



Driver Manual
cifX Device Driver
Linux (Kernel 2.6.x)
V1.0.x.x

Hilscher Gesellschaft für Systemautomation mbH

www.hilscher.com

DOC090201DRV04EN | Revision 4 | English | 2010-07 | Released | Public

Table of Contents

1	Introduction.....	3
1.1	About this Document.....	3
1.2	List of Revisions	3
1.3	Overview	4
1.4	Requirement.....	5
1.5	Features	5
1.6	Limitations	5
1.7	CD Contents.....	6
1.8	Terms, Abbreviations and Definitions	6
1.9	References	6
1.10	Legal Notes	7
1.10.1	Copyright.....	7
1.10.2	Important Notes.....	7
1.10.3	Exclusion of Liability	8
1.10.4	Export.....	8
2	Licensing Terms	9
3	Installation.....	10
3.1	Prerequisites	10
3.2	Compilation of the netX UIO Kernel Module	11
3.2.1	Compilation of the UIO Kernel Module via console	12
3.3	Compilation of the cifX Userspace Library.....	13
3.3.1	Compilation of the cifX Userspace Library via console.....	13
3.3.2	Compilation of the cifX Userspace Library via IDE	14
3.4	Compilation of the cifX Example Program	16
3.4.1	Compilation of the cifX Example Program via Console	16
3.4.2	Compilation of the cifX Example Program via IDE	17
3.5	Loading netX UIO Driver Module	18
3.6	Firmware and Configuration File Storage	19
3.6.1	Device Identification via Single Directory.....	20
3.6.2	Device Identification via Device and Serial Number	21
3.6.3	Device Identification via Slotnumber (rotary switch)	22
4	Linux Driver Specific Information.....	23
4.1	Additional Structures	23
4.1.1	Structure CFX_LINUX_INIT	23
4.1.2	Structure CFX_DEVICE_T	24
4.2	Additional Functions	25
4.2.1	cfXDriverInit().....	25
4.2.2	cfXDriverDeinit()	25
4.2.3	cfXGetDriverVersion().....	26
4.2.4	cfXGetDeviceCount().....	26
4.2.5	cfXFindDevice().....	27
4.2.6	cfXDeleteDevice().....	27
4.3	Driver/Library Startup Procedure	28
4.4	Device Configuration (device.conf)	29
5	Using SYCON.net to Configure the Fieldbus System.....	30
6	Programming with the cifX Linux Driver.....	31
7	Appendix	32
7.1	List of Tables	32
7.2	List of Figures.....	32
7.3	Contacts	33

1 Introduction

1.1 About this Document

This manual describes the Hilscher cifX driver for Linux and its architecture.

The driver offers access to the Hilscher netX based hardware (e.g. CIFX50) with the same functional API as the cifX device driver for Windows.

1.2 List of Revisions

Rev	Date	Name	Chapter	Revision
1	2008-11-13	RM/MT	all	Created
2	2009-01-26	MT		prerequisites updated (added pkg-config) Updated to netX Toolkit 0.941 (Big Endian and netX50 support included) Structure CIFX_DEVICE_T updated PCI support can now be disabled by configuration switch
3	2009-10-02	MT		Updated to netX Toolkit 0.946 Parameters for adjusting thread priorities added Channel directory must be created using lower-case
4	2010-06-01	SD		Updated udev rule to get write access Added compilation with eclipse Update functions / limitations / little changes for driver version 1.0.0.0 Added cifXGetDriverVersion() Update file storage if rotary switch is used

Table 1: List of Revisions

1.3 Overview

The cifX Linux driver runs as a library in userspace and accesses the card via a UIO kernel module (Userspace I/O).

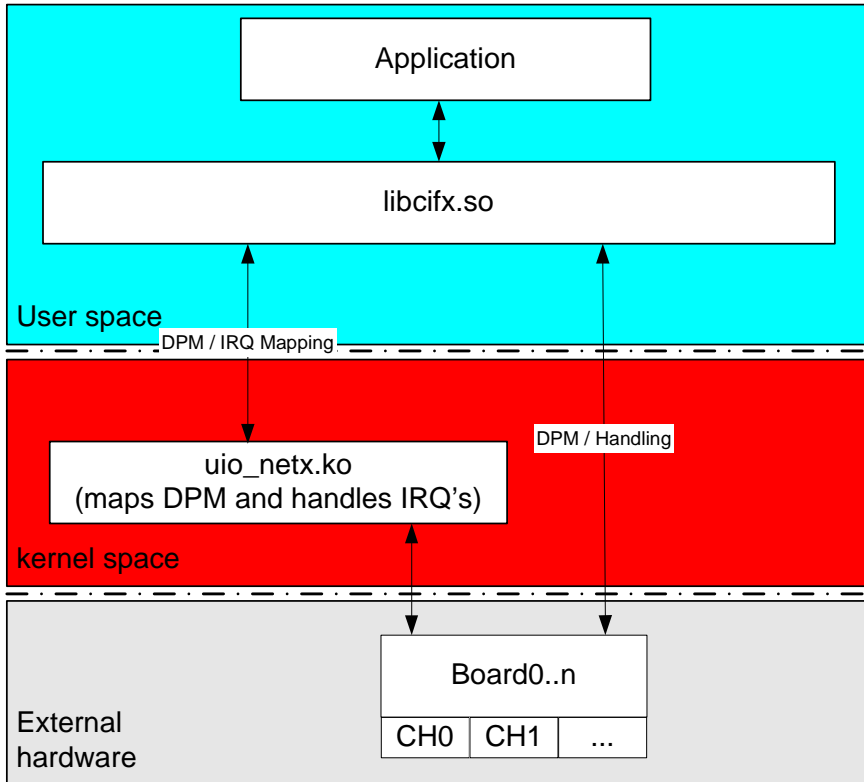


Figure 1: Linux cifX Driver Architecture

1.4 Requirement

Mandatory:

- Linux Kernel 2.6 (tested with 2.6.26)
- libpthread, librt
- cifX board (PCI/PCIe), NXSB-PCA / NXSB100 / NXHX board or NX-PCA-PCI / NXHX or netX Chip DPM connections).
- If building with configure:
pkg-config utility for automatically finding/configuring needed libraries
If building with eclipse: eclipse environment (V3.5.2) with CDT-plugin (V6.0.2)

Optional:

- Linux standard libraries libpciaccess V0.10.2 or later (always needed for cifX PCI cards, support can be disabled by defining `CIFX_TOOLKIT_DISABLEPCI`)

1.5 Features

- Based on the netX Toolkit source V1.0.1.0
- Unlimited number of cifX boards supported
- Support for NXSB-PCA or NX-PCA-PCI boards included

1.6 Limitations

- No Interrupt support for NXSB-PCA and NX-PCA-PCI boards
- On Big Endian machines the user is responsible for converting send/receive packets from/to Little Endian. This is **NOT** automatically done inside the toolkit.
- Interrupt support only available for devices handled through `uio_netx` kernel module
- Only one application can access a card simultaneously. For multi-application access to a single card, a special application needs to be implemented by user.
- Toolkit needs to be run as "root" or with a user that has the following rights:
 - read/write access to the PCI configuration registers (i.e. `"/sys/class/uio/uio<n>/device/config"`)
 - Mapping of DPM to user space (see "mmap" and "ulimit -l")
 - read/write access to devices `"/dev/uio<n>"`
 - read/write access to `/dev/mem` (for user added devices)
- Online diagnostics access via SYCON.net needs a TCP/IP Server functionality integrated into the user application. An example stand alone server is offered with the linux driver.

1.7 CD Contents

Folder	Content
documentation	Driver documentation
driver	
libcifx	cifX Linux driver source (autoconf project / eclipse project)
uio_kernel_patches	netX UIO driver patch for kernel 2.6.26 / 2.6.27 / 2.6.30 (see www.kernel.org)
uio_netx	netx uio driver as module
examples	cifX example application
basedir	Example card configuration directory (copy to /opt/cifx or to your own base directory)
cifxsample	Example application for testing toolkit functions (autoconf project / eclipse project)
cifX TCPServer	Example stand alone TCP server for SYCON.net diagnostic access

Table 2: CD Contents

1.8 Terms, Abbreviations and Definitions

Term	Description
cifX	C ommunication I nterface based on netX
comX	C ommunication M odule based on netX
PCI	P eripheral C omponent I nterconnect
UIO	U userspace I I/O
API	A pplication P rogramming I nterface
DPM	D ual- P ort M emory Physical interface to all communication board Note: DPM is also sometimes used for PROFIBUS- DP Master

Table 3: Terms, Abbreviations and Definitions

1.9 References

This document based on the following documents respectively specification:

- [1] Hilscher Gesellschaft für Systemautomation mbH: Driver Manual cifX Device Driver - Windows 2000/XP/Vista/7/CE V1.0.x.x. Revision 15, english, 2010

Table 4: References

1.10 Legal Notes

1.10.1 Copyright

© 2009-2010 Hilscher Gesellschaft für Systemautomation mbH

All rights reserved.

The images, photographs and texts in the accompanying material (manual, accompanying texts, documentation, etc.) are protected by German and international copyright law as well as international trade and protection provisions. You are not authorized to duplicate these in whole or in part using technical or mechanical methods (printing, photocopying or other methods), to manipulate or transfer using electronic systems without prior written consent. You are not permitted to make changes to copyright notices, markings, trademarks or ownership declarations. The included diagrams do not take the patent situation into account. The company names and product descriptions included in this document may be trademarks or brands of the respective owners and may be trademarked or patented. Any form of further use requires the explicit consent of the respective rights owner.

1.10.2 Important Notes

The manual, accompanying texts and the documentation were created for the use of the products by qualified experts, however, errors cannot be ruled out. For this reason, no guarantee can be made and neither juristic responsibility for erroneous information nor any liability can be assumed. Descriptions, accompanying texts and documentation included in the manual do not present a guarantee nor any information about proper use as stipulated in the contract or a warranted feature. It cannot be ruled out that the manual, the accompanying texts and the documentation do not correspond exactly to the described features, standards or other data of the delivered product. No warranty or guarantee regarding the correctness or accuracy of the information is assumed.

We reserve the right to change our products and their specification as well as related manuals, accompanying texts and documentation at all times and without advance notice, without obligation to report the change. Changes will be included in future manuals and do not constitute any obligations. There is no entitlement to revisions of delivered documents. The manual delivered with the product applies.

Hilscher Gesellschaft für Systemautomation mbH is not liable under any circumstances for direct, indirect, incidental or follow-on damage or loss of earnings resulting from the use of the information contained in this publication.

1.10.3 Exclusion of Liability

The software was produced and tested with utmost care by Hilscher Gesellschaft für Systemautomation mbH and is made available as is. No warranty can be assumed for the performance and flawlessness of the software for all usage conditions and cases and for the results produced when utilized by the user. Liability for any damages that may result from the use of the hardware or software or related documents, is limited to cases of intent or grossly negligent violation of significant contractual obligations. Indemnity claims for the violation of significant contractual obligations are limited to damages that are foreseeable and typical for this type of contract.

It is strictly prohibited to use the software in the following areas:

- for military purposes or in weapon systems;
- for the design, construction, maintenance or operation of nuclear facilities;
- in air traffic control systems, air traffic or air traffic communication systems;
- in life support systems;
- in systems in which failures in the software could lead to personal injury or injuries leading to death.

We inform you that the software was not developed for use in dangerous environments requiring fail-proof control mechanisms. Use of the software in such an environment occurs at your own risk. No liability is assumed for damages or losses due to unauthorized use.

1.10.4 Export

The delivered product (including the technical data) is subject to export or import laws as well as the associated regulations of different countries, in particular those of Germany and the USA. The software may not be exported to countries where this is prohibited by the United States Export Administration Act and its additional provisions. You are obligated to comply with the regulations at your personal responsibility. We wish to inform you that you may require permission from state authorities to export, re-export or import the product.

2 Licensing Terms

The Hilscher cifX Linux driver consists of several modules.

- **uio_netx** Created by Linutronix GmbH and available from www.kernel.org

This module is licensed under GPL V2 and can be used under this terms.

- **libcifx** Offered by Hilscher Gesellschaft für Systemautomation mbH

This library is a userspace library and an intellectual property of the

Hilscher Gesellschaft für Systemautomation mbH.

The source code and library can be used for internal development, modification and debugging purpose.

Distribution of the original **libcifx** source code, parts of the **libcifx** source code or modifications based on it is prohibited.

Binary distribution for use in products is allowed.

3 Installation

The Linux UIO driver needs to be compiled for your kernel, if it is not yet included in your Linux distribution.

Checking if the UIO driver is included in the Linux distribution can be done by using the following command:

```
[ -f /lib/modules/`uname -r`/kernel/drivers/uio/uio_netx.ko ] && echo "Found cifX UIO module" || echo "ATTENTION: NO cifX UIO module found"
```

The cifX userspace library always needs to be compiled.

This chapter describes the manual installation procedure consisting of the compilation and installation of both modules (libcifx, uio_netx) including the cifX example program.

3.1 Prerequisites

- Kernel sources 2.6.26 or later (if kernel module is not included in your Linux distribution)
- GCC 4.x.x (tested with version 4.4.2)

For PCI card support:

- Library and development package of **libpciaccess** (V0.10.3 or later)

3.2 Compilation of the netX UIO Kernel Module

If the kernel module must be compiled you should refer to your Linux distribution on how to build the kernel.

The following steps describing the generic build procedure of a kernel and may differ from your kernel build mechanism.

- Change to your working directory (e.g. /usr/src)


```
cd /usr/src
```
- Extract the kernel sources


```
tar xjf linux-source-2.6.26.tar.bz2
```
- Apply the patch from the CD

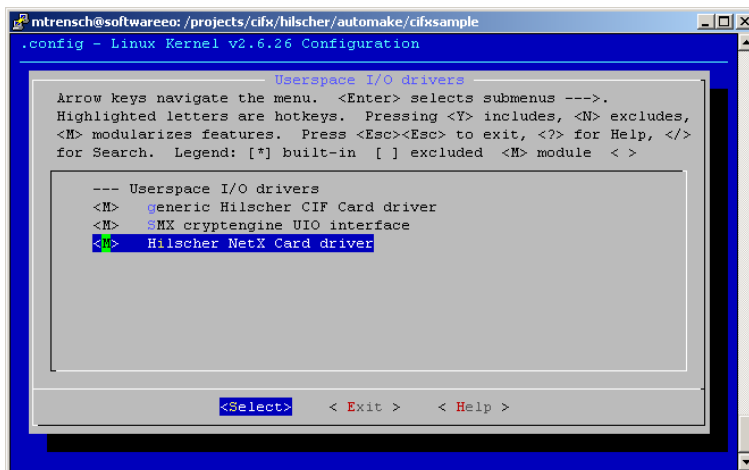

```
patch -p0 < /mnt/cdrom/driver/kernel_patches/uiio-netx.patch
```
- Change to the kernel build directory


```
cd linux-source-2.6.26
```
- Load your old kernel configuration via command line or inside "make menuconfig"


```
make oldconfig
```
- Configure your kernel to include UIO ("Userpace I/O drivers") and uiio_netx ("Hilscher NetX Card driver")

make menuconfig

Enable "Device Drivers / Userspace I/O Drivers / Hilscher netX Card Driver"



- Optional: Rebuild the kernel (only necessary if Hilscher netX Card driver should be a built-in driver, not a module)


```
make all install
```
- Build and install the modules


```
make modules modules_install
```

3.2.1 Compilation of the UIO Kernel Module via console

The next lines describing an alternative way to build the module.

- Change to your working directory (e.g. /usr/src)
cd /usr/src
- Extract the module sources from the cdrom
tar xvjf /mnt/cdrom/driver/uiio_netx/uiio_netx.tar.bz2
- Change to libcifx build directory
cd uiio_netx
- Run the makefile
make
- Copy module “uio_netx.ko” in the directory specified for modules (distribution dependent)
cp uio_netx.ko /lib/modules/\$(uname -r)/kernel/drivers/uiio/
- Update the list of the modul-dependencies
depmod

At this point the module is only installed. Module loading is described in chapter 3.5. To load the module automatically every time at system start, you must add a startup script (/etc/init.d).

3.3 Compilation of the cifX Userspace Library

The userspace library contains the cifX Toolkit with all necessary Linux adaption. This library needs to be build for your system. Make sure it is installed in your library search path.

3.3.1 Compilation of the cifX Userspace Library via console

Installation Procedure:

- Change into your working directory (e.g. `cd ~`)
- Extract the libcifx sources from the cdrom
`tar xvjf /mnt/cdrom/driver/libcifx/libcifx-1.0.0.0.tar.bz2`
- Change to libcifx build directory
`cd libcifx_1.0.0.0`
- Run the configure script

`./configure`

Option	Parameter	Description
<code>--prefix</code>	Installation path	Sets the path where the library (subdirectory <code>lib</code>) and include files (subdirectory <code>include/cifx</code>) will be installed. Default: <code>/usr/local</code>
<code>--enable-debug</code>	none	Enables debug symbols for the generated library
<code>--disable-pci</code>	none	Disable PCI support. This will remove all links to <code>libpciaccess</code> . Note: When compiling without PCI support, the driver cannot handle cifX PCI cards any more
<code>--enable-verbose</code>	none	Enable verbose outputs to console
<code>--enable-single-directory</code>	none	Use subdirectory <code>"deviceconfig/FW/channelx"</code> beneath base directory for firmware/configuration file storage. Note: This will force all handled devices to use the same firmware/configuration
<code>PCIACCESS_CFLAGS</code> <code>PCIACCESS_LIBS</code>	compiler parameters	Force the usage of the given parameters for the <code>libpciaccess</code> and don't use <code>pkg-config</code>

Table 5: Additional libcifx Configuration Options

- Build all source modules
`make all`
- Install the library and include files
`make install`

Example for compilation without using pkg-config

```
./configure PCIACCESS_CFLAGS="-I/opt/pciaccess/include -L/opt/pciaccess/lib"
  PCIACCESS_LIBS="-lpciaccess"
make all
make install
```

3.3.2 Compilation of the cifX Userspace Library via IDE

Get the eclipse environment from <http://www.eclipse.org/downloads/>. Depending on the download, additionally you will need the CDT-plugins (<http://www.eclipse.org/cdt/downloads.php>). They are required to build and debug C/C++ projects. For more documentation see <http://www.eclipse.org/cdt/>. There is also information about how to start and develop under the eclipse environment.

When eclipse is installed and the workspace path is set, you can load the predefined cifX library project as follows:

- Change into working directory (e.g. `cd /home/~/.workspace/`)
- Extract the libcifx sources from the cdrom
`tar xvjf /mnt/cdrom/driver/libcifx/libcifx-1.0.0.0.tar.bz2`
- Start eclipse
- Select **File > Import** and choose in the folder **General, Existing Projects into Workspace**
- Select the path to the extracted sources and load the shown pre-selected project

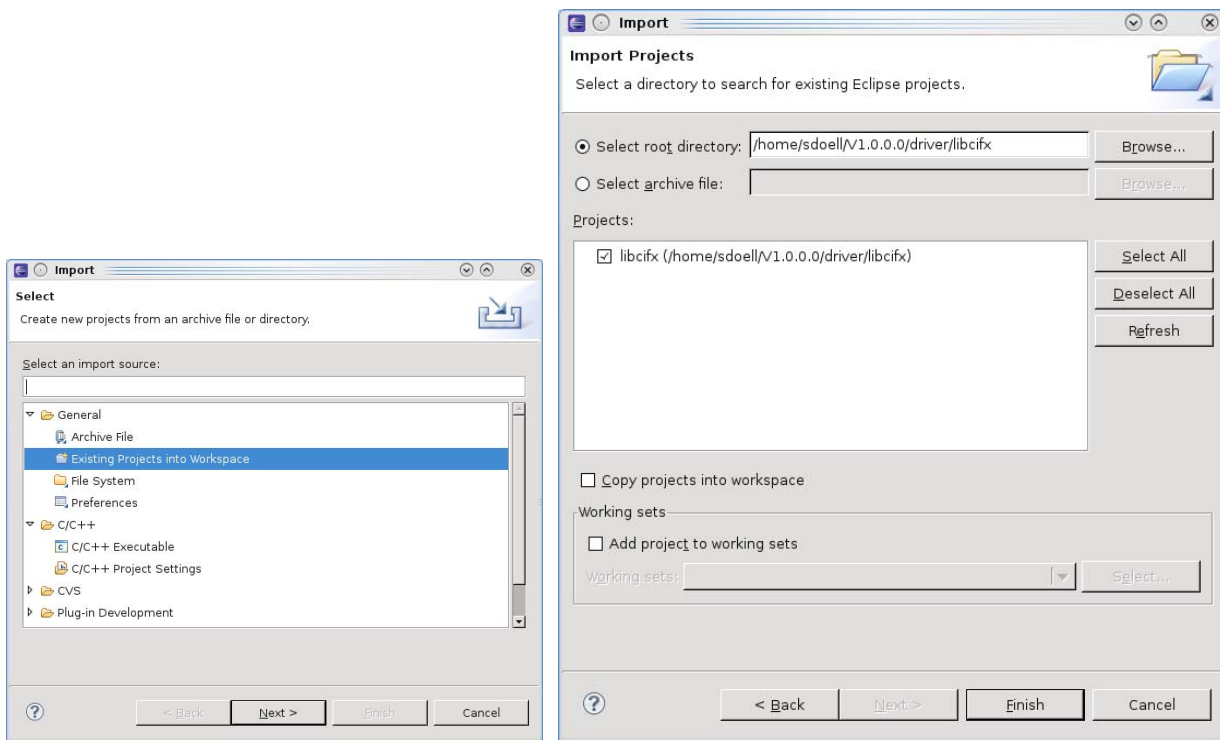


Figure 2: Eclipse IDE – Import Project

After importing the project, with a right click on libcifx, which is shown in the project explorer, extended settings can be done.

- Right click on the project **libcifx**
- Select **Properties > C/C++ Build > Settings**
- Under the tab **Tool Settings > Symbols** you can define or undefine special compile parameters

The default setting is a debug version (g3) without any optimization. The following compiler flags can be set additionally.

Option	Parameter	Description
DEBUG	compiler parameters	Enables debug symbols for the generated library
CIFX_TOOLKIT_DISABLEPCI	compiler parameters	Disable PCI support. This will remove all links to libpciaccess. Note: When compiling without PCI support, the driver cannot handle cifX PCI cards any more
VERBOSE	compiler parameters	Enable verbose outputs to console
CIFX_TOOLKIT_USESINGLE_DIRECTORY	compiler parameters	Use subdirectory " <i>deviceconfig/FW/channelx</i> " beneath base directory for firmware/configuration file storage. Note: This will force all handled devices to use the same firmware/configuration

Table 6: Additional libcifx Configuration Options

The default location of the library is "*~/libcifx/Default*" and must be copied to the installation path (*/usr/local/lib/*). Finally run the next three steps:

- Change into installation directory (`cd /usr/local/lib/`)
- Run `ldconfig` to register library and create a link

ldconfig

- Create a symbolic link to the cifxlink

ln -s libcifx.so.1 libcifx.so

Note: The required include files must also be copied to the installation path (*/usr/local/include/cifx/*).

- *./src/cifxlinux.h*
- *./src/Toolkit/cifXUser.h*
- *./src/Toolkit/cifXErrors.h*
- *./src/Toolkit/cifXEndianness.h*
- *./src/Toolkit/TLR_Types.h*
- *./src/Toolkit/rcX_Public.h*
- *./src/Toolkit/rcX_User.h*

3.4 Compilation of the cifX Example Program

The cifX example program uses the first available netX board for some basic tests. Before using the test application make sure you have compiled and installed the cifX library (see 3.3)

3.4.1 Compilation of the cifX Example Program via Console

Installation Procedure:

- Change into your working directory (e.g. `cd ~`)
- Extract the *cifxsample* sources from the cdrom
`tar xvjf /mnt/cdrom/example/cifxsample.tar.bz2`
- Change to *cifxsample* build directory
`cd cifxsample`
- Run the configure script

`./configure`

Option	Parameter	Description
<code>--prefix</code>	Installation path	Sets the path where the program will be installed. Default: <code>/usr/local</code>
<code>--enable-debug</code>	None	Enables debug symbols for the generated library
<code>--with-cifx-lib</code>	Path to cifX library	Needs to be set if your installation target of the cifX library is not in your default library search path
<code>--with-cifx-include</code>	Path to cifXUser.h, etc.	Needs to be set if the cifX includes are not in your default include path

Table 7: Additional *cifxsample* Configuration Options

- Build all source modules
`make all`
- Optional: Install the program
`make install`

Example for compilation without using pkg-config

```
./configure libcifx_CFLAGS="-I/usr/local/include/cifx -L/usr/local/lib"
libcifx_LIBS="-lcifx -lpthread -lrt" PCIACCESS_CFLAGS="-I/opt/pciaccess/include
-L/opt/pciaccess/lib" PCIACCESS_LIBS="-lpciaccess"
make all
```

3.4.2 Compilation of the cifX Example Program via IDE

Like mentioned before, to open the example project you must copy the entire *cifxsample* folder in your local workspace.

- Change into working directory (e.g. `cd /home/~/.workspace/`)
- Extract the *cifxsample* sources from the cdrom
`tar xvjf /mnt/cdrom/example/cifxsample.tar.bz2`
- Start eclipse and import the project like noted in 3.3.2

Before compiling the example, the library *libcifx* must be installed (see 3.3).

The default search path for the header is “`/usr/local/include/cifx`”. If another path is used, set the include path to the specified one.

- Right click to the project **cifxsample**
- Select **Properties > C/C++ Build > Settings**
- Under the tab **Tool Settings > Directories** you can set a new or additional include path

Further you can print some debug information with defining a compiler flag *DEBUG* (set compiler flags, see 3.3.2).

Option	Parameter	Description
DEBUG	compiler parameters	Enables debug information output. (Disabled by default)

Table 8: Additional *cifxsample* Configuration Options

The eclipse debug environment can be used after compiling the project. When the library *libcifx* is build in debug version, it is also possible to step into the driver functions.

3.5 Loading netX UIO Driver Module

To load the UIO driver module you will need to login as root and enter the following command.

```
modprobe uio_netx
```

Note: To automatically load the UIO driver module, check the manual of your Linux distribution. Usually kernel modules loaded at startup are placed in `/etc/modules`.

Using netX UIO Driver as user (non-root):

If you want to correctly access the UIO driver as user you will need to make sure the user has read / write access to the following device nodes:

- `/dev/uio<n>`
- `/sys/class/uio/uio<n>/device/config`

This can automatically be done by writing a udev rule (see example below):

```
/etc/udev/netx.rules
SUBSYSTEMS=="pci",ATTRS{vendor}=="0x15cf",ATTRS{device}=="0x0000",MODE="0666",PROGRAM="/bin/bash -c 'chmod 0666 /sys/class/uio/uio%n/device/config'"
```

To allow mapping of the DPM to a user application you will need to make sure, that the application is allowed to *mmap* enough memory. You can check the current memory lock limit using the following command, which returns the maximum possible mapped memory in kB :

```
ulimit -l
```

3.6 Firmware and Configuration File Storage

cifX cards are not using any flash memory to store a firmware or configuration on the card. Every time the card is powered-up the firmware and configuration must be downloaded to the hardware.

This chapter describes where to store these files depending on how to cards should be identified. Identification of the a card can be done in three different ways (single directory / device and serial number / *Slotnumber*) described below.

Note: Firmware and configurations are not stored on the hardware and must be downloaded each time the card is powered-up.

It is the task of the driver to initialize the card and therefore the driver has to know which files must be loaded to the hardware.

To allow device specific configuration, every file that needs to be downloaded must be stored in an own folder. These folder reside under a global base folder (default: "/opt/cifx") and can be manually changed during driver initialization.

- Use a single direcotry

If only one cifX device needs to be supported, a predefined directory can be used by setting the driver symbol `CIFX_TOOLKIT_USESINGLE_DIRECTORY` accordingly. The firmware and configuration file must reside in the subdirectory *FW*.

- Use the *Slotnumber* (hardware rotary switch)

The *Slotnumber* serves to distinguish cifX cards from each other clearly, especially if more cifX cards are installed in one PC. The *Slotnumber* must be set at the cifX card using the Rotary Switch *Slotnumber*. While *Slotnumber* 0 means, that the cifX card is identified via its device and serial number, values from 1 to 9 corresponds to the *Slotnumber* 1 to 9. The firmware and configuration file must reside in the subdirectory *Slot_<1..9>*.

- Use the device and serial number (default)

If the cifX device is not equipped with a rotary switch or the *Slotnumber* should not be used, the device is identified by its device and serial number. The firmware and configuration file must reside in the subdirectory */<Device Number>/<Serial Number>/*.

3.6.1 Device Identification via Single Directory

The following table describes the different subdirectory levels, using Single Directory:

Subdirectory	Description
<BASEDIR>	Base directory Can be changed during userspace library initialization. This directory must contain the second stage PCI bootloader (NXCIF50-RTE.bin) Default: "/opt/cifx"
deviceconfig	Device specific configuration files
FW	If uses single directory is defined the search path is set to "<BASEDIR>/deviceconfig/FW". Contains <i>device.conf</i> storing device specific settings NOTE: This directory must contain the rcX base firmware if loadable modules are used.
channel<#>	Channel specific files NOTE: Currently only channel 0 is supported

Table 9: Firmware and Configuration File Storage - Single Directory

Sample file structure for a cifX device with device number 1250100 and serial number 20217:

```
+ <BASEDIR>/deviceconfig
|
|--+ FW
|   |-- device.conf
|   |--+ channel0
|       |-- cifXdps.nxf
|       |-- warmstart.dat
|
|--+ channel1
|--+ channel2
|--+ channel3
|--+ channel4
|--+ channel5
```

3.6.2 Device Identification via Device and Serial Number

Note: `<Device Number>/<Serial Number>` are shown on the device hardware label.

Example:

Hardware Label Entry: **1250.100 / 20217**

Directory Entry: **"/ 1250100 / 20217"**

The following table describes the different subdirectory levels, without using rotary switch:

Subdirectory	Description
<BASEDIR>	Base directory Can be changed during userspace library initialization. This directory must contain the second stage PCI bootloader (NXCIF50-RTE.bin) Default: "/opt/cifx"
deviceconfig	Device specific configuration files
<Device Number>	Device number of the device File storage for cifX cards with a given device and serial number
<Serial Number>	Serial number of the device Contains <i>device.conf</i> storing device specific settings NOTE: This directory must contain the rcX base firmware if loadable modules are used.
channel<#>	Channel specific files NOTE: Currently only channel 0 is supported

Table 10: Firmware and Configuration File Storage - Device and Serial Number

Sample file structure for a cifX device with device number 1250100 and serial number 20217:

```
+ <BASEDIR>/deviceconfig
|
|--+ 1250100
|   |--+ 20217
|       |-- device.conf
|       |--+ channel0
|           |--cifXdps.nxf
|           |--warmstart.dat
|       |--+ channel1
|       |--+ channel2
|       |--+ channel3
|       |--+ channel4
|       |--+ channel5
... |--+ 20300
```

3.6.3 Device Identification via Slotnumber (rotary switch)

The following table describes the different subdirectory levels, if the device provides a rotary switch:

Subdirectory	Description
<BASEDIR>	Base directory Can be changed during userspace library initialization. This directory must contain the second stage PCI bootloader (NXCIF50-RTE.bin) Default: "/opt/cifx"
deviceconfig	Device specific configuration files
Slot_<1..9>	If device provides a rotary switch the files will be stored under Slot_<rotary switch set>. (Only if the switch is set !=0) Contains <i>device.conf</i> storing device specific settings NOTE: This directory must contain the rcX base firmware if loadable modules are used.
channel<#>	Channel specific files NOTE: Currently only channel 0 is supported

Table 11: Firmware and Configuration File Storage - Rotary Switch

Sample file structure for a cifX device identified by a *Slotnumber 2* and loadable module support:

```
+ <BASEDIR>
|
|-- NETX100-BSL.BIN
|
|--+deviceconfig
|   |
|   |--+ 1250100
|   |
|   |--+ Slot_1
|   |--+ Slot_2
|   |
|   |   |--+ channel0
|   |   |   |
|   |   |   |--nx100dpm.nxo (loadable module)
|   |   |   |--config.nxd (fieldbus database)
|   |   |
|   |   |--+ channel1
|   |   |--+ channel2
|   |   |--+ channel3
|   |   |--+ channel4
|   |   |--+ channel5
|   |
|   |-- device.conf (configuration file)
|   |-- cifXrcX.nxf (rcX base firmware)
```

4 Linux Driver Specific Information

The Linux driver needs some special initialization compared to the standard Windows driver, as it is not executed by the kernel at system startup. The driver (libcifx) is linked to an application and needs to be correctly configured to work.

To enable the use of the cifX driver by an application, some special functions are provided. These functions are described in the following chapters. Chapter 4.3 describes the correct use and sequence of the functions.

4.1 Additional Structures

Some of the Linux specific functions need parameters provided through structures. The structures and the meaning of the internal data are described in the following chapter.

4.1.1 Structure CIFX_LINUX_INIT

This structure is used to initialize the cifX driver.

Element	Datatype	Description
init_options	int	Driver Initialization options: 0 = CIFX_DRIVER_INIT_NOSCAN Driver does not scan for available cards 1 = CIFX_DRIVER_INIT_AUTOSCAN Driver scans for all available cards and adds them to the application
base_dir	const char*	Set the base directory of the driver, can be NULL to use the default of <i>'/opt/cifx'</i>
poll_interval	unsigned long	Polling interval in milliseconds [ms] for non-interrupt cards. Used for Change of State (COS) signaling Can be 0 to use the default of 500ms. Use CIFX_POLLINTERVAL_DISABLETHREAD to disable polling completely
poll_priority	int	Priority of the polling thread
trace_level	unsigned long	Set the trace level of the driver.
user_card_cnt	int	Number of user cards to add to driver (e.g. if a card is connected via DPM and cannot be accessed using the netX UIO driver).
user_cards	struct CIFX_DEVICE_T*	Pointer to an array of user added cards. Length of the array must be given in <i>user_card_cnt</i> .

Table 12: Structure CIFX_LINUX_INIT Definition

4.1.2 Structure CIFX_DEVICE_T

This structure describes a cifX device which should be added to the driver. This structure can be acquired through *cifXFindDevice()* or filled by the user if a custom card should be added.

Element	Datatype	Description
dpm	unsigned char*	Virtual Pointer to card DPM
dpmaddr	unsigned long	Physical address of the DPM (this value is used to detect the PCI card linked to the DPM)
dpmlen	unsigned long	Size of the DPM in bytes
uio_num	int	UIO number of the device. Set to -1 if not using a UIO device
uio_fd	int	File handle to UIO device. Set to -1 if not using a UIO device NOTE: Interrupt mode is only available on UIO devices with at least 64kB DPM
pci_card	int	0 = Card is a non-PCI card, which comes with a firmware in FLASH memory (no reset during start-up) 1 = Card is a PCI card, which is reset on every start
force_ram	int	0 = Auto-detect card storage (PCI = RAM, DPM = Flash) 1 = Force usage of RAM only on this card. (This will execute a HW reset and download Bootloader / Firmware on every start of the card)
notify	PFN_CIFX_NOTIFY_EVENT	Callback that is made at several stages when initializing a device. This allows the user to setup DPM and timings (if they are different from the netX ROM Loader settings) Pass NULL to suppress callback
userparam	void*	User specific parameter assigned to device

Table 13: Structure CIFX_DEVICE_T Definition

4.2 Additional Functions

This chapter describes functions which are only available for the Linux version of the driver. These functions need to be used to initialize the driver for the use inside an application.

4.2.1 cifXDriverInit()

This function must be called before accessing any driver function. It initializes the driver and adds the needed devices to the control of the libcifx shared library.

Function call:

```
int32_t cifXDriverInit(struct CIFX_LINUX_INIT* init_params)
```

Arguments:

Argument	Data type	Description
init_params	struct CIFX_LINUX_INIT_T*	Initialization parameters (see 4.1.1 for details)

Return Values:

CIFX_NO_ERROR (0) if the driver was successfully initialized.

4.2.2 cifXDriverDeinit()

Un-initialize the driver and remove all devices from the control of the libcifx shared library. After calling this function the application must not access any cifX driver API function any more.

Function call:

```
void cifXDriverDeinit(void)
```

Arguments:

None

4.2.3 cifXGetDriverVersion()

This function returns the version of the cifX driver for Linux.

Function call:

```
int32_t cifXGetDriverVersion ( uint32_t ulSize, char* szVersion);
```

Arguments:

Argument	Data Type	Description
ulSize	unsigned long	Size of buffer referenced by parameter <i>szVersion</i>
szVersion	char*	Buffer to return driver version string

Return Values:

Return Values	
CIFX_NO_ERROR	Memory mapping successful
CIFX_INVALID_BUFFERSIZE	Size of supplied buffer is too small

4.2.4 cifXGetDeviceCount()

Query the number of available UIO devices. Device detection only works through the netX UIO driver.

Function call:

```
int cifXGetDeviceCount(void)
```

Arguments:

None

Return Values:

Number of detected devices.

4.2.5 cifXFindDevice()

Build a CIFX_DEVICE_T structure for a given device.

The structure can be used by an application if only some specific cards should be used. Therefore the application has to add them manually to the driver as "user_cards" (see 4.1.1).

This can be done by calling *cifXDriverInit()* with AUTO_SCAN disabled and passing the card information in the *user_cards* parameter.

Function call:

```
struct CIFX_DEVICE_T* cifXFindDevice(int num)
```

Arguments:

Argument	Data type	Description
num	int	Device number of the chosen device. Range: 0..cifXGetDeviceCount()

Return Values:

Pointer to the device information structure of the given device. If the device number is invalid or the device is already used by another application, this function returns NULL.

4.2.6 cifXDeleteDevice()

Delete a CIFX_DEVICE_T structure that was returned by *cifXFindDevice()*. This needs to be done **after** the driver un-initialization to clean up all internally used administration data and allocated memory areas.

Function call:

```
void cifXDeleteDevice(struct CIFX_DEVICE_T* device)
```

Arguments:

Argument	Data type	Description
device	struct CIFX_DEVICE_T*	Pointer to a device returned by <i>cifXFindDevice()</i>

4.3 Driver/Library Startup Procedure

The driver startup procedure can be controlled by the user.

The following two use cases are available:

- Automatically add all found uio_netx based devices and optionally add user specific devices
- Skip uio_netx device scan and only add user specific device

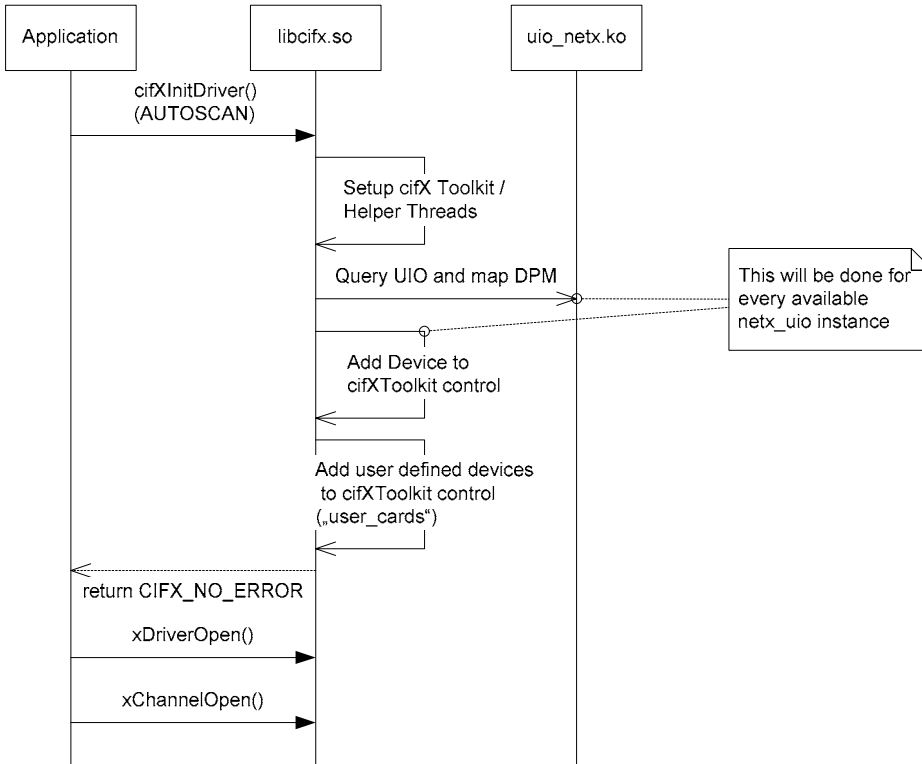


Figure 3: Initialization of libcifx with CIFX_DRIVER_INIT_AUTOSCAN

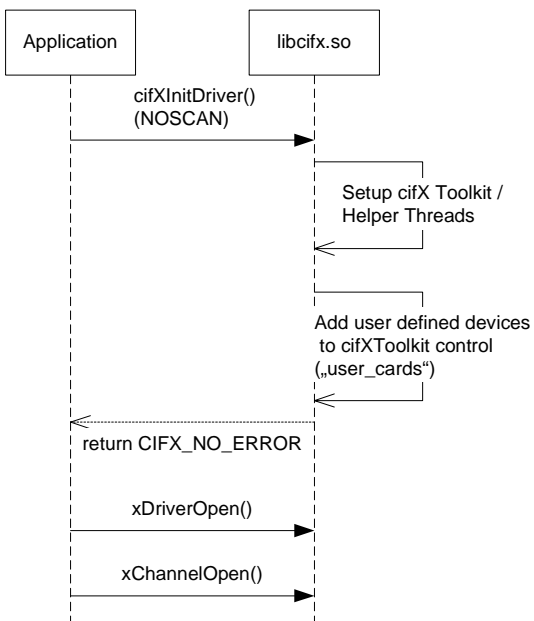


Figure 4: Initialization of libcifx with CIFX_DRIVER_INIT_NOSCAN

4.4 Device Configuration (device.conf)

Parameters like a unique alias name and interrupt support can be enabled per device. The configuration file must be called *'device.conf'* and must reside in the top level directory of a device, identified by the device number and serial number (e.g. *'/opt/cifx/1250100/20217/'*).

The file may contain the following keys:

Key	Datatype	Description
alias	char[16]	Alias name for the device. Must be less than 16 characters.
irq	string "yes"/"no"	yes/no to enable/disable IRQ on the device
irq_prio	int	Priority of the ISR handler thread
irq_sched	string	Setup alternate ISR scheduling algorithm (see man pages) "fifo" - FIFO scheduling (see SCHED_FIFO) "rr" - Real-Time Scheduling (see SCHED_RR)

Table 14: *device.conf* Parameters

Sample device.conf:

```
#Sample device configuration file
alias=PROFIBUS
irq=no
irq_prio=1
irq_sched=fifo
```

5 Using SYCON.net to Configure the Fieldbus System

The Hilscher Fieldbus hardware will be configured by a Windows application called SYCON.net. SYCON.net is based on the FDT/DTM concept and generates the configuration files for the hardware. It is also able to update the firmware for a specific card.

Please use the following steps to create a configuration:

- Install SYCON.net
- Open SYCON.net and create a configuration
- Store the SYCON.net configuration project and export the configuration from SYCON.net into a database file.
- Copy the database and the firmware files to the device configuration directory.
- Now start/restart the cifX Linux driver. This will load the firmware and configuration into the cifX card.

SYCON.net is also able to connect to a remote device supporting the Hilscher "cifX Diagnostics and Remote Access" functions.

An example standalone TCP/IP server, offering these functions, can be found in the examples of the linux driver.

6 Programming with the cifX Linux Driver

The cifX Linux driver offers the same interface like the cifX Windows driver and therefore the cifX Device Driver manual can be used. This manual describes the driver functions (API), error codes and shows some program examples.

Note: As the driver is contained in the library linked to your application, you will need to initialize the driver by calling the function "*cifXDriverInit*".

Initialization Example:

```
struct CIFX_LINUX_INIT init =
{
    .init_options = CIFX_DRIVER_INIT_AUTOSCAN, // Find all PCI cards automatically
    .base_dir     = NULL, // default to /opt/cifx
    .poll_interval = 0, // Use default poll interval (500ms)
    .trace_level  = 255, // Enable all debugging outputs to log file
    .user_card_cnt = 0, // no user defined cards
};
/* First of all initialize toolkit */
long lRet = cifXDriverInit(&init);

/* TODO: Insert your application here */

cifXDriverDeinit();
```

The installation CD also includes an "Example" directory with Linux specific examples.

7 Appendix

7.1 List of Tables

Table 1: List of Revisions	3
Table 2: CD Contents	6
Table 3: Terms, Abbreviations and Definitions	6
Table 4: References	6
Table 5: Additional libcifx Configuration Options	13
Table 6: Additional libcifx Configuration Options	15
Table 7: Additional cifxsample Configuration Options	16
Table 8: Additional cifxsample Configuration Options	17
Table 9: Firmware and Configuration File Storage - Single Directory	20
Table 10: Firmware and Configuration File Storage - Device and Serial Number	21
Table 11: Firmware and Configuration File Storage - Rotary Switch	22
Table 12: Structure CIFX_LINUX_INIT Definition	23
Table 13: Structure CIFX_DEVICE_T Definition	24
Table 14: device.conf Parameters	29

7.2 List of Figures

Figure 1: Linux cifX Driver Architecture	4
Figure 2: Eclipse IDE – Import Project	14
Figure 3: Initialization of libcifx with CIFX_DRIVER_INIT_AUTOSCAN	28
Figure 4: Initialization of libcifx with CIFX_DRIVER_INIT_NOSCAN	28

7.3 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 Ges.f.Systemaut. mbH
Shanghai Representative Office
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.
New Delhi - 110 025
Phone: +91 11 40515640
E-Mail: info@hilscher.in

Italy

Hilscher Italia srl
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.
Suwon-Si, 443-810
Phone: +82-31-204-6190
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