



Driver Manual
cifX Device Driver
VxWorks
V5.5/V6.2

Hilscher Gesellschaft für Systemautomation mbH

www.hilscher.com

DOC090602DRV02EN | Revision 2 | English | 2010-06 | Released | Public

Table of Contents

1	Introduction.....	3
1.1	About this Document.....	3
1.1.1	Overview.....	3
1.2	List of Revisions.....	4
1.3	Terms, Abbreviations and Definitions.....	4
1.4	References.....	4
1.5	Requirement.....	5
1.6	Features.....	5
1.7	Limitations.....	5
1.8	CD Contents.....	6
1.9	Legal Notes.....	7
1.9.1	Copyright.....	7
1.9.2	Important Notes.....	7
1.9.3	Exclusion of Liability.....	8
1.9.4	Export.....	8
2	Licensing Terms.....	9
3	Installation.....	10
3.1	MMU Support and Settings.....	10
3.2	Building the cifX Driver.....	12
3.2.1	Initialization at System Startup.....	13
3.2.2	Initialization during Application Execution.....	16
3.3	Firmware and Configuration File Storage.....	18
4	VxWorks Driver Specific Information.....	20
4.1	Additional Structures.....	20
4.1.1	Structure VXW_CIFXDRV_PARAMETERS_T.....	20
4.1.2	Structure VXW_CIFXDRV_DEVICEENTRY_T.....	21
4.1.3	Structure VXW_CIFXDRV_DEVICEBUSINFO_T.....	21
4.2	Additional functions.....	22
4.2.1	cifXInitDriver().....	22
4.2.2	cifXDeinitDriver().....	23
4.2.3	cifXFindDevice().....	24
4.2.4	cifXMemMap().....	25
4.2.5	cifXGetDriverVersion().....	26
4.3	Driver Startup Procedure.....	27
4.4	Device Configuration (device.conf).....	28
5	Programming with the cifX VxWorks Driver.....	29
6	Appendix.....	30
6.1	List of Tables.....	30
6.2	List of Figures.....	30
6.3	Contacts.....	31

1 Introduction

1.1 About this Document

Wind River VxWorks is a real-time operating system (RTOS) and the fundamental run-time component of the Tornado II (VxWorks 5.x releases) and the Wind River Workbench (VxWorks 6.x releases) development platform. VxWorks is flexible, scalable, reliable and available on all popular CPU platforms.

This manual describes the Hilscher cifX driver for VxWorks and its architecture. The driver offers access to the Hilscher netX based hardware (e.g. CIFX50, comX) with the same functional API as the cifX device driver for Windows and offers transparent access to the different devices.

1.1.1 Overview

The cifX VxWorks driver is available as a library built around the cifX Toolkit. Any application which needs to access a cifX device can use the device specific functions provided by this driver library. The concept of the cifX device driver is illustrated in the subsequent figure.

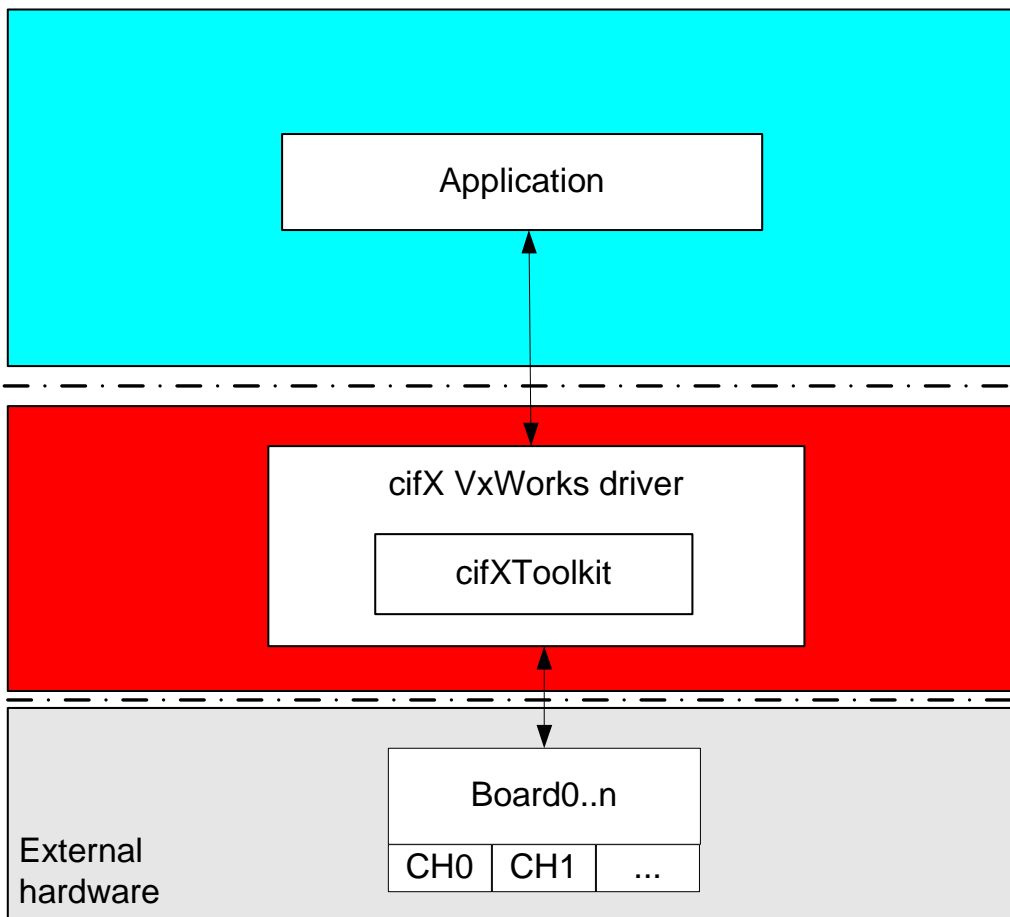


Figure 1 : VxWorks cifX Driver Architecture

1.2 List of Revisions

Rev	Date	Name	Chapter	Revision
1	2009-06-09	SS	All	Created
2	2010-05-31	SS	3.3	Support for loadable modules
			3.3, 4.4	Support for slot numbers
			3.3, 4.1.1, 4.4	Driver parameter <i>fSingleDir</i> added to support single firmware directory
			3.2.1.1	Update screenshot
			4.2	Return types of additional functions adapted to <code>stdint</code> data types
			4.2.2	Return value of <code>cifXDeinitDriver()</code> changed
			4.2.5	Description for new function <code>cifXGetDriverVersion()</code> added
			4.4	DMA mode configuration added

Table 1: List of Revisions

1.3 Terms, Abbreviations and Definitions

Term	Description
cifX	Communication Interface based on netX
comX	C ommunication M odule based on netX
PCI	P eripheral C omponent I nterconnect
API	A pplication P rogramming I nterface
DPM	D ual- P ort M emory Physical interface to all communication board (DPM is also used for PROFIBUS- DP Master).
CDF	C omponent D escription F ile
BSP	B oard S upport P ackage
DMA	D irect M emory A ccess

Table 2: Terms, Abbreviations and Definitions

1.4 References

This document is based on the following documents:

- [1] Hilscher Gesellschaft für Systemautomation mbH: cifX Device Driver - Windows 2000/XP/Vista/7/CE V1.0.x.x. Revision 15, 2010
- [2] VxWorks Kernel Programmers Guide 6.2 Section 5.9 Virtual Memory Management
- [3] VxWorks Kernel Programmers Guide 6.2 Section 2.9 Custom Kernel Components

Table 3: References

1.5 Requirement

- VxWorks V5.5 or
- VxWorks V6.2

1.6 Features

- Based on the cifX Toolkit source (V1.0.1.0)
- Unlimited number of cifX boards supported
- Support for NXSB-PCA or NX-PCA-PCI boards included (PCI-Adapter to a netX DPM)
- Interrupt support for PCI based devices
- DMA data transfer for I/O data
- Support for loadable modules
- Interrupt notification for applications

1.7 Limitations

- No DMA support for NXSB-PCA, NX-PCA-PCI and CIFX104 boards
- No 64 bit support
- 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

1.8 CD Contents

Folder	Content
Documentation	Driver documentation
Driver	
VxWorks 6.2\WindRiver_Workbench	Project files for Wind River Workbench (VxWorks 6.2)
.metadata	VxWorks 6.2 project environment
VxWCIFXDrv	Driver sources and project file
cifxToolkit	cifX Toolkit sources
cifXDrvTest	Driver test application
VxWorks 5.5\Tornado	Project files for Tornado II (VxWorks 5.5)
VxWCIFXDrv	Driver sources and project file
cifxToolkit	cifX Toolkit sources
cifXDrvTest	Driver test application
Component	Source files for integration of cifX driver into the VxWorks image
comps	
src	Component description file
vxWorks	Configlette file
Hilscher	Sources for VxWorks system component (VxWorks 5.5 and 6.2)
cifXDrv	Driver sources
cifxToolkit	cifX Toolkit sources
Example Basedir	Example card configuration directory (copy to your own base directory)

Table 4 : CD Contents

1.9 Legal Notes

1.9.1 Copyright

© 2009-2010 Hilscher Gesellschaft für Systemautomation mbH

All rights reserved.

The images, photographs and texts in the accompanying material (user 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.

1.9.2 Important Notes

The user 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 user 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 user 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 user 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.

1.9.3 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.

1.9.4 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 cifX VxWorks driver offers full source code for the netX chip DPM adaptation to VxWorks.

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

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

Binary distribution for use in products is allowed.

3 Installation

The installation CD includes project files for the development environment Tornado II (VxWorks 5.5) and the development environment Wind River Workbench (VxWorks 6.2). In addition to the driver source, both projects contain example applications.

3.1 MMU Support and Settings

The cifXVxWDriver needs direct access to the dual port memory area of a netX based hardware. This access depends also on the connection of the hardware to the host system (PCI / ISA / DPM). Depending on the connection, the host system must provide the correct memory mappings and memory access masks (read/write and non cached) or the driver fails during hardware access.

A VxWorks target system requires a complete configuration of the hardware including all necessary system memory areas and areas used by the hardware. The configuration takes place in the *sysPhysMemDesc[]* table, located in *sysLib.c*. This table is used by the MMU (memory management unit) to setup the target hardware and to provide access to the defined areas during system startup [2]. The configuration of the memory areas can be done statically, by modifying the *sysPhysMemDesc[]* table or dynamically by calling the dedicated function *sysMmuMapAdd()*. Both methods will be illustrated by the following example.

Example:

A cifX PCI device is located at physical memory address 0xEC000000. The size of the dual port memory is 64kB.

Static MMU Configuration:

In case of a one-to-one mapping of physical and virtual addresses, virtual and physical addresses are identical. For a static configuration the *sysPhysMemDesc[]* table must be extended by the address configuration of the cifX card. Parts of the customized *sysPhysMemDesc[]* table are displayed below.

```
PHYS_MEM_DESC sysPhysMemDesc [] =
{
  /* cifX PCI device @ Physical address 0xEC000000 */
  {
    (VIRT_ADDR)0xEC000000, /* For VxWorks 5.5 use (void *)0xEC000000 here */
    (PHYS_ADDR)0xEC000000, /* For VxWorks 5.5 use (void *)0xEC000000 here */
    0x10000,
    VM_STATE_MASK_VALID | VM_STATE_MASK_WRITABLE | VM_STATE_MASK_CACHEABLE,
    VM_STATE_VALID      | VM_STATE_WRITABLE      | VM_STATE_CACHEABLE_NOT
  },
  DUMMY_MMU_ENTRY,
  DUMMY_MMU_ENTRY
};
```

Dynamic MMU Configuration:

Adding memory area definitions dynamically, must be done at system startup, before initialization of the MMU (*usrMmuInit()*, see *prjConfig.c* of your VxWorks image). As proposed in the VxWorks Kernel Programmers Guide, dynamic memory area definitions could be performed during execution of *sysHwInit()* [2].

The cifX driver provides the function *cifXMemMap()* to dynamically add a memory area definition for a certain cifX device (see section 4.2.4).

The example shows the dynamic configuration for each installed cifX PCI device in the target system.

```
VXW_CIFXDRV_DEVICEENTRY_T tDevEntry = {0};
int iDevNum = 0;

/* Browse cifX devices and perform mapping of dual port memory */
while (cifXFindDevice (&tDevEntry, iDevNum) )
{
    ++iDevNum;
    cifXMemMap (&tDevEntry);
}
```

3.2 Building the cifX Driver

To use a cifX device on your VxWorks environment the device driver must be initialized with the VxWorks specific driver routines, described in section 4. This initialization can be done at system startup or within an application. To perform the initialization during system startup the cifX device driver must be included as a system component into the VxWorks image. For an easy integration process VxWorks provides a graphical component editor which is an inherent part of its integrated development environment. If the initialization should be done during execution of an application, the driver initialization routines must be placed inside the application. Both methods will be introduced below.

- Initialization at System Startup

Therefore the driver must be included into the VxWorks image and becomes a part of it. The driver source in the project workspace does not need to be builded with the development environment. How to integrate and build the driver for the VxWorks image is described in section 3.2.1.

- Initialization during Application Execution

If the driver should be initialized by an application the driver must be build as a downloadable module via the development environment, described in section 3.2.2.

3.2.1 Initialization at System Startup

The VxWorks IDE (Tornado or Wind River Workbench) provides facilities to arrange and configure system components to build a customized VxWorks image [3]. The cifX driver can be built into a custom VxWorks image by adding a new system component to the VxWorks development environment. For this purpose the cifX driver source, a configlet file and a *Component Description File*, which describes the cifX driver component has to be placed into the VxWorks installation tree.

Follow this steps to add the cifX driver as a component to your VxWorks development environment:

- Make sure your Tornado II or Wind River Workbench development environment is closed
- Copy the whole *'Driver/Component/Hilscher'* directory from the driver CD to your VxWorks installation tree: *'installdir/target/config'*
- Place the component description file and the configlet file by copying the whole *'Driver/Component/comps'* directory from the driver CD to your VxWorks installation tree: *'installdir/target/config'*
- Open a windows command shell
VxWorks 5.5: execute *'installdir/host/x86-win32/bin/torVars.bat'*
VxWorks 6.2: execute *'installdir/wrenv.exe -p vxworks-6.2'*
- Navigate to the *'installdir/target/config/Hilscher/cifXDrv'* directory in your VxWorks installation tree and type *'make CPU=PENTIUM4'* (use your CPU architecture here)
- VxWorks 5.5: Open the *'VxWCIFXDriver.wsp'* with the Tornado II environment
VxWorks 6.2: Open the workspace folder *'WindRiver_Workbench'* with the Wind River Workbench
- Create your VxWorks image, open the kernel configuration, include the required components for the cifX driver and rebuild the image
- Build the example application.
Define the macro CIFXDRVINIT_STARTUP (default) in the example source (cifXDrvTest.c / cifXIOTest.c) before building

3.2.1.1 Handle the cifX Driver Components

If the cifX driver components are included properly, the graphical component tree of the VxWorks development environment includes the new Folder <Hilscher GmbH>. This folder provides the following components:

- <cifX driver>
- <cifX PCI>
- <cifX DPM>

To build the cifX driver into the VxWorks image the <cifX driver> component must be included. For a correct initialization of the cifX driver the parameters associated with this component needs to be adjusted (e.g. base directory, poll interval and trace level; see section 4.1.1).

By including the <cifX PCI> component the driver scans for PCI based cifX devices in the system. A DPM based cifX device will be added if the <cifX DPM> component is included (Currently only one DPM card can be added via the graphical component editor). The components parameters physical address, IRQ number and the size of the dual port memory must be specified.

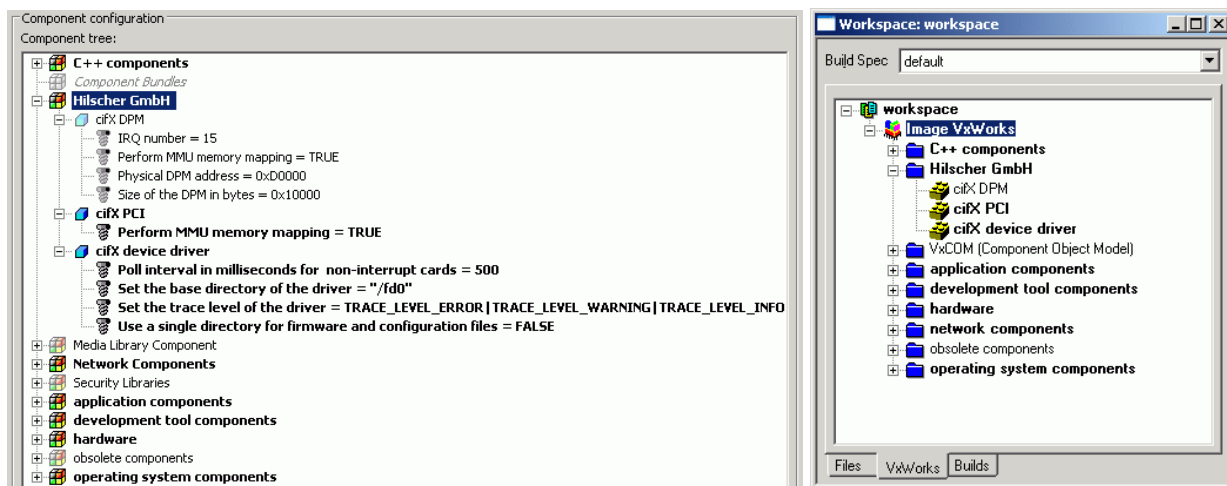


Figure 2 : Component Configuration with VxWorks 6.2 and VxWorks 5.5

As discussed in section 3.1 the target system must provide the correct memory mappings for each cifX device. The mapping of the dual port memory areas of the cifX cards is done dynamically at system startup, if the parameter <PCI_MMU>/<DPM_MMU> is enabled.

If the parameter is disabled, the area of the dual port memory must be included in the `sysPhysMemDesc[]` table (`sysLib.c`) of your BSP (see section 3.1).

3.2.1.2 Using the Test Application

After system startup cifX devices can be accessed via the application programming interface of the already known cifX Device Driver Interface [1].

Note: The example applications at the installation CD using the macro CIFXDRVINIT_STARTUP (Default setting = defined) to define whether driver initialization is already performed at startup or should be performed by the application. This enables the applications to be usable in both cases.

The following C application demonstrates the minimum functions which must be called to enable an application to work with a cifX device if the driver initialization is done at system startup.

```
#include <cifXVxWorks.h>
#include <cifXUser.h>
#include <cifXErrors.h>

/*****
/*! The main function
*   \return 0 on success
*/
*****/
int main(int argc, char* argv[])
{
    CIFXHANDLE hDriver = NULL;
    long       lRet     = CIFX_NO_ERROR;

    /* Open the cifX driver */
    lRet = xDriverOpen(&hDriver);
    if(CIFX_NO_ERROR != lRet)
    {
        printf("Error opening driver. lRet=0x%08X\r\n", lRet);
    } else
    {
        /* Work with the cifX API */

        /* Close the cifX driver */
        xDriverClose(hDriver);
    }
    return 0;
}
```

3.2.2 Initialization during Application Execution

The cifX device driver initialization can be processed in an application by calling the driver initialization routines inside the application.

Note: As the MMU is already initialized, dynamic configuration of the memory area cannot be performed at this point. For this reason the memory configuration should be done by modifying the *sysPhysMemDesc[]* table manually (see section 3.1).

The workspaces on the installation CD includes the cifX driver source to build the cifX driver as downloadable module. To initialize the driver by an application, follow this steps:

- Copy the required VxWorks workspace to your development system
- VxWorks 5.5: Open the 'VxWCIFXDriver.wsp' with the Tornado II environment
VxWorks 6.2: Open the workspace folder 'WindRiver_Workbench' with the Wind River Workbench
- Undefine the macro CIFXDRVINIT_STARTUP in the example source (cifXDrvTest.c / cifXIOTest.c)
- Rebuild the cifX driver and the included example application

Note: The example applications at the installation CD using the macro CIFXDRVINIT_STARTUP (Default setting = defined) to define whether driver initialization is already performed at startup or should be performed by the application. This enables the applications to be usable in both cases.

The following C application demonstrates the minimum functions which must be called to enable an application to work with a cifX device if device driver initialization is carried out to the application.

```
#include <cifXVxWorks.h>
#include <cifXUser.h>
#include <cifXErrors.h>

/*****
/*! The main function
*   \return 0 on success
*/
*****/
int main(int argc, char* argv[])
{
    VXW_CIFXDRV_PARAMETERS_T tDriverParams = {0};
    CIFXHANDLE                hDriver      = NULL;
    long                      lRet         = CIFX_NO_ERROR;

    /* Driver scans for all available cards */
    tDriverParams.fScanPCI      = TRUE;

    /* Set the trace level of the driver */
    tDriverParams.ulTraceLevel  = TRACE_LEVEL_ERROR;

    /* Set the base directory of the driver */
    tDriverParams.szDriverBaseDir = "/hd0/cifX";

    /* Do not use single firmware directory */
    tDriverParams.fSingleDir = FALSE;

    /* Polling intervall in milliseconds for non-interrupt cards */
    tDriverParams.ulPollInterval = 500;

    /* No DPM cards will be added to the driver */
    tDriverParams.ulUserDevCount = 0;
    tDriverParams.ptUserDevList  = NULL;

    /* Init the driver */
    cifXInitDriver(&tDriverParams);

    /* Open the cifX driver */
    lRet = xDriverOpen(&hDriver);
    if(CIFX_NO_ERROR != lRet)
    {
        printf("Error opening driver. lRet=0x%08X\r\n", lRet);
    } else
    {
        /* Work with the cifX device */
        /* ... */
        /* Close the cifX driver */
        xDriverClose(hDriver);
    }
    return 0;
}
```

3.3 Firmware and Configuration File Storage

cifX PCI 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.

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 and must be passed during driver initialization (Parameter *szDriverBaseDir*, see section 4.1.1).

To assign the firmware and configuration files to a cifX device clearly, the driver provides the options below:

- If only one cifX device needs to be supported, a predefined directory can be used by setting the driver parameter *fSingleDir* (see section 4.1.1) accordingly. The firmware and configuration file must reside in the subdirectory *FW*.
- The Slot Number serves to distinguish cifX cards from each other clearly, especially if more cifX cards are installed into the very same PC. The Slot Number must be set at the cifX card using the Rotary Switch Slot Number. While Slot Number 0 means, that the cifX card is identified via its device and serial number, values from 1 to 9 corresponds to the Slot Number 1 to 9. The firmware and configuration file must reside in the subdirectory *Slot_<1..9>*.
- If the cifX device is not equipped with a Rotary Switch or the Slot Number 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>*.

The following table describes the different subdirectory levels:

Subdirectory	Description
<BASEDIR>	Base directory (<i>szDriverBaseDir</i> , see section 4.1.1) Must be set during driver initialization. This directory must contain the second stage PCI bootloader (e.g. NETX100-BSL.bin)
FW Slot_<1..9> <Device Nr>_<Serial Nr>	Device and serial number of the device or slot number if the device provides a rotary switch. If the slot number is 0 the device and serial number is used to identify the device. The files always resides in the single directory if the corresponding option is used. Note: The configuration file (device.conf) must reside here! Note: This directory must contain the rcX base firmware if loadable modules are used.
Channel<#>	Channel specific files (loadable modules, monolithic firmware files, fieldbus database files) Note: Currently only channel 0 is supported

Table 5 : Firmware and Configuration File Storage

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

```
+ <BASEDIR>
|
|-- NETX100-BSL.BIN (second stage PCI bootloader)
|
|--+ 1250100_20217
|   |--+ Channel0
|       |--cifXdps.nxf (monolithic firmware)
|       |--config.nxd (fieldbus database)
|   |--+ Channel1
|   |--+ Channel2
|   |--+ Channel3
|   |--+ Channel4
|   |--+ Channel5
|   |-- device.conf (configuration file)
```

Sample file structure for a cifX device identified by Slot number 2 and loadable module support:

```
+ <BASEDIR>
|
|-- NETX100-BSL.BIN (second stage PCI bootloader)
|
|--+ 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 VxWorks Driver Specific Information

The VxWorks driver needs some special initialization functions and structures as described in section 3.2.

4.1 Additional Structures

Some of the VxWorks 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 VXW_CIFXDRV_PARAMETERS_T

This structure is used to initialize the cifX driver.

Element	Data Type	Description
fScanPCI	BOOL	Driver Initialization options: 0 = CIFX_DRIVER_INIT_NOSCAN / FALSE Driver does not scan for PCI cards. Cards can be added manually by ulUserDevCount and ptUserDevList parameters. 1 = CIFX_DRIVER_INIT_AUTOSCAN / TRUE Driver scans for all available PCI cards and adds them to the application.
ulTraceLevel	unsigned long	Set the trace level of the driver: 1 = TRACE_LEVEL_DEBUG 2 = TRACE_LEVEL_INFO 4 = TRACE_LEVEL_WARNING 8 = TRACE_LEVEL_ERROR
ulPollInterval	unsigned long	Polling interval in milliseconds [ms] for non-interrupt driven cards (used for Change of State (COS) signaling) 0 = 500ms default
szDriverBaseDir	const char*	Set the base directory of the driver, can be NULL to use the default of '/hd0/cifX' NULL = '/hd0/cifX' default
fSingleDir	BOOL	The driver will only look into '/szDriverBaseDir/FW' for the firmware. This can be used on single cifX OEM devices to prevent the need for a unique cifX device directory (NOTE: This only support one cifX device) 0 = FALSE
ulUserDevCount	unsigned long	Number of user cards entries in the ptUserDevList to add to the driver (e.g. if a card is connected via DPM). 0 = none
ptUserDevList	struct VXW_CIFXDRV_ DEVICEENTRY_T **	Array of user added cards. Number of entries are defined by ulUserDevCount. See section 4.1.2.

Table 6 : Structure Definition of VXW_CIFXDRV_PARAMETERS_T

4.1.2 Structure VXW_CIFXDRV_DEVICEENTRY_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	Data Type	Description
ulPhysicalAddress	unsigned long	Physical address of the DPM (this value is used to detect the PCI card linked to the DPM)
blrqNumber	unsigned char	Interrupt number
pvDPMAAddress	void*	Virtual Pointer to card DPM
ulDPMSize	unsigned long	Size of the DPM in bytes
fPCICard	BOOL	0 = FALSE Device is connected via DPM. 1 = TRUE Device is connected to PCI bus
pfnNotify	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
tBusInfo	struct VXW_CIFXDRV_DEVICEBUSINFO_T	Bus information, see section 4.1.3

Table 7 : Structure Definition of VXW_CIFXDRV_DEVICEENTRY_T

4.1.3 Structure VXW_CIFXDRV_DEVICEBUSINFO_T

Bus information structure used to store bus specific information.

Element	Data Type	Description
iBusNo	int	Bus number
iDeviceNo	int	Device number
iFuncNo	int	Function number

Table 8 : Structure Definition of VXW_CIFXDRV_DEVICEBUSINFO_T

4.2 Additional functions

This chapter describes functions which are only available for the VxWorks version of the driver. These functions are used to initialize the cifX device driver. The driver initialization can be performed inside an application or at system startup.

4.2.1 cifXInitDriver ()

This function must be called before accessing any driver function. The cifX driver initialization includes discovering all available cifX PCI devices and downloading the firmware and configuration files.

Note: If PCI scan is enabled via the driver parameter *fScanPCI* (see section 4.1.1) the *cifXInitDriver()* search and initialize every available cifX PCI card. PCI cards can be added manually, by using the *cifXFindDevice()* routine (see section 4.2.3) and calling the driver initialization routine with disabled PCI Scan.

Note: For none PCI cards, the application has to create a *VXW_CIFXDRV_DEVICEENTRY_T* with the corresponding device information. Afterwards, the structure must be handed to the *cifXInitDriver()* routine via the driver parameters *ulUserDevCount* and *ptUserDevList* (see section 4.1.1).

Function call:

```
int32_t cifXInitDriver (VXW_CIFXDRV_PARAMETERS_T* ptDriverParams);
```

Arguments:

Argument	Data Type	Description
ptDevEntry	VXW_CIFXDRV_PARAMETERS_T*	Driver parameters, see section 4.1.1

Return Values:

Return Values	
CIFX_NO_ERROR	Driver initialization successful
CIFX_DRV_INIT_ERROR	Driver initialization failed (no cifX device available)

Example:

```
VXW_CIFXDRV_PARAMETERS_T tDriverParams = {0};

/* Set driver parameters */
tDriverParams.fScanPCI = TRUE;
tDriverParams.ulUserDevCount = 0;

/* Scan for all available cifX PCI devices and initialize the cifX device driver */
cifXInitDriver (&tDriverParams);
```

4.2.2 cifXDeinitDriver()

Deinitialize the driver and remove all devices from the control of the cifX driver library. After calling this function the application must not access any cifX driver API function any more.

Function call:

```
void cifXDeinitDriver ( void);
```

Arguments:

Argument	Data Type	Description
none		

Return Values:

Return Values	
CIFX_NO_ERROR	Driver deinitialization successful
CIFX_DEV_HW_PORT_IS_USED	At least one channel has an open reference so deny deinitialization

4.2.3 cifXFindDevice()

This function scans for a cifX PCI device in the system and builds a `VXW_CIFXDRV_DEVICEENTRY_T` structure for each discovered device. The discovered device information can be handed to the driver initialization routine via the driver parameters `ulUserDevCount` and `ptUserDevList` (see 4.1.1). The PCI scan of the driver initialization routine should be disabled via the driver parameter `fScanPCI`, to prevent the discovery of every available PCI device in the system.

Function call:

```
BOOL cifXFindDevice ( VXW_CIFXDRV_DEVICEENTRY_T*   ptDevEntry,
                    int                             iNum);
```

Arguments:

Argument	Data Type	Description
ptDevEntry	VXW_CIFXDRV_DEVICEENTRY_T*	Pointer to a VXW_CIFXDRV_DEVICEENTRY_T structure, to place returned values in
iNum	int	Number of the device in the system 0 = first device

Return Values:

Return Values	
TRUE	A device with number iNum was found
FALSE	A device with number iNum could not be found

Example:

```
VXW_CIFXDRV_PARAMETERS_T  tDriverParams  = {0};
VXW_CIFXDRV_DEVICEENTRY_T tDevEntry     = {0};

/* Find the first cifX PCI device */
if (cifXFindDevice (&tDevEntry, 0))
{
    tDriverParams.fScanPCI      = FALSE;
    tDriverParams.ulUserDevCount = 1;
    tDriverParams.ptUserDevList = &tDevEntry;

    /* initialize the cifX device driver */
    cifXInitDriver (&tDriverParams);
}
```

4.2.4 cifXMemMap()

This function allows to dynamically add the memory mappings and set the memory access masks of the dual port memory. The function uses the `VXW_CIFXDRV_DEVICEENTRY_T` structure, which can be build by the `cifXFindDevice()` function.

Note: Dynamic memory mappings must be done at system startup before initialization of the MMU (`usrMmulnit()`, see `prjConfig.c` of your VxWorks image).

Function call:

```
BOOL cifXMemMap ( VXW_CIFXDRV_DEVICEENTRY_T* ptDevEntry);
```

Arguments:

Argument	Data Type	Description
ptDevEntry	VXW_CIFXDRV_DEVICEENTRY_T*	Pointer to a VXW_CIFXDRV_DEVICEENTRY_T structure, to place returned values in

Return Values:

Return Values	
CIFX_NO_ERROR	Memory mapping successful
CIFX_MEMORY_MAPPING_FAILED	Memory mapping failed

Example:

```
VXW_CIFXDRV_DEVICEENTRY_T tDevEntry = {0};
int iDevNum = 0;

/* Browse cifX devices and perform mapping of dual port memory */
while (cifXFindDevice (&tDevEntry, iDevNum) )
{
    ++iDevNum;
    cifXMemMap(&tDevEntry);
}
```

4.2.5 cifXGetDriverVersion()

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

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

4.3 Driver Startup Procedure

The driver startup procedure can be controlled by the user.

The following two use cases are available:

- Automatically add all found cifX PCI devices and optionally add user specific devices
- Skip cifX PCI device scan and only add user specific device

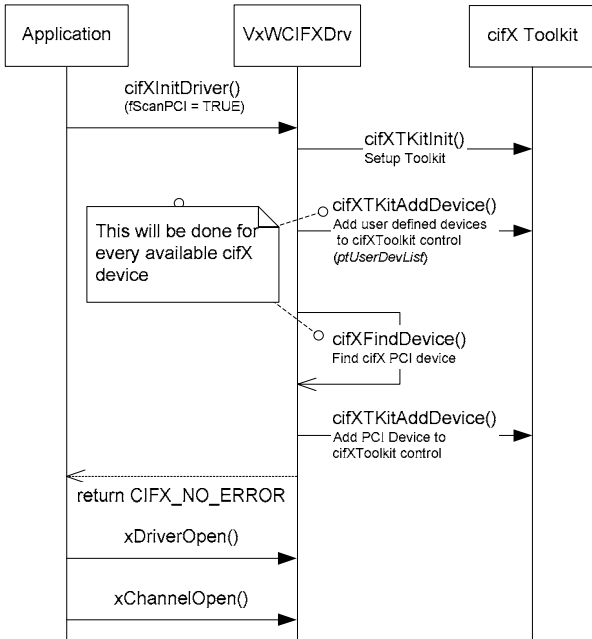


Figure 3: Initialization of the cifX Driver with fScanPCI = CIFX_DRIVER_INIT_AUTOSCAN / TRUE

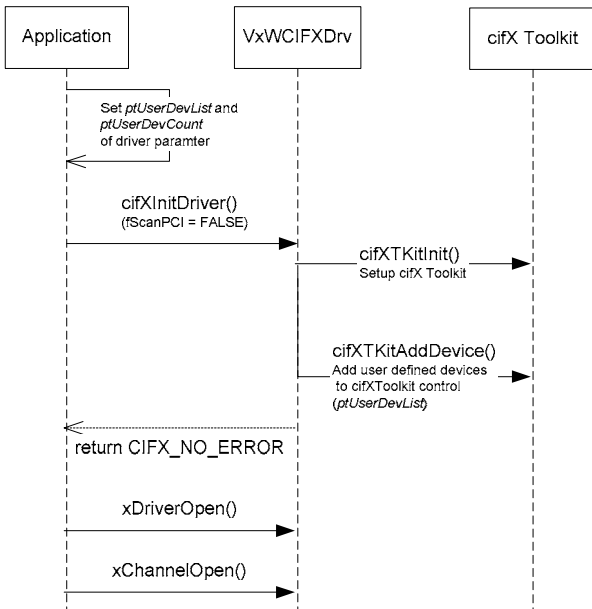


Figure 4: Initialization of the cifX Driver with fScanPCI = CIFX_DRIVER_INIT_NOSCAN / FALSE

4.4 Device Configuration (device.conf)

Each device has several parameters which can be configured via a per device configuration file. Available parameters are shown below.

The configuration file must be called '*device.conf*' and must reside in the top level directory of a device, e.g. (base directory '*/hd0/cifx*')

- '*/hd0/cifx/1250100_20217/device.conf*' if a device is identified by its device/serial number
- '*/hd0/cifx/Slot_1/device.conf*' if a device is identified by slot number 1
- '*/hd0/cifx/FW/device.conf*' if a single directory is used

Note: Currently DMA support is only provided by cifX50 and netPLC cards.

The file may contain the following keys:

Key	Data Type	Description
alias	char[16]	Alias name for the device. Must be less than 16 characters.
irq	string	'yes' = enable IRQ on the device 'no' = disable IRQ on the device
dma	string	'on' = Switch channels into DMA mode 'off' = Switch off DMA mode for all channels 'leave' = Leave communication channels in actual mode

Table 9 : Device Configuration Parameters - *device.conf*

Sample device.conf:

```
#Sample device configuration file
alias=PROFIBUS
irq=yes
dma=on
```

5 Programming with the cifX VxWorks Driver

The Application Programming Interface (API) of the Hilscher VxWorks driver is based on the already known [1]. Therefore the cifX Device Driver manual can be used. This manual describes the driver functions (API), error codes and shows some program examples. The installation CD also includes a '*cifXDrvTest*' directory with a VxWorks specific example.

6 Appendix

6.1 List of Tables

Table 1: List of Revisions	4
Table 2: Terms, Abbreviations and Definitions	4
Table 3: References	4
Table 4 : CD Contents.....	6
Table 5 : Firmware and Configuration File Storage.....	18
Table 6 : Structure Definition of VXW_CIFXDRV_PARAMETERS_T	20
Table 7 : Structure Definition of VXW_CIFXDRV_DEVICEENTRY_T	21
Table 8 : Structure Definition of VXW_CIFXDRV_DEVICEBUSINFO_T.....	21
Table 9 : Device Configuration Parameters - device.conf	28

6.2 List of Figures

Figure 1 : VxWorks cifX Driver Architecture	3
Figure 2 : Component Configuration with VxWorks 6.2 and VxWorks 5.5	14
Figure 3: Initialization of the cifX Driver with fScanPCI = CIFX_DRIVER_INIT_AUTOSCAN / TRUE.....	27
Figure 4: Initialization of the cifX Driver with fScanPCI = CIFX_DRIVER_INIT_NOSCAN / FALSE.....	27

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 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