guarantee or legal responsibility for erroneous information or liability of any kind. You are hereby made aware that descriptions found in the user's manual, the accompanying texts and the documentation neither represent a guarantee nor any indication on proper use as stipulated in the agreement or a promised attribute. It cannot be ruled out that the user's manual, the accompanying texts and the documentation do not completely match the described attributes, standards or any other data for the delivered product. A warranty or guarantee with respect to the correctness or accuracy of the information is not assumed.

We reserve the right to modify our products and the specifications for such as well as the corresponding documentation in the form of a user's manual, operating manual and/or any other document types and accompanying texts at any time and without notice without being required to notify of said modification. Changes shall be taken into account in future manuals and do not represent an obligation of any kind, in particular there shall be no right to have delivered documents revised. The manual delivered with the product shall apply.

Under no circumstances shall Hilscher Gesellschaft für Systemautomation mbH be liable for direct, indirect, ancillary or subsequent damage, or for any loss of income, which may arise after use of the information contained herein.

Liability disclaimer

The hardware and/or software was created and tested by Hilscher Gesellschaft für Systemautomation mbH with utmost care and is made available as is. No warranty can be assumed for the performance or flawlessness of the hardware and/or software under all application conditions and scenarios and the work results achieved by the user when using the hardware and/or software. Liability for any damage that may have occurred as a result of using the hardware and/or software or the corresponding documents shall be limited to an event involving willful intent or a grossly negligent violation of a fundamental contractual obligation. However, the right to assert damages due to a violation of a fundamental contractual obligation shall be limited to contract-typical foreseeable damage.

It is hereby expressly agreed upon in particular that any use or utilization of the hardware and/or software in connection with

- Flight control systems in aviation and aerospace;
- Nuclear fission processes in nuclear power plants;
- Medical devices used for life support and
- Vehicle control systems used in passenger transport

shall be excluded. Use of the hardware and/or software in any of the following areas is strictly prohibited:

- For military purposes or in weaponry;
- For designing, engineering, maintaining or operating nuclear systems;
- In flight safety systems, aviation and flight telecommunications systems;
- In life-support systems;
- In systems in which any malfunction in the hardware and/or software may result in physical injuries or fatalities.

You are hereby made aware that the hardware and/or software was not created for use in hazardous environments, which require fail-safe control mechanisms. Use of the hardware and/or software in this kind of environment shall be at your own risk; any liability for damage or loss due to impermissible use shall be excluded.

Warranty

Hilscher Gesellschaft für Systemautomation mbH hereby guarantees that the software shall run without errors in accordance with the requirements listed in the specifications and that there were no defects on the date of acceptance. The warranty period shall be 12 months commencing as of the date of acceptance or purchase (with express declaration or implied, by customer's conclusive behavior, e.g. putting into operation permanently).

The warranty obligation for equipment (hardware) we produce is 36 months, calculated as of the date of delivery ex works. The aforementioned provisions shall not apply if longer warranty periods are mandatory by law pursuant to Section 438 (1.2) BGB, Section 479 (1) BGB and Section 634a (1) BGB [Bürgerliches Gesetzbuch; German Civil Code] If, despite of all due care taken, the delivered product should have a defect, which already existed at the time of the transfer of risk, it shall be at our discretion to either repair the product or to deliver a replacement product, subject to timely notification of defect.

The warranty obligation shall not apply if the notification of defect is not asserted promptly, if the purchaser or third party has tampered with the products, if the defect is the result of natural wear, was caused by unfavorable operating conditions or is due to violations against our operating regulations or against rules of good electrical engineering practice, or if our request to return the defective object is not promptly complied with.

Costs of support, maintenance, customization and product care

Please be advised that any subsequent improvement shall only be free of charge if a defect is found. Any form of technical support, maintenance and customization is not a warranty service, but instead shall be charged extra.

Additional guarantees

Although the hardware and software was developed and tested in-depth with greatest care, Hilscher Gesellschaft für Systemautomation mbH shall not assume any guarantee for the suitability thereof for any purpose that was not confirmed in writing. No guarantee can be granted whereby the hardware and software satisfies your requirements, or the use of the hardware and/or software is uninterruptable or the hardware and/or software is fault-free.

It cannot be guaranteed that patents and/or ownership privileges have not been infringed upon or violated or that the products are free from third-party influence. No additional guarantees or promises shall be made as to whether the product is market current, free from deficiency in title, or can be integrated or is usable for specific purposes, unless such guarantees or promises are required under existing law and cannot be restricted.

Confidentiality

The customer hereby expressly acknowledges that this document contains trade secrets, information protected by copyright and other patent and ownership privileges as well as any related rights of Hilscher Gesellschaft für Systemautomation mbH. The customer agrees to treat as confidential all of the information made available to customer by Hilscher Gesellschaft für Systemautomation mbH and rights, which were disclosed by Hilscher Gesellschaft für Systemautomation mbH and that were made accessible as well as the terms and conditions of this agreement itself.

The parties hereby agree to one another that the information that each party receives from the other party respectively is and shall remain the intellectual property of said other party, unless provided for otherwise in a contractual agreement.

The customer must not allow any third party to become knowledgeable of this expertise and shall only provide knowledge thereof to authorized users as appropriate and necessary. Companies associated with the customer shall not be deemed third parties. The customer must obligate authorized users to confidentiality. The customer should only use the confidential information in connection with the performances specified in this agreement.

The customer must not use this confidential information to his own advantage or for his own purposes or rather to the advantage or for the purpose of a third party, nor must it be used for commercial purposes and this confidential information must only be used to the extent provided for in this agreement or otherwise to the extent as expressly authorized by the disclosing party in written form. The customer has the right, subject to the obligation to confidentiality, to disclose the terms and conditions of this agreement directly to his legal and financial consultants as would be required for the customer's normal business operation.

Export provisions

The delivered product (including technical data) is subject to the legal export and/or import laws as well as any associated regulations of various countries, especially such laws applicable in Germany and in the United States. The products / hardware / software must not be exported into such countries for which export is prohibited under US American export control laws and its supplementary provisions. You hereby agree to strictly follow the regulations and to yourself be responsible for observing them. You are hereby made aware that you may be required to obtain governmental approval to export, reexport or import the product.

6.4 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