SIEMENS

Quick Start
PROFINET Driver V2.2

Getting Started

04/2020
A5E42795137-AB

Legal information

Warning notice system

This manual contains notices you have to observe in order to ensure your personal safety, as well as to prevent damage to property. The notices referring to your personal safety are highlighted in the manual by a safety alert symbol, notices referring only to property damage have no safety alert symbol. These notices shown below are graded according to the degree of danger.

⚠ DANGER

indicates that death or severe personal injury **will** result if proper precautions are not taken.

⚠ WARNING

indicates that death or severe personal injury **may** result if proper precautions are not taken.

⚠ CAUTION

indicates that minor personal injury can result if proper precautions are not taken.

NOTICE

indicates that property damage can result if proper precautions are not taken.

If more than one degree of danger is present, the warning notice representing the highest degree of danger will be used. A notice warning of injury to persons with a safety alert symbol may also include a warning relating to property damage.

Qualified Personnel

The product/system described in this documentation may be operated only by **personnel qualified** for the specific task in accordance with the relevant documentation, in particular its warning notices and safety instructions. Qualified personnel are those who, based on their training and experience, are capable of identifying risks and avoiding potential hazards when working with these products/systems.

Proper use of Siemens products

Note the following:

⚠ WARNING

Siemens products may only be used for the applications described in the catalog and in the relevant technical documentation. If products and components from other manufacturers are used, these must be recommended or approved by Siemens. Proper transport, storage, installation, assembly, commissioning, operation and maintenance are required to ensure that the products operate safely and without any problems. The permissible ambient conditions must be complied with. The information in the relevant documentation must be observed.

Trademarks

All names identified by ® are registered trademarks of Siemens AG. The remaining trademarks in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owner.

Disclaimer of Liability

We have reviewed the contents of this publication to ensure consistency with the hardware and software described. Since variance cannot be precluded entirely, we cannot guarantee full consistency. However, the information in this publication is reviewed regularly and any necessary corrections are included in subsequent editions.

Siemens AG
Digital Industries
Postfach 48 48
90026 NÜRNBERG
GERMANY

A5E42795137-AB
℗ 04/2020 Subject to change

Table of contents

Introduction 1

1.1 Overview

Purpose of this documentation

These compact instructions aim to provide you with a quick introduction to using "PROFINET Driver for controller".

Conventions

The terms "PROFINET Driver for controller" and "PN Driver" are used synonymously in this manual.Please also observe notes marked as follows:

Note

A note contains important information on the product described in the documentation, on the handling of the product, or on the section of the documentation to which particular attention should be paid.

Target group of the documentation

This document is intended for software developers and should help them create an executable user program in a very short amount of time. This requires the following basic knowledge:

- Programming experience with C/C++
- User knowledge in the operating systems Windows or Linux depending on the used variant
- Experience with PROFINET systems
- Basic knowledge of the configuration software TIA Portal or PNConfigLib
- General knowledge of automation technology

What is PN Driver?

PN Driver is a PROFINET controller development kit for RT and IRT.

What are the requirements?

You can run PN Driver on a PC or on an embedded device of your choice. For detailed requirements of the sample variants, you can refer to the corresponding chapters in this document. To port PN Driver to another operating system or to a different hardware platform, you should refer to "How to Port PN Driver V2.2 Manual".

Even though a hardware configuration can also be created without an engineering system, we recommend using the TIA Portal engineering system. You can also use the PNConfigLib tool that allows you to create PROFINET projects, perform consistency checks to ensure their validation and compile them. It is intended to be a lightweight solution to create and compile PROFINET projects. If you use neither TIA Portal nor PNConfigLib, you have to make sure that all the used XML input or output files originate from reliable sources.

Note

The hardware components that are stated as requirements for running the PN Driver software (e.g. CP 1625 board, Intel Ethernet adapters) are not included in the product, customers are responsible for the procurement process.

New functions of PN Driver V2.2

- Linux IP stack support
- Standard Linux Ethernet Driver support
- Calculation tool of Isochronous mode parameters
- Firmware download via SSH
- Hardware configuration with TIA Portal V16
- Enable/Disable SNMP functionality
- Isochronous Real Time (IRT) Performance Upgrade

Note

Only little endian systems are supported.

Note

PN Driver runs only as 32-bit application.

Note

There are two types of startup for PROFINET devices: "Startup Mode Legacy" and "Startup Mode Advanced". Both startup types are supported by PN Driver in RT communication, but PN Driver only supports the "Startup Mode Advanced" startup type in IRT communication.

Application examples

The PN Driver software package includes a set of application examples which offers a quick start into programming your own applications. The application examples also provide an overview of the functionality of PN Driver.

Technical specifications

Table 1- 1 Technical specifications of the PN Driver variants

PN Driver V2.2	Windows	Linux	Linux Native	IOT20x0	CP1625 Host	CP1625 Stand-alone
Max. number of PROFINET devices	16	128	128	64	28	128
Max. number of Fast Startup devices	-	32	-	-	32	32
Supported send cycles RT	32 ms	1, 2, 4 ms	1 ms	1, 2, 4 ms	1, 2, 4 ms	1, 2, 4 ms
Supported send cycles IRT & Isochronous mode	-	-	-	-	0.250, 0.375, 0.5, 0.625, 0.75, 0.875, 1, 2, 4 ms	0.5, 0.625, 0.75, 0.875, 1, 2, 4 ms
Supported number of PROFINET IRT devices	-	-	-	-	32	32
Number of supported frames per milliseconds	4 frames per 32 ms	13 frames per 1 ms	13 frames per 1 ms	8 frames per 1 ms	64 frames per 1 ms	64 frames per 1 ms
Max. data length of a submodule in bytes	1024	1024	1024	1024	1024	1024
Max. number of input addresses in bytes	8192	8192	8192	8192	8192	8192
Max. number of output addresses in bytes	8192	8192	8192	8192	8192	8192
Max. data record length in bytes	32768	32768	32768	32768	32768	32768

Overview of the supplied documentation

Table 1- 2 PN Driver documentation

Documentation	Contents
PROFINET IO-Base User Programming Interface Programming Manual	This document describes the IO-Base API which represents the interface for creating your own user programs.
PROFINET Driver for Controller Engineering Interface Programming and Operating Manual	If an engineering system is not available, hardware configuration can also take place by creating an XML configuration file. This document describes how you can create a hardware configuration based on an XML file.
How to Port PN Driver V2.2 Manual	This document describes how to port the PN Driver software to another operating system or to a different hardware platform.
Readme	This document includes a short introduction about the new features coming with the release. The document also contains registration and licensing information.

Overview of the supplied PNConfigLib documentation

Table 1- 3 PNConfigLib documentation

Documentation	Contents
PNConfigLib User Manual and Documentation	This document provides technical details to configure PROFINET controllers with PNConfigLib and includes application examples.
XSD Documentation	This document gives a list of all configuration parameters defined in input files with their explanation.
PROFINET Controller Attribute List	This document includes PROFINET attributes with their descriptions, types, default values and ranges to help users develop their own PROFINET controller.
Readme	This document includes a short introduction about the new features coming with the release. The document also contains registration and licensing information.

Additional support

If you have questions regarding the PN Driver that are not addressed in the documentation, please contact your local representative at the Siemens office nearest to you.

Please send questions, comments and suggestions regarding this manual in writing to the specified e-mail addresses below.

For useful product information about PN Driver and PNConfigLib, please visit the following address (https://support.industry.siemens.com/cs/products/6es7195-3aa00-0ya0). In addition, you can find general information on the Internet (https://www.siemens.com/profinet-development).

Technical contact information worldwide

Siemens Sanayi ve Ticaret A.Ş
Office Address:
Yakacık Caddesi No 111
34870 Istanbul, Turkey

E-mail: (mailto:profinet.devkits.industry@siemens.com)

Technical contact information for the U.S.

PROFI Interface Center
(http://www.profiinterfacecenter.com)
Office Address:
Siemens Industry, Inc.
C/O The PROFI Interface Center
One Internet Plaza Johnson City,
TN 37604

Phone: +1 (423) 262-2576
E-mail: (mailto:PIC.industry@siemens.com)

Technical contact information for China

The PROFI Interface Center China
Office Address:
7, Wangjing Zhonghuan Nanlu
100102 Beijing

Phone: +86- (10-) 6476-4725
E-mail: (mailto:Profinet.cn@siemens.com)

1.2 Security information

Siemens provides products and solutions with industrial security functions that support the secure operation of plants, systems, machines and networks.

In order to protect plants, systems, machines and networks against cyber threats, it is necessary to implement – and continuously maintain – a holistic, state-of-the-art industrial security concept. Siemens' products and solutions constitute one element of such a concept.

Customers are responsible for preventing unauthorized access to their plants, systems, machines and networks. Such systems, machines and components should only be connected to an enterprise network or the internet if and to the extent such a connection is necessary and only when appropriate security measures (e.g. firewalls and/or network segmentation) are in place.

For additional information on industrial security measures that may be implemented, please visit (https://www.siemens.com/industrialsecurity).

Siemens' products and solutions undergo continuous development to make them more secure. Siemens strongly recommends that product updates are applied as soon as they are available and that the latest product versions are used. Use of product versions that are no longer supported, and failure to apply the latest updates may increase customers' exposure to cyber threats.

To stay informed about product updates, follow us on Twitter (@ProductCERT), register to our advisory mailing list or subscribe to the Siemens Industrial Security RSS Feed under (https://new.siemens.com/global/en/products/services/cert.html#Subscriptions).

1.3 PROFINET security guidelines

As PROFINET controller, PN Driver must abide by the security guidelines of the PROFINET International Organization.

The security and installation guidelines of PROFINET International Organization can be found on the Internet under (https://www.profibus.com).

In particular we refer to:

PROFINET Planung / Design
Version: 1.38
Order No.: 8.061 / 8.062 (German / English)

PROFINET Montage / Cabling and Assembly
Version: 2.8
Order No.: 8.071 / 8.072 (German / English)

PROFINET Inbetriebnahme / Commissioning
Version: 1.44
Order No.: 8.081 / 8.082 (German / English)

PROFINET Security Guideline
Version: 2.0
Order No.: 7.002 / 7.001 (English / German)

PROFINET IO Security Level 1
Version: 1.2.1.1
Order No.: 7.302 (English)

1.4 Open Source Software

The product/system described in this document may use Open Source Software or any similar software of a third party (hereinafter referred to as "OSS"). The OSS is listed in the Readme_OSS-file of the product.

The purchaser of the product/system described in this document (hereinafter referred to as "the Customer") is responsible for the right to use OSS that is required for the product to operate safely and without any problems in accordance with the respective license conditions of the OSS.

1.5 Disclaimer for third-party software updates

This product includes third-party software. Siemens AG only provides a warranty for updates/patches of the third-party software, if these have been officially released by Siemens AG. Otherwise, updates/patches are undertaken at your own risk.

1.6 Notes on protecting administrator accounts

A user with administrator privileges has extensive access and manipulation options in the system. Therefore, ensure there are adequate safeguards for protecting the administrator accounts to prevent unauthorized changes. To do this, use secure passwords and a standard user account for normal operation. Other measures, such as the use of security policies, should be applied as needed.

Quick start for Windows

2

2.1 Quick start for Windows

Overview

This section describes the most important steps required for commissioning PN Driver under Windows 10. PN Driver V2.2 is tested only with Windows 10 64-bit OS.

Customers are responsible for the target system build (PN Driver under Windows) with the help of the source code of PN Driver V2.2, application examples and the relevant documentation. PN Driver must be run as a 32-bit application under Windows.

PN Driver Windows variant is only for demonstration purposes. It is not meant to run as a proper IO controller because Windows OS does not support RT.

Requirements

You need a WinPcap installed PC with a network adapter. You can use any network adapter that is supported by WinPcap device driver. To create your own user programs based on the IO-Base API, we recommend using a development environment, such as Microsoft Visual Studio 2017.

Copy the CD contents to a path on your Windows PC. Then execute the steps described in the following sections.

2.2 Installing WinPcap Developer's Pack

Procedure

1. Download the WinPcap Developer’s Pack from the website (https://www.winpcap.org/devel.htm).
2. Create a folder called "driver" under the path "[..]\pn_driver\src\".
3. Extract the downloaded zip file and copy the folder "WpdPack" to the folder "[..]\pn_driver\src\driver\".

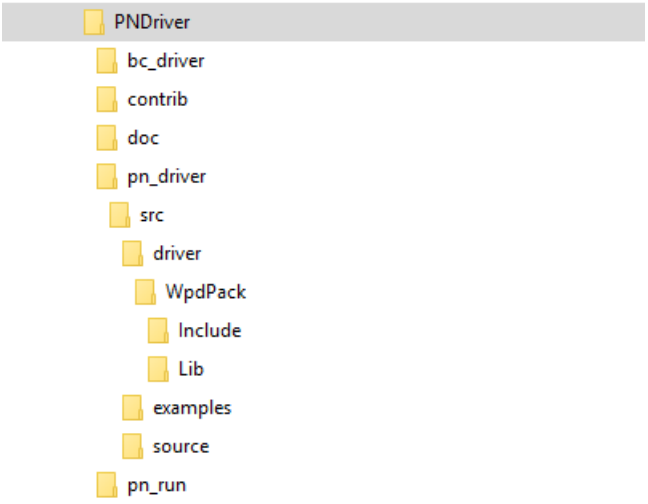


Figure 2-1 PN Driver file structure

The test application uses the files "wpcap.dll" and "packet.dll" to access the local network adapters.

PN Driver V2.2 is tested with version 4.1.2 of the developer’s pack.

2.3 Installing WinPcap

Procedure

1. Download the WinPcap driver for Windows from the WinPcap website (https://www.winpcap.org/).
2. Perform the installation.

PN Driver V2.1 is tested with version 4.1.3 of the WinPcap driver for Windows.

Note

Notice that WinPcap Developer's pack and WinPcap versions with which PN Driver V2.2 is tested are different from each other. This is because "there is no Developer's package specific for WinPcap 4.1.3 and the current 4.1.2 package is compatible with WinPcap 4.1.3" as stated in WinPcap website.

Note

Although WinPcap website states that the product is not officially supported any more, PN Driver V2.2 is tested with WinPcap and works properly.

2.4 Installing Visual Studio 2017

The Microsoft Visual Studio 2017 development environment lets you run the test application provided on the installation package under Windows or you can develop your own application by using PN Driver.

Procedure

1. Download a Microsoft Visual Studio 2017 product of your choice (version 15.9.1 or higher) for Windows from the Microsoft website (https://www.visualstudio.com/vs/older-downloads/).
2. Perform the installation. During installation, do not forget to include the "Desktop development with C++" workload.

2.5 Disabling network protocols that are not utilized

Since PN Driver includes an IP stack, it is important that all network protocols (TCP/IP, File and Printer Sharing, Client for Microsoft Networks, etc.) are disabled on the network adapter used by PN Driver.

Procedure

1. Make sure that all network protocols are disabled for the network adapter on which PN Driver will run.

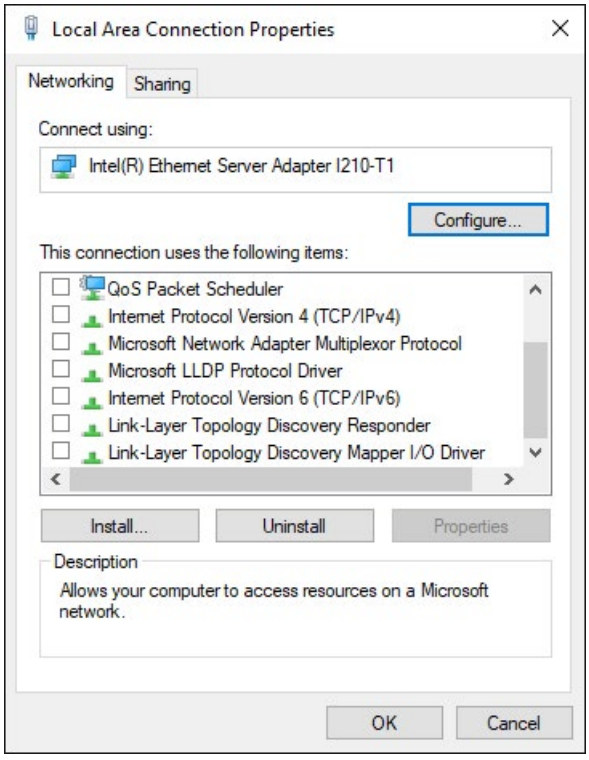


Figure 2-2 Disabling network protocols

2.6 Disabling the PNIO adapter of the PG/PC interface

If a copy of a configuration software such as TIA Portal is installed on your computer, the PNIO adapters of the PG/PC interface must be disabled.

Procedure

1. Ensure that all PNIO adapters of the PG/PC interface are disabled.

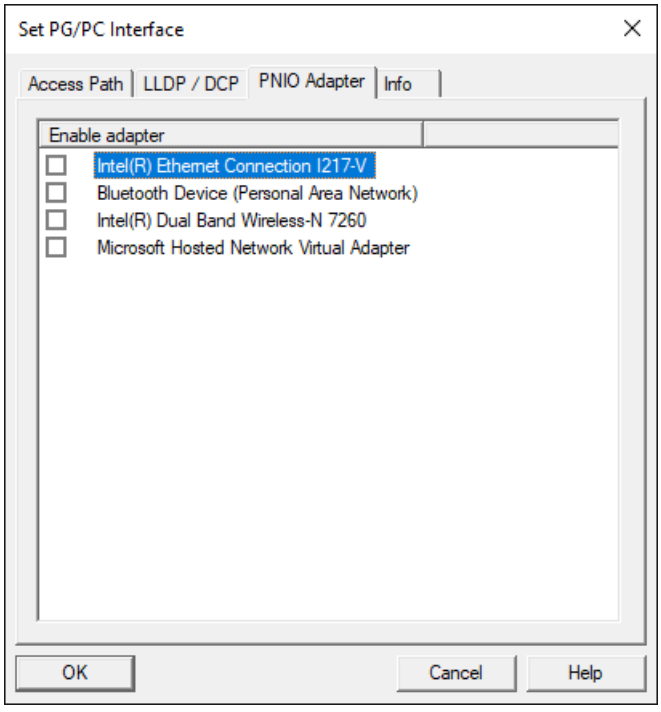


Figure 2-3 Disabling PNIO adapters of the PG/PC interface

2.7 Start application example

The enclosed CD includes application examples. The execution and debugging of the "test_app" is described below as an example.

Requirement

The following paths are important for executing the application example:

- Microsoft Visual Studio 2017 project folder:
  "[..]\pn_driver\src\examples\test_app\win32\test_app.sln"
- Source files: "[..]\pn_driver\src\examples\test_app\src\"
- Example XML configuration file: "[..]\pn_driver\src\examples\test_app\win32\
  PNDriverBase_TestApp\Station_1.PN Driver_1.PNDriverConfiguration.XML"

Procedure

1. Start Visual Studio 2017 and open the "Solution" under the path "[..]\pn_driver\src\examples\test_app\win32\test_app.sln". Two projects are loaded in this step: "pndriver_w32" and "test_app".

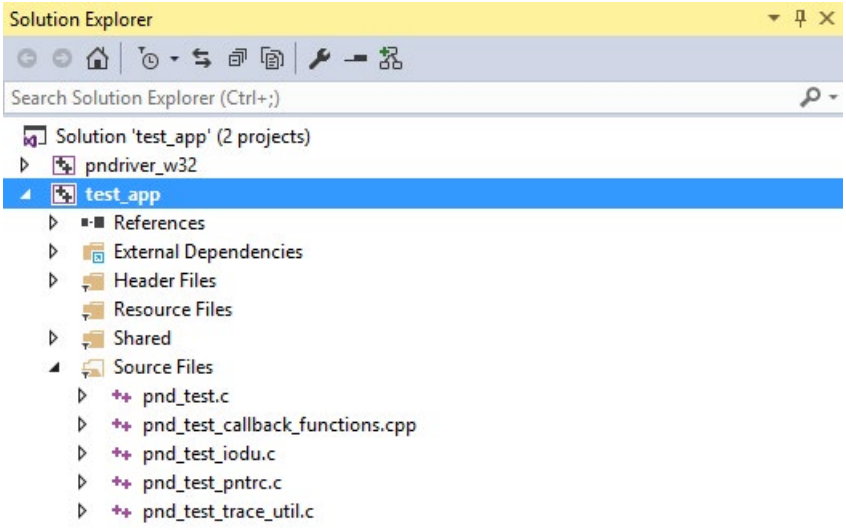


Figure 2-4 Solution Explorer

Before PN Driver can be started or debugging can take place, you must ensure that the application can find the related XML configuration file. A prepared XML configuration file is available in the directory of the application examples. If you want to control a different HW configuration with this application, you must copy the configuration file to the working directory. See the section Application examples (Page 102).

2. Under the "test_app" project, open the file "pnd_test.c".

3. Set a breakpoint on the "int main()" function and start the program.
   When you start the program for the first time, the projects are initially compiled.

   If the program is executed further, the functions that can be executed by the test application are listed.

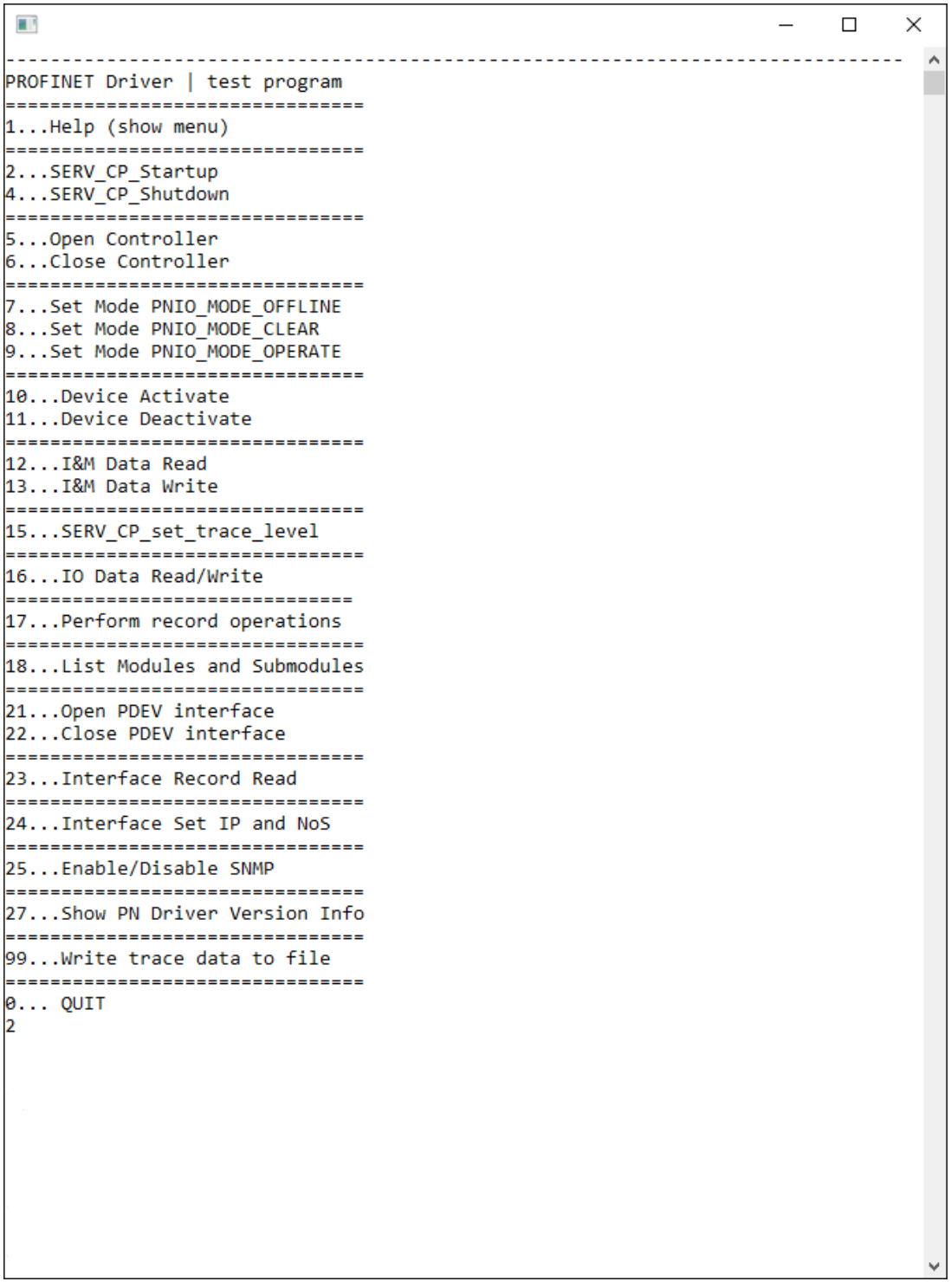


Figure 2-5 Menu of the test application

4. Enter "2" to call the start function. If the WinPcap driver has been installed correctly, a list with the network adapters is shown. Select the network adapter that you want to use for the PROFINET communication. When the HW configuration is downloaded without errors and PN Driver is started, "0" is output as return value.

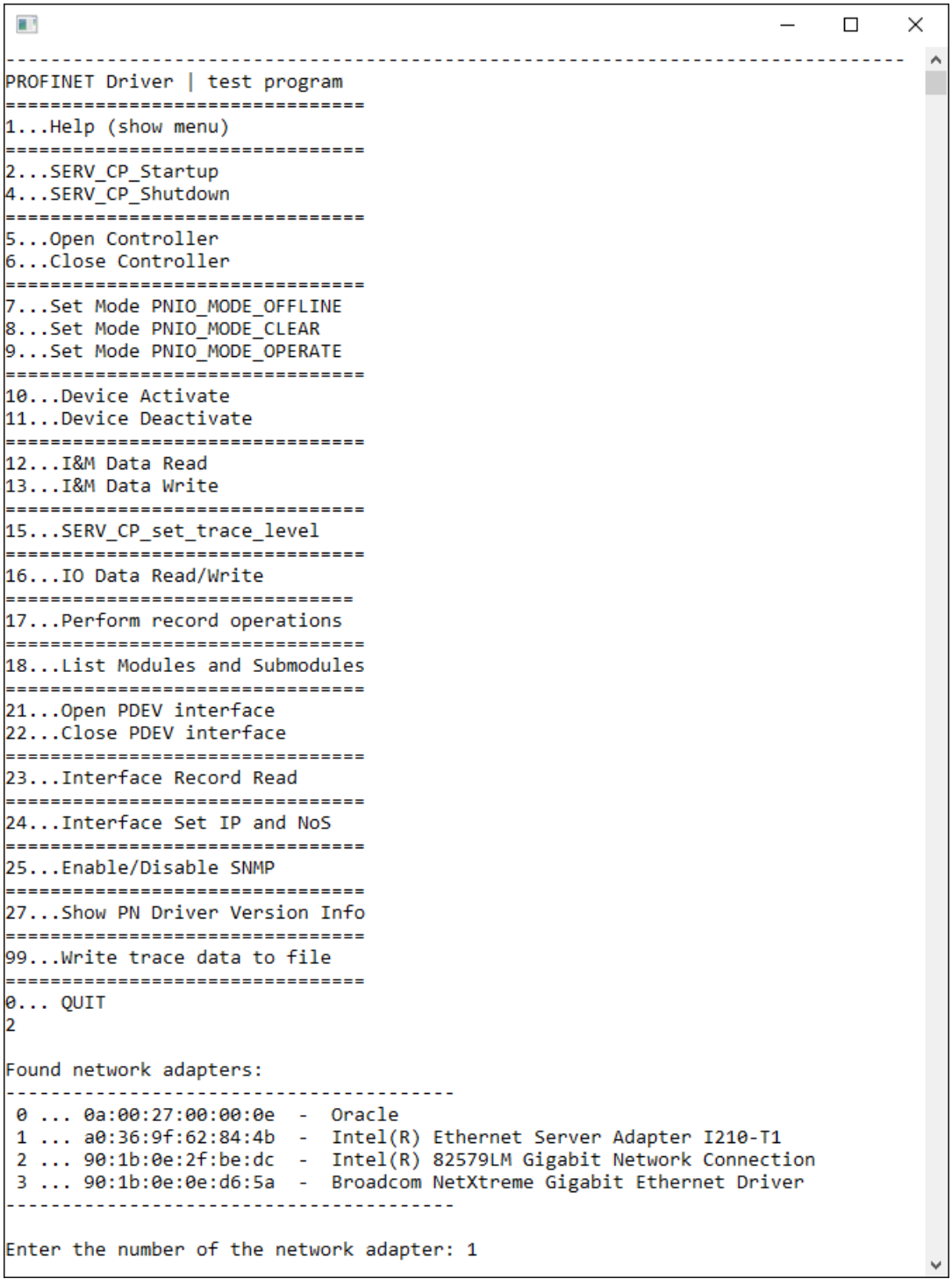


Figure 2-6 List of network adapters

Note

For an application developed for PN Driver V1.0 to run under PN Driver V2.1 or PN Driver V2.2, you must make the following changes in the source code:

1. When starting PN Driver, insert the new function "SERV_CP_init" before calling the function "SERV_CP_startup".
2. Update the function "SERV_CP_startup" according to the specification in the manual "PROFINET IO-Base User Programming Interface".
3. When shutting down PN Driver, call the function "SERV_CP_undo_init" after calling the function "SERV_CP_shutdown".

Details on the respective functions are available in the manual "PROFINET IO-Base User Programming Interface".

Note

PN Driver V1.1 is compatible with PN Driver V2.1 and PN Driver V2.2.

Quick start for Linux 3

3.1 Quick start for Linux

Overview

This section describes the steps required for commissioning PN Driver under Linux.

The following statements refer to Debian 9.3 with the Linux kernel V4.9 including real-time patch. To keep updated for any Debian version upgrade, please refer to the PN Driver Readme document .

Customers are responsible for the target system build (PN Driver under Linux) with the help of the source code of PN Driver V2.2, application examples and the relevant documentation files.

3.2 PN Driver Linux variants

There are two PN Driver variants that can run on a Linux PC with a standard ethernet adapter: Linux Native and Linux. These variants have the following different features and constraints:

Linux Native variant: It is possible to use the same network interface for both IP communication of third party applications and PROFINET communication with this PN Driver variant.

Please note that the configurations which are described in the section Using PROFINET interface for IP communication (Page 90) are mandatory for this variant even if the interface is only used for PROFINET.

PN Driver Linux Native variant can be used with any ethernet adapter as long as it is installed to the system with its driver correctly.

The interface that is used by Linux Native variant must not be used for any other purposes. The real-time performance could be influenced by not following this rule. For TCP and UDP communication the created virtual interface should be used.

PROFINET feature Fast Startup is not supported by this variant.

Linux variant: You must have PNDevDriver installed on your system. This is the device driver used by PN Driver and it is provided in the PN Driver delivery. The supported ethernet adapters are Intel I210 ("Springville") and Intel 82574L ("Hartwell"). The PROFINET interface can be used only by PN Driver, so you should have an additional ethernet adapter for any other type of network communication.

PROFINET feature Fast Startup is supported by this variant.

Note

We recommend using the Eclipse development environment under Linux.

PN Driver only supports little endian environments. PN Driver only works as a 32-bit application under 32-bit or 64-bit operating systems.

3.3 Installing Debian

Debian images are available on the Debian website (https://www.debian.org/) and can be installed with a USB stick. For installation, you can copy the *.iso image to a USB stick (for example, with UNetbootin from (https://unetbootin.github.io)).

The USB stick must be formatted in FAT32 and should be blank.

Procedure

1. Install Debian.
   - You can use the default settings during installation.
   - User name and password as well as the Superuser(Root) password must be chosen to comply with your security regulations.

   After the installation is completed and the PC is restarted, the login window is displayed.
2. Enter the previously specified user name and the specified password.

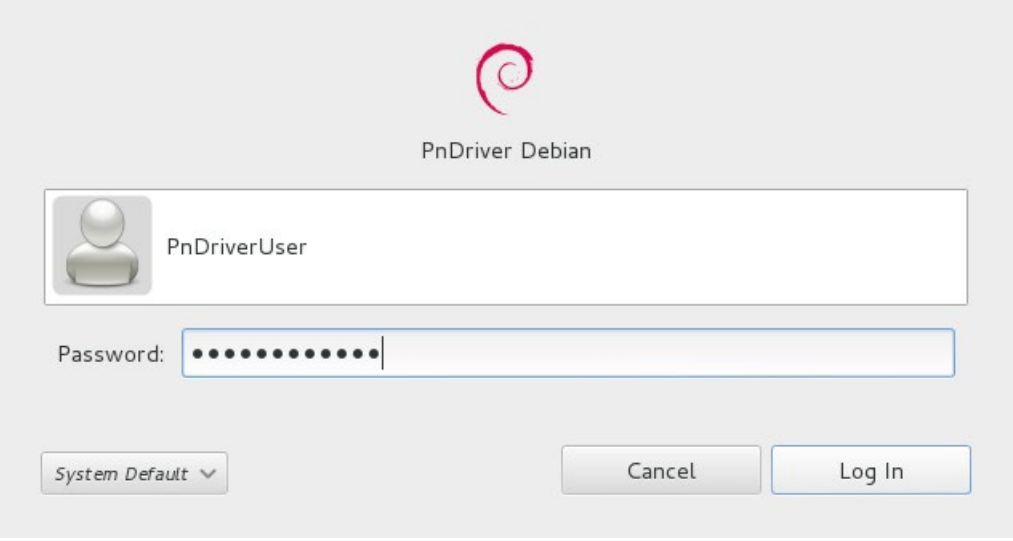


Figure 3-1 Debian login

Note

Standard Linux and Linux toolchain are not parts of the product PN Driver V2.2.

3.4 Enabling admin rights for the user

You need superuser rights to run example application. To get these rights, you may need to proceed as follows. This step highly depends on the options chosen during installation.

Procedure

1. Start a terminal and enter "su".
2. Confirm the password prompt with the superuser password specified during installation.
3. If you have not already installed ‘sudo’ package, you need to run the following command:
   `apt-get install sudo`
4. Add your user to the superuser group ("sudo") by entering "`adduser PnDriverUser sudo`". ("PnDriverUser" is the user name used in the example).

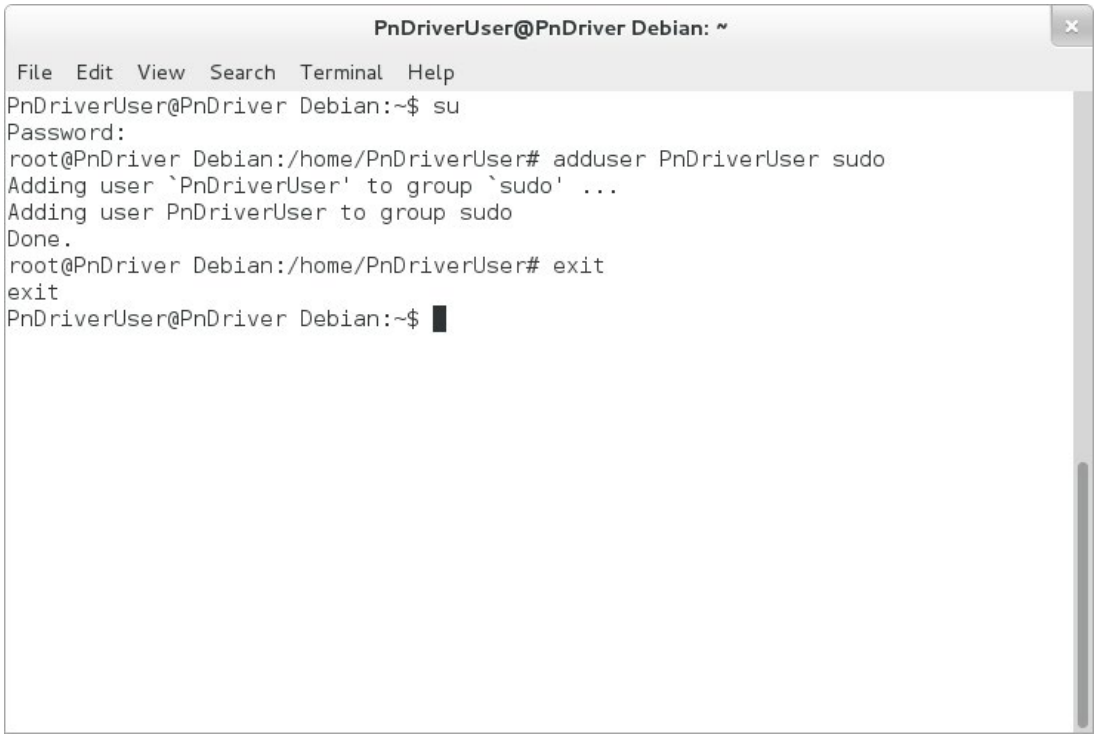


Figure 3-2 Enabling admin rights

5. Log off with the current user and log on again for the changes to become effective.

3.5 Installing the real-time Linux kernel

Procedure

1. To ensure the real-time capability of the Linux system, download and install the real-time kernel from the repository:
   - for 32-bit systems:
     ```
     sudo apt install 4.9.0-11-rt-686
     ```
   - for 64-bit systems:
     ```
     sudo apt install 4.9.0-11-rt-amd64
     ```
2. When the CPU is not active, parts of it are switched to idle mode. This saves power for the PC but it takes longer for the CPU until it can compute actively. This wake up mode slows down cyclic tasks with short cycle times (ms and µs). You therefore need to change the scheduling mechanism of the real-time Linux kernel with one of the following methods:
   - Add the boot parameter "idle=poll" below the kernel you use in the file "/boot/grub/grub.cfg". Example:
     ```
     linux  /boot/vmlinuz-4.9.0-11-rt-686-pae root=UUID=xxx ro quiet idle=poll
     ```
   - If changes in kernel boot parameters are not allowed or if this setting needs to be enabled/disabled on demand, the user application may use the Power Management Quality of Service (PM QOS) interface, which is mapped to /dev/cpu_dma_latency, to achieve the same effect. Writing zero to /dev/cpu_dma_latency will emulate the idle=poll behavior.

If no desirable Linux kernel with real-time support ("rt") is available, you must download the source code files of the required Linux kernel, edit them with a Realtime preemption (RT-Preempt) patch and compile the kernel yourself. You can download the kernel and the available Realtime patches at kernel.org kernel.org (https://www.mirrorservice.org/sites/ftp.kernel.org/pub/linux/kernel/)

NOTICE
While power management and frequency scaling are disabled with idle=poll parameter or /dev/cpu_dma_latency interface, turbo mode should be disabled from the BIOS setting to prevent any negative effects on the CPU for long-term usage.

3.6 Installing other required packages

Requirements

You may need the following to execute the instructions listed below:

Note

Please make sure that your machine has internet access and the packet servers are reachable. These steps highly depend on the chosen options of the operating system installation.

Proxy server

If a proxy server is required, edit the file "/etc/apt/apt.conf.d/80proxy" by entering the respective proxy settings as follows:

```
Acquire::http::Proxy "..."; //("..." = http://user:password@proxy.server:port/)

Acquire::https::Proxy "..."; //("..." = http://user:password@proxy.server:port/)
```

Sources for updates

You need to store the sources for updates in order to be able to download the required packages. To store the sources for updates, please follow the steps below:

1. Open the file “/etc/apt/sources.list”.
2. There comment out "`cdrom`".
3. Uncomment the following and extend with “non-free”:
   ```
   deb http://deb.debian.org/debian stretch-updates main contrib non-free
   deb-src http://deb.debian.org/debian stretch-updates main contrib non-free
   ```
4. Add:
   ```
   deb http://deb.debian.org/debian stretch main contrib non-free
   deb-src http://deb.debian.org/debian stretch main contrib non-free
   ```
5. Save the list. Then enter the following in the terminal program:
   ```
   sudo find /var/lib/apt -type f -exec rm {} \+
   sudo apt update
   ```

Procedure

1. Install the required Linux header files. For example, enter the following in the terminal program for 64 bit systems:
   ```
   sudo apt install linux-headers-4.9.0-11-rt-amd64
   ```
   Make sure that the headers match the currently used kernel. You can read out the kernel version with the command "`uname -a`".
2. To build the PN Driver application, you must install "gcc", "make" and additional packages. To do so, enter the following in the terminal program:
   ```
   sudo apt install build-essential
   ```
3. Install the following packages to get the debugging option:
   ```
   sudo apt install gdb gdbserver gdb-multiarch
   ```
4. PN Driver is a 32-bit application. If you have a 64-bit Debian installation but want to run 32-bit applications, you must add the i386 architecture to the package manager and update the package database. To do so, enter the following in the terminal program:
   ```
   sudo dpkg --add-architecture i386
   sudo apt update
   sudo apt install libc6-dev-i386
   sudo apt install g++-multilib
   ```
   Next you must install the matching C++ standard library:
   ```
   sudo apt install libstdc++6:i386
   ```

3.7 Using the PN Device Driver for the Linux variant

Overview

You can skip this section if the Linux Native variant will be used. Linux variant of PN Driver requires PN Device Driver (PNDevDrv) to use the network adapters. You must compile the modules yourself as described below.

Note

When we talk about "32-bit" or "64-bit" in this section, we are always referring to the installed operating system and not the application. This means you use a 64-bit Linux kernel module for a 64-bit operating system even when a 32-bit application is executed.

After a restart, the PNDevDrv must be loaded again unless you have set your system to do it automatically as described in the section Loading the PN Device Driver automatically (Page 30).

3.7.1 Compiling the PN Device Driver

You must compile the Linux kernel module. To do so, you need the directory "bc_driver/".

Procedure

1. When you have installed all the build tools mentioned above (see section Installing other required packages), you can generate the PN Device Driver by entering the following command in the directory "[..]/bc_driver/src/pndevdrv/bin/Linux":
   - in 32-bit systems:
     ```
     sudo make modules IS_32=1
     ```
   - in 64-bit systems:
     ```
     sudo make modules
     ```

   The result is a "*.ko" file which can then be loaded. Keep in mind that a "_32.ko" file can only be compiled and loaded on a 32-bit Debian; a "_64.ko" file can only be used with 64-bit systems.
2. In addition to the actual driver, a shared object file must be generated; this file corresponds to a DLL in Windows. You can generate this shared object file with the "Makefile_SharedObject" in "[..]/bc_driver/src/pndevdrv/bin/Linux":
   - in 32-bit systems:
     ```
     make -f Makefile_SharedObject PnDev_DriverU32.so CCFLAGS=-DMAP_32BIT=0
     ```
   - in 64-bit systems:
     ```
     make -f Makefile_SharedObject all
     ```

   As a result, a shared object file ("*.so") for 32-bit or 64-bit applications is generated and stored in the same folder.

3.7.2 Loading the PN Device Driver

Procedure

1. To load the PN Device Driver temporarily (until the next restart), use the following command in the same folder in which the "PnDevDrv_*.ko" module is located:
   - for 32-bit systems:
     ```
     sudo insmod PnDevDrv_32.ko
     ```
   - for 64-bit systems:
     ```
     sudo insmod PnDevDrv_64.ko
     ```

3.7.3 Binding the PCI card

Overview

To get the PNDevDrv running for a specific PCI Card, you need to bind the card to the driver. To execute the following procedure, admin rights are necessary. For detailed information, please refer to the section Enabling admin rights for the user (Page 26)

Procedure

1. Find out the PCI location of the corresponding card. Use the command "lspci –k".
2. Identify if there is already a Linux kernel driver in use. If there is, unbind the PCI device. The following command is an example of doing this.

   ```
   echo 0000:01:00.0 > /sys/bus/pci/devices/0000:01:00.0/driver/unbind
   ```

3. A supported unbound card can be bound to PNDevDrv with:

   ```
   echo 0000:01:00.0 > /sys/bus/pci/drivers/pndevdrv/bind
   ```

4. When executing the command "lspci –k" again, the used Linux kernel driver for the corresponding PCI card should be

   ```
   "Kernel driver in use: pndevdrv".
   ```

3.7.4 Unbinding the PCI card

Overview

The PNDevDrv can be unbound from a specific PCI card.

Note

Unbinding and binding the PNDevDrv can be used for resetting the card if there has been a crash.

Procedure

1. Find out the PCI location of the corresponding card. Use the command "`lspci -k`".
2. A bound card can be unbound by executing one of the following commands with the corresponding PCI location.

   ```
   "echo 0000:01:00.0 > /sys/bus/pci/drivers/pndevdrv/unbind"
   ```

   or

   ```
   "echo 0000:01:00.0 > /sys/bus/pci/devices/0000:01:00.0/driver/unbind"
   ```

3. When executing the command "lspci –k" again, the line "`Kernel driver in use:`" should not be present.

3.7.5 Unloading the PN Device Driver

If the application you have developed crashes, we recommend that you reload the PN Device Driver.

Procedure

1. Enter the following in the terminal program:
   - for 32-bit systems:
     ```
     sudo rmmod PnDevDrv_32
     ```
   - for 64-bit systems:
     ```
     sudo rmmod PnDevDrv_64
     ```
2. Then you can load the PN Device Driver once again (see the section Loading the PN Device Driver (Page 30)).

The error message "Error: Module PnDevDrv_xx is in use" indicates that the driver is being used by another application in the system. This application must be closed first.

3.7.6 Loading the PN Device Driver automatically

Linux supports automatic loading of drivers during booting. For this, you must do the following:

Procedure

1. Copy the Linux kernel module "*.ko" to a folder that is available during system start, e.g. "/usr/local/". (In case of encrypted home drives, "/home/<Username>/[..]" might not be readable.)
2. For Linux to find the driver during booting, a symbolic link to the Linux kernel module must be created:
   ```
   sudo ln -s /usr/local/PnDevDrv_64.ko /lib/modules/4.9.0-11-rt-amd64
   ```
   The input "`4.9.0-11-rt-amd64`" in the example depends on the installed real-time Linux kernel. Make sure that the path information is correct.
3. For the automated binding of the PNDevDrv to the corresponding card, a file called pndev.rules needs to be created at /etc/udev/rules.d/.
   This file unbinds the card from an already existing driver and binds it to the PNDevDrv. To do so, add the following lines to pndev.rules file:
   ```
   ACTION=="add", KERNEL=="0000:01:00.0", SUBSYSTEM=="pci", RUN+="/bin/sh -c 'echo 0000:01:00.0 > /sys/bus/pci/devices/0000:01:00.0/driver/unbind'"
   ACTION=="add", KERNEL=="0000:01:00.0", SUBSYSTEM=="pci", RUN+="/bin/sh -c 'modprobe PnDevDrv_64'"
   ACTION=="add", KERNEL=="0000:01:00.0", SUBSYSTEM=="pci", RUN+="/bin/sh -c 'echo 0000:01:00.0 > /sys/bus/pci/drivers/pndevdrv/bind'"
   ```
   Please note that you must adapt the PCI bus addresses according to your system.
4. Internal start routines of Linux can be updated with the following two commands, so that the driver can be loaded during the next restart:
   ```
   sudo depmod –a
   sudo update-initramfs -u
   ```

3.7.7 Determining errors during loading of the PN Device Driver

If loading of the PNDevDrv ("insmod") were to fail, you can output the last system messages and try to determine what caused the error.

Procedure

1. To do so, enter:
   `dmesg`

Possible causes of error are:

- Insufficient memory for mapping (for 32-bit systems)
  - Error message during loading: "Permission denied!" - `dmesg` indicates that "vmap" could not be executed
  - Solution: Start 32-bit systems with the boot parameter "vmalloc=xxxM".
- Other drivers loaded for the network adapters (igb, e1000e)
  - Solution: Unbind/bind the driver from/to the PCI card.
- Driver is already loaded
  - Solution: Unload/load the driver.
- Insufficient coherent memory available (for DMA transfer).
  - Solution: Load driver during booting when memory is not yet very fragmented.
- Path to "*.ko" file incorrect (with loading during boot process)
  - Solution: Update the path.

3.8 Additional Configuration for the Linux Native variant

If the Linux Native variant is used, the following settings that are specific to this variant must be done.

3.8.1 Disabling DHCP

DCHP clients must be deactivated for the interface which is used for PROFINET communication.

Adding the following line to the file "/etc/network/interfaces" would deactivate DHCP clients for the interface (assuming the interface name is eth0):

- `iface eth0 inet manual`

3.8.2 Removing IP address

The interface which is used for PROFINET communication must not have any IP address since it is used to capture raw packets by PN Driver.

This setting is not necessary if the setting which is described in Disabling DHCP (Page 33) has already been done and the system has been rebooted or the networking system has been restarted afterwards. Otherwise, the following command would remove the IP address (assuming the interface name is eth0):

- `ip addr flush dev eth0`

3.8.3 Disabling ARP Protocol

The ARP protocol must be disabled for the interface used by the Linux Native variant. Otherwise, two ARP responses will be sent on the wire for ARP probes, one from PN Driver and the other one from Linux IP stack.

Adding the following line to a created "/etc/sysctl.d/pn_arp.conf" file would disable ARP for the interface used by the Linux Native variant (assuming the interface name is eth0):

- `net.ipv4.conf.eth0.arp_ignore=8`

3.8.4 Preventing duplicated packets

In the Linux Native variant, incoming packets are received by both PN Driver and Linux IP stack. As PN Driver also sends the non-PROFINET packets to the Linux IP stack, Linux IP stack gets the same non-PROFINET packet twice. To prevent duplicate packets the following commands need to be executed, before PN Driver is started (assuming the interface name is eth0):

- `sudo tc qdisc add dev eth0 handle ffff: ingress`
- `sudo tc filter add dev eth0 parent ffff: matchall skip_hw action drop`

These commands are not persistent and they need to be run after every PC restart or binding/unbinding the driver to/from the PCI Card.

3.8.5 Adapting LED Blink functionality to your hardware

Linux Native variant can be used with any ethernet adapter, but the LED handling solution provided by PN Driver may not be suitable for every adapter. In this case, the user needs to implement a hardware specific LED handling for "DCP Show Location" service of PROFINET. For more details please refer to the related section of "How to Port PN Driver V2.2 Manual".

3.9 Installing and starting Eclipse IDE

The Eclipse development environment may be used for starting and debugging PN Driver under Linux.

Procedure

1. Under Debian, you can install Eclipse with the package manager "apt". To do so, enter:
   ```
   sudo apt install eclipse-cdt
   ```
2. Start Eclipse for debugging with superuser (root) rights:
   ```
   sudo eclipse
   ```
3. After the installation, you can load the test application in Eclipse using [**File > Import > General > Existing Projects into Workspace**].
4. In the **Import** dialog as root directory, navigate to the CD directory "[..]/pn_driver/src/examples". Eclipse now displays all existing projects.
5. Select "pndriver_32" and "test_app" projects under the following paths respectively depending on the variant which you want to build

   For Linux variant;

   [..]/pn_driver/src/examples/shared/linux32

   [..]/pn_driver/src/examples/test_app/linux32

   For Linux Native variant;

   [..]/pn_driver/src/examples/shared/linux32_native

   [..]/pn_driver/src/examples/test_app/linux32_native

   Once these projects are imported into Eclipse, you can use the test application for debugging. All settings required for the build process are already implemented.

6. Compile the two projects using **[Project > Build All]**. The compilation process uses a makefile that is available in the subdirectory "build" under the project directories which are selected in the above step.

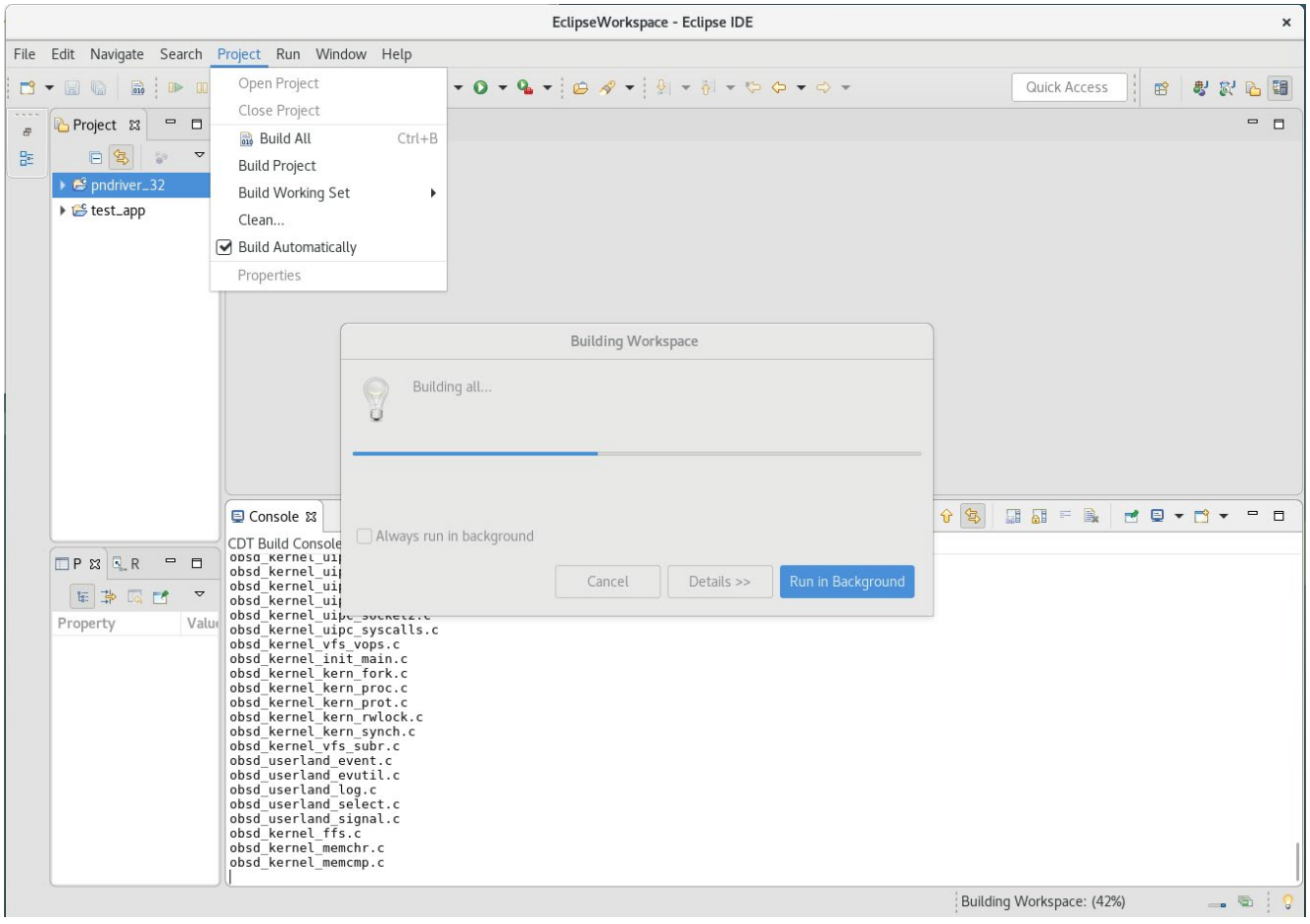


Figure 3-3 Eclipse - Build All

During debugging you must ensure that the starting conditions regarding the XML file and the shared object file are observed. Both files must be located in the same directory as the executed PN Driver. See the section Starting the application example (Page 37).

3.10 Starting the application example

Requirements

Before you start PN Driver or run debugging, you must ensure that the application can locate the corresponding XML file as well as the shared object file of the PN Device Driver if the Linux variant is used. Both files must be stored in the same directory as the executable PN Driver file. The directory for the test application is the same as the project directory which is selected in chapter Additional Configuration for the Linux Native variant (Page 33)

- This directory already includes a prepared test XML file for hardware configuration.
- If Linux variant is used, for communication between PN Driver and the PNDevDrv, the associated shared object file "PnDev_DriverU32.so" that is available in the directory "[..]/bc_driver/src/pndevdrv/bin/Linux" must be placed in the same directory as the executable PN Driver file.
- Because PN Driver changes the thread priority in the real-time Linux kernel, the application must be started with superuser permissions. For debugging, this is also true for the Eclipse development environment from which the application is started.

If the Linux Native variant is used, the configuration steps which are described in Additional Configuration for the Linux Native variant (Page 33) must be done before running the application.

Note

When you start the PN Driver application in Eclipse in debugging mode, the application may no longer respond. Exit Eclipse and restart it.

To avoid errors, start the PN Driver application in Eclipse in Run mode.

Procedure

1. You can now create a breakpoint with Eclipse in the "main()" function ("[..]/pn_driver/src/examples/test_app/src/pnd_test.c") which stops the program at the beginning during debugging.
2. For debugging, right-click the "test_app" project and select using **[Debug As > 1 Local C/C++ Application]**.

   Note

   To detect crashes in the application, a time-out of 2 s is executed when you start PN Driver. You can disable this timeout, which disrupts debugging, in the file "[..]/pn_driver/src/source/pnd/src/agent/pnd_agent.cpp", function "pnd_agent_startup", when calling "eps_open(xxx, xxx, 2);". To do so, replace the last parameter (timeout period in s, here: "2") with "0" (infinite waiting).

3. After starting PN Driver (by running the command "`sudo ./test_app`" or with Eclipse), it will show an overview of all detected and supported Ethernet adapters. By entering "2" (for "SERV_CP_startup"), you can select a card that is to be used by PN Driver.

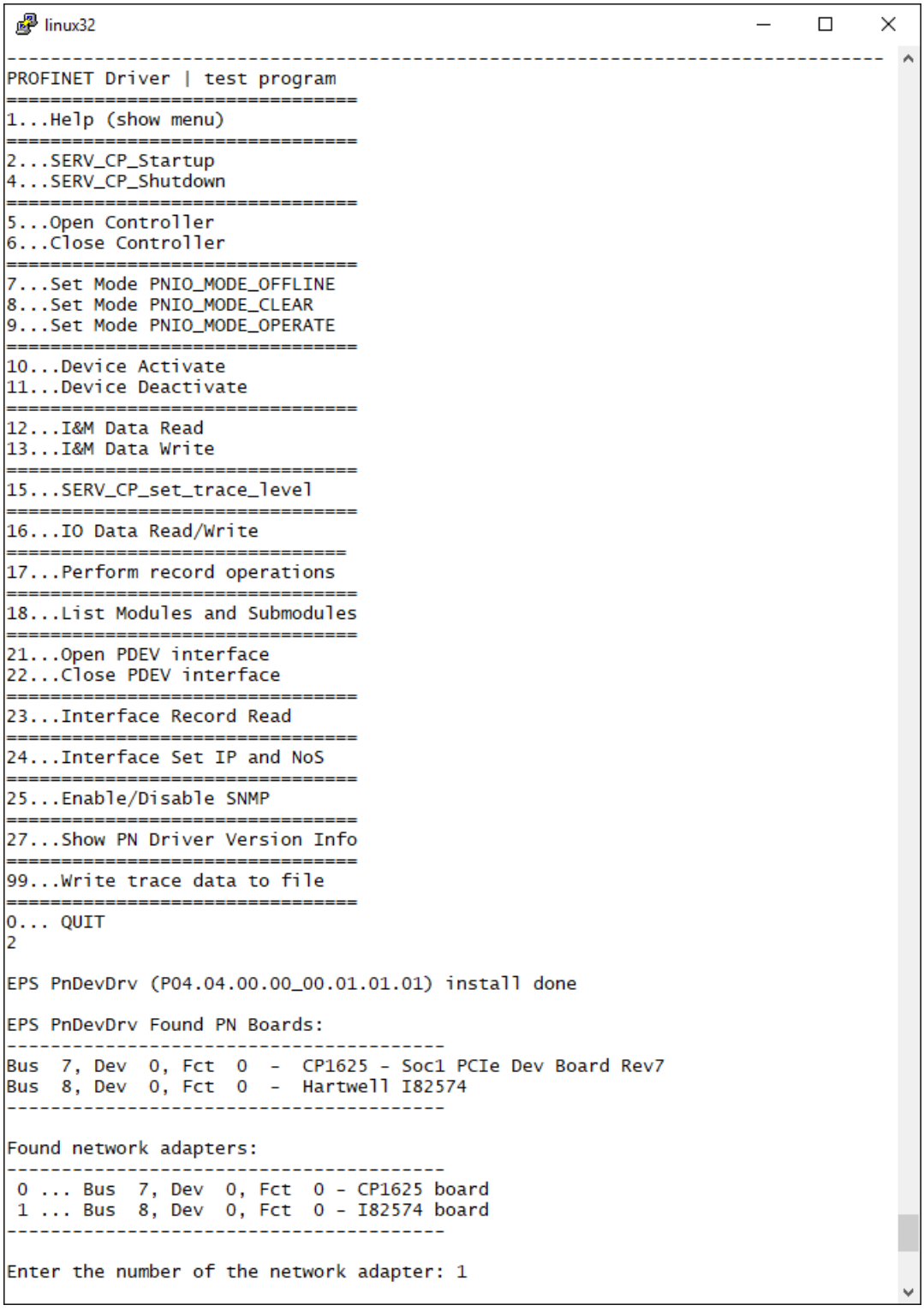


Figure 3-4 Selecting a network adapter

Quick start for IOT20x0

4

4.1 Quick start for IOT20x0

Overview

This section describes the steps required for commissioning PN Driver under a custom Linux operation system built with Yocto Project (https://www.yoctoproject.org).

The following statements refer to standard Yocto image for IOT20x0 where x stands for 0, 2 or 4. This image can be built by using IOT20x0 Board Support Package IOT20x0 Board Support Package (https://github.com/siemens/meta-iot2000/releases/tag/V2.2.0) with Yocto 2.1.

Customers are responsible for the target system build (PN Driver under IOT20x0 hardware device) with the help of the source code of PN Driver V2.2, application examples and the relevant documentation files.

Requirements

You need an IOT20x0 hardware device with a valid boot image and PNDevDrv installed. We recommend using the Eclipse development environment under Linux Host Development System, especially for remote debugging purposes.

Note

PN Driver only works as a 32-bit application under 32-bit or 64-bit operating systems. Please note that IOT20x0 platform has a 32-bit operating system.

4.2 Installing Yocto Image

Requirements

You need a build host with a minimum of 50 GB of free disc space that is running a supported Linux distribution. In this chapter Linux Host Development System refers to Debian 9.3 with the Linux kernel V4.9 including RT patch. For more details, refer to the section Installing Debian (Page 25). For a list of other distributions that support the Yocto Project, refer to (https://www.yoctoproject.org/docs/2.1/mega-manual/mega-manual.html#detailed-supported-distros). To keep updated for any Debian version upgrade, please refer to the PN Driver Readme document.

4.2.1 Download IOT20x0 Board Support Package

IOT20x0 Board Support Package which you can download from (https://github.com/siemens/meta-iot2000/releases/tag/V2.2.0) is required to build standard Yocto image for IOT20x0.

To download the package run:

- `git clone https://github.com/siemens/meta-iot2000`

To use specific V2.2.0, run the following command under your repository directory:

- `git checkout tags/V2.2.0`

This package contains the following elements: meta-iot2000-bsp and meta-iot2000-example. meta-iot2000-example is an extended version of meta-iot2000-bsp and it can be used to build an exemplary image with additional tools and services on top of meta-iot2000-bsp. meta-iot2000-bsp does not configure the network. For preconfigured IP settings, you should use meta-iot2000-example package.

For updates and detailed information please refer to (https://github.com/siemens/meta-iot2000).

4.2.2 Using PREEMPT_RT kernel

The recipes-rt recipes provide package and image recipes for using and testing the PREEMPT_RT Linux kernel. In order to build the image with the linux-yocto-rt kernel, file "kas-rt.yml" under the directory [..]/meta-iot2000/meta-iot2000-example has the value for "PREFERRED_PROVIDER_virtual/kernel" configuration parameter.

Additional packages need to be included in the image. In order to include these packages, add attributes “CORE_IMAGE_EXTRA_INSTALL” and “IMAGE_INSTALL_append” to the file “kas-rt.yml”. The final content of the file “kas-rt.yml” must be as follows with the added lines highlighted:

```
header:
  version: 2
  includes:
    - kas.yml


target: iot2000-example-image-rt


local_conf_header:
  rt_kernel: |
    PREFERRED_PROVIDER_virtual/kernel = "linux-cip-rt"
    CORE_IMAGE_EXTRA_INSTALL = "libstdc++"
    IMAGE_INSTALL_append = " pciutils"
```

4.2.3 Building the Image

IOT2000 Board Support Package uses Yocto 2.1. Before you start the building process, you have to install the required packages according to the Host Development System. For a list of required packages please refer to (https://www.yoctoproject.org/docs/2.1/mega-manual/mega-manual.html#packages)

Building process is done by using the kas build tool. The kas build tool provides an easy mechanism to setup BitBake based projects by using a project configuration file. For detailed information, please refer to (https://github.com/siemens/kas)

If the meta-iot2000 package is downloaded from the link given in section 4.2.1 as a compressed archive file (not by using git as described), the following highlighted line must be added to the meta-iot2000-example/kas.yml file:

```
header:
 version: 2
 includes:
 - file: meta-iot2000-bsp/kas.yml
 repo: meta-iot2000
distro: poky-iot2000
target: iot2000-example-image
repos:
 meta-iot2000:
 path: <PATH TO META-IOT2000 FOLDER>
 layers:
 meta-iot2000-example:
meta-intel-iot-middleware:
 url: https://git.yoctoproject.org/git/meta-intel-iot-middleware
 refspec: fc8eabfa4fb54802d3f97123b9d2954450175e33
meta-nodejs:
 url: https://github.com/imyller/meta-nodejs
 refspec: eec531e97a17bfd406f3bf76dee4057dcf5286a4
local_conf_header:
 package_ipk: |
 PACKAGE_CLASSES = "package_ipk"
```

The `<PATH TO META-IOT2000 FOLDER>` part must be replaced by the path of the meta-iot2000 folder.

To build the image, run;

- `sed -i 's/wagi\/linux-cip-rt/cip\/linux-cip/g' meta-iot2000-bsp/recipes-kernel/linux/linux-cip-rt_4.4.bb`
- `kas build meta-iot2000-example/kas-rt.yml`

4.2.4 Creating a bootable media

After build process has finished, the created image can be found under the directory [..]/meta-iot2000/build/tmp/deploy/images/iot2000/. Copy this image file (e.g. "iot2000-example-image-rt-iot2000.wic") onto the microSD card via a tool for writing images to USB sticks or SD/CF cards.

Unless you add new configurations or features to the configuration files of the Yocto Linux image, you don’t need to build a new one.

There are two ways to boot the image: from SD card or from USB stick.

For updates and detailed information please refer to (https://github.com/siemens/meta-iot2000/).

Requirements

It is assumed that the user already has the required hardware and software components available and has already set up a connection configuration like the one described below. You should note that this is only one example option for a development environment for PN Driver under IOT20x0 hardware device.

Hardware components can be listed as below:

- An IOT20x0 hardware device with a valid boot image (e.g. built by Yocto, refer to section Installing Yocto Image (Page 39)) to run the PN Driver application on.
- An Ethernet cable to connect IOT20x0 and the development PC.

The figure below shows an example connection setting for SIMATIC IOT20x0.

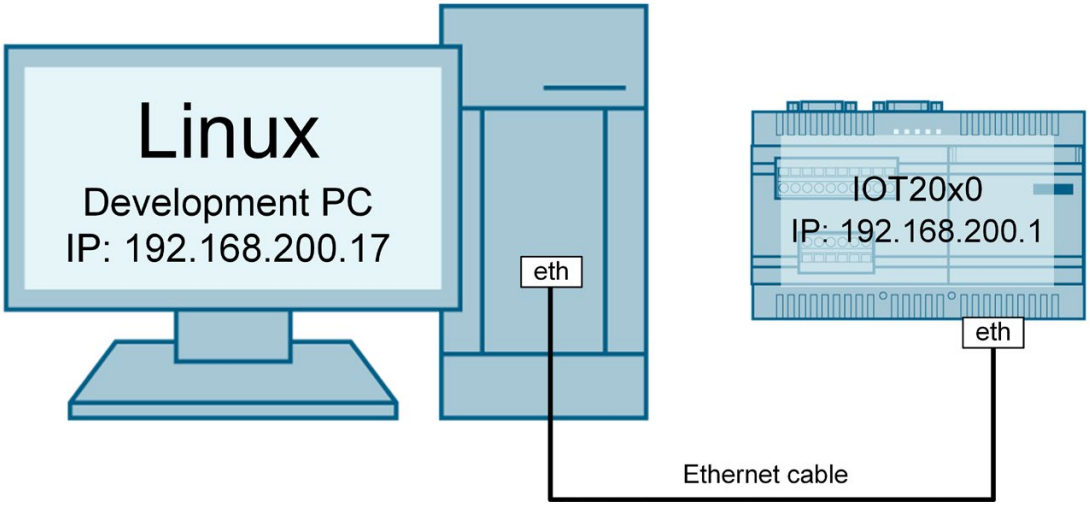


Figure 4-1 An example connection configuration between IOT20x0 and Linux based development PC

Additional Notes

You can use Linux VM on a Windows PC to download or debug the PN Driver application.

The figure below shows an example connection setting for SIMATIC IOT20x0 device.

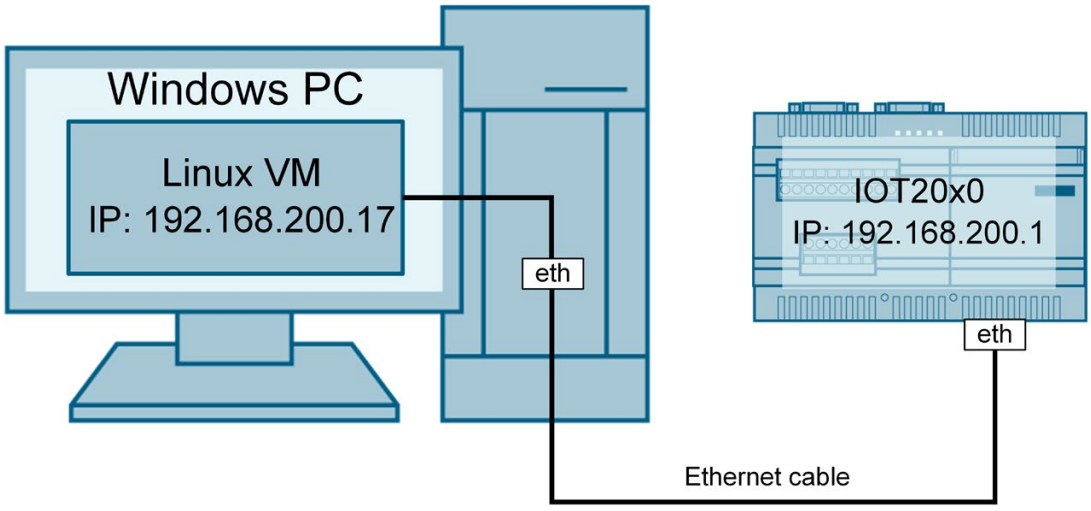


Figure 4-2 An example connection configuration between IOT20x0 and Windows based development PC

4.3 Using the PN Device Driver

Overview

PN Driver requires the PN Device Driver (PNDevDrv) for accessing the hardware and handling the interrupts. You must compile the necessary modules yourself as described below.

4.3.1 Compiling the PN Device Driver

To compile PNDevDrv, use the BitBake tool. BitBake is a generic task execution engine and executes tasks according to provided metadata that is stored in recipe (.bb), configuration (.conf) and class (.bbclass) files.

For detailed information please refer to (https://www.yoctoproject.org/docs/2.1/bitbake-user-manual/bitbake-user-manual.html).

All necessary files for compiling the PNDevDrv are under "[..]/bc_driver/src/pndevdrv-iot" directory which includes the recipe file ("pndevdrv.bb) and the files that are used to compile the Linux kernel module (under "[..]/bc_driver/src/pndevdrv-iot /files/" directory).

Procedure

1. After you have built a valid Yocto image (see the section Installing Yocto Image (Page 39)), you can generate the PN Device Driver. You should find the script "oe-init-build-env" under "[..]/meta-iot2000/poky/" directory. Before running BitBake commands, you need to run an environment setup script to set up the OpenEmbedded build environment as follows:

   – `. oe-init-build-env ../build`

   A message like “You can now run ‘bitbake <target>’” will appear in the shell.

2. Copy the "pndevdrv-iot" folder from "[..]/bc_driver/src/" directory to "[..]/poky/meta-skeleton/recipes-kernel" directory.

3. Make sure that the path of the added layer ([..]/poky/meta-skeleton) is added to the BBLAYERS variable in bblayers.conf file which is located under "[..]/meta-iot2000/build/conf" directory.

4. Run the following commands in "[..]/meta-iot2000/build". Clean the old object files before compiling the sources. To do so, enter:

   – `bitbake -f -c clean pndevdrv`

   To compile:

   – `bitbake -f -c compile pndevdrv`

   And to build:

   – `bitbake pndevdrv`

   The build result should be found under "[..]/meta-iot2000/build/tmp/work/iot2000-poky-linux/pndevdrv/1.0-r0" directory. The newly generated "PnDevDrv_32.ko" file can be used to install PNDevDrv at boot routine.

In addition to the actual driver, a shared object file must be generated. You can generate this shared object file with the "Makefile_SharedObject" in "[..]/bc_driver/src/pndevdrv/bin/Linux".

1. The standard Yocto Project SDK, which provides a cross-development toolchain and libraries, must be installed. Before installation, you should first generate Linux SDK. To do so, run the following command in "[..]/meta-iot2000":

   – `kas build meta-iot2000-example/kas-sdk-linux-x64.yml`

   The build result should be found under "[..]/meta-iot2000/build/tmp/deploy/sdk" directory. To install Yocto SDK, run the newly generated "poky-iot2000-glibc-x86_64-iot2000-example-image-i586-nlp-32-toolchain-2.2.2.sh"

2. Open a new shell and set the current directory to "[..]/bc_driver/src/pndevdrv/bin/Linux".

   Then, set environment variables. To do so, enter:

   – `. /opt/poky-iot2000/2.2.2/environment-setup-i586-nlp-32-poky-linux`
   (Please pay attention to the blank between '.' and '/' characters)

   It is assumed that script is generated and located under "/opt/poky-iot2000/2.2.2/" directory.

3. To generate the shared object file, enter the following:

   – `make -f Makefile_SharedObject`

   As a result, the two shared object files ("*.so") for 32-bit and 64-bit applications are generated and stored in the same folder ("[..]/bc_driver/src/pndevdrv/bin/Linux").

4.3.2 Loading the PN Device Driver

Procedure

1. Copy PnDevDrv_32.ko to target device (IOT20x0 hardware device) (see the section Transferring files from the host to the target (Page *)). Do not use the home directory as a location because it is not accessible at boot time. For example, you can use "/usr/src" directory.
   - `cp PnDevDrv_32.ko /usr/src/`
2. Link the Linux kernel module. To do so, enter the following:
   - `ln -s /usr/src/PnDevDrv_32.ko /lib/modules/4.4.105-cip15-rt10`
3. After having linked PnDevDrv_32.ko, execute:
   - `depmod -a`
4. Copy "pndev.rules" file which is located under "[..]/bc_driver/src/pndevdrv/bin/Linux/iot2000/" directory to "/etc/udev/rules.d/" directory in target device.
   - `cp pndev.rules /etc/udev/rules.d`
5. Restart IOT20x0 hardware device to make the changes take place. To do so, enter:
   - `reboot`
6. Check if the PNDevDrv is installed correctly. To do so, enter:
   - `lsmod`

   and see if PnDevDrv_32 is listed among the installed modules.

4.3.3 Unloading the PN Device Driver

If you want to unload PN Device Driver from your target device (IOT20x0 hardware device), you can do so by following the procedure below.

Procedure

1. To remove PNDevDrv Linux kernel module, enter:
   - `rmmod PnDevDrv_32`
2. Then you can load PNDevDrv once again. Enter the directory where you have located the Linux kernel module (e.g. "/usr/src")
   - `cd /usr/src`

   Then run the following command:
   - `insmod PnDevDrv_32.ko`

4.4 Building the PN Driver application

The compiler "i586-poky-linux-gcc" is used to build applications for IOT20x0 and therefore Yocto Project SDK must be installed before building the PN Driver application. Before installation, you should first build Linux SDK. To do so, run the following command in "[..]/meta-iot2000":

- `kas build meta-iot2000-example/ kas-sdk-linux-x64.yml`

The build result should be found under "[..]/meta-iot2000/build/tmp/deploy/sdk" directory. To install Yocto SDK, run the newly generated "poky-iot2000-glibc-x86_64-iot2000-example-image-i586-nlp-32-toolchain-2.2.2.sh".

PN Driver applications for IOT20x0 can be built on the command line or in the Eclipse IDE.

4.4.1 Build on Command Line

Procedure

To build PN Driver application on the command line:

1. Start a new shell and set the current directory to "[..]/pn_driver/src/examples/shared/iot2000/build".

   Then set the build environment to the Linux compiler. To do so, enter:

   – `. /opt/poky-iot2000/2.2.2/environment-setup-i586-nlp-32-poky-linux`
   (Please pay attention to the blank between '.' and '/' characters)

2. Build PN Driver as a library by calling "make" (or "make debug" if you want to build in debug mode) in "[..]/pn_driver/src/examples/shared/iot2000/build". Make sure that "libpndriver.a" has been created under "[..]/pn_driver/src/examples/shared/iot2000/build/lib" directory.
3. Build the application within the same shell and link the PN Driver library to the application by calling "make" (or "make debug" if you want to build in debug mode) in 'iot2000' folder of the application, e.g. "[..]/pn_driver/src/examples/test_app/iot2000/build". Make sure that binary of the application has been created under e.g. "[..]/pn_driver/src/examples/test_app/iot2000" directory.

4.4.2 Build in Eclipse IDE

Requirement

The Eclipse development environment must be ready for use for starting and debugging PN Driver under Linux (virtual Linux) machine. In order to use remote debugging you will need a more recent version of Eclipse than provided by the Eclipse package in Debian 9.3, such as Eclipse Oxygen. To install the Eclipse Oxygen, run the following commands:

- `wget http://ftp.fau.de/eclipse/technology/epp/downloads/release/oxygen/3a/eclipse-cpp-oxygen-3a-linux-gtk-x86_64.tar.gz`
- `sudo tar xfz eclipse-cpp-oxygen-3a-linux-gtk-x86_64.tar.gz -C /opt/`
- `sudo ln -s /opt/eclipse/eclipse /usr/bin`

Procedure

To build PN Driver application in Eclipse IDE:

1. Start a new shell and set the build environment to the Linux compiler. To do so, enter:
   - `. /opt/poky-iot2000/2.2.2/environment-setup-i586-nlp-32-poky-linux`
2. Start the Eclipse IDE from the same shell with the following command:
   - `eclipse`
3. Load the application for IOT20x0 that is provided with PN Driver in Eclipse using [File > Import > General > Existing Projects in Workspace]
4. Select the two projects "pndriver_32" (located in "[..]/pn_driver/src/examples/shared/iot2000/") and "test_app" (located in "[..]/pn_driver/src/examples/test_app/iot2000/") and click "Finish".
5. Compile the two projects using [Project > Build All].

6. Make sure that libpndriver.a has been created. It is located under "[..]/pn_driver/src/examples/shared/iot2000/build/lib". Also "test_app" should be located under "[..]/pn_driver/src/examples/test_app/iot2000".

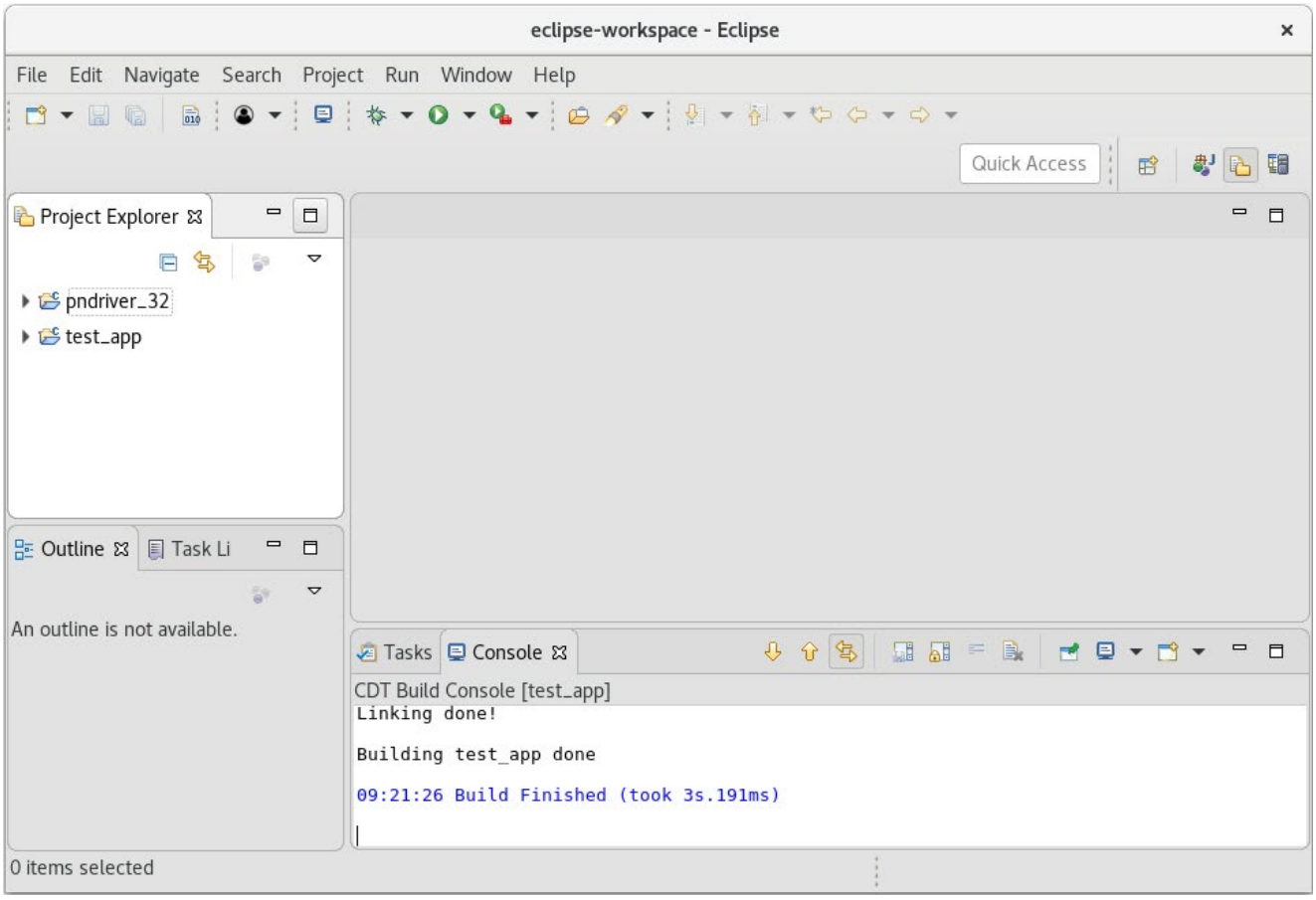


Figure 4-3 Eclipse - Build All

4.5 Connecting to the target device

You can access the target device (IOT20x0 hardware device) via remote connection. For this purpose “Secure Shell” (SSH) protocol can be used. SSH enables secure system administration and file transfer.

Before connecting to IOT20x0 hardware device from Linux Development System, IP address configurations must be done. In our example, IP addresses “192.168.200.17” and “192.168.200.1” are assigned to Linux Development System and IOT20x0 hardware device, respectively. Note that IP Suite is already configured for IOT20x0 hardware device during Yocto image generation. You must configure the IP Suite for your Linux Development System.

After IP configuration has been done successfully, open a new shell in Linux Development System and enter:

- `ssh root@192.168.200.1`

Additional Notes

If you want to connect to the target device via serial console, serial port settings are as follows:

Table 4- 1 Hardware components of the test application

Setting	Value
Baudrate	115200
Data	8 Bits
Parity	None
Stop bits	1
Hardware flow control	No
Software flow control	No

4.6 Transferring files from the host to the target

After having connected successfully to the target IOT20x0 hardware device, transfer the necessary files to run the PN Driver application. File transfer from the host (Development Machine) to target (IOT20x0 hardware device) can be done via USB flash drive or remote access.

4.6.1 File transfer via USB flash drive

File transfer from the host to the target can be done via USB flash drive. Also, for running the PN Driver application for IOT20x0 hardware device, the USB flash drive can be used.

1. Generate the following files:
   - For PN Device Driver shared object file ("PnDev_DriverU32.so") please see the section Compiling the PN Device Driver (Page 43),
   - For binary file of the PN Driver application (e.g. "test_app" and libpndriver.a (optional)) please see the section Building the PN Driver application (Page 47),
   - Hardware configuration file (a test XML file is provided under iot2000/PNDriverBase_TestApp directory which is located under application directory)
2. Copy the files mentioned above to the USB flash drive (under the same directory).
3. Locate USB flash drive to the USB port on the IOT20x0 hardware device and mount. To do so, follow the commands below:
   - `mkdir /mnt/usb`
   - `mount /dev/sda1 /mnt/usb`
4. Check the content of USB flash drive under /mnt/usb folder. Note that /mnt/usb is given as an example here, and you can use any folder to mount the USB drive.

Note

If the name of your device is other than sda1, modify the command (mount /dev/sda1 /mnt/usb) accordingly. You can see the list of your devices by running the command “cat /proc/partitions”.

4.6.2 File transfer via Remote Access

With the use of remote access to the IOT20x0 hardware device with two Ethernet controllers (only IOT2040 has 2 interfaces), it is possible to transfer the required files via Remote System Explorer (RSE) toolkit of Eclipse IDE.

Required files are:

- PnDev_DriverU32.so,
- Binary file of PN Driver application (e.g. “test_app” and libpndriver.a (optional)),
- Hardware configuration file

The RSE allows you to connect and work with a variety of remote systems, e.g. displaying remote file systems, transferring files between hosts, performing remote search and executing commands.

Before configuring a new connection between Linux Host Development System and IOT20x0 hardware device, IP address configurations must be done. In our example, IP addresses “192.168.200.17” and “192.168.200.1” are assigned to Linux Host Development System and IOT20x0 hardware device respectively.

You can use the “ping” command to test the remote connection between Linux Host Development System and IOT20x0 hardware device.

To configure a new connection:

1. Install the Remote System Explorer (RSE) and Target Communication Framework (TCF) plugins in your Eclipse IDE using [Help > Install New Software]

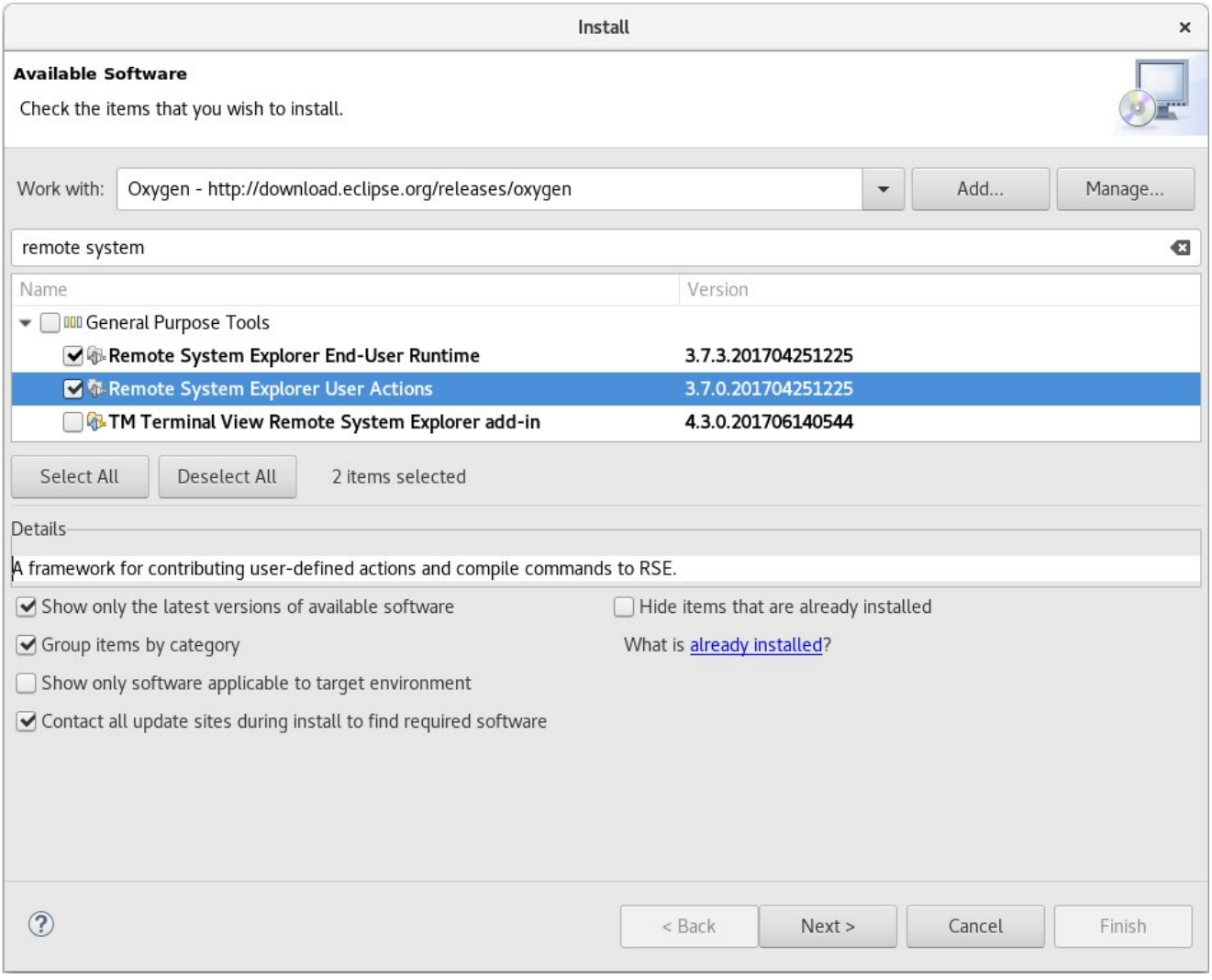


Figure 4-4 Install RSE plugin

2. In Eclipse, switch to the RSE perspective using [Window > Open Perspective > Remote System Explorer].

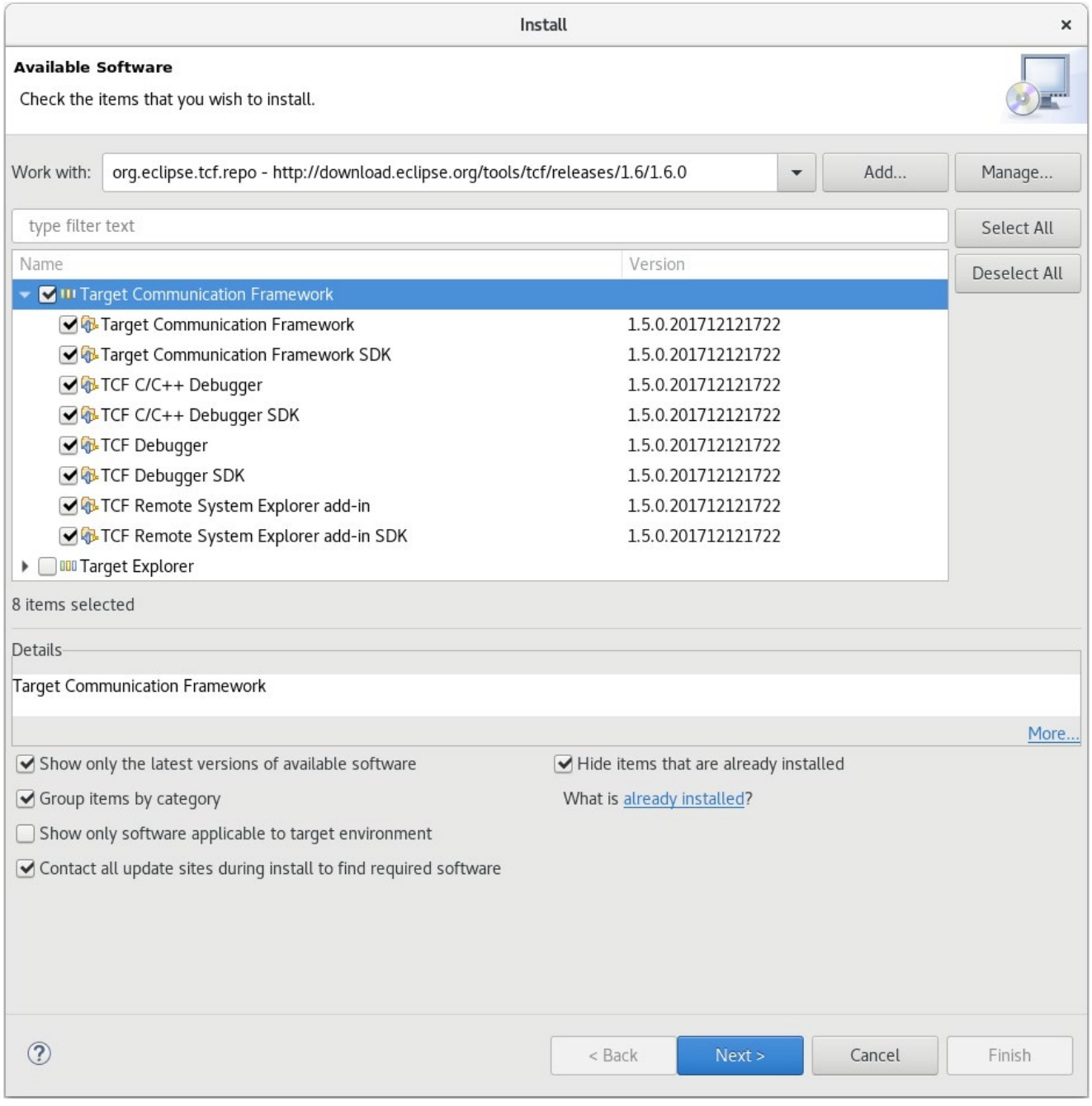


Figure 4-5 Install TCF plugin

3. In Remote System Explorer view:

   Right-click and select “New/Connection”.

4. In “New Connection” dialog, select “TCF” as “System type” and click “Next” button.

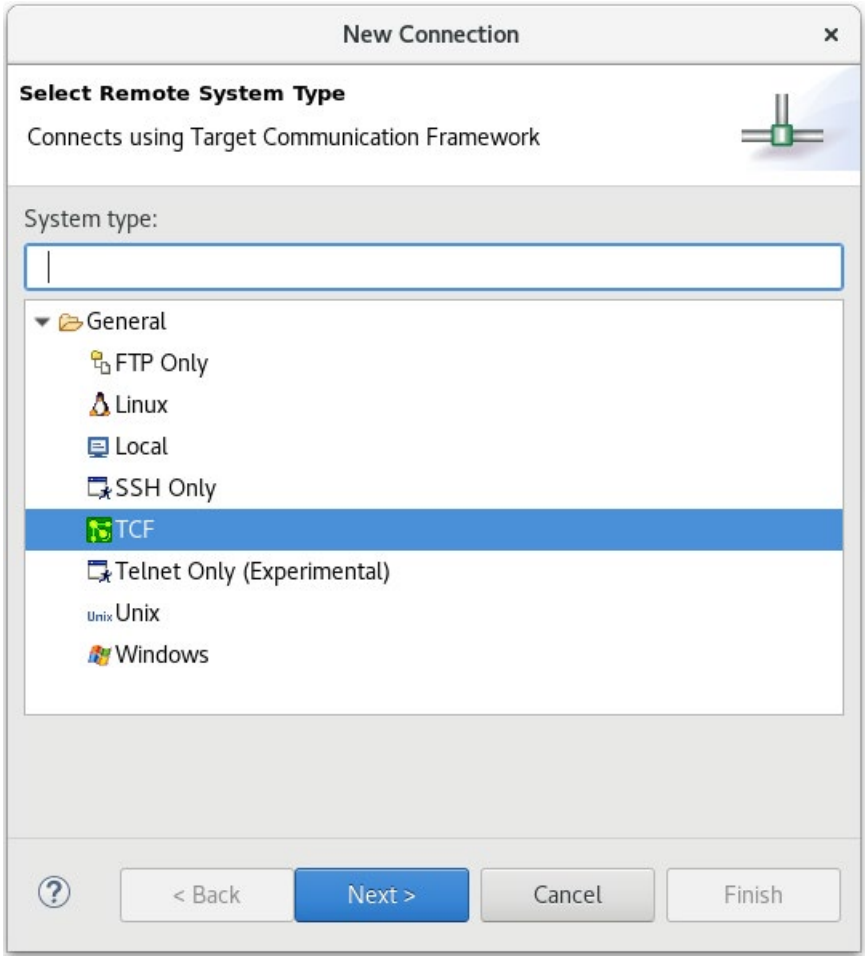


Figure 4-6 Generate new TCF connection

5. Enter IP address of the target machine as “Host name” (IOT20x0 hardware device) and an arbitrary string to name new connection as “Connection name”.

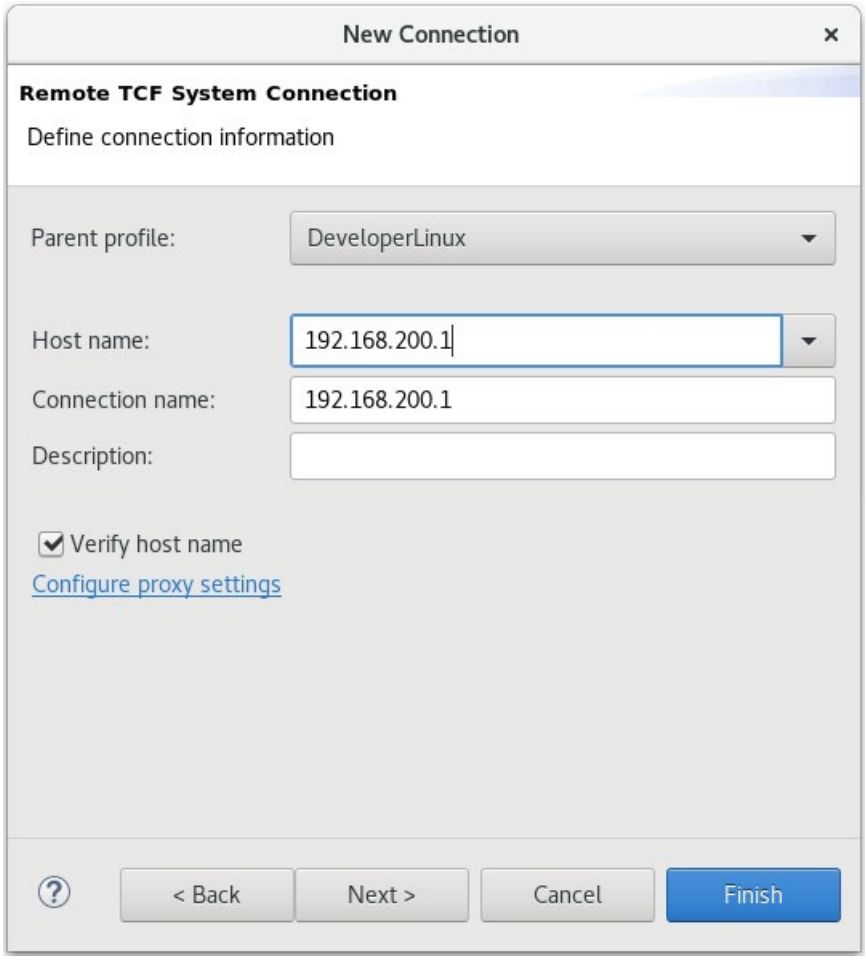


Figure 4-7 Configure new TCF connection

6. To verify the remote connection, check the window as shown in the figure below. Enter “root” as “User ID”. No password is required.

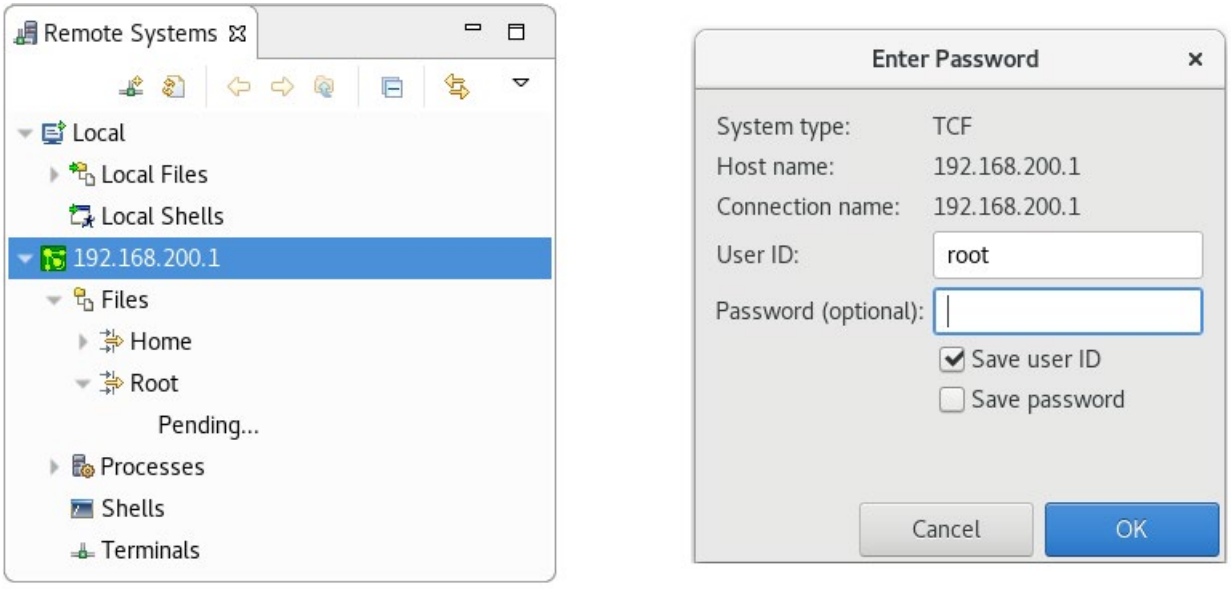


Figure 4-8 User ID and password configurations of new TCF connection

7. A new connection should be displayed in "Remote Systems" view and you should be able to view the file system of your target device.

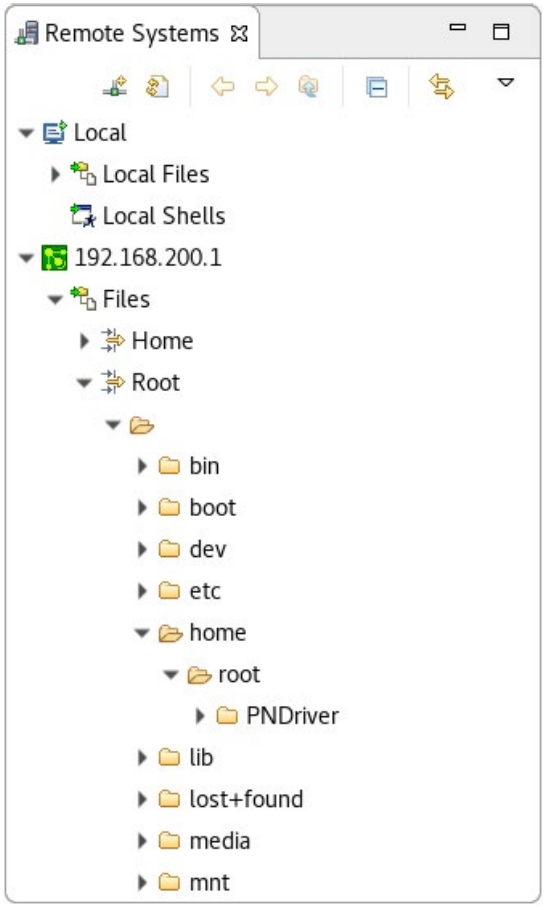


Figure 4-9 View content of the file system

4.7 Running the application on the target

Requirements

Before you start PN Driver, you must ensure that the application can locate the corresponding XML file as well as the shared object file of the PN Device Driver. Both files must be stored in the same directory as the executable PN Driver file.

After connection is established to the target device via remote connection (see the section Connecting to the target device (Page 50)), the PN Driver application must be run manually via SSH. To do so:

1. Set the current directory to the directory where PN Driver application binary is located on your target device.
2. Run the PN Driver application binary:
   - `./test_app`

When you run the application, a similar screen on console as in the figure below is displayed.

```
--------------------------------------------------------------------------------
PROFINET Driver | test program
================================
1...Help (show menu)
================================
2...SERV_CP_Startup
4...SERV_CP_Shutdown
================================
5...Open Controller
6...Close Controller
================================
7...Set Mode PNIO_MODE_OFFLINE
8...Set Mode PNIO_MODE_CLEAR
9...Set Mode PNIO_MODE_OPERATE
================================
10...Device Activate
11...Device Deactivate
================================
12...I&M Data Read
13...I&M Data Write
================================
15...SERV_CP_set_trace_level
================================
16...IO Data Read/Write
================================
17...Perform record operations
================================
18...List Modules and Submodules
================================
19...Start Netload Test
================================
21...Open PDEV interface
22...Close PDEV interface
================================
23...Interface Record Read
================================
24...Interface Set IP and NoS
================================
25...Enable/Disable SNMP
================================
26...Show PN Driver Version Info
================================
99...Write trace data to file
================================
0... QUIT
2

EPS PnDevDrv (P04.03.00.00_00.01.05.01) install done

EPS PnDevDrv Found PN Boards:
----------------------------------------
Bus  0, Dev 20, Fct  7  -  Intel Quark IOT2000
----------------------------------------

Found network adapters:
----------------------------------------
 0 ... Bus  0, Dev 20, Fct  7 - IOT2000 board
----------------------------------------

Enter the number of the network adapter: 0
```

Figure 4-10 PN Driver application menu on IOT20x0

4.8 Debugging the PN Driver application

You can debug the PN Driver application via GNU debugger (GDB) or remotely via Eclipse IDE.

4.8.1 Debugging the PN Driver application via GNU Debugger

GDB, the GNU Project Debugger, allows you to see what is going on 'inside' another program while it executes. For detailed information please refer to (https://www.gnu.org/software/gdb/).

Procedure

The PN Driver application should be run manually via remote connection. To do so:

1. Set the current directory to the directory where PN Driver application binary is located on your IOT20x0 hardware device.
2. To start the GDB, use the following command:
   - `gdb ./test_app`
3. Before runnig the application, set the breakpoints. For example:
   - `(gdb)break pnd_test.c:19 #to set a break point for file pnd_test.c, line 19`
4. To run the application, enter:
   - `run`
5. To exit the GDB, use the following command:
   - `quit`

4.8.2 Debugging the PN Driver application via Remote Debugging

Requirement

The Eclipse development environment must be ready for use for starting and debugging PN Driver under Linux (virtual Linux) machine.

Procedure

To debug the PN Driver application in Eclipse IDE:

1. Start a new shell and set the build environment to the Linux compiler. To do so, enter:
   - `. /opt/poky-iot2000/2.2.2/environment-setup-i586-nlp-32-poky-linux`
2. Start the Eclipse IDE from the same shell with the following command:
   - `eclipse`

3. Load the application for IOT20x0 that is supplied with PN Driver in Eclipse using [File > Import > General > Existing Projects] in the workspace.

   Select the two projects "pndriver_32" (located in "[..]/pn_driver/src/examples/shared/iot2000/") and "test_app" (located in "[..]/pn_driver/src/examples/test_app/iot2000/") and click "Finish".

4. After having loaded the PN Driver application, Debug Configuration should be done. In “Debug Configurations” menu, following fields should be filled in as below:

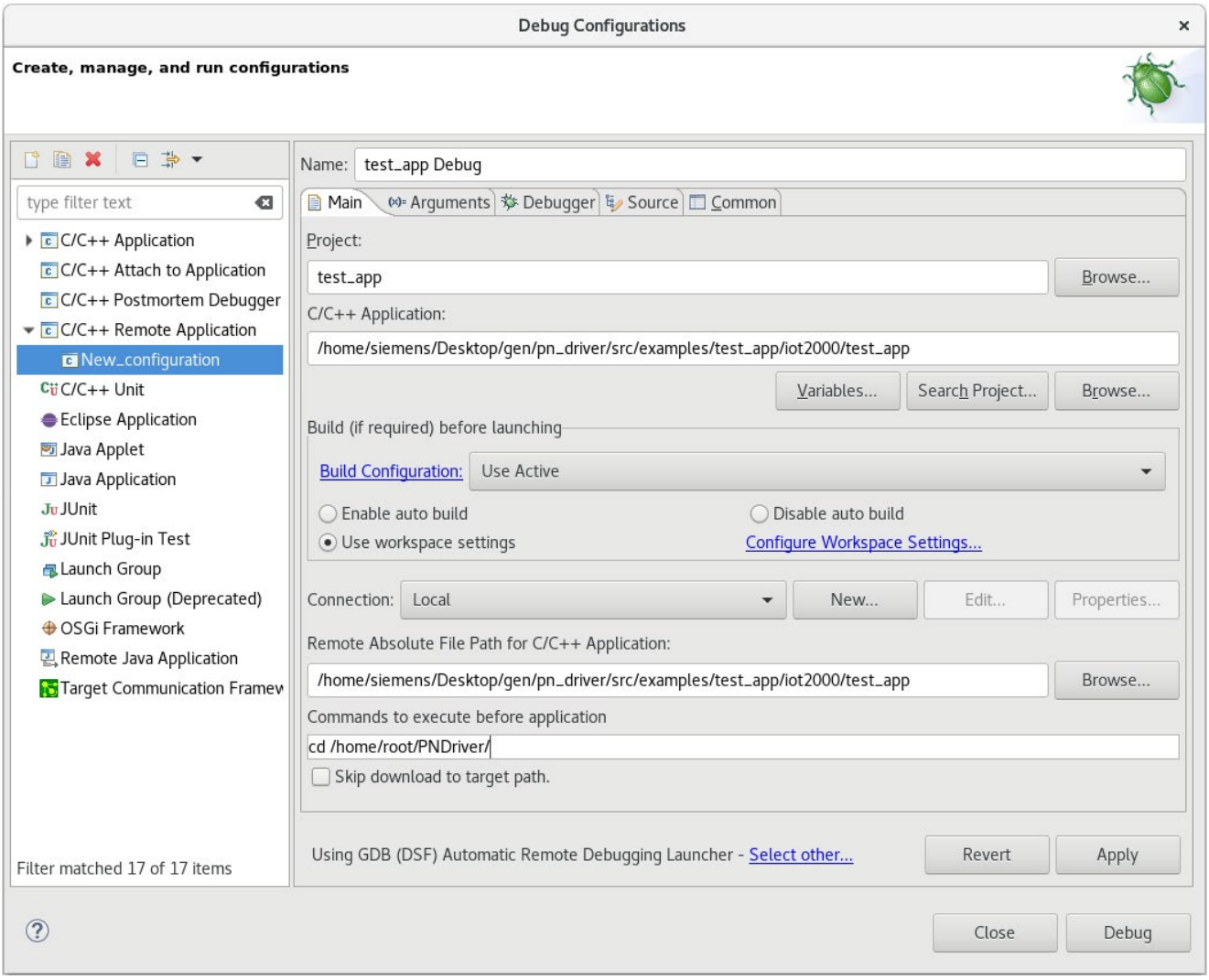


Figure 4-11 Remote Debug Configurations – “Main” tab

- “Main” tab:
  - C/C++ Application: Path to the directory where PN Driver test application is located under Host Development System.
  - Connection: IP address of IOT20x0 hardware device.
  - Remote Absolute File Path for C/C++ Application: Path to the directory where PN Driver test application binary is located in your target device.
  - Commands to execute before application: Command to set current path to the path where PN Driver test application binary is located in your target device.
- "Source" tab:
  - Add “Source Lookup Path” for both PN Driver library and test application.

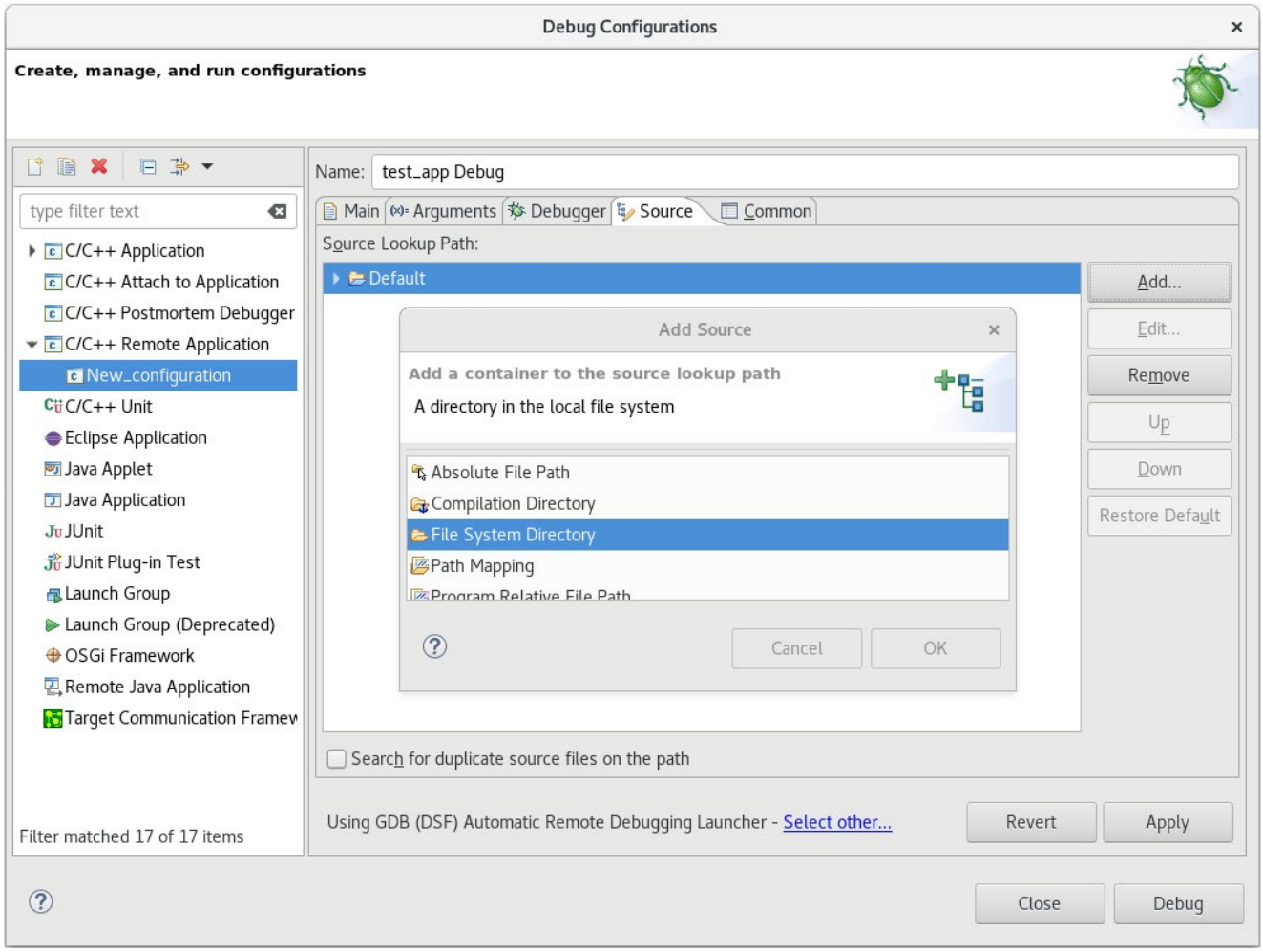


Figure 4-12 Remote Debug Configurations – “Source” tab

- “Debugger” tab:
  - GDB debugger: "/opt/poky-iot2000/2.2.2/sysroots/x86_64-pokysdk-linux/usr/bin/i586-poky-linux/i586-poky-linux-gdb"
  - GDB command line: “.gdbinit”

Quick start for CP1625 Stand-alone 5

5.1 Quick start for CP1625 Stand-alone

Overview

This section describes the steps required for commissioning PN Driver under a custom Linux operating system, built with Buildroot (https://buildroot.org/).

Customers are responsible for the target system build (PN Driver under CP 1625Dev board) with the help of the source code of PN Driver V2.2, application examples and the relevant documentation files.

Requirements

You need a CP 1625Dev board and Linux Host Development System for both building the Buildroot image and PN Driver application. You must configure your board for stand-alone usage. For more details, please refer to CP 1625 operating instructions in (https://support.industry.siemens.com/cs/ww/en/view/109756564). You also need a cable for serial connection. We recommend using TTL-232R-RPI cable from FTDI.

Note

PN Driver CP1625 Stand-alone variant only supports little endian environments and 32-bit operating systems.

5.2 Installing Buildroot Image

Requirements

uildroot is designed to run in Linux environments. For details about system requirements, please refer to (https://buildroot.org/downloads/manual/manual.html#requirement). In this chapter Linux Host Development System refers to Debian 9.3 with the Linux kernel V4.9 including RT patch. For more details, refer to the section Installing Debian (Page 25).

To keep updated for any Debian version upgrade, please refer to the PN Driver Readme document.

5.2.1 Downloading Buildroot package

Buildroot is required as Linux system generation tool for building the Linux image. Recommended Buildroot version is buildroot-2017.02.9 which can be downloaded from (https://buildroot.org/downloads/).

Download “buildroot-2017.02.9.tar.gz” from the download page and extract it. To do so, run the following command in the directory where the downloaded package is located:

- `tar -xvf buildroot-2017.02.9.tar.gz`

Buildroot has dependencies toon some software packages. For detailed information please refer to (https://buildroot.org/docs.html).

5.2.2 Configuring Buildroot

Buildroot has to be configured for PN Driver. To do so, copy the following add-on file from PN Driver CD to the directory where the Buildroot package is located:

"[..]/contrib/Buildroot/siemens_cp1625_br2017.02.9_addon.tar.gz"

Then extract the file by running the following command:

- `tar -xvf siemens_cp1625_br2017.02.9_addon.tar.gz`

5.2.3 Building the image

In order to build the Linux Image run the following commands:

- `cd buildroot-2017.02.9`
- `make clean`
- `make cp1625_nand_defconfig`
- `make`

“"rootfs.ubi" will be created under "[..]/buildroot-2017.02.9/output/images" directory after the build process has been successfully completed. Note that internet connection is required when running commands above for the first time.

5.2.4 Adding custom files to the Linux image

To run PN Driver application on CP 1625Dev board, you need to transfer the application binary and the configuration file to the board. For details about creating the application binary, please refer to the section Building PN Driver application (Page 72). Buildroot's overlay structure is used for integrating files in the Linux image. To do so, follow the procedure below.

Procedure

1. Copy the files that you want to be included in the image (e.g. the application binary and the configuration file) to the directory "[..]/buildroot-2017.02.9/board/cp1625/rootfs_overlay/root".
2. Run the following command in "[..]/buildroot-2017.02.9"
   - `make`
3. The image called "rootfs.ubi" will be created in "output/images" directory. Use this image to boot CP 1625Dev board as described in the section Booting the image (Page 69).

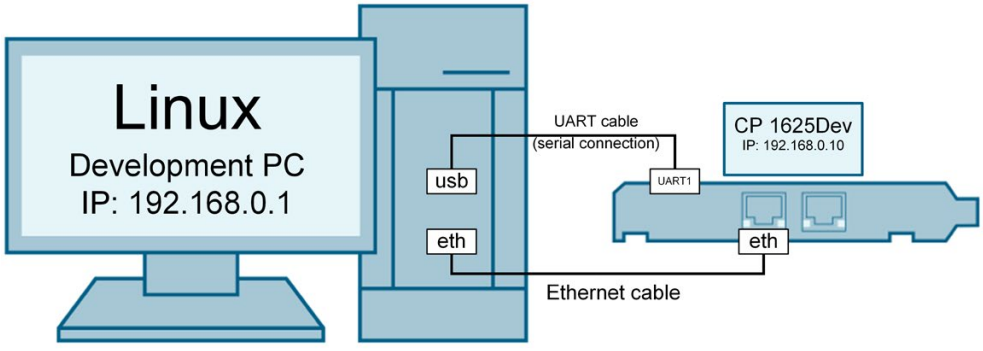
5.2.5 Connecting to the target device via serial console

Connection to CP 1625Dev board from your host development system is possible via serial console. UART1 serial interface is used for the console connection. Serial port settings are as follows:

Table 5- 1 Hardware components of the test application

Setting	Value
Baudrate	115200
Data	8 bits
Parity	None
Stop bits	1
Hardware flow control	No
Software flow control	No

The figure below shows an example connection setting for CP 1625Dev board:



UART cable is for the access to the console. Ethernet cable is for the communication via TFTP.

Figure 5-1 An example connection configuration for CP1625 Stand-alone

5.2.6 Flashing the bootloader

In order to boot your CP 1625Dev board, you need a bootloader which was created when you build the image as described in the section Building the image (Page 65). “bootloader.bin” will be under "[..]/buildroot-2017.02.9/output/images” directory. You must copy it to your Linux PC. To flash the bootloader, follow the procedure below in your Linux PC.

Procedure

1. Configure your board for PCIe usage. For more details, please refer to CP 1625 operating instructions in (https://support.industry.siemens.com/cs/ww/en/view/109756564).
2. Plug the board in a Linux PC.
3. Build the bootloader application by running "make" in "[..]/bc_driver/src/pndevdrv/tools/cp1625_flash_bootloader". The application binary will be created in this directory.
4. In order to run this application on CP 1625Dev board, you need to use PNDevDrv. For details of building and installing the PNDevDrv kernel module and binding it to your board, refer to the section Using the PN Device Driver for the Linux variant (Page 29). You also need to build a shared library for PNDevDrv and copy it where your application is located. The procedure for this is also explained in the same section.
5. Start the application by running the following command.
   - `sudo ./cp1625_flash_bootloader64 (or cp1625_flash_bootloader32)`
6. Select your CP 1625Dev board. If it is the only device, it will be selected automatically.
7. Select option “(1) CP1625 – Flash NAND Bootloader”.
8. Enter the path for “bootloader.bin”.
9. Select option “(c) Cancel” to close the application.
10. Shutdown your PC and unplug the CP 1625Dev board.
11. Reconfigure it for stand-alone usage.

5.2.7 Booting the image

To boot CP 1625Dev board with the image created in the host development system, you must use TFTP server. To do so, follow the procedure below.

Procedure

1. TFTP client that runs on CP 1625Dev board assumes IP address of the TFTP server is 192.168.0.1. Bind 192.168.0.1 IP address on Build Host machine where TFTP server will run.
2. Install the TFTP server by running the following command:
   - `sudo apt install tftpd-hpa`
3. Configure the TFTP server by editing the file “/etc/default/tftpd-hpa”. The content of the file should be as follows:

```
# /etc/default/tftpd-hpa

TFTP_USERNAME="tftp"

TFTP_DIRECTORY="/var/lib/tftpboot"

TFTP_ADDRESS="192.168.0.1:69"

TFTP_OPTIONS="--secure"
```

4. Copy the Buildroot image in the “/var/lib/tftpboot/” directory by running the following command:
   - `sudo cp [..]/buildroot-2017.02.9/output/images/rootfs.ubi /var/lib/tftpboot/`
5. Connect the Ethernet port 1 of CP 1625Dev board to your Linux Host Development System.

   For the following steps you must be able to run commands on the board. To do so, you must connect to the board from your host via serial console. Refer to the section Connecting to the target device via serial console (Page 67) for details of serial connection.

6. Power up the board and stop autoboot by pressing ‘s’ to enter “U-Boot” menu.

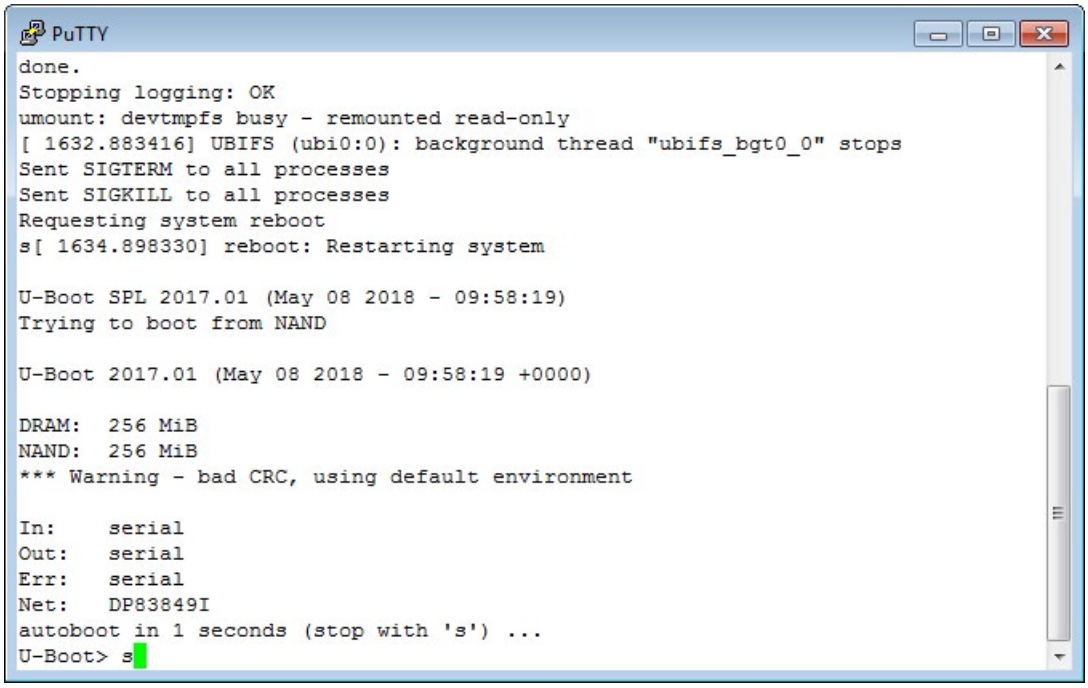


Figure 5-2 Open "U-Boot" menu

7. Load ‘rootfs.ubi’ file from TFTP server by running the following command:
   - `run prgubiimg`

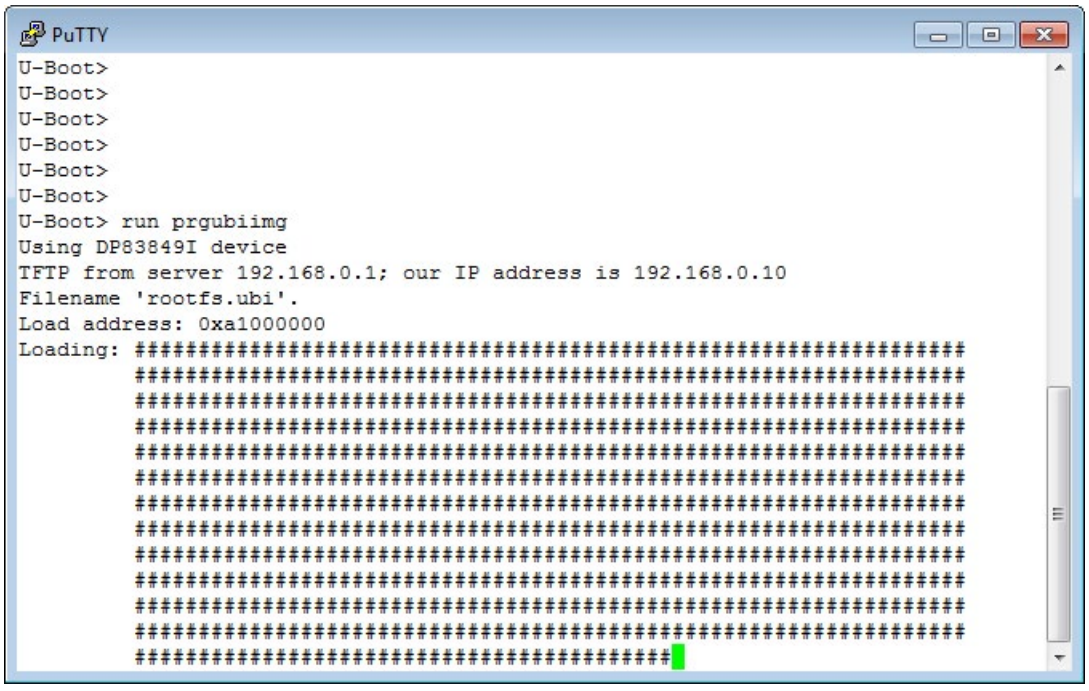


Figure 5-3 Load rootfs.ubi file

8. After loading has been finished, a message like “PROGRAM SUCCEEDED” will be shown. Then you should reboot the board by entering “reset”. The board will boot with the image taken from TFTP server.

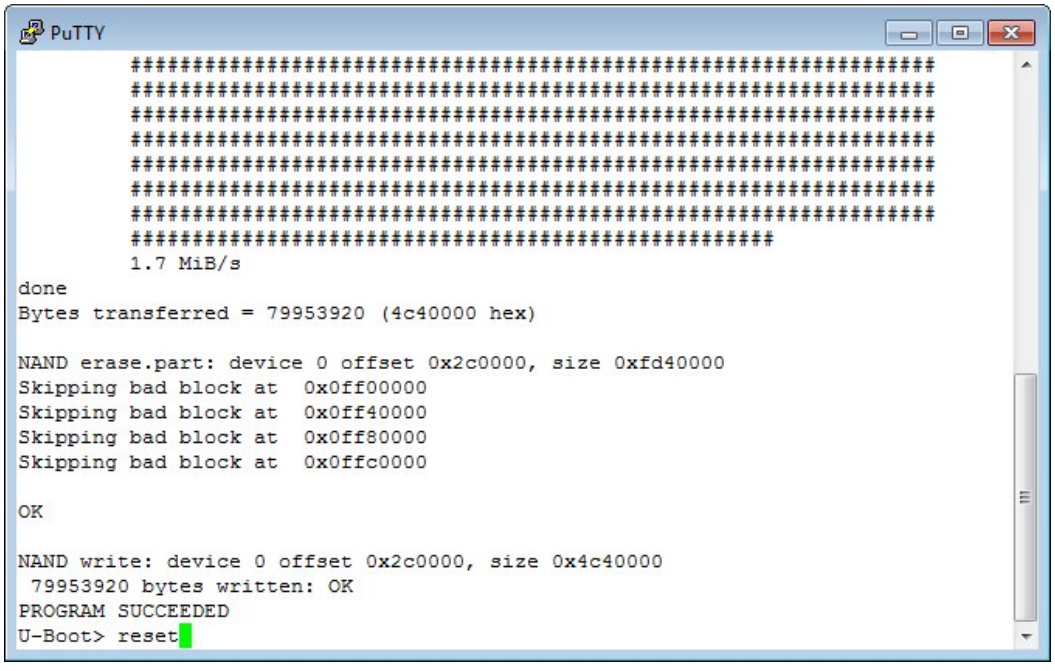


Figure 5-4 Reset CP 1625Dev board

5.3 Building the PN Driver application

Buildroot cross compilation toolchain is used to build applications for CP1625 Stand-alone variant. The toolchain is provided in the Buildroot package.

Procedure

To build the PN Driver application, follow the steps below:

1. Start a new shell and set the current directory to "[..]/buildroot-2017.02.9/". Then set the build environment to the Linux compiler. To do so, run the following command:
   - `source ./set_toolchain_env_buildroot_complete.sh`
2. If you want to use the same network interface for both IP communication of third party applications and PROFINET communication, build PN Driver as a library by running "make" (or "make debug" if you want to build in debug mode) in "[..]/pn_driver/src/examples/shared/cp1625_standalone_native/build". Otherwise, run "make" (or "make debug" if you want to build in debug mode) in "[..]/pn_driver/src/examples/shared/cp1625_standalone/build". Make sure that "libpndriver.a" has been created in the subdirectory "lib".

   Please refer to the section Using PROFINET interface for IP communication (Page 90) for the details regarding IP communication over the PROFINET interface.
3. Build the application within the same shell and link the PN Driver library to the application by running "make" (or "make debug" if you want to build in debug mode) in "cp1625_standalone_native" or "cp1625_standalone" folder of the application, depending on which version of the library has been built in the previous step, e.g., "[..]/pn_driver/src/examples/test_app/cp1625_standalone_native/build" or "[..]/pn_driver/src/examples/test_app/cp1625_standalone/build". Make sure that binary of the application has been created in the parent directory.

5.4 Running the application on the target

After CP 1625Dev board has booted with an image that contains PN Driver application and necessary hardware configuration file, you can run the application by executing the following command:

- `./test_app`

When you run the application, a similar screen is displayed as shown in the figure below:

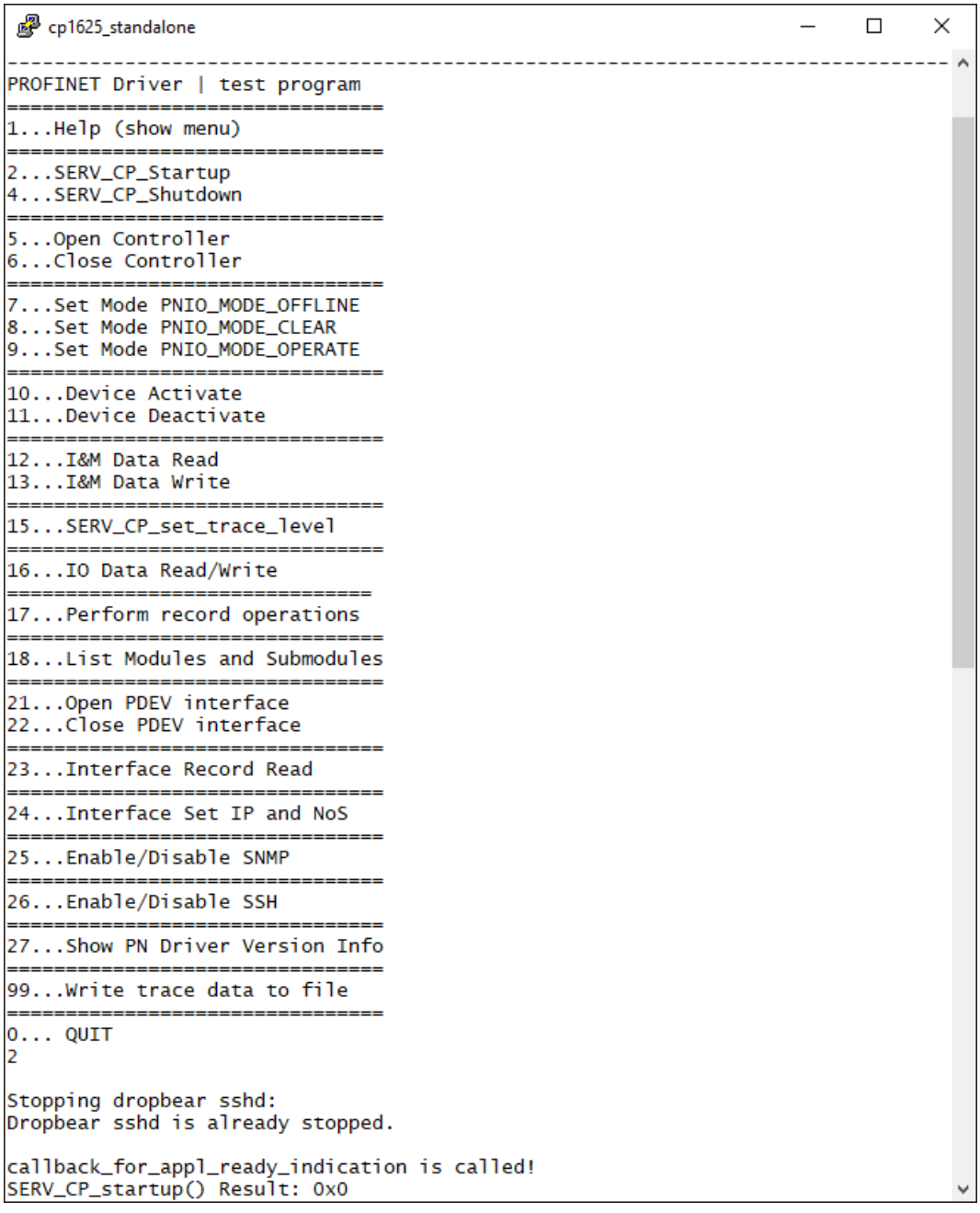


Figure 5-5 PN Driver application menu on CP1625 Stand-alone

5.5 Transferring files from the target to the host

Firmware download or file exchange to CP1625 Stand-alone variant can be performed either by updating the whole file system image from u-boot using serial terminal or by using Secure Shell (SSH). With the introduction of the standard Linux networking stack support, applications are able to use the ethernet connection with the created virtual interface, over which SSH connections can be made for file exchange and shell access. By this way, big files can be exchanged faster in comparison with the serial port connection, but serial port connection is required in the initial setup since PN Driver must be running to use SSH.

5.5.1 Transferring files from the target to the host using serial port

It is possible to transfer files from CP 1625Dev board to your host development system via serial console. Different transfer protocols are available depending on the serial console application that you use, such as ExtraPutty or Tera Term. For example you can use ZMODEM with ExtraPutty as follows.

Procedure

1. Start sending the file to the host by running the following command:
   - `sz -e -b -X filename`
2. Start receiving the file in ExtraPutty: [File Transfer > ZMODEM > Receive]
3. Enter the name of the file you want to transfer and the file will be transferred into the ExtraPutty folder.

5.5.2 Transferring files between the target and the host using Secure Shell (SSH)

It is possible to transfer files between the CP 1625Dev board and the host development system using SSH if you are using the same network interface for both PROFINET and other IP based network communication. This feature is only supported by the CP1625 Stand-alone variant.

To use the ethernet connection, two preconditions must be satisfied:

- **PN Driver must be running:** This is achieved by calling "SERV_CP_startup()" IO-Base function. Once "SERV_CP_startup()" returns without any error, a virtual network interface emerges within Linux system. The virtual network interface has the name of "pnio1" and it is accessible for the applications running on the same host as PN Driver until the PN Driver is shutdown via "SERV_CP_shutdown()" IO-Base function. Once SERV_CP_shutdown() returns without any error, the virtual network interface disappears.
- **SSH server must be enabled:** You can enable/disable SSH server via PN Driver example application or through some initialization scripts during the booting up phase of the board as follows:

Procedure

1. First, to be able to login through SSH, the public key of the host development system must be provided to the CP 1625Dev board. To generate the public/private keys use "ssh-keygen" application by running the following command:
   - `ssh-keygen`
2. Then, copy the public key that is created on the host development system into a file called "authorized_keys" by running the following command:
   - `cat id_rsa.pub >> /home/user_name/.ssh/authorized_keys`
3. And copy the "authorized_keys" file under the "[..]/buildroot-2017.02.9/board/cp1625/rootfs_overlay/root/.ssh" directory.
4. Now you can enable/disable the SSH server via the PN Driver example application ("test_app"). After building PN Driver application and booting the CP 1625Dev board, you can use menu item "Enable/Disable SSH" to enable or disable SSH.

   You can also make the selection persistent by using the ssh configuration file "enable_ssh.txt" which is located under "src/examples/test_app/cp1625_standalone_native/" folder. To enable the SSH server, set "enable_ssh" to 1. You can disable the SSH server by setting "enable_ssh" to 0. After PN Driver is started via "SERV_CP_startup()" IO-Base function, ssh configuration file is checked. The PN Driver application takes the required actions to start or stop the SSH server according to the saved selection. This configuration can be also updated according to the selection that you make via the menu item "Enable/Disable SSH".

5. You can also enable/disable SSH server through initialization scripts during bootup. First you must create a script and name it according to the naming convention rules. The files that begin with ‘S’ are run to start a system service. Therefore you must name your script according to this rule.

   The content of an example script file "S99testapp" may be as follows:

```
#!/bin/sh

#

# run test_app

#

HW_PATH="/root/my_config.xml"

case "$1" in

 start)

 printf "starts test_app: "

 /root/test_app '-a' '-f' $HW_PATH

 ;;

 stop)

 ;;

esac
```

   You must change the name of your application and the related command line arguments according to your own PN Driver example application.

   Also you must change the user rights of the script file by running the following command:

   – `chmod +x S99testapp`

   Then you must place it under the "[..]/buildroot-2017.02.9/board/cp1625/rootfs_overlay/etc/init.d" directory. This script will be called by the master script in ASCII sort order when you boot the system. According to the persistent data stored in "enable_ssh.txt", related enable/disable action will be taken by the PN Driver application after start up.

   After you enable SSH via menu item or using initialization script, you can transfer file between the host PC and the CP 1625Dev board with the following commands:

6. Use su command on host development PC.
7. Connect to the CP 1625Dev board with "ssh root@<CP1625_IP_address>" command.
8. Open the target folder where you want to locate the file on the CP 1625Dev board.
9. Copy the file from host development PC to the CP 1625Dev board by running the following command:

   – `scp siemens@HostPC_IP_address:/home/siemens/CD/pn_driver/src/examples/test_app/cp1625_standalone_native/test_app`

Quick start for CP1625 Host

6

6.1 Quick start for CP1625 Host

Overview

This section describes the steps required for commissioning PN Driver in a distributed system consisting of a Linux host and an embedded device which runs on a custom Linux operating system, built with Buildroot (https://buildroot.org/).

In CP1625 Host variant, it is possible to run PN Driver V2.2 distributed on Linux host PC and on CP 1625 PCIe card. With the help of the processing power of the host PC, you are able to profit both from the features of CP 1625 board and from the variety of applications on the host PC, at the same time.

The customer is responsible for the target system build with the help of the source code of PN Driver V2.2, application examples and the relevant documentation files.

Requirements

You need a Linux host development PC running on Debian 9.3 with Linux kernel V4.9 including RT patch. Please refer to the section Installing Debian (Page 25) for more details. You also need a CP 1625 or a CP 1625Dev board inserted into a free PCIe slot on the host machine. In case you use a CP 1625Dev board, you must configure it for PCIe usage. For more details, please refer to CP 1625 operating instructions in (https://support.industry.siemens.com/cs/ww/en/view/109756564). To keep updated for any Debian version upgrade, please refer to the PN Driver Readme document. Throughout this chapter, both CP 1625 and CP 1625Dev boards are referred to as CP 1625 board.

Note

PN Driver only supports little endian operating systems on host and CP 1625 board. PN Driver only works as a 32-bit application under 32-bit or 64-bit operating systems.

6.2 Changing local port range

If it is required to have IP communication parallel to PROFINET using the same network interface, local port range of the Linux system must be changed. This can be done as follows:

Copy the following add-on file from PN Driver CD to the root directory “/”.

"[..]/contrib/Linux_Host/siemens_linux_host_addon.tar.gz"

Then extract the file by running the following command.

- `sudo tar -xvf siemens_linux_host_addon.tar.gz`

Please refer to the section Using PROFINET interface for IP communication (Page 90) for the details regarding this limitation and IP communication over the PROFINET interface in general.

6.3 Installing Buildroot image

6.3.1 Downloading Buildroot package

Buildroot is required as Linux system generation tool, which is used to build the Linux image for the CP 1625 board. Suggested Buildroot version is buildroot-2017.02.9 which can be downloaded from (https://buildroot.org/downloads/).

Download “buildroot-2017.02.9.tar.gz” from the download page and extract it. To do so, run the following command in the directory where the downloaded package is located:

- `tar -xvf buildroot-2017.02.9.tar.gz`

Buildroot has dependencies on other software packages. For detailed information please refer to (https://buildroot.org/docs.html)

6.3.2 Configuring Buildroot

Buildroot has to be configured for PN Driver. To do so, copy the following addon file from PN Driver CD to the directory where the Buildroot package is located:

"[..]/contrib/Buildroot/siemens_cp1625_br2017.02.9_addon.tar.gz"

Then extract the file by running the following command:

- `tar -xvf siemens_cp1625_br2017.02.9_addon.tar.gz`

6.3.3 Building the image

Note that internet connection is required when running the commands listed below for the first time. In order to build the Linux Image run the following commands:

- `cd buildroot-2017.02.9`
- `make clean`
- `make cp1625_pci_defconfig`
- `make`

"uboot+linux.bin" will be created under "output/images" directory after the build process has been successfully completed.

6.3.4 Adding custom files to the Linux image

You need the firmware application to run on CP 1625 board. Therefore you need to transfer the application's binary to the board. For details about how to create the firmware application binary, please refer to the section Building PN Driver application (Page 80). Buildroot's overlay structure is used for integrating files in the Linux image. To do so, follow the procedure below.

Procedure

1. Copy the files that you want to be included in the image (e.g. firmware application binary) to the directory "[..]/buildroot-2017.02.9/board/cp1625/rootfs_overlay/root"
2. Run the following command in "[..]/buildroot-2017.02.9"
   - `make`
3. The image called "uboot+linux.bin" will be created in "[..]/buildroot-2017.02.9/output/images" directory. Use this image to run PN Driver application as described in the section Running PN Driver application (Page 81).

6.3.5 Configuring CP1625 firmware application as an auto boot application

To start firmware application automatically on CP 1625 board, edit the file "[..]/buildroot-2017.02.9/board/cp1625/rootfs_overlay/root/.profile" as follows:

1. Uncomment the last line
2. Change the name of the application to pnd_main. The last line of the file should be "/root/pnd_main"

6.4 Using the PN Device Driver

To use CP 1625 board under Linux, PN Driver requires the PN Device Driver (PNDevDrv). For details on how to use PNDevDrv under Debian 9.3 with the Linux kernel V4.9, see the section Using the PN Device Driver for the Linux variant (Page 29)

6.5 Building the PN Driver application

In CP1625 Host variant you must build both firmware and the example application.

6.5.1 Building the firmware application

Buildroot cross compilation toolchain is used to build applications for CP1625 Host variant. The toolchain is provided in the Buildroot package.

Procedure

To build the firmware application, follow the steps below:

1. Start a new shell and set the current directory to "[..]/buildroot-2017.02.9/". Then you should set the build environment to the Linux compiler. To do so, run the following command:
   - `source ./set_toolchain_env_buildroot_complete.sh`
2. If you want to use the same network interface for both IP communication of third party applications and PROFINET communication, build the firmware application by running "make" (or "make debug" if you want to build in debug mode) in "[..]/pn_driver/src/examples/shared/cp1625_fw_native/build", otherwise run "make" in "[..]/pn_driver/src/examples/shared/cp1625_fw/build". Make sure that the application binary "pnd_main" is created in the parent directory.

Please refer to the section Using PROFINET interface for IP communication (Page 90) for the details regarding IP communication over the PROFINET interface.

6.5.2 Building the example application

Procedure

To build the example application, follow the steps below:

1. Start a new shell.
2. Build PN Driver as a library by running "make" (or "make debug" if you want to build in debug mode) in "[..]/pn_driver/src/examples/shared/cp1625_host_native/linux_host/build" or "[..]/pn_driver/src/examples/shared/cp1625_host/linux_host/build" depending on the IP communication choice, which is mentioned above . Make sure that "libpndriver.a" has been created in the subdirectory "lib".
3. Build the application within the same shell and link the PN Driver library to the application by running "make" (or "make debug" if you want to build in debug mode) in "cp1625_host_native" or "cp1625_host" folder of the application depending on the IP communication choice, which is mentioned above, e.g., "[..]/pn_driver/src/examples/test_app/cp1625_host_native/build" or "[..]/pn_driver/src/examples/test_app/cp1625_host/build". Make sure that binary of the application has been created in the parent directory.

6.6 Running the PN Driver application

Requirements

Before you start the PN Driver application, copy firmware image “uboot+linux.bin”, shared object file “PnDev_DriverU32.so” and the prepared hardware configuration file to the same directory as the executable PN Driver example application. Make sure you have inserted the PNDevDrv kernel object as described in the section Loading the PN Device Driver (Page 30).

Procedure

To start PN Driver application, follow the steps below:

1. Change directory to where your application binary is located:
   - `cd [..]/pn_driver/src/examples/test_app/cp1625_host/`

   or

   - `cd [..]/pn_driver/src/examples/test_app/cp1625_host_native/`

   depending on the IP communication choice.
2. Run the application:
   - `sudo ./test_app`

When you run the application, a similar screen is displayed as shown in the figure below:

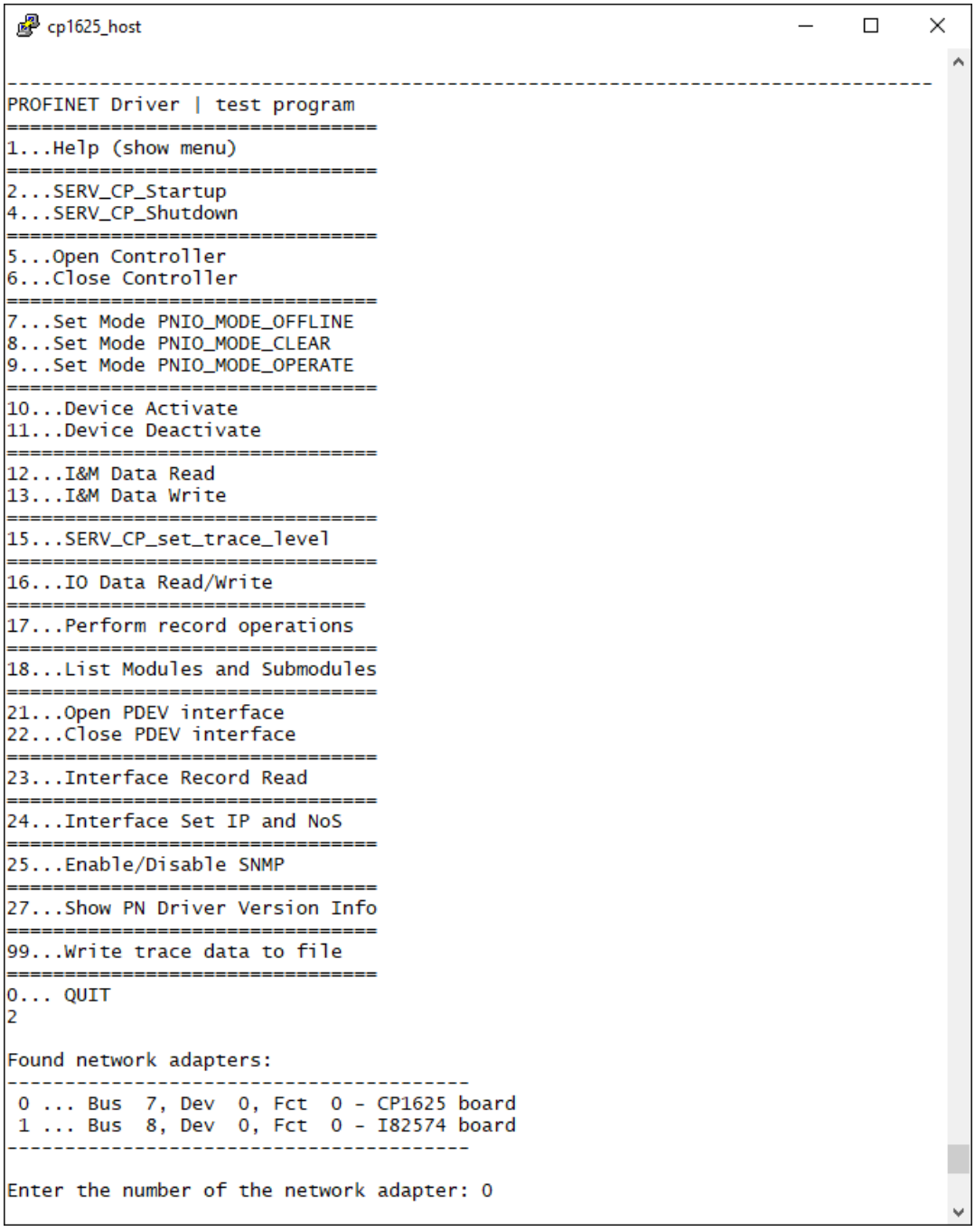


Figure 6-1 PN Driver application menu on CP1625 Host

Hardware configuration in engineering system 7

Overview

To generate the required XML configuration files, two alternatives are presented: TIA Portal and PNConfigLib.

Note

PN Driver does not perform a consistency check of hardware configuration files. Customers are responsible for the protection and security of hardware configuration.

7.1 Hardware configuration in the TIA Portal

Overview

This section describes the basic steps for generating the required XML configuration files with the TIA Portal

TIA Portal as of V16 is used as a basis. You also need the Hardware Support Package for PN Driver V2.2.

7.1.1 Installing the Hardware Support Package for PN Driver V2.2

Requirements

The project view of the TIA Portal is open.

Procedure

1. Click "Support Packages" under "Options".
   The "Detailed information" dialog opens.
2. You have the following options:
   - If the support package for PN Driver V2.2 (HSP0307) is already on your computer, you can add it to the "Local Support Packages" list by selecting "Add from file system".
   - If the support package for PN Driver V2.2 (HSP0307) is not yet on your computer, download it by clicking on the "Download from the Internet" link displayed at the dialog. Then you can add it to the "Local Support Packages" list from the file system.
3. Select the "HSP_V16_0307_001_Other_PNDriver_2.2.isp16" support package.

4. Click "Install".
5. Follow the instructions on the screen.

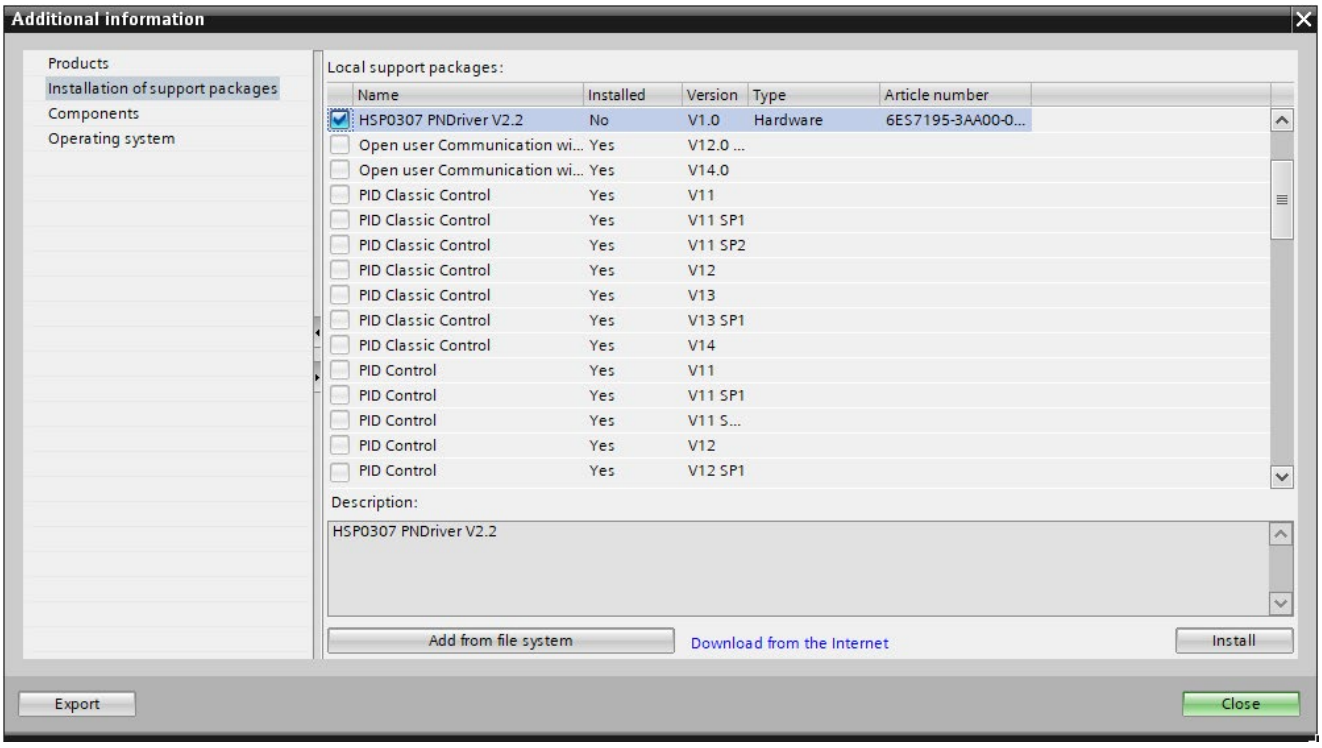


Figure 7-1 Installing support packages

Result: The version information "V2.2" is displayed for the existing PROFINET Driver record in the hardware catalog under "Communications Modules" – "PROFINET/Ethernet" – "PROFINET Driver" – "6ES7195-3AA00-0YA0".

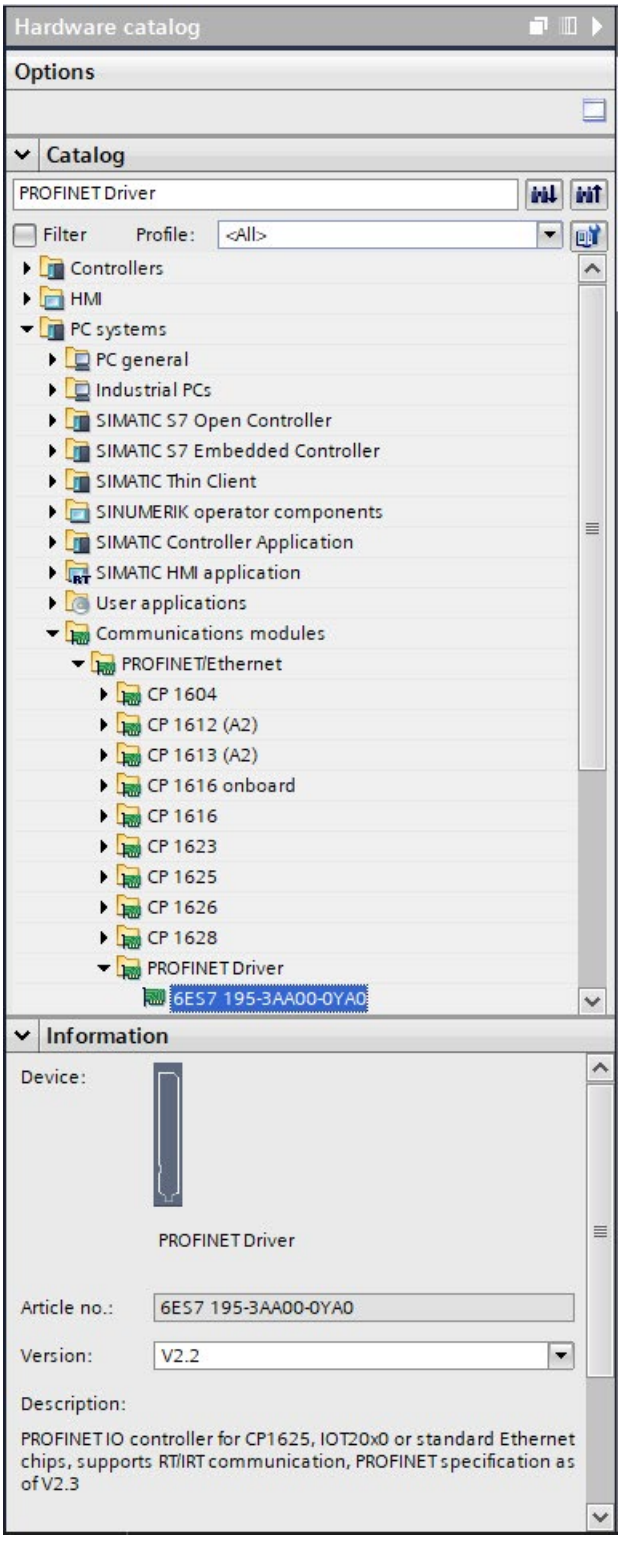


Figure 7-2 PN Driver in the hardware catalog

7.1.2 Generating an XML configuration file

Requirements

The network view of the TIA Portal is open.

Procedure

1. Add PN Driver to the project. You have the following options:
   - Drag PN Driver from the hardware catalog to the network view.
   - Copy and paste the PN Driver into the network view.
   - Double-click the PN Driver entry in the hardware catalog.
2. Select the "PN Driver Station" in the network view and switch to the device view.

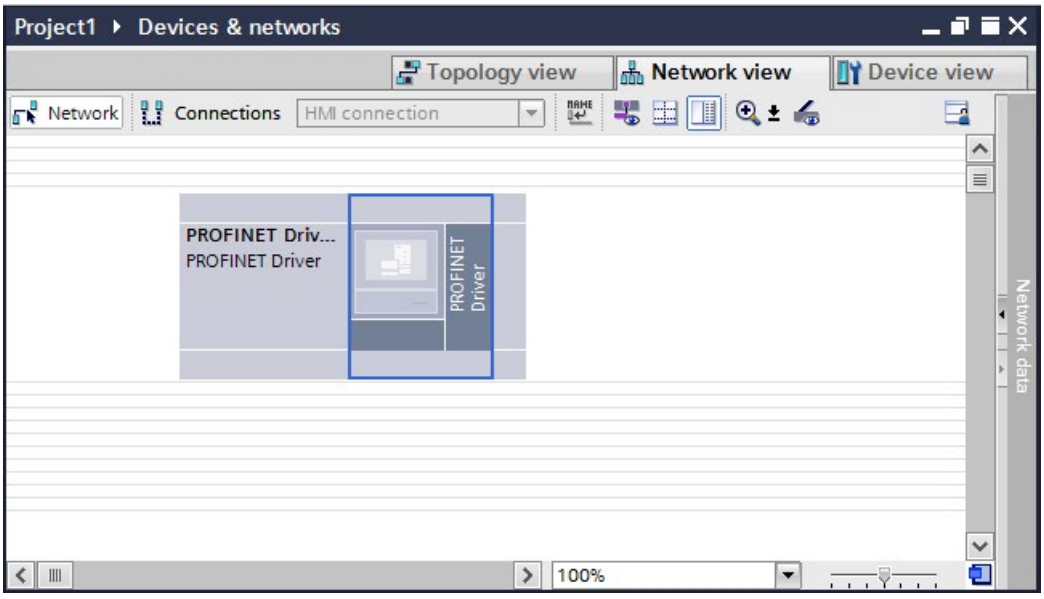


Figure 7-3 PN Driver station in the network view

3. Select the desired interface submodule (for Linux, Linux Native, Windows, etc.) in the hardware catalog under "Communications Modules" – "PROFINET/Ethernet" – "Interface Submodules".

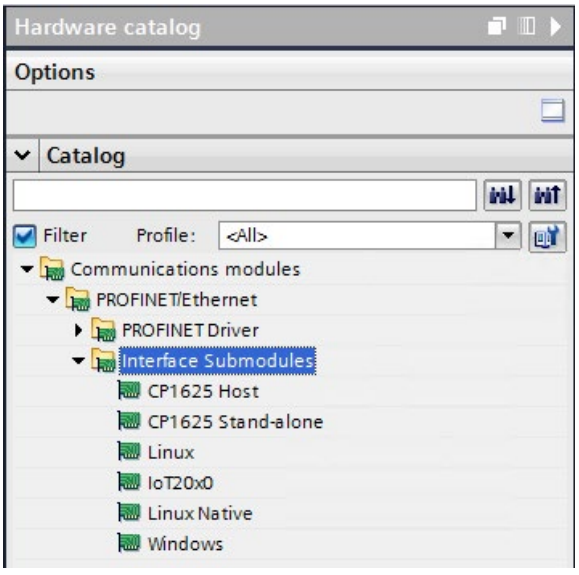


Figure 7-4 Interface submodule in the hardware catalog

4. Perform your hardware configuration as usual.
5. Compile the entire project.
   A consistency check is run when the configuration is compiled. Inconsistencies are displayed as messages in the Inspector window under "Info".
6. When your configuration is consistent, TIA Portal automatically generates the XML configuration file for the PN Driver. The storage path of the XML configuration file and other messages are displayed in the Inspector window under "Info".

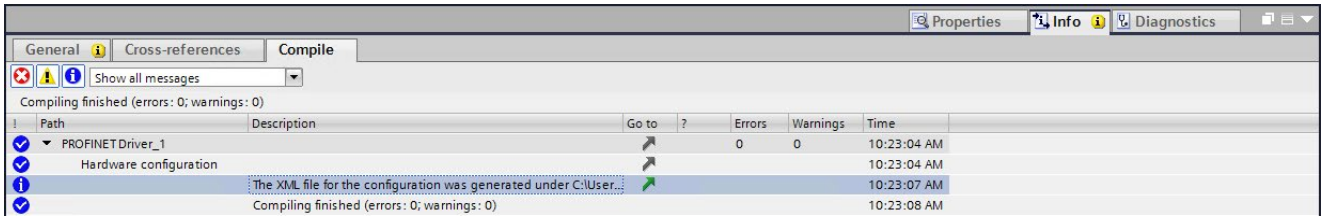


Figure 7-5 Storage path of the XML configuration file in the Inspector window

7. Double-click the message "Configuration XML file has been generated under…" to open the Windows Explorer.
8. Integrate the generated XML configuration file in your system.

7.2 Hardware configuration in PNConfigLib

Overview

PNConfigLib is a library that allows you to create PROFINET projects, to perform consistency checks to ensure their validation, and to compile these projects. It provides an API to allow you to call PNConfigLib in your own code.

This section explains PNConfigLib briefly. For more detailed information, please refer to PNConfigLib manuals inside PNConfigLib software package.

7.2.1 Generating an XML configuration file

From the user point of view, PNConfigLib has only one single entry point called “Run”. This method accepts the paths of the input files (Configuration, ListOfNodes and optional Topology) as inputs and returns a “PNConfigLibResult” object as an output.

```
public PnConfigLibResult Run(string configFilePath,
            string listOfNodesFilePath,
            string topologyFilePath = null)
```

Figure 7-6 PNConfigLib’s single entry point is the “Run” method

Requirements

The input files of PNConfigLib consist of two required and one optional XML files: ListOfNodes.xml (mandatory), Configuration.xml (mandatory), Topology.xml (optional).

ListOfNodes.xml: A "node" represents an IO device or an IO controller, which is configured in the "Configuration" file. The user should present here all the nodes with their identifiers. IO devices are described with GSDML files provided by the device manufacturer. The content of the GSDML consists of configuration information, parameters, modules, diagnostic, alarms, vendor and device identification. PNConfigLib imports the GSDML files indicated in the ListOfNodes.

Configuration.xml: The nodes defined in the ListOfNodes file are configured in the Configuration file. The IO controller and IO device settings, their modules and submodules, subnets, sync domains etc. are defined here.

Topology.xml: The port interconnections are defined in the optional Topology file. In this file, the user may provide information on how devices are connected to each other.

You can find examples for ListOfNodes.xml, Configuration.xml and Topology.xml in the PNConfigLib manual.

Procedure

1. Use PNConfigLib to read the provided input files.
2. Use PNConfigLib to compile the project. If there is no error, an XML based configuration is created for each IO controller within the project.
3. Integrate the generated XML configuration file in your system.

Additional Notes

In order to generate an XML configuration file, you can use PnConfigLib.dll to create your own application or you can use PNConfigLibRunner.exe.

With PNConfigLibRunner.exe you can call PnConfigLib.dll from command prompt with the following parameters:

PNConfigLibRunner.exe –c “Configuration.xml” –l “ListOfNodes.xml” –t “Topology.xml” –o “OutputFolderPath”

- -c, --configuration Required. Path to Configuration XML file
- -l, --listofnodes Required. Path to ListOfNodes XML file
- -t, --topology Optional. Path to Topology XML file
- -o, --output Optional. Output folder
- --help Displays help screen
- --version Displays version information

For more details, refer to the PNConfigLib manual.

Using PROFINET interface for IP communication 8

Overview

It is possible to use the same interface both for PROFINET communication and for IP communication of other applications when PN Driver is running on Linux.

The following PN Driver variants support this feature:

1. Linux Native
2. CP1625 Stand-alone
3. CP1625 Host

How to activate this feature is described under the platform specific chapters, e.g., Quick start for Linux (Page 24) for the Linux Native variant.

Please refer to "IO-Base User Programming Interface" document for the details regarding IO-Base functions that are mentioned in this chapter.

You may refer to the function manual "PROFINET with STEP 7 V16" (https://support.industry.siemens.com/cs/ww/en/view/49948856) for the settings which must be done in the engineering system such as IP assignment to the controller.

8.1 How to use PROFINET interface for socket connections

PN Driver must be running in order to enable the IP communication of other applications in parallel. This is achieved by calling "SERV_CP_startup()" IO-Base function.

Once "SERV_CP_startup()" returns without any error, a virtual network interface emerges within Linux system. Controller application can continue with its internal tasks. Virtual network interface is not affected by other calls to IO-Base unless PN Driver is shutdown via "SERV_CP_shutdown()" IO-Base function. Once SERV_CP_shutdown() returns without any error, the virtual network interface disappears.

The virtual network interface has the name of "pnio1" and it is accessible for the applications running on the same host as PN Driver. For example, the list of the available network interfaces that is displayed by Linux "ip" utility would include "pnio1" interface as well as other interfaces. This could be achieved by the following command.

- `ip link show`

This means that you can also create your own socket application or add the functionality to the controller application and use the PROFINET interface as the network interface. Please note that having another interface in the same subnet as "pnio1" on the same host might cause routing problems which could result in communication failure.

8.2 Limitations

Using the PROFINET interface for socket connections is possible with the listed limitations below.

8.2.1 Transport protocols

Only IP based protocols (e.g. TCP and UDP) are supported. Non-IP based protocols are not supported.

8.2.2 Local port range

Dynamic local port range of Linux must be shifted. The new range must be same as IANA dynamic port range that is specified as 49152 – 65535. Some of the Linux distributions including the one that is used in PN Driver Linux based variants have a different default dynamic local port range.

There are two reasons for this adjustment.

- PN Driver UDP packet filtering mechanism will drop the inbound UDP frames with a destination port number outside the IANA dynamic port range.

  In order to prevent PROFINET UDP frames and/or the UDP frames which belong to a 3rd party application from being dropped at the reception, the local dynamic port range of Linux must be shifted.

- According to the PROFINET specification, IANA dynamic port range must be used for PROFINET UDP communication. PN Driver's PROFINET stack does not bind its UDP socket to a specific port. Instead, it assumes that the dynamic local port range is already the same as the IANA dynamic port range.

With this adjustment, the port range is shifted for both TCP and UDP communication using any network interface. However, in most of the cases, the port number that your client socket is bound to is not critical for the server side. In addition, it is still possible to bind the client TCP sockets to specific ports explicitly if it is necessary.

One example as a problematic side effect of the UDP packet filtering is UDP based DNS requests which are sent through "pnio1" interface. If the port range is not adjusted appropriately as explained above, DNS requests sent from the PN-Interface will arrive the server but the response might be dropped by UDP packet filtering. It is also possible that even if the port range is adjusted, the response may still be dropped since some of the DNS stub resolvers use specifically UDP port 53 regardless of the configured port range. Please make sure that the local DNS stub resolver in your system does not have this behavior.

Changing port range is already configured within Buildroot settings for CP1625 Stand-alone variant. There is no need to take extra action once you have created the image as explained in the section Installing Buildroot Image (Page 64). However, it must be configured manually for Linux Native and CP1625 Host variants as described below.

8.2.2.1 Changing local port range for Linux Native and CP1625 Host variants

A configuration file is provided in the PN Driver CD to configure the port range at each boot as from 49152 to 60999.

Copy the following add-on file from PN Driver CD to the root directory "/".

"[..]/contrib/Linux_Host/siemens_linux_host_addon.tar.gz"

Then extract the file by running the following command.

- `sudo tar -xvf siemens_linux_host_addon.tar.gz`

8.2.3 Default Gateway and IP assignment

8.2.3.1 Default Gateway assignment via external tools and DHCP

Default gateway address must not be set via external tools or DHCP client. It must be set only by PN Driver to be PROFINET compliant and the supported ways are the ones described in Supported methods for IP and Default Gateway assignment (Page 94).

All external settings for default gateway assignment must be avoided.

The necessary configuration must be done to prevent DHCP client from assigning a default gateway to network interfaces which should use DHCP even if the interface is not used for PROFINET. This could be done via network manager on Debian by selecting "Use this connection only for resources on its network" option for the active connection profile of each interface. The following screenshot illustrates the setting.

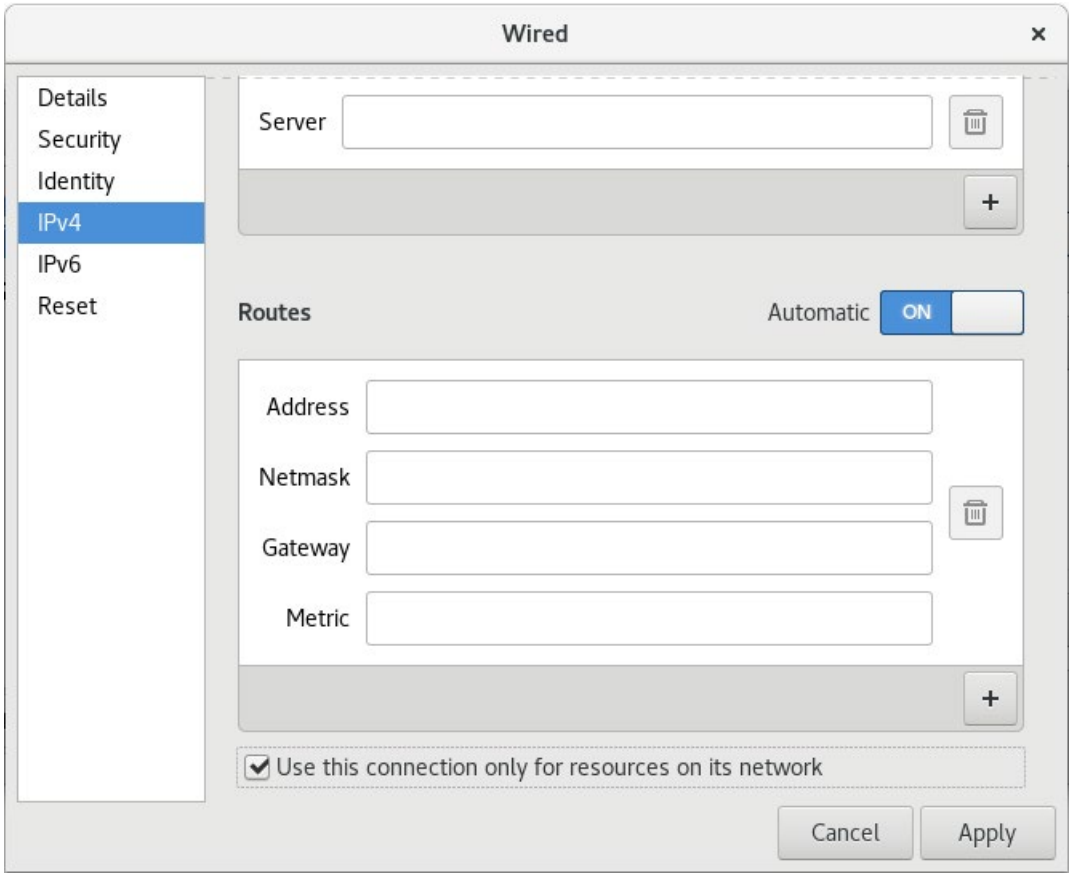


Figure 8-1 The setting for default gateway

You can also configure this setting from terminal. To do so, follow the steps below:

1. List all the interfaces managed by the network manager, and see which connection profile they are using with the following command

   – `nmcli`

   The output of this command will be a list of interfaces in the following format:

   `<interface name>: connected to <connection profile name>`

2. For each interface, run the following command:

   – `sudo nmcli connection modify "<connection profile name>" ipv4.never-default true`

8.2.3.2 Supported methods for IP and Default Gateway assignment

Changing IP and gateway address via OS tools such as "ip" would cause undefined behavior. Only the methods which are described in this section must be used for this purpose.

Please note that, the assigned default gateway will be valid for the entire Linux IP stack.

The IP assignment to "pnio1" interface and default gateway assignment can be done by one of the following methods as long as the gateway is in the same subnet as "pnio1" interface and thus as the PROFINET system or there is no default gateway.

1. IP address assignment to the PROFINET interface of the controller in the hardware configuration XML file. This could be done via the engineering system.
2. The engineering system or a tool with DCP support such as PRONETA (PROFINET Network Analyzer - Configuration and Diagnostic Tool), which can be downloaded from (https://support.industry.siemens.com/cs/document/67460624/proneta-basic-3-0-0-4-commissioning-and-diagnostics-tool-for-profinet?dti=0&lc=en-WW)), might provide this functionality.
3. Setting the IP address within the controller application calling "PNIO_interface_set_ip_and_nos()" IO-Base function.

Setting the IP suite at runtime with option 2 or 3 might be necessary for the use cases such as "Multiple use IO systems".

In order to use the options 2 and 3, the hardware configuration must allow adapting the IP address of the controller outside the engineering project. Please see Figure 8-3 for the relevant setting in TIA portal.

You may want to use a default gateway which is not in the same subnet as "pnio1" interface to access another network such as company network or Internet. The following image illustrates this scenario.

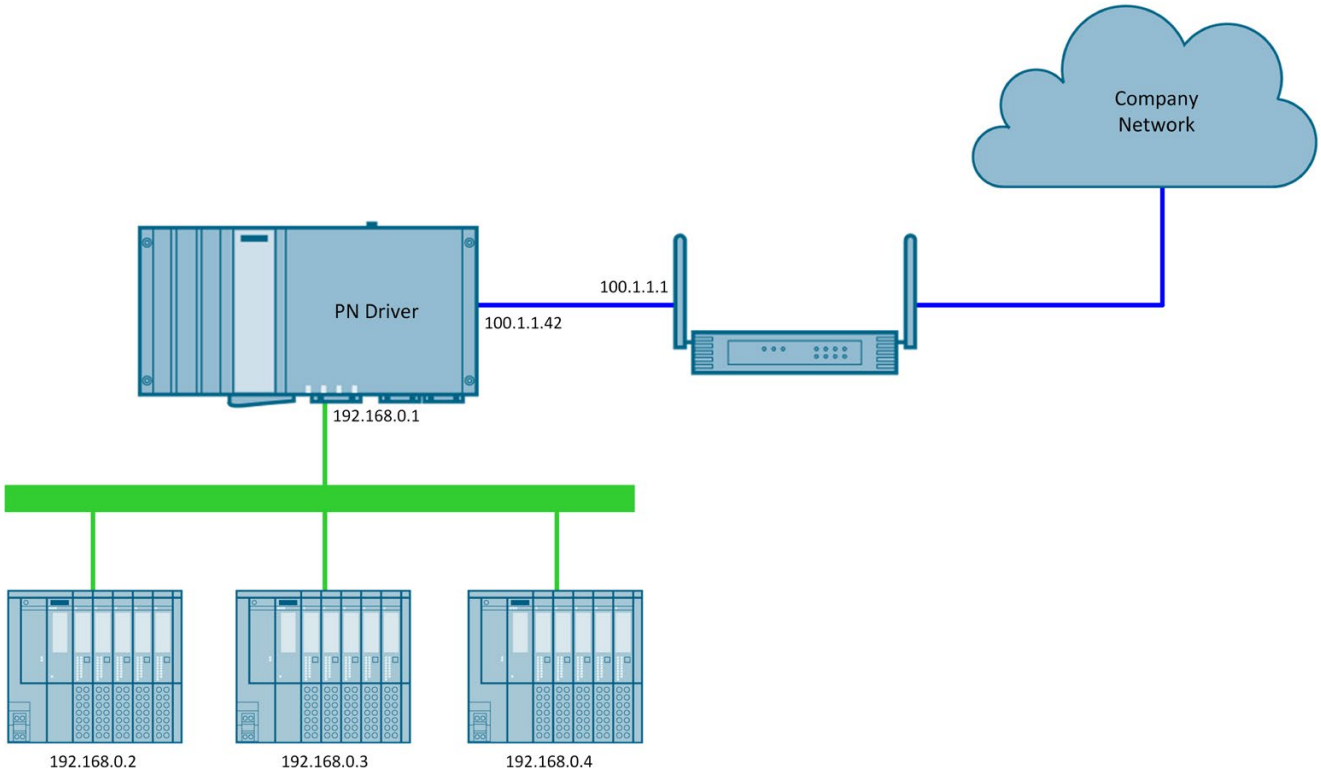


Figure 8-2 Using default gateway for company network access

In this case, the IP assignment to "pnio1" interface and to the IO devices must be done via DCP and the hardware configuration must be specified, that the IP address for both the controller and the IO devices may be adapted locally. The following images illustrate the necessary settings in TIA Portal for the controller and the IO device respectively.

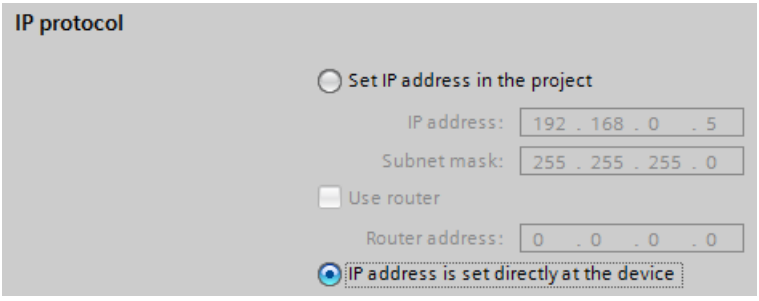


Figure 8-3 Setting for the controller

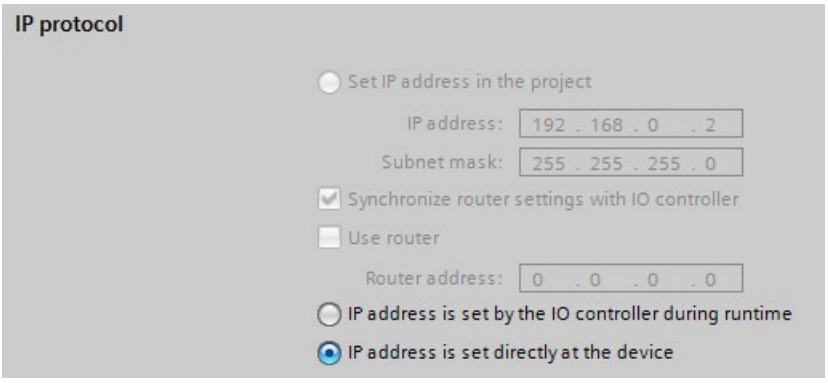


Figure 8-4 Setting for the IO device

Default gateway to "pnio1" interface can be set together with the IP address. There must not be any default gateway assigned to the IO devices so, only IP address assignment must be done.

Following images illustrate the assignment via PRONETA for the controller and one of the IO devices in Figure 8-2 respectively.

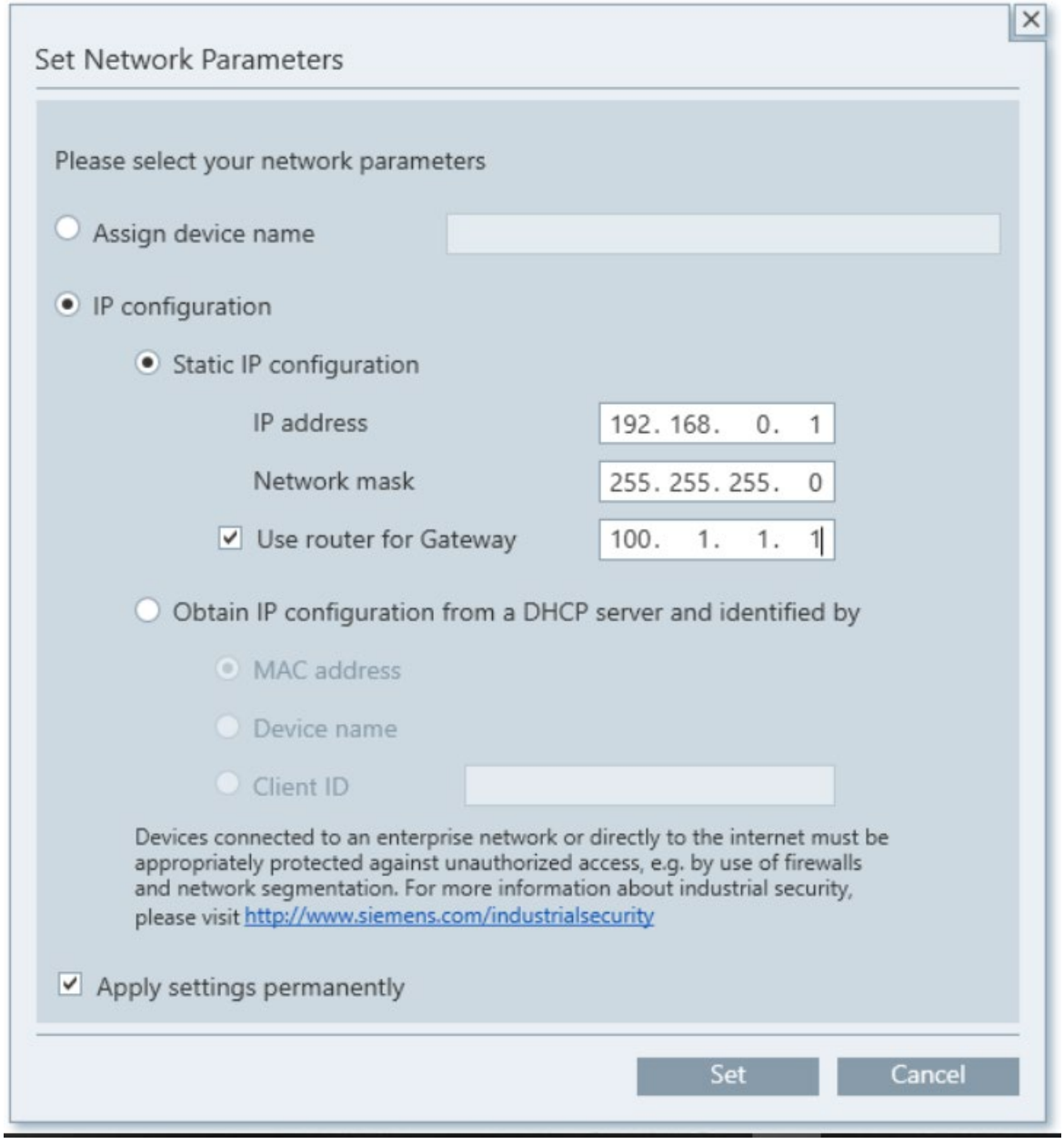


Figure 8-5 IP & Gateway assignment to controller via PRONETA

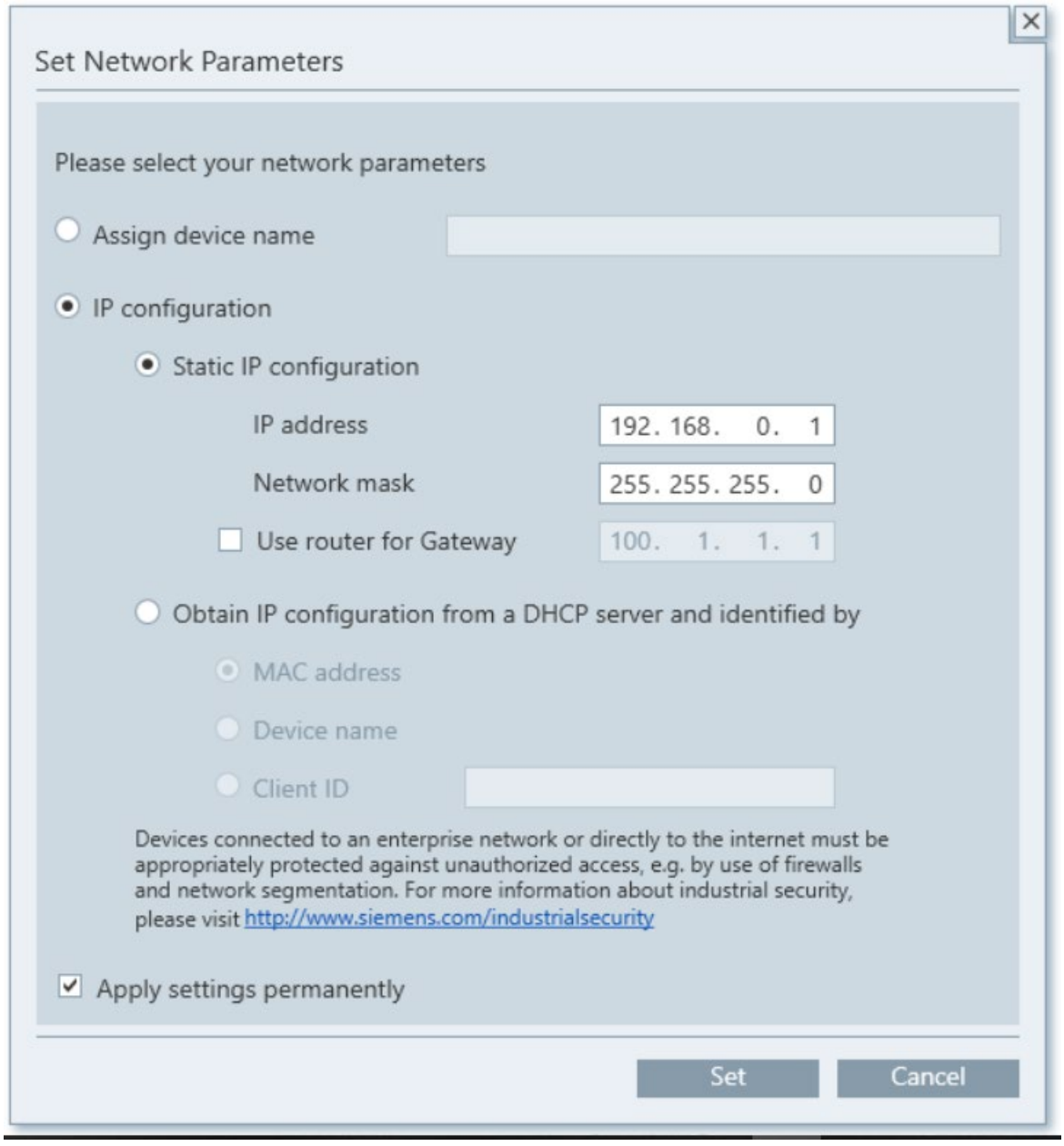


Figure 8-6 IP assignment to IO device via PRONETA

It might be necessary to enable "Multiple use IO system" feature on the IO system first to be able to make the setting for the local IP assignment on the IO device.

Since the default gateway is outside the subnet of the PROFINET interface "pnio1" with this change, standard gateway of the controller becomes inactive. This situation is indicated with PNIO_ALARM_MULTIPLE_INTERFACE alarm and "Application Ready" indication at runtime.

Please refer to IO-Base user programming interface document for the details of the alarm and the indication. Please be aware that, changing IP and gateway address at runtime may cause failure of the existing socket connections in the system.

8.2.3.3 Disabling DHCP IP assignment for Linux Native and CP1625 Host variants

IP assignment via DHCP is not allowed. It is already not possible to assign an IP address to "pnio1" interface of CP1625 Stand-alone variant via DHCP; however, it must be disabled manually for "pnio1" interface of Linux Native and CP1625 Host variants.

Adding the following line to "/etc/network/interfaces" file would disable DHCP client for "pnio1" interface:

- `iface pnio1 inet manual`

DHCP IP assignment for the interface used for the Linux Native variant is also not allowed and must be disabled. This can be done in the same way, but using the name of the interface instead of pnio1. For example if eth0 is used for the Linux Native variant, the following line would disable DHCP client for eth0 interface:

- `iface eth0 inet manual`

8.2.4 Bandwidth limitation

Data infeed from "pnio1" interface for data types other than cyclic real-time data is limited to 3k octets per millisecond to avoid critical network load and share the communication bandwidth fairly between devices. This includes e.g. PROFINET alarms, PROFINET records, UDP and TCP frames.

We recommend to limit the sending bandwidth as 3k octets per millisecond maximum for other non-real time communication nodes in your network as well to prevent network problems due to high load.

8.2.5 SNMP

- It is not possible to use an SNMP agent other than the internal SNMP agent of PN Driver at the standard SNMP ports.
- The SNMP agent provides interface group content for only "pnio1" interface. The other interfaces in the system and relevant data is not provided.
- SNMP counter values of OSI model layer 3 and the above layers correspond to the entire IP stack and thus all interfaces. Counter values which are that of layer 2 and layer 1 correspond to only "pnio1" interface.
- There is no interface to the user to define additional MIB objects, only RFC1213 MIB-II is supported.

8.2.6 Firewall

You may need to check the firewall settings of your system to allow PROFINET communication via "pnio1" interface since PROFINET UDP traffic may be subject to the firewall rules.

With improper firewall settings, the PROFINET UDP frames which are sent from the IO devices to the controller can be blocked by the firewall. This would cause connection problems between the controller and the IO devices.

8.3 Network planning

Network load might become an issue when PROFINET and non-real time communication are used together. Siemens offers a tool, SINETPLAN, to calculate and simulate network load for PROFINET networks.

With SINETPLAN, you can optimize the use of network bandwidth and avoid problems that could arise during the commissioning and the production. For more information, please refer to (https://new.siemens.com/global/en/products/automation/industrial-communication/profinet/portfolio/sinetplan.html)

Application examples

9

Overview

The PN Driver software package includes a set of application examples which offer a quick start into programming your own application. These examples are very useful and helpful for you to get familiar with PN Driver. These examples can be listed as follows:

- Test application
- Multiple use IO systems
- PNIO diagnostics
- Configuration control of IO systems
- Option handling
- Receiving alarms
- Isochronous mode
- Isochronous calculation

An XML file which includes the hardware configuration is stored for each application example in the respective folder.

9.1 Test application

The test application (“test_app") uses the hardware configuration as shown in the figure below.

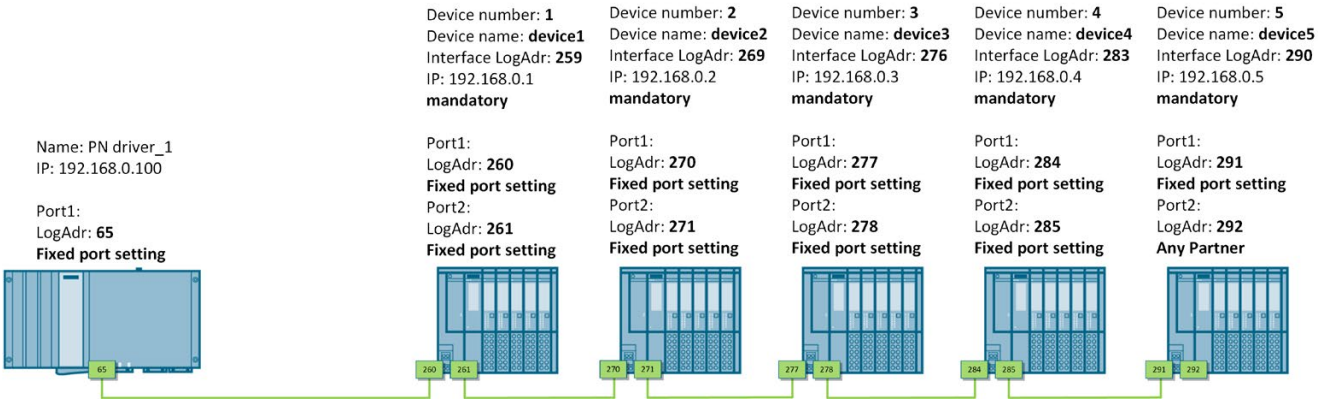


Figure 9-1 Hardware configuration of the test application

The table below lists the included hardware components.

Table 9- 1 Hardware components of the test application

Component	Number	Article number	Firmware
PN Driver	1	6ES7195-3AA00-0YA0	V2.2
SIMATIC ET 200SP, PROFINET interface module IM 155-6 PN ST	5	6ES7155-6AA01-0BN0	V4.1
Light-colored BaseUnit (BU...D)	5	6ES7193-6BP20-0DA0	
Dark-colored BaseUnit (BU...B)	5	6ES7193-6BP20-0BA0	
DI 8x24VDC ST	5	6ES7131-6BF01-0BA0	V0.0
DQ 8x24VDC/0.5A ST	5	6ES7132-6BF01-0BA0	V0.0

9.1.1 Startup options

The test application can be executed with various startup options in terms of reading the configuration file. Refer to the table below for these options and their usage.

Table 9- 2 Startup options of the test application

./test_app	Starts with default configuration under PNDriverBase_TestApp folder.
./test_app –f <file_path>	Starts with the configuration file given by <file_path>.
./test_app –d <directory_path>	Lists all the xml files in the directory given by <directory_path>. User chooses one of them to start test_app.

9.1.2 Menu Items

Menu items in the test application example are explained as follows:

1. **Help**
   Shows the menu again.
2. **SERV_CP_Startup**
   Starts all the internal tasks and configures PNIO stack according to the configuration. It must be called before “Open Controller””.
4. **SERV_CP_Shutdown**
   This must be called as the last function when exiting the user program. After the return of the function, all internal threads have been exited and the entire local memory has been released again.
5. **Open Controller**
   Registers PN Driver as an IO controller with the IO-Base functions. It also registers callback functions for setting the mode, device activation/deactivation, diagnostics requests and IO system reconfiguration.
6. **Close Controller**
   This function deregisters PN Driver as an IO Controller. All registered callback functions are also deregistered with this function.
7. **Set Mode PNIO_MODE_OFFLINE**
   With this function, you change the mode of the controller to OFFLINE.
8. **Set Mode PNIO_MODE_CLEAR**
   With this function, you change the mode of the controller to CLEAR.
9. **Set Mode PNIO_MODE_OPERATE**
   With this function, you change the mode of the controller to OPERATE.
10. **Device Activate**
    This sets up a connection between your IO controller (PN Driver) and an IO device. You need to specify which IO device you want to activate by entering a logical address associated with it. Each IO device can have several modules and therefore also can have several addresses. It is enough to address just one of these.
11. **Device Deactivate**
    This disconnects an IO device from PN Driver. You need to specify which IO device you want to deactivate by entering a logical address associated with it.
12. **I&M Data Read**
    With this function, PN Driver sends a read identification and maintenance data record job to a module of an IO device. You should enter the logical address of the module for which the read data record job is intended.
13. **I&M Data Write**
    With this function, PN Driver sends a write identification and maintenance data record job to a module of an IO device. You should enter the logical address of the module for which the write data record job is intended.
15. **SERV_CP_set_trace_level**
    With this function, the trace level of individual internal components can be changed during operation. The function can be called at any time after the "SERV_CP_startup()" function. When you call this function, you will first be asked the component for which you want to change trace level, and then the level you want to set.

16. **IO Data Read/Write**
    After selecting this menu item, you call either PNIO_data_read() or PNIO_data_write(). These functions read input data from and write IO data to the process image, respectively.
17. **Perform record operations**
    You can perform record read and record write operations following the submenu that appears when you choose this menu item
18. **List Modules and Submodules**
    This lists the details of modules and submodules that are connected to PN Driver in the hardware configuration.
21. **Open PDEV interface**
    This opens the local Ethernet interface to use its functions, and registers interface callback functions for setting IP suite and name of station and reading remanent file.
22. **Close PDEV Interface**
    This closes local Ethernet interface.
23. **Interface Record Read**
    You can perform record read from the local Ethernet interface using this menu.
24. **Interface Set IP and NoS**
    You can change the "IP Suite" and/or the station name of the local Ethernet interface by using the submodule that appears when you choose this menu item.
25. **Enable/Disable SNMP**
    SNMP functionality can be activated or deactivated by using the submodule that appears when you choose this menu item.
26. **Enable/Disable SSH**
    SSH server can be activated or deactivated by using the submodule that appears when you choose this menu item. This menu item appears when PN Driver runs under CP 1625Dev board "PN Driver CP1625 Stand-alone" and if you are using the same network interface for both PROFINET and other IP based network communication.
27. **Show PN Driver Version Info**
    This menu item displays the version of the actual used PN Driver version.
99. **Write trace data to file**
    This menu item appears when circular trace mechanism is activated (#define PNDTEST_USE_CIRCLE_TRACE) in the application. With this method, trace data will be held in RAM until a FATAL/exception occurs. Also, you can write this trace data to a file by using this menu item before exiting the user program.
0. **QUIT**
    You can quit the application by entering 0.

9.2 Multiple use IO systems

The application example shows how to use the feature "multiple use IO systems".

The application example for "multiple use IO systems" ("multiple_use_io_systems") uses the hardware configuration as shown in the figure below. For the details of hardware components such as article number and firmware version please refer to the section Test application (Page 102)

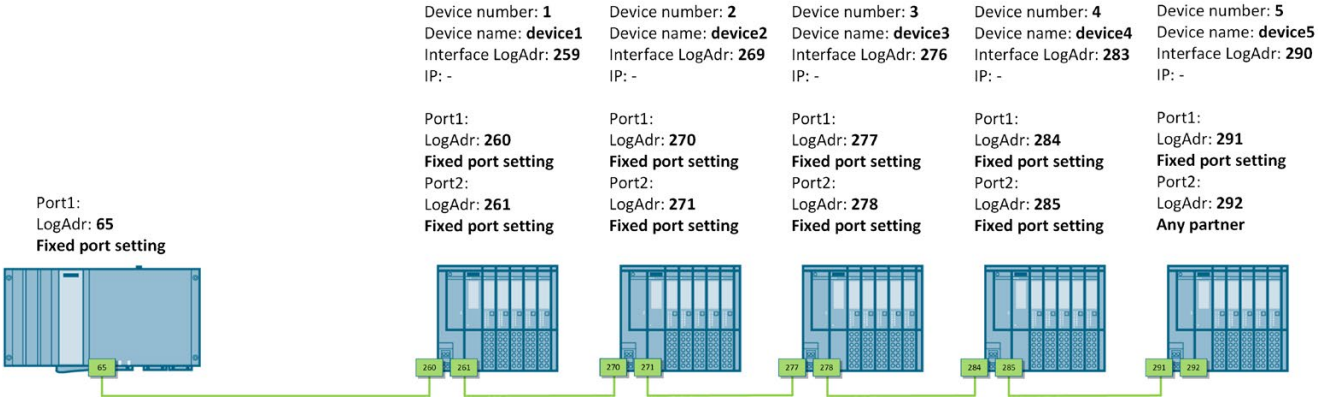


Figure 9-2 Hardware configuration of multiple use IO systems and PNIO diagnostics

9.2.1 Menu Items

Menu items in the test application example are explained as follows:

1. **Help**
   Shows the menu again.
2. **Startup**
   Starts all the internal tasks and configures PNIO stack according to the configuration. Opens the local Ethernet interface and registers callback functions for setting IP suite and name of station and reading remanent file. Registers PN Driver as an IO controller and also changes the mode of the controller to OPERATE.
3. **Start Tailoring Process**

   With this function you can start the tailoring process and set the IP suite and name of station of PN Driver. The process of setting the IP suite and name of station of PN Driver leads the tailoring process of each IO device.

   With this application you can also store IP suite and name of station values remanently in a file named "rema.xml". With the next start of the application, when you startup, IP suite and name of station values will be taken from "rema.xml" file.

0. **QUIT**
   You can quit the application by entering 0.

9.3 PNIO diagnostics

The application shows how to gather configuration and diagnosis information of IO devices. Besides the device information, information about network parameters of the local IO interface is also provided. The application example for PNIO Diagnostics ("pnio_ctrl_diag") uses the same hardware configuration and the same hardware components as the test application. For details, see the section Test application (Page 102)

9.3.1 Menu Items

Menu items in the test application example are explained as follows:

1. **Help**
   Shows the menu again.
2. **Startup**
   Starts all the internal tasks and configures PNIO stack according to the configuration. Opens the local Ethernet interface and registers callback functions for setting IP suite and name of station and reading remanent file. Registers PN Driver as an IO controller and also changes the mode of the controller to OPERATE.
3. **Show Submodule List**
   Displays a list of all configured submodules that are connected to PN Driver in the hardware configuration.
4. **Show Device Diagnostic**
   Displays the status information of IO device for which the logical address is given.
5. **Show Device Configuration and Diagnostics**
   With this function, diagnosis information and all modules/submodules of each IO device are displayed
6. **Show Address Info**
   Displays the network parameters of the local IO interface.
7. **Set Device I&M Values**
   With this function, you can set the identification and maintenance (I&M) data record values for a module of an IO device. You should enter the logical address of the module. Then you should enter Function and Location tag (I&M1), build in date tag (I&M2), descriptor tag (I&M3) and signature tag (I&M4).

0. **QUIT**
   You can quit the application by entering 0.

9.4 Configuration control for IO systems

The application example shows how to use the feature "configuration control of IO systems". At startup, one of the two possible configurations can be chosen. One hardware configuration for optional IO devices and one hardware configuration for flexible topology can be found in the folder for this example.

This application example uses the same hardware configuration and the same hardware components as the test application. For the details of hardware components such as article number and firmware version please refer to the section Test application (Page 102).

9.4.1 Optional IO devices

The hardware configuration used for optional IO devices is shown in the figure below.

Note that the IO device with the device name "device1" is not physically included in the hardware configuration. The created application example is used to inform PN Driver with "PNIO_reconfig_iosystem" about this state to commission the plant.

Additionally, in the hardware configuration "device2" is projected as "Optional IO-Device" and "device3", "device4" and "device5" are projected as mandatory IO device. Each port has a fixed partner port. Also note that, even if "device1" is not physically included in the hardware configuration, "device1" is projected as "Optional IO-Device" in the engineering configuration.

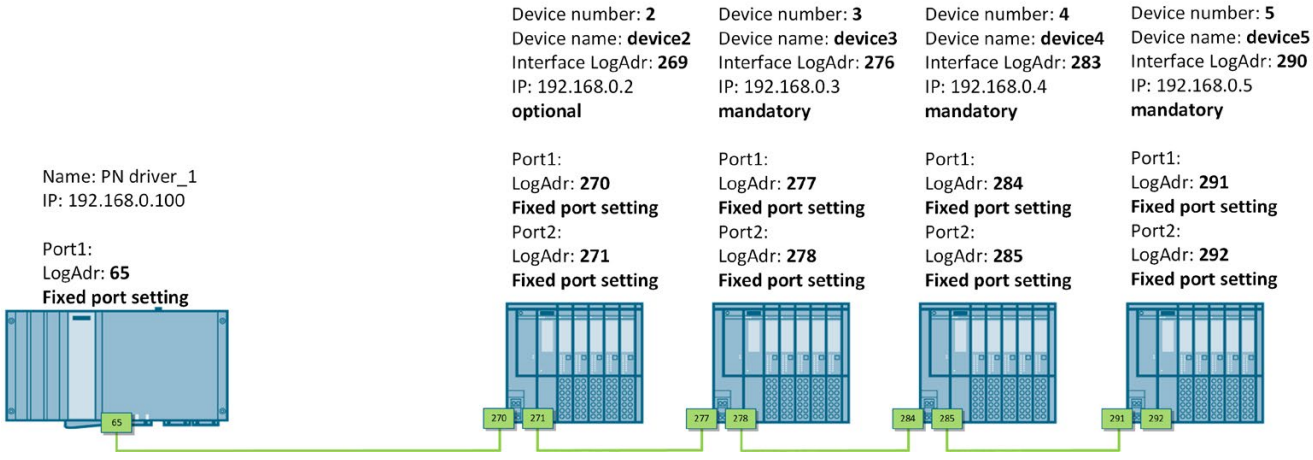


Figure 9-3 Physical hardware configuration of optional IO devices

9.4.2 Flexible topology

The hardware configuration used for flexible topology is shown in the figure below.

Note that the order of the IO devices is changed in the physical configuration. The application example is created to inform PN Driver about this state to commission the plant.

Additionally, in the hardware configuration the port settings of each IO device and IO controller are set to "Set Partner by user program".

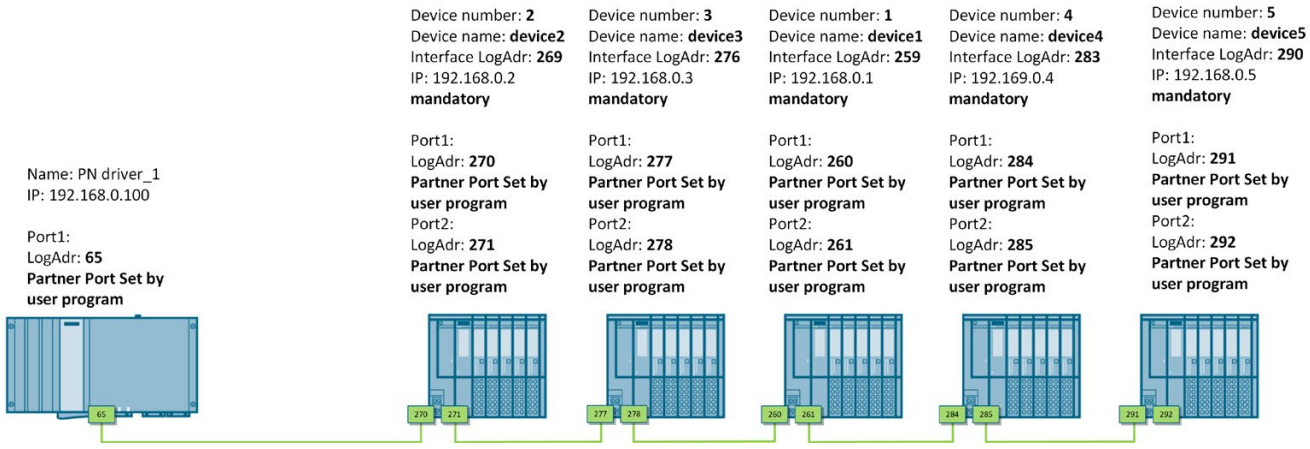


Figure 9-4 Physical hardware configuration of flexible topology

9.4.3 Menu Items

Menu items in the test application example are explained as follows:

1. **Help**
   Shows the menu again.
2. **Startup**
   Starts all the internal tasks and configures PNIO stack according to the chosen configuration (1 for optional IO devices configuration and 2 for flexible topology). Opens the local Ethernet interface. Registers PN Driver as an IO controller and callback function for IO system reconfiguration. Also changes the mode of the controller to OPERATE.
4. **Deactivate All IO Devices**
   With this function you can avoid diagnostic messages on the IO devices by deactivating all IO devices including the mandatory ones.
5. **Execute Tailoring Process**
   Tailoring process is done with the parameters specified in "DeviceList" and "PortInterconnectionList" and then the activation of all IO devices which are mandatory or optional and listed in "DeviceList" is triggered.

   "DeviceList" contains the logical addresses of the optional and physically present IO devices. For example; for optional IO devices configuration "DeviceList" has only one entry for "device2" which is optional and physically present. For flexible topology configuration "DeviceList" does not have any entry because all of the IO devices are projected as mandatory.

   "PortInterconnectionList" contains the logical addresses of the port interconnections. For example; for optional IO devices configuration "PortInterconnectionList" is empty because each port has a fixed partner port. For flexible topology configuration "PortInterconnectionList" contains 5 entries ([65,270], [271, 277], [278, 260], [261, 284], [285, 291]).
0. **QUIT**
   You can quit the application by entering 0.

9.5 Option handling

The application example shows how to use the feature "configuration control". You can activate a variant by writing a "control data record" to the IO device which describes the expected configuration and then read a "feedback data record" from IO device which describes the actual configuration of the IO device. Description of the variants and configured hardware configuration of ET 200SP are given in the table below.

Table 9- 3 Description of the variants

			Slot Numbers of ET 200SP				
			1	**2**	**3**	**4**	**5**
Projected configuration of the SIMATIC ET 200SP		Modules:	M1	M2	M3	M4	SM
		BaseUnits:	BU_D	BU_B	BU_B	BU_B	
Real configuration of the SIMATIC ET 200SP	Variant 2	Modules:	M1	M4	SM	---	---
		BaseUnits:	BU_D	BU_B			
	Variant 3	Modules:	M1	M3	M4	M2	SM
		BaseUnits:	BU_D	BU_B	BU_B	BU_B	
	Variant 4	with Modules:	M1	M2	M3	M4	SM
		with BU cover:	M1	BC	M3	M4	
		BaseUnits:	BU_D	BU_B	BU_B	BU_B	
	Variant 5	Modules:	M1	M2	M3	M4	SM
		BaseUnits:	BU_D	BU_B	BU_D	BU_B	
	Variant 6	with Modules:	M3	M4	M1	SM	---
		With BU cover:	M3	BC	M1		
		BaseUnits:	BU_D	BU_B	BU_D		

The application example for option handling ("option_handling") uses the hardware configuration as shown in the figure below.

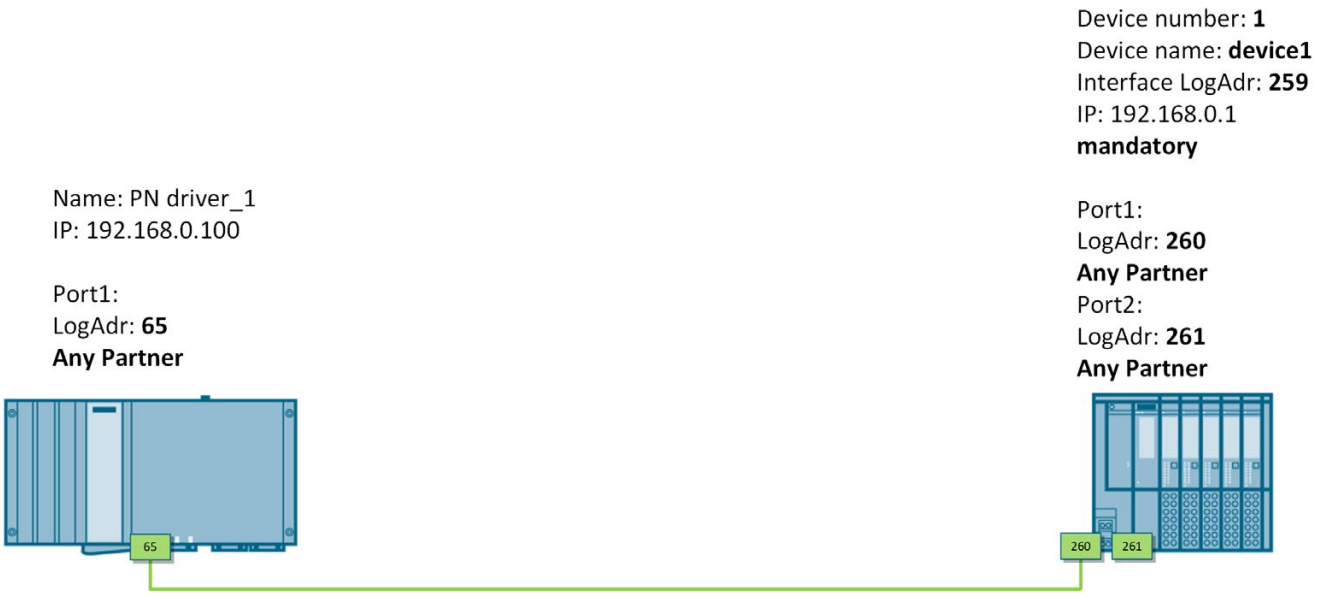


Figure 9-5 Physical hardware configuration of option handling

The required components are listed in the table below.

Table 9- 4 Hardware components of the configuration control for IO devices

Component	Name	Number	Article number	Firmware Version
PN Driver		1	6ES7195-3AA00-0YA0	V2.2
SIMATIC ET 200SP, PROFINET bundle IM, IM 155-6 PN ST		1	6ES7155-6AA01-0BN0	V4.1
Light-colored BaseUnit (BU...D)	BU_D	1	6ES7193-6BP20-0DA0	
Dark-colored BaseUnit (BU...B)	BU_B	3	6ES7193-6BP20-0BA0	
DI 8x24VDC ST	M1	1	6ES7131-6BF01-0AB0	V0.0
DI 8x24VDC ST	M2	1	6ES7131-6BF01-0AB0	V0.0
DQ 4x24VDC/2A ST	M3	1	6ES7132-6BD20-0BA0	V1.1
DQ 8x24VDC/0.5A ST	M4	1	6ES7132-6BF01-0BA0	V0.0
BU cover	BC	1	6ES7133-6CV15-1AM0	
Server module that comes with the device bundle of ET 200SP as shown in the second record of this table	SM	1	6ES7 193-6PA00-0AA0	V1.1

9.5.1 Menu Items

Menu items in the test application example are explained as follows:

1. **Help**
   Shows the menu again.
2. **Startup**
   Starts all the internal tasks and configures PNIO stack according to the chosen configuration (1 for optional IO devices configuration and 2 for flexible topology). Opens the local Ethernet interface. Registers PN Driver as an IO controller and callback function for IO system reconfiguration. Also changes the mode of the controller to OPERATE.
3. **Enable Variant 2**
   With this function, you can build the "control data record" for Variant 2 (module 1 (M1) is inserted to slot 1 (S1), module 2 (M2) and module 3 (M3) are hidden and module 4 (M4) is inserted to slot 2 (S2)). "Control data record" is written to IO device then "feedback data record" is read from IO device.
4. **Enable Variant 3**
   With this function, you can build the "control data record" for Variant 3 (module 1 (M1) is inserted to slot 1 (S1), module 2 (M2) is inserted to slot 4 (S4), module 3 (M3) is inserted to slot 2 (S2) and module 4 (M4) is inserted to slot 3 (S3)). "Control data record" is written to IO device then "feedback data record" is read from IO device.
5. **Enable Variant 4**
   With this function, you can build the "control data record" for Variant 4 (module 1 (M1) is inserted to slot 1 (S1), module 2 (M2) is inserted to slot 2 (S2) or a BU cover is inserted to slot 2 (S2), module 3 (M3) is inserted to slot 3 (S3) and module 4 (M4) is inserted to slot 4 (S4)). "Control data record" is written to IO device then "feedback data record" is read from IO device.
6. **Enable Variant 5**
   With this function, you can build the "control data record" for Variant 5 (module 1 (M1) is inserted to slot 1 (S1), module 2 (M2) is inserted to slot 2 (S2), module 3 (M3) is inserted to slot 3 (S3) and a new potential group is opened and module 4 (M4) is inserted to slot 4 (S4)). "Control data record" is written to IO device then "feedback data record" is read from IO device.
7. **Enable Variant 6**
   With this function, you can build the "control data record" for Variant 6 (module 1 (M1) is inserted to slot 3 (S3) and a new potential group is opened, module 2 (M2) is hidden, module 3 (M3) is inserted to slot 1 (S1) and module 4 (M4) is inserted to slot 2 (S2) or a BU cover is inserted to slot 2 (S2)). "Control data record" is written to IO device then "feedback data record" is read from IO device.
0. **QUIT**
   You can quit the application by entering 0.

9.6 Receiving alarms

The application example shows how to receive alarms from the local PDEV interface or from any other IO device.

The application example for receiving alarms ("receiving_alarms") uses the same hardware configuration and the same hardware components as the test application. For details, see the section Test application. (Page 102)

9.6.1 Menu Items

Menu items in the test application example are explained as follows:

1. **Help**
   Shows the menu again.
2. **Startup**
   Starts all the internal tasks and configures PNIO stack according to the chosen configuration (1 for optional IO devices configuration and 2 for flexible topology). Opens the local Ethernet interface. Registers PN Driver as an IO controller and callback function for IO system reconfiguration. Also changes the mode of the controller to OPERATE.
3. **Show Alarm Buffer**
   Received alarm information is stored in an alarm buffer. With this function, you can display all the alarm items.
4. **Init Alarm Buffer**
   With this function, you can delete all the alarm items and clear the alarm buffer.

0. **QUIT**
   You can quit the application by entering 0.

9.7 Isochronous mode

The application example for isochronous mode ("isochronous_mode") uses the hardware configuration as shown in the figure below.

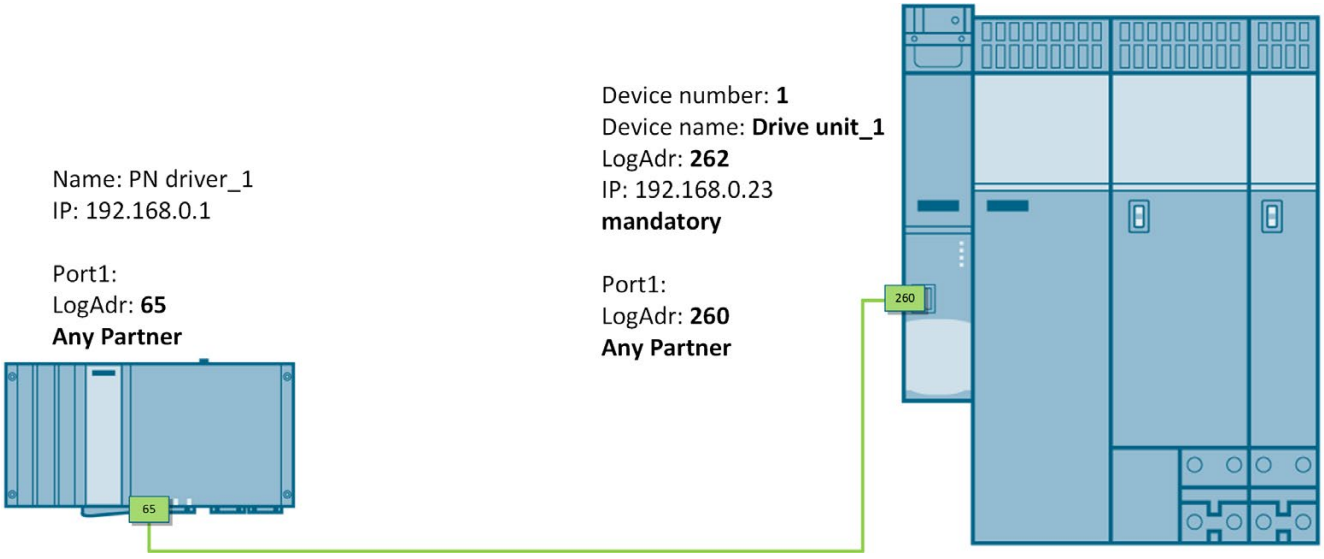


Figure 9-6 Physical hardware configuration of isochronous mode

The required device is listed in the table below. There are basically 2 servo motors with wheels (blue and red), a double motor module, a line module, a sensor module and a control unit (CU320-2 PN) inside the specified device.

Table 9- 5 Hardware components of the configuration control for IO devices

Component	Number	Article number
Training Case SINAMICS S120	1	6ZB2480-0CN00

PN Driver supports ISO mode feature for only CP1625 Host and CP1625 Stand-alone variants. For details see the sections Quick start for CP1625 Stand-alone (Page 64) for CP1625 Stand-alone and Quick start for CP1625 Host (Page 77) for CP1625 Host variants.

Before starting the application example, you need to load the project to the control unit using TIA Portal from port X150. If you need to see the online diagnostics while controlling the device with PN Driver you can connect the control unit over port X127.

Application is capable of setting the PN Driver as a controller for the training case, changing the speed of the servo motors, start-stop of the motors, read encoder position of the motors and write the encoder position to a txt file.

In multiple fields of Isochronous Applications, e.g. motion, it is necessary to uncover whether a data value is from the current cycle or some past cycle. For this purpose "PROFI drive" which is the modular device profile for drive devices developed by PROFIBUS and PROFINET International (PI), defines the so called "Sign-Of-Life" as companion value to the process data. The "Sign-Of-Life" companion value has a range of 0..15, whereas "0" is defined as "applications are not in sync". The sign of life bit is altered in each cycle in the application.

9.7.1 Menu Items

Menu items in the test application are explained as follows:

1. **Help**
   Shows the menu again.
2. **SERV_CP_Startup**
   Starts all the internal tasks and configures PNIO stack according to the configuration. It must be called before "Open Controller".
4. **SERV_CP_Shutdown**
   This must be called as the last function when exiting the user program. After the return of the function, all internal threads have been exited and the entire local memory has been released again.
5. **Open Controller**
   Registers PN Driver as an IO controller with the IO-Base functions. It also registers callback functions for setting the mode, device activation/deactivation, diagnostics requests and IO system reconfiguration.
6. **Close Controller**
   This function deregisters PN Driver as an IO controller. All registered callback functions are also deregistered with this function.
7. **Set Mode PNIO_MODE_OFFLINE**
   With this function, you change the mode of the controller to OFFLINE.
8. **Set Mode PNIO_MODE_CLEAR**
   With this function, you change the mode of the controller to CLEAR.
9. **Set Mode PNIO_MODE_OPERATE**
   With this function, you change the mode of the controller to OPERATE.
10. **Set PN Driver as Controller**
    Makes the training case controllable from PN Driver. The related bit is set and control unit and modules accept data from a controller.
11. **Set Servo Motor Speeds**
    With this function, you can set servo motor speeds (1-6000 rpm) (Red Wheel is 1, Blue Wheel is 2.). Note that, the speed is limited for safety.
12. **Start Servo Motors**
    With this function, motors start rotating.
13. **Stop Servo Motors**
    With this function, motors stop.
14. **Read 1 second Encoder Position Data**
    With this function, you can read the encoder position for 1 second whenever needed.
15. **Write 1 second Encoder Position Data to Text File**
    After reading the encoder position, with this function, you can write it to text file after it is read.
0. **QUIT**
   You can quit the application by entering 0.

In this application, an example working sequence of the training case with PN Driver in ISO Mode can be like below:

2. SERV_CP_Startup

5. Open Contoller

9. Set Mode PNIO_MODE_OPERATE

10. Set PN Driver as Controller

11. Set Servo Motor Speeds (1-6000 rpm)

12. Servo Motors (rotating)

14. Read 1 second Encoder Position Data (if needed)

15. Write 1 second Encoder Position Data to Text File (if needed)

13. Stop Servo Motors

0. QUIT

Please note that RDY light or OPT LEDs on the control unit (CU320-2 PN) may flash red at 2Hz, if the mode of the controller is changed to PNIO_MODE_OFFLINE or PNIO_MODE_CLEAR after PNIO_MODE_OPERATE. If it happens, these LEDs turn to green once the FN key is pressed on the operator panel.

9.8 Isochronous calculation

The application example for isochronous calculation ("iso_calc") illustrates how to calculate the time left for the user application in isochronous mode for a given hardware configuration. If you are using PN Driver in isochronous mode, you may want to know for how long your user program can safely run in each cycle without violating isochronous mode. This example calculates the available free time for the user application in each cycle remaining after the system process time is deduced. The system process time is the total value of IO copy times and all kinds of latencies and jitters depending on the hardware and the operating system. In case of isochronous mode violation, the application stops measuring with a notification. You can either use this application as it is and follow the menu to calculate the remaining application time for a hardware configuration of your choice, or you can adapt the source code into an existing application as described in this section.

In order to get meaningful results, you must use a hardware configuration in which isochronous mode is activated for IO controller and at least some of the IO devices and modules. An example hardware configuration is provided for illustration purposes. Note that PN Driver does not need a connection with actual IO devices for this application to work correctly. Therefore, you can also perform the calculation with a hypothetical configuration.

The application measures the entire processing time and gives the time left for application through a formula. The sample output is shown in the figure below:

```
                              ISO CALCULATION RESULTS
=============================================================================================================================
Maximum Processing time (Max TCA_End) = 188 usec
Time Left For Application             = Application Cycle -  188 usec

Application Cycle = TDC x 31.25 usec : please refer to your configuration for TDC value

Process inputs valid        [ time between the start of the IRT data cycle and the start of the application     ]
        min        142 usec     max        153 usec   average        143 usec

Process outputs transferred [ time between the start of the application and end of application output processing]
        min         34 usec     max         36 usec   average         34 usec


-----------------------------------------------------------------------------
```

Figure 9-7 Sample results

PN Driver supports ISO mode feature for CP1625 Host and CP1625 Stand-alone variants. Additionally, this application is implemented to support only IPO model (read Inputs – Processing – write Outputs). OIP model (write Outputs - read Inputs – Processing) is not supported by this application in isochronous mode. Since PN Driver automatically uses the OIP model for CACF values greater than 1, this application example must be used with a hardware configuration where CACF=1; otherwise, the results will not be reliable.

9.8.1 Interpretation of ISO calculation output

To be able to interpret the results of the calculation, you need to be aware of the following concepts related to isochronous mode:

TDC (Time Data Cycle/Application Data Cycle): This attribute indicates the time for the application data cycle. This time span includes all parts of the isochronous data cycle to convey inputs and outputs. TDC is a binary multiple (i.e. 1, 2, 4, 8…) multiple of SendClock. For PROFINET IO the Application Data Cycle is the so called update time of the isochronously operated IO devices.

TCA_Start: It is the time elapsed between the beginning of the application cycle and the moment the input data becomes available in the communication memory.

TCA_End: It is the time elapsed between the beginning of the application cycle and the moment the output data is written to the communication memory.

TCA_Valid: It describes the time span between the start of the application cycle and the point of time when the input data is available for the Isochronous Application.

TKI (Input PIP processing time): The time needed for processing and transfer of the input data available in the communication memory to the Input PIP before they are ready for use by the application.

TKO (Output PIP processing time): The time needed for processing and transfer of the output data of the PIP before they are ready in the communication memory for sending.

TAppl (Application processing time): The processing time of the user application (TKI / TKO not included). A user application reads input values from the input PIP, does some processing and writes output values to the output PIP.

Sync PI / Sync PO depicts the buffering of inputs and outputs for the Isochronous Application. The main advantage of the buffering is that the IO controller has a consistent image of the process signals for the duration of one cycle of the Application.

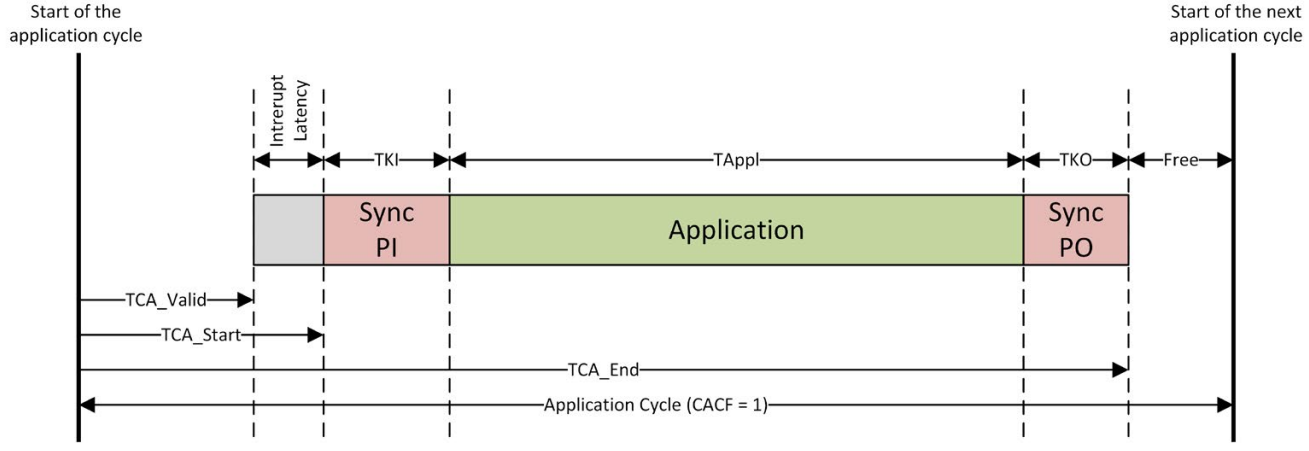


Figure 9-8 Timeline of an application cycle

9.8.2 Menu Items

Menu items in the test application are explained as follows:

1. **Help**
   Shows the menu again.
2. **Startup**
   Starts all the internal tasks and configures PNIO stack according to the configuration. Registers PN Driver as an IO controller with the IO-Base functions. It also registers callback functions for setting the mode, device activation/deactivation, alarm requests, start/stop and opFault.
   Change the mode of the controller to OPERATE. ISO sync and interrupts start.
3. **Start Calculation**
   "Measurement is started" is printed on screen. This message will retain on screen till results will be available once user issues Menu Item 4
4. **Stop Calculation**
   Measurement is stopped and the results are displayed on screen. "ISO Measurement Results" are printed on screen.
5. **Insert TAppl Time**
   Before starting the calculation, you can optionally insert a time period inside the application time using this menu item. In this case, the results should be interpreted as the remaining time after this much time is already used by the application.

0. **QUIT**
   You can quit the application by entering 0.

9.8.3 Enabling debug mode when needed

A compilation flag called "DEBUG_ISO_CALC" is used for more details to be printed to the console while the measurement is running. It can be activated by building the application with the command "make debug".

9.8.4 Adapting calculation source code into an existing application

You can also adapt the source code of this example into an existing application to analyze how much more time you can use for processing data. To do so, following steps must be taken, details of which are explained in the subsequent sections:

1. Import relevant source files and include them in your Makefile
2. Insert your application program in the proper callback function
3. Call functions necessary to manage calculation lifecycle

9.8.4.1 Import relevant source files and include them in your Makefile

The files "iso_calc.cpp" and "iso_calc.h" under the path "[..]/pn_driver/src/examples/iso_calc/src" need to be included in your project.

Your typical Makefile would be similar to the Makefiles which you can find under variant directories; for example "[..]/pn_driver /src/examples/iso_calc/cp1625_host/build". You need to adapt your Makefile to compile "iso_calc.cpp" and keep "iso_calc.h" under an include directory recognized by your Makefile.

9.8.4.2 Insert your application program in the proper callback function

The following callback function will be called at each ISO application cycle if it has been registered with the callback event "PNIO_CP_CBE_STARTOP_CYCLEINFO_IND". Application program should be inserted just before "PNIO_CP_set_opdone" function is called in the function "callback_for_start_op_ind_cycleinfo" of the "iso_calc.cpp" module.

```
void callback_for_start_op_ind_cycleinfo(PNIO_CP_CBE_PRM* pCbfPrm)

{

 ...

 //Please insert your Application User Program here

 // ....

 //End of Application

 PNIO_CP_set_opdone(g_ApplHandle, &(pCbfPrm->u.StartOp.CycleInfo));

 ...

}
```

9.8.4.3 Call functions necessary to manage calculation lifecycle

Before you start the measurement, you first need to startup PN Driver, register it as an IO controller, register necessary callback functions and set the controller mode to "OPERATE". You may use your own callback functions for other callback events, but make sure that the callback events "PNIO_CP_CBE_STARTOP_CYCLEINFO_IND" and "PNIO_CP_CBE_OPFAULT_IND" are registered with the functions "callback_for_start_op_ind_cycleinfo()" and "callback_for_op_fault_ind()", respectively. These functions are defined in "iso_calc.cpp" and they are responsible for the measurement logic.

To start the measurement, call the function "isoStartCalc()". Please note that the measurement will start, if the controller mode is set to OPERATE.

To stop the measurement, call the function "isoStopDisplayCalc()". This function stops the measurement and displays the results on console.

If any isochronous mode violation occurs, then the function "callback_for_op_fault_ind" is called automatically and this function stops the calculation and displays the fault timing.

Note

Please note that you can still register your own callback functions with the callback events "PNIO_CP_CBE_STARTOP_CYCLEINFO_IND" and "PNIO_CP_CBE_OPFAULT_IND". If you choose to do so, you need to adapt them to implement the measurement logic by calling the function "updateIsoModeTiming" properly. You can refer to the definitions of the functions "callback_for_start_op_ind_cycleinfo" and "callback_for_op_fault_ind" given in "iso_calc.cpp" module for the implementation of the measurement logic.

9.9 References

- Additional information on the use of the configuration control can be found on the Internet (https://support.industry.siemens.com/cs/ww/en/view/29430270).
- Additional information on multiple use IO systems and on configuration control for IO systems is available in the function manual PROFINET (https://support.industry.siemens.com/cs/ww/en/view/49948856).
- Additional information on PROFI drive and its usage can be found on the Internet (https://www.profibus.com/technology/profidrive/)

9.10 Opening archived TIA projects in the TIA Portal

1. To open an archived TIA project, select the menu **Project > Open**.

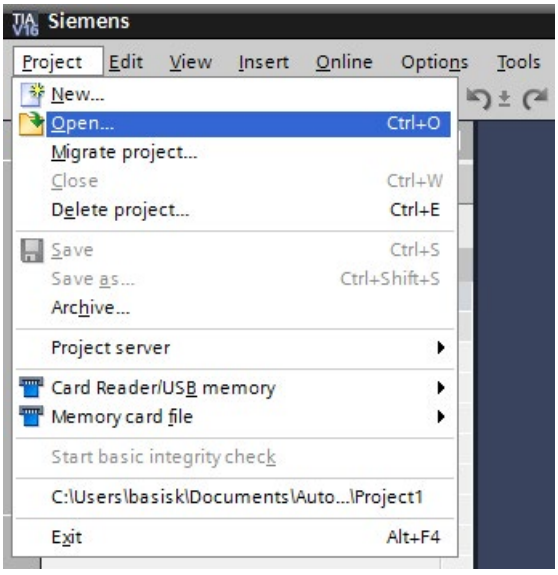


Figure 9-9 TIA Portal - Open

2. In the next window, you first select the archive file of the required project followed by the target directory to which the project should be extracted.

Appendix

A

A.1 Abbreviations / Glossary of terms

API	**A**pplication **P**rogramming **I**nterface
BC	**B**aseUnit **C**over
BU_B	Dark BaseUnit
BU_C	Light BaseUnit (opens a new potential group)
CACF	**C**ontroller **A**pplication **C**ycle **F**actor
CPU	**C**entral **P**rocessing **U**nit
DCP	**D**iscovery and **C**onfiguration **P**rotocol
DLL	**D**ynamically **L**inked **L**ibrary
GDB	**G**NU **P**roject **D**ebugger
GSDML	PROFINET **G**eneral **S**tation **D**escription (GSD) file in XML format
HSP	**H**ardware **S**upport **P**ackage
IO	**I**nput/**O**utput
IPO	Read **I**nputs - **P**rocessing - Write **O**utputs
IRT	**I**sochronous **R**eal **T**ime
Mx	**M**odule **x** (I/O module)
OIP	Write **O**utputs - Read **I**nputs - **P**rocessing
PDEV	**P**hysical **Dev**ice
PN	**P**ROFI**N**ET
PNConfigLib	**P**ROFINET **C**onfiguration **L**ibrary
PNIO	**P**ROFI**N**ET **IO**
PM QOS	**P**ower **M**anagement **Q**uality **o**f **S**ervice
RSE	**R**emote **S**ystem **E**xplorer
RT	**R**eal **T**ime
SDK	**S**oftware **D**evelopment **K**it
SM	**S**erver **M**odule
SNMP	**S**imple **N**etwork **M**anagement **P**rotocol
SSH	**S**ecure **Sh**ell
TCF	**T**arget **C**ommunication **F**ramework
USP	**U**ser **D**atagram **P**rotocol
UUID	**U**niversal **U**nique **Id**entifier
XML	E**x**tensible **M**arkup **L**anguage

Index

S

T

V

W

Y