PROFIdrive Application example AC4

Application example

https://support.industry.siemens.com/cs/ww/en/view/109757402

Siemens Industry Online Support

Warranty and liability

Note

The Application Examples are not binding and do not claim to be complete regarding the circuits shown, equipping and any eventuality. The Application Examples do not represent customer-specific solutions. They are only intended to provide support for typical applications. You are responsible for ensuring that the described products are used correctly. These Application Examples do not relieve you of the responsibility to use safe practices in application, installation, operation and maintenance. When using these Application Examples, you recognize that we cannot be made liable for any damage/claims beyond the liability clause described. We reserve the right to make changes to these Application Examples at any time without prior notice.
If there are any deviations between the recommendations provided in these Application Examples and other Siemens publications − e.g. Catalogs − the contents of the other documents have priority.

We do not accept any liability for the information contained in this document.
Any claims against us – based on whatever legal reason – resulting from the use of the examples, information, programs, engineering and performance data etc., described in this Application Example shall be excluded. Such an exclusion shall not apply in the case of mandatory liability, e.g. under the German Product Liability Act ("Produkthaftungsgesetz"), in case of intent, gross negligence, or injury of life, body or health, guarantee for the quality of a product, fraudulent concealment of a deficiency or breach of a condition which goes to the root of the contract ("wesentliche Vertragspflichten"). The damages for a breach of a substantial contractual obligation are, however, limited to the foreseeable damage, typical for the type of contract, except in the event of intent or gross negligence or injury to life, body or health. The above provisions do not imply a change of the burden of proof to your detriment.
Any form of duplication or distribution of these Application Examples or excerpts hereof is prohibited without the expressed consent of the Siemens AG.

Security information

Siemens provides products and solutions with industrial security functions that support the secure operation of plants, systems, machines and networks.
In order to protect plants, systems, machines and networks against cyber threats, it is necessary to implement – and continuously maintain – a holistic, state-of-the-art industrial security concept. Siemens' products and solutions only form one element of such a concept.
Customer is responsible to prevent unauthorized access to its plants, systems, machines and networks. Systems, machines and components should only be connected to the enterprise network or the internet if and to the extent necessary and with appropriate security measures (e.g. use of firewalls and network segmentation) in place.
Additionally, Siemens' guidance on appropriate security measures should be taken into account. For more information about industrial security, please visithttps://www.siemens.com/industrialsecurity.

Siemens' products and solutions undergo continuous development to make them more secure. Siemens strongly recommends to apply product updates as soon as available and to always use the latest product versions. Use of product versions that are no longer supported, and failure to apply latest updates may increase customer's exposure to cyber threats.
To stay informed about product updates, subscribe to the Siemens Industrial Security RSS Feed under https://www.siemens.com/industrialsecurity.

Table of contents

1 Preface

1.1 Purpose of the manual

The user documentation describes the software functionality of the PROFIdrive application example of Application Class 4 (AC4) based on the Evaluation Kit, EK-ERTEC 200P-2 PN IO V4.5.0. It is based on the Programming and Operating Manual "Interface description PROFINET IO Development Kits V4.5.0" /2/ of the Evaluation Kit. The PROFIdrive application example is realized in accordance with PROFIdrive V4.2/1/.

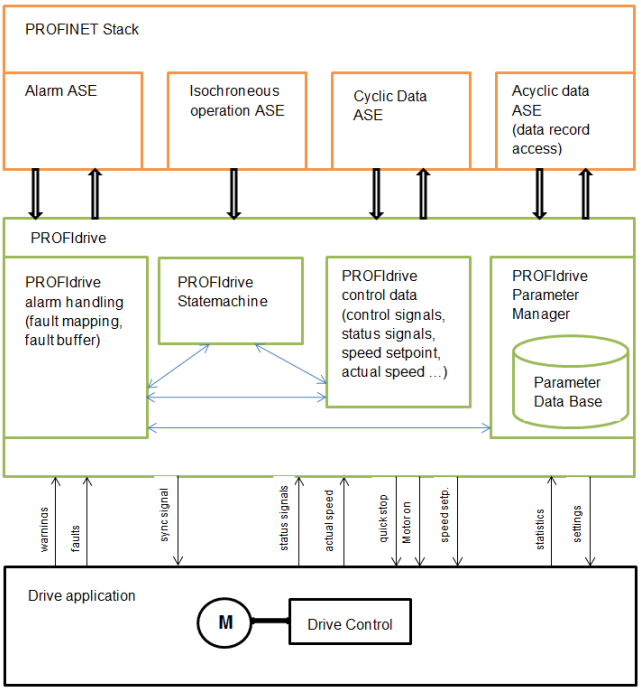
1.2 Target group for the manual

This manual is intended for software developers and application developers who want to use the Evaluation Kit for new products with the PROFIdrive profile or to port the example implementation of the PROFIdrive profile to a different PROFINET hardware platform. This manual applies to ERTEC 200P-based platforms. A good understanding of PROFINET IO and PROFIdrive are required in order to be able to use the PROFINET firmware stack and the PROFIdrive application example.

2 Introduction

The PROFIdrive profile is the tried-and-tested device profile for drive units based on PROFIBUS and PROFINET. With the application example presented here, you can shorten the development time for the realization of a PROFIdrive connection of a drive unit. The application example is based on the PROFINET stack for the EK ERTEC 200P-2 Evaluation Kit. It can also serve as the basis for porting to other platforms and communication stacks. Since the PI ENCODER profile bases on the PROFIdrive profile, parts of this application example can also be used for the implementation of an encoder device in accordance with the ENCODER profile (for example Parameter Channel, Parameter Set, Encoder Interface, Diagnosis, Fault Buffer)

PROFIdrive is a standardized application interface for drives and encoders and is defined in IEC61800-7 Parts 203, 303, as well as Chinese Recommendatory National Standard GB/T 25740.



Overview: PROFIdrive (green) as the application layer between the PROFINET stack (orange) and the drive application (black) from drive perspective

The PROFINET Stack is the firmware version supplied with the Evaluation Kit. PROFIdrive is implemented by the application example presented here. The "Drive application" is simulated in the application example by a simple simulation (PT1 element) and a motor encoder function.

The functionality of the PROFIdrive application example includes:

- Implementation of a drive object (a P-device consisting of exactly one Drive Unit and exactly one Drive Object) of the application class (AC) 4 with speed setpoint interface.
- Cyclic data exchange via standard frame 1 (control word, 16-bit speed setpoint, status word, 16-bit actual speed value), standard frame 2 (control word, 32 bit actual speed value, status word, 32 bit actual speed value, sign of life) and standard frame 3 (control word, 32-bit speed setpoint, status word, 32-bit actual speed value, sign of life, Encoder1 control word, Encoder1 status word, encoder value XIST1, encoder value XIST2)
- Acyclic data exchange through PROFIdrive parameter channel via BMP protocol (parameter access)
- PROFIdrive Diagnostics
- PROFIdrive General State Machine (drive state machine)
- Simulation of a drive control system (simulation)
- PROFIdrive Encoder channel mit Encoder State Machine (sensor interface)

A GSDML file exists for the PROFIdrive application example. The GSDML file is based on the GSDML file of the Evaluation Kit supplemented with entries for the PROFIdrive application example (PROFIdrive API).

The application example was tested with the PROFIdrive PROFINET Profile Tester V5.0. The PROFIdrive PROFINET Profile Tester is the test tool that the PI test labs use for PROFIdrive certification. The profile tester can also be used for the accompanying test in the development of PROFIdrive profile drives.

Link to the PROFIdrive PROFINET Profile Tester:

https://www.profibus.com/download/profidrive-profinet-profile-tester/

The PROFIdrive Implementation Guide can serve as a planning aid for a PROFIdrive implementation.

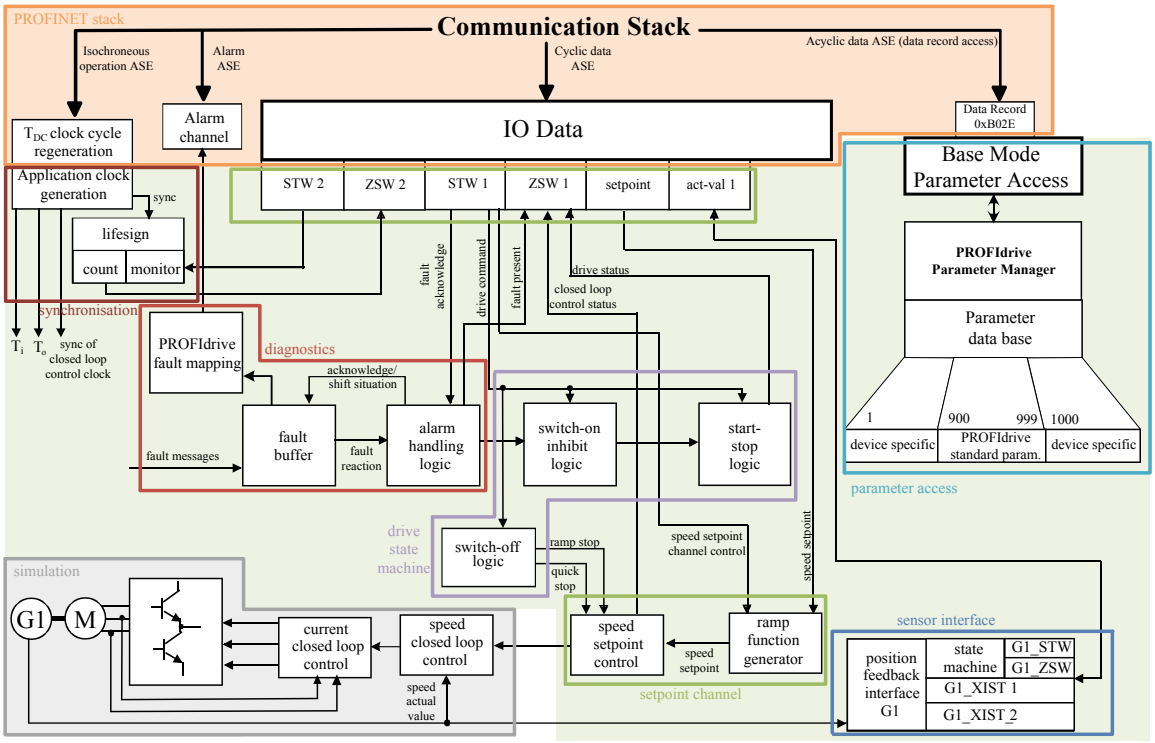
https://kb.hilscher.com/display/PROFIDRIVE/PROFIdrive+Implementation+Guide

The PROFIdrive specification can be downloaded here:

https://www.profibus.com/download/profidrive-profile-drive-technology/

3 PROFIdrive model of the application example

The following text describes the realized application example of the Application Example 4 in more detail.



Overview screen of PROFIdrive Application Class AC4 "Time-synchronized, speed regulated drive" Example from /1/ Figure 136

3.1 PROFINET device model according to PROFIdrive (see /1/ Section 8.4)

The general PROFINET device model breaks down a device into modules and submodules in the respective slots and subslots. Slots and subslots should be considered placeholders, meaning modules can be plugged into free slots or submodules into free subslots. The structure is generic and hierarchical, meaning a device contains one or more slots, a module contains one or more subslots.

Slot 0 is the Device Access Point (DAP) and describes the bus interface with its properties. The actual device functionality (for example inputs/outputs/drive axes) is modeled starting with Slot 1 in subslots.

Logical structure of the PROFIdrive P Device in the application example:

<table>
<tr><td colspan="2">Slot 0 (API = 0)</td><td colspan="3">Slot 1 (API = 0x3A00 PROFIdrive)</td></tr>
<tr><td>Subslot 0</td><td></td><td>Subslot 0</td><td>Subslot 1</td><td>Subslot 2</td></tr>
<tr><td rowspan="2"></td><td rowspan="2"></td><td rowspan="2"></td><td>Modules Access Point (MAP)</td><td>Standard Frame 1/2/3 (submodule ID = PROFIdrive frame number)</td></tr>
<tr><td>Contains parameters Access Point and alarm channel</td><td></td></tr>
<tr><td colspan="2">P-Device</td><td colspan="3">Drive Object 1</td></tr>
</table>

3.2 Cyclic data (IO AR)

Standard frames 1, 2 and 3 are implemented according to PROFIdrive standard /1/.

Standard frame 1 to /1/ Section 6.3.4.3.2

IO Data number (word number)	Setpoint	Actual value
1	STW1	ZSW1
2	NSOLL_A	NIST_A

Standard frame 1 has a size of 4 octets (bytes). STW1, NSOLL_A, ZSW1, NIST_A are 16-bit-data (Word). Input Data and Output Data are each 4 bytes over the same subslot 2.

STW1 is implemented according to "Speed control mode" /1/ Section 6.3.2.2. The optional function Jog1/Jog2 is not implemented. No device-specific control bits are implemented.

ZSW1 is implemented in "Speed control mode" as in /1/ Section 6.3.2.5. No device-specific status bits are implemented.

NSOLL_A and NIST_A have the data type N2 (0x4000 ≙ 100%, see /1/ Section 5.3.2). NIST_A returns the speed of the simulated control system (PT1 element).

Standard frame 2 from /1/ Section 6.3.4.3.3

IO Data number (word number)	Setpoint	Actual value
1	STW1	ZSW1
2	NSOLL_B	NIST_B
3		
4	STW2	ZSW2

The standard frame 2 has a size of 8 octets (bytes). STW1, STW2, ZSW1, ZSW2 are 16-bit data. NSOLL_B, NIST_B are 32-bit data. Input Data and Output Data are each 8 bytes over the same subslot 2.

STW1 and ZSW1 are implemented as in standard frame 1.

NSOLL_B and NIST_B have the data type N4 (0x40000000 ≙ 100%, see /1/ Section 5.3.2). NIST_B returns the speed of the simulated control system (PT1 element).

STW2 and ZSW2 include the signs of life Sign-Of-Life (see /1/ Section 6.3.12) and are implemented without manufacturer-specific data.

Standard frame 3 from /1/ Section 6.3.4.3.4

IO Data number (word number)	Setpoint	Actual value
1	STW1	ZSW1
2	NSOLL_B	NIST_B
3		
4	STW2	ZSW2
5	G1_STW	G1_ZSW
6		G1_XIST1
7		
8		G1_XIST2
9		

Standard frame 3 has a size of 10 octets (bytes) input und 18 octets (bytes) output. STW1, STW2, G1_STW, ZSW1, ZSW2 and G1_ZSW are each 16 bit files. NSOLL_B, NIST_B, G1_XIST1 and G1_XIST2 are each 32-bit files. Input Data are 10 bytes and Output Data are 18 bytes over the same subslot 2.

STW1 and ZSW1 are implemented as in standard frame 1.

NSOLL_B, NIST_B, STW2 and ZSW2 are implemented as in standard frame 2.

G1_STW is realized as in /1/ Table 108 and G1_ZSW as in /1/ Table 109. The simulated encoder currently only supports the functions Position, Park and Error acknowledgement (no referencing, no zero position adjustment, no probes),

G1_XIST1 returns the current position of the simulated encoder (see /1/ Section 6.3.6).

G1_XIST2 transfers the absolute position and in the case of encoder errors returns the error code (see /1/ Table 106 und Table 107).

3.3 Parameter channel (BMP protocol via Record Data ASE)

The implemented parameter channel (see /1/ Section 6.2.3) supports:

- Only one parameter per parameter job, this means no multi-parameter access (see /1/ Section 6.2.3.7)
- Standard types (see /1/ Section 5.2)
- PROFIdrive specific data types (see /1/ Section 5.3)
- Parameter value (see /1/ Section 6.2.1.2)
- Parameter description (see /1/ Section 6.2.1.3)
- Text (see /1/ Section 6.2.1.4)
- Data block length of 240 bytes (default setting, can be changed, see /1/Section 6.2.3.2)

Only the mandatory PAP data record 0xB02E for the "Base Mode Parameter Access - Local" is implemented. The optional data record 0xB02F for the "Base Mode Parameter Access - Global" is not implemented (is also not recommended). Other data records are not defined by PROFIdrive and have not been implemented.

5 parameter channels are implemented. This allows 4 IO controllers and 1 IO supervisor (device access) to be accessed simultaneously via separate ARs. The scope of 4 IO controllers and 1 IO supervisor is defined by the PROFINET-IO application example on which the PROFIdrive application example is based.

List of implemented parameters:

PNU	Parameter description	Write	Impl.	Valid.
100 -200	Examples of manufacturer-specific parameters for the configuration of the application (Parameters for ramp generator, threshold values, gain factors of the control system)	rw	V	L
300	Exemplary parameter for selecting the source for the frame selection (PROFINET configuration or PROFIdrive parameter PNU00922)	rw	V	L
700 -819	Exemplary manufacturer-specific parameters for tests with the PROFIdrive PROFINET Profile Tester	rw	V	L
900	Setpoint frame (DO IO DATA)	ro	O	L
907	Actual value frame (DO IO DATA)	ro	O	L
922	Frame selection	rw	M	L
925	Number of Controller Sign-Of-Life failures which will be tolerated	rw	O	L
930	Operating mode	rw	M	L
944	Fault message counter	ro	M	L
945	Fault code	ro	O	L
947	Fault number	ro	M	L
948	Fault time	ro	O	L
949	Fault value	ro	O	L
950	Scaling of the fault buffer	ro	O	L
951	Fault number list with text	ro	O	L
952	Fault situation counter	rw	O	L
953	Warning parameter	ro	O	L
964	Drive Unit identification data block	ro	M	G
965	Profile identification number	ro	M	L
974	Base Mode Parameter Access identification	ro	O	G
975	DO identification	ro	O	L
979	Sensor format	ro	M	L
980	Number list of defined parameter	ro	M	L
60000	Velocity reference value	ro	M	L
61000	NameOfStation	ro	O	G
61001	IpOfStation	ro	O	G
61002	MacOfStation	ro	O	G
61003	DefaultGatewayOfStation	ro	O	G
61004	SubnetMaskOfStation	ro	O	G

Write: ro – read only, rw – read-/writeable; Impl.: M – mandatory, O – optional, V – vendor specific;
Valid.: L – local, G - Global

3.4 Alarms (Alarm ASE)

All alarms are optional according to PROFIdrive. For example, alarms are generated as errors (faults) or warnings. The faults/warnings are set and reset in the application example for the demonstration via parameter PNU00700-PNU00702.

3.5 State machine

The general state machine is implemented according to /1/ Section 6.3.3.1.

3.6 Synchronization (Isochronous Mode Application ASE)

For the isochronous mode (mandatory for AC4) the synchronization of the PROFINET Stack is used and additionally synchronization via Sign of life (see /1/ Section 6.3.12) implemented.

3.7 Application

The control system (controller, inverter, motor, encoder) is simulated as a PT1 element.

4 Implementation of the application example in DevKit ERTEC 200P-2

4.1 Overview

The PROFIdrive_AC4 example is implemented in accordance with PROFIdrive V4.2 /1/.

The PROFIdrive_AC4 example requires the installation of the firmware stack for the Evaluation Kit (see /2/ and /3/). It is based on the application example "App3_IsoApp", which is located under `<InstallPath>\pnio_src\application\App3_IsoApp` (see /2/ Section 2.4 "Application examples").

The following procedure is recommended to use the PROFIdrive_AC4 example with the Evaluation Kit:

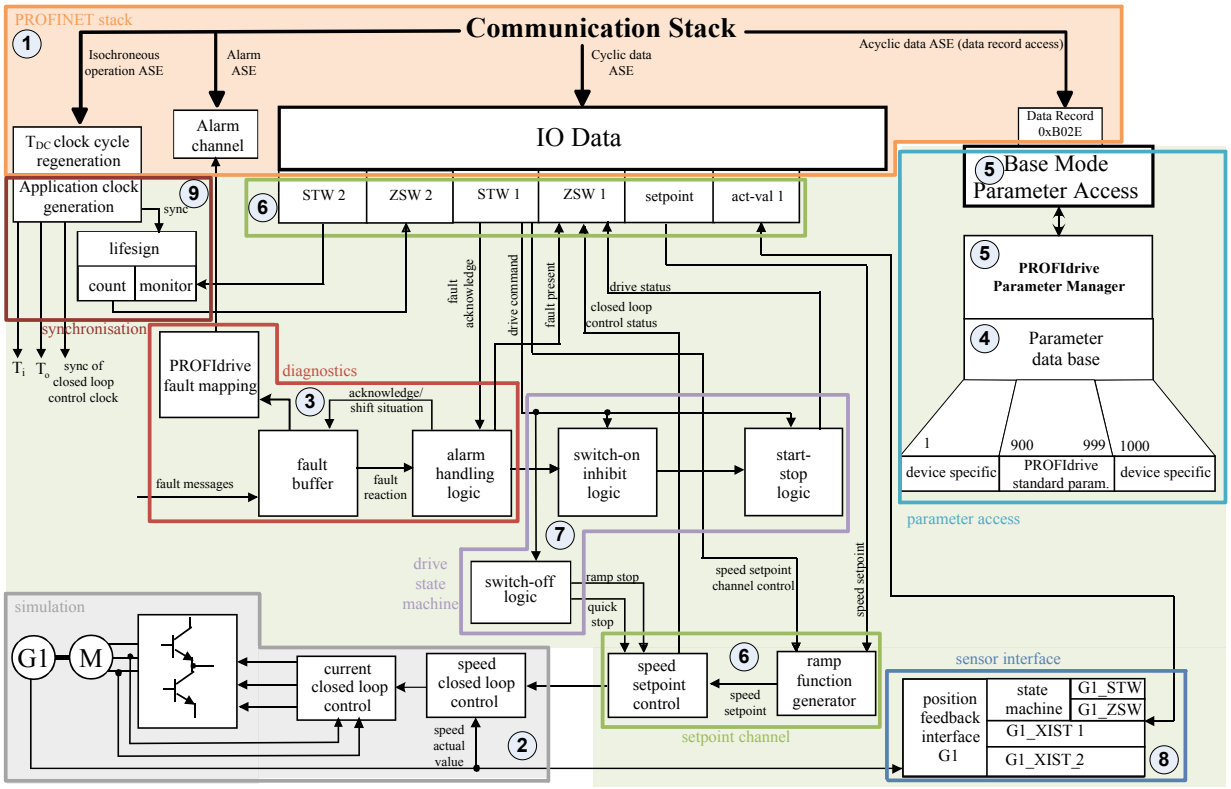
1. Complete commissioning of the Evaluation Kit as described in /3/ Section 4 "Quickstart Guide".
2. Copy the PROFIdrive source files from
   `<SRC_PROFIdrive_AC4_example.zip>\pnio_src\application\App44_PROFIdrive_AC4`
   to
   `<InstallPath>\pnio_src\application\App44_PROFIdrive_AC4`
3. Selection of the application example is carried out in the `usrapp_cfg.h` header file in `<InstallPath>\pnio_src\application\App_common` by means of the following entry:
   `#define  EXAMPL_DEV_CONFIG_VERSION  44`
4. New creation of the project and loading of the PROFIdrive_AC4 example and loading into the Evaluation Kit

For the configuration of a PROFINET project the example includes the modified GSDML file in
`<SRC_PROFIdrive_AC4_example.zip>\GSDML .`

4.2 Files for App44_PROFIdrive_AC4

Quick overview of the files of the application example:

File	Brief description	Fig. No.
usriod_main_pdrvac4.c	Main program of the PROFINET example Start of the PROFINET IO stack, initialization of the PROFIdrive application, login of the call-up of the PROFIdrive application at the isochronous interrupt, main loop with start of functions via keyboard of PROFINET functions (for example, firmware updates)	1
iodapi_event_pdrvac4.c	PROFINET interface with messages from PROFINET events to the application Contains functions which the PROFINET IO stack calls when events such as connection/disconnection, alarm reception, etc. occur and thus informs the application.	1
PnUsr_Api_pdrvac4.c	PROFINET subroutines for the user example. If necessary, the functions contained can be used as a function library in the customer application.	1
pdrv_application_ac4.c pdrv_application_ac4.h	PROFIdrive application 1ms time slice with call of the other PROFIdrive modules, simulation of the control system, terminal outputs for PROFIdrive events, implementation of the application parameters PNU100-PNU819	2
pdrv_diagnostics_ac4.c pdrv_diagnostics_ac4.h	PROFIdrive diagnostics with fault and warning handling	3
pdrv_parameter_ac4.inc	Macro file with list of the implemented PROFIdrive parameters	4
pdrv_pardatabase_ac4.c pdrv_pardatabase_ac4.h	PROFIdrive parameter database Creates the parameter database from pdrv_parameter.inc	4
pdrv_parmanager_ac4.c pdrv_parmanager_ac4.h	PROFIdrive Parameter Manager Realizes the PROFIdrive Base Mode Parameter Access	5
pdrv_setpointchannel_ac4.c pdrv_setpointchannel_ac4.h	PROFIdrive setpoint channel Implements access to the signals of standard frames 1 & 2; selection of the setpoint source based on the general state, ramp generator	6
pdrv_statemachine_ac4.c pdrv_statemachine_ac4.h	PROFIdrive General State Machine	7
pdrv_sensor_ac4.c pdrv_sensor_ac4.h	PROFIdrive encoder: Implements the encoder state machine and the simulated absolute value encoder.	8
pdrv_synchronization_ac4.c pdrv_synchronization_ac4.h	PROFIdrive synchronization Implements the synchronization via signs of life with the evaluation of CACF	9
pdrv_types_ac4.h	Definitions of PROFIdrive data types	



- Mapping of the source files to the functionality (see table)

4.3 usriod_main_pdrvac4.c

The module is created based on `<InstallPath>\pnio_src\application\App3_IsoApp\iodapi_event_isoapp.c` . It includes the startup of the PROFINET IO stack, the initialization of the PROFIdrive application, the initialization and start of the 1ms timer for the PROFIdrive application and main loop with the start of PROFINET functions (for example firmware updates) via keyboard.

In addition, the functionalities of the PNU61000-PNU61004 parameters are implemented with parameter read routines `uPdrv_RfPnu61000()` to `uPdrv_RfPnu61002()`.

4.4 iodapi_event_pdrvac4.c

The module is created based on `<InstallPath>\pnio_src\application\App3_IsoApp\iodapi_event_isoapp.c` . It contains functions that the PROFINET IO stack calls when events such as connection/disconnection, alarm receipt, etc. occur and thereby informs the application.

In the `PNIO_cbf_data_write()` function, the actual values ZSW1, ZSW2, NIST_A, NIST_B, G1_ZSW, G1_XIST1, G1_XIST2 are transferred from the PROFIdrive setpoint channel to the PROFINET stack. This handles the different byte order (Endianess, ERTEC Little Endian, PROFINET Big Endian).

In the `PNIO_cbf_data_read()` function, the setpoints STW1, STW2, NSOLL_A, NSOLL_B, G1_STW are transferred from the PROFINET stack to the PROFIdrive setpoint channel. This handles the different byte order (Endianess, ERTEC Little Endian, PROFINET Big Endian).

In the `PNIO_cbf_ar_connect_ind()` function, the establishment of the PROFINET-AR connection is communicated to the PROFIdrive Parameter Manager, which then reserves a parameter channel for this connection.

In the `PNIO_cbf_ar_disconn_ind()` function, the PROFINET-AR disconnection is communicated to the PROFIdrive Parameter Manager that then closes the parameter channel for this connection and releases the parameter channel instance.

In the `PNIO_cbf_ar_indata_ind()` function, the initial receipt of setpoints from the PROFINET stack is reported to the PROFIdrive setpoint channel.

In the `PnUsr_cbf_rec_read()` function, the read request of the PROFIdrive Base Mode Parameter Access Record is passed to the PROFIdrive Parameter Manager, which processes the read request accordingly and supplies the contents of the record.

In the `PnUsr_cbf_rec_write()` function, the write request PROFIdrive Base Mode Parameter Access Record is passed to the PROFIdrive Parameter Manager, which processes the write request accordingly and returns a corresponding response. Similarly the IsoM data record (IsochronousModeData, Index 0x8030) is evaluated here and checked for permitted values.

In the `PNIO_cbf_ar_ownership_ind()` function the frame function is realized according to PROFINET configuration (Expected Configuration Data) at PNU00300=0. The different PROFIdrive frames are displayed via different submodule IDs. At a frame exchange the PROFINET configuration and the drive actual status (Real Configuration Data) deviate from each other. In the function, it is checked whether a permitted frame is selected in the PROFINET configuration. If this is the case, the driver actual state is adapted (pulling and plugging of the submodule). Since the AC4-PROFIdrive application example includes the functions of the AC1-PROFIdrive application example compatibly a deviation of the PROFINET configuration to this effect is accepted here (see `PDRV_MODULE_ID_MAP1`).

The functionalities of the parameters PNU00300 "Source frame selection", PNU00922 "Frame selection", PNU00900 "Setpoint frame" and PNU00907 "Actual value frame" are implemented. The source for the frame selection is set in the parameter PNU00300. At PNU00300 = 0 (default value), the frame selection according to the PROFINET configuration (Expected Configuration Data) is carried out in the `PNIO_cbf_ar_ownership_ind()` function. At PNU00300 = 1, the frame selection is carried out via the PNU00922 parameter in the parameter write routine `uPdrv_WfPnu00922()`.

4.5 pdrv_application_ac4.c

The module is an example for the drive application. It consists of the `PdrvApp_main()` function, which is called cyclically every 1ms by the operating system. This function cyclically calls the PROFIdrive functions such as PROFIdrive general state machine, ramp generator, threshold value calculations, setpoint value channel, control system simulation, actual value transfer, encoder simulation and Parameter Manager. For demonstration purposes, calculated data are output and can be displayed in a terminal program via the serial interface of the Evaluation Kit.

The module also implements application-dependent PROFIdrive functions such as standstill detection `bPdrvApp_IsAxisStandstill()`, transition acknowledgment for the PROFIdrive general state machine `bPdrvApp_IsTransitionCondition()`, a 1s-Timer uPdrvApp_GetTimer1s() and a 1ms timer `uPdrvApp_GetTimer1ms()` .

Furthermore, the parameter read routines and parameter write routines for the application parameters and for the test parameters are implemented for testing with the Profile Tester.

4.6 pdrv_diagnostics_ac4.c

The module implements PROFIdrive diagnostic mechanisms (see /1/ Section 6.3.8 "Diagnostics"). Both the PROFIdrive "Complete Fault Buffer" (see /1/ Table 114 "Fault buffer parameters") as well as the PROFIdrive warnings (see /1/ Section 6.3.8.2 "Warning mechanism") are implemented. In addition the PROFINET IO Alarm ASE (see /1/ Section 8.8.2) is used to communicate diagnostics.

List of implemented parameters of the "Fault Buffer" and "Warning mechanism":

PNU	Parameter description	Write	Impl.	Valid.
944	Fault message counter	ro	M	L
945	Fault code	ro	O	L
947	Fault number	ro	M	L
948	Fault time	ro	O	L
949	Fault value	ro	O	L
950	Scaling of the fault buffer	ro	O	L
951	Fault number list with text	ro	O	L
952	Fault situation counter	rw	O	L
953	Warning parameter	ro	O	L

Write: ro – read only, rw – read-/writeable; Impl.: M – mandatory, O – Optional; Valid.: L – local, G - Global

The `PdrvDiag_SetFaultMsg()` function is used to set a fault and entry in the PROFIdrive fault buffer. A corresponding message is also generated via the standard PROFINET alarm mechanism. The `PdrvDiag_AckFaultSit()` function acknowledges a fault and is called with a positive edge of the STW1 Bit 7 and when the PNU00952=0 parameter is reset (see /1/ Figure 61 "Fault acknowledgment for the fault buffer mechanism").

The `PdrvDiag_AckSenFault()` function acknowledges a fault if it only consists of an encoder error, and is called by the encoder state machine.

The `bPdrvDiag_IsFaultOff1()`, `bPdrvDiag_IsFaultOff2()`, `bPdrvDiag_IsFaultOff3()` functions return whether a fault is active with the corresponding fault reaction "Power down", "Coast stop" or "Quick stop"

.

The `PdrvDiag_SetWarning()` function is used to set and reset a warning. A corresponding message is also generated via the standard PROFINET alarm mechanism. Furthermore, the parameter read routines and parameter write routines are implemented for the parameters listed above.

4.7 pdrv_pardatabase_ac4.c

This module implements the parameter database in which the parameters defined in `pdrv_parameter_ac4.inc` file are created using macros in C structures.

The `ptPdrvPar_GetParObj()` function searches for a parameter object in the parameter database and returns a pointer to the parameter object if it is successful.

In addition to the properties listed in /1/ Section 6.2.1.3 "Parameter description", a parameter object contains 3 different access functions. The parameter read routine is called when the parameter access is read and fills the response buffer with the requested current parameter values. The write routine is called during write access and gets the written values from the job buffer and enters these into the memory associated with the parameter. The text routine is called when an additional text is requested (see /1/ Section 6.2.1.4 "Text") and returns a pointer to the requested text.

Furthermore, the parameter read routines and parameter write routines are implemented for various parameters.

4.8 pdrv_parmanager_ac4.c

This module implements the PROFIdrive Base Mode Parameter Access (see /1/ Figure 131).

When a PROFINET connection is established, the `bPdrvPar_EstablishCxn()` function is called. This checks whether a PROFIdrive parameter channel is available for the PROFINET connection and reserves a PROFIdrive parameter channel for further use with this PROFINET connection (AR). In the process the PROFIdrive parameter channel is reset. If the same PROFINET connection is closed, the `bPdrvPar_DisconnCxn()` function is called and the PROFIdrive parameter channel is also closed and released for later use. Currently 5 PROFIdrive parameter channel instances are available simultaneously (configurable in `PARCXN_NR` constant). The restriction to 5 PROFIdrive parameter channel instances is based on the restriction of the used PROFINET Evaluation Kit EK200P-2 V4.5. This allows a maximum of 5 simultaneous PROFINET connections to be maintained, of which there are a 4 controller connections maximum as well as one device access connection.

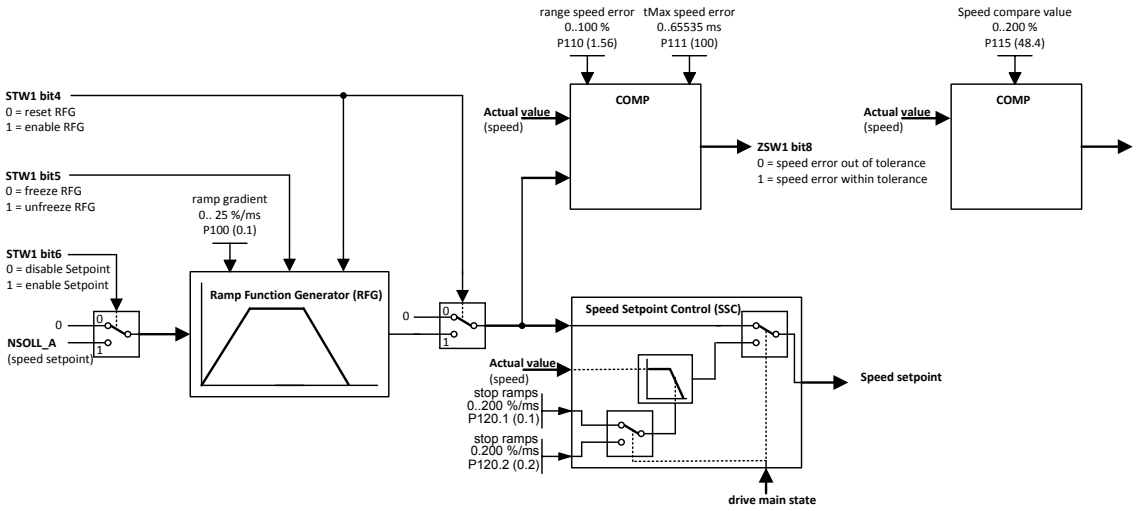
When receiving record requests for the Base Mode-Parameter Access Local Record (0xB02E), the `uPdrvPar_WriteReqCxn()` or `uPdrvPar_ReadReqCxn()` functions are correspondingly called.
These generate the response to this record request in accordance with the state of the associated PROFIdrive parameter channel (see /1/ Table 33). With a valid Write.req, the PROFIdrive parameter request is transferred to the request buffer for further processing. With a valid Read.req, the response is taken from the response buffer.

The `PdrvPar_ProcessReq()` function is called cyclically. With each call the function checks if there is a parameter request to be processed in one of the parameter channels, the parameter request is processed and the result stored in the response buffer. If there are parameter jobs to be processed in several parameter channels, the parameter jobs are processed sequentially according to the round robin method. Note: The PROFIdrive BMP transport protocol is also in the same way by Profienergy for accessing the Profienergy Service Access Point (SAP). Therefore this code example can be used as basis for the Profienergy SAP.

The different byte order (Endianess, ERTEC Little Endian, PROFINET Big Endian) is handled when processing the parameter jobs.

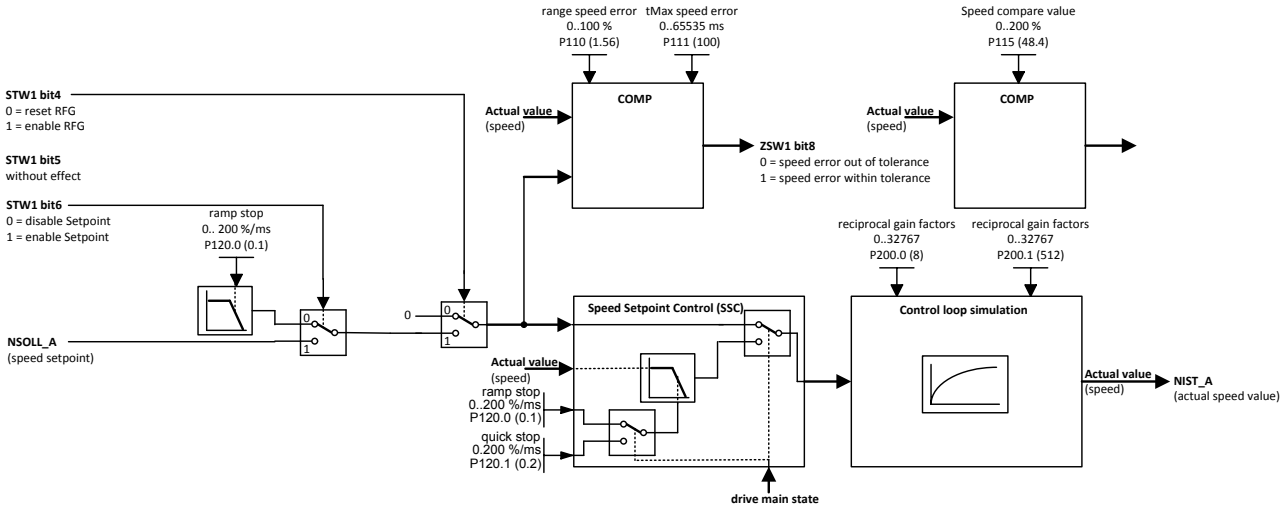
4.9 pdrv_setpointchannel_ac4.c

This module implements the setpoint channel. The cyclical setpoints of the standard frames are stored here for further use and are made available with get functions.



Setpoint editing, if PNU00930=1

The ramp generator (see /1/ Figure 29) is implemented without jogging functionality in the `nPdrvSpc_CalcRfg()` function. At PNU00930=1 the function is called up by the main function `PdrvApp_main()`.



Setpoint processing, if PNU00930=3

The reduced speed setpoint value channel (see /1/ Figure 31) is implemented without jogging functionality in the `nPdrvSpc_CalcReducedSsc()` function. At PNU00930=3 the function is called up by the main function `PdrvApp_main()`.

The `nPdrvSpc_CalcSsc()` function implements a speed setpoint control which, depending on the state of the general state machine, either passes through the ramp generator setpoint unchanged or realizes an OFF ramp with normal or rapid stop speed.

4.10 pdrv_statemachine_ac4.c

The `ePdrvSma_AxisGeneralStateMachine()` function implements the general state machine that is described in /1/ Figure 27 with the state graphs. The current status of the general state machine can be queried with the `ePdrvSma_GetAxisMainState()` function.

4.11 pdrv_sensor_ac4.c

This module realizes a simple encoder simulation and the encoder state machine (see /1/ Section 6.3.6).

The encoder state machine `PdrvSen_SensorStateMachine()` realizes the states and state transitions according to /1/ Figure 52. The optional states SD14 "Parking & error acknowledgement", SD15 "Parking & error" and the respective state transitions TD24 to TD29 are currently not implemented.

The encoder simulation `nPdrvSen_Sensor()` calculates a position value from the actual speed NIST_B and makes the position available as G1_XIST1. The encoder simulation solely supports the state s SD1 "Normal operation", SD2 "Error acknowledgement", SD3 "Error" and SD12 "Parking" and has not implemented additional functionalities (SD4 to SD11).

In addition the parameter read routine for parameter PNU00979 is implemented here.

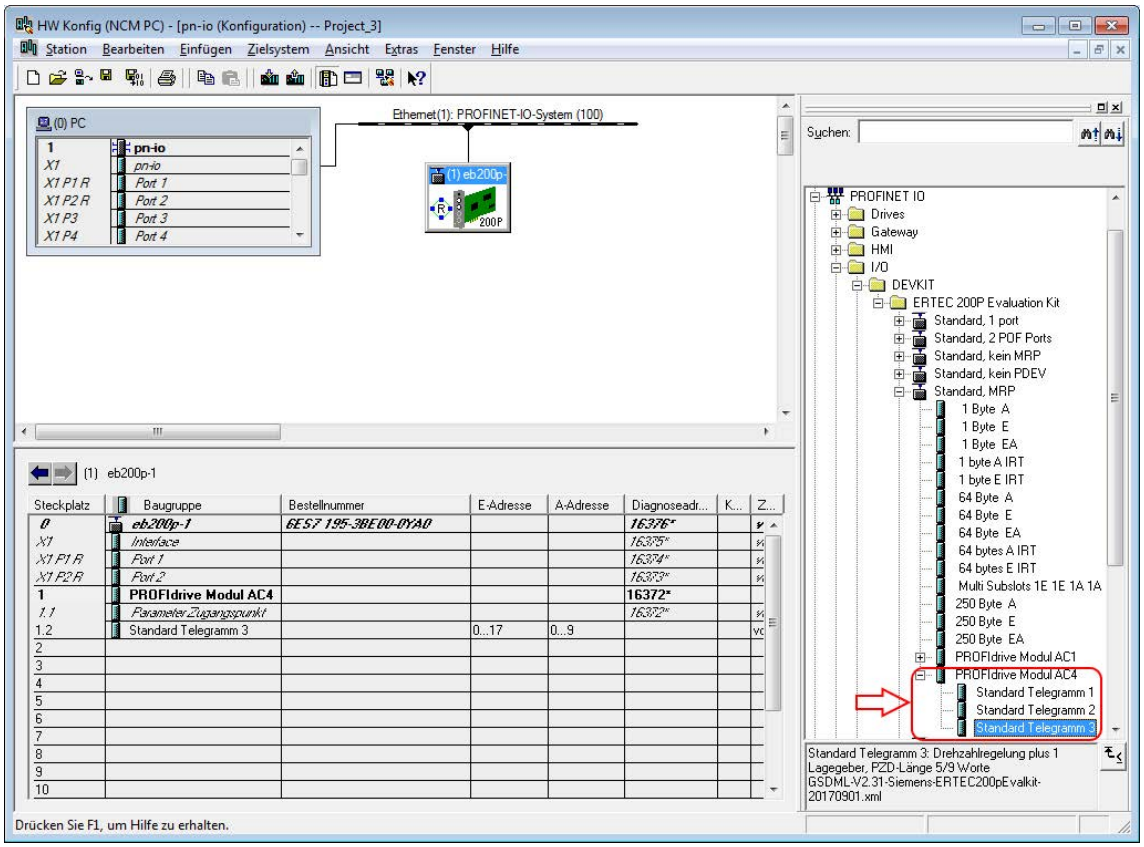
4.12 pdrv_synchronization_ac4.c

The isochronous mode is implemented by the used PROFINET stack. The PROFIdrive main function `PdrvApp_main()` is called up accordingly. In `PdrvSyn_CheckSignOfLife()` only the synchronization of the application via the sign of life mechanism (see /1/ Section 6.3.12) is realized. The state of the sign of life synchronization can be sampled via the `bPdrvSyn_IsSyncedSignOfLife()` function. The CACF factor required for the evaluation of the master sign of life is taken from the ISOM parameter block transferred during the establishment of the connection.

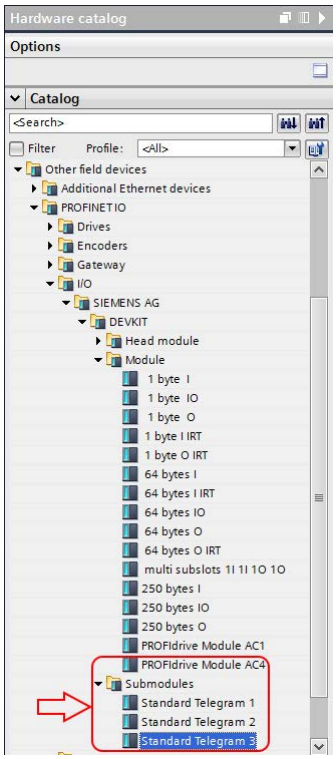
Furthermore, the parameter read routine and parameter write routine are implemented for parameter PNU00925. The sign of life monitoring can be deactivated by writing the 0xFFFF value in PNU925.

4.13 GSDML file

The PROFIdrive module (Drive Object DO) has been added to the GSDML file of the EK-ERTEC 200P-2 PN IO V4.5.0 Evaluation Kit. The PROFIdrive module thus appears in the same category as the other IO modules of the Evaluation Kit.



Example for hardware configuration in the program NCM PC (Profile Tester)



PROFIdrive example in the hardware catalog of the TIA Portal

5 Adaptation of the application example

5.1 Identification data

In addition to the changes to the PROFINET stack that are described in /3/ Section 7.2.1, the following identification data have to be adapted in the application example.

The `PDRV_MODULE_ID_MAP` module ID has to be configured in `pdrv_application.h` (manufacturer-specific).

The identification data HAVE TO be configured in the `pdrv_application.h` file (for more on this, see parameters PNU00964, PNU00975 in /1/ Table 118 and Table 120).

Constant	Description
PDRV_ID_MANUFACTURER	Manufacturer code according to PNO
PDRV_ID_DUTYPE	Drive Unit Type (manufacturer-specific)
PDRV_ID_DOTYPE	Drive Object Type (manufacturer-specific)
PDRV_ID_FWVERSION	Firmware version
PDRV_ID_FWDATE_Y	Year of the firmware date
PDRV_ID_FWDATA_DM	Day and month of the firmware date

5.2 Diagnostics

The faults and warnings implemented as examples are to be removed and the faults and warnings occurring in the new application should be implemented.

This involves

- Removing the entries `FAULT_USER0` to `FAULT_USER9 and WARNING_USER0` to `WARNING_USER9` in the `pdrv_diagnostics.*` files
- Removing the parameters PNU00700 to PNU00702 in the `pdrv_parameter.inc` file and the associated implementations (tags, functions) in `pdrv_application.c` (example for testing with the Profile Tester)
- New warnings are to be assigned to a fault class of the PROFINET standard alarm functionality and entered in the `m_tWarningAttribs[]` table. For more than 16 warnings, the PNU00954 to PNU00960 parameters have to be implemented.
- New faults are stored in the `m_tFaultAttribs[]` table.
  The following must be assigned to a fault for this:
  * A fault code (PNU00945)
  * A fault number (PNU00947)
  * A fault reaction (OFF1 ≙ Motor OFF, OFF2 ≙ Pulse lock, OFF3 ≙ Rapid stop, none)
  * A fault class of the PROFINET standard alarm functionality
  * A fault text (16 characters, PNU00951)
  .
- The `PdrvDiag_SetFaultMsg()` and `PdrvDiag_SetWarning()` are to be called correspondingly in the new application

A check must be carried out whether a new application requires a consistency check during the diagnostic processing. In the example application, no consistency protection is required since all software components are processed sequentially with the same priority.

5.3 Parameter

The parameters implemented as examples must be removed and the parameters occurring in the new application must be implemented.

This involves

- Removing the parameters PNU00100 to PNU00200 in the pdrv_parameter.inc file and the associated implementations (tags, functions) in `pdrv_application.c` (parameters of the example application)
- Removing the parameters PNU00700 to PNU00819 in the `pdrv_parameter.inc` file and the associated implementations (tags, functions) in `pdrv_application.c` (parameters for testing with the profile tester)
- Removing the parameter PNU00300 in the `pdrv_parameter.inc` file and selecting one of the two variants for the frame selection in `iodapi_event_pdrvac4.c`. If possible, the frame should be selected according to the PROFINET configuration. In this case, the PNU00922 parameter only displays the configured frame.
- The parameters PNU00900 to PNU00999 in the `pdrv_parameter.inc` file must be adapted to the changed conditions.
- The application-specific parameters must be implemented for the new application.

Each parameter requires a read routine `pfnRead()`. Writable parameters also require a write routine `pfnWrite()`. If a parameter contains additional texts, a text routine `pfnText()` is required.

Note: It is recommended to generate the parameter database automatically from a separate parameter database. The macro method selected in the example application makes it difficult to find syntax errors and formal errors in the parameter database, which is implemented in the `pdrv_pardatabase.c` file.

The example application did not implement retentive storage for parameters and therefore also does not implement an explicit reset to the factory state.

The following additional setting options for parameter processing are provided:

Constant	Description
PDRV_PAR_BLOCKSIZE	Data block size of a parameter request/response (see /1/ Section 6.2.3.2)
PARCXN_NR	Number of parameter channels (simultaneously possible parameter connections). The PROFINET stack has to be adapted accordingly if there is a change.

5.4 Isochronous Mode Data (IsoM)

The data for the isochronous operation are to be adapted in the `pdrv_application.h` file:

Constant	Description
PDRV_ISO_TDC_MIN	T_DC_Min minimal time data cycle [ns]
PDRV_ISO_TDC_MAX	T_DC_Max maximal time data cycle [ns]
PDRV_ISO_IOI_MIN	T_IO_InputMin minimal time for data input [ns]
PDRV_ISO_IOO_MIN	T_IO_OutputMin minimal time for data output [ns]

In the `case 0x8030` of the `PnUsr_cbf_rec_write()` function the application specific reactions to changes of the isochronous mode data are to be implemented.

5.5 Application

The example application in the `pdrv_application.c` file consists of a very simple simulation of the full control system (actuator, motor, measured value acquisition) using a PT1 element and in the `pdrv_sensor.c` file of a simple encoder function. These simulations are to be replaced by your own application and the encoder implementation.

The application has to call the following PROFIdrive functions in the correct order with correct parameters:

- General state machine `ePdrvSma_AxisGeneralStateMachine()`
- Ramp generator `nPdrvSpc_CalcRfg()` or reduced setpoint channel `nPdrvSpc_CalcReducedSsc()`
- Setpoint operation `nPdrvSpc_CalcSsc()`
- Tolerance calculation for ZSW1 Bit 8 `bPdrvSpc_CalcSpeedWithinTolerance()`
- Comparison for ZSW1 Bit 10 `bPdrvSpc_CalcSpeedReached()`
- Actual value transfer `PdrvSpc_SetNistA()`
- Encoder interface `nPdrvSen_Sensor()`, `PdrvSpc_SetG1Xist1()`, `PdrvSen_SensorStateMachine()`, `nPdrvSen_GetGxXist2()`
- Parameter Manager `PdrvPar_ProcessReq()`

Typically, with the exception of the parameter manager, the functions are executed in a constant bus cycle with the same priority as the actual drive application. The parameter manager is often executed with a lower priority and not necessarily with a constant bus cycle. In the present example, all functions with the same priority are processed every 1 ms, which is why consistency checks were omitted in the example. If an application is implemented with different execution priorities, appropriate measures (interrupt locks, alternating buffers) must be implemented at appropriate points for consistency.

Furthermore, the following functions must be implemented:

- Standstill detection `bPdrvApp_IsAxisStandstill()` (used in the general state machine)
- Transition acknowledgment `bPdrvApp_IsTransitionCondition()` (used in the general state machine)
- 1s-Timer `uPdrvApp_GetTimer1s()` (used with fault buffer PNU00948, optional)
- 1ms-Timer `uPdrvApp_GetTimer1ms()` (used in encoder state machine)

The transition acknowledgment bPdrvApp_IsTransitionCondition() is used to delay the state propagation of the general state machine by the application, if the application is not yet ready for the subsequent state. This is in each case for the general state transition from S1 Switch-on inhibit to S2 Ready for switch-on, from S2 Ready for switch-on to S3 Ready for operation, from S3 Ready for operation to S4 Operation (see /1/ Figure 27 "General State Diagram").

The nPdrvSen_Sensor() encoder simulation is to be replaced by an encoder implementation. This also applies for the functions associated with the encoder function:

- Encoder error `bPdrvSen_IsSensorError()`
- Validity of the encoder position `bPdrvSen_IsGxXist1Valid()`
- Reference value found `bPdrvSen_IsRefValFound()`
- Reference marks found `bPdrvSen_IsRefMarkFound()`
- Encoder measurement active `bPdrvSen_IsMeasureActive()`

5.6 Dynamic Servo Control (DSC)

Dynamic Servo Control (DSC) function is a closed-loop control structure which is calculated in the fast speed controller cycle and supplied with setpoints from the controller in the position control cycle. This way higher position controller gains can be achieved. This allows the rigidity and the dynamics of the control circuit to be increased. Requirement is the isochronous mode with encoder interface. (See /1/ Section 6.3.5)

In addition to the changes specified in the preceding Section 5.5 "Application", the following changes are additionally required:

- Realizing a drive regulator with position controller and speed pre-control (see /1/ Figure 42)
- Implementing standard frame 5 (see /1/ Section 6.3.4.3.6)
- Parameter PNU900 Increase number of elements from 10 to 18 in `pdrv_parameter_ac4.inc`
- Parameter PNU922 Increase upper limit from 3 to 5 in `pdrv_parameter_ac4.inc`
  and possibly modify `uPdrv_WfPnu00922()` so that the value 4 is not permissible
- Extension of the GSDML file with standard frame 5

Standard frame 5 is an extension of standard frame 3 with the signals XERR und KPC. The implementation of standard frame 3 can therefore serve as a template. Corresponding code locations can be found by searching for the string "TLG3" in the source files iodapi_event_pdrvac4.c and pdrv_application_ac4.h. The signals XERR and KPC have to be included into the setpoint value channel (for example NSOLL_B signal with PdrvSpc_SetNsollB()) and to be processed in the drive controller.

5.7 Communication stack

When changing the communication stack, the modules `iodapi_event_pdrvac4.c` and `usriod_main_pdrvac4.c` have to be adapted or replaced.

PROFIdrive uses/requires the following functions:

- Communication connection `PNIO_cbf_ar_connect_ind()`
- Communication disconnection `PNIO_cbf_ar_disconn_ind()`
- Detection of the first successful communication exchange `PNIO_cbf_ar_indata_ind()`
- Send cyclic data `PNIO_cbf_data_write()`
- Receive cyclic data `PNIO_cbf_data_read()`
- Read data record `PnUsr_cbf_rec_read()`
- Write data record `PnUsr_cbf_rec_write()`
- Channel diagnostics `uPdrvUsr_ChanDiag()`
- Pulling and plugging a submodule `uPdrv_WfPnu00922()` (when using multiple standard frames, optional)
- PROFINET and TCP/IP station data `uPdrv_RfPnu61000()` - `uPdrv_RfPnu61002()` (optional)

In the modules `iodapi_event_pdrvac4.c` and `usriod_main_pdrvac4.c`, the code locations for PROFIdrive can be easily found by searching for the string "Pdrv".

When changing the processor platform or the communication stack, take into account note that the ERTEC 200P of Evaluation Kit works in the Little Endian format and that PROFINET uses Big Endian format.

6 Appendix

6.1 Terms / Abbreviations

API	Application Interface
API	Application Process Identifier
AC	Application Class (PROFIdrive)
ASE	Application Service Element (PROFIdrive)
DAP	Drive Access Point (PROFINET)
DSC	Dynamic Servo Control (PROFIdrive)
DO	Drive Object (PROFIdrive)
DU	Drive Unit (PROFIdrive)
MAP	Module Access Point (PROFIdrive)
PAP	Parameter Access Point (PROFIdrive)
P-Device	Peripheral device (PROFIdrive)

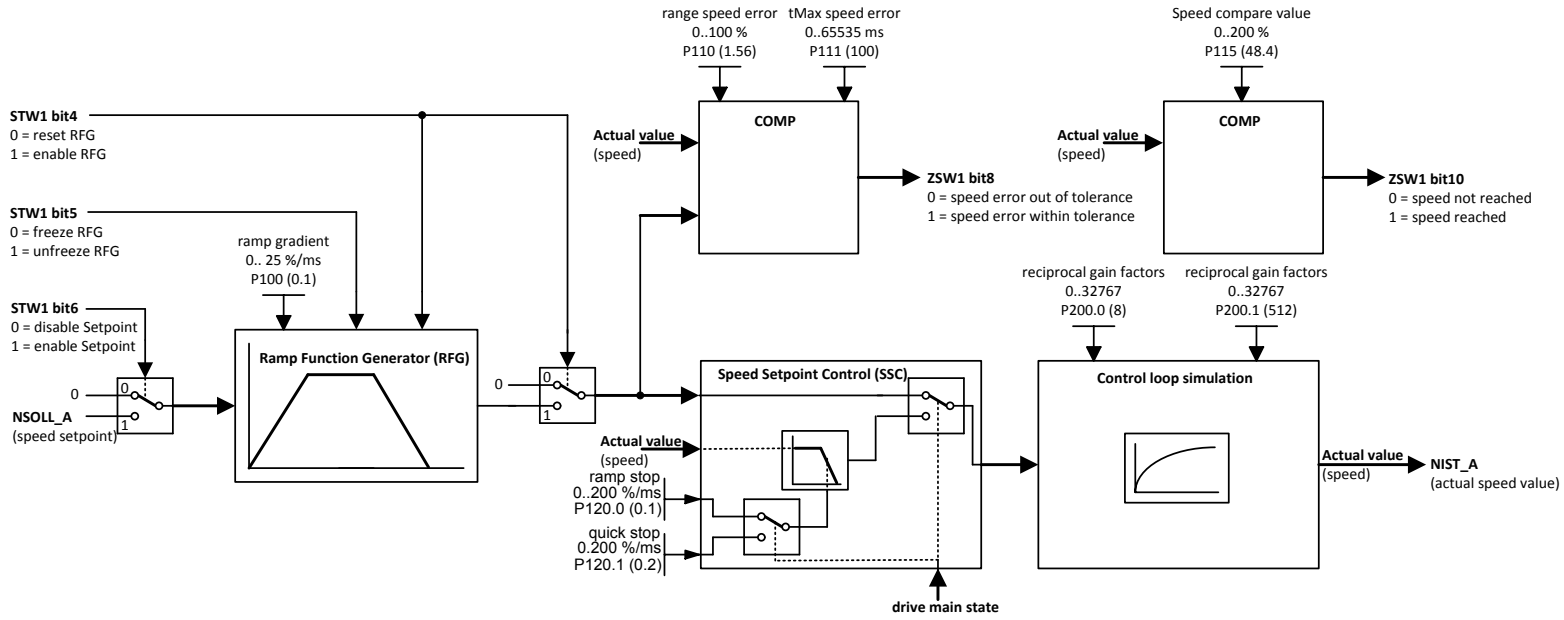
6.2 References

/1/ PROFIBUS User Organization: Profile Drive Technology PROFIdrive Version 4.2 October 2015
https://www.profibus.com/download/profidrive-profile-drive-technology/

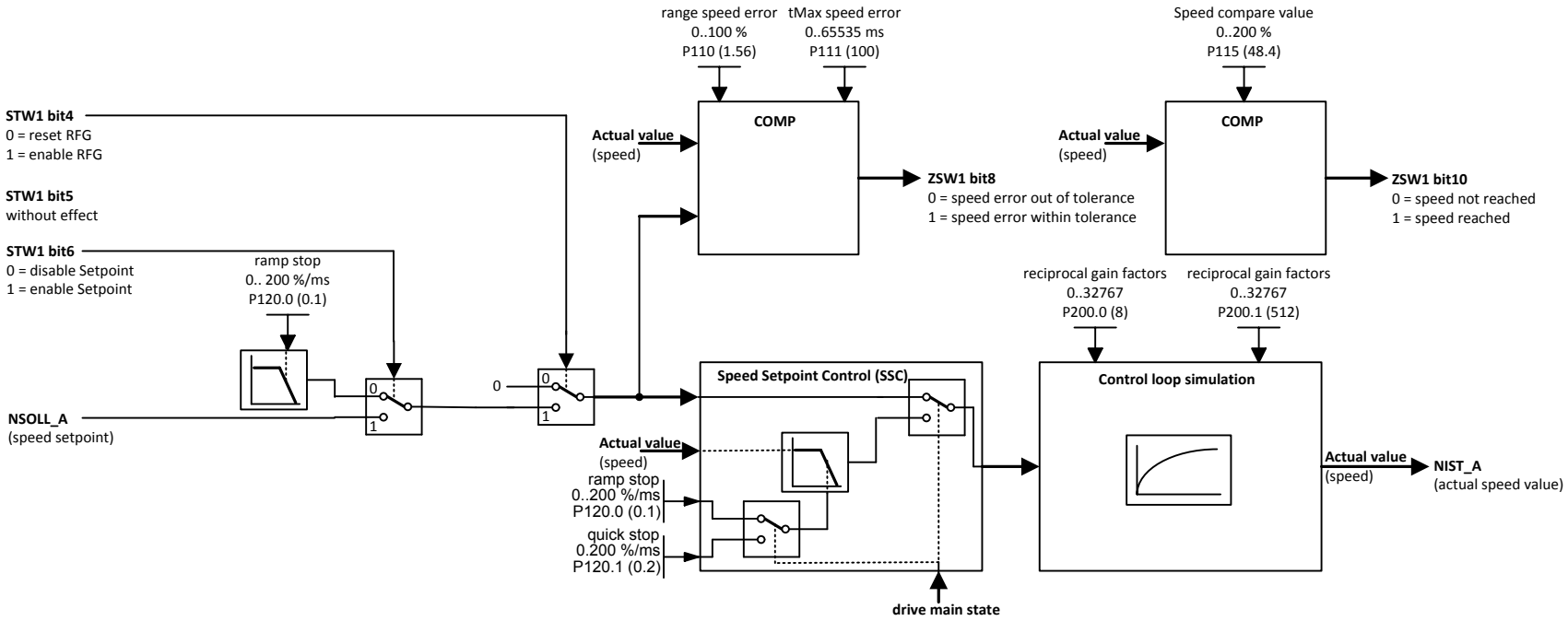
/2/ Siemens AG: "Interface description PROFINET IO Development Kits V4.5.0" Programming and Operating Manual, A5E33638878-AC, 11/2016

/3/ Siemens AG: "Guidelines for Evaluation Kit ERTEC 200P-2 V4.5.0" Programming and Operating Manual, A5E03855331-AC, 11/2016

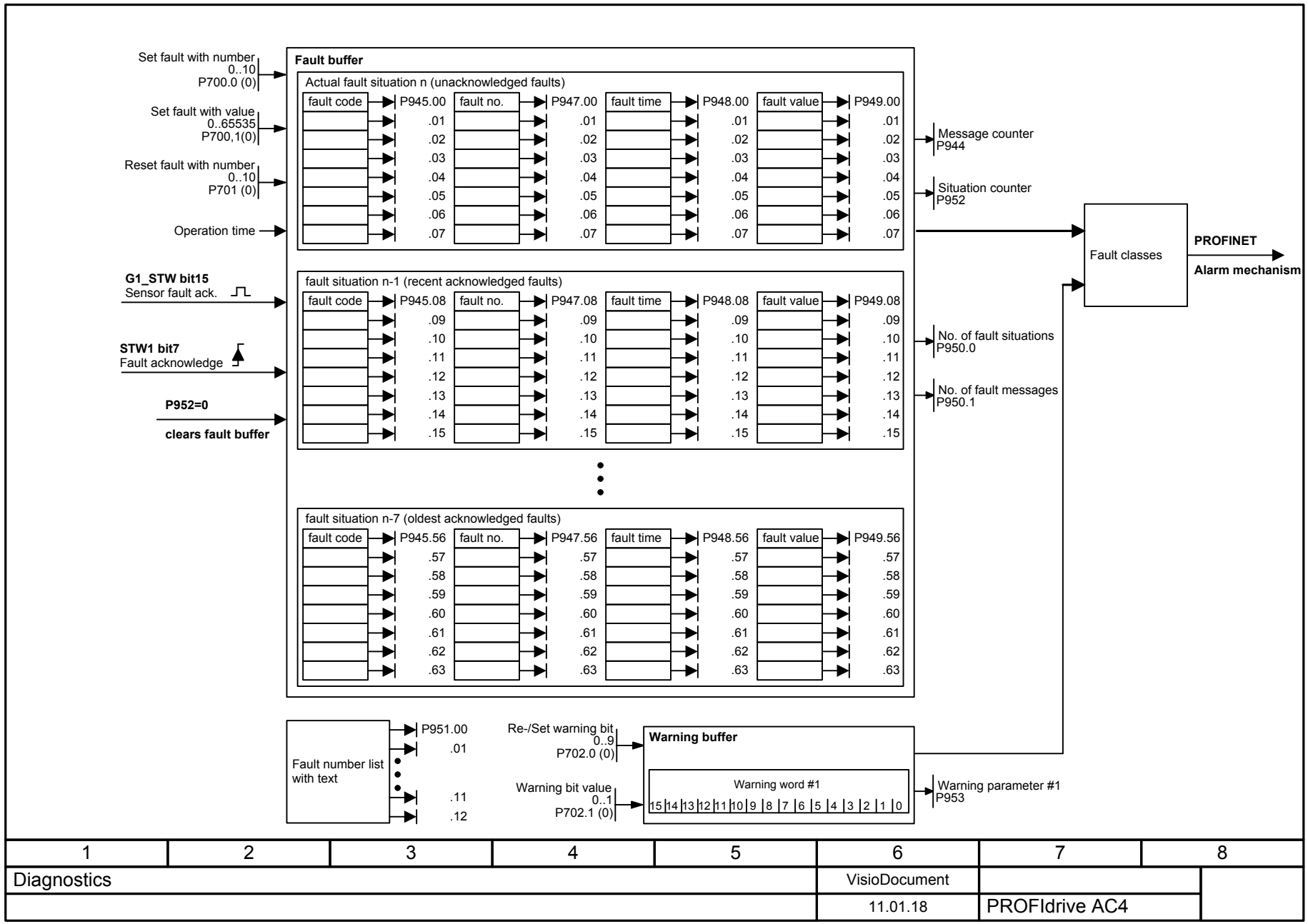
6.3 Function charts



1	2	3	4	5	6	7	8
Setpoint handling with PNU00930=1					VisioDocument		
					11.01.18	PROFIdrive AC4	



1	2	3	4	5	6	7	8
Setpoint handling with PNU00930=3					VisioDocument		
					11.01.18	PROFIdrive AC4	



1	2	3	4	5	6	7	8
Diagnostics					VisioDocument		
					11.01.18	PROFIdrive AC4	