APPLICATION NOTE

TPS-1

PROFINET Device Design Guideline for TPS-1

R19AN0035ED0302
Rev. 3.02
August 9, 2024

Introduction

The TPS-1 is a single-chip PROFINET interface component integrating a CPU, a 2-port switch supporting latest PROFINET specifications, the Ethernet PHYs and peripheral modules to interface to the application layer of any application building a PROFINET IO device.

This document describes the all aspects for developing a PROFINET field device with the TPS-1. It is intended to be guideline through all steps of a PROFINET device development in order to avoid unpleasant surprises at the customer.

Target Device

TPS-1 (MC-10105F1-821-FNA-M1-A)

When using this application note with other Renesas products than TPS-1, careful evaluation is recommended after making modifications to comply with the alternate product.

Contents

1. Introduction and Overview

The present document describes the necessary steps for developing a PROFINET field device with the TPS-1.

The TPS-1 is designed as communication controller, which by itself (for simple applications) or in combination with an application CPU builds a field device.

PROFINET is an established, flexible and powerful industrial Ethernet network based on the IEEE 802 standard. PROFINET has prevailed in recent years, not only because of its open architecture but also because of its strengths. There is the powerful diagnostic model, the possibility of coexisting with Internet protocols on the same cable or the scalability of the communication.

So much performance is not quite for nothing. PROFINET is no simple standard. A PROFINET development requires the cooperation of all disciplines involved.

- Mechanics
- Hardware
- Software
- Product introduction

This document discusses TPS-1 specific topics so that an implementation can be tackled quickly.

The TPS-1 does not only consist of a silicon device as hardware but also of a larger number of software products and tools that support the development decisively.

The TPS Development Toolkit belongs to the TPS-1 and can be downloaded (after registration) free of charge from the Phoenix Contact Software website

https://www.phoenixcontact-software.com/en/downloads

1.1 Presumptions

This description presumes that the basic hard and software properties (e.g. IO Mode, Peripheral Mode, etc.) are known. If required, the TPS-1 data sheet and user's manual must be consulted. The "TPS-1 Reference Manual" (Driver description) is contained in the TPS Development kit and describes the software interface to the host CPU. The fundamentals of PROFINET protocols must be known. The PROFINET user organization PI (PROFIBUS and PROFINET International) e.g. offers multiple 1-day training courses per year on this topic - see PNO website

http://www.profibus.com/nc/trainingevents/

A summary of PROFINET topics is available in the book "Industrial Communication using PROFINET" by Manfred Popp; PI order no.: 4,182 (german version: "Industrielle Kommunikation mit PROFINET by Manfred Popp, PI order no. 4181)

Document Type	Description	Document Title	Document No.
Data Sheet	Hardware overview and electrical characteristics	TPS-1 Datasheet	R19DS0069EJ0106 (or newer)
User's manual for Hardware	Hardware specifications (pin assignments, memory maps, peripheral function specifications, electrical characteristics, timing charts) and operation description.	TPS-1 User's Manual for Hardware	R19UH0081ED0105 (or newer)

2. Structure of TPS Development Toolkit

For the development of a PROFINET field device, besides the ASIC, a software package is also needed. This software packet contains all programs and tools for the implementation and test of a field device. One only requires the development environment for the given host CPU which is typically available from the device manufacturer.

The TPS Development Toolkit can be downloaded after registration from the following address:

www.phoenixcontact-software.com/en/downloads

Since the TPS Development Toolkit is a commercial piece of software, registration by name is required.

A combination of different stack versions (e.g. 1.2 and 1.3) has not been tested, and the function cannot be guaranteed. Users are explicitely discouraged to mix components from different toolkit versions.

Figure 2-1 shows the interaction of the different toolkit hardware and software components; this is explained in the subsequent chapters in more detail.

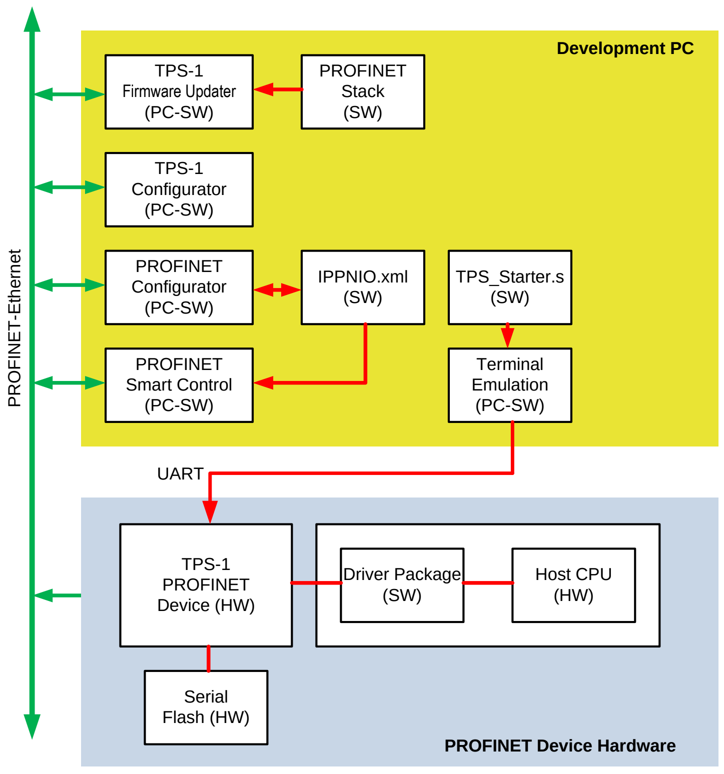
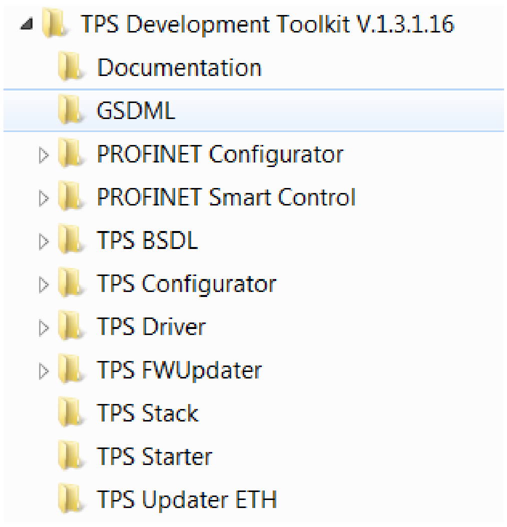

Development PC
TPS-1 Firmware Updater (PC-SW)
PROFINET Stack (SW)
TPS-1 Configurator (PC-SW)
PROFINET Configurator (PC-SW)
IPPNIO.xml (SW)
TPS_Starter.s (SW)
PROFINET Smart Control (PC-SW)
Terminal Emulation (PC-SW)
PROFINET-Ethernet
UART
TPS-1 PROFINET Device (HW)
Driver Package (SW)
Host CPU (HW)
Serial Flash (HW)
PROFINET Device Hardware

Figure 2-1: Interaction between HW and SW components

2.1 Overview of TPS Development Toolkit

The following figure shows the directory structure of the TPS Development Toolkit.



```
◢ TPS Development Toolkit V.1.3.1.16
    Documentation
    GSDML
  ▷ PROFINET Configurator
  ▷ PROFINET Smart Control
  ▷ TPS BSDL
  ▷ TPS Configurator
  ▷ TPS Driver
  ▷ TPS FWUpdater
    TPS Stack
    TPS Starter
    TPS Updater ETH
```

Figure 2-2: Overview TPS Development Toolkit

Note: The version number may vary when newer versions of the toolkit have been released.

2.2 TPS-1 Firmware

The TPS-1 firmware consists of:

- TPS Updater ETH
- TPS Stack
- Hardware configuration

Before loading the firmware, first the hardware configuration of TPS-1 must be loaded in the TPS Flash (using the TPS Configurator). The hardware configuration is done via a graphical user interface and is then transferred into Flash.

The same also applies to the TPS Updater and the TPS Stack. For this, the firmware updater is used.

2.3 TPS-1 Documentation

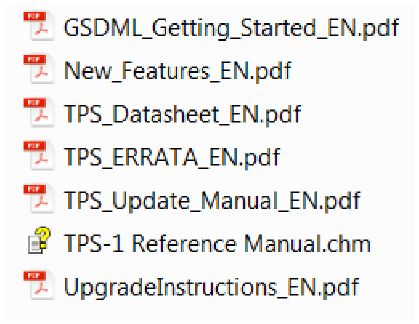

GSDML_Getting_Started_EN.pdf
New_Features_EN.pdf
TPS_Datasheet_EN.pdf
TPS_ERRATA_EN.pdf
TPS_Update_Manual_EN.pdf
TPS-1 Reference Manual.chm
UpgradeInstructions_EN.pdf

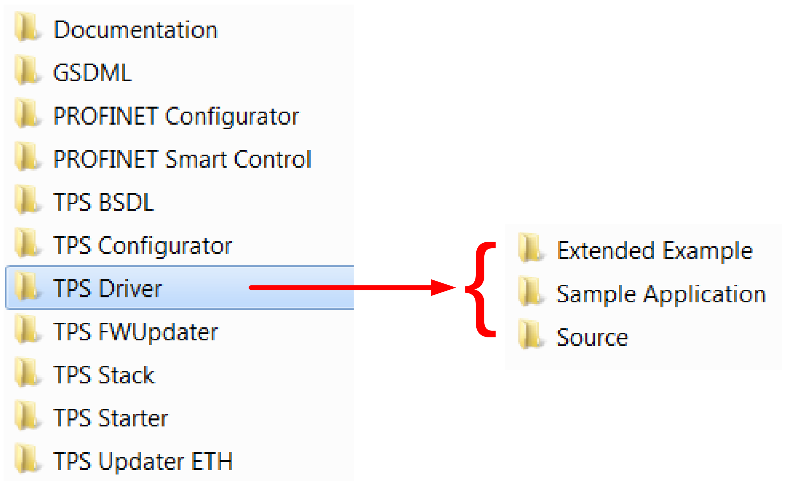
Figure 2-3: TPS Development Toolkit Documentation

In this directory, all the necessary documents for the TPS-1 and the Driver (z. B. TPS-1 Reference Manual) can be found. The document “GSDML Getting_Started_EN.pdf” gives an introduction into the structure and use of a GSD file.

2.4 TPS Driver (API Host Application)

The TPS Driver is an API for use on a host CPU. The software is available in source code and is linked to the application code. The Driver facilitates the user application to be easily connected to the TPS-1. The TPS Driver is written in ANSI C and can be used in most cases as it is.

Caution: Alterations in the Driver should be avoided. Especially, depending upon the software version, the files in the „Source” directory are subject to change and must be interchangeable.


Documentation
GSDML
PROFINET Configurator
PROFINET Smart Control
TPS BSDL
TPS Configurator
TPS Driver
TPS FWUpdater
TPS Stack
TPS Starter
TPS Updater ETH

Extended Example
Sample Application
Source

Figure 2-4: Driver code overview

2.4.1 Driver API Files (Source)

The Driver API files are divided into three parts:

- Communication interface (SPI1_Master.*)
- Driver interface (TPS_1_API.*)
- Compile Information (TPS_1_user.h)

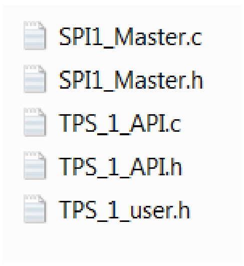

SPI1_Master.c
SPI1_Master.h
TPS_1_API.c
TPS_1_API.h
TPS_1_user.h

Figure 2-5: Driver application programming interface files

All accesses to the TPS-1 DPRAM are encapsulated by access functions (for example, TPS_SetValue8 ()) and with the help of the pre-processor setting (e.g. SPI_INTERFACE from TPS_user.h), it is decided here via which interface the TPS-1 is to be addressed. For the SPI interface, the TPS_SPIWriteData() and TPS_SPIReadData() functions still need to be implemented. The elementary transfer parameters, which can be processor specific, must be provided here.

2.4.2 Sample Application

The sample application consists of the „main.c“ file. All functions that are necessary for the simple operation of a PROFINET field device are pooled together in this file. In the example, a device is configured which consists only of a DAP, a slot, and a subslot. Two data bytes are output, and two data bytes are read.

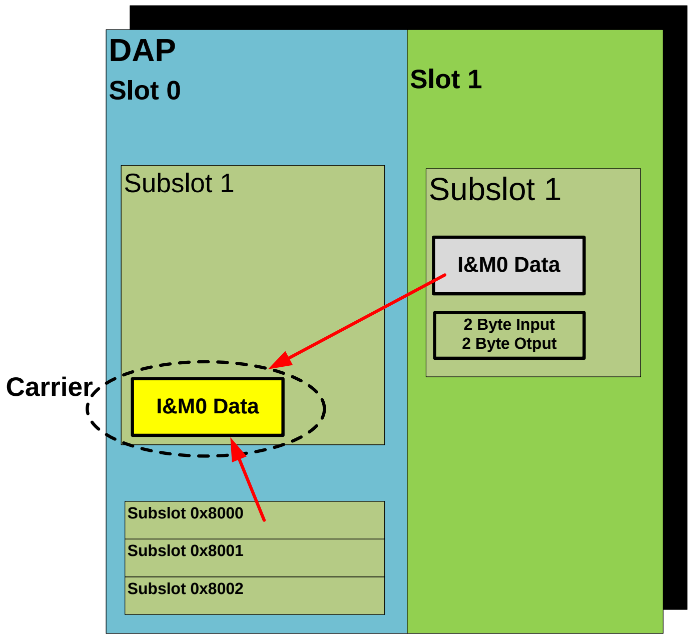


Figure 2-6: Device model for sample application

2.4.3 Extended Application

The extended application gives further examples to facilitate implementation of PROFINET device features. Here, e.g. pull and plug operations, adaptation to the target configuration, TPS-1 configuration and firmware update via the DPRAM are illustrated.

2.4.4 Modifications Made for the Renesas TPS-1 Solution Kits

Customers that use one of the Renesas TPS-1 solution kits (YCONNECT-IT-TPS-1 or YCONNECT-IT-TPS-1L) have several versions of the API and the sample application. Additionally to what is described in chapters 2.4.1 and 2.4.2 the TPS-1 solution kits contain CPU-specific adaptations of these software elements. The supported target CPUs are the

- the µPD70F3767 from the Renesas V850 MCU family (used as well on the solution kit hardware board)
- the R5F5630E from the Renesas RX63x MCU family
- the R5F52315ADFP from the Renesas RX23x MCU family (only YCONNECT-IT-TPS-1L)
- the R7FS7G27H3A01CFC from the Renesas Synergy MCU family (only YCONNECT-IT-TPS-1L)

Files from the Phoenix Contact Software original and the Renesas adaptations should not be mixed.

2.4.5 Modifications for Different Application Processors

Before compiling the Driver it is necessary to check the preprocessor option (TPS_1_user.h). Data structures inside the file TPS_1_API.h must be packed.

This is necessary because the TPS-1 internal CPU uses packed structures and if an application CPU does not follow these instructions you will not find the correct addresses inside the DPRAM.

Figure 2-7 shows a short code snip from TPS_1_user.h. If you use other tool chains than IAR EWV850, you have to check, how your specific compiler handles data packing. Normally you will have to fill up the `#else` case in the code snip with code needed to “emulate” the behavior or the IAR EWV850 compiler.

```
/*-------------------------------------------------------------------------*/
/* For IDE's other than IAR Workbench you have to define your own          */
/* packing directive. Otherwise the structure alignment might be wrong!    */
/*-------------------------------------------------------------------------*/
#if defined(__IAR_SYSTEMS_ICC__)
    #define PRE_PACKED _Pragma("pack(1)")
    #define POST_PACKED _Pragma("pack()")
#else
    #define PRE_PACKED
    #define POST_PACKED
#endif
```

Figure 2-7: Data packing instructions in TPS_1_user.h

2.5 GSDML Directory

The GSDML directory contains two examples of GSDML files that match to the „Sample Application“(GSDML-V2.31-Phoenix_Contact-TPS1-Template-20150603.xml), and the „Extended Application“(GSDML-V2.31-Phoenix_Contact-TPS1-Extended-20151026.xml) respectively. You should modify these files stepwise and after successful commissioning only.

Note: The creation dates given here correspond to version V1.3.1.16 of the TPS-1 development toolkit. Newer versions may have newer GSDML files.

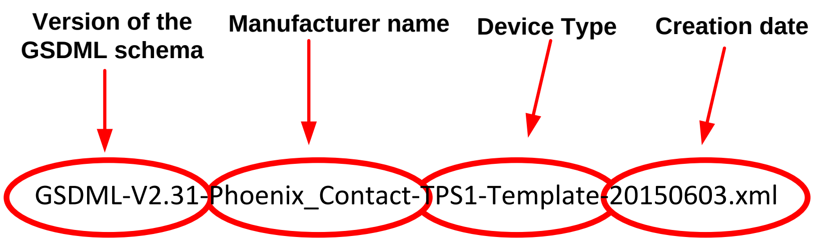

Version of the GSDML schema
Manufacturer name
Device Type
Creation date
GSDML-V2.31-Phoenix_Contact-TPS1-Template-20150603.xml

Figure 2-8: GSDML name convention

Figure 2-8 describes the structure of a name for a description file. The file name must be compliant to this convention and will be verified during the certification.

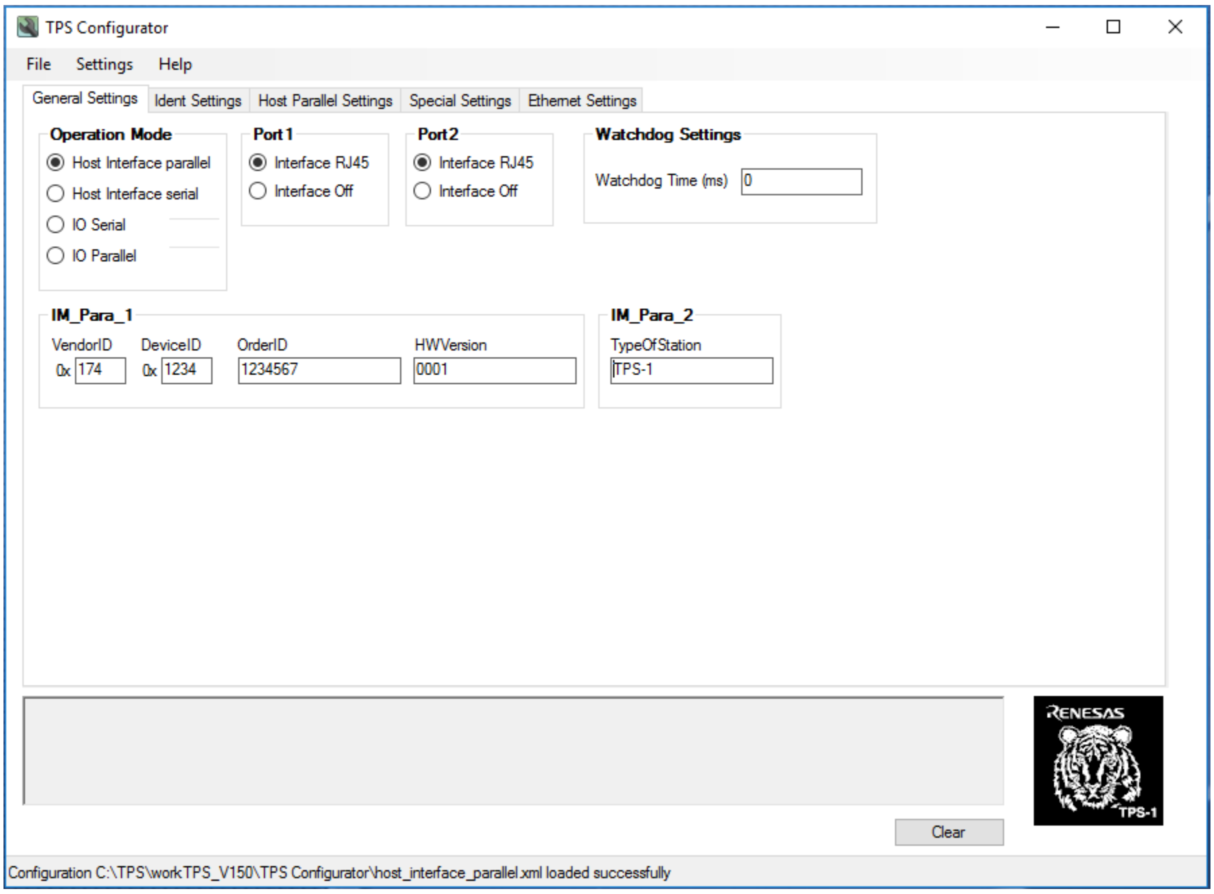
This scheme version is not related to the PROFINET version.

2.6 Test and Development Tools

For the test and subsequent commissioning, additional software tools are available in the TPS Development Toolkit. The following chapters provide a brief overview of the use of these tools; they do not replace studying the manuals and other related documents.

2.6.1 TPS Configurator

The TPS Configurator is a tool for configuring the TPS-1 hardware. The desired characteristics for specific device development are determined by a graphical user interface and then transferred to the TPS Flash using the Ethernet interface.



Operation Mode: [✓ Host Interface parallel] Host Interface serial IO Serial IO Parallel

Port 1: [✓ Interface RJ45] Interface Off

Port 2: [✓ Interface RJ45] Interface Off

Watchdog Settings: Watchdog Time (ms) 0

IM_Para_1: VendorID 0x 174, DeviceID 0x 1234, OrderID 1234567, HWVersion 0001

IM_Para_2: TypeOfStation TPS-1

Configuration C:\TPS\work TPS_V150\TPS Configurator\host_interface_parallel.xml loaded successfully

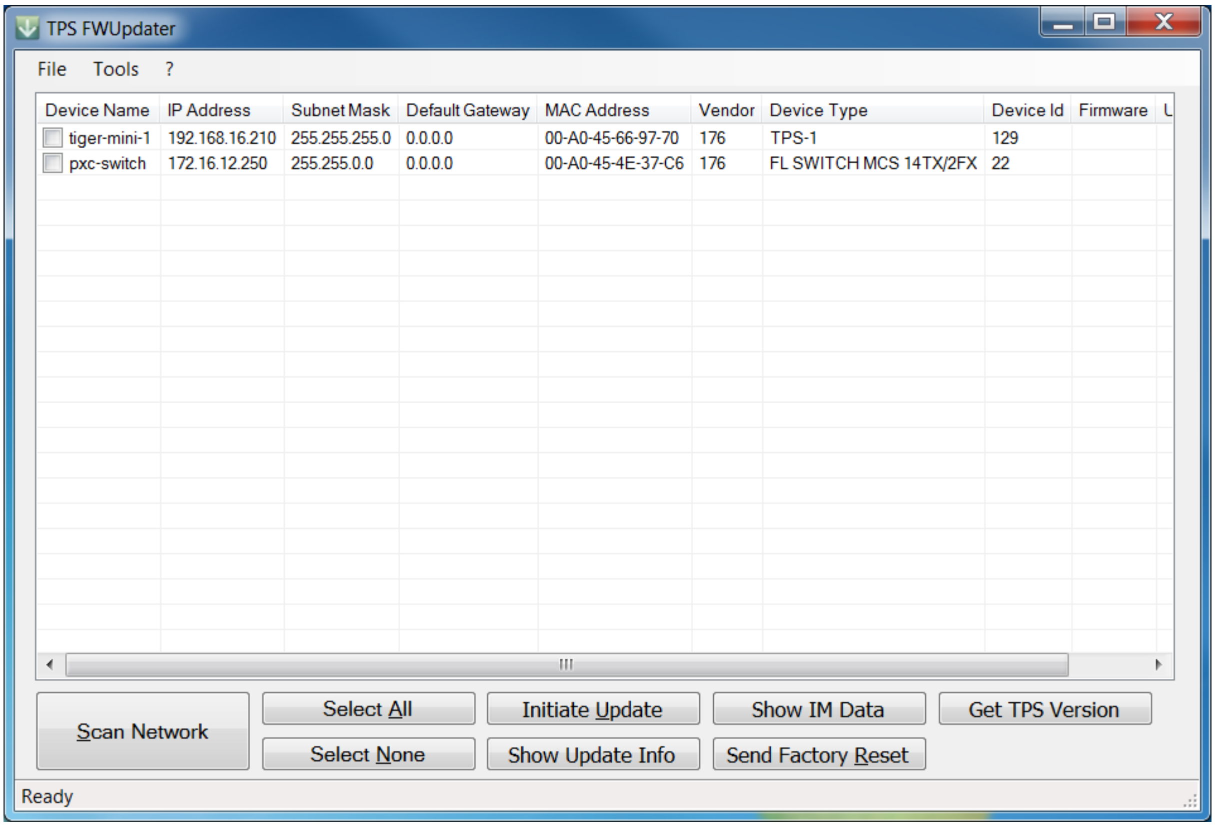
Figure 2-9: Start screen TPS Configurator

With the TPS Configurator, the operating mode and the interfaces used (e.g. Ethernet copper, Ethernet fiber optic, SPI interface, parallel interface) can be selected.

Note: Since TPS Development Toolkit V1.7.0 in TPS Configurator, fiber optic interface support is no longer available and a new tab “Special Setting” has been introduced to customize BER setting. Attention!! changing the default BER setting can have a negative effect on the functionality of the system.

2.6.2 TPS FWUpdater

The TPS FWUpdater tool is used to initialize the firmware (TPS Updater and TPS Stack). In the finished PROFINET field device, this tool can be used for an update of the TPS Firmware.

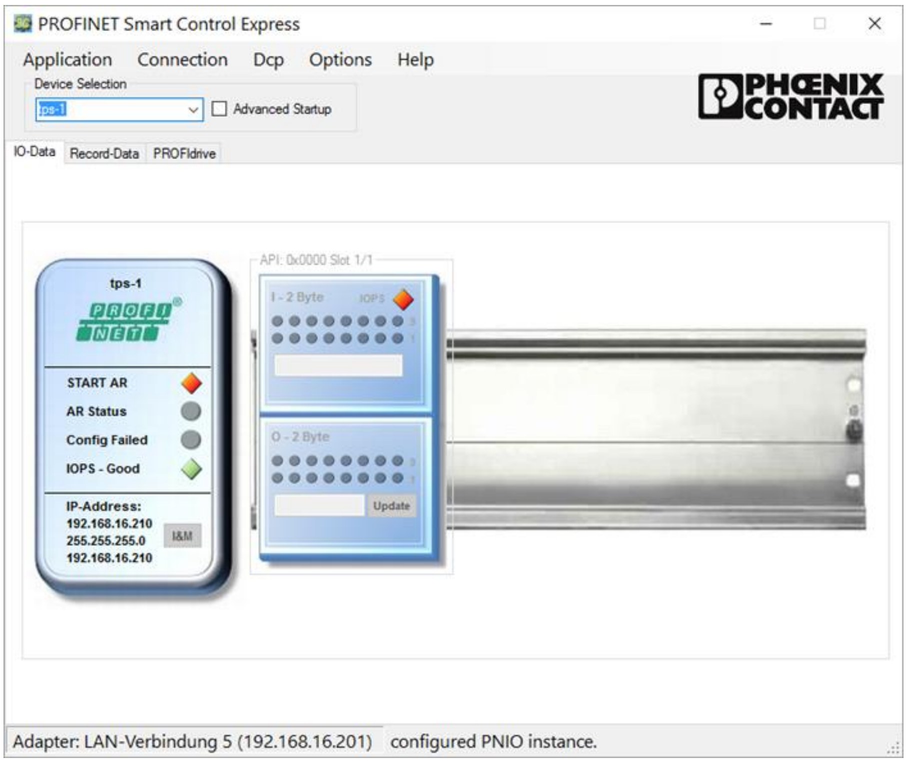


Device Name	IP Address	Subnet Mask	Default Gateway	MAC Address	Vendor	Device Type	Device Id	Firmware
tiger-mini-1	192.168.16.210	255.255.255.0	0.0.0.0	00-A0-45-66-97-70	176	TPS-1	129	
pxc-switch	172.16.12.250	255.255.0.0	0.0.0.0	00-A0-45-4E-37-C6	176	FL SWITCH MCS 14TX/2FX	22	

Figure 2-10: Power-up-screen TPS FWUpdater

2.6.3 PROFINET Smart Control Express

The PROFINET Smart Control Express application is a software IO controller (PLC) in reduced form, useful for making initial steps towards the development of a module. It is possible to establish a connection to a device. It is possible to exchange cyclic and acyclic data so that individual development steps can be tested easily. PROFINET Smart Control Express is installed on a PC (see Figure 2-1).



Device Selection: tps-1

Advanced Startup

API: 0x0000 Slot 1/1

tps-1

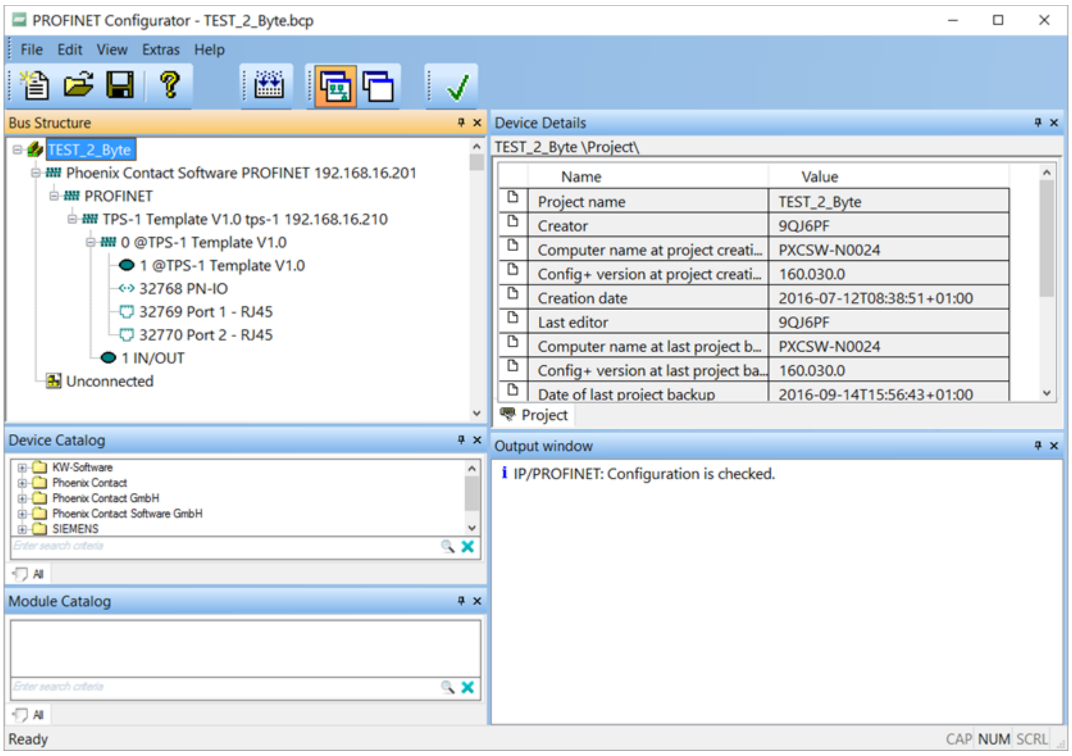
IP-Address:
192.168.16.210
255.255.255.0
192.168.16.210

Adapter: LAN-Verbindung 5 (192.168.16.201) configured PNIO instance.

Figure 2-11: PROFINET Smart Control Express

2.6.4 PROFINET Configurator Express

Configuration of a PROFINET network is a must. A PROFINET Configurator Express is included in the Toolkit for this task. The IPPNIO.xml file created here is loaded by the PROFINET Smart Control and defines the network (see Figure 2-1).



```
TEST_2_Byte
  Phoenix Contact Software PROFINET 192.168.16.201
    PROFINET
      TPS-1 Template V1.0 tps-1 192.168.16.210
        0 @TPS-1 Template V1.0
          1 @TPS-1 Template V1.0
          32768 PN-IO
          32769 Port 1 - RJ45
          32770 Port 2 - RJ45
        1 IN/OUT
  Unconnected
```

TEST_2_Byte \Project\

Name	Value
Project name	TEST_2_Byte
Creator	9QJ6PF
Computer name at project creati...	PXCSW-N0024
Config+ version at project creati...	160.030.0
Creation date	2016-07-12T08:38:51+01:00
Last editor	9QJ6PF
Computer name at last project b...	PXCSW-N0024
Config+ version at last project ba...	160.030.0
Date of last project backup	2016-09-14T15:56:43+01:00

```
KW-Software
Phoenix Contact
Phoenix Contact GmbH
Phoenix Contact Software GmbH
SIEMENS
```

```
i IP/PROFINET: Configuration is checked.
```

Figure 2-12: PROFINET Configurator Express

3. Operating Modes of TPS-1

The TPS-1 can be operated in two different modes. For simple IO applications (maximum 48 GPIOs) the chip can use an internal application. For complex and demanding applications, an application processor can be connected, which processes the PROFINET device specific firmware.

3.1 I/O Configuration (Local IO)

In this operating mode, the PROFINET interface cannot be programmed by the user. The use of the 48 GPIOs is set-up with the TPS Configurator. It is possible to set-up Inputs and outputs, as well as diagnostic inputs. If the number of IOs proves to be insufficient, then it is possible to use external IO devices via an SPI master and to drive up to 340 bytes with it.

3.2 Host Interface Modus (Parallel or Serial)

In the Host Interface mode, another processor is connected to the TPS-1 (Peripheral Interface) which operates either via parallel (8 or 16-bit data bus width) or an SPI interface. Also, here the hardware configuration is done using the TPS Configurator.

4. Implementation of a Field Device

For the configuration of a TPS based PROFINET device, a few software program steps must be executed which are described below. The description given here is no substitute for reading the DRIVER program code.

4.1 PROFINET Device Model

The process data addressing is derived from the structure of a PROFINET field device. PROFINET has a uniform addressing model and one can differentiate between compact and modular field devices. In case of modular devices, a different device function can be selected through add-on hardware.

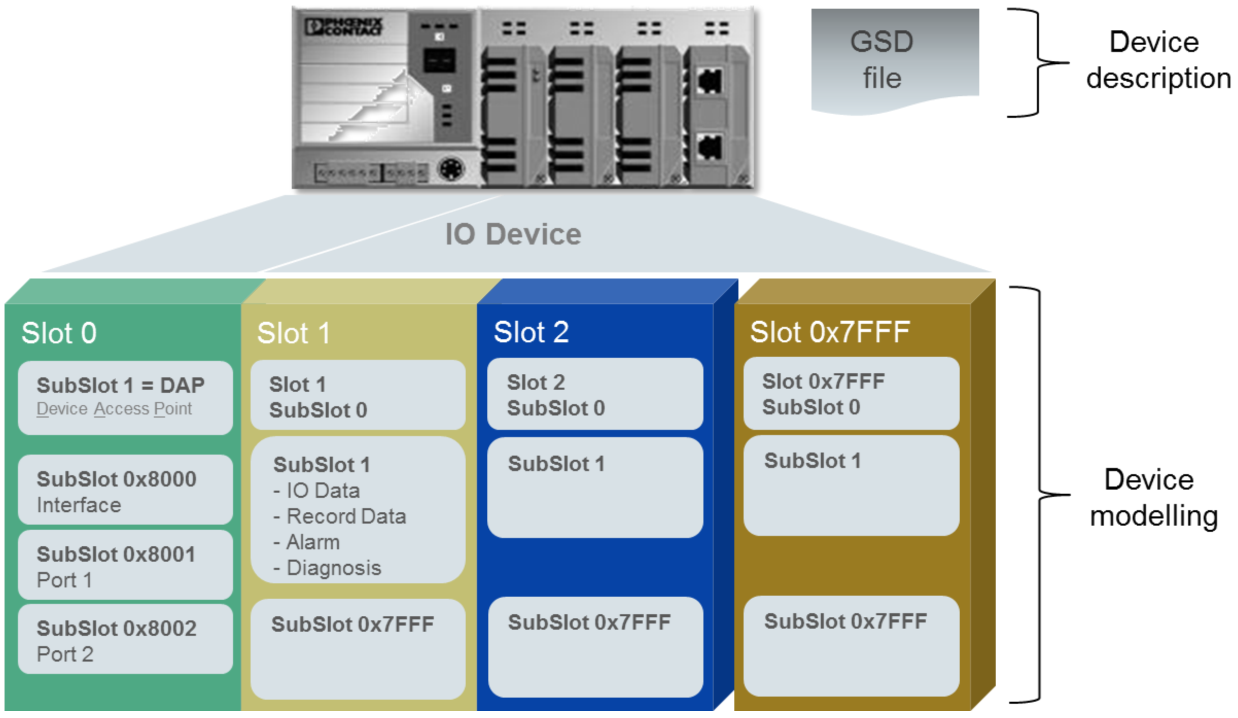


Figure 4-1: PROFINET device model

The following parts are to be differentiated:

- the device with Slot 0, Subslot 1 as DAP (Device Access Point)
- individual peripheral modules by slots
- individual I/O channels of a peripheral module within subslots
- program parts/functions within a peripheral module by indices (only in acyclic data transaction)
- assignment of alarms based on the corresponding message

Device expansion and its possibilities are described in the GSD file. Only combinations which are described here can be configured using the engineering tool of the PROFINET controller

For device configurations, precisely these settings are made by the device manufacturer’s (application CPU) firmware.

4.2 Device Configuration

Figure 4-2 describes the sequence of necessary steps that a device must have executed in host mode up to the start of connection (Connect.Req). The names of the related functions in the API are shown in red.

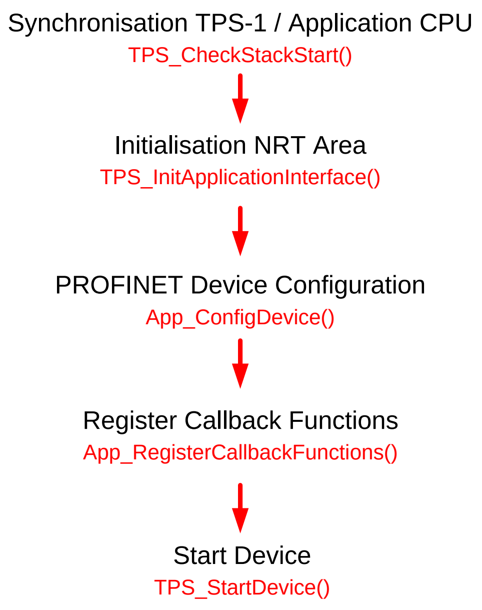

Synchronisation TPS-1 / Application CPU
TPS_CheckStackStart()
Initialisation NRT Area
TPS_InitApplicationInterface()
PROFINET Device Configuration
App_ConfigDevice()
Register Callback Functions
App_RegisterCallbackFunctions()
Start Device
TPS_StartDevice()

Figure 4-2: Device configuration sequence

The necessary operations are carried out in the DPRAM of the TPS-1. The NRT area starts from address 0x8000 and contains the configurations for the device, the Slot / Subslot configuration and the mailboxes for acyclic data exchange (Record Data).

The steps illustrated in Figure 4-2 will be explained in more detail in the next chapters.

4.2.1 Synchronisation TPS-1 / Application CPU

After turning on the power supply or after a reset, the TPS-1 begins to load the stack from the TPS Flash to start. After the TPS-1 has completed its initialization, it writes the so-called magic number at address 0x8000 and the size of the NRT area at address 0x8004. These values are always accessible here and can be used for test purposes (these values apply to stack version V1.4.0.14 and may be subject to change in the newer versions):

• Magic number	0x0400 0009
• NRT area size	0x0000 8000

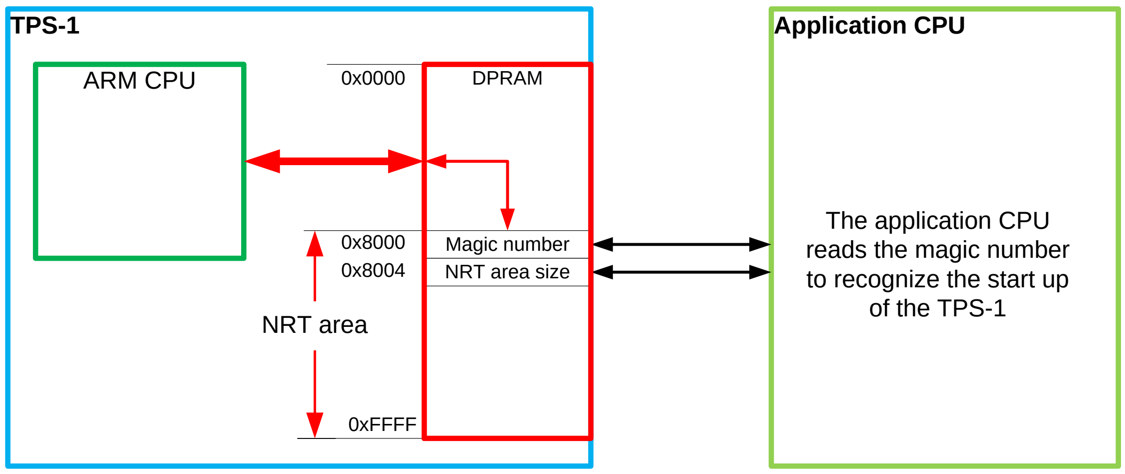


Figure 4-3: Start-synchronisation

This status can be checked by calling the ***TPS_CheckStackStart()*** function. This function always provides the DRIVER in the latest version.

Note: The TPS-1 Flash can also be programmed without a functional host connection via the Ethernet interface.

4.2.2 Initialisation of the NRT Area

By calling the TPS InitApplication Interface () function, the basic settings for the memory areas (DPRAM) and global variables are made. No alterations should be made here. You will find a complete picture of the NRT area in the **TPS-1 Reference Manual.chm**

Address		Name	Description
0x8000	USIGN32	dwMagicNumber	Show the correct start of the TPS-1; actual value 0x0400 0009
0x8004	USIGN32	dwNrtMemSize	Show the NRT Area Size (0x8000)
0x8008	USIGN16	wVendorID	PROFINET ID of the vendor
0x800A	USIGN16	wDeviceID	Device ID (chosen by the vendor)
0x800C	USIGN8	byStationName [STATION_NAME_LEN]	Max. 240 character - length of "Name of Station" without terminating
0x80FC	USIGN16	wReserved	
0x80FE	USIGN16	wDeviceVendorTypeLength	Length of the Device Vendor Type

Address		Name	Description
0x8100	USIGN8	byDeviceVendorType [TYPE_OF_STATION_STRING_LEN]	Max. 25 character - length of "Type of Station"
0x8119	USIGN8	bySerialnumber [IM0_SERIALNUMBER_LEN]	Max. 16 character - length of serial number
0x8129	USIGN8	byOrderId[IM0_ORDERID_LEN]	Max 20 character - length of order ID
0x813D	USIGN8	byPadding [TYPE_OF_STATION_PADDING]	Padding (92 byte)
0x8199	USIGN8	dwFastStarupParam [SIZE_FSU_PARAMETER]	Fast start up parameter - max. 20 bytes
0x81AD	USIGN32	dwIOBufferLenAr [MAX_ARS_SUPPORTED]	CR dependent length of the IO data
0x81B9	USIGN32	dwApduAddrForCr [MAX_ARS_SUPPORTED]	Offset of the APDU for the output data of an AR
0x81C5	USIGN32	dwHostProtoSelector	Protocol selector (see example)

0x81C9	USIGN16	wLEDState	Bit(1:on/0:Off)
			0:BF, 1:SF, 2:MD, 3:MR, 4:LinkP1, 5:LinkP2, 6-15:Reserved;

0x81CB	USIGN16	wResetOption	FactroryToReset option
0x81CD	USIGN32	dwTPSFWVersion	
0x81D1	USIGN32	dwAPIVersion	Contain the driver version (e.g. 0x14 -> 1.4)
0x81D5	USIGN32	dwReservedValue	
0x81D9	USIGN8	byAppConfMode	Access Mode for name of station
0x81DA	USIGN8	byInterfaceMac [MAC_ADDRESS_SIZE]	MAC address device (6 byte)
0x81E0	USIGN8	byPort1Mac [MAC_ADDRESS_SIZE]	MAC address port 1 (6 byte)
0x81E6	USIGN8	byPort2Mac [MAC_ADDRESS_SIZE]	MAX address port 2 (6 byte)
0x81EC	USIGN8	byRevisionPrefix	Element of the firmware version
0x81ED	USIGN8	byRevisionFunctionalEnhancement	Element of the firmware version
0x81EE	USIGN8	byBugFix	Element of the firmware version
0x81EF	USIGN8	byInternalChange	Element of the firmware version
0x81F0	USIGN32	dwIPAddress	Device IP Address (4 byte)
0x81F4	USIGN32	dwSubnetMask	Device Subnet Mask (4 byte)
0x81F8	USIGN32	dwGateway	Device Gateway Address (4 byte)
0x8200	USIGN16	Max Number of IOAR	Possible number of active Application Relations (2 byte)
0x8202	USIGN16	First Record Mailbox	

Table 4-1: NRT memory address

The table above shows the addresses and parameters at the beginning of the NRT area. The structure is valid in version 1.5.1.2 of the TPS-1 stack. Please refer to the structure NRT_APP_CONFIG_HEAD from the file TPS_1_API.h for more information.

4.2.3 Device Configuration

The device configuration determines the structure and addressing of the field device.

The first step is to call the function ***TPS_AddAPI()***. This function sets up the API (Application Process Identifier) in the NRT area (the function should not be altered). The typical value for the API (Application Process Interface) is the 0x0000. Other applications may require other values (e.g. PROFIdrive)

The DRIVER is configured by the ***App_ConfigDevice()*** function.

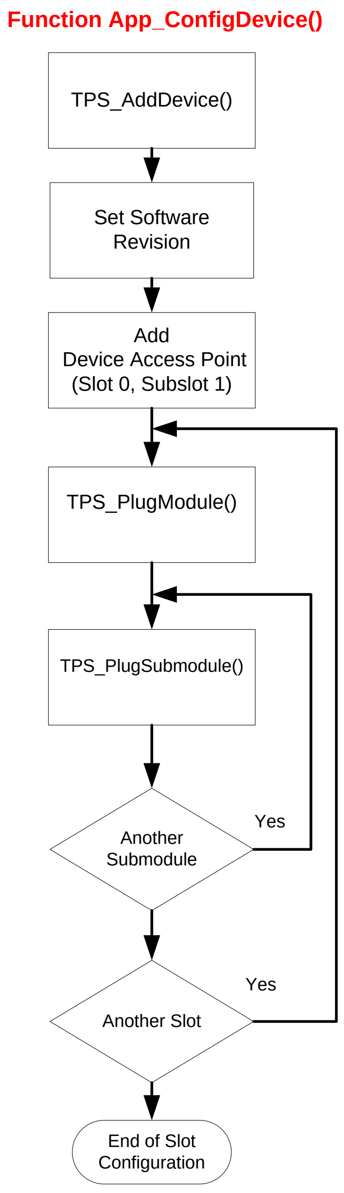


Figure 4-4: Device configuration with the App_ConfigDevice() function

Within the App_ConfigDevice() function, the DAP (Slot 0 / Subslot 1) is set first. In a further step the software revision is set. The software revision is required at various places (e.g. for I&M data). The TPS_Add Device () function creates several entries in the NRT area and mailboxes, which serve to manage the device.

1) Adding a Module (Slot)

In the device GSD, each module (slot) must be described in the ModuleList. The GSD data are used by the engineering tool for PROFINET controller setting. A typical module entry is shown in

Figure **4-5**.

```
<ModuleList>
        <ModuleItem ID="ID_Mod_01" ModuleIdentNumber="0x00000002">
          <ModuleInfo CategoryRef="ID_Out">
            <Name TextId="TOK_TextId_Module_1IO" />
            <InfoText TextId="TOK_InfoTextId_Module_1IO" />
            <HardwareRelease Value="1.0" />
            <SoftwareRelease Value="1.0" />
          </ModuleInfo>
          <VirtualSubmoduleList>
            .
            .
          </VirtualSubmoduleList>
        </ModuleItem>
</ModuleList>
```

Figure 4-5: Module entry in the GSDML

All further modules are added by the ***TPS_PlugModule()*** function. It is important to note at this point that the **„ModuleIdentNumber“** must as well be available in the GSD file. Deviating values prevent later a connection of the PROFINET controller to the field device

```
SLOT* TPS_PlugModule (API_LIST* pzApi,
                      USIGN16   wSlotNumber,
                      USIGN32   dwModuleIdentNumber)
```

Figure 4-6: Function call TPS_PlugModule()

A detailed description of the function can be found in [2]. The function returns a handle on the slot to which one or more subslots can be connected.

2) Adding Submodules to the Device

With the TPS_PlugSubmodule() function, a submodule is added to the configuration. A submodule is a structure for data exchange. A **VirtualSubmoduleItem** describes a submodule (always in conjunction with a module).

```
<VirtualSubmoduleList>
  <VirtualSubmoduleItem ID="1" SubmoduleIdentNumber="0x0002" API="0">
    <IOData IOPS_Length="1" IOCS_Length="1">
       <Input Consistency="Item consistency">
         <DataItem TextId="T_ID_IN_2BYTE" DataType="OctetString" Length="2"
                   UseAsBits="true" />
       </Input>
       <Output Consistency="Item consistency">
```

```
            <DataItem TextId="T_ID_OUT_2BYTE" DataType="OctetString"
                      Length="2" UseAsBits="true" />
        </Output>
     </IOData>
         .
         .
     <ModuleInfo>
       <Name TextId="TOK_TextId_Module_1IO" />
         <InfoText TextId="TOK_InfoTextId_Module_1IO" />
     </ModuleInfo>
   </VirtualSubmoduleItem>
</VirtualSubmoduleList>
```

Figure 4-7: VirtualSubmoduleItem entry in the GSD

The number of parameters passed gets larger because the data exchange structures are created here. Within the NRT area an administrative structure for a module and submodule is created, which contains all information about the data to be exchanged.

```
     SUBSLOT* TPS_PlugSubmodule(SLOT*       pzSlotHandle,
                               USIGN16     wSubslotNumber,
                               USIGN32     dwSubmoduleIdentNumber,
                               USIGN16     wInitParameterSize,
                               USIGN16     wNumberOfChannelDiag,
                               USIGN16     wSizeOfInputData,
                               USIGN16     wSizeOfOutputData,
                               T_IM0_DATA* pzIM0Data,
                               BOOL        bIM0Carrier)
```

Figure 4-8: Calling parameter of the TPS_PlugSubmodule

Among the parameters passed, the "wInitParameterSize" is important; it sets the size of initial parameters to be passed following the Connect.Req. The initial parameters are loaded from the GSDML file. An adequately sized memory space must be kept free within the NRT area.

wInitParameterSize = HeaderSize + InitParameterSize; (HeaderSize 6 Byte)

If substitute values have been defined for the field device, then these are stored behind the initial parameters in the NRT area. A header (6 bytes) is available also for the substitute values.

wInitParameterSize = HeaderSize + InitParameterSize +HeaderSize + SUBSTITUTE_CONFIG_SIZE);

The pointer „pzIM0Data" passes the address of a memory area containing the data. The Boolean value of „bIM0Carrier" indicates whether I&M0 data belong directly to the submodule or whether the pointer references to a different submodule (Carrier). This process is discussed in Chapter 5.1 in more detail.

4.2.4 Registration of Callback Functions

The communication between the TPS-1 and a connected host CPU is done via an event register. The event register query is organized via the ***TPS_CheckEvents()*** function. For each event, a callback function is invoked, which carries out the necessary actions as required. The list of events is given in the TPS-1 user manual [1].

The Callback functions are registered with the „***App_RegisterCallbackFunctions()***“function.

4.2.5 Setting the Device Software Function

For many queries, the TPS-1 must deliver the application software (e.g. I&MO) version. For setting this version, constants are defined in the **TPS_1_user.h** file.

Name of the constant	Definition
SOFTWARE_REVISION_PREFIX	V
SOFTWARE_REVISION_FUNCTIONAL_ENHANCEMENT	10
SOFTWARE_REVISION_BUGFIX	1
SOFTWARE_REVISION_INTERNAL_CHANGE	20

Table 4-2: Constants for the software version

Note: This is **not** the TPS-1 Stack version. For the certification, „V“ must be entered as SOFTWARE REVISION PREFIX. These constants are used in the ***App_ConfigDevice()*** and ***App_InitIMData()*** functions. Both points must match for successful certification.

4.2.6 Device Start (TPS_StartDevice())

The „***TPS_StartDevice()***“ function signals to the TPS-1 that all configurations have been carried out and the data can be checked. The TPS-1 checks the data in the NRT area and then goes into operation. From this point onwards one can communicate with the field device via the PROFINET interface.

Note: If configuration problems occur, the debug information from the TPS-1 UART interface must be evaluated. For this, the debug version of the TPS-1 stack must be transferred into the TPS-1 Flash.

4.3 Communication TPS-1 and Host CPU (Event Communication)

Following the device start, the event registers of the TPS-1 are polled in a cyclic loop **(TPS_CheckEvents ()).** There are registers in the host CPU direction and back to the TPS-1 for confirmation and additional calls.

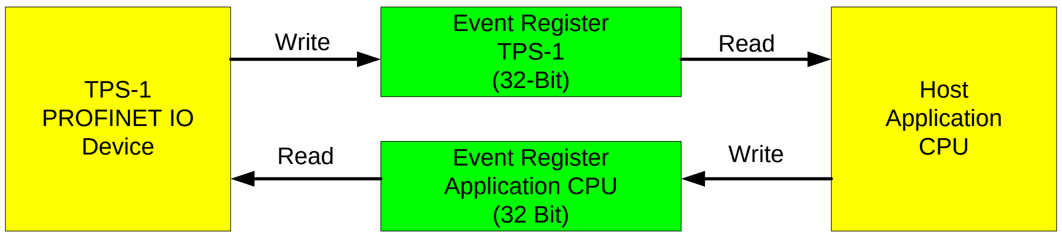


Figure 4-9: Event Communication TPS-1 - Host Application

The list of occurring events is available in the TPS-1 User Manual [1]. Events must always be reset after the processing so that then a new event can be displayed here.

An incoming Connect.Req would, for example, trigger the AR0 event „EVENT_ONCONNECT_REQ_REC_0“.

In the sample application, the function TPS_CheckEvents() is called cyclically (polling). It is also possible to configure certain events in such a way that their occurrence triggers an interrupt (INT_OUT, Pin K11). One or more events can be selected as interrupt triggers.

5. Identification & Maintenance Functions (I&M)

The functions I&M serve to read the basic information from a field device. The information is defined in the data structures in the specification and is divided into I&M0 to I&M4. The data is read via Record-Data-Read and written with Record-Data-Write.

I&M Parameter	Status	Index	Access
I&M0	mandatory [1]	0xAFF0	R
I&M1	optional [2]	0xAFF1	R/W
I&M2	optional [2]	0xAFF2	R/W
I&M3	optional [2]	0xAFF3	R/W
I&M4	optional [2]	0xAFF4	R/W
I&M0FilterData	mandatory [1]	0xF840	R

Note 1: Each PROFINET device must provide at least one submodule as device representative which supports I&MO functions.

Note 2: Mandatory, read- and writeable, for at least one subslot, e.g. interface submodule

Table 5-1: I&M data

The individual parameters can be found in the corresponding PROFINET specification.

Note: It must be possible to write data into IM1 to IM3 on a carrier, if this IM is available. The application must retain the data.

5.1 Assignment of the I&M Data

I&M data are assigned to each submodule. It is not necessary to set-up individual I&M data for every submodule. For example, a data set may exist for the DAP. All other modules then possess a pointer pointing to these data (carrier) as illustrated in Figure 5-1.

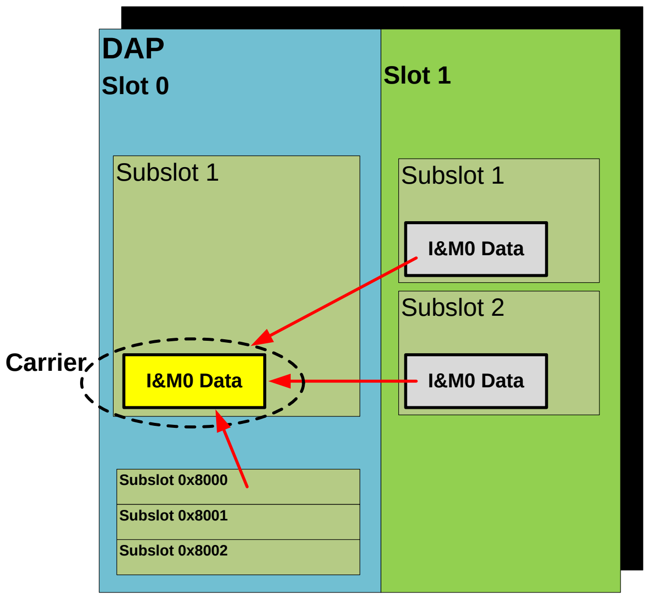


Figure 5-1: I&M data in a compact field device

The TPS driver provides the TPS_PlugSubmodule function for plugging-in a submodule.

```
SUBSLOT* TPS_PlugSubmodule(SLOT* pzSlotHandle,
USIGN16     wSubslotNumber,
USIGN32     dwSubmoduleIdentNumber,
USIGN16     wInitParameterSize,
USIGN16     wNumberOfChannelDiag,
USIGN16     wSizeOfInputData,
USIGN16     wSizeOfOutputData,
T_IM0_DATA* pzIM0Data,
BOOL        bIM0Carrier)
```

Figure 5-2: TPS_Plug_Submodule function

The parameter **„BOOL bIMOCarrier“** specifies whether the pointer **„pzIM0Data“** is a carrier module (TPS_TRUE) or is it a reference to the carrier (TPS_FALSE).

Each submodule can receive its own I&M data. Especially in modular field devices, module-specific I&M data exists. Figure 21 describes the relationship in detail.

Individual modules possess e.g. an order number and a dedicated description. If required, the operator can identify these easily.

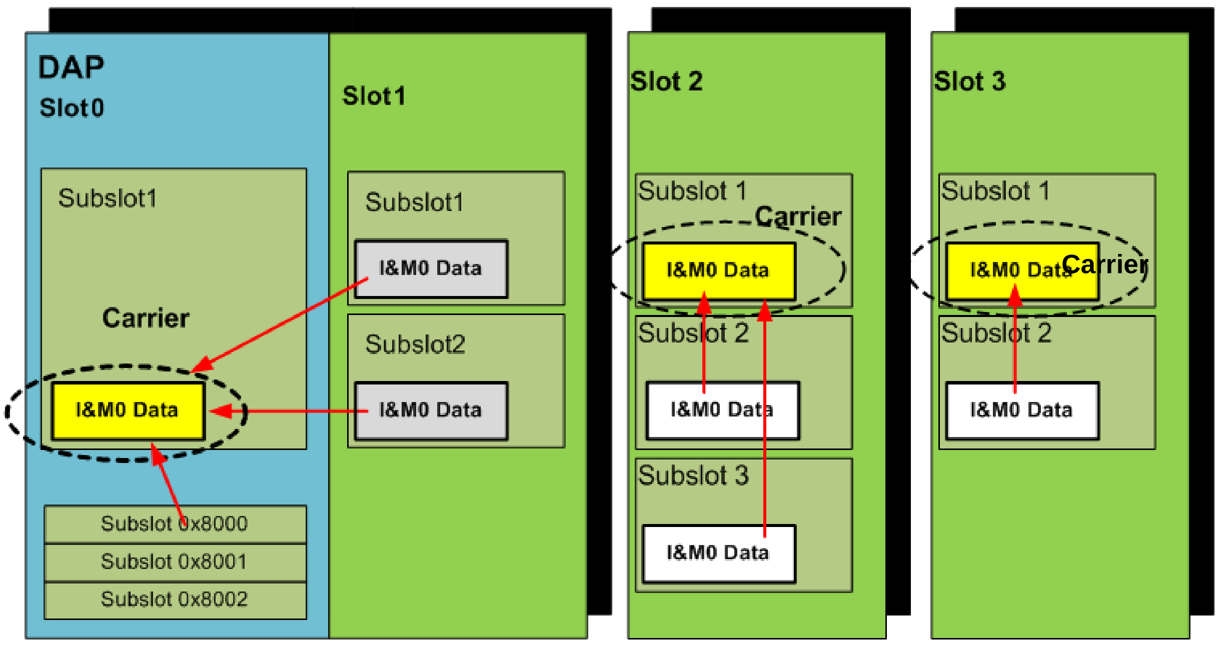
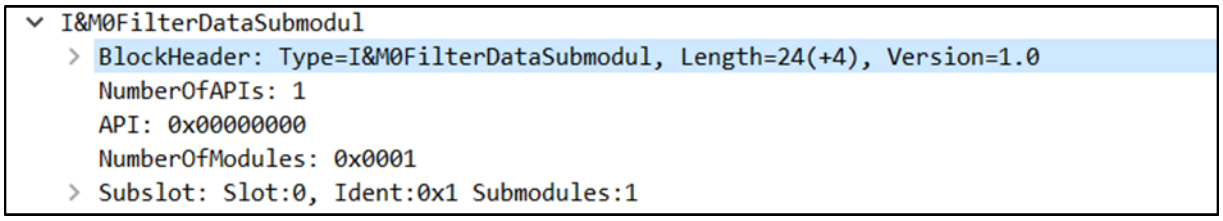


Figure 5-3: I&M-Data for modular field devices

Care must be taken during implementation that all submodules deliver the corresponding data.

5.2 Using I&M Filter Data (Index 0xF840)

The TPS-1 stack answers the index 0xF840. If you are requesting this information, the TPS-1 provides all submodules that are carrier for I&M data.

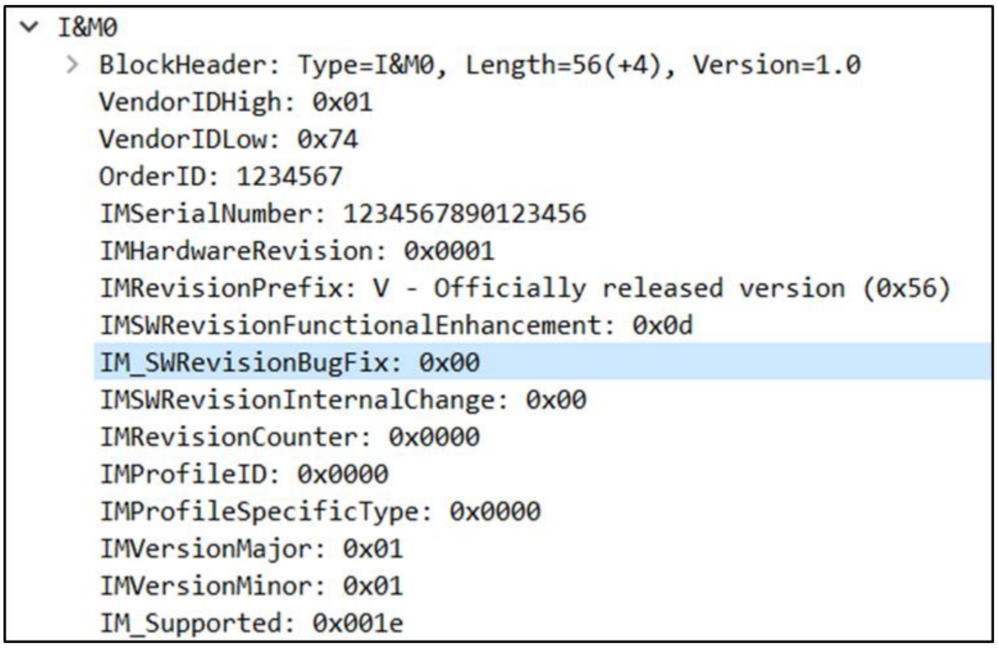


```
I&M0FilterDataSubmodul
    BlockHeader: Type=I&M0FilterDataSubmodul, Length=24(+4), Version=1.0
    NumberOfAPIs: 1
    API: 0x00000000
    NumberOfModules: 0x0001
    Subslot: Slot:0, Ident:0x1 Submodules:1
```

Figure 5-4: I&M0FilterDataSubmodul

In a second step the controller asks for I&M0 data (index 0xAFF0). Inside I&M0 data you get the information IM_Supported. This variable decode the available I&M Data as shown in Figure 5-5.

It is necessary to register the supported I&M data for each carrier. For registration the function **TPS_RegisterIMDataForSubslots()** (please refer the reference manual) must be called.



```
I&M0
    BlockHeader: Type=I&M0, Length=56(+4), Version=1.0
    VendorIDHigh: 0x01
    VendorIDLow: 0x74
    OrderID: 1234567
    IMSerialNumber: 1234567890123456
    IMHardwareRevision: 0x0001
    IMRevisionPrefix: V - Officially released version (0x56)
    IMSWRevisionFunctionalEnhancement: 0x0d
    IM_SWRevisionBugFix: 0x00
    IMSWRevisionInternalChange: 0x00
    IMRevisionCounter: 0x0000
    IMProfileID: 0x0000
    IMProfileSpecificType: 0x0000
    IMVersionMajor: 0x01
    IMVersionMinor: 0x01
    IM_Supported: 0x001e
```

Figure 5-5: I&M0 data with IM_Supported structure

5.3 Initialisation of I&M Data (Device Start-up)

In the application example, I&M data is created with the **„App_InitIMData()“** function. This function creates a complete data set (IM0 – IM4) in the example. Depending upon the field device, the manufacturer must then specify which data he wishes to deliver (this must match the GSDML tag **“Writeable_IM_Records”**).

The function **„TPS_RegisterIMDataForSubslot()“** assigns a created data set to a submodule. This function is called by the function **“TPS_PlugSubmodule()”**, e.g. during the start up.

While writing I&M data (I&M1…4), the application must ensure that the necessary data are retained. I&M Write data are received in the "***AppWriteIMData ()***" function. If necessary, a routine must be built-in to ensure the data retention.

The function **“onImDataChanged()”** (registered callback function) must store new I&M data to your permanent memory (e.g. Flash memory).

Note: If I&M1-4 are supported it is necessary to store and restore data during start up out of your permanent memory. The structures g_pIM1_Data, g_pIM2_Data, g_pIM3_Data, g_pIM4_Data must be filled. In the extended example you find a solution for restoring I&M data (**restoreImFromFlash()**).

6. Establishing a Connection Between IO Controller and IO Device

The first step for establishing a connection between an IO Controller and an IO Device is to search for an IO Device. In PROFINET networks, specific devices are not searched by the IP address; rather each device is assigned a **„NameOfStation“** which serves as a unique device reference.

6.1 Searching the Device

The search for a specific device name is done with the DCP protocol. The search is triggered by sending the PNO multicast address (01:0E:CF:00:00:00), to which only PROFINET devices react.

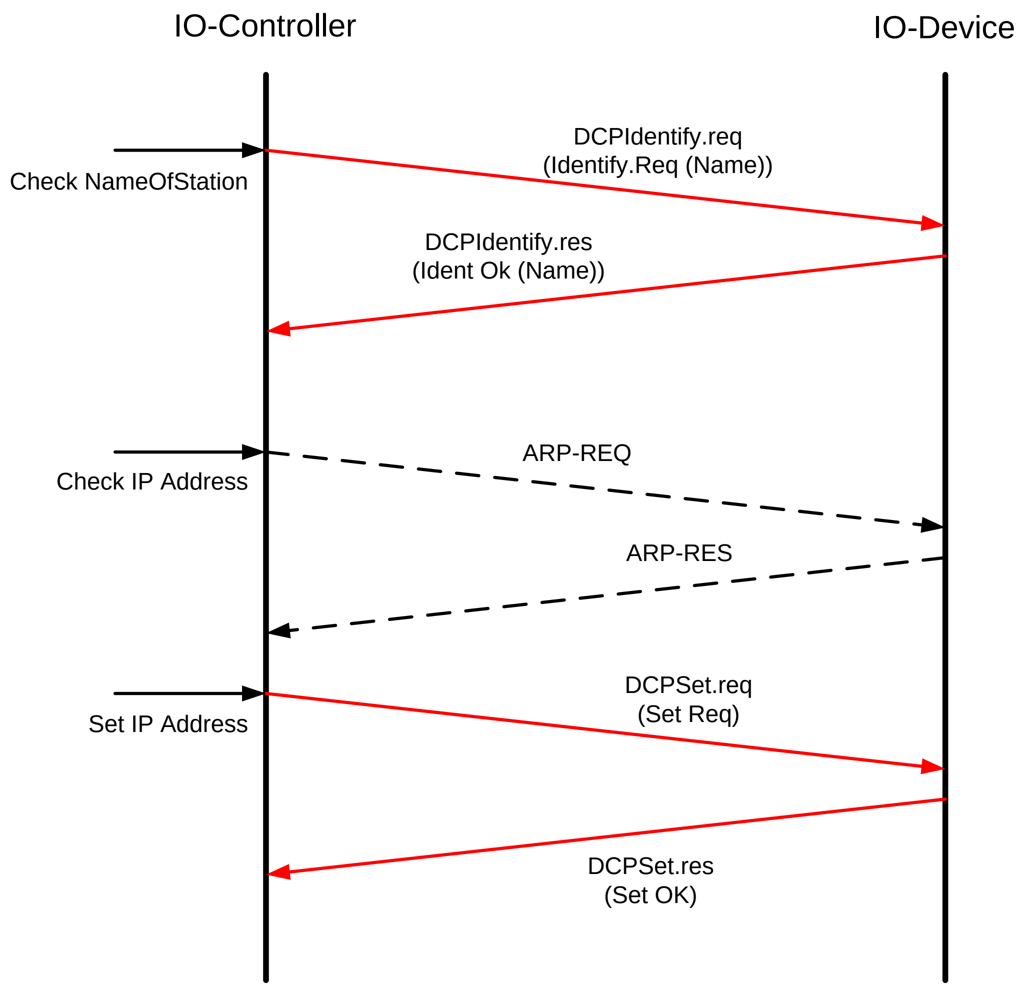


Figure 6-1: Identify Request

If required, the DCPSet.req assigns a new IP address to the IO Device. This address is derived from the configured data. Figure 6-1 illustrates the process.

6.2 Connection Set Up

After addressing the IO Device, the IO Controller can establish a connection. In general, an IO Controller builds a separate connection with each participating IO Device. A description of the individual steps can e.g. be found in [9].

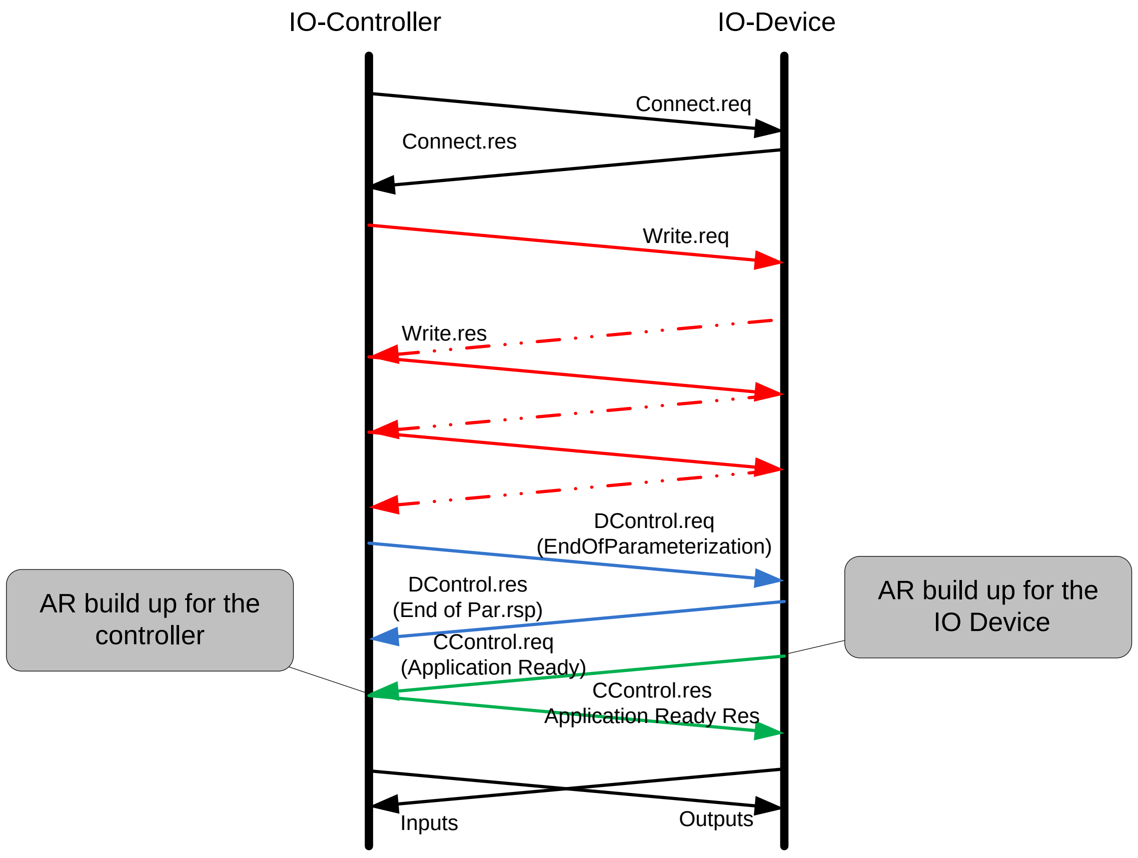


Figure 6-2: Connection set up

As soon as the IO Controller has finished writing all initialisation parameters (red in Figure 6-2), a Control.req (ParameterEnd) is sent (blue). This event is forwarded to the application. The parameters sent previously are associated with the data supporting subslots. The customer implementation must now process this data before the Control.req sends a Ready to the IO Controller (green), and a connection is established (before sending the Application Ready the consumer and the provider status must be set-up correctly).

7. Acyclic Data Exchange via Record Data

PROFINET offers certain options so that besides the cyclic data traffic for transferring the process data, it is also possible to exchange data (e.g. read and write parameter) non-cyclically between the initiator and the responder via the „Record Data Communication Relation."

Acyclic data do not have high priority and are only sent if required, using the RPC-Protocol. The acyclic services can be executed by the controller as well as by the supervisor.

Write.Requests are allowed only within an established CR. Read.Requests can also be executed without establishing a connection in advance as it does not affect the process.

Chapter 7.2 provides an overview of frequently used PROFINET records. Some of these records must be processed with a host application.

Note: The record data exchange described here refers only to the TPS-1 with an attached host CPU. In the local IO mode, the TPS-1 responds to all record queries.

7.1 General Procedure for Record Data Exchange

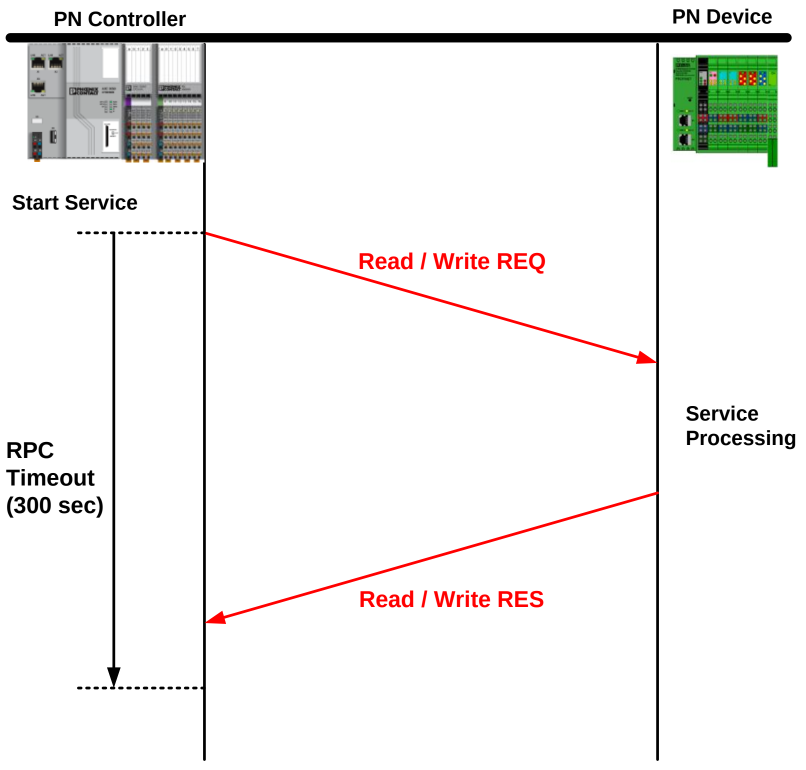


Figure 7-1: Record-Data Request / Response

A controller sends e.g. a RecordRead.Request, and after processing, receives a RecordRead.Response back - either from the TPS-1 firmware or from the device application.

The corresponding event is then set in the event register via the event interface. After that, the corresponding processing routine is called via the ***TPS_CheckEvents()*** function. Table 7-1 shows some typical events for the handling of acyclic data.

Event	Meaning:
TPS_EVENT_ONREADRECORD	A RecordRead.REQ has arrived. This event is only set for requests which cannot be answered by the TPS firmware.
TPS_EVENT_ONWRITERECORD	A RecordWrite.REQ has arrived. This event is only set for requests which cannot be answered by the TPS firmware.
APP_EVENT_RECORD_DONE	The application has processed a Record.Request; the answer is available in the mailbox and can be returned by the TPS-1.

Table 7-1: Acyclic events

The incoming requests are processed in two steps. Some requests are already implemented in the DRIVER (e.g. Write-IM1-0xAFF1, etc.). These will be answered by the DRIVER without own implementation in the application. A Callback function is registered in the default branch of a Switch statement; in the case of another request, this function will be executed.

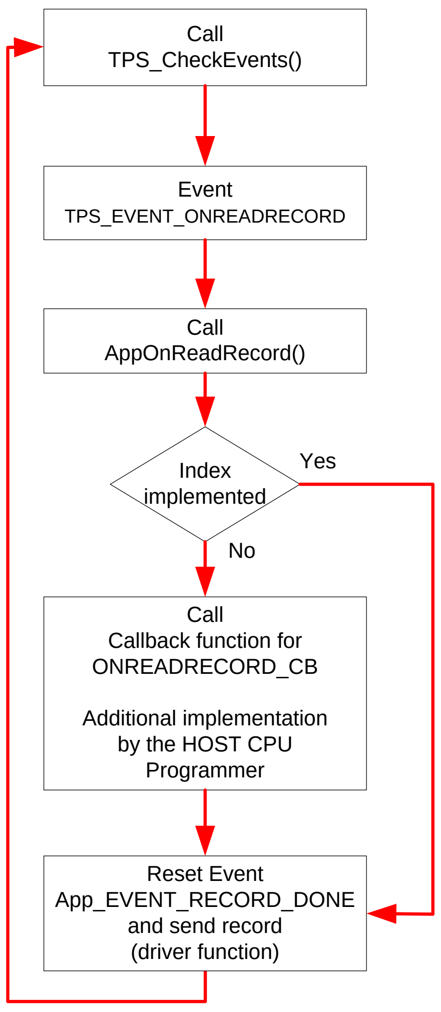


Figure 7-2: Acyclic data exchange flow for “Read.Request”

In this Callback function, the index to be processed must be added to the customer implementation, and the implementation must then be saved.

The same applies to the Write-Request. The flow is identical. Taking the indices 0x8028 and 0x8029 as an example, chapter 7.2 describes the process in detail.

7.2 Processing of Indices 0x8028 and 0x8029

By polling the indices "Record Input Data Object Element" (0x8028) and "Record Output Data Object Element" (0x8029), a controller or a supervisor can check the current input and output data without affecting the cyclic messages.

The event „TPS_EVENT_ONREADRECORD“ indicates that a „ReadRecordRequest“ has occurred. Then, within the TPS_CheckEvent() function, the AppOnReadRecord() function (in file TPS_1_API.c) is called.

```
AppOnReadRecord()

{
       switch(mailBoxInfo.wIndex)
           case RECORD_INDEX_RECORD_INPUT_DATA_OBJECT:
                   AppReadRecordDataObject(..);
       break;
           case RECORD_INDEX_RECORD_OUTPUT_DATA_OBJECT:
                 AppReadRecordDataObject(..);
             break;
             default:
                   break;
}
```

Figure 7-3: Function body AppOnReadRecord()

The AppOnReadRecordDataObject() function must be adapted for the planned application. The function then provides the subslot data.

Note: This adaptation is mandatory for each field device.

7.3 PROFINET IO Record overview (Selection)

PROFINET Index	**Description**	**Information**	**Used by: 1) TPS-1 Stack 2) Application**	**Suitable submodules**
0x0000 - 0x7FFF	user specific	definded by user	2)	
0x2000 - 0x20FF	**reserved by TPS-1**	**Do not use!**	1), 2)	
0x8000	ExpectedIdentification Data	This returns the current expected configuration for the specified connection.	1)	All
0x8001	RealIdentificationData for one Subslot	This returns the currend expected configuration for the current device.	1)	All
0x800C	Diagnosis, Maintenance, Qualified and Status for one Subslot		1)	All
0x8020	PDIRSubframeData for one Subslot			
0x8027	PDPortDataRealExtended for one Subslot.	Mandatory for port submodule in version 2.32 (2.31 optional)	1)	
0x8028	RecordInputDataObject Element	This record returns the input data object for one subslot.	2)	All
0x8029	RecordOutputDataObject Element	This record returns the output data for one subslot.	2)	All
0x802A	PDPortDataReal for one Subslot	Reading the LLDP information of one physical subslot via record data.	1)	Port
0x802B	PDPortDataCheck for one subslot	Reading Port Data, e.g. Mau Type, Link State, etc.	1)	Port
0x802C	PDIRData for one subslot	Forwarding information for IRT data.		
0x802D	PDSyncData for one subslot with SyncID value 0			
0x802E	Reserved (legacy)			
0x802F	PDPortDataAdjust	Adjustment of port parameter (e.g. MAU-type)	1)	Port
0x8030	IsochronousModeData for one Subslot		1), 2)	Slot

PROFINET Index	**Description**	**Information**	**Used by: 1) TPS-1 Stack 2) Application**	**Suitable submodules**
0xAFF0	I&M0 (mandatory) read only	Return of information - the structure is defined in the specification.	2)	All
0xAFF1	I&M1 (optional) read/write	Return of information - the structure is defined in the specification.	2)	All
0xAFF2	I&M2 (optional) read/write	Return of information - the structure is defined in the specification.	2)	All
0xAFF3	I&M3 (optional)	Return of information - the structure is defined in the specification.	2)	All
0xAFF4	I&M4 (optional)	Return of information - the structure is defined in the specification.	2)	All
0xE040	WriteMultiple	In the start-up phase, several blocks of initial parameters are transferred in a Write-Record to a device.		
0xF840	I&M0FilterData	This query provides the carrier containing I&M0 data. To determine the supported I&M, a query about the I&M0 data of the carrier must be made.	1), 2)	All
0xF841	PDRealData	Information data of all PDV submodules	1)	
0xFBFF	Trigger Index	Trigger index for the RPC connection inside the CMSM		

Table 7-2: Table PROFINET IO Records

8. Cyclic Data Exchange

With the help of the GSDML file, the controller determines the structure of the cyclic data output (Output) and data input (Input). Using the **Connect.Req**, the controller informs the device how this structure looks like. The device must adapt itself to these requirements.

If a device is not in possession of the expected module or submodule, then the „TPS-1“ generates a ModuleDiffBlock, which is attached to the **Connect.Req.** If needed, the controller can operate the connection partially with it.

Figure 8-1 illustrates the basic sequence of data exchange. Messages are sent in each direction and stored in the corresponding data buffers. The respective devices then fetch the data from or write into the data buffer. By default, data exchange follows the Consumer-Provider model.

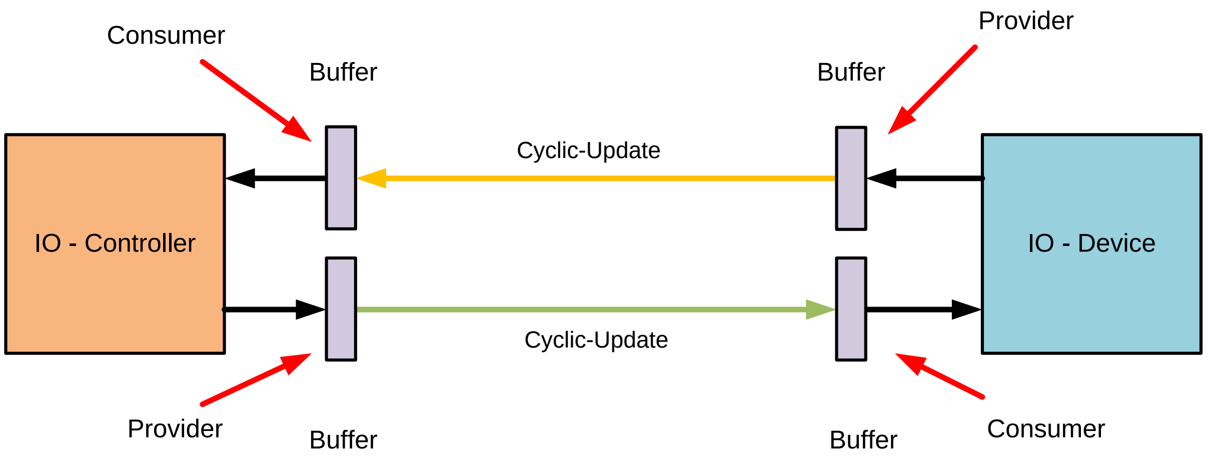


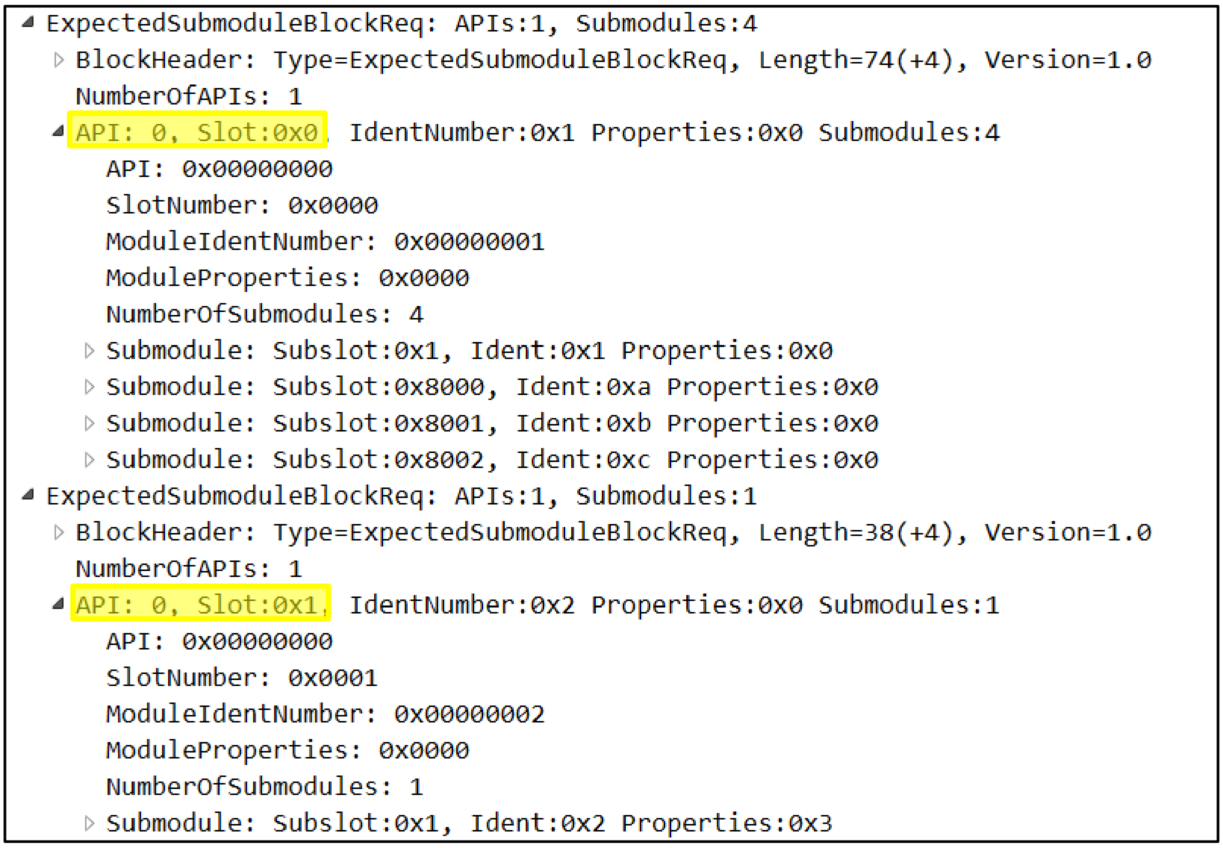
Figure 8-1: Cyclic data exchange Controller - Device

8.1 Connect Request by the Controller

The **Connect.Req** represents the central entry point in the cyclic data exchange. As soon as the device has completed its ramp-up, the controller can connect to it. The Connect.Req provides the expected configuration to the device (ExpectedSubmoduleBlockReq). The device compares it with its slot, subslot configuration and sends back a Connect.Res to the controller (see chapter 6.2).

After the **Connect.Req**, the initial parameters (Write.Request) are sent, and at the end the device reports „ApplicationReady“ and starts with cyclic data exchange.

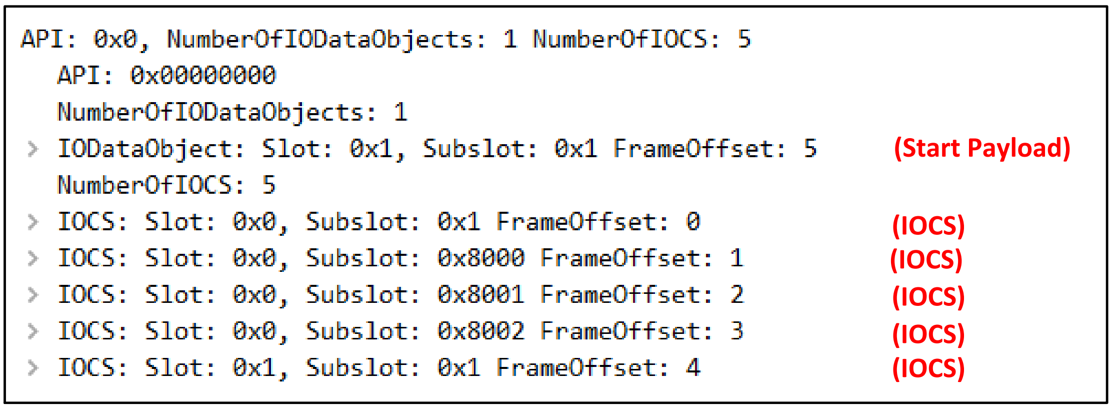
Figure 8-2 shows a part of a **Connect.Req**, where the expected configuration is described. The module provides only one slot and one subslot each for the data exchange.



```
◢ ExpectedSubmoduleBlockReq: APIs:1, Submodules:4
  ▷ BlockHeader: Type=ExpectedSubmoduleBlockReq, Length=74(+4), Version=1.0
    NumberOfAPIs: 1
  ◢ API: 0, Slot:0x0, IdentNumber:0x1 Properties:0x0 Submodules:4
      API: 0x00000000
      SlotNumber: 0x0000
      ModuleIdentNumber: 0x00000001
      ModuleProperties: 0x0000
      NumberOfSubmodules: 4
    ▷ Submodule: Subslot:0x1, Ident:0x1 Properties:0x0
    ▷ Submodule: Subslot:0x8000, Ident:0xa Properties:0x0
    ▷ Submodule: Subslot:0x8001, Ident:0xb Properties:0x0
    ▷ Submodule: Subslot:0x8002, Ident:0xc Properties:0x0
◢ ExpectedSubmoduleBlockReq: APIs:1, Submodules:1
  ▷ BlockHeader: Type=ExpectedSubmoduleBlockReq, Length=38(+4), Version=1.0
    NumberOfAPIs: 1
  ◢ API: 0, Slot:0x1, IdentNumber:0x2 Properties:0x0 Submodules:1
      API: 0x00000000
      SlotNumber: 0x0001
      ModuleIdentNumber: 0x00000002
      ModuleProperties: 0x0000
      NumberOfSubmodules: 1
    ▷ Submodule: Subslot:0x1, Ident:0x2 Properties:0x3
```

Figure 8-2: Expected Configuration block in a Connect.Req

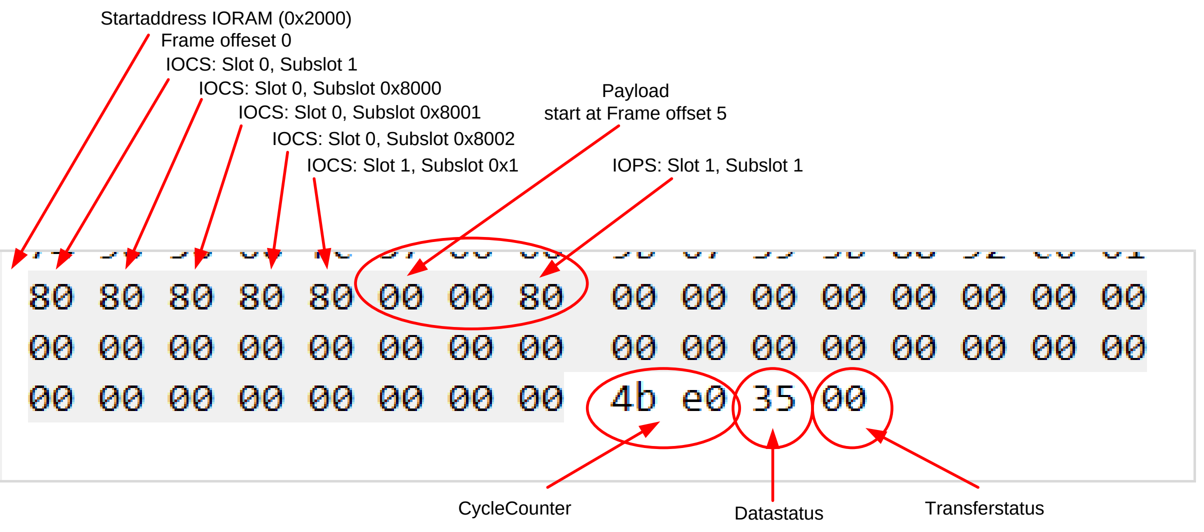
Within the Connect.Req, the controller also specifies where the subslots, as well as the provider and consumer status related data, are located. The order may differ for each controller. For accessing these data, pointers within the subslot data must be used. These pointers are managed by the TPS-1 and provide the correct position for the output and input data (the offset is calculated from the beginning of the IO RAM - 0x2000).



```
API: 0x0, NumberOfIODataObjects: 1 NumberOfIOCS: 5
   API: 0x00000000
   NumberOfIODataObjects: 1
 > IODataObject: Slot: 0x1, Subslot: 0x1 FrameOffset: 5      (Start Payload)
   NumberOfIOCS: 5
 > IOCS: Slot: 0x0, Subslot: 0x1 FrameOffset: 0              (IOCS)
 > IOCS: Slot: 0x0, Subslot: 0x8000 FrameOffset: 1           (IOCS)
 > IOCS: Slot: 0x0, Subslot: 0x8001 FrameOffset: 2           (IOCS)
 > IOCS: Slot: 0x0, Subslot: 0x8002 FrameOffset: 3           (IOCS)
 > IOCS: Slot: 0x1, Subslot: 0x1 FrameOffset: 4              (IOCS)
```

Figure 8-3: Position of the cyclic data in IO RAM

The data position shown in **Figure 8-3** is illustrated in Figure 8-4 the message form as one would find it in the memory.


Startaddress IORAM (0x2000)
Frame offeset 0
IOCS: Slot 0, Subslot 1
IOCS: Slot 0, Subslot 0x8000
IOCS: Slot 0, Subslot 0x8001
IOCS: Slot 0, Subslot 0x8002
IOCS: Slot 1, Subslot 0x1
Payload
start at Frame offset 5
IOPS: Slot 1, Subslot 1
CycleCounter
Datastatus
Transferstatus

```
80 80 80 80 80 00 00 80  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  4b e0 35 00
```

Figure 8-4: Output data representation in IO RAM

The TPS-1 memorizes the order of the data and places it as per controller instruction e.g. in the receive buffer. The host CPU can now access this data.

8.2 Data Access to Receive and Send Buffers

For cyclic data exchange (IO data), the TPS-1 provides a triple buffer mechanism in receive and send direction. This ensures that a new buffer is always available for data processing.

The processing cycle begins with fetching the current data buffer (***TPS_UpdateOutputData(bActiveIOAR)***). By calling this function, the current output buffer becomes available. The data are displayed in the IO RAM and the output data can be processed.

The application compiles the input data and places it in the input buffer using the ***TPS_WriteInputData(pzSubmodule, g_byIOData, wSizeInputData, IOXS_GOOD)*** function. At the same time, the provider status for the submodule (***pzSubmodule)*** processed here is set. If needed, this process must be executed for each of the plugged-in submodules. After the input data have been placed in the IORAM, the ***TPS_SetOutputIocs***() function is called to set the consumer status.

With this, all data are available in the IORAM (Input), and the buffer can be submitted to TPS-1 for sending with ***(TPS_UpdateInputData(bActiveIOAR))***.

This completes the transfer of cyclic data.

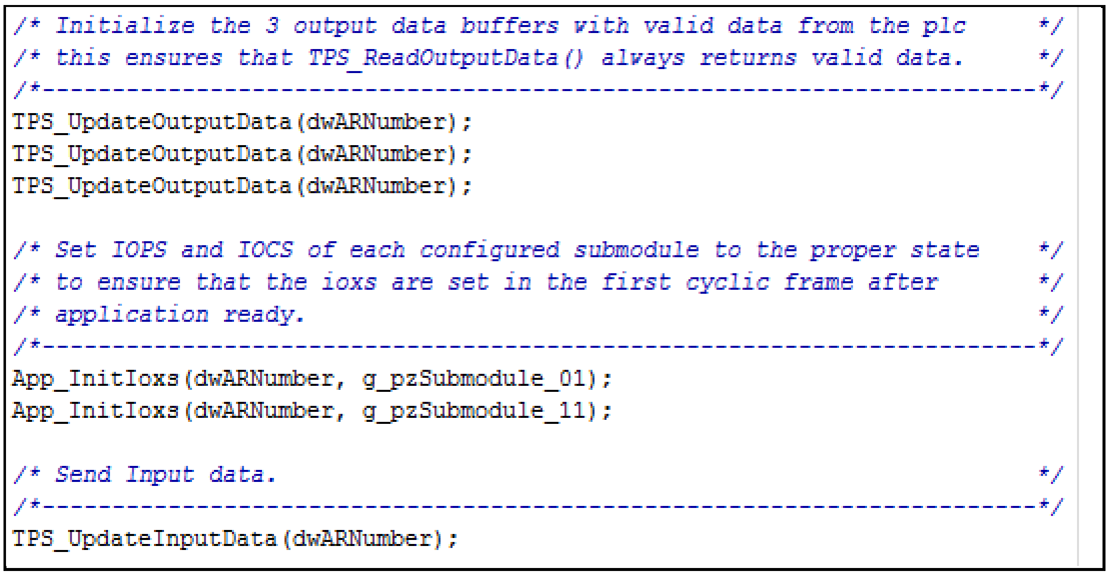
8.3 Provider and Consumer Status

Each submodule receives (if configured in the GSDML) and sends cyclic data. For received data, the provider status (IOXS) is checked and for the input data, the consumer status is set.

While sending the Application Ready event, it must be ensured that the provider and consumer status are set correctly. In the Callback function ***App_OnPrmEndCallback()***, the output buffer is updated three times at the end of the function. The result is that all the buffers contain the correct data and controller status.

Also, for all subslots with data, a provider status initialisation must be carried out (***App_InitIoxs()***).In the sample code it is necessary to do this for slot 0, subslot 1 and slot 1, subslot 1.

This completes the start-up initialisation of the IORAM. The last step is to send the cyclic frame to the controller.



```
/* Initialize the 3 output data buffers with valid data from the plc    */
/* this ensures that TPS_ReadOutputData() always returns valid data.    */
/*--------------------------------------------------------------------*/
TPS_UpdateOutputData(dwARNumber);
TPS_UpdateOutputData(dwARNumber);
TPS_UpdateOutputData(dwARNumber);

/* Set IOPS and IOCS of each configured submodule to the proper state    */
/* to ensure that the ioxs are set in the first cyclic frame after       */
/* application ready.                                                    */
/*--------------------------------------------------------------------*/
App_InitIoxs(dwARNumber, g_pzSubmodule_01);
App_InitIoxs(dwARNumber, g_pzSubmodule_11);

/* Send Input data.                                                      */
/*--------------------------------------------------------------------*/
TPS_UpdateInputData(dwARNumber);
```

Figure 8-5: Initialization out of function App_OnPrmEndCallback()

Inside the function App_On ConnectCallback() the module and submodule state must be set to MODULE_OK (function TPS SetModuleState() and TPS SetSubmoduleState()).

8.4 Providing Initial Parameters for the Field Device

After the controller has received the Connect.Res from the field device, it starts to send initial parameters to the field device (Write.Req). The initial parameters are based on what is specified in the GSDML file.

```
<ParameterRecordDataItem Index="1234" Length="2">

<Name TextId="TOK_ExampleParameter" />

<Const Data="0xAB,0xCD" />

</ParameterRecordDataItem>
```

Figure 8-6: Initial parameter set for two bytes

The length of the initial parameters (**wInitParameterSize**) must be specified when the subslot is plugged-in. It is a Header (six bytes) plus the actual number of bytes. With the parameter length entry, space within the subslot data is reserved. During parameter transfer, the TPS-1 automatically enters these into the free space. This structure can be accessed with the pointer ***„pt_init_records"***.

8.5 Query on new cyclic output data

The GSD file specifies the period in which the cyclic data sent to a device. The function ***TPS_UpdateOutputData(bActiveIOAR)*** enables the current buffer of the Three-Buffer-Unit of IO-RAM. However, this is not an indication that the data is new. The register PN_EVENT_LOW (0x003C) includes two bits (bit 7 and bit 8) that indicate that new data have come from the controller.

Name	**PN_EVENT_LOW**	
Address	0x003C	
Bits	**Name**	**Description**
31:0	**Event bits**	
	Bit 7	Receive new data on AR1 (set to „1")
	Bit 8	Receive new data on AR0 (set to „1")

Table 8-1: TPS-1 PN_EVENT_LOW register

After receiving and processing the particular bit, it must be set back again by setting a correlated acknowledge bit in the register HOST_IRQ_ACK_LOW.

Name	**HOST_IRQ_ACK_LOW**	
Address	0x0020	
Bits	**Name**	**Description**
31:0	**Acknowledge bits**	
	Bit 7	„0“ event bit is not set back „1“ event bit is set back
	Bit 8	

Table 8-2: TPS-1HOST_IRQ_ACK_LOW register

The sample code below shows the query for AR0 and reset of the event bit by writing the register ***HOST_IRQ_ACK_LOW.***

```
/* Read PN_EVENT_LOW                                                   */
/*---------------------------------------------------------------------*/
TPS_GetValue32((USIGN8*)PN_EVENT_LOW, &dwEventLow);
/* Mask bit 7 (AR1) and bit 8 (AR0)                                    */
/*---------------------------------------------------------------------*/
dwEventLow &= 0x0180;
/* Bit 8 show a message for AR0                                        */
/*---------------------------------------------------------------------*/
if (dwEventLow == 0x0100)
{   printf("DEBUG_API > API: PN_EVENT_LOW %x\n",dwEventLow);
    /* Acknowledge AR0                                                 */
    /*-----------------------------------------------------------------*/
    TPS_SetValue32((USIGN8*)HOST_IRQ_ACK_LOW, dwEventLow);
}
```

Figure 8-7: Program code for requesting new cyclic data

9. IRT Communication and IRT Application

9.1 IRT Communication

The term isochronous (Greek: isokhronos, from isos = 'equal' + khronos = 'time') is used when an operation repeats in precisely equal time intervals. In PROFINET, the isochronous behaviour of cycles is produced by the highly accurate (deviations less than 1 µs) synchronisation between the devices. This allows reading the input signals and activating the output signals at the same time within a system.

In the PROFINET context, the isochronous behaviour may refer to both the communication itself as well as the application. Thereby, the communication can be isochronous without the application being isochronous, but not vice versa. Isochronous applications always require isochronous communication.

For the isochronous communication, the bandwidth of the Ethernet communication is divided into two time slices (see Figure 9-1). One, the so-called RED-phase in which only IRT data packets (PROFINET RTC3 telegrams) on the network are allowed; and the second, the GREEN phase in which the normal Ethernet communication, as well as normal, not isochronous PROFINET communication are allowed. By dividing the bandwidth, a deterministic communication is achieved, i.e. the time of arrival of the IRT telegrams (RTC 3 data packets) is always exactly predictable. This is a prerequisite for high-precision isochronous drive control systems, as e.g. required by the newspaper printing machines.

Here, the isochronous cycle and the beginning of the RED-phase are synchronized with high precision in under less than 1 microsecond. Furthermore, in the RED phase, the individual packets are matched exactly with each other. The beginning of the RED-phase may also trigger local events (interrupts).

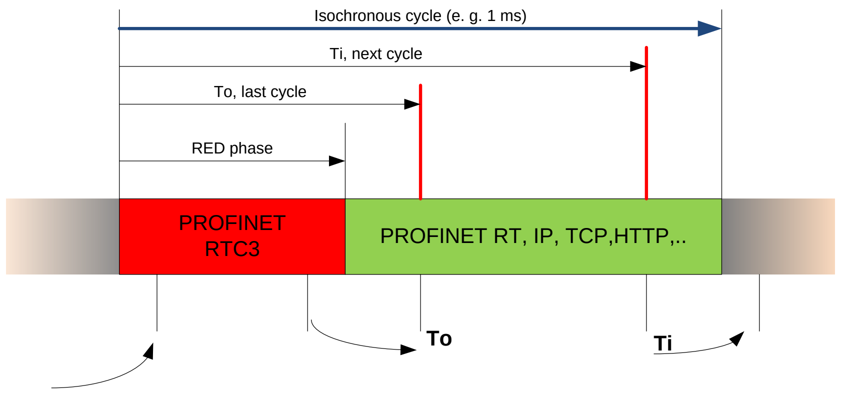


Figure 9-1: IRT cycle display

9.2 Isochronous Application

In an isochronous application, two signals relative to beginning of the RED phase are provided:

- **Ti**: At this time, all devices involved in the isochronous operation must back up their inputs and provide these for the transfer
- **To**: At this time, all devices involved in the isochronous operation must write their output data.

Since the timings are defined relative to the start of the RED-phase, the isochronous communication can be used to exchange the input data secured at time Ti and the output data required at time To in time between the PROFINET field device and the central control (SPS, PC, PAC, ..) with very low fluctuation (jitter). Also, the processing of the data in the controller is synchronous to this cycle.

As a result, a closed control loop can be built in a single cycle (e.g. 250 µs).

9.3 IRT Applications With the TPS-1

The TPS-1 provides various signals for operation in an IRT domain which allow an easy realisation of an isochronous application.

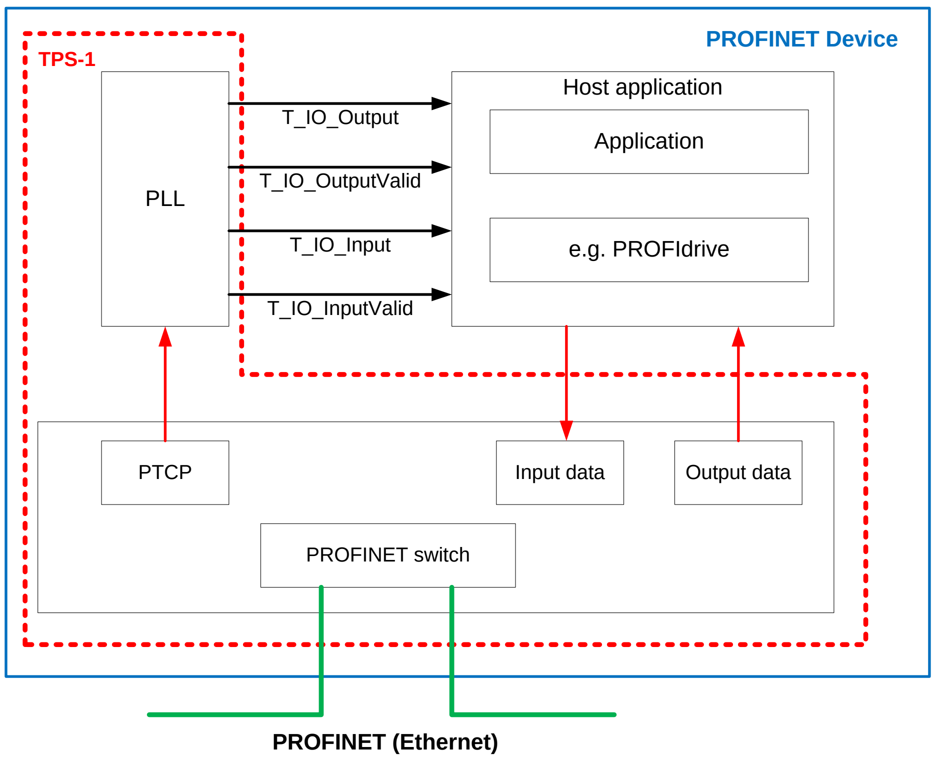


Figure 9-2: IRT-signals of TPS-1

For a typical IRT application, the **T_IO_Output** (To, T2) and **T_IO_Input** (Ti, T1) signals are required. The internal PLL of the TPS-1 synchronises itself to the TEST_SYNC (Tsync) signal which is used to indicate the beginning of a new isochronous cycle.

The outputs T1 and T2 are connected to the interrupt inputs of the host CPU and trigger the processing of input and output data.

TPS-1 Pin	Application/Use
T1 (J11)	Ti (T_IO_Input)
T2 (H11)	To (T_IO_Output)
T3 (G11)	T_IO_InputValid
T4 (F11)	T_IO_OutputValid
TEST_SYNC (N12)	Start of bus cycle

Table 9-1: IRT Communication Signals

Figure 9-3 shows the location of the Ti and To signals relative to the frame synchronizing signal Tsync.

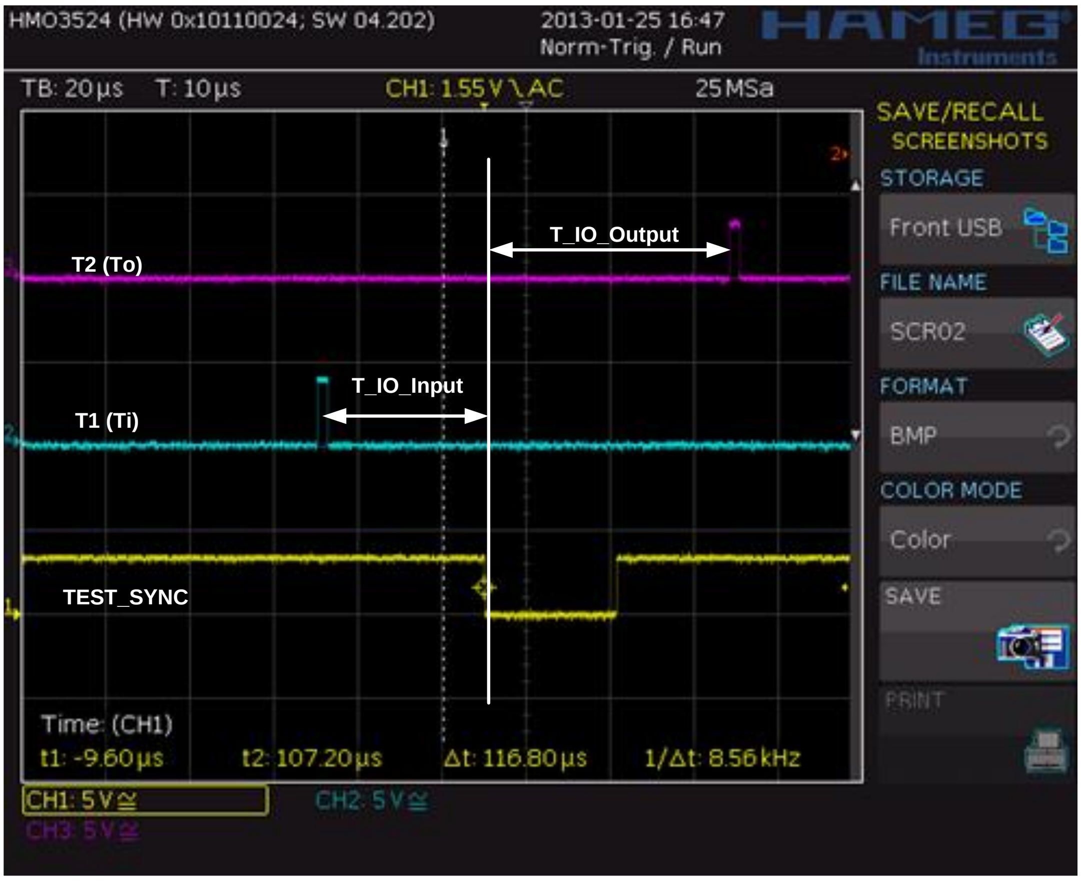


Figure 9-3: Ti and To signals display

9.4 IRT Keywords in the GSDML File

The „Isochronous mode“ is optional; a field device supporting this mode must have the following entry in the GSDML for a module:

```
<IsochroneMode       T_DC_Base="4"
                     T_DC_Min="1"
                     T_DC_Max="8"
                     T_IO_Base="1000"
                     T_IO_InputMin="40"
                     T_IO_OutputMin="40"
                     IsochroneModeRequired="true" />
```

Figure 9-4: Isochronous mode entry for a GSDML

The following table describes the GSDML attributes which must be set-up in the GSDML of a module. The individual attributes must be determined by the device manufacturer.

GSDML Keywordt	Description
T_DC_Base	T_DC time Base (Data exchange cycle time) in 31.25 µs units.
T_DC_Min	Minimum T_DC time in T_DC _Base units.
T_DC_Max	Minimum T_DC time in T_DC _Base units.
T_IO_Base	Time base of T_IO_Input, T_IO_Output, T_IO_InputMin, T_IO_OutputMin in Nanoseconds.
T_IO_InputMin	Minimum time necessary to fetch the input and to update the input buffer (in T_IO_Base units).
T_IO_OutputMin	Minimum time necessary to fetch the output and to update the submodule (in T_IO_Base units).
IsochroneModeRequired	This attribute shows whether a field device or a submodule demands an IRT mode for the operation. RT_CLASS_3 must be used for this communication.

Table 9-2: Keywords for isochronous mode in the GSDML

10. ResetToFactory Settings

In PROFINET, two methods are provided for the reset of a device to the delivery state. These methods are determined by Suboption 5 (“FactoryReset”) and Suboption 6 (“ResetToFactory”).

You can filter reset messages with the wireshark filter:

Wireshark Filter: pn_dcp.block_qualifier_reset

10.1 Factory Reset

A “Factory Reset” shall set the following data:

- the NameOfStation to “”
- the IP Address, the subnet mask and the standard gateway to 0.0.0.0
- all other parameters to the manufacturer’s default value.

10.2 ResetToFactory Settings

Every PROFINET device must be capable of being set to conditions as supplied to customer (“Reset to factory settings”). This must be possible even during cyclic data exchange. This is necessary if you want to put a device into another factory automation machinery.

Typically, a DCP.Set service activates the “ResetToFactory settings”.

The TPS-1 example software supports the reset modes 2 and 8 (please refer the PROFINET standard regarding this topic). The TPS-1 Stack handles mode 2. An event informs the application about the ResetToFactory and its mode.

Reset option	Meaning
2	Mandatory – Reset Interface Reset communication parameters – this is done by the TPS-1 stack
8	Optional – Reset Device Reset all stored data in the IO-Device to its factory values. The I&M1 to I&M4 data must be handled by the application and the customer has to implement the desired functionality.

Figure 10-1: Reset options for ResetToFactory settings

You register the callback function „***onDCPResetToFactory()***“ (TPS_RegisterDCPCallbackPara()). The function **AppOnResetFactorySettings()** checks „**wResetOption**“. Option 2 is fulfilled by the TPS Stack itself. The TPS-1 example software supports only options 2 and 8. Function „***AppFactoryResetIM1_4()***” deletes the I&M data inside the application program (volatile). In a second step I&M data must be deleted inside the „nonvolatile memory”. The application developer must do this.

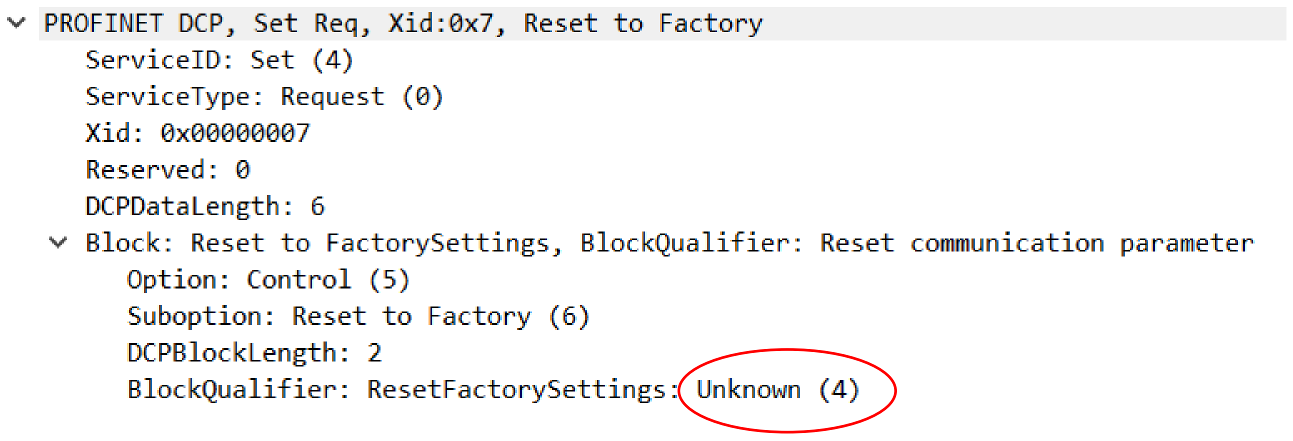
Our example uses a part of the TPS-1 flash (see function “***TPS_WriteUserDataToFlash()*** ”). The registered function “***onDCPResetToFactory()***” deletes I&M data in this flash array.

The supported ResetToFactory modes are listed in the GSDML file as “DeviceAccessPointItem” as shown in Figure 10-2.

```
<DeviceAccessPointItem
          ID="TPS-1 Template"
          .........
                    ResetToFactoryModes="2"
          .........
          ParameterizationSpeedupSupported="true">
```

Figure 10-2: Snip from GSDML file

The following Figure 10-3 shows a DCP request that calls Reset Option 2 (Reset all communication parameter).



```
PROFINET DCP, Set Req, Xid:0x7, Reset to Factory
   ServiceID: Set (4)
   ServiceType: Request (0)
   Xid: 0x00000007
   Reserved: 0
   DCPDataLength: 6
   Block: Reset to FactorySettings, BlockQualifier: Reset communication parameter
      Option: Control (5)
      Suboption: Reset to Factory (6)
      DCPBlockLength: 2
      BlockQualifier: ResetFactorySettings: Unknown (4)
```

Figure 10-3: DCP Request Reset To Factory

(4) → option 2

(16) → option 8

11. Ethernet Communication – TCP/IP Channel

The Ethernet communication refers today primarily to TCP/IP and UDP/IP data transfer. Of course, still many other protocols which are not so visible also belong to the Ethernet communication.

Typically, multiple applications run on a target device using e.g. TCP/IP. To ease distinguishing between such different processes, a port number is assigned to each application. For various application programs and services, fixed port numbers are assigned.

For TPS-1, a communication channel is implemented which passes the data traffic coming over the PROFINET lines to the field device, onto the host CPU. A TCP/IP stack implemented on the host CPU can then e.g. operate a Web-server independent of the PROFINET communication.

11.1 TCP/IP Channel

The TPS-1 provides the option of supplying a TCP/IP stack on a host CPU with Ethernet frames. This property can e.g. be used to implement a Web-server on a field device as shown in Figure 11-1.

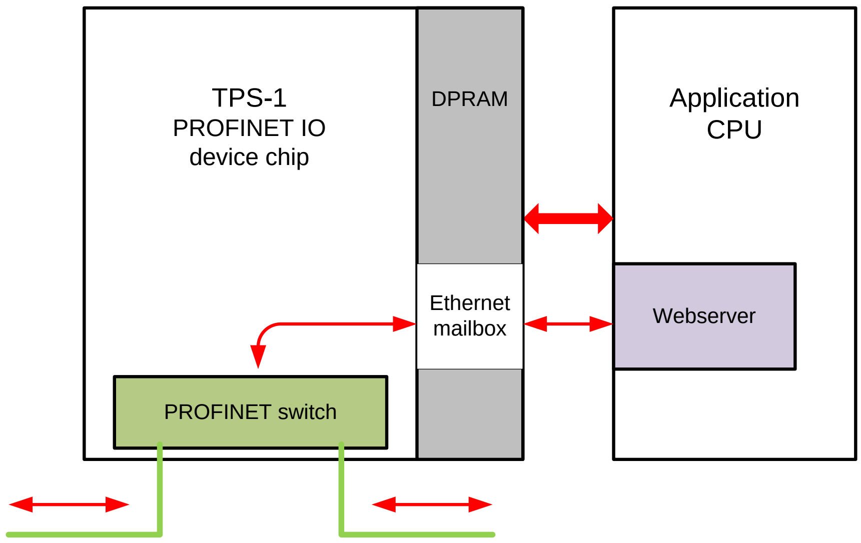


Figure 11-1: Block diagram TCP/IP communication

The internal PROFINET switch evaluates all incoming messages. Messages not belonging to the PROFINET or its protocols are diverted into the Ethernet mailbox in the DPRAM.

At the same time an event is set (**TPS_EVENT_ETH_FRAME_REC**), which then calls an earlier registered Callback function (***App_OnEthernetReceive() – TPS_CheckEvents()***). The same applies in the send direction. If a frame is placed in the send mailbox, then the **APP_EVENT_ETH_FRAME_SEND** event is set, the (***TPS_SendEthernet Frame()*** function is called, and the TPS-1 sends the frame to its destination.

A complete frame is placed in the TPS-1 mailbox. No pre-processing takes place. A queue is built in TPS-1 so that more messages can be cached. However, it must be ensured that messages are picked-up in time, otherwise, they may be discarded and have to be resent.

11.2 Commissioning of the TCP/IP Channel

For commissioning the channel, a compiler switch must be set (**TPS_1_user.h**).

- #define USE_ETHERNET_INTERFACE

This switch performs the registration of the Callback function and the initialisation of the Ethernet mailbox. Further settings are not required.

The following port numbers (sender port - physical) are allowed:

PORT_NR_1	0x1	
PORT_NR_2	0x2	
PORT_NR_1_2	0x3	
PORT_INTERN	0x4	(Special case: only internal messages)

The mailbox in the function ***_onEthPacketReceive ()*** is set again to **„empty“** at the end of processing.

11.3 Ethernet Mirror Application

If desired, a mirror application can be activated for testing (**TPS_1_user.h**):

- #define USE_ETHERNET_MIRROR_APPLICATION

The mirror application sends back the received frame at once. The mirror application is implemented in the ***onEthPacketReceive()*** function.

11.4 Extension of the Host API for selective reception of Ethernet frames

To enable the host application to receive or send special protocol frames from the PN stack, a new parameter (USIGN32 dwHostProtoSelector) is added to the Host API. This parameter can be changed by the host application at runtime. The parameter dwHostProtoSelector is a bit field, where individual bits represent a specific protocol and can be activated independently of each other. The exact bit assignment is shown in the following Table 11-1.

Bit number	Description
0	ARP
1	PTCP (without Frame-IDs: 0x0080, 0x0020, 0xFF20)
2	PN-alarm (high / low)
3	DCP
4	RT/IRT (only those that are not received by PN consumers)
5	SNMP

Bit number	Description
6	RPC
7	UDO (all UDP frames with the appropriate IP address. So far only factory settings from TPS-Configurator / By default in the release versions of the FW from V1.2. x on)
8	ICMP (only ping request / ping responses are always forwarded to the host application)
9	LLDP
10	IP_ALL (all IP frames, also those with an unknown IP address, including all UDP, SNMP, RPC...)
11-15	Reserved
16-31	Mirroring of the bits 0 -15 (redundancy)

Table 11-1: Bit assignment for frame transfer

To set the filter, the following new function is created in the Host API:

USIGN32 TPS_ForwardProtocolsToHost (USIGN32 dwProtoSelector)

The input parameter dwProtoSelector is copied into the variable dwHostProtoSelector in NRT-Area. The content of the parameter dwHostProtoSelector is read and returned as return value.

11.5 List of Used Port Numbers

The following list of port numbers contains protocols with the higher occurrence.

Port number (decimal)	Name	Transport	Description
15	NETSTAT		Network status
19	CHARGEN		Character generator
25	SMTP	TCP	Sending e-mails via Simple Mail Transfer Protocol
67	BOOTP (Server)	UDP	Bootstrap Protocol
68	BOOTP (Client)	UDP	Bootstrap Protocol
69	TFTP	TCP	Trivial File Transfer Protocol
80	HTTP	TCP	Accessing webserver with Hyper Text Transfer Protocol
110	POP3	TCO	Fetching e-mail via Post Office Protocol
111	RPC		Remote Procedure Calls
143	IMAP	TCP	Processing e-mail via Internet Mail Access Protocol
161	SNMP	UDP	Simple Network Management Protocol
162	SNMP-Trap	UDP	SNMP-Traps/Events
443	HTTPS	TCP	Encrypting http via SSL/TLS
34962	PROFINET RT Unicast	UDP	PROFINET IO
34963	PROFINET RT Multicast	UDP	PROFINET IO
34964	PROFINET Context Manager	UDP	PROFINET IO

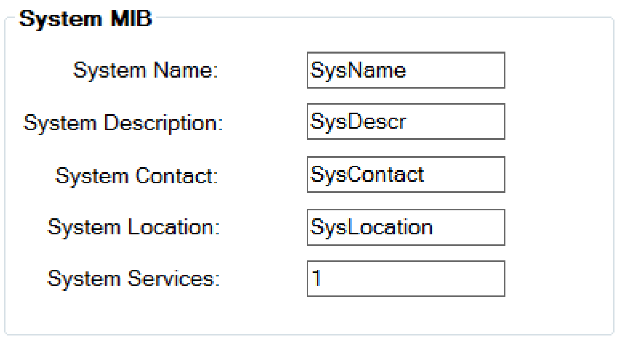
Table 11-2: Important port numbers

12. SNMP Server

For the conformance Class B and Class C, an SNMP server must be present on a PROFINET device that meets the MIB II requirements.

SNMP is primarily responsible for the transport of control data to allow the flow of management information as well as status and statistic data between network components and a management system. This protocol is typically used in the administration of computer networks.

The parameters that are required for the SNMP server can be set with the TPS Configurator as shown in Figure 12-1. Since version 1.5 the parameter System Description is built by the stack firmware and must not be configured. The TPS-1 Stack processes the SNMP parameter. The application has no access to these parameters.


System MIB

System Name:	SysName
System Description:	SysDescr
System Contact:	SysContact
System Location:	SysLocation
System Services:	1

Figure 12-1: TPS Configurator - Ident Settings

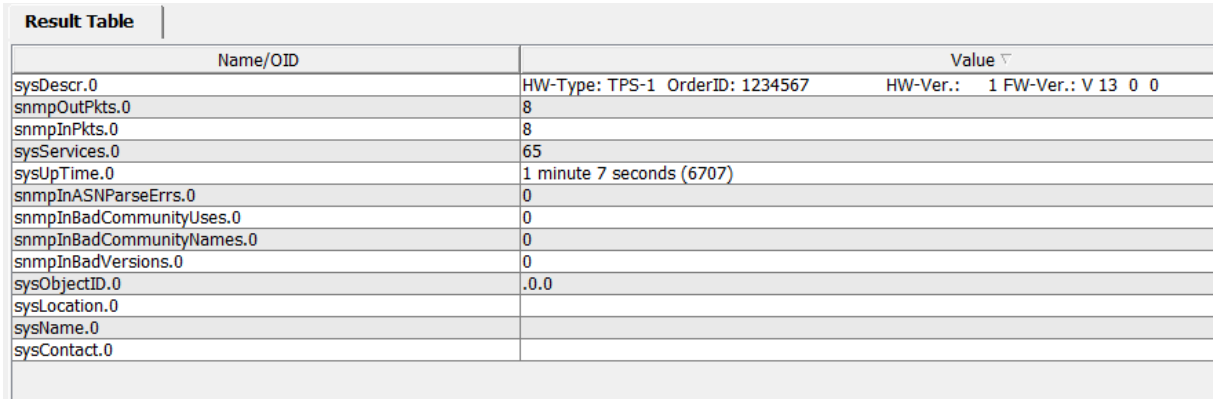
The SNMP parameters are stored in the TPS-1 Flash memory. Every change (e.g. System Name) is saved in the TPS-1 Flash memory. After a voltage failure, the changed data are available again.

Parameter Name	Parameter Description
System Description	A textual description of the entity. This value should include the full name and version identification of the system's hardware type, software operating system, and networking software. It is mandatory that this only contain printable ASCII characters (build by the stack firmware).
System Name	An administratively assigned name for this managed node. By convention, this is the node's fully-qualified domain name.
System Contact	The textual identification of the contact person for this managed node, together with information on how to contact this person.
System Location	The physical location of this node (e.g., “telephone closet, 3rd floor”).
System Services	A value, which indicates the set of services that this entity primarily offers.

Table 12-1: SNMP parameter description

Many SNMP parameters are supplied by the TPS Stack and cannot be influenced by the application. Figure 12-2 below shows all available parameters.

The SNMP-server can be accessed via **Port 161** on the PROFINET device.



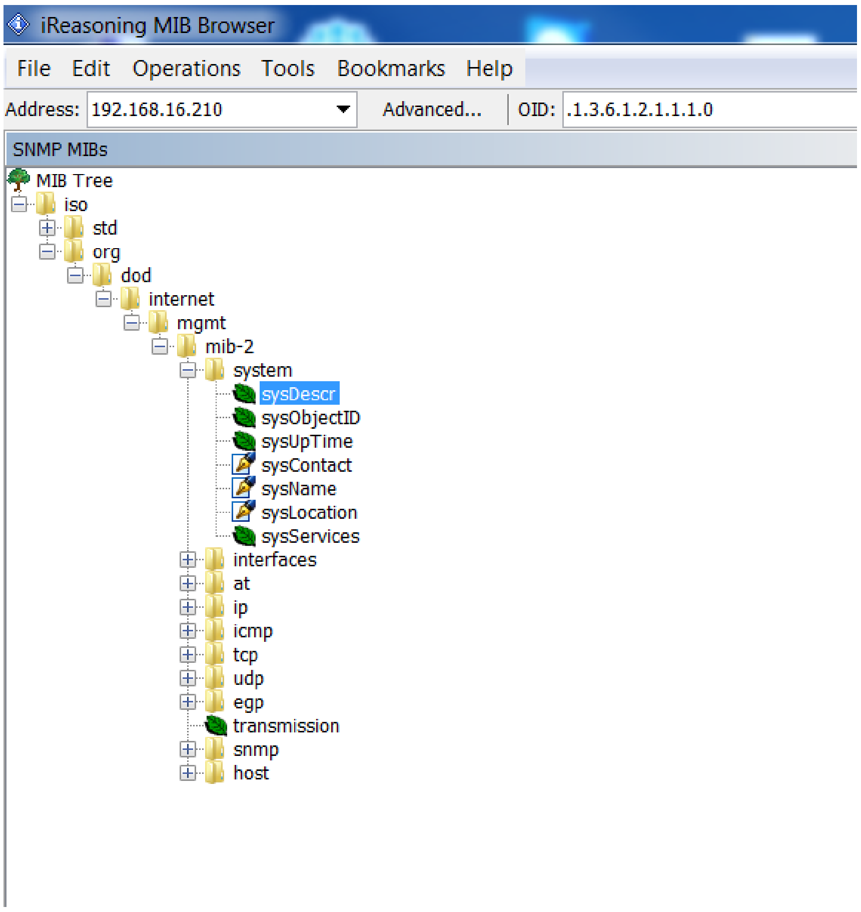
Result Table

Name/OID	Value
sysDescr.0	HW-Type: TPS-1 OrderID: 1234567 HW-Ver.: 1 FW-Ver.: V 13 0 0
snmpOutPkts.0	8
snmpInPkts.0	8
sysServices.0	65
sysUpTime.0	1 minute 7 seconds (6707)
snmpInASNParseErrs.0	0
snmpInBadCommunityUses.0	0
snmpInBadCommunityNames.0	0
snmpInBadVersions.0	0
sysObjectID.0	.0.0
sysLocation.0	
sysName.0	
sysContact.0	

Figure 12-2: Overview SNMP Parameter

12.1 SNMP MIB II for TPS-1

For reading the SNMP data e.g. the „iReasoning MIB Browser“ can be used. The searched entries are found through the device IP address in combination with the OID as shown in Figure 12-3.



Address: 192.168.16.210

OID: .1.3.6.1.2.1.1.1.0

```
MIB Tree
iso
  std
  org
    dod
      internet
        mgmt
          mib-2
            system
              sysDescr
              sysObjectID
              sysUpTime
              sysContact
              sysName
              sysLocation
              sysServices
            interfaces
            at
            ip
            icmp
            tcp
            udp
            egp
            transmission
            snmp
            host
```

Figure 12-3: SNMP MIB tree representation

13. PROFINET Security Class 1

To increase cybersecurity, new mechanisms have been defined in the PROFINET standard 2.4MU4, which are available in version V1.8.1.2 in the TPS-1 firmware.

The following extensions are available:

- SNMP Configuration
- DCP Configuration
- GSD-Signature

(**Note:** The GSD signature is a method of signing the GSDML file of a device manufacturer and thus checking for changes by the engineering. Signing is the responsibility of the device manufacturer and is not part of the TPS-1 firmware.)

13.1 Changes regarding the GSDML-file

Necessary entries in the GSD file to activate Security Class 1:

<CertificationInfo ConformanceClass="C" ApplicationClass="Isochronous" NetloadClass= "III" **SecurityClass="1"**/>

The entry **SecurityClass="1"** describes the implemented security class (this entry belongs to the **"DeviceAccessPointList"**).

13.2 SNMP Configuration

The following table shows the behavior of the SNMP server with "Security Class 1".

	PN-Standard	Security Class 1
SNMP version	V1	V1
Service can be disabled	No	Yes
Default behavior	enabled	Disabled (*)
Community Name	Fixed: "public" / "private"	User defined

Table 13-1: Comparison between legacy and Security Class 1 behavior

Note (*): The locking of the SNMP server is controlled via the CIMSNMPRecord in the start-up. It is not necessary for the device to be locked when new.

It will be useful to Change the Community Name before read and write access to individual settings.

The configuration of the device SNMP server is done by the engineering tool.

GSDML entry for supporting the SNMP-Server:

<InterfaceSubmoduleItem SNMP_FeaturesSupported=„SNMPAdjust"/>

If the token “SNMP_FeaturesSupported.SNMPAdjust” is set, the records CIMSNMPAdjust and CIMSNMPReject are supported:

- CIMSNMPAdjust (0x8200)
- CIMSNMPReal (0x8201) – Read only

13.3 DCP Configuration

If the GSDML Tag **DCP_FeaturesSupported** is set to **“RejectDCPSet”** and an AR is established (operating mode) ist is not possible to send critical DCP commands (e.g., ResetToFactory, SetName).

<InterfaceSubmoduleItem DCP_FeaturesSupported=„RejectDCPSet“/>

13.4 Signing of GSD-files

By using **signed** GSD files (GSDX), an engineering user can verify the following aspects:

- Is the GSD file unchanged and exactly as intended by the manufacturer,
- Is the manufacturer authorized by PNO to sign GSD files for the used VendorID.

The signing process must be done by the manufacturer.

14. Update of the TPS-1 Firmware

The TPS-1 firmware consists of the TPS Updater and the TPS Stack. The TPS Updater takes care of the reception of new software and for copying it into the TPS-1 external Flash. The TPS Stack constitutes the entire PROFINET communication functionality.

A firmware update can be done via one of the Ethernet interfaces (alternatively POF) or the DPRAM by the application processor (Chapter xyz).

The process of initialisation and update of the hardware configuration as well as the firmware is described in the **TPS-1 Firmware Update Manual** [7] in detail. In the following only a summary of the steps is given

.

14.1 Hardware Configuration of the TPS-1

The hardware configuration is carried out with the TPS Configurator. The selected configuration is downloaded into TPS Flash after loading and starting the **TPS Starter.s** program via UART interface. The TPS-1 now expects hardware configuration. Before the transfer of the **TPS Starter.s** program, the flash must be erased.

After successful transfer, the TPS-1 is reset. This completes the hardware configuration.

Note: If the TPS-1 hardware configuration is to be written, then no other programs should operate on the Ethernet interface at that time. The TPS-1 receives all frames in this state.

14.2 Preparation of Flash Images

After transferring the image files, a header at the beginning of the file is checked to ensure that the right file has been transferred. The **VendorID**, the **DeviceID** and the **Hardware-Revision** of the module are used for this test. This process is necessary to prevent update of other modules. Each image must be prepared in this way for the transfer. See [7].

14.3 Firmware Update via Ethernet

Using the FWUpdater, now, the TPS Updater and the TPS Stack are loaded and programmed. See also [7].

15. Production Environment (Default Image)

In a production environment, it is not acceptable to go through the steps hardware configuration, programming the TPS Updater and finally to program the TPS Stack. A default image file is provided to simplify and accelerate the process in a production environment.

The default image file is delivered in two versions. One for the use with RJ45 interface and the other for use with POF interfaces.

The image can be found in TPS_Stack directory:

- TPS_Default_Image_ETH.hex

These images include a hardware configuration, the TPS Updater, and the TPS Stack. The configuration block is artificially corrupted.

The serial flash device, that is connected to the TPS-1, is programmed with the default image before soldering and then assembled with other components of the module. It is also possible to program the image with the boundary-scan functions into the flash (in-system programming)

The deliberately corrupted configuration block causes the TPS-1 to request a configuration block when the power is switched on for the first time. This operation must always be carried out because the configuration block also contains the MAC addresses and the serial numbers which are of course unique for each module.

The configuration block can be written with the TPS Configurator. But also a batch file can call the **FS_PROG.exe** program that writes the data via the Ethernet connection in the TPS Flash.

A detailed description of the FS_PROG.exe, its invocation and error codes can be found in the TPS Configurator help functions.

16. Special TPS-1 Properties

16.1 Automatic Adaption to the Target Configuration

Complex devices such as IO-Link gateways or modular devices require greater flexibility with respect to configuration. For the TPS-1, adaptation to the target configuration has been implemented. Here, while connecting to a controller (Connect.Req), the decision is made how the field device should be configured.

During the ramp-up, the maximum possible expansion level is entered in the configuration. The **ModuleIdentNumber** and the **SubmoduleIdentNumber** are set to „0". The compiler switch **„USE_AUTOCONF_MODULE"** must be set. Permanently plugged (fixed) modules are entered with their **ModuleIdentNumber** or **SubmoduleIdentNumber**.

In the case of a **Connect.Req**, the TPS-1 enters the configuration (**Module ID Number and SubmoduleIdentNumber**) requested by the controller. In a next step **(*App_OnConnectCallback()*)** the application must then check, whether it is possible to meet the request.

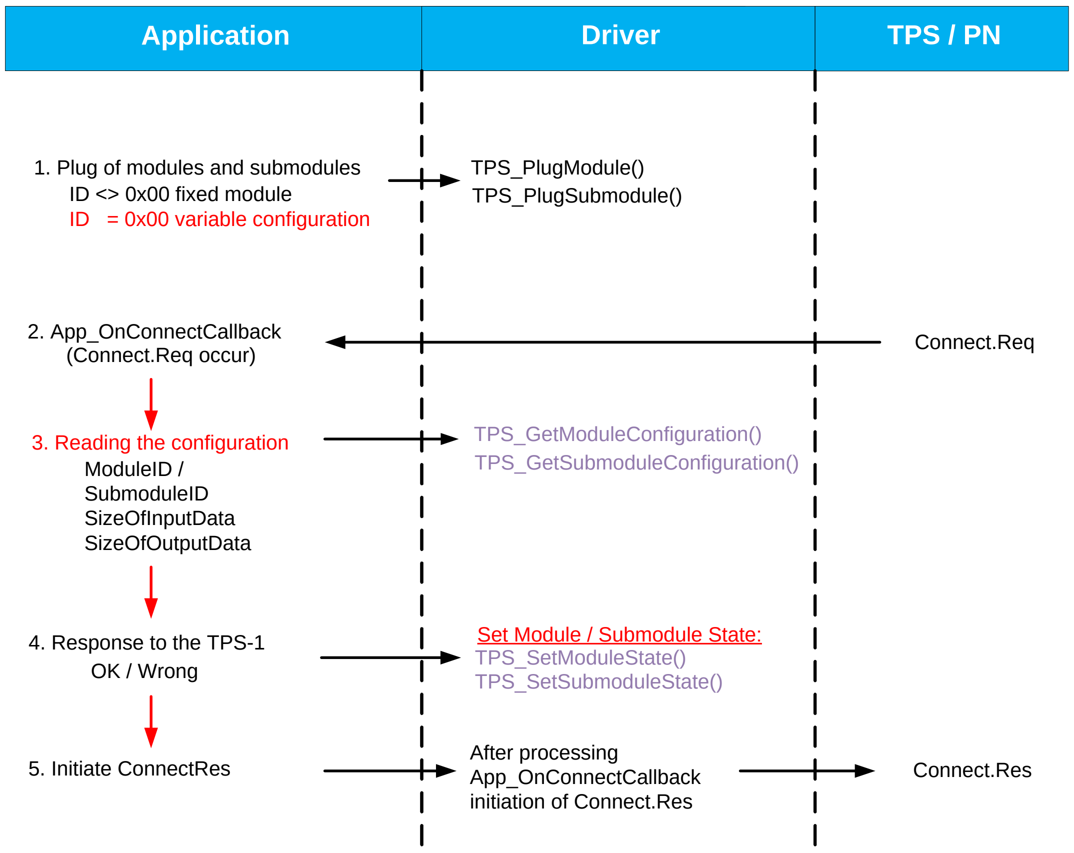


Figure 16-1: Adaption to the target configuration

Among others, the following functions are available for processing the modules.

The following functions listed in are required in this context.

Function name	**Description**
TPS_GetModuleConfiguration()	Delivers the current ModuleNumber for the given slot
TPS_GetSubmoduleConfiguration()	Delivers the current SubmoduleIdentNumber and the size of the IO-Data in the IO-RAM
TPS_SetModuleState()	Sets the status of a given module. This function must be called for each slot in the function ***OnConnectCallback()*** so that if required, a ModuleDiffBlock can be created in Connect.Res.
TPS_SetSubmoduleState()	Sets the status of a given submodule. This function must be invoked for each configured submodule in the function ***OnConnectCallback()*** so that if required, a ModuleDiffBlock can be created

Table 16-1: Function involved in adaptation to expected configuration

An example is available in the extended example application in the TPS-1 Development Kit.

Note: A status must be assigned to each module and submodule so that if required, a ModuleDiffBlock can be created properly.

16.2 Transferring Initial Parameters

After the successful Connect.Req, the controller starts to send the initial parameters for each configured subslot. The transfer is done with Write.Req and is received by the device. It is also possible to send data for multiple subslots in one frame (in the GSDML, the attribute must have ***„MultipleWriteSupported=true“***).

If all parameters are loaded into the device, the controller marks the end of parametrization with a „DControl.req“-frame (**„EndOFParametrization“**).

The application software then checks the data of all configured subslots and sends an **"Application Ready"** message to the controller. This completes the AR build-up and the sending of cyclic data begins.

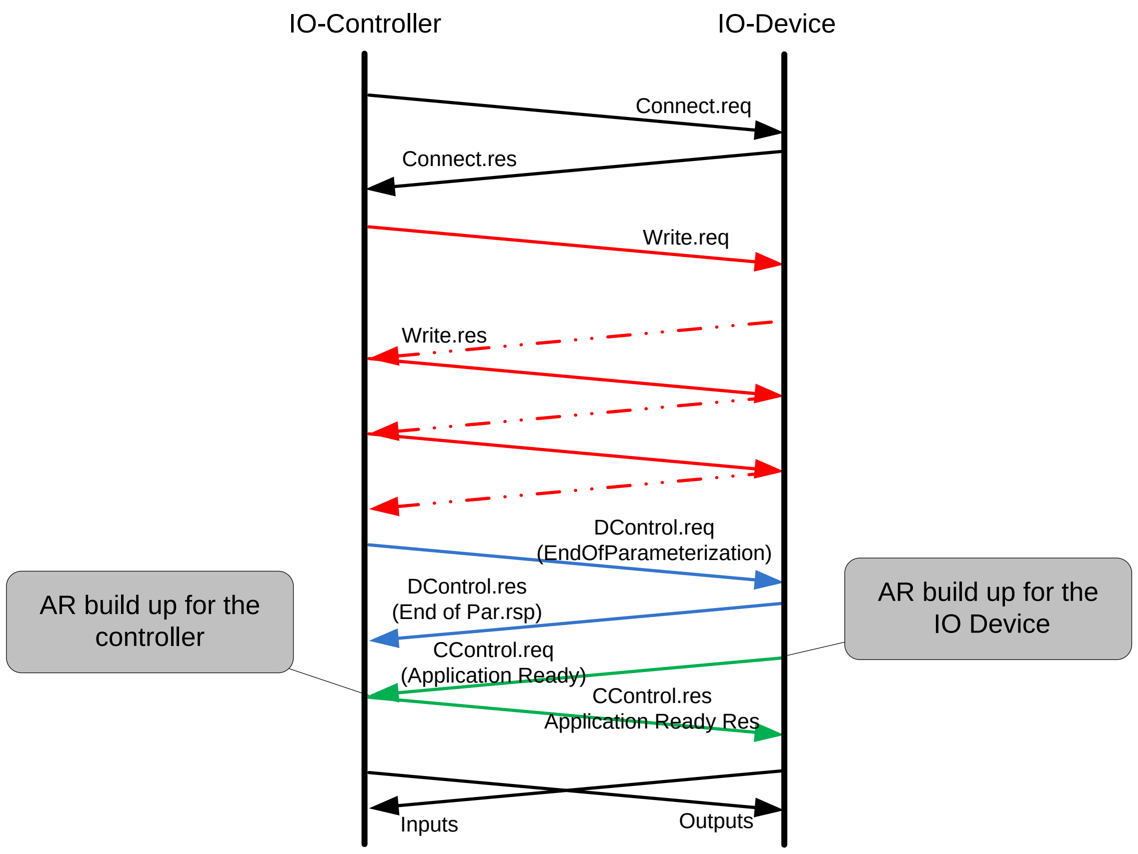


Figure 16-2: Device ramp up (Connect.Req)

The TPS-1 firmware assigns the initialisation data to the corresponding subslots. The TPS-1 puts data in DPRAM (NRT area). The application can access this data after completion of the parameter transfer. The end of the transfer is signalled by the event ***„TPS_EVENT_ON_PRM_END_DONE_IOARx“.*** For this event, the previously registered Callback function ***App_OnPrmEndCallback()*** is called.

All necessary device settings should be made in this function. In a next step, the device would send ***„DeviceReady“*** to the controller and the cyclic operation starts.

```
typedef struct subslot
{
..
       USIGN32*      pt_size_init_records;
       USIGN32*      pt_size_init_records_used;
       USIGN8*       pt_init_records;
..
} SUBSLOT;
```

Figure 16-3: Initial Parameter Pointer of the subslot data

Initial parameters can be accessed via the corresponding pointer. During the subslot configuration, it should be ensured that sufficient space is reserved in the subslot data (see TPS_PlugSubmodule ()).

16.3 TPS-1 Hardware Configuration via DPRAM

Typically, the TPS-1 hardware configuration is done with the **TPS Configurator** or with the **FS_PROG.exe** program.

However, for certain applications it is also possible to transfer the configuration block from the host CPU via DPRAM. In this case, however, a default configuration must be present in the TPS-1 Flash and the TPS Updater and TPS-1 Stack must already be programmed in TPS-1 Flash.

For using this transfer channel the compiler switch **„USE_TPS_COMMUNICATION_CHANNEL“** must be set in the TPS_1_user.h file. Additionally, the compiler switch **„USE_ETHERNET_INTERFACE“** must be set**.** This initializes the Ethernet mailbox.

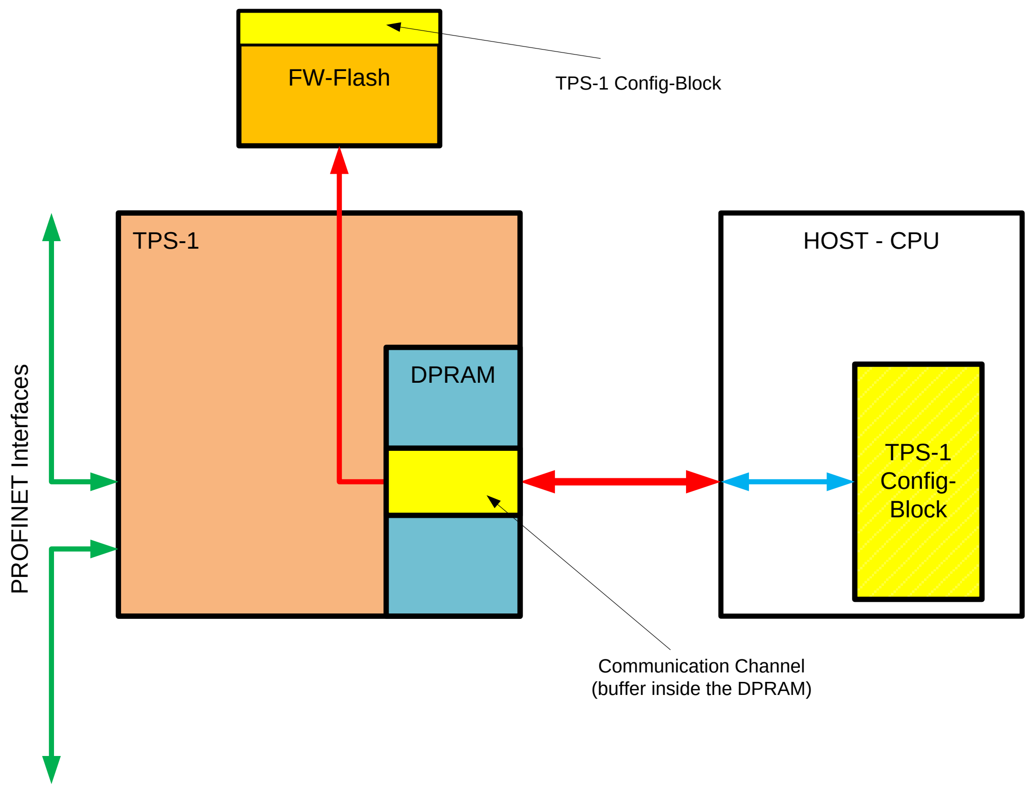


Figure 16-4: TPS-1 configuration via DPRAM

16.3.1 Generating the Configuration Block

During the transfer of a configuration to a TPS-1 module, the TPS Configurator generates besides an * .xml file also a file with the extension * .xml.c.

Typical examples for these files would be:

- host_interface_parallel.xml
- host_interface_parallel.xml.c

The leading name is freely selectable by the user.

The C-file consists of a C-array containing all data of the configuration block (pbyConfigurationsData[]) as it had already been transferred once.

A list of constants representing the offsets follows the array. These offsets, allow direct addressing and changing of parameters if necessary. Here, for example, it would be possible to enter another MAC addresses.

16.3.2 Transferring the Configuration Block via DPRAM

The ***„TPS_WRITEConfigBlockToFlash()"*** (Driver) function call transfers the configuration block via the DPRAM to TPS-1 and enters it in the TPS-1 Flash.

After the transfer, the TPS-1 must be reset. After that, the data are also available in the NRT area.

Note:

A valid IP address must be available for the device. The same interface is used to change the TPS Configuration as if the configuration were sent from the network via the Ethernet interface. An IP address like 0.0.0.0 is not a valid address.

16.4 TPS-1 Stack Update via DPRAM

For certain applications, it is necessary to update the firmware of the TPS-1 (TPS Updater and TPS Stack) from the host CPU. For that, the Driver provides functions with which the firmware can be rewritten via the DPRAM.

For the activation of the communication channel, the following compiler switches must be set (File TPS_1_user.h).

- #define USE_ETHERNET_INTERFACE
- #define USE_TPS_COMMUNICATION_CHANNEL
- #define FW_UPDATE_OVER_HOST

By activating the transfer channel, for example, a TPS Stack update by the application CPU can be started. On a device specific path, the image is transferred to the application CPU and passed on from there to the TPS-1.

The image to be transferred must be prepared as done during programming via the Ethernet interface (VendorID, DeviceID, etc. must be entered - see the Update Manual).

With the ***TPS_StartFWUpdater()*** function call, the implementation of TPS Stack is stopped, and the TPS Updater present in the TPS Flash is loaded and started.

Then, with the ***TPS_WriteFWImageToFlash()*** function, the image is transferred to the TPS-1 and programmed in the flash device.

After the transfer is complete, the TPS Updater is stopped and the TPS Stack is restarted with the ***TPS_StartFWStack()*** function.

Depending upon the performance of the application CPU, it may be necessary to change the transfer time. The function ***TPS_SetUpdaterTimeout()*** sets a time which determines the timeout. With the function ***TPS_GetUpdaterTimeout(),*** the set time can be read.

Note: A block-by-block transfer of the stack has already been tested successfully at a customer, but is not part of the standard API delivery. The driver function ***"TPS_WriteFWImageFragmentToFlash()"*** sends only one fragment of the complete image. The customer must implement the remaining functionality by himself. The advantage would the usability of an application processor with smaller memory footprint.

16.4.1 Starting the TPS Updater

The TPS-1 must be switched into update mode for this. By calling the ***TPS_StartFWUpdater()*** function, the TPS Stack program execution is stopped and the TPS starts with the execution of the TPS Updater.

The ***TPS_StartFWUpdater()*** function writes a command structure in the DPRAM (NRT area) out of which the TPS-CPU derives the further steps.

```
typedef struct _dpr_header
{
    USIGN32 dwHostSign;
    USIGN32 dwHostEvent;
    USIGN32 dwUpdaterEvent;
    USIGN32 dwErrorCode;
    FW_UPD_MAILBOX pbFwStartAddr;
} DPR_HOST_HEADER;
```

Figure 16-5: Update command structure

The constant ***„HOST_LIFE_SIGN“*** is entered in the variable ***„dwHostSign“,*** and a TPS CPU software RESET is triggered via the Event Unit. During the new ramp-up, this memory cell is evaluated, and the TPS-1 starts the TPS Updater. In response, the host application expects the constant „UPDATER_LIFE_SIGN” in the memory cell, as soon as this is detected, the application CPU enters the „UPDATER_LIFE_SIGN” again and can start with data transfer.

16.4.2 Transferring the Requested Firmware Image

After the TPS-1 has been switched into update mode, data transfer can start. For this purpose the function

- TPS_WriteFWImageToFlash(USIGN8* pbyImage,USIGN32dwLengthOfImage)

is called. The image start address and length are passed to the function.

The data transport control is done via the ***„FW_UPD_MAILBOX“*** structure which is a constituent part of the ***„DPR_HOST_HEADER“*** structure.

```
typedef struct _dpr_fragment_mbx
{
    USIGN32 dwFlag;
    USIGN32 dwFragLen;
    USIGN8  bData[FW_FRAG_LEN];
} FW_UPD_MAILBOX;
```

Figure 16-6: Send control structure image

The variable ***„dwFlag“*** can have the following states:

• #define MAILBOX_EMPTY	0x00
• #define MAILBOX_BUSY	0x01
• #define MAILBOX_LAST_BLOCK	0x02

As soon as a transfer block is written in the mailbox, the flag is set to ***„MAILBOX_BUSY“*** and the TPS-1 recognizes that a new block has arrived. As soon as the mailbox has been emptied by TPS-1, the flag is set again to ***„MAILBOX_EMPTY“.*** If the last block is written, the flag is set to ***„MAILBOX_LAST_BLOCK“*** and the TPS-1 starts to check the transferred image. After checking, the image is stored in the flash memory.

16.4.3 Starting the TPS-Stack

After the TPS Updater has transferred the requested image in the TPS-1 Flash, the TPS Stack must be restarted. The function ***„TPS_StartFWStack()“*** ensures a new start of the TPS-1 Stack. The function is called at the end of **TPS_WriteFWImageToFlash()**.

16.5 Using TPS-1 Flash for Host Application Data

Since TPS-1 stack version V1.4 a developer can use a memory block of 4k byte data inside the TPS-1 flash memory. This flash memory can used for any data the developer want, but he is also completely responsible for this block.

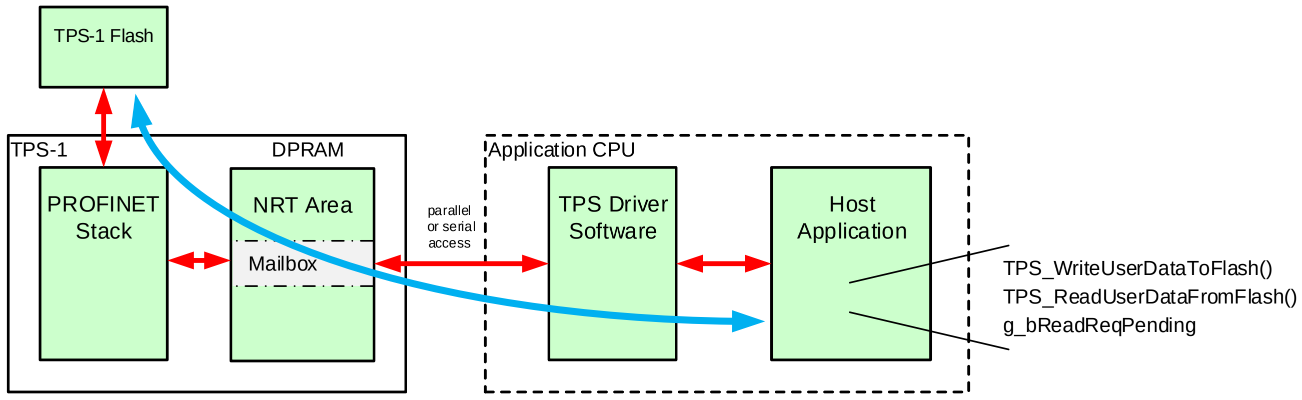


Figure 16-7: Transfer of private Flash data

To reach the TPS-1 Flash the TPS firmware must be in operation and the internal Ethernet Channel must be activated (#define USE_ETHERNET_INTERFACE).

For managing this data block the DRIVER provides two functions:

- ***TPS_WriteUserDataToFlash(wAddr, pbyUserData, wLength)***
- ***TPS_ReadUserDataFromFlash(wAddr, wLength)***

The function "***TPS_WriteUserDataToFlash()***" stores data to the flash memory of the TPS-1. The developer must take care for the structure and addresses of the data blocks. For a safe execution of a command the variable "**g_bReadReqPending**" must be referred; a TPS_TRUE indicates a running command.

Reading data from the TPS-1 flash is performed by the function "***TPS_ReadUserDataFromFlash()***". You can find the detailed parameters in the TPS Reference Manual [2].

The transfer of the payload data is organized via the internal Ethernet Channel as illustrated in Figure 16-7. The related record index is **RECORD_INDEX_USER_DATA_IN_TPS_FLASH** (0x2012). One single message can contain a maximum of 1342-byte data (max. payload). If you want to transfer more data, please manage the write address by your own.

You can find an example of a read operation in the extended example code. If you want to realize a write request you must at first transfer the response out of the mailbox into the receive buffer. The function "***onTpsMessageReceive()***" delivers the response (e.g. data by a read request).

16.6 Generation of Process and Diagnosis Alarms

16.6.1 General Diagnosis and Alarm Processing

PROFINET offers specific mechanisms to illustrate diagnoses. PROFINET differentiates between alarms from the process attached to an IO Device (Process alarms) and alarms generated by the IO Devices themselves (Diagnosis alarm). A diagnosis must be entered in the "Diagnosis database" of a device; if necessary, a diagnostics alarm is also sent to the IO Controller. All available diagnosis messages from an IO Device can be read via a standard path. Standard errors are described in the specification. The texts are stored in the engineering tool. Private messages must be stored in the GSD file.

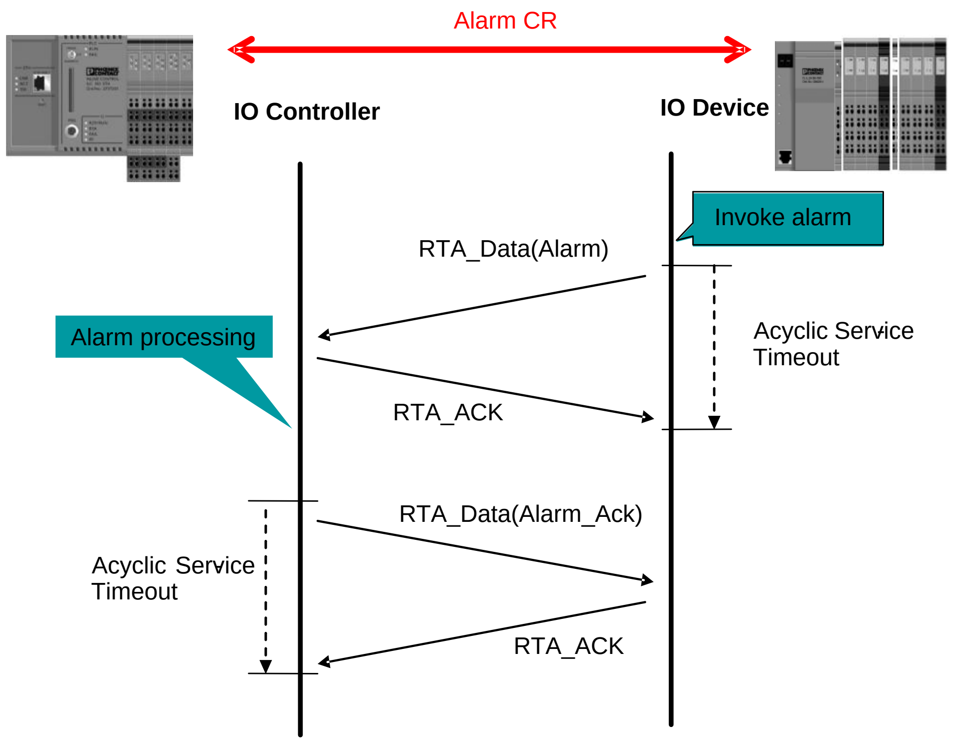


Figure 16-8: Sending diagnostic alarms

If an IO Device detects a diagnosis, it sends a ***Diagnosis Appear*** alarm to the IO Controller. Additionally, the IO Device generates a corresponding entry in its diagnosis buffer (in the case of TPS-1, the diagnosis buffer is managed by the TPS Stack). Entries in the diagnosis buffer can also be made without corresponding alarm. Examples for this are the preventive diagnosis messages that indicate a state of wear, or messages that indicate when the next maintenance is due.

If an entry is generated in the diagnosis buffer, then the ***„DataStatus“*** automatically shows in bit ***„StationProblemIndicator“*** (*APDU_Status.DataStatus.StationProblemIndicator*) of the cyclic data sent by the device, that at least one diagnosis entry is present.

The IO Controller can read the diagnosis information on the NRT channel using ***ReadRecord.Request*** and take appropriate action.

Process alarms do not result in the setting of the ***APDU-Status***. They are analysed directly in the control system. Process alarms do not result in an entry in a diagnosis memory.

Detailed use of alarms and diagnoses can be found in the Guideline **„Diagnosis for PROFINET IO.“** [8].

16.6.2 DRIVER Functions for Diagnosis and Alarm Processing

The Driver provides the following functions for processing alarms and diagnosis:

Name of function	Description
TPS_SendDiagAlarm()	This function sends a diagnosis alarm notification to an IO Controller. Before transfer, the diagnosis must be entered in the diagnosis ASE (***TPS_DiagChannelAdd()***).
TPS_DiagSetChannelProperties()	This function builds a „channel properties block“ that is used for the ***TPS_DiagChannelAdd()*** function.
TPS_DiagChannelAdd()	This function makes the diagnosis entry in the diagnosis database.
TPS_DiagChannelRemove()	This function removes a diagnosis entry from the diagnosis database.
TPS_DiagSetChangeState()	With this function the „Maintenance Status“ and the „Specifier“ of an already existing diagnosis entry can be changed.

Table 16-2: DRIVER functions for handling of diagnostics

The parametrisation of the diagnostic functions can be found in the TPS-1 Reference Manual.

16.6.3 Example of a Diagnosis Alarm

During the configuration of a subslot (***TPS_PlugSubmodule()***), a parameter (***wNumberOfChannelDiag***) is passed which specifies the number of possible diagnoses for this subslot. In the sub-slot data, a corresponding storage area is created.

```
pt_size_chan_diag;  /*Pointer to maximum number of diagnosis*/

pt_chan_diag;       /*Pointer to the first diagnosis entry of this subslot */
```

Figure 16-9: Diagnoses information of the subslot data

If e.g. a plugged submodule detects a short circuit in a connected cable, then this is reported to the IO Controller using a diagnosis alarm.

The following steps must be taken to enter a diagnostic alarm:

- TPS_DiagSetChannelProperties(&wChannelProperties,,,)

  `/* Preparing the channel properties for diagnostics */`

- TPS_DiagChannelAdd(,,..)

  `/* Enter diagnostics */`

- TPS_SendDiagAlarm(,,..)

  `/* Send appear alarm */`

By invoking the ***TPS_DiagChannelAdd()*** function, a diagnosis is entered in the subslot diagnosis buffer. The diagnosis entry structure can be found in the file TPS_1_API.h (DPR_DIAG_ENTRY).

The diagnosis alarm is triggered by the ***TPS_SendDiagAlarm()*** function and is passed to the IO Controller. If e.g. the error status does not exist anymore, then the diagnosis is deleted from the diagnosis queue and a diagnosis alarm (off) is sent.

The following steps must be taken to delete a diagnostic alarm:

- TPS_DiagSetChangeState(,,,)

  `/* Change of the diagnosis entry from appear to disappear, etc. */`

- TPS_SendDiagAlarm(,,,)

  `/* Send disappear alarm */`

- TPS_DiagChannelRemove()

  `/* After acknowledge, delete diagnosis entry */`

16.7 PROFINET System Redundancy S2 with the TPS-1

PROFINET System Redundancy S2 specifies the connection of two redundant controllers with one PROFINET device (or several). The basis for data processing within the TPS-1 is the "System Redundancy Layer Device" (SRL-D).

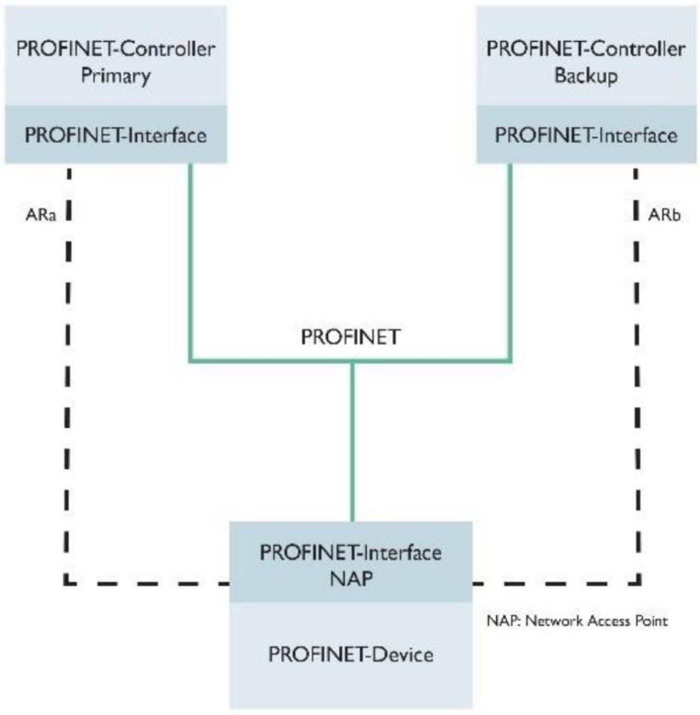


Figure 16-10: S2-Redundancy with the TPS-1

The mechanisms for deciding which controller (primary or backup) to use are decided in a communication layer between the controllers and cannot be decided by a device. The TPS-1 Firmware handle S2 Redundancy without management by the application software.

17. GSD (General Station Description) for the TPS-1

17.1 What is a GSD?

The GSD is a formalized technical description of the PROFINET field device which contains all the information for the data transport and engineering.

These are:

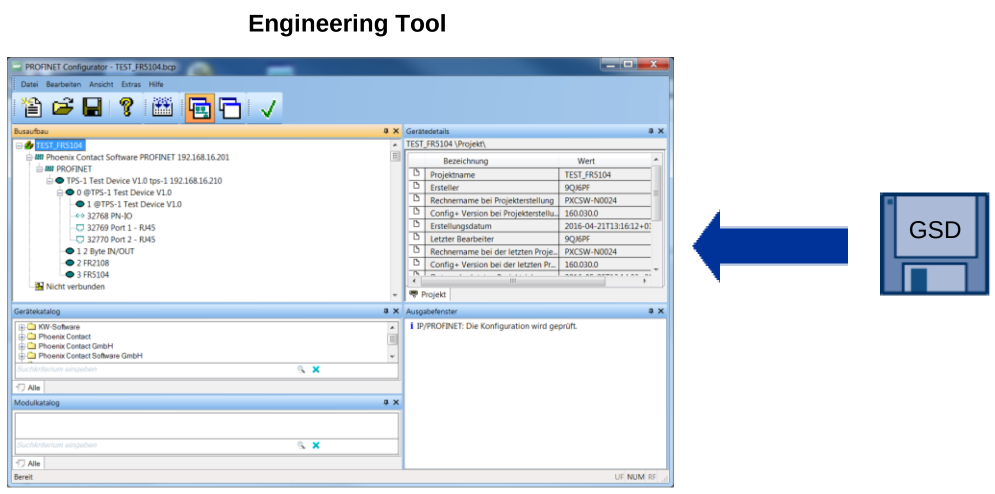
- Communication parameters, communication capabilities
- Device structure (as far as relevant for the communication: modules, submodules)
- Catalogue information (device description,...)
- Structure of cyclic data and ramp-up parameters
- Definition of diagnosis information (only alarms)
- Engineering information (icons, pictures, texts, values)
- Order numbers (for selection and order processing)

The GSD does not describe:

- Complex user interface (graphics, charts, wizards)
- Dependencies (e.g. of variables among each other)
- Complex slot rules
- Applicative diagnostics
- Device specific business logic
- Mechanical data, connection diagrams

The GSD is indispensable for a PROFINET field device because this is the only gateway to engineering. Every manufacturer of a PROFINET field device must create a corresponding GSD file whose verification is part of the certification test.

GSD data is the sole source for the engineering tool to gain knowledge about the device as shown in **Figure 17-1**. For this purpose, the GSD file is read once in the engineering tool (e.g. the programming device). After that, the field device can e.g. be configured from the product catalogue of the engineering tools



PROFINET Configurator - TEST_FR5104.bcp

Datei Bearbeiten Ansicht Extras Hilfe

Busaufbau

```
TEST_FR5104
  Phoenix Contact Software PROFINET 192.168.16.201
    PROFINET
      TPS-1 Test Device V1.0 tps-1 192.168.16.210
        0 @TPS-1 Test Device V1.0
          1 @TPS-1 Test Device V1.0
          32768 PN-IO
          32769 Port 1 - RJ45
          32770 Port 2 - RJ45
        1 2 Byte IN/OUT
        2 FR2108
        3 FR5104
  Nicht verbunden
```

Gerätedetails

TEST_FR5104 \Projekt\

Bezeichnung	Wert
Projektname	TEST_FR5104
Ersteller	9QJ6PF
Rechnername bei Projekterstellung	PXCSW-N0024
Config+ Version bei Projekterstellu...	160.030.0
Erstellungsdatum	2016-04-21T13:16:12+0
Letzter Bearbeiter	9QJ6PF
Rechnername bei der letzten Proje...	PXCSW-N0024
Config+ Version bei der letzten Pr...	160.030.0

Projekt

Gerätekatalog

```
KW-Software
Phoenix Contact
Phoenix Contact GmbH
Phoenix Contact Software GmbH
```

Suchkriterium eingeben

Alle

Modulkatalog

Suchkriterium eingeben

Alle

Ausgabefenster

IP/PROFINET: Die Konfiguration wird geprüft.

Bereit UF NUM RF

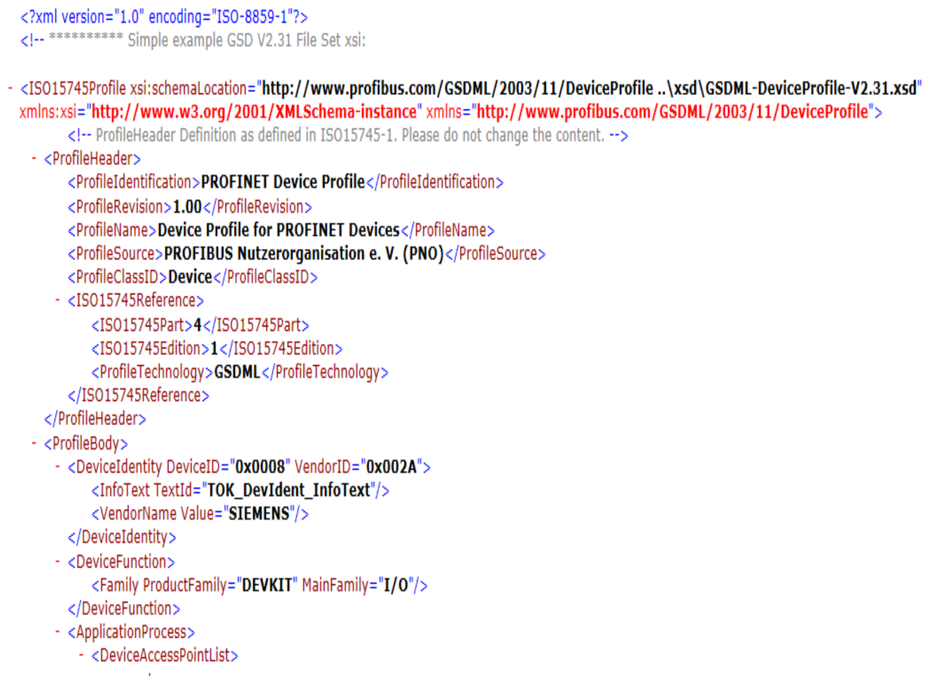
Figure 17-1: GSD and engineering tool

During the device development, the GSD, and its properties must be taken into consideration at an early stage so that the device functions can also be properly mapped to the GSD file. If it is done too late, it may lead to unnecessary modification efforts. You can find further general information about GSD in chapter 2 of „PROFINET System Description, Technology and Application“(german version: *„PROFINET Systembeschreibung, Technologie und Anwendung“*), which can be downloaded free of charge under:

http://www.profibus.com/download/technical-descriptions-books

17.2 What is the GSDML (GSD Markup Language)?

The GSD file is an XML file, which can be created and processed with standard tools. GSDML is the descriptive language of GSD file, which defines the device properties over multiple layers and is thus well suited for hierarchical illustration of PROFINET field devices. The following figure shows e.g. the general part of the identification of field device of a simple GSD file.



```
 <?xml version="1.0" encoding="ISO-8859-1"?>
 <!-- ********** Simple example GSD V2.31 File Set xsi:

- <ISO15745Profile xsi:schemaLocation="http://www.profibus.com/GSDML/2003/11/DeviceProfile ..\xsd\GSDML-DeviceProfile-V2.31.xsd"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.profibus.com/GSDML/2003/11/DeviceProfile">
     <!-- ProfileHeader Definition as defined in ISO15745-1. Please do not change the content. -->
  - <ProfileHeader>
     <ProfileIdentification>PROFINET Device Profile</ProfileIdentification>
     <ProfileRevision>1.00</ProfileRevision>
     <ProfileName>Device Profile for PROFINET Devices</ProfileName>
     <ProfileSource>PROFIBUS Nutzerorganisation e. V. (PNO)</ProfileSource>
     <ProfileClassID>Device</ProfileClassID>
   - <ISO15745Reference>
       <ISO15745Part>4</ISO15745Part>
       <ISO15745Edition>1</ISO15745Edition>
       <ProfileTechnology>GSDML</ProfileTechnology>
     </ISO15745Reference>
   </ProfileHeader>
  - <ProfileBody>
   - <DeviceIdentity DeviceID="0x0008" VendorID="0x002A">
       <InfoText TextId="TOK_DevIdent_InfoText"/>
       <VendorName Value="SIEMENS"/>
     </DeviceIdentity>
   - <DeviceFunction>
       <Family ProductFamily="DEVKIT" MainFamily="I/O"/>
     </DeviceFunction>
   - <ApplicationProcess>
     - <DeviceAccessPointList>
```

Figure 17-2: General part of a GSD file in XML illustration

17.3 Which Information can be Found in the GSD File?

The interaction between GSD file and engineering is illustrated in Figure 17-3. It shows e.g. how, based on the information from GSD file (see also XML illustrations in the previous figure) the engineering classifies the field device in its product catalogue.

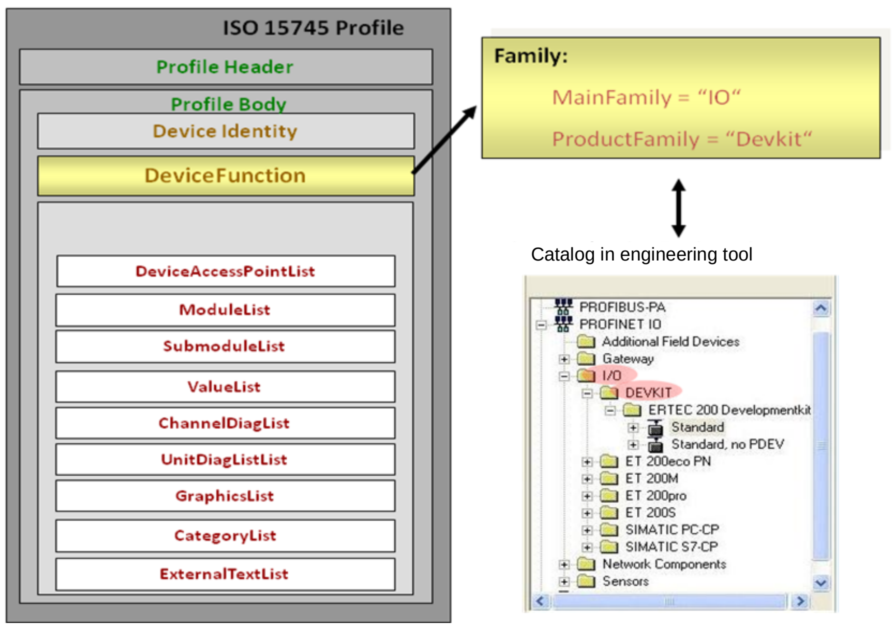


Figure 17-3: Interaction GSD and engineering

The basic structure of the GSD file can be seen from the block diagram in **Figure 17-3**.

17.3.1 Profile Header

General profile information like e.g. profile name, profile version, the publisher of the profile are stored in the profile header.

17.3.2 Profile Body

The actual field device data is located in the profile body. It contains the information for unambiguous identification, indicates to which product family the device belongs and describes all the communication properties of the field device. Depending upon the functions and structure of the device, this main part of the GSD file can be very extensive, because information

- about communication capabilities
- about configuration
- of parametrisation
- of illustration
- of diagnosis
- about references to graphics and texts

is necessary. What needs to be considered for the Vendor ID and Device ID in the GSD?

The PROFINET field device of a manufacturer must be unambiguously identifiable in the engineering tool and the subsequent real automation system. This is achieved by the combination of the company ID (Vendor ID) and a specific device identifier (DeviceID).

The DeviceID which must be unambiguous in the manufacturing company is specified by the manufacturer himself. To avoid multiple assignments, it is recommended to coordinate the assignment with the marketing or product management within the company. Block wise assignment to departments or maintaining a central list has proven to be a pragmatic approach.

Besides the VendorID and DeviceID approach, which is essential for the engineering and production data traffic, there is also the possibility to identify a field device in the engineering through its order number. This option is convenient for simplifying the ordering process.

Note: In real life, it may happen that device manufacturers, who realise their PROFINET connection with a communication module, leave the VendorID of the communication module or its manufacturer unchanged in the GSD file. This leads in subsequent system diagnostics to misinterpretation, because not the manufacturer of the device is displayed, but rather the module supplier. It is, therefore, important to ensure that the VendorID of the device supplier is used.

17.4 What are the Application Implications on the GSD?

The application requirements demand certain device functions which in turn are mirrored in the GSD file. The philosophy of the GSD is that every technical feature is described by a separate element or attribute. In case there are technical dependencies to other features, then it is described in the GSDML specifications and will also be verified by the GSD Checker (see chapter xyz).

Moreover, there are marketing rules that are non-technical in nature. Thus, if “x” and “y” are supported, then “z” must also be supported. This serves to reduce the number of variants. To this end, in particular, both the „Conformance Classes" as well as the „Application Classes" play a role.

17.4.1 Conformance Classes (CC)

The conformance classes provide a convenient summary of various minimum requirements. Through the certification of a field device to a conformance class it is ensured that the selected field devices support unambiguously defined minimum requirements on functionality and interoperability. The conformance classes are geared to the needs of specific fields, such as e.g. factory automation and motion control.

17.4.2 Application Classes (Optional)

Application Classes define device characteristics so that the application function can be provided and it is interoperable with the system properties. In an application class, the minimum scope for the relevant application in an automation system is defined.

The following application classes are defined:

- Isochronous Application
- Process automation
- High performance
- Controller to Controller
- Functional safety

If the manufacturer specifies an application class in the GSD file, the corresponding minimum scope is also tested in the certification test.

17.5 What are the Key Issues in „Life Cycle Management“ of GSD?

The life cycle management of the GSD can be influenced by newer versions of the GSDML or newer versions of the PROFINET field device, see also chapter 10.

17.5.1 What are the Implications of Changing the GSDML?

Further developments of the PROFINET technology may affect the GSDML. In the context of further development of the GSDML, it is ensured that the current GSD files are always compatible with a newer GSDML version. That means, an engineering tool with a newer GSDML version can also handle GSDs based on older GSDML versions (upwards compatibility).

A workaround in the other direction is also available. If a GSD based on a new GSDML Version is to be handled by an engineering tool which only supports an older GSDML version, then it can be specified as to which GSD objects should be over read by the (older) tool, without this leading to a fault behaviour. For this purpose, the attribute „RequiredSchemaVersion“(current version or later) for the objects DAP, module and submodule was introduced.

17.5.2 Do Further Developments of the Field Device Influence the GSD File?

In case the PROFINET field device is developed further and its functions are enhanced, then for the same PROFINET level or the same GSDML version, only a newer edition of the GSD file is required that includes the new functions.

If modules are added to a modular field device, renewed certification is not required, although the GSD file needs to be expanded indeed. Only if new functions are also added with new models, which were not testable before, then, a re-certification is required. Example: If a device did not support PROFIsafe earlier, and if now a PROFIsafe module is added, then the PROFIsafe part must be re-certified. This applies then to the DAP as well as to the actual PROFIsafe module.

If, from a newer PNIO level onwards new features are added and can thus only be described as a newer GSDML version, then the entire GSD must be upgraded to the newer GSDML version.

17.5.3 How do I Provide the GSD to my Customers?

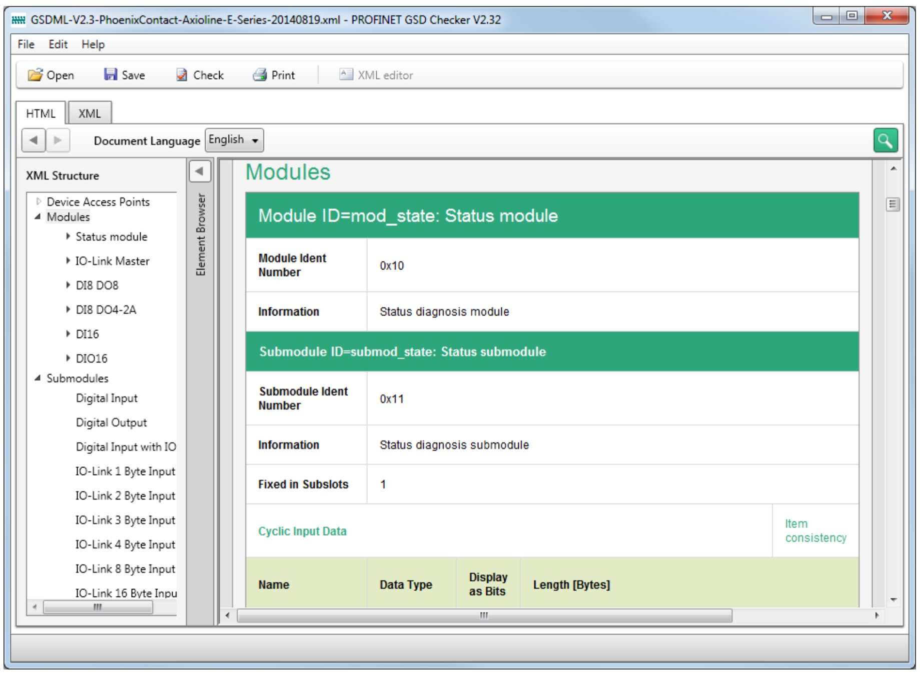
For the use of PROFINET field devices, the GSD file must be available and therefore is usually delivered with the field device. Many manufacturers also offer the GSD file along with the device presentation for download via Internet. Thus, the manufacturer can always provide the latest version. Such examples can be found in the Internet product catalogue of the field device vendors.

17.6 PROFINET GSD Checker Tool

The PROFINET GSD Checker is a piece of software for checking the GSD file. For PI members, this tool is available free of charge on the PI website http://www.profibus.com/nc/download/software-and-tools/downloads/profinet-xml-viewer- v22/display/ and it essentially offers two functions:

- Displaying the contents of GSD in a concise HTML representation (see Figure 17-4).
- Validation of a GSD against the schema files and reviewing the rules, which go beyond the schema and are documented in the GSDML specification (see Figure 17-5).

Above that, the checker also provides the schema documentation (a more user-friendly, better readable form of the schema) for GSD. A simple XML editor is already included in the PROFINET GSD Checker. However, the GSD Checker also allows the integration of another XML editor.



Modules

Module ID=mod_state: Status module

Module Ident Number	0x10
Information	Status diagnosis module

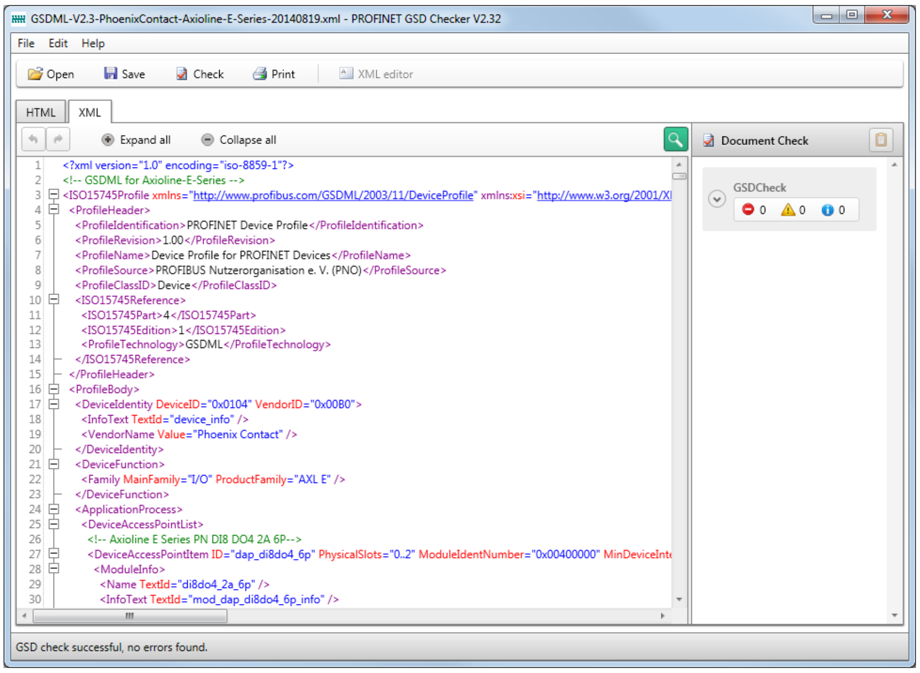
Submodule ID=submod_state: Status submodule

Submodule Ident Number	0x11
Information	Status diagnosis submodule
Fixed in Subslots	1

Cyclic Input Data — Item consistency

Name	Data Type	Display as Bits	Length [Bytes]

Figure 17-4: HTML representation in the GSD checker



```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- GSDML for Axioline-E-Series -->
<ISO15745Profile xmlns="http://www.profibus.com/GSDML/2003/11/DeviceProfile" xmlns:xsi="http://www.w3.org/2001/XI
 <ProfileHeader>
  <ProfileIdentification>PROFINET Device Profile</ProfileIdentification>
  <ProfileRevision>1.00</ProfileRevision>
  <ProfileName>Device Profile for PROFINET Devices</ProfileName>
  <ProfileSource>PROFIBUS Nutzerorganisation e. V. (PNO)</ProfileSource>
  <ProfileClassID>Device</ProfileClassID>
  <ISO15745Reference>
   <ISO15745Part>4</ISO15745Part>
   <ISO15745Edition>1</ISO15745Edition>
   <ProfileTechnology>GSDML</ProfileTechnology>
  </ISO15745Reference>
 </ProfileHeader>
 <ProfileBody>
  <DeviceIdentity DeviceID="0x0104" VendorID="0x00B0">
   <InfoText TextId="device_info" />
   <VendorName Value="Phoenix Contact" />
  </DeviceIdentity>
  <DeviceFunction>
   <Family MainFamily="I/O" ProductFamily="AXL E" />
  </DeviceFunction>
  <ApplicationProcess>
   <DeviceAccessPointList>
    <!-- Axioline E Series PN DI8 DO4 2A 6P-->
    <DeviceAccessPointItem ID="dap_di8do4_6p" PhysicalSlots="0..2" ModuleIdentNumber="0x00400000" MinDeviceInte
     <ModuleInfo>
      <Name TextId="di8do4_2a_6p" />
      <InfoText TextId="mod_dap_di8do4_6p_info" />
```

GSD check successful, no errors found.

Figure 17-5: XML representation in the GSD checker

17.7 Good Practices

17.7.1 Creating the GSD

A pragmatic approach for creating GSD is via the XML checker and the supplied examples (in the TPS Development Toolkit). Using the examples, the correct structure can easily be identified, and it can serve as the basis for the new product.

If built on the technology components (e.g. ASIC, Development Kit) as a platform, you get a functioning device software with sample application and sample GSD files. Also here, it must be ensured that the sample GSD is adapted to the individual field device.

17.7.2 Testing the GSD

The structure of an XML document can be tested with the help of a schema file. In this context, one speaks of the “validation” of a document. In this process, among other things, it is checked whether the element structure and the attributes used in the XML document comply with the schema definition. The schema file e.g. defines whether an attribute must be present and which values are allowed for the attribute.

For using the GSD Checker to check the GSD, no know-how about the function and use of schema files is required because these are automatically installed and used for validation by the PROFINET GSD Checker.

Also, one or two frequently used integrated engineering tools are also very useful to ensure that the GSD is processed as desired, that the graphics are displayed accordingly, and finally, whether the result then also meets the expectations.

17.7.3 Adaptation to a new GSDML schema

The required schema files e.g. for the GSDML 2:31 are:

- GSDML-CommNetworkProfile-v2.31.xsd
- GSDML-DeviceProfile-v2.31.xsd
- GSDML-Primitives-v2.31.xsd

To use the new schema files , they must be copied in the following directory:

\program files(x86)\PhoenixContactSoftware\PROFINET Configurator\Schemas\GSDML\.

After new start, the new schema is available, and the GSDML can be imported.

18. Wireshark Recordings

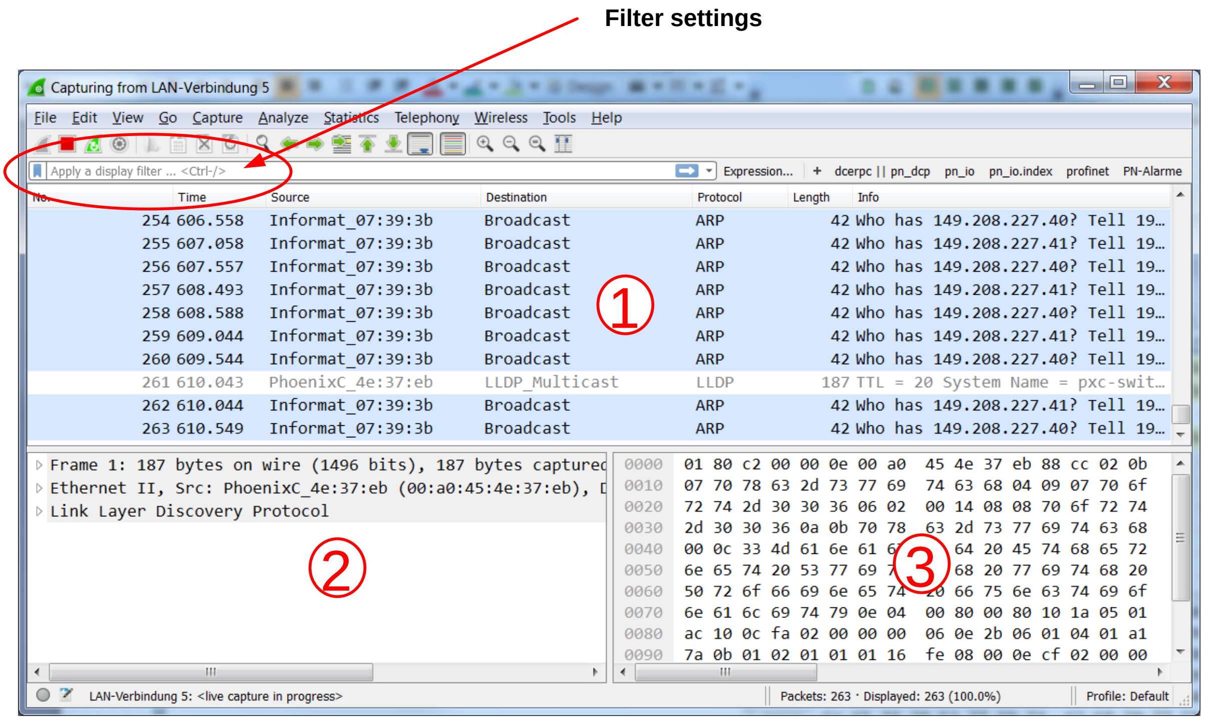
Note: The settings and the masks described in this chapter correspond to Wireshark Version 2.0.2. These may differ in other versions. Wireshark can be downloaded free of charge from the Internet.

Since a PROFINET network is established via switches, recording of data traffic is not possible without additional measures. If you connect a computer with Wireshark running via a normal switch port, then you do not see the frames exchanged because the switch forwards only to the port through which the device identified with the MAC address is accessible. So-called managed switches offer the possibility to operate a port as “mirror port” and to specify which port should be monitored on the mirror port.

The Wireshark tool provides data in the form of individual packets either during or after the recording of data traffic of a network interface (Ethernet connection). The data are clearly arranged and can be analysed.

In such data transaction, a large number of packets can be sent. To process these efficiently, the Wireshark provides filtering functions that can reduce the data flood.

Note: For the processing of communication problems in PROFINET, a comprehensive data traffic recording is a pre-condition for analysis.


Filter settings
Capturing from LAN-Verbindung 5
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
+ dcerpc || pn_dcp pn_io pn_io.index profinet PN-Alarme
①
②
③

```
No.     Time     Source            Destination     Protocol Length Info
    254 606.558  Informat_07:39:3b Broadcast       ARP        42 Who has 149.208.227.40? Tell 19…
    255 607.058  Informat_07:39:3b Broadcast       ARP        42 Who has 149.208.227.41? Tell 19…
    256 607.557  Informat_07:39:3b Broadcast       ARP        42 Who has 149.208.227.40? Tell 19…
    257 608.493  Informat_07:39:3b Broadcast       ARP        42 Who has 149.208.227.41? Tell 19…
    258 608.588  Informat_07:39:3b Broadcast       ARP        42 Who has 149.208.227.40? Tell 19…
    259 609.044  Informat_07:39:3b Broadcast       ARP        42 Who has 149.208.227.41? Tell 19…
    260 609.544  Informat_07:39:3b Broadcast       ARP        42 Who has 149.208.227.40? Tell 19…
    261 610.043  PhoenixC_4e:37:eb LLDP_Multicast  LLDP      187 TTL = 20 System Name = pxc-swit…
    262 610.044  Informat_07:39:3b Broadcast       ARP        42 Who has 149.208.227.41? Tell 19…
    263 610.549  Informat_07:39:3b Broadcast       ARP        42 Who has 149.208.227.40? Tell 19…
```

```
▷ Frame 1: 187 bytes on wire (1496 bits), 187 bytes capturec
▷ Ethernet II, Src: PhoenixC_4e:37:eb (00:a0:45:4e:37:eb), [
▷ Link Layer Discovery Protocol
```

```
0000  01 80 c2 00 00 0e 00 a0  45 4e 37 eb 88 cc 02 0b
0010  07 70 78 63 2d 73 77 69  74 63 68 04 09 07 70 6f
0020  72 74 2d 30 30 36 06 02  00 14 08 08 70 6f 72 74
0030  2d 30 30 36 0a 0b 70 78  63 2d 73 77 69 74 63 68
0040  00 0c 33 4d 61 6e 61 6[illegible] [illegible] 64 20 45 74 68 65 72
0050  6e 65 74 20 53 77 69 7[illegible]  68 20 77 69 74 68 20
0060  50 72 6f 66 69 6e 65 74  20 66 75 6e 63 74 69 6f
0070  6e 61 6c 69 74 79 0e 04  00 80 00 80 10 1a 05 01
0080  ac 10 0c fa 02 00 00 00  06 0e 2b 06 01 04 01 a1
0090  7a 0b 01 02 01 01 01 16  fe 08 00 0e cf 02 00 00
```

```
LAN-Verbindung 5: <live capture in progress>    Packets: 263 · Displayed: 263 (100.0%)    Profile: Default
```

Figure 18-1: Wireshark start screen

Figure 18-1 shows a typical start screen for Wireshark. The important fields are the packet list (1), the field packet details (2) and the field packet bytes (3).

To begin with recording, an Ethernet interface must be selected. Under the **Tab Capture -> Option**, an interface for the recording can be selected. With a push of the Start button the recording begins.

In the field **Apply a display filter**, a filter can be edited (e.g. **dcerpc || pn_dcp**) only to see the messages which are related to a connection set-up (Connect.Req). All other messages are hidden, but not deleted.

For meaningful PROFINET analysis, it is important to record the outgoing as well as the incoming data traffic. To achieve this, a managed switch or a TAP (Test Access Port) is required; Figure 18-2 illustrates its usage.

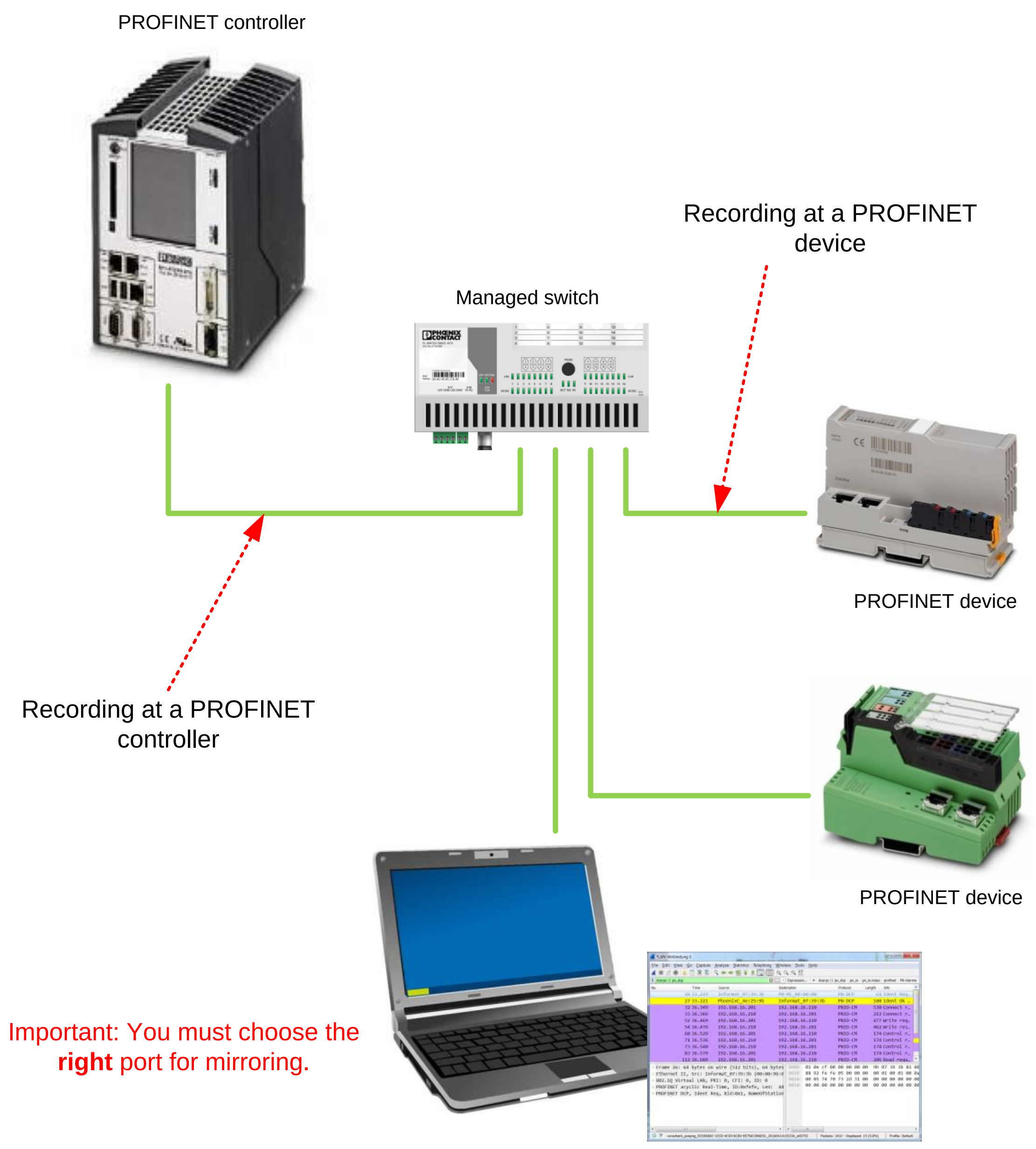

PROFINET controller
Recording at a PROFINET device
Managed switch
PROFINET device
Recording at a PROFINET controller
PROFINET device
Important: You must choose the **right** port for mirroring.

Figure 18-2: Measurements in a network

A network TAP however is inserted into the transfer path and provides all receive and send messages to the interface.

18.1 Filters for PROFINET

The following tables show some filters which are of significance for PROFINET data traffic analysis.

Protocol	Filter	Wireshark display (protocol column)	Description
DCP	**pn_dcp**	PN-DCP	**D**iscovery and Basic **C**onfiguration **P**rotocol - a data protocol as per IEC 61158, which is used in PROFINET to recognize and to configure the devices.
DCE/RPC	**dcerpc**	PNIO-CM	**D**istributed **C**omputing **E**nvironment/**R**emote **P**rocedure **C**all (DEC/RPC) – technology for the realisation of inter-process communication. It facilitates function calls in other address spaces (other computers).
RT	**pn_io**	PNIO	**P**ROFI**N**ET **IO** (PN_IO); PROFINET/RT: cyclic exchange of data
ARP	**arp**	ARP	**A**ddress **R**esolution **P**rotocol – the physical address (MAC) of a device is determined.
IP	**ip**		**I**nternet **P**rotocol – network protocol above the network access.
PN_PTCP	**Pn_ptcp**	PN_PTCP	This protocol is used for synchronization of the device clocks (e.g.. measurement of the line delay)
UDP	**udp**		**U**ser **D**atagram **P**rotocol – connectionless protocol belonging to the transport layer of the Internet protocol suite.
ICMP	**imcp**	ICMP	**I**nternet **C**ontrol **M**essage **P**rotocol – protocol for information exchange and error messages via the Internet protocol.

Table 18-1: Wireshark filter values

A filter can be set to analyse the data after recording is finished. But it is also possible to activate a filter during the recording so that only dedicated messages are shown.

18.2 Filter Proposals

For the analysis of a connection set-up e.g. the following filter combination may be useful:

pn_dcp || dcerpc

With the following filter, all PROFINET related messages except for the cyclic data (pn_io), can be filtered:

(dcerpc || pn_io.send_seq_num || udp.port == 0x0202) && !icmp

The following filter shows PROFINET messages including the cyclic data:

pn_io e.g.: pn_io.index==0xF841

The search for a particular IP address:

ip.addr==192.168.16.210

If you want to see all PROFINET alarms, you should choose this filter:

(pn_rt.frame_id>=0xFC00 && pn_rt.frame_id<=0xFCFF) ||
(pn_rt.frame_id>=0xFE00 && pn_rt.frame_id<=0xFEFC)

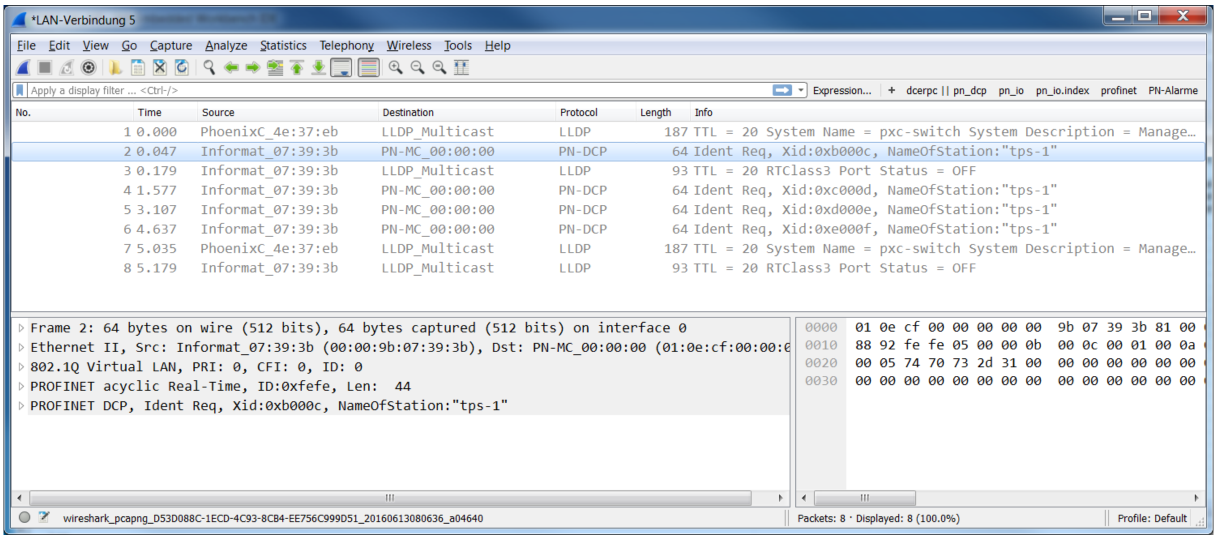
A reference to the possible filters and protocols can be found under the web address given below:

https://www.wireshark.org/docs/dfref/

18.3 Typical Problem Cases

18.3.1 Station Name is not Correct

The PROFINET Controller searches for the station „tps-1“. There is no response to the Ident Req in the 2nd message in Figure 18-3. The Ident Req is repeated several times. Here, the name assignment must be checked. The station could also not be accessed via the network (Check IP parameter).



```
No.  Time     Source               Destination       Protocol  Length  Info
  1 0.000   PhoenixC_4e:37:eb   LLDP_Multicast    LLDP       187 TTL = 20 System Name = pxc-switch System Description = Manage…
  2 0.047   Informat_07:39:3b   PN-MC_00:00:00    PN-DCP      64 Ident Req, Xid:0xb000c, NameOfStation:"tps-1"
  3 0.179   Informat_07:39:3b   LLDP_Multicast    LLDP        93 TTL = 20 RTClass3 Port Status = OFF
  4 1.577   Informat_07:39:3b   PN-MC_00:00:00    PN-DCP      64 Ident Req, Xid:0xc000d, NameOfStation:"tps-1"
  5 3.107   Informat_07:39:3b   PN-MC_00:00:00    PN-DCP      64 Ident Req, Xid:0xd000e, NameOfStation:"tps-1"
  6 4.637   Informat_07:39:3b   PN-MC_00:00:00    PN-DCP      64 Ident Req, Xid:0xe000f, NameOfStation:"tps-1"
  7 5.035   PhoenixC_4e:37:eb   LLDP_Multicast    LLDP       187 TTL = 20 System Name = pxc-switch System Description = Manage…
  8 5.179   Informat_07:39:3b   LLDP_Multicast    LLDP        93 TTL = 20 RTClass3 Port Status = OFF
```

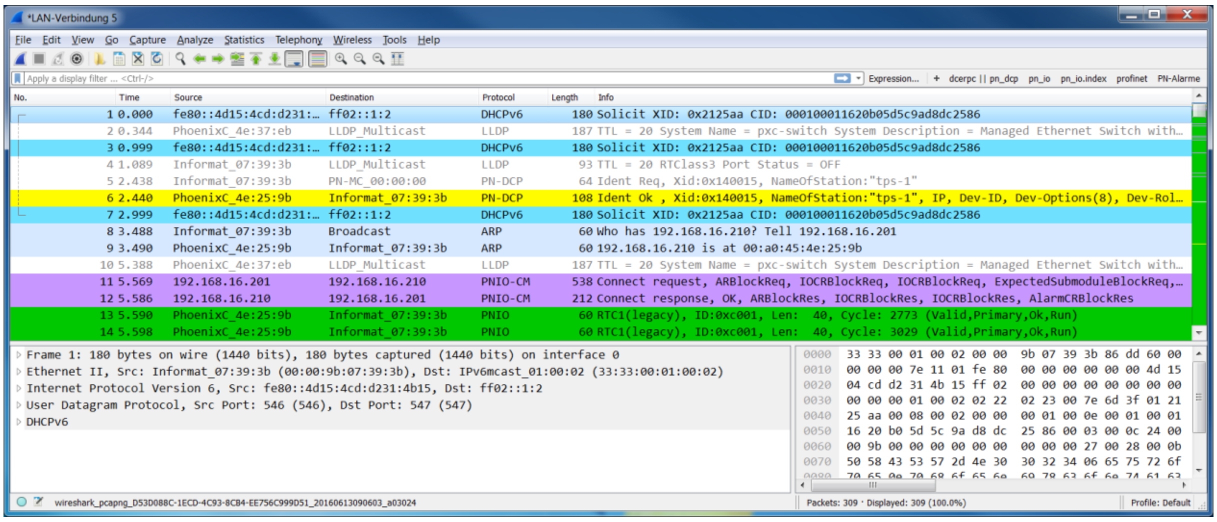
```
▷ Frame 2: 64 bytes on wire (512 bits), 64 bytes captured (512 bits) on interface 0
▷ Ethernet II, Src: Informat_07:39:3b (00:00:9b:07:39:3b), Dst: PN-MC_00:00:00 (01:0e:cf:00:00:0
▷ 802.1Q Virtual LAN, PRI: 0, CFI: 0, ID: 0
▷ PROFINET acyclic Real-Time, ID:0xfefe, Len:  44
▷ PROFINET DCP, Ident Req, Xid:0xb000c, NameOfStation:"tps-1"
```

```
0000  01 0e cf 00 00 00 00 00  9b 07 39 3b 81 00
0010  88 92 fe fe 05 00 00 0b  00 0c 00 01 00 0a
0020  00 05 74 70 73 2d 31 00  00 00 00 00 00 00
0030  00 00 00 00 00 00 00 00  00 00 00 00 00 00
```

wireshark_pcapng_D53D088C-1ECD-4C93-8CB4-EE756C999D51_20160613080636_a04640 | Packets: 8 · Displayed: 8 (100.0%) | Profile: Default

Figure 18-3: Searched station does not respond

Figure 18-3 shows correct identification of a station and the subsequent Connect.Req. After this the connection to the IO controller ramps up.



```
No.  Time     Source                  Destination        Protocol  Length  Info
  1 0.000   fe80::4d15:4cd:d231:…  ff02::1:2          DHCPv6     180 Solicit XID: 0x2125aa CID: 000100011620b05d5c9ad8dc2586
  2 0.344   PhoenixC_4e:37:eb      LLDP_Multicast     LLDP       187 TTL = 20 System Name = pxc-switch System Description = Managed Ethernet Switch with…
  3 0.999   fe80::4d15:4cd:d231:…  ff02::1:2          DHCPv6     180 Solicit XID: 0x2125aa CID: 000100011620b05d5c9ad8dc2586
  4 1.089   Informat_07:39:3b      LLDP_Multicast     LLDP        93 TTL = 20 RTClass3 Port Status = OFF
  5 2.438   Informat_07:39:3b      PN-MC_00:00:00     PN-DCP      64 Ident Req, Xid:0x140015, NameOfStation:"tps-1"
  6 2.440   PhoenixC_4e:25:9b      Informat_07:39:3b  PN-DCP     108 Ident Ok , Xid:0x140015, NameOfStation:"tps-1", IP, Dev-ID, Dev-Options(8), Dev-Rol…
  7 2.999   fe80::4d15:4cd:d231:…  ff02::1:2          DHCPv6     180 Solicit XID: 0x2125aa CID: 000100011620b05d5c9ad8dc2586
  8 3.488   Informat_07:39:3b      Broadcast          ARP         60 Who has 192.168.16.210? Tell 192.168.16.201
  9 3.490   PhoenixC_4e:25:9b      Informat_07:39:3b  ARP         60 192.168.16.210 is at 00:a0:45:4e:25:9b
 10 5.388   PhoenixC_4e:37:eb      LLDP_Multicast     LLDP       187 TTL = 20 System Name = pxc-switch System Description = Managed Ethernet Switch with…
 11 5.569   192.168.16.201         192.168.16.210     PNIO-CM    538 Connect request, ARBlockReq, IOCRBlockReq, IOCRBlockReq, ExpectedSubmoduleBlockReq,…
 12 5.586   192.168.16.210         192.168.16.201     PNIO-CM    212 Connect response, OK, ARBlockRes, IOCRBlockRes, IOCRBlockRes, AlarmCRBlockRes
 13 5.590   PhoenixC_4e:25:9b      Informat_07:39:3b  PNIO        60 RTC1(legacy), ID:0xc001, Len:  40, Cycle: 2773 (Valid,Primary,Ok,Run)
 14 5.598   PhoenixC_4e:25:9b      Informat_07:39:3b  PNIO        60 RTC1(legacy), ID:0xc001, Len:  40, Cycle: 3029 (Valid,Primary,Ok,Run)
```

```
▷ Frame 1: 180 bytes on wire (1440 bits), 180 bytes captured (1440 bits) on interface 0
▷ Ethernet II, Src: Informat_07:39:3b (00:00:9b:07:39:3b), Dst: IPv6mcast_01:00:02 (33:33:00:01:00:02)
▷ Internet Protocol Version 6, Src: fe80::4d15:4cd:d231:4b15, Dst: ff02::1:2
▷ User Datagram Protocol, Src Port: 546 (546), Dst Port: 547 (547)
▷ DHCPv6
```

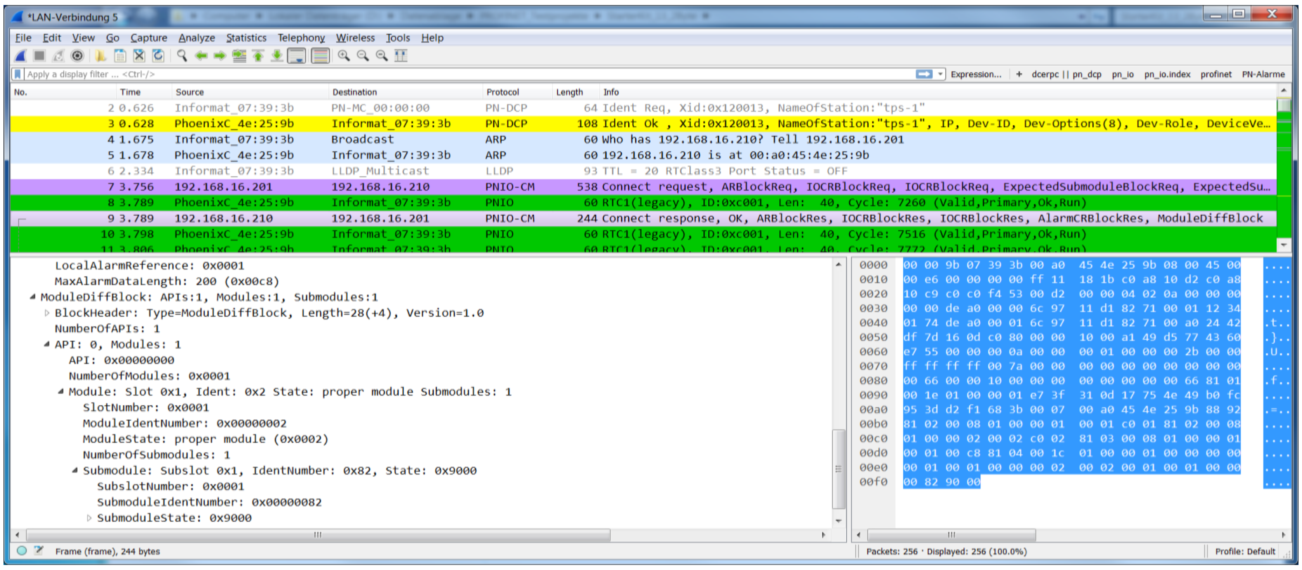
```
0000  33 33 00 01 00 02 00 00  9b 07 39 3b 86 dd 60 00
0010  00 00 00 7e 11 01 fe 80  00 00 00 00 00 00 4d 15
0020  04 cd d2 31 4b 15 ff 02  00 00 00 00 00 00 00 00
0030  00 00 00 01 00 02 02 22  02 23 00 7e 6d 3f 01 21
0040  25 aa 00 08 00 02 00 00  00 01 00 0e 00 01 00 01
0050  16 20 b0 5d 5c 9a d8 dc  25 86 00 03 00 0c 24 00
0060  00 9b 00 00 00 00 00 00  00 00 00 27 00 28 00 0b
0070  50 58 43 53 57 2d 4e 30  30 32 34 06 65 75 72 6f
```

wireshark_pcapng_D53D088C-1ECD-4C93-8CB4-EE756C999D51_20160613090603_a03024 | Packets: 309 · Displayed: 309 (100.0%) | Profile: Default

Figure 18-4: Station is identified correctly

18.3.2 A ModuleDiffBlock is Created During Connect Req

A PROFINET device generates a ModuleDiffBlock if the expected IO controller configuration does not match with the actual device configuration. You can detect in the ModuleDiffBlock, which problem has occurred. In the example shown in Figure 18-5, a submodule with wrong submodule number is plugged in the Slot 1, Subslot1 (here 0x82, expected 0x02).


*LAN-Verbindung 5
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
+ dcerpc || pn_dcp pn_io pn_io.index profinet PN-Alarme

```
No.   Time    Source              Destination         Protocol  Length Info
    2 0.626   Informat_07:39:3b   PN-MC_00:00:00      PN-DCP      64 Ident Req, Xid:0x120013, NameOfStation:"tps-1"
    3 0.628   PhoenixC_4e:25:9b   Informat_07:39:3b   PN-DCP     108 Ident Ok , Xid:0x120013, NameOfStation:"tps-1", IP, Dev-ID, Dev-Options(8), Dev-Role, DeviceVe…
    4 1.675   Informat_07:39:3b   Broadcast           ARP         60 Who has 192.168.16.210? Tell 192.168.16.201
    5 1.678   PhoenixC_4e:25:9b   Informat_07:39:3b   ARP         60 192.168.16.210 is at 00:a0:45:4e:25:9b
    6 2.334   Informat_07:39:3b   LLDP_Multicast      LLDP        93 TTL = 20 RTClass3 Port Status = OFF
    7 3.756   192.168.16.201      192.168.16.210      PNIO-CM    538 Connect request, ARBlockReq, IOCRBlockReq, IOCRBlockReq, ExpectedSubmoduleBlockReq, ExpectedSu…
    8 3.789   PhoenixC_4e:25:9b   Informat_07:39:3b   PNIO        60 RTC1(legacy), ID:0xc001, Len:  40, Cycle: 7260 (Valid,Primary,Ok,Run)
    9 3.789   192.168.16.210      192.168.16.201      PNIO-CM    244 Connect response, OK, ARBlockRes, IOCRBlockRes, IOCRBlockRes, AlarmCRBlockRes, ModuleDiffBlock
   10 3.798   PhoenixC_4e:25:9b   Informat_07:39:3b   PNIO        60 RTC1(legacy), ID:0xc001, Len:  40, Cycle: 7516 (Valid,Primary,Ok,Run)
   11 3.806   PhoenixC_4e:25:9b   Informat_07:39:3b   PNIO        60 RTC1(legacy), ID:0xc001, Len:  40, Cycle: 7772 (Valid,Primary,Ok,Run)
```

```
      LocalAlarmReference: 0x0001
      MaxAlarmDataLength: 200 (0x00c8)
  ◢ ModuleDiffBlock: APIs:1, Modules:1, Submodules:1
    ▷ BlockHeader: Type=ModuleDiffBlock, Length=28(+4), Version=1.0
      NumberOfAPIs: 1
    ◢ API: 0, Modules: 1
        API: 0x00000000
        NumberOfModules: 0x0001
      ◢ Module: Slot 0x1, Ident: 0x2 State: proper module Submodules: 1
          SlotNumber: 0x0001
          ModuleIdentNumber: 0x00000002
          ModuleState: proper module (0x0002)
          NumberOfSubmodules: 1
        ◢ Submodule: Subslot 0x1, IdentNumber: 0x82, State: 0x9000
            SubslotNumber: 0x0001
            SubmoduleIdentNumber: 0x00000082
          ▷ SubmoduleState: 0x9000
```

```
0000  00 00 9b 07 39 3b 00 a0  45 4e 25 9b 08 00 45 00   ....
0010  00 e6 00 00 00 00 ff 11  18 1b c0 a8 10 d2 c0 a8   ....
0020  10 c9 c0 c0 f4 53 00 d2  00 00 04 02 0a 00 00 00   ....
0030  00 00 de a0 00 00 6c 97  11 d1 82 71 00 01 12 34   ....
0040  01 74 de a0 00 01 6c 97  11 d1 82 71 00 a0 24 42   .t..
0050  df 7d 16 0d c0 80 00 00  10 00 a1 49 d5 77 43 60   .}..
0060  e7 55 00 00 00 0a 00 00  00 01 00 00 00 2b 00 00   .U..
0070  ff ff ff ff 00 7a 00 00  00 00 00 00 00 00 00 00   ....
0080  00 66 00 00 10 00 00 00  00 00 00 00 00 66 81 01   .f..
0090  00 1e 01 00 00 01 e7 3f  31 0d 17 75 4e 49 b0 fc   ....
00a0  95 3d d2 f1 68 3b 00 07  00 a0 45 4e 25 9b 88 92   .=..
00b0  81 02 00 08 01 00 00 01  00 01 c0 01 81 02 00 08   ....
00c0  01 00 00 02 00 02 c0 02  81 03 00 08 01 00 00 01   ....
00d0  00 01 00 c8 81 04 00 1c  01 00 00 01 00 00 00 00   ....
00e0  00 01 00 01 00 00 00 02  00 02 00 01 00 01 00 00   ....
00f0  00 82 90 00                                        ....
```

Frame (frame), 244 bytes | Packets: 256 · Displayed: 256 (100.0%) | Profile: Default

Figure 18-5: Connect Res with ModuleDiffBlock

19. Mechanical Requirements

19.1 Are There any Special Requirements for Housing and Plug?

Some PROFINET field devices must meet requirements for higher protection classes which provide protection against contact, water and dirt penetration. A commonly used protection class for this is e.g. IP65. IP stands for International Protection, and the number 65 means sealed against dust and water jets. Further protection classes are specified in the International Standard ISO 20653. If such requirements exist for a PROFINET field device then housing, plugs and sockets must comply with the respective IP protection class.

19.2 Which Cables Does PROFINET Use?

19.2.1 PROFINET-Copper Cabling

A PROFINET copper cable is typically a 4-wire, shielded copper cable (star quad).

The various cable types differ

- by the structure of wires (fixed / flexible)
- and/or the sheath.

The wires are colour coded.

In 4-wire cables, the cores of the pair 1 have a yellow and an orange insulation, whereas the wires of the pair 2 have a blue and a white insulation. The cores of the pairs are arranged crossed opposite to one another.

8-wire PROFINET copper cables consist of 4 pairs of lines with green, blue, orange and brown colours, and the associated white wire. Like in standard Ethernet applications, the maximum distance between the communication end-points for copper cables is limited to 100 m. This transfer path is defined as PROFINET end-to-end link.

19.2.2 PROFINET Fibre Optic Cabling

In areas where there is a likelihood of electromagnetic interference or high potential differences, you should use fibre optic cables for connecting automation islands and systems.

The use of fibre-optic cables eliminates the electromagnetic interference and earthing related equalizing currents through the shields of PROFINET copper cables.

The advantages of fibre optic transmission technology over copper cables are:

- FOCs bridge in general greater distances than copper cable
- FOCs provide electrical isolation between the coupled system components
- FOCs are insensitive to electromagnetic interference (EMI)

In PROFINET, four different types of fibres for fibre optic cables are used. The selection of a fibre type must be made by taking into account the demands made on the automation projects.

The following fibre types are available for selection:

- Plastic fibres with plastic optical fibre (POF)
- Glass fibre (multimode, single mode or hard cladded silica fibre (HCF))

Depending on the wavelength used and the respective attenuation, each fibre type is suitable only for a limited transmission distance. Table 19-1 lists the typical data for the beforementioned fibre types.

Fibre Type	**Core Diameter**	**Sheath Diameter**	**Transmission Distance (Typical values)**
POF	980µm	1000 µm	up to 50 m
HCF	200µm	230 µm	up to 100 m
Multimode	50 or 62.5 µm	125 mµ	up to 2000 m
Single mode	9 - 10 µm	125 µm	up to 14000 m

Table 19-1: Achievable transmission distances of FOC fibre types

19.3 Which Connectors are Available for PROFINET?

PROFINET cables are equipped with connectors on both sides. The combination of a connector on cable and socket is seen as a pair.

19.3.1 RJ45 Connectors for Copper Cables

The RJ45 connector is suitable for connecting a terminal device and network components. An important criterion for use of connectors is the controllability on-site. In the switch cabinet area, the RJ45 connector is used in its IP20 version. Outside the switch cabinet, the harsh environment must be taken into consideration. Here, the RJ45 push-pull in IP65 or IP67 version is used.

Also, the standard RJ45 has the advantage that it can be used to connect to laptops or engineering tools quickly and easily for servicing. The RJ45 connector in IP20 version is standardised in the IEC 60603-7. The RJ45 push-pull connector in IP67 is standardized in the IEC 61076-3-117 and is mainly used in the German automotive industry as a standard connector for PROFINET. Both versions are illustrated in Figure 19-1.

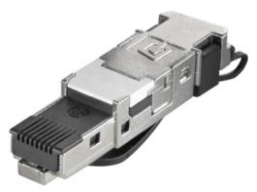
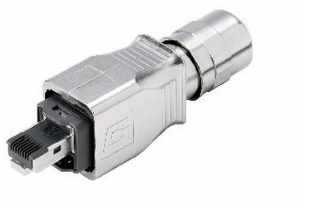
	
RJ45-connector in IP20	RJ45-push-pull-connector in IP67

Figure 19-1: RJ45 connector for copper cable (from [10])

19.3.2 M12 Connector for Copper Cables

For use in harsh industrial environments with IP67 protection, PI has specified the M12, a connector which provides a secure connection for sensors/actuators. The M12 is standardized in the IEC 61076-2-101.

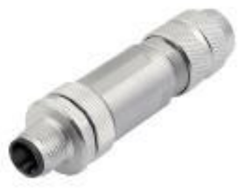
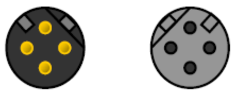
	
Encoding the connector face	
	
Connector	Socket
M12-D encoded connector in IP67	

Figure 19-2: M12 connector for copper cable (from [10])

19.3.3 Connectors for Fibre Optic Cables

For PROFINET data transmission via fibre optic, the SCRJ and the LC duplex connectors are provided. The basic versions of these connectors are designed for use in switch cabinets (IP20 protection class). For harsh environments or IP65 / IP67 requirements, the SCRJ push-pull version is used. An IP65 / IP67 connector with LC-Duplex connector is currently in the planning phase for the next revision of the PROFINET guideline.

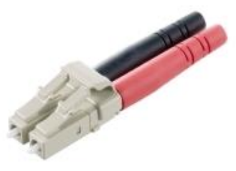
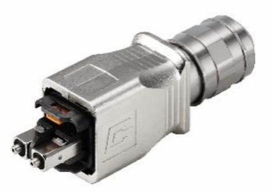
		
SCRJ-connector in IP20	LC duplex connector in IP 20	SCRJ-Push-Pull-connector in IP67

Figure 19-3: Connector for FO cable (from [10])

19.3.4 Connector Types BFOC and SC for FOC

The use of BFOC / 2.5 (IEC 60874-10) class connectors and the SC-plug system (IEC 60874-14) are not recommended for new automation systems.

19.3.5 Signal Connector

The standard 10-pin push-pull variant 14 connectors are used for PROFINET signal applications. These are characterized by the fact that up to 10 signal inputs can be plugged into a single push-pull connector. The connector is standardised in IEC / PAS 61076-3- 119 internationally.

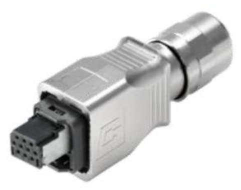

Signal connector push-pull connector, IP65

Figure 19-4: Signal connector (from [10])

19.4 What is Important While Integrating Plugs and Connectors into the Device?

The integration of copper or FOC interfaces follows the tolerance position on the printed circuit board and the wall thicknesses of the device as well as the tolerances of the employed interface. The device manufacturer must design the interface position optimally to ensure a secure connection of the device with the interface. Drawings for PROFINET copper and FOC connections, that allow easy integration of the interface, are available in the Guideline „Cabling and Interconnection Technology“. This guideline is available free of charge on the PI website for downloading.

http://www.profibus.com/download/installation-guide

Figure 19-5 and Figure **19-6** show examples from the guideline.

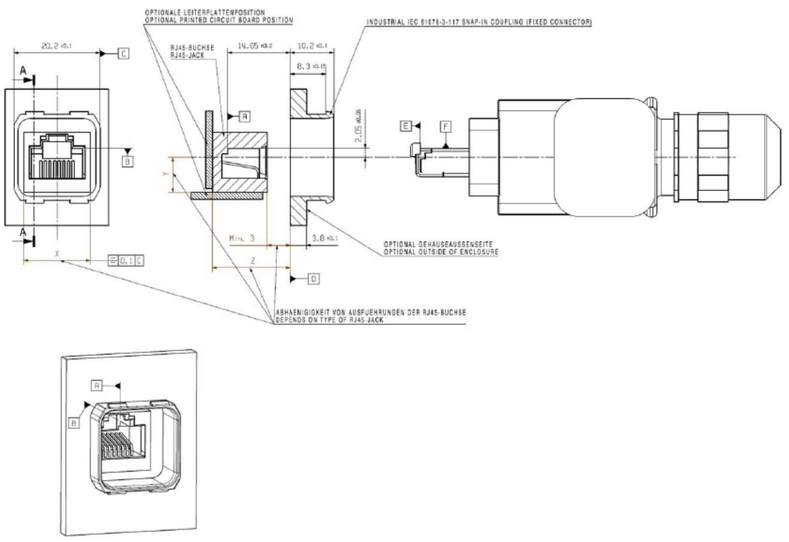

OPTIONALE LEITERPLATTENPOSITION
OPTIONAL PRINTED CIRCUIT BOARD POSITION
INDUSTRIAL IEC 61076-3-117 SNAP-IN COUPLING (FIXED CONNECTOR)
20.2 ±0.1
RJ45-BUCHSE
RJ45-JACK
14.65 ±0.2
10.2 ±0.1
8.3 ±0.05
2.05 ±0.05
A
B
C
D
E
F
X
Y
Z
Min. 3
3.8 ±0.1
0.1 C
OPTIONAL GEHAUSEAUSSENSEITE
OPTIONAL OUTSIDE OF ENCLOSURE
ABHAENIGIGKEIT VON AUSFUEHRUNGEN DER RJ45-BUCHSE
DEPENDS ON TYPE OF RJ45-JACK

Figure 19-5: Design rules for RJ45 push-pull type connector (from [10])

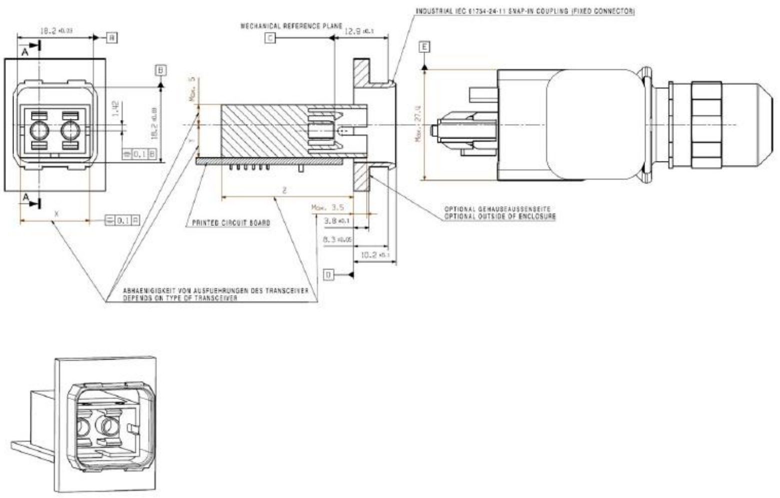

INDUSTRIAL IEC 61754-24-11 SNAP-IN COUPLING (FIXED CONNECTOR)
MECHANICAL REFERENCE PLANE
18.2 ±0.03
12.9 ±0.1
1.42
18.2 ±0.08
Max. 5
Max. 27.4
0.1 B
0.1 A
A
B
C
D
E
X
Y
Z
Max. 3.5
3.8 ±0.1
8.3 ±0.05
10.2 ±0.1
PRINTED CIRCUIT BOARD
OPTIONAL GEHAUSEAUSSENSEITE
OPTIONAL OUTSIDE OF ENCLOSURE
ABHAENIGIGKEIT VON AUSFUEHRUNGEN DES TRANSCEIVER
DEPENDS ON TYPE OF TRANSCEIVER

Figure 19-6: Design rules for SCRJ push-pull type connector (from [10])

19.4.1 Multiport Connector

When using RJ 45 Multiport Jacks, attention must be paid to the compatibility with industrial-grade IP20 RJ 45. Due to the industry-grade design, the field-installable connectors are built a bit larger than patch cable connectors in the office environment. Therefore, while using very compact Multiport Jacks, one may encounter incompatibilities with industrial-grade RJ 45 connectors. Figure 19-7 illustrates the problem.

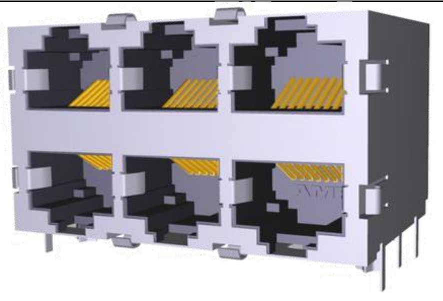
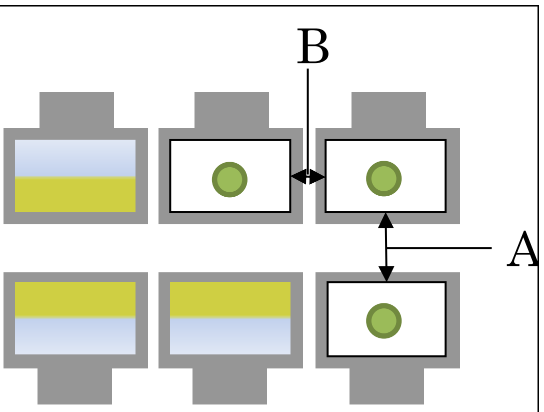
	
Standard Multiport Jack	Distance between individual ports

Figure 19-7: Multiport device (from [10])

Due to the non-standardised clearance in case of Multiport RJ45 Jacks, attention must be paid to the clearance between A and B while selecting the multiport jack. This should be dimensioned so that common, industry-grade RJ 45 connectors can be used.

Further information on the copper cabling or FO cabling can also be found in the Cabling and Interconnection Technology Guideline for PROFINET.

Note: More information for installation and earthing of copper cabling can be found in the PROFINET Installation Guideline (*PROFINET Montagerichtlinie Order No.: 8.071*). It is available free of charge for download.

http://www.profibus.com/download/specifications-standards/

19.5 What is Important for Shielding and Earthing?

PROFINET operates with shielded copper cables. Shielded cables are earthed at both ends of the cable connection. The earthing is mostly done to the shield of the RJ45 connector. This shield potential is connected directly to functional earth (FE) or protective earth (PE). For this earth potential a low impedance connection should be provided on the device (e.g., for 2.5-4 mm² Cu). The shield of the Ethernet cable is not a viable substitute for the local FE/PE connection on the unit - the cross-currents interfere with the Ethernet data transfer!

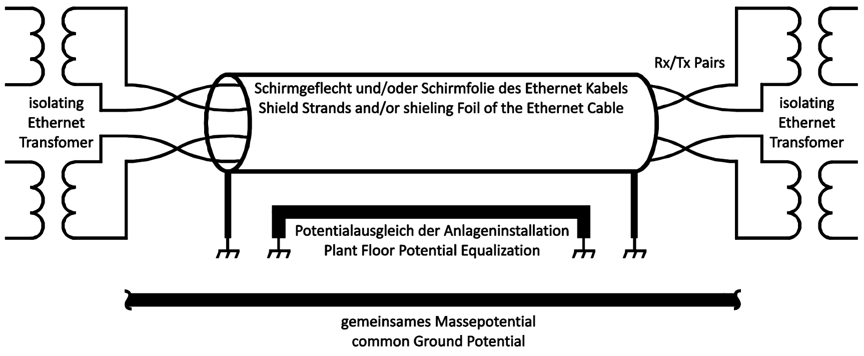

Rx/Tx Pairs
Schirmgeflecht und/oder Schirmfolie des Ethernet Kabels
Shield Strands and/or shieling Foil of the Ethernet Cable
isolating Ethernet Transfomer
isolating Ethernet Transfomer
Potentialausgleich der Anlageninstallation
Plant Floor Potential Equalization
gemeinsames Massepotential
common Ground Potential

Figure 19-8: Equipotential bonding and earthing (from [10])

Figure 19-9: 8-wire PROFINET cable (from [10])

In a DIN-compliant installation, a consistent earthing (equipotential bonding) is provided; cross-currents on the shield are therefore not to be expected. If the cross currents cannot be excluded or eliminated, the implementation of a fibre optic path is an effective measure.

Note: Additional information is available in the "PROFINET Design Guideline" (Version 1.14, December 2014; Order number 8.062; PROFIBUS User Organisation (*PROFIBUS Nutzerorganisation e.V.*)). It is available free of charge for download.

http://www.profibus.com/download/specifications-standards/

19.6 Should the MAC Address be Visible on the Device?

In the delivery state, no name is set for a PROFINET IO Device but rather it can only be accessed over a MAC address. This is permanently stored in the device, globally unique and, in general, it cannot be changed. The cyclic useful data exchange takes place only through addressing via MAC address when the controller (IO Controller) and the IO Device are on the same sub-network. Many PROFINET field devices have the MAC address printed on the housing or the nameplate. This eases commissioning and any subsequent search for the MAC address locally. For this reason, printing or laser-etching the MAC address is recommended on the device in such a way that it is readable even after installation.

19.7 Must LEDs be Mounted?

19.7.1 Status LEDs

Each PROFINET device must be supplied with at least one (1) Status LED - this LED displays the service **„DCP Signal“**. In TPS-1, the Link-LED of the Ethernet interface is used for this service. Basically, this service can also be provided through other methods - e.g. with a display. If an implementation different from an LED is considered, it is advisable to come to an agreement with the envisaged testing laboratory for certification about the different type of display before the development or design of the device.

Further requirements are not included in the PROFINET standard. TPS-1 has four status LEDs which are directly controlled by the ASIC (the circuit diagram is shown in the TPS-1 User’s Manual [1]). Table 19-2 summarizes the function of these LEDs.

LED	Color	Pin	State	Description
LED_BF_OUT	red	B13		Bus Communication:
			ON	No link status available.
			Flashing	Link status ok; no communication link to a PROFINET-Controller.
			OFF	The PROFINET-Controller has an active communication link to this PROFINET-Device.
LED_SF_OUT	red	B11		System Fail:
			ON	PROFINET diagnostic exists.
			OFF	No PROFINET diagnostic.
LED_MT_OUT	yellow	B10		Maintenance Required: Manufacturer specific – depends on the ability of the device.
LED_READY_OUT	green	C10		Device Ready:
			OFF	TPS-1 has not started correctly.
			Flashing	TPS-1 is waiting for the synchronization of the Host CPU (firmware start is complete).
			ON	TPS-1 has started correctly.

Table 19-2: TPS-1 Status LEDs for PROFINET

19.7.2 Link/Activity LEDs

Link/Activity LEDs indicate the status of the physical Ethernet connection and should be available for each port. They should be arranged so that the correlation between port and LED can be seen easily. Particularly during commissioning these indicators provide important information.

Note: As described above, the Link-LED is used for the „DCP Signal“ service. In a switch cabinet, the visibility of this LED must be ensured.

20. Certification of a PROFINET Field Device

20.1 Is it Mandatory to Certify for PROFINET?

Yes, a PROFINET field device must be certified. If you market your field device with PROFINET connection, it must be proven with a PROFINET certificate. Equipment manufacturers and end customers thus have the guarantee that the field device in a PROFINET system shows a standard-compliant behavior from the perspective of communication and therefore the interoperability between the PROFINET field devices from different manufacturers is guaranteed. Please keep in mind that the later a problem is detected, the higher the debugging costs usually are. The follow-up costs of a communication problem, which occurs in an automation system, are usually many times higher than the cost of certification.

20.2 General Procedure for Obtaining a Certificate

1. Make an appointment with a PI accredited test laboratory (PITL's) of your choice in good time during the development phase. Keep in mind that depending on the workload of the PITL's, a lead time of some weeks to months may be necessary to obtain a test date. Contact the PITLs well in time and ask for time constraints. PITLs also answer the questions related to the cost of certification. You can find a list of accredited PITL e.g. under the following link:

   http://www.profibus.com/pi-organization/institutions-support/test-labs/.

2. Each PROFINET device has a VendorID and a DeviceID, which are stored both in the GSD file and on the device itself. You can apply for a VendorID with the PI; it is valid worldwide.
3. Create a GSD file during the implementation phase. **Note:** Consider early, how you would describe your PROFINET device model in the GSD file. In case of questions, the PROFINET Competence Centre (PICC) would be glad to help you
4. On the certification date, deliver a production device and the corresponding GSD file to the PITL. Additional equipment and documentation, as far as needed for the commissioning of the unit during the certification test, must also be provided. After completion of the test, the certification laboratory generates a test report and hands it over to the device manufacturer.
5. After passing the test, on the basis of the test protocol, the device manufacturer can apply for a certificate at the PROFINET PNO office.

20.3 What Needs to be Clarified or Prepared for a Certification by the Manufacturer?

- A PROFINET device must always be submitted with the corresponding GSD file for certification.
- Each PROFINET device has a VendorID and a DeviceID both of which are stored in the GSD file and on the device itself, A VendorID is applied for only once and is valid worldwide. A DeviceID is assigned by the device manufacturer himself, whereby it must be ensured that the combination of Vendor and DeviceID must be unique worldwide so that there is exactly one PROFINET device which is described by its GSD file.
- The device manufacturer specifies in the test application, which DAP (Device Access Point) in the GSD file (in case it contains several DAPs) is to be tested. A DAP is the part of a field device, which includes the bus connection and the user program. In general only one DAP out of the GSD file is tested and is noted in the test protocol. You can find a short description of the DAP in the PROFINET System description (*PROFINET Systembeschreibung*), chapter 2.2. „Device model of a device.“
- Specify the conformance class (A, B or C) targeted for the device. Depending upon the conformance class entered in the GSD, additional certifications tests are carried out (e.g. testing the isochronous operation for conformance class C). For a short description of conformance classes, please refer to PROFINET system description (*PROFINET Systembeschreibung*), chapter 1.1 „Conformance Classes.“
- Optionally, an application class can be specified in the GSD file. Certain minimum functionalities for the application are assigned per definition to an application class. The certification lab then checks by the entered application class, whether the device complies with all the assigned functions of this application class properly (see chapter 17.4.2).
- Within the framework of certification test, a network load test (Security Level 1 test) is also prescribed. The device manufacturer specifies according to which network load class (1, 2 or 3) the device is to be tested. Network load class 1 thereby has the lowest, class 3, the highest requirements. For more information about the network load test, refer to the document „Guideline PN IO Netload“; this is a constituent part of the PROFINET test bundle which can be downloaded free of charge by PI members from the PNO website. The link to this is: http://www.profibus.com/nc/download/software-and-tools/downloads/profinet-io-test-bundle/display/
- Compliance with the respective national and international regulations must be ensured by the manufacturer.
- For a certification test according to conformance class C, a synchronization pin to allow testing of the isochronous operation of the device with an oscilloscope must be accessible. This pin must be taken into account already during the design phase of the device. As it is required only for the certification, this pin is not necessary for the later production later for the production devices

20.4 Can I Continue to use the Certificate of the Technology Provider?

No, this is not possible. When using pre-certified technologies, you as a device manufacturer, however, do not have to be familiar with all the details of the PROFINET standard indeed, and the risk that errors occur during the tests is reduced significantly.

A PROFINET certification test is always the test of a complete device comprising PROFINET protocol unit (e.g. HW + Protocol stack + GSD) and the application. This is necessary because the interaction between the application and PROFINET protocol unit (operating the PROFINET user interface through the application) can affect the execution of the PROFINET protocol. The certification, therefore, is intended to rule out that an application malfunction or the protocol unit may lead to malfunctions of the PROFINET communication.

20.5 Checklist for PROFINET Certification

Before PROFINET certification, the following items must be checked:

- The I&M0 data must be correctly filled with your own values.
- If I&M1, I&M2 and I&M3 parameters are present, they must be stored remanently. With a ResetToFactory, these must then be deleted. If necessary you can use the 4 kByte Flash Area inside the TPS-1 flash (see reference guide) for this purpose.
- Connect Tsync signal to a pad that is accessible during the conformance testing (only required for conformance class C)
- Acyclic record data index 0x8028 and 0x8029 must be processed by the application. Please check each subslot.
- Fill in your own software version (in function ***configDevice(void)*)*****.** The numbering must be the same as in the GSDML.
- For each subslot used the consumer status (IOCS) and the provider status (IOPS) must be set correctly. First set must be done in the function ***onPrmEndReq()*** (event **TPS_EVENT_ON_PRM_END_DONE_IOARx**). This is necessary to send a correct status before “Application Ready” is sent to the controller.
- Check the available NumberOfARs. In some cases you need two - but normally only one (IOC-AR).
- The settings of the slots and subslots (ID’s) must be identical in the GSDML and in the program when configuring (***configDevice(void)***).
- Initial parameters must be checked (requires the subslot initial parameter?). The GSDML file must not contain any parameters that are not supported by the application.

20.6 Which Tests are Performed for Certification?

A number of test systems covering certain aspects are used for certification. This chapter briefly describes the test systems used including the test objectives.

The test systems are largely automated. The general test procedure and especially the given test setup is described in the test specification (e.g. for PROFINET version V2.34.1 -> Testspec-PN_2572_V234_Feb17.pdf). It is mandatory to configure the test setup and the devices described there. In most test cases you need additional hardware. Neighborhood information from these test setups is used in many places to test interoperability. Table 20-1 summarizes the available tester packages and their scope.

Test name	Purpose
Automated RT-Tester	Real Time and Interoperability Test
SPIRTA Test Bundle	IRT Test (dedicated IRT hardware required)
TED-Check	Topology test
Net Load Test	Security Level 1 test (there are three Netload classes defined)
GSD-Checker	Checking GSDML for formal correctness

Table 20-1: Used test systems for certification

21. Checklist for Hardware Development (TPS-1)

The items listed here are to be seen as notes supplementing the TPS-1 user's manual [1]. Detailed information can be found there. Before starting with the board layout; these items must be reviewed again.

21.1 Power Supply Concept

In the TPS-1 data sheet, a power supply proposal is made. Besides the supply to the digital sections (1.0V, 1.5V, and 3.3V), also analogue circuits exist that need to be decoupled. Based on the overview in Figure 21-1 these power supplies should be checked once again before starting layout. It is important at this point that the DGND and AGND are separate from each other connected at a suitable point via e.g. an inductor.

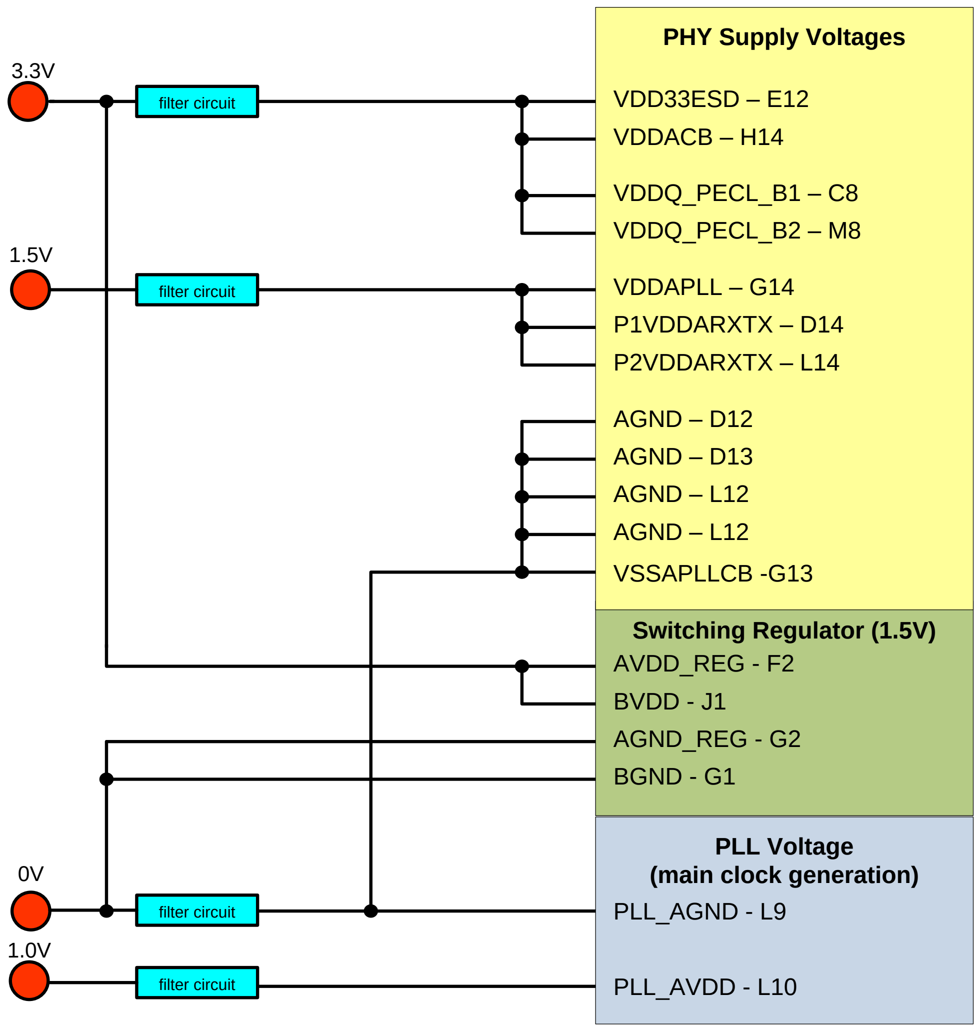


Figure 21-1: Power supply concept for TPS-1

21.2 JTAG Interface Circuit

The JTAG circuitry should be rechecked once again especially under the aspect of wiring the TRSTN signal; if you do not follow this recommendation, sporadic ramp-up problems may pop up.

21.2.1 Unused JTAG Interface

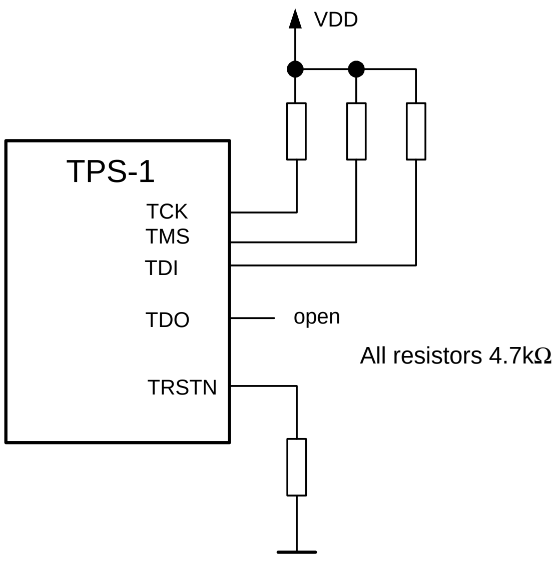


Figure 21-2: Unused JTAG interface

21.2.2 Used JTAG Interface

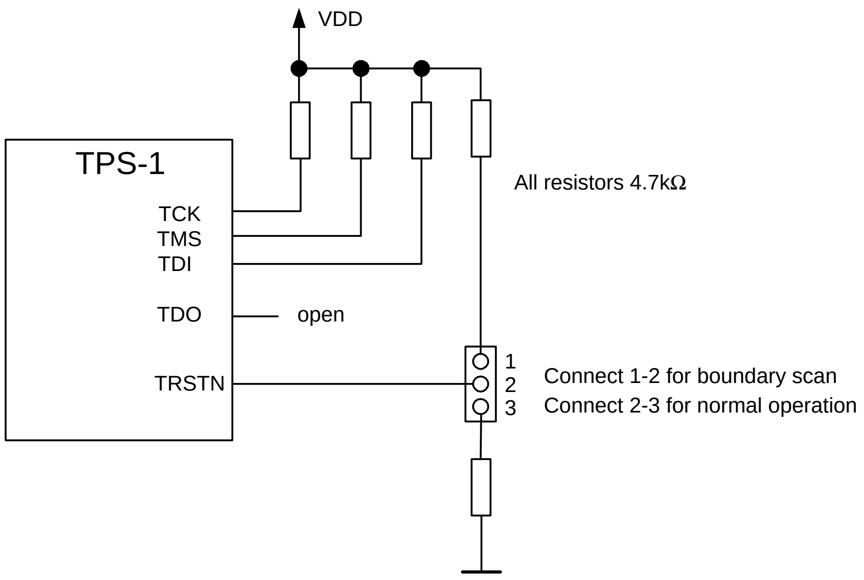

VDD
TPS-1
All resistors 4.7kΩ
TCK
TMS
TDI
TDO
open
TRSTN
1
2
3
Connect 1-2 for boundary scan
Connect 2-3 for normal operation

Figure 21-3: Used JTAG interface (for boundary scan)

21.3 Setting the Switching Regulator / POR

These input signals are used to control on-chip functions. These signals should be rechecked.

- TEST1 (H3)
- TEST2 (G3)
- TEST3 (E1)

The pins mentioned above control the regulator and POR and must be connected correctly as shown in Table 21-1.

TEST3 (Pin E1)	TEST2 (Pin G3)	TEST1 (Pin H3)	Function
0	0	0	Normal mode: regulator and POR on.
0	0	1	Only POR mode: regulator off, POR on.
0	1	0	Regulator and POR circuitry switched off [1]
			All other options reserved for test

Note 1: This setting keeps the TPS-1 permanently in reset and cannot be used for normal device operation.

Table 21-1: Switching regulator and POR settings

21.4 Power Supply and RESET

The TPS-1 provides a POR, which releases the TPS-1 RESET only after the required supply voltage (3.3 V) has ramped up. The TPS-1 does not monitor any other required voltage. Such monitoring and the necessary triggering of a RESET in the case of too low voltage must be done by the power supply module.

21.5 Other Individual Signals

Signal group	Signal name	Description
Production test	TMC1 (E10)	Pull down (10k)
	TMC2 (K10)	Pull down (10k)
	TESTDOUT5 (D8)	Can be left open (Test PHY)
	TESTDOUT6 (D9)	Can be left open (Test PHY)
	TESTDOUT7 (L8)	Can be left open (Test PHY)
	TEST_1_IN (D6)	Pull down (10k)
	TEST_2_IN (D7)	Pull down (10k)
PHY	EXTRES (H13)	Resistor (12k4) to AGND
Boundary scan	TM0 (L4)	Pull down (1k)
	TM1 (J10)	Pull down (1k)
IRT signals	TEST_SYNC (N12)	Clock signal for certification – should be connected to a pad
	T1 (J11)	If IRT is used connect to interrupt input of the host CPU (if not used, leave open).
	T2 (H12)	If IRT is used connect to interrupt input of the host CPU (if not used, leave open).
	T3 (G11)	If IRT is used connect to interrupt input of the host CPU (if not used, leave open).
	T4 (F11)	If IRT is used connect to interrupt input of the host CPU (if not used, leave open).
Boot loader	BOOT_1 (P12)	Pull Down (10k), signal should be configurable (if set to high, the internal boot loader is started after RESET).

Table 21-2: Individual signals

21.6 LED Status Signals

The status LEDs must be connected to Vdd (via a resistor). When active, the TPS-1 output is set to logical 0.

LED	Pin	Colour	Function
LED_BF	B13	red	Bus fail occur
LED_SF	B11	red	System fail occur
LED_READY	C10	green	Device ready
LED_MT	B10	yellow	Maintenance demanded / required

Table 21-3: Overview status LEDs

The functional meaning of the LEDs is described in the TPS-1 device hardware documentation [1].

21.7 Network Interface

To make best use of the capabilities of the PHYs, that are included in the TPS-1, the network interface should be designed carefully. Careless network interface designs will not necessarily result in immediate device failures, but in reduced usable cable length and more frequent transmission errors.

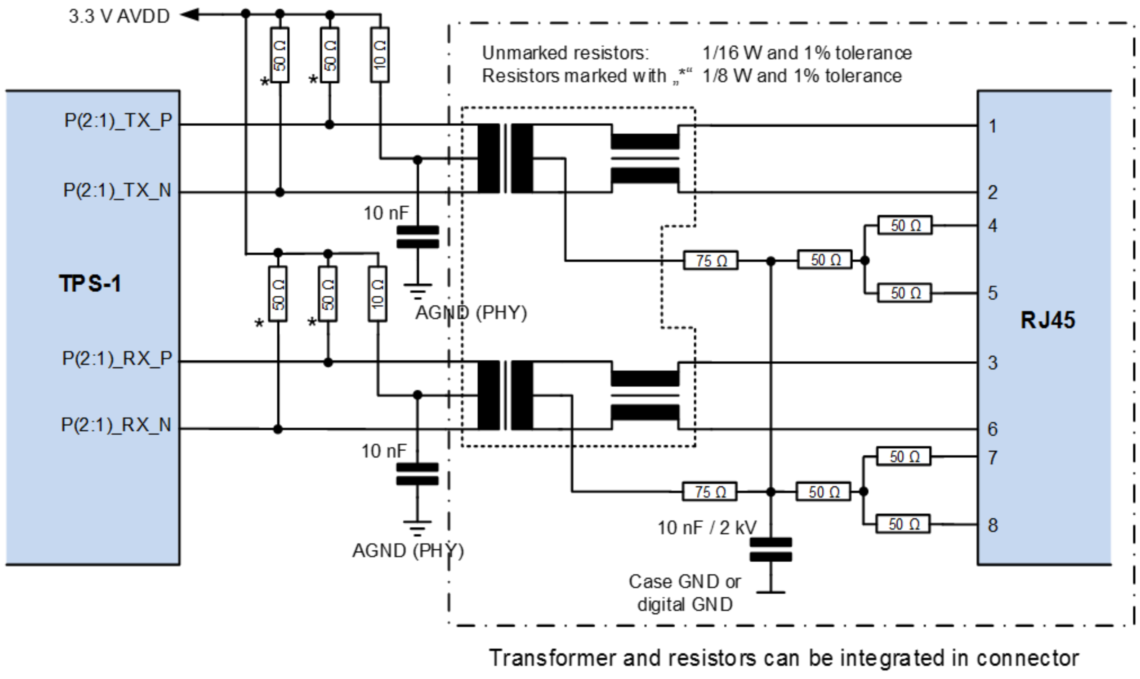

3.3 V AVDD
Unmarked resistors: 1/16 W and 1% tolerance
Resistors marked with „*“ 1/8 W and 1% tolerance
P(2:1)_TX_P
P(2:1)_TX_N
TPS-1
P(2:1)_RX_P
P(2:1)_RX_N
50 Ω
10 Ω
10 nF
AGND (PHY)
75 Ω
10 nF / 2 kV
Case GND or digital GND
1
2
4
5
3
6
7
8
RJ45
Transformer and resistors can be integrated in connector

Figure 21-4: Recommended circuit for network interface

Figure 21-4 shows the recommended circuit for a typical copper-based network interface. Your design considerations should touch the following points:

- RJ45 connectors with integrated transformers or usage of discrete transformers
  Integrated transformers will save cost and board space, while discrete transformers are regarded as higher quality and allow easier implementation of additional ESD protection circuitry. Recommendations for connectors and transformers are described in the TPS-1 user manual [1].
- selection of proper analogue supply voltages and GND
  Analogue supply voltage should as well be used for the external pull-up resistor in Figure 21-4.
- Signal lines between TPS-1 and the RJ45 connector
  The signal lines should not only be short, but must be also routed as impedance controlled lines. Furthermore you should avoid crossings between the Ethernet signal lines and other “noisy” signals like clock or address/data buses.

TPS-1 based systems that works in a electrically very noisy environment may need additional HW countermeasures against ESD. Figure 21-5 illustrates a typical circuit for this purpose.

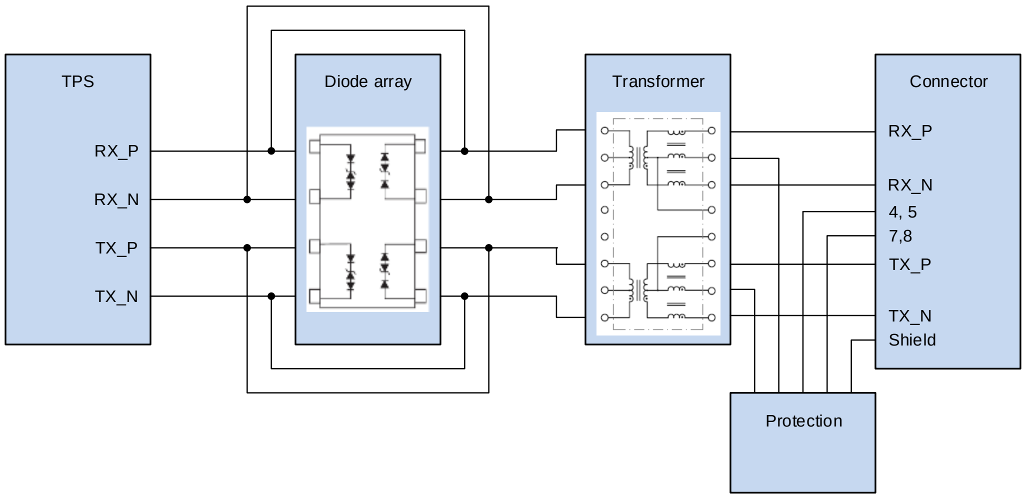


Figure 21-5: Typical ESD protection circuit

21.8 Reset Concept

In case of a reset, it should be noted that the TPS-1 can also always be reset by the host CPU. Situations may arise which can only be overcome by a reset. A detailed description is available in the TPS-1 device hardware documentation.

21.9 Watchdog Concept

The TPS-1 provides a signal (WD_OUT, pin B12) that indicates to the connected host CPU that a watchdog is triggered in the TPS-1. If necessary, after the TPS-1 reset, the NRT areas must be reconfigured (like in the case of reset).

For monitoring the connected host CPU, TPS-1 provides the input WD_IN (pin A11). This pin must be triggered by the host CPU (can also be switched off completely). If there is no trigger, then the TPS-1 switches off its peripheral interface and can no longer be addressed by the host CPU. This situation can only be resolved by TPS-1 reset. Here, the monitoring concept must be checked again.

Note: The WD_OUT and WD_IN signals should be monitored under all circumstances in order to avoid undefined system states.

21.10 Device Identification – MAC Addresses

For the TPS-1 operation, three MAC addresses are needed:

- Device interface
- Ethernet port 1
- Ethernet port 2

Each Ethernet port requires a MAC address to identify it uniquely.

The MAC addresses - in addition to the serial numbers - are assigned during the manufacturing process and are permanently written in the TPS-1 configuration. The MAC address of the device should also be readable on the device housing in installed state (e.g. in a switch cabinet).

The MAC addresses are entered with the TPS Configurator and stored in the TPS-1 external Flash.

MAC addresses can be purchased from the *Profibus User Organisation* (smaller quantities) or the IEEE.

Profibus User Organisation:	www.profibus.com or www.profinet.com
IEEE:	http://standards.ieee.org/develop/regauth/oui/index.html

22. Checklist for Software Implementation

The subsequent Table 22-1 lists a couple of check items to verify the status of the software implementation.

Test item	Result:
VendorID changed to own identifier	
DeviceID entered (own administration)	
Serial number - clarified creation and administration issues	
I&M0 data creation (initImData(void) (mandatory for each device)	
I&M1 -3 data retention must be ensured by the device firmware (for certification)	
Correct software version must be implemented (configDevice() -> SoftwareVersion)	
Implementation Record Request for index **0x8028** (RecordInputDataObjectElement for one Subslot – enhance AppOnReadRecord())	
Implementation Record Request for index **0x8029** RecordOutputDataObjectElement for one Subslot - enhance AppOnReadRecord())	
Implemented diagnostic alarm (if requested)	
Implemented process alarms (if requested)	
Tested **Reset to Factory Settings** (reset option 2 and 8 or more if necessary)	
GSD file – change VendorID, DeviceID and manufacturer	
Set **IOPS** for input data and set the **IOCS** (cyclic data IO-RAM) (TPS_WriteInputData())	

Table 22-1: Software checklist

A. Abbreviations and Terms

AGND	Analogue Ground Plane
AIDA	Automation Initiative of German Automobile Manufacturers (*Automatisierungsinitiative Deutscher Automobilhersteller*)
ANSI C	C Programming Language development as per ANSI X3.159-1989
APDU-Status	Application Protocol Data Unit Status – contains status information related to the cyclic transfer of user data (Cycle Count, Data Status, and Transfer Status)
API	Application Programming Interface
API	Application Process Identifier (PROFINET)
AR	Application Relation - Application relation between a provider and a consumer (PROFINET).
ASE	Application Service Entity
ASIC	Application specific integrated circuit
CC	Conformance Class
CDT	C/C++ Development Toolkit
Connect.Req	Connect.Request: Message, with which a controller attempts to establish a connection to a field device
Connect.Res	Connect.Response: Response of a field device to a controller attempting to establish a connection
Controller-AR (IOS)	The IOC AR (AR Controller) serves to exchange cyclic input and output data within a unicast or multicast connection, acyclic data via Read/Write services and bidirectional alarms
CR	Communication Relation
DAP	Device Access Point
DCP	Discovery and Configuration Protocol; this protocol is used e.g. to set the „Name Of Station“, to transfer and set the IP address and other parameters
Device Access AR (DA)	Device Access refers to the build-up of an AR between a higher level controller and a device. The device access allows cyclic reading and writing of data.
DeviceID	Device identification assigned by the device manufacturer
DGND	Digital Ground Plane
DPRAM	Dual-Port-RAM, a RAM, in which it is possible to have simultaneous read or write accesses from two sides
Driver	Collective term for the functions and structures of the Driver interface, which can be used by a host application
EMC	Electromagnetic Compatibility
FPGA	Field Programmable Gate Array
GCC	GNU Compiler Collection
GDB	GNU Project Debugger
GNU	A UNIX-based operating system (GNU-Project)

GPIO	General Purpose Input / Output – Signal pin of an integrated circuit, which, depending upon programming can be configured as an input or an output pin
GSD	General Station Description
GSDML	GSD Markup Language
Hardware-Revision	Specifies the hardware version of the field device is queried in I&M0 data as well.
IP20	protected against solid foreign bodies having diameter above 12.5 mm
IP65	sealed against dirt and water jets
IP67	sealed against dust and temporary immersion
I&M-Functions	Identification and Maintenance; services that provide support during commissioning and maintenance to read informative data from a field device (e.g. version identifications).
ICMP	Internet Control Message Protocol; Internet protocol for reporting faults in a network environment
Implicit AR (IMP)	The implicit AR (ARUUID = 0) defines an application relationship between controller/supervisor and device for acyclic reading of data from a device. The controller must not establish a separate AR for this purpose. It is always existent and can be used by the higher level controller.
IOCS	Input Output Consumer Status
IOPS	Input Output Provider Status
IP	Internet Protocol; the protocol that ensures the transfer of data on the Internet from the end node to end node
IRT	Isochronous Real-time
LLDP	Link Discovery Protocol Data (IEEE 802.1 AB); the LLDP is a vendor independent Layer-2 protocol that provides the ability to exchange information between neighbouring devices
M12	Circular connector with metric thread
MAC-Address	Media Access Control Address; this is also referred to as an Ethernet address and serves to identify an Ethernet node. The Ethernet address is 6 bytes long (IPv4) and is assigned by the IEEE
MAU	Medium Attachment Unit (physical interface of a port)
ModuleIdentNumber	Unambiguous description of a module that must be uniform across a field device
NameOfStation	Individual device name, by which a controller can find a device
NRT-Area	Non-Real Time-Area; storage area in the DPRAM of the TPS-1 starting at address 0x8000, which contains the device configuration and mailboxes
Outline View	An Outline View is a summary of the contents of a structured file (variables, structures, functions, classes, etc.).
PDEV	Physical Device
Peer-to-Peer	Peer-to-Peer (P2P) connection (English: peer = „equals", „Coequal”) and computer to computer connection are synonymous designations for communication among equals; here, referring to a computer network (see Client-Server System)

PNO	PROFIBUS User Organisation (PROFIBUS Nutzerorganisation e.V.) (http://www.profibus.com/)
POF	Polymeric optical fibre
POR	Power On Reset
Project Explorer	The Project Explorer provides an overview of packages and their contents are belonging to a project.
RJ45	standard Ethernet connector
RTA	Real Time Acyclic
RTC	Real Time Cyclic
Slot	A PROFINET addressing level found in automobiles
SCRJ	Connector for FOC transmission
SubmoduleIdentNumber	This identification must be unique within the submodules of a specific module.
Subslot	Insertion slot in a real device within a slot; the subslot is that part of the device over which data are exchanged.
Supervisor AR (IOS)	The Supervisor AR is provided for data exchange of a supervisor with a device and has the same features as an IOC-AR.
TAP	Test Access Port; passive access point of a network connection over which data can be read for analysis purposes
TCP	Transmission Control Protocol; superimposed protocol of IP, to ensure secure data exchange and flow control
TED	Topology Engineering and Discovery (part of test package for certification)
TFTP	Trivial File Transfer Protocol; service to access files on a remote computer over a network
TLV	Type Length Value; format that is used in network protocols and file formats to transfer a variable number of attributes e.g. in a message
TPS Stack	TPS-1 Firmware for operating the PROFINET communication system
TPS Updater	Separate part of the firmware with which the firmware can be updated
VendorID	Manufacturer identification assigned by the PROFIBUS user organisation
UDP	User Datagram Protocol; insecure Multicast- / Broadcast telegram
UUID	Universal Unique Identifier; regulates the unique identification of a certain functionality or data in PROFINET
XML	Extensible Markup Language

B. List of Documents

[1] **TPS-1 User Manual**, document number R19UH0081ED0105 or newer, Renesas Electronics

[2] **TPS-1 Reference Manual**, included in TPS-1 Development Toolkit version V1.3.1.16 or newer

[3] **TPS-1 Document Change Notification**, document number R19TU0002ED0200 or newer, Renesas Electronics

[4] Jäger, Edgar: **Industrial Ethernet**, Heidelberg, Verlagsgruppe Hüthig, ISBN 978-3-7785-4031-2

[5] **Renesas Semiconductor Package Mount Manual**, (Rev.1.01, Mar 2011) (R50ZZ0003EJ010)

[6] **GSDML Getting Started** included in TPS-1 Development Toolkit version V1.3.1.16 or newer
Simple Instructions for Creating a Device Description Data (GSD) file for PROFINET Version February 2015, Phoenix Contact Software GmbH

[7] **TPS-1 Firmware Update Manual**, version V1.4 included in TPS-1 Development Toolkit version V1.3.1.16 or newer

[8] **Diagnosis for PROFINET IO**; Guideline for PROFINET Version 1.1, March 2015; Order number 7.142; PROFIBUS Nutzerorganisation e.V.

[9] Popp, Manfred: **Industrial communication with PROFINET (*Industrielle Kommunikation mit PROFINET*)** PROFIBUS Nutzerorganisation e. V., 2014

[10] **PROFINET Design Guideline**;
Version 1.14, December 2014; Order number 8.062; PROFIBUS Nutzerorganisation e.V.

Website and Support

Renesas Electronics Website
  http://www.renesas.com/

Inquiries
  http://www.renesas.com/contact/

Revision History

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	Nov. 3, 2016		1st edition
2.00	Jan. 12, 2018		Major update - added information related to YCONNECT-IT-TPS-1L - added chapter 2.4.5 - added chapter 5.2 - extended chapter 10 - extension in Table 11-1 - added chapter 16.5 - extension in Table 17-1 - various minor corrections, typos etc.
3.00	April 11, 2019		Major update - added Table 4-1 - extension and modification in Table 7-2 - added new Figure 10-3 in chapter 10.2 - extended chapter 16.6.3 - added Table 12-1 - extension and modification in Table 22-1 - added new chapter 11.4 Extension of the Host API for selective reception of Ethernet frames - various minor corrections, typos etc.
3.01	March 18, 2020		Minor update - updated Table 21-2 , TEST_1_IN and TEST_2_IN - pull down (10k) - new photo in Figure 2-9 and a note available - adjust text in chapter 16.4.1 , 16.4.3 and 18.2
3.02	August 9, 2024		Minor update - changes in chapter 14 - additional note in chapter 15.3.2 - changes comment 0x81C9 in Table 2 - added chapter 13 (Information for Security Class 1) - added chapter 16.7 (PROFINET System Redundancy S2 with the TPS-1)

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Handling of Unused Pins

   Handle unused pins in accordance with the directions given under Handling of Unused Pins in the manual.

   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment when power is supplied.

   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of Microprocessing unit or Microcontroller unit products in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL II Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141