

Application example for implementation of the PROFIsafe driver on the Evaluation Kit EK-ERTEC 200P-2 V4.7

Getting Started

Introduction	1
GSDML configuration	2
Commissioning the DevKit	3
Preparation/configuration of standard source files	4
Configuration of the PROFIsafe driver	5
State machine	6
Embedding the PROFIsafe driver into the F-device	7
Sequence diagrams	8
Program flow diagram	9
PROFIsafe functions	10
Integration of PROFIsafe driver source files into the application example	11
Mapping the error numbers	12
LED status of the DevKit	13
User LED output	14
References	A

Legal information

Warning notice system

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

DANGER

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

WARNING

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

CAUTION

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

NOTICE

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

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

Qualified Personnel

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

Proper use of Siemens products

Note the following:

WARNING

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

Trademarks

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

Disclaimer of Liability

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

Preface

This document describes integration of the PROFIsafe driver into the software of the EK-ERTEC200P-2 Evaluation Kit. A single-channel hardware architecture and a dual-channel hardware architecture can be simulated in the implementation.

For the dual-channel implementation, the second microcontroller is simulated, as is the second independent timer. The application example serves as an example of the firmware implementation. However, with its existing hardware, it does not constitute a safety-oriented module and has no safety certificate.

The GSDML has been configured for the PROFIsafe application example so that the fail-safe telegrams conform to PROFIsafe Profile V2.6 MU1.

The user data is thus structured as follows:

- 1 byte user data (input and output)
- 1 byte status/control byte
- 4 bytes CRC

In addition, the implementation builds on the code of the Evaluation Kit. The user should have a basic understanding of the Evaluation Kit before adding the PROFIsafe driver to it. The basic functions of the Evaluation Kit are revisited and explained once again in this description. The PROFIsafe driver supports multi-instance operation. This means that multiple (up to 32) F-devices can be installed on one device. Only one instance is configured in this application example.

NOTICE
Safety operation in a real F-application The purpose of the application example is to present a possible implementation of the PROFIsafe driver. The application example is not suitable for safety operation in a real F-application.

Table of contents

1	Introduction.....	7
1.1	Requirements.....	7
2	GSDML configuration.....	8
2.1	Preparation.....	8
2.2	Extensions – Step by step.....	8
2.2.1	DAP (Device Access Point).....	8
2.2.2	Module (slot submodules).....	9
2.2.3	Optional changes.....	14
2.2.4	Integrating the GSDML file in the TIA Portal.....	15
3	Commissioning the DevKit.....	16
3.1	Installing tools.....	16
3.2	Configuration for Eclipse.....	18
3.2.1	Setting up Eclipse for the DevKit.....	18
3.2.2	Replacing PSD-lib in C-files.....	20
3.2.3	Configuration for the debugger.....	22
3.2.4	Configuration of USB terminal connection.....	28
3.2.5	Build of application firmware "App_05.bin".....	28
3.2.6	Loading and starting the PSD F-application.....	30
3.3	Loading the application firmware into flash memory.....	32
3.3.1	Preparation.....	33
3.3.1.1	Application firmware.....	33
3.3.1.2	Configuring TcpFwLoader_EB200P.bat.....	33
3.3.2	Loading procedure.....	33
3.4	Reading out the firmware version data.....	36
3.5	F-address assignment (initialization).....	37
3.5.1	Writing the F-source address and F-destination address.....	37
4	Preparation/configuration of standard source files.....	39
4.1	Preparation of standard source files.....	39
4.2	Setting EXAMPL_DEV_CONFIG_VERSION.....	39
4.3	Changes in the source file usriod_main.c.....	40
5	Configuration of the PROFIsafe driver.....	43
5.1	Header file p_c_config.h.....	43
5.1.1	NOF_INSTANCES.....	43
5.1.2	REDUNDANT.....	43
5.1.3	Length configuration.....	44

5.1.4	Source/destination address; SIL class.....	44
5.1.5	Telegram properties.....	44
5.2	Header file psd_interface.h.....	45
5.2.1	Slot and subslot.....	45
5.2.2	Parameters for telegram properties.....	45
5.2.3	Interface for BusEvents.....	45
5.2.4	Diagnostics defines.....	46
5.2.5	Diagnostics type DiagType.....	47
6	State machine.....	48
6.1	Structure of state machine.....	48
6.2	State diagram.....	49
6.3	Transition table.....	50
7	Embedding the PROFIsafe driver into the F-device.....	52
7.1	Embedding the PROFIsafe driver into the F-device.....	52
8	Sequence diagrams.....	54
8.1	Error-free sequence.....	54
8.1.1	Startup of configuration/parameter assignment.....	54
8.1.2	Transition from STOP to RUN.....	55
8.1.3	Cyclic data exchange.....	56
8.1.4	Stopping of PROFIsafe communication.....	57
8.2	Error scenarios.....	58
8.2.1	Errors during parameter assignment.....	58
8.2.2	Telegram error in state "CYCLIC_DATAEX; DATAEX_OUT_RCV".....	59
9	Program flow diagram.....	60
9.1	PnUsr_DeviceSetup().....	61
9.2	PNIO_cbf_rec_write().....	62
9.3	PNIO_cbf_data_read().....	65
9.4	PNIO_cbf_data_write().....	68
9.5	PNIO_cbf_ar_connect_ind().....	71
9.6	DG_update().....	72
10	PROFIsafe functions.....	74
10.1	psd_InitInstance().....	74
10.2	psd_FParBuild().....	74
10.3	psd_Config().....	75
10.4	psd_Run().....	75
10.5	psd_Stop().....	76
10.6	psd_RecvFOutTele().....	76

10.7	psd_GetFOutData().....	77
10.8	psd_SetFInData().....	77
10.9	psd_SendFInTele().....	78
11	Integration of PROFIsafe driver source files into the application example.....	79
11.1	p_c_fapplication.c.....	79
11.2	p_c_si.c.....	79
11.3	p_c_pseudo.c.....	80
11.4	DiagnosticManager.c.....	82
11.5	Psd_interface.c.....	83
11.6	DevKit PnUsr_Api.c.....	85
11.7	DevKit iodapi_event.c.....	85
11.8	DevKit usriod_main.c.....	86
12	Mapping the error numbers.....	87
13	LED status of the DevKit.....	88
14	User LED output.....	89
A	References.....	91
	Glossary.....	92

Introduction

1.1 Requirements

Basic understanding of the following is required:

- PROFIsafe Starter Kit V3.5.2 (6ES7195-3BF03-0YAO) (not required for commissioning)
- EvalKit ERTEC200P V4.7.0 (6ES7195-3BE00-0YAO)
- PROFIsafe_3192_V26MU1_Aug18.pdf
- profisafe_driver_v2_2_3_programming_manual_en-US.pdf and Evaluation Kit EK-ERTEC200P-2 PN IO V4.7.0
 - Interface_Description_PN_IO_DevKits_V4.7.0.pdf
 - Guideline_EvalKit_ERTEC200P_V4.7.0.pdf
 - Manual_ERTEC200P-2_V1_0.pdf
 - Howto use J-link JTAG Debugger_on_EB200P_V1_1.pdf
 - GSDML_GettingStarted_V1_5.pdf
- GSDML Specification
 - GSDML-Spec_2352_V243_May22.pdf
- PROFINET communication
- C programming language
- J-Link debugger

Required software and tools

- TIA Portal V17
 - Example project: Application_Example_PSD_DevKit_ERTEC200P_4.7
 - GSDML-V2.35-Siemens-ERTEC200pEvalkit-20210601.xml (

...DevKit4.7_PSD_Image\contributions\GSDML\GSDML-V2.35-Siemens-

ERTEC200pEvalkit-20210601.xml)

NOTE

The enclosed GSDML with the V2.35 schema can no longer be used for PROFIsafe certification. It serves here as an illustrative example and has to be converted to the current GSDML schema for certification.

GSDML configuration

2.1 Preparation

To configure the DevKit as an F-device during the engineering, you need an extension in the GSDML file.

The original GSDML file of the DevKit is located in folder "...\\contributions\\GSDML\\(GSDML-V2.35-Siemens-ERTEC200pEvalkit-20210601.xml)".

The extension of the GSDML is an integral component of the application example.

A TIA test project with GSDML PSD_DevKit_Testproduct_V16 that has been adapted for the application example is available in subdirectory contributions\\Simatic_TIA

The following tools and documentations served as the basis for the extension:

- GSDML_GettingStarted_V1_5.pdf
- PROFINET GSD Checker
- GSDML-Spec_2352_V243_May22.pdf

If no changes are made to DAP and submodules, items 2.2.1 to 2.2.4 are omitted.

NOTE

The GSDML-DeviceProfile-V2.35.xsd used serves as an example. An implementation of the PROFIsafe device, including a new PROFINET device, must be made with the currently valid GSDML-DeviceProfile, if a product is to be released with it.

2.2 Extensions – Step by step

2.2.1 DAP (Device Access Point)

Copy DAP3 from the GSDML, rename it to DAP9 and add the fail-safe functions to it (Standard MRP PROFIsafe Profile V2.6.1).

Assign a unique ModuleIdNumber for the F-device in DAP9.

The following figure shows an excerpt from the GSDML:

```
<DeviceAccessPointItem ID="DAP_9" PhysicalSlots="0..1" ModuleIdentNumber="0x00000009"  
  <ModuleInfo CategoryRef="ID_ERTEC200PDEVKit">
```

Figure 2-1 GSDML DAP9

You can configure the F-module in the F-device (DAP9) on slots 1 to 16.

```
<UseableModules>  
  <ModuleItemRef ModuleItemTarget="ID_Mod_71" AllowedInSlots="1" />  
</UseableModules>
```

Figure 2-2 GSDML ModuleItem

For configuration of the F-application, the ModuleIdentNumber is defined in a define in header file "psd_interface.h" (see section 5.2 [\(Page 45\)](#)).

2.2.2 Module (slot submodules)

Submodule "ID_MOD_01" with one byte input and one byte output serves as the basis.

Using the structure of this submodule as a basis, create and configure the fail-safe-specific extensions in the new submodule "ID_Mod_71". The new submodule can be found in the configuration under "1 byte IO PSD 2.6.1".

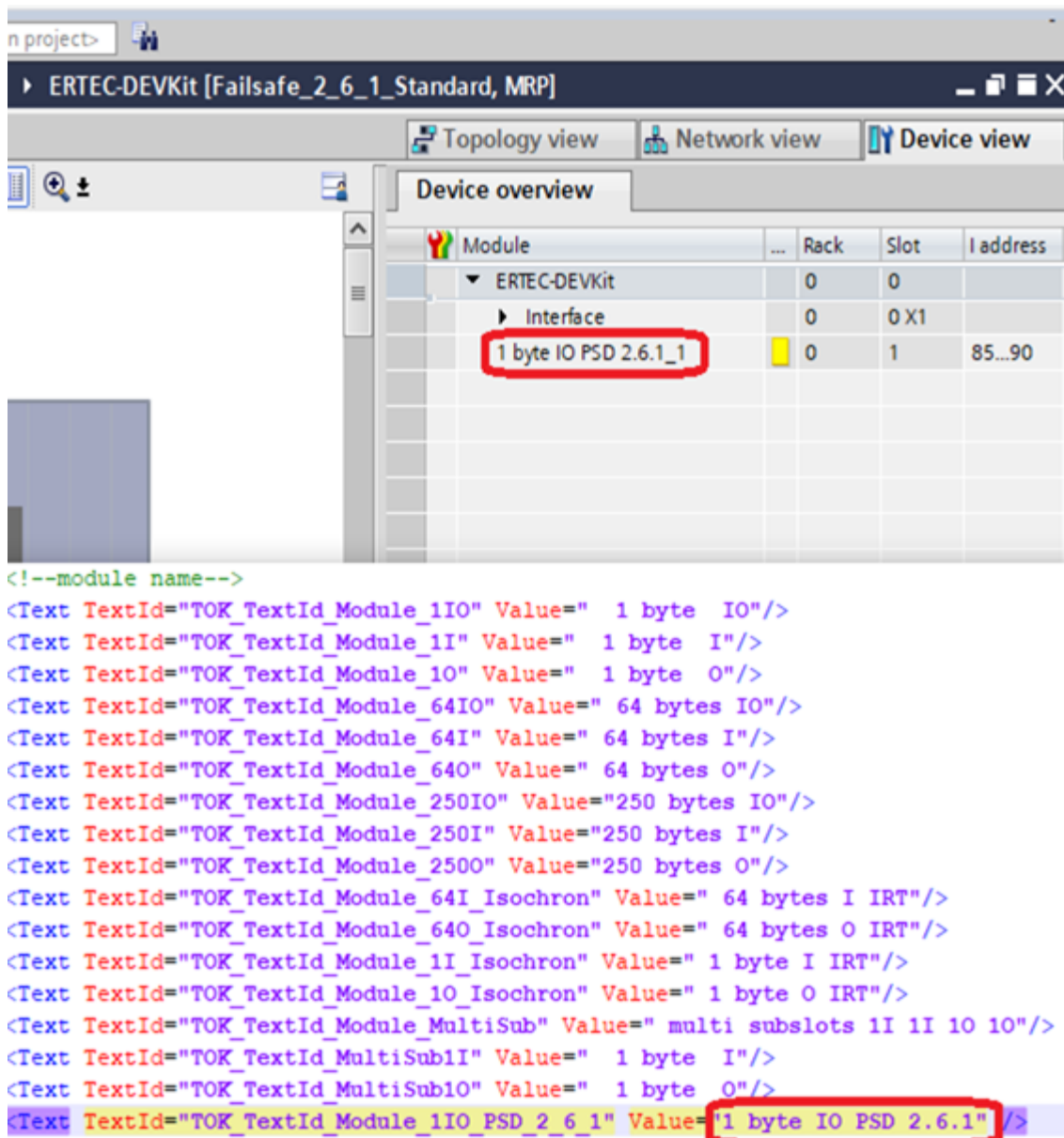


Figure 2-3 GSDML module name

Specify the ID and "ModuleIdentNumber".

Excerpt from the GSDML:

```
544 <ModuleItem ID="ID_Mod_71" ModuleIdentNumber="0x00000071">
```

Figure 2-4 ModuleIdentNumber

Change the "VirtualSubmoduleList" so that you can configure the DevKit as an F-device:

```
<VirtualSubmoduleList>
  <VirtualSubmoduleItem ID="702" SubmoduleIdentNumber="0x0001" API="0" PROFIsafeSupp
    <IOData F_IO_StructureDescVersion="1" F_IO_StructureDescCRC="60025">
      <Input Consistency="All items consistency">
        <DataItem DataType="Unsigned8" UseAsBits="true" TextId="F_ID_INPUT" >
          <BitDataItem TextId="Bit 0" BitOffset="0"/>
          <BitDataItem TextId="Bit 1" BitOffset="1"/>
          <BitDataItem TextId="Bit 2" BitOffset="2"/>
          <BitDataItem TextId="Bit 3" BitOffset="3"/>
          <BitDataItem TextId="Bit 4" BitOffset="4"/>
          <BitDataItem TextId="Bit 5" BitOffset="5"/>
          <BitDataItem TextId="Bit 6" BitOffset="6"/>
          <BitDataItem TextId="Bit 7" BitOffset="7"/>
        </DataItem>
        <DataItem DataType="F_MessageTrailer5Byte" TextId="F_ID_PSD_TRAILER" />
      </Input>
      <Output Consistency="All items consistency">
        <DataItem DataType="Unsigned8" UseAsBits="true" TextId="F_ID_OUTPUT" />
        <DataItem DataType="F_MessageTrailer5Byte" TextId="F_ID_PSD_TRAILER" />
      </Output>
    </IOData>
    <RecordDataList>
      <ParameterRecordDataItem Index="1" Length="4" TransferSequence="0">
        <Name TextId="T_general_parameter"/>
        <Const Data="0x20,0x01,0x56,0x78" ByteOffset="0"/>
      </ParameterRecordDataItem>
      <F_ParameterRecordDataItem F_ParamDescCRC="16197" Index="128">
        <F_Check_iPar DefaultValue="NoCheck" Visible="false" Changeable="fal
        <F_SIL DefaultValue="SIL3" Visible="true" Changeable="false" Allowed
        <F_CRC_Length DefaultValue="4-Byte-CRC" Visible="true" Changeable="f
        <F_CRC_Seed />
        <F_Passivation DefaultValue="Device/Module" AllowedValues="Device/Mo
        <F_Block_ID DefaultValue="1" Visible="true" Changeable="false" />
        <F_Par_Version DefaultValue="1" AllowedValues="1" Visible="true" Cha
        <F_Source_Add DefaultValue="1" AllowedValues="1..65534" />
        <F_Dest_Add DefaultValue="1" AllowedValues="1..65534" />
        <F_WD_Time DefaultValue="150" AllowedValues="10..65534" />
        <F_Par_CRC DefaultValue="57377" />
        <F_iPar_CRC DefaultValue="4180666059" Changeable="true" Visible="true" />
      </F_ParameterRecordDataItem>
    </RecordDataList>
  </VirtualSubmoduleItem>
</VirtualSubmoduleList>
```

Figure 2-5 GSDML module

```

817      <!--F-Common-->
818      <Text TextId="F_ID_INPUT" Value="Input values" />
819      <Text TextId="F_ID_OUTPUT" Value="Output values" />
820      <Text TextId="I_PARAMETER" Value="I-Parameter" />
821      <Text TextId="F_ID_PSD_TRAILER" Value="PROFIsafe" />
822      <Text TextId="General parameters" Value="General Parameters" />
823      <Text TextId="Bit 0" Value="Bit 0" />
824      <Text TextId="Bit 1" Value="Bit 1" />
825      <Text TextId="Bit 2" Value="Bit 2" />
826      <Text TextId="Bit 3" Value="Bit 3" />
827      <Text TextId="Bit 4" Value="Bit 4" />
828      <Text TextId="Bit 5" Value="Bit 5" />
829      <Text TextId="Bit 6" Value="Bit 6" />
830      <Text TextId="Bit 7" Value="Bit 7" />

```

Figure 2-6 Resolving F-values

The most important elements are:

- "VirtualSubmoduleItem ID"
- "PROFIsafeSupported"
- "F-MessageTrailer5Byte"
- All properties that are listed under "F_ParameterRecordDataItem"

The bits under "Input" stand for the user data in the PROFIsafe telegram. The property "F-MessageTrailer5Byte" adds a 5-byte long PROFIsafe trailer to the telegram. This consists of 1 byte status/control byte and 4 bytes CRC.

All properties under "F_ParameterRecordDataItem" stand for the F-parameters that are sent to the DevKit when a connection is established.

As the final configuration step, provide diagnostics for the F-device. The diagnostics texts have been standardized and are described in document "PROFIsafe-Profile_3192_V261_Aug14.pdf", page 39.

Excerpt from the GSDML:

```
</ChannelDiagItem>
<!--PROFIsafe Driver Diag -->
<ChannelDiagItem ErrorType="64">
  <Name TextId="Mismatch of safety destination address (F_Dest_Add)" />
  <Help TextId="Help_Mismatch of safety destination address (F_Dest_Add)" />
</ChannelDiagItem>
<ChannelDiagItem ErrorType="65">
  <Name TextId="Safety destination address not valid (F_Dest_Add)" />
  <Help TextId="Help_Safety destination address not valid (F_Dest_Add)" />
</ChannelDiagItem>
<ChannelDiagItem ErrorType="66">
  <Name TextId="Safety source address not valid (F_Source_Add)" />
  <Help TextId="Help_Safety source address not valid (F_Source_Add)" />
</ChannelDiagItem>
<ChannelDiagItem ErrorType="67">
  <Name TextId="Safety watchdog time value is 0 ms (F_WD_Time or F_WD_Time2)" />
  <Help TextId="Help_Safety watchdog time value is 0 ms (F_WD_Time or F_WD_Time2)" />
</ChannelDiagItem>
<ChannelDiagItem ErrorType="68">
  <Name TextId="Parameter F_SIL exceeds SIL from specific device application" />
  <Help TextId="Help_Parameter F_SIL exceeds SIL from specific device application" />
</ChannelDiagItem>
<ChannelDiagItem ErrorType="69">
  <Name TextId="Parameter F_CRC_Length does not match the generated value" />
  <Help TextId="Help_Parameter F_CRC_Length does not match the generated value" />
</ChannelDiagItem>
<ChannelDiagItem ErrorType="70">
  <Name TextId="Version of F-Parameter set or F_Block_ID incorrect" />
  <Help TextId="Help_Version of F-Parameter set or F_Block_ID incorrect" />
</ChannelDiagItem>
<ChannelDiagItem ErrorType="71">
  <Name TextId="CRC1-Fault" />
  <Help TextId="Help_CRC1-Fault" />
</ChannelDiagItem>
<ChannelDiagItem ErrorType="72">
  <Name TextId="Device specific diagnostic information, see manual" />
  <Help TextId="Help_Device specific diagnostic information, see manual" />
</ChannelDiagItem>
<ChannelDiagItem ErrorType="73">
  <Name TextId="Save iParameter watchdog time exceeded" />
  <Help TextId="Help_Save iParameter watchdog time exceeded" />
</ChannelDiagItem>
<ChannelDiagItem ErrorType="74">
  <Name TextId="Restore iParameter watchdog time exceeded" />
  <Help TextId="Help_Restore iParameter watchdog time exceeded" />
</ChannelDiagItem>
<ChannelDiagItem ErrorType="75">
  <Name TextId="Inkonsistente iParameter (iParCRC Fehler)" />
  <Help TextId="Help_Inkonsistente iParameter (iParCRC Fehler)" />
</ChannelDiagItem>
```

Figure 2-7 GSDML diagnostics texts

```

<!-- PROFIsafe -->
<Text TextId="Mismatch of safety destination address (F_Dest_Add)" Value="Mismatch of safety destination address (F_Dest_Add)" />
<Text TextId="Help_Mismatch of safety destination address (F_Dest_Add)" Value="Help_Mismatch of safety destination address (F_Dest_Add)" />
<Text TextId="Safety destination address not valid (F_Dest_Add)" Value="Safety destination address not valid (F_Dest_Add)" />
<Text TextId="Help_Safety destination address not valid (F_Dest_Add)" Value="Help_Safety destination address not valid (F_Dest_Add)" />
<Text TextId="Safety source address not valid (F_Source_Add)" Value="Safety source address not valid (F_Source_Add)" />
<Text TextId="Help_Safety source address not valid (F_Source_Add)" Value="Help_Safety source address not valid (F_Source_Add)" />
<Text TextId="Safety watchdog time value is 0 ms (F_WD_Time or F_WD_Time2)" Value="Safety watchdog time value is 0 ms (F_WD_Time or F_WD_Time2)" />
<Text TextId="Help_Safety watchdog time value is 0 ms (F_WD_Time or F_WD_Time2)" Value="InfoSafety watchdog time value is 0 ms (F_WD_Time or F_WD_Time2)" />
<Text TextId="Parameter F_SIL exceeds SIL from specific device application" Value="Parameter F_SIL exceeds SIL from specific device application" />
<Text TextId="Help_Parameter F_SIL exceeds SIL from specific device application" Value="Help_Parameter F_SIL exceeds SIL from specific device application" />

```

Figure 2-8 Resolving diagnostics texts of the GSDML

Open the GSDML file with the program "PROFINET GSD Checker". Select the "XML" tab and run "Check". Normally, error messages regarding incorrect checksums are now displayed.

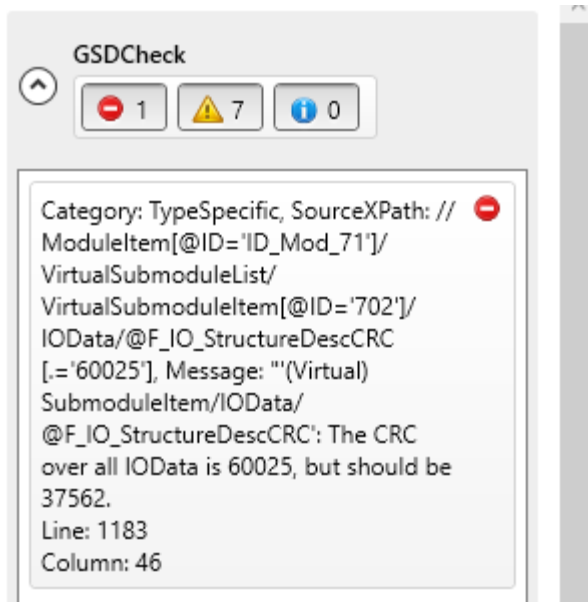


Figure 2-9 Error message of the GSD checker

Change the checksums so that the GSDML is executable.

2.2.3 Optional changes

Optionally, you can also change values such as "Text ID", "GraphicItemTarget", etc., in the GSDML file. However, these values do not affect the technical behavior of the submodule. Adapt the fail-safe-specific and general text IDs in the course of the implementation.

Excerpt from the GSDML file:

```
<Text TextId="TOK_TextId_Module_1IO_PSD_2_6_1" Value="1 byte IO PSD 2.6.1" />
```

Figure 2-10 Excerpt of optional changes in GSDML

You can install the GSDML file in other PNO-compliant engineering systems. In this example, the GSDML file was installed in TIA Portal V17.

2.2.4 Integrating the GSDML file in the TIA Portal

Integrate the GSDML file in the TIA Portal as follows:

1. In the TIA Portal, select menu "Options > Manage general station description files (GSD)".
2. In the "Manage general station description files" dialog, select the path to the GSDML file.
3. Click the "Install" button.

The DevKit can then be found under the following path:

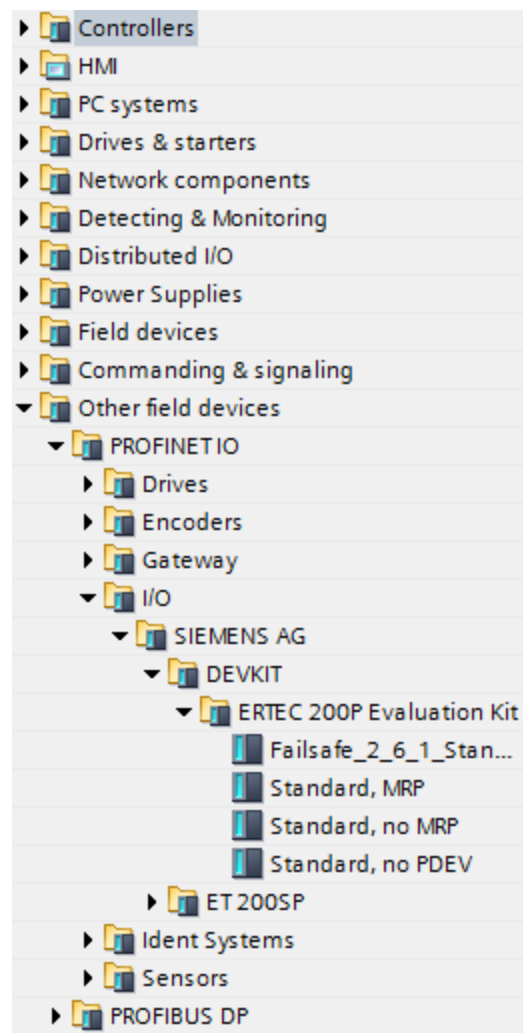


Figure 2-11 Path in the hardware catalog

Commissioning the DevKit

3.1 Installing tools

Requirement

Carry out the following steps for the installation:

1. Copy the content of the Evaluation Kit CD to a local drive.
2. Call the file "install.bat" there.
3. Replace the curl version in the file "get_dependencies.ps1" (7.70) with the latest version (7.84.0_3 or newer) as version 7.70 has been removed from the website (<https://curl.haxx.se/windows/>).

Installing tools

Successful installation requires that the tools of the DevKit be installed using the file "...\\DevKit4.7_PSD_Image\\setup\\install.bat".

Carry out the following steps for installation of the toolchain:

1. Unpack the image.
2. Open the image with administrator rights cmd in the setup directory.
3. Then call the install.bat file.

You can find more information on the installation in the guideline EvalKit_ERTEC200P_V4.7.0.pdf, sections 4.1.2 ... 4.1.4.

The default hardware settings are retained (see Manual...\\DevKit4.7_PSD_Image\\doc\\HW\\Tech_Doc\\Manual_EB200P-2_V1_0_3.pdf, section 7 - Settings on the EB200P-2).

If the installation was successfully completed, you receive the following information:

```

c:\. Eingabeaufforderung

Downloading arm_gcc
~~~~~
% Total      % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100  247  100  247    0     0   473      0  --:--:--  --:--:--  --:--:--   478
100 130M  100 130M    0     0 7604k      0  0:00:17  0:00:17  --:--:-- 7899k
~~~~~
Unzipping arm_gcc
~~~~~
Downloading MinGW
~~~~~
% Total      % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
  0     0    0     0    0     0    0      0  --:--:--  0:00:01  --:--:--    0
100 351k  100 351k    0     0  136k      0  0:00:02  0:00:02  --:--:-- 298k
~~~~~
Unzipping MinGW
~~~~~
-----
Install MinGW packages
-----
Adding CygPath File
-----
Copy ARM_GCC into msys
-----
Copy ECOS into msys
-----
CleanUp Temp
-----
Completed
-----

D:\DevKit4.7_PSD_Image\setup>

```

Figure 3-1 Installation completed

3.2 Configuration for Eclipse

3.2.1 Setting up Eclipse for the DevKit

Carry out the following steps to set up the DevKit:

1. In the subdirectory, open the file installed_tools\Eclipse\eclipse.exe as administrator. The following error may occur when opening Eclipse:
Failed to load the JNI shared library "C:\Program Files(x86)\Zulu\zulu-8-jre\bin\client\jvm.dll".
The Eclipse version installed is a 64-bit version. Therefore a 64-bit Java version must also be used. Please point to the corresponding Java version in the Path environment variable of your system.
2. Create a new directory as workspace.
3. Select "Import existing projects".
4. As root directory subdirectory, select pn_iodevkits\src\projects.

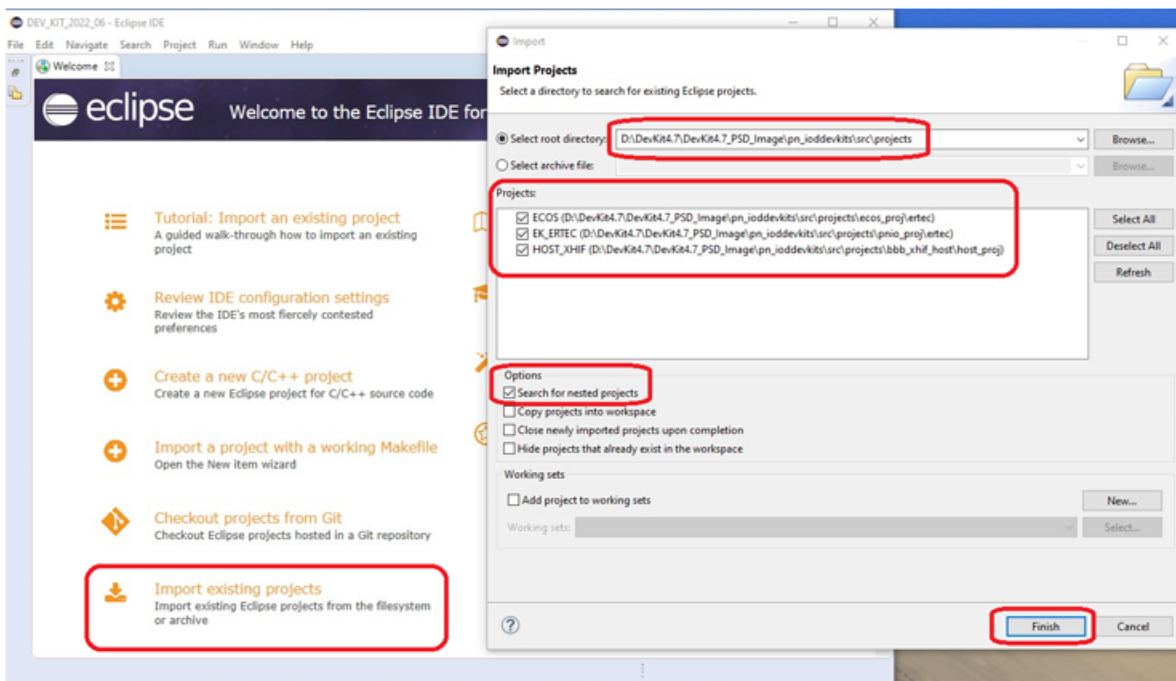


Figure 3-2 Importing Eclipse projects

NOTE

If the subdirectory contains the file pn_iodevkits\src\projects\ecos_target\eb200p, you must delete the directory eb200p_ecos_build completely.

Open the ECOS project in Eclipse and run Build targets \ 1*32bit NOR-FLASH 32bit.

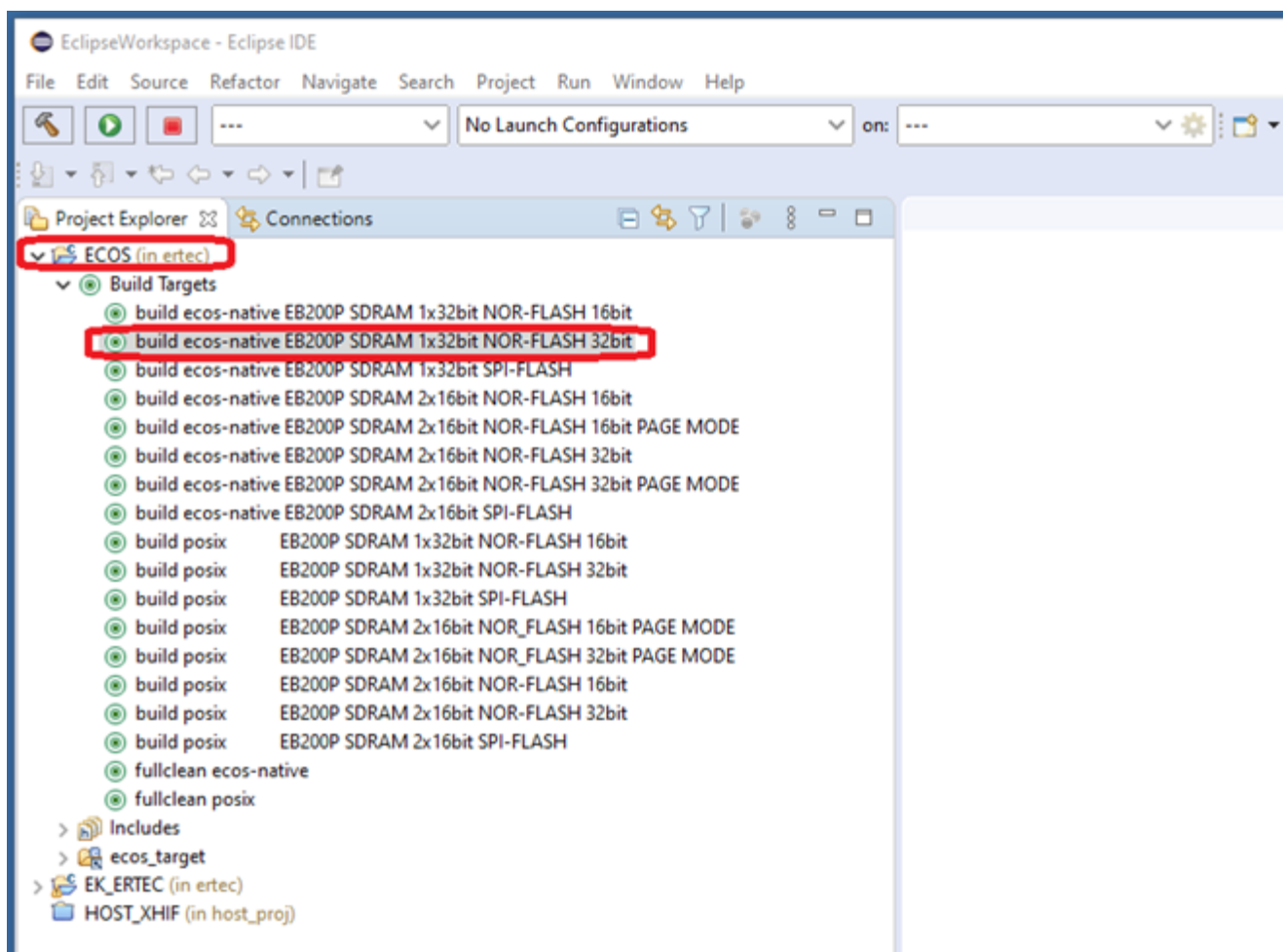


Figure 3-3 Compiling ECOS

Double-click the selected build.

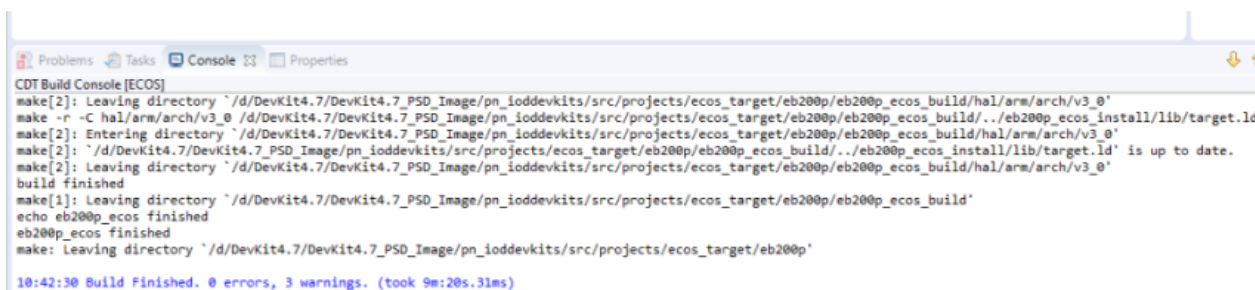


Figure 3-4 Compile_successful

3.2.2 Replacing PSD-lib in C-files

The sources for the F-application example are located in subdirectory `pn_iodevkits\src\application\App5_FAILSAFE_PSD`.

The application example contains the PROFIsafe driver V3.5.2 only as a library compiled for single-channel operation. If you would like to compile or debug the PROFIsafe sources for other operating modes, you need the Starter Kit CD of PROFIsafe driver V3.5.2 or higher (MLFB 1P-6ES7159-3BF030YAO).

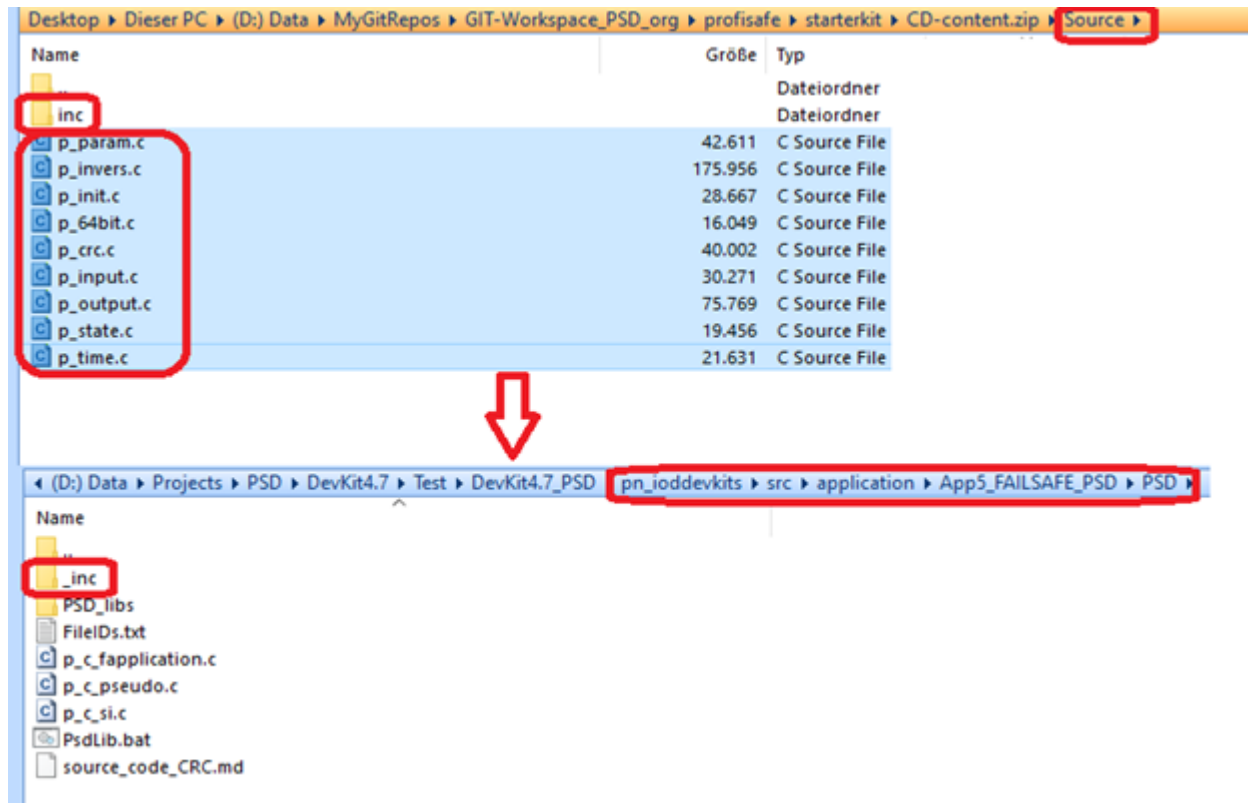


Figure 3-5 Integrating source files

Integrating the sources

1. Copy the sources from the Starter Kit CD subdirectory `Source` to subdirectory `pn_iodevkits\src\application\App5_FAILSAFE_PSD\PSD`.
2. Copy the header files from the Starter Kit CD subdirectory `Source\inc` to subdirectory `pn_iodevkits\src\application\App5_FAILSAFE_PSD\PSD\inc`, **with the exception of files `p_c_config.h` and `p_c_si.h`**.
3. Refresh the Eclipse project so that the source files are visible in the Project Explorer.

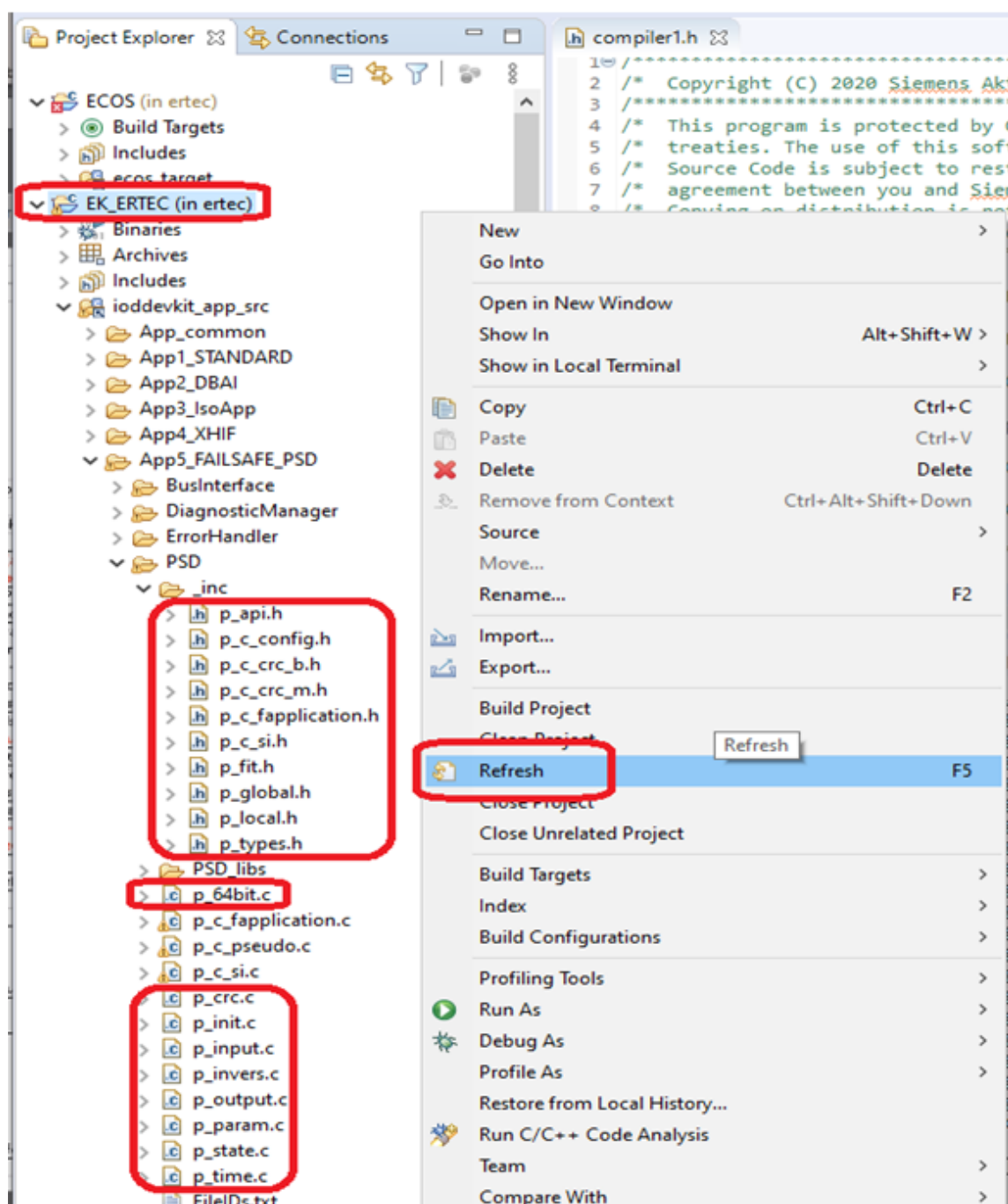


Figure 3-6 Refresh Eclipse project

Removing PSD library files from Eclipse Linker settings

- In the shortcut menu, select project EK_ERTEC\Properties\C/C++Build\Settings\GCC C Linker\Libraries
- Remove the libraries.

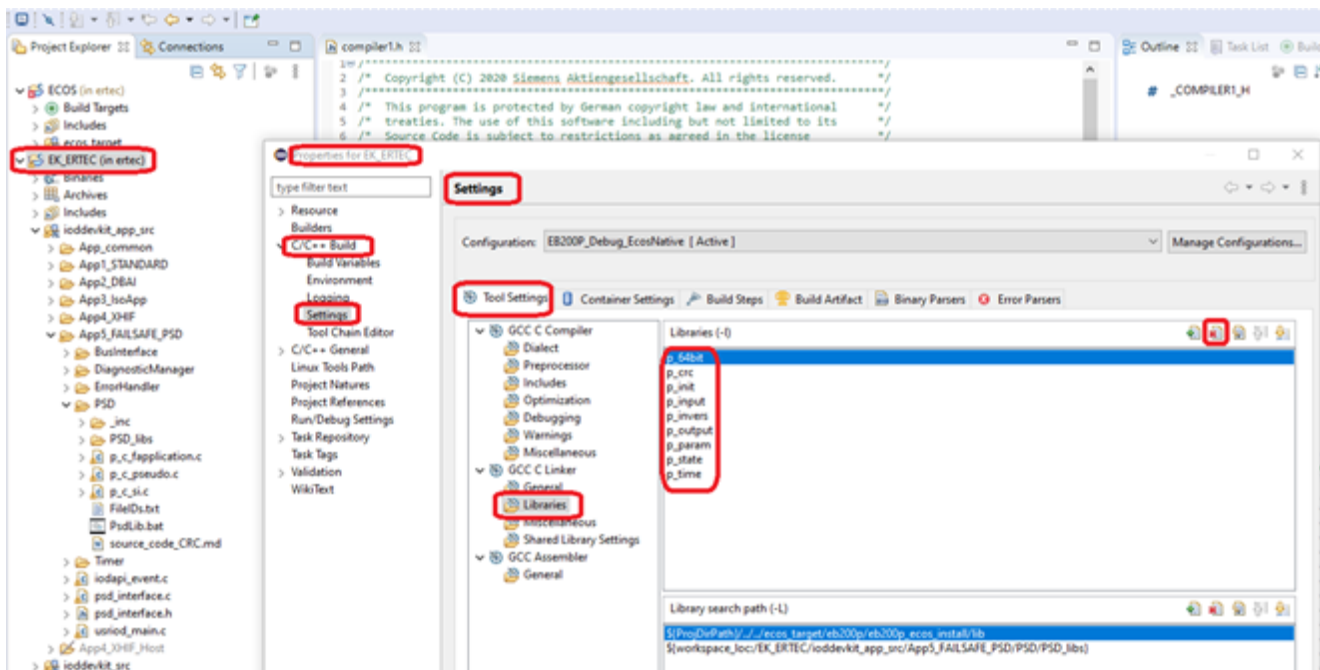


Figure 3-7 Removing PSD libraries from Linker settings

3.2.3 Configuration for the debugger

Debugger

1. Open the configuration in Eclipse using shortcut menu Run\Debug Configurations.
2. Select the file GDB Hardware Debugging\EK_ERTEC_EB200P_Size_EcosNative. Make the settings as described in the following screenshots.
3. Download the driver for TUSB3410 here:
<https://new.siemens.com/global/en/products/buildings/support/fire-safety-driver.html>.
4. Debug configurations for SEGGER J-Link.

NOTE

You can find information on Segger J-Link at: <https://www.segger.com/downloads/jlink>
There you will also find the file "JLinkGDBServer.exe".

Under "C/C++ Application:", press "Search Project..." and select "PNIO4ECOS" in the window.

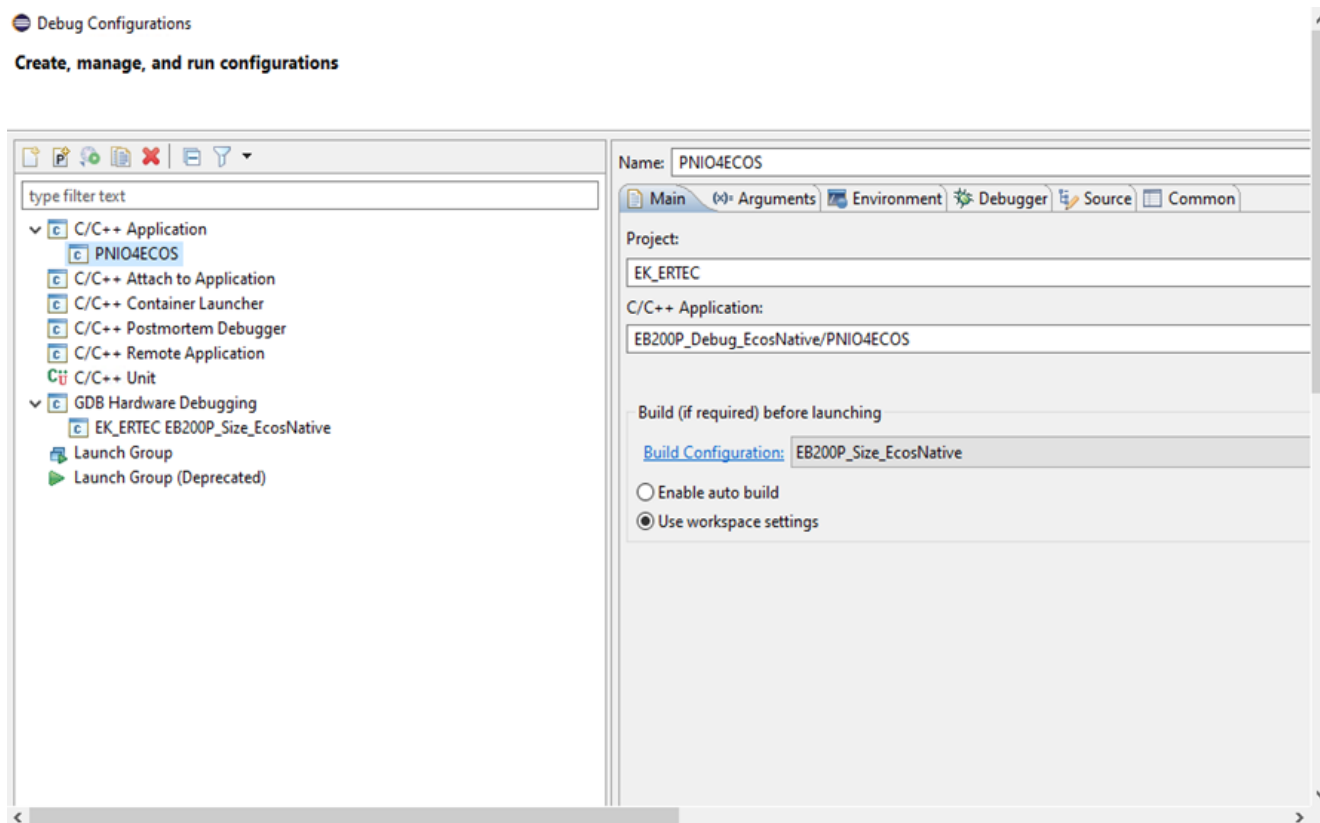


Figure 3-8 Application

Commissioning the DevKit

3.2 Configuration for Eclipse

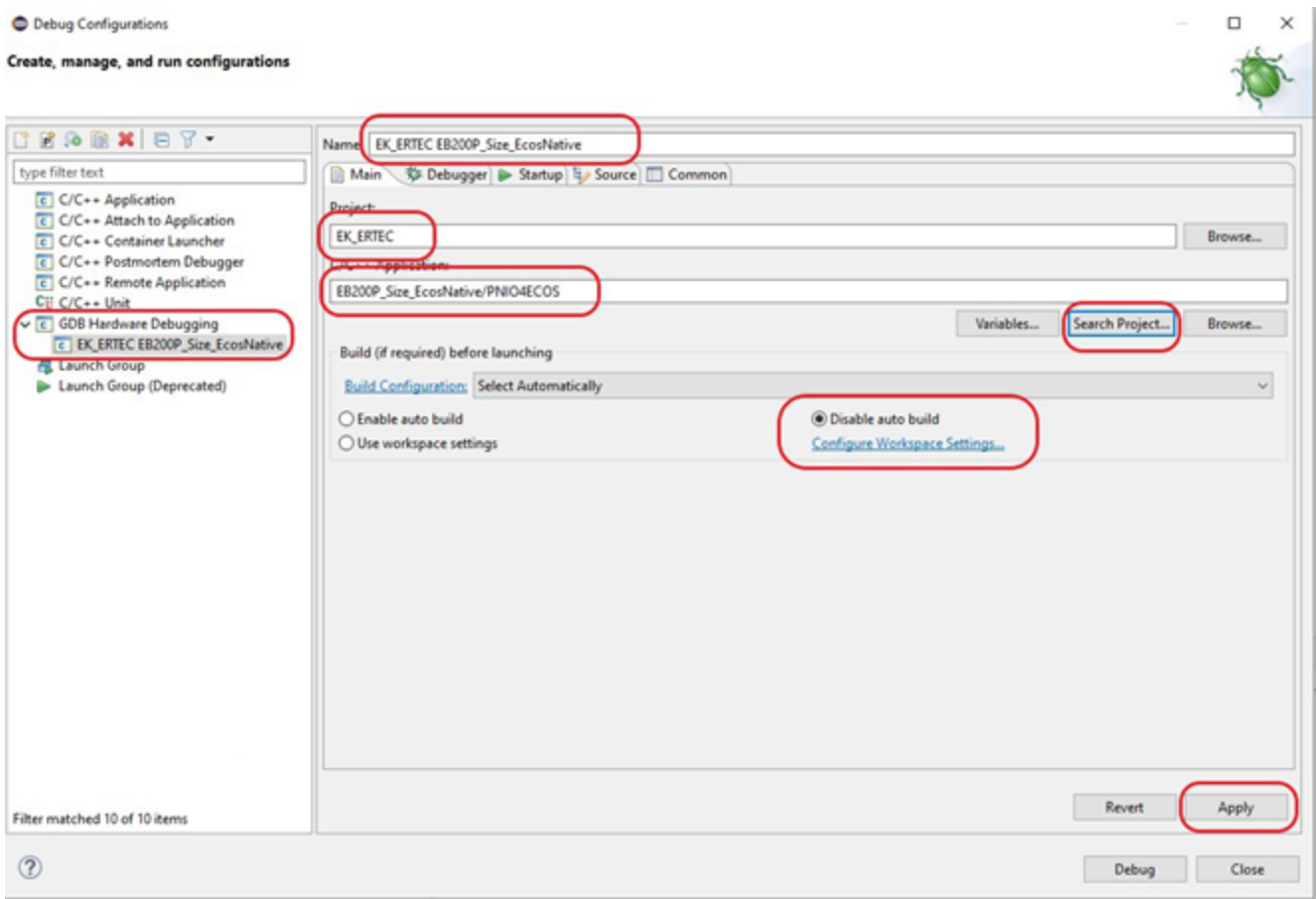


Figure 3-9 Debugger Configuration 1

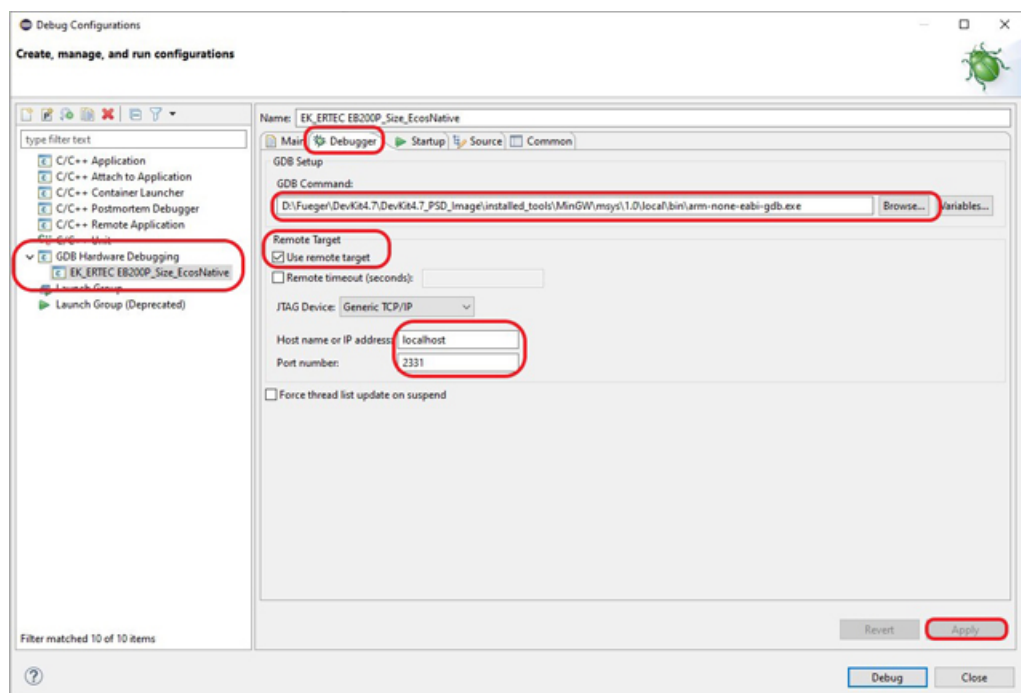


Figure 3-10 Debugger Configuration 2

The "GDB Command:" is located – dependent on your file structure – in the following folder:
`${project_loc}\..\..\..\..\installed_tools\MinGW\msys\1.0\local\bin\arm-none-eabi-gdb.exe`

Startup

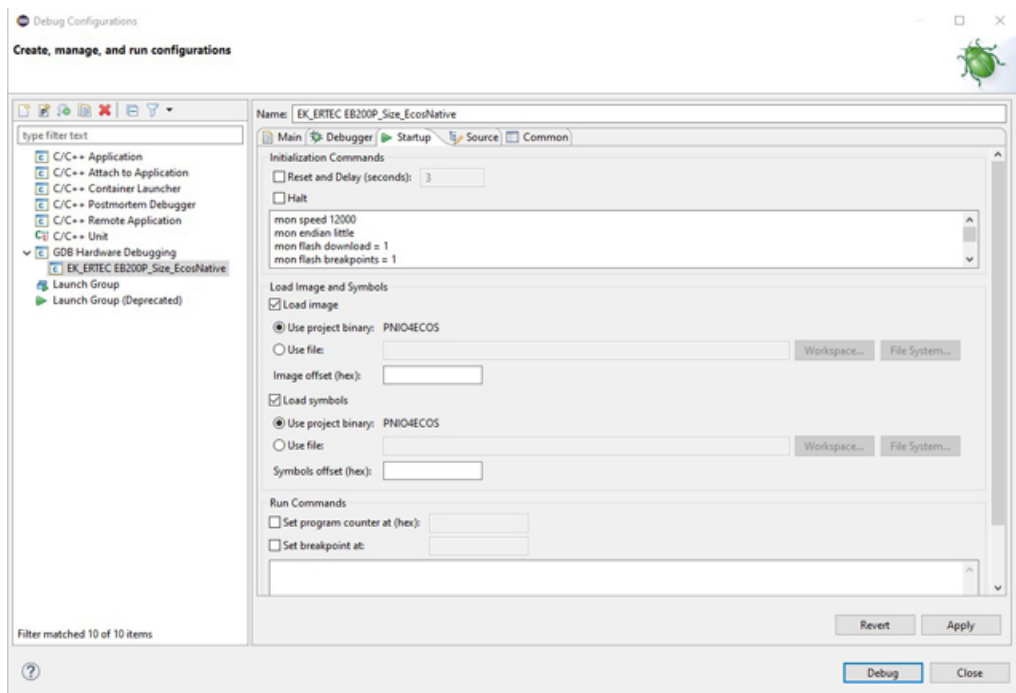


Figure 3-11 Debug Configurations_Startup

If you have installed the application example using the supplied "batch" file, the GDB command must be configured relative to the project path.

Initialization commands

```
mon speed 12000
mon endian little
mon flash download = 1
mon flash breakpoints = 1
mon reset 1
mon mww 0x4000f078 0x00ffffff
mon mww 0x10d00004 0x40000080
mon mww 0x10d0000c 0x000003d0
mon mww 0x10d00010 0x3ffffff2
mon mww 0x10d00014 0x3ffffff2
mon mww 0x10d00018 0x3ffffff2
mon mww 0x10d0001c 0x3ffffff2
mon mww 0x10d00020 0x01974600
mon mww 0x10d00028 0x0
mon mww 0x10d0002c 0x0
mon mww 0x10d00030 0x42
mon mww 0x10d00008 0x00002521
```

Source

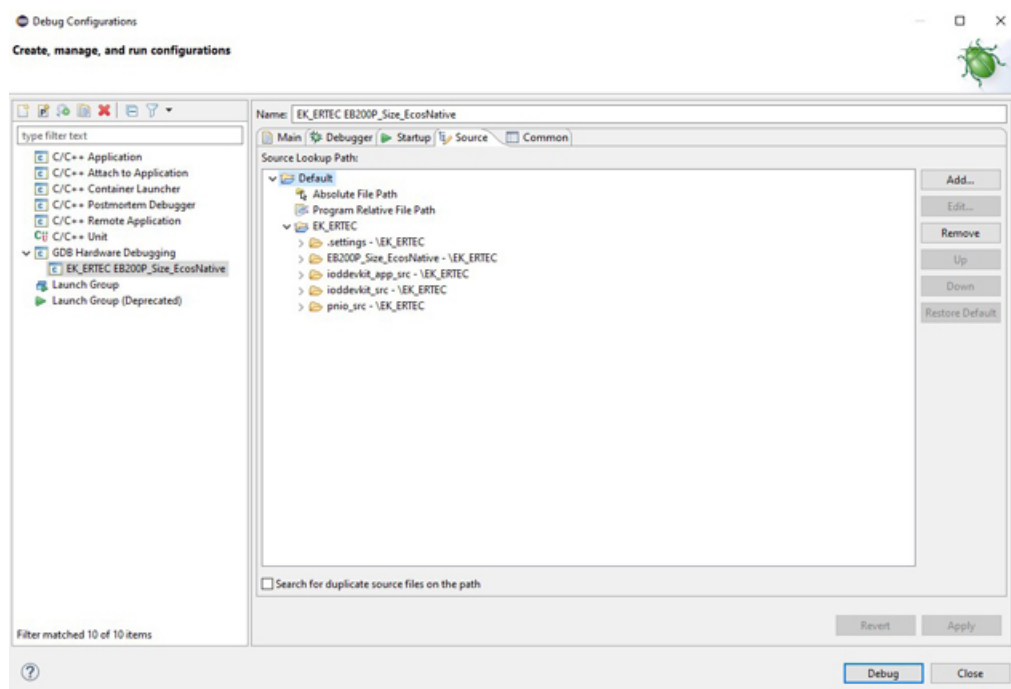


Figure 3-12 Debug Configurations_Source

To load the firmware into RAM, open the "Debug" tab and select the configuration "EB200P_Size_EcosNative".

3.2.4 Configuration of USB terminal connection

In order to change the configuration of the USB terminal connection, follow the procedure described in document "Guideline_EvalKit_ERTEC200P_V4.7.0.pdf" (sections 4.1.3. and 4.2.2). In addition, you must install a terminal program (e.g. Tera Term) on the development computer and configure it as follows: D:\Tools\TeraTerm\teraterm-4.106

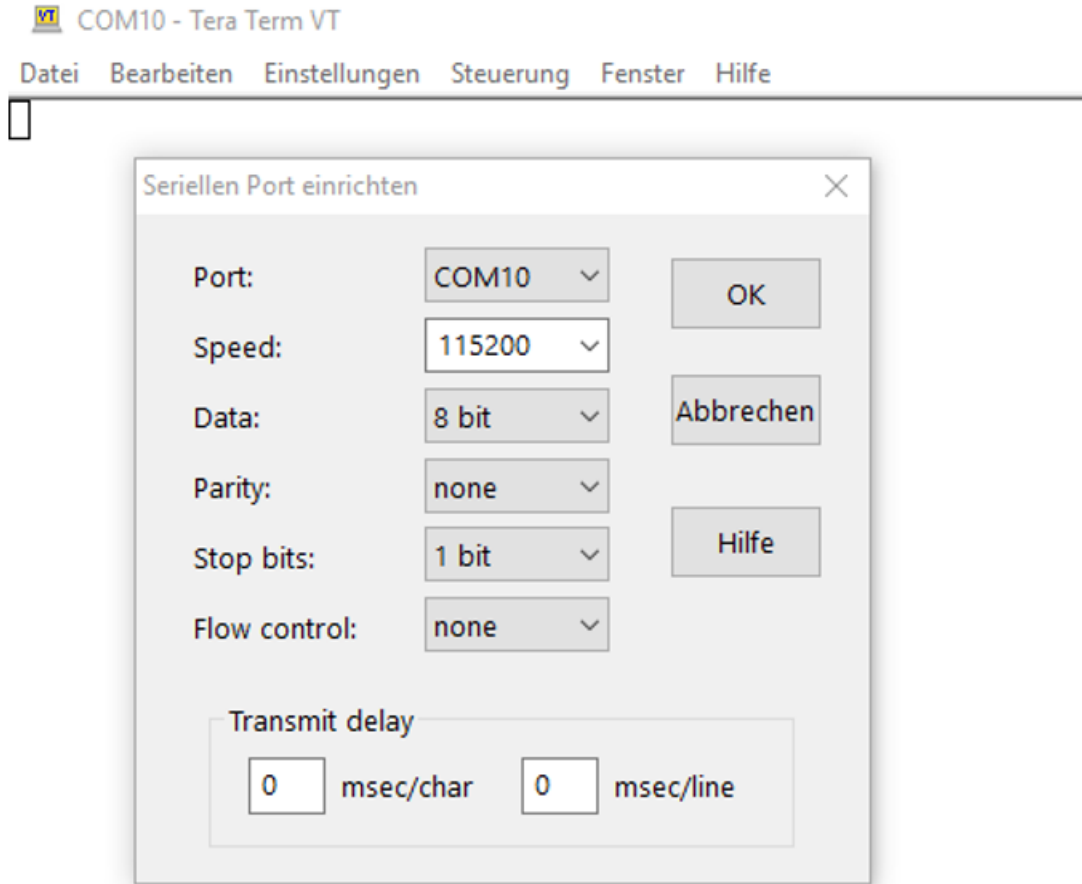


Figure 3-13 Configuration of terminal program

3.2.5 Build of application firmware "App_05.bin"

An executable program must exist in order to create a build of the application firmware. Right-click "EK_ERTEC".

Then compile the EK_ERTEC project.

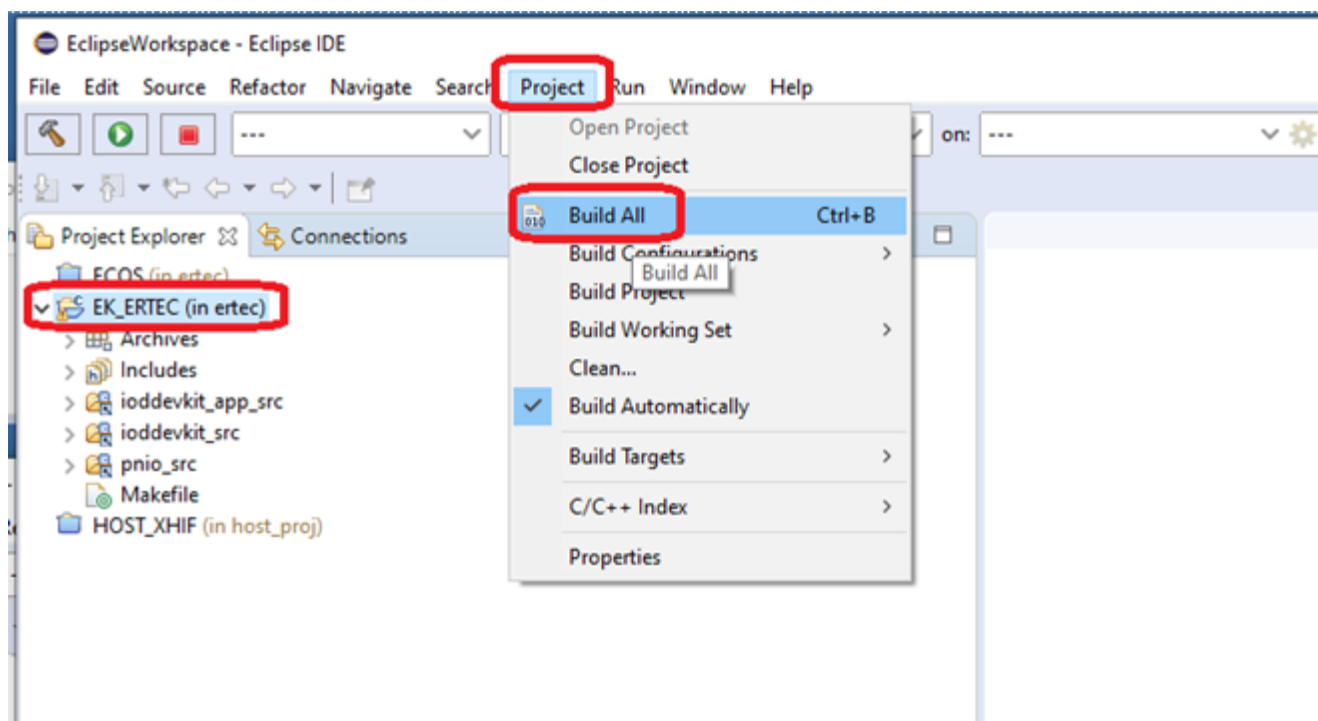


Figure 3-14 Creating a build

Select "Build Project".

The build process is output in the console window of Eclipse.

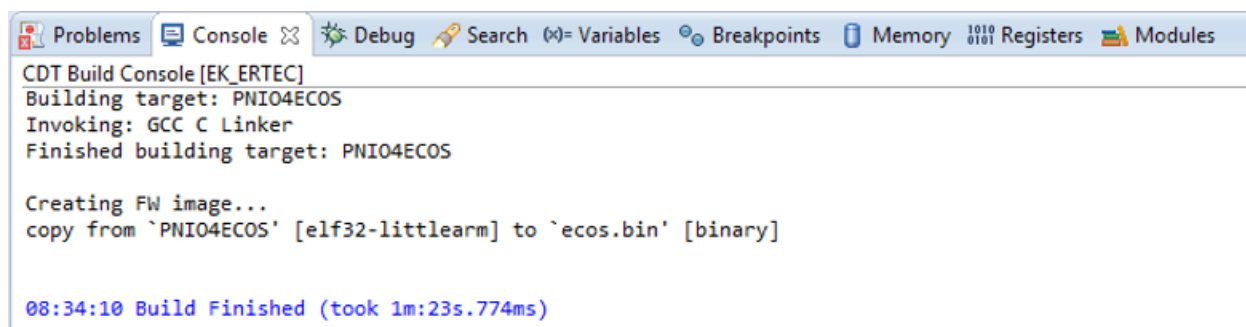


Figure 3-15 Console output following a successful build

If the build was completed successfully, proceed as follows:

- Copy the binary file ecos.bin from subdirectory pn_iodevkit\src\projects\pnio_proj\ertec\EB200P_Size_EcosNative to subdirectory Tools\TcpFwLoader
- Rename the copied binary ecos.bin to ecos200p_app_05.bin

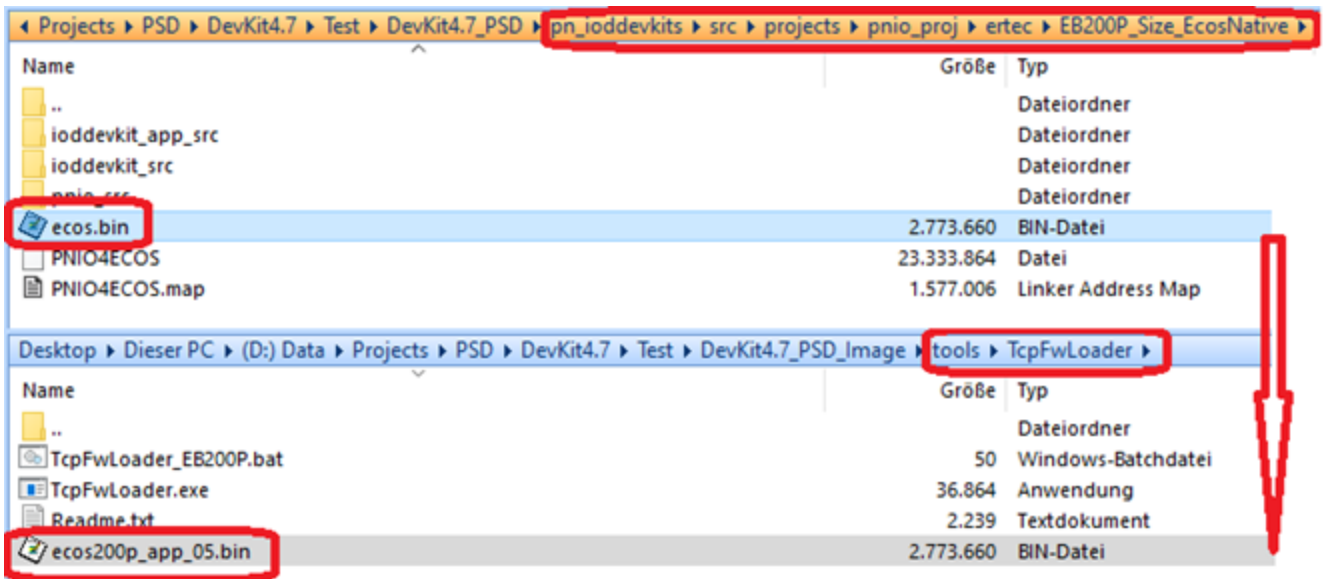


Figure 3-16 Copying the binary

3.2.6 Loading and starting the PSD F-application

Before you begin loading and starting the PSD F-application, the IP address and the PROFINET device name must be assigned by the TIA Portal. Otherwise, no connection to the ERTEC will be detected.

If a PSD F-application has not yet been loaded, you must load this into the RAM of the ERTEC200P DevKit once via the Segger J-Link. To do this, create the firmware as described in section "Build of application firmware" and start the firmware with the "Launch" button.

For the debugging you must configure the SEGGER J-Link (sections 8.1 to 8.5 of file "Guideline_EvalKit_ERTEC200P_V4.7.0.pdf").

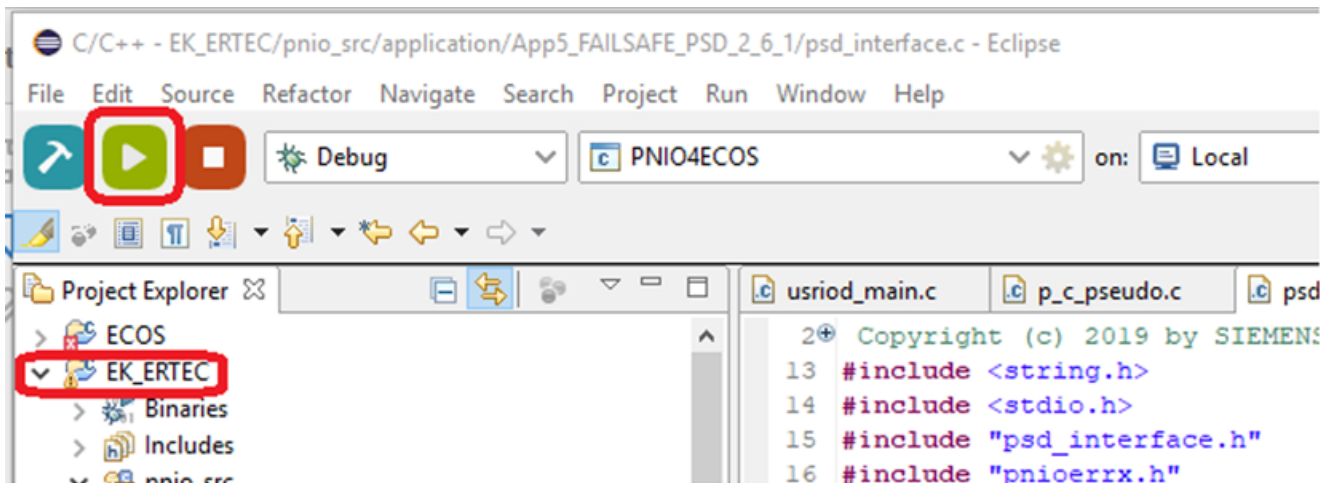


Figure 3-17 Loading the PSD F-application into RAM and starting it

Starting the debug

1. Step:
 - Start the SEGGER J-Link GDB Server.
2. Step:
 - In the Project Explorer, select the file "EB200P_Debug_EcosNative".
 - Press "Run". The "Debug Configurations" window opens.
 - Select the "Startup" tab.
 - Then press the "Debug" button.

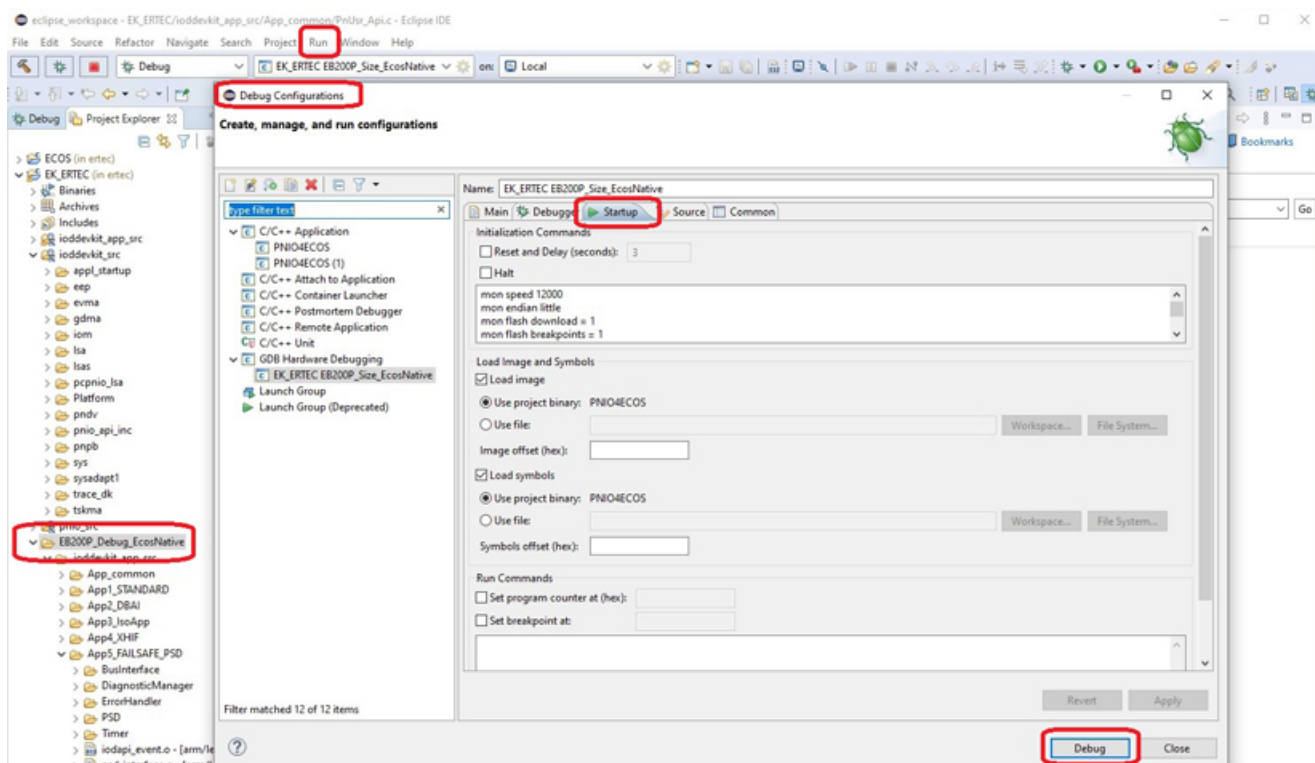


Figure 3-18 Debugger START

3.3 Loading the application firmware into flash memory

In parallel with this, start the console program (Tera Term). Establish a connection to the PC via a USB interface. After correct Power On of the PSD F-application, the terminal console should look like this:

```

COM10 - Tera Term VT
Datei Bearbeiten Einstellungen Steuerung Fenster Hilfe
IP address = 192.168.000.051, Subnet mask = 255.255.255.000, Default router = 192.168.000.051
##CONNECT_IND AR=1 AR type=1 sendclock=32, reduction_ratio_in=2, reduction_ratio_out=2, sessionKey=1025, hostIP=192.168.000.011
##OWNERSHIP_IND AR = 1 number of submodules = 5
  Api=0 Slot=0 Sub=1 ModID=9 SubID=1 OwnSessKey=1025 isWrong=0
  Api=0 Slot=0 Sub=32768 ModID=9 SubID=2 OwnSessKey=1025 isWrong=0
  Api=0 Slot=0 Sub=32769 ModID=9 SubID=3 OwnSessKey=1025 isWrong=0
  Api=0 Slot=0 Sub=32770 ModID=9 SubID=3 OwnSessKey=1025 isWrong=0
  Api=0 Slot=1 Sub=1 ModID=113 SubID=1 OwnSessKey=1025 isWrong=0
##WRITE startup-Record from GSD file , Api=0 Slot=1 Subslot=1 Len=4, Sequ_nr=6
##WRITE_REC RQ, Api=0 Slot=1 Subslot=1 Index=0x1, Len=4, Sequ_nr=6
##REC_DATA = 0x20 0x01 0x56 0x78

**** F-Parameter received ****
##PARAM END for Api=0, Slot=0, Sub=1, ArNum=1, Session=1025
##PARAM END for Api=0, Slot=0, Sub=32768, ArNum=1, Session=1025
##PARAM END for Api=0, Slot=0, Sub=32769, ArNum=1, Session=1025
##PARAM END for Api=0, Slot=0, Sub=32770, ArNum=1, Session=1025
##PARAM END for Api=0, Slot=1, Sub=1, ArNum=1, Session=1025
new IO controller output provider status (OPS) = 0x60 in slot 1, subslot 1
new IO controller input consumer status (ICS) = 0x60 in slot 1, subslot 1
##READY FOR INPUT UPDATE DURING AR_STARTUP ARnum=1
##AR IN-Data event indication received, ArNum = 1h, Session = 0h
new IO controller output provider status (OPS) = 0x80 in slot 1, subslot 1
new IO controller input consumer status (ICS) = 0x80 in slot 1, subslot 1
**** param OK ****
**** out passivated ****
**** communication err ****
Asynchronous request ArNum=1 Alarmtype=2 Api=0 Slot=1 Subslot=1 Status = 1
User tag 1
**** out passivated ****
Asynchronous request ArNum=1 Alarmtype=2 Api=0 Slot=1 Subslot=1 Status = 1
User tag 0
**** F output OK ****

```

Figure 3-19 Terminal console after successful Power On of the PSD F-application

If you have not yet transferred any F-parameters, the section including " **** F-Parameter received **** " is omitted, and the Sync LED flashes at a frequency of 2 Hz. No F-parameter assignment is required to load the PSD F-applications into flash memory. The console message " **** F output OK **** " indicates that PSD F-application is exchanging process data.

3.3 Loading the application firmware into flash memory

To load a new firmware version onto the board, you must carry out the following steps:

1. Copy the binary file (ecos200p_app_05.bin) and rename it.
2. Adapt the TcpFwLoader batch file.
3. Start Tera Term and TcpFwLoader.
4. Load the firmware.
5. Restart the DevKit board.

3.3.1 Preparation

3.3.1.1 Application firmware

An executable program with build must exist.

If you have performed the steps in section 3.1.4 Build of application firmware "App_05.bin", you will find the build in directory
pn_iodevkits\src\projects\pnio_proj\ertec\EB200P_Size_EcosNative.

1. Copy the binary ecos.bin from subdirectory
pn_iodevkits\src\projects\pnio_proj\ertec\EB200P_Size_EcosNative to subdirectory
Tools\TcpFwLoader.
2. Rename the binary ecos.bin to ecos200p_app_05.bin.

3.3.1.2 Configuring TcpFwLoader_EB200P.bat

You must adapt the file "TcpFwLoader_EB200P.bat" in folder "Tools\TcpFwLoader".

Transfer parameter 1 for the tcpFwLoader function contains the designation of the binary file of the firmware. In this case: "ecos200p_app_05.bin"

Transfer parameter 2 contains the IP address of the F-device (DevKit board). In this implementation, the address is 192.168.0.51. The address was assigned during engineering.

You must permanently assign transfer parameter 3 with 999.

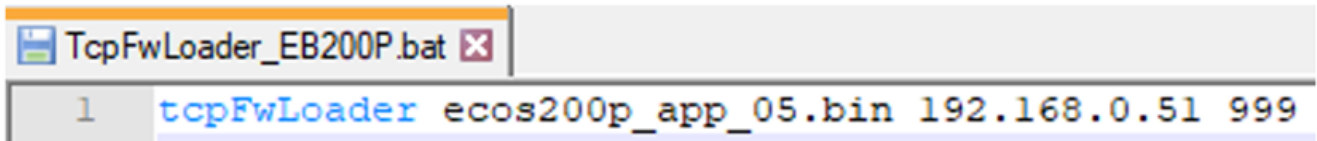


Figure 3-20 Setting the file "TcpFwLoader_EB200P.bat"

3.3.2 Loading procedure

This section shows the procedure for loading the firmware graphically.

3.3 Loading the application firmware into flash memory

Since two separate interfaces (TCP/IP and USB) are used for the firmware update process, you can also load the firmware update via a PC. The following figure shows two PCs.

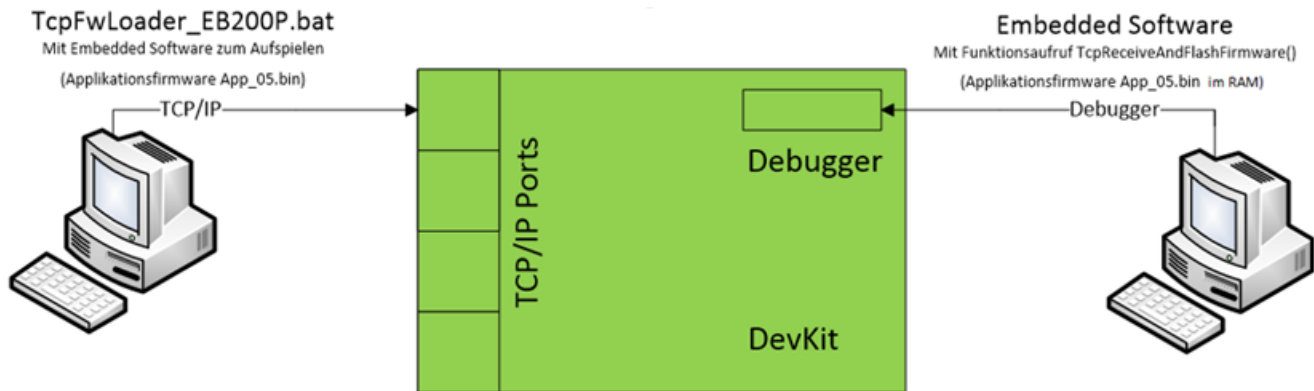


Figure 3-21 Loading the application firmware into flash memory

Start the application firmware in the debugger. If the firmware has already been loaded in the DevKit board, this step is omitted.

Open the terminal console window (Tera Term) and enter an "f".

```
COM3 - Tera Term VT
Datei Bearbeiten Einstellungen Steuerung Fenster Hilfe
new IO controller output provider status (OPS) = 0x60 in slot 1, subslot 1
new IO controller input consumer status (ICS) = 0x60 in slot 1, subslot 1
new IO controller output provider status (OPS) = 0x80 in slot 1, subslot 1
new IO controller input consumer status (ICS) = 0x80 in slot 1, subslot 1
**** out passivated ****
**** F output OK ****
f
```

Figure 3-22 Starting firmware update console

Then load the application firmware ("ecos200p_app_05.bin") onto the DevKit board. To do this, call "TcpFwLoader_EB200P.bat" (left PC). A CMD window opens. The loading progress is displayed in this window and in the terminal console.

When the terminal console stops, enter a "1" to select the flash type (NOR flash) and confirm your input.

The following steps are required to load the firmware.

1. Start the Tera Term console.
2. Start the TcpFwLoader.
3. Enter an "f" in the Tera Term console.

```

new IO controller output provider status (OPS) = 0x60 in slot 1, subslot 1
new IO controller input consumer status (ICS) = 0x60 in slot 1, subslot 1
new IO controller output provider status (OPS) = 0x80 in slot 1, subslot 1
new IO controller input consumer status (ICS) = 0x80 in slot 1, subslot 1
**** out passivated ****
**** F output OK ****
f

-----
TCP interface init ... OK
TCP interface wait on connection ... OK, established
Receive firmware image header ... Image size = 4132024
Allocate memory ... done
Receive firmware image data
.....
finished, 4132024 bytes received
Checksum = 0xe7a175d0 OK

!!! DO NOT SWITCH POWER OFF !!!

ERASE & PROGRAM FLASH:

Choose flash to be loaded:
[1] NOR flash
[2] SPI flash
1 ←

```

Figure 3-23 Stop of firmware update for entry of the flash type

3.4 Reading out the firmware version data

After a successful update, the command line and terminal should display the following:

```

COM10 - Tera Term VT
Datei Bearbeiten Einstellungen Steuerung Fenster Hilfe
TCP interface init ... OK
TCP interface wait on connection ... OK, established
Receive firmware image header ... Image size = 4128032
Allocate memory ... done
Receive firmware image data
.....1

finished, 4128032 bytes received
Checksum = 0xe7a2eb74 OK

!!! DO NOT SWITCH POWER OFF !!!

ERASE & PROGRAM FLASH:

Choose flash to be loaded:
[1]  NOR flash
[2]  SPI flash

(1) Erase flash ...
.....
OK, done

(2) Erase Verify ... OK, done

(3) Program Flash ...
.....
OK, done

(4) Program Verify ... OK, done
***Bsp_nv_data_clear***
NvData : Set Default Values

OK, Flashing firmware finished
-----
=> RESET your system, to start new firmware ...
-----
***Bsp nv data store: Completed ***

```

Figure 3-24 Firmware update successfully completed

To activate the new firmware, you must disconnect the supply voltage from the DevKit board. Then reconnect the DevKit board to the supply voltage.

The flash content is copied into RAM and started. In this case, the executable firmware is "ecos200p_app_05.bin"

After each firmware update, you must load the F-source address and F-destination address into non-volatile RAM of the device F-address assignment (initialization) (Page 37)

3.4 Reading out the firmware version data

To call the version data of the F-application in the terminal console, write a "V" in the Tera Term console. The firmware CRC and the version are then output in the console.



Figure 3-25 Firmware version console output

PSD-CRC corresponds to the CRC of PROFIsafe driver sources defined in "p_c_config.h" (depending on single channel or multi-channel PSD build variant). The CRCs have been defined according to "source_code_CRC.md" in the PSD directory. The version corresponds to the VersionId specified in "p_c_config.h".

```
/* get Firmware Version crc */
#define VersionData 0x06
#define VersionID ((P_Word)0x2F01u /* version 47 testversion 01 */
#ifdef REDUNDANT
    #define VersionCRC ((P_DWord)0x6B227F0Bul) /* source crc */
#else
    #define VersionCRC ((P_DWord)0xAB47F4F5ul /* source crc */
#endif
/***** CRC-Tab Memory-Model *****/
```

Figure 3-26 Definition of CRC and firmware version in p_c_config.h

3.5 F-address assignment (initialization)

The F-addresses must match the addresses configured in the TIA Portal. Once the F-addresses have been loaded, a Power Off/On is required.

3.5.1 Writing the F-source address and F-destination address

F-source address and F-destination address

F-source address

Open the Tera Term console and enter an "A".

The instruction to type in the F-source address in hex format follows. Type this in hex format into the console. See figure "Initialization" below.

3.5 F-address assignment (initialization)

F-destination address

The instruction to type in the F-destination address in hex format follows. Type this in hex format into the console. See figure "Initialization" below.

```
***Bsp_nv_data_store: Completed ***
new IO controller output provider status (OPS) = 0x60 in slot 1, subslot 1
new IO controller input consumer status (ICS) = 0x60 in slot 1, subslot 1
new IO controller output provider status (OPS) = 0x80 in slot 1, subslot 1
new IO controller input consumer status (ICS) = 0x80 in slot 1, subslot 1
A
type in F- source address in hex format eg. 0x0001 for source address 1
0x0001
type in F- destination address in hex format eg. 0x0064 for destination address 100
0x0064
***Bsp_nv_data_store:NvDataType=b, Size=4, CheckSum: ffff55e5 ***
**** Profisafe Source and Destination - Address set ****
**** param OK ****
```

Figure 3-27 Initialization

Preparation/configuration of standard source files

4.1 Preparation of standard source files

In order for the DevKit to also match the extended GSDML (section 2), you must change the source code.

- To do this, copy the directory "App1_STANDARD".
- Rename it to "App5_FAILSAFE_PSD_2_6_1".

4.2 Setting EXAMPL_DEV_CONFIG_VERSION

The "EXAMPL_DEV_CONFIG_VERSION" defined in App_common\usrapp_cfg.h specifies the example application to be compiled.

In this case: "App5_FAILSAFE_PSD_2_6_1". The definition takes place on a continuing basis and builds on the existing applications of the DevKit.

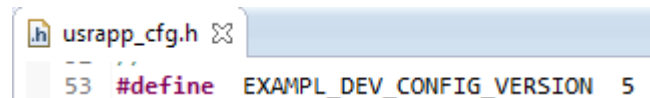


Figure 4-1 Setting of define EXAMPL_DEV_CONFIG_VERSION

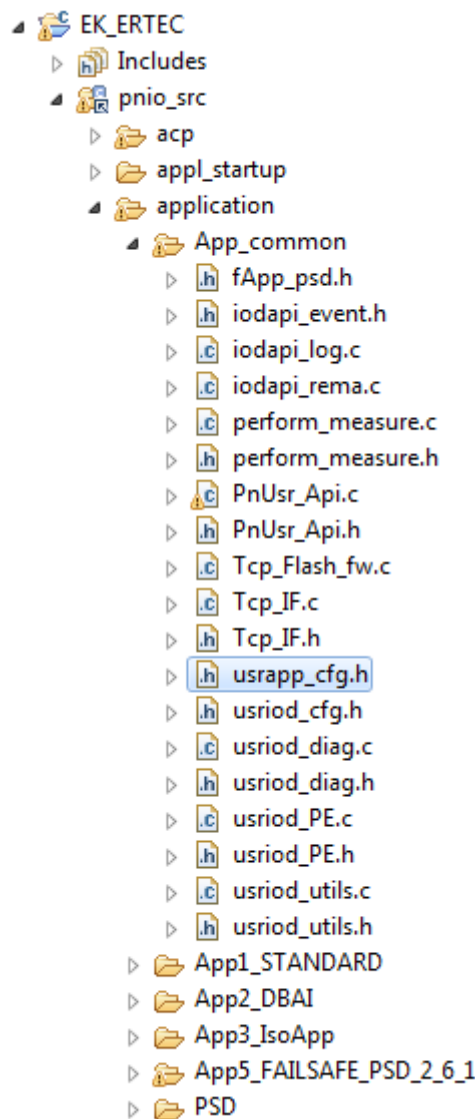


Figure 4-2 Path usrapp_cfg.h

To prevent function duplications, all "App5_FAILSAFE_PSD_2_6_1"-specific codes are referenced as follows:

```
88 #if (EXAMPL_DEV_CONFIG_VERSION == 5)
```

Figure 4-3 Define, to eliminate conflicts (function duplications)

4.3 Changes in the source file usriod_main.c

Directory "App5_FAILSAFE_PSD_2_6_1" contains the source file "usriod_main.c", in which the global variable "IoSubList" has been declared and initialized.

The settings/defines that are changed here must match the GSDML (DAP, ModuleIdentNumber SubModule, etc.). The settings defined here match the example GSDML of the TIA project contained in DevKit 4.7.

```
static PNIO_SUB_LIST_ENTRY IoSubList []
= {
    // Api Slot Subslot, ModId, SubId, InLen, OutLen, I&M0 support
    { 0, 0, 1, MODULE_ID_DAP, SUBMOD_ID_DEFAULT, 0, 0, PNIO_IMO_SUBMODULE + PNIO_IMO_DEVICE },
    { 0, 0, 0x8000, MODULE_ID_DAP, SUBMOD_ID_PDEV_IF, 0, 0, PNIO_IMO_SUBMODULE }, // PDEV interface
    { 0, 0, 0x8001, MODULE_ID_DAP, SUBMOD_ID_PDEV_PORT, 0, 0, PNIO_IMO_SUBMODULE }, // PDEV port1
    #if (IOD_CFG_PDEV_NUMOF_PORTS >= 2)
    { 0, 0, 0x8002, MODULE_ID_DAP, SUBMOD_ID_PDEV_PORT, 0, 0, PNIO_IMO_SUBMODULE }, // PDEV port2
    #endif
    #if (IOD_CFG_PDEV_NUMOF_PORTS >= 3)
    { 0, 0, 0x8003, MODULE_ID_DAP, SUBMOD_ID_PDEV_PORT, 0, 0, PNIO_IMO_SUBMODULE }, // PDEV port3
    #endif
    #if (IOD_CFG_PDEV_NUMOF_PORTS >= 4)
    { 0, 0, 0x8004, MODULE_ID_DAP, SUBMOD_ID_PDEV_PORT, 0, 0, PNIO_IMO_SUBMODULE }, // PDEV port4
    #endif
    { 0, 1, 1, EXAMPL_MOD_ID_SLOT1, SUBMOD_ID_DEFAULT, 1, 1, PNIO_IMO_SUBMODULE } // IO subslot
    /*{ 0, 2, 1, EXAMPL_MOD_ID_SLOT2, SUBMOD_ID_DEFAULT, 1, 1, PNIO_IMO_SUBMODULE } // IO subslot*/
};
```

Figure 4-4 Variable `IoSubList` with entries adapted to the GSDML

The same applies to the global variable "`IoSubList`" in header file "`usrapp_cfg.h`".

```
63 #if (PNIOD_PLATFORM & PNIOD_PLATFORM_EB200P)
64 #define MODULE_ID_DAP_NO_PDEV 0x01 // DAP 1 no PDEV, for old pnioc controller only
65 #define MODULE_ID_DAP_2PORT_NO_MRP 0x02 // DAP 2 standard, no MRP capability, but Fast Startup
66 #define MODULE_ID_DAP_SWITCH 0x03 // DAP 3, including switch with more than one port
67 #define MODULE_ID_DAP_1PORT 0x04 // DAP 4 standard, no MRP capability, but Fast Startup
68 #define MODULE_ID_DAP_POF_1PORT 0x05 // DAP 5, including physical device for 1 fiber optic (POF)
69 #define MODULE_ID_DAP_POF 0x06 // DAP 6, including physical device for fiber optic (POF)
70 #define MODULE_ID_DAP_S2 0x07 // DAP 7, including switch with more than one port, S2 redundancy enabled
71 #define MODULE_ID_DAP_DR 0x08 // DAP 8, S2 redundancy with DR (CiR)
72 #define MODULE_ID_DAP_FAILSAFE_2_6_1 0x09 // Set in the GSDML of the DevKit
73 #else
74 #define MODULE_ID_DAP_NO_PDEV 0x01 // DAP 1 no PDEV, for old pnioc controller only
75 #define MODULE_ID_DAP_1PORT 0x02 // DAP 2 standard, no MRP capability, but Fast Startup
76 #endif
77
78 /* Consistency check - Some config combinations are invalid -> inform user */
115 #if (IOD_INCLUDE_POF == 0)
116 #if (IOD_INCLUDE_MRP == 1)
117 #define MODULE_ID_DAP MODULE_ID_DAP_FAILSAFE_2_6_1
118 #else
119 #define MODULE_ID_DAP MODULE_ID_DAP_2PORT_NO_MRP
120 #endif
121 #else
122 #define MODULE_ID_DAP MODULE_ID_DAP_POF
123 #endif
```

Figure 4-5 Define of the DAP

4.3 Changes in the source file `usriod_main.c`

The number for the definition "MODULE_ID_DAP_FAILSAFE_2_6_1" can be found in the GSDML under the name "ModuleIdentNumber" in the DAP (see Figure 2-1 [\(Page 8\)](#)).

```
94 // *-----  
95 // *   define IO module IDs  
96 // *   (must fit to GSD file)  
97 // *-----  
98 #define IO_MODULE_1_BYTE_INOUT      0x20  
99 #define IO_MODULE_1_BYTE_IN         0x21  
100 #define IO_MODULE_1_BYTE_OUT        0x22  
101 #define IO_MODULE_64_BYTE_INOUT     0x29  
102 #define IO_MODULE_64_BYTE_IN        0x30  
103 #define IO_MODULE_64_BYTE_OUT       0x31  
104 #define IO_MODULE_64_BYTE_IN_IRT    0x50  
105 #define IO_MODULE_64_BYTE_OUT_IRT   0x51  
106 #define IO_MODULE_MULTISUBSLOT      0x60  
107  
108 #define IO_MODULE_FAILSAFE_2_6_1    0x71
```

Figure 4-6 Define of the module

The number for the definition "IO_MODULE_FAILSAFE_2_6_1" has been defined in the GSDML under the name "ModuleItemTarget" in the DAP (see Figure 2-2 [\(Page 8\)](#)).

Configuration of the PROFIsafe driver

The "Application example for implementation of the PROFIsafe driver on the Evaluation Kit EK-ERTEC 200P-2 V4.7" corresponds with single-channel operation of the PROFIsafe driver. Changes to the configuration of the PROFIsafe driver require integration of the sources from the StarterKit CD, as described in section 3.1.2 Configuration for the debugger.

5.1 Header file p_c_config.h

5.1.1 NOF_INSTANCES

Set the define "NOF_INSTANCES" in this implementation to 1 since the PROFIsafe driver is implemented with one instance.

```
/* Number of PSD-Instances. All configuration parameters are valid for all PSD-Instances */
/* maximum 32 instances allowable */
/* example: NOF_INSTANCES 3: the instances with the instance IDs INST_01, INST_02 and INST_03 are usable */
#ifndef NOF_INSTANCES
#define NOF_INSTANCES 1
#define INST INST_01
#endif
```

Figure 5-1 Define for PROFIsafe instances

See profisafe_driver_v2_2_2_programming_manual_en-US.pdf (section 5.1).

5.1.2 REDUNDANT

You can set or comment out the define "REDUNDANT". Both operating modes are supported by this implementation.

```
27 /* with define = Two Channel */
28 /* no define = One Channel */
29 #define REDUNDANT
```

Figure 5-2 Define for REDUNDANT operating mode

Dual-channel operation

With the define "REDUNDANT", the PROFIsafe driver operates as in a dual-channel controller architecture. The DevKit has a single-channel hardware design. The necessary dual-channel architecture is simulated in the existing application (App_05). For this purpose, the functions psd_OutTransfer(), psd_OutSync(), psd_InTransfer() and psd_InSync() are provided by the application to synchronize the "two" software controllers. You can find a description of the functions in profisafe_driver_v2_2_2_programming_manual_en-US.pdf (section 4.4).

Single-channel operation

Without the define "REDUNDANT", the PROFIsafe driver operates as a single-channel driver. The inverse functions are called in this case. See

profisafe_driver_v2_2_2_programming_manual_en-US.pdf (section 5.2 for redundancy, section 4.5.8 for single-channel operation).

5.1.3 Length configuration

The defines "OUTDATA_MAX_LEN_INST_01" and "INDATA_MAX_LEN_INST_01" define the user data length of the F-telegram. Change the defines to match the settings in the GSDML. (see Optional changes (Page 14))

In this implementation, the user data of the F-telegrams has a length of 1 byte.

The other instances are set to 0 as they are not being used.

```

/***** Input/Output data length *****/
/* Number of user data output- and input bytes for each instance: */
/* maximal 123 I/O-Bytes in CRC-Seed mode, */
/* maximal 12 I/O-Bytes in no-CRC-Seed mode, */
/* 0 no I/O-Bytes */
#define OUTDATA_MAX_LEN_INST_01 1u
#define INDATA_MAX_LEN_INST_01 1u
#define OUTDATA_MAX_LEN_INST_02 1u
#define INDATA_MAX_LEN_INST_02 1u
#define OUTDATA_MAX_LEN_INST_03 1u
#define INDATA_MAX_LEN_INST_03 1u
#define OUTDATA_MAX_LEN_INST_04 0
#define INDATA_MAX_LEN_INST_04 0

```

Figure 5-3 Setting of the length configuration

5.1.4 Source/destination address; SIL class

These parameters specify the fail-safe address assignment and the safety integrity level (SIL) of the F-module.

The define "F_DEST_ADD" matches the destination address. Since the DevKit has neither a DIP switch nor a coding element for setting the address, the destination address is permanently set to 0x0001.

The F-source address is taken from the configuration parameters. The SIL class has been permanently set to SIL3 in the F-application ("Fapp_Init" function in "PSD\p_c_pseudo.c").

NOTE

You must configure the PROFIsafe destination address and SIL as defined in the application during engineering. Different settings result in a parameter error.

5.1.5 Telegram properties

The defines "IN_TELE_LEN" and "OUT_TELE_LEN" define the length of the F-telegrams with user data (1 byte), control/status byte (1 byte) and CRC (4 bytes).

The define "FPAR_LEN" is defined with a length of 14 bytes and 2 bytes optional WD2. This configuration has been defined in the GSDML file (see section Module (slot submodules) (Page 9), figure 2-5)).

The settings/defines must match the GSDML.

You can find more information on the F-parameter structure in the document PROFIsafe-Profile_3192_V261_Aug14.pdf (section 8.1.9.).

5.2 Header file `psd_interface.h`

The file "psd_interface.h" can be found under "`..\DevKit4.7_PSD_Image\pn_ioddevkits\src\application\App5_FAILSAFE_PSD`". It contains all the declarations used for the PROFIsafe implementation.

5.2.1 Slot and subslot

The defines "FAPP5_SLOT_NR" == 1 and "FAPP5_SUBLSOT_NR" == 1 correspond with a modeling with one slot and one subslot. See section 2.2.2 Module (slot submodules).

5.2.2 Parameters for telegram properties

The defines "F_PAR_DS" and "F_PAR_LEN" define the data record and telegram length of the F-parameters.

You can find more information on the F-parameter structure in the document PROFIsafe-Profile_3192_V261_Aug14.pdf (section 8.1.9.).

The defines "I_PAR_DS", "I_PAR_LEN" and "I_PAR_USER_DATA" define the data record and telegram length of the I-parameters. I-parameters are not used in the F-application of this implementation.

You can find more information on the I-parameter structure in the document PROFIsafe-Profile_3192_V261_Aug14.pdf (section 8.2.).

5.2.3 Interface for BusEvents

The structure BusEvents defines control events for the state machine of the F-application. The events are set or reset, depending on their state, both in the F-application of the PSD and from the DevKit interface.

The type BusEvents is declared as follows:

```
typedef struct
{
    /**
     * New parameters have been received. The application may reject or assign new
     * parameters.
     */
    unsigned char hasNewParameter;
    unsigned char hasNewFParameter;
    /**
     * error state set on hard fail 1 = hard fail 0 = OK
     */
    unsigned char errorState;
    /**
     * Process data output disabled message received from bus. (e.g. CPU in Stop mode)
     */
    unsigned char processDataOutputIsDisabled;
    /**
     * A new process data telegram has been received.
     */
    unsigned char hasNewProcessOutputData;
    /**
     * Bus has been reset and now it is up again.
     */
    unsigned char busHasBeenReset;
    /**
     * New process data received.
     */
    unsigned char procDatReceived;
    /**
     * Process data for sending to FHost prepared.
     */
    unsigned char procDatSendRequ;
    /**
     * IO device Application ready received from PROFINet
     */
    unsigned char iodApplicationReady;
    /**
     * fault insertion test active or version request over record 240
     */
    unsigned char faultInsertionTest;
}BusEvents;
```

Figure 5-4 Type BusEvents

5.2.4 Diagnostics defines

Defines such as "FAPP_F_DEST_ADD_MISMATCH" are error codes that are displayed by the F-host. For security reasons, the PROFIsafe driver uses Hamming codes as return values for its functions. The return values must be mapped onto the standardized error codes by the application.

The mapping is described in section 12 Mapping the error numbers ([Page 87](#)).

You need defines such as "NO_ELEMENT_ACTIVE" and "NOT_QUITTED" for diagnostics handling.

5.2.5 Diagnostics type DiagType

The diagnostics type "DiagType" has been defined as follows:

```

95 // *****
96 // *   Diagnose-structure
97 // *****
98
99 typedef struct
100 {
101     P_Byte isDiagCreateQuitted;
102     P_Byte isDiagDeleteQuitted;
103     PNIO_UINT16 activeErrorCode;
104     PNIO_UINT16 diagQueue;
105 }DiagType;

```

Figure 5-5 Diagnostics structure

The diagnostics type is needed for diagnostics handling.

Explanation of the variables:

isDiagCreateQuitted	TRUE: Create – Acknowledgment
isDiagDeleteQuitted	TRUE: Delete – Acknowledgment
activeErrorCode	Mapped error number
diagQueue	Diagnostic buffer

State machine

The state machine described below is used for the F-application. The application state machine is controlled by events and states from the PSD. The implementation of the F-application is based on sources of PROFIsafe StarterKit V3.5.1 (6ES7195-3BF03-0YA0).

6.1 Structure of state machine

A dedicated state machine is used for the F-application.

The variables "fApp_State", "param_state", "conf_state" and "com_state", which contain the status of the state machine, have been declared globally for this purpose. The variables have been declared in file "p_c_pseudo.c" in directory "App5_FAILSAFE_PSD\PSD" and initialized as follows:

```
/* global variables */
P_Byte      error_state = 0;          /* TRUE / FALSE */
P_Byte      f_app_state = INITIALIZE; /* INITIALIZE / NO_CYCLIC_DATAEX / CYCLIC_DATAEX */
P_Byte      param_state = 0;          /* INITIALIZE / PARAM_FPAR_OK / PARAM_FPAR_NOT_OK */
P_Byte      conf_state = 0;           /* INITIALIZE / CONFIG_OK / CONFIG_NOT_OK */
P_Byte      com_state = 0; /* INITIALIZE / DATAEX_OUT_RCV / DATAEX_OUT_GET / DATAEX_IN_SET / DATAEX_IN_SEND */
```

Figure 6-1 Declaration and initialization of the state machine

The **F-application** "f_app_state" has three states:

INITIALIZE	State after Power On, application has not yet been initialized
NO_CYCLIC_DATAEX	F-application has been initialized but not yet parameterized. No process data exchange is taking place.
CYCLIC_DATAEX	F-application has been parameterized, and a plausibility check of the F-parameters was OK. Cyclic data exchange is active

Parameter assignment "param_state" (F-parameters received) is signaled with the following states:

INITIALIZE	No F-parameters have been received yet
PARAM_FPAR_OK	Check of F-parameters was OK
PARAM_FPAR_NOT_OK	Error detected during check of F-parameters

States of **F-telegram length configuration** "conf_state" are:

INITIALIZE	No length configuration received yet
CONFIG_OK	Length configuration received and result of telegram length check matches sum of process data length and PSD trailer
CONFIG_NOT_OK	Error detected during check of length configuration; discrepancy in process data length or length of PROFIsafe trailer

The cyclic **F-process data exchange** "com_state" has the following states:

INITIALIZE	No process data exchange is taking place yet (after Power On).
------------	--

DATAEX_OUT_RCV	F-application is ready to receive the next output telegram.
DATAEX_OUT_GET	A new valid output telegram has been received from the F-host.
DATAEX_IN_SET	F-application is ready to set the input data.
DATAEX_IN_SEND	Input data has been set, F-application is ready to send the input telegram to the F-host.

6.2 State diagram

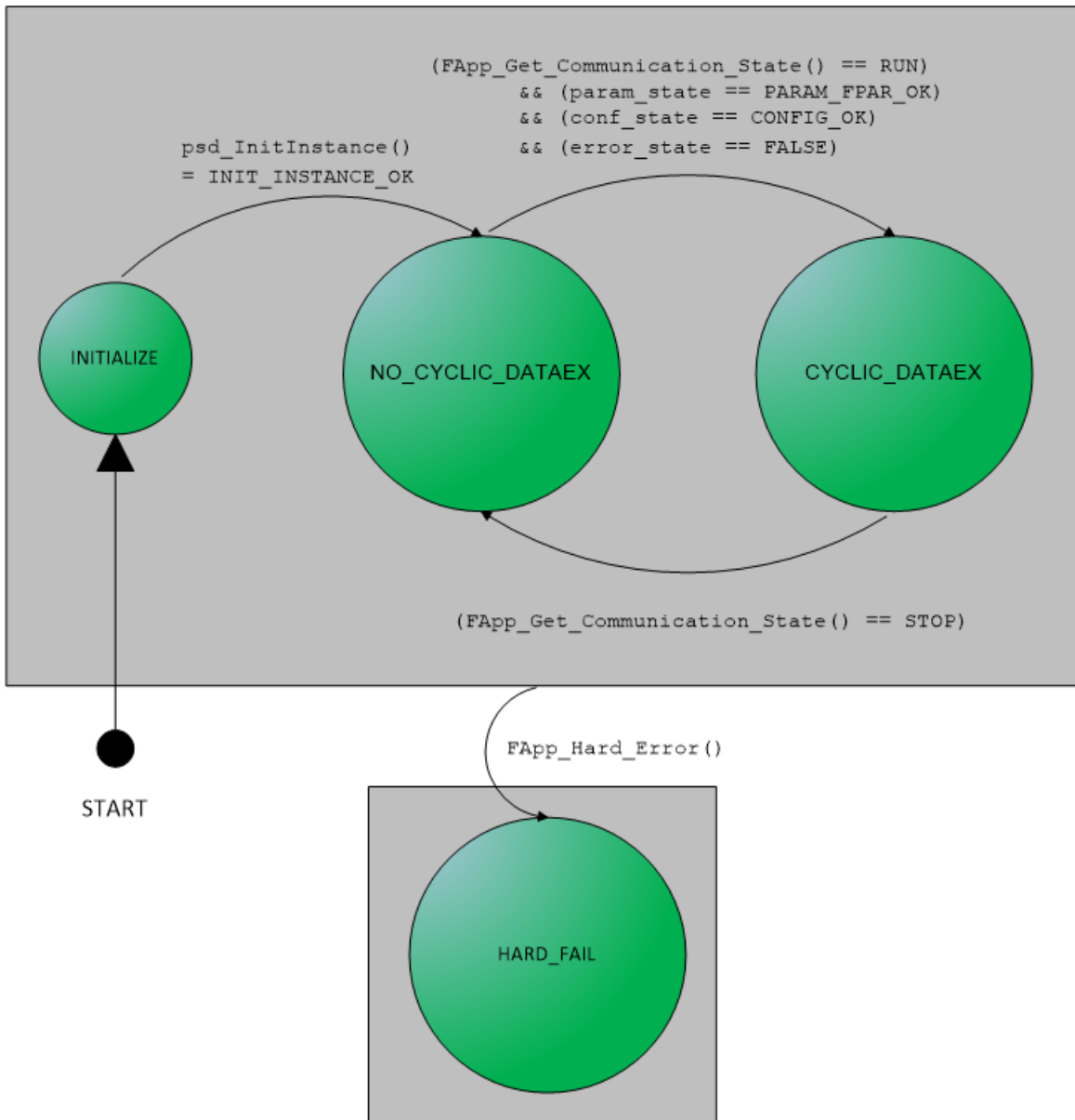


Figure 6-2 State diagram

6.3 Transition table

START → INITIALIZE

After Power On, the F-application initializes with default values. The state machine is set to INITIALIZE state.

INITIALIZE → NO_CYCLIC_DATAEX

Once the PROFINET device has received its configuration data, the application and PROFIsafe driver are initialized with call of psd_InitInstance().

After error-free completion (return value: "INIT_INSTANCE_OK"), the state machine is in NO_CYCLIC_DATAEX state.

NO_CYCLIC_DATAEX

When the F-parameter data record is received (in this application: data record number 128), the function psd_FParBuild() is called. If executed successfully (return value: FPAR_BUILD_OK), the PROFIsafe driver goes to param_state PARAM_FPAR_OK; if an error is detected, it goes to PARAM_FPAR_NOT_OK.

After receipt of new length configuration user data from the F-host, the function "psd_Config()" is called. This sets "conf_state" to CONFIG_OK if the check is error-free and to CONFIG_NOT_OK if an error is detected.

Besides process data, the cyclically called DevKit function PNIO_cbf_data_read() (iodapi_event.c) supplies the io controller provider status (IOPS). This sets the event processDataOutputsDisabled using interface function psdInterfaceProcessDataReceived() (psd_interface.c).

When processDataOutputsDisabled = 1, the function FApp_Get_Communication_State() (p_c_pseudo.c) called in the state machine supplies return value "STOP"; when processDataOutputsDisabled = 0, "RUN" is returned.

NO_CYCLIC_DATAEX → CYCLIC_DATAEX

If no error is present (event errorState = 0), parameter assignment was successful (param_state = PARAM_FPAR_OK), length check was error-free (conf_State = CONFIG_OK) and communication is in "RUN" state (IOPS value 0x80), the state machine changes to CYCLIC_DATAEX state. The subsequent call of FApp_Start_Communication (p_c_pseudo.c) sets com_state to DATAEX_OUT_RCV and starts the cyclic process data exchange via PROFIsafe.

CYCLIC_DATAEX

When com_state = DATAEX_OUT_RCV, the F-application is ready to receive a new process data telegram. Event procDatReceived = 1 starts the interface function psdInterfaceReceiveProcessData (psd_interface.c). This copies the process output data received from the F-host. The subsequent call of FApp_Read_FOutput_Telegram checks the output data and, if OK, sets com_state to DATAEX_OUT_GET. If the output telegram is error-free, the data is copied to the process data buffer and switched to the corresponding output LED ports. com_state changes to DATAEX_IN_SET. The function psd_SetInData (p_c_pseudo.c) prepares the next input data telegram for the F-host. com_state changes to DATAEX_IN_SEND and thereby signals readiness to send the input telegram. psdInterfaceSendProcessData (psd_interface.c) copies the data of the input telegram to the data record buffer and sets the event procDatSendRequ. When com_state = DATAEX_OUT_RCV, the cyclic process data exchange via PROFIsafe restarts.

CYCLIC_DATAEX → NO_CYCLIC_DATAEX

Besides process data, the cyclically called DevKit function PNIO_cbf_data_read() (iodapi_event.c) supplies the io controller provider status (IOPS). This sets the event processDataOutputsDisabled using interface function psdInterfaceProcessDataReceived() (psd_interface.c).

When `processDataOutputsDisabled = 1`, the function `FApp_Get_Communication_State()` (`p_c_pseudo.c`) called in the state machine supplies return value "STOP". When `FApp_Stop_Communication()` (`p_c_pseudo.c`) is called, the PROFIsafe communication stops, process data becomes 0 and the state machine switches to `NO_CYCLIC_DATAEX` state.

... → **HARD_FAIL**

If a PROFIsafe function supplies impermissible return values, the function `Fapp_Hard_Error()` is called. This sets event `errorState = 1`. User data including relevant GPIOs are set to fail-safe values and the PROFIsafe communication is stopped. The `HARD_FAIL` state can be assumed from any state of the state machine.

Embedding the PROFI-safe driver into the F-device

7.1 Embedding the PROFI-safe driver into the F-device

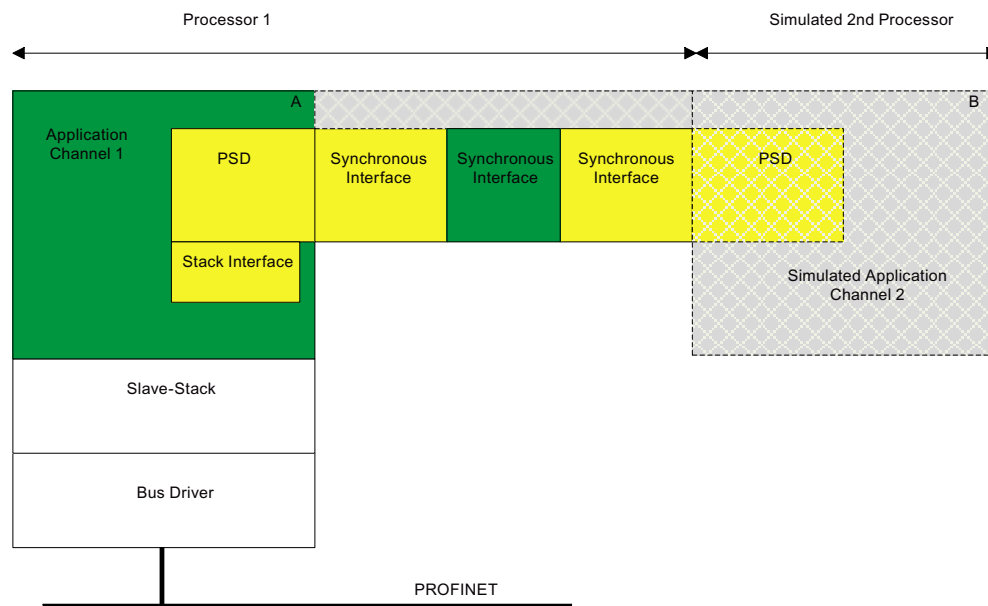


Figure 7-1 Embedding the F-device in redundant operation

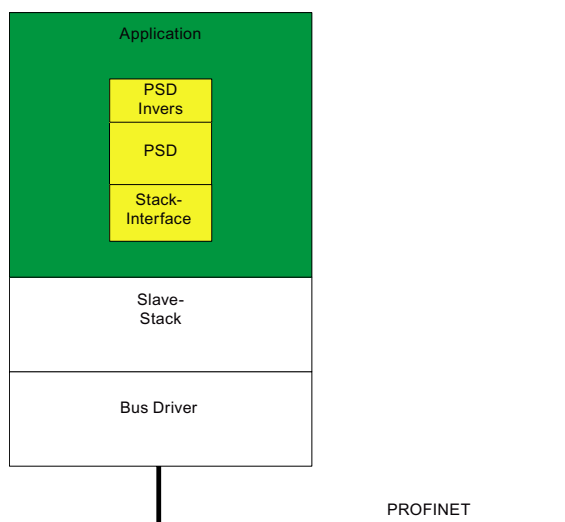


Figure 7-2 Embedding the F-device in non-redundant operation

NOTE

Implementation on single-channel and dual-channel architecture is described in section 2.1 of manual "PROFIsafe Driver V2.2.3 for F-Slaves" (link to manual: <https://support.industry.siemens.com/cs/ww/en/view/109769384>)

Sequence diagrams

8.1 Error-free sequence

8.1.1 Startup of configuration/parameter assignment

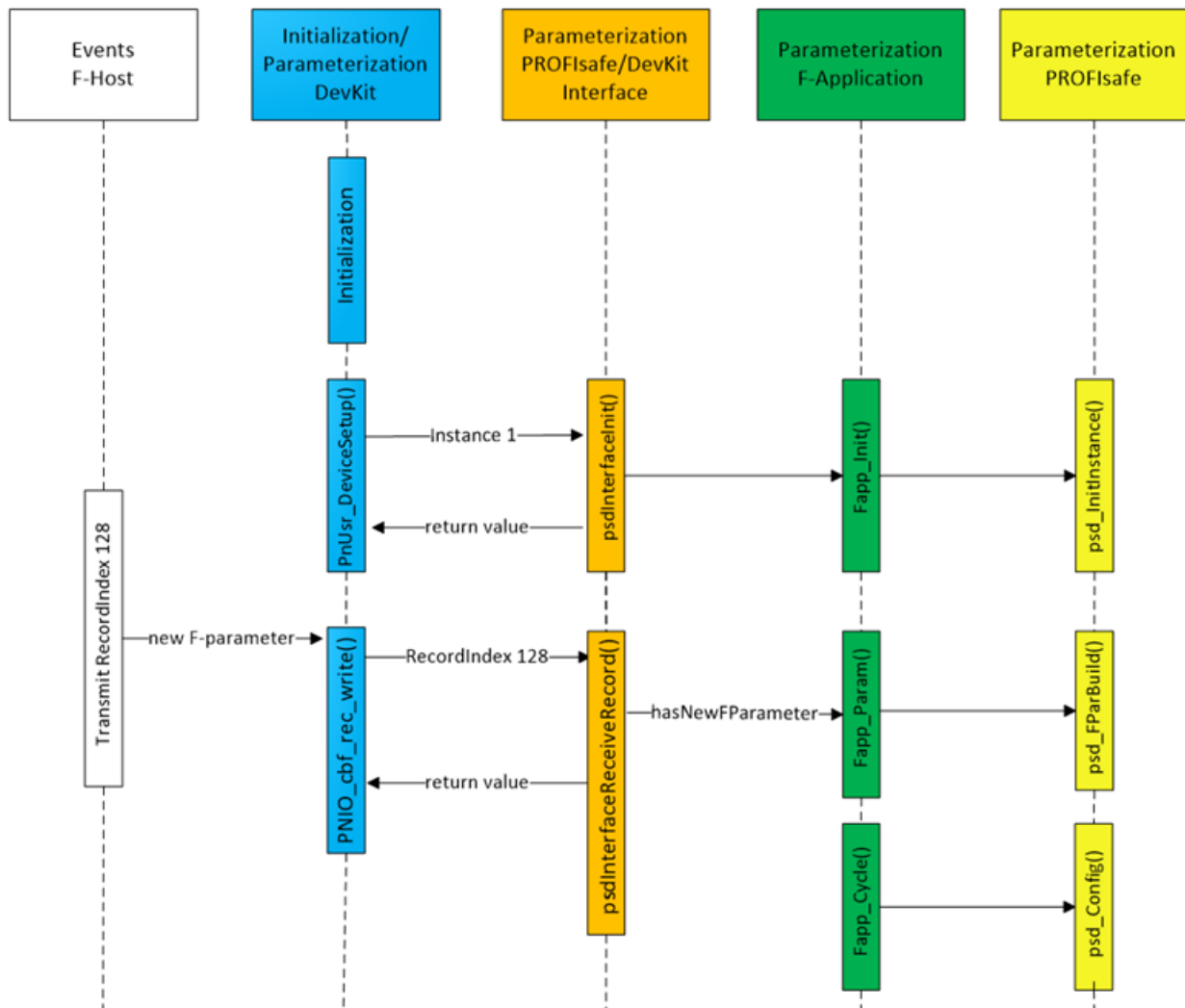


Figure 8-1 Startup of application

8.1.2 Transition from STOP to RUN

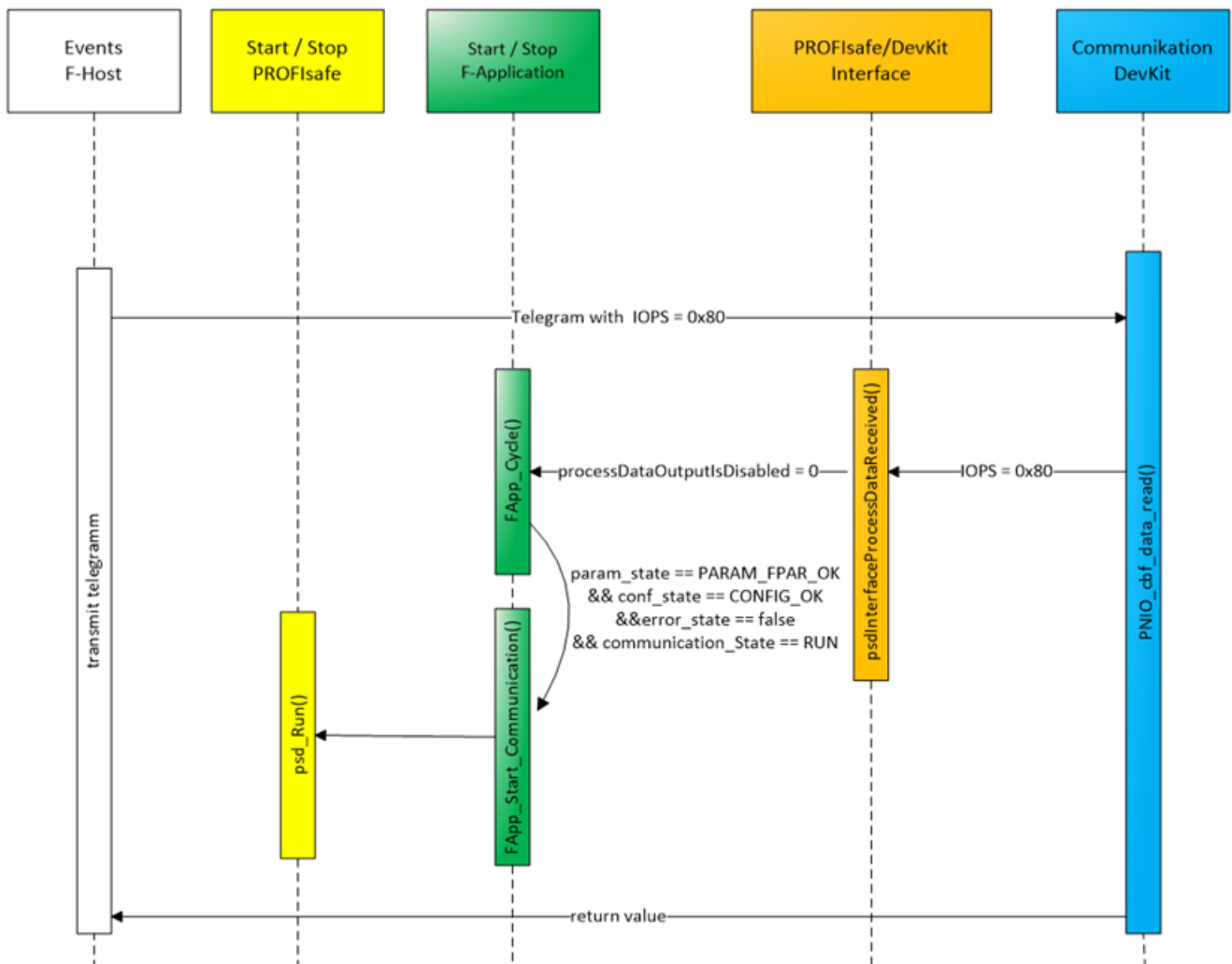


Figure 8-2 Startup of communication

8.1.3 Cyclic data exchange

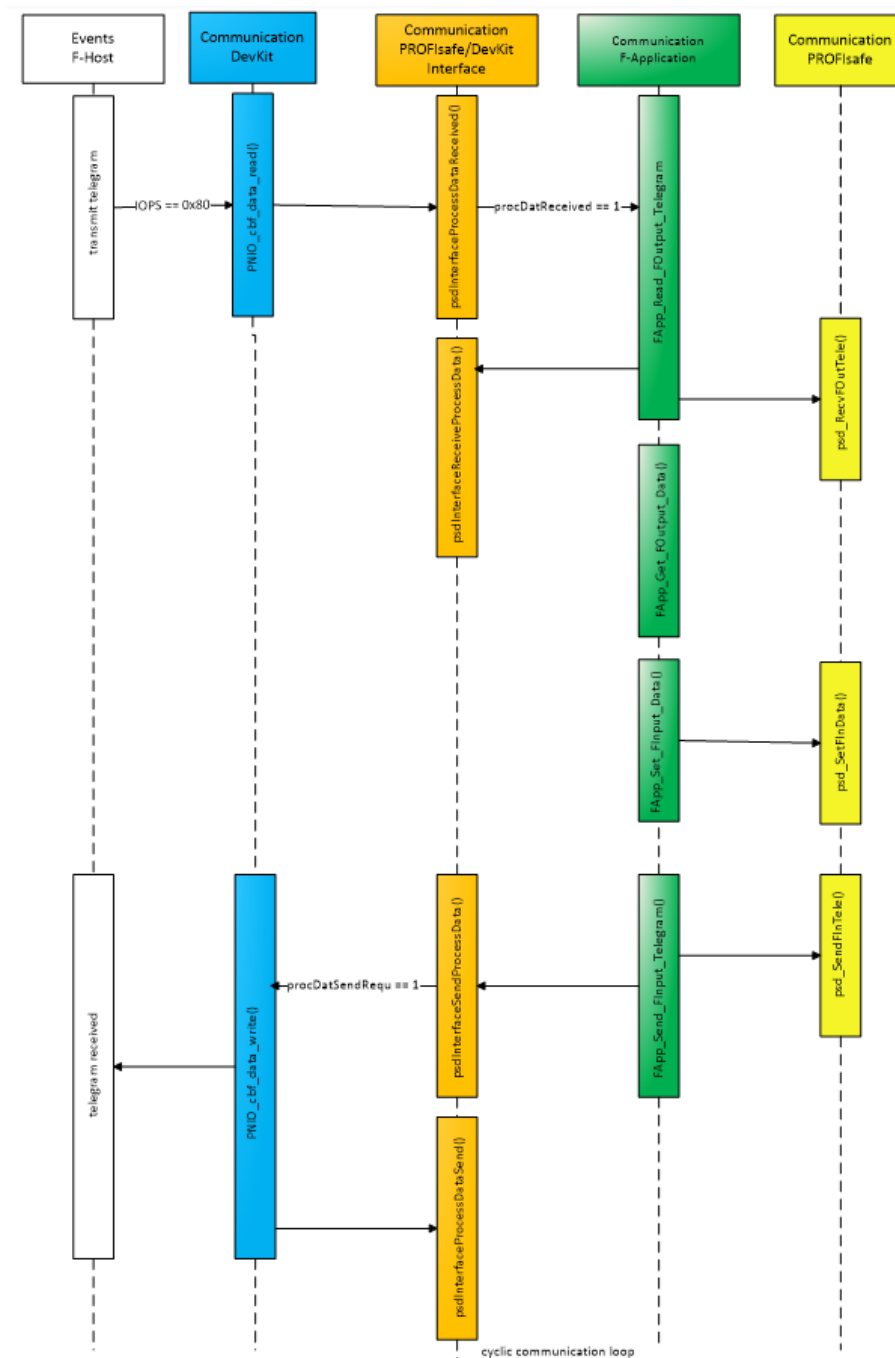


Figure 8-3 Cyclic data exchange

8.1.4 Stopping of PROFIsafe communication

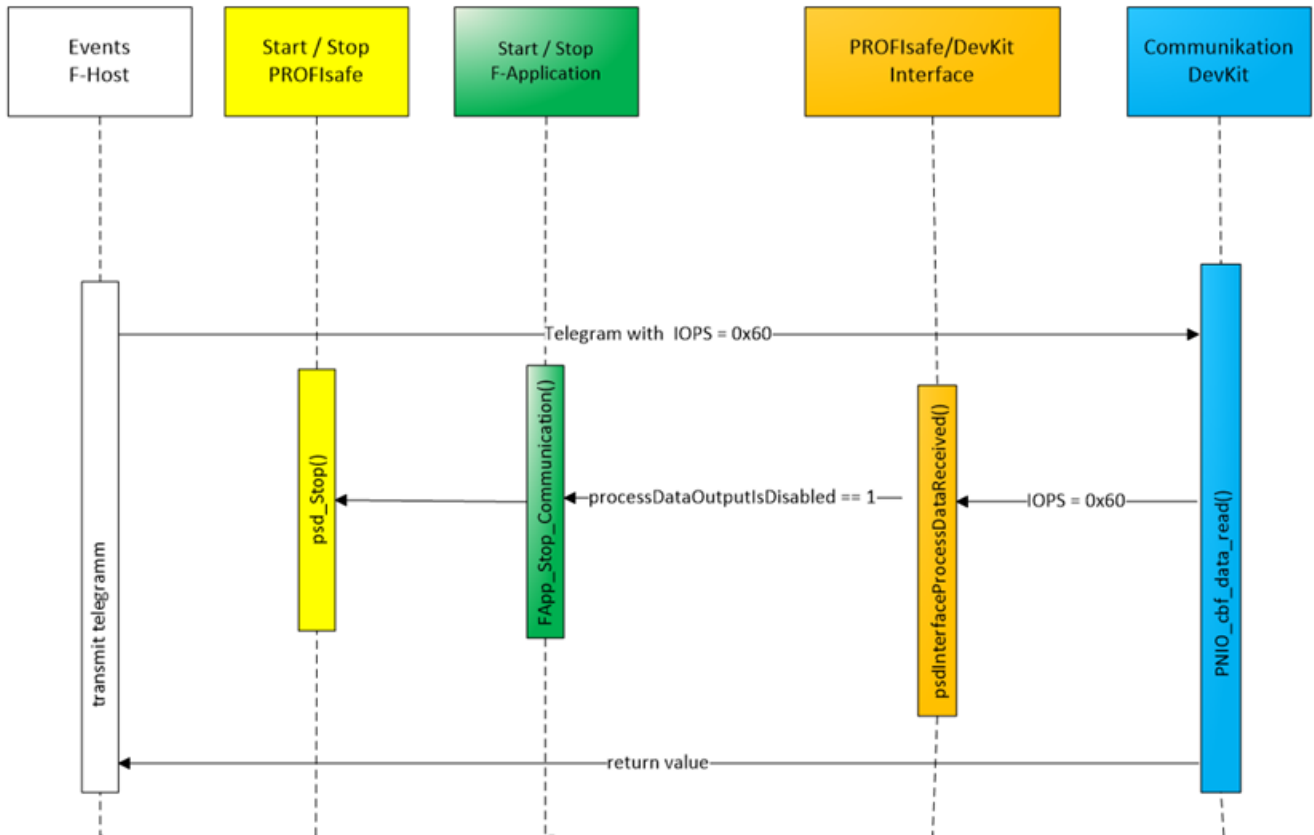


Figure 8-4 Stop of PROFIsafe communication

8.2 Error scenarios

8.2.1 Errors during parameter assignment

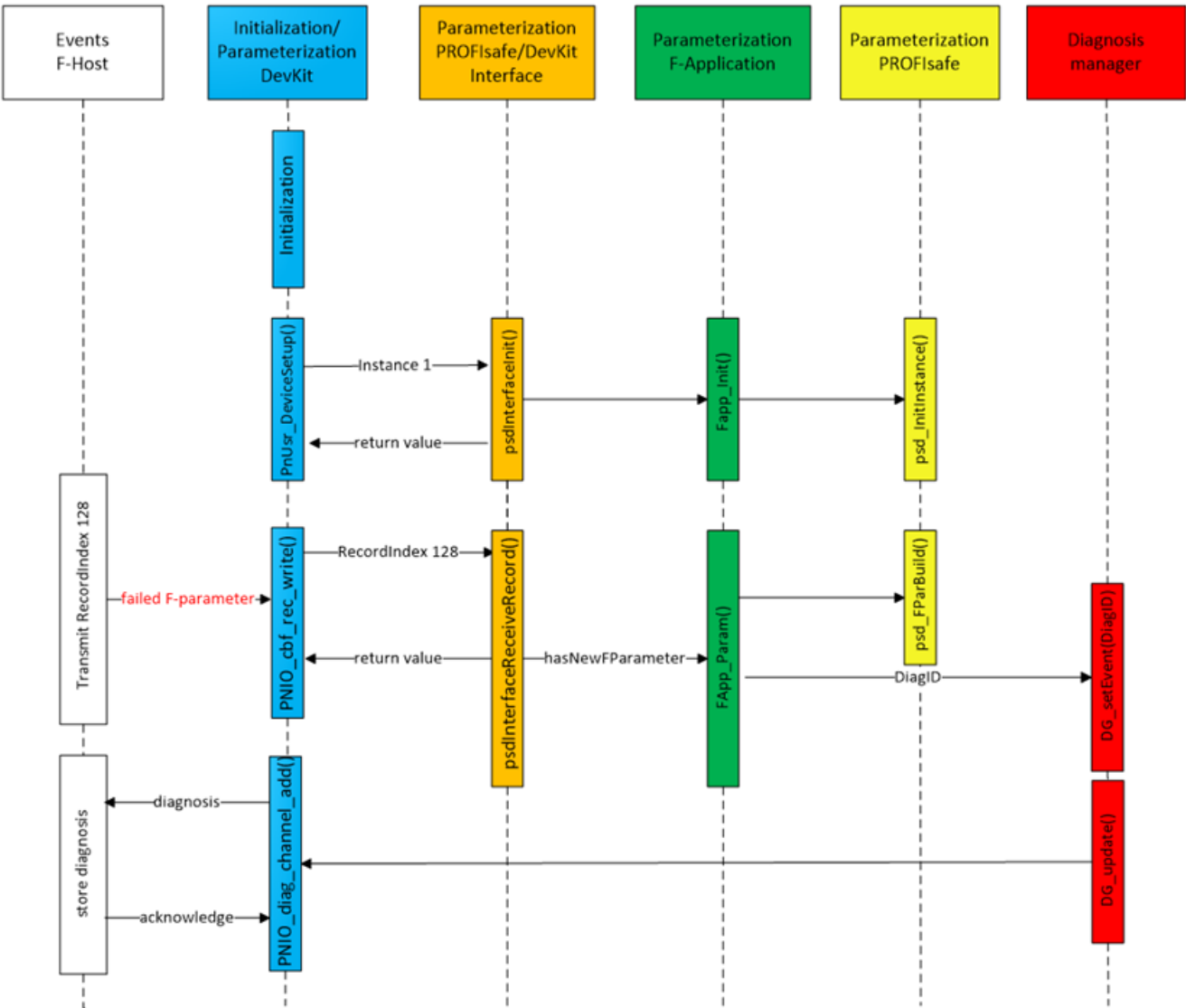
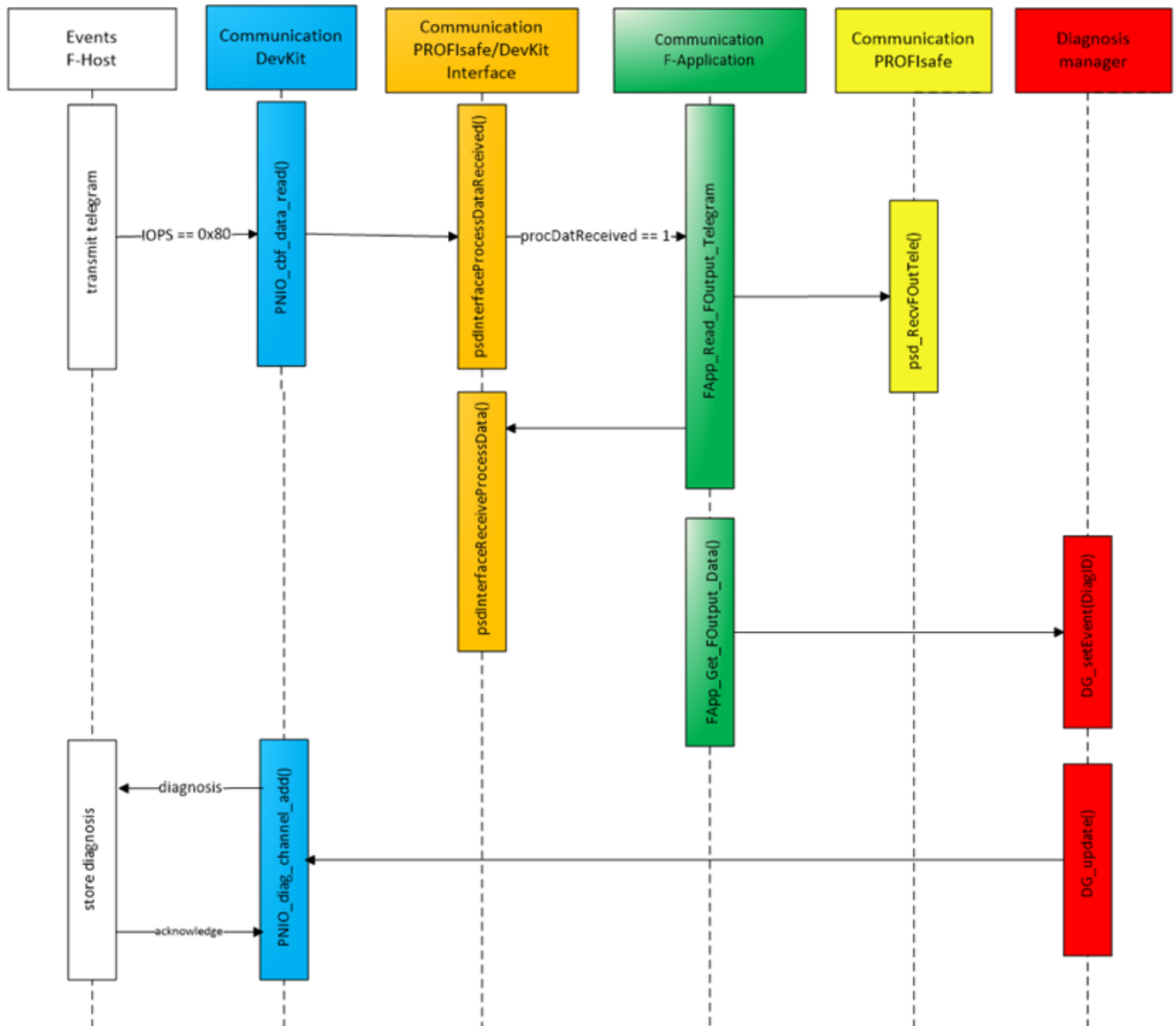


Figure 8-5 Incorrect parameter assignment

8.2.2 Telegram error in state "CYCLIC_DATAEX; DATAEX_OUT_RCV"



* An error can occur if the telegram is corrupted during transmission or if there is an error during copying in the memory (inconsistency).

Figure 8-6 Telegram error

Program flow diagram

The following legend applies to all diagrams in section 9

Legende:







	DevKit -Funktionen
	F-Applikation/DevKit Interface
	F-Applikation
	PROFIsafe
	Diagnose
	Hard Fail

Figure 9-1 Legend

The functions shown in the program flow diagram are presented in more detail below based on the PROFIsafe driver.

The diagnostics is examined separately since it is processed in its own thread.

PROFIsafe properties of the application are initialized (corresponding to FW parameters).

The outputs are initialized to zero (fail-safe values)

The figure below shows the program flow diagram of the DevKit:

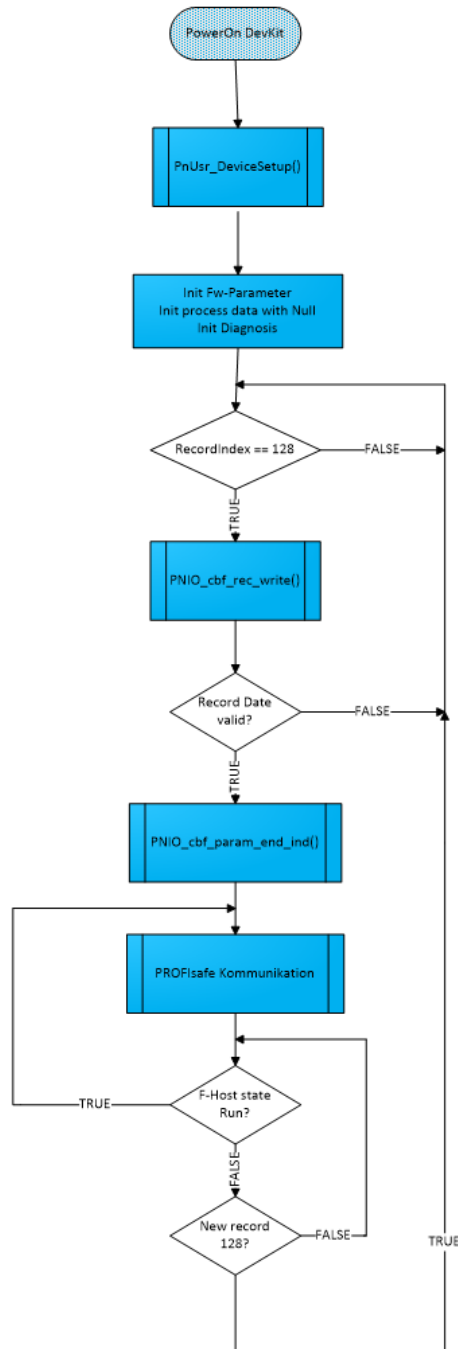


Figure 9-2 Flow of application - new

9.1 PnUsr_DeviceSetup()

The function "PnUsr_DeviceSetup()" is called when the DevKit is started.

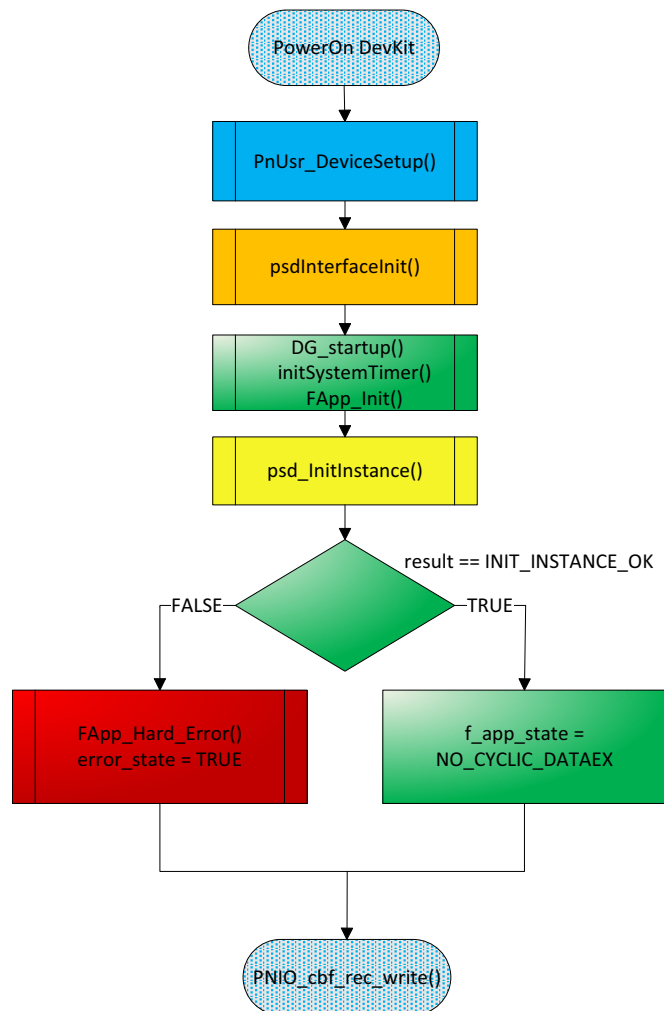


Figure 9-3 PnUsr_DeviceSetup()

The PROFIsafe driver is initialized in this function with the instance 1.

9.2 PNIO_cbf_rec_write()

Evaluating F-parameters

If a record data record is received, this function is called.

In the example application, only the F-parameters are evaluated. The F-parameters are sent by the F-host with data record 128.

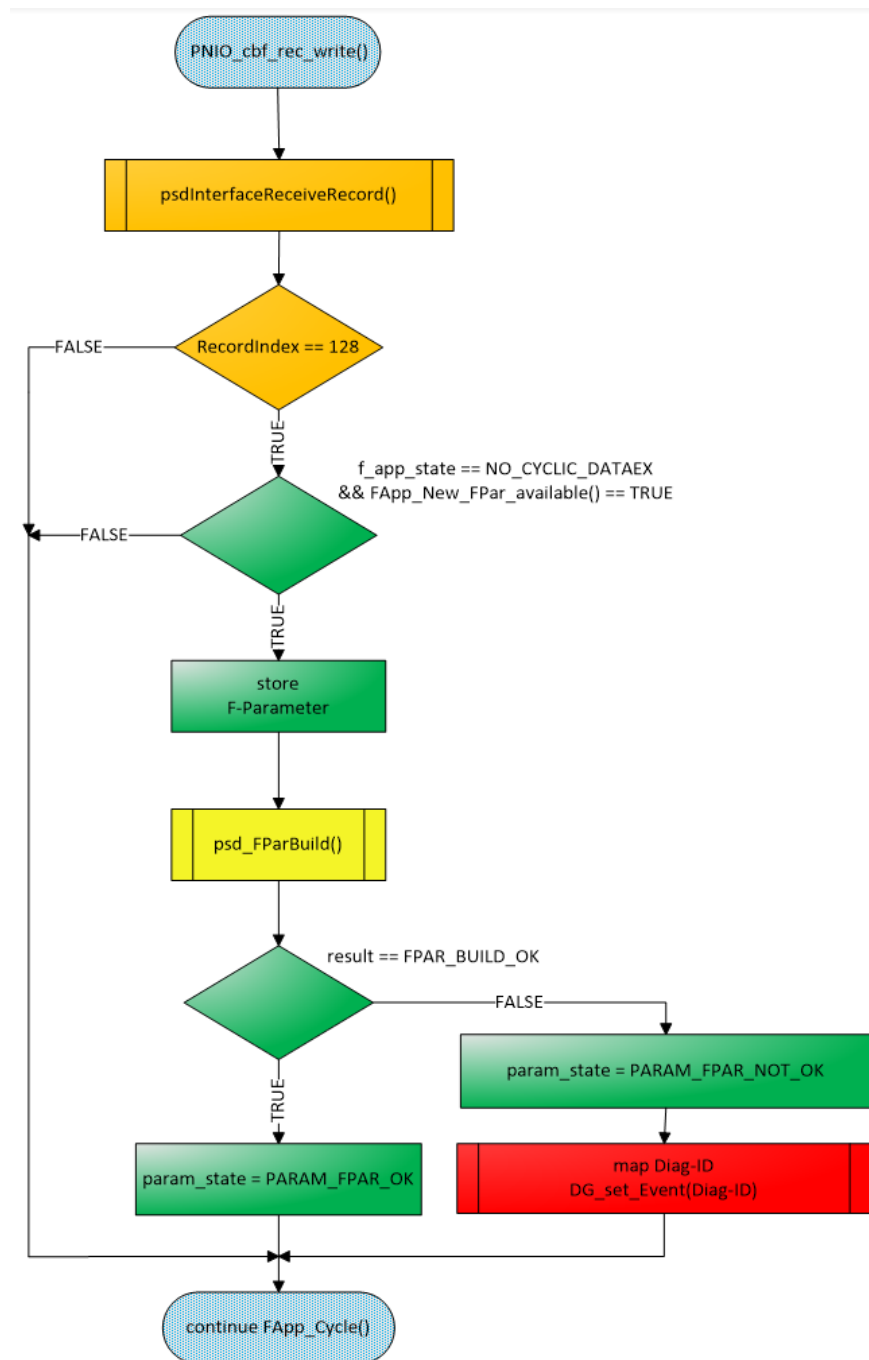


Figure 9-4 Evaluation of F-parameters

Length configuration of process data

This block is called in NO_CYCLIC_DATAEX state of the state machine.

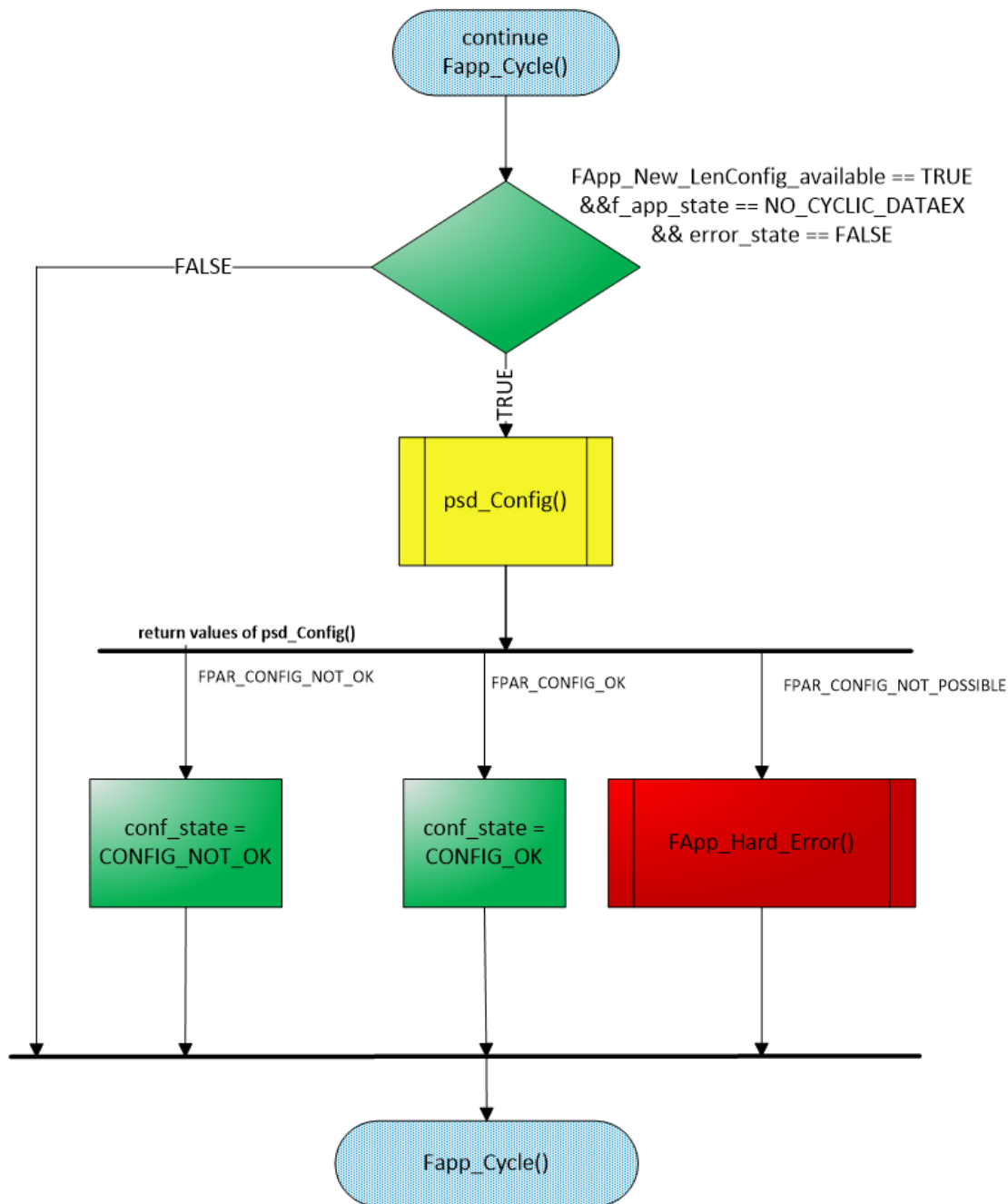


Figure 9-5 Configuration

9.3 PNIO_cbf_data_read()

F-host status

The IOPS is generated by the F-host and shows whether the F-host is in RUN or STOP state:
IOPS 0x80 (GOOD) = F-host in RUN
IOPS 0x60 (BAD) = F-host in STOP

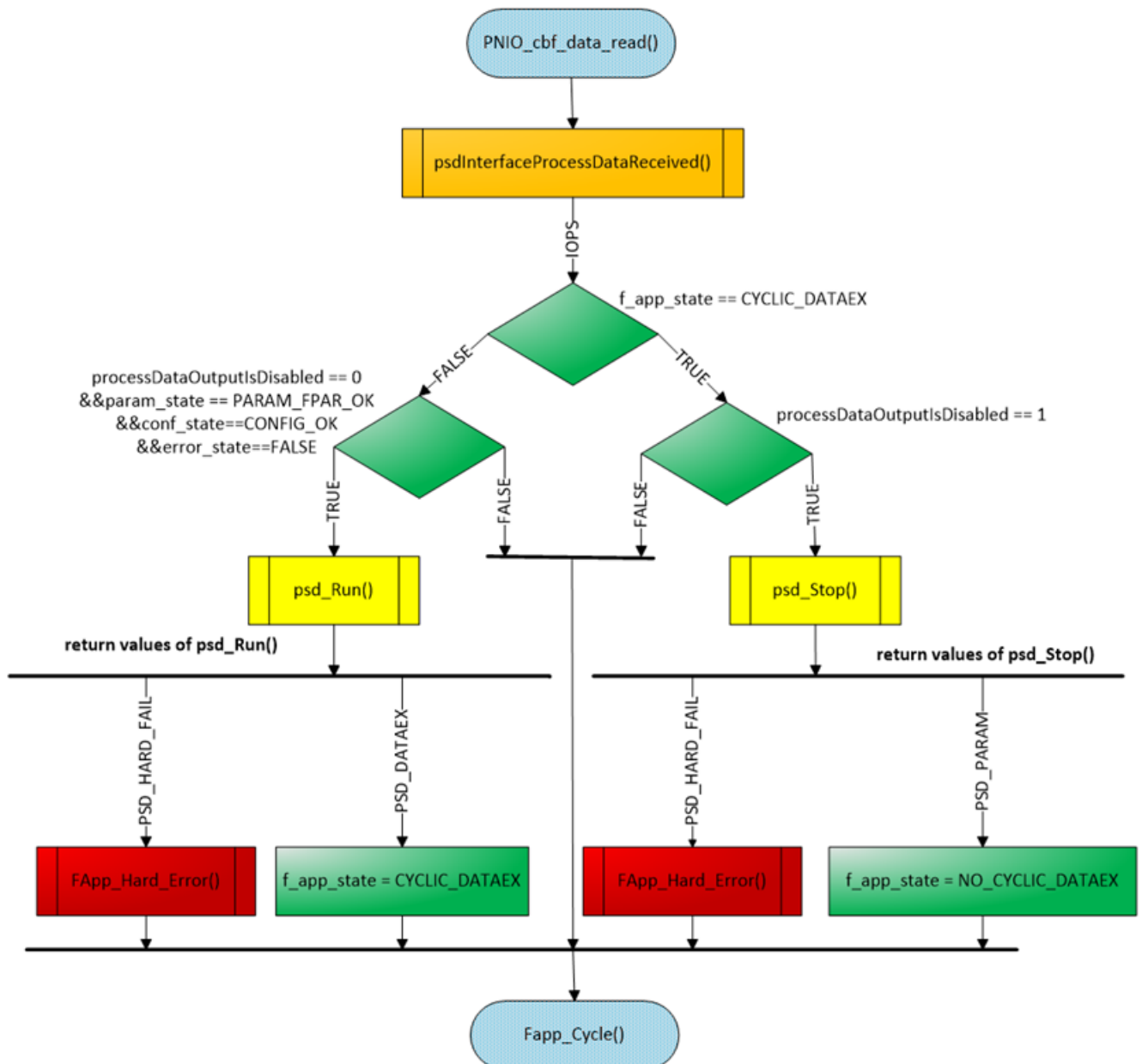


Figure 9-6 F-host status

Communication - psd_RecvFOutTele()

The time values are read in using the function `OsGetTime_ms()` and passed to the function `psd_RecvFOutTele()`.

The function `psd_RecvFOutTele()` expects two independently generated time values. Only one time value is generated for the application example.

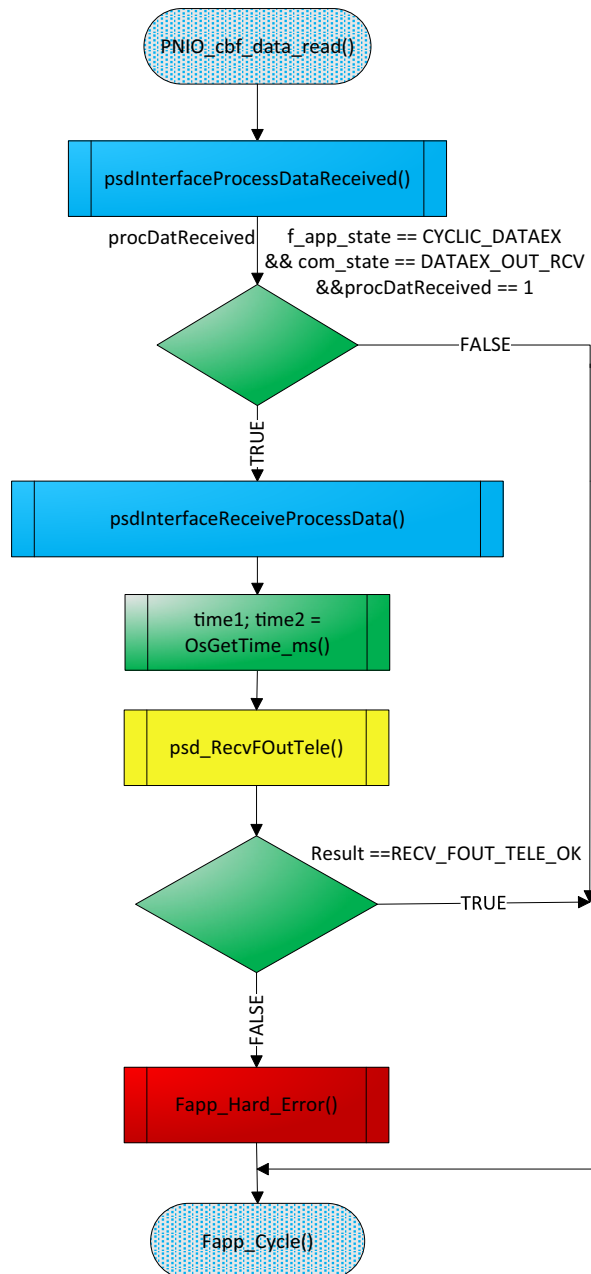


Figure 9-7 Communication - psd_RecvFOutTele()

Communication - psd_GetFOutData()

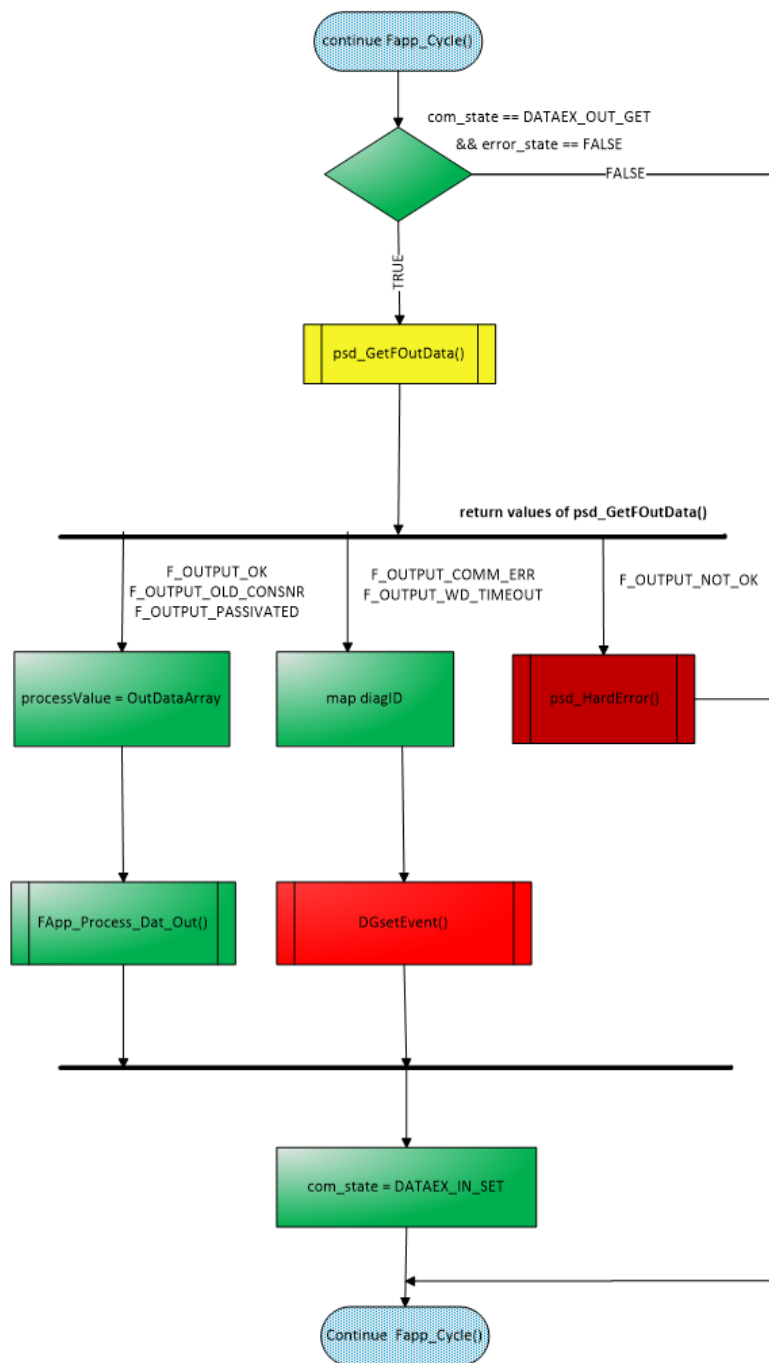


Figure 9-8 Communication - psd_GetFOutData()

The process data memory OutDataArray is passed to the function psd_GetFOutData(). Once the telegram has been verified in the function psd_GetFOutData(), the user data is activated or, if an error is detected, overwritten with zero.

9.4 PNIO_cbf_data_write()

Communication - psd_SetFinData()

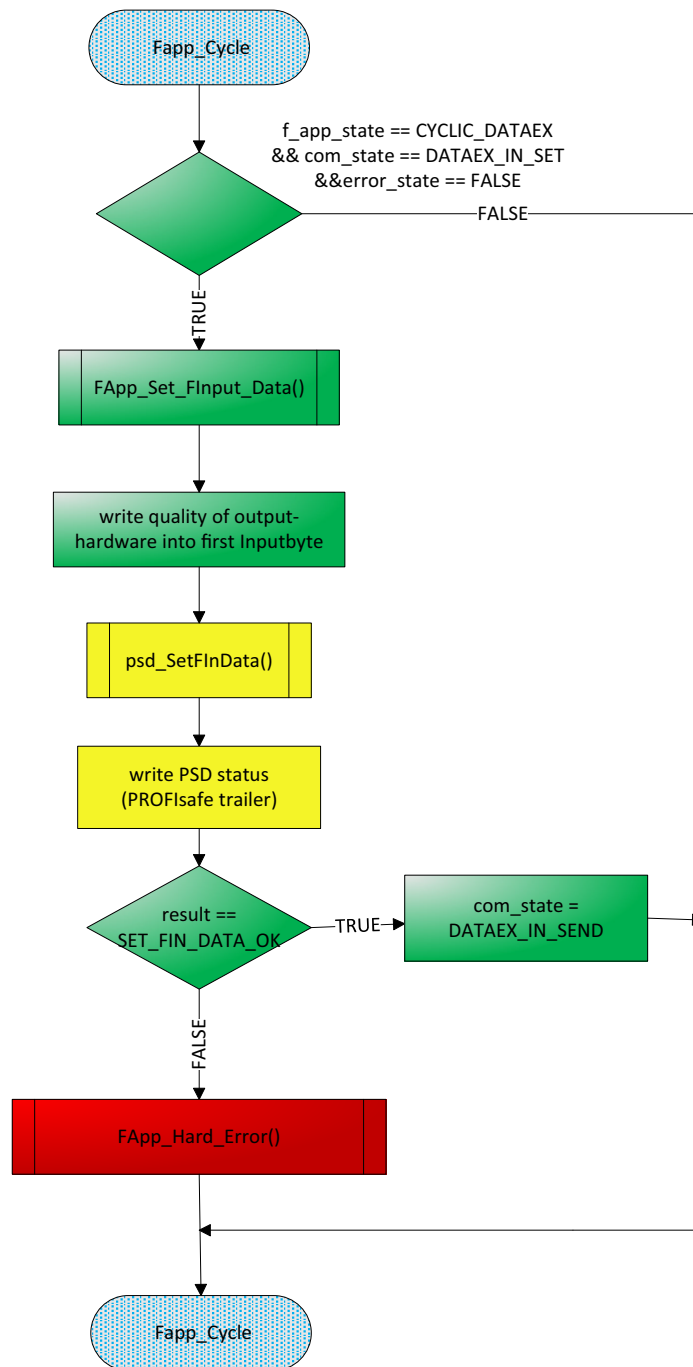


Figure 9-9 Communication - psd_SetFinData()

Communication - psd_SendFinTele()

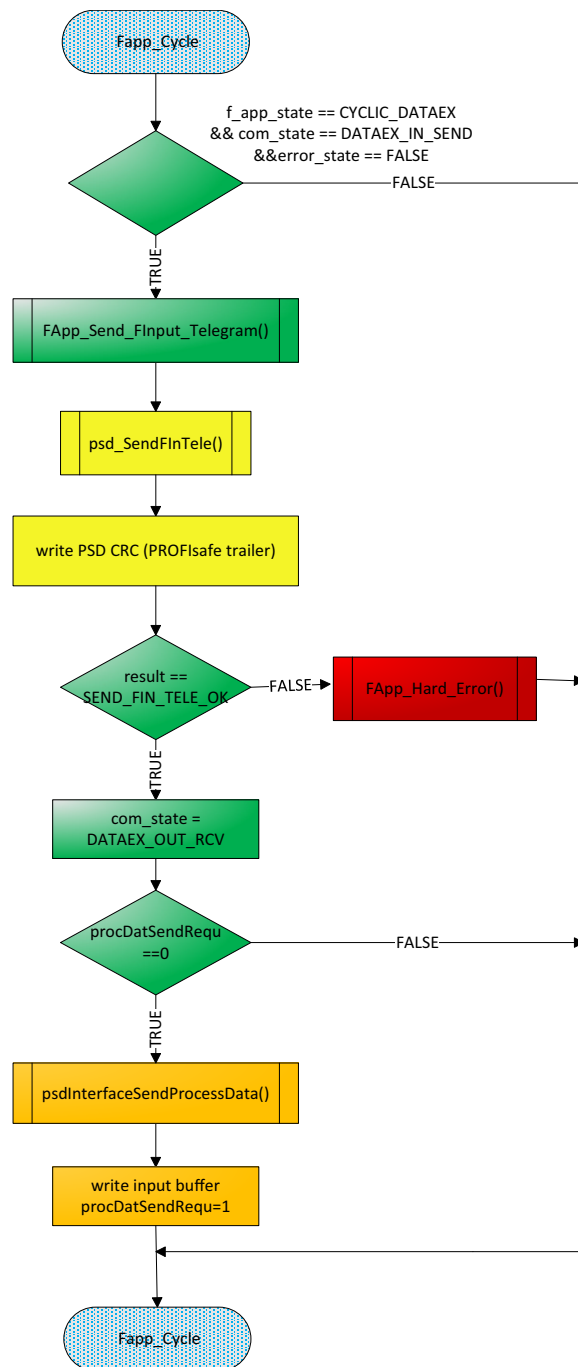


Figure 9-10 Communication - psd_SendFinTele()

Communication - PNIO_cbf_data_write()

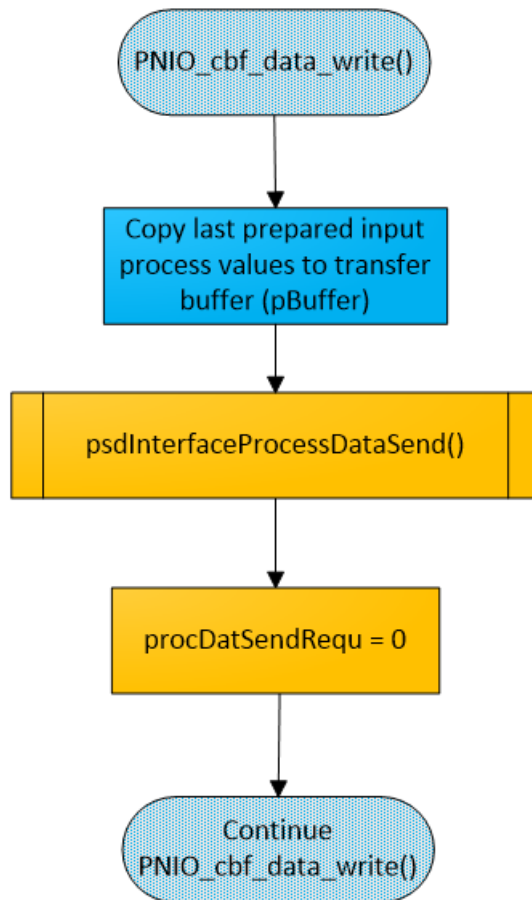


Figure 9-11 PNIO_cbf_data_write()

`PNIO_cbf_data_write` is called if an input telegram is sent from the DevKit to the F-host. Here, the process input data preassigned in the `p_fApp_PNIO_Send` buffer (status of F-outputs + PROFIsafe status + PROFIsafe crc) is copied to the output buffer and the event `procDatSendRequ` is reset.

Event `procDatSendRequ = 0` signals to the F-application to copy new process input data into the `p_fApp_PNIO_Send` buffer for the next `com_state == DATAEX_IN_SEND`.

9.5 PNIO_cbf_ar_connect_ind()

PNIO_cbf_ar_connect_ind()

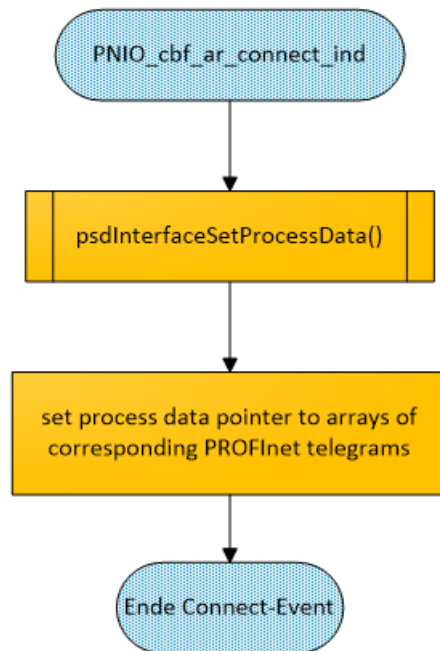


Figure 9-12 PNIO_cbf_ar_connect_ind()

This block is not shown in the general chart since it can occur at any time.

The function is called when the F-host is disconnected from the DevKit.

The pointers that point to the PROFINET telegrams are set to zero since no telegrams are received from that moment on.

9.6 DG_update()

Diagnostic buffer

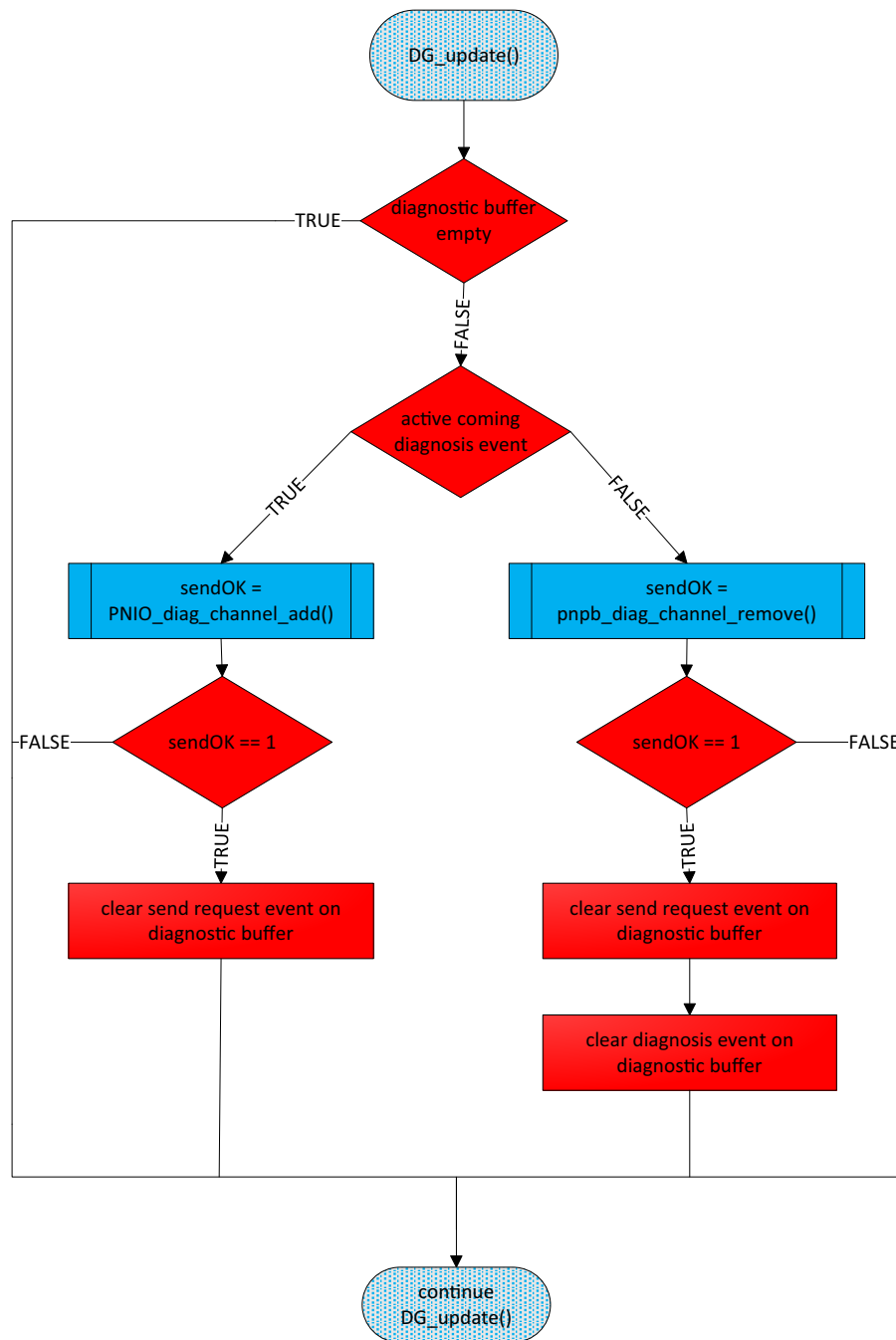


Figure 9-13 Diagnostic buffer

The diagram describes how the `DG_update()` task handles diagnosis events.

In the event of incoming diagnosis events, the diagnostic buffer is filled with incoming diagnosis events from the function "DG_setEvent()". "DG_clearEvent()" sets the state of the diagnosis event as outgoing.

The data structure of the diagnostic buffer "diagEvents_" has been defined in DiagnosticManager.h as follows:

```
typedef struct
{
    unsigned short ID;
    unsigned char sendToBus;
}DiagEvent;
```

Figure 9-14 DiagnosticManager

ID corresponds with the diagnosis event ID, sendToBus describes the state of the diagnosis event.

- sendToBus =0: Diagnosis event has already been sent to the bus or, if ID = 0, the diagnostic buffer is empty
- sendToBus =1: Incoming diagnosis event
- sendToBus =2: Outgoing diagnosis event

PROFIsafe functions

10.1 psd_InitInstance()

psd_InitInstance()	
Source file:	p_c_pseudo.c
Function:	Fapp_Init
Transfer parameters	
INST_01	Instance 1
DYNAMIC_TELE_LEN_CONF_ON	Dynamic telegram length configuration active.

Evaluation of return value:

If the return value is "INIT_INSTANCE_OK", the state "NO_CYCLIC_DATAEX" is assumed.

If the return value is "INIT_INSTANCE_NOT_OK", the function FApp_Hard_Error() is called.

10.2 psd_FParBuild()

psd_FParBuild()	
Source file:	p_c_pseudo.c
Function:	FApp_Param
Transfer parameters	
INST_01	Instance 1
fParameterData	"Firmware parameters" such as destination address, source address and SIL are stored in this variable.
fParameterDataLength	Length of data structure of F-parameter.
iParameterCrc16	I-parameters are not supported in this example.
iParameterCrc32	I-parameters are not supported in this example.

Evaluation of return value:

Based on the return value of psd_FParBuild() (error detected), a diagnosis event is sent to the thread DiagnoseHandling() with the function addDiagnose().

The return values are mapped onto the error numbers from the GSDML file and passed to the function addDiagnose().

If execution of the function is successful, the state "PARAM_FPAR_OK" is assumed.

10.3 psd_Config()

psd_Config()	
Source file:	p_c_pseudo.c
Function:	FApp_LenConfig
Transfer parameters	
INST_01	Instance 1

Evaluation of return value:

If an error is detected, the F-application changes to safe state with FApp_Hard_Error().

If the check is successful, the state "conf_state = CONFIG_OK" is assumed.

NOTE

For the length configuration, a length of 1 byte was entered for OUTDATA_MAX_LEN_INST_01 and INDATA_MAX_LEN_INST_01 in header file "p_c_config.h", since it was defined this way in the GSDML file of the DevKit. (see Length configuration ([Page 44](#)))

For the function psd_GetConfigPtr(), the telegram lengths were determined and passed. Here, the telegram has a length of 6 bytes (1 byte user data + 1 byte status byte + 4 bytes CRC). The telegram length was defined in section Interface for BusEvents ([Page 45](#)).

10.4 psd_Run()

psd_Run()	
Source file:	p_c_pseudo.c
Function:	FApp_Start_Communication
Transfer parameters	
INST_01	Instance 1

Evaluation of return value:

If the return value is "PSD_DATAEX", the state "f_app_state = CYCLIC_DATAEX" is assumed. As a result, the F-application changes to state PROFIsafe process data exchange.

10.6 psd_RecvFOutTele()

If an error is detected, the F-application changes to safe state with FApp_Hard_Error().

NOTE

After the DevKit has received a PROFINET frame, the IOPS status is read out. If this is 0x80 (F-host in Run) and param_state == PARAM_FPAR_OK, conf_state == CONFIG_OK and error_state == FALSE, the F-application changes to state CYCLIC_DATAEX (process data exchange).

10.5 psd_Stop()

psd_Stop()	
Source file:	p_c_pseudo.c
Function:	FApp_Start_Communication
Transfer parameters	
INST_01	Instance 1

Evaluation of return value:

Return value PSD_PARAM is not evaluated in the example.

The F-application changes to NO_CYCLIC_DATAEX state and stops the PROFIsafe process data exchange. PSD com_state changes to INITIALIZE state.

NOTE

The IOPS status is read out after every receipt of a PROFINET telegram by the DevKit. If IOPS is 0x60 (F-host in Stop), the event "processDataOutputsDisabled" is set. The event is evaluated multiple times in FApp_Cycle. When Event = 1 (active), psd_Stop is called using Fapp_Stop_Communication().

10.6 psd_RecvFOutTele()

psd_RecvFOutTele()	
Source file:	p_c_pseudo.c
Function:	FApp_Read_FOutput_Telegram
Transfer parameters	
INST_01	Instance 1
time1	Time stamp in ms
time2	Time stamp of simulated CPU 2 in ms

Evaluation of return value:

If the return value of function psd_RecvFOutTele() is not "RECV_FOUT_TELE_OK", the F-application changes to safe state with FApp_Hard_Error().

NOTE

The transfer parameters "time1" and "time2" should be generated by 2 independent timers. However, since only one F-device is being simulated here, both values are the same (see section "Embedding the PROFIsafe driver into the F-device ([Page 52](#))").

10.7 psd_GetFOutData()

psd_GetFOutData()	
Source file:	p_c_pseudo.c
Function:	FApp_Get_FOutput_Data
Transfer parameters	
INST_01	Instance 1
& FOutputDataCB	Buffer for PSD control bytes
& OutDataArray	Buffer for process output data
OUTDATA_MAX_LEN_INST_01	User data length without PROFIsafe trailer, here: 1 byte

Evaluation of return value:

Based on the return value of psd_GetFOutData (error detected), an "incoming" diagnosis event is sent using IDG_setEvent(event ID) if an error is detected.

The return values are mapped onto the error numbers from the GSDML file. If the check is error free, the state com_state = DATAEX_IN_SET is assumed (prepare process input data and status byte for F-host input telegram).

10.8 psd_SetFInData()

psd_SetFInData()	
Source file:	p_c_pseudo.c
Function:	FApp_Set_FInput_Data
Transfer parameters	
INST_01	Instance 1

10.9 psd_SendFInTele()

& FInputDataSB	Buffer for PSD status byte
& InDataArray	Buffer for process input data
INDATA_MAX_LEN_INST_01	User data length without PROFIsafe trailer, here: 1 byte

Evaluation of return value:

If the return value of function psd_SetFInData() is not SET_FIN_DATA_OK, the F-application changes to safe state with FApp_Hard_Error().

If no error is detected, the state com_state = DATAEX_IN_SEND is assumed (send process input data to the F-host).

10.9 psd_SendFInTele()

psd_SendFInTele()	
Source file:	p_c_pseudo.c
Function:	FApp_Send_FInput_Telegram
Transfer parameters	
INST_01	Instance 1

Evaluation of return value:

If the return value of function psd_SendFInTele() is not "SEND_FIN_TELE_OK", the F-application changes to safe state with FApp_Hard_Error().

If no error is detected, the state com_state = DATAEX_OUT_RCV is assumed (receipt of F-output data from the F-host).

Integration of PROFIsafe driver source files into the application example

11

If you want to debug the function of the PROFIsafe driver on the DevKit, you must use the original sources files of the PROFIsafe driver. Three source files ("p_c_pseudo.c", "p_c_fapplication.c" and "p_c_si.c") have been provided for this purpose on the CD of the PROFIsafe Starter Kit V3.5.1 (6ES7194-3BF03-0YA0) in folder "Example".

11.1 p_c_fapplication.c

psd_OutTransfer(), psd_OutSync(), psd_InTransfer(), psd_InSync()

The functions psd_OutTransfer(), psd_OutSync(), psd_InTransfer() and psd_InSync() simulate a data exchange between two processors with the static variable SyncValues. These functions are only called when the define "REDUNDANT" has been set.

In this application example, the two processors are emulated on one processor, and the synchronization functions are therefore executed without errors.

disable_int()

No changes were made here since the PROFIsafe driver can also operate without an interrupt disable in this application.

enable_int()

No changes were made here since the PROFIsafe driver can also operate without an interrupt disable in this application.

11.2 p_c_si.c

psd_SetFParPtr()

This function copies the F-parameters received from the F-host into the static parameter buffer prm_fpar defined in the same file.

psd_GetFParPtr()

This function returns a pointer that points to the F-parameters.
This has been stored in a global variable "prm_fpar" in the same file.

11.3 p_c_pseudo.c

psd_GetConfigPtr()

This function returns a pointer to the PROFIsafe telegram length.
This has been stored in a global variable "io_range" in the same file.

psd_GetInputBuffer()

This function returns the pointer to the array with the cyclic process input data.
This has been stored in a global variable "InDataArray" in "p_c_pseudo.c".

psd_InputSendAck()

The return value is always "FIN_TELE_UPDATE_OK".

psd_GetOutputTelegram()

This function returns the pointer to the array with the cyclic process output data.
This has been stored in a global variable "OutDataArray" in "p_c_pseudo.c".

11.3 p_c_pseudo.c

calcFParCRC()

This function calculates a 16-bit CRC for the console output following a CRC error.
Terminal programs such as Tera Term or similar tools can be used for the console output.

Fapp_Process_Dat_Out()

This function switches process output data to the output LEDs of the DevKit.

Fapp_New_FPar_available()

This function evaluates the events "hasNewParameter" and "hasNewFParameter" and returns "TRUE" when an event is active. The events are reset at the same time.

Fapp_Get_Communication_State()

This function evaluates the event processDataOutputsDisabled and returns "STOP" when the event is active. When Event = 0, the return value is "RUN".

Fapp_Hard_Error()

This function calls all error handlers (stop PROFIsafe communication, stop process data output, switch fail-safe values, console output of line number and FileID) and sets the errorState event.

main()

The main() function is replaced by the MainAppl() function in this implementation example. The definition is made in usriod_main.c.

Fapp_Cycle()

The state machine of the PSD has been adapted to the firmware of the DevKit (see State machine [\(Page 48\)](#)).

Fapp_Init()

This function initializes all application-specific data of the PSD during Power On and has been adapted to the firmware of the DevKit. The initialization of interface events and specific F-parameters, such as source address, destination address and SIL, has been added.

If an error occurs during initialization of the PSD instance, the F-application goes to safe state with Fapp_Hard_Error().

If initialization is completed without errors, f_app_state changes to NO_CYCLIC_DATAEX (wait for parameters).

(see flow diagrams Startup of configuration/parameter assignment [\(Page 54\)](#) and PnUsr_DeviceSetup() [\(Page 61\)](#))

Fapp_Param()

This function is called when interface event hasNewFParameter is active in f_app_state NO_CYCLIC_DATAEX; it checks the F-parameters received from the bus.

Depending on the return value of the called FParBuild(), an "incoming" diagnosis event is sent using DG_setEvent(event ID) if an error is detected. Event IDs are mapped onto error numbers of the GSDML. If the check is error-free, then param_state = PARAM_FPAR_OK.

(see flow diagrams Startup of configuration/parameter assignment [\(Page 54\)](#) and Errors during parameter assignment [\(Page 57-58\)](#)).

Fapp_Read_FOutput_Telegram()

This function checks the output telegram received from the F-host. For the time stamps time1 and time2 (timeout check), the same timers are read by the operating system of the DevKit.

(see diagrams Cyclic data exchange Cyclic data exchange [\(Page 55\)](#), Telegram error in state "CYCLIC_DATAEX; DATAEX_OUT_RCV" Telegram error in state "CYCLIC_DATAEX;

DATAEX_OUT_RCV" [\(Page 58\)](#) and Communication - psd_RecvFOutTele()

PNIO_cbf_data_read() [\(Page 64\)](#) and the description of the psd_RecvFOutTele() function psd_RecvFOutTele() [\(Page 76\)](#))

Fapp_GetFoutData()

This function gets the checked output data as well as bit information of the control byte for the process data output.

Depending on the return value of the called `psd_GetFOutData()`, an "incoming" diagnosis event is sent using `DG_setEvent(event ID)` if an error is detected. Event IDs are mapped onto error numbers of the GSDML. If the check is error free, then `com_state = DATAEX_IN_SET` (prepare process input data and status bytes for F-host input telegram).

See diagrams Cyclic data exchange Cyclic data exchange (Page 55), Telegram error in state "CYCLIC_DATAEX; DATAEX_OUT_RCV" Telegram error in state "CYCLIC_DATAEX; DATAEX_OUT_RCV" (Page 58) and Communication - `psd_GetFOutData()`

`PNIO_cbf_data_read()` (Page 64) and the description of the `psd_GetFOutData()` function `psd_GetFOutData()` (Page 77).

Fapp_Set_FInput_Data()

This function assigns process input data and the PSD status byte for the F-host with function call `psd_SetFInData()`. If the return value of function `psd_SetFInData()` is not `SET_FIN_DATA_OK`, the F-application changes to safe state with `FApp_Hard_Error()`.

If no error is detected, then state `com_state = DATAEX_IN_SEND` (send process input data to the F-host).

See diagrams Cyclic data exchange Cyclic data exchange (Page 55) and Communication – `psd_SetFInData()` `PNIO_cbf_data_write()` (Page 67) and the description of the `psd_SetFInData()` function `psd_SetFInData()` (Page 77)

Fapp_Send_FInput_Telegram()

This function prepares the F-input telegram for the F-host with function `psd_SendFInTele()`. If the return value of function `psd_SendFInTele()` is not `SEND_FIN_TELE_OK`, the F-application changes to safe state with `FApp_Hard_Error()`.

If no error is detected, then state `com_state = DATAEX_OUT_RCV` (receive F-output data from the F-host).

See diagrams Cyclic data exchange Cyclic data exchange (Page 55) and Communication – `psd_SetFInTele()` `PNIO_cbf_data_write()` (Page 67) and the description of the `psd_SetFInTele()` function `psd_SendFInTele()` (Page 78)

11.4 DiagnosticManager.c

DG_setEvent()

This function enters diagnosis event numbers in the diagnostic buffer as "incoming".

The error number that was defined together with its diagnostics text in the GSDML is needed as transfer parameter.

DG_clearEvent()

This function removes diagnosis event numbers from the diagnostic buffer. "Outgoing diagnosis event".

The error number that was defined together with its diagnostics text in the GSDML is needed as transfer parameter.

DG_update()

This function is called cyclically as a task; it processes all diagnostics entered in the buffer. See diagram for DG_update() task for handling diagnosis events DG_update() [\(Page 71\)](#)

11.5 Psd_interface.c

All functions implemented in the file "Psd_interface.c" serve as an interface between the ERTEC 200P DevKit software and PROFIsafe F-application. The DevKit and F-application cycles run asynchronously. Therefore, the program flow is controlled using events.

psdInterfaceInit()

This function, which is called from DevKit PnUser_DeviceSetup() during Power On, initializes the PROFIsafe driver, diagnostics manager and F-application. Return value is PNIO_OK or, if an error is detected, PNIO_NOT_OK.

See diagram Startup of configuration/parameter assignment [\(Page 54\)](#).

psdInterfaceSetLED()

This function, which is called in F-application Fapp_Process_Dat_Out(), switches process output data (or 0, if an error is detected) to the LEDs of the DevKit.

psdInterfaceF_App_Cycle()

This function, which is called from DevKit MainAppl(), implements the F-application. It is processed cyclically (see State machine [\(Page 48\)](#))

psdInterfaceReceiveRecord()

This function is called from DevKit PNIO_cbf_rec_write() when a data record is received. Depending on the data record number, control events (hasNewFParameter, faultInsertionTest etc.) are set for the F-application and the data is copied to the relevant buffer of the F-application. (see diagram PNIO_cbf_rec_write()). Return value is PNIO_OK or, if an error is detected, PNIO_NOT_OK.

psdInterfaceSendRecord()

This function is called from DevKit PNIO_cbf_rec_read() following a read data record request (F-parameter, Trace, fault insertion test). Depending on the data record number, the data of the F-application is copied to the transfer buffer of the DevKit. Return value is PNIO_OK or, if an error is detected, PNIO_NOT_OK.

psdInterfaceSetProcessData()

This function is called from DevKit PNIO_cbf_ar_connect_ind() during initialization after Power On; it sets the process data pointer of the F-application to the corresponding buffer of the DevKit.

See diagram PNIO_cbf_ar_connect_ind() [\(Page 70\)](#).

psdInterfaceProcessDataReceived()

This function, which is called from DevKit PNIO_cbf_data_read() upon receipt of process data, sets the event procDatReceived and, depending on IOPS, the event processDataOutputsDisabled.

See diagrams "F-Host Status" and PNIO_cbf_data_read() [\(Page 64\)](#) "Communication – psd_RecvFOutTele()" PNIO_cbf_data_read() [\(Page 64\)](#)

psdInterfaceReceiveProcessData()

This function, which is called from F-application Fapp_Cycle() following event procDatReceived, copies process output data from the buffer of the DevKit to the process data buffer of the F-application.

See diagram "Communication – psd_RecvFOutTele()" PNIO_cbf_data_read() [\(Page 64\)](#)

psdInterfaceSetProcessData()

This function, which is called from F-application Fapp_Cycle(), copies process input data from the process data buffer of the F-application to the transfer buffer of the DevKit and sets the event procDatSendRequ.

See diagram "Communication – psd_SendFInTele()" PNIO_cbf_data_write() [\(Page 67\)](#)

psdInterfaceProcessDataSend()

This function, which is called from DevKit PNIO_cbf_data_write, resets the event procDatSendRequ after successful sending of the process input data to the F-host, thereby releasing the DevKit transfer buffer for new process input data of the F-application.

See diagram "Communication – PNIO_cbf_data_write()" PNIO_cbf_data_write() [\(Page 67\)](#)

11.6 DevKit PnUsr_Api.c

PnUsr_DeviceSetup()

The initialization function psdInterfaceInit() of the F-application interface is called here.

PNIO_cbf_rec_read()

This function is extended to record indices 240 (Fatal Error Report) and 201 (PSD Trace Report); it calls the interface function psdInterfaceSendRecord().

```
// *-----*
// * handle special indices for I&M
// *-----*
switch (RecordIndex)
{
    case 0x00F0: //record 240 for version request, fault insertion tests and fatal error report
        Status = psdInterfaceSendRecord(RecordIndex,pBuffer,pBufLen);
        break;
    case 0x00C9: //record 201 PSD trace report
        Status = psdInterfaceSendRecord(RecordIndex,pBuffer,pBufLen);
        break;
```

Figure 11-1 PNIO_cbf_rec_read()

PNIO_cbf_rec_write()

This function is extended to record indices 128, 129 and 240; it calls the interface function psdInterfaceReceiveRecord().

```
// *----- IM 0 ... IM 4-----*
switch (RecordIndex)
{
    case 0x80a0:
        Status = PROFIenergy_RequestHandler (pAddr, pBufLen, pBuffer, pPnioState, ArNum);
        break;

    case 0x0080: //record 128 for F-parameters
    case 0x0081: //record 129 for I-parameters
    case 0x00F0: //record 240 for version request and fault insertion tests
        Status = psdInterfaceReceiveRecord(RecordIndex,pBuffer,pBufLen);
        break;
```

Figure 11-2 PNIO_cbf_rec_write()

11.7 DevKit iodapi_event.c

PNIO_cbf_data_write()

Call of the interface function psdInterfaceProcessDataSend(). See description of interface functionPsd_interface.c ([Page 83](#)).

11.8 DevKit usriod_main.c

PNIO_cbf_data_read()

Call of the interface function `psdInterfaceProcessDataReceived()`. See description of interface `functionPsd_interface.c` ([Page 83](#)).

PNIO_cbf_ar_connect_ind()

Call of the interface function `psdInterfaceSetProcessData()`. See description of interface `functionPsd_interface.c` ([Page 83](#)).

11.8 DevKit usriod_main.c

MainAppl()

Call of the interface function `psdInterfaceF_App_Cycle()`. See description of interface `functionPsd_interface.c` ([Page 83](#)).

Mapping the error numbers

Since the return values of `psd_FParBuild()` and `psd_GetFOutData()` do not correspond with the error numbers defined in the GSDML together with their error texts, the return values are mapped onto the PROFIsafe-specific error numbers.

The diagnostics texts have been standardized and are contained in PROFIsafe profile "PROFIsafe-Profile_3192_V261_Aug14.pdf", page 39.






The table below presents an overview of the transformed error numbers:

Hex	Number	PSD return values Hex	Description	psd_FParBuild()	psd_GetFOutData()
0x40	64	0x4A	Mismatch of safety destination address (F_Dest_Add)	X	
0x41	65	0x41	Safety destination address not valid (F_Dest_Add)	X	
0x42	66	0x54 / 0x2C	Safety source address not valid or mismatch (F_Source_Add)	X	
0x43	67	0x79	Safety watchdog time value is 0 ms (F_WD_Time, F_WD_Time_2)	X	
0x44	68	0x86	Parameter "F_SIL" exceeds SIL from specific device application	X	
0x45	69	0x98	Parameter "F_CRC_Length" does not match the generated values	X	
0x46	70	0xAB	Version of F-Parameter set incorrect	X	
0x47	71	0xB5	Data inconsistent in received F-Parameter block (CRC1 error)	X	
0x48	72	0xCD / 0x72	Device specific or unspecified diagnosis information, see manual		X
0x4B	75	0xE0	Inconsistent iParameters (iParCRC error)	X	
0x4C	76	0x1F	F_Block_ID not supported	X	
0x4D	77	0x67	Transmission error: data inconsistent (CRC2 error)		X
0x4E	78	0x79	Transmission error: timeout (F_WD_Time or F_WD_Time_2 elapsed)		X

The right side of the table indicates which function sends which diagnoses.

LED status of the DevKit

Various LEDs are controlled in this implementation to enable you to better identify an error source.

LEDs			Meaning	Remedy
SYNC	ERROR	MAINT		
 Flashes	On	 Flashes	Hard error situation	Power Off/ Power On
Off	 Flashes	Off	Incoming diagnosis event active	Evaluate diagnostics and handle the error
 Flashes	Off	Off	No parameter assignment	In case of sustained flashing, check the parameter assignment in the TIA Portal.
 On	Off	Off	Cyclic data exchange	---

User LED output

LEDs are located above port X50 (Figure 18) of the DevKit. These are representative of outputs that are switched.

In this application, only the first 4 LEDs or, in non-redundant operating mode, the last 4 LEDs are switched. Which LED is switched depends on the first 4 bits of the process data.

The left-most LED is the LSB and the fourth LED from the left is the MSB.

In non-redundant operating mode, the last 4 LEDs are additionally switched starting from the right inversely to the left 4 LEDs.

blue	Status LED (connector for GPIOs connected)
red	LEDs for output of user data (left LSB, right MSB)
green	LEDs for output of user data (in non-redundant operating mode)

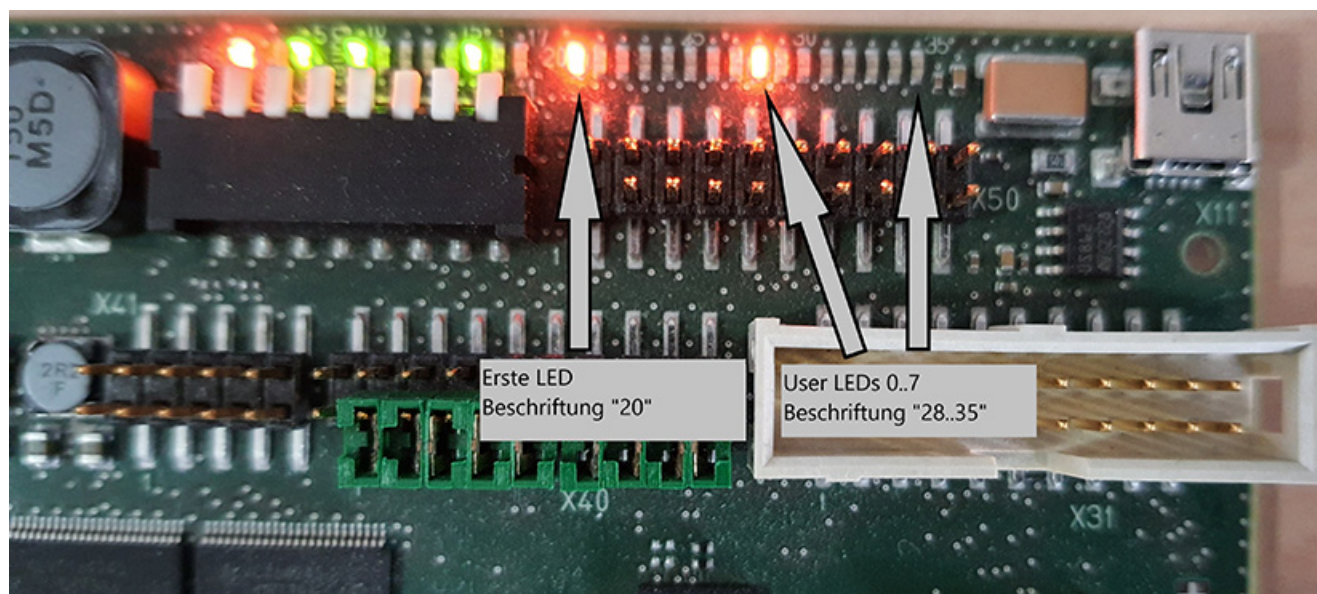


Figure 14-1 User LEDs of the DevKit

NOTE

The first LED from the left is always lit because it merely indicates whether the connector for user GPIOs has been plugged in.

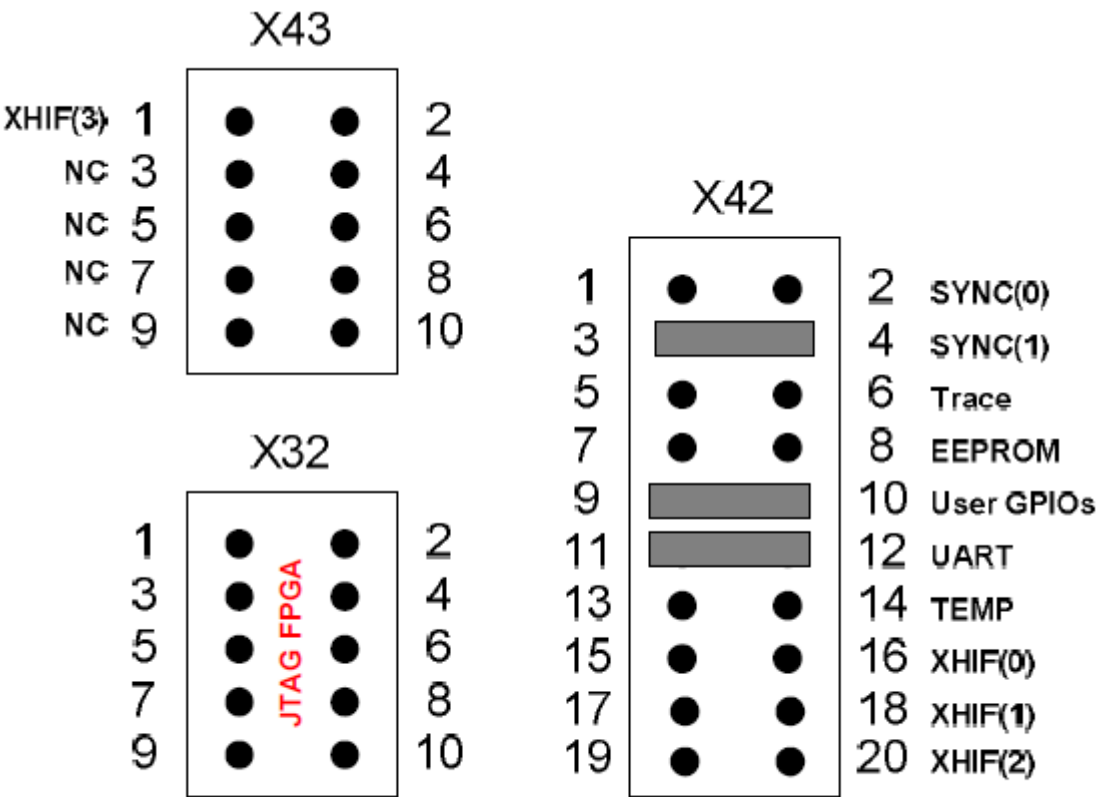


Figure 7 EB 200P default configuration of onboard circuits

Figure 14-2 Connector for user GPIOs

The connector for user GPIOs must have been plugged in (Port X42, Pin 9, 10) so that the GPIOs can be controlled by the application.

References

1.

PDF file:	Component of
PROFIsafe-Profile_3192_V26MU1_Aug18.pdf	Part of the PROFIsafe Starter Kit V3.5.2
profisafe_driver_v2_2_3_programming_manual_en-US.pdf	Part of the PROFIsafe Starter Kit V3.5.2
Interface_Description_PN_IO_DevKits_V4.7.0	Part of EvalKit_ERTEC200P
Guideline_EvalKit_ERTEC200P_V4.7.0.pdf	Part of EvalKit_ERTEC200P
Manual_ERTEC200P-2_V1_0.pdf	Part of EvalKit_ERTEC200P
Howto use J-link JTAG Debugger_on_EB200P_V1_1.pdf	Part of the application example
GSDML_GettingStarted_V1_5.pdf	Part of the application example
GSDML-Spec_2352_V243_May22.pdf	Current version available on profibus.com

Glossary

CRC

Cyclic Redundancy Check

DevKit

Evaluation Kit EK-ERTEC200P PN IO V4.3

Embedded Software

Colloquial term for firmware.
Hardware-level software that is installed on the F-device.

F-device

Passive communication partner that is able to execute the PROFIsafe protocol and that is prompted by the F-host to exchange data.

F-host

Data processing unit that is able to execute the PROFIsafe protocol and that prompts the F-device to exchange data.

FIT

Fault Insertion Test

F-parameter

PROFIsafe parameter

FW

Firmware, also "embedded software"

GPIO

General Purpose Input Output
Pins that can be used as output or input.

GSDML

General Station Description Markup Language

HW

Hardware

PROFIsafe trailer

Composed of status/control byte and the security information.

PSD

PROFIsafe driver for F-device

SIL

Safety Integrity Level

WD

Watchdog for runtime monitoring