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 ......................................................................................................................................... 5
   1.5 Requirement ....................................................................................................................................... 6
   1.6 Supported Hardware ............................................................................................................................ 6
   1.7 Features ............................................................................................................................................ 6
   1.8 Limitations ................................................................................................................................-------- 6
   1.9 CD Contents ..................................................................................................................................... 7
   1.10 Legal Notes ..................................................................................................................................... 8
       1.10.1 Copyright .................................................................................................................................. 8
       1.10.2 Important Notes ......................................................................................................................... 8
       1.10.3 Exclusion of Liability ................................................................................................................ 9
       1.10.4 Export ...................................................................................................................................... 9

2 Licensing Terms .................................................................................................................................... 10

3 VxWorks 7 ............................................................................................................................................. 11
   3.1 Building the cifX Driver ..................................................................................................................... 11
       3.1.1 Initialization at System Startup ................................................................................................. 12
       3.1.2 Create the driver as a DKM from the Workbench ..................................................................... 15
       3.1.3 Add the CIFX driver interface to an application ...................................................................... 15
   3.2 Driver Specific Information .............................................................................................................. 17
       3.2.1 registerCIFXDrv () .................................................................................................................... 17

4 VxWorks 5.x/6.x .................................................................................................................................... 18
   4.1 Installation ........................................................................................................................................ 18
       4.1.1 MMU Support and Settings ........................................................................................................ 18
       4.1.2 Building the cifX Driver ............................................................................................................. 20
   4.2 VxWorks Driver Specific Information ............................................................................................. 26
       4.2.1 Additional Structures ................................................................................................................ 26
       4.2.2 Additional functions .................................................................................................................... 28
       4.2.3 Driver Startup Procedure ......................................................................................................... 33

5 Firmware and Configuration File Storage .............................................................................................. 34

6 Device Configuration (device.conf) ...................................................................................................... 36

7 Programming with the cifX VxWorks Driver ....................................................................................... 37

8 Appendix .............................................................................................................................................. 38
   8.1 List of Tables ................................................................................................................................... 38
   8.2 List of Figures ................................................................................................................................... 38
   8.3 Contacts ......................................................................................................................................... 39
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 and later) 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 either as a library (VxWorks 5.x/6.x) or as a VxBUS GEN 2 driver (VxWorks 7.0) 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](image-url)
1.2 List of Revisions

<table>
<thead>
<tr>
<th>Rev</th>
<th>Date</th>
<th>Name</th>
<th>Chapter</th>
<th>Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2009-06-09</td>
<td>SS</td>
<td>All</td>
<td>Created</td>
</tr>
<tr>
<td>2</td>
<td>2010-05-31</td>
<td>SS</td>
<td>5, 6</td>
<td>Support for loadable modules</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5, 4.2.1.1, 6</td>
<td>Support for slot numbers</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>Driver parameter fSingleDir added to support single firmware directory</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4.2.2</td>
<td>Update screenshot</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4.2.2.2</td>
<td>Return types of additional functions adapted to stdint data types</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4.2.2.5</td>
<td>Description for new function cifXGetDriverVersion() added</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td>DMA mode configuration added</td>
</tr>
<tr>
<td>3</td>
<td>2014-11-21</td>
<td>RM</td>
<td>1.4</td>
<td>Support of VxWorks 6.7/6.9 added.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SS</td>
<td>1.6</td>
<td>Information about CIFX API manual added</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1.9</td>
<td>Section Supported Hardware added.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4.1.2.1/4.1.2.2</td>
<td>Installation instructions adapted to new driver project structure.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td>Channel directory is case sensitive.</td>
</tr>
<tr>
<td>4</td>
<td>2015-07-27</td>
<td>RM</td>
<td>1.5, 1.6, 1.7, 1.8</td>
<td>Added VxWorks 7.0 Support</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LC</td>
<td>1.9</td>
<td>Sections Requirement, Supported Hardware, Features and Limitations updated</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>Section CD Contents updated</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5, 6</td>
<td>Chapter VxWorks 7 added</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Restructure document (VxWorks version independent chapters added)</td>
</tr>
</tbody>
</table>

Table 1: List of Revisions

1.3 Terms, Abbreviations and Definitions

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cifX</td>
<td>Communication Interface based on netX</td>
</tr>
<tr>
<td>comX</td>
<td>Communication Module based on netX</td>
</tr>
<tr>
<td>PCI</td>
<td>Peripheral Component Interconnect</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>DPM</td>
<td>Dual-Port Memory</td>
</tr>
<tr>
<td></td>
<td>Physical interface to all communication board (DPM is also used for PROFIBUS-DP Master).</td>
</tr>
<tr>
<td>CDF</td>
<td>Component Description File</td>
</tr>
<tr>
<td>BSP</td>
<td>Board Support Package</td>
</tr>
<tr>
<td>DMA</td>
<td>Direct Memory Access</td>
</tr>
</tbody>
</table>

Table 2: Terms, Abbreviations and Definitions
1.4 References

This document is based on the following documents:


Table 3: References
1.5 Requirement
- VxWorks V5.5/V6.2/V6.7/V6.9
- VxWorks V7.0

1.6 Supported Hardware
- VxWorks V5.x/6.x:
  cifX board (PCI / PCIe), netJACK 100, netPLC
  NXSB-PCA / NXSB100 / NXHX board
  NX-PCA-PCI / NXHX
  CIFX 104 (ISA)
- VxWorks V7.0:
  cifX board (PCI / PCIe), netJACK 100, netPLC

1.7 Features
- VxWorks 5.x/6.x: Based on the cifX Toolkit source V1.2.0.1
- VxWorks 7.0: Based on the cifX Toolkit source V1.3.0.0
- 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

1.8 Limitations
- No DMA support for NXSB-PCA, NX-PCA-PCI and CIFX104 boards
- VxWorks 5.x/6.x:
  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
- VxWorks 7.0:
  CifX Driver for only supports netX based PCI and PCIe hardware
## 1.9 CD Contents

<table>
<thead>
<tr>
<th>Folder</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSL</td>
<td>netX bootloader images</td>
</tr>
<tr>
<td>Diagnostic and Remote Access</td>
<td>Windows based remote diagnose client</td>
</tr>
<tr>
<td>Documentation</td>
<td>Driver documentation</td>
</tr>
<tr>
<td><strong>VxWorks_5_6</strong></td>
<td></td>
</tr>
<tr>
<td>API</td>
<td>Header files for cifX application development</td>
</tr>
<tr>
<td>Component</td>
<td>Source files for integration of cifX driver into the VxWorks image</td>
</tr>
<tr>
<td>comp</td>
<td></td>
</tr>
<tr>
<td>src</td>
<td>Component description file</td>
</tr>
<tr>
<td>vxWorks</td>
<td>Configlette file</td>
</tr>
<tr>
<td>Hilscher</td>
<td>Sources for VxWorks driver component</td>
</tr>
<tr>
<td>cifXDrv</td>
<td>Driver sources</td>
</tr>
<tr>
<td>cifXToolkit</td>
<td>cifX Toolkit sources</td>
</tr>
<tr>
<td>Driver</td>
<td>cifX driver sources and project file</td>
</tr>
<tr>
<td>cifXToolkit</td>
<td>cifX Toolkit sources</td>
</tr>
<tr>
<td><strong>Examples</strong></td>
<td></td>
</tr>
<tr>
<td>Basedir</td>
<td>Example card configuration directory (copy to your own base directory)</td>
</tr>
<tr>
<td>cifXDrvTest</td>
<td>cifX test application</td>
</tr>
<tr>
<td>cifXTCPServer</td>
<td>cifX TCP server application for remote diagnose</td>
</tr>
<tr>
<td><strong>VxWorks_7</strong></td>
<td></td>
</tr>
<tr>
<td>cifXAPI</td>
<td>Header files for cifX application development</td>
</tr>
<tr>
<td>cifXDrvInterface</td>
<td>Driver Interface source files for cifX application development</td>
</tr>
<tr>
<td>Driver</td>
<td>cifX driver sources and project file</td>
</tr>
<tr>
<td>cifXToolkit</td>
<td>cifX Toolkit sources</td>
</tr>
<tr>
<td>Driver_image_components</td>
<td>Source files for integration of cifX driver into the VxWorks image</td>
</tr>
<tr>
<td>cdf</td>
<td>Component definition file</td>
</tr>
<tr>
<td>h</td>
<td></td>
</tr>
<tr>
<td>hwif</td>
<td></td>
</tr>
<tr>
<td>drv</td>
<td></td>
</tr>
<tr>
<td>cifx</td>
<td>Component parameter definition file</td>
</tr>
<tr>
<td>src</td>
<td>Sources for VxWorks driver component</td>
</tr>
<tr>
<td>cifx</td>
<td>Driver sources</td>
</tr>
<tr>
<td>cifXToolkit</td>
<td>cifX Toolkit sources</td>
</tr>
<tr>
<td><strong>Examples</strong></td>
<td></td>
</tr>
<tr>
<td>Basedir</td>
<td>Example card configuration directory (copy to your own base directory)</td>
</tr>
<tr>
<td>cifXDrvTest</td>
<td>cifX test application</td>
</tr>
<tr>
<td>cifXDrvTest_console</td>
<td>cifX test console application</td>
</tr>
<tr>
<td>cifXTCPServer</td>
<td>cifX TCP server application for remote diagnose</td>
</tr>
</tbody>
</table>

*Table 4: CD Contents*
1.10 Legal Notes

1.10.1 Copyright

© 2009-2014 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.

1.10.2 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.
1.10.3 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 counters, 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 VxWorks 7

Wind River has separated VxWorks 7 into user space and kernel space. The cifX Driver for VxWorks 7 is implemented as a VxBUS GEN 2 driver and can be accessed by IO Control calls.

Note: cifX Driver for VxWorks 7 currently supports only netX based PCI and PCIe hardware.

3.1 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 Driver Specific Information on page 17. 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 runtime, the driver can be started as a DKM 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 build with the development environment. How to integrate and build the driver for the VxWorks image is described in section Initialization at System Startup on page 12.

- Create the driver as a DKM from the Workbench

  If the driver should be started by as a DKM application the driver must be build as a downloadable module via the development environment, described in section Create the driver as a DKM from the Workbench on page 15.
3.1.1 Initialization at System Startup

The VxWorks IDE (Wind River Workbench) provides facilities to arrange and configure system components to build a customized VxWorks image [4]. 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 configlette file and a Component Description File, which describes the cifX driver component has to be placed into the VxWorks installation tree.

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

- Make sure Wind River Workbench development environment is closed
- Copy the content of the subdirectory 'Driver-image-components' into an existing driver package directory
- Add the cifx driver to the existing driver package makefile by opening ‘./WindRiver/VxWorks-7/pkg/os/drv/vxbus/drv-x.x.x.x/src/Makefile’ and add “cifx” to the following line of the make file:
  
  SUBDIRS = intCtlr i2c spi gpio timer dma busCtlr sio resource sensor cifx

- Add the actual target directory path to the driver “makefile”:
  open the "cifx" driver makefile under ‘./WindRiver/VxWorks-7/pkg/os/drv/vxbus/drv-x.x.x.x/src/cifx/Makefile’ and change the target directory path corresponding to your version e.g.
  
  TGT_DIR = $(WIND_BASE)/pkg/os/drv/vxbus/drv-x.x.x.x/

- Add the actual target directory path to the toolkit makefile:
  open the "cifXTtoolkit” makefile under ‘./WindRiver/VxWorks-7/pkg/os/drv/vxbus/drv-x.x.x.x/src/cifx/cifXToolkit/Makefile’ and change the target directory path corresponding to your version e.g.
  
  TGT_DIR = $(WIND_BASE)/pkg/os/drv/vxbus/drv-x.x.x.x/

- Create your VxWorks image, open the kernel configuration, include the required components for the cifX driver and rebuild the image
- Create a new Real Time Process or Downloadable Kernel Module -Workbench project based on your VIP project
- Copy the content of the ‘./VxWorks_7/Examples/cifXDrvTest’ into the created workspace.
- Use the workspace project ‘Properties -> Build Properties -> Paths’ settings to add these additional include path settings, e.g. (base directory ‘C:\WindRiver\workspace’):
  
  IC:/WindRiver/workspace/<Your-Project-Name>
  IC:/WindRiver/workspace/<Your-Project-Name>/cifXAPI
  IC:/WindRiver/workspace/<Your-Project-Name>/cifXDrvInterface

- Build the example application
### 3.1.1.1 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 component `<Hilscher cifX Device Driver>` in folder `<hardware (default) / Device Drivers>`.

To build the cifX driver into the VxWorks image the `<Hilscher cifX Device 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 registerCIFXDrv () on page 17).

**Components**

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Drivers</td>
<td>FOLDER_DRIVERS</td>
<td>SELECT_PC_CONSOLE_K..</td>
<td></td>
</tr>
<tr>
<td>PC Console keyboard driver</td>
<td>SELECT_PC_CONSOLE_V..</td>
<td>DRI_ADIUCHD</td>
<td></td>
</tr>
<tr>
<td>PC Console video driver</td>
<td></td>
<td>DRI_ADIUCHD</td>
<td></td>
</tr>
<tr>
<td>ADDITTIFramework Devices</td>
<td></td>
<td>DRI_ADIUCHD</td>
<td></td>
</tr>
<tr>
<td>COMIFAC Interrupt Controller Driver</td>
<td></td>
<td>DRI_ADIUCHD</td>
<td></td>
</tr>
<tr>
<td>DMA support for Finisar xDMA compatible devices</td>
<td>DRI_DMA_FDI_FSL_6MM</td>
<td>DRI_ADIUCHD</td>
<td></td>
</tr>
<tr>
<td>POST Bus driver</td>
<td></td>
<td>DRI_ADIUCHD</td>
<td></td>
</tr>
<tr>
<td>POST memory driver</td>
<td></td>
<td>DRI_ADIUCHD</td>
<td></td>
</tr>
<tr>
<td>POSTalec FlexI支配 Driver</td>
<td>DRI_FSL_FDI_FSL_FSM</td>
<td>DRI_ADIUCHD</td>
<td></td>
</tr>
<tr>
<td>POSTalec FDC bus controller driver</td>
<td>DRI_FDC</td>
<td>DRI_ADIUCHD</td>
<td></td>
</tr>
<tr>
<td>POSTalec SPI bus device</td>
<td>DRI_ADIUCHD</td>
<td>DRI_ADIUCHD</td>
<td></td>
</tr>
<tr>
<td>Hilscher cifX Device Driver</td>
<td>DRY_HILSCHER_CIF</td>
<td>DRY_HILSCHER_CIF</td>
<td></td>
</tr>
<tr>
<td>Poll interval in milliseconds for non-interrupt cards</td>
<td>POLL_INTERVAL</td>
<td>500</td>
<td></td>
</tr>
<tr>
<td>Set the base directory of the driver</td>
<td>BASE_DIR</td>
<td>string</td>
<td><code>/var/cifX</code></td>
</tr>
<tr>
<td>Set the trace level of the driver</td>
<td>TRACELVL</td>
<td>string</td>
<td><code>/var/cifX</code></td>
</tr>
<tr>
<td>Use a single firmware directory</td>
<td>SINGLE_DIR</td>
<td>BOOL</td>
<td>TRUE</td>
</tr>
<tr>
<td>D2C IPMC4 firmware driver</td>
<td>DRI_IPMC4</td>
<td>DRI_ADIUCHD</td>
<td></td>
</tr>
<tr>
<td>D2C GPIO Driver for MAIN/1X</td>
<td>DRI_GPIO_FDI_M4 ...</td>
<td>DRI_ADIUCHD</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2 : Component Configuration with VxWorks 7.0**
3.1.1.2 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].

The following C application demonstrates the minimum functions which must be called to enable an application to work with a cifX device.

```c
#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;
}
```
3.1.2 Create the driver as a DKM from the Workbench

The cifX device driver can be started as a Downloadable Kernel Module application.

- Create a new VxWorks workspace or use an existing one on your development system
- Create a new DKM-Workbench project based on your VIP project
- Copy the content of the ‘./VxWorks_7/Driver’ into the created DKM workspace.
- Use the Workbench DKM project ‘Properties -> Build Properties -> Paths’ settings to add two additional include path settings, e.g. (base directory ‘C:\WindRiver\workspace\’):
  IC:\WindRiver\workspace\<Your-Project-Name>\
  IC:\WindRiver\workspace\<Your-Project-Name>\cifXToolkit
- Use the Workbench DKM project properties ‘Properties -> Build Properties -> Defines’ to add additional compiler defines described in the cifX VxWorks driver manual e.g. -DCIFX_TOOLKIT_DMA / -DCIFX_TOOLKIT_ENABLE_DSR_LOCK
- Build the cifX driver

3.1.3 Add the CIFX driver interface to an application

The cifX device driver can be used in a DKM or RTP application by calling the cifX API routines inside the application.

The installation CD includes the cifX driver interface sources to integrate the cifX driver. To use the cifX API, follow these steps:

- Copy the files from the ‘cifXAPI’ directory into your DKM or RTP Workbench project
- Copy ‘cifXDrvInterface’ directory to your Workbench project directory
- Use the Workbench DKM project ‘Properties -> Build Properties -> Paths’ settings to add these additional include path settings, e.g. (base directory ‘C:\WindRiver\workspace\’):
  IC:\WindRiver\workspace\<Your-Project-Name>\
  IC:\WindRiver\workspace\<Your-Project-Name>\cifXAPI\
  IC:\WindRiver\workspace\<Your-Project-Name>\cifXDrvInterface
- Include the ‘cifXUser.h’ into your application source module
- Call the cifX driver functions described in [1]

**Note:** You can also use the ‘cifXAPI’ from an own subdirectory of your Workbench project. Therefore copy the whole ‘cifXAPI’ directory to your project directory and add another include path to the ‘cifXAPI’ directory into the project properties.
The following C application demonstrates the minimum functions which must be called to enable an application to work with a cifX device.

```c
#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 Driver Specific Information

3.2.1 registerCIFXDrv()

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

Function call:

```c
STATUS registerCIFXDrv (BOOL            fSingleDir,
                         unsigned long   ulTraceLevel,
                         unsigned long   ulPollInterval,
                         char*           szDriverBaseDir);
```

**Arguments:**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fSingleDir</td>
<td>BOOL</td>
<td>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)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = FALSE</td>
</tr>
<tr>
<td>ulTraceLevel</td>
<td>unsigned long</td>
<td>Set the trace level of the driver: 1 = TRACE_LEVEL_DEBUG, 2 = TRACE_LEVEL_INFO, 4 = TRACE_LEVEL_WARNING, 8 = TRACE_LEVEL_ERROR</td>
</tr>
<tr>
<td>ulPollInterval</td>
<td>unsigned long</td>
<td>Polling interval in milliseconds [ms] for non-interrupt driven cards (used for Change of State (COS) signaling) default = 500ms</td>
</tr>
<tr>
<td>szDriverBaseDir</td>
<td>char*</td>
<td>Set the base directory of the driver, can be NULL to use the default directory setting default = '/hd0/cifX'</td>
</tr>
</tbody>
</table>

**Return Values:**

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK</td>
<td>Driver initialization successful</td>
</tr>
<tr>
<td>ERROR</td>
<td>Driver initialization failed (no cifX device available)</td>
</tr>
</tbody>
</table>

**Example:**

```c
registerCIFXDrv ( FALSE, /* fSingleDir, */
                 TRACE_LEVEL_ERROR, /* ulTraceLevel, */
                 500, /* ulPollInterval, */
                 "/bd0/cifX" /* szDriverBaseDir */);
```
4 VxWorks 5.x/6.x

4.1 Installation

The installation CD includes project files for the development environment Tornado II (VxWorks 5.5) and the development environment Wind River Workbench (VxWorks 6.2/6.7/6.9). In addition to the driver source, both projects contain example applications.

4.1.1 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.

```c
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 diver provides the function *cifXMemMap()* to dynamically add a memory area definition for a certain cifX device (see section *cifXMemMap()* on page 31).

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

```c
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);
}
```
4.1.2 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 VxWorks Driver Specific Information on page 26. 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 build with the development environment. How to integrate and build the driver for the VxWorks image is described in section Initialization at System Startup on page 21.

- 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 Initialization during Application Execution on page 24.
4.1.2.1 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 configlette file and a Component Description File, which describes the cifX driver component has to be placed into the VxWorks installation tree.

Follow these 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 ‘Component/Hilscher’ directory from the driver CD to your VxWorks installation tree ‘installdir/target/config’ (VxWorks 6.9: ‘installdir/target/3rdparty’)
- Place the component description file and the configlette file by copying the whole ‘Component/comps’ directory from the driver CD to your VxWorks installation tree: ‘installdir/target/config’ (VxWorks 6.9: ‘installdir/target/3rdparty’)
- 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’
  - VxWorks 6.7: execute ‘installdir/wrenv.exe –p vxworks-6.7’
  - VxWorks 6.9: execute ‘installdir/wrenv.exe –p vxworks-6.9’
- Navigate to the ‘installdir/target/config/Hilscher/cifXDrv’ (VxWorks 6.9: installdir/target/3rdparty/Hilscher/cifXDrv) directory in your VxWorks installation tree and type ‘make CPU=PENTIUM4’ (use your CPU architecture here)
- Create your VxWorks image, open the kernel configuration, include the required components for the cifX driver and rebuild the image
- VxWorks 5.5: Import the example project ‘Examples/cifXDrvTest/cifXDrvTest.wpj’ to your Tornado II workspace
  - VxWorks 6.2/6.7/6.9: Import the example project ‘Examples/cifXDrvTest’ to your Wind River Workbench workspace
- Build the example application.
  - Define the macro CIFXDRVINIT_STARTUP (default) in the example source (cifXDrvTest.c) before building
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 Structure VXW_CIFXDRV_PARAMETERS_T on page 26).

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.

As discussed in section MMU Support and Settings on page 18 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 MMU Support and Settings on page 18).
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.

```c
#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\n", lRet);
  } else {
    /* Work with the cifX API */
    /* Close the cifX driver */
    xDriverClose(hDriver);
  }
  return 0;
}
```
4.1.2.2 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 **MMU Support and Settings** on page 18).

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

- Create a new VxWorks workspace or use an existing one on your development system
- VxWorks 5.5: Import the ‘Driver/VxWCIFXDriver.wpj’ project to your Tornado II workspace
  VxWorks 6.2/6.7/6.9: Import the driver project folder ‘Driver’ to your Wind River Workbench workspace
- Undefine the macro `CIFXDRVINIT_STARTUP` in the example source (cifXDrvTest.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.

```c
#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;
}
```
## 4.2 VxWorks Driver Specific Information

The VxWorks driver needs some special initialization functions and structures as described in section *Building the cifX Driver* on page 20.

### 4.2.1 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.

#### 4.2.1.1 Structure VXW_CIFXDRV_PARAMETERS_T

This structure is used to initialize the cifX driver.

<table>
<thead>
<tr>
<th>Element</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fScanPCI</td>
<td>BOOL</td>
<td>Driver Initialization options: Driver does not scan for PCI cards. Cards can be added manually by ulUserDevCount and ptUserDevList parameters.</td>
</tr>
<tr>
<td>ulTraceLevel</td>
<td>unsigned long</td>
<td>Set the trace level of the driver: 1 = TRACE_LEVEL_DEBUG  2 = TRACE_LEVEL_INFO  4 = TRACE_LEVEL_WARNING  8 = TRACE_LEVEL_ERROR</td>
</tr>
<tr>
<td>ulPollInterval</td>
<td>unsigned long</td>
<td>Polling interval in milliseconds [ms] for non-interrupt driven cards (used for Change of State (COS) signaling) 0 = 500ms default</td>
</tr>
<tr>
<td>szDriverBaseDir</td>
<td>const char*</td>
<td>Set the base directory of the driver, can be NULL to use the default of '/hd0/cifX' NULL = '/hd0/cifX' default</td>
</tr>
<tr>
<td>fSingleDir</td>
<td>BOOL</td>
<td>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</td>
</tr>
<tr>
<td>ulUserDevCount</td>
<td>unsigned long</td>
<td>Number of user cards entries in the ptUserDevList to add to the driver (e.g. if a card is connected via DPM) 0 = none</td>
</tr>
<tr>
<td>ptUserDevList</td>
<td>struct VXW_CIFXDRV_DEVICEENTRY_T **</td>
<td>Array of user added cards. Number of entries are defined by ulUserDevCount. See section 4.2.1.2.</td>
</tr>
</tbody>
</table>

*Table 5 : Structure Definition of VXW_CIFXDRV_PARAMETERS_T*
4.2.1.2 Structure VXW_CIFXDRVDEVICEENTRY_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.

<table>
<thead>
<tr>
<th>Element</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ulPhysicalAddress</td>
<td>unsigned long</td>
<td>Physical address of the DPM (this value is used to detect the PCI card linked to the DPM)</td>
</tr>
<tr>
<td>blrqNumber</td>
<td>unsigned char</td>
<td>Interrupt number</td>
</tr>
<tr>
<td>pvDPMAddress</td>
<td>void*</td>
<td>Virtual Pointer to card DPM</td>
</tr>
<tr>
<td>ulDPMSize</td>
<td>unsigned long</td>
<td>Size of the DPM in bytes</td>
</tr>
</tbody>
</table>
| 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_CIFXDRVDEVICEBUSINFO_T | Bus information, see section 4.2.1.3 |

Table 6: Structure Definition of VXW_CIFXDRVDEVICEENTRY_T

4.2.1.3 Structure VXW_CIFXDRVDEVICEBUSINFO_T

Bus information structure used to store bus specific information.

<table>
<thead>
<tr>
<th>Element</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iBusNo</td>
<td>int</td>
<td>Bus number</td>
</tr>
<tr>
<td>iDeviceNo</td>
<td>int</td>
<td>Device number</td>
</tr>
<tr>
<td>iFuncNo</td>
<td>int</td>
<td>Function number</td>
</tr>
</tbody>
</table>

Table 7: Structure Definition of VXW_CIFXDRVDEVICEBUSINFO_T
4.2.2 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.

4.2.2.1 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 Structure VXW_CIFXDRV_PARAMETERS_T on page 26) the cifXInitDriver() search and initialize every available cifX PCI card. PCI cards can be added manually, by using the cifXFindDevice() routine (see section cifXFindDevice() on page 30) 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 Structure VXW_CIFXDRV_PARAMETERS_T on page 26).

**Function call:**

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

**Arguments:**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ptDevEntry</td>
<td>VXW_CIFXDRV_PARAMETERS_T*</td>
<td>Driver parameters, see section 4.2.1.1</td>
</tr>
</tbody>
</table>

**Return Values:**

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFX_NO_ERROR</td>
<td>Driver initialization successful</td>
</tr>
<tr>
<td>CIFX_DRV_INIT_ERROR</td>
<td>Driver initialization failed (no cifX device available)</td>
</tr>
</tbody>
</table>

**Example:**

```c
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);
```
4.2.2.2 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:

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

Arguments:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values:

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFX_NO_ERROR</td>
<td>Driver deinitialization successful</td>
</tr>
<tr>
<td>CIFX_DEV_HW_PORT_IS_USED</td>
<td>At least one channel has an open reference so deny deinitialization</td>
</tr>
</tbody>
</table>
4.2.2.3  cifXFindDevice()

This function scans for a cifX PCI device in the system and builds a VXW_CIFXDRVDEVICEENTRY_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 section Structure VXW_CIFXDRV_PARAMETERS_T on page 26). 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:

```c
BOOL cifXFindDevice ( VXW_CIFXDRVDEVICEENTRY_T* ptDevEntry,
                       int iNum);
```

Arguments:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ptDevEntry</td>
<td>VXW_CIFXDRVDEVICEENTRY_T*</td>
<td>Pointer to a VXW_CIFXDRVDEVICEENTRY_T structure, to place returned values in</td>
</tr>
<tr>
<td>iNum</td>
<td>int</td>
<td>Number of the device in the system 0 = first device</td>
</tr>
</tbody>
</table>

Return Values:

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>A device with number iNum was found</td>
</tr>
<tr>
<td>FALSE</td>
<td>A device with number iNum could not be found</td>
</tr>
</tbody>
</table>

Example:

```c
VXW_CIFXDRV_PARAMETERS_T tDriverParams  = {0};
VXW_CIFXDRVDEVICEENTRY_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);
}
```
4.2.2.4  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 (usrMmuInit(), see prjConfig.c of your VxWorks image).

Function call:

BOOL cifXMemMap ( VXW_CIFXDRV_DEVICEENTRY_T*  ptDevEntry);

Arguments:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ptDevEntry</td>
<td>VXW_CIFXDRV_DEVICEENTRY_T*</td>
<td>Pointer to a VXW_CIFXDRV_DEVICEENTRY_T structure, to place returned values in</td>
</tr>
</tbody>
</table>

Return Values:

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFX_NO_ERROR</td>
<td>Memory mapping successful</td>
</tr>
<tr>
<td>CIFX_MEMORY_MAPPING_FAILED</td>
<td>Memory mapping failed</td>
</tr>
</tbody>
</table>

Example:

VXW_CIFXDRVDEVICEENTRY_T tDevEntry = {0};
int iDevNum = 0;
/* Browse cifX devices and perform mapping of dual port memory */
while (cifXFindDevice(&tDevEntry, iDevNum) )
{
    ++iDevNum;
    cifXMemMap(&tDevEntry);
}
4.2.2.5 cifXGetDriverVersion()

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

Function call:

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

Arguments:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ulSize</td>
<td>unsigned long</td>
<td>Size of buffer referenced by parameter szVersion</td>
</tr>
<tr>
<td>szVersion</td>
<td>char*</td>
<td>Buffer to return driver version string</td>
</tr>
</tbody>
</table>

Return Values:

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFX_NO_ERROR</td>
<td>Memory mapping successful</td>
</tr>
<tr>
<td>CIFX_INVALID_BUFFERSIZE</td>
<td>Size of supplied buffer is too small</td>
</tr>
</tbody>
</table>
4.2.3 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

![Diagram of driver startup procedure]

Figure 5: Initialization of the cifX Driver with fScanPCI = CIFX_DRIVER_INIT_AUTOSCAN / TRUE

![Diagram of driver startup procedure]

Figure 6: Initialization of the cifX Driver with fScanPCI = CIFX_DRIVER_INIT_NOSCAN / FALSE
5 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 folders reside under a global base folder and must be passed during driver initialization (Parameter szDriverBaseDir, for VxWorks 5.x/6.x see section Structure VXW_CIFXDRV_PARAMETERS_T on page 26, for VxWorks 7 see section registerCIFXDrv () on page 17).

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 (for VxWorks 5.x/6.x see section Structure VXW_CIFXDRV_PARAMETERS_T on page 26, for VxWorks 7 see section registerCIFXDrv () on page 17) 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:

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

Table 8 : 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)
  |  ```
  |  ```  1250100_20217
  |  ```    ```
  |  ```    ``` channel0
  |  ```    ```   ```-- nx100dpm.nxo (loadable module)
  |  ```    ```   ```-- config.nxd (fieldbus database)
  |  ```    ```   ```-- channel1
  |  ```    ```    ```-- channel2
  |  ```    ```    ```-- channel3
  |  ```    ```    ```-- channel4
  |  ```    ```    ```-- channel5
  |  ```-- device.conf (configuration file)
  |  ```-- cifXrcX.nxf (rcX base firmware)
```
6 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 cifX and netPLC cards.

The file may contain the following keys:

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

Table 9 : Device Configuration Parameters - device.conf

Sample device.conf:

```plaintext
#Sample device configuration file
alias=PROFIBUS
irq=no
dma=on
```
7 Programming with the cifX VxWorks Driver

The Application Programming Interface (API) of the Hilscher VxWorks driver is based on the already known CIFX API manual [1].

The CIFX API manual describes the API functions, error codes and shows some program examples. The installation CD also includes an example directory ('Examples/cifXDrvTest') containing a VxWorks specific example.
8 Appendix

8.1 List of Tables

Table 1: List of Revisions ................................................................. 4
Table 2: Terms, Abbreviations and Definitions .................................. 4
Table 3: References ............................................................................ 5
Table 4 : CD Contents ........................................................................ 7
Table 5 : Structure Definition of VXW_CIFXDRV_PARAMETERS_T ........ 26
Table 6 : Structure Definition of VXW_CIFXDRV_DEVICEENTRY_T .......... 27
Table 7 : Structure Definition of VXW_CIFXDRV_DEVICEBUSINFO_T ........ 27
Table 8 : Firmware and Configuration File Storage ............................... 34
Table 9 : Device Configuration Parameters - device.conf .......................... 36

8.2 List of Figures

Figure 1: VxWorks cifX Driver Architecture .................................................. 3
Figure 2 : Component Configuration with VxWorks 7.0 ................................. 13
Figure 3 : Workbench application with CIFX API and driver interface included .......................................................... 16
Figure 4: Component Configuration with VxWorks 6.2/6.7/6.9 and VxWorks 5.5 ............................................. 22
Figure 5: Initialization of the cifX Driver with fScanPCI = CIFX_DRIVER_INIT_AUTOSCAN / TRUE ..................... 33
Figure 6: Initialization of the cifX Driver with fScanPCI = CIFX_DRIVER_INIT_NOSCAN / FALSE ..................... 33
8.3 Contacts

Headquarters

Germany
Hilscher Gesellschaft für Systemautomation mbH
Rheinstrasse 15
65795 Hattersheim
Phone: +49 (0) 6190 9907-0
Fax: +49 (0) 6190 9907-50
E-Mail: info@hilscher.com
Support
Phone: +49 (0) 6190 9907-99
E-Mail: de.support@hilscher.com

Subsidiaries

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

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

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

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

Switzerland
Hilscher Swiss GmbH
4500 Solothurn
Phone: +41 (0) 32 623 6633
E-Mail: info@hilscher.ch
Support
Phone: +1 630-505-5301
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