BS EN 61784-3-12:2010

BSI Standards Publication

Industrial communication networks — Profiles -

Part 3-12: Functional safety fieldbuses — Additional specifications for CPF 12

raising standards worldwide™

National foreword

This British Standard is the UK implementation of EN 61784-3-12:2010.

The UK participation in its preparation was entrusted to Technical Committee AMT/7, Industrial communications: process measurement and control, including fieldbus.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

ISBN 978 0 580 72032 1

ICS 25.040.40; 35.100.05

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 30 September 2010.

Amendments issued since publication

Date	Text affected

EUROPEAN STANDARD

NORME EUROPÉENNE

EUROPÄISCHE NORM

EN 61784-3-12

August 2010

ICS 25.040.40; 35.100.05

English version

Industrial communication networks - Profiles - Part 3-12: Functional safety fieldbuses - Additional specifications for CPF 12

(IEC 61784-3-12:2010)

Réseaux de communication industriels -
Partie 3-12: Bus de terrain à sécurité fonctionnelle -
Spécifications complémentaires pour le CPF 12
(CEI 61784-3-12:2010)

Industrielle Kommunikationsnetze -
Profile -
Teil 3-12: Funktional sichere Übertragung bei Feldbussen -
Zusätzliche Festlegungen für die Kommunikationsprofilfamilie 12
(IEC 61784-3-12:2010)

This European Standard was approved by CENELEC on 2010-07-01. CENELEC members are bound to comply with the CEN/CENELEC Internal Regulations which stipulate the conditions for giving this European Standard the status of a national standard without any alteration.

Up-to-date lists and bibliographical references concerning such national standards may be obtained on application to the Central Secretariat or to any CENELEC member.

This European Standard exists in three official versions (English, French, German). A version in any other language made by translation under the responsibility of a CENELEC member into its own language and notified to the Central Secretariat has the same status as the official versions.

CENELEC members are the national electrotechnical committees of Austria, Belgium, Bulgaria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Norway, Poland, Portugal, Romania, Slovakia, Slovenia, Spain, Sweden, Switzerland and the United Kingdom.

CENELEC

European Committee for Electrotechnical Standardization
Comité Européen de Normalisation Electrotechnique
Europäisches Komitee für Elektrotechnische Normung

Management Centre: Avenue Marnix 17, B - 1000 Brussels

Ref. No. EN 61784-3-12:2010 E

Foreword

The text of document 65C/591A/FDIS, future edition 1 of IEC 61784-3-12, prepared by SC 65C, Industrial networks, of IEC TC 65, Industrial-process measurement, control and automation, was submitted to the IEC-CENELEC parallel vote and was approved by CENELEC as EN 61784-3-12 on 2010-07-01.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. CEN and CENELEC shall not be held responsible for identifying any or all such patent rights.

The following dates were fixed:

– latest date by which the EN has to be implemented at national level by publication of an identical national standard or by endorsement	(dop)	2011-04-01
– latest date by which the national standards conflicting with the EN have to be withdrawn	(dow)	2013-07-01

Annex ZA has been added by CENELEC.

Endorsement notice

The text of the International Standard IEC 61784-3-12:2010 was approved by CENELEC as a European Standard without any modification.

In the official version, for Bibliography, the following notes have to be added for the standards indicated:

IEC 61158 series	NOTE Harmonized in EN 61158 series (not modified).
IEC 61496 series	NOTE Harmonized in EN 61496 series (partially modified).
IEC 61508-1:2010	NOTE Harmonized as EN 61508-1:2010 (not modified).
IEC 61508-4:2010	NOTE Harmonized as EN 61508-4:2010 (not modified).
IEC 61508-5:2010	NOTE Harmonized as EN 61508-5:2010 (not modified).
IEC 61511 series	NOTE Harmonized in EN 61511 series (not modified).
IEC 61784-1	NOTE Harmonized as EN 61784-1.
IEC 61784-5 series	NOTE Harmonized in EN 61784-5 series (not modified).
IEC 61800-5-2	NOTE Harmonized as EN 61800-5-2.
IEC 62061	NOTE Harmonized as EN 62061.
ISO 10218-1	NOTE Harmonized as EN ISO 10218-1.
ISO 12100-1	NOTE Harmonized as EN ISO 12100-1.
ISO 13849-1	NOTE Harmonized as EN ISO 13849-1.
ISO 13849-2	NOTE Harmonized as EN ISO 13849-2.

Annex ZA
(normative)

Normative references to international publications with their corresponding European publications

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

NOTE When an international publication has been modified by common modifications, indicated by (mod), the relevant EN/HD applies.

Publication	Year	Title	EN/HD	Year
IEC 60204-1	-	Safety of machinery - Electrical equipment of machines - Part 1: General requirements	EN 60204-1	-
IEC 61000-6-2	-	Electromagnetic compatibility (EMC) - Part 6-2: Generic standards - Immunity for industrial environments	EN 61000-6-2	-
IEC 61131-2	-	Programmable controllers - Part 2: Equipment requirements and tests	EN 61131-2	-
IEC 61158-2	-	Industrial communication networks - Fieldbus specifications - Part 2: Physical layer specification and service definition	EN 61158-2	-
IEC 61158-3-12	-	Industrial communication networks - Fieldbus specifications - Part 3-12: Data-link layer service definition - Type 12 elements	EN 61158-3-12	-
IEC 61158-4-12	-	Industrial communication networks - Fieldbus specifications - Part 4-12: Data-link layer protocol specification - Type 12 elements	EN 61158-4-12	-
IEC 61158-5-12	-	Industrial communication networks - Fieldbus specifications - Part 5-12: Application layer service definition - Type 12 elements	EN 61158-5-12	-
IEC 61158-6-12	-	Industrial communication networks - Fieldbus specifications - Part 6-12: Application layer protocol specification - Type 12 elements	EN 61158-6-12	-
IEC 61326-3-1	-	Electrical equipment for measurement, control and laboratory use - EMC requirements - Part 3-1: Immunity requirements for safety-related systems and for equipment intended to perform safety-related functions (functional safety) - General industrial applications	EN 61326-3-1	-

Publication	Year	Title	EN/HD	Year
IEC 61326-3-2	-	Electrical equipment for measurement, control and laboratory use - EMC requirements - Part 3-2: Immunity requirements for safety-related systems and for equipment intended to perform safety-related functions (functional safety) - Industrial applications with specified electromagnetic environment	EN 61326-3-2	-
IEC 61508	Series	Functional safety of electrical/electronic/programmable electronic safety-related systems	EN 61508	Series
IEC 61784-2	-	Industrial communication networks - Profiles - Part 2: Additional fieldbus profiles for real-time networks based on ISO/IEC 8802-3	EN 61784-2	-
IEC 61784-3	2010	Industrial communication networks - Profiles - Part 3: Functional safety fieldbuses - General rules and profile definitions	EN 61784-3	2010
IEC 61918	-	Industrial communication networks - Installation of communication networks in industrial premises	EN 61918	-

CONTENTS

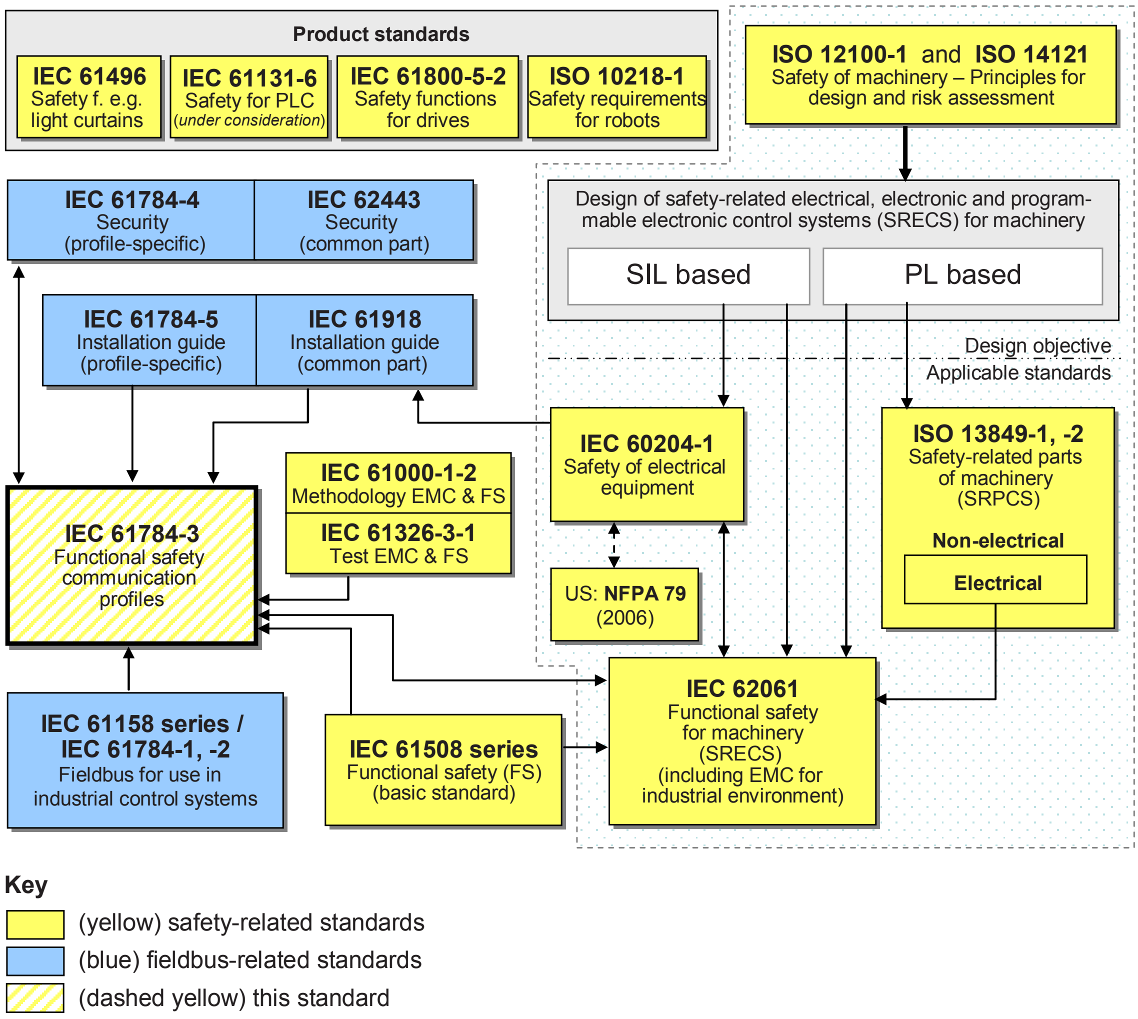
0 Introduction

0.1 General

The IEC 61158 fieldbus standard together with its companion standards IEC 61784-1 and IEC 61784-2 defines a set of communication protocols that enable distributed control of automation applications. Fieldbus technology is now considered well accepted and well proven. Thus many fieldbus enhancements are emerging, addressing not yet standardized areas such as real time, safety-related and security-related applications.

This standard explains the relevant principles for functional safety communications with reference to IEC 61508 series and specifies several safety communication layers (profiles and corresponding protocols) based on the communication profiles and protocol layers of IEC 61784-1, IEC 61784-2 and the IEC 61158 series. It does not cover electrical safety and intrinsic safety aspects.

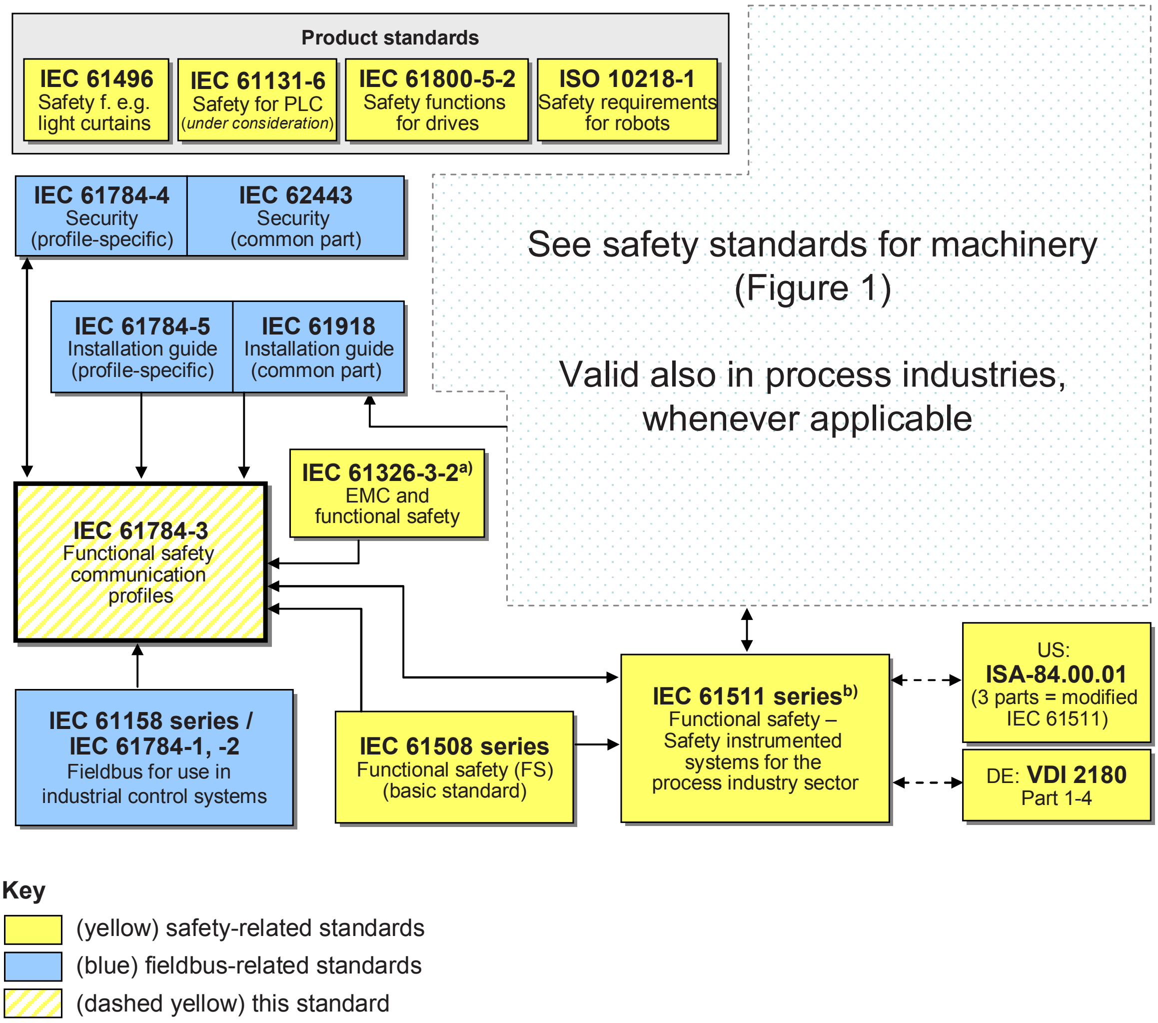
Figure 1 shows the relationships between this standard and relevant safety and fieldbus standards in a machinery environment.



NOTE Subclauses 6.7.6.4 (high complexity) and 6.7.8.1.6 (low complexity) of IEC 62061 specify the relationship between PL (Category) and SIL.

Figure 1 – Relationships of IEC 61784-3 with other standards (machinery)

Figure 2 shows the relationships between this standard and relevant safety and fieldbus standards in a process environment.



[a] For specified electromagnetic environments; otherwise IEC 61326-3-1.

[b] EN ratified.

Figure 2 – Relationships of IEC 61784-3 with other standards (process)

Safety communication layers which are implemented as parts of safety-related systems according to IEC 61508 series provide the necessary confidence in the transportation of messages (information) between two or more participants on a fieldbus in a safety-related system, or sufficient confidence of safe behaviour in the event of fieldbus errors or failures.

Safety communication layers specified in this standard do this in such a way that a fieldbus can be used for applications requiring functional safety up to the Safety Integrity Level (SIL) specified by its corresponding functional safety communication profile.

The resulting SIL claim of a system depends on the implementation of the selected functional safety communication profile within this system – implementation of a functional safety communication profile in a standard device is not sufficient to qualify it as a safety device.

This standard describes:

- basic principles for implementing the requirements of IEC 61508 series for safety-related data communications, including possible transmission faults, remedial measures and considerations affecting data integrity;
- individual description of functional safety profiles for several communication profile families in IEC 61784-1 and IEC 61784-2;
- safety layer extensions to the communication service and protocols sections of the IEC 61158 series.

0.2 Patent declaration

The International Electrotechnical Commission (IEC) draws attention to the fact that it is claimed that compliance with this document may involve the use of patents concerning the functional safety communication profiles for family 12 as follows, where the [xx] notation indicates the holder of the patent right:

DE 10 2004 044 764.0	[BE]	Datenübertragungsverfahren und Automatisierungssystem zum Einsatz eines solchen Datenübertragungsverfahrens
EP 05 733 921.0	[BE]	Sicherheitssteuerung

IEC takes no position concerning the evidence, validity and scope of these patent rights.

The holders of these patents rights have assured the IEC that they are willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holders of these patent rights are registered with IEC.

Information may be obtained from:

[BE] Beckhoff Automation GmbH
Eiserstrasse 5, 33415 Verl
GERMANY

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those identified above. IEC shall not be held responsible for identifying any or all such patent rights.

INDUSTRIAL COMMUNICATION NETWORKS – PROFILES –

Part 3-12: Functional safety fieldbuses – Additional specifications for CPF 12

1 Scope

This part of the IEC 61784-3 series specifies a safety communication layer (services and protocol) based on CPF 12 of IEC 61784-2 and IEC 61158 Type 12. It identifies the principles for functional safety communications defined in IEC 61784-3 that are relevant for this safety communication layer.

NOTE 1 It does not cover electrical safety and intrinsic safety aspects. Electrical safety relates to hazards such as electrical shock. Intrinsic safety relates to hazards associated with potentially explosive atmospheres.

This part[1] defines mechanisms for the transmission of safety-relevant messages among participants within a distributed network using fieldbus technology in accordance with the requirements of IEC 61508 series[2] for functional safety. These mechanisms may be used in various industrial applications such as process control, manufacturing automation and machinery.

This part provides guidelines for both developers and assessors of compliant devices and systems.

NOTE 2 The resulting SIL claim of a system depends on the implementation of the selected functional safety communication profile within this system – implementation of a functional safety communication profile according to this part in a standard device is not sufficient to qualify it as a safety device.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IEC 60204-1, *Safety of machinery – Electrical equipment of machines – Part 1: General requirements*

IEC 61000-6-2, *Electromagnetic compatibility (EMC) – Part 6-2: Generic standards – Immunity for industrial environments*

IEC 61131-2, *Programmable controllers – Part 2: Equipment requirements and tests*

IEC 61158-2, *Industrial communication networks – Fieldbus specifications – Part 2: Physical layer specification and service definition*

IEC 61158-3-12, *Industrial communication networks – Fieldbus specifications – Part 3-12: Data-link layer service definition – Type 12 elements*

---

1 In the following pages of this standard, “this part” will be used for “this part of the IEC 61784-3 series”.

2 In the following pages of this standard, “IEC 61508” will be used for “IEC 61508 series”.

IEC 61158-4-12, *Industrial communication networks – Fieldbus specifications – Part 4-12: Data-link layer protocol specification – Type 12 elements*

IEC 61158-5-12, *Industrial communication networks – Fieldbus specifications – Part 5-12: Application layer service definition – Type 12 elements*

IEC 61158-6-12, *Industrial communication networks – Fieldbus specifications – Part 6-12: Application layer protocol specification – Type 12 elements*

IEC 61326-3-1, *Electrical equipment for measurement, control and laboratory use – EMC requirements – Part 3-1: Immunity requirements for safety-related systems and for equipment intended to perform safety related functions (functional safety) – General industrial applications*

IEC 61326-3-2, *Electrical equipment for measurement, control and laboratory use – EMC requirements – Part 3-2: Immunity requirements for safety-related systems and for equipment intended to perform safety related functions (functional safety) – Industrial applications with specified electromagnetic environment*

IEC 61508 (all parts), *Functional safety of electrical/electronic/programmable electronic safety-related systems*

IEC 61784-2, *Industrial communication networks – Profiles – Part 2: Additional fieldbus profiles for real-time networks based on ISO/IEC 8802-3*

IEC 61784-3:2010[3], *Industrial communication networks – Profiles – Part 3: Functional safety fieldbuses – General rules and profile definitions*

IEC 61918, *Industrial communication networks – Installation of communication networks in industrial premises*

3 Terms, definitions, symbols, abbreviated terms and conventions

3.1 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

3.1.1 Common terms and definitions

3.1.1.1
availability
probability for an automated system that for a given period of time there are no unsatisfactory system conditions such as loss of production

3.1.1.2
black channel
communication channel without available evidence of design or validation according to IEC 61508

3.1.1.3
communication channel
logical connection between two end-points within a *communication system*

3 In preparation.

3.1.1.4
communication system
arrangement of hardware, software and propagation media to allow the transfer of *messages* (ISO/IEC 7498 application layer) from one application to another

3.1.1.5
connection
logical binding between two application objects within the same or different devices

3.1.1.6
Cyclic Redundancy Check (CRC)
<value> redundant data derived from, and stored or transmitted together with, a block of data in order to detect data corruption

<method> procedure used to calculate the redundant data

NOTE 1 Terms “CRC code” and "CRC signature", and labels such as CRC1, CRC2, may also be used in this standard to refer to the redundant data.

NOTE 2 See also [34], [35][4].

3.1.1.7
error
discrepancy between a computed, observed or measured value or condition and the true, specified or theoretically correct value or condition

[IEC 61508-4:2010[5]], [IEC 61158]

NOTE 1 Errors may be due to design mistakes within hardware/software and/or corrupted information due to electromagnetic interference and/or other effects.

NOTE 2 Errors do not necessarily result in a *failure* or a *fault*.

3.1.1.8
failure
termination of the ability of a functional unit to perform a required function or operation of a functional unit in any way other than as required

NOTE 1 The definition in IEC 61508-4 is the same, with additional notes.

[IEC 61508-4:2010, modified], [ISO/IEC 2382-14.01.11, modified]

NOTE 2 Failure may be due to an *error* (for example, problem with hardware/software design or message disruption)

3.1.1.9
fault
abnormal condition that may cause a reduction in, or loss of, the capability of a functional unit to perform a required function

NOTE IEV 191-05-01 defines “fault” as a state characterized by the inability to perform a required function, excluding the inability during preventive maintenance or other planned actions, or due to lack of external resources.

[IEC 61508-4:2010, modified], [ISO/IEC 2382-14.01.10, modified]

---

4 Figures in square brackets refer to the bibliography.

5 To be published.

3.1.1.10
fieldbus
communication system based on serial data transfer and used in industrial automation or process control applications

3.1.1.11
fieldbus system
system using a *fieldbus* with connected devices

3.1.1.12
frame
denigrated synonym for DLPDU

3.1.1.13
Frame Check Sequence (FCS)
redundant data derived from a block of data within a DLPDU (frame), using a hash function, and stored or transmitted together with the block of data, in order to detect data corruption

NOTE 1 An FCS can be derived using for example a CRC or other hash function.

NOTE 2 See also [34], [35].

3.1.1.14
hash function
(mathematical) function that maps values from a (possibly very) large set of values into a (usually) smaller range of values

NOTE 1 Hash functions can be used to detect data corruption.

NOTE 2 Common hash functions include parity, checksum or CRC.

[IEC/TR 62210, modified]

3.1.1.15
hazard
state or set of conditions of a system that, together with other related conditions will inevitably lead to harm to persons, property or environment

3.1.1.16
master
active communication entity able to initiate and schedule communication activities by other stations which may be masters or slaves

3.1.1.17
message
ordered series of octets intended to convey information
[ISO/IEC 2382-16.02.01, modified]

3.1.1.18
performance level (PL)
discrete level used to specify the ability of safety-related parts of control systems to perform a safety function under foreseeable conditions
[ISO 13849-1]

3.1.1.19
protective extra-low-voltage (PELV)
electrical circuit in which the voltage cannot exceed a.c. 30 V r.m.s., 42,4 V peak or d.c. 60 V in normal and single-fault condition, except earth faults in other circuits

NOTE A PELV circuit is similar to an SELV circuit that is connected to protective earth.

[IEC 61131-2]

3.1.1.20
redundancy
existence of means, in addition to the means which would be sufficient for a functional unit to perform a required function or for data to represent information

NOTE The definition in IEC 61508-4 is the same, with additional example and notes.

[IEC 61508-4:2010, modified], [ISO/IEC 2382-14.01.12, modified]

3.1.1.21
reliability
probability that an automated system can perform a required function under given conditions for a given time interval (t1,t2)

NOTE 1 It is generally assumed that the automated system is in a state to perform this required function at the beginning of the time interval.

NOTE 2 The term "reliability" is also used to denote the reliability performance quantified by this probability.

NOTE 3 Within the MTBF or MTTF period of time, the probability that an automated system will perform a required function under given conditions is decreasing.

NOTE 4 Reliability differs from availability.

[IEC 62059-11, modified]

3.1.1.22
risk
combination of the probability of occurrence of harm and the severity of that harm

NOTE For more discussion on this concept see Annex A of IEC 61508-5:2010[6].

[IEC 61508-4:2010], [ISO/IEC Guide 51:1999, definition 3.2]

3.1.1.23
safety communication layer (SCL)
communication layer that includes all the necessary measures to ensure safe transmission of data in accordance with the requirements of IEC 61508

3.1.1.24
safety data
data transmitted across a safety network using a safety protocol

NOTE The Safety Communication Layer does not ensure safety of the data itself, only that the data is transmitted safely.

3.1.1.25
safety device
device designed in accordance with IEC 61508 and which implements the functional safety communication profile

3.1.1.26
safety extra-low-voltage (SELV)
electrical circuit in which the voltage cannot exceed a.c. 30 V r.m.s., 42,4 V peak or d.c. 60 V in normal and single-fault condition, including earth faults in other circuits

NOTE An SELV circuit is not connected to protective earth.

---

6 To be published.

[IEC 61131-2]

3.1.1.27
safety function
function to be implemented by an E/E/PE safety-related system or other risk reduction measures, that is intended to achieve or maintain a safe state for the EUC, in respect of a specific hazardous event

NOTE The definition in IEC 61508-4 is the same, with an additional example and reference.

[IEC 61508-4:2010, modified]

3.1.1.28
safety function response time
worst case elapsed time following an actuation of a safety sensor connected to a fieldbus, before the corresponding safe state of its safety actuator(s) is achieved in the presence of errors or failures in the safety function channel

NOTE This concept is introduced in IEC 61784-3:2010[7], 5.2.4 and addressed by the functional safety communication profiles defined in this part.

3.1.1.29
safety integrity level (SIL)
discrete level (one out of a possible four), corresponding to a range of safety integrity values, where safety integrity level 4 has the highest level of safety integrity and safety integrity level 1 has the lowest

NOTE 1 The target failure measures (see IEC 61508-4:2010, 3.5.17) for the four safety integrity levels are specified in Tables 2 and 3 of IEC 61508-1:2010[8].

NOTE 2 Safety integrity levels are used for specifying the safety integrity requirements of the safety functions to be allocated to the E/E/PE safety-related systems.

NOTE 3 A safety integrity level (SIL) is not a property of a system, subsystem, element or component. The correct interpretation of the phrase “SILn safety-related system” (where n is 1, 2, 3 or 4) is that the system is potentially capable of supporting safety functions with a safety integrity level up to n.

[IEC 61508-4:2010]

3.1.1.30
safety measure
<this standard> measure to control possible communication *errors* that is designed and implemented in compliance with the requirements of IEC 61508

NOTE 1 In practice, several safety measures are combined to achieve the required safety integrity level.

NOTE 2 Communication *errors* and related safety measures are detailed in IEC 61784-3:2010, 5.3 and 5.4.

3.1.1.31
safety-related application
programs designed in accordance with IEC 61508 to meet the SIL requirements of the application

3.1.1.32
safety-related system
system performing *safety functions* according to IEC 61508

---

7 In preparation.

8 To be published.

3.1.1.33
slave
passive communication entity able to receive messages and send them in response to another communication entity which may be a master or a slave

3.1.2 CPF 12: Additional terms and definitions

3.1.2.1
fail-safe data
expression for data that are set to a predefined value in case of initialization or error

NOTE In this part, the value of the fail-safe data should always be set to "0".

3.1.2.2
FSoE Connection
unique relationship between the FSoE Master and an FSoE Slave

3.1.2.3
FSoE Cycle
communication cycle with one Safety Master PDU and the corresponding Safety Slave PDU

3.1.2.4
SafeInput
safety process data transferred from the FSoE Slave to the FSoE Master

3.1.2.5
SafeOutput
safety process data transferred from the FSoE Master to the FSoE Slave

3.1.2.6
Safety Master PDU
safety PDU transferred from the FSoE Master to the FSoE Slave

3.1.2.7
Safety Slave PDU
safety PDU transferred from the FSoE Slave to the FSoE Master

3.2 Symbols and abbreviated terms

3.2.1 Common symbols and abbreviated terms

CP	Communication Profile	[IEC 61784-1]
CPF	Communication Profile Family	[IEC 61784-1]
CRC	Cyclic Redundancy Check	
DLL	Data Link Layer	[ISO/IEC 7498-1]
DLPDU	Data Link Protocol Data Unit	
EMC	Electromagnetic Compatibility	
EUC	Equipment Under Control	[IEC 61508-4:2010]
E/E/PE	Electrical/Electronic/Programmable Electronic	[IEC 61508-4:2010]
FAL	Fieldbus Application Layer	[IEC 61158-5]
FCS	Frame Check Sequence	
FS	Functional Safety	
FSCP	Functional Safety Communication Profile	
MTBF	Mean Time Between Failures	
MTTF	Mean Time To Failure	
PDU	Protocol Data Unit	[ISO/IEC 7498-1]

PELV	Protective Extra Low Voltage	
PhL	Physical Layer	[ISO/IEC 7498-1]
PL	Performance Level	[ISO 13849-1]
PLC	Programmable Logic Controller	
SCL	Safety Communication Layer	
SELV	Safety Extra Low Voltage	
SIL	Safety Integrity Level	[IEC 61508-4:2010]

3.2.2 CPF 12: Additional symbols and abbreviated terms

ASIC	Application specific integrated circuit	
FSoE	Failsafe over CPF 12	
ID	Identifier	
UML	Unified Modeling Language	[ISO/IEC 19501]

3.3 Conventions

The conventions used for the descriptions of objects services and protocols are described in IEC 61158-3-12, IEC 61158-4-12, IEC 61158-5-12 and IEC 61118-6-12.

As appropriate, this part uses flow charts and UML Sequence Diagrams to describe concepts.

In state diagrams states are represented as boxes, state transitions are shown as arrows. Names of states and transitions of the state diagram correspond to the names in the textual listing of the state transitions.

The textual listing of the state transitions is structured as follows, see also Table 1.

The first row contains the name of the transition. The second row contains the condition for the transition. The third row contains the action(s) that shall take place. The last row contains the next state.

Table 1 – State machine description elements

Transition	Condition	Action	Next State

Each state with its transitions is described in a separate subclause. For each event that can occur in a state a separate subclause is inserted.

4 Overview of FSCP 12/1 (Safety-over-EtherCAT™)

Communication Profile Family 12 (commonly known as EtherCAT™[9]) defines communication profiles based on IEC 61158-2 Type 12, IEC 61158-3-12, IEC 61158-4-12, IEC 61158-5-12 and IEC 61158-6-12.

9 EtherCAT™ and Safety-over-EtherCAT™ are trade names of Beckhoff, Verl. This information is given for the convenience of users of this International Standard and does not constitute an endorsement by IEC of the trade name holder or any of its products. Compliance to this standard does not require use of the trade names EtherCAT™ or Safety-over-EtherCAT™. Use of the trade names EtherCAT™ or Safety-over-EtherCAT™ requires permission of Beckhoff, Verl.

The basic profile(s) CP 12/1 and CP 12/2 are defined in IEC 61784-2. The CPF 12 functional safety communication profile FSCP 12/1 (Safety-over-EtherCAT™[9]) is based on the CPF 12 basic profiles in IEC 61784-2 and the safety communication layer specifications defined in this part.

FSCP 12/1 describes a protocol for transferring safety data up to SIL3 between FSCP 12/1 devices. Safety PDUs are transferred by a subordinate fieldbus that is not included in the safety considerations, since it can be regarded as a black channel. The Safety PDU exchanged between two communication partners is regarded by the subordinate fieldbus as process data that are exchanged cyclically.

FSCP 12/1 uses a unique master/slave relationship between the FSoE Master and an FSoE Slave; it is called FSoE Connection (Figure 3).In the FSoE Connection, each device only returns its own new message once a new message has been received from the partner device. The complete transfer path between FSoE Master and FSoE Slave is monitored by a separate watchdog timer on both devices, in each FSoE Cycle.

The FSoE Master can handle more than one FSoE Connection to support several FSoE Slaves.

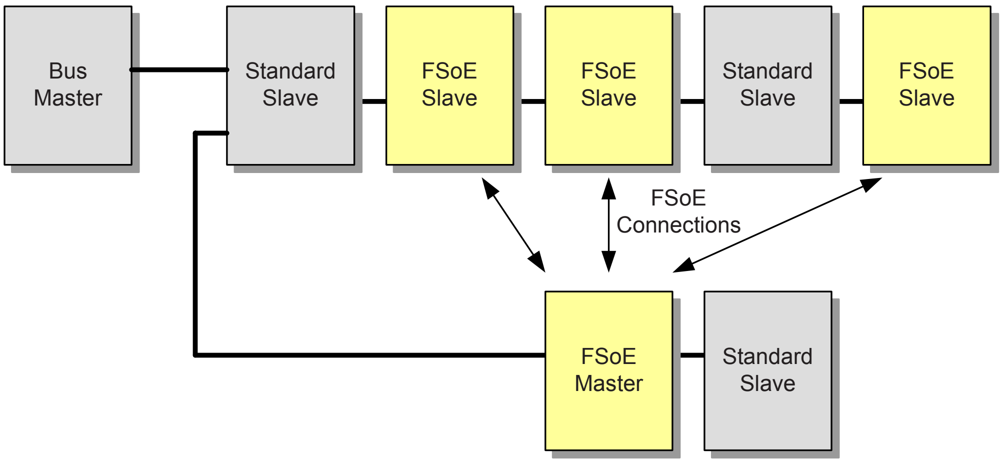


Figure 3 – Basic FSCP 12/1 system

The integrity of the safety data transfers is ensured as follows:

- session-number for detecting buffering of a complete startup sequence;
- sequence number for detecting interchange, repetition, insertion or loss of whole messages;
- unique connection identification for safely detecting misrouted messages via a unique address relationship;
- watchdog monitoring for safely detecting delays not allowed on the communication path
- cyclic redundancy checking for data integrity for detecting message corruption from source to sink.

State transitions are initiated by the FSoE Master and acknowledged by the FSoE Slave. The FSoE state machine also involves exchange and checking of information for the communication relation.

5 General

5.1 External document providing specifications for the profile

The following document is useful in understanding the design of FSCP 12/1 protocol:

- GS-ET-26 [33]

5.2 Safety functional requirements

The following requirements shall apply to the development of devices that implement the FSCP 12/1 protocol. The same requirements were used in the development of FSCP 12/1.

- The FSCP 12/1 protocol is designed to support Safety Integrity Level 3 (SIL 3) (see IEC 61508).
- Implementations of FSCP 12/1 shall comply with IEC 61508.
- The basic requirements for the development of the FSCP 12/1 protocol are defined in IEC 61784-3.
- FSCP 12/1 protocol is implemented using a black channel approach; there is no safety related dependency on the standard CPF 12 communication profiles. Transmission equipment such as controllers, ASICs, links, couplers, etc. shall remain unmodified.
- Environmental conditions shall be according to general automation requirements mainly IEC 61326-3-1 for the safety margin tests, unless there are specific product standards.
- Safety communication and non safety relevant communication shall be independent. However, non safety relevant devices and safety devices shall be able to use the same communication channel.
- Implementation of the FSCP 12/1 protocol shall be restricted to the communication end devices (FSoE Master and FSoE Slave).
- There shall always be a 1:1 communication relationship between an FSoE Slave and its FSoE Master.
- The safety communication shall not restrict the minimum cycle time of the communication system.

5.3 Safety measures

The safety measures used in the FSCP 12/1 to detect communication errors are listed in Table 2. The safety measures shall be processed and monitored within each safety device.

Table 2 – Communication errors and detection measures

Communication errors	Safety measures				
	Sequence number (see 7.1.3.4)	Time expectation (see 6.2) [a]	Connection authentication (see 7.2.2.4) [b]	Feedback Message (see 7.2.1)	Data integrity assurance (see 7.1.3)
Corruption					X
Unintended repetition	X				X
Incorrect sequence	X				X
Loss	X	X		X	X
Unacceptable delay		X		X	X
Insertion	X				X
Masquerade		X		X	X
Addressing			X		
Revolving memory failures within switches	X				X

[a] In this standard the instance is called "FSoE Watchdog".

[b] In this standard the instance is called "FSoE Connection ID".

5.4 Safety communication layer structure

The FSCP 12/1 protocol is layered on top of the standard network protocol. Figure 4 shows how the protocol is related to the CPF 12 layer. The safety layer accept safety data from the safety-related application and transfers these data via the FSCP 12/1 protocol.

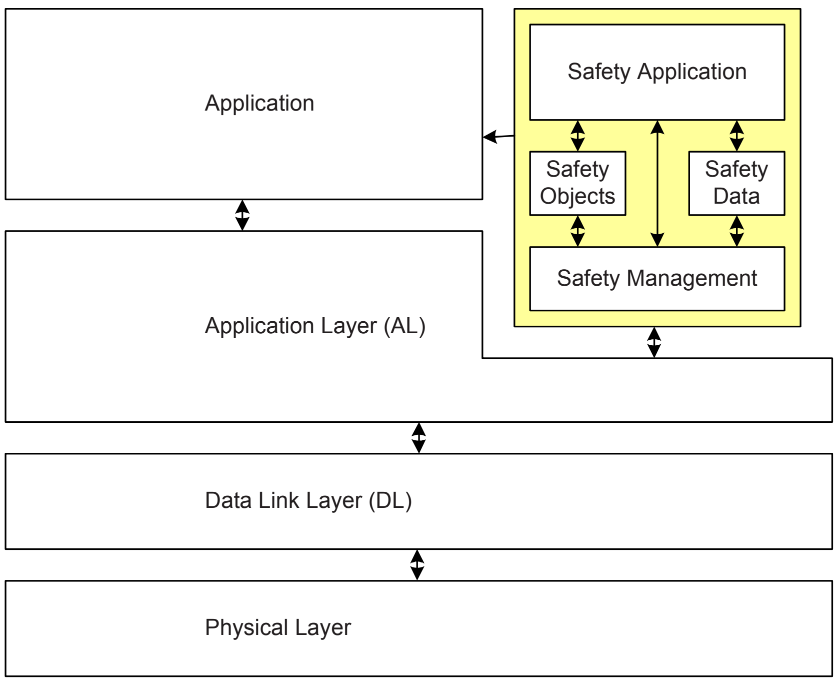


Figure 4 – FSCP 12/1 software architecture

A safety PDU containing the safety data and the required error detection measures is included in the communication process data objects (PDO). The mapping in the process data of the communication system and the start-up of the communication state machine is not part of the safety protocol.

The calculation of the residual error probability for the FSCP 12/1 protocol takes no credit of the error detection mechanisms of the communication system. This means that the protocol can also be transferred via other communication systems. Any transmission link can be used, including fieldbus systems, Ethernet or similar transfer routes, optical fibre cables, copper cables, or even radio links.

5.5 Relationships with FAL (and DLL, PhL)

5.5.1 General

This safety communication layer is designed to be used in conjunction with CPF 12 communication profiles. But it is not restricted to this communication profile.

5.5.2 Data types

Profiles defined in this part support all the CPF 12 data types as defined in IEC 61158-5-12.

6 Safety communication layer services

6.1 FSoE Connection

The connection between two FSCP 12/1 communication partners (FSCP 12/1 nodes) is referred to as FSoE Connection. In an FSoE Connection one communication partner is always the FSoE Master, the other one the FSoE Slave.

The FSoE Master initialises the FSoE Connection after power-on or after a communication fault, while the FSoE Slave is limited to responses. The FSoE Master sets the safety-related communication parameters and optionally the safety-related application parameters of the FSoE Slave.

The safety process data transferred from the FSoE Master to the FSoE Slave are referred to as SafeOutputs. The safety data transferred from the FSoE Slave to the FSoE Master are referred to as SafeInputs.

The Safety PDU transferred from the FSoE Master to the FSoE Slave is referred to as Safety Master PDU. The Safety PDU transferred from the FSoE Slave to the FSoE Master is referred to as Safety Slave PDU.

6.2 FSoE Cycle

The FSoE Master sends the Safety Master PDU to the FSoE Slave and starts the FSoE Watchdog.

After checking the integrity of the Safety PDU, the FSoE Slave transfers the SafeOutputs to the Safety Application. It calculates the Safety Slave PDU with the SafeInputs from the Safety Application and sends this PDU to the FSoE Master. The FSoE Slave also starts its FSoE watchdog. This is shown in Figure 5.

After receiving a valid Safety Slave PDU an FSoE Cycle is finished.

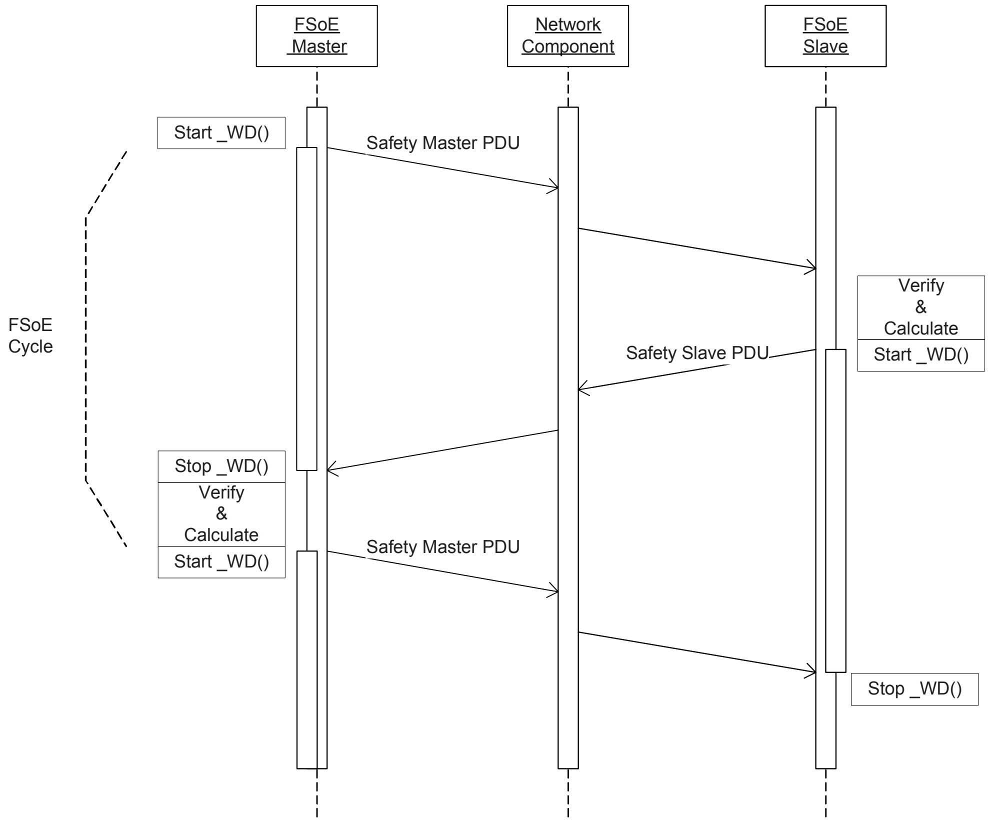


Figure 5 – FSoE Cycle

6.3 FSoE services

For each FSoE Connection, the FSoE Master shall support an FSoE Master handler to control the associated FSoE Slave.

For FSoE Master to FSoE Master Communication, the FSoE Master should support one or several FSoE Slave handler. Figure 6 shows the possible FSoE functionalities in the FSoE Master and FSoE Slave devices.

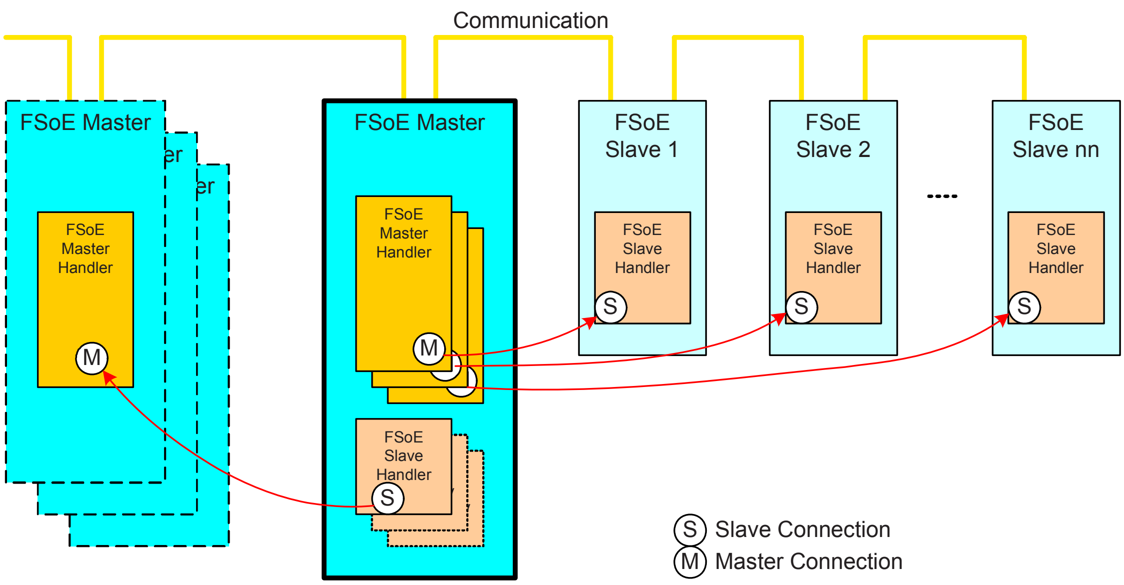


Figure 6 – FSCP 12/1 communication structure

7 Safety communication layer protocol

7.1 Safety PDU format

7.1.1 Safety PDU structure

Figure 7 shows the structure of one safety PDU embedded in a Type 12 PDU. In Table 3 the general structure of the Safety PDU is listed.

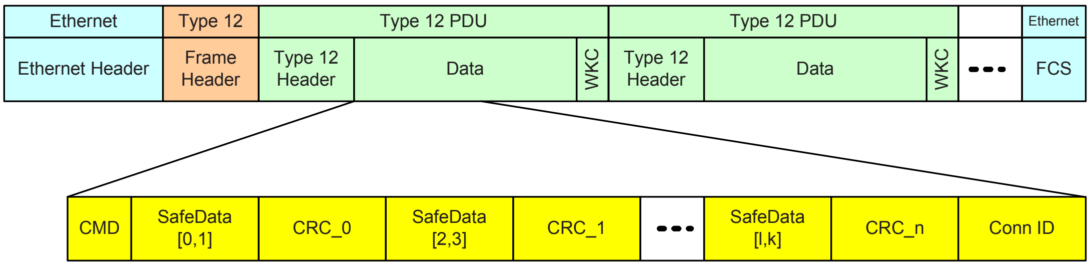


Figure 7 – Safety PDU for CPF 12 embedded in Type 12 PDU

The Safety PDU is cyclically transferred via the subordinate fieldbus. Each FSCP 12/1 node detects a new Safety PDU if at least one bit within the Safety PDU has changed.

The Safety PDU has a variable length specified in the device description of the FSoE Slave. The safety data length can be 1 octet or an even number of octets. The safety data length can be different in input and output direction.

The shorter of the two safety data lengths in the Safety Master PDU and the Safety Slave PDU determines how many safety data are used during the initialisation phase of the FSoE Connection with parameter information. In the longer of the two PDUs the remaining safety data are assigned zero.

Table 3 – General Safety PDU

Octet	Name	Description
0	Command	Command
1	SafeData[0]	safety data, octet 0
2	SafeData[1]	safety data, octet 1
3	CRC_0_Lo	low octet (bits 0-7) of the 16-bit CRC_0
4	CRC_0_Hi	high octet (bits 8-15) of the 16-bit CRC_0
5	SafeData[2]	safety data, octet 2
6	SafeData[3]	safety data, octet 3
7	CRC_1_Lo	low octet (bits 0-7) of the 16-bit CRC_1
8	CRC_1_Hi	high octet (bits 8-15) of the 16-bit CRC_1
…	…	…
(n-1) × 2-1	SafeData[n-2]	safety data, octet n-2
(n-1) × 2	SafeData[n-1]	safety data, octet n-1
(n-1) × 2+1	CRC_(n-2)/2_Lo	low octet (bits 0-7) of the 16-bit CRC_(n-2)/2
(n-1) × 2+2	CRC_(n-2)/2_Hi	high octet (bits 8-15) of the 16-bit CRC_(n-2)/2
(n-1) × 2+3	Conn_Id_Lo	unique connection ID, low octet
(n-1) × 2+4	Conn_Id_Hi	unique connection ID, high octet

The Safety PDU can transfer n safety data octets. Two octets of data are transferred by a 2-octet CRC.

The shortest Safety PDU consists of 6 octets, which can be used to transfer 1 octet of safety data, as shown in Table 4.

Table 4 – Shortest Safety PDU

Octet	Name	Description
0	Command	Command
1	SafeData[0]	safety data, octet 0
2	CRC_0_Lo	low octet (bits 0-7) of the 16-bit CRC_0
3	CRC_0_Hi	high octet (bits 8-15) of the 16-bit CRC_0
4	Conn_Id_Lo	unique connection ID, low octet
5	Conn_Id_Hi	unique connection ID, high octet

7.1.2 Safety PDU command

The Safety PDU command determines the meaning of the safety data based on the scheme shown in Table 5.

Table 5 – Safety PDU command

Command	Description
0x36	ProcessData
0x2A	Reset
0x4E	Session
0x64	Connection
0x52	Parameter
0x08	FailSafeData

7.1.3 Safety PDU CRC

7.1.3.1 CRC calculation

Two octets of safety data are transferred by a corresponding two octet CRC.

In addition to the transferred data (command, data, ConnID), the CRC_0 of the Safety PDU also includes a virtual sequence number, the CRC_0 of the last received Safety PDU and three additional zero octets. If only one octet safety data is transferred, the SafeData[1] is skipped in the calculation.

```
CRC_0 :=     f(received CRC_0, ConnID, Sequence_Number, command, SafeData[0],
               SafeData[1], 0x000000)
```

Table 6 shows the calculation sequence for CRC_0.

Table 6 – CRC_0 calculation sequence

Step	Argument
1	received CRC_0 (bit 0-7)
2	received CRC_0 (bit 8-15)
3	ConnID (bit 0-7)
4	ConnID (bit 8-15)
5	Sequence_Number (bit 0-7)
6	Sequence_Number (bit 8-15)
7	Command
8	SafeData[0]
9	SafeData[1]
10	0
11	0
12	0

The CRC_i (0 < i <= ((n-2)/2)) of the Safety PDU additionally includes the index i of the CRC.

```
CRC_i :=      f(received CRC_0, ConnID, Sequence_Number, command, i,
              SafeData[i × 2], SafeData[i × 2 + 1], 0)
```

Table 7 shows the calculation sequence for CRC_i.

Table 7 – CRC_i calculation sequence (i>0)

Step	Argument
1	received CRC_0 (bit 0-7)
2	received CRC_0 (bit 8-15)
3	ConnID (bit 0-7)
4	ConnID (bit 8-15)
5	Sequence_Number (bit 0-7)
6	Sequence_Number (bit 8-15)
7	Command
8	Index i (bit 0-7)
9	Index i (bit 8-15)
10	SafeData[2 × i]
11	SafeData[2 × I + 1]
12	0
13	0
14	0

7.1.3.2 CRC polynomial selection

The polynomial 0x139B7 is used for calculating the CRCs and is referred to as the Safety polynomial.

In order to allow the Safety PDU to be transported via a black channel whose transfer characteristics are not included in the safety considerations, a bit fault rate of 10^{-2} shall be used for determining the residual error probability. The residual error probability shall not exceed 10^{-9}.

Safety is ensured based on the FSoE Master and the FSoE Slave switching to the reset state (i.e. safe state) as soon as an error is detected.

All CRC calculation factors except safety data have a fixed expected value, so that only safety data have to be considered in the calculation of the residual error probability.

The mathematical proof showing that the residual error probability with the Safety polynomial for 16-bit safety data and a bit fault rate of 10^{-2} does not exceed 10^{-9} is included in separate document covering quantitative fault detection.

7.1.3.3 CRC inheritance

Inclusion (inheritance) of the CRC_0 of the last received telegram in the CRC calculation ensures that two consecutive Safety Master PDUs or Safety Slave PDUs differ, even if the other data remain unchanged.

Inheritance of CRC_0 also ensures safe and consistent transfer of data that are distributed over several Type 12 PDUs due to their length.

The CRC_0 of the received Safety PDU is included in the calculation of all CRC_i for the Safety PDU to be sent.

Table 8 shows an example for CRC_0 inheritance.

Table 8 – Example for CRC_0 inheritance

FSoE Cycle	FSoE Master		FSoE Slave	
	old CRC_0	new CRC_0	old CRC_0	new CRC_0
j-1	CRC_0 (2 × j - 3)	CRC_0 (2 × j - 2)	CRC_0 (2 × j - 2)	CRC_0 (2 × j - 1)
j	CRC_0 (2 × j - 1)	CRC_0 (2 × j)	CRC_0 (2 × j)	CRC_0 (2 × j + 1)
j+1	CRC_0 (2 × j + 1)	CRC_0 (2 × j + 2)	CRC_0 (2 × j + 2)	CRC_0 (2 × j + 3)

In FSoE Cycle j the FSoE Master receives a Safety Slave PDU with CRC_0 (2 × j - 1). Since the value of CRC_0 (2 × j - 2), which was included in the calculation of CRC_0 (2 × j - 1), was calculated by the FSoE Master in FSoE Cycle (j – 1), the FSoE Master can check CRC_0 (2 × j - 1) in the Safety Slave PDU.

In turn, in FSoE Cycle j, the FSoE Slave receives the Safety Master PDU with CRC_0 (2 × j) and is also able to check this PDU, since CRC_0 (2 × j - 1) calculated by the FSoE Slave in FSoE Cycle (j – 1) was included in the calculation.

7.1.3.4 Sequence number

In Table 8 CRC_0 (2 × j) may equal CRC_0 (2 × j - 2). With short Safety PDUs this could lead to the Safety Master PDU in FSoE Cycle (j – 1) being the same as the Safety Master PDU in FSoE Cycle j, with the result that the FSoE Slave would not recognise the Safety Master PDU as a new PDU in FSoE Cycle j and the FSoE Watchdog would be triggered.

The CRCs of the Safety Master PDU therefore includes a virtual 16-bit master sequence number, which the FSoE Master increments with each new Safety Master PDU. The CRC of the Safety Slave PDU also includes a virtual 16-bit slave sequence number, which is incremented by the FSoE Slave with each new Safety Slave PDU.

If CRC_0 (2 × j) equals CRC_0 (2 × j - 2) despite these measures, the master sequence number is incremented further until CRC_0 (2 × j) is not equal CRC_0 (2 × j - 2). This algorithm is used both for generating the Safety Master PDU by the FSoE Master and for checking the Safety Master PDU by the FSoE Slave.

If CRC_0 (2 × j + 1) equals CRC_0 (2 × j - 1), the slave sequence number is incremented further until CRC_0 (2 × j + 1) is not equal CRC_0 (2 × j - 1). This algorithm is used both for generating the Safety Slave PDU by the FSoE Slave and for checking the Safety Slave PDU by the FSoE Master.

The sequence number can assume values between 1 and 65 535. After 65 535 the sequence starts again with 1, i.e. 0 is left out.

7.1.3.5 CRC index

If more than 2 octets of safety data and therefore 2 or several CRCs (for example CRC_0 and CRC_1) are transferred, the measures described above are not sufficient for detecting all reversal options within a Safety PDU, see example in Table 9.

Table 9 – Example for 4 octets of safety data with interchanging of octets 1-4 with 5-8

Octet	Name	Description
0	Command	Command
1	SafeData[2]	safety data, octet 2
2	SafeData[3]	safety data, octet 3
3	CRC_1_Lo	low octet (bits 0-7) of the 16-bit CRC_1
4	CRC_1_Hi	high octet (bits 8-15) of the 16-bit CRC_1
5	SafeData[0]	safety data, octet 0
6	SafeData[1]	safety data, octet 1
7	CRC_0_Lo	low octet (bits 0-7) of the 16-bit CRC_0
8	CRC_0_Hi	high octet (bits 8-15) of the 16-bit CRC_0
9	Conn_Id_Lo	low octet (bits 0-7) of the unique connection ID
10	Conn_Id_Hi	low octet (bits 0-7) of the unique connection ID

Index i (2 octect value) is therefore also included in the respective CRC_i. This enables detection of reversal of octets 1-4 and 5-8.

7.1.3.6 Additional zero octets

The residual error probability is calculated via the ratio of detected errors and undetected errors. Undetected errors are essentially errors already detected by the CRC polynomial for the black channel (standard polynomial), since these errors are not apparent in the Safety layer due to the fact that they are filtered out beforehand. The worst case ratio between detected errors (errors that are not detected by the standard polynomial but by the CRC polynomial of the Safety layer) and undetected errors (errors already detected by the standard polynomial) occurs if the standard polynomial is divisible by the Safety polynomial.

Three zero octets are included in the calculation in order to ensure adequate independence between the two polynomials in this case.

7.1.3.7 Session ID

Particularly in fieldbuses transferred via Ethernet, a faulty device storing telegrams (for example a switch) may lead to a correct sequence of telegrams being inserted at the wrong time. CRC inheritance means that a Safety PDU sequence always depends on the history.

Transferring a randomly generated session ID during setup of the FSoE Connection ensures that two Safety PDU sequences differ after power-on.

The session ID can assume values between 0 and 65 535.

7.2 FSCP 12/1 communication procedure

7.2.1 Message cycle

FSCP 12/1 communication operates with an acknowledged message cycle (FSoE Cycle), i.e. the FSoE Master sends a Safety Master PDU to the FSoE Slave and expects a Safety Slave PDU back. Only then is the next Safety Master PDU generated.

7.2.2 FSCP 12/1 node states

7.2.2.1 General

While the FSoE Connection is established, the FSCP 12/1 nodes take on different states before the safety data become valid and the safely state is exited.

Figure 8 shows the FSoE node states.

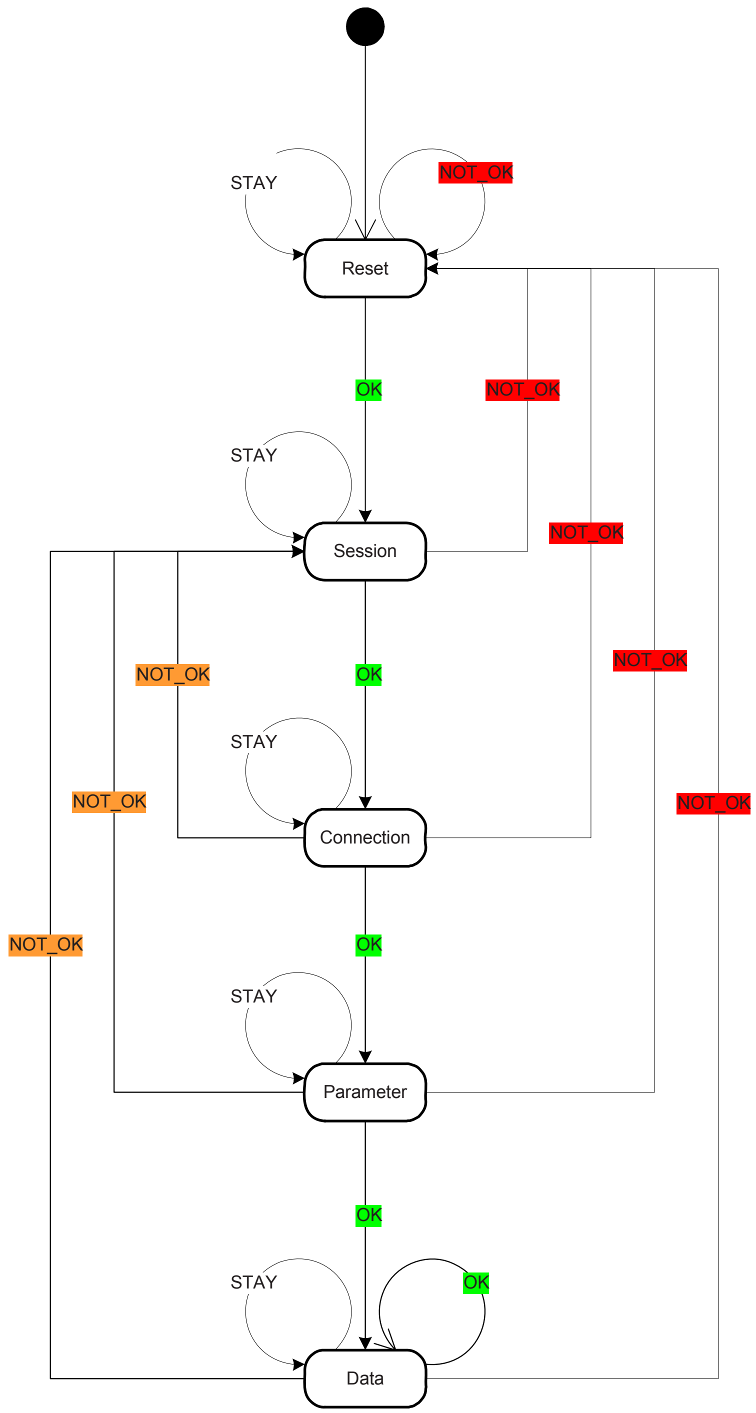


Figure 8 – FSCP 12/1 node states

After a power-on or an FSoE communication error the FSoE Master and Slave are in the reset state. The FSoE nodes also switch to reset-state if they detect an error in the communication or the local application. After an FSoE Reset command from the FSoE Slave, the FSoE Master switches to session state (transitions in orange). After a reset command from the FSoE Master, the FSoE Slave switches to reset state. The data state can then be assumed via the session, connection and parameter states. The safe output state can only be exited in the data state.

7.2.2.2 Reset state

The reset state is used to re-initialise the FSoE Connection after the power-on or an FSoE communication error. The FSoE Master exits the reset state when it sends a Safety Master PDU with the Session command to the FSoE Slave. The FSoE Slave exits the reset state when it receives a valid Safety Master PDU with the Session command.

In the reset state the sequence number and the CRC of the last telegram used in the CRC calculation are reset.

Table 10 shows an example of the Safety Master PDU for 4 octets of safety data with the reset command.

Table 10 – Safety Master PDU for 4 octets of safety data with command = Reset after restart (reset connection) or error

Octet	Name	Description
0	Command	*Reset*
1	SafeData[0]	error code (bit 0-7), 0 for restart
2	SafeData[1]	unused (= 0)
3	CRC_0_Lo	low octet (bits 0-7) of the 16-bit CRC_0
4	CRC_0_Hi	high octet (bits 8-15) of the 16-bit CRC_0
5	SafeData[2]	unused (= 0)
6	SafeData[3]	unused (= 0)
7	CRC_1_Lo	low octet (bits 0-7) of the 16-bit CRC_1
8	CRC_1_Hi	high octet (bits 8-15) of the 16-bit CRC_1
9	Conn_Id_Lo	unused (= 0)
10	Conn_Id_Hi	unused (= 0)

The FSoE Slave acknowledges Reset command by setting the SafeData to 0.

Table 11 shows an example of the Safety Slave PDU for 4 octets of safety data with the reset command.

Table 11 – Safety Slave PDU for 4 octets of safety data with command = Reset for acknowledging a Reset command from the FSoE Master

Octet	Name	Description
0	Command	*Reset*
1	SafeData[0]	0
2	SafeData[1]	0
3	CRC_0_Lo	low octet (bits 0-7) of the 16-bit CRC_0
4	CRC_0_Hi	high octet (bits 8-15) of the 16-bit CRC_0
5	SafeData[2]	unused (= 0)
6	SafeData[3]	unused (= 0)
7	CRC_1_Lo	low octet (bits 0-7) of the 16-bit CRC_1
8	CRC_1_Hi	high octet (bits 8-15) of the 16-bit CRC_1
9	Conn_Id_Lo	unused (= 0)
10	Conn_Id_Hi	unused (= 0)

The FSoE Slave also sends a Safety PDU with the Reset command during a restart (reset connection) or in the event of an error. This is shown in Table 12 as an example for 4 octets of safety data.

Table 12 – Safety Slave PDU for 4 octets of safety data with command = Reset after restart (reset connection) or error

Octet	Name	Description
0	Command	*Reset*
1	SafeData[0]	error code (bit 0-7), 0 for restart
2	SafeData[1]	unused (= 0)
3	CRC_0_Lo	low octet (bits 0-7) of the 16-bit CRC_0
4	CRC_0_Hi	high octet (bits 8-15) of the 16-bit CRC_0
5	SafeData[2]	unused (= 0)
6	SafeData[3]	unused (= 0)
7	CRC_1_Lo	low octet (bits 0-7) of the 16-bit CRC_1
8	CRC_1_Hi	high octet (bits 8-15) of the 16-bit CRC_1
9	Conn_Id_Lo	unused (= 0)
10	Conn_Id_Hi	unused (= 0)

The FSoE Master acknowledges the Reset command by sending a Safety Master PDU with the Session command.

7.2.2.3 Session state

During the transition to or in the session state, a 16-bit Master Session ID is transferred from the FSoE Master to the FSoE Slave, which in turn responds with its own Slave Session ID. Both FSoE nodes generate the Session ID as a random number that is only used to differentiate multiple Safety PDU sequences in the event of several restarts of the FSoE Connection.

Table 13 shows an example of the Safety Master PDU for 4 octets of safety data with the Session command.

Table 13 – Safety Master PDU for 4 octets of safety data with command = Session

Octet	Name	Description
0	Command	*Session*
1	SafeData[0]	*Master Session Id*, octet 0
2	SafeData[1]	*Master Session Id*, octet 1
3	CRC_0_Lo	low octet (bits 0-7) of the 16-bit CRC_0
4	CRC_0_Hi	high octet (bits 8-15) of the 16-bit CRC_0
5	SafeData[2]	unused (= 0)
6	SafeData[3]	unused (= 0)
7	CRC_1_Lo	low octet (bits 0-7) of the 16-bit CRC_1
8	CRC_1_Hi	high octet (bits 8-15) of the 16-bit CRC_1
9	Conn_Id_Lo	unused (= 0)
10	Conn_Id_Hi	unused (= 0)

The FSoE Slave acknowledges Session command by sending back the Slave Session ID. Table 14 shows an example of the Safety Slave PDU for 4 octets of SafeData with the Session command.

Table 14 – Safety Slave PDU for 4 octets of safety data with command = Session

Octet	Name	Description
0	Command	*Session*
1	SafeData[0]	*Slave Session Id*, octet 0
2	SafeData[1]	*Slave Session Id*, octet 1
3	CRC_0_Lo	low octet (bits 0-7) of the 16-bit CRC_0
4	CRC_0_Hi	high octet (bits 8-15) of the 16-bit CRC_0
5	SafeData[2]	unused (= 0)
6	SafeData[3]	unused (= 0)
7	CRC_1_Lo	low octet (bits 0-7) of the 16-bit CRC_1
8	CRC_1_Hi	high octet (bits 8-15) of the 16-bit CRC_1
9	Conn_Id_Lo	unused (= 0)
10	Conn_Id_Hi	unused (= 0)

If the Safety PDU contains at least 2 octets of safety data, the Session ID can be transferred with one FSoE Cycle. If, on the other hand, the Safety PDU only contains 1 octet of safety data, the Session ID shall be transferred with two FSoE Cycles.

The value of the Session ID has no safety relevance, i.e. a switch in the FSoE Node receiving the Safety PDU does not need to be examined from a safety perspective. The Connection ID for the Session command is therefore set to 0.

The FSoE Master exits the session state once it has transferred the complete session ID and received the associated acknowledgements from the FSoE Slave by sending a Safety PDU with the Connection command to the FSoE Slave. The FSoE Slave exits the session state when it receives a Safety PDU with the Connection command from the FSoE Master.

Both the FSoE Master and the FSoE Slave would also exit the Session state if they detect an FSoE communication error.

In the FSoE Master, after receipt of a RESET command, both the sequence number and the CRC of the last telegram used in the CRC calculation are reset.

7.2.2.4 Connection state

In the connection state, a 16-bit Connection ID is transferred from the FSoE Master to the FSoE Slave. The Connection ID shall be unique and is generated by the safety configurator of the FSoE Master. If several FSoE Masters are present in the communication system, the user shall ensure that the Connection IDs used are unique.

In addition to the 16-bit Connection ID the unique FSoE Slave Address is also transferred. Table 15 shows the content of the safety data transferred in the connection state.

Table 15 – Safety data transferred in the connection state

SafetyData Octet	Description
0	low octet (bits 0-7) of the connection ID
1	high octet (bits 8-15) of the connection ID
2	low octet (bits 0-7) of the FSoE Slave Address
3	high octet (bits 8-15) of the FSoE Slave Address

Depending on the length of the safety data, up to 4 FSoE Cycles are required. For a Safety PDU with 4 octets of safety data only one FSoE Cycle is required, this is shown in Table 16 and Table 17.

Table 16 – Safety Master PDU for 4 octets of safety data in Connection state

Octet	Name	Description
0	Command	*Connection*
1	SafeData[0]	Connection Id, low octet
2	SafeData[1]	Connection Id, high octet
3	CRC_0_Lo	low octet (bits 0-7) of the 16-bit CRC_0
4	CRC_0_Hi	high octet (bits 8-15) of the 16-bit CRC_0
5	SafeData[2]	FSoE Slave Address, low octet
6	SafeData[3]	FSoE Slave Address, high octet
7	CRC_1_Lo	low octet (bits 0-7) of the 16-bit CRC_1
8	CRC_1_Hi	high octet (bits 8-15) of the 16-bit CRC_1
9	Conn_Id_Lo	Connection Id, low octet
10	Conn_Id_Hi	Connection Id, high octet

The FSoE Slave acknowledges the Connection command by sending back the safety data.

Table 17 – Safety Slave PDU for 4 octets of safety data in Connection state

Octet	Name	Description
0	Command	*Connection*
1	SafeData[0]	Connection Id, low octet
2	SafeData[1]	Connection Id, high octet
3	CRC_0_Lo	low octet (bits 0-7) of the 16-bit CRC_0
4	CRC_0_Hi	high octet (bits 8-15) of the 16-bit CRC_0
5	SafeData[2]	FSoE Slave Address, low octet
6	SafeData[3]	FSoE Slave Address, high octet
7	CRC_1_Lo	low octet (bits 0-7) of the 16-bit CRC_1
8	CRC_1_Hi	high octet (bits 8-15) of the 16-bit CRC_1
9	Conn_Id_Lo	Connection Id, low octet
10	Conn_Id_Hi	Connection Id, high octet

The FSoE Slave Address shall be unique in the communication system. It can be set at the respective FSoE Slave device. By transferring the FSoE Slave Address together with the Connection ID, the FSoE Slave can check whether it was actually addressed, so that invalid addressing would be detected. Since the Connection ID is also unique in the communication system, the Connection ID is always sent in the following Safety PDUs, so that both the FSoE Master and the FSoE Slave can detect whether they are addressed with the telegram. The unique Connection ID therefore enables 65 535 FSoE Connections to be realised in the communication system (Connection ID = 0 is not permitted).

7.2.2.5 Parameter state

In the parameter state, safety-related communication and device specific safety-related application parameters are transferred. The latter can have any length. CRC inheritance ensures safety and consistent parameter transfer.

Table 18 shows the content of the safety data transferred in the parameter state.

Table 18 – Safety data transferred in the parameter state

Safety data Octet	Description
0	low octet (bits 0-7) length of the communication parameters in octets (= 2)
1	high octet (bits 8-15) length of the communication parameters in octets (= 0)
2	low octet (bits 0-7) of the FSoE watchdog (in ms)
3	high octet (bits 8-15) of the FSoE watchdog (in ms)
4	low octet (bits 0-7) length of the application parameters in octets
5	high octet (bits 8-15) length of the application parameters in octets
6	1st octet of the safety-related application parameter
…	
n+5	nth octet of the safety-related application parameter

The number of FSoE Cycles in the parameter state depends on the length of the safety-related application parameters and the length of the safety data in the Safety PDU. If not all safety data octets are required in the last FSoE Cycle, they shall be transferred as 0.

Two FSoE Cycles are required for a Safety PDU with 4 octets of safety data and 2 octets of safety-related application parameter; this is shown in Table 19 to Table 22.

Table 19 – First Safety Master PDU for 4 octets of safety data in parameter state

Octet	Name	Description
0	Command	*Parameter*
1	SafeData[0]	low octet (bits 0-7) length of the communication parameters in octets (= 2)
2	SafeData[1]	high octet (bits 8-15) length of the communication parameters in octets (= 0)
3	CRC_0_Lo	low octet (bits 0-7) of the 16-bit CRC_0
4	CRC_0_Hi	high octet (bits 8-15) of the 16-bit CRC_0
5	SafeData[2]	low octet (bits 0-7) of the FSoE watchdog (in ms)
6	SafeData[3]	high octet (bits 8-15) of the FSoE watchdog (in ms)
7	CRC_1_Lo	low octet (bits 0-7) of the 16-bit CRC_1
8	CRC_1_Hi	high octet (bits 8-15) of the 16-bit CRC_1
9	Conn_Id_Lo	Connection Id, low octet
10	Conn_Id_Hi	Connection Id, high octet

The FSoE Slave acknowledges a correct Parameter command by sending back the safety data.

Table 20 – First Safety Slave PDU for 4 octets of safety data in parameter state

Octet	Name	Description
0	Command	*Parameter*
1	SafeData[0]	low octet (bits 0-7) length of the communication parameters in octets (= 2)
2	SafeData[1]	high octet (bits 8-15) length of the communication parameters in octets (= 0)
3	CRC_0_Lo	low octet (bits 0-7) of the 16-bit CRC_0
4	CRC_0_Hi	high octet (bits 8-15) of the 16-bit CRC_0
5	SafeData[2]	low octet (bits 0-7) of the FSoE watchdog (in ms)
6	SafeData[3]	high octet (bits 8-15) of the FSoE watchdog (in ms)
7	CRC_1_Lo	low octet (bits 0-7) of the 16-bit CRC_1
8	CRC_1_Hi	high octet (bits 8-15) of the 16-bit CRC_1
9	Conn_Id_Lo	Connection Id, low octet
10	Conn_Id_Hi	Connection Id, high octet

The FSoE Master sends the second Safety Master PDU once it has correctly received the first Safety Slave PDU.

Table 21 – Second Safety Master PDU for 4 octets of safety data in parameter state

Octet	Name	Description
0	Command	*Parameter*
1	SafeData[0]	low octet (bits 0-7) length of the application parameters in octets (= 2)
2	SafeData[1]	high octet (bits 8-15) length of the application parameters in octets (= 0)
3	CRC_0_Lo	low octet (bits 0-7) of the 16-bit CRC_0
4	CRC_0_Hi	high octet (bits 8-15) of the 16-bit CRC_0
5	SafeData[2]	1st octet of the safety-related application parameter
6	SafeData[3]	2nd octet of the safety-related application parameter
7	CRC_1_Lo	low octet (bits 0-7) of the 16-bit CRC_1
8	CRC_1_Hi	high octet (bits 8-15) of the 16-bit CRC_1
9	Conn_Id_Lo	Connection Id, low octet
10	Conn_Id_Hi	Connection Id, high octet

The FSoE Slave acknowledges a correct Parameter command by sending back the safety data.

Table 22 – Second Safety Slave PDU for 4 octets of safety data in parameter state

Octet	Name	Description
0	Command	*Parameter*
1	SafeData[0]	low octet (bits 0-7) length of the application parameters in octets (= 2)
2	SafeData[1]	high octet (bits 8-15) length of the application parameters in octets (= 0)
3	CRC_0_Lo	low octet (bits 0-7) of the 16-bit CRC_0
4	CRC_0_Hi	high octet (bits 8-15) of the 16-bit CRC_0
5	SafeData[2]	1^{st} octet of the safety-related application parameter
6	SafeData[3]	2^{nd} octet of the safety-related application parameter
7	CRC_1_Lo	low octet (bits 0-7) of the 16-bit CRC_1
8	CRC_1_Hi	high octet (bits 8-15) of the 16-bit CRC_1
9	Conn_Id_Lo	Connection Id, low octet
10	Conn_Id_Hi	Connection Id, high octet

The FSoE Watchdog and the safety-related application parameters are configured via a safety configurator of the FSoE Master.

7.2.2.6 Data state

7.2.2.6.1 Valid data

While in the previous states the number of FSoE Cycles was fixed, in the data state FSoE Cycles are transferred until either a communication error occurs or an FSoE node is stopped locally. The FSoE Master sends SafeOutputs to the FSoE Slave.

Table 23 shows an example of the Safety Master PDU for 4 octets of SafeOutputs with the ProcessData command.

Table 23 – Safety Master PDU for 4 octets of ProcessData in data state

Octet	Name	Description
0	Command	*ProcessData*
1	SafeData[0]	1^{st} octet of SafeOutputs
2	SafeData[1]	2^{nd} octet of SafeOutputs
3	CRC_0_Lo	low octet (bits 0-7) of the 16-bit CRC_0
4	CRC_0_Hi	high octet (bits 8-15) of the 16-bit CRC_0
5	SafeData[2]	3^{rd} octet of SafeOutputs
6	SafeData[3]	4^{th} octet of SafeOutputs
7	CRC_1_Lo	low octet (bits 0-7) of the 16-bit CRC_1
8	CRC_1_Hi	high octet (bits 8-15) of the 16-bit CRC_1
9	Conn_Id_Lo	Connection Id, low octet
10	Conn_Id_Hi	Connection Id, high octet

The FSoE Slave acknowledges the Safety Master PDU and sends SafeInputs to the FSoE Master.

Table 24 shows an example of the Safety Slave PDU for 4 octets of SafeInputs with the ProcessData command.

Table 24 – Safety Slave PDU for 4 octets of ProcessData in data state

Octet	Name	Description
0	Command	*ProcessData*
1	SafeData[0]	1st octet of SafeInputs
2	SafeData[1]	2nd octet of SafeInputs
3	CRC_0_Lo	low octet (bits 0-7) of the 16-bit CRC_0
4	CRC_0_Hi	high octet (bits 8-15) of the 16-bit CRC_0
5	SafeData[2]	3rd octet of SafeInputs
6	SafeData[3]	4th octet of SafeInputs
7	CRC_1_Lo	low octet (bits 0-7) of the 16-bit CRC_1
8	CRC_1_Hi	high octet (bits 8-15) of the 16-bit CRC_1
9	Conn_Id_Lo	Connection Id, low octet
10	Conn_Id_Hi	Connection Id, high octet

7.2.2.6.2 FailSafeData command

If the FSoE Master locally detects that the SafeOutputs are not valid or are to be switched to safe state, it sends the FailSafeData command.

Table 25 shows an example of the Safety Master PDU for 4 octets of FailsafeData with the FailsafeData command.

Table 25 – Safety Master PDU for 4 octets of fail-safe data in data state

Octet	Name	Description
0	Command	*FailSafeData*
1	SafeData[0]	Fail-safe Data = 0
2	SafeData[1]	Fail-safe Data = 0
3	CRC_0_Lo	low octet (bits 0-7) of the 16-bit CRC_0
4	CRC_0_Hi	high octet (bits 8-15) of the 16-bit CRC_0
5	SafeData[2]	Fail-safe Data = 0
6	SafeData[3]	Fail-safe Data = 0
7	CRC_1_Lo	low octet (bits 0-7) of the 16-bit CRC_1
8	CRC_1_Hi	high octet (bits 8-15) of the 16-bit CRC_1
9	Conn_Id_Lo	Connection Id, low octet
10	Conn_Id_Hi	Connection Id, high octet

If the FSoE Slave locally detects that the SafeInputs are not valid or are to be switched to safe state, it sends the FailSafeData command.

Table 26 shows an example of the Safety Slave PDU for 4 octets of FailsafeData with the FailsafeData command.

Table 26 – Safety Slave PDU for 4 octets of fail-safe data in data state

Octet	Name	Description
0	Command	*FailSafeData*
1	SafeData[0]	Fail-safe Data = 0
2	SafeData[1]	Fail-safe Data = 0
3	CRC_0_Lo	low octet (bits 0-7) of the 16-bit CRC_0
4	CRC_0_Hi	high octet (bits 8-15) of the 16-bit CRC_0
5	SafeData[2]	Fail-safe Data = 0
6	SafeData[3]	Fail-safe Data = 0
7	CRC_1_Lo	low octet (bits 0-7) of the 16-bit CRC_1
8	CRC_1_Hi	high octet (bits 8-15) of the 16-bit CRC_1
9	Conn_Id_Lo	Connection Id, low octet
10	Conn_Id_Hi	Connection Id, high octet

The transfer of ProcessData or FailSafeData is independent of the command of the received Safety PDU. It only depends on local circumstances.

7.3 Reaction on communication errors

An FSoE node can detect the errors listed in Table 27.

Table 27 – FSoE communication error

Error	Description
Unexpected command	The received command is not allowed in the state
Unknown command	The received command is not defined
Invalid connection ID	The connection does not match the connection ID transferred in the connection state
CRC error	At least one of the received CRC_i does not match the calculated CRC_i
Watchdog has expired	No valid Safety PDU was received within the FSoE watchdog time
Invalid FSoE Alave Address	The FSoE Slave Address transferred in the Connection state does not match the local address set on the FSoE Slave
Invalid SafeData	The safety data sent back by the FSoE Slave in the Session, Connection and Parameter states do not match the expected values
Faulty SafePara	The SafePara sent to the FSoE Slave in the Parameter state are faulty
Invalid communication parameter length	The length of the communication parameter is wrong
Invalid communication parameter	The content of the communication parameter is wrong
Invalid application parameter length	The length of the application parameter is wrong
Invalid application parameter	The content of the application parameter is wrong

If an FSoE node detects a communication error, a Reset command is sent, as well as the associated error code in SafeData[0] for diagnostic purposes. The FSoE Master then switches to the Session state, the FSoE Slave to the Reset state. The FSoE communication error codes are listed in Table 28.

Table 28 – FSoE communication error codes

Error Code	Description
0	Local reset or acknowledgement of a RESET command
1	Unexpected command (INVALID_CMD)
2	Unknown command (UNKNOWN_CMD)
3	Invalid connection ID (INVALID_CONNID)
4	CRC error (INVALID_CRC)
5	Watchdog has expired (WD_EXPIRED)
6	Invalid FSoE Slave Address (INVALID_ADDRESS)
7	Invalid safety data (INVALID_DATA)
8	Invalid communication parameter length (INVALID_COMMPARALEN)
9	Invalid communication parameter data (INVALID_COMPARA)
10	Invalid application parameter length (INVALID_USERPARALEN)
11	Invalid application parameter data (INVALID_USERPARA)
0x80-0xFF	Invalid SafePara (device-specific)

7.4 State table for FSoE Master

7.4.1 FSoE Master state machine

7.4.1.1 Overview

Depending on the communication procedure, the FSoE Master can have the states listed in Table 29.

Table 29 – States of the FSoE Master

State	Description
Reset	The FSoE Connection is reset (outputs are in safe state)
Session	The session ID is being transferred (outputs are in safe state)
Connection	The connection ID is being transferred (outputs are in safe state)
Parameter	The parameters are being transferred (outputs are in safe state)
Data	Process data or fail-safe data are being transferred (outputs are only active if the *ProcessData* command is received)

The state diagram for the FSoE Master is shown in Figure 9.

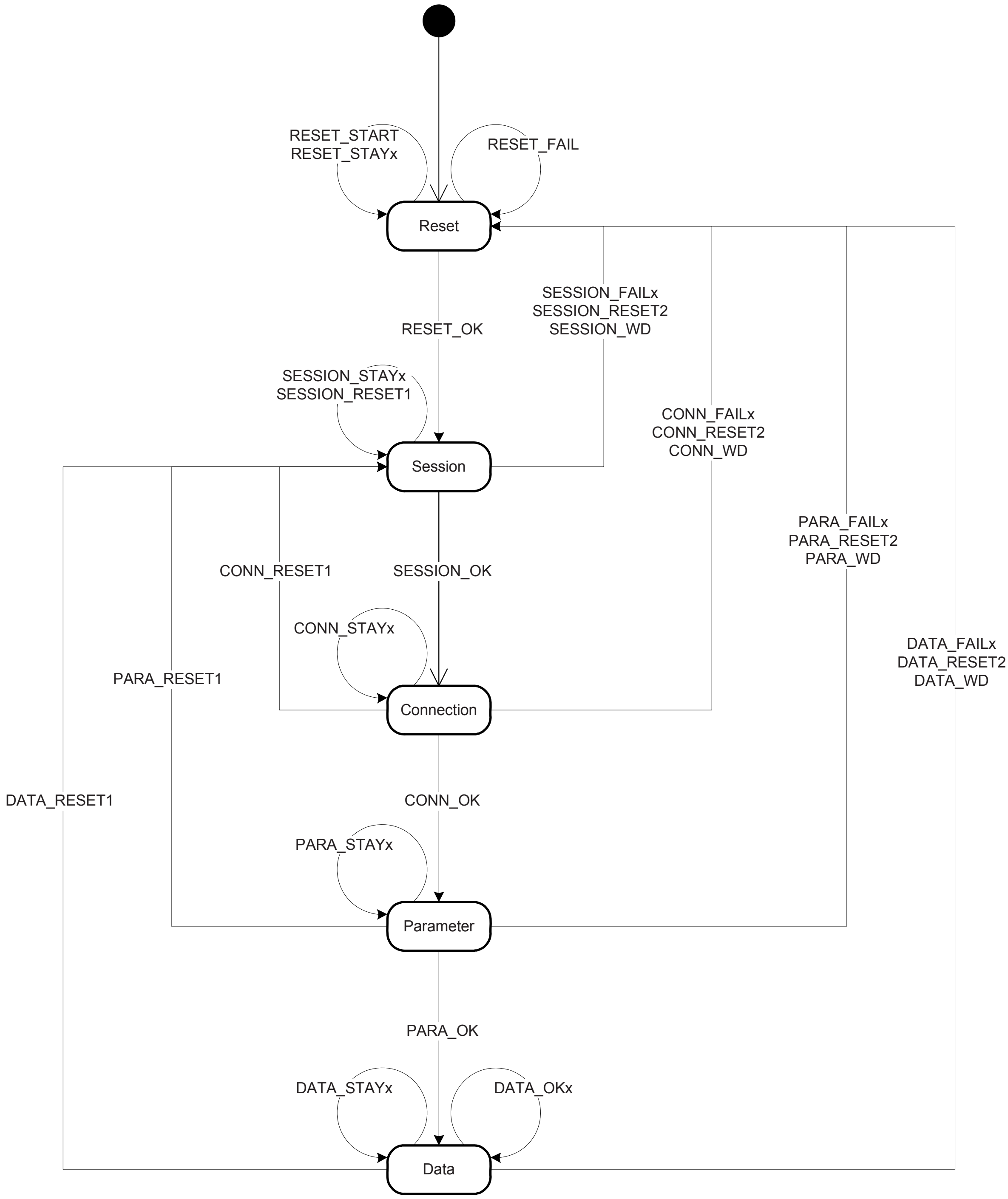


Figure 9 – State diagram for FSoE Master

For each state, the following sections analyse the events that can occur in the FSoE Master. Each event is considered under conditions with different actions or subsequent states.

7.4.1.2 Events

An event can include different parameters, which are referred to in the state tables. Table 30 lists the used events.

Table 30 – Events in the FSoE Master state table

Event	Description
Frame received	A Safety PDU was received, i.e. at least one bit within the Safety PDU has changed Parameters: Frame = received Safety PDU Frame.Command = command of the received Safety PDU Frame.Crc0 = CRC_0 of the received Safety PDU Frame.ConnId = connection ID of the received Safety PDU Frame.SafeData = safety data of the received Safety PDU
Watchdog expired	The FSoE watchdog has expired, i.e. no Safety PDU was received within the watchdog time Parameters: none
Reset Connection	Request via a local interface to reset the FSoE Connection. This event shall be triggered on power-on in order to start communication with the FSoE Slave Parameters: none
Set Data Command	Request via a local interface to switch the SafeOutputs to the safe state or to exit the safe state Parameters: DataCmd = *FailSafeData* or *ProcessData*

7.4.1.3 Actions

Depending on different conditions, certain actions are carried out if an event occurs. In the state tables the actions are shown as function calls or variable assignments.

Table 31 lists the used functions in the FSoE Master state table.

Table 31 – Functions in the FSoE Master state table

Function	Description
SendFrame(cmd, safeData, lastCrc, connId, seqNo, oldCrc, bNew)	An FSoE Master frame is sent Parameters: cmd = frame command SafeData = reference to safety data sent with the frame lastCrc = CRC_0 of the last Safety Slave PDU included in the CRC calculation for the frame connId = Connect ID to be entered in the frame and included in the CRC calculation seqNo = Pointer to the Master Sequence Number included in the CRC calculation for the frames. The incremented (perhaps several times) seqNo is returned oldCrc: Pointer to the CRC_0 of the last sent Safety Master PDU. The calculated CRC_0 is returned bNew: if bNew = TRUE and oldCrc equals the calculated crc, the CRC calculation is repeated with the incremented seqNo until the calculated crc is not equal the oldCrc (procedure according to 7.1.3.4)

Table 32 lists the used variables in the FSoE Master state table.

Table 32 – Variables in the FSoE Master state table

Variable	Description
LastCrc	CRC_0 of the last sent Safety Master PDU (initialised with 0 on power-on)
OldMasterCrc	CRC_0 of the last sent Safety Master PDU (initialised with 0 on power-on)
OldSlaveCrc	CRC_0 of the last received Safety Slave PDU (initialised with 0 on power-on)
MasterSeqNo	Master Sequence Number to be used in the CRC for next Safety Master PDU (initialised with 0 on power-on)
SlaveSeqNo	Expected Slave Sequence Number to be used in the CRC for next Safety Slave PDU (initialised with 0 on power-on)
SessionId	Randomly generated session ID (initialised with 0 on power-on)
DataCommand	Indicates whether the *ProcessData* or *FailSafeData* command is sent in the Data state. Initialised with *FailSafeData* on power-on
BytesToBeSent	If several Safety PDUs have to be sent in the Session, Connection or Parameter state, this variable indicates how many octets are still to be sent (initialised with 0 on power-on)
ConnData	ConnData consist of the connection ID and the FSoE Slave Address. Initialised by the safety configurator on power-on according to the configuration ConnData.ConnId: ConnectionId of the FSoE Connection
SafePara	The SafePara consist of the safety communication parameters and the safety application parameters. Initialised by the safety configurator on power-on according to the configuration SafePara.Watchdog: FSoE watchdog
SafeParaSize	Indicates the SafePara length. Initialised by the safety configurator on power-on according to the configuration data
SafeOutputs	Contains the process values of the safety outputs sent to the FSoE Slave. Initialised with FS_VALUE (Fail-safe Data = 0) on power-on
SafeInputs	Contains the process values of the safety inputs received by the FSoE Slave. Initialised with FS_VALUE (Fail-safe Data = 0) on power-on
CommFaultReason	Indicates the error code in the event of a communication error
SecondSessionFrameSent	If two Safety PDUs have to be sent in state Session this variable indicates if the second PDU is already sent. This variable is set to FALSE by the macro CREATE_SESSION_ID

7.4.1.4 Macros

Certain functionalities are consolidated in macros in order to keep the state tables transparent.

Table 33 lists the used macros in the FSoE Master state table.

Table 33 – Macros in the FSoE Master state table

Macro	Description
IS_CRC_CORRECT(frame, lastCrc, seqNo, oldCrc, bNew)	This macro checks whether the CRCs of the received Safety Slave PDU are correct Parameters: Frame = received frame lastCrc = CRC_0 of the last sent Safety Master PDU included in the CRC calculations for the received PDU seqNo = Slave Sequence Number included in the CRC calculations for the received frame. The incremented (perhaps several times) seqNo is returned oldCrc: Pointer to the CRC_0 of the last received Safety Slave PDU. The CRC_0 of the received telegram is returned

Macro	Description
	bNew: if bNew = TRUE and oldCrc equals the calculated crc, the CRC calculation is repeated with the incremented seqNo until the calculated crc is not equal the oldCrc (procedure according to 7.1.3.4)
UPDATE_BYTES_TO_BE_SENT (bytesSent)	This macro checks how many octets in the Session, Connection and Parameter states are still to be sent before the state is changed Parameters: bytesSent = Number of octets still to be sent
IS_SAFEDATA_CORRECT (frame, expectedData, bytesSent)	This macro checks whether the SafeData of the received Safety Slave PDUmatch the expected data Parameters: Frame = received frame expectedData = Reference to the expected data bytesSent = Number of octets sent
START_WD(watchdog)	This macro resets the watchdog and starts a monitoring timer Parameters: Watchdog = monitoring time in ms
CREATE_SESSION_ID	This macro generates a random session ID The variable SecondSessionFrameSent is reset to FALSE
ADR	This macro generates a reference (pointer) to a variable

7.4.2 Reset state

7.4.2.1 Frame received event

Transition	Condition	Action	Next State

RESET_OK	`Frame.Command = Reset`	`SessionId := CREATE_SESSION_ID();`
		`SendFrame(Session,`
		`ADR(SessionId),`
		`LastCrc,`
		`0,`
		`ADR(MasterSeqNo),`
		`ADR(OldMasterCrc),`
		`FALSE);`
		`LastCrc = SendFrame.Crc0;`
		`BytesToBeSent :=`
		`UPDATE_BYTES_TO_BE_SENT(2);`
		`START_WD(SafePara.Watchdog);`
			Session

RESET_STAY1	`Frame.Command <> Reset`	`LastCrc := 0`
		`OldMasterCrc := 0;`
		`OldSlaveCrc := 0;`
		`MasterSeqNo := 1;`
		`SlaveSeqNo := 1;`
		`DataCommand := FailSafeData;`
		`CommFaultReason := 0;`
		`SendFrame(Reset,`
		`ADR(CommFaultReason),`
		`LastCrc,`
		`0,`
		`ADR(MasterSeqNo),`
		`ADR(OldMasterCrc),`
		`FALSE);`
		`MasterSeqNo := 1;`
			Reset

7.4.2.2 Watchdog expired event

Transition	Condition	Action	Next State
RESET_WD	Watchdog expired	SessionId := CREATE_SESSION_ID(); SendFrame(Session, ADR(SessionId), LastCrc, 0, ADR(MasterSeqNo), ADR(OldMasterCrc), FALSE); LastCrc = SendFrame.Crc0; BytesToBeSent := UPDATE_BYTES_TO_BE_SENT(2); START_WD(SafePara.Watchdog);	Session

7.4.2.3 Reset connection event

Transition	Condition	Action	Next State
RESET_START		LastCrc := 0 OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := FailSafeData; CommFaultReason := 0; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(MasterSeqNo), ADR(OldMasterCrc), FALSE); MasterSeqNo := 1; START_WD(SafePara.Watchdog);	Reset

7.4.2.4 Set Data Command event

Transition	Condition	Action	Next State
RESET_STAY2		DataCommand := DataCmd;	Reset

7.4.3 Session state

7.4.3.1 Frame received event

Transition	Condition	Action	Next State
SESSION_OK	Frame.Command = *Session* AND BytesToBeSent = 0 AND IS_CRC_CORRECT(Frame, LastCrc, ADR(SlaveSeqNo), ADR(OldSlaveCrc), TRUE) = TRUE	LastCrc := Frame.Crc0; SendFrame(Connection, ADR(ConnData), LastCrc, ConnData.ConnId, ADR(MasterSeqNo), ADR(OldMasterCrc), TRUE); LastCrc := SendFrame.Crc0; BytesToBeSent := UPDATE_BYTES_TO_BE_SENT(4); START_WD(SafePara.Watchdog);	Connection

Transition	Condition	Action	Next State
SESSION_FAIL1	Frame.Command = *Session* AND IS_CRC_CORRECT(Frame, LastCrc, ADR(SlaveSeqNo), ADR(OldSlaveCrc), TRUE) = FALSE AND SecondSessionFrameSent = TRUE	LastCrc := 0 OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_CRC; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(MasterSeqNo), ADR(OldMasterCrc), FALSE); MasterSeqNo := 1; START_WD(SafePara.Watchdog);	Reset
SESSION_STAY2	Frame.Command = *Session* AND IS_CRC_CORRECT(Frame, LastCrc, ADR(SlaveSeqNo), ADR(OldSlaveCrc), TRUE) = FALSE AND SecondSessionFrameSent = FALSE	START_WD(SafePara.Watchdog);	Session
SESSION_STAY1	Frame.Command = *Session* AND BytesToBeSent <> 0 AND IS_CRC_CORRECT(Frame, LastCrc, ADR(SlaveSeqNo), ADR(OldSlaveCrc), TRUE) = TRUE	LastCrc := Frame.Crc0; SendFrame(Session, ADR(SessionId [2-BytesToBeSent]), Frame.Crc0, 0, ADR(MasterSeqNo), ADR(OldMasterCrc), TRUE); LastCrc := SendFrame.Crc0; BytesToBeSent := UPDATE_BYTES_TO_BE_SENT(BytesToBeSent); SecondSessionFrameSent := TRUE; START_WD(SafePara.Watchdog);	Session
SESSION_RESET1	Frame.Command = *Reset*	LastCrc := 0 OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; SessionId := CREATE_SESSION_ID(); DataCommand := *FailSafeData*; SendFrame(Session, ADR(SessionId), LastCrc, 0, ADR(MasterSeqNo), ADR(OldMasterCrc), FALSE); LastCrc = SendFrame.Crc0; BytesToBeSent := UPDATE_BYTES_TO_BE_SENT(2); START_WD(SafePara.Watchdog);	Session

Transition	Condition	Action	Next State
SESSION_FAIL3	Frame.Command = *Connection* OR Frame.Command = *Parameter* OR Frame.Command = *ProcessData* OR Frame.Command = *FailSafeData*	LastCrc := 0 OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_CMD; SendFrame(Reset, ADR(CommfaultReason), LastCrc, 0, ADR(MasterSeqNo), ADR(OldMasterCrc), FALSE); MasterSeqNo := 1; START_WD(SafePara.Watchdog);	Reset
SESSION_FAIL4	Frame.Command <> *Reset* AND Frame.Command <> *Session* AND Frame.Command <> *Connection* AND Frame.Command <> *Parameter* AND Frame.Command <> *ProcessData* AND Frame.Command <> *FailSafeData*	LastCrc := 0 OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := UNKNOWN_CMD; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(MasterSeqNo), ADR(OldMasterCrc), FALSE) MasterSeqNo := 1; START_WD(SafePara.Watchdog);	Reset

7.4.3.2 Watchdog expired event

Transition	Condition	Action	Next State
SESSION_WD		LastCrc := 0 OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := FailSafeData; CommFaultReason := WD_EXPIRED; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(MasterSeqNo), ADR(OldMasterCrc), FALSE); MasterSeqNo := 1; START_WD(SafePara.Watchdog);	Reset

7.4.3.3 Reset connection event

Transition	Condition	Action	Next State
SESSION_RESET2		LastCrc := 0 OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := FailSafeData; CommFaultReason := 0; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(MasterSeqNo), ADR(OldMasterCrc), FALSE); MasterSeqNo := 1; START_WD(SafePara.Watchdog);	Reset

7.4.3.4 Set Data Command event

Transition	Condition	Action	Next State
SESSION_STAY2		DataCommand := DataCmd;	Session

7.4.4 Connection state

7.4.4.1 Frame received event

Transition	Condition	Action	Next State
CONN_OK	Frame.Command = *Connection* AND BytesToBeSent = 0 AND Frame.ConnId = ConnData.ConnId AND IS_SAFEDATA_CORRECT(Frame, ADR(ConnData), 4-BytesToBeSent) = TRUE AND IS_CRC_CORRECT(Frame, LastCrc, ADR(SlaveSeqNo), ADR(OldSlaveCrc), TRUE) = TRUE	LastCrc := Frame.Crc0; SendFrame(Parameter, ADR(SafePara), Frame.Crc0, ConnData.ConnId, ADR(MasterSeqNo), ADR(OldMasterCrc), TRUE); LastCrc := SendFrame.Crc0; BytesToBeSent := UPDATE_BYTES_TO_BE_SENT(SafeParaSize); START_WD(SafePara.Watchdog);	Parameter
CONN_FAIL1	Frame.Command = *Connection* AND Frame.ConnId = ConnData.ConnId AND IS_SAFEDATA_CORRECT(Frame, ADR(ConnData), 4-BytesToBeSent) = TRUE AND IS_CRC_CORRECT(Frame, LastCrc, ADR(SlaveSeqNo), ADR(OldSlaveCrc), TRUE) = FALSE	LastCrc := 0 OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_CRC; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(MasterSeqNo), ADR(OldMasterCrc), FALSE); MasterSeqNo := 1; START_WD(SafePara.Watchdog);	Reset

Transition	Condition	Action	Next State
CONN_FAIL2	Frame.Command = *Connection* AND Frame.ConnId = ConnData.ConnId AND IS_SAFEDATA_CORRECT(Frame, ADR(ConnData), 4-BytesToBeSent) = FALSE	LastCrc := 0 OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_DATA; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(MasterSeqNo), ADR(OldMasterCrc), FALSE); MasterSeqNo := 1; START_WD(SafePara.Watchdog);	Reset
CONN_FAIL3	Frame.Command = *Connection* AND Frame.ConnId <> ConnData.ConnId	LastCrc := 0 OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_CONNID; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(MasterSeqNo), ADR(OldMasterCrc), FALSE); MasterSeqNo := 1; START_WD(SafePara.Watchdog);	Reset
CONN_STAY1	Frame.Command = *Connection* AND BytesToBeSent <> 0 AND Frame.ConnId = ConnData.ConnId AND IS_SAFEDATA_CORRECT(Frame, ADR(ConnData), 4-BytesToBeSent) = TRUE AND IS_CRC_CORRECT(Frame, LastCrc, ADR(SlaveSeqNo), ADR(OldSlaveCrc), TRUE) = TRUE	LastCrc := Frame.Crc0; SendFrame(Connection, ADR(ConnData[4- BytesToBeSent]), Frame.Crc0, ConnData.ConnId, ADR(MasterSeqNo), ADR(OldMasterCrc), TRUE); LastCrc := SendFrame.Crc0; BytesToBeSent := UPDATE_BYTES_TO_BE_SENT(BytesToBeSent); START_WD(SafePara.Watchdog);	Connection

Transition	Condition	Action	Next State
CONN_RESET1	Frame.Command = *Reset*	LastCrc := 0 OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; SessionId := CREATE_SESSION_ID(); SendFrame(Session, ADR(SessionId), LastCRC, 0, ADR(MasterSeqNo), ADR(OldMasterCrc), FALSE); LastCrc = SendFrame.Crc0 BytesToBeSent := UPDATE_BYTES_TO_BE_SENT(2); START_WD(SafePara.Watchdog);	Session
CONN_FAIL4	Frame.Command = *Session* OR Frame.Command = *Parameter* OR Frame.Command = *ProcessData* OR Frame.Command = *FailSafeData*	LastCrc := 0 OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_CMD; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(MasterSeqNo), ADR(OldMasterCrc), FALSE); MasterSeqNo := 1; START_WD(SafePara.Watchdog);	Reset
CONN_FAIL5	Frame.Command <> *Reset* AND Frame.Command <> *Session* AND Frame.Command <> *Connection* AND Frame.Command <> *Parameter* AND Frame.Command <> *ProcessData* AND Frame.Command <> *FailSafeData*	LastCrc := 0 OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := UNKNOWN_CMD; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(MasterSeqNo), ADR(OldMasterCrc), FALSE); MasterSeqNo := 1; START_WD(SafePara.Watchdog);	Reset

7.4.4.2 Watchdog expired event

Transition	Condition	Action	Next State

CONN_WD		`LastCrc := 0`
		`OldMasterCrc := 0;`
		`OldSlaveCrc := 0;`
		`MasterSeqNo := 1;`
		`SlaveSeqNo := 1;`
		`DataCommand := FailSafeData;`
		`CommFaultReason := WD_EXPIRED;`
		`SendFrame(Reset,`
		`ADR(CommFaultReason),`
		`LastCrc,`
		`0,`
		`ADR(MasterSeqNo),`
		`ADR(OldMasterCrc),`
		`FALSE);`
		`MasterSeqNo := 1;`
		`START_WD(SafePara.Watchdog);`
			Reset

7.4.4.3 Reset connection event

Transition	Condition	Action	Next State

CONN_RESET2		`LastCrc := 0`
		`OldMasterCrc := 0;`
		`OldSlaveCrc := 0;`
		`MasterSeqNo := 1;`
		`SlaveSeqNo := 1;`
		`DataCommand := FailSafeData;`
		`CommFaultReason := 0;`
		`SendFrame(Reset,`
		`ADR(CommFaultReason),`
		`LastCrc,`
		`0,`
		`ADR(MasterSeqNo),`
		`ADR(OldMasterCrc),`
		`FALSE);`
		`MasterSeqNo := 1;`
		`START_WD(SafePara.Watchdog);`
			Reset

7.4.4.4 Set Data Command event

Transition	Condition	Action	Next State
CONN_STAY2		`DataCommand := DataCmd;`	Connection

7.4.5 Parameter state

7.4.5.1 Frame received event

Transition	Condition	Action	Next State
PARA_OK	Frame.Command = *Parameter* AND BytesToBeSent = 0 AND Frame.ConnId = ConnData.ConnId AND IS_SAFEDATA_CORRECT(Frame, ADR(SafePara), SafeParaSize- BytesToBeSent) = TRUE AND IS_CRC_CORRECT(Frame, LastCrc, ADR(SlaveSeqNo), ADR(OldSlaveCrc), TRUE) = TRUE	LastCrc := Frame.Crc0; SendFrame(DataCommand, ADR(SafeOutputs), Frame.Crc0, ConnData.ConnId, ADR(MasterSeqNo), ADR(OldMasterCrc), TRUE); LastCrc := SendFrame.Crc0; START_WD(SafePara.Watchdog);	Data
PARA_FAIL1	Frame.Command = *Parameter* AND Frame.ConnId = ConnData.ConnId AND IS_SAFEDATA_CORRECT(Frame, ADR(SafePara), SafeParaSize- BytesToBeSent) = TRUE AND IS_CRC_CORRECT(Frame, LastCrc, ADR(SlaveSeqNo), ADR(OldSlaveCrc), TRUE) = FALSE	LastCrc := 0 OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_CRC; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(MasterSeqNo), ADR(OldMasterCrc), FALSE); MasterSeqNo := 1; START_WD(SafePara.Watchdog);	Reset
PARA_FAIL2	Frame.Command = *Parameter* AND Frame.ConnId = ConnData.ConnId AND IS_SAFEDATA_CORRECT(Frame, ADR(SafePara), SafeParaSize- BytesToBeSent) = FALSE	LastCrc := 0 OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_DATA; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(MasterSeqNo), ADR(OldMasterCrc), FALSE); MasterSeqNo := 1; START_WD(SafePara.Watchdog);	Reset

Transition	Condition	Action	Next State
PARA_FAIL3	Frame.Command = *Parameter* AND Frame.ConnId <> ConnData.ConnId	LastCrc := 0 OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_CONNID; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(MasterSeqNo), ADR(OldMasterCrc), FALSE); MasterSeqNo := 1; START_WD(SafePara.Watchdog);	Reset
PARA_STAY1	Frame.Command = *Parameter* AND BytesToBeSent <> 0 AND Frame.ConnId = ConnData.ConnId AND IS_SAFEDATA_CORRECT(Frame, ADR(SafePara), SafeParaSize- BytesToBeSent) = TRUE AND IS_CRC_CORRECT(Frame, LastCrc, ADR(SlaveSeqNo), ADR(OldSlaveCrc), TRUE) = TRUE	LastCrc := Frame.Crc0; SendFrame(Parameter, ADR(SafePara[SafeParaSize- BytesToBeSent]), Frame.Crc0, ConnData.ConnId, ADR(MasterSeqNo), ADR(OldMasterCrc), TRUE); LastCrc := SendFrame.Crc0; BytesToBeSent := UPDATE_BYTES_TO_BE_SENT(BytesToBeSent); START_WD(SafePara.Watchdog);	Parameter
PARA_RESET1	Frame.Command = *Reset*	LastCrc := 0 OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; SessionId := CREATE_SESSION_ID(); SendFrame(Session, ADR(SessionId), LastCRC, 0, ADR(MasterSeqNo), ADR(OldMasterCrc), FALSE); LastCrc = SendFrame.Crc0 BytesToBeSent := UPDATE_BYTES_TO_BE_SENT(2); START_WD(SafePara.Watchdog);	Session

Transition	Condition	Action	Next State
PARA_FAIL4	Frame.Command = *Session* OR Frame.Command = *Connection* OR Frame.Command = *ProcessData* OR Frame.Command = *FailSafeData*	LastCrc := 0 OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_CMD; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(MasterSeqNo), ADR(OldMasterCrc), FALSE); MasterSeqNo := 1; START_WD(SafePara.Watchdog);	Reset
PARA_FAIL5	Frame.Command <> *Reset* AND Frame.Command <> *Session* AND Frame.Command <> *Connection* AND Frame.Command <> *Parameter* AND Frame.Command <> *ProcessData* AND Frame.Command <> *FailSafeData*	LastCrc := 0 OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := UNKNOWN_CMD; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(MasterSeqNo), ADR(OldMasterCrc), FALSE); MasterSeqNo := 1; START_WD(SafePara.Watchdog);	Reset

7.4.5.2 Watchdog expired event

PARA_WD		LastCrc := 0 OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := FailSafeData; CommFaultReason := WD_EXPIRED; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(MasterSeqNo), ADR(OldMasterCrc), FALSE); MasterSeqNo := 1; START_WD(SafePara.Watchdog);	Reset

7.4.5.3 Reset connection event

Transition	Condition	Action	Next State

PARA_RESET2		`LastCrc := 0`
		`OldMasterCrc := 0;`
		`OldSlaveCrc := 0;`
		`MasterSeqNo := 1;`
		`SlaveSeqNo := 1;`
		`DataCommand := FailSafeData;`
		`CommFaultReason := 0;`
		`SendFrame(Reset,`
		`ADR(CommFaultReason),`
		`LastCrc,`
		`0,`
		`ADR(MasterSeqNo),`
		`ADR(OldMasterCrc),`
		`FALSE);`
		`MasterSeqNo := 1;`
		`START_WD(SafePara.Watchdog);`
			Reset

7.4.5.4 Set Data Command event

Transition	Condition	Action	Next State
PARA_STAY2		`DataCommand := DataCmd;`	Parameter

7.4.6 Data state

7.4.6.1 Frame received event

Transition	Condition	Action	Next State

DATA_OK1	`Frame.Command =`
	ProcessData
	`AND`
	`Frame.ConnId =`
	`ConnData.ConnId`
	`AND`
	`IS_CRC_CORRECT(Frame,`
	`LastCrc,`
	`ADR(SlaveSeqNo),`
	`ADR(OldSlaveCrc),`
	`TRUE) = TRUE`
		`SafeInputs := Frame.SafeData;`
		`LastCrc := Frame.Crc0;`
		`SendFrame(DataCommand,`
		`ADR(SafeOutputs),`
		`Frame.Crc0,`
		`ConnData.ConnId,`
		`ADR(MasterSeqNo),`
		`ADR(OldMasterCrc),`
		`TRUE);`
		`LastCrc := SendFrame.Crc0;`
		`START_WD(SafePara.Watchdog);`
			Data

DATA_OK2	`Frame.Command =`
	FailSafeData
	`AND`
	`Frame.ConnId =`
	`ConnData.ConnId`
	`AND`
	`IS_CRC_CORRECT(Frame,`
	`LastCrc,`
	`ADR(SlaveSeqNo),`
	`ADR(OldSlaveCrc),`
	`TRUE) = TRUE`
		`SafeInputs := FS_VALUE;`
		`LastCrc := Frame.Crc0;`
		`SendFrame(DataCommand,`
		`ADR(SafeOutputs),`
		`Frame.Crc0,`
		`ConnData.ConnId,`
		`ADR(MasterSeqNo),`
		`ADR(OldMasterCrc),`
		`TRUE);`
		`LastCrc := SendFrame.Crc0;`
		`START_WD(SafePara.Watchdog);`
			Data

Transition	Condition	Action	Next State
DATA_FAIL1	(Frame.Command = *ProcessData* OR Frame.Command = *FailSafeData)* AND Frame.ConnId = ConnData.ConnId AND IS_CRC_CORRECT(Frame, LastCrc, ADR(SlaveSeqNo), ADR(OldSlaveCrc), TRUE) = FALSE	LastCrc := 0 OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; SafeInputs := FS_VALUE; CommFaultReason := INVALID_CRC; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(MasterSeqNo), ADR(OldMasterCrc), FALSE); MasterSeqNo := 1; START_WD(SafePara.Watchdog);	Reset
DATA_FAIL2	(Frame.Command = *ProcessData* OR Frame.Command = *FailSafeData)* AND Frame.ConnId <> ConnData.ConnId	LastCrc := 0 OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; SafeInputs := FS_VALUE; CommFaultReason := INVALID_CONNID SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(MasterSeqNo), ADR(OldMasterCrc), FALSE); MasterSeqNo := 1; START_WD(SafePara.Watchdog);	Reset
DATA_RESET1	Frame.Command = *Reset*	LastCrc := 0 OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; SafeInputs := FS_VALUE; SessionId := CREATE_SESSION_ID(); SendFrame(Session, ADR(SessionId), LastCRC, 0, ADR(MasterSeqNo), ADR(OldMasterCrc), FALSE); LastCrc = SendFrame.Crc0 BytesToBeSent := UPDATE_BYTES_TO_BE_SENT(2); START_WD(SafePara.Watchdog);	Session

Transition	Condition	Action	Next State
DATA_FAIL3	Frame.Command = Session OR Frame.Command = Connection OR Frame.Command = Parameter	LastCrc := 0 OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := FailSafeData; CommFaultReason := INVALID_CMD; SafeInputs := FS_VALUE; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(MasterSeqNo), ADR(OldMasterCrc), FALSE); MasterSeqNo := 1; START_WD(SafePara.Watchdog);	Reset
DATA_FAIL4	Frame.Command <> Reset AND Frame.Command <> Session AND Frame.Command <> Connection AND Frame.Command <> Parameter AND Frame.Command <> ProcessData AND Frame.Command <> FailSafeData	LastCrc := 0 OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := FailSafeData; SafeInputs := FS_VALUE; CommFaultReason := UNKNOWN_CMD; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(MasterSeqNo), ADR(OldMasterCrc), FALSE); MasterSeqNo := 1; START_WD(SafePara.Watchdog);	Reset

7.4.6.2 Watchdog expired event

Transition	Condition	Action	Next State
DATA_WD		LastCrc := 0 OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := FailSafeData; SafeInputs := FS_VALUE; CommFaultReason := WD_EXPIRED SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(MasterSeqNo), ADR(OldMasterCrc), FALSE); MasterSeqNo := 1; START_WD(SafePara.Watchdog);	Reset

7.4.6.3 Reset connection event

Transition	Condition	Action	Next State

DATA_RESET2		`LastCrc := 0`
		`OldMasterCrc := 0;`
		`OldSlaveCrc := 0;`
		`MasterSeqNo := 1;`
		`SlaveSeqNo := 1;`
		`DataCommand := FailSafeData;`
		`SafeInputs := FS_VALUE;`
		`CommFaultReason := 0;`
		`SendFrame(Reset,`
		`ADR(CommFaultReason),`
		`LastCrc,`
		`0,`
		`ADR(MasterSeqNo),`
		`ADR(OldMasterCrc),`
		`FALSE);`
		`MasterSeqNo := 1;`
		`START_WD(SafePara.Watchdog);`
			Reset

7.4.6.4 Set Data Command event

Transition	Condition	Action	Next State
DATA_STAY		`DataCommand := DataCmd;`	Data

7.5 State table for FSoE Slave

7.5.1 FSoE Slave state machine

7.5.1.1 Overview

Depending on the communication procedure, the FSoE Slave can have the states listed in Table 34.

Table 34 – States of the FSoE Slave

State	Description
Reset	The FSoE Connection is reset (outputs are in safe state)
Session	The session ID is being transferred (outputs are in safe state)
Connection	The connection ID is being transferred (outputs are in safe state)
Parameter	The parameters are being transferred (outputs are in safe state)
Data	Process data or fail-safe data are being transferred (outputs are only active if the *ProcessData* command is received)

The state diagram for the FSoE State is shown in Figure 10.

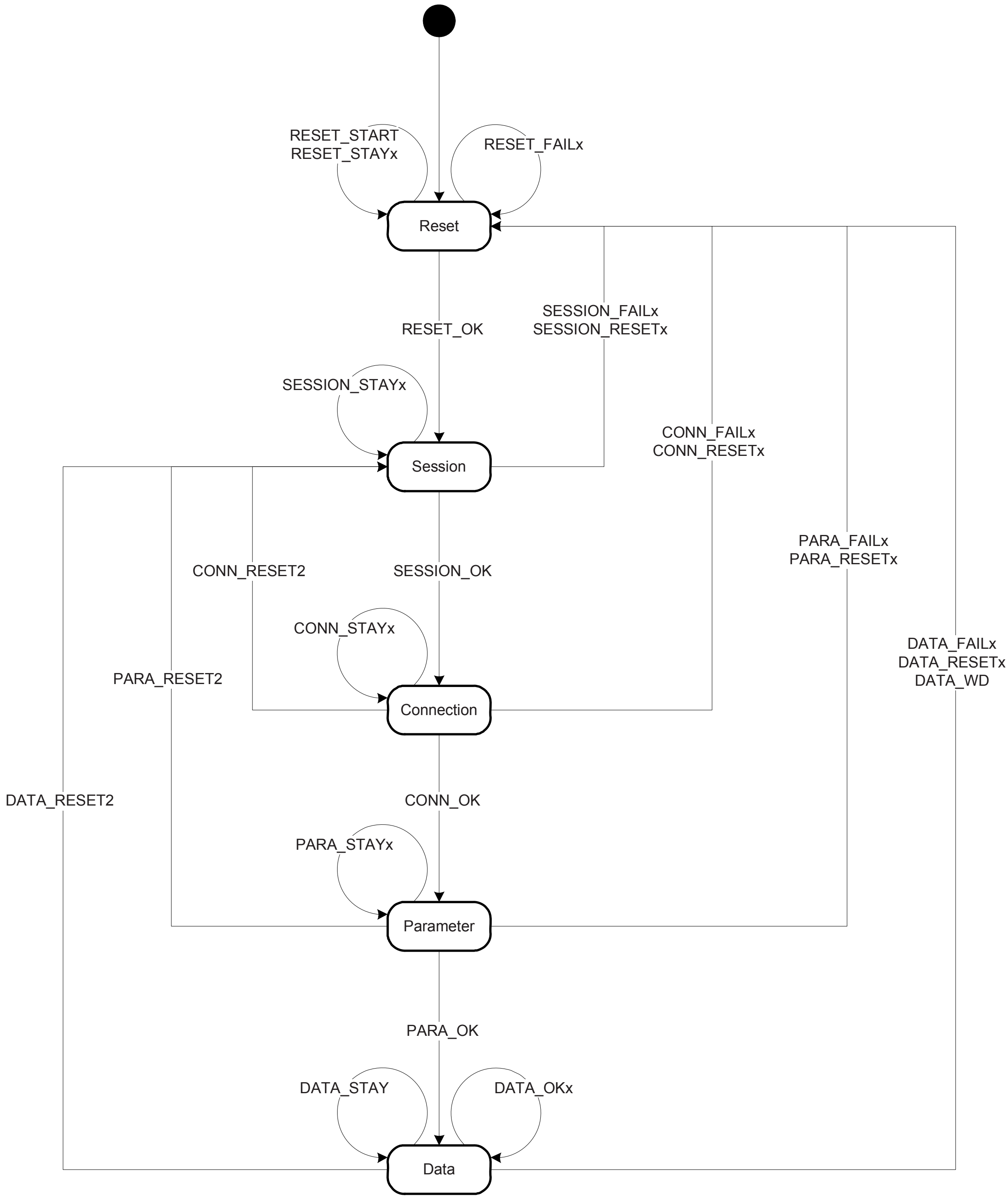


Figure 10 – State diagram for FSoE Slave

For each state, the following sections analyse the events that can occur in the FSoE Slave. Each event is considered under conditions with different actions or subsequent states.

7.5.1.2 Events

An event can include different parameters, which are referred to in the state tables. Table 35 lists the used events.

Table 35 – Events in the FSoE Slave state table

Event	Description
Frame received	A Safety PDU was received, i.e. at least one bit within the PDU has changed Parameters: Frame = received Safety PDU Frame.Command = command of the received Safety PDU Frame.Crc0 = CRC_0 of the received Safety PDU Frame.ConnId = connection ID of the received Safety PDU Frame.SafeData = safety data of the received Safety PDU
Watchdog expired	The FSoE watchdog has expired, i.e. no new Safety PDU was received within the watchdog time Parameters: none
Reset Connection	Request via a local interface to reset the FSoE Connection Parameters: none
Set Data Command	Request via a local interface to switch the SafeInputs to the safe state or to exit the safe state Parameters: DataCmd: FailSafeData or ProcessData

7.5.1.3 Actions

Depending on different conditions certain actions are carried out if an event occurs. In the state tables, the actions are shown as function calls or variable assignments.

Table 36 lists the used functions in the FSoE Slave state table.

Table 36 – Functions in the FSoE Slave state table

Function	Description
SendFrame(cmd, safeData, lastCrc, connId, seqNo, oldCrc, bNew)	A Safety Slave PDU is sent Parameters: Cmd = frame command SafeData = reference to safety data sent with the frame lastCrc = CRC_0 of the last Safety Master PDU included in the CRC calculation for the frame connId = Connect ID to be entered in the frame and included in the CRC calculation seqNo = Pointer to the Slave Sequence Number included in the CRC calculation for the frames. The incremented (perhaps several times) seqNo is returned. oldCrc: Pointer to the CRC_0 of the last sent Safety Slave PDU. The calculated CRC_0 is returned bNew: if bNew = TRUE and oldCrc equals the calculated crc, the CRC calculation is repeated with the incremented seqNo until the calculated crc is not equal the oldCrc (procedure according to 7.1.3.4)

Table 37 lists the used variables in the FSoE Slave state table.

Table 37 – Variables in the FSoE Slave state table

Variable	Description
LastCrc	CRC_0 of the last Safety Slave PDU (initialised with 0 on power-on)
OldMasterCrc	CRC_0 of the last received Safety Master PDU (initialised with 0 on power-on)
OldSlaveCrc	CRC_0 of the last sent Safety Slave PDU (initialised with 0 on power-on)
MasterSeqNo	Expected Master Sequence Number to be used in the CRC for next Safety Master PDU (initialised with 0 on power-on)
SlaveSeqNo	Slave Sequence Number to be used in the CRC for next Safety Slave PDU (initialised with 0 on power-on)
InitSeqNo	Variable containing initialisation sequence number 1
DataCommand	Indicates whether the *ProcessData* or *FailSafeData* command is sent in the Data state. Initialised with *FailSafeData* on power-on
BytesToBeSent	If several Safety PDUs have to be sent in the Session, Connection or Parameter state, this variable indicates how many octets are still to be sent (initialised with 0 on power-on)
ConnectionId	In the Connection state the ConnectionID is received by the FSoE Master (initialised with 0 on power-on)
ConnectionData	In the Connection state the ConnectionData are received by the FSoE Master (initialised with 0 on power-on)
SlaveAddress	The FSoE Slave Address is initialised via a local interface on power-on (generally an external address switch)
SafePara	The SafePara are received by the FSoE Master in the Parameter state and initialised depending on the device SafePara.Watchdog: FSoE watchdog (initialised with 0 on power-on)
ExpectedSafeParaSize	Indicates the expected SafePara length
SafeOutputs	Contains the process values of the safety outputs received by the FSoE Master. Initialised with FS_VALUE (Fail-safe Data = 0) on power-on
SafeInputs	Contains the process values of the safety inputs sent to the FSoE Master. Initialised with FS_VALUE (Fail-safe Data = 0) on power-on
CommFaultReason	Indicates the error code in the event of a communication error

7.5.1.4 Macros

Certain functionalities are consolidated in macros in order to keep the state tables transparent.

Table 38 lists the used macros in the FSoE Slave state table.

Table 38 – Macros in the FSoE Slave state table

Macro	Description
IS_CRC_CORRECT(frame, lastCrc, seqNo, oldCrc, bNew)	This macro checks whether the CRCs of the received Safety Master PDU are correct Parameters: frame: received PDU lastCrc: CRC_0 of the last sent Safety Slave PDU included in the CRC calculations for the received frame seqNo: Pointer to the Master Sequence Number used in the CRC calculations for the received frame. The incremented (perhaps several times) seqNo is returned oldCrc: Pointer to the CRC_0 of the last received Safety Master PDU. The

Macro	Description
	CRC_0 of the received telegram is returned bNew: if bNew = TRUE and oldCrc equals the calculated crc, the CRC calculation is repeated with the incremented seqNo until the calculated crc is not equal the oldCrc (procedure according to 7.1.3.4)
UPDATE_BYTES_TO_BE_SENT(bytesSent)	This macro checks how many octets in the Session, Connection and Parameter states are still to be sent before the state is changed Parameters: bytesSent: number of octets still to be sent
IS_SAFE_PARA_CORRECT(safePara)	This macro checks whether the SafePara received in the Parameter state are correct Parameters: safePara: pointer to received SafePara
STORE_DATA(dst, src)	This macro stores the received safety data of a Safety PDU Parameters: dst: pointer to the target address src: pointer to the source address
GET_PARA_FAULT()	This macro returns an error code if the SafePara were invalid
START_WD(watchdog)	This macro resets the watchdog and starts a monitoring timer with the specified watchdog Parameters: watchdog: monitoring time in ms
STOP_WD()	This macro stops the monitoring timer and resets the watchdog
ADR	This macro generates a reference (pointer) to a variable

7.5.2 Reset state

7.5.2.1 Frame received event

Transition	Condition	Action	Next State
RESET_OK	Frame.Command = *Session* AND IS_CRC_CORRECT(Frame, LastCrc, ADR(MasterSeqNo), ADR(OldMasterCrc), FALSE) = TRUE	LastCrc := Frame.Crc0; SessionId := CREATE_SESSION_ID(); SendFrame(Session, ADR(SessionId), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); LastCrc := SendFrame.Crc0; BytesToBeSent := UPDATE_BYTES_TO_BE_SENT(2);	Session
RESET_FAIL1	Frame.Command = *Session* AND IS_CRC_CORRECT(Frame, LastCrc, ADR(MasterSeqNo), ADR(OldMasterCrc), FALSE) = FALSE	LastCrc := 0 OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_CRC; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset

Transition	Condition	Action	Next State
RESET_STAY1	Frame.Command = *Reset*	LastCrc := 0 OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; InitSeqNo := 1; DataCommand := FailSafeData; CommFaultReason := 0; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset
RESET_FAIL2	(Frame.Command = *Connection* OR Frame.Command = *Parameter* OR Frame.Command = Process*Data* OR Frame.Command = FailSafe*Data*)	LastCrc := 0 OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_CMD; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset
RESET_FAIL3	(Frame.Command <> *Reset* AND Frame.Command <> *Session* AND Frame.Command <> *Connection* AND Frame.Command <> *Parameter* AND Frame.Command <> Process*Data* AND Frame.Command <> FailSafe*Data*)	LastCrc := 0 OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := UNKNOWN_CMD; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset

7.5.2.2 Watchdog expired event

Cannot occur in this state because the watchdog has not yet been started.

7.5.2.3 Reset connection event

Transition	Condition	Action	Next State
RESET_START		LastCrc := 0 OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; InitSeqNo := 1; DataCommand := FailSafeData; CommFaultReason := 0; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset

7.5.2.4 Set Data Command event

Transition	Condition	Action	Next State
RESET_STAY2		DataCommand := DataCmd;	Reset

7.5.3 Session state

7.5.3.1 Frame received event

Transition	Condition	Action	Next State
SESSION_OK	Frame.Command = *Connection* AND BytesToBeSent = 0 AND Frame.ConnId <> 0 AND IS_CRC_CORRECT(Frame, LastCrc, ADR(MasterSeqNo), ADR(OldMasterCrc), TRUE) = TRUE	STORE_DATA(ADR(ConnectionData), ADR(Frame.SafeData)); ConnectionId := Frame.ConnId; LastCrc := Frame.Crc0; SendFrame(*Connection*, ADR(Frame.SafeData), LastCrc, ConnectionId, ADR(SlaveSeqNo), ADR(OldSlaveCrc), TRUE); LastCrc := SendFrame.Crc0; BytesToBeSent := UPDATE_BYTES_TO_BE_SENT(4);	Connection
SESSION_FAIl1	Frame.Command = *Connection* AND BytesToBeSent = 0 AND Frame.ConnId <> 0 AND IS_CRC_CORRECT(Frame, LastCrc, ADR(MasterSeqNo), ADR(OldMasterCrc), TRUE) = FALSE	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_CRC; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset

Transition	Condition	Action	Next State
SESSION_FAIL2	Frame.Command = *Connection* AND BytesToBeSent = 0 AND Frame.ConnId = 0	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_CONNID; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset
SESSION_FAIL3	Frame.Command = *Connection* AND BytesToBeSent <> 0	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_CMD; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset
SESSION_STAY1	Frame.Command = *Session* AND BytesToBeSent <> 0 AND IS_CRC_CORRECT(Frame, LastCrc, ADR(MasterSeqNo), ADR(OldMasterCrc), TRUE) = TRUE	LastCrc := Frame.Crc0; SendFrame(Session, ADR(SessionId[2- BytesToBeSent]), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), TRUE); LastCrc := SendFrame.Crc0; BytesToBeSent := UPDATE_BYTES_TO_BE_SENT(BytesToBeSent);	Session
SESSION_STAY2	Frame.Command = *Session* AND IS_CRC_CORRECT(Frame, 0, ADR(InitSeqNo), ADR(OldMasterCrc), FALSE) = TRUE	LastCrc := Frame.Crc0; MasterSeqNo := InitSeqNo; InitSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; SessionId := CREATE_SESSION_ID(); SendFrame(Session, ADR(SessionID), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); LastCrc := SendFrame.Crc0; BytesToBeSent := UPDATE_BYTES_TO_BE_SENT(2);	Session

Transition	Condition	Action	Next State
SESSION_FAIL4[a]	Frame.Command = *Session* AND IS_CRC_CORRECT(Frame, LastCrc, ADR(MasterSeqNo), ADR(OldMasterCrc), TRUE) = FALSE AND IS_CRC_CORRECT(Frame, 0, ADR(InitSeqNo), ADR(OldMasterCrc), FALSE) = FALSE	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; InitSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_CRC; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset
SESSION_FAIL5[a]	Frame.Command = *Session* AND BytesToBeSent = 0 AND IS_CRC_CORRECT(Frame, LastCrc, ADR(MasterSeqNo), ADR(OldMasterCrc), TRUE) = TRUE AND IS_CRC_CORRECT(Frame, 0, ADR(InitSeqNo), ADR(OldMasterCrc), FALSE) = FALSE	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_CMD; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset
SESSION_RESET1	Frame.Command = *Reset* AND IS_CRC_CORRECT(Frame, 0, ADR(InitSeqNo), ADR(OldMasterCrc), FALSE) = TRUE	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; InitSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := 0; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset
SESSION_FAIL6	Frame.Command = *Reset* AND IS_CRC_CORRECT(Frame, 0, ADR(InitSeqNo), ADR(OldMasterCrc), FALSE) = FALSE	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; InitSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_CRC; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset

Transition	Condition	Action	Next State
SESSION_FAIL7	Frame.Command = *Parameter* OR Frame.Command = *ProcessData* *OR* Frame.Command = *FailSafeData*	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_CMD; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset
SESSION_FAIL8	(Frame.Command <> *Reset* AND Frame.Command <> *Session* AND Frame.Command <> *Connection* AND Frame.Command <> *Parameter* AND Frame.Command <> *ProcessData* AND Frame.Command <> *FailSafeData*)	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := UNKNOWN_CMD; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset

a The two states SESSION_FAIL4 and SESSION_FAIL5 are the only states where two CRC-checks should be calculated. The only difference is a different CommFaultReason, i.e. only diagnosis information, not safety relevant. It is allowed to reduce these states to one state; in that state only the condition "IS_CRC_CORRECT(Frame, 0, ADR(InitSeqNo), ADR(OldMasterCrc), FALSE) = FALSE" shall be checked with the CommFaultReason := INVALID_CRC.

7.5.3.2 Watchdog expired event

Cannot occur in this state because the watchdog has not yet been started.

7.5.3.3 Reset connection event

Transition	Condition	Action	Next State
SESSION_RESET2		LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := FailSafeData; CommFaultReason := 0; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset

7.5.3.4 Set Data Command event

Transition	Condition	Action	Next State
SESSION_STAY3		DataCommand := DataCmd;	Session

7.5.4 Connection state

7.5.4.1 Frame received event

Transition	Condition	Action	Next State
CONN_OK	Frame.Command = *Parameter* AND BytesToBeSent = 0 AND Frame.ConnId = ConnectionId AND ConnectionData.ConnectionId = ConnectionId AND ConnectionData.SlaveAddress = SlaveAddress AND IS_CRC_CORRECT(Frame, LastCrc, ADR(MasterSeqNo), ADR(OldMasterCrc), TRUE) = TRUE	STORE_DATA(ADR(SafePara), ADR(Frame.SafeData)); LastCrc := Frame.Crc0; SendFrame(*Parameter*, ADR(Frame.SafeData), LastCrc, ConnectionId, ADR(SlaveSeqNo), ADR(OldSlaveCrc), TRUE); LastCrc := SendFrame.Crc0; BytesToBeSent := UPDATE_BYTES_TO_BE_SENT(ExpectedSafeParaSize);	Parameter
CONN_FAIL1	Frame.Command = *Parameter* AND BytesToBeSent = 0 AND Frame.ConnId = ConnectionId AND ConnectionData.ConnectionId = ConnectionId AND ConnectionData.SlaveAddress = SlaveAddress AND IS_CRC_CORRECT(Frame, LastCrc, ADR(MasterSeqNo), ADR(OldMasterCrc), TRUE) = FALSE	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_CRC; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset
CONN_FAIL2	Frame.Command = *Parameter* AND BytesToBeSent = 0 AND Frame.ConnId = ConnectionId AND ConnectionData.ConnectionId = ConnectionId AND ConnectionData.SlaveAddress <> SlaveAddress	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_ADDR; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset

Transition	Condition	Action	Next State
CONN_FAIL3	Frame.Command = *Parameter* AND BytesToBeSent = 0 AND (Frame.ConnId <> ConnectionId OR ConnectionData.ConnectionId <> ConnectionId)	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_CONNID; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset
CONN_FAIL4	Frame.Command = *Parameter* AND BytesToBeSent <> 0	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_CMD; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset
CONN_STAY1	Frame.Command = *Connection* AND BytesToBeSent <> 0 AND Frame.ConnId = ConnectionId AND IS_CRC_CORRECT(Frame, LastCrc, ADR(MasterSeqNo), ADR(OldMasterCrc), TRUE) = TRUE	STORE_DATA(ADR(Connection[4-BytesToBeSent]), ADR(Frame.SafeData)); LastCrc := Frame.Crc0; SendFrame(Connection, ADR(Frame.SafeData), LastCrc, ConnectionId, ADR(SlaveSeqNo), ADR(OldSlaveCrc), TRUE); LastCrc := SendFrame.Crc0; BytesToBeSent := UPDATE_BYTES_TO_BE_SENT(BytesToBeSent);	Connection
CONN_FAIL5	Frame.Command = *Connection* AND BytesToBeSent <> 0 AND Frame.ConnId = ConnectionId AND IS_CRC_CORRECT(Frame, LastCrc, ADR(MasterSeqNo), ADR(OldMasterCrc), TRUE) = FALSE	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_CRC; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset

Transition	Condition	Action	Next State
CONN_FAIL6	Frame.Command = *Connection* AND BytesToBeSent <> 0 AND Frame.ConnId <> ConnectionId	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_CONID; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset
CONN_FAIL7	Frame.Command = *Connection* AND BytesToBeSent = 0	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_CMD; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset
CONN_RESET1	Frame.Command = *Reset* AND IS_CRC_CORRECT(Frame, 0, ADR(InitSeqNo), ADR(OldMasterCrc), FALSE) = TRUE	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; InitSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := 0; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset

Transition	Condition	Action	Next State

CONN_FAIL8	`Frame.Command = `*`Reset`*
	`AND`
	`IS_CRC_CORRECT(Frame, 0,`
	`ADR(InitSeqNo),`
	`ADR(OldMasterCrc),`
	`FALSE) = FALSE`
		`LastCrc := 0;`
		`OldMasterCrc := 0;`
		`OldSlaveCrc := 0;`
		`MasterSeqNo := 1;`
		`SlaveSeqNo := 1;`
		`InitSeqNo := 1;`
		`DataCommand := `*`FailSafeData`*`;`
		`CommFaultReason :=`
		`INVALID_CRC;`
		`SendFrame(Reset,`
		
		`ADR(CommFaultReason),`
		`LastCrc,`
		`0,`
		`ADR(SlaveSeqNo),`
		`ADR(OldSlaveCrc),`
		`FALSE);`
		`SlaveSeqNo := 1;`
			Reset

CONN_RESET2	`Frame.Command = `*`Session`*
	`AND`
	`IS_CRC_CORRECT(Frame, 0,`
	`ADR(InitSeqNo),`
	`ADR(OldMasterCrc),`
	`FALSE) = TRUE`
		`LastCrc := Frame.Crc0;`
		`MasterSeqNo := 2;`
		`InitSeqNo := 1;`
		`SlaveSeqNo := 1;`
		`DataCommand := `*`FailSafeData`*`;`
		`SessionId :=`
		`CREATE_SESSION_ID();`
		`SendFrame(Session,`
		`ADR(SessionID),`
		`LastCrc,`
		`0,`
		`ADR(SlaveSeqNo),`
		`ADR(OldSlaveCrc),`
		`FALSE);`
		`LastCrc := SendFrame.Crc0;`
		`BytesToBeSent :=`
		`UPDATE_BYTES_TO_BE_SENT(2);`
			Session

CONN_FAIL9	`Frame.Command = `*`Session`*
	`AND`
	`IS_CRC_CORRECT(Frame, 0,`
	`ADR(InitSeqNo),`
	`ADR(OldMasterCrc),`
	`FALSE) = FALSE`
		`LastCrc := 0;`
		`OldMasterCrc := 0;`
		`OldSlaveCrc := 0;`
		`MasterSeqNo := 1;`
		`SlaveSeqNo := 1;`
		`InitSeqNo := 1;`
		`DataCommand := `*`FailSafeData`*`;`
		`CommFaultReason :=`
		`INVALID_CRC;`
		`SendFrame(Reset,`
		
		`ADR(CommFaultReason),`
		`LastCrc,`
		`0,`
		`ADR(SlaveSeqNo),`
		`ADR(OldSlaveCrc),`
		`FALSE);`
		`SlaveSeqNo := 1;`
			Reset

Transition	Condition	Action	Next State
CONN_FAIL10	Frame.Command = *ProcessData* OR Frame.Command = *FailSafeData*	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_CMD; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset
CONN_FAIL11	(Frame.Command <> *Reset* AND Frame.Command <> *Session* AND Frame.Command <> *Connection* AND Frame.Command <> *Parameter* AND Frame.Command <> *ProcessData* AND Frame.Command <> *FailSafeData*)	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := UNKNOWN_CMD; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset

7.5.4.2 Watchdog expired event

Cannot occur in this state because the watchdog has not yet been started.

7.5.4.3 Reset connection event

Transition	Condition	Action	Next State
CONN_RESET3		LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := FailSafeData; CommFaultReason := 0; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset

7.5.4.4 Set Data Command event

Transition	Condition	Action	Next State
CONN_STAY2		DataCommand := DataCmd;	Connection

7.5.5 Parameter state

7.5.5.1 Frame received event

Transition	Condition	Action	Next State
PARA_OK1	Frame.Command = *ProcessData* AND BytesToBeSent = 0 AND Frame.ConnId = ConnectionId AND IS_SAFE_PARA_CORRECT(SafePara) = TRUE AND IS_CRC_CORRECT(Frame, LastCrc, ADR(MasterSeqNo), ADR(OldMasterCrc), TRUE) = TRUE	Watchdog := SafePara.Watchdog; SafeOutputs := Frame.SafeData; LastCrc := Frame.Crc0; SendFrame(*DataCommand*, ADR(SafeInputs), LastCrc, ConnectionId, ADR(SlaveSeqNo), ADR(OldSlaveCrc), TRUE); LastCrc := SendFrame.Crc0; START_WD(Watchdog);	Data
PARA_OK2	Frame.Command = *FailSafeData* AND BytesToBeSent = 0 AND Frame.ConnId = ConnectionId AND IS_SAFE_PARA_CORRECT(SafePara) = TRUE AND IS_CRC_CORRECT(Frame, LastCrc, ADR(MasterSeqNo), ADR(OldMasterCrc), TRUE) = TRUE	Watchdog := SafePara.Watchdog; SafeOutputs := FS_VALUE; LastCrc := Frame.Crc0; SendFrame(*DataCommand*, ADR(SafeInputs), LastCrc, ConnectionId, ADR(SlaveSeqNo), ADR(OldSlaveCrc), TRUE); LastCrc := SendFrame.Crc0; START_WD(Watchdog);	Data
PARA_FAIL1	(Frame.Command = *ProcessData* OR Frame.Command = *FailSafeData*) AND BytesToBeSent = 0 AND Frame.ConnId = ConnectionId AND IS_SAFE_PARA_CORRECT(SafePara) = TRUE AND IS_CRC_CORRECT(Frame, LastCrc, ADR(MasterSeqNo), ADR(OldMasterCrc), TRUE) = FALSE	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_CRC; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset

Transition	Condition	Action	Next State
PARA_FAIL2	(Frame.Command = *ProcessData* OR Frame.Command = *FailSafeData)* AND BytesToBeSent = 0 AND Frame.ConnId = ConnectionId AND IS_SAFE_PARA_CORRECT(SafePara) = FALSE	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := GET_PARA_FAULT; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset
PARA_FAIL3	(Frame.Command = *ProcessData* OR Frame.Command = *FailSafeData)* AND BytesToBeSent = 0 AND Frame.ConnId <> ConnectionId	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_CONNID; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset
PARA_FAIL4	(Frame.Command = *ProcessData* OR Frame.Command = *FailSafeData)* AND BytesToBeSent <> 0	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_CMD; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset
PARA_STAY1	Frame.Command = *Parameter* AND BytesToBeSent <> 0 AND Frame.ConnId = ConnectionId AND IS_CRC_CORRECT(Frame, LastCrc, ADR(MasterSeqNo), ADR(OldMasterCrc), TRUE) = TRUE	STORE_DATA(ADR(SafePara[ExpectedSafeParaSize- BytesToBeSent]), ADR(Frame.SafeData)); LastCrc := Frame.Crc0; SendFrame(*Parameter*, ADR(Frame.SafeData), LastCrc, ConnectionId, ADR(SlaveSeqNo), ADR(OldSlaveCrc), TRUE); LastCrc := SendFrame.Crc0; BytesToBeSent := UPDATE_BYTES_TO_BE_SENT(BytesToBeSent);	Parameter

Transition	Condition	Action	Next State
PARA_FAIL5	Frame.Command = *Parameter* AND BytesToBeSent <> 0 AND Frame.ConnId = ConnectionId AND IS_CRC_CORRECT(Frame, LastCrc, ADR(MasterSeqNo), ADR(OldMasterCrc), TRUE) = FALSE	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_CRC; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset
PARA_FAIL6	Frame.Command = *Parameter* AND BytesToBeSent <> 0 AND Frame.ConnId <> ConnectionId	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_CONNID; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset
PARA_FAIL7	Frame.Command = *Parameter* AND BytesToBeSent = 0	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_CMD; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset
PARA_RESET1	Frame.Command = *Reset* AND IS_CRC_CORRECT(Frame, 0, ADR(InitSeqNo), ADR(OldMasterCrc), FALSE) = TRUE	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; InitSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := 0; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset

Transition	Condition	Action	Next State
PARA_FAIL8	Frame.Command = *Reset* AND IS_CRC_CORRECT(Frame, 0, ADR(InitSeqNo), ADR(OldMasterCrc), FALSE) = FALSE	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; InitSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_CRC; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset
PARA_RESET2	Frame.Command = *Session* AND IS_CRC_CORRECT(Frame, 0, ADR(InitSeqNo), ADR(OldMasterCrc), FALSE) = TRUE	LastCrc := Frame.Crc0; MasterSeqNo := 2; InitSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; SessionId := CREATE_SESSION_ID(); SendFrame(Session, ADR(SessionID), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); LastCrc := SendFrame.Crc0; BytesToBeSent := UPDATE_BYTES_TO_BE_SENT(2);	Session
PARA_FAIL9	Frame.Command = *Session* AND IS_CRC_CORRECT(Frame, 0, ADR(InitSeqNo), ADR(OldMasterCrc), FALSE) = FALSE	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; InitSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_CRC; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset
PARA_FAIL10	Frame.Command = *Connection*	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := INVALID_CMD; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset

Transition	Condition	Action	Next State
PARA_FAIL11	(Frame.Command <> *Reset* AND Frame.Command <> *Session* AND Frame.Command <> *Connection* AND Frame.Command <> *Parameter* AND Frame.Command <> FailSafe*Data* *AND* Frame.Command <> Process*Data*)	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; CommFaultReason := UNKNOWN_CMD; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset

7.5.5.2 Watchdog expired event

Cannot occur in this state because the watchdog has not yet been started.

7.5.5.3 Reset connection event

Transition	Condition	Action	Next State
PARA_RESET3		LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := FailSafeData; CommFaultReason := 0; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset

7.5.5.4 Set Data Command event

Transition	Condition	Action	Next State
PARA_STAY2		DataCommand := DataCmd;	Parameter

7.5.6 Data state

7.5.6.1 Frame received event

Transition	Condition	Action	Next State
DATA_OK1	Frame.Command = *ProcessData* AND Frame.ConnId = ConnectionId AND IS_CRC_CORRECT(Frame, LastCrc, ADR(MasterSeqNo), ADR(OldMasterCrc), TRUE) = TRUE	SafeOutputs := Frame.SafeData; LastCrc := Frame.Crc0; SendFrame(DataCommand, ADR(SafeInputs), LastCrc, ConnectionId, ADR(SlaveSeqNo), ADR(OldSlaveCrc), TRUE); LastCrc := SendFrame.Crc0; START_WD(Watchdog);	Data
DATA_OK2	Frame.Command = *FailSafeData* AND Frame.ConnId = ConnectionId AND IS_CRC_CORRECT(Frame, LastCrc, ADR(MasterSeqNo), ADR(OldMasterCrc), TRUE) = TRUE	SafeOutputs := FS_VALUE; LastCrc := Frame.Crc0; SendFrame(DataCommand, ADR(SafeInputs), LastCrc, ConnectionId, ADR(SlaveSeqNo), ADR(OldSlaveCrc), TRUE); LastCrc := SendFrame.Crc0; START_WD(Watchdog);	Data
DATA_FAIL1	(Frame.Command = *ProcessData* OR Frame.Command = *FailSafeData)* AND Frame.ConnId = ConnectionId AND IS_CRC_CORRECT(Frame, LastCrc, ADR(MasterSeqNo), ADR(OldMasterCrc), TRUE) = FALSE	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; SafeOutputs := FS_VALUE; STOP_WD(); CommFaultReason := INVALID_CRC; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset
DATA_FAIL2	(Frame.Command = *ProcessData* OR Frame.Command = *FailSafeData)* AND Frame.ConnId <> ConnectionId	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; SafeOutputs := FS_VALUE; STOP_WD(); CommFaultReason := INVALID_CONNID; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset

Transition	Condition	Action	Next State
DATA_RESET1	Frame.Command = *Reset* AND IS_CRC_CORRECT(Frame, 0, ADR(InitSeqNo), ADR(OldMasterCrc), FALSE) = TRUE	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; InitSeqNo := 1; DataCommand := *FailSafeData*; SafeOutputs := FS_VALUE; STOP_WD(); CommFaultReason := 0; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset
DATA_FAIL3	Frame.Command = *Reset* AND IS_CRC_CORRECT(Frame, 0, ADR(InitSeqNo), ADR(OldMasterCrc), FALSE) = FALSE	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; InitSeqNo := 1; DataCommand := *FailSafeData*; SafeOutputs := FS_VALUE; STOP_WD(); CommFaultReason := INVALID_CRC; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset
DATA_RESET2	Frame.Command = *Session* AND IS_CRC_CORRECT(Frame, 0, ADR(InitSeqNo), ADR(OldMasterCrc), FALSE) = TRUE	LastCrc := Frame.Crc0; MasterSeqNo := 2; InitSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; SafeOutputs := FS_VALUE; STOP_WD(); SessionId := CREATE_SESSION_ID(); SendFrame(Session, ADR(SessionID), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); LastCrc := SendFrame.Crc0; BytesToBeSent := UPDATE_BYTES_TO_BE_SENT(2);	Session

Transition	Condition	Action	Next State
DATA_FAIL4	Frame.Command = *Session* AND IS_CRC_CORRECT(Frame, 0, ADR(InitSeqNo), ADR(OldMasterCrc), FALSE) = FALSE	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; InitSeqNo := 1; DataCommand := *FailSafeData*; SafeOutputs := FS_VALUE; STOP_WD(); CommFaultReason := INVALID_CRC; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset
DATA_FAIL5	Frame.Command = *Connection* OR Frame.Command = *Parameter*	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; SafeOutputs := FS_VALUE; STOP_WD(); CommFaultReason := INVALID_CMD; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset
DATA_FAIL6	(Frame.Command <> *Reset* AND Frame.Command <> *Session* AND Frame.Command <> *Connection* AND Frame.Command <> *Parameter* AND Frame.Command <> FailSafe*Data* *AND* Frame.Command <> Process*Data*)	LastCrc := 0; OldMasterCrc := 0; OldSlaveCrc := 0; MasterSeqNo := 1; SlaveSeqNo := 1; DataCommand := *FailSafeData*; SafeOutputs := FS_VALUE; STOP_WD(); CommFaultReason := UNKNOWN_CMD; SendFrame(Reset, ADR(CommFaultReason), LastCrc, 0, ADR(SlaveSeqNo), ADR(OldSlaveCrc), FALSE); SlaveSeqNo := 1;	Reset

7.5.6.2 Watchdog expired event

Transition	Condition	Action	Next State

DATA_WD		`LastCrc := 0;`
		`OldMasterCrc := 0;`
		`OldSlaveCrc := 0;`
		`MasterSeqNo := 1;`
		`SlaveSeqNo := 1;`
		`DataCommand := FailSafeData;`
		`SafeOutputs := FS_VALUE;`
		`STOP_WD();`
		`CommFaultReason := WD_EXPIRED;`
		`SendFrame(Reset,`
		`ADR(CommFaultReason),`
		`LastCrc,`
		`0,`
		`ADR(SlaveSeqNo),`
		`ADR(OldSlaveCrc),`
		`FALSE);`
		`SlaveSeqNo := 1;`
			Reset

7.5.6.3 Reset connection event

Transition	Condition	Action	Next State

DATA_RESET3		`LastCrc := 0;`
		`OldMasterCrc := 0;`
		`OldSlaveCrc := 0;`
		`MasterSeqNo := 1;`
		`SlaveSeqNo := 1;`
		`DataCommand := FailSafeData;`
		`SafeOutputs := FS_VALUE;`
		`STOP_WD();`
		`CommFaultReason := 0;`
		`SendFrame(Reset,`
		`ADR(CommFaultReason),`
		`LastCrc,`
		`0,`
		`ADR(SlaveSeqNo),`
		`ADR(OldSlaveCrc),`
		`FALSE);`
		`SlaveSeqNo := 1;`
			Reset

7.5.6.4 Set Data Command event

Transition	Condition	Action	Next State
DATA_STAY		`DataCommand := DataCmd;`	Data

8 Safety communication layer management

8.1 FSCP 12/1 parameter handling

The FSCP 12/1 protocol supports an inherent download of the FSoE Slave parameter from the FSoE Master in the Parameter State.

8.2 FSoE communication parameters

The FSoE communication between the FSoE Master and the FSoE Slave uses the FSoE communication parameters defined in Table 39.

Table 39 – FSoE Communication parameters

Name	Data Type	Range	Description
FSoE Session ID	UINT16	0 .. 2^{16}	Randomly generated FSoE Session ID
FSoE Connection ID	UINT16	1 … 2^{16}	Unique Connection ID between FSoE Master and FSoE Slave
FSoE Sequence Number	UINT16	Init: 0 1 … 2^{16}	Increased in each FSoE Cycle
FSoE Slave Address	UINT16	1 … 2^{16}	Unique FSoE Slave Address for each FSoE Device
FSoE Watchdog time	UINT16	1 … 2^{16}	Watchdog time for the FSoE Connection in ms

9 System requirements

9.1 Indicators and switches

9.1.1 Indicator states and flash rates

The indicator states and flash rates are defined in Table 40 and in Figure 11. The times listed shall be met with a tolerance of less than +/- 20%.

Table 40 – Indicator States

Indicator State	Definition
on	The indicator shall be constantly on
off	The indicator shall be constantly off
single flash	The indicator shall show one short flicker (50ms) followed by an off phase of at least 200 ms
flickering	The indicator shall turn on and off iso-phase with a frequency of 10 Hz: on for 50 ms and off for 50 ms
blinking	The indicator shall turn on and off iso-phase with a frequency of 2,5 Hz: on for 200 ms followed by off for 200 ms
flickering with 1 flash	The indicator shall show first flickering for 500 ms than one off phase (500 ms) than a short flash (200 ms) followed by a long off phase (1 000 ms)
flickering with 2 flashes	The indicator shall show first flickering for 500 ms than one off phase (500 ms) than a sequence of two short flashes (200 ms), separated by an off phase (200 ms) followed by a long off phase (1 000 ms)
flickering with *n* flashes	The indicator shall show first flickering for 500 ms than one off phase (500 ms) than a sequence of *n* short flashes (200 ms), separated by an off phase (200 ms) followed by a long off phase (1 000 ms)

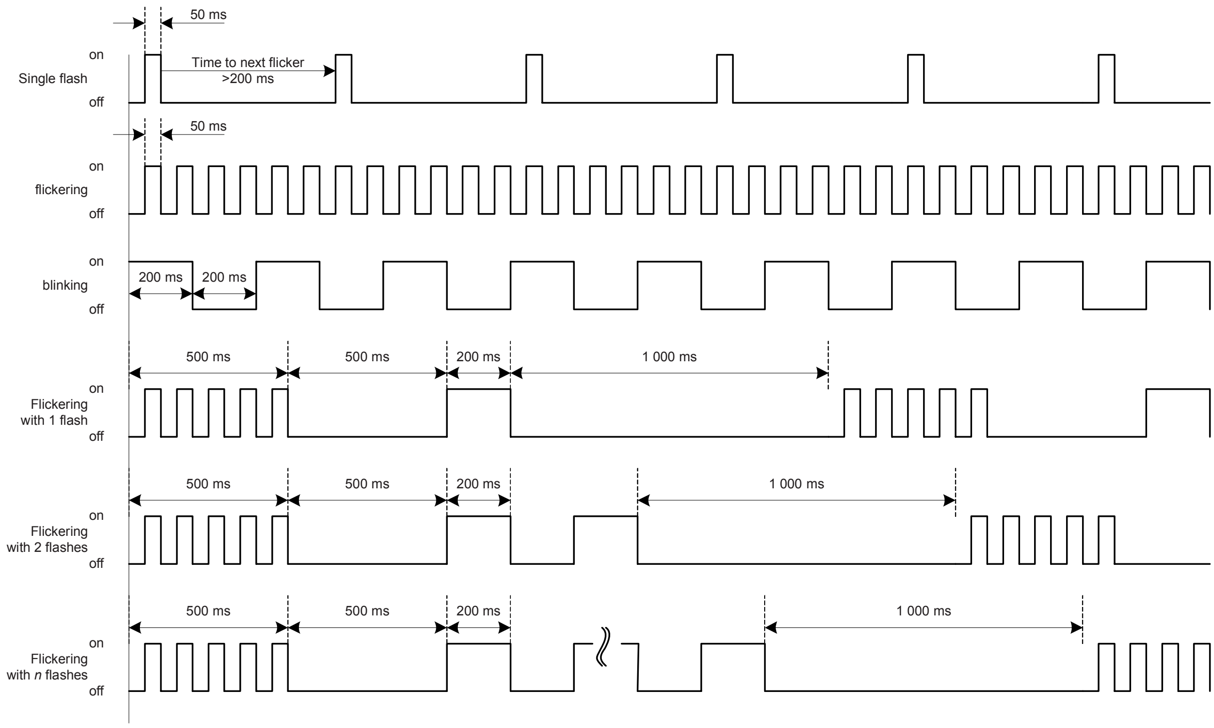


Figure 11 – Indicator flash rates

9.1.2 Indicators

9.1.2.1 Required indicators

Devices supporting the FSCP 12/1 protocol should have a status indicator to support visual inspection and troubleshooting of the FSoE Connection. If a device supports any of the indicators described here, they shall adhere to the specifications.

Additional indicators may be implemented.

9.1.2.2 FSoE STATUS indicator

The FSoE STATUS indicator shall show the status of the FSoE Connection. The colour shall be green.

In order to meet space constraints, labelling of the FSoE STATUS indicator may be omitted. If it is labelled, it shall be labelled with one of the following (no case sensitivity):

- FS
- FSoE
- FSoE Status

The FSoE STATUS indicator states are specified in Table 41.

Table 41 – FSoE STATUS indicator states

Indicator State	Definition	FSoE State
off	Initialising	Pre-Reset
blinking	Ready for Parameterization	Reset, Session, Connection, Parameter
on	Normal operation	Process Data
single flash	Failsafe-Data	Failsafe Data

Indicator State	Definition	FSoE State
flickering	Unspecified Error in the FSoE Connection	All
flickering with 1 flash	Error in F-Parameter	Parameter
flickering with 2 flashes	Error in Application Parameter	Parameter
flickering with 3 flashes	Wrong Safety Address	Connection
flickering with 4 flashes	Wrong Command	All
flickering with 5 flashes	Watchdog-Error	All
flickering with 6 flashes	CRC-Error	All

The indication of an error (flickering) shall last until the FSoE Connection changes from Reset to Session.

At least one full sequence of a flickering with *n* flashes indication shall be shown.

9.2 Installation guidelines

This part specifies a protocol and services for a safety communication system based on IEC 61158 Type 12. However, usage of safety devices with the safety protocol specified in this part requires proper installation. All devices connected to a safety communication system defined in this part shall fulfil SELV/PELV requirements, which are specified in the relevant IEC standards such as IEC 60204-1. Further relevant installation guidelines are specified in IEC 61918.

9.3 Safety function response time

9.3.1 General

To determine the safety function response time, the safety function is decomposed into several components shown in Figure 12.

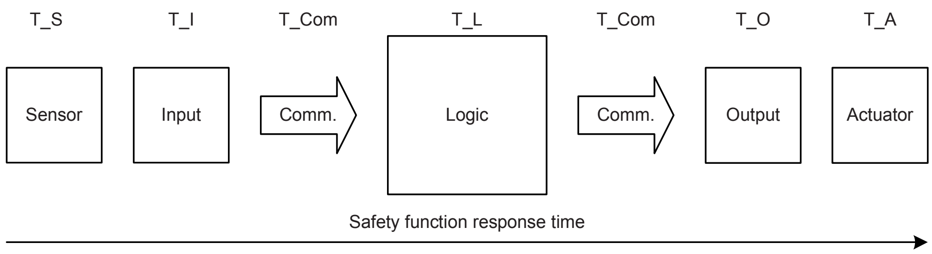


Figure 12 – Components of a safety function

Not all components need to be present in a system. The sensor (for example light curtain or Emergency Stop button) converts the physical signal into an electrical signal. This signal can be connected to an input device (for example safety input) that converts the signal into logical information. This logical information is transferred via the safety communication network to the safety logic. The safety logic (for example safety PLC) combines this and/or other input information to logical output information. The output information is transferred to the output device (for example safety output) and converted into an electrical signal. This signal is connected to the actuator, which performs the physical reaction, for example switch off the power of a drive.

General assumptions regarding communication errors are as follow:

- All components work asynchronous.
- The processing of the input signals / information is independent to the processing of the output signals / information. This means that each side can have its own time behaviour.
- In order to calculate the safety function response time, only one error or failure shall be assumed in the overall system. This error or failure shall be assumed to occur in that part of the signal path, which contributes the maximum difference time between its worst case delay time and its watchdog time. This means that concurrent failures are not considered.

Table 42 defines the times of the components.

Table 42 – Definition of times

Time	Name	Description
T_SFR	Safety Function Response Time	Safety function response time from the physical input signal to the reaction on the actuator
T_InCon	Input connection time	Time to transfer the physical input signal to the safety logic
T_OutCon	Output connection time	Time to transfer the calculated output signal from the safety logic to the actuator
T_S	Sensor-Time	Conversion time of the safety sensor
T_I	Input-Time	Delay time of the safety input device
T_Com	Communication Time	Communication cycle time of the communication network
T_L	Logic Time	Delay time of the logic (cycle)
T_O	Output Time	Delay time of the safety output device
T_A	Actuator Time	Conversion time of the safety actuator
T_WD_In	Input Watchdog time	FSoE Watchdog time of the input connection
T_WD_Out	Output Watchdog time	FSoE Watchdog time of the output connection
ΔT	Watchdog margin	Additional margin on minimum watchdog time

Because of the assumption that all components work asynchronously, the worst case time for each component is twice the delay of the component. This is the case, if the proceeding information becomes available just after the process has started. The worst case times are marked with a *_wc* suffix.

9.3.2 Determination of FSoE Watchdog time

In Figure 13 the basic schema for the calculation of the FSoE Watchdog time for the input and the output connection is shown.

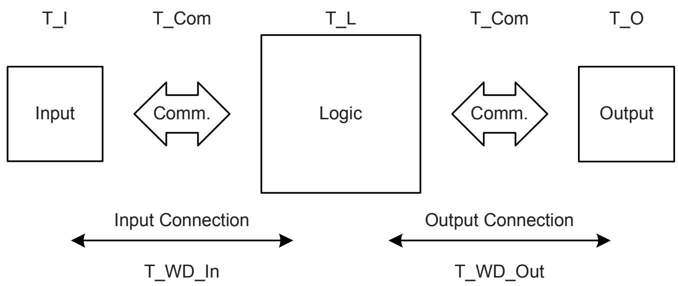


Figure 13 – Calculation of the FSoE Watchdog times for input and output connections

To determine the watchdog time of the input connection T_WD_In, Equation (1) can be used:

$$T\_WD\_In = T\_I\_wc + T\_Com\_wc + T\_L\_wc + T\_Com\_wc + \Delta T = 2 \times T\_I + 4 \times T\_Com + 2 \times T\_L + \Delta T \quad (1)$$

By analogy, Equation (2) calculates the watchdog time of the output connection T_WD_Out:

$$T\_WD\_Out = T\_Com\_wc + T\_L\_wc + T\_Com\_wc + T\_O\_wc + \Delta T = 4 \times T\_Com + 2 \times T\_L + 2 \times T\_O + \Delta T \quad (2)$$

9.3.3 Calculation of the worst case safety function response time

In Figure 14, the basic schema for the calculation of the worst case safety function response time is shown.

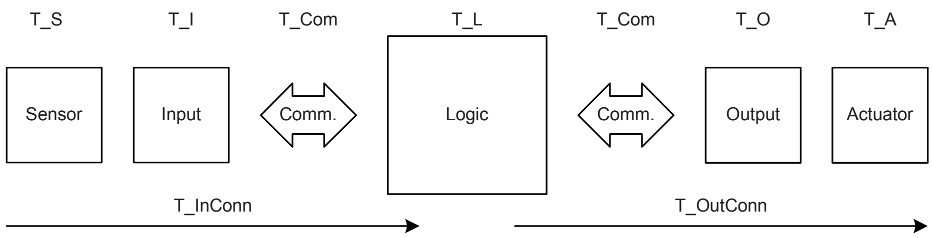


Figure 14 – Calculation of the worst case safety function response time

The time to transfer the sensor signal information to the safety logic T_InConn can be calculated as:

$$T\_InConn = T\_S\_wc + T\_I\_wc + T\_Com\_wc + T\_L\_wc = 2 \times T\_S + 2 \times T\_I + 2 \times T\_Com + 2 \times T\_L \quad (3)$$

The worst case time to get the safe state information from the sensor signal to the safety logic T_InConn_wc occurs when the input communication is interrupted and the input connection watchdog time expires. In this case the fail-safe values of the Input signals are used in the safety-logic. It can be calculated as:

$$T\_InConn\_wc = T\_S\_wc + T\_WD\_In = 2 \times T\_S + T\_WD\_In \quad (4)$$

The time to get the calculated output signal from the safety logic to the actuator T_OutConn can be calculated as:

$$T\_OutConn = T\_L\_wc + T\_Com\_wc + T\_O\_wc + T\_A\_wc = 2 \times T\_L + 2 \times T\_Com + 2 \times T\_O + 2 \times T\_A \quad (5)$$

The worst case time to get the calculated output signal from the safety logic to the actuator T_OutConn_wc occurs when the output communication is interrupted and the output connection watchdog time expires. In this case the fail-safe values in the output device are activated. It can be calculated as:

$$T\_OutConn\_wc = T\_L\_wc + T\_WD\_Out + T\_A\_wc = 2 \times T\_L + T\_WD\_In + 2 \times T\_A \quad (6)$$

In order to calculate the safety function response time one error or failure shall be assumed in that signal path, which contributes the maximum difference time between its worst case delay time and its watchdog time.

To determine the worst case safety function response time T_SFR_wc, Equation (7) can be used:

$$T\_SFR\_wc = \max\{T\_InConn\_wc + T\_OutConn \ ; \ T\_OutConn\_wc + T\_InConn\} \quad (7)$$

System manufacturers shall provide their individual adapted calculation method if necessary.

9.4 Duration of demands

The duration of demand by the safety-related application to the safety communication layer may be present as long as, or, longer than, the Process Safety Time or the FSCP 12/1 timeout (FSoE Watchdog).

9.5 Constraints for calculation of system characteristics

9.5.1 General

The FSCP 12/1 makes no restrictions regarding:

- minimum communication cycle time;
- number of safety data per FSoE Device;
- underlying communication system.

All devices shall provide electrical safety SELV/PELV.

Safety devices are designed for normal industrial environment according to IEC 61000-6-2 or IEC 61131-2 and provide increased immunity according to IEC 61326-3-1 or IEC 61326-3-2.

The communication path is arbitrary; it can be a fieldbus system, Ethernet or similar paths, fibre optics, CU-wires or even wireless transmission. There are no restrictions or requirements on bus coupler or other devices in the communication path.

The additional insertion of three zero octets in the CRC calculation, together with the CRC inheritance, guarantee the independence of the underlying communication even if the same CRC polynomial is used.

The communication interface in the Safety Devices can be a one channel interface. It may be a redundant interface due to availability.

The connection between the FSoE Devices is a Master to Slave Connection. The FSoE Master has one or several FSoE Connections to one or several FSoE Slaves. The FSoE Slave only reacts on the FSoE Master. Up to 65 535 FSoE Connections can be distinguished in a system.

9.5.2 Probabilistic considerations

Every detected error in the safety communication shall initiate a transition in the reset state, i.e. in a safe state. This transition shall not occur more than once in 5 hours, i.e. the residual error rate shall be better than 10^{-2}/h.

It is proved that the CRC Polynomial with the insertion of three zero octets (so called virtual bits) guarantees the independence to the underlying standard check.

The Type 12 PDU consists of a safety and a standard part. The safety part is embedded in the standard part. Figure 15 shows the PDU consisting of the SafetyData ND_{safety}, the virtual Bits with length d_{safety} = 24 bit, the Safety FCS_{safety}, the standard payload data $ND_{standard}$ and the standard $FCS_{standard}$.

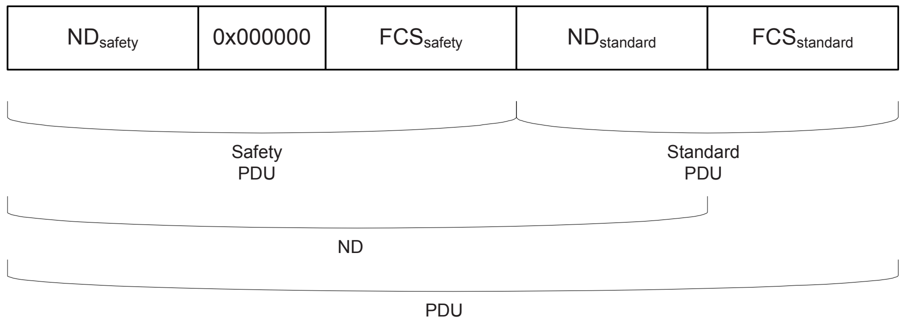


Figure 15 – Safety PDU embedded in standard PDU

The following requirements have been derived:

- $x^{dsafety}$+1 and the Generator polynomial are prime to each other;
- the number of virtual bits d_{safety} is lower or equal the number of bits for the standard part ($d_{safety} \leq n_{standard}$);
- the residual error rate is below 10^{-9}/h.

With the primitive Safety Polynomial 139B7h these requirements are fulfilled under the following conditions:

- the number of safety data bits is 8 or 16 (ND_{safety} = 8 or ND_{safety} = 16);
- the number of virtual bits is 24 (d_{safety} = 24);
- the minimum number of standard bits is 16 ($ND_{standard} \geq 16$);
- the maximum number of standard bits is 12 144 ($ND_{standard} \leq$ 12 144);

  NOTE Proof has been provided for up to 12 144 bits (1 518 octets) as used for Ethernet as a maximum DPDU length.

- the standard bits can contain again safety data blocks, consisting of safety data and FCS_{safety}.

In Figure 16, the residual error rate for 8, 16, and 24 bit safety data is shown. With a maximum bit error probability of 10^{-2}/h, the residual error rate is below 10^{-9}/h for 8 and 16 bit safety data. 24 bit safety data is not used within this protocol.

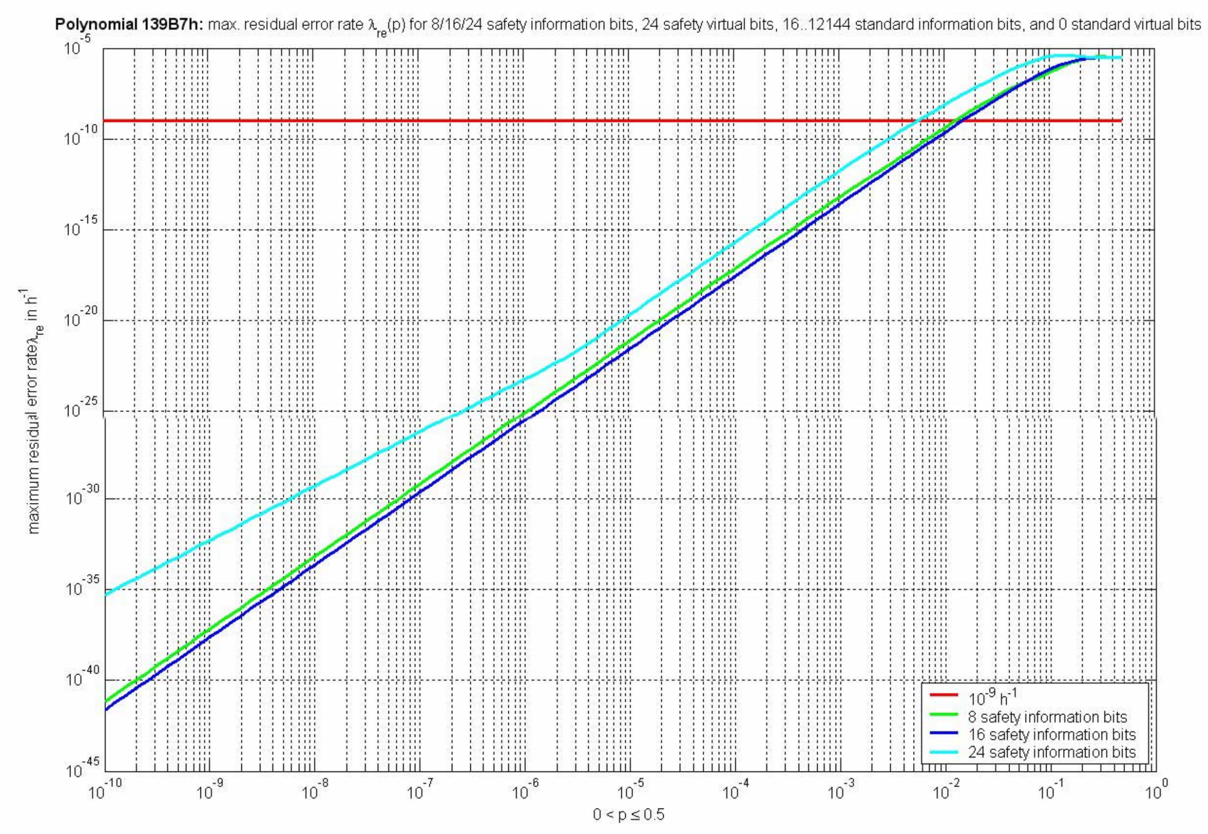


Figure 16 – Residual error rate for 8/16/24 bit safety data and up to 12 144 bit standard data

9.6 Maintenance

There are no special maintenance requirements for this protocol.

9.7 Safety manual

Implementers of this part shall supply a safety manual with following information, at a minimum:

- The safety manual shall inform the users of constraints for calculation of system characteristics, see 9.5.
- The safety manual shall inform the users of their responsibilities in the proper parameterization of the device.

In addition to the requirements of this clause the safety manual shall follow all requirements in IEC 61508.

10 Assessment

It is highly recommended that implementers of FSCP 12/1 obtain verification from an independent competent assessment body for all functional safety aspects of the product, both the protocol and any application. It is highly recommended that implementers of FSCP 12/1 obtain proof that a suitable conformance test has been performed by an independent competent assessment body.

Annex A
(informative)

Additional information for functional safety communication profiles of CPF 12

A.1 Hash function calculation

The following code for a Safety PDU represents an example of how to calculate the CRCs of the Safety PDU. The three trailing zeros are already taken into account in the tables.

```
****************************************************************
**  Parameter: psPacket    - FSCP12/1 Safety PDU
**             startCrc    - Startvalue of CRC Calculatoin
**             seqNo       - SeqNo
**             oldCRC      - CRC_0 of the last received/send Safety Slave PDU
**             bRcvDir     - bRcvDir = True:  calc of CRCs of the received Frame
**                           bRcvDir = False: calc of CRCs for the send Frame
**             size        - size of Safety PDU
**
**  Return:  bSuccess - TRUE: CRC korrekt
**
***************************************************************/

UINT8 CalcCrc(SAFETY_PDU *psPacket, UINT16 startCrc, UINT16 * seqNo, UINT16 oldCrc,
UINT8 bRcvDir, UINT8 size)
{
       UINT8 bSuccess = FALSE;
       UINT16 w1,w2;              // temporary values
       UINT16 crc;
       UINT16 crc_common;         // common part of CRC calculation,
                                  // includes CRC_0, Conn-ID, Sequence-No., Cmd
       UINT8 *pCrc = &psPacket->au8Data[2]; // pointer to CRC Low-Byte
       UINT8 *pSafeData                  // pointer to SafeData Low-Byte

if ( size > 6 )                   // that means 2 or a multiple of two safety data
       pCrc++;                    // → Crc0 Low-Byte at Byteoffset 3 instead of 2
do
{
       crc = 0;                   // reset crc

// Sequence for calcultaion:
// old CRC-Lo, old CRC-Hi, ConnId-Lo, ConnId-Hi, SeqNo-Lo, SeqNo-Hi, Command,
// (Index,) Data

       // CRC-Lo
       w1 = aCRCTab1[((UINT8 *) &crc)[HI_BYTE]];      // look at the CRC-table
       w2 = aCRCTab2[((UINT8 *) &startCrc)[0]];       // look at the CRC-table
       w1 = w1 XOR w2;
       ((UINT8 *) &crc)[HI_BYTE] = ((UINT8 *) &w1)[HI_BYTE] XOR ((UINT8 *)
                                   &crc)[LO_BYTE];
       ((UINT8 *) &crc)[LO_BYTE] = ((UINT8 *) &w1)[LO_BYTE];

       // CRC-Hi
       w1 = aCRCTab1[((UINT8 *) &crc)[HI_BYTE]];
       w2 = aCRCTab2[((UINT8 *) &startCrc)[1]];
       w1 = w1 XOR w2;
       ((UINT8 *) &crc)[HI_BYTE] = ((UINT8 *) &w1)[HI_BYTE] XOR ((UINT8 *)
                                   &crc)[LO_BYTE];
       ((UINT8 *) &crc)[LO_BYTE] = ((UINT8 *) &w1)[LO_BYTE];

       // ConnId-Lo
       w1 = aCRCTab1[((UINT8 *) &crc)[HI_BYTE]];
       w2 = aCRCTab2[psPacket->au8Data[size-2]];
       w1 = w1 XOR w2;
```

```
        ((UINT8 *) &crc)[HI_BYTE] = ((UINT8 *) &w1)[HI_BYTE] XOR ((UINT8 *)
                                     &crc)[LO_BYTE];
        ((UINT8 *) &crc)[LO_BYTE] = ((UINT8 *) &w1)[LO_BYTE];

        // ConnId-Hi
        w1 = aCRCTab1[((UINT8 *) &crc)[HI_BYTE]];
        w2 = aCRCTab2[psPacket->au8Data[size-1]];
        w1 = w1 XOR w2;
        ((UINT8 *) &crc)[HI_BYTE] = ((UINT8 *) &w1)[HI_BYTE] XOR ((UINT8 *)
                                     &crc)[LO_BYTE];
        ((UINT8 *) &crc)[LO_BYTE] = ((UINT8 *) &w1)[LO_BYTE];

        // SeqNo-Lo
        w1 = aCRCTab1[((UINT8 *) &crc)[HI_BYTE]];
        w2 = aCRCTab2[((UINT8 *) seqNo)[LO_BYTE]];
        w1 = w1 XOR w2;
        ((UINT8 *) &crc)[HI_BYTE] = ((UINT8 *) &w1)[HI_BYTE] XOR ((UINT8 *)
                                     &crc)[LO_BYTE];
        ((UINT8 *) &crc)[LO_BYTE] = ((UINT8 *) &w1)[LO_BYTE];

        // SeqNo-Hi
        w1 = aCRCTab1[((UINT8 *) &crc)[HI_BYTE]];
        w2 = aCRCTab2[((UINT8 *) seqNo)[HI_BYTE]];
        w1 = w1 XOR w2;
        ((UINT8 *) &crc)[HI_BYTE] = ((UINT8 *) &w1)[HI_BYTE] XOR ((UINT8 *)
                                     &crc)[LO_BYTE];
        ((UINT8 *) &crc)[LO_BYTE] = ((UINT8 *) &w1)[LO_BYTE];

        // Command
        w1 = aCRCTab1[((UINT8 *) &crc)[HI_BYTE]];
        w2 = aCRCTab2[psPacket->au8Data[OFFS_COMMAND]];
        w1 = w1 XOR w2;
        ((UINT8 *) &crc)[HI_BYTE] = ((UINT8 *) &w1)[HI_BYTE] XOR ((UINT8 *)
                                     &crc)[LO_BYTE];
        ((UINT8 *) &crc)[LO_BYTE] = ((UINT8 *) &w1)[LO_BYTE];

        // CRC part that is common for all other crc-calculations is saved
        crc_common = crc;

        // Data [0]
        w1 = aCRCTab1[((UINT8 *) &crc)[HI_BYTE]];
        w2 = aCRCTab2[psPacket->au8Data[OFFS_DATA]];
        w1 = w1 XOR w2;
        ((UINT8 *) &crc)[HI_BYTE] = ((UINT8 *) &w1)[HI_BYTE] XOR ((UINT8 *)
                                     &crc)[LO_BYTE];
        ((UINT8 *) &crc)[LO_BYTE] = ((UINT8 *) &w1)[LO_BYTE];

        // if 2 Byte Safety data → calculate next Byte into the crc
        if ( size > 6 )
        {
               // Data [1]
               w1 = aCRCTab1[((UINT8 *) &crc)[HI_BYTE]];
               w2 = aCRCTab2[psPacket->au8Data[OFFS_DATA+1]];
               w1 = w1 XOR w2;
               ((UINT8 *) &crc)[HI_BYTE] = ((UINT8 *) &w1)[HI_BYTE] XOR ((UINT8 *)
                                            &crc)[LO_BYTE];
               ((UINT8 *) &crc)[LO_BYTE] = ((UINT8 *) &w1)[LO_BYTE];
        }

        // UPDATE_SEQ_NO
        seqNo[0]++;
        if (seqNo[0] == 0)
               seqNo[0]++;

} while ( crc == oldCrc && (bRcvDir & NEW_CRC) != 0 );
               // as long as resulting crc is the same like oldCrc

if (bRcvDir) // for receive direction
{       if ( ((UINT8 *) &crc)[HI_BYTE] == pCrc[OFFS_CRC_HI-OFFS_CRC_LO]
             && ((UINT8 *) &crc)[LO_BYTE] == pCrc[0] )
```

```
        {       // for receive direction
                // CRC is correct
                bSuccess = TRUE;
        }
}
else                    // for send direction
{
        // insert Checksum
        pCrc[OFFS_CRC_HI-OFFS_CRC_LO] = ((UINT8 *) &crc)[HI_BYTE];
        pCrc[0] = ((UINT8 *) &crc)[LO_BYTE];
}


// if more than 2 Byte Safety Data are transferred,
// CRC_1 and so forth must be calculated
if ( size > 10 )
{
        UINT16 i     = 1;
        pSafeData    = pCrc+2;     // set pSafeData to the SafeData Low-Byte
                                   // of the next part = SafeData[2]
        pCrc          += 4;        // set pCrc to CRC_i Low-Byte
        size -= 7;                 // substract first part of the frame
        while ( size >= 4 ) // as long as other parts follow
        {
                // Start-CRC
                crc = crc_common;   // this part is already calculated above

                // i (Bit 0-7)      // calculate index
                w1 = aCRCTab1[((UINT8 *) &crc)[HI_BYTE]];
                w2 = aCRCTab2[((UINT8 *) &i)[LO_BYTE]];
                w1 = w1 XOR w2;
                ((UINT8 *) &crc)[HI_BYTE] = ((UINT8 *) &w1)[HI_BYTE] XOR ((UINT8 *)
                                              &crc)[LO_BYTE];
                ((UINT8 *) &crc)[LO_BYTE] = ((UINT8 *) &w1)[LO_BYTE];

                // i (Bit 8-15)
                w1 = aCRCTab1[((UINT8 *) &crc)[HI_BYTE]];
                w2 = aCRCTab2[((UINT8 *) &i)[HI_BYTE]];
                w1 = w1 XOR w2;
                ((UINT8 *) &crc)[HI_BYTE] = ((UINT8 *) &w1)[HI_BYTE] XOR ((UINT8 *)
                                              &crc)[LO_BYTE];
                ((UINT8 *) &crc)[LO_BYTE] = ((UINT8 *) &w1)[LO_BYTE];

                // Data 2*i
                w1 = aCRCTab1[((UINT8 *) &crc)[HI_BYTE]];
                w2 = aCRCTab2[pSafeData[0]];
                w1 = w1 XOR w2;
                ((UINT8 *) &crc)[HI_BYTE] = ((UINT8 *) &w1)[HI_BYTE] XOR ((UINT8 *)
                                              &crc)[LO_BYTE];
                ((UINT8 *) &crc)[LO_BYTE] = ((UINT8 *) &w1)[LO_BYTE];

                // Data 2*i+1
                w1 = aCRCTab1[((UINT8 *) &crc)[HI_BYTE]];
                w2 = aCRCTab2[pSafeData[1]];
                w1 = w1 XOR w2;
                ((UINT8 *) &crc)[HI_BYTE] = ((UINT8 *) &w1)[HI_BYTE] XOR ((UINT8 *)
                                              &crc)[LO_BYTE];
                ((UINT8 *) &crc)[LO_BYTE] = ((UINT8 *) &w1)[LO_BYTE];

                if ( ((UINT8 *) &crc)[HI_BYTE] == pCrc [1]
                  && ((UINT8 *) &crc)[LO_BYTE] == pCrc [0] )
                {
                        // CRC is correct
                }
                else
                {
                        bSuccess = FALSE;
                        if ( bRcvDir == 0)  // for send direction
                        {
                                // insert Checksum
```

```
                        pCrc [1] = ((UINT8 *) &crc)[HI_BYTE];
                        pCrc [0] = ((UINT8 *) &crc)[LO_BYTE];
                    }
                }

                size      -= 4;            // substract this part of the frame
                pSafeData += 4;            // set to next SafeData Low Byte
                pCrc0     += 4;            // set to next CRC_i Low Byte
                i++;                       // increment Index
            }
        }

        return bSuccess;
}
```

The following two tables are used:

```
aCrcTab1: ARRAY[0..255] OF WORD :=
16#0000,16#39B7,16#736E,16#4AD9,16#E6DC,16#DF6B,16#95B2,16#AC05,16#F40F,16#CDB8,
16#8761,16#BED6,16#12D3,16#2B64,16#61BD,16#580A,16#D1A9,16#E81E,16#A2C7,16#9B70,
16#3775,16#0EC2,16#441B,16#7DAC,16#25A6,16#1C11,16#56C8,16#6F7F,16#C37A,16#FACD,
16#B014,16#89A3,16#9AE5,16#A352,16#E98B,16#D03C,16#7C39,16#458E,16#0F57,16#36E0,
16#6EEA,16#575D,16#1D84,16#2433,16#8836,16#B181,16#FB58,16#C2EF,16#4B4C,16#72FB,
16#3822,16#0195,16#AD90,16#9427,16#DEFE,16#E749,16#BF43,16#86F4,16#CC2D,16#F59A,
16#599F,16#6028,16#2AF1,16#1346,16#0C7D,16#35CA,16#7F13,16#46A4,16#EAA1,16#D316,
16#99CF,16#A078,16#F872,16#C1C5,16#8B1C,16#B2AB,16#1EAE,16#2719,16#6DC0,16#5477,
16#DDD4,16#E463,16#AEBA,16#970D,16#3B08,16#02BF,16#4866,16#71D1,16#29DB,16#106C,
16#5AB5,16#6302,16#CF07,16#F6B0,16#BC69,16#85DE,16#9698,16#AF2F,16#E5F6,16#DC41,
16#7044,16#49F3,16#032A,16#3A9D,16#6297,16#5B20,16#11F9,16#284E,16#844B,16#BDFC,
16#F725,16#CE92,16#4731,16#7E86,16#345F,16#0DE8,16#A1ED,16#985A,16#D283,16#EB34,
16#B33E,16#8A89,16#C050,16#F9E7,16#55E2,16#6C55,16#268C,16#1F3B,16#18FA,16#214D,
16#6B94,16#5223,16#FE26,16#C791,16#8D48,16#B4FF,16#ECF5,16#D542,16#9F9B,16#A62C,
16#0A29,16#339E,16#7947,16#40F0,16#C953,16#F0E4,16#BA3D,16#838A,16#2F8F,16#1638,
16#5CE1,16#6556,16#3D5C,16#04EB,16#4E32,16#7785,16#DB80,16#E237,16#A8EE,16#9159,
16#821F,16#BBA8,16#F171,16#C8C6,16#64C3,16#5D74,16#17AD,16#2E1A,16#7610,16#4FA7,
16#057E,16#3CC9,16#90CC,16#A97B,16#E3A2,16#DA15,16#53B6,16#6A01,16#20D8,16#196F,
16#B56A,16#8CDD,16#C604,16#FFB3,16#A7B9,16#9E0E,16#D4D7,16#ED60,16#4165,16#78D2,
16#320B,16#0BBC,16#1487,16#2D30,16#67E9,16#5E5E,16#F25B,16#CBEC,16#8135,16#B882,
16#E088,16#D93F,16#93E6,16#AA51,16#0654,16#3FE3,16#753A,16#4C8D,16#C52E,16#FC99,
16#B640,16#8FF7,16#23F2,16#1A45,16#509C,16#692B,16#3121,16#0896,16#424F,16#7BF8,
16#D7FD,16#EE4A,16#A493,16#9D24,16#8E62,16#B7D5,16#FD0C,16#C4BB,16#68BE,16#5109,
16#1BD0,16#2267,16#7A6D,16#43DA,16#0903,16#30B4,16#9CB1,16#A506,16#EFDF,16#D668,
16#5FCB,16#667C,16#2CA5,16#1512,16#B917,16#80A0,16#CA79,16#F3CE,16#ABC4,16#9273,
16#D8AA,16#E11D,16#4D18,16#74AF,16#3E76,16#07C1;
```

```
aCrcTab2: ARRAY[0..255] OF WORD :=
16#0000,16#7648,16#EC90,16#9AD8,16#E097,16#96DF,16#0C07,16#7A4F,16#F899,16#8ED1,
16#1409,16#6241,16#180E,16#6E46,16#F49E,16#82D6,16#C885,16#BECD,16#2415,16#525D,
16#2812,16#5E5A,16#C482,16#B2CA,16#301C,16#4654,16#DC8C,16#AAC4,16#D08B,16#A6C3,
16#3C1B,16#4A53,16#A8BD,16#DEF5,16#442D,16#3265,16#482A,16#3E62,16#A4BA,16#D2F2,
16#5024,16#266C,16#BCB4,16#CAFC,16#B0B3,16#C6FB,16#5C23,16#2A6B,16#6038,16#1670,
16#8CA8,16#FAE0,16#80AF,16#F6E7,16#6C3F,16#1A77,16#98A1,16#EEE9,16#7431,16#0279,
16#7836,16#0E7E,16#94A6,16#E2EE,16#68CD,16#1E85,16#845D,16#F215,16#885A,16#FE12,
16#64CA,16#1282,16#9054,16#E61C,16#7CC4,16#0A8C,16#70C3,16#068B,16#9C53,16#EA1B,
16#A048,16#D600,16#4CD8,16#3A90,16#40DF,16#3697,16#AC4F,16#DA07,16#58D1,16#2E99,
16#B441,16#C209,16#B846,16#CE0E,16#54D6,16#229E,16#C070,16#B638,16#2CE0,16#5AA8,
16#20E7,16#56AF,16#CC77,16#BA3F,16#38E9,16#4EA1,16#D479,16#A231,16#D87E,16#AE36,
16#34EE,16#42A6,16#08F5,16#7EBD,16#E465,16#922D,16#E862,16#9E2A,16#04F2,16#72BA,
16#F06C,16#8624,16#1CFC,16#6AB4,16#10FB,16#66B3,16#FC6B,16#8A23,16#D19A,16#A7D2,
16#3D0A,16#4B42,16#310D,16#4745,16#DD9D,16#ABD5,16#2903,16#5F4B,16#C593,16#B3DB,
16#C994,16#BFDC,16#2504,16#534C,16#191F,16#6F57,16#F58F,16#83C7,16#F988,16#8FC0,
16#1518,16#6350,16#E186,16#97CE,16#0D16,16#7B5E,16#0111,16#7759,16#ED81,16#9BC9,
16#7927,16#0F6F,16#95B7,16#E3FF,16#99B0,16#EFF8,16#7520,16#0368,16#81BE,16#F7F6,
16#6D2E,16#1B66,16#6129,16#1761,16#8DB9,16#FBF1,16#B1A2,16#C7EA,16#5D32,16#2B7A,
16#5135,16#277D,16#BDA5,16#CBED,16#493B,16#3F73,16#A5AB,16#D3E3,16#A9AC,16#DFE4,
16#453C,16#3374,16#B957,16#CF1F,16#55C7,16#238F,16#59C0,16#2F88,16#B550,16#C318,
16#41CE,16#3786,16#AD5E,16#DB16,16#A159,16#D711,16#4DC9,16#3B81,16#71D2,16#079A,
16#9D42,16#EB0A,16#9145,16#E70D,16#7DD5,16#0B9D,16#894B,16#FF03,16#65DB,16#1393,
16#69DC,16#1F94,16#854C,16#F304,16#11EA,16#67A2,16#FD7A,16#8B32,16#F17D,16#8735,
16#1DED,16#6BA5,16#E973,16#9F3B,16#05E3,16#73AB,16#09E4,16#7FAC,16#E574,16#933C,
16#D96F,16#AF27,16#35FF,16#43B7,16#39F8,16#4FB0,16#D568,16#A320,16#21F6,16#57BE,
16#CD66,16#BB2E,16#C161,16#B729,16#2DF1,16#5BB9;
```

A.2 …

Void

Annex B
(informative)

Information for assessment of the functional safety communication profiles of CPF 12

Information about test laboratories which test and validate the conformance of FSCP 12/1 products with IEC 61784-3-12 can be obtained from the National Committees of the IEC or from the following organization:

EtherCAT Technology Group
Ostendstrasse 196
90482 Nuremberg
GERMANY

Phone: +49-911-54056-20
Fax: +49-911-54056-29
E-mail: info@ethercat.org
URL: www.ethercat.org

Bibliography

[1] IEC 60050 (all parts), *International Electrotechnical Vocabulary*

NOTE See also the IEC Multilingual Dictionary – Electricity, Electronics and Telecommunications (available on CD-ROM and at <http://www.electropedia.org>)

[2] IEC/TS 61000-1-2, *Electromagnetic compatibility (EMC) – Part 1-2: General – Methodology for the achievement of the functional safety of electrical and electronic equipment with regard to electromagnetic phenomena*

[3] IEC 61131-6[10], *Programmable controllers – Part 6: Functional safety*

[4] IEC 61158 (all parts), *Industrial communication networks – Fieldbus specifications*

[5] IEC 61496 (all parts), *Safety of machinery – Electro-sensitive protective equipment*

[6] IEC 61508-1:2010[11], *Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 1: General requirements*

[7] IEC 61508-4:2010[11], *Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 4: Definitions and abbreviations*

[8] IEC 61508-5:2010[11], *Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 5: Examples of methods for the determination of safety integrity levels*

[9] IEC 61511 (all parts), *Functional safety – Safety instrumented systems for the process industry sector*

[10] IEC 61784-1, *Industrial communication networks – Profiles – Part 1: Fieldbus profiles*

[11] IEC 61784-4[12], *Industrial communication networks – Profiles – Part 4: Secure communications for fieldbuses*

[12] IEC 61784-5 (all parts), *Industrial communication networks – Profiles – Part 5: Installation of fieldbuses – Installation profiles for CPF x*

[13] IEC 61800-5-2, *Adjustable speed electrical power drive systems – Part 5-2: Safety requirements – Functional*

[14] IEC/TR 62059-11, *Electricity metering equipment – Dependability – Part 11: General concepts*

[15] IEC 62061, *Safety of machinery – Functional safety of safety-related electrical, electronic and programmable electronic control systems*

[16] IEC/TR 62210, *Power system control and associated communications – Data and communication security*

[17] IEC 62280-1, *Railway applications – Communication, signalling and processing systems – Part 1: Safety-related communication in closed transmission systems*

[18] IEC 62280-2, *Railway applications – Communication, signalling and processing systems – Part 2: Safety-related communication in open transmission systems*

[19] IEC 62443 (all parts), *Industrial communication networks – Network and system security*

[20] ISO/IEC Guide 51:1999, *Safety aspects — Guidelines for their inclusion in standards*

[21] ISO/IEC 2382-14, *Information technology – Vocabulary – Part 14: Reliability, maintainability and availability*

[22] ISO/IEC 2382-16, *Information technology – Vocabulary – Part 16: Information theory*

[23] ISO/IEC 7498 (all parts), *Information technology – Open Systems Interconnection – Basic Reference Model*

---

10 In preparation.

11 To be published.

12 Proposed new work item under consideration.

[24] ISO/IEC 19501, *Information technology – Open Distributed Processing – Unified Modeling Language (UML) Version 1.4.2*

[25] ISO 10218-1, *Robots for industrial environments – Safety requirements – Part 1: Robot*

[26] ISO 12100-1, *Safety of machinery – Basic concepts, general principles for design – Part 1: Basic terminology, methodology*

[27] ISO 13849-1, *Safety of machinery – Safety-related parts of control systems – Part 1: General principles for design*

[28] ISO 13849-2, *Safety of machinery – Safety-related parts of control systems – Part 2: Validation*

[29] ISO 14121, *Safety of machinery – Principles of risk assessment*

[30] EN 954-1:1996[13], *Safety of machinery – Safety related parts of control systems – General principles for design*

[31] ANSI/ISA-84.00.01-2004 (all parts), *Functional Safety: Safety Instrumented Systems for the Process Industry Sector*

[32] VDI/VDE 2180 (all parts), *Safeguarding of industrial process plants by means of process control engineering*

[33] GS-ET-26[14], *Grundsatz für die Prüfung und Zertifizierung von Bussystemen für die Übertragung sicherheitsrelevanter Nachrichten*, May 2002. HVBG, Gustav-Heinemann-Ufer 130, D-50968 Köln ("*Principles for Test and Certification of Bus Systems for Safety relevant Communication*")

[34] ANDREW S. TANENBAUM, *Computer Networks*, 4th Edition, Prentice Hall, N.J., ISBN-10:0130661023, ISBN-13: 978-0130661029

[35] W. WESLEY PETERSON, *Error-Correcting Codes*, 2nd Edition 1981, MIT-Press, ISBN 0-262-16-039-0

[36] BRUCE P. DOUGLASS, *Doing Hard Time*, 1999, Addison-Wesley, ISBN 0-201-49837-5

[37] *New concepts for safety-related bus systems*, 3rd International Symposium "Programmable Electronic Systems in Safety Related Applications ", May 1998, from Dr. Michael Schäfer, BG-Institute for Occupational Safety and Health.

[38] DIETER CONRADS, *Datenkommunikation*, 3rd Edition 1996, Vieweg, ISBN 3-528-245891

[39] German IEC subgroup DKE AK 767.0.4: *EMC and Functional Safety*, Spring 2002

[40] NFPA79 (2002), *Electrical Standard for Industrial Machinery*

[41] GUY E. CASTAGNOLI, *On the Minimum Distance of Long Cyclic Codes and Cyclic Redundancy-Check Codes*, 1989, Dissertation No. 8979 of ETH Zurich, Switzerland

[42] GUY E. CASTAGNOLI, STEFAN BRÄUER, and MARTIN HERRMANN, *Optimization of Cyclic Redundancy-Check Codes with 24 and 32 Parity Bits*, June 1993, IEEE Transactions On Communications, Volume 41, No. 6

[43] SCHILLER F and MATTES T: *An Efficient Method to Evaluate CRC-Polynomials for Safety-Critical Industrial Communication*, Journal of Applied Computer Science, Vol. 14, No 1, pp. 57-80, Technical University Press, Łódź,Poland, 2006

[44] SCHILLER F and MATTES T: *Analysis of CRC-polynomials for Safety-critical Communication by Deterministic and Stochastic Automata*, 6th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes, SAFEPROCESS 2006, pp. 1003-1008, Beijing, China, 2006

---

13 To be replaced by ISO 13849-1 and/or IEC 62061.

14 This document has been one of the starting points for this part. It is currently undergoing a major revision.

This page deliberately left blank

British Standards Institution (BSI)

BSI is the independent national body responsible for preparing British Standards and other standards-related publications, information and services.
It presents the UK view on standards in Europe and at the international level.

It is incorporated by Royal Charter.

Revisions

British Standards are updated by amendment or revision. Users of British Standards should make sure that they possess the latest amendments or editions.

It is the constant aim of BSI to improve the quality of our products and services. We would be grateful if anyone finding an inaccuracy or ambiguity while using this British Standard would inform the Secretary of the technical committee responsible, the identity of which can be found on the inside front cover.

Tel: +44 (0)20 8996 9001 Fax: +44 (0)20 8996 7001

BSI offers Members an individual updating service called PLUS which ensures that subscribers automatically receive the latest editions of standards.

Tel: +44 (0)20 8996 7669 Fax: +44 (0)20 8996 7001
Email: plus@bsigroup.com

Buying standards

You may buy PDF and hard copy versions of standards directly using a credit card from the BSI Shop on the website **www.bsigroup.com/shop.** In addition all orders for BSI, international and foreign standards publications can be addressed to BSI Customer Services.

Tel: +44 (0)20 8996 9001 Fax: +44 (0)20 8996 7001
Email: orders@bsigroup.com

In response to orders for international standards, it is BSI policy to supply the BSI implementation of those that have been published as British Standards, unless otherwise requested.

Information on standards

BSI provides a wide range of information on national, European and international standards through its Knowledge Centre.

Tel: +44 (0)20 8996 7004 Fax: +44 (0)20 8996 7005
Email: knowledgecentre@bsigroup.com

Various BSI electronic information services are also available which give details on all its products and services.

Tel: +44 (0)20 8996 7111 Fax: +44 (0)20 8996 7048
Email: info@bsigroup.com

BSI Subscribing Members are kept up to date with standards developments and receive substantial discounts on the purchase price of standards. For details of these and other benefits contact Membership Administration.

Tel: +44 (0)20 8996 7002 Fax: +44 (0)20 8996 7001
Email: membership@bsigroup.com

Information regarding online access to British Standards via British Standards Online can be found at **www.bsigroup.com/BSOL**

Further information about BSI is available on the BSI website at **www.bsigroup.com/standards**

BSI Group Headquarters

389 Chiswick High Road London W4 4AL UK

Tel +44 (0)20 8996 9001
Fax +44 (0)20 8996 7001
www.bsigroup.com/standards

raising standards worldwide™