SIMATIC

PROFIsafe driver V2.1

for F-Slaves

Manual

Edition 05 / 2010

Technical safety guidelines

This manual contains notices intended to ensure your personal safety, as well as to protect the product and connected equipment against damage. The notices are highlighted by a triangular warning symbol and are graded according to severity as follows:

Safety information

This is an important notice for the approval and failsafe use of the product.

Note

Draws your attention to particularly important information on the product, handling the product, or to a particular part of the documentation.

Trademarks

SIMATIC®, SIMATIC HMI® and SIMATIC NET® are trademarks of Siemens AG.

All other names appearing in this document may be trademarks; use of such names by third parties for their own purposes may infringe upon owners rights.

Siemens AG

Industry Sector

Division Industry-Automation

P.O. Box 4848, D- 90327 Nuremberg, Germany

Contents

Table of illustrations

1 Preface

1.1 Purpose of the manual

This manual describes the procedure for integrating the "PROFIsafe driver for F-Slaves" (hereinafter abbreviated to **PSD**) in various runtime environments.

1.2 Scope oF-Application

The PSD for F-Slaves is used in PROFIBUS and PROFINET applications with safety-oriented field devices. These include, for example,: remote I/Os, laser scanners, photo-electric guards, drives, valves, robots, operator panels, pressure transmitters, overflow safeguard, gateways to other failsafe buses, and many other systems or devices. In this manual, the devices that are failsafe are preceded by the letter "**F**", for example F-Slave and F-Host.

1.3 What's New

This manual, which refers to PSD V2.1, includes the following changes and

amendments compared to the previous edition:

- Some links to textmarks corrected.
- Description of the status byte bit_1 "Device Fault" changed.
- Restrictions relating the ban of code optimization (PSD_08) changed.

1.4 Customer benefits

The certified PSD for slaves reduces the development and certification overhead for F-device manufacturers. Development of a PSD by a PNO member offers the following benefits:

- The F-communication operates in the same manner for all F-Slaves that use the PSD (secures PROFIsafe profile /1/).
- You only need to go through the implementation, testing and certification procedures once.

1.5 Target group

F-device development department at Siemens and partner companies.

1.6 References

/1/	PROFIsafe – Profile for Safety Technology on PROFIBUS and PROFINET IO Order No. 3.192b, Version 2.4, March 2007 Publisher: PROFIBUS Nutzerorganisation e.V. Haid-und-Neu-Str. 7, D-76131 Karlsruhe Phone: ++ 721 / 96 58 590, Fax:++ 721 / 96 58 589 pi@profibus.com, http://www.profibus.com/
/2/	IEC 61508 Functional Safety of electrical/electronic/programmable electronic safety-related systems Part 1: General requirements (version 1999) Part 2: Requirements for electrical/electronic/programmable electronic safety-related systems (version: 2000) Part 3: Software requirements (version: 1999) Part 4: Definitions and abbreviations (version: 1999) Part 5: Examples of methods for the determination of safety integrity levels (version: 1999) Part 6: Guideline on the application of IEC61508-2 and IEC61508-3 (version: 2000) Part 7: Overview of techniques and measures (version: 2000)
/3/	PRQA QA C for C/C++ Ver. 7.0 http://www.programmingresearch.com/QAMISRA.html?pnl=1_3#1_3
/4/	PROFIsafe Starter Kit

Other unreferenced literature:

- Programming guidelines for the design of software of safety-related systems
  Wirtschaftsverlag NW, 1998
  Publisher: Federal Institute for Occupational Safety and Health
  Author: Dr. Schaefer. ISBN 3 89701 212 X
- PROFIBUS – DP/DPV1 Basics, tips and tools for users
  Author: Manfred Popp, Hüthig, 2000, ISBN 3-7785-2781-9
- The Rapid Way to PROFIBUS-DP
  Author: Manfred Popp, PROFIBUS User Organization, order no. 4.071
- Industrielle Kommunikation mit PROFINET,
  Author: Manfred Popp, PROFIBUS User Organization, order no. 4.071
- PROFIsafe Requirements for Installation, Immunity and Electrical Safety
  V1.0, order no. 2.242 2.242

1.7 Other support

You can obtain further information or answers to questions via the ComDEC e-mail address: ComDeC@siemens.com

For more information about PROFIBUS, please go to the homepage http://www.profibus.com/

2 Product overview

2.1 Description of the PROFIsafe driver (PSD)

The PSD is a firmware module for F-Slaves. It contains the monitoring mechanisms for safety-oriented communication as defined in the "PROFIsafe – Profile for Safety Technology on PROFIBUS DP and PROFINET IO" profile. The PROFIsafe driver allows manufacturers of F-Slaves to easily connect their F-Slaves to the standard PROFIBUS as well as standard PROFINET and conduct failsafe communication with one F-communications partner (F-Host).

With the PSD, equipment manufacturers are able to design and implement F-devices with failsafe point-to-point communication via PROFIBUS or PROFINET. The PSD can be integrated in different runtime environments via "generation" (hardware/firmware environments).

2.2 Safe communication between F-Host and F-Slave

The PSD's main task in terms of safety is to ensure failsafe exchange of user data with an F-Host using the required F-Parameter assignment. The structure of a complete input / output telegram corresponds to the specification in the PROFIsafe-Profile (Fig. "Complete F-message structure") /1/. In the following, these expressions are used in the same way as in the following diagram.

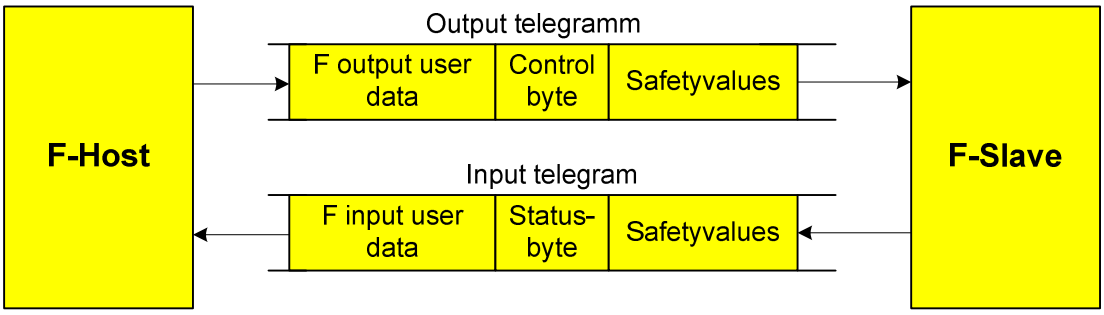

Fig. 2-1: Structure of the F-telegrams

For the user of the PSD, the following data is relevant:

Output telegram:

- F output user data (structure is application-specific)
- control byte.bit_0: iPar_EN
  (F-Host sets parameter request for i-Parameters)
- control byte.bit_1: OA_Req
  (F-Slave can display operator acknowledge request)
- control byte.bit_4: activate_FV
  (is utilized by F-Host to induce the F-Slave to transmit failsafe values at its outputs)

Input telegram:

- F input user data (structure is application-specific)
- status byte.bit_0: iPar_OK
  (F-Slave has adopted the i-Parameters)
- status byte.bit_1: Device_Fault
  (F-Slave has detected a safety relevant malfunction)
- status byte.bit_4: FV_activated
  (F-Slave uses failsafe values at its outputs)

The meanings of the bits in the status / control byte are described in the PROFIsafe-Profile (Fig. "Status Byte" or "Control Byte") /1/.

Safety information

Requirement: PSD_16

The status byte.bit_1 shall be set by the F-Application for at least two (2) changes of the consecutive number (2 x returnvalue "F_OUTPUT_OK" of the PSD-function psd_GetFOutData()), if the F-Slave is not able to guarantee the safety integrity of the process data to be transmitted. Signal name is "Device Fault".

2.3 Error detection mechanisms in the PSD

A variety of measures are implemented in the PSD to detect errors. These take effect immediately on processing the PSD code, which means that an additional, cyclical PSD self-test function is not required to be started.

- In the single-channel PSD, a high-quality flow-control is fitted in all interface functions to detect falsifications in the program flow. The functions are processed logically in dual-channel, whereby in the second channel the data is stored as inverted, which can also be seen as the function call-ups of the PSD proceed.
- In the dual-channel PSD, a synchronization function helps to implement a flow-control in cyclical operation.

2.4 Standards and guidelines

The PSD was developed according to the SIL3 requirements of IEC 61508-1 and -3. On the single-channel variant, the SIL2 requirements as regards RAM, ROM and program flow monitor from IEC 61508-2 have also been taken into account and implemented.
It meets the requirements of the standards and guidelines with regard to functional safety as stated in the report issued with the safety certificate (TÜV certificate). The latest TÜV documents are available in the Internet at (http://support.automation.siemens.com/WW/view/de/11669702/134200).

Safety information

Requirement: PSD_21

Analysis with the verification code in the file p_verify.c /4/ must be performed to verify the compiler.

3 Function principle of the PSD

3.1 Integration of the PSD in F-Slaves

The following diagrams show the FW blocks integrated in F communication and the integration of the PSD.

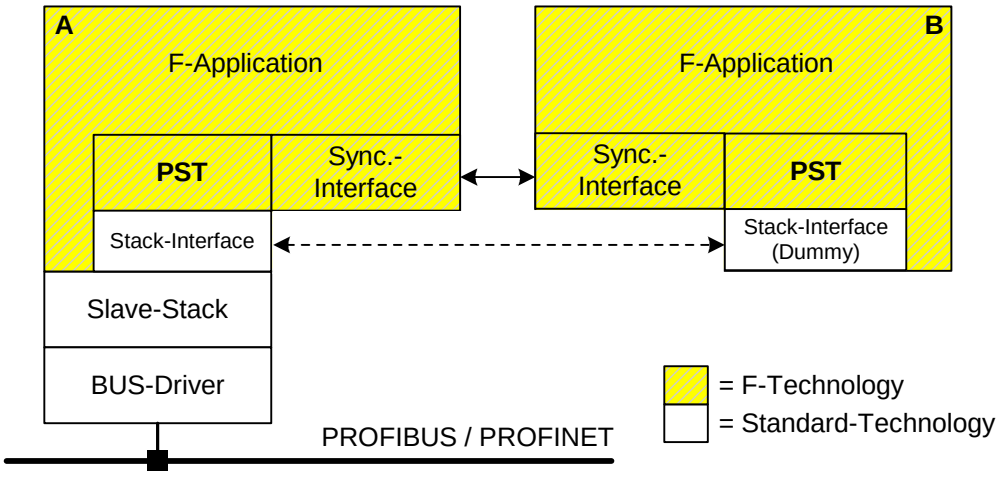

Fig. 3-1: Integration of the PSD (dual-channel)

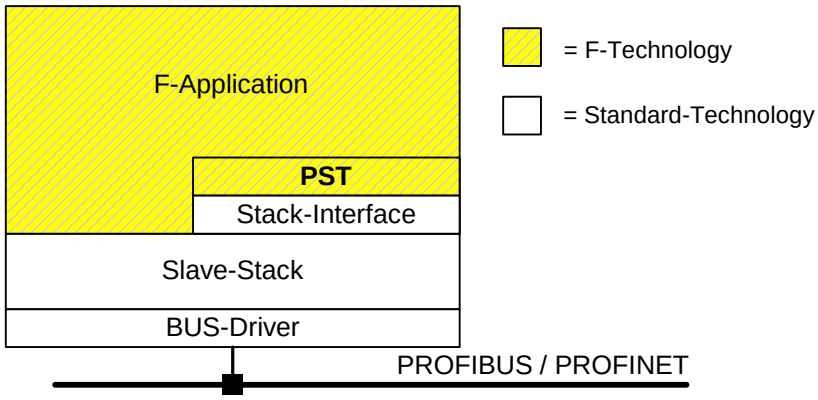

Fig. 3-2: Integration of the PSD (single-channel)

Slave-Stack: controls communication via PROFIBUS or PROFINET.

Stack-Interface: for adapting the PSD to different Slave-Stacks.

F-Application: specific device application that uses the PSD for communication via the PROFIBUS or PROFINET with cyclic F-telegrams.

Synchronization-Interface: for data exchange.

The tasks of the FW blocks defined below are intended as orientation for the design of an F-Slave.

3.1.1 Tasks of the Slave-Stack

The following tasks are handled by the Slave-Stack:

- The Slave-Stack receives F-Parameters and optional i-Parameters.
- The Slave-Stack receives the output telegram.
- The Slave-Stack transmits the input telegrams.
- The Slave-Stack recognized the BUS events such as "Start / stop of the cyclical data exchange".

3.1.2 Stack-Interface tasks

The following tasks are assigned to the Stack-Interface:

- The Stack-Interface extracts F-Parameters from the parameter telegram of the Slave-Stack and transfers them to the PSD.
- The Stack-Interface takes over the output telegrams from the Slave-Stack and transfers it to the PSD.
- On request, the Stack-Interface provides the PSD with a buffer to accept the input telegram.
- The Stack-Interface accepts the buffer with the input telegram.
- With the Slave-Stack, the Stack-Interface controls the transmission of the input telegram to the F-Host.

3.1.3 Tasks of F-Application

The implementation of the F-Application lies outside of the PSD and has to perform the following tasks:

- The F-Application controls the entire sequence.
- The F-Application handles the initialization of the Slave-Stack.
- The F-Application distributes (with dual-channel) the F-data packets to both channels with the help of the Synchronization-Interface.
- The F-Application takes over the output data and the control byte from the PSD and processes them.
- The F-Application generates the input data as well as certain bits for the status byte and transfers them to the PSD.

3.1.4 The Synchronization-Interface

The synchronization-sequence, which takes place in cyclic operation, is shown in the folling picture:

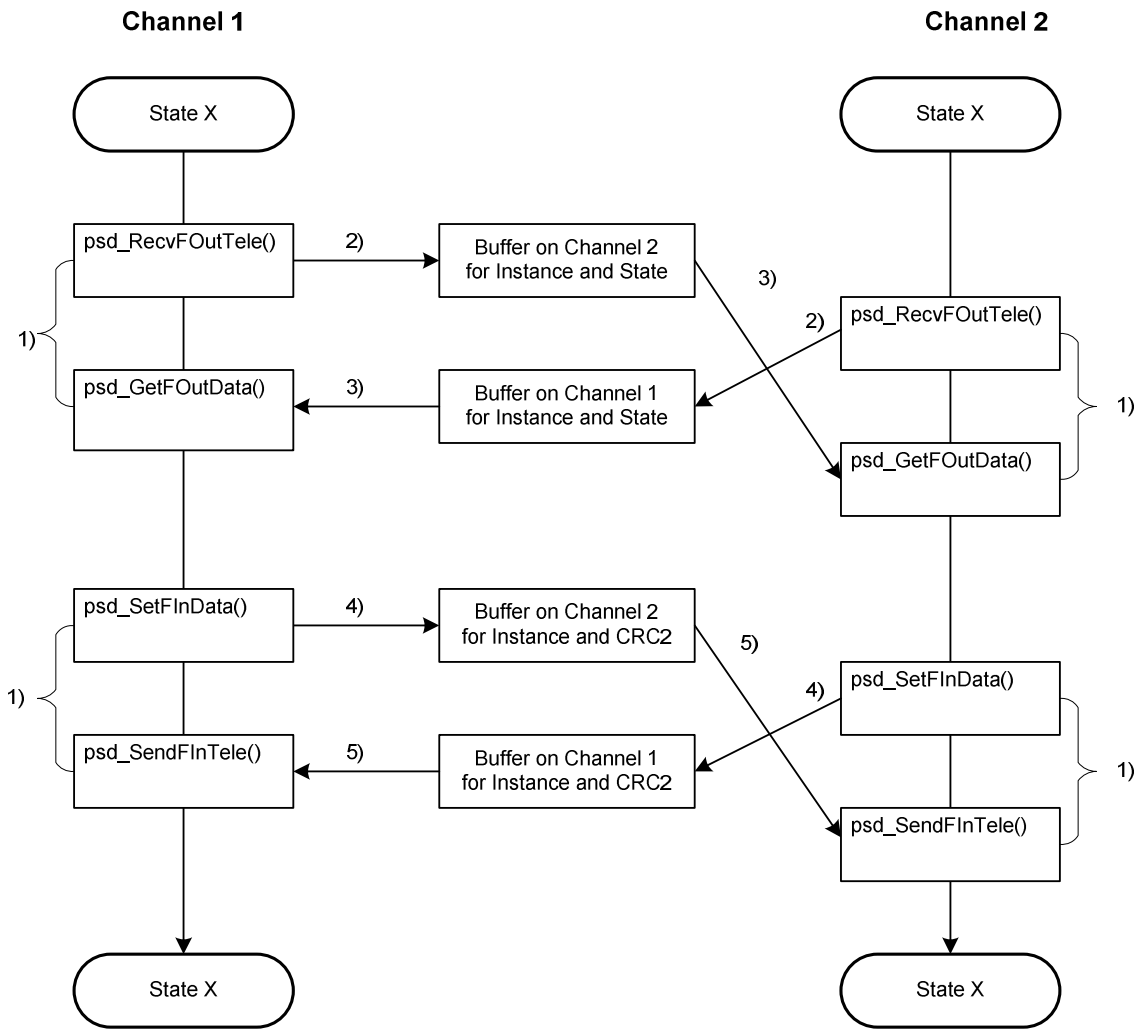

Fig. 3-3: Operation of the asynchron synchronization

1) = the calling order of the API-functions (see Chapter 5) is checked by the PSD.

2) = call of the API-function psd_OutTransfer(Instance, State),

3) = call of the API-function psd_OutSync(Instance, State),

4) = call of the API-function psd_InTransfer(Instance, CRC2),

5) = call of the API-function psd_InSync(Instance, CRC2)

In a first step, the PSD sends the synchronization data to the other channel.

In a second step, the PSD gets the meanwhile received synchronization data and compares it with the data of its own channel. A difference leads to the state PSD_HARD_FAIL.

After the transfer of the synchronization data to the PSD, the synchronization data have to be deleted.

The bufferd data have to be read out only one time (for example the bufferd data are deleted after read out).

3.2 Structure of an F-Application

In principle, the structure of an F-Application comprises the following states:

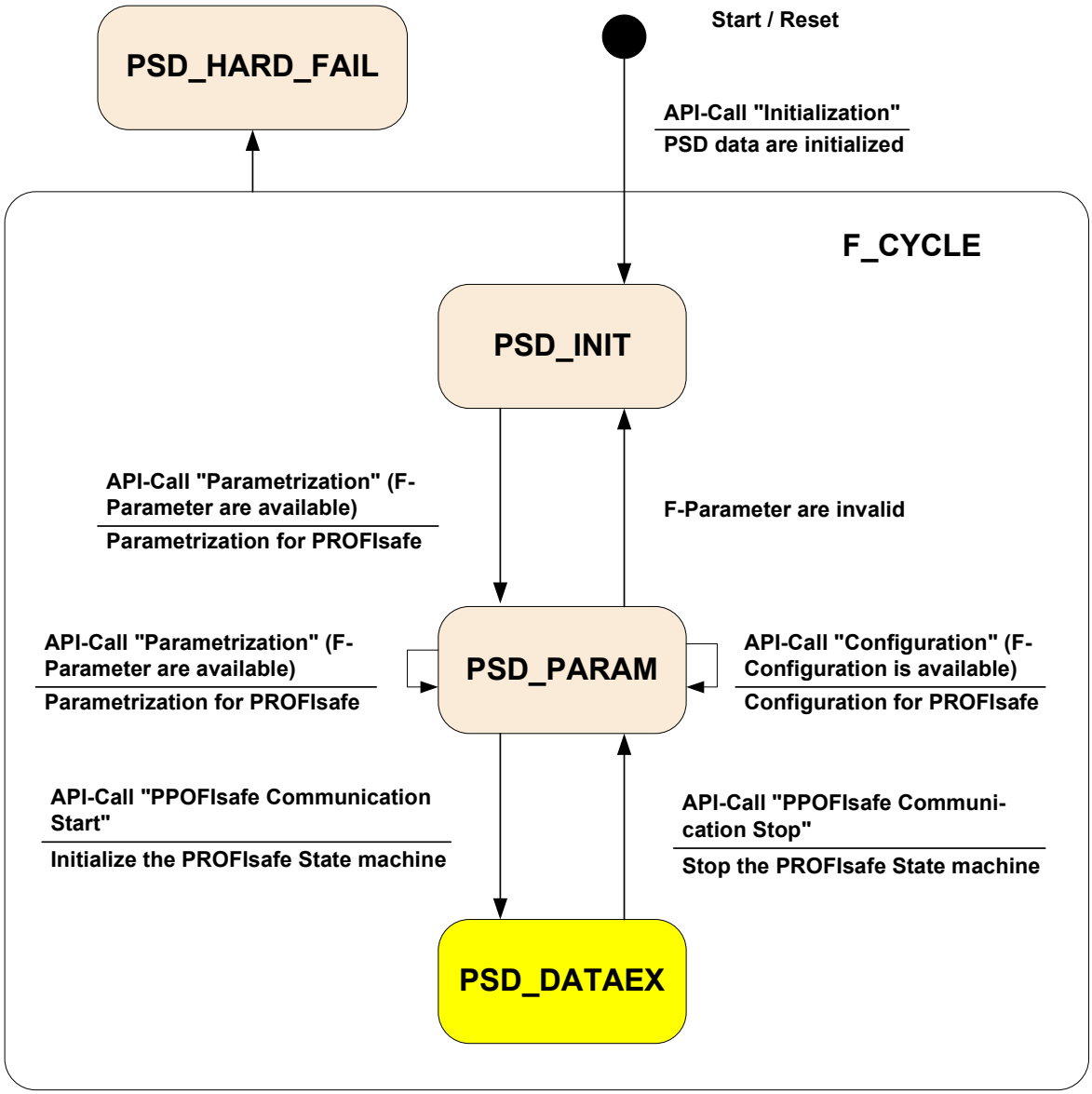

Fig. 3-4: Entire sequence of the PROFIsafe slave_FW

3.2.1 Description of the states

F_CYCLE
F-Application cycle.

PSD_INIT
PSD instance is initialized without error.

PSD_PARAM
Valid F-Parameters are available. The configuration of the user data length can take place in this status if it was set during initialization. The PSD instance is ready for cyclical data exchange.

PSD_DATAEX
The cyclical data exchange with the F-Host was started. The PSD instance function psd_Run(…) has been called and cyclical processing of the input/output telegrams has started.

PSD_HARD_FAIL
The PSD instance has detected an inconsistency probably due to a hardware problem. With the single-channel generation variant, the instance will set the interface variable “pSafetyVar” to the value “P_SAFETY_CONST_FALSE.” The PSD instance calls up the function "hard_err(…)" from the F-Application. The F-Application sets the safe status with this function.

3.2.2 Description of the state transitions

START→PSD_INIT
After PowerOn the F-Application initializes all instances of the PSD by starting psd_InitInstance(…). After an error-free run, the respective PSD instance will be in the PSD_INIT state.

F-CYCLE→PSD_HARD-FAIL
Every inconsistency or hardware error that is recognized by a PSD instance will cause the state PSD_HARD_FAIL.

PSD_INIT→PSD_PARAM
The F-Application Slave-Stack indicates the reception of F-Parameters with **[SLAVE_STACK_F_PAR_AVAILABLE]**.
After that, the F-Application assigns parameters to the PSD instance by starting the function psd_FParBuild(…). The status PSD_PARAM will be reached if valid F-Parameters are available.

PSD_PARAM→PSD_INIT
The psd_FParBuild(…) function has detected invalid F-Parameters.

PSD_PARAM→PSD_PARAM
After receiving new, valid F-Parameters from the Slave-Stack, the F-Application reassigns parameters to the PSD instance.

PSD_PARAM→PSD_PARAM
After receiving the user data length configuration, the PSD instance sets the length for the input and output user data.

PSD_PARAM→PSD_DATAEX
The Slave-Stack has indicated to the F-Application the transition into the cyclical data exchange with **[RUN]**. Subsequently, the F-Application starts the function psd_Run(…).

PSD_DATAEX→PSD_PARAM
The Slave-Stack has indicated to the F-Application the exit from the cyclical data exchange with **[STOP]**. Subsequently the F-Application starts the function psd_Stop(…).

3.2.3 Activities of the PSD in state PSD_DATAEX

3.2.3.1 Processing the output telegrams

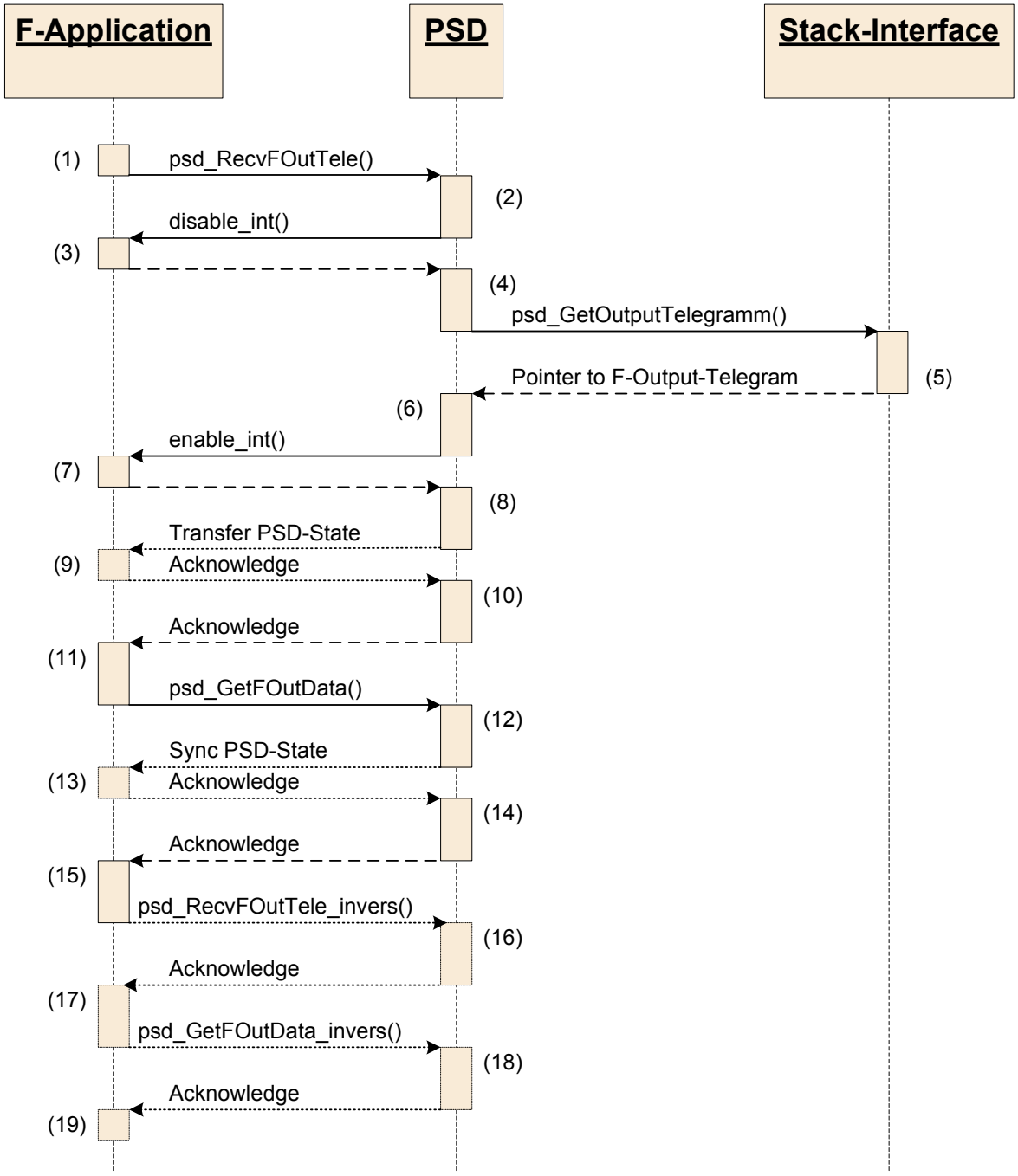

Fig. 3-5: Process the cyclical output telegram

(1) The F-Application starts the PSD function psd_RecvFOutTele(…) to accept and check an output telegram from the Stack-Interface and test it. It will also transfer its latest time values to the PSD.

(2) The PSD calculates the timeout situation based on the transferred time values. It will request an interrupt lock for consistent acceptance of the output telegram.

(3) Set the interrupt lock.

(4) The PSD requests a pointer to the buffer with the output telegram from the Slave-Stack-Interface.

(5) The Slave-Stack-Interface fills the buffer with the output telegram and transfers a pointer to the PSD.

(6) The PSD adopts the output telegram into its internal data structure. In case of single channel, the PSD will also fill the inverse data structure with the output telegram. It requests cancellation of the interrupt lock from the F-Application.

(7) Cancel the interrupt lock.

(8) The PSD checks the output telegram according to the PROFIsafe profile. In case of dual-channel, the PSD will makes its current status available to the other channel.

(9) The F-Application transfers the PSD status to the other channel in case of dual channel.

(10) The PSD exits the function psd_RecvFOutTele().

(11) The F-Application checks the return value of psd_RecvFOutTele(...). To receive the output data, it starts the function psd_GetFOutData(…).

(12) The PSD will request the status of the other channel from the F-Application in case of dual channel.

(13) The F-Application transfers the status of the other channel to the PSD in case of dual channel.

(14) The PSD compares its current status with the status of the other channel in case of dual channel. It then fills the buffer of the F-Application with the output data and the control byte.

(15) The F-Application checks the return value of psd_GetFOutData (…) and processes the output telegram. In the single-channel generating version, the “inverse component” of the F-Application will start the function psd_RecvFOutTele_invers(…).

(16) In case of single channel, inverse processing will take place with psd_RecvFOutTele().

(17) In case of single channel, the “inverse component” of the F-Application checks the return value of psd_RecvFOutTele_invers(). To receive the inverse output data, it starts the function psd_GetFOutData_invers().

(18) In case of single channel, inverse processing will take place with psd_GetFOutData_invers().

(19) In case of single channel, the “inverse component” of the F-Application checks the return value of psd_GetFOutData_Invers() and processes the inverse output telegram.

Safety information

Requirement: PSD_09

Invalid return values must lead to the F-Application state **HARD_FAIL**.

Safety information

Single channel generation variant

Requirement: PSD_18

The F-Application has to test the consistency of the normal and inverse F-output user data as well as control byte IDs. The consistency ckeck has to be done before output to the process.
If the PSD reports a communication error, the received control byte is invalid and the evaluation of the control byte IDs isn´t permissible.

Safety information

Single channel generation variant

Requirement: PSD_22

When using the single channel-PSD in an actor or sensor, all safety critical secondary faults have to be uncovered by the surrounding of the PSD.

Safety information

Single channel generation variant

Requirement: PSD_23

When using the single channel-PSD in an actor, accidental hardware-faults have to be uncoverd within the process-fault-tolerance time by the surroundings.

Safety information

Dual channel generation variant

Requirement: PSD_24

The F-Application has to test the consistency of the F-output user data as well as control byte IDs of both hardware channels. The consistency ckeck has to be done before output to the process.
If the PSD reports a communication error, the received control byte is invalid and the evaluation of the control byte IDs isn´t permissible.

3.2.3.2 Processing the input telegrams

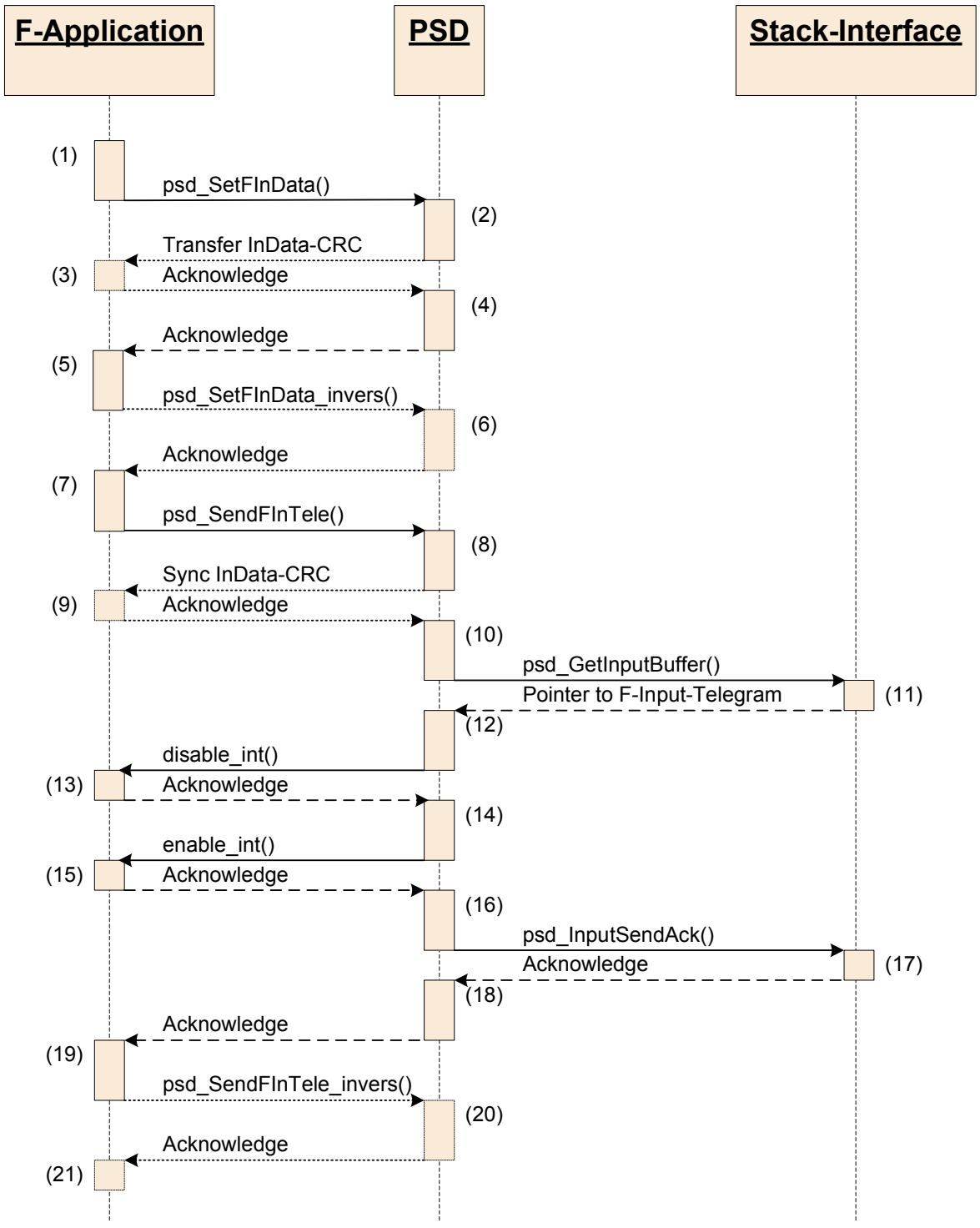

Fig. 3-6: Processing the cyclical input telegrams

(1) To transfer the input data, the F-Application starts the function psd_SetFInData().

(2) The PSD reads the input data and certain bits for the structure of the status-byte and starts creating the input telegram.

(3) In case of dual channel, the PSD will make the CRC of the input telegram available to the other channel.

(4) The PSD exits the function psd_SetFInData().

(5) The F-Application evaluates the return value of psd_SetFInData(). In case of single channel, the “inverse component” of the F-Application starts the function psd_SetFInData_invers().

(6) In the single-channel generation version, the PSD processes the inverse function of psd_SetFInData().

(7) In case of single channel, the F-Application evaluates the return value of psd_SetFInData_invers(). The F-Application starts the function psd_SendFInTele() to send the input telegram.

(8) The PSD creates the input telegram according to the PROFIsafe profile. It requests the CRC of the other channel in case of dual channel and completes the input telegram. In case of single channel, the PSD will use the CRC of its inverse processing.

(9) In case of dual channel, the F-Application will provide the CRC of the other channel to the PSD.

(10) The PSD uses the function psd_GetInputBuffer() to request a buffer for the input telegram from the Stack-Interface.

(11) The Stack-Interface returns a pointer to the buffer for the input telegram to the PSD.

(12) The PSD request an interrupt lock from the F-Application for consistent filling of the send buffer.

(13) The F-Application sets an interrupt lock.

(14) The PSD fills the send buffer and then requests release of the interrupt lock from the F-Application.

(15) The F-Application releases the interrupt lock.

(16) The PSD uses the function psd_InputSendAck() to prompt the Stack-Interface to send the input telegram to the F-Host.

(17) The Stack-Interface sends the input telegram.

(18) The PSD checks the return value of psd_InputSendAck() and exits the function psd_SendFInTele().

(19) In case of single channel, the “inverse component” of the F-Application starts the inverse processing of psd_SendFInTele().

(20) In case of single channel, the PSD processes the function psd_SendFInTele_invers().

(21) In case of single channel, the “inverse component” of the F-Application checks the return value of psd_RecvFOutTele_invers().

Safety information

Requirement: PSD_09

Invalid return values must lead to the F-Application state **HARD_FAIL**.

4 Files of the PSD

The following tables list all the files that can be found on the CD-ROM for the PSD. The files include a description of the interface functions, which are explained briefly in the table below. A detailed description of these interfaces is available in the following chapters.

4.1 Files that must not be changed

File	Interface function	Task
p_crc.c	psd_CRC16(…)	Calculation of a 16-bit CRC via a memory section
	psd_CRC32(…)	Calculation of a 32-bit CRC via a memory section
p_init.c	psd_InitInstance(…)	Initialize a PSD instance
p_state.c	psd_GetState(…)	Get PSD state
	psd_Stop(…)	Stop cyclical data exchange of the PSD
	psd_Run(…)	PSD is in cyclical data exchange
p_input.c	psd_SendFInTele(…)	Transfer F input telegram to Stack-Interface
	psd_SetFInData(…)	Transfer F input data to PSD
p_output.c	psd_GetFOutData(…)	Transfer F output data to F-Application
	psd_RecvFOutTele(…)	Check F output telegram
p_param.c	psd_GetFPar(…)	Read F_SIL and F_iPar_CRC
	psd_FParBuild(…)	Check F-Parameters
	psd_Config(…)	Set user data length
p_time.c	-	Internal timer management
p_types.h	-	Declaration of higher-level data types
p_api.h	-	Interface eclaration
p_crc.h	-	Local type declaration for p_crc.c
p_global.h	-	Global type declaration for F-Application
p_init.h	-	Local type declaration for p_init.c
p_input.h	-	Local type declaration for p_input.c
p_invers.h	-	Inverse type declarations for single channel generation version of the PSD
p_local.h	-	PSD local type declarations
p_output.h	-	Local type declaration for p_output.c
p_param.h	-	Local type declaration for p_param.c
p_state.h	-	Local type declaration for p_state.c
p_time.h	-	Local type declaration for p_time.c

Additionally for single-channel generation version

File	Interface function	Task
p_invers.c	psd_GetFOutData_Invers(…)	Returned value of the inverse calculation path of the function psd_GetFOutData(...)
	psd_RecvFOutTele_Invers(…)	Returned value of the inverse calculation path of the function psd_RecvFOutTele (…)
	psd_SendFInTele_Invers (…)	Returned value of the inverse calculation path of the function psd_SendFInTele (...)
	psd_SetFInData_Invers (…)	Returned value of the inverse calculation path of the function psd_SetFInData(...)
	psd_InitInstance_Invers (…)	Returned value of the inverse calculation path of the function psd_InitInstance (...)
	psd_FParBuild_Invers (…)	Returned value of the inverse calculation path of the function psd_FParBuild (...)
	psd_Config_Invers (…)	Returned value of the inverse calculation path of the function psd_Config (...)
	psd_GetState_Invers (…)	Returned value of the inverse calculation path of the function psd_GetState (...)
	psd_Stop_Invers (…)	Returned value of the inverse calculation path of the function psd_Stop (...)
	psd_Run_Invers (…)	Returned value of the inverse calculation path of the function psd_Run (...)
	psd_GetFPar_Invers (…)	Returned value of the inverse calculation of the function psd_GetFPar (...)

Safety information

Requirement: PSD_02

The PSD files must not be changed.

The tool **CRC-Check** is available to verify that the PSD sources used have not been changed. The tool forms a signature over each PSD file that must not be changed as well as an overall signature over all PSD sources.

For installation and use of the tool **CRC-Check**, read and comply with the information in the ReadMe file in the directory <CD>:\CRC_Check\ .

The following diagram shows the corresponding CRC values for the source files. These CRC values must be compared to the currently calculated CRC values.

No.	File	CRC
1	D:\F-Treiber\Source\crc_gen\p_crc.c	919E 3169
2	D:\F-Treiber\Source\crc_gen\p_init.c	2A57 9023
3	D:\F-Treiber\Source\crc_gen\p_input.c	AE6B 861C
4	D:\F-Treiber\Source\crc_gen\p_invers.c	C040 CE69
5	D:\F-Treiber\Source\crc_gen\p_output.c	79C2 9873
6	D:\F-Treiber\Source\crc_gen\p_param.c	D4B C458
7	D:\F-Treiber\Source\crc_gen\p_state.c	D890 7758
8	D:\F-Treiber\Source\crc_gen\p_time.c	9A8C E756
9	D:\F-Treiber\Source\crc_gen\p_api.h	69BD FD35
10	D:\F-Treiber\Source\crc_gen\p_crc.h	E622 D32A
11	D:\F-Treiber\Source\crc_gen\p_global.h	60BF 943C
12	D:\F-Treiber\Source\crc_gen\p_init.h	1163 BC2A
13	D:\F-Treiber\Source\crc_gen\p_input.h	4EC3 334B
14	D:\F-Treiber\Source\crc_gen\p_invers.h	C33D 1266
15	D:\F-Treiber\Source\crc_gen\p_local.h	A318 CE4B
16	D:\F-Treiber\Source\crc_gen\p_output.h	E28B 1AD9
17	D:\F-Treiber\Source\crc_gen\p_param.h	BFCB E84F
18	D:\F-Treiber\Source\crc_gen\p_state.h	1AA1 0C28
19	D:\F-Treiber\Source\crc_gen\p_time.h	DCEE CBC7
20	D:\F-Treiber\Source\crc_gen\p_types.h	4C9E 87D9
21	Total:	6C81 A67B

Fig. 4-1: CRC-values of the certified PSD-sources

Safety information

Requirement: PSD_03

The CRC values of the PSD files (individual and overall CRC) must match the values of the CRC check tool.

4.2 Files that may be adapted

File	Task
p_c_config.h	Definition of generation parameters
p_c_crc_m.h	Memory allocation of CRC table
p_c_crc_b.h	Restoring default memory model
p_c_fapplication.h	Adaptation level for F-Application
p_c_si.c	Functions of stack inferface
p_c_si.h	Function prototypes of stack inferface

4.3 Procedure for startup

The following steps are to be taken to use PSD:

- Implement surrounding FW modules (Stack-Interface , Synchronization-Interface, F-Application).
- Provide the functions required by the PSD (output macros) and match to prototype, which are available in p_c_fapplication.h.

- Adapt the generation parameters in p_c_Config.h.
- Include p_api.h in the surrounding FW modules.
- Adapt the desired memory model, the names of the memory classes in p_c_config or the memory segmentation in p_c_crc_m.h, p_c_crc_b.h. Each of the settings is also activated in p_c_config.h via Defines.

Note

A pseudo code /4/ is enclosed to illustrate the PSD integration.
(Files: p_c_pseudo_1c.c or p_c_pseudo_2c.c).

---

5 Interface description

Function interface

The PSD can be used in different system environments. The files described in chapter 4 are available for this purpose.

Data interface

The PSD uses buffers located in a separate memory section that is created and managed statically by the PSD to store the input and output telegrams.

Memory attributes and optional compiler instructions for memory segment control can be used to manage the storage location according to the tool chain.

Safety information

Requirement: PSD_04

Only the PSD interface functions declared in the file p_api.h may be used.

Safety information

Requirement: PSD_05

Only the PSD interface functions declared in the file p_global.h may be used.

Safety information

Requirement: PSD_06

The defines from p_global.h must be used for evaluation of the PSD returns.

Safety information

Requirement: PSD_09

Invalid return values must lead to the F-Application state **HARD_FAIL.**

5.1 Necessary PSD interface functions

The functions described below are to be integrated in the F-Application as described in chapter 3.2.

The instance number calculation from the bus telegrams is bus specific. For PROFINET for example, the instance number is calculated with the received submodul number.

For the adressing of an instance, the defines "INST_xx" can be used. The structure of each delivered instance number "pInstance" to the PSD is:

16Bit-value	Bit 0- 7:	(1th-complement of instance number) - 1
	Bit 8-15:	instance number - 1

example for instance 1: 0x00FF
example for instance 2: 0x01FE (see also implementation in p_global.h)

5.1.1 psd_InitInstance(…)

When this function is started, the PSD is given the information for the indicated instance whether or not its data length is going to be configured dynamically.

Function:	**psd_InitInstance(…)**	
Prototype:	*P_Byte **psd_InitInstance*** *(P_Word pInstance,* *P_Byte pDynamicTeleLenConf);*	
Meaning:	• Initialize PSD instance • Check the transfer parameters	
Transfer parameters:	**Value range**	**Meaning**
pInstance	INST_01…INST_X X <= NOF_INSTANCES	Instance ID
pDynamicTeleLenConf	DYNAMIC_TELE_LEN_CONF_ON	The maximally values are included in p_c_config.h. The latest data length of the instance are read in via the function psd_GetConfigPtr(…).
	DYNAMIC_TELE_LEN_CONF_OFF	Data lengths of the instance are determined in p_c_config.h.
Return parameter:	P_Byte	
Value range		**Meaning**
INIT_INSTANCE_OK		Initialization successful

Safety information

From the point of view of the F-Application only **INIT_INSTANCE_OK** is permitted.

Requirement: PSD_09

Invalid return values must lead to the F-Application state **HARD_FAIL**

5.1.2 psd_FParBuild(…)

When this function is started, the PSD is given the F-Address (determined independently of the F-Parameters) and the maximum SIL of the F-Application (stored permanently in the F-Application) as well as the iPar-CRC (CRC3). The value range of the transferred parameters is checked by the PSD.
As part of this function, the PSD will retrieve the F-Parameters with the function psd_GetFParPtr (…), checks the CRC via the parameter block and executes a plausibility check. The acceptance of the F-Parameters is paranthesized with the functions disable_int(…) and enable_int(…) to guarantee consistency during parameter acceptance.

Function:	**psd_FParBuild(…)**
Prototype:	*P_Byte **psd_FParBuild*** *(P_Word pInstance,* *Fw_FPar pFwPar,* *P_Word psd_CRC3);*
Meaning:	• Configure PSD for instance • Check transfer parameters • Reading and checking F-Parameters

Transfer parameters:	Value range	Meaning
pInstance	INST_01…INST_X X <= NOF_INSTANCES	Instance ID
pFwPar	Fw_FPar	Device parameter
psd_CRC3	0 - 0xFFFF	Transfer of i-Par-CRC (CRC3) If “F_Check_iPar“=1, then iPar-CRC will be integrated into the calculation of FPar-CRC (CRC1). In this case, psd_CRC3 must not be "0". If "F_Check_iPar" = 0 or "F_Block_ID" = "001", psd_CRC3 is not evaluated. (See PROFIsafe profile /1/, chap. 8.1.4.3 and chap. 8.1.5.2)
Return parameter:	P_Byte	

Value range	Meaning
FPAR_BUILD_OK	Parameter assignment successful
FPAR_F_DEST_ADD_MISMATCH	Incorrect target address
FPAR_F_DEST_ADD_NOT_VALID	Invalid target address (0 or 0xFFFF)
FPAR_F_SOURCE_ADD_NOT_VALID	Invalid source address (0 or 0xFFFF)
FPAR_F_WD_TIME_NULL	Watchdog time is invalid (==0)
FPAR_F_SIL_ERR	Transferred SIL class too high
FPAR_CRC1_ERR	F-Par-CRC – error
FPAR_BUILD_NOT_OK	Inconsistent parameter assignment

Structure of the device parameters:

With dual channel generation version

```
typedef struct Fw_FPar_t {
  P_Word      FDestAdd
  P_Byte      FSil
} Fw_FPar;
```

With single channel generation version

```
typedef struct Fw_FPar_t {
  P_Word      FDestAdd
  P_Word      FDestAdd_inv
  P_Byte      FSil
  P_Byte      FSil_inv
} Fw_FPar;
```

Safety information

From the point of the F-Application only

- **FPAR_BUILD_RETURN_OK**
- **FPAR_F_DEST_ADD_MISMATCH**
- **FPAR_F_DEST_ADD_NOT_VALID**
- **FPAR_F_SOURCE_ADD_NOT_VALID**
- **FPAR_F_WD_TIME_NULL**
- **FPAR_F_SIL_ERR**
- **FPAR_CRC1_ERR**
- **FPAR_BUILD_NOT_OK**

are permitted.

Requirement: PSD_09

Invalid return values must lead to the F-Application state **HARD_FAIL**.

Safety information

Requirement: PSD_11

The SIL must correspond to the SIL of the F-Slave and must be determined independent of the F-Parameters.

Safety information

Requirement: PSD_12

The F-Address must be stored in the F-Application. An extraction of the F-Address or other values from the F-Parameters is not permitted.

5.1.3 psd_Config(…)

The PSD in this function sets the data length of the instance, which it either (depending on the setting of this instance) reads in via the function psd_GetConfigPtr(…) or adopts the length directly from the settings of the file p_c_config.h. The acceptance of the data lengths is paranthesized with the functions disable_int(…) and enable_int(…) to guarantee consistency.

Function:	**psd_Config(…)**	
Prototype:	*P_Byte **psd_Config** (P_Word pInstance);*	
Meaning:	• Setting data length of PSD instance	
Transfer parameters:	**Value range**	**Meaning**
pInstance	INST_01…INST_X X <= NOF_INSTANCES	Instance ID
Return parameter:	P_Byte	
Value range		**Meaning**
FPAR_CONFIG_OK		Configuration successful
FPAR_CRC_LENGTH		Invalid length information

Safety information

From the point of the F-Application only

- **FPAR_CONFIG_OK**
- **FPAR_CRC_LENGTH**

are permitted.

Requirement: PSD_09

Invalid return values must lead to the F-Application state **HARD_FAIL.**

5.1.4 psd_Run(…)

The function switches the PSD into the cyclical data exchange and returns the internal state of the PSD.

Function:	**psd_Run(…)**	
Prototype:	*P_Byte **psd_Run** (P_Word pInstance);*	
Meaning:	Place PSD instance in cyclical data exchange	
Transfer parameters:	**Value range**	**Meaning**
pInstance	INST_01…INST_X X <= NOF_INSTANCES	Instance ID
Return parameter:	P_Byte	
Value range	**Meaning**	
PSD_DATAEX	Error-free cyclical data traffic	

Safety information

From the point of view of the F-Application only **PSD_DATAEX** is permitted.

Requirement: PSD_09

Invalid return values must lead to the F-Application state **HARD_FAIL.**

5.1.5 psd_Stop(…)

The function is intended to terminate the cyclical data exchange of the PSD and permit parameterization.

Function:	**psd_Stop(…)**	
Prototype:	*P_Byte **psd_Stop** (P_Word pInstance);*	
Meaning:	Terminate cyclical data exchange in PSD instance	
Transfer parameters:	**Value range**	**Meaning**
pInstance	INST_01…INST_X X <= NOF_INSTANCES	Instance ID
Return parameter:	P_Byte	
Value range	**Meaning**	
PSD_PARAM	Assigned	

Safety information

From the point of view of the F-Application only **PSD_PARAM** is permitted.

Requirement: PSD_09

Invalid return values must lead to the F-Application state **HARD_FAIL.**

5.1.6 psd_RecvFOutTele(…)

The function transfers the time values in 1ms intervals and accepts the output telegram from the F-Master in the PSD for the indicated instance. The PSD reads in the output data via the function psd_GetOutputTelegram(..). The acceptance of the output data is parenthesized with the functions disable_int(…) and enable_int(…) to guarantee consistency during parameter acceptance.

Function:	**psd_RecvFOutTele(...)**	
Prototype:	*P_Byte **psd_RecvFOutTele*** *(P_Word pInstance, P_Word ClockValue, P_Word ClockValueC2);*	
Meaning:	Acceptance of 1ms increments and the cyclical output telegram in the PSD_zykout buffer.	
Transfer parameters:	**Value range**	**Meaning**
pInstance	INST_01…INST_X X <= NOF_INSTANCES	Instance ID
ClockValue		1msec-clock value
ClockValue2		1msec-clock value
Return parameter:	P_Byte	
Value range		**Meaning**
RECV_FOUT_TELE_OK		Output telegram adopted

Safety information

From the point of view of the F-Application only **RECV_FOUT_TELE_OK** is permitted.

Requirement: PSD_09

Invalid return values must lead to the F-Application state **HARD_FAIL.**

5.1.7 psd_RecvFOutTele_Invers(...)

This function is to be used in the single-channel version. The function transfers the inverse time values in 1ms intervals and creates the inverse data in the internal memory of the PSD for the indicated instance.

Function:	**Psd_RecvFOutTele_Invers(...)**	
Prototype:	*P_Byte **psd_RecvFOutTele_invers*** *(P_Word pInstance, P_Word ClockValue_inv, P_Word ClockValueC2_inv);*	
Meaning:	Acceptance of the inverse 1ms increments and buildup of the inverse data in the PSD buffer.	
Transfer parameters:	**Value range**	**Meaning**
pInstance	INST_01…INST_X X <= NOF_INSTANCES	Instance ID
ClockValue_inv		Inverter 1msec-clock value
ClockValue2_inv		Inverter 1msec-clock value
Return parameter:	P_Byte	
Value range		**Meaning**
RECV_FOUT_TELE_OK_INV		Inverse-data created

Safety information

From the point of view of the F-Application only **RECV_FOUT_TELE_OK_INV** is permitted.

Requirement: PSD_09

Invalid return values must lead to the F-Application state **HARD_FAIL.**

Safety information

Requirement: PSD_19

The returned values of the 'inverse' functions used are to be evaluated accordingly.

5.1.8 psd_GetFOutData(…)

The function is used to check the output telegrams from the master according to the indicated PROFIsafe profile /1/. The F-Application is provided with the F-output user data, the control byte, and the error status of the function.

Function:	**psd_GetFOutData(…)**	
Prototype:	*P_Byte* ***psd_GetFOutData*** (P_Word pInstance, Fw_OutputDataCB P_GLOBDAT_ATTR_POINTER* FFwOutputDataCB, P_Byte P_GLOBDAT_ATTR_POINTER* pOutDataArray, P_Byte pOutDataLen);	
Meaning:	Checks the output telegram, forms the F-output user data as well as the control byte	
Transfer parameters:	**Value range**	**Meaning**
pInstance	INST_01…INST_X X <= NOF_INSTANCES	Instance ID
FFwOutputDataCB,		Pointer to a structure to store control byte IDs
pOutDataArray		Pointer to a data area for storage of user data
pOutDataLen		Length of the user data area
Return parameter:	P_Byte	

Value range	Meaning
F_OUTPUT_OK	F telegram from master is valid and the F-output user data can be adopted by the F-Application.
F_OUTPUT_OLD_CONSNR	
F_OUTPUT_CLEAR	The PSD has received a zero-telegram from the F-Host. This telegram will be discarded but time monitoring will continue. The F-output user data will remain unchanged. The F-Application can retain its last valid output user data.
F_OUTPUT_COMM_ERR	An error in the PROFIsafe communication was detected. The F-output user data are passivated.
F_OUTPUT_WD_TIMEOUT	A timeout was detected in the communication. The F-output user data are passivated.
F_OUTPUT_PASSIVATED	An error was detected in the communication or the F-Host has set the bit_4 (FV-to-be-activated) to 1 in the control byte. The F-output user data are passivated.

Safety information

From the point of view of the F-Application for the return values:
F_OUTPUT_COMM_ERR,
F_OUTPUT_WD_TIMEOUT,
F_OUTPUT_PASSIVATED
you will have to set the F-output user data according to the application into the safe state.

Requirement: PSD_13

For:
F_OUTPUT_COMM_ERR,
F_OUTPUT_WD_TIMEOUT,
F_OUTPUT_PASSIVATED
the F-Application must ensure the safe state for the process.

Safety information

From the point of view of the F-Application, only

- **F_OUTPUT_OK**
- **F_OUTPUT_OLD_CONSNR**
- **F_OUTPUT_CLEAR**
- **F_OUTPUT_COMM_ERR**
- **F_OUTPUT_WD_TIMEOUT**
- **F_OUTPUT_PASSIVATED**

are permitted.

Requirement: PSD_09

Invalid return values must lead to the F-Application state **HARD_FAIL.**

Control byte ID	
typedef *struct* Fw_OutputDataCB_t {	
P_Byte	CB_B0_iParEN
P_Byte	CB_B1_OAReq
P_Byte	CB_B4_activate_FV
} Fw_OutputDataCB;	

Safety information

Dual channel generation variant

Requirement: PSD_24

The F-Application has to test the consistency of the F-output user data as well as control byte IDs of both hardware channels. The consistency ckeck has to be done before output to the process.
If the PSD reports a communication error, the received control byte is invalid and the evaluation of the control byte IDs isn´t permissible.

5.1.9 psd_GetFOutData_Invers(…)

This function is to be used in the single-channel version. The function provides the F-Application with the inverse F-output user data, the inverse control byte, and the inverse error status of the indicated instance.

Function:	**psd_GetFOutData_Invers(…)**	
Prototype:	*P_Byte* ***psd_GetFOutData_invers*** (P_Word pInstance, Fw_OutputDataCB P_GLOBDAT_ATTR_POINTER* FFwOutputDataCB_inv, P_Byte P_GLOBDAT_ATTR_POINTER* pOutDataArray_inv, P_Byte pOutDataLen);	
Meaning:	Checks the output telegram, forms the F output user data as well as the control byte	
Transfer parameters:	**Value range**	**Meaning**
pInstance	INST_01…INST_X X <= NOF_INSTANCES	Instance ID
FFwOutputDataCB_inv		Pointer to a structure to store inverse control byte IDs
pOutDataArray_inv		Pointer to a data area for storage of inverse user data
pOutDataLen		Length of the user data area
Return parameter:	P_Byte	

Value range	Meaning
F_OUTPUT_OK_INV	F telegram from master is valid and the inverse F output user data can be adopted by the F-Application.
F_OUTPUT_OLD_CONSNR_INV	
F_OUTPUT_CLEAR_INV	The PSD has received a zero-telegram from the F-Host. This telegram will be discarded but time monitoring will continue. The F output user data will remain unchanged. The F-Application can retain its last valid output user data.
F_OUTPUT_COMM_ERR_INV	An error in the PROFIsafe communication was detected. The inverse F output user data are passivated.
F_OUTPUT_WD_TIMEOUT_INV	A timeout was detected in the communication. The inverse F output user data are passivated.
F_OUTPUT_PASSIVATED_INV	An error was detected in the communication or the F-Host has set the bit_4 (FV-to-be-activated) to 1 in the control byte. The inverse F output user data are passivated.

Safety information

From the point of view of the F-Application for the return values:
F_OUTPUT_COMM_ERR_INV,
F_OUTPUT_WD_TIMEOUT_INV,
F_OUTPUT_PASSIVATED_INV
you will have to set the F output user data according to the application into the safe state.

Requirement: PSD_13

For:
F_OUTPUT_COMM_ERR,
F_OUTPUT_WD_TIMEOUT,
F_OUTPUT_PASSIVATED
the F-Application must ensure the safe state for the process

Safety information

From the point of view of the F-Application, only

- **F_OUTPUT_OK_INV**
- **F_OUTPUT_OLD_CONSNR_INV**
- **F_OUTPUT_CLEAR_INV**
- **F_OUTPUT_COMM_ERR_INV**
- **F_OUTPUT_WD_TIMEOUT_INV**
- **F_OUTPUT_PASSIVATED_INV**

are permitted.

Requirement: PSD_09

Invalid return values must lead to the F-Application state **HARD_FAIL**.

Control byte ID	
typedef *struct* Fw_OutputDataCB_t {	
P_Byte	CB_B0_iParEN
P_Byte	CB_B1_OAReq
P_Byte	CB_B4_activate_FV
} Fw_OutputDataCB_inv;	

Safety information

Requirement: PSD_19

The returned values of the 'inverse' functions used are to be evaluated accordingly.

Safety information

Single channel generation variant

Requirement: PSD_18

The F-Application has to test the consistency of the normal and inverse F-output user data as well as control byte IDs. The consistency ckeck has to be done before output to the process.
If the PSD reports a communication error, the received control byte is invalid and the evaluation of the control byte IDs isn´t permissible

5.1.10 psd_SetFInData(…)

The function takes over the status IDs and the F-input user data of the indicated instance in a PSD internal buffer from the F-Application.

Function	**psd_SetFInData()**	
Prototype:	*P_Byte* ***psd_SetFInData*** *(*P_Word pInstance, const Fw_InputtDataSB P_GLOBDAT_ATTR_POINTER* FFwInputDataSB, const P_Byte P_GLOBDAT_ATTR_POINTER* pInDataArray, P_Byte pInDataLen);	
Meaning:	Transfer of F input user data as well as status byte IDs to an internal PSD buffer	
Transfer parameters:	**Value range**	**Meaning**
pInstance	INST_01…INST_X X <= NOF_INSTANCES	Instance ID
FFwInDataSB		Pointer to a structure to take over control byte IDs
pInDataArray		Pointer to a data area to take over user data
pInDataLen		Length of the input user data
Return parameter:	P_Byte	
Value range		**Meaning**
SEND_FIN_DATA_OK		Data are adopted

Safety information

From the point of view of the F-Application only **SEND_FIN_DATA_OK** is permitted.

Requirement: PSD_09

Invalid return values must lead to the F-Application state **HARD_FAIL.**

Status byte ID	
typedef *struct* Fw_InputDataSB_t {	
P_Byte	SB_B0_iParOk
P_Byte	SB_B1_DeviceFault
P_Byte	CB_B4_FVactivated
} Fw_InputDataSB;	

5.1.11 psd_SetFInData_invers(…)

This function is to be used in the single-channel version. The function takes over the inverse F-input user data as well as the status byte IDs of the indicated instance in a PSD internal buffer from the F-Application.

Function	**psd_SetFInData_invers()**	
Prototype:	*P_Byte* ***psd_SetFInData_invers*** *(*P_Word pInstance, const Fw_InputtDataSB P_GLOBDAT_ATTR_POINTER* FFwInputDataSB_inv, const P_Byte P_GLOBDAT_ATTR_POINTER* pInDataArray_inv, P_Byte pInDataLen);	
Meaning:	Transfer of F input user data as well as status byte IDs to an internal PSD buffer	
Transfer parameters:	**Value range**	**Meaning**
pInstance	INST_01…INST_X X <= NOF_INSTANCES	Instance ID
FFwInDataCB_inv		Pointer to a structure to take over the inverse control byte IDs
pInDataArray_inv		Pointer to a data area to take over the inverse input user data
pInDataLen		Length of the input user data
Return parameter:	P_Byte	
Value range		**Meaning**
SEND_FIN_DATA_OK_INV		Data are adopted

Safety information

From the point of view of the F-Application only **SEND_FIN_DATA_OK_INV** is permitted.

Requirement: PSD_09

Invalid return values must lead to the F-Application state **HARD_FAIL.**

Safety information

Requirement: PSD_19

The returned values of the 'inverse' functions used are to be evaluated accordingly.

5.1.12 psd_SendFInTele(…)

The function compiles the input telegram of the indicated instance from the F input user data as well as status byte IDs transferred from the F-Application and passes this to the Stack-Interface .

Function:	**psd_SendFInTele()**	
Prototype:	*P_Byte **psd_SendFInTele**(P_Word pinstance);*	
Meaning:	Generate and send the input telegram	
Transfer parameters:	**Value range**	**Meaning**
pInstance	INST_01…INST_X X <= NOF_INSTANCES	Instance ID
Return parameter:	P_Byte	
Value range		**Meaning**
SEND_FIN _TELE_OK		Input telegram sent

Safety information

From the point of view of the F-Application only **SEND_FIN_TELE_OK** is permitted.

Requirement: PSD_09

Invalid return values must lead to the F-Application state **HARD_FAIL.**

5.1.13 psd_SendFInTele_invers(…)

This function is to be used in the single-channel version. The function compiles the inverse input telegram of the indicated instance from the F input user data as well as status byte IDs transferred from the F-Application.

Function:	**psd_SendFInTele_invers()**	
Prototype:	*P_Byte **psd_SendFInTele_invers**(P_Word pinstance);*	
Meaning:	Generate and send the input telegram	
Transfer parameters:	**Value range**	**Meaning**
pInstance	INST_01…INST_X X <= NOF_INSTANCES	Instance ID
Return parameter:	P_Byte	
Value range		**Meaning**
SEND_FIN _TELE_OK_INV		Input telegram sent

Safety information

From the point of view of the F-Application only **SEND_FIN_TELE_OK_INV** is permitted.

Requirement: PSD_09

Invalid return values must lead to the F-Application state **HARD_FAIL.**

Safety information

Requirement: PSD_19

The returned values of the 'inverse' functions used are to be evaluated accordingly.

5.2 Necessary PSD-Interface variable

5.2.1 pSafetyVar

The interface variable **pSafetyVar** enables the F-Application to report a HARD_FAIL to the PSD in single channel. This leads to a CRC falsification of the input telegram.

Safety information

Requirement: PSD_20

If the F-Application goes into the state **HARD_FAIL,**, the falsification variable **PSD_safety_var** must be set to the value **P_SAFETY_CONST_FALSE.**

5.3 Necessary functions to be implemented

5.3.1 hard_err(…)

In case of an error, the PSD sets the PSD status PSD_HARD_FAIL. The PSD then calls up the function hard_err(…) in p_c_fapplication.c so that the F-Application can respond to the error.

Function:	hard_err(...)	
Prototype:	*void* ***hard_err*** *(P_Word pInstance);*	
Meaning:	Specific reaction of the F-Application to fatal errors.	
Transfer parameters:	**Value range**	**Meaning**
pInstance	INST_01…INST_X X <= NOF_INSTANCES	Instance ID
Return parameter:	**Value range**	**Meaning**
none		

Safety information

Requirement: PSD_10

The function hard_err(...) is to be implemented by the F-Application. It must lead for all used instances to a HALT state.

Safety information

Requirement: PSD_17

All interface functions except hard_err(...) must return to PSD.

5.3.2 psd_GetFParPtr(…)

The prototype of the function is available in the file p_c_si.h and has to be implemented by the F-Application. It provides a pointer to the PSD for the F-Parameters of the indicated instance. The buffer must be provided by the Stack-Interface.

Function	psd_GetFParPtr(..)	
Prototype:	*P_Byte** ***psd_GetFParPtr*** *(P_Word pInstance);*	
Meaning:	The PSD requests a pointer to the F-Parameters of the instance from the Stack-Interface.	
Transfer parameters:	**Value range**	**Meaning**
pInstance	INST_01…INST_X X <= NOF_INSTANCES	Instance ID
Return parameter:	**Value range**	**Meaning**
P_Byte*	NIL != NIL	No pointer to the buffer for F-Parameters of instance present. Pointer to the buffer for F-Parameters of instance is present.

5.3.3 psd_GetConfigPtr(...)

The prototype of the function is available in the file p_c_si.h and has to be implemented by the F-Application. It provides a pointer to the PSD for the structure with the data length of the indicated instance. The structure must be provided by the Stack-Interface.

Function	psd_GetConfigPtr(...)	
Prototype:	*P_Byte** ***psd_GetConfigPtr*** *(P_Word pInstance);*	
Meaning:	The PSD requests a pointer to the data lengths of the instance from the Stack-Interface.	
Transfer parameters:	**Value range**	**Meaning**
pInstance	INST_01…INST_0X X <= NOF_INSTANCES	Instance ID
Return parameter:	**Value range**	**Meaning**
P_Byte*	NIL != NIL	No pointer to the buffer for the data lengths of the instance present. Pointer to the buffer for the data lengths of the instance present. The buffer consists of an array of 2 bytes. The first byte contains the length of the input telegram. The second byte contains the length of the output telegram. The PSD calculates the length of the user data depend on parameterization.

5.3.4 psd_GetOutputTelegram(...)

The prototype of the function is available in the file p_c_si.h and has to be implemented by the F-Application. It provides a pointer to the PSD for the output data of the indicated instance. The buffer must be provided by the Stack-Interface.

Function	psd_GetOutput_Telegram(..)	
Prototype:	*P_Byte** ***psd_GetOutput_Telegram*** *(P_Word pInstance);*	
Meaning:	The PSD requests a pointer to the output data of the instance from the Stack-Interface .	
Transfer parameters:	**Value range**	**Meaning**
pInstance	INST_01…INST_X X <= NOF_INSTANCES	Instance ID
Return parameter:	**Value range**	**Meaning**
P_Byte*	NIL != NIL	No pointer to the buffer for output data of the instance present. Pointer to the buffer for the output data of the instance is present.

5.3.5 psd_GetInputBuffer(...)

The prototype of the function is available in the file p_c_si.h and has to be implemented by the F-Application. It provides a pointer to the PSD for the input data area of the indicated instance. The buffer must be provided by the Stack-Interface.

Function	psd_GetInputBuffer(..)	
Prototype:	*P_Byte** ***psd_GetInputBuffer*** *(P_Word pInstance* *P_Byte InTeleLen*);*	
Meaning:	The PSD requests a pointer to the input data area of the instance from the Stack-Interface.	
Transfer parameters:	**Value range**	**Meaning**
pInstance	INST_01…INST_X X <= NOF_INSTANCES	Instance ID
InTeleLen		Length of the input data area
Return parameter:	**Value range**	**Meaning**
P_Byte*	NIL != NIL	No pointer to the buffer for input data area of the instance present. Pointer to the buffer for input data area of the instance present.

5.3.6 psd_InputSendAck(...)

The prototype of the function is available in the file p_c_si.h and has to be implemented by the F-Application. It informs the Stack-Interface that the input data and F data were copied by the PSD to the input data area of the indicated instance (predefined by the Stack-Interface using the function psd_Get-InputBuffer()).

Function	psd_InputSendAck(...)	
Prototype:	*P_Byte* ***psd_InputSendAck*** *(P_Word pInstance);*	
Meaning:	The PSD informs the Stack-Interface that the input data were entered in the input data area of the instance.	
Transfer parameters:	**Value range**	**Meaning**
pInstance	INST_01…INST_X X <= NOF_INSTANCES	Instance ID
Return parameter:	**Value range**	**Meaning**
P_Byte	FIN_TELE_UPDATE_OK FIN_TELE_UPDATE_ERR	The input telegram was taken over by the Stack-Interface . Error on taking over the input telegram

5.3.7 enable_int(...)

The prototype of the function is available in the file p_c_application.h and has to be implemented by the F-Application. The PSD indicates that the interrupt lock for this instance has to be released once again.

Function	enable_int(...)	
Prototype:	*void **enable_int** (P_Word pInstance);*	
Meaning:	The PSD informs the Stack-Interface that the interrupt lock of the instance may be removed once again.	
Transfer parameters:	**Value range**	**Meaning**
pInstance	INST_01…INST_X X <= NOF_INSTANCES	Instance ID
Return parameter:	**Value range**	**Meaning**
none		

5.3.8 disable_int (..)

The prototype of the function is available in the file p_c_application.h and has to be implemented by the F-Application. The PSD requests an interrupt lock of the indicated instance because it has to access "unfamilar" data.

Function	enable_int(...)	
Prototype:	*void **disable_int** (P_Word pInstance);*	
Meaning:	The PSD requests an interrupt lock of the instance from the Stack-Interface because it has to access data.	
Transfer parameters:	**Value range**	**Meaning**
pInstance	INST_01…INST_X X <= NOF_INSTANCES	Instance ID
Return parameter:	**Value range**	**Meaning**
none		

5.4 Necessary to implement functions for the dual-channel generation version

In the dual-channel generation variant, the PSD requires a Synchronization-Interface to align the data.

The prototype of the functions is defined in the file p_c_fapplication.h and it enables the PSD to access the Synchronization-Interface provided by the F-Application. The file p_c_fapplication.c is available for the adaptation.

5.4.1 psd_OutTransfer(…)

This function is called up from the PSD in the function **psd_RecvFOutTele()**. It initiates transfer of the PSD status to other channel for specific instances.

Function	psd_OutTransfer	
Prototype:	*P_Byte **psd_OutTransfer*** *(P_Word pInstance, P_Byte State);*	
Meaning:	The PSD transfers the PSD status of an interface to the Synchronization-Interface. The function initiates transfer to the second channel so that it can access the PSD status with the function psd_OutSync() at a later time.	
Transfer parameters:	**Value range**	**Meaning**
pInstance	INST_01…INST_X X <= NOF_INSTANCES	Instance ID
State		Status to be transmitted
Return parameter:	**Value range**	**Meaning**
P_Byte	TRANSFER_OK TRANSFER_NOT_OK	Synchronization-Interface reports all ok Synchronization-Interface reports error

5.4.2 psd_OutSync(…)

This function is called up from the PSD in the function **psd_GetFOutData()**. It transmits an instance specific pointer to a byte. The function enters the PSD status that was received by the Synchronization-Interface from the second channel in the meantime.

Function	psd_OutSync	
Prototype:	*P_Byte **psd_OutSync*** *(P_Word pInstance, P_Byte* State);*	
Meaning:	The PSD transmits a pointer to a byte. The function enters the PSD status that was received by the Synchronization-Interface from the second channel for specific instances.	
Transfer parameters:	**Value range**	**Meaning**
pInstance	INST_01…INST_X X <= NOF_INSTANCES	Instance ID
**State*		Pointer to a status byte
Return parameter:	**Value range**	**Meaning**
P_Byte	SYNC_OK SYNC_NOT_OK	Synchronization-Interface reports all ok Synchronization-Interface reports error

Safety information

Requirement: PSD_25

The F-Application must provide a Synchronization-Interface for dual-channel PSD generation. Mirroring of the data to be swapped should be prohibited.

5.4.3 psd_InTransfer(…)

This function is called up from the PSD in the function **psd_SetFInData()**. It initiates transfer of the input telegram CRC to other channel for specific instances.

Function	psd_InTransfer	
Prototype:	*P_Byte **psd_InTransfer*** *(P_Word pInstance, P_DWord CRCValue);*	
Meaning:	The PSD transfers the CRC2 of the input telegram of an instance to the Synchronization-Interface. The function initiates transfer to the second channel so that it can access the CRC2 with the function psd_InSync() at a later time.	
Transfer parameters:	**Value range**	**Meaning**
pInstance	INST_01…INST_X X <= NOF_INSTANCES	Instance ID
CRCValue		CRC value to be transmitted
Return parameter:	**Value range**	**Meaning**
P_Byte	TRANSFER_OK TRANSFER_NOT_OK	Stack-Interface reports all ok Stack-Interface reports error

5.4.4 psd_InSync(…)

This function is called up from the PSD in the function **psd_SendFInTele()**. It transmits an instance specific pointer to a DWord. The function enters the CRC value that was received by the Synchronization-Interface from the second channel in the meantime.

Function	psd_OutSync	
Prototype:	*P_Byte **psd_InSync*** *(P_Word pInstance, P_DWord* CRCValue);*	
Meaning:	The PSD transmits a pointer to a DWord. The function enters the CRC2 of the input telegram that was received by the Synchronization-Interface from the second channel for specific instances.	
Transfer parameters:	**Value range**	**Meaning**
pInstance	INST_01…INST_X X < = NOF_INSTANCES	Instance ID
**CRCValue*		Pointer to a CRC value
Return parameter:	**Value range**	**Meaning**
P_Byte	SYNC_OK SYNC_NOT_OK	Synchronization-Interface reports all ok Synchronization-Interface reports error

Safety information

Requirement: PSD_25

The F-Application must provide a Synchronization-Interface for dual-channel PSD generation. Mirroring of the data to be swapped should be prohibited.

5.5 Optional PSD interface functions

Safety information

Requirement: PSD_17

All interface functions except hard_err(...) must return to PSD.

5.5.1 Help functions for CRC calculation

5.5.1.1 psd_CRC16(…)

The function calculates a 16-bit CRC across a data area. The generator polynomial 0x14EAB is used.

Function:	**psd_CRC16(…)**	
Prototype:	P_Word **psd_CRC16** *(P_DWord pLength,* *P_Word PStartValue,* *const P_Byte pData[],* *P_Byte pCalcDirection);*	
Meaning:	CRC16 calculation across any memory section	
Transfer parameters:	**Value range**	**Meaning**
pLength		Data length for calculation
PStartValue		Start-CRC16 value for calculation
pData		Pointer to data for calculation
pCalcDirection	FORWARD BACKWARD	It is calculated as of the start address of the data of the CRC16 value via the values according to the length information in increasing order. It is calculated as of the end address of the data of the CRC16 value via the values according to the length information in decreasing order.
Return parameter:	P_Word	
Value range		**Meaning**
0 – 0xFFFF		16-bit CRC value

5.5.1.2 psd_CRC32(…)

The function calculates a 32-bit CRC across a data area. The generator polynomial 0x1F4ACFB13 is used.

Function:	**psd_CRC32(…)**
Prototype:	P_DWord **psd_CRC32** *(P_DWord pLength,* *P_DWord PStartValue,* *const P_Byte pData[],* *P_Byte pCalcDirection);*
Meaning:	CRC16 calculation across any memory section

Transfer parameters:	Value range	Meaning
pLength		Data length for calculation
PStartValue		Start-CRC16 value for calculation
pData		Pointer to data for calculation
pCalcDirection	FORWARD BACKWARD	It is calculated as of the start address of the data of the CRC32 value via the values according to the length information in increasing order. It is calculated as of the end address of the data of the CRC32 value via the values according to the length information in decreasing order.
Return parameter:	P_DWord	
Value range		**Meaning**
0 – 0xFFFF FFFF		32-bit CRC value

5.5.2 psd_GetState(…)

The function can be used to query the current PSD state.

Function:	psd_GetState(…)	
Prototype:	*P_Byte **psd_GetState**(P_Word pinstance);*	
Meaning:	Output of PSD status of instance	
Transfer parameters:	**Value range**	**Meaning**
pInstance	INST_01…INST_X X <= NOF_INSTANCES	Instance ID
Return parameter:	P_Byte	
Value range		**Meaning**
PSD_INIT		Initialized
PSD_PARAM		Assigned
PSD_DATAEX		Error-free cyclical data traffic
PSD_HARD_FAIL		Fatal internal error ➔ Call up the function hard_err(...)

5.5.3 psd_GetFPar(…)

Use this function to read out the set SIL value and of the F_iPar_CRC.

Function:	psd_GetFPar(…)	
Prototype:	*P_Byte **psd_GetFPar**(P_Word pInstance Fw_PsdFPar* pFwFParam);*	
Meaning:	Read out the SIL value and the F_iPar_CRC value	
Transfer parameters:	**Value range**	**Meaning**
pInstance	INST_01…INST_X X <= NOF_INSTANCES	Instance ID
pFwFParam		Pointer to device parameter structure
Return parameter:	**Value range**	**Meaning**
P_Byte	**GET_FPAR_OK**	Call is correct

Structure of the device parameter structure:

With dual channel generation version:

```
typedef struct Fw_PsdFPar_t {
  P_DWord      FiParCRC
  P_Byte       FSil
} Fw_PsdFPar;
```

With single channel generation version:

```
typedef struct Fw_PsdFPar_t {
  P_DWord      FiParCRC
  P_DWord      FiParCRC_inv
  P_Byte       FSil
  P_Byte       FSil_inv
} Fw_PsdFPar;
```

5.5.4 Inverse PSD Functions

In the single-channel generation variant, the PSD provides a corresponding inverse function for each main function. The functions described below support a single-channel software architecture of the F-Application. The functions deliver the results of the inverse calculation path of the PSD. The meaning of the returned values can be found in the main functions.

Safety information

Requirement: PSD_19

The returned values of the 'inverse' functions used are to be evaluated accordingly.

5.5.4.1 psd_InitInstance_Invers (…)

Function:	**psd_InitInstance_Invers (…)**	
Prototype:	*P_Byte* ***psd_InitInstance_Invers*** *(P_Word pInstance);*	
Meaning:	Return of the inverse initialization status of the PSD instance	
Transfer parameters:	**Value range**	**Meaning**
pInstance	INST_01…INST_X X <= NOF_INSTANCES	Instance ID
Return parameter:	P_Byte	
Value range		**Meaning**
INIT_INSTANCE_OK_INV		Initialization successful

5.5.4.2 psd_FParBuild_Invers(…)

Function:	**psd_FParBuild_Invers(…)**
Prototype:	*P_Byte* ***psd_FParBuild_Invers*** *(P_Word pInstance);*

Meaning:	Return of the inverse parameter status of the PSD instance	
Transfer parameters:	**Value range**	**Meaning**
pInstance	INST_01…INST_0X X <= NOF_INSTANCES	Instance ID
Return parameter:	P_Byte	

Value range	**Meaning**
FPAR_BUILD_OK_INV	Parameter assignment successful
FPAR_F_DEST_ADD_MISMATCH_INV	Incorrect target address
FPAR_F_DEST_ADD_NOT_VALID_INV	Invalid target address
FPAR_F_SOURCE_ADD_NOT_VALID_INV	Invalid source address
FPAR_F_WD_TIME_NULL_INV	Watchdog time is invalid (==0)
FPAR_F_SIL_ERR _INV	Transferred SIL class too high
FPAR_CRC1_ERR _INV	The CRC of the F-Parameters is invalid
FPAR_BUILD_NOT_OK_INV	Inconsistent parameter assignment

5.5.4.3 psd_Config_Invers (…)

Function:	**psd_Config_Invers (…)**	
Prototype:	*P_Byte **psd_Config_Invers** (P_Word pInstance);*	
Meaning:	Return of the inverse configuration status of the PSD instance.	
Transfer parameters:	**Value range**	**Meaning**
pInstance	INST_01…INST_X X <= NOF_INSTANCES	Instance ID
Return parameter:	P_Byte	

Value range	**Meaning**
FPAR_CONFIG_OK_INV	Configuration successful
FPAR_CRC_LENGTH_INV	Invalid length information

5.5.4.4 psd_Run_Invers (…)

Function:	**psd_Run_Invers(…)**	
Prototype:	*P_Byte **psd_Run_Invers** (P_Word pInstance);*	
Meaning:	Return of the inverse PSD status of the PSD instance.	
Transfer parameters:	**Value range**	**Meaning**
pInstance	INST_01…INST_X X <= NOF_INSTANCES	Instance ID
Return parameter:	P_Byte	

Value range	**Meaning**
PSD_DATAEX_INV	Error-free cyclical data traffic

5.5.4.5 psd_Stop_Invers (…)

Function:	**psd_Stop_Invers (…)**	
Prototype:	*P_Byte **psd_Stop** (P_Word pInstance);*	
Meaning:	Return of the inverse PSD status of the PSD instance.	
Transfer parameters:	**Value range**	**Meaning**
pInstance	INST_01…INST_X X <= NOF_INSTANCES	Instance ID

Return parameter:	P_Byte
Value range	**Meaning**
PSD_PARAM_INV	Assigned

5.5.4.6 psd_GetFPar_Invers (…)

Function:	**psd_GetFPar_Invers (...)**	
Prototype:	*P_Byte **psd_GetFPar_Invers**(P_Word pInstance);*	
Meaning:	Return of the inverse acknowledgment of psd_GetFPar() of the PSD instance	
Transfer parameters:	**Value range**	**Meaning**
pInstance	INST_01…INST_X X <= NOF_INSTANCES	Instance ID
Return parameter:	**Value range**	**Meaning**
P_Byte	**GET_FPAR_OK_INV**	Call is correct

5.5.4.7 psd_GetState_Invers (…)

Function:	**psd_GetState_Invers (...)**	
Prototype:	*P_Byte **psd_GetState_Invers** (P_Word pinstance);*	
Meaning:	Return of the inverse PSD status of the PSD instance.	
Transfer parameters:	**Value range**	**Meaning**
pInstance	INST_01…INST_0X X <= NOF_INSTANCES	Instance ID
Return parameter:	P_Byte	

Value range	**Meaning**
PSD_INIT_INV	Initialized
PSD_PARAM_INV	Assigned
PSD_DATAEX_INV	Error-free cyclical data traffic
PSD_HARD_FAIL_INV	Fatal internal error ➔ Call up the function hard_err(...)

6 Configuration of the PSD

The generation parameters (compiler switches) described below can be found in the PSD files p_c_config.h.

6.1 Generation variant

#undef REDUNDANT
Single-channel generation variant Deployment of the PSD on a single-channel hardware architecture. The PSD stores all internal data as normal and inversely. The PSD functions use different algorithms to calculate on the normal and inverse data.
#define REDUNDANT
Dual-channel generation variant Deployment of the PSD on a dual-channel hardware architecture. The processing is dual-channel. The data alignment takes place via the Synchronization-Interface.

6.2 Memory space optimization for CRC tables and functions

#undef CRC32
Tables and functions that are required exclusively for the 4-byte CRC are not generated. The PSD does **not** support the 32-bit CRC required for the cyclical input and/or output telegrams in V1 mode and V2 mode according to the PROFIsafe profile.
#define CRC32
Tables and functions that are required exclusively for the 4-byte CRC are generated. The PSD does support the 32-bit CRC required for the cyclical input and/or output telegrams in V1 mode and V2 mode according to the PROFIsafe profile.

6.3 Instance information

#define NOF_INSTANCES 1
Number of instances. A length information is required for each used instance of F output and F input user data in p_c_config.h. Value range: 1 – 32

6.4 F output user data range

#define OUTDATA_MAX_LEN_INST_01 2 **:** **#define OUTDATA_MAX_LEN_INST_32 X**
Use Defines to set the maximum number of cyclical F-output user data for each used instance. 0 means that no output user data are used. Value range: 0 – 12 bytes for 2-byte CRC in V1 mode 0 – 12 bytes for 3□byte CRC in V2 mode 0 – 122 bytes for 4□byte CRC in V1 mode 0 – 123 bytes for 4□byte CRC in V2 mode

6.5 F input user data range

#define INDATA_MAX_LEN_INST_01 2 **:** **#define INDATA_MAX_LEN_INST_32 X**
Use Defines to set the maximum number of cyclical F-input user data for each used instance. 0 means that no input user data are used. Value range: 0 – 12 bytes for 2-byte CRC in V1 mode 0 – 12 bytes for 3□byte CRC in V2 mode 0 – 122 bytes for 4□byte CRC in V1 mode 0 – 123 bytes for 4□byte CRC in V2 mode

#undef INST_ACCESS_POINTER
For instances >1 addressing to the PSD instance data takes place via array pointers. Depending on the compiler type, this can mean a faster access method. With only one instance, addressing will take place via array pointers with array index "0" as constant.
#define INST_ACCESS_POINTER
For instances >1 addressing to the PSD instance data takes place via array pointers. Depending on the compiler type, this can mean a faster access method.

6.6 Memory attributes

The memory attributes depend on the used tool chain. Use the keywords accordingly.

#define P_GLOBDAT_ATTR_POINTER
Memory attribute for pointers accessing PSD global variables.

#define P_GLOBDAT_ATTR
Memory attribute for storing the PSD global variables.

#define P_GLOBDAT_I_ATTR
Memory attribute for saving PSD global variables that can be different on both channels (in case of two channels).

#define CRC_DAT_ATTR
Memory attribute for storing the CRC tables.

#define P_CODE_ATTR const
Memory attribute for the code area.

6.7 Memory model CRC tables.

This switch turns on/off the memory model for the CRC tables.

The memory attributes depend on the used tool chain. Use the keywords accordingly.

#define p_crctab_memory
The files "p_c_crc_m.h" and "p_c_crc_b.h" are included. In p_c_crc_m.h you can set the tool chain specific memory model for CRC tables. In p_c_crc_b.h you can return to the memory model for the PSD code.

#undef p_crctab_memory
The files "p_c_crc_m.h" and "p_c_crc_b.h" are not included.

7 Requirements

In order to meet the requirements of the corresponding safety level of the PSD, the F-Application must meet the requirements summarized below.

7.1 Requirements independent of the generation variant

PSD_01 The normative requirements according to the standards on which they are based must be adhered to with regard to the F-Application.

PSD_02 The PSD files must not be changed.

PSD_03 The CRC values of the PSD files (individual and overall CRC) must match the values of the CRC check tool.

PSD_04 Only the PSD interface functions declared in the file p_api.h may be used.

PSD_05 Only the PSD interface functions declared in the file p_global.h may be used.

PSD_06 The defines from p_global.h must be used for evaluation of the PSD returns.

PSD_07 The development tools are to be qualified by the manufacturer of the F-Slave.

PSD_08 If code optimization is used, the restrictions described in chapter 8.3 have to be considered.

PSD_09 Invalid return values must lead to the F-Application state **HARD_FAIL**.

The following return values are permitted:

Function	Permitted return values
psd_InitInstance(…)	**INIT_INSTANCE_OK**
psd_FParBuild(…)	**FPAR_BUILD_OK** **FPAR_F_DEST_ADD_MISMATCH** **FPAR_F_DEST_ADD_NOT_VALID** **FPAR_F_SOURCE_ADD_NOT_VALID** **FPAR_F_WD_TIME_NULL** **FPAR_F_SIL_ERR** **FPAR_CRC1_ERR** **FPAR_BUILD_NOT_OK**
psd_Config(…)	**FPAR_CONFIG_OK**

Function	Permitted return values
psd_Run(…)	**PSD_DATAEX**
psd_Stop(…)	**PSD_PARAM**
psd_RecvFOutTele(…)	**RECV_FOUT_TELE_OK**
psd_GetFOutData(…)	**F_OUTPUT_OK** **F_OUTPUT_OLD_CONSNR** **F_OUTPUT_CLEAR** **F_OUTPUT_COMM_ERR** **F_OUTPUT_WD_TIMEOUT** **F_OUTPUT_PASSIVATED**
psd_SetFInData(…)	**SEND_FIN_DATA_OK**
psd_SendFInTele(…)	**SEND_FIN _TELE_OK**
psd_GetState(…)	**PSD_INIT** **PSD_PARAM** **PSD_DATAEX** **PSD_HARD_FAIL**
psd_GetFPar(…)	**GET_FPAR_OK**

The permitted return value range of the "inverse" functions (only for the single channel generation variant) correspond to those of the main functions.

PSD_10 The function hard_err(...) is to be implemented by the F-Application. It must lead for all used instances to a HALT state.

PSD_11 The SIL must correspond to the SIL of the F-Slave and must be determined independent of the F-Parameters.

PSD_12 The F-Address must be stored in the F-Application. An extraction of the F-Address or other values from the F-Parameters is not permitted.

PSD_13 For:
F_OUTPUT_COMM_ERR,
F_OUTPUT_WD_TIMEOUT,
F_OUTPUT_PASSIVATED
the F-Application must ensure the safe state for the process.

PSD_14 A PROFIsafe-Conformance test must be run by an accredited inspection laboratory (see PNO).

PSD_15 The test cases specified in chapter 8 are to be run.

PSD_16 The status byte.bit_1 shall be set by the F-Application for at least two (2) changes of the consecutive number (2 x returnvalue "F_OUTPUT_OK" of the PSD-function psd_GetFOutData()), if

the F-Slave is not able to guarantee the safety integrity of the process data to be transmitted. Signal name is "Device Fault".

PSD_17 All interface functions except hard_err(...) must return to PSD.

7.2 Requirements for the single-channel generation variant

PSD_18 The F-Application has to test the consistency of the normal and inverse F-output user data as well as control byte IDs. The consistency ckeck has to be done before output to the process. If the PSD reports a communication error, the received control byte is invalid and the evaluation of the control byte IDs isn´t permissible.

PSD_19 The returned values of the 'inverse' functions used are to be evaluated accordingly.

PSD_20 If the F-Application goes into the state **HARD_FAIL,**, the falsification variable **PSD_safety_var** must be set to the value **P_SAFETY_CONST_FALSE**.

PSD_21 Analysis with the verification code in the file p_verify.c /4/ must be performed to verify the compiler.

PSD_22 When using the single channel-PSD in an actor or sensor, all safety critical secondary faults have to be uncovered by the surrounding of the PSD.

PSD_23 When using the single channel-PSD in an actor, accidental hardware-faults have to be uncoverd within the process-fault-tolerance time by the surroundings.

7.3 Requirements for the dual-channel generation variant

PSD_24 The F-Application has to test the consistency of the F-output user data as well as control byte IDs of both hardware channels. The consistency ckeck has to be done before output to the process. If the PSD reports a communication error, the received control byte is invalid and the evaluation of the control byte IDs isn´t permissible.

PSD_25 The F-Application must provide a Synchronization-Interface for dual-channel PSD generation. Mirroring of the data to be swapped should be prohibited.

8 Test specification

8.1 Fault insertion test

The specific implementation of the PSD error detection mechanism (program operating control (single channel), Sync-Interface (dual channel), consistency comparison of PSD internal double + inverse data, etc.) are proven by the fault insertion test.

The test code necessary for the test is inserted with the help of macro into the C code of the PSD or directly in the F-Application C code. The PSD test code is contained as an example in the file p_c_fault_insert.h and must be modifed for the specific F-Slave. The define "FAULT_INSERTION" must be commented out in the file p_global.h.

Each individual fault insertion test must be "switched on" in the file; the entire code must be compiled and executed. In each test the state of HARD_FAIL must be attained during the course of PROFIsafe communication.

Safety information

Requirement: PSD_15

The test cases specified in chapter 8 are to be run.

8.1.1 Single channel and dual channel generation variants

The following tests must be performed:

No.	Description	Expected result
1	The parameter F_ iPar_CRC will be falsified. Error injection by the macro FI_PSD_025	Reaction of the PSD: The PSD detects the configuration error. Reaction of the F-Application: The secure state of the process values, dependent upon the application, is determined.
2	The PSD is given an excessive length via the function psd_Config(..). Error injection by the macro FI_PSD_024	The PSD detects an error and calls up the function hard_err(...).
3	The PSD is not set into the cyclical data exchange (pst_run is not started). Error injection through F-Application	Reaction of the F-Host: The F-Host detects a timeout.
4	The PSD function psd_SetFInData(..) is given an excessively large instance. Error injection by the macro FI_PSD_001	The PSD detects an error and calls up the function hard_err(...).
5	The PSD function psd_SetFInData(..) is given an incorrect data length. Error injection by the macro FI_PSD_002	The PSD detects an error and calls up the function hard_err(...).
6	The PSD function psd_SetFInData() is terminated prematurely. Error injection by the macro FI_PSD_003	The PSD detects an error and calls up the function hard_err(...).
7	The PSD is given an excessive data length via the function psd_GetInputBuffer(..). Error injection by the macro FI_PSD_005	The PSD detects an error and calls up the function hard_err(...).
8	The PSD function psd_SendFInTele(..) is given an incorrect CRC value. Error injection by the macro FI_PSD_006	The PSD detects an error and calls up the function hard_err(...).
9	The PSD function psd_SetFInTele() is terminated prematurely. Error injection by the macro FI_PSD_011	The PSD detects an error and calls up the function hard_err(...).
10	The PSD function psd_RecvFOutTele(..) is given an incorrect CRC value. Error injection by the macro FI_PSD_013	The PSD detects an error and calls up the function hard_err(...).
11	The PSD function psd_RecvFOutTele() is terminated prematurely. Error injection by the macro FI_PSD_017	The PSD detects an error and calls up the function hard_err(...).

8.1.2 Single channel generation variant

The following tests must be performed:

No.	Description	Expected result
12	Recognition of a falsfication of the process control on a logical channel via FlowControl value Error injection by the macro FI_PSD_151	Reaction of the PSD: The process control of the PSD detects an error and calls up the function hard_err(...).
13	Recognition of a falsfication of the process control on a logical channel via FlowControl_inv value Error injection by the macro FI_PSD_152	Reaction of the PSD: The process control of the PSD detects an error and calls up the function hard_err(...).

8.1.3 Dual channel generation variant

The following tests must be performed:

No.	Description	Expected result
14	Detection of an incorrect Sync.-Quit on a channel when sending the PSD, change to state PSD_HARD_FAIL: In function psd_SetFInData() the Sync_Quit to TRANSFER_NOT_OK is falsified after call up of psd_InTransfer() on a channel. Error injection by the macro FI_PSD_102	Reaction of the PSD: The PSD detects a synchronization error on a channel and calls up the function hard_err(...). A synchronization timeout is generated on the other channel and the function hard_err(...) is called up.
15	Detection of an incorrect Sync.-Quit on a channel when sending the PSD, change to state PSD_HARD_FAIL: In function psd_SetFInTele() the Sync_Quit to SYNC_NOT_OK is falsified after call up of psd_InSync () on a channel. Error injection by the macro FI_PSD_104	Reaction of the PSD: The PSD detects a synchronization error on a channel and calls up the function hard_err(...). A synchronization timeout is generated on the other channel and the function hard_err(...) is called up.

8.2 Qualification of the development tools

Safety information

Requirement: PSD_07

The development tools are to be qualified by the manufacturer of the F-Slave.

8.3 Code generation

Code generation for the single channel variant with use of code optimization has to retain the logical dual channel model in the code.

Safety information

Requirement: PSD_21

Analysis with the verification code in the file p_verify.c /4/ must be performed to verify the compiler.

8.4 Compiler verification code for single channel generation variants

The verification code in the file p_verify.c shows how the compiler used in the PSD converts code sequences used in the PSD. To prove redundancy in the code, all instances marked with "M:" must be met. All instances marked with "S:" show the implementation of C instructions in Assembler and are for information purposes only. In the comments, the term "move" is used for assignments and the term "compare" is used for comparisons. These terms are to be interpreted according to the controller used.

Safety information

Requirement: PSD_21

Analysis with the verification code in the file p_verify.c /4/ must be performed to verify the compiler.

8.5 PROFIsafe-Conformance test

Safety information

Requirement: PSD_14

A PROFIsafe-Conformance test must be run by an accredited inspection laboratory (see PNO).

9 Appendix

9.1 Memory requirement

The memory requirement has been determined using the list files with a generation for the DPC31 EKit. A Keil-C-Compiler V7.01 was used.

9.1.1 Single-channel generation

Compiler switches	Data [bytes]	CRC tables [bytes]	Code [bytes]
#define NOF_INSTANCES 1 #undef REDUNDANT #undef CRC32 #undef INST_ACCESS_POINTER	approximately 438	approximately 1536	approximately 15534

9.1.2 Dual-channel generation

Compiler switches	Data [bytes]	CRC tables [bytes]	Code [bytes]
#define NOF_INSTANCES 1 #define REDUNDANT #undef CRC32 #undef INST_ACCESS_POINTER	approximately 147	approximately 1536	approximately 6995

9.2 Static code analysis

The tool "QA C" was used for the static code analysis. The default settings with MISRA supplementary module were used. The following table summarizes the messages from all configurations. (The list of messages can be different in the individual configurations.)

QA C Message category	QA C Message number and text	Comment
(2) Switch statements	3352: This 'switch' statement contains only two execution paths.	When using different configurations, "case" branches (through Compiler switch) may no longer be present.
(3) Redundancy	3199: The value of <variable_name> is never used following this assignment.	In single instance operation, "pID" is always 0 and for performance reasons "pID" is replaced with "0" in the single instance configuration in case of data pointers. (The configuration-dependent design of the function header is not necessary.) To suppress the Compiler warning a redundant assignment is used.

QA C Message category	QA C Message number and text	Comment
(4) M2CM Rule 8.7	1514: The object <object_name> is only referenced by function <function_name>, in the translation unit where it is defined.	Certain global data are only referenced by the operator of PSD internal by pointers. This external or indirect referencing is not detected as referencing by QA C.
(4) M2CM Rule 8.10	1504: The object <object_name> is only referenced in the translation unit where it is defined.	The CRC tables must be accessible for PSD users. The tables are part of the PSD interface but are also used internally.
(4) M2CM Rule 8.10	1505: The function <function_name> is only referenced in the translation unit where it is defined.	Certain functions must be accessible for PSD users that are referenced/used internally. The functions are part of the PSD interface. The inverse path does not follow the physical path of the standard path, but the automatically generated functions are all housed in the same compilation unit (source file). This means functions that are referenced in the standard path by other sources are referenced by the same source in the inverse path only.
(4) M2CM Rule 11.4	310: Casting to different object pointer type.	Intentional type conversion with loss to determine the storage format (Intel or Motorola) or intentional conversion in which an integer date is used as byte array in the CRC calculation.
(4) M2CM Rule 13.7	3556: The result of this logical operation is always 'false'.	Certain control structures for multi-instance operation are not completely executed/utilized in single instance operation.
(4) M2CM Rule 13.7	3559: The result of this logical operation is always 'false'.	Certain control structures for multi-instance operation are not completely executed/utilized in single instance operation.
(4) M2CM Rule 14.1	1503: The function <function_name> is defined but is not used within this project.	Interface functions are referenced by the PSD users (external) only.
(4) M2CM Rule 14.1	3201: This statement is unreachable.	Certain control structures for multi-instance operation are not completely executed/utilized in single instance operation.
(4) M2CM Rule 14.10	2004: No concluding 'else' exists in this 'if'-'else'-'if' statement.	No "else" branch is required in this "if-else-if" structures.
(4) M2CM Rule 19.4	3412: Macro defines an unrecognised code-fragment.	-This macro definition is necessary for performance reasons to permit pointer access. (There are platforms in which pointer access is much faster than array access)
(4) M2CM Rule 19.7	3453: A function could probably be used instead of this function-like macro.	A function is hidden behind a macro in this case for configuration reasons. The callup can be turned on or off depending on the configuration.

QA C Message category	QA C Message number and text	Comment
(6) Implementation defined	288: [I] Source file <file_name> has comments containing characters which are not members of the basic source character set.	The $ character in the comment is necessary for automatic generation of the inverse branch.

9.3 Development documents

The documents listed below were created during development and are the basis for certification.

- PROFIsafe Profil /1/
- Safety requirement specification
- Safety plan
- Feature Specification
- FMEA
- Software design
- Safety Criticality Analysis
- Module test specification
- Module test reports
- Integration test specification
- Integration test report
- Fault-Insertion test specification
- Fault-Insertion test report
- PROFIsafe-Conformity test
- PROFIsafe-Conformity test report
- Manual

10 Terms and acronyms

CRC	**C**yclic **R**edundancy **C**heck
DP	**D**ecentralized **P**eriphery
EPROM	**E**rasable **p**rogrammable **r**ead-**o**nly **m**emory
F	**F**ail-safe
FW	**F**irm**w**are
HW	**H**ard**w**are
ISR	**I**nterrupt **S**ervice **R**outine
PNO	**P**ROFIBUS-User-**O**rganisation URL: http://www.profibus.com/
Prm	**P**arameter
PSD	**P**ROFI**s**afe **S**lave Driver
SI	DP-**S**tack-**I**nterface
SIL	**S**afety **I**ntegrity **L**evel
SW	**S**oft**w**are
WD	**W**atch**d**og, for runtime monitoring
zero-telegram	All the bytes of a telegram contain "0"