IEC 61784-3-12

Edition 1.0 2010-06

INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour inside

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

Réseaux de communication industriels – Profils – Partie 3-12: Bus de terrain de sécurité fonctionnelle – Spécifications supplémentaires pour CPF 12

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland

Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00
info@iec.ch
www.iec.ch

About the IEC

The International Electrotechnical Commission (IEC) is the leading global organization that prepares and publishes International Standards for all electrical, electronic and related technologies.

About IEC publications

The technical content of IEC publications is kept under constant review by the IEC. Please make sure that you have the latest edition, a corrigenda or an amendment might have been published.

Useful links:

IEC publications search - www.iec.ch/searchpub

The advanced search enables you to find IEC publications by a variety of criteria (reference number, text, technical committee,…).
It also gives information on projects, replaced and withdrawn publications.

IEC Just Published - webstore.iec.ch/justpublished

Stay up to date on all new IEC publications. Just Published details all new publications released. Available on-line and also once a month by email.

Electropedia - www.electropedia.org

The world's leading online dictionary of electronic and electrical terms containing more than 30 000 terms and definitions in English and French, with equivalent terms in additional languages. Also known as the International Electrotechnical Vocabulary (IEV) on-line.

Customer Service Centre - webstore.iec.ch/csc

If you wish to give us your feedback on this publication or need further assistance, please contact the Customer Service Centre: csc@iec.ch.

A propos de la CEI

La Commission Electrotechnique Internationale (CEI) est la première organisation mondiale qui élabore et publie des Normes internationales pour tout ce qui a trait à l'électricité, à l'électronique et aux technologies apparentées.

A propos des publications CEI

Le contenu technique des publications de la CEI est constamment revu. Veuillez vous assurer que vous possédez l'édition la plus récente, un corrigendum ou amendement peut avoir été publié.

Liens utiles:

Recherche de publications CEI - www.iec.ch/searchpub

La recherche avancée vous permet de trouver des publications CEI en utilisant différents critères (numéro de référence, texte, comité d'études,…).
Elle donne aussi des informations sur les projets et les publications remplacées ou retirées.

Just Published CEI - webstore.iec.ch/justpublished

Restez informé sur les nouvelles publications de la CEI. Just Published détaille les nouvelles publications parues. Disponible en ligne et aussi une fois par mois par email.

Electropedia - www.electropedia.org

Le premier dictionnaire en ligne au monde de termes électroniques et électriques. Il contient plus de 30 000 termes et définitions en anglais et en français, ainsi que les termes équivalents dans les langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International (VEI) en ligne.

Service Clients - webstore.iec.ch/csc

Si vous désirez nous donner des commentaires sur cette publication ou si vous avez des questions contactez-nous: csc@iec.ch.

IEC 61784-3-12

Edition 1.0 2010-06

INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour inside

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

Réseaux de communication industriels – Profils – Partie 3-12: Bus de terrain de sécurité fonctionnelle – Spécifications supplémentaires pour CPF 12

INTERNATIONAL ELECTROTECHNICAL COMMISSION

COMMISSION ELECTROTECHNIQUE INTERNATIONALE

PRICE CODE
CODE PRIX XD

ICS 25.040.40, 35.100.05

ISBN 978-2-88912-944-7

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

INDUSTRIAL COMMUNICATION NETWORKS – PROFILES –

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

FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
5) IEC itself does not provide any attestation of conformity. Independent certification bodies provide conformity assessment services and, in some areas, access to IEC marks of conformity. IEC is not responsible for any services carried out by independent certification bodies.
6) All users should ensure that they have the latest edition of this publication.
7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.

International Standard IEC 61784-3-12 has been prepared by subcommittee 65C: Industrial networks, of IEC technical committee 65: Industrial process measurement, control and automation.

This bilingual version (2012-02) corresponds to the monolingual English version, published in 2010-06.

The text of this standard is based on the following documents:

FDIS	Report on voting
65C/591A/FDIS	65C/603/RVD

Full information on the voting for the approval of this standard can be found in the report on voting indicated in the above table.

The French version of this standard has not been voted upon.

This publication has been drafted in accordance with the ISO/IEC Directives, Part 2.

A list of all parts of the IEC 61784-3 series, published under the general title *Industrial communication networks – Profiles – Functional safety fieldbuses*, can be found on the IEC website.

The committee has decided that the contents of this publication will remain unchanged until the stability date indicated on the IEC web site under "http://webstore.iec.ch" in the data related to the specific publication. At this date, the publication will be

- reconfirmed,
- withdrawn,
- replaced by a revised edition, or
- amended.

IMPORTANT – The 'colour inside' logo on the cover page of this publication indicates that it contains colours which are considered to be useful for the correct understanding of its contents. Users should therefore print this document using a colour printer.

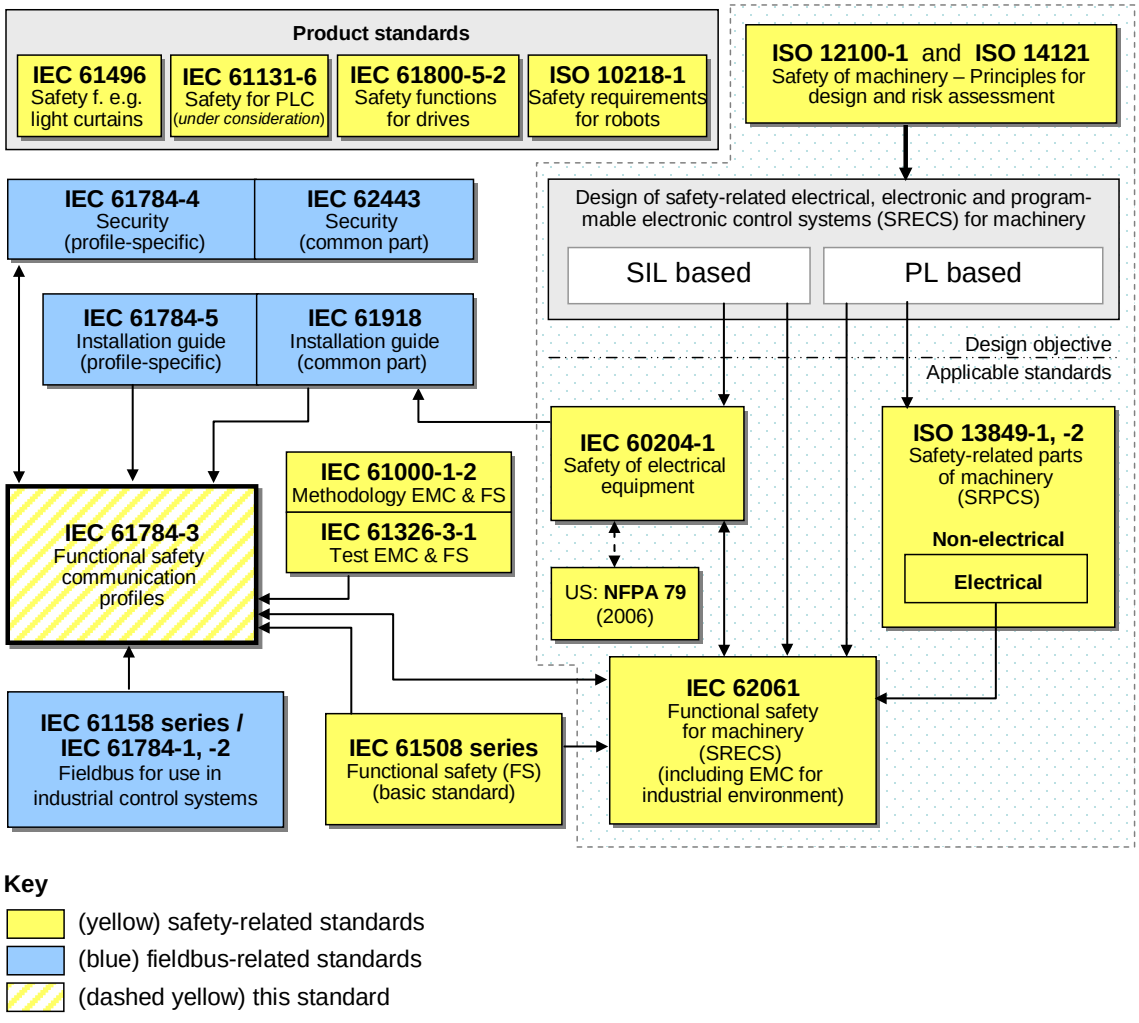
0 Introduction

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.

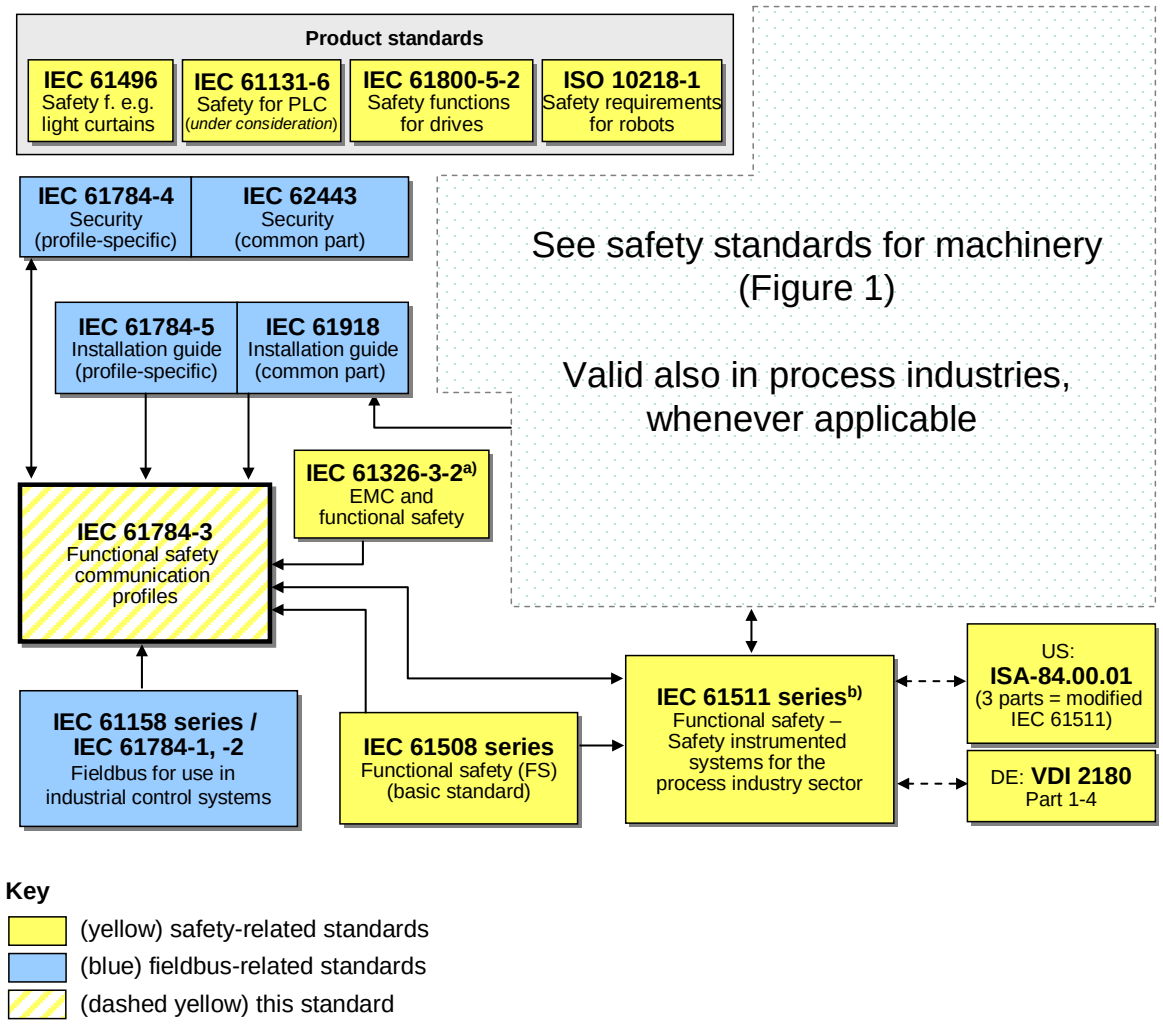
Figure 1 shows the relationships between this standard and relevant safety and fieldbus standards in a machinery environment.



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.

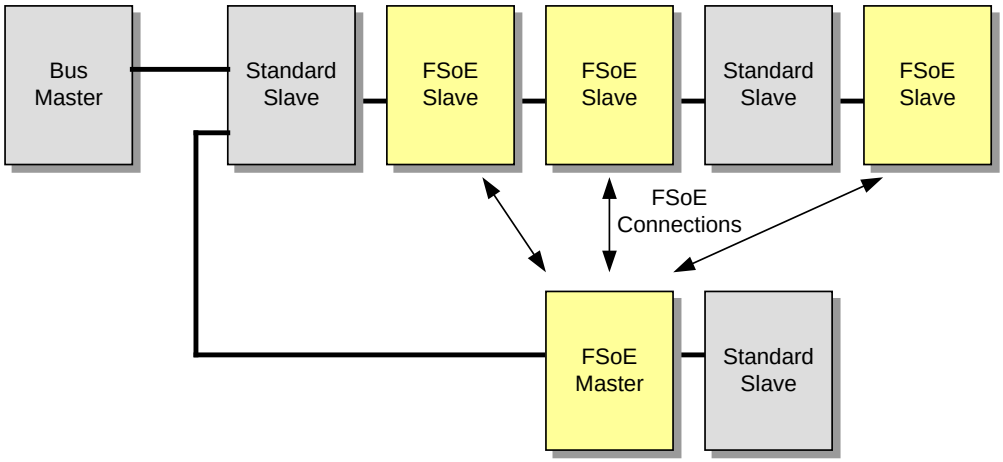


Figure 3 – Basic FSCP 12/1 system

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.

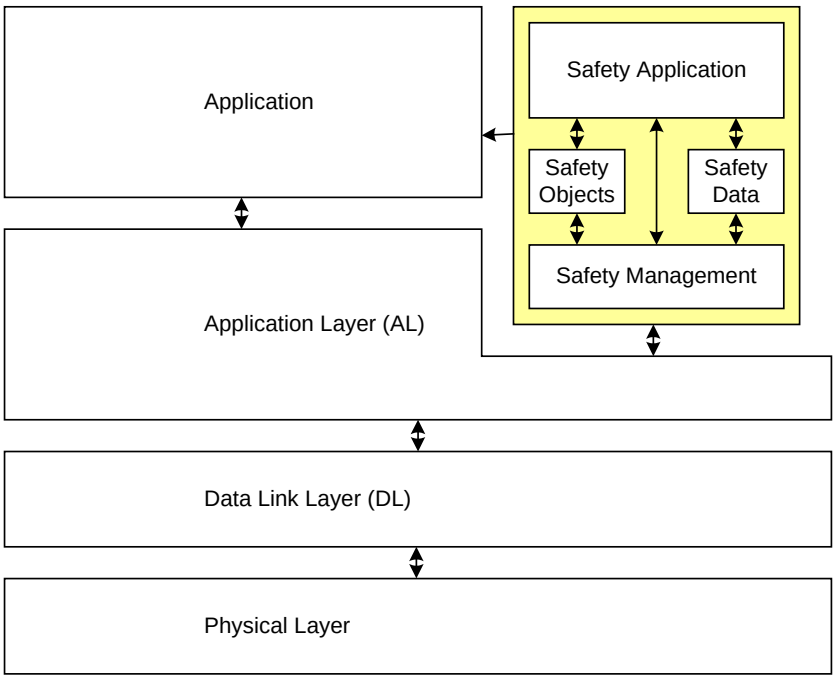


Figure 4 – FSCP 12/1 software architecture

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.

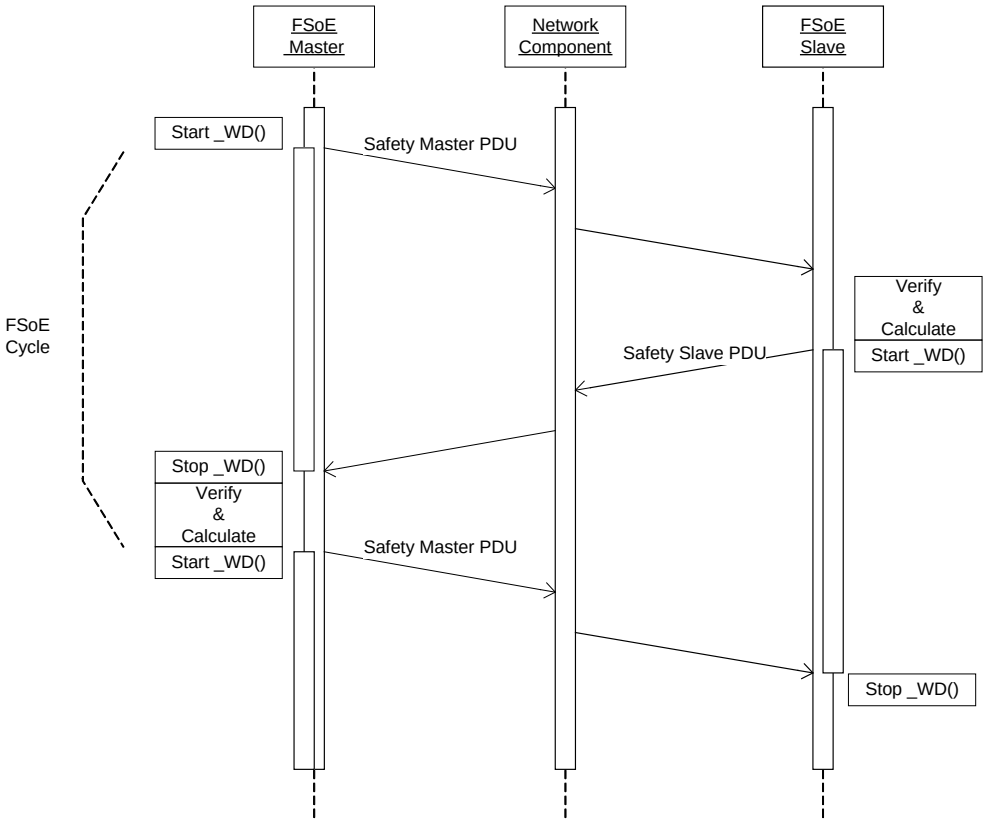


Figure 5 – FSoE Cycle

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.

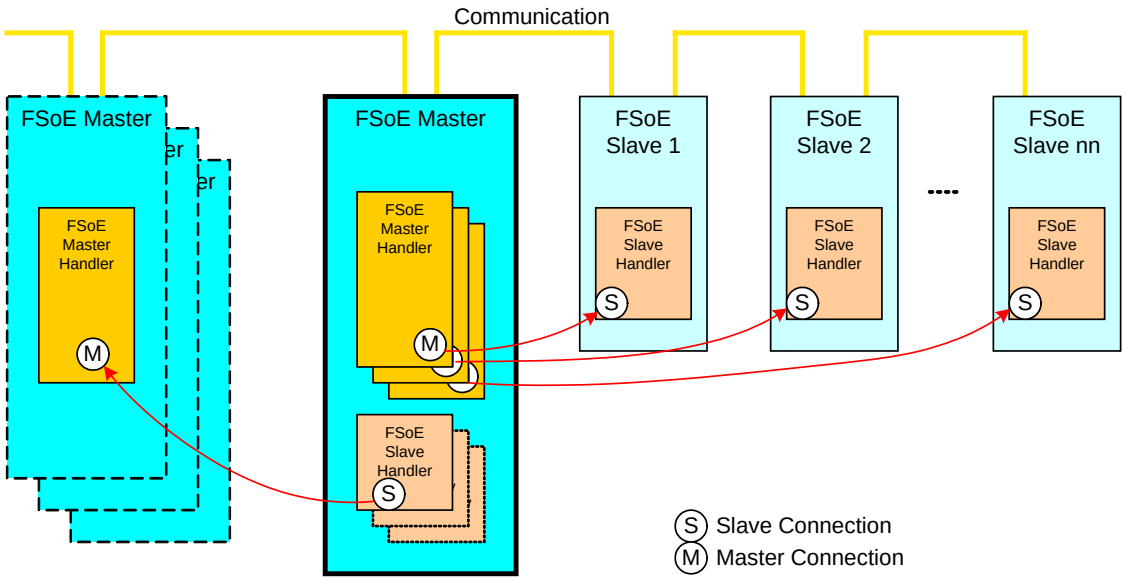


Figure 6 – FSCP 12/1 communication structure

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.

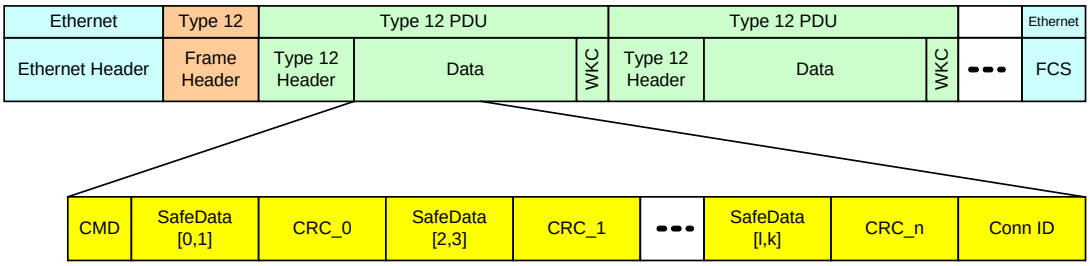


Figure 7 – Safety PDU for CPF 12 embedded in Type 12 PDU

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.

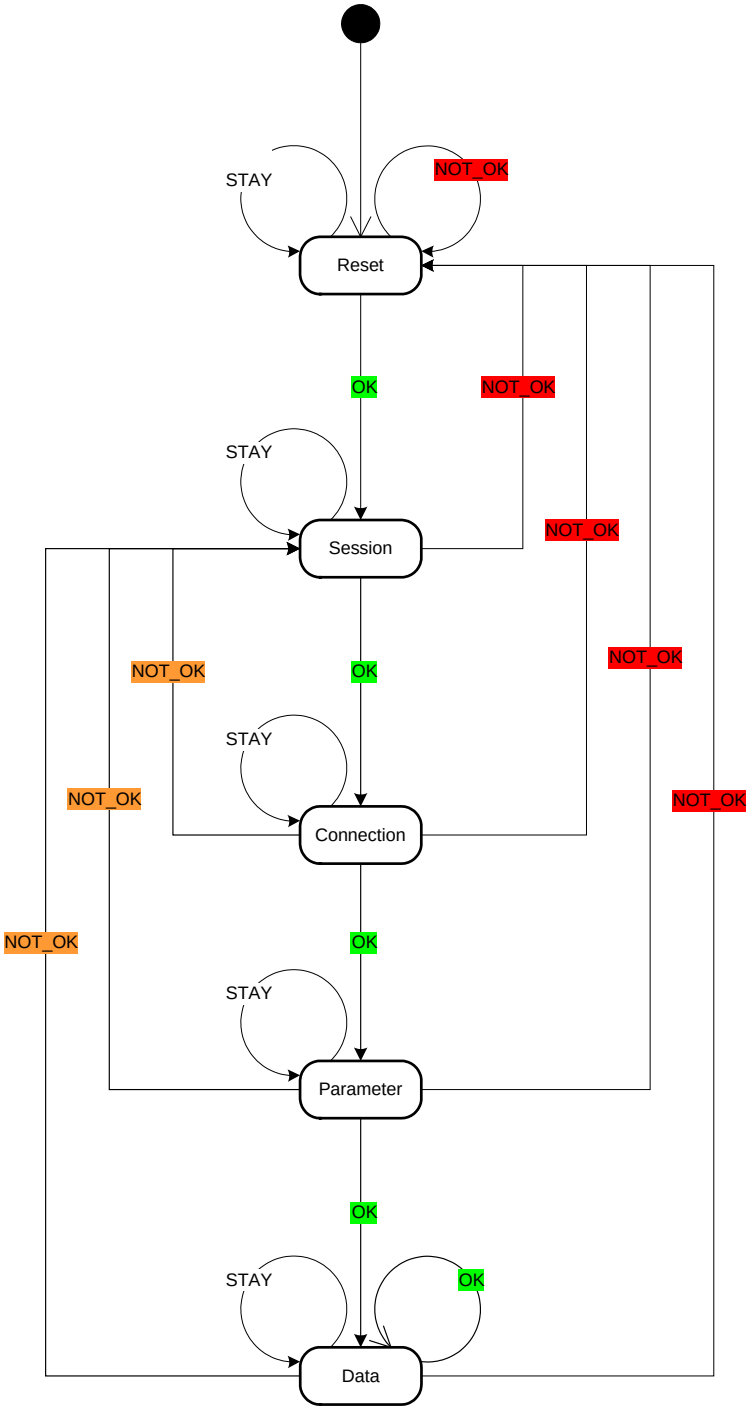


Figure 8 – FSCP 12/1 node states

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]	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 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]	1st octet of SafeOutputs
2	SafeData[1]	2nd 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]	3rd octet of SafeOutputs
6	SafeData[3]	4th 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.

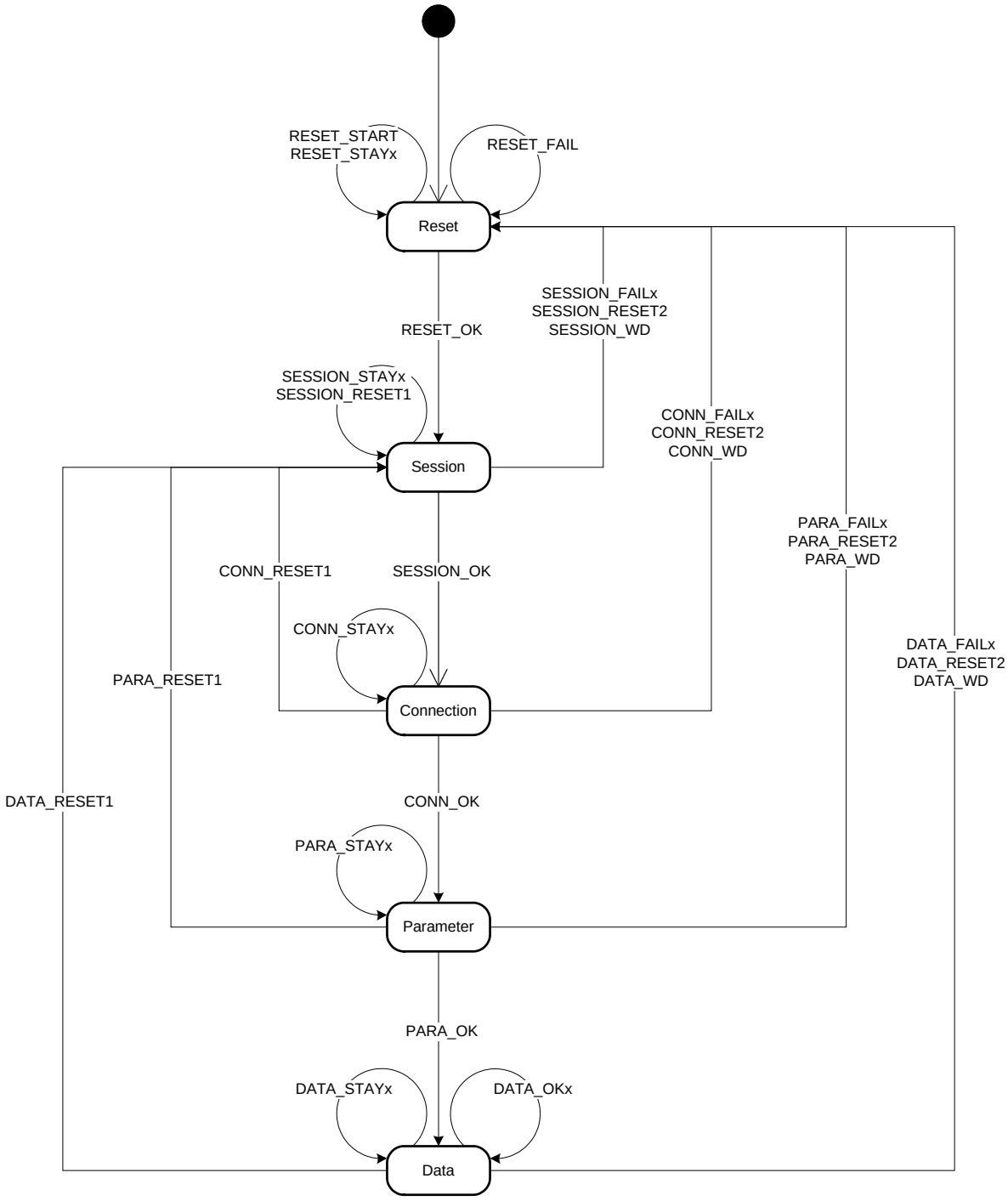


Figure 9 – State diagram for FSoE Master

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.

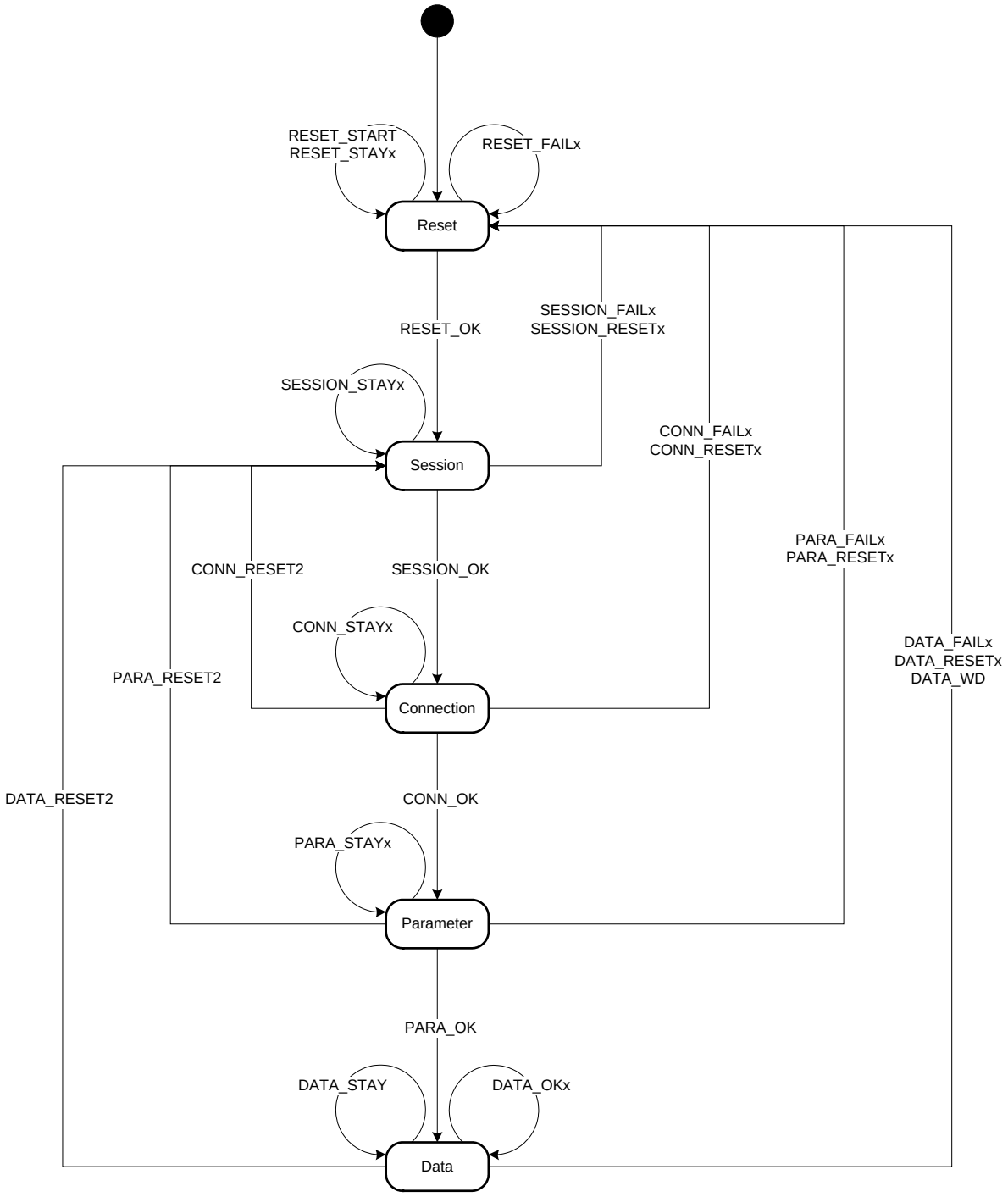


Figure 10 – State diagram for FSoE Slave

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)

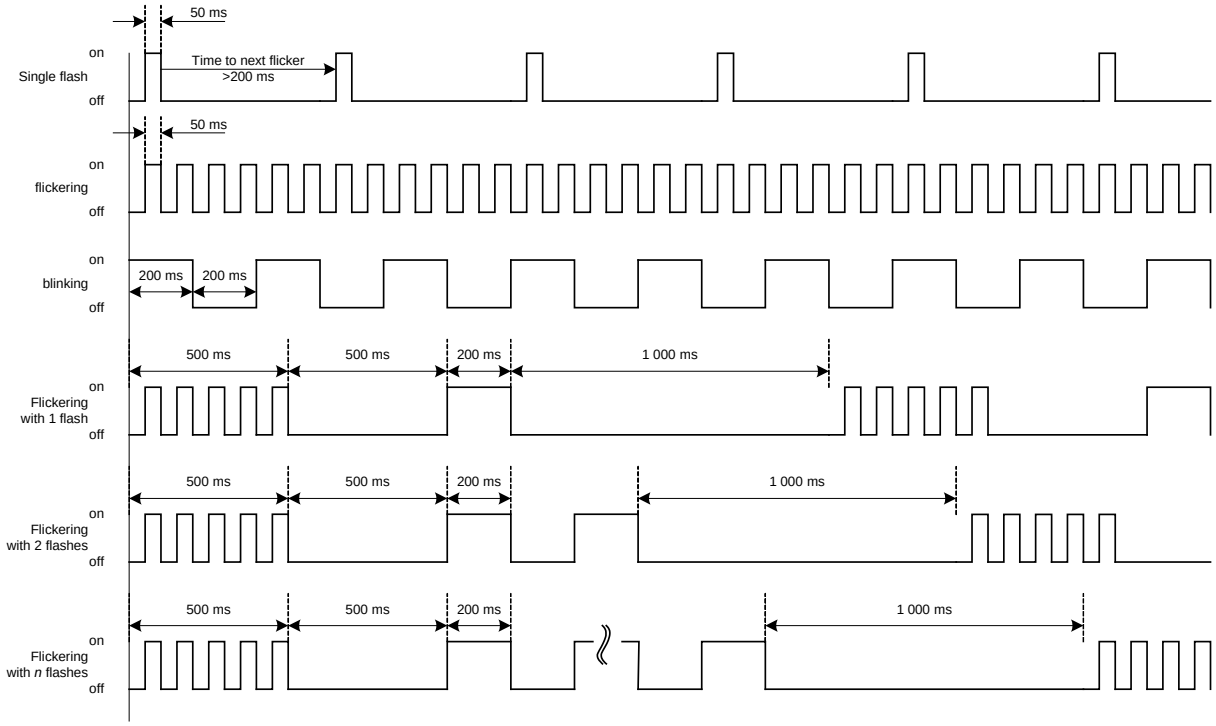


Figure 11 – Indicator flash rates

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.

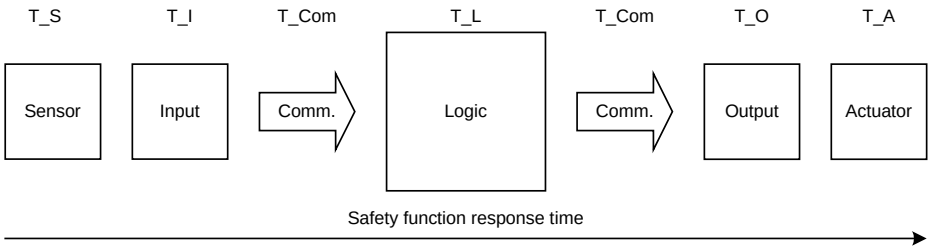


Figure 12 – Components of a safety function

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.

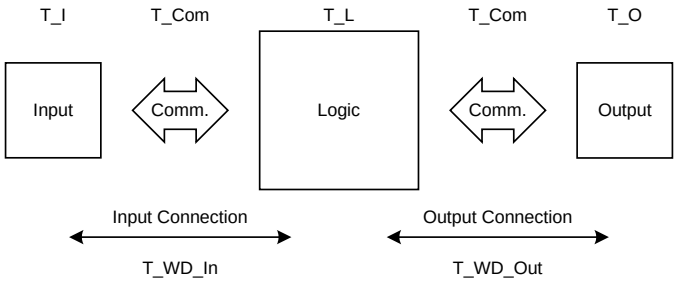


Figure 13 – Calculation of the FSoE Watchdog times for input and output connections

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

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

$$= 2 \times T\_I + 4 \times T\_Com + 2 \times T\_L + \Delta T$$

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

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

$$= 4 \times T\_Com + 2 \times T\_L + 2 \times T\_O + \Delta T$$

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.

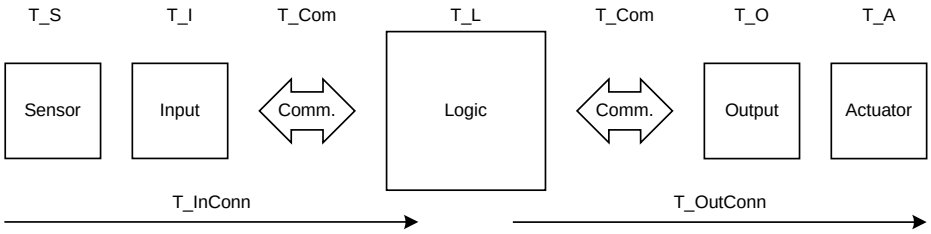


Figure 14 – Calculation of the worst case safety function response time

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

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

$$= 2 \times T\_S + 2 \times T\_I + 2 \times T\_Com + 2 \times T\_L$$

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:

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

$$= 2 \times T\_S + T\_WD\_In$$

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

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

$$= 2 \times T\_L + 2 \times T\_Com + 2 \times T\_O + 2 \times T\_A$$

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:

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

$$= 2 \times T\_L + T\_WD\_In + 2 \times T\_A$$

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 $FCS_{standard}$.

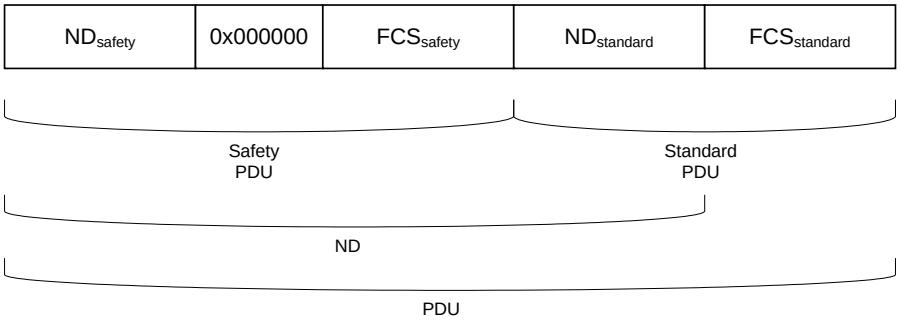


Figure 15 – Safety PDU embedded in standard PDU

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.

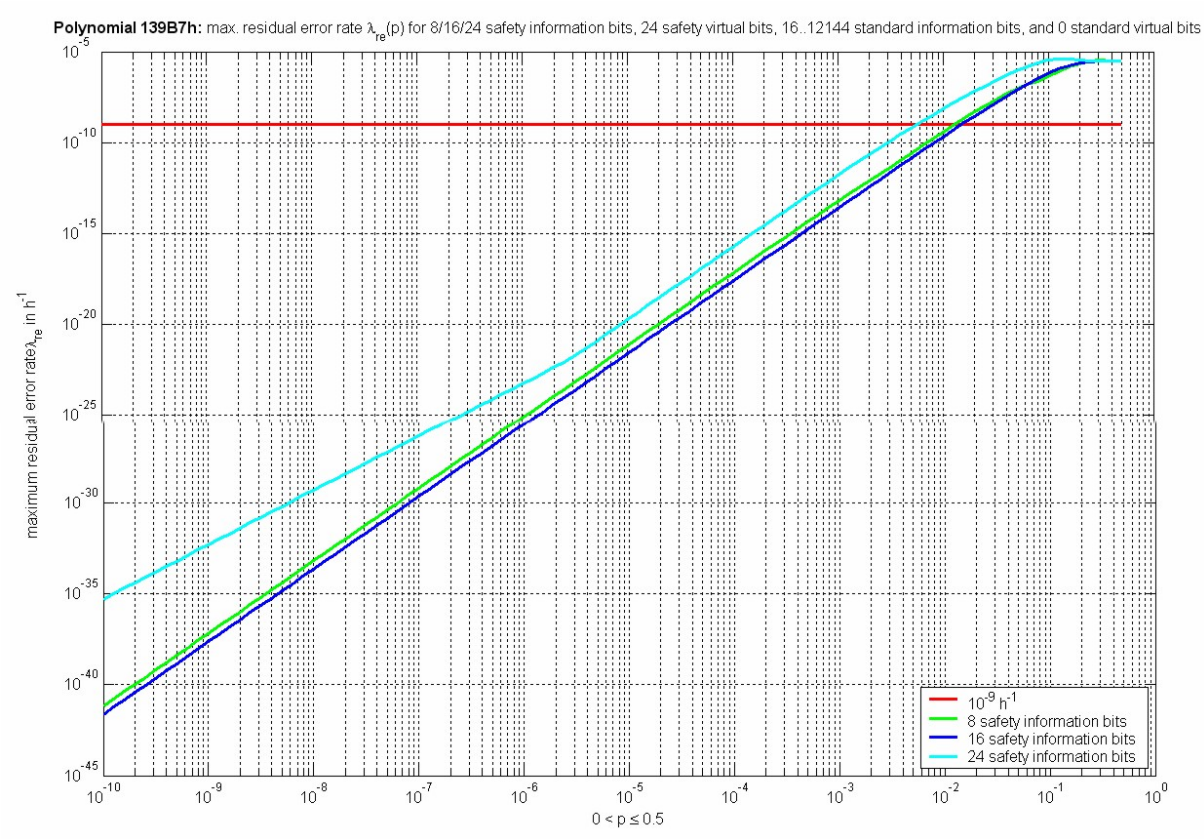


Figure 16 – Residual error rate for 8/16/24 bit safety data and up to 12 144 bit standard data

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*

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

---

10 In preparation.

11 To be published.

12 Proposed new work item under consideration.

[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.

SOMMAIRE

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

_____

RÉSEAUX DE COMMUNICATION INDUSTRIELS – PROFILS –

Partie 3-12: Bus de terrain de sécurité fonctionnelle – Spécifications supplémentaires pour CPF 12

AVANT-PROPOS

1) La Commission Electrotechnique Internationale (CEI) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de la CEI). La CEI a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. A cet effet, la CEI – entre autres activités – publie des Normes internationales, des Spécifications techniques, des Rapports techniques, des Spécifications accessibles au public (PAS) et des Guides (ci-après dénommés "Publication(s) de la CEI"). Leur élaboration est confiée à des comités d'études, aux travaux desquels tout Comité national intéressé par le sujet traité peut participer. Les organisations internationales, gouvernementales et non gouvernementales, en liaison avec la CEI, participent également aux travaux. La CEI collabore étroitement avec l'Organisation Internationale de Normalisation (ISO), selon des conditions fixées par accord entre les deux organisations.
2) Les décisions ou accords officiels de la CEI concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, étant donné que les Comités nationaux de la CEI intéressés sont représentés dans chaque comité d'études.
3) Les Publications de la CEI se présentent sous la forme de recommandations internationales et sont agréées comme telles par les Comités nationaux de la CEI. Tous les efforts raisonnables sont entrepris afin que la CEI s'assure de l'exactitude du contenu technique de ses publications; la CEI ne peut pas être tenue responsable de l'éventuelle mauvaise utilisation ou interprétation qui en est faite par un quelconque utilisateur final.
4) Dans le but d'encourager l'uniformité internationale, les Comités nationaux de la CEI s'engagent, dans toute la mesure possible, à appliquer de façon transparente les Publications de la CEI dans leurs publications nationales et régionales. Toutes divergences entre toutes Publications de la CEI et toutes publications nationales ou régionales correspondantes doivent être indiquées en termes clairs dans ces dernières.
5) La CEI elle-même ne fournit aucune attestation de conformité. Des organismes de certification indépendants fournissent des services d'évaluation de conformité et, dans certains secteurs, accèdent aux marques de conformité de la CEI. La CEI n'est responsable d'aucun des services effectués par les organismes de certification indépendants.
6) Tous les utilisateurs doivent s'assurer qu'ils sont en possession de la dernière édition de cette publication.
7) Aucune responsabilité ne doit être imputée à la CEI, à ses administrateurs, employés, auxiliaires ou mandataires, y compris ses experts particuliers et les membres de ses comités d'études et des Comités nationaux de la CEI, pour tout préjudice causé en cas de dommages corporels et matériels, ou de tout autre dommage de quelque nature que ce soit, directe ou indirecte, ou pour supporter les coûts (y compris les frais de justice) et les dépenses découlant de la publication ou de l'utilisation de cette Publication de la CEI ou de toute autre Publication de la CEI, ou au crédit qui lui est accordé.
8) L'attention est attirée sur les références normatives citées dans cette publication. L'utilisation de publications référencées est obligatoire pour une application correcte de la présente publication.

La Norme internationale CEI 61784-3-12 a été établie par le sous-comité 65C: Réseaux de communication industriels, du comité d'études 65 de la CEI: Mesure, commande et automation dans les processus industriels.

La présente version bilingue (2012-02) correspond à la version anglaise monolingue publiée en 2010-06.

Le texte anglais de cette norme est issu des documents 65C/591A/FDIS et 65C/603/RVD.

Le rapport de vote 65C/603/RVD donne toute information sur le vote ayant abouti à l'approbation de cette norme.

La version française de cette norme n'a pas été soumise au vote.

Cette publication a été rédigée selon les Directives ISO/CEI, Partie 2.

Une liste de toutes les parties de la série CEI 61784-3, présentées sous le titre général *Réseaux de communication industriels – Profils – Bus de terrain de sécurité fonctionnelle*, est disponible sur le site Web de la CEI.

Le comité a décidé que le contenu de cette publication ne sera pas modifié avant la date de stabilité indiquée sur le site web de la CEI sous "http://webstore.iec.ch" dans les données relatives à la publication recherchée. A cette date, la publication sera

- reconduite,
- supprimée,
- remplacée par une édition révisée, ou
- amendée.

IMPORTANT – Le logo "couleur inside" qui se trouve sur la page de couverture de cette publication indique qu'elle contient des couleurs qui sont considérées comme utiles à une bonne compréhension de son contenu. Les utilisateurs devraient, par conséquent, imprimer cette publication en utilisant une imprimante couleur.

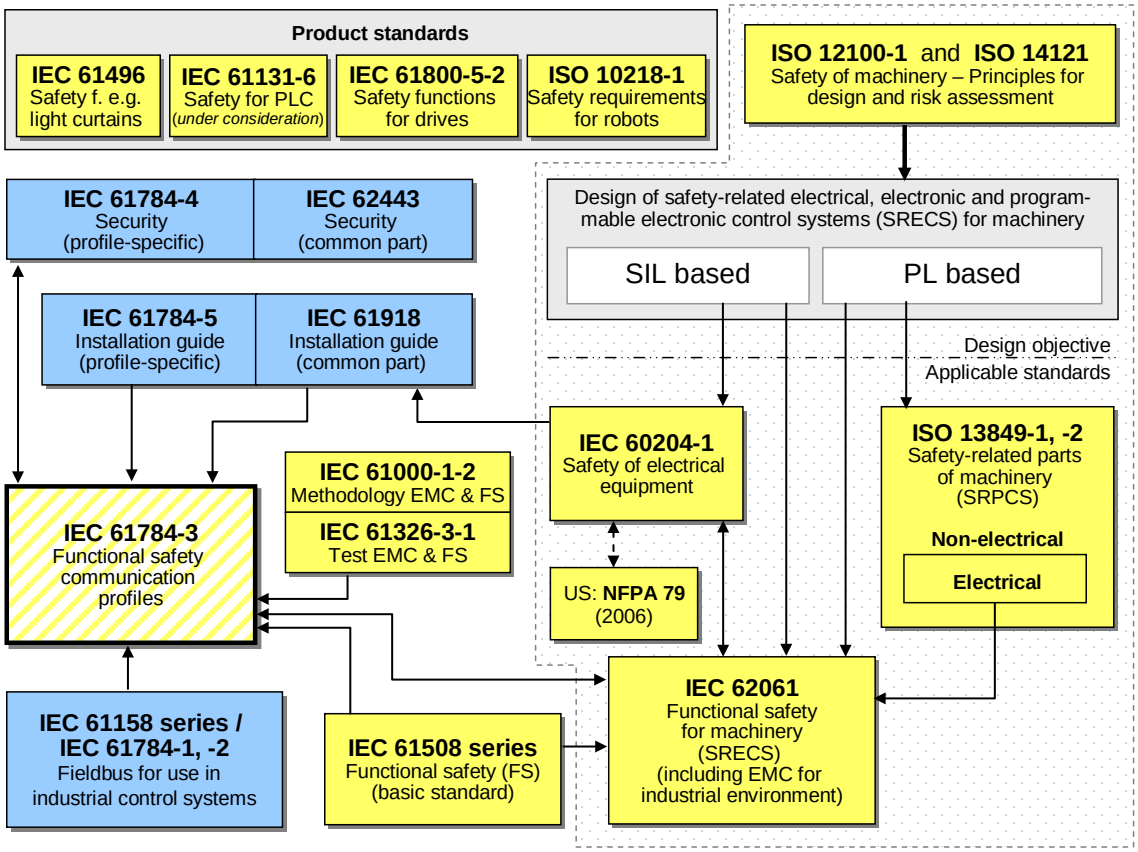
0 Introduction

0.1 Généralités

La norme CEI 61158 relative aux bus de terrain, ainsi que ses normes associées CEI 61784-1 et CEI 61784-2, définit un ensemble de protocoles de communication qui assurent la commande répartie d'applications automatisées. La technologie de bus de terrain est désormais reconnue et bien éprouvée. Ainsi de nombreuses améliorations des bus de terrain se développent pour traiter de domaines non encore normalisés tels que les applications en temps réel relatives à la sécurité et de sûreté.

La présente norme définit les principes pertinents applicables aux communications en termes de sécurité fonctionnelle en référence à la série CEI 61508, et spécifie plusieurs couches de communication de sécurité (profils et protocoles correspondants) basées sur les profils de communication et les couches de protocoles de la CEI 61784-1, la CEI 61784-2 et la série CEI 61158. Elle ne couvre pas les aspects relatifs à la sécurité électrique et à la sécurité intrinsèque.

La Figure 1 illustre les relations entre la présente norme et les normes pertinentes relatives à la sécurité et au bus de terrain dans un environnement machines.



Key

(yellow) safety-related standards
(blue) fieldbus-related standards
(dashed yellow) this standard

Légende

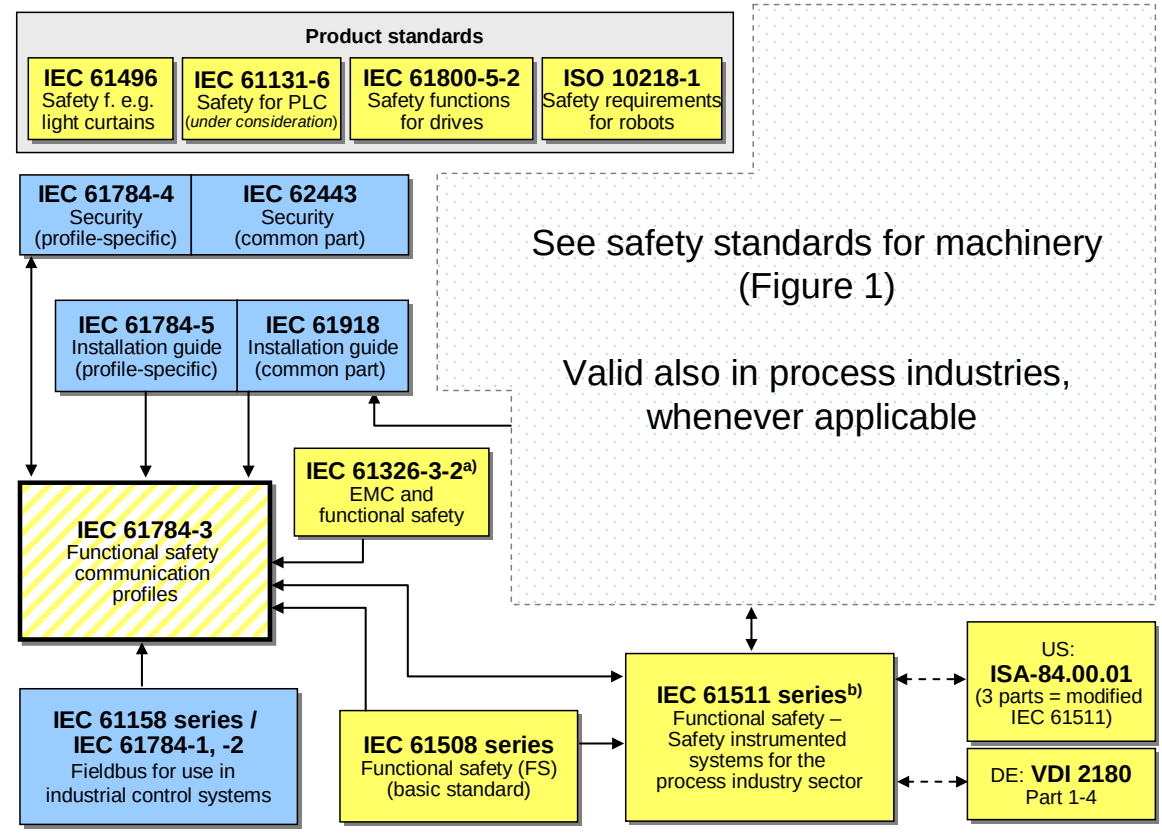
Anglais	Français
Product standards	Normes de produits
Safety function, e.g. light curtains	Fonction de sécurité, par exemple rideaux de lumière
Safety for PLC (under consideration)	Sécurité relative aux automates programmables (à l'étude)
Safety functions for drives	Fonctions de sécurité applicables aux entraînements
Safety requirements for robots	Exigences de sécurité applicables aux robots
Safety of machinery - ... assessment	Sécurité des machines – principes généraux de conception et appréciation du risque
Security (profile-specific)	Sûreté (spécifique au profil)
Security (common part)	Sûreté (partie commune)
Design of safety-related .... for machinery	Conception des systèmes de commande électriques, électroniques et électroniques programmables relatifs à la sécurité pour les machines
SIL based	Basé sur SIL
PL based	Basé sur PL
Installation guide (profile-specific)	Guide d'installation (spécifique au profil)
Installation guide (common part)	Guide d'installation (partie commune)

Anglais	Français
Design objective	Objectif de conception
Applicable standards	Normes applicables
Safety of electrical equipment	Sécurité des équipements électriques
Safety-related parts of machinery	Parties des systèmes de commande relatives à la sécurité
Non-electrical	Non électrique
Electrical	Electrique
Methodology EMC & functional safety	Méthodologie en matière de compatibilité électromagnétique & sécurité fonctionnelle
Test EMC & functional safety	Essai CEM et sécurité fonctionnelle
Functional safety communication profiles	Profils de communication de sécurité fonctionnelle
IEC 61158 series / Fieldbus for use in industrial control systems	Série CEI 61158 / Bus de terrain pour utilisation dans des systèmes de commande industriels
IEC 61508 series, Functional safety (basic standard)	Série CEI 61508 Sécurité fonctionnelle (norme de base)
Functional safety for machinery .... for industrial environment)	Sécurité fonctionnelle des systèmes de commande électriques, électroniques et électroniques programmables (y compris les interférences électromagnétiques dans l'environnement industriel)
Key	Légende
(yellow) safety-related standards	(jaune) normes relatives à la sécurité
(blue) fieldbus-related standards	(bleu) normes relatives au bus de terrain
(dashed) yellow) this standard	(jaune pointillé) la présente norme
IEC	CEI

NOTE Les paragraphes 6.7.6.4 (complexité élevée) et 6.7.8.1.6 (faible complexité) de la CEI 62061 précisent la relation entre le niveau de performance PL (catégorie) et le niveau d'intégrité de sécurité SIL.

Figure 1 – Relations entre la CEI 61784-3 et d'autres normes (machines)

La Figure 2 illustre les relations entre la présente norme et les normes pertinentes relatives à la sécurité et au bus de terrain dans un environnement de transformation.



Key

(yellow) safety-related standards
(blue) fieldbus-related standards
(dashed yellow) this standard

Légende

Anglais	Français
Product standards	Normes de produits
Safety function, e.g. light curtains	Fonction de sécurité, par exemple barrières photo électriques
Safety for PLC (under consideration)	Sécurité relative aux automates programmables (à l'étude)
Safety functions for drives	Fonctions de sécurité applicables aux entraînements
Safety requirements for robots	Exigences de sécurité applicables aux robots
Security (profile-specific)	Sûreté (spécifique au profil)
Security (common part)	Sûreté (partie commune)
Installation guide (profile-specific)	Guide d'installation (spécifique au profil)
Installation guide (common part)	Guide d'installation (partie commune)
See safety standards for machinery (Figure 1)	Voir normes de sécurité pour les machines (Figure 1)
Valid also in process industries, whenever applicable	Valable également dans les industries de transformation, le cas échéant
Functional safety communication profiles	Profils de communication de sécurité fonctionnelle

Anglais	Français
IEC 61326-3-2 a) EMC and functional safety	CEI 61326-3-2 a) CEM & sécurité fonctionnelle
IEC 61158 series/ IEC 61784-1-2, Fieldbus for use in industrial control systems	Série CEI 61158/ CEI 61784-1,-2 Bus de terrain pour utilisation dans des systèmes de commande industriels
IEC 61508 series, Functional safety (basic standard)	Série CEI 61508 Sécurité fonctionnelle (norme de base)
IEC 61511 seriesb) Functional safety–safety instrumented systems for the process industry sector	Série CEI 61511b) sécurité fonctionnelle – systèmes instrumentés de sécurité pour le secteur des industries de transformation
US: ISA 84.00.1 (3 parts = modified IEC 61511)	US: ISA 84.00.1 (3 parties = CEI 61511 modifiée)
DE : VDI 2180 Part 1 –4	DE : VDI 2180 Parties 1 à 4
Key	Légende
(yellow) safety-related standards	(jaune) normes relatives à la sécurité
(blue) fieldbus-related standards	(bleu) normes relatives au bus de terrain
(dashed) yellow) this standard	(jaune pointillé) la présente norme

[a] Pour des environnements électromagnétiques spécifiés; Autrement, la CEI 61326-3-1 s'applique.

[b] EN ratifiée.

Figure 2 – Relations entre la CEI 61784-3 et d'autres normes (transformation)

Les couches de communication de sécurité mises en oeuvre dans le cadre de systèmes relatifs à la sécurité conformément à la série CEI 61508, assurent la confiance nécessaire à accorder à la transmission de messages (information) entre deux participants ou plus sur un bus de terrain dans un système sécuritaire, ou une fiabilité suffisante dans le comportement de sécurité en cas d'erreurs ou de défaillances du bus de terrain.

Les couches de communication de sécurité spécifiées dans la présente norme permettent de garantir cette assurance en utilisant un bus de terrain dans des applications nécessitant une sécurité fonctionnelle jusqu'au niveau d'intégrité de sécurité (SIL) spécifié par son profil de communication de sécurité fonctionnelle correspondant.

La revendication du SIL qui en résulte pour un système dépend de la mise en oeuvre du profil de communication de sécurité fonctionnelle retenu au sein du système – la mise en oeuvre du profil de communication de sécurité fonctionnelle dans un dispositif normal ne suffit pas à le qualifier de dispositif de sécurité.

La présente norme décrit:

— les principes de base de mise en œuvre des exigences de la série CEI 61508 pour les communications de données relatives à la sécurité, y compris les défauts de transmission potentiels, les mesures correctives et les considérations concernant l'intégrité des données;

— la description individuelle des profils de sécurité fonctionnelle pour plusieurs familles de profils de communication dans les CEI 61784-1 et CEI 61784-2;

— les extensions de la couche de sécurité aux sections relatives au service et aux protocoles de communication de la série CEI 61158.

0.2 Déclaration de droits de propriété

La Commission électrotechnique internationale (CEI) attire l'attention sur le fait qu'il est déclaré que la conformité avec le présent document peut impliquer l'utilisation de brevets concernant les profils de communication de sécurité fonctionnelle pour la famille 12 comme suit, où la notation [xx] désigne le détenteur des droits de propriété:

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

La CEI ne prend pas position eu égard à la preuve, la validité et la portée de ces droits de propriété.

Les détenteurs de ces droits de propriété ont donné l'assurance à la CEI qu'ils consentent à négocier des licences avec des demandeurs du monde entier, en des termes et à des conditions raisonnables et non discriminatoires. A ce propos, la déclaration des détenteurs de ces droits de propriété est enregistrée à la CEI.

Des informations peuvent être obtenues auprès de:

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

L'attention est par ailleurs attirée sur le fait que certains des éléments du présent document peuvent faire l'objet de droits de propriété intellectuelle ou de droits analogues autres que ceux identifiés ci-dessus. La CEI ne saurait être tenue pour responsable de ne pas avoir identifié de tels droits de propriété et de ne pas avoir signalé leur existence.

RÉSEAUX DE COMMUNICATION INDUSTRIELS – PROFILS –

Partie 3-12: Bus de terrain de sécurité fonctionnelle – Spécifications supplémentaires pour CPF 12

1 Domaine d'application

La présente partie de la série CEI 61784-3 spécifie une couche de communication sécuritaire (services et protocole) fondée sur la CPF 12 de la CEI 61784-2 et le type 12 de la CEI 61158. Elle identifie les principes applicables aux communications de sécurité fonctionnelle définies dans la CEI 61784-3, et appropriés à cette couche de communication de sécurité.

NOTE 1 Elle ne couvre pas les aspects relatifs à la sécurité électrique et à la sécurité intrinsèque. La sécurité électrique concerne les dangers tels que les chocs électriques. La sécurité intrinsèque concerne les dangers associés aux atmosphères explosibles.

La présente partie [1] définit les mécanismes de transmission des messages propres à la sécurité entre les participants d'un réseau réparti, en utilisant la technologie de bus de terrain conformément aux exigences de la série CEI 61508 [2] concernant la sécurité fonctionnelle. Ces mécanismes peuvent être utilisés dans diverses applications industrielles, telles que la commande de processus, l'usinage automatique et les machines.

La présente partie fournit des lignes directrices tant pour les développeurs que pour les évaluateurs de dispositifs et systèmes conformes.

NOTE 2 La revendication du SIL qui en résulte pour un système dépend de la mise en oeuvre du profil de communication de sécurité fonctionnelle retenu au sein du système – la mise en oeuvre du profil de communication de sécurité fonctionnelle, conforme à la présente partie, dans un dispositif normal ne suffit pas à le qualifier de dispositif de sécurité.

2 Références normatives

Les documents de référence suivants sont indispensables pour l'application du présent document. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

CEI 60204-1, *Sécurité des machines – Equipement électrique des machines – Partie 1: Règles générales*

CEI 61000-6-2, *Compatibilité électromagnétique (CEM) – Partie* 6-2: *Normes génériques – Immunité pour les environnements industriels*

IEC 61131-2, *Programmable controllers – Part 2: Equipment requirements and tests* (disponible uniquement en anglais)

IEC 61158-2, *Industrial communication networks – Fieldbus specifications – Part 2: Physical layer specification and service definition* (disponible uniquement en anglais)[3]

---

1 Dans les pages suivantes de la présente norme, "la présente partie" se substitue à "cette partie de la série CEI 61784-3".

2 Dans les pages suivantes de la présente norme, "CEI 61508" se substitue à "série CEI 61508".

3 Les publications monolingues des séries IEC 61158 et IEC 61784 sont actuellement en cours de traduction.

IEC 61158-3-12, *Industrial communication networks – Fieldbus specifications – Part 3-12: Data-link layer service definition – Type 12 elements* (disponible uniquement en anglais)

IEC 61158-4-12, *Industrial communication networks – Fieldbus specifications – Part 4-12: Data-link layer protocol specification – Type 12 elements* (disponible uniquement en anglais)

IEC 61158-5-12, *Industrial communication networks – Fieldbus specifications – Part 5-12: Application layer service definition – Type 12 elements* (disponible uniquement en anglais)

IEC 61158-6-12, *Industrial communication networks – Fieldbus specifications – Part 6-12: Application layer protocol specification – Type 12 elements* (disponible uniquement en anglais)

CEI 61326-3-1, *Matériel électrique de mesure, de commande et de laboratoire – Exigences relatives à la CEM – Partie 3-1: Exigences d'immunité pour les systèmes relatifs à la sécurité et pour les matériels destinés à réaliser des fonctions relatives à la sécurité (sécurité fonctionnelle) – Applications industrielles générales*

CEI 61326-3-2, *Matériel électrique de mesure, de commande et de laboratoire – Exigences relatives à la CEM – Partie 3-2: Exigences d'immunité pour les systèmes relatifs à la sécurité et pour les matériels destinés à réaliser des fonctions relatives à la sécurité (sécurité fonctionnelle) – Applications industrielles dont l'environnement électromagnétique est spécifié*

CEI 61508 (toutes parties), *Sécurité fonctionnelle des systèmes électriques / électroniques / électroniques programmables relatifs à la sécurité*

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

IEC 61784-3:2010[4], *Industrial communication networks – Profiles – Part 3: Functional safety fieldbuses – General rules and profile definitions* (disponible uniquement en anglais)

IEC 61918, *Industrial communication networks – Installation of communication networks in industrial premises* (disponible uniquement en anglais)

3 Termes, définitions, symboles, abréviations et conventions

3.1 Termes et définitions

Pour les besoins du présent document, les termes et définitions suivants s'appliquent.

3.1.1 Termes et définitions communs

3.1.1.1
disponibilité
probabilité, pour un système automatisé, qu'il ne se produise pas de conditions opérationnelles non satisfaisantes, telles que la perte de production, pendant une période donnée

3.1.1.2
canal noir
canal de communication sans preuve existante de conception ou de validation conformément à la CEI 61508

---

4 En cours d'élaboration.

3.1.1.3
canal de communication
connexion logique entre deux points limites d'un *système de communication*

3.1.1.4
système de communication
disposition de matériels, logiciels et vecteurs de propagation destinée à permettre la transmission de *messages* (couche d'application définie dans l'ISO/CEI 7498) d'une application à une autre

3.1.1.5
connexion
liaison logique entre deux objets d'application de dispositifs identiques ou différents

3.1.1.6
contrôle de redondance cyclique (CRC[5])
<valeur> donnée redondante déduite, et enregistrée ou transmise simultanément, d'un bloc de données afin de détecter toute corruption des données

<méthode> procédure utilisée pour calculer les données redondantes

NOTE 1 Les termes « code CRC » et « signature CRC », et les étiquettes telles que CRC1, CRC2, peuvent également être utilisés dans la présente norme pour se référer aux données redondantes.

NOTE 2 Voir également [34], [35]][6].

3.1.1.7
erreur
écart ou discordance entre une valeur ou une condition calculée, observée ou mesurée, et la valeur ou la condition vraie, prescrite ou théoriquement correcte

[CEI 61508-4:2010[7]], [CEI 61158]

NOTE 1 Les erreurs peuvent être dues à des erreurs de conception du matériel/logiciel et/ou des informations altérées du fait de perturbations électromagnétiques et/ou autres effets.

NOTE 2 Les erreurs ne produisent nécessairement pas une *défaillance* ou une *panne*.

3.1.1.8
défaillance
cessation de l'aptitude d'une unité fonctionnelle à accomplir une fonction requise ou fonctionnement d'une unité fonctionnelle d'une toute autre manière que celle requise

NOTE 1 La définition de la CEI 61508-4 est identique avec des notes complémentaires.

[CEI 61508-4:2010, modifiée], [ISO/CEI 2382-14.01.11, modifiée]

NOTE 2 Une défaillance peut être due à une *erreur* (par exemple, problème de conception matérielle/logicielle ou rupture de message).

3.1.1.9
panne
condition anormale susceptible de provoquer la réduction, ou la perte, de capacité d'une unité fonctionnelle à accomplir une fonction requise

---

5 CRC = *Cyclic Redundancy Check*

6 Les chiffres entre crochets font référence à la bibliographie.

7 A publier.

NOTE Le VEI 191-05-01 définit la « panne » comme un état caractérisé par l'incapacité à accomplir une fonction requise, à l'exclusion de l'incapacité au cours de la période de maintenance préventive ou autres actions planifiées, ou du fait de l'absence de ressources externes.

[CEI 61508-4:2010, modifiée], [ISO/CEI 2382-14.01.10, modifiée]

3.1.1.10
bus de terrain
système de communication reposant sur le transfert de données en série et utilisé dans des applications d'automatisation industrielle ou de commande de processus

3.1.1.11
système de bus de terrain
système utilisant un *bus de terrain* avec des dispositifs connectés

3.1.1.12
trame
synonyme discrédité de DLPDU

3.1.1.13
séquence de contrôle de trame (FCS[8])
données redondantes issues d'un bloc de données d'une DLPDU (trame), utilisant une fonction de hachage, et enregistrées ou transmises avec le bloc de données, afin de déterminer l'altération des données

NOTE 1 Il est possible de calculer une FCS à l'aide, par exemple, d'un CRC ou d'une autre fonction de hachage.

NOTE 2 Voir également [34], [35].

3.1.1.14
fonction de hachage
fonction (mathématique) de mise en correspondance des valeurs d'un ensemble (éventuellement) très grand de valeurs en une plage de valeurs (habituellement) plus petite

NOTE 1 Les fonctions de hachage peuvent être utilisées pour déterminer l'altération de données.

NOTE 2 Les fonctions de hachage courantes incluent la parité, la somme de contrôle ou le CRC.

[CEI/TR 62210, modifiée]

3.1.1.15
danger
état ou ensemble de conditions d'un système qui, avec d'autres conditions associées, entraîne inévitablement un préjudice pour les personnes, les biens ou l'environnement

3.1.1.16
maître
entité de communication active capable le lancer et de planifier les activités de communication d'autres stations qui peuvent être des maîtres ou des esclaves

3.1.1.17
message
série ordonnée d'octets destinée à communiquer des informations
[ISO/CEI 2382-16.02.01, modifiée]

8 FCS = *Frame Check Sequence*

3.1.1.18
niveau de performance (PL[9])
niveau discret utilisé pour spécifier la capacité des parties relatives à la sécurité des systèmes de commande à accomplir une fonction de sécurité dans des conditions prévisibles [ISO 13849-1]

3.1.1.19
très basse tension de protection (TBTP ou PELV)
circuit électrique dans lequel la tension ne peut pas dépasser en c.a. 30 V efficace, 42,4 V crête ou en c.c. 60 V, dans des conditions normales et dans des conditions de défaut simple, à l'exception des défauts à la terre dans d'autres circuits électriques

NOTE Un circuit TBTP est similaire à un circuit TBTS qui est relié à la terre de protection.

[CEI 61131-2]

3.1.1.20
redondance
existence de moyens, outre les moyens qui se révéleraient suffisants pour qu'une unité fonctionnelle accomplisse une fonction requise ou que des données représentent une information

NOTE La définition de la CEI 61508-4 est identique, avec des exemples et des notes supplémentaires.

[CEI 61508-4:2010, modifiée], [ISO/CEI 2382-14.01.12, modifiée]

3.1.1.21
fiabilité
probabilité qu'un système automatisé puisse accomplir une fonction requise dans des conditions données pendant un intervalle de temps donné (t1,t2)

NOTE 1 On suppose, en général, que le système automatisé est en état d'accomplir la fonction requise au début de l'intervalle.

NOTE 2 Le terme «fiabilité» est aussi employé en français pour désigner la performance de fiabilité quantifiée par cette probabilité.

NOTE 3 Au cours de la période MTBF (temps moyen entre défaillances) ou MTTF (durée moyenne de fonctionnement avant défaillance), il y a réduction de la probabilité qu'un système automatisé puisse accomplir une fonction requise dans des conditions données.

NOTE 4 La fiabilité diffère de la disponibilité.

[CEI 62059-11, modifiée]

3.1.1.22
risque
combinaison de la probabilité d'occurrence d'un dommage ou préjudice et de la gravité de ce dernier

NOTE Pour plus d'informations sur ce concept, se reporter à l'Annexe A de la CEI 61508-5:2010[10].

[CEI 61508-4:2010], [ISO/CEI Guide 51:1999, définition 3.2]

3.1.1.23
couche de communication de sécurité (SCL[11])
couche de communication qui comprend toutes les mesures nécessaires permettant d'assurer la transmission de données en toute sécurité conformément aux exigences de la CEI 61508

---

9 PL = *Performance Level*

10 A publier.

11 SCL = *Safety Communication Layer*

3.1.1.24
données de sécurité
données transmises par un réseau de sécurité utilisant un protocole de sécurité

NOTE La couche de communication de sécurité ne garantit pas la sécurité des données proprement dites, mais uniquement la transmission en toute sécurité de ces dernières.

3.1.1.25
dispositif de sécurité
dispositif conçu conformément à la CEI 61508 et qui met en œuvre le profil de communication de sécurité fonctionnelle

3.1.1.26
très basse tension de sécurité (TBTS ou SELV)
circuit électrique dont la tension ne peut pas dépasser en c.a. 30 V efficace, 42,4 V crête ou en c.c. 60 V, dans des conditions normales et dans des conditions de défaut simple, y compris les défauts à la terre dans d'autres circuits électriques

NOTE Un circuit TBTS n'est pas relié à la terre de protection.

[CEI 61131-2]

3.1.1.27
fonction de sécurité
fonction à réaliser par un système E/E/PE relatif à la sécurité ou par un dispositif externe de réduction de risque, prévue pour assurer ou maintenir un état de sécurité de l'EUC par rapport à un événement dangereux spécifique

NOTE La définition donnée dans la CEI 61508-4 est identique avec ajout d'un exemple et d'une référence.

[CEI 61508-4:2210, modifiée]

3.1.1.28
temps de réponse de la fonction de sécurité
temps écoulé du cas le plus défavorable suite à l'activation d'un capteur de sécurité relié à un bus de terrain, avant que ne soit atteint l'état de sécurité correspondant de son (ses) actionneur(s) de sécurité, du fait d'erreurs ou de défaillances avérées dans le canal de fonction de sécurité

NOTE Ce concept est introduit dans 5.2.4 de la CEI 61784-3:2010[12] et traité dans le cadre des profils de communication de sécurité fonctionnelle définis dans la présente partie.

3.1.1.29
niveau d'intégrité de sécurité (SIL[13])
niveau discret (un sur quatre niveaux possibles), correspondant à une plage de valeurs d'intégrité de sécurité, où le niveau d'intégrité de sécurité 4 est le niveau le plus élevé et le niveau d'intégrité de sécurité 1 est le niveau le plus faible

NOTE 1 Les mesures cible des défaillances (voir 3.5.17 dans la CEI 61508-4:2010) applicables aux quatre niveaux d'intégrité de sécurité sont spécifiées dans les Tableaux 2 et 3 de la CEI 61508-1:2010[14].

NOTE 2 Les niveaux d'intégrité de sécurité sont utilisés pour spécifier les exigences d'intégrité de sécurité des fonctions équivalentes à attribuer aux systèmes E/E/PE relatifs à la sécurité.

NOTE 3 Le niveau d'intégrité de sécurité (SIL) n'est pas une propriété d'un système, sous-système, élément ou composant. L'interprétation correcte de l'expression « système sécuritaire avec SILn » (où n est égal à 1, 2, 3 ou 4) signifie que le système est potentiellement capable de prendre en charge des fonctions de sécurité avec un niveau d'intégrité de sécurité jusqu'à n.

---

12 En cours d'élaboration.

13 SIL = *Safety Integrity Level*

14 A publier.

[CEI 61508-4:2010]

3.1.1.30
mesure de sécurité
<la présente norme> mesure permettant de contrôler les *erreurs* de communication éventuelles, qui est conçue et mise en œuvre conformément aux exigences de la CEI 61508

NOTE 1 Dans la pratique, plusieurs mesures de sécurité sont combinées pour atteindre le niveau d'intégrité de sécurité requis.

NOTE 2 Les *erreurs* de communication et les mesures de sécurité associées sont détaillées en 5.3 et 5.4 de la CEI 61784-3:2010.

3.1.1.31
application relative à la sécurité
programmes conçus conformément à la CEI 61508 pour satisfaire aux exigences SIL de l'application

3.1.1.32
système relatif à la sécurité
système qui exécute des *fonctions de sécurité* conformément à la CEI 61508

3.1.1.33
esclave
entité de communication passive capable de recevoir des messages et de les envoyer en réponse à une autre entité de communication qui peut être un maître ou un esclave

3.1.2 CPF 12: Termes et définitions supplémentaires

3.1.2.1
données de sécurité intégrée
caractérisation de données qui sont mises à une valeur pré-définie en cas d'initialisation ou d'erreur

NOTE Dans la présente partie, il convient que la valeur des données de sécurité intégrée soit toujours mise à "0".

3.1.2.2
connexion FSoE
relation unique entre le Maître FSoE et un Esclave FSoE

3.1.2.3
cycle FSoE
cycle de transmission avec une PDU de Maître de Sécurité et la PDU d'Esclave de Sécurité correspondante

3.1.2.4
SafeInput (entrée sûre)
données de processus de sécurité transmises de l'Esclave FSoE au Maître FSoE

3.1.2.5
SafeOutput (sortie sûre)
données de processus de sécurité transmises du Maître FSoE à l'Esclave FSoE

3.1.2.6
PDU de Maître de Sécurité
PDU de sécurité transmise du Maître FSoE à l'Esclave FSoE

3.1.2.7
PDU d'Esclave de Sécurité
PDU de sécurité transmise de l'Esclave FSoE au Maître FSoE

3.2 Symboles et abréviations

3.2.1 Symboles et abréviations communs

CP	Profil de communication (*Communication Profile*)	[CEI 61784-1]
CPF	Famille de profils de communication (*Communication Profile Family*)	[CEI 61784-1]
CRC	Contrôle de redondance cyclique (*Cyclic Redundancy Check*)	
DLL	Couche Liaison de données (*Data Link Layer*)	[ISO/CEI 7498-1]
DLPDU	Unité de données de protocole de liaison de données (*Data Link Protocol Data Unit*)	
CEM	Compatibilité électromagnétique	
EUC	Equipement commandé (*Equipment Under Control*)	[CEI 61508-4:2010]
E/E/PE	Electrique/électronique/électronique programmable (*Electrical/Electronic/Programmable Electronic*)	[CEI 61508-4:2010]
FAL	Couche Application de bus de terrain (*Fieldbus Application Layer*)	[CEI 61158-5]
FCS	Séquence de contrôle de trame (*Frame Check Sequence*)	
FS	Sécurité fonctionnelle (*Functional Safety*)	
FSCP	Profil de communication de sécurité fonctionnelle (*Functional Safety Communication Profile*)	
MTBF	temps moyen entre défaillances (*Mean Time Between Failures*)	
MTTF	durée moyenne de fonctionnement avant défaillance (*Mean Time To Failure*)	
PDU	Unité de données de protocole (*Protocol Data Unit*)	[ISO/CEI 7498-1]
TBTP ou PELV	Très Basse Tension de Protection	
PhL	Couche Physique (*Physical Layer*)	[ISO/CEI 7498-1]
PL	Niveau de performance (*Performance Level*)	[ISO 13849-1]
PLC	Automate programmable (*Programmable Logic Controller*)	
SCL	Couche de communication de sécurité (*Safety Communication Layer*)	
TBTS ou SELV	Très Basse Tension de Sécurité	
SIL	Niveau d'intégrité de sécurité (*Safety Integrity Level*)	[CEI 61508-4:2010]

3.2.2 CPF 12: Symboles et abréviations supplémentaires

ASIC	circuit intégré d'application spécifique (*Application Specific Integrated Circuit*)	
FSoE	Communication de sécurité intégrée sur CPF 12 (*Failsafe over CPF 12*)	
ID	Identifiant	
UML	Langage de modélisation unifié (*Unified Modeling Language*)	[ISO/CEI 19501]

3.3 Conventions

Les conventions utilisées pour décrire les objets, services et protocoles sont définies dans les normes CEI 61158-3-12, CEI 61158-4-12, CEI 61158-5-12 et CEI 61118-6-12.

La présente partie utilise, le cas échéant, des organigrammes et des diagrammes séquentiels en UML pour décrire certaines notions.

Dans les diagrammes d'état, les états sont représentés par des cases et les transitions d'états par des flèches. Les désignations des états et transitions du diagramme d'état correspondent à ceux de la liste textuelle des transitions d'états.

Cette liste de transitions d'états est structurée de la manière suivante (voir également le Tableau 1):

La première rangée donne la dénomination de la transition. La deuxième rangée indique la condition de réalisation de la transition. La troisième rangée définit la ou les actions qui doivent avoir lieu. La dernière rangée donne l'état suivant.

Tableau 1 – Eléments descriptifs d'un diagramme d'état

Transition	Condition	Action	Etat suivant

Chaque état et ses transitions sont décrits dans un paragraphe séparé. Il est inséré un paragraphe séparé pour chaque événement qui peut avoir lieu dans un état donné.

4 Vue d'ensemble du FSCP 12/1 (Safety-over-EtherCAT™)

La famille de profils de communication 12 (communément appelée EtherCAT™[15]) définit des profils de communication sur la base des normes CEI 61158-2 Type 12, CEI 61158-3-12, CEI 61158-4-12, CEI 61158-5-12 et CEI 61158-6-12.

Le ou les profils de base CP 12/1 et CP 12/2 sont définis dans la CEI 61784--2. Le profil de communication de sécurité fonctionnelle FSCP 12/1 (Safety-over-EtherCAT™[15]) repose sur les profils de base de la famille CPF 12 spécifiés dans la CEI 61784-2 et sur les spécifications de la couche de communication de sécurité définies dans la présente partie.

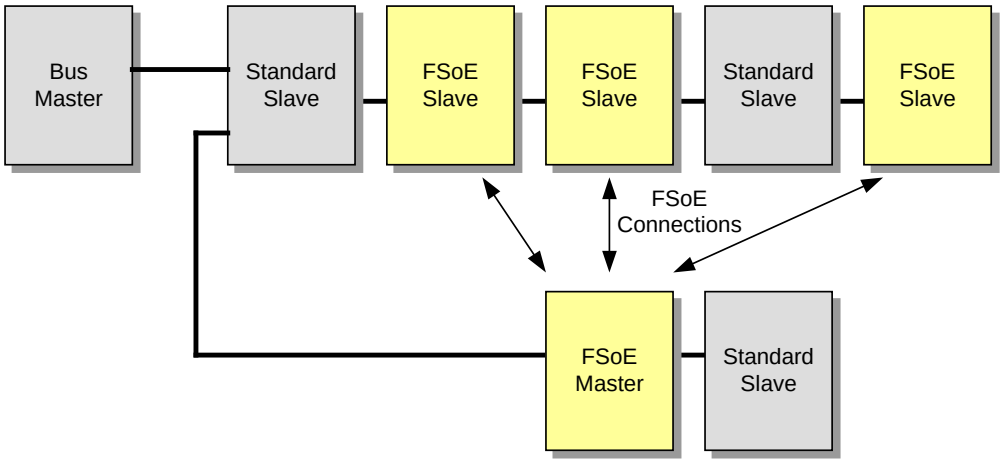
Le profil FSCP 12/1 décrit un protocole de sécurité qui permet le transfert de données de sécurité jusqu'au niveau SIL 3 entre des dispositifs de type FSCP 12/1. Les PDU de sécurité sont transmises par un bus de terrain subordonné qui n'est pas pris en compte en termes de sécurité, car il peut être considéré comme un canal noir. Les PDU de sécurité échangées entre deux partenaires de communication sont considérées par le bus de terrain subordonné comme des données de processus échangées de manière cyclique.

Le FSCP 12/1 utilise une relation maître/esclave unique entre le Maître FSoE et un Esclave FSoE; Cette relation est appelée Connexion FSoE (Figure 3). Dans la Connexion FSoE, chaque dispositif renvoie uniquement son propre nouveau message après qu'un nouveau message ait été reçu du dispositif partenaire. L'ensemble du trajet de transfert entre Maître FSoE et Esclave FSoE est surveillé, au cours de chaque Cycle FSoE, par un chien de garde séparé présent sur les deux dispositifs.

Le Maître FSoE peut traiter plusieurs Connexions FSoE de manière à prendre en charge plusieurs Esclaves FSoE.

---

15 EtherCAT™ et Safety-over-EtherCAT™ désignent les appellations commerciales de Beckhoff, Verl. Cette information est donnée à l'intention des utilisateurs de la présente Norme internationale et ne signifient nullement que la CEI approuve ou recommande le détenteur de la marque ou de l'un quelconque de ses produits. La conformité à la présente norme ne nécessite pas l'utilisation des marques commerciales EtherCAT™ ou Safety-over-EtherCAT™. L'utilisation des marques commerciales EtherCAT™ ou Safety-over-EtherCAT™ nécessite l'autorisation de Beckhoff, Verl.



Légende

Anglais	Français
Bus master	Maître bus
Standard slave	Esclave normal
FsoE slave	Esclave FsoE
FsoE connections	Connexions FsoE
FsoE master	Maître FSoE

Figure 3 – Système FSCP 12/1 de base

L'intégrité des transferts de données de sécurité est assurée comme suit:

— un numéro de session permet de détecter la mise en mémoire-tampon d'une séquence de démarrage complète;

— un numéro de séquence permet de détecter l'échange, la répétition, l'insertion ou la perte de messages entiers;

— une identification unique de connexion permet de détecter en toute sécurité des messages mal acheminés grâce à une relation d'adresses unique;

— un chien de garde permet de détecter en toute sécurité les retards non autorisés sur le trajet de communication

— un contrôle de redondance cyclique permet d'assurer l'intégrité des données et de détecter une éventuelle altération des messages entre l'origine et la destination.

Les transitions d'états sont lancées par le Maître FSoE et acquittées par l'Esclave FSoE. Le diagramme d'état FSoE implique également l'échange et le contrôle d'informations dans la relation de communication.

5 Généralités

5.1 Document externe de spécifications applicables au profil

Le document suivant est utile pour bien comprendre la conception du protocole FSCP 12/1:

- GS-ET-26 [33]

5.2 Exigences fonctionnelles de sécurité

Les exigences suivantes doivent s'appliquer au développement de dispositifs qui mettent en œuvre le protocole FSCP 12/1. Les mêmes exigences ont été appliquées au développement de ce protocole.

- Le protocole FSCP 12/1 est conçu de manière à prendre en charge le niveau d'intégrité de sécurité 3 (SIL 3) (voir la CEI 61508).
- Les mises en œuvre du FSCP 12/1 doivent être conformes à la CEI 61508.
- Les exigences de base pour le développement du protocole FSCP 12/1 sont définies dans la CEI 61784-3.
- Le protocole FSCP 12/1 est mis en œuvre en appliquant la méthode du canal noir; aucune dépendance sécuritaire ne s'applique aux profils de communication CPF 12 normaux. Le matériel de transmission (tels que les appareils de commande, les ASIC, les liaisons, les coupleurs, etc.) doit rester inchangé.
- Les conditions relatives à l'environnement doivent être conformes aux exigences généralement applicables à l'automatisation et notamment la CEI 61326-3-1 en ce qui concerne les essais de marge de sécurité, à moins qu'il n'existe des normes de produits spécifiques.
- Les communications de sécurité et les communications non sécuritaires doivent être indépendantes. Les dispositifs sécuritaires et non sécuritaires doivent cependant être capables d'utiliser le même canal de communication.
- La mise en oeuvre du protocole FSCP 12/1 doit être limitée aux dispositifs terminaux de communication (Maître FSoE et Esclave FSoE).
- Une relation de communication 1:1 doit toujours exister entre un Esclave FSoE et son Maître FSoE.
- La communication de sécurité ne doit pas limiter la durée de cycle minimale du système de communication.

5.3 Mesures de sécurité

Les mesures de sécurité appliquées dans le protocole FSCP 12/1 pour détecter les erreurs de communication sont énumérées dans le Tableau 2. Les mesures de sécurité doivent être traitées et surveillées au sein de chaque dispositif de sécurité.

Tableau 2 – Erreurs de communication et mesures de détection

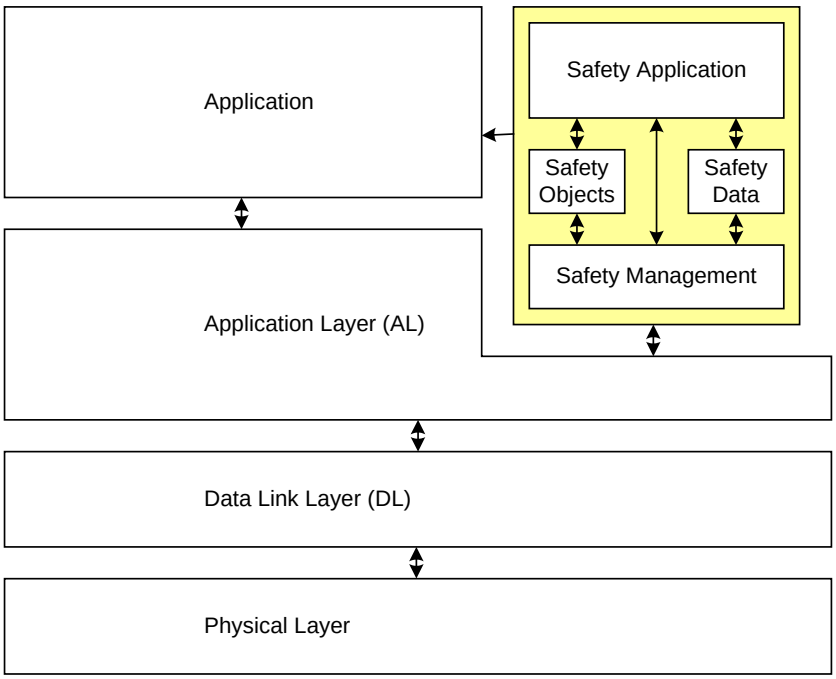
Erreurs de communication	Mesures de sécurité				
	Numéro de séquence (voir 7.1.3.4)	Délai d'attente (voir 6.2) [a]	Authentification de connexion (voir 7.2.2.4) [b]	Message de retour d'informations (voir 7.2.1)	Assurance d'intégrité des données (voir 7.1.3)
Corruption					X
Répétition non prévue	X				X
Séquence incorrecte	X				X
Perte	X	X		X	X
Retard inacceptable		X		X	X
Insertion	X				X
Déguisement		X		X	X
Adressage			X		
Défaillances de mémoire tournante des commutateurs	X				X

[a] Dans la présente norme, l'instance est appelée "Chien de garde FSoE".

[b] Dans la présente norme, l'instance est appelée "ID de Connexion FSoE ".

5.4 Structure de la couche de communication de sécurité

Le protocole FSCP 12/1 est placé en couche au-dessus du protocole de réseau normalisé. La Figure 4 illustre la relation entre le protocole et la couche CPF 12. La couche de sécurité accepte les données de sécurité en provenance de l'application sécuritaire et transfère ces données via le protocole FSCP 12/1.



Légende

Anglais	Français
Safety application	Application de sécurité
Safety objects	Objets de sécurité
Safety data	Données de sécurité
Safety management	Gestion de sécurité
Application layer	Couche Application
Data link layer	Couche Liaison de données
Physical layer	Couche physique

Figure 4 – Architecture logicielle du protocole FSCP 12/1

Une PDU de sécurité contenant les données de sécurité et les mesures de détection d'erreurs requises est comprise dans les objets de données de processus de communication (PDO). La mise en correspondance dans les données de processus du système de communication et du démarrage du diagramme d'état de communication ne fait pas partie du protocole de sécurité.

Le calcul de la probabilité d'erreur résiduelle propre au protocole FSCP 12/1 ne bénéficie pas des mécanismes de détection d'erreurs du système de communication. En d'autres termes, le protocole peut également être transmis par l'intermédiaire d'autres systèmes de communication. Toute liaison de transmission peut être utilisée, y compris des systèmes de bus de terrain, l'Ethernet ou des trajets de transfert similaires, des câbles à fibre optique, des câbles en cuivre, voire des liaisons radioélectriques.

5.5 Relations avec la FAL (et DLL, PhL)

5.5.1 Généralités

Cette couche de communication de sécurité est conçue pour être utilisée avec les profils de communication CPF 12. Elle ne se limite toutefois pas à ce profil de communication.

5.5.2 Types de données

Les profils définis dans la présente partie prennent en charge tous les types de données CPF 12 définis dans la CEI 61158-5-12.

6 Services de la couche de communication de sécurité

6.1 Connexion FSoE

La connexion entre deux partenaires de communication FSCP 12/1 (nœuds FSCP 12/1) est appelée connexion FSoE. Dans une connexion FSoE, l'un des partenaires de communication est toujours le maître FSoE et l'autre est l'esclave FSoE.

Le maître FSoE initialise la connexion FSoE à la mise sous tension ou suite à un défaut de communication, tandis que l'esclave FSoE se limite à des réponses. Le maître FSoE établit les paramètres de communication sécuritaires et, de manière facultative, les paramètres de l'application sécuritaire de l'esclave FSoE.

Les données du processus de sécurité transmises du maître FSoE à l'esclave FSoE sont appelées SafeOutputs (sorties sûres). Les données de sécurité transmises de l'esclave FSoE au maître FSoE sont appelées SafeInputs (entrées sûres).

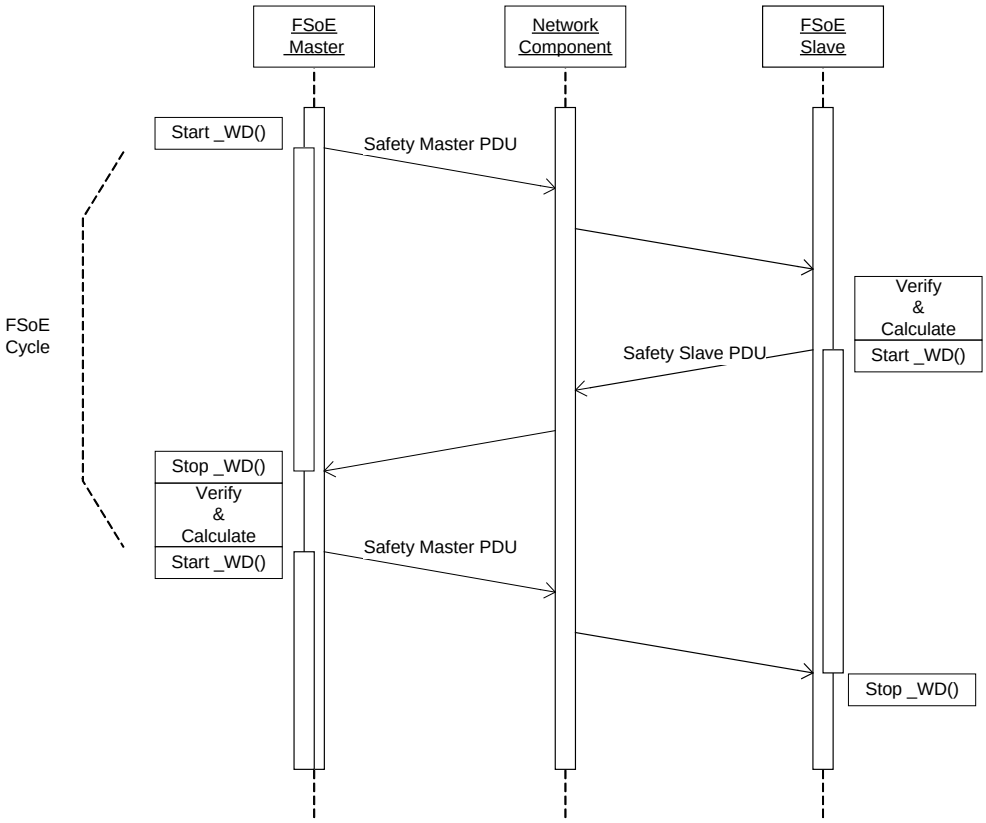
La PDU de sécurité transmise du maître FSoE à l'esclave FSoE est appelée PDU de maître de sécurité. La PDU de sécurité transmise de l'esclave FSoE au maître FSoE est appelée PDU d'esclave de sécurité.

6.2 Cycle FSoE

Le maître FSoE envoie les PDU de maître de sécurité à l'esclave FSoE et lance le chien de garde FSoE.

Après avoir vérifié l'intégrité des PDU de sécurité, l'esclave FSoE transfère les SafeOutputs à l'application de sécurité. Il calcule la PDU d'esclave de sécurité avec les SafeInputs provenant de l'application de sécurité et envoie cette PDU au maître FSoE. L'esclave FSoE lance également son chien de garde FSoE. Ce processus est illustré en Figure 5.

Un cycle FSoE se termine après réception d'une PDU d'esclave de sécurité valide.



Légende

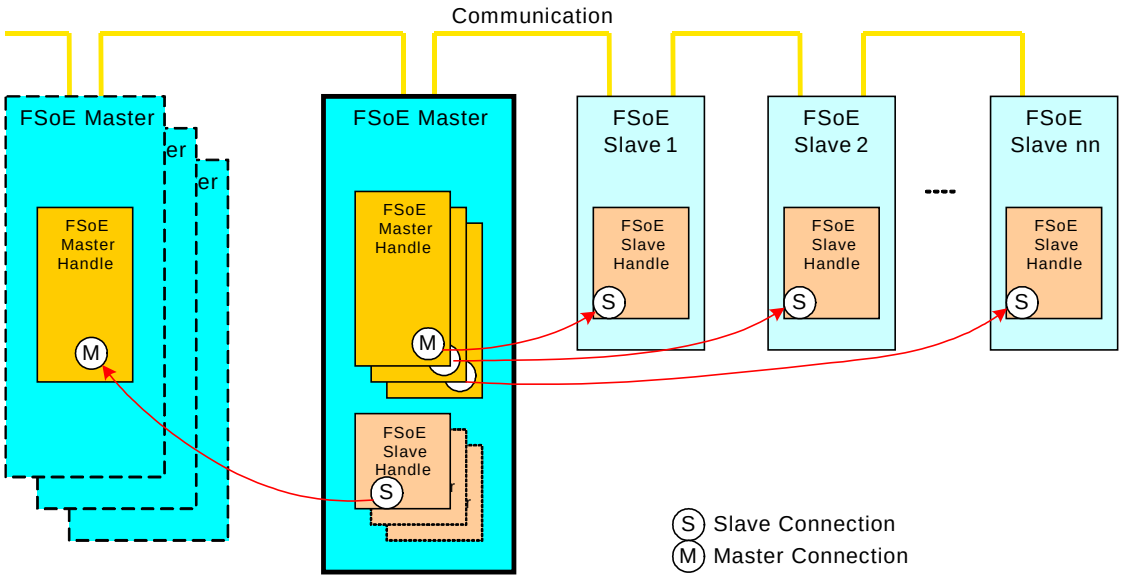
Anglais	Français
FsoE master	Maître FSoE
Network component	Composant de réseau
FSoE slave	Esclave FSoE
Safety master PDU	PDU de maître de sécurité
FSoE cycle	Cycle FSoE
Safety slave PDU	PDU d'esclave de sécurité
Verify & calculate	Vérifier & calculer

Figure 5 – Cycle FSoE

6.3 Services FSoE

Lors de chaque connexion FSoE, le maître FSoE doit prendre en charge un gestionnaire de maître FSoE pour commander l'esclave FSoE correspondant.

Pour une communication entre maîtres FSoE, il convient que le maître FSoE puisse prendre en charge un ou plusieurs gestionnaires d'esclaves FSoE. La Figure 6 illustre les fonctionnalités FSoE disponibles dans le dispositif maître et esclave FSoE.



Légende

Anglais	Français
FSoE master	Maître FSoE
FSoE slave	Esclave FSoE
FSoE master handler	Gestionnaire de maître FSoE
FsoE slave handler	Gestionnaire d'esclave FSoE
Slave connection	Connexion esclave
Master connection	Connexion esclave

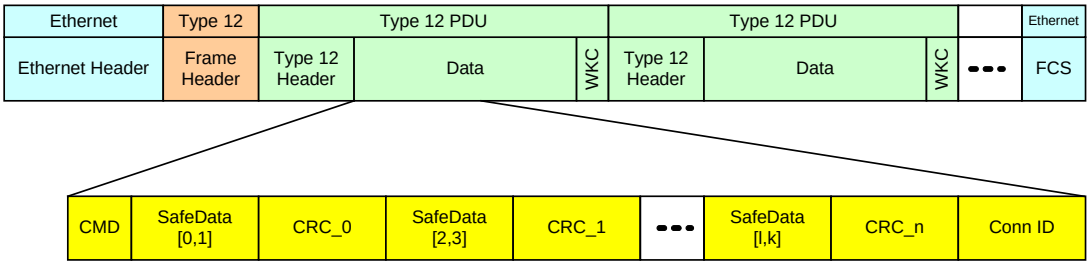
Figure 6 – Structure de communication FSCP 12/1

7 Protocole de couche de communication de sécurité

7.1 Format de PDU de sécurité

7.1.1 Structure de PDU de Sécurité

La Figure 7 illustre la structure d'une PDU de sécurité intégrée dans une PDU de Type 12. La structure générale d'une PDU de sécurité est présentée dans le Tableau 3.



Légende

Anglais	Français
Type 12 PDU	PDU de Type 12
Ethernet header	En-tête Ethernet
Frame header	En-tête de trame
Type 12 header	En-tête de Type 12
Data	Données
Conn ID	ID conn

Figure 7 – PDU de sécurité pour CPF 12 intégrée dans une PDU de type 12

La transmission des PDU de sécurité s'effectue de manière cyclique via le bus de terrain subordonné. Chaque nœud FSCP 12/1 détecte une nouvelle PDU de sécurité si au moins un bit dans la PDU de sécurité a changé.

La PDU de sécurité a une longueur variable spécifiée dans la description de dispositif de l'esclave FSoE. La longueur des données de sécurité peut être de 1 octet ou d'un nombre pair d'octets. La longueur des données de sécurité peut être différente en fonction du sens, entrée et sortie.

La plus courte des deux longueurs de données de sécurité dans la PDU de maître de sécurité et dans la PDU d'esclave de sécurité détermine le nombre de données de sécurité utilisées au cours de la phase d'initialisation de la connexion FSoE avec des informations concernant les paramètres. Dans la PDU la plus longue, les données de sécurité restantes sont mises à zéro.

Tableau 3 – PDU de sécurité générale

Octet	Nom	Description
0	Command	Commande
1	SafeData[0]	données de sécurité, octet 0
2	SafeData[1]	données de sécurité, octet 1
3	CRC_0_Lo	octet de poids faible (bits 0 à 7) du CRC_0 sur 16 bits
4	CRC_0_Hi	octet de poids fort (bits 8 à 15) du CRC_0 sur 16 bits
5	SafeData[2]	données de sécurité, octet 2
6	SafeData[3]	données de sécurité, octet 3
7	CRC_1_Lo	octet de poids faible (bits 0 à 7) du CRC_1 sur 16 bits
8	CRC_1_Hi	octet de poids fort (bits 8 à 15) du CRC_1 sur 16 bits
…	…	…
(n-1) × 2-1	SafeData[n-2]	données de sécurité, octet n-2
(n-1) × 2	SafeData[n-1]	données de sécurité, octet n-1
(n-1) × 2+1	CRC_(n-2)/2_Lo	octet de poids faible (bits 0 à 7) du CRC_(n-2)/2 sur 16 bits
(n-1) × 2+2	CRC_(n-2)/2_Hi	octet de poids fort (bits 8 à 15) du CRC_(n-2)/2 sur 16 bits
(n-1) × 2+3	Conn_Id_Lo	Identifiant unique de connexion, octet de poids faible
(n-1) × 2+4	Conn_Id_Hi	Identifiant unique de connexion, octet de poids fort

La PDU de sécurité peut transférer n octets de données de sécurité. Deux octets de données sont transmis par un CRC à 2 octets.

La PDU de sécurité la plus courte est constituée de 6 octets, qui peuvent être utilisés pour transférer 1 octet de données de sécurité, comme présenté dans le Tableau 4.

Tableau 4 – PDU de sécurité courte

Octet	Nom	Description
0	Command	Commande
1	SafeData[0]	données de sécurité, octet 0
2	CRC_0_Lo	octet de poids faible (bits 0 à 7) du CRC_0 sur 16 bits
3	CRC_0_Hi	octet de poids fort (bits 8 à 15) du CRC_0 sur 16 bits
4	Conn_Id_Lo	Identifiant unique de connexion, octet de poids faible
5	Conn_Id_Hi	Identifiant unique de connexion, octet de poids fort

7.1.2 PDU de sécurité "Commande"

La PDU de sécurité "Commande" détermine la signification des données de sécurité sur la base du schéma du Tableau 5.

Tableau 5 – PDU de sécurité "Commande"

Commande	Description
0x36	ProcessData
0x2A	Réinitialisation
0x4E	Session
0x64	Connexion
0x52	Paramètre
0x08	FailSafeData

7.1.3 CRC de PDU de sécurité

7.1.3.1 Calcul du CRC

Deux octets de données de sécurité sont transmis par un CRC à deux octets correspondant.

Outre les données transmises (commande, données, ConnID), le CRC_0 de la PDU de sécurité inclut également un numéro de séquence virtuel, le CRC_0 de la dernière PDU de sécurité reçue et trois octets supplémentaires à zéro. Si un seul octet de données de sécurité est transmis, les SafeData[1] (données sûres) sont omises dans le calcul.

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

Le Tableau 6 présente la séquence de calcul du CRC_0.

Tableau 6 – Séquence de calcul du CRC_0

Etape	Argument
1	CRC_0 (bit 0 à 7) reçu
2	CRC_0 (bit 8 à 15) reçu
3	ConnID (bit 0 à 7)
4	ConnID (bit 8 à 15)
5	Sequence_Number (bit 0 à 7)
6	Sequence_Number (bit 8 à 15)
7	Commande
8	SafeData[0]
9	SafeData[1]
10	0
11	0
12	0

Le CRC_i (0 < i <= ((n-2)/2)) de la PDU de sécurité inclut en outre l'indice i de la CRC.

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

Le Tableau 7 présente la séquence de calcul du CRC_i.

Tableau 7 – Séquence de calcul du CRC_i (i>0)

Etape	Argument
1	CRC_0 (bit 0 à 7) reçu
2	CRC_0 (bit 8 à 15) reçu
3	ConnID (bit 0 à 7)
4	ConnID (bit 8 à 15)
5	Sequence_Number (bit 0 à 7)
6	Sequence_Number (bit 8 à 15)
7	Commande
8	Indice i (bit 0 à 7)
9	Indice i (bit 8 à 15)
10	SafeData[2 × i]
11	SafeData[2 × I + 1]
12	0
13	0
14	0

7.1.3.2 Sélection du polynôme CRC

Le polynôme 0x139B7 est utilisé pour calculer les CRC; il est appelé polynôme de sécurité.

Un taux de couverture binaire de 10^{-2} doit être utilisé pour déterminer la probabilité d'erreur résiduelle afin de permettre la transmission de PDU de sécurité via un canal noir dont les caractéristiques de transfert ne sont pas prises en compte dans les considérations de sécurité. La probabilité d'erreur résiduelle ne doit pas dépasser 10^{-9}.

La sécurité est assurée sur la base de la commutation du maître FSoE et de l'esclave FSoE à l'état de réinitialisation (c'est-à-dire à l'état de sécurité) dès la détection d'une erreur.

Tous les facteurs utilisés pour le calcul des CRC, à l'exception des données de sécurité, ont une valeur fixe prévue, de façon à ce que seules les données de sécurité soient prises en compte dans le calcul de la probabilité d'erreur résiduelle.

La preuve mathématique démontrant que la probabilité d'erreur résiduelle avec le polynôme de sécurité pour des données de sécurité de 16 bits et un taux de couverture binaire de 10^{-2} ne dépasse pas 10^{-9}, est décrite dans un document séparé traitant de la détection quantitative d'erreurs.

7.1.3.3 Héritage de CRC

L'inclusion (héritage) du CRC_0 du dernier télégramme reçu dans le calcul du CRC permet de s'assurer que deux PDU consécutives de maître de sécurité ou d'esclave de sécurité sont différentes, même si les autres données demeurent inchangées.

L'héritage du CRC_0 assure également une transmission sûre et cohérente des données qui sont réparties sur plusieurs PDU de Type 12 du fait de leur longueur.

Le CRC_0 de la PDU de sécurité reçue est inclus dans le calcul de tous les CRC_i pour la PDU de sécurité à envoyer.

Le Tableau 8 présente un exemple d'héritage de CRC_0.

Tableau 8 – Exemple d'héritage du CRC_0

Cycle FSoE	Maître FSoE		Esclave FSoE	
	ancien CRC_0	nouveau CRC_0	ancien CRC_0	nouveau 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)

Dans le cycle FSoE j, le maître FSoE reçoit une PDU d'esclave de sécurité avec CRC_0 (2 × j - 1). Etant donné que la valeur de CRC_0 (2 × j - 2), qui était incluse dans le calcul de CRC_0 (2 × j - 1), a été calculée par le maître FSoE au cours du cycle FSoE (j – 1), le maître FSoE peut vérifier CRC_0 (2 × j - 1) dans la PDU d'esclave de sécurité.

De la même manière, au cours du cycle FSoE j, l'esclave FSoE reçoit la PDU de maître de sécurité avec CRC_0 (2 × j) et est également capable de vérifier cette PDU, puisque CRC_0 (2 × j - 1) calculé par l'esclave FSoE au cours du cycle FSoE (j – 1) a été inclus dans le calcul.

7.1.3.4 Numéro de séquence

Dans le Tableau 8 CRC_0 (2 × j) peut être égal à CRC_0 (2 × j - 2). Avec des PDU de sécurité courtes, il est possible que la PDU de maître de sécurité dans le cycle FSoE (j – 1) soit identique à la PDU de maître de sécurité dans le cycle FSoE j, de telle sorte que l'esclave FSoE ne reconnaîtrait pas la PDU de maître de sécurité comme étant une nouvelle PDU dans le cycle FSoE j et le chien de garde FSoE serait alors déclenché.

Les CRC de PDU de maître de sécurité incluent par conséquent un numéro de séquence maître virtuel de 16 bits que le maître FSoE incrémente à chaque nouvelle PDU de maître de sécurité. Le CRC de la PDU d'esclave de sécurité comprend également un numéro de séquence esclave virtuel de 16 bits, qui est incrémenté par l'esclave FSoE à chaque nouvelle PDU d'esclave de sécurité.

Si CRC_0 (2 × j) égal CRC_0 (2 × j - 2) malgré ces mesures, le numéro de séquence maître est encore incrémenté jusqu'à ce que CRC_0 (2 × j) ne soit plus égal à CRC_0 (2 × j - 2). Cet algorithme est utilisé à la fois pour que le maître FSoE génère la PDU de maître de sécurité et pour que l'esclave FSoE vérifie la PDU de maître de sécurité.

Si CRC_0 (2 × j + 1) égal CRC_0 (2 × j - 1), le numéro de séquence esclave est encore incrémenté jusqu'à ce que CRC_0 (2 × j + 1) ne soit plus égal à CRC_0 (2 × j - 1). Cet algorithme est utilisé à la fois pour que l'esclave FSoE génère la PDU d'esclave de sécurité et pour que le maître FSoE vérifie la PDU d'esclave de sécurité.

Le numéro de séquence peut prendre des valeurs entre 1 et 65 535. Après 65 535, la séquence repart de nouveau à 1, c'est-à-dire que 0 n'est pas utilisé.

7.1.3.5 Indice de CRC_0

Si plus de 2 octets de données de sécurité et par conséquent 2 ou plusieurs CRC (par exemple CRC_0 et CRC_1) sont transmis, les mesures décrites ci-dessus ne suffisent pas à détecter toutes les options d'inversion dans une PDU de sécurité (voir l'exemple du Tableau 9).

Tableau 9 – Exemple pour des données de sécurité de 4 octets avec échange des octets 1 à 4 par les octets 5 à 8

Octet	Nom	Description
0	Command	Commande
1	SafeData[2]	données de sécurité, octet 2
2	SafeData[3]	données de sécurité, octet 3
3	CRC_1_Lo	octet de poids faible (bits 0 à 7) du CRC_1 sur 16 bits
4	CRC_1_Hi	octet de poids fort (bits 8 à 15) du CRC_1 sur 16 bits
5	SafeData[0]	données de sécurité, octet 0
6	SafeData[1]	données de sécurité, octet 1
7	CRC_0_Lo	octet de poids faible (bits 0 à 7) du CRC_0 sur 16 bits
8	CRC_0_Hi	octet de poids fort (bits 8 à 15) du CRC_0 sur 16 bits
9	Conn_Id_Lo	octet de poids faible (bits 0 à 7) de l'identifiant unique de connexion
10	Conn_Id_Hi	octet de poids faible (bits 0 à 7) de l'identifiant unique de connexion

L'indice i (valeur de 2 octets) est par conséquent également inclus dans le CRC_i correspondant. Ceci permet la détection d'inversion des octets 1 à 4 et 5 à 8.

7.1.3.6 Octets à zéro supplémentaires

La probabilité d'erreur résiduelle est calculée en utilisant le rapport entre les erreurs détectées et les erreurs non détectées. Les erreurs non détectées sont essentiellement des erreurs déjà détectées par le polynôme CRC pour le canal noir (polynôme normalisé), car ces erreurs ne sont pas apparentes dans la couche de sécurité parce qu'elles sont préalablement filtrées. Le rapport le plus défavorable entre erreurs détectées (erreurs qui ne sont pas détectées par le polynôme normalisé mais par le polynôme CRC de la couche de sécurité) et erreurs non détectées (erreurs déjà détectées par le polynôme normalisé) a lieu si le polynôme normalisé est divisible par le polynôme de sécurité.

Trois octets à zéro sont inclus dans le calcul pour assurer une indépendance suffisante entre les deux polynômes dans ce cas précis.

7.1.3.7 ID de session

Dans le cas particulier des bus de terrain dont les données sont transmises via Ethernet, un dispositif défectueux qui stocke des télégrammes (par exemple un commutateur) peut entraîner l'insertion d'une séquence de télégrammes au mauvais moment. L'héritage CRC signifie qu'une séquence de PDU de sécurité dépend également de l'historique.

Le fait de transmettre un ID de session généré au hasard au moment de l'établissement de la connexion FSoE assure que les deux séquences de PDU de sécurité seront différentes lors de leur mise sous tension.

L'ID de session peut prendre des valeurs entre 0 et 65 535.

7.2 Procédure de communication FSCP 12/1

7.2.1 Cycle de message

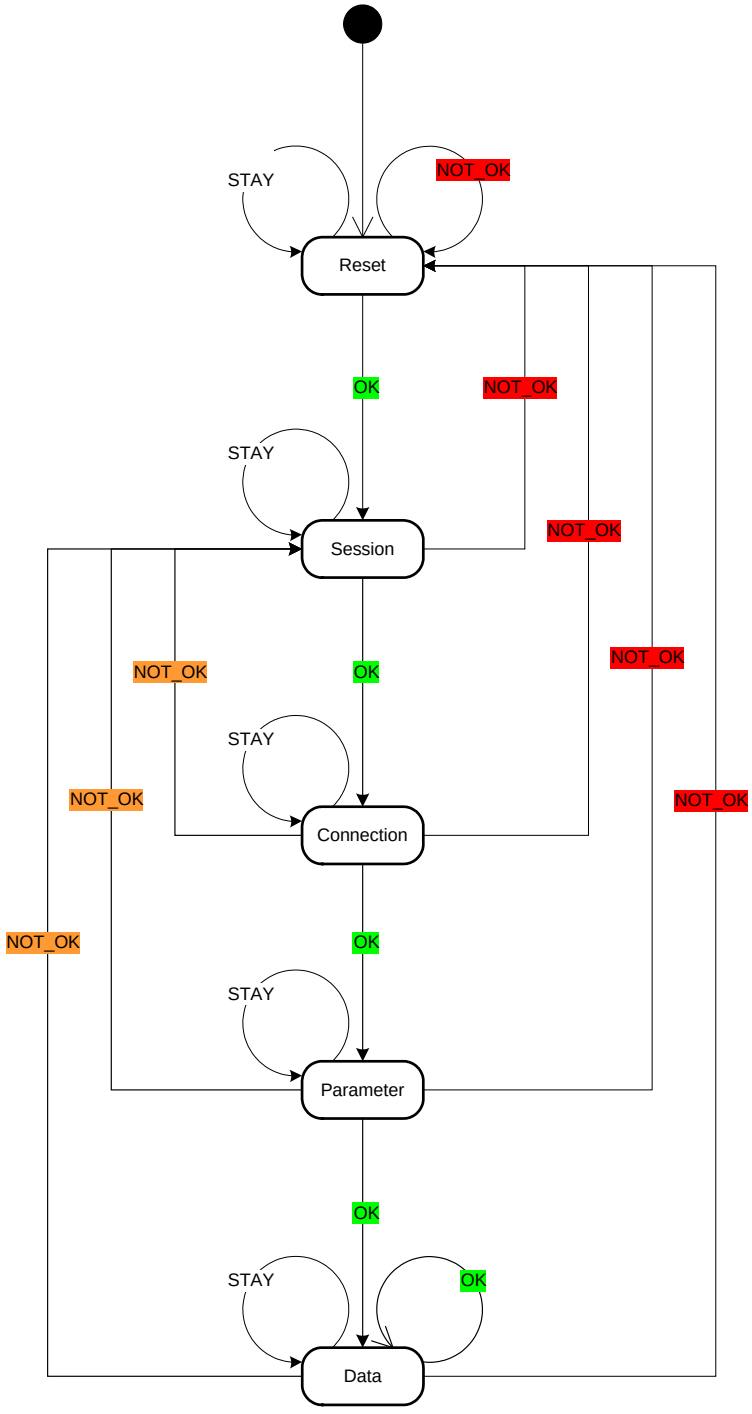
La communication FSCP 12/1 fonctionne avec un cycle de message acquitté (cycle FSoE), c'est-à-dire que le maître FSoE envoie une PDU de maître de sécurité à l'esclave FSoE et attend en retour une PDU d'esclave de sécurité. C'est seulement ensuite que la PDU de maître de sécurité suivante est générée.

7.2.2 Etats de nœuds FSCP 12/1

7.2.2.1 Généralités

Lors de l'établissement d'une connexion FSoE, les nœuds FSCP 12/1 prennent différents états avant que les données de sécurité ne deviennent valides et que l'état de sécurité ne soit abandonné.

La Figure 8 illustre les états des nœuds FSoE.



Légende

Anglais	Français
STAY	RESTE
Reset	Réinitialisation
Connection	Connexion
Parameter	Paramètre
Data	données

Figure 8 – Etats de nœuds FSCP 12/1

Après mise sous tension ou en cas d'erreur de communication FSoE, le maître et l'esclave FSoE sont à l'état de réinitialisation. Les nœuds FSoE passent également à l'état de réinitialisation s'ils détectent une erreur dans la communication ou l'application locale. Après une commande de réinitialisation FSoE de l'esclave FSoE, le maître FSoE passe à l'état de session (les transitions sont représentées en couleur orange). Après une commande de réinitialisation provenant du maître FSoE, l'esclave FSoE passe à l'état de réinitialisation. L'état Données peut alors être pris en charge via les états Session, Connexion et Paramètre. On ne peut sortir de l'état Sortie sûre que dans l'état Données.

7.2.2.2 Etat Réinitialisation

L'état de réinitialisation est utilisé pour réinitialiser la connexion FSoE après mise sous tension ou après une erreur de communication FSoE. Le maître FSoE abandonne l'état de réinitialisation lorsqu'il utilise la commande Session pour envoyer une PDU de maître de sécurité à l'esclave FSoE. L'esclave FSoE sort de l'état de réinitialisation lorsqu'il reçoit, au moyen de la commande Session, une PDU de maître de sécurité valide.

Dans l'état de réinitialisation, le numéro de séquence et le CRC du dernier télégramme utilisé dans le calcul du CRC sont réinitialisés.

Le Tableau 10 donne un exemple de PDU de maître de sécurité pour 4 octets de données de sécurité avec la commande de réinitialisation.

Tableau 10 – PDU de Maître de Sécurité pour 4 octets de données de sécurité avec la commande = Réinitialisation après redémarrage (réinitialisation de la connexion) ou erreur

Octet	Nom	Description
0	Command	*Réinitialisation*
1	SafeData[0]	code d'erreur (bit 0 à 7), 0 pour le redémarrage
2	SafeData[1]	non utilisé (= 0)
3	CRC_0_Lo	octet de poids faible (bits 0 à 7) du CRC_0 sur 16 bits
4	CRC_0_Hi	octet de poids fort (bits 8 à 15) du CRC_0 sur 16 bits
5	SafeData[2]	non utilisé (= 0)
6	SafeData[3]	non utilisé (= 0)
7	CRC_1_Lo	octet de poids faible (bits 0 à 7) du CRC_1 sur 16 bits
8	CRC_1_Hi	octet de poids fort (bits 8 à 15) du CRC_1 sur 16 bits
9	Conn_Id_Lo	non utilisé (= 0)
10	Conn_Id_Hi	non utilisé (= 0)

L'esclave FSoE acquitte la commande de réinitialisation en mettant SafeData à 0.

Le Tableau 11 donne un exemple de PDU d'esclave de sécurité pour 4 octets de données de sécurité avec la commande Réinitialisation.

Tableau 11 – PDU d'Esclave de Sécurité pour 4 octets de données de sécurité avec la commande = Réinitialisation pour acquittement d'une commande Réinitialisation en provenance du Maître FSoE

Octet	Nom	Description
0	Command	*Réinitialisation*
1	SafeData[0]	0
2	SafeData[1]	0
3	CRC_0_Lo	octet de poids faible (bits 0 à -7) du CRC_0 sur 16 bits

Octet	Nom	Description
4	CRC_0_Hi	octet de poids fort (bits 8 à 15) du CRC_0 sur 16 bits
5	SafeData[2]	non utilisé (= 0)
6	SafeData[3]	non utilisé (= 0)
7	CRC_1_Lo	octet de poids faible (bits 0 à 7) du CRC_1 sur 16 bits
8	CRC_1_Hi	octet de poids fort (bits 8 à 15) du CRC_1 sur 16 bits
9	Conn_Id_Lo	non utilisé (= 0)
10	Conn_Id_Hi	non utilisé (= 0)

L'Esclave FSoE envoie également une PDU de Sécurité avec la commande Réinitialisation lors d'un redémarrage (réinitialisation de la connexion) ou en cas d'erreur. Ce processus est présenté dans le Tableau 12 qui donne un exemple pour 4 octets de données de sécurité.

Tableau 12 – PDU d'Esclave de Sécurité pour 4 octets de données de sécurité avec la commande = Réinitialisation après redémarrage (réinitialisation de la connexion) ou erreur

Octet	Nom	Description
0	Command	*Réinitialisation*
1	SafeData[0]	code d'erreur (bit 0 à 7), 0 pour le redémarrage
2	SafeData[1]	non utilisé (= 0)
3	CRC_0_Lo	octet de poids faible (bits 0 à -7) du CRC_0 sur 16 bits
4	CRC_0_Hi	octet de poids fort (bits 8 à 15) du CRC_0 sur 16 bits
5	SafeData[2]	non utilisé (= 0)
6	SafeData[3]	non utilisé (= 0)
7	CRC_1_Lo	octet de poids faible (bits 0 à 7) du CRC_1 sur 16 bits
8	CRC_1_Hi	octet de poids fort (bits 8 à 15) du CRC_1 sur 16 bits
9	Conn_Id_Lo	non utilisé (= 0)
10	Conn_Id_Hi	non utilisé (= 0)

Le Maître FSoE acquitte la commande Réinitialisation en envoyant une PDU de Maître de Sécurité au moyen de la commande Session.

7.2.2.3 Etat Session

Au cours de la transition vers ou dans l'état session, un ID de Session Maître sur 16 bits est transmis du Maître FSoE à l'Esclave FSoE, qui répond en renvoyant en retour son propre ID de Session Esclave. Les deux nœuds FSoE génèrent l'ID de Session comme un numéro aléatoire qui est uniquement utilisé pour différencier plusieurs séquences de PDU de Sécurité dans le cas où il y a plusieurs redémarrages de la Connexion FSoE.

Le Tableau 13 présente un exemple de la PDU de Maître de Sécurité pour 4 octets de données de sécurité au moyen de la commande Session.

Tableau 13 – PDU de Maître de Sécurité pour 4 octets de données de sécurité avec la commande = Session

Octet	Nom	Description
0	Command	*Session*
1	SafeData[0]	*Id de Session Maître*, octet 0
2	SafeData[1]	*Id de Session Maître*, octet 1

Octet	Nom	Description
3	CRC_0_Lo	octet de poids faible (bits 0 à 7) du CRC_0 sur 16 bits
4	CRC_0_Hi	octet de poids fort (bits 8 à 15) du CRC_0 sur 16 bits
5	SafeData[2]	non utilisé (= 0)
6	SafeData[3]	non utilisé (= 0)
7	CRC_1_Lo	octet de poids faible (bits 0 à 7) du CRC_1 sur 16 bits
8	CRC_1_Hi	octet de poids fort (bits 8 à 15) du CRC_1 sur 16 bits
9	Conn_Id_Lo	non utilisé (= 0)
10	Conn_Id_Hi	non utilisé (= 0)

L'Esclave FSoE acquitte une commande Session en renvoyant l'ID de Session Esclave. Le Tableau 14 donne un exemple de PDU d'Esclave de Sécurité pour 4 octets de SafeData au moyen de la commande Session.

Tableau 14 – PDU d'Esclave de Sécurité pour 4 octets de données de sécurité avec la commande = Session

Octet	Nom	Description
0	Command	*Session*
1	SafeData[0]	*ID de Session Esclave*, octet 0
2	SafeData[1]	*ID de Session Esclave*, octet 1
3	CRC_0_Lo	octet de poids faible (bits 0 à 7) du CRC_0 sur 16 bits
4	CRC_0_Hi	octet de poids fort (bits 8 à 15) du CRC_0 sur 16 bits
5	SafeData[2]	non utilisé (= 0)
6	SafeData[3]	non utilisé (= 0)
7	CRC_1_Lo	octet de poids faible (bits 0 à 7) du CRC_1 sur 16 bits
8	CRC_1_Hi	octet de poids fort (bits 8 à 15) du CRC_1 sur 16 bits
9	Conn_Id_Lo	non utilisé (= 0)
10	Conn_Id_Hi	non utilisé (= 0)

Si la PDU de Sécurité contient au moins 2 octets de données de sécurité, l'ID de Session peut être transféré avec un Cycle FSoE. Si, d'autre part, la PDU de Sécurité contient uniquement 1 octet de données de sécurité, l'ID de Session doit être transféré avec deux Cycles FSoE.

La valeur de l'ID de Session n'a pas de pertinence en termes de sécurité, c'est-à-dire qu'il n'est pas nécessaire qu'une commutation dans le Nœud FSoE qui reçoit la PDU de Sécurité soit examinée du point de vue de la sécurité. L'ID de Connexion pour la commande Session est par conséquent mis à 0.

Le Maître FSoE sort de l'état session après avoir transféré l'ID de session complet et après réception des acquittements correspondants de l'Esclave FSoE en envoyant une PDU de Sécurité au moyen de la commande Connexion vers l'Esclave FSoE. L'Esclave FSoE sort de l'état session lorsqu'il reçoit une PDU de Sécurité, avec la commande Connexion, du Maître FSoE.

En cas de détection d'une erreur de communication FSoE, le Maître FSoE et l'Esclave FSoE sortent également de l'état Session.

Dans le Maître FSoE, après réception d'une commande RESET (RÉINITIALISATION), le numéro de séquence et le CRC du dernier télégramme utilisé dans le calcul du CRC sont tous deux réinitialisés.

7.2.2.4 Etat Connexion

En l'état connexion, un ID de Connexion sur 16 bits est transféré du Maître FSoE à l'Esclave FSoE. L'ID de Connexion doit être unique et il est généré par le configurateur de sécurité du Maître FSoE. Si plusieurs Maîtres FSoE sont présents sur le système de communication, l'utilisateur doit s'assurer que les ID de Connexion utilisés sont uniques.

Outre l'ID de Connexion sur 16 bits, l'Adresse d'Esclave FSoE unique est également transmise. Le Tableau 15 présente le contenu des données de sécurité transférées dans l'état Connexion.

Tableau 15 – Données de sécurité transmises dans l'état Connexion

Octet de Données de sécurité	Description
0	octet de poids faible (bits 0 à 7) de l'identifiant de connexion
1	octet de poids fort faible (bits 8 à 15) de l'identifiant de connexion
2	octet de poids faible (bits 0 à 7) de l'Adresse d'Esclave FSoE
3	octet de poids fort (bits 8 à 15) de l'Adresse d'Esclave FSoE

En fonction de la longueur des données de sécurité, il peut être nécessaire d'utiliser jusqu'à 4 Cycles FSoE. Pour une PDU de Sécurité à 4 octets de données de sécurité, un seul Cycle FSoE est nécessaire, comme présenté dans le Tableau 16 et dans le Tableau 17.

Tableau 16 – PDU de Maître de Sécurité pour 4 octets de données de sécurité dans l'état Connexion

Octet	Nom	Description
0	Command	*Connexion*
1	SafeData[0]	Identifiant de connexion, octet de poids faible
2	SafeData[1]	Identifiant de connexion, octet de poids fort
3	CRC_0_Lo	octet de poids faible (bits 0 à 7) du CRC_0 sur 16 bits
4	CRC_0_Hi	octet de poids fort (bits 8 à 15) du CRC_0 sur 16 bits
5	SafeData[2]	Adresse d'Esclave FSoE, octet de poids faible
6	SafeData[3]	Adresse d'Esclave FSoE, octet de poids fort
7	CRC_1_Lo	octet de poids faible (bits 0 à 7) du CRC_1 sur 16 bits
8	CRC_1_Hi	octet de poids fort (bits 8 à 15) du CRC_1 sur 16 bits
9	Conn_Id_Lo	Identifiant de connexion, octet de poids faible
10	Conn_Id_Hi	Identifiant de connexion, octet de poids fort

L'Esclave FSoE acquitte une commande Connexion en renvoyant les données de sécurité.

Tableau 17 – PDU d'Esclave de Sécurité pour 4 octets de données de sécurité dans l'état Connexion

Octet	Nom	Description
0	Command	*Connexion*
1	SafeData[0]	Identifiant de connexion, octet de poids faible
2	SafeData[1]	Identifiant de connexion, octet de poids fort
3	CRC_0_Lo	octet de poids faible (bits 0 à 7) du CRC_0 sur 16 bits
4	CRC_0_Hi	octet de poids fort (bits 8 à 15) du CRC_0 sur 16 bits
5	SafeData[2]	Adresse d'Esclave FSoE, octet de poids faible
6	SafeData[3]	Adresse d'Esclave FSoE, octet de poids fort
7	CRC_1_Lo	octet de poids faible (bits 0 à 7) du CRC_1 sur 16 bits
8	CRC_1_Hi	octet de poids fort (bits 8 à 15) du CRC_1 sur 16 bits
9	Conn_Id_Lo	Identifiant de connexion, octet de poids faible
10	Conn_Id_Hi	Identifiant de connexion, octet de poids fort

L'Adresse d'Esclave FSoE doit être unique dans le système de communication. Elle peut être établie au niveau du dispositif Esclave FSoE correspondant. En transférant l'Adresse d'Esclave FSoE avec l'ID de Connexion, l'Esclave FSoE peut vérifier s'il était réellement concerné, de sorte qu'un adressage invalide serait détecté. Sachant que l'ID de Connexion est également unique sur le système de communication, il est toujours envoyé dans les PDU de Sécurité suivantes, de façon à ce que le Maître FSoE et l'Esclave FSoE puissent savoir si le télégramme les concerne. L'ID de Connexion unique permet par conséquent de réaliser 65 535 Connexions FSoE sur le système de communication (l'ID de Connexion = 0 n'est pas autorisé).

7.2.2.5 Etat Paramètre

Dans l'état paramètre, la communication sécuritaire et les paramètres de l'application sécuritaire spécifique au dispositif sont transmis. Ces paramètres peuvent avoir n'importe quelle longueur. L'héritage CRC assure une transmission des paramètres en toute sécurité et cohérence.

Le Tableau 18 présente le contenu des données de sécurité transmises dans l'état Paramètre.

Tableau 18 – Données de sécurité transmises dans l'état Paramètre

Octet de données de sécurité	Description
0	octet de poids faible (bits 0 à 7) longueur des paramètres de communication, en octets (= 2)
1	octet de poids fort (bits 8 à -15) longueur des paramètres de communication, en octets (= 0)
2	octet de poids faible (bits 0 à 7) du chien de garde FSoE (en ms)
3	octet de poids fort (bits 8 à -15) du chien de garde FSoE (en ms)
4	octet de poids faible (bits 0 à 7) longueur des paramètres d'application, en octets
5	octet de poids fort (bits 8 à 15) longueur des paramètres d'application, en octets
6	1er octet du paramètre de l'application sécuritaire
…	
n+5	n^{ième} octet du paramètre de l'application sécuritaire

Le nombre de Cycles FSoE dans l'état paramètre dépend de la longueur des paramètres de l'application sécuritaire et de la longueur des données de sécurité dans la PDU de Sécurité. Si tous les octets de données de sécurité ne sont pas nécessaires dans le dernier Cycle FSoE, ils doivent être transmis à 0.

Deux Cycles FSoE sont nécessaires pour une PDU de Sécurité ayant 4 octets de données de sécurité et 2 octets de paramètre d'application sécuritaire; comme illustré du Tableau 19 au Tableau 22.

Tableau 19 – Première PDU de Maître de Sécurité pour 4 octets de données de sécurité dans l'état Paramètre

Octet	Nom	Description
0	Command	*Paramètre*
1	SafeData[0]	octet de poids faible (bits 0 à 7) longueur des paramètres de communication, en octets (= 2)
2	SafeData[1]	octet de poids fort (bits 8 à -15) longueur des paramètres de communication, en octets (= 0)
3	CRC_0_Lo	octet de poids faible (bits 0 à 7) du CRC_0 sur 16 bits
4	CRC_0_Hi	octet de poids fort (bits 8 à 15) du CRC_0 sur 16 bits
5	SafeData[2]	octet de poids faible (bits 0 à 7) du chien de garde FSoE (en ms)
6	SafeData[3]	octet de poids fort (bits 8 à -15) du chien de garde FSoE (en ms)
7	CRC_1_Lo	octet de poids faible (bits 0 à 7) du CRC_1 sur 16 bits
8	CRC_1_Hi	octet de poids fort (bits 8 à 15) du CRC_1 sur 16 bits
9	Conn_Id_Lo	Identifiant de connexion, octet de poids faible
10	Conn_Id_Hi	Identifiant de connexion, octet de poids fort

L'Esclave FSoE acquitte une commande Paramètre correcte en renvoyant les données de sécurité.

Tableau 20 – Première PDU d'Esclave de Sécurité pour 4 octets de données de sécurité dans l'état Paramètre

Octet	Nom	Description
0	Command	*Paramètre*
1	SafeData[0]	octet de poids faible (bits 0 à 7) longueur des paramètres de communication, en octets (= 2)
2	SafeData[1]	octet de poids fort (bits 8 à -15) longueur des paramètres de communication, en octets (= 0)
3	CRC_0_Lo	octet de poids faible (bits 0 à 7) du CRC_0 sur 16 bits
4	CRC_0_Hi	octet de poids fort (bits 8 à 15) du CRC_0 sur 16 bits
5	SafeData[2]	octet de poids faible (bits 0 à 7) du chien de garde FSoE (en ms)
6	SafeData[3]	octet de poids fort (bits 8 à -15) du chien de garde FSoE (en ms)
7	CRC_1_Lo	octet de poids faible (bits 0 à 7) du CRC_1 sur 16 bits
8	CRC_1_Hi	octet de poids fort (bits 8 à 15) du CRC_1 sur 16 bits
9	Conn_Id_Lo	Identifiant de connexion, octet de poids faible
10	Conn_Id_Hi	Identifiant de connexion, octet de poids fort

Le Maître FSoE envoie la seconde PDU de Maître de Sécurité après avoir correctement reçu la première PDU d'Esclave de Sécurité.

Tableau 21 – Seconde PDU de Maître de Sécurité pour 4 octets de données de sécurité dans l'état Paramètre

Octet	Nom	Description
0	Command	*Paramètre*
1	SafeData[0]	octet de poids faible (bits 0 à 7) longueur des paramètres d'application, en octets (= 2)
2	SafeData[1]	octet de poids fort (bits 8 à -15) longueur des paramètres d'application, en octets (= 0)
3	CRC_0_Lo	octet de poids faible (bits 0 à 7) du CRC_0 sur 16 bits
4	CRC_0_Hi	octet de poids fort (bits 8 à 15) du CRC_0 sur 16 bits
5	SafeData[2]	1er octet du paramètre de l'application sécuritaire
6	SafeData[3]	2ème octet du paramètre de l'application sécuritaire
7	CRC_1_Lo	octet de poids faible (bits 0 à 7) du CRC_1 sur 16 bits
8	CRC_1_Hi	octet de poids fort (bits 8 à 15) du CRC_1 sur 16 bits
9	Conn_Id_Lo	Identifiant de connexion, octet de poids faible
10	Conn_Id_Hi	Identifiant de connexion, octet de poids fort

L'Esclave FSoE acquitte une commande de paramètre correcte en renvoyant les données de sécurité.

Tableau 22 – Seconde PDU d'Esclave de Sécurité pour 4 octets de données de sécurité dans l'état Paramètre

Octet	Nom	Description
0	Command	*Paramètre*
1	SafeData[0]	octet de poids faible (bits 0 à 7) longueur des paramètres d'application, en octets (= 2)
2	SafeData[1]	octet de poids fort (bits 8 à -15) longueur des paramètres d'application, en octets (= 0)
3	CRC_0_Lo	octet de poids faible (bits 0 à 7) du CRC_0 sur 16 bits
4	CRC_0_Hi	octet de poids fort (bits 8 à 15) du CRC_0 sur 16 bits
5	SafeData[2]	1er octet du paramètre de l'application sécuritaire
6	SafeData[3]	2ème octet du paramètre de l'application sécuritaire
7	CRC_1_Lo	octet de poids faible (bits 0 à 7) du CRC_1 sur 16 bits
8	CRC_1_Hi	octet de poids fort (bits 8 à 15) du CRC_1 sur 16 bits
9	Conn_Id_Lo	Identifiant de connexion, octet de poids faible
10	Conn_Id_Hi	Identifiant de connexion, octet de poids fort

Le chien de garde FSoE et les paramètres de l'application sécuritaire sont configurés au moyen d'un configurateur de sécurité du Maître FSoE.

7.2.2.6 Etat Données

7.2.2.6.1 Données valides

Tandis que dans les états précédents, le nombre de Cycles FSoE était fixe, dans l'état données, les Cycles FSoE sont transmis jusqu'à ce qu'une erreur de communication

survienne ou qu'un nœud FSoE s'arrête localement. Le Maître FSoE envoie des SafeOutputs à l'Esclave FSoE.

Un exemple de PDU de Maître de Sécurité pour 4 octets de SafeOutputs, au moyen de la commande ProcessData (données de processus), est donné dans le Tableau 23.

Tableau 23 – PDU de Maître de Sécurité pour 4 octets de ProcessData dans l'état Données

Octet	Nom	Description
0	Command	*ProcessData*
1	SafeData[0]	1er octet de SafeOutputs
2	SafeData[1]	2ème octet de SafeOutputs
3	CRC_0_Lo	octet de poids faible (bits 0 à 7) du CRC_0 sur 16 bits
4	CRC_0_Hi	octet de poids fort (bits 8 à 15) du CRC_0 sur 16 bits
5	SafeData[2]	3ème octet de SafeOutputs
6	SafeData[3]	4ème octet de SafeOutputs
7	CRC_1_Lo	octet de poids faible (bits 0 à 7) du CRC_1 sur 16 bits
8	CRC_1_Hi	octet de poids fort (bits 8 à 15) du CRC_1 sur 16 bits
9	Conn_Id_Lo	Identifiant de connexion, octet de poids faible
10	Conn_Id_Hi	Identifiant de connexion, octet de poids fort

L'Esclave FSoE acquitte la PDU de Maître de Sécurité et envoie des SafeInputs au Maître FSoE.

Le Tableau 24 donne un exemple de PDU d'esclave de sécurité pour 4 octets de SafeInputs avec la commande ProcessData.

Tableau 24 – PDU d'Esclave de Sécurité pour 4 octets de ProcessData dans l'état Données

Octet	Nom	Description
0	Command	*ProcessData*
1	SafeData[0]	1er octet de SafeInputs
2	SafeData[1]	2nd octet de SafeInputs
3	CRC_0_Lo	octet de poids faible (bits 0 à 7) du CRC_0 sur 16 bits
4	CRC_0_Hi	octet de poids fort (bits 8 à 15) du CRC_0 sur 16 bits
5	SafeData[2]	3ème octet de SafeInputs
6	SafeData[3]	4ème octet de SafeInputs
7	CRC_1_Lo	octet de poids faible (bits 0 à 7) du CRC_1 sur 16 bits
8	CRC_1_Hi	octet de poids fort (bits 8 à 15) du CRC_1 sur 16 bits
9	Conn_Id_Lo	Identifiant de connexion, octet de poids faible
10	Conn_Id_Hi	Identifiant de connexion, octet de poids fort

7.2.2.6.2 Commande FailSafeData (données de sécurité intégrée)

Si le Maître FSoE détecte localement que les SafeOutputs ne sont pas valides ou doivent passer à un état sûr, il envoie la commande FailSafeData.

Le Tableau 25 donne un exemple de PDU de Maître de Sécurité pour 4 octets de FailsafeData avec la commande FailsafeData.

Tableau 25 – PDU de Maître de Sécurité pour 4 octets de données de sécurité intégrée dans l'état Données

Octet	Nom	Description
0	Command	*FailSafeData*
1	SafeData[0]	Données de sécurité intégrée = 0
2	SafeData[1]	Données de sécurité intégrée = 0
3	CRC_0_Lo	octet de poids faible (bits 0 à 7) du CRC_0 sur 16 bits
4	CRC_0_Hi	octet de poids fort (bits 8 à 15) du CRC_0 sur 16 bits
5	SafeData[2]	Données de sécurité intégrée = 0
6	SafeData[3]	Données de sécurité intégrée = 0
7	CRC_1_Lo	octet de poids faible (bits 0 à 7) du CRC_1 sur 16 bits
8	CRC_1_Hi	octet de poids fort (bits 8 à 15) du CRC_1 sur 16 bits
9	Conn_Id_Lo	Identifiant de connexion, octet de poids faible
10	Conn_Id_Hi	Identifiant de connexion, octet de poids fort

Si l'Esclave FSoE détecte localement que les SafeInputs ne sont pas valides ou doivent passer à un état sûr, il envoie la commande FailSafeData.

Le Tableau 26 donne un exemple de PDU d'Esclave de Sécurité pour 4 octets de FailsafeData avec la commande FailsafeData.

Tableau 26 – PDU d'Esclave de Sécurité pour 4 octets de données de sécurité intégrée dans l'état Données

Octet	Nom	Description
0	Command	*FailSafeData*
1	SafeData[0]	Données de sécurité intégrée = 0
2	SafeData[1]	Données de sécurité intégrée = 0
3	CRC_0_Lo	octet de poids faible (bits 0 à 7) du CRC_0 sur 16 bits
4	CRC_0_Hi	octet de poids fort (bits 8 à 15) du CRC_0 sur 16 bits
5	SafeData[2]	Données de sécurité intégrée = 0
6	SafeData[3]	Données de sécurité intégrée = 0
7	CRC_1_Lo	octet de poids faible (bits 0 à 7) du CRC_1 sur 16 bits
8	CRC_1_Hi	octet de poids fort (bits 8 à 15) du CRC_1 sur 16 bits
9	Conn_Id_Lo	Identifiant de connexion, octet de poids faible
10	Conn_Id_Hi	Identifiant de connexion, octet de poids fort

La transmission de ProcessData ou FailSafeData est indépendante de la commande des PDU de sécurité reçues. Elle est uniquement fonction des circonstances locales.

7.3 Réaction en cas d'erreurs de communication

Un nœud FSoE peut détecter les erreurs énumérées dans le Tableau 27.

Tableau 27 – Erreurs de communication FSoE

Erreur	Description
Commande imprévue	La commande reçue n'est pas admise dans l'état courant
Commande inconnue	La commande reçue n'est pas définie
ID de connexion non valide	La connexion ne correspond pas à l'ID de connexion transmis dans l'état Connexion
Erreur de CRC	Au moins un des CRC_i reçus ne correspond pas au CRC_i calculé
Expiration du chien de garde	Aucune PDU de sécurité valide n'a été reçue dans le délai imparti du chien de garde FSoE
Adresse d'Esclave FSoE non valide	L'Adresse d'Esclave FSoE transmise dans l'état Connexion ne correspond pas à l'adresse locale établie sur l'Esclave FSoE
SafeData non valide	Les données de sécurité renvoyées par l'Esclave FSoE dans les états Session, Connexion et Paramètre ne correspondent pas aux valeurs attendues.
SafePara erronés	Les SafePara, envoyés à l'Esclave FSoE dans l'état Paramètre, sont erronés
Longueur du paramètre de communication non valide	La longueur du paramètre de communication est incorrecte
Paramètre de communication non valide	Le contenu du paramètre de communication est incorrect
Longueur du paramètre d'application non valide	La longueur du paramètre d'application est incorrecte
Paramètre d'application non valide	Le contenu du paramètre d'application est incorrect

Si un nœud FSoE détecte une erreur de communication, une commande Réinitialisation est envoyée, ainsi que le code d'erreur correspondant dans SafeData[0] à des fins de diagnostic. Le Maître FSoE passe alors à l'état Session et l'Esclave FSoE à l'état Réinitialisation. Les codes d'erreurs de communication FSoE sont énumérés dans le Tableau 28.

Tableau 28 – Codes d'erreurs de communication FSoE

Code d'erreur	Description
0	Réinitialisation locale ou acquittement d'une commande RESET (Réinitialisation)
1	Commande imprévue (INVALID_CMD)
2	Commande inconnue (UNKNOWN_CMD)
3	ID de connexion non valide (INVALID_CONNID)
4	Erreur de CRC (INVALID_CRC)
5	Expiration du chien de garde (WD_EXPIRED)
6	Adresse d'Esclave FSoE non valide (INVALID_ADDRESS)
7	Données de sécurité non valides (INVALID_DATA)
8	Longueur de paramètre de communication non valide (INVALID_COMMPARALEN)
9	Données de paramètre de communication non valides (INVALID_COMPARA)
10	Longueur de paramètre d'application non valide (INVALID_USERPARALEN)
11	Données de paramètre d'application non valides (INVALID_USERPARA)
0x80-0xFF	SafePara non valide (spécifique au dispositif)

7.4 Table d'états pour Maître FSoE

7.4.1 Diagramme d'état de Maître FSoE

7.4.1.1 Vue d'ensemble

En fonction de la procédure de communication, le Maître FSoE peut prendre les états énumérés dans le Tableau 29.

Tableau 29 – Etats du Maître FSoE

Etat	Description
Reset (Réinitialisation)	La connexion FSoE est réinitialisée (les sorties sont en état de sécurité)
Session	L'ID de session est en cours de transfert (les sorties sont en état de sécurité)
Connection (Connexion)	L'ID de connexion est en cours de transfert (les sorties sont en état de sécurité)
Parameter (Paramètre)	Les paramètres sont en cours de transfert (les sorties sont en état de sécurité)
Data (Données)	Les données de processus ou les données de sécurité intégrée sont en cours de transfert (les sorties ne sont actives que si la commande *ProcessData* est reçue)

Le diagramme d'état du Maître FSoE est illustré en Figure 9.

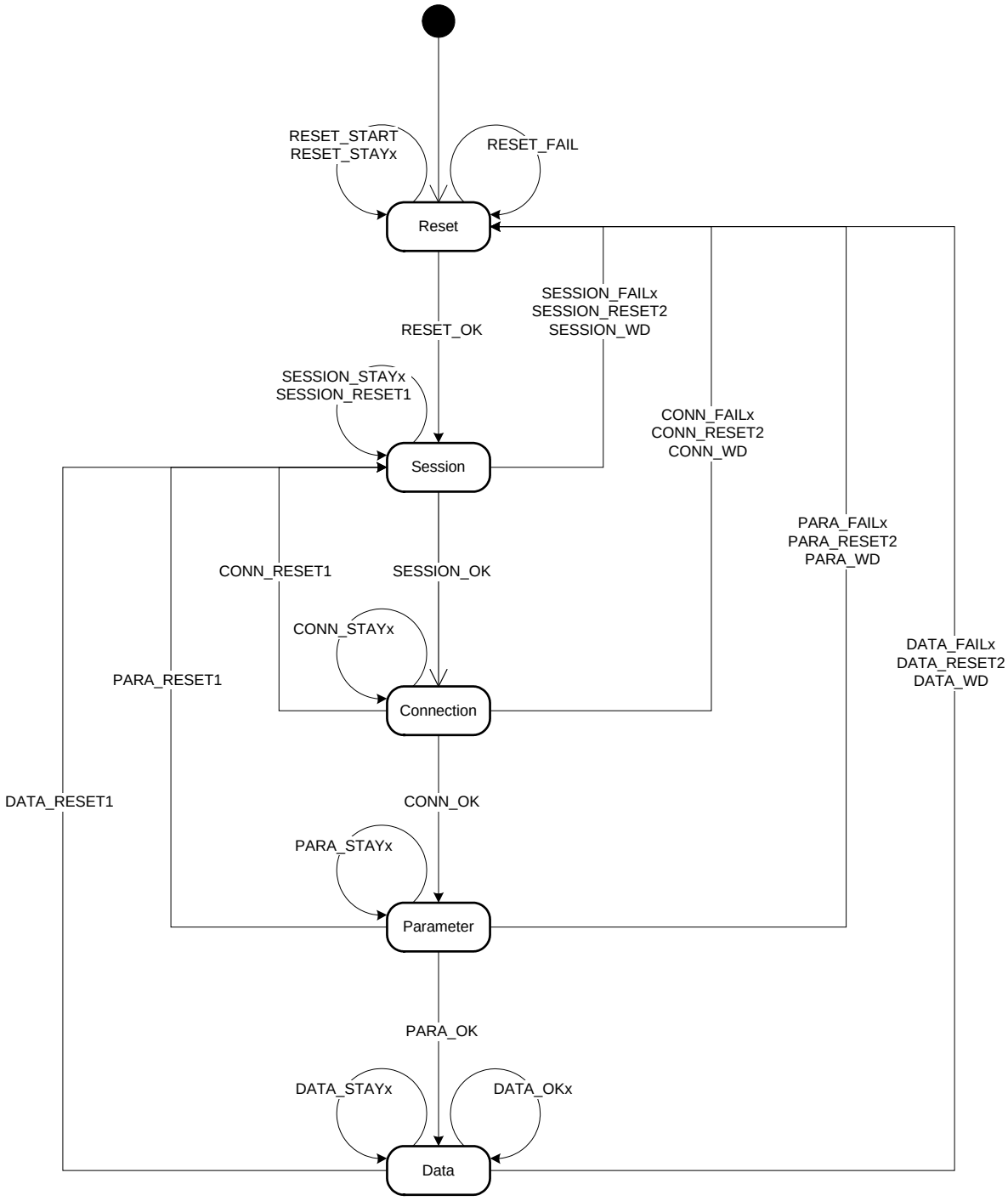


Figure 9 – Diagramme d'état du Maître FSoE

Les paragraphes ci-après analysent, pour chaque état, les événements qui peuvent affecter le Maître FSoE. Chaque événement est pris en compte sous certaines conditions comportant différentes actions ou états ultérieurs.

7.4.1.2 Evénements

Un événement peut inclure différents paramètres auxquels il est fait référence dans les Tables d'états. Le Tableau 30 énumère les événements considérés.

Tableau 30 – Evénements dans la table d'états de Maître FSoE

Evénement	Description
trame reçue	Une PDU de sécurité a été reçue, ce qui signifie qu'au moins un bit dans la PDU de sécurité a changé Paramètres: Trame = PDU de sécurité reçue Frame.Command = commande de la PDU de sécurité reçue Frame.Crc0 = CRC_0 de la PDU de sécurité reçue Frame.ConnId = ID de connexion de la PDU de sécurité reçue Frame.SafeData = données de sécurité de la PDU de sécurité reçue
Expiration du chien de garde	Le chien de garde FSoE a expiré, ce qui signifie qu'aucune PDU de sécurité n'a été reçue dans le délai imparti du chien de garde Paramètres: Aucun
Réinitialiser la Connexion	Demande, via une interface locale, de réinitialisation de la Connexion FSoE. Cet événement doit être déclenché à la mise sous tension pour démarrer la communication avec l'Esclave FSoE Paramètres: Aucun
Etablir la Commande de Données	Demande, via une interface locale, de passage des SafeOutputs à l'état de sécurité ou de sortie de l'état de sécurité Paramètres: DataCmd = *FailSafeData* ou *ProcessData*

7.4.1.3 Actions

En fonction des différentes conditions, certaines actions sont réalisées si un événement a lieu. Dans les tables d'états, les actions sont présentées comme des invocations de fonctions ou des attributions de variables.

Le Tableau 31 énumère les fonctions utilisées dans la table d'états de Maître FSoE.

Tableau 31 – Fonctions dans la table d'états de Maître FSoE

Fonction	Description
SendFrame(cmd, safeData, lastCrc, connId, seqNo, oldCrc, bNew)	Une trame de Maître FSoE est envoyée Paramètres: cmd = commande de trame SafeData = référence aux données de sécurité envoyées avec la trame lastCrc = CRC_0 de la dernière PDU d'Esclave de Sécurité incluse dans le calcul du CRC pour la trame connId = ID de connexion à insérer dans la trame et inclus dans le calcul du CRC seqNo = Pointeur vers le numéro de séquence Maître inclus dans le calcul du CRC pour les trames. Le seqNo incrémenté (peut-être plusieurs fois) est renvoyé oldCrc: Pointeur vers le CRC_0 de la dernière PDU de Maître de Sécurité envoyée. Le CRC_0 calculé est renvoyé bNew: Si bNew = VRAI et oldCrc est égal au CRC calculé, le calcul du CRC est recommencé avec le seqNo incrémenté jusqu'à ce que le CRC calculé ne soit plus égal à oldCrc (procédure conformément au 7.1.3.4)

Le Tableau 32 énumère les variables utilisées dans la table d'états de Maître FSoE.

Tableau 32 – Variables dans la table d'états de Maître FSoE

Variable	Description
LastCrc	CRC_0 de la dernière PDU de Maître de sécurité envoyée (initialisée à zéro à la mise sous tension)
OldMasterCrc	CRC_0 de la dernière PDU de Maître de sécurité envoyée (initialisée à zéro à la mise sous tension)
OldSlaveCrc	CRC_0 de la dernière PDU d'Esclave de sécurité reçue (initialisée à zéro à la mise sous tension)
MasterSeqNo	Numéro de séquence Maître à utiliser dans le CRC pour la PDU de Maître de Sécurité suivante (initialisée à zéro à la mise sous tension)
SlaveSeqNo	Numéro de séquence Esclave attendu à utiliser dans le CRC pour la PDU d'Esclave de Sécurité suivante (initialisée à zéro à la mise sous tension)
SessionId	ID de session généré de manière aléatoire (initialisé à zéro à la mise sous tension)
DataCommand	Indique si la commande *ProcessData* ou *FailSafeData* est envoyée dans l'état Données. Initialisée avec *FailSafeData* à la mise sous tension
BytesToBeSent	Si plusieurs PDU de sécurité doivent être envoyées dans l'état Session, Connexion ou Paramètre, cette variable indique le nombre d'octets qui restent à envoyer (initialisée à zéro à la mise sous tension)
ConnData	ConnData est constitué de l'ID de connexion et de l'Adresse d'Esclave FSoE. Initialisé par le configurateur de sécurité à la mise sous tension, en fonction de la configuration ConnData.ConnId: ConnectionId de la Connexion FSoE
SafePara	SafePara est constitué des paramètres de communication de sécurité et des paramètres d'application de Sécurité. Initialisé par le configurateur de sécurité à la mise sous tension, en fonction de la configuration SafePara.Watchdog: Chien de garde FSoE
SafeParaSize	Indique la longueur de SafePara. Initialisé par le configurateur de sécurité à la mise sous tension, en fonction des données de configuration
SafeOutputs	Contient les valeurs de processus des sorties de sécurité envoyées à l'Esclave FSoE. Initialisé à la valeur FS_VALUE (Données de sécurité intégrée = 0) à la mise sous tension
SafeInputs	Contient les valeurs de processus des entrées de sécurité reçues par l'Esclave FSoE. Initialisé à la valeur FS_VALUE (Données de sécurité intégrée = 0) à la mise sous tension
CommFaultReason	Indique le code d'erreur en cas d'erreur de communication
SecondSessionFrameSent	Si deux PDU de sécurité doivent être envoyées dans l'état Session, cette variable indique si la deuxième PDU est déjà envoyée. Cette variable est mise sur "FAUX" par la macro CREATE_SESSION_ID

7.4.1.4 Macros

Certaines fonctionnalités sont consolidées dans des macros pour préserver la transparence des Tables d'états.

Le Tableau 33 énumère les macros utilisées dans la table d'états de Maître FSoE.

Tableau 33 – Macros dans la table d'états de Maître FSoE

Macro	Description
IS_CRC_CORRECT(frame, lastCrc, seqNo, oldCrc, bNew)	Cette macro vérifie si les CRC de la PDU d'Esclave de Sécurité reçue sont corrects Paramètres: Trame = trame de réception lastCrc = CRC_0 de la dernière PDU de Maître de Sécurité envoyée, incluse dans les calculs du CRC pour la PDU reçue

Macro	Description
	seqNo = Numéro de séquence Esclave inclus dans les calculs du CRC pour la trame reçue. Le seqNo incrémenté (peut-être plusieurs fois) est renvoyé oldCrc: Pointeur vers le CRC_0 de la dernière PDU d'Esclave de Sécurité reçue. Le CRC_0 du télégramme reçu est renvoyé bNew: Si bNew = VRAI et oldCrc est égal au CRC calculé, le calcul du CRC est recommencé avec le seqNo incrémenté jusqu'à ce que le CRC calculé ne soit plus égal à oldCrc (procédure conformément au 7.1.3.4)
UPDATE_BYTES_TO_BE_SENT (bytesSent)	Cette macro vérifie le nombre d'octets qui restent à envoyer dans les états Session, Connexion et Paramètre avant que l'état ne soit modifié Paramètres: bytesSent = Nombre d'octets restant à envoyer
IS_SAFEDATA_CORRECT (frame, expectedData, bytesSent)	Cette macro vérifie si les SafeData de la PDU d'Esclave de Sécurité reçue correspondent aux données attendues Paramètres: Trame = trame de réception expectedData = Référence aux données attendues bytesSent = Nombre d'octets envoyés
START_WD(watchdog)	Cette macro réinitialise le chien de garde et lance un temporisateur de surveillance Paramètres: chien de garde = période de surveillance en ms
CREATE_SESSION_ID	Cette macro génère un ID de session aléatoire La variable SecondSessionFrameSent est réinitialisée à "FAUX"
ADR	Cette macro génère une référence (pointeur) à une variable

7.4.2 Etat Reset (Réinitialisation)

7.4.2.1 Evénement Trame reçue

Transition	Condition	Action	Etat suivant

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;`
			Réinitialisation

7.4.2.2 Evénement Expiration du chien de garde

Transition	Condition	Action	Etat suivant
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 Evénement Réinitialiser la Connexion

Transition	Condition	Action	Etat suivant
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);	Réinitialisation

7.4.2.4 Evénement Etablir la Commande de Données

Transition	Condition	Action	Etat suivant
RESET_STAY2		DataCommand:= DataCmd;	Réinitialisation

7.4.3 Etat Session

7.4.3.1 Evénement Trame reçue

Transition	Condition	Action	Etat suivant
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);	Connexion

Transition	Condition	Action	Etat suivant
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);	Réinitialisation
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	Etat suivant
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);	Réinitialisation
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);	Réinitialisation

7.4.3.2 Evénement Expiration du chien de garde

Transition	Condition	Action	Etat suivant
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);	Réinitialisation

7.4.3.3 Evénement Réinitialiser la Connexion

Transition	Condition	Action	Etat suivant
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);	Réinitialisation

7.4.3.4 Evénement Etablir la Commande de Données

Transition	Condition	Action	Etat suivant
SESSION_STAY2		DataCommand:= DataCmd;	Session

7.4.4 Etat Connexion

7.4.4.1 Evénement Trame reçue

Transition	Condition	Action	Etat suivant
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);	Paramètre
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);	Réinitialisation

Transition	Condition	Action	Etat suivant
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);	Réinitialisation
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);	Réinitialisation
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);	Connexion

Transition	Condition	Action	Etat suivant
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);	Réinitialisation
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);	Réinitialisation

7.4.4.2 Evénement Expiration du chien de garde

Transition	Condition	Action	Etat suivant
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);	Réinitialisation

7.4.4.3 Evénement Réinitialiser la Connexion

Transition	Condition	Action	Etat suivant
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);	Réinitialisation

7.4.4.4 Evénement Etablir la Commande de Données

Transition	Condition	Action	Etat suivant
CONN_STAY2		DataCommand:= DataCmd;	Connexion

7.4.5 Etat Paramètre

7.4.5.1 Evénement Trame reçue

Transition	Condition	Action	Etat suivant
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);	Données
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);	Réinitialisation
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);	Réinitialisation

Transition	Condition	Action	Etat suivant
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);	Réinitialisation
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);	Paramètre
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	Etat suivant
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);	Réinitialisation
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);	Réinitialisation

7.4.5.2 Evénement Expiration du chien de garde

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);	Réinitialisation

7.4.5.3 Evénement Réinitialiser la Connexion

Transition	Condition	Action	Etat suivant
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);	Réinitialisation

7.4.5.4 Evénement Etablir la Commande de Données

Transition	Condition	Action	Etat suivant
PARA_STAY2		DataCommand:= DataCmd;	Paramètre

7.4.6 Etat Données

7.4.6.1 Evénement Trame reçue

Transition	Condition	Action	Etat suivant
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);	Données
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);	Données

Transition	Condition	Action	Etat suivant
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);	Réinitialisation
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);	Réinitialisation
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	Etat suivant
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);	Réinitialisation
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);	Réinitialisation

7.4.6.2 Evénement Expiration du chien de garde

Transition	Condition	Action	Etat suivant
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);	Réinitialisation

7.4.6.3 Evénement Réinitialiser la Connexion

Transition	Condition	Action	Etat suivant
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);	Réinitialisation

7.4.6.4 Evénement Etablir la Commande de Données

Transition	Condition	Action	Etat suivant
DATA_STAY		DataCommand:= DataCmd;	Données

7.5 Table d'états pour Esclave FSoE

7.5.1 Diagramme d'état d'Esclave FSoE

7.5.1.1 Vue d'ensemble

En fonction de la procédure de communication, l'Esclave FSoE peut prendre les états énumérés dans le Tableau 34.

Tableau 34 – Etats de l'Esclave FSoE

Etat	Description
Reset (Réinitialisation)	La connexion FSoE est réinitialisée (les sorties sont en état de sécurité)
Session	L'ID de session est en cours de transfert (les sorties sont en état de sécurité)
Connection (Connexion)	L'ID de connexion est en cours de transfert (les sorties sont en état de sécurité)
Parameter (Paramètre)	Les paramètres sont en cours de transfert (les sorties sont en état de sécurité)
Data (Données)	Les données de processus ou les données de sécurité intégrée sont en cours de transfert (les sorties ne sont actives que si la commande *ProcessData* est reçue)

Le diagramme d'état de l'Esclave FSoE est illustré en Figure 10.

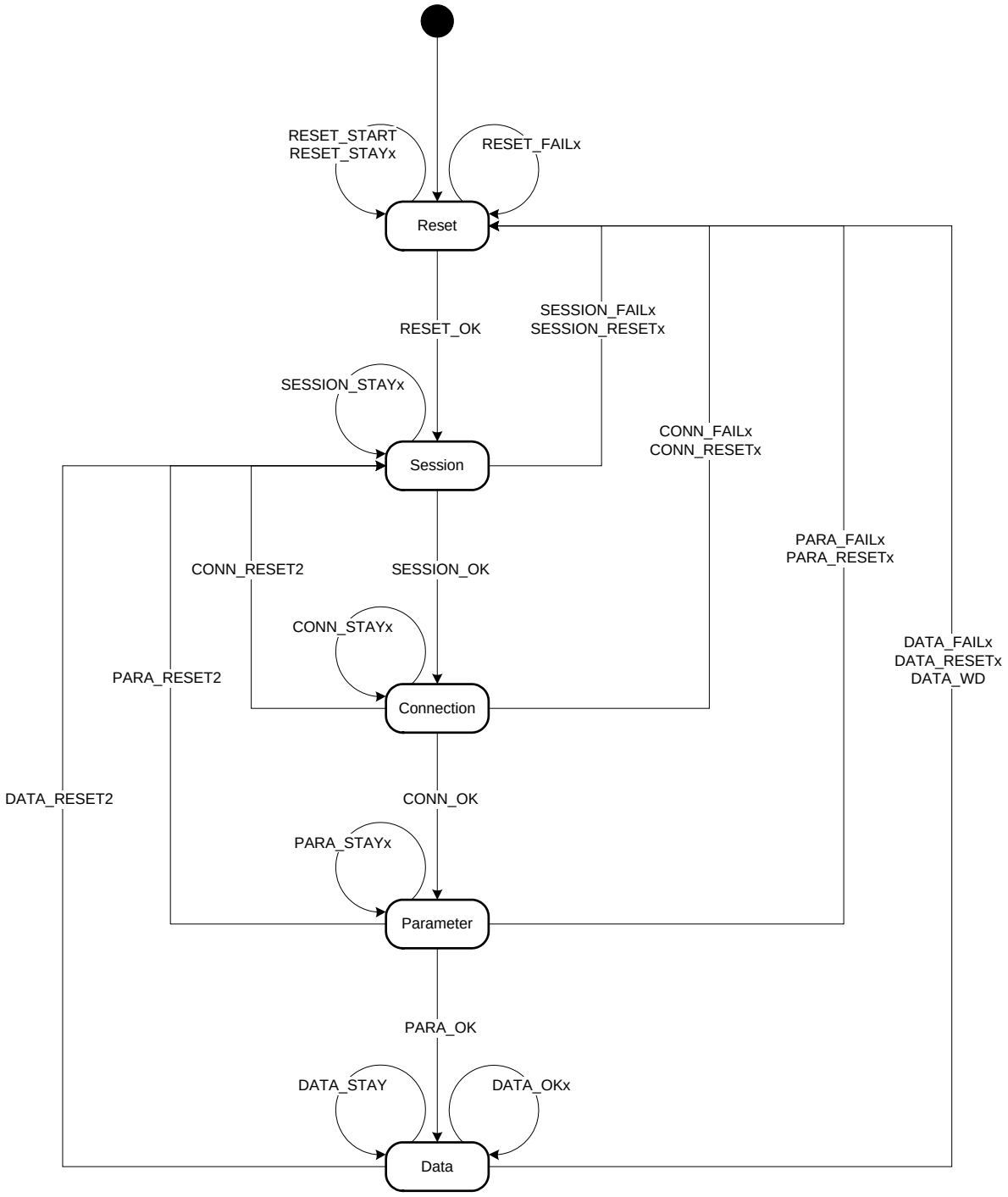


Figure 10 – Diagramme d'état de l'Esclave FSoE

Les paragraphes ci-après analysent, pour chaque état, les événements qui peuvent affecter l'Esclave FSoE. Chaque événement est pris en compte sous certaines conditions comportant différentes actions ou états ultérieurs.

7.5.1.2 Evénements

Un événement peut inclure différents paramètres auxquels il est fait référence dans les Tables d'états. Le Tableau 35 énumère les événements considérés.

Tableau 35 – Evénements dans la table d'états d'Esclave FSoE

Evénement	Description
trame reçue	Une PDU de sécurité a été reçue, ce qui signifie qu'au moins un bit dans la PDU a changé Paramètres: Trame = PDU de sécurité reçue Frame.Command = commande de la PDU de sécurité reçue Frame.Crc0 = CRC_0 de la PDU de sécurité reçue Frame.ConnId = ID de connexion de la PDU de sécurité reçue Frame.SafeData = données de sécurité de la PDU de sécurité reçue
expiration du chien de garde	Le chien de garde FSoE a expiré, ce qui signifie qu'aucune nouvelle PDU de sécurité n'a été reçue dans le délai imparti du chien de garde Paramètres: Aucun
Réinitialiser la Connexion	Demande, via une interface locale, de réinitialisation de la Connexion FSoE. Paramètres: Aucun
Etablir la Commande de Données	Demande, via une interface locale, de passage des SafeInputs à l'état de sécurité ou de sortie de l'état de sécurité Paramètres: DataCmd: FailSafeData ou ProcessData

7.5.1.3 Actions

En fonction des différentes conditions, certaines actions sont réalisées si un événement a lieu. Dans les tables d'états, les actions sont présentées comme des invocations de fonctions ou des attributions de variables.

Le Tableau 36 énumère les fonctions utilisées dans la table d'états d'Esclave FSoE.

Tableau 36 – Fonctions dans la table d'états d'Esclave FSoE

Fonction	Description
SendFrame(cmd, safeData, lastCrc, connId, seqNo, oldCrc, bNew)	Une PDU d'Esclave de Sécurité est envoyée Paramètres: cmd = commande de trame SafeData = référence aux données de sécurité envoyées avec la trame lastCrc = CRC_0 de la dernière PDU de Maître de Sécurité incluse dans le calcul du CRC pour la trame connId = ID de connexion à insérer dans la trame et inclus dans le calcul du CRC seqNo = Pointeur vers le numéro de séquence Esclave inclus dans le calcul du CRC pour les trames. Le seqNo incrémenté (peut-être plusieurs fois) est renvoyé oldCrc: Pointeur vers le CRC_0 de la dernière PDU d'Esclave de Sécurité envoyée. Le CRC_0 calculé est renvoyé bNew: Si bNew = VRAI et oldCrc est égal au CRC calculé, le calcul du CRC est recommencé avec le seqNo incrémenté jusqu'à ce que le CRC calculé ne soit plus égal à oldCrc (procédure conformément au 7.1.3.4)

Le Tableau 37 énumère les variables utilisées dans la table d'états d'Esclave FSoE.

Tableau 37 – Variables dans la table d'états d'Esclave FSoE

Variable	Description
LastCrc	CRC_0 de la dernière PDU d'Esclave de sécurité (initialisé à zéro à la mise sous tension)
OldMasterCrc	CRC_0 de la dernière PDU de Maître de Sécurité reçue (initialisé à zéro à la mise sous tension)
OldSlaveCrc	CRC_0 de la dernière PDU d'Esclave de sécurité envoyée (initialisé à zéro à la mise sous tension)
MasterSeqNo	Numéro de séquence Maître attendu, à utiliser dans le CRC pour la PDU de Maître de Sécurité suivante (initialisé à zéro à la mise sous tension)
SlaveSeqNo	Numéro de séquence Esclave à utiliser dans le CRC pour la PDU d'Esclave de Sécurité suivante (initialisé à zéro à la mise sous tension)
InitSeqNo	Variable contenant le numéro de séquence d'initialisation 1
DataCommand	Indique si la commande *ProcessData* ou *FailSafeData* est envoyée dans l'état Données. Initialisée avec *FailSafeData* à la mise sous tension
BytesToBeSent	Si plusieurs PDU de sécurité doivent être envoyées dans l'état Session, Connexion ou Paramètre, cette variable indique le nombre d'octets qui restent à envoyer (initialisée à zéro à la mise sous tension)
ConnectionId	Dans l'état Connexion, le ConnectionID est reçu par le Maître FSoE (initialisé à zéro à la mise sous tension)
ConnectionData	Dans l'état Connexion, les ConnectionData sont reçues par le Maître FSoE (initialisées à zéro à la mise sous tension)
SlaveAddress	L'Adresse d'Esclave FSoE est initialisée via une interface locale à la mise sous tension (généralement un commutateur d'adresses externe)
SafePara	Les SafePara sont reçus par le Maître FSoE dans l'état Paramètre et initialisés en fonction du dispositif SafePara.Watchdog: chien de garde FSoE (initialisé à zéro à la mise sous tension)
ExpectedSafeParaSize	Indique la longueur de SafePara attendue
SafeOutputs	Contient les valeurs de processus des sorties de sécurité reçues par le Maître FSoE. Initialisé à la valeur FS_VALUE (Données de sécurité intégrée = 0) à la mise sous tension
SafeInputs	Contient les valeurs de processus des entrées de sécurité envoyées au Maître FSoE. Initialisé à la valeur FS_VALUE (Données de sécurité intégrée = 0) à la mise sous tension
CommFaultReason	Indique le code d'erreur en cas d'erreur de communication

7.5.1.4 Macros

Certaines fonctionnalités sont consolidées dans des macros pour préserver la transparence des Tables d'états.

Le Tableau 38 énumère les macros utilisées dans la table d'états d'Esclave FSoE.

Tableau 38 – Macros dans la table d'états d'Esclave FSoE

Macro	Description
IS_CRC_CORRECT(frame, lastCrc, seqNo, oldCrc, bNew)	Cette macro vérifie si les CRC de la PDU de Maître de Sécurité reçue sont corrects Paramètres: trame: PDU reçue lastCrc: CRC_0 de la dernière PDU d'Esclave de Sécurité envoyée, incluse dans les calculs du CRC pour la trame reçue

Macro	Description
	seqNo: Pointeur vers le numéro de séquence Maître utilisé dans les calculs du CRC pour la trame reçue. Le seqNo incrémenté (peut-être plusieurs fois) est renvoyé oldCrc: Pointeur vers le CRC_0 de la dernière PDU de Maître de Sécurité reçue. Le CRC_0 du télégramme reçu est renvoyé bNew: Si bNew = VRAI et oldCrc est égal au CRC calculé, le calcul du CRC est recommencé avec le seqNo incrémenté jusqu'à ce que le CRC calculé ne soit plus égal à oldCrc (procédure conformément au 7.1.3.4)
UPDATE_BYTES_TO_BE_SENT(bytesSent)	Cette macro vérifie le nombre d'octets qui restent à envoyer dans les états Session, Connexion et Paramètre avant que l'état ne soit modifié Paramètres: bytesSent: Nombre d'octets restant à envoyer
IS_SAFE_PARA_CORRECT(safePara)	Cette macro vérifie si les SafePara reçus dans l'état Paramètre sont corrects Paramètres: safePara: Pointeur vers SafePara reçus
STORE_DATA(dst, src)	Cette macro conserve les données de sécurité reçues d'une PDU de sécurité Paramètres: dst: Pointeur vers l'adresse de destination src: Pointeur vers l'adresse d'origine
GET_PARA_FAULT()	Cette macro renvoie un code d'erreur si les SafePara s'avèrent non valides
START_WD(watchdog)	Cette macro réinitialise le chien de garde et lance un temporisateur de surveillance avec le chien de garde spécifié. Paramètres: chien de garde: période de surveillance en ms
STOP_WD()	Cette macro arrête le temporisateur de surveillance et réinitialise le chien de garde
ADR	Cette macro génère une référence (pointeur) à une variable

7.5.2 Etat Réinitialisation

7.5.2.1 Evénement Trame reçue

Transition	Condition	Action	Etat suivant

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

Transition	Condition	Action	Etat suivant
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;	Réinitialisation
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;	Réinitialisation
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;	Réinitialisation
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;	Réinitialisation

7.5.2.2 Evénement Expiration du chien de garde

Ne peut pas avoir lieu dans cet état car le chien de garde n'a pas encore démarré.

7.5.2.3 Evénement Réinitialiser la Connexion

Transition	Condition	Action	Etat suivant
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;	Réinitialisation

7.5.2.4 Evénement Etablir la Commande de Données

Transition	Condition	Action	Etat suivant
RESET_STAY2		DataCommand:= DataCmd;	Réinitialisation

7.5.3 Etat Session

7.5.3.1 Evénement Trame reçue

Transition	Condition	Action	Etat suivant
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);	Connexion
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;	Réinitialisation

Transition	Condition	Action	Etat suivant
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;	Réinitialisation
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;	Réinitialisation
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	Etat suivant
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;	Réinitialisation
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;	Réinitialisation
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;	Réinitialisation
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;	Réinitialisation

Transition	Condition	Action	Etat suivant
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;	Réinitialisation
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;	Réinitialisation
a Les deux états SESSION_FAIL4 et SESSION_FAIL5 sont les deux seuls états pour lesquels il convient de calculer les deux CRC-checks. La seule différence est une CommFaultReason différente, c'est-à-dire uniquement des informations de diagnostic non sécuritaires. Il est admis de réduire ces états à un seul état; dans cet état, seule la condition "IS_CRC_CORRECT(Frame, 0, ADR(InitSeqNo), ADR(OldMasterCrc), FALSE) = FALSE" doit être vérifiée avec CommFaultReason:= INVALID_CRC.			

7.5.3.2 Evénement Expiration du chien de garde

Ne peut pas avoir lieu dans cet état car le chien de garde n'a pas encore démarré.

7.5.3.3 Evénement Réinitialiser la Connexion

Transition	Condition	Action	Etat suivant
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;	Réinitialisation

7.5.3.4 Evénement Etablir la Commande de Données

Transition	Condition	Action	Etat suivant
SESSION_STAY3		DataCommand:= DataCmd;	Session

7.5.4 Etat Connexion

7.5.4.1 Evénement Trame reçue

Transition	Condition	Action	Etat suivant
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);	Paramètre
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;	Réinitialisation
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;	Réinitialisation

Transition	Condition	Action	Etat suivant
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;	Réinitialisation
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;	Réinitialisation
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);	Connexion
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;	Réinitialisation

Transition	Condition	Action	Etat suivant
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;	Réinitialisation
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;	Réinitialisation
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;	Réinitialisation

Transition	Condition	Action	Etat suivant
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;	Réinitialisation
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;	Réinitialisation

Transition	Condition	Action	Etat suivant
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;	Réinitialisation
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;	Réinitialisation

7.5.4.2 Evénement Expiration du chien de garde

Ne peut pas avoir lieu dans cet état car le chien de garde n'a pas encore démarré.

7.5.4.3 Evénement Réinitialiser la Connexion

Transition	Condition	Action	Etat suivant
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;	Réinitialisation

7.5.4.4 Evénement Etablir la Commande de Données

Transition	Condition	Action	Etat suivant
CONN_STAY2		DataCommand:= DataCmd;	Connexion

7.5.5 Etat Paramètre

7.5.5.1 Evénement Trame reçue

Transition	Condition	Action	Etat suivant
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);	Données
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);	Données
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;	Réinitialisation

Transition	Condition	Action	Etat suivant
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;	Réinitialisation
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;	Réinitialisation
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;	Réinitialisation
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);	Paramètre

Transition	Condition	Action	Etat suivant
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;	Réinitialisation
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;	Réinitialisation
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;	Réinitialisation
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;	Réinitialisation

Transition	Condition	Action	Etat suivant
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;	Réinitialisation
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;	Réinitialisation
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;	Réinitialisation

Transition	Condition	Action	Etat suivant
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;	Réinitialisation

7.5.5.2 Evénement Expiration du chien de garde

Ne peut pas avoir lieu dans cet état car le chien de garde n'a pas encore démarré.

7.5.5.3 Evénement Réinitialiser la Connexion

Transition	Condition	Action	Etat suivant
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;	Réinitialisation

7.5.5.4 Evénement Etablir la Commande de Données

Transition	Condition	Action	Etat suivant
PARA_STAY2		DataCommand:= DataCmd;	Paramètre

7.5.6 Etat Données

7.5.6.1 Evénement Trame reçue

Transition	Condition	Action	Etat suivant
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);	Données
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);	Données
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;	Réinitialisation
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;	Réinitialisation

Transition	Condition	Action	Etat suivant
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;	Réinitialisation
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;	Réinitialisation
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	Etat suivant
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;	Réinitialisation
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;	Réinitialisation
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;	Réinitialisation

7.5.6.2 Evénement Expiration du chien de garde

Transition	Condition	Action	Etat suivant
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;	Réinitialisation

7.5.6.3 Evénement Réinitialiser la Connexion

Transition	Condition	Action	Etat suivant
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;	Réinitialisation

7.5.6.4 Evénement Etablir la Commande de Données

Transition	Condition	Action	Etat suivant
DATA_STAY		DataCommand:= DataCmd;	Données

8 Gestion de la couche de communication de sécurité

8.1 Traitement des paramètres FSCP 12/1

Le protocole FSCP 12/1 prend en charge un téléchargement inhérent du paramètre de l'Esclave FSoE à partir du Maître FSoE dans l'état Paramètre.

8.2 Paramètres de communication FSoE

La communication FSoE entre le Maître FSoE et l'Esclave FSoE utilise les paramètres de communication FSoE définis dans le Tableau 39.

Tableau 39 – Paramètres de communication FSoE

Nom	Type de Données	Domaine	Description
ID de session FSoE	UINT16	0 à 2^{16}	ID de session FSoE généré de manière aléatoire
ID de Connexion FSoE	UINT16	1 à 2^{16}	Identifiant unique de connexion entre Maître FSoE et Esclave FSoE
Numéro de séquence FSoE	UINT16	Init: 0 1 à 2^{16}	Incrémenté à chaque Cycle FSoE
Adresse d'Esclave FSoE	UINT16	1 à 2^{16}	Adresse unique d'Esclave FSoE pour chaque dispositif FSoE
Période (délai imparti) du chien de garde FSoE	UINT16	1 à 2^{16}	Délai de chien de garde pour la Connexion FSoE, en ms

9 Exigences relatives au système

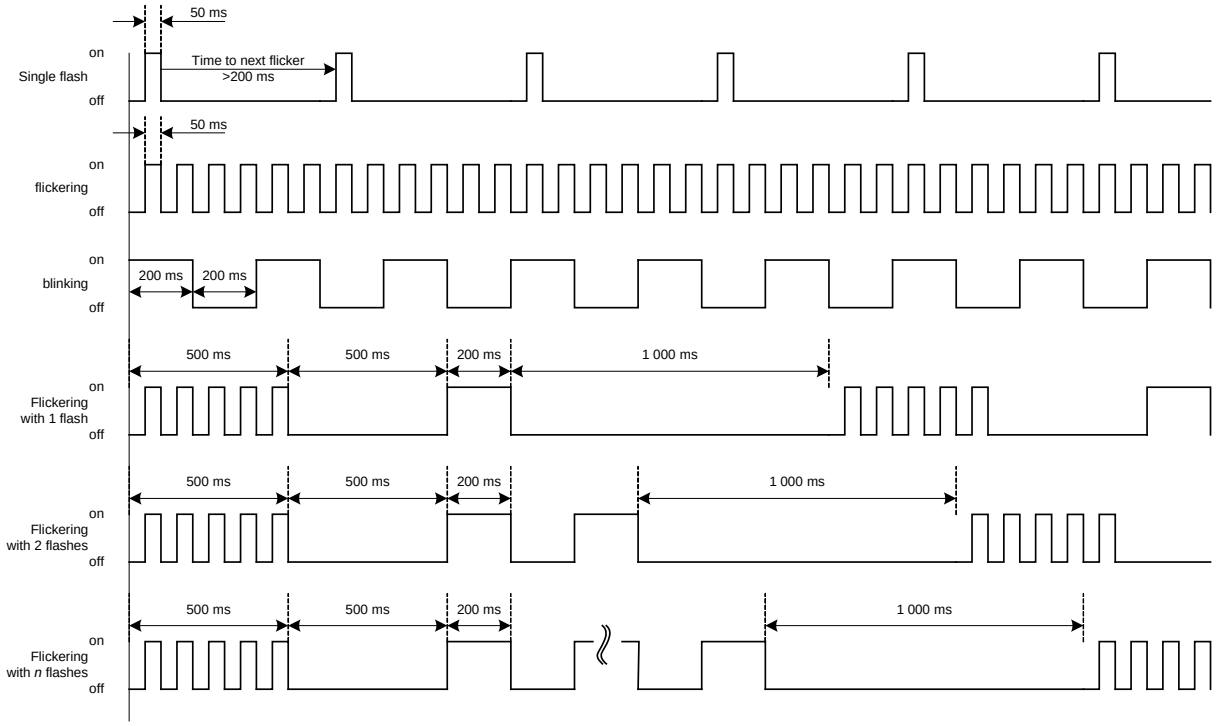
9.1 Voyants et commutateurs

9.1.1 Etats des voyants et fréquences de clignotement

Les états des voyants et les fréquences de clignotement sont définis dans le Tableau 40 et illustrés en Figure 11. Les durées indiquées doivent être respectées avec une tolérance inférieure à +/- 20%.

Tableau 40 – Etats des voyants

Etat du voyant	Définition
Allumé	Le voyant doit être allumé de manière constante
Eteint	Le voyant doit être éteint de manière constante
un seul clignotement	Le voyant doit présenter un scintillement court (50 ms) suivi d'une phase d'extinction d'au moins 200 ms
scintillement	Le voyant doit s'allumer et s'éteindre en isophase à une fréquence de 10 Hz: allumage pendant 50 ms et extinction pendant 50 ms
papillotement	Le voyant doit s'allumer et s'éteindre en isophase à une fréquence de 2,5 Hz: allumage pendant 200 ms suivi d'une extinction pendant 200 ms
scintillement avec 1 clignotement	Le voyant doit présenter un premier scintillement pendant 500 ms puis une phase d'extinction (500 ms), puis un clignotement court (200 ms) suivi d'une phase d'extinction longue (1000 ms)
scintillement avec 2 clignotements	Le voyant doit présenter un premier scintillement pendant 500 ms puis une phase d'extinction (500 ms), puis une suite de deux clignotements courts (200 ms), séparés par une phase d'extinction (200 ms), suivie d'une phase d'extinction longue (1000 ms)
scintillement avec *n* clignotements	Le voyant doit présenter un premier scintillement pendant 500 ms puis une phase d'extinction (500 ms), puis une suite de *n* clignotements courts (200 ms), séparés par une phase d'extinction (200 ms), suivie d'une phase d'extinction longue (1000 ms)



Légende

Anglais	Français
Single flash	Un seul clignotement
Time to next flicker	
Flickering	scintillement
blinking	papillotement
Flickering with 1 flash	Scintillement avec 1 clignotement
Flickering with 2 flashes	Scintillement avec 2 clignotements
Flickering with *n* flashes	Scintillement avec *n* clignotements

Figure 11 – Fréquences de clignotement des voyants

9.1.2 Voyants

9.1.2.1 Voyants exigés

Il convient que les dispositifs prenant en charge le protocole FSCP 12/1 disposent de voyants d'état pour l'examen visuel et le dépannage de la Connexion FSoE. Lorsqu'un dispositif prend en charge l'un des voyants décrits dans la présente norme, ces derniers doivent être conformes aux spécifications.

D'autres voyants peuvent être mis en œuvre.

9.1.2.2 Voyant STATUS (d'ETAT) FSoE

Le voyant STATUS FSoE doit indiquer l'état de la Connexion FSoE. Il doit être de couleur verte.

Pour satisfaire à des contraintes d'espace, il est admis de ne pas utiliser d'étiquette pour le voyant STATUS FSoE. Si le voyant est étiqueté, il doit utiliser l'une des indications suivantes (la casse n'a aucune importance):

- FS

- FSoE
- Status FSoE

Les états du voyant STATUS FSoE sont spécifiés dans le Tableau 41.

Tableau 41 – Etats du voyant STATUS FSoE

Etat du voyant	Définition	Etat FSoE
Eteint	En cours d'initialisation	Pré-réinitialisation
papillotement	Prêt pour le paramétrage	Réinitialisation, Session, Connexion, Paramètre
Allumé	Fonctionnement normal	Données de processus
un seul clignotement	Données de Sécurité intégrée	Données de sécurité intégrée
scintillement	Erreur non spécifiée dans la Connexion FSoE	Tous
scintillement avec 1 clignotement	Erreur de paramètre F	Paramètre
scintillement avec 2 clignotements	Erreur de Paramètre d'Application	Paramètre
scintillement avec 3 clignotements	Adresse de sécurité incorrecte	Connexion
scintillement avec 4 clignotements	Commande incorrecte	Tous
scintillement avec 5 clignotements	Erreur de chien de garde	Tous
scintillement avec 6 clignotements	Erreur de CRC	Tous

L'indication d'une erreur (scintillement) doit persister jusqu'à ce que la Connexion FSoE passe de Réinitialisation à Session.

Il doit être présenté au moins une séquence entière de scintillement avec *n* clignotements.

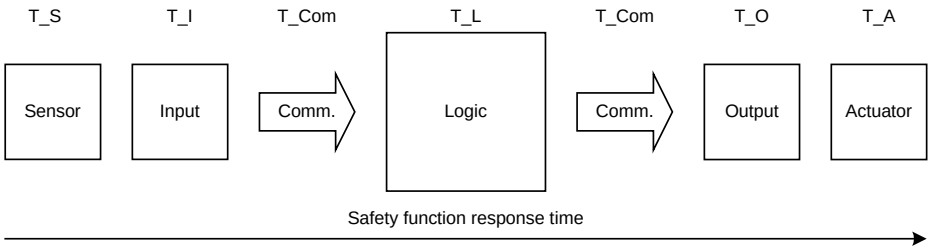
9.2 Recommandations d'installation

La présente partie de la norme spécifie un protocole et des services pour un système de communication de sécurité fondé sur le Type 12 de la CEI 61158. Cependant, l'utilisation de dispositifs de sécurité avec le protocole de sécurité spécifié dans la présente partie exige une installation correcte. Tous les dispositifs connectés à un système de communication de sécurité défini dans la présente partie doivent satisfaire aux exigences TBTS/TBTP spécifiées dans les normes CEI pertinentes, telles que la CEI 60204-1. La CEI 61918 donne d'autres recommandations d'installation pertinentes.

9.3 Temps de réponse de la fonction de sécurité

9.3.1 Généralités

Pour pouvoir déterminer le temps de réponse de la fonction de sécurité, cette dernière est décomposée en plusieurs composantes comme illustré en Figure 12.



Légende

Anglais	Français
Sensor	Capteur
Input	Entrée
Logic	Logique
Output	Sortie
Actuator	Actionneur
Safety function response time	Temps de réponse de la fonction de sécurité

Figure 12 – Composantes d'une fonction de sécurité

Il n'est pas nécessaire que toutes les composantes soient présentes dans un système donné. Le capteur (par exemple un rideau de lumière ou un bouton d'arrêt d'urgence) convertit le signal physique en signal électrique. Ce signal peut être connecté à un dispositif d'entrée (par exemple une entrée de sécurité) qui convertit le signal électrique en information logique. Le réseau de communication de sécurité permet de transmettre cette information logique à la logique de sécurité. La logique de sécurité (par exemple un automate programmable de sécurité) combine cette information logique et/ou d'autres informations d'entrée en informations logiques de sortie. Les informations de sortie sont transmises au dispositif de sortie (par exemple une sortie de sécurité) et converties en signal électrique. Ce signal est connecté à l'actionneur qui réagit physiquement, par exemple en interrompant l'alimentation d'un entraînement.

Les hypothèses à caractère général relatives aux erreurs de communication sont les suivantes:

- Toutes les composantes fonctionnent de manière asynchrone.
- Le traitement des signaux / informations d'entrée est indépendant du traitement des signaux / informations de sortie. Ceci veut dire que chaque côté peut avoir son propre comportement temporel.
- Pour calculer le temps de réponse de la fonction de sécurité, on doit supposer, pour l'ensemble du système, une seule erreur ou défaillance. Cette erreur ou défaillance doit être supposée avoir lieu dans la partie du trajet du signal qui contribue à la différence temporelle maximale entre son temps de retard le plus défavorable et le délai imparti par son chien de garde. Ceci veut dire que des défaillances simultanées ne sont pas prises en compte.

Le Tableau 42 définit les temps des composantes.

Tableau 42 – Définition des temps

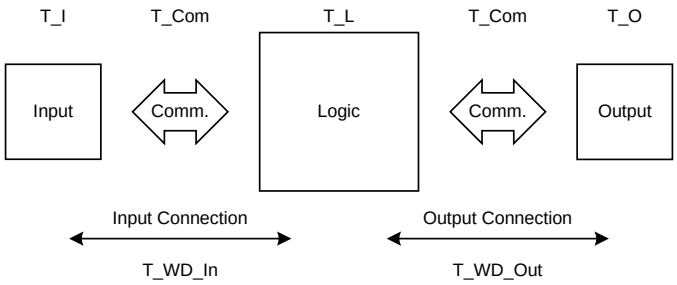
Temps	Nom	Description
T_SFR	Temps de réponse de la fonction de sécurité	Temps de réponse de la fonction de sécurité, du signal d'entrée physique à la réaction de l'actionneur
T_InCon	Temps de connexion d'entrée	Temps de transmission du signal d'entrée physique à la logique de sécurité
T_OutCon	Temps de connexion de sortie	Temps de transmission du signal de sortie calculé, de la logique

Temps	Nom	Description
		de sécurité à l'actionneur
T_S	Temps de détection (capteur)	Temps de conversion du capteur de sécurité
T_I	Temps d'entrée	Temps de retard du dispositif d'entrée de sécurité
T_Com	Temps de communication	Durée de cycle de transmission du réseau de communication
T_L	Temps logique	Temps de retard de la logique (cycle)
T_O	Temps de sortie	Temps de retard du dispositif de sortie de sécurité
T_A	Temps d'activation (actionneur)	Temps de conversion de l'actionneur de sécurité
T_WD_In	Temps du chien de garde d'entrée	Délai du chien de garde FSoE de la connexion d'entrée
T_WD_Out	Temps du chien de garde de sortie	Délai du chien de garde FSoE de la connexion de sortie
ΔT	Marge du chien de garde	Marge supplémentaire appliquée au temps minimal du chien de garde

Sachant que toutes les composantes sont supposées fonctionner de manière asynchrone, le temps le plus défavorable pour chaque composante est égal à deux fois le retard de la composante. C'est le cas si les informations de fonctionnement sont disponibles juste après que le processus ait commencé. Les temps les plus défavorables sont marqués du suffixe a *_wc*.

9.3.2 Détermination du temps du chien de garde FSoE

La Figure 13 illustre le schéma de base pour le calcul du délai imparti du chien de garde FSoE pour la connexion d'entrée et de sortie.



Légende

Anglais	Français
Input	Entrée
Logic	Logique
Output	Sortie
Input connection	Connexion d'entrée
Output connection	Connexion de sortie

Figure 13 – Calcul des temps de chien de garde FsoE pour les connexions d'entrée et de sortie

L'Equation (1) peut être utilisée pour la détermination du temps de chien de garde de connexion d'entrée T_WD_In:

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

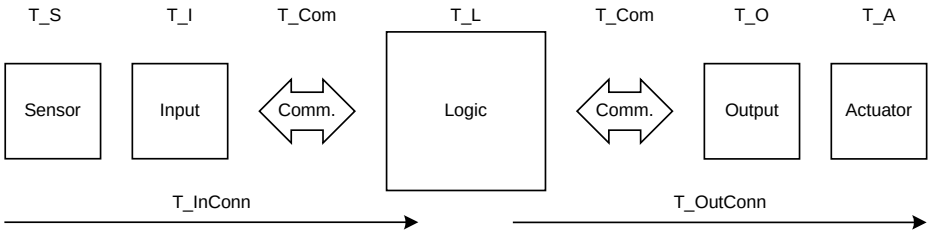
$$= 2 \times T\_I + 4 \times T\_Com + 2 \times T\_L + \Delta T$$

L'Equation (2) calcule, par analogie, le temps du chien de garde de connexion de sortie 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 Calcul du temps de réponse de la fonction de sécurité le plus défavorable

La Figure 14 illustre le schéma de base pour le calcul du temps de réponse le plus défavorable de la fonction de sécurité.



Légende

Anglais	Français
Input	Entrée
Logic	Logique
Output	Sortie
Sensor	Capteur
Actuator	actionneur

Figure 14 – Calcul du temps de réponse le plus défavorable de la fonction de sécurité

Le temps nécessaire à la transmission des informations du signal du capteur à la logique de sécurité, T_InConn, peut être calculé de la manière suivante:

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

Le temps le plus défavorable pour obtenir les informations d'état de sécurité du signal du capteur à la logique de sécurité, T_InConn_wc, est obtenu lorsque la communication d'entrée est interrompue et que le délai du chien de garde de connexion d'entrée expire. Dans ce cas, les valeurs de sécurité intégrée des signaux d'entrée sont utilisées dans la logique de sécurité. Ce temps peut être calculé de la manière suivante:

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

Le temps nécessaire pour obtenir le signal de sortie calculé de la logique de sécurité à l'actionneur, T_OutConn, peut être calculé de la manière suivante:

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

Le temps le plus défavorable pour obtenir le signal de sortie calculé de la logique de sécurité à l'actionneur, T_OutConn_wc, est obtenu lorsque la communication de sortie est interrompue

et que le délai du chien de garde de connexion de sortie expire. Dans ce cas, les valeurs de sécurité intrinsèque sont activées dans le dispositif de sortie. Ce temps peut être calculé de la manière suivante:

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

Pour le calcul du temps de réponse de la fonction de sécurité, on doit supposer que le trajet du signal comporte une erreur ou une défaillance qui contribue au différentiel de temps maximal entre son temps de retard le plus défavorable et le délai de chien de garde.

L'Equation (7) peut être utilisée pour déterminer le temps de réponse de la fonction de sécurité le plus défovrable T_SFR_wc:

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

Les fabricants de systèmes doivent fournir, si nécessaire, leur propre méthode de calcul adaptée.

9.4 Durée des demandes

La durée de la demande émise par l'application sécuritaire à la couche de communication de sécurité peut être égale ou supérieure au temps de sécurité de processus ou au délai imparti du profil FSCP 12/1 (chien de garde FSoE).

9.5 Contraintes de calcul des caractéristiques du système

9.5.1 Généralités

Le FSCP 12/1 n'impose aucune restriction concernant:

- le temps minimal du cycle de transmission;
- le nombre de données de sécurité par dispositif FSoE;
- le système de communication sous-jacent.

Tous les dispositifs doivent assurer une sécurité électrique TBTS/TBTP.

Les dispositifs de sécurité sont conçus pour un environnement industriel normal, conformément à la CEI 61000-6-2 ou à la CEI 61131-2 ainsi que pour une immunité accrue, conformément à la CEI 61326-3-1 ou à la CEI 61326-3-2.

Le trajet de communication est arbitraire; il peut s'agir d'un système de bus de terrain, de l'Ethernet ou de trajets similaires, de câbles à fibre optique, de câbles en cuivre, voire de liaisons radioélectriques. Il n'existe aucune restriction ou exigence concernant le coupleur ou autres dispositifs du trajet de communication.

L'insertion supplémentaire de trois octets à zéro dans le calcul du CRC ainsi que l'héritage du CRC, garantissent l'indépendance de la transmission sous-jacente même si le même polynôme CRC est utilisé.

L'interface de communication des dispositifs de sécurité peut être une interface à un seul canal. Il peut s'agir d'une interface redondante pour assurer une meilleure disponibilité.

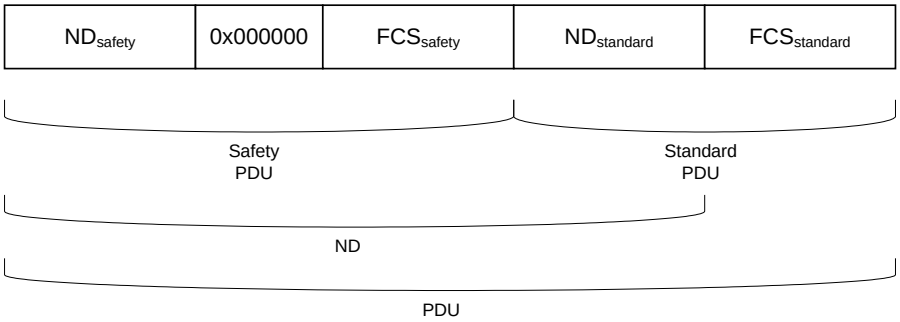
La connexion entre les dispositifs FSoE est une Connexion Maître-Esclave. Le Maître FSoE dispose d'une ou de plusieurs Connexions FSoE à un ou plusieurs Esclaves FSoE. L'Esclave FSoE ne réagit qu'au Maître FSoE. Il peut être établi dans un système jusqu'à 65 535 Connexions FSoE.

9.5.2 Considérations d'ordre probabiliste

Chaque erreur détectée dans la communication de sécurité doit lancer une transition vers l'état de réinitialisation, c'est-à-dire un état de sécurité. Cette transition ne doit avoir lieu plus d'une fois en 5 heures, c'est-à-dire que le taux d'erreurs résiduelles doit être meilleur que 10^{-2}/h.

Il est démontré que le polynôme CRC avec insertion de trois octets à zéro (appelés bits virtuels) garantit l'indépendance du contrôle normal sous-jacent.

La PDU de Type 12 est constituée d'une partie sécurité et d'une partie normale. La partie sécurité est intégrée dans la partie normale. La Figure 15 illustre la PDU, qui est constituée des données de sécurité SafetyData ND_{safety}, de bits virtuels de longueur d_{safety} = 24 bits, de la séquence de contrôle de sécurité FCS_{safety}, des données utiles normales $ND_{standard}$ et de la séquence de contrôle normale $FCS_{standard}$.



Légende

Anglais	Français
Safety PDU	PDU de sécurité
Standard PDU	PDU normale

Figure 15 – PDU de sécurité intégrée dans une PDU normale

Les exigences suivantes sont induites:

— $x^{dsafety}+1$ et le polynôme générateur sont premiers entre eux;

— le nombre de bits virtuels d_{safety} est inférieur ou égal au nombre de bits pour la partie normale ($d_{safety} \leq n_{standard}$);

— le taux d'erreurs résiduelles est inférieur à 10^{-9}/h.

Avec le polynôme de sécurité primitif 139B7h, ces exigences sont satisfaites dans les conditions suivantes:

— le nombre de bits de données de sécurité est de 8 ou 16 (ND_{safety} = 8 ou ND_{safety} = 16);

— le nombre de bits virtuels est de 24 (d_{safety} = 24);

— le nombre minimal de bits normalisés est de 16 ($ND_{standard} \geq 16$);

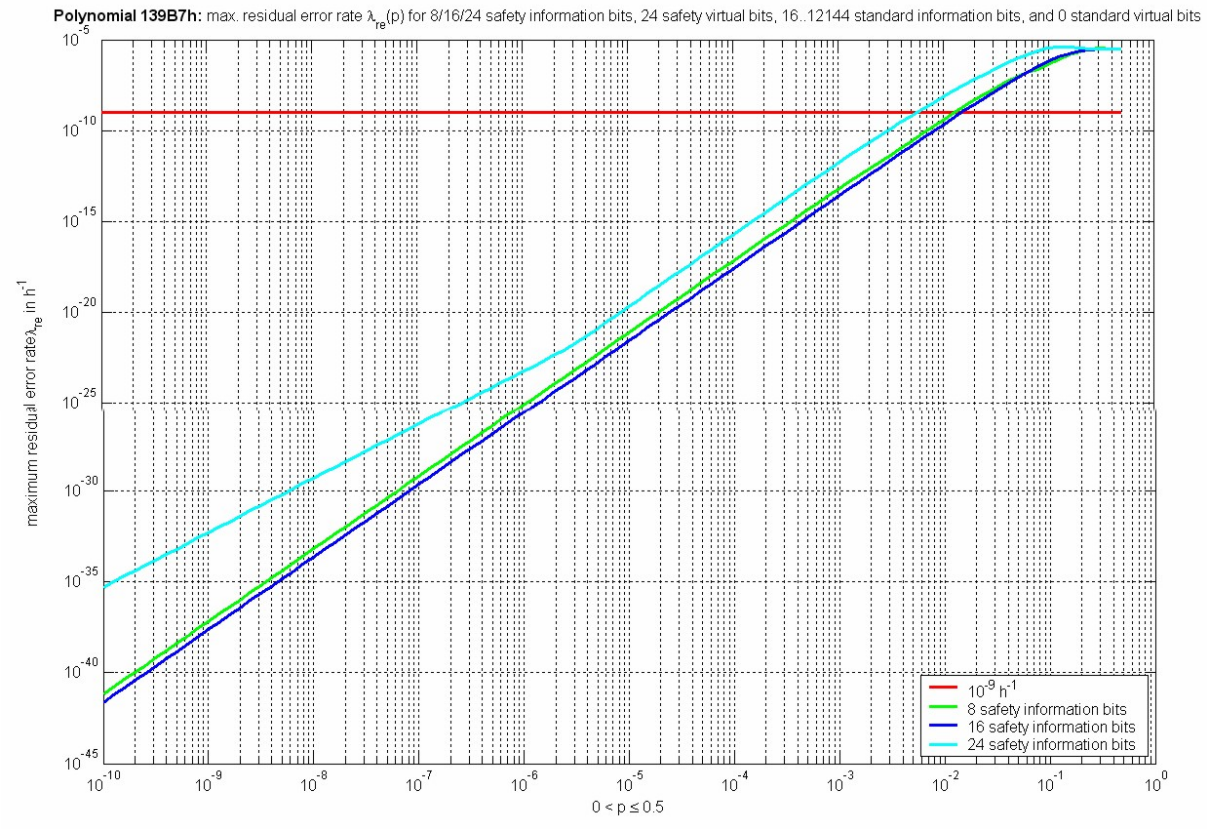
— le nombre maximal de bits normalisés est de 12 144 ($ND_{standard} \leq$ 12 144);

NOTE Il a été démontré qu'une longueur maximale de DPDU allant jusqu'à 12 144 bits (1 518 octets) pouvait être utilisée, comme pour l'Ethernet.

— les bits normalisés peuvent de nouveau contenir des blocs de données de sécurité, constitués de données de sécurité et de FCS_{safety}.

La Figure 16 illustre le taux d'erreurs résiduelles pour 8, 16 et 24 bits de données de sécurité. Avec une probabilité maximale d'erreur sur les bits de 10^{-2}/h, le taux d'erreurs résiduelles est

inférieur à 10^{-9}/h pour des données de sécurité de 8 et de 16 bits. Dans ce protocole, les données de sécurité de 24 bits ne sont pas utilisées.



Légende

Anglais	Français
Polynomial 139B7h: max. residual error rate λ_{re} for 8/16/24 safety information bits, 24 saftey virtual bits, 16....12144 standard information bits and 0 standard virtual bits.	Polynôme 139B7h: taux max. d'erreurs résiduelles λ_{re} pour 8/16/24 bits d'information de sécurité, 24 bits virtuels de sécurité, 16...12144 bits d'information normalisés et 0 bit virtuel normalisé
Maximum residual error rate	Taux maximal d'erreurs résiduelles
8 safety ...	8 Bits d'information de sécurité
16 safety ...	16 bits d'information de sécurité
24 safety ...	24 bits d'information de sécurité

Figure 16 – Taux d'erreurs résiduelles pour des données de sécurité de 8/16/24 bits et jusqu'à 12 144 bits de données normales

9.6 Maintenance

Ce protocole ne fait l'objet d'aucune exigence de maintenance particulière.

9.7 Manuel de sécurité

Les ingénieurs d'application de la présente partie doivent fournir un manuel de sécurité comportant au minimum les informations suivantes:

- Le manuel de sécurité doit informer les utilisateurs des contraintes de calcul des caractéristiques du système (voir 9.5).
- Le manuel de sécurité doit informer les utilisateurs de leurs responsabilités quant à un paramétrage correct du dispositif.

Outre les exigences du présent article, le manuel de sécurité doit satisfaire aux exigences définies dans la CEI 61508.

10 Evaluation

Il est fortement recommandé aux ingénieurs d'application du FSCP 12/1 d'obtenir, auprès d'un organisme d'évaluation compétent et indépendant, la vérification du protocole ainsi que de toute application, pour tous les aspects relatifs à la sécurité fonctionnelle du produit. Il est fortement recommandé aux ingénieurs d'application du FSCP 12/1 d'obtenir la preuve qu'un organisme d'évaluation compétent et indépendant a réalisé un essai de conformité approprié.

Annexe A
(informative)

Informations supplémentaires pour les profils de communication de sécurité fonctionnelle CPF 12

A.1 Calcul de la fonction de hachage

Le code d'une PDU de sécurité représenté ci-après donne un exemple de calcul des CRC pour la PDU de sécurité. Les trois octets de fin à zéro sont déjà pris en compte dans les tableaux.

```
************************************************************
**  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;
}
```

Les deux tables suivantes sont utilisées:

```
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 ...

Vide

Annexe B
(informative)

Information pour l'évaluation des profils de communication de sécurité fonctionnelle CPF 12

Des informations sur les laboratoires d'essai qui vérifient et valident la conformité des produits FSCP 12/1 avec la CEI 61784-3-12 peuvent être obtenues auprès des comités nationaux de la CEI ou de l'institution suivante:

EtherCAT Technology Group
Ostendstrasse 196
90482 Nuremberg
ALLEMAGNE

Téléphone: +49-911-54056-20
Télécopie: +49-911-54056-29
Courriel: info@ethercat.org
URL: www.ethercat.org

Bibliographie[16]

[1] CEI 60050 (toutes parties), *Vocabulaire Electrotechnique International*

NOTE Voir également le dictionnaire multilingue de la CEI – Electricité, électronique et télécommunications (disponible sur CD-ROM et à l'adresse <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 (disponible uniquement en anglais)*

[3] IEC 61131-6[17], *Programmable controllers – Part 6: Functional safety* (disponible uniquement en anglais)

[4] IEC 61158 (toutes parties), *Industrial communication networks – Fieldbus specifications* (disponible uniquement en anglais)

[5] CEI 61496 (toutes parties), *Sécurité des machines – Equipements de protection électro-sensibles*

[6] CEI 61508-1:2010[18], *Sécurité fonctionnelle des systèmes électriques / électroniques / électroniques programmables relatifs à la sécurité – Partie 1: Exigences générales*

[7] CEI 61508-4:2010[18], *Sécurité fonctionnelle des systèmes électriques / électroniques / électroniques programmables relatifs à la sécurité – Partie 4: Définitions et abréviations*

[8] CEI 61508-5:2010[18], *Sécurité fonctionnelle des systèmes électriques / électroniques / électroniques programmables relatifs à la sécurité – Partie 5: Exemples de méthodes de détermination des niveaux d'intégrité de sécurité*

[9] CEI 61511 (toutes parties), *Sécurité fonctionnelle – Systèmes instrumentés de sécurité pour le secteur des industries de transformation*

[10] IEC 61784-1, *Industrial communication networks – Profiles – Part 1 : Fieldbus profiles* (disponible uniquement en anglais)

[11] IEC 61784-4[19], *Industrial communication networks – Profiles – Part 4: Secure communications for fieldbuses* (disponible uniquement en anglais)

[12] IEC 61784-5 (toutes parties), *Industrial communication networks – Profiles – Part 5: Installation of fieldbuses – Installation profiles for CPF x* (disponible uniquement en anglais)

[13] IEC 61800-5-2, *Adjustable speed electrical power drive systems – Part 5-2: Safety requirements – Functional* (disponible uniquement en anglais)

[14] CEI/TR 62059-11, *Equipements de comptage de l'électricité – Sûreté de fonctionnement – Partie 11: Concepts généraux*

[15] CEI 62061, *Sécurité des machines – Sécurité fonctionnelle des systèmes de commande électriques, électroniques et électroniques programmables relatifs à la sécurité*

[16] IEC/TR 62210, *Power system control and associated communications – Data and communication security* (disponible uniquement en anglais)

[17] CEI 62280-1, *Applications ferroviaires – Systèmes de signalisation, de télécommunication et de traitement – Partie 1: Communication de sécurité sur des systèmes de transmission fermés*

[18] CEI 62280-2, *Applications ferroviaires – Systèmes de signalisation, de télécommunication et de traitement – Partie 2: Communication de sécurité sur des systèmes de transmission ouverts*

[19] IEC 62443 (toutes parties), *Industrial communication networks – Network and system security* (disponible uniquement en anglais)

---

16 Les publications monolingues des séries IEC 61158 et IEC 61784 sont actuellement en cours de traduction.

17 En cours d'élaboration.

18 A publier.

19 Proposition d'un nouveau sujet d'étude soumise à examen.

[20] ISO/CEI Guide 51, *Aspects liés à la sécurité – Principes directeurs pour les inclure dans les normes*

[21] ISO/CEI 2382-14, *Technologies de l'information – Vocabulaire – Partie 14: Fiabilité, maintenabilité et disponibilité*

[22] ISO/CEI 2382-16, *Technologies de l'information – Vocabulaire – Partie 16: Théorie de l'information*

[23] ISO/CEI 7498 (toutes parties), *Technologies de l'information – Interconnexion des systèmes ouverts (OSI) – Modèle de référence de base*

[24] ISO/CEI 19501, *Technologies de l'information – Traitement distribué ouvert – Langage de modélisation unifié (UML), version 1.4.2*

[25] ISO 10218-1, *Robots et dispositifs robotiques – Exigences de sécurité pour les robots industriels – Partie 1: Robots*

[26] ISO 12100-1, *Sécurité des machines - Notions fondamentales, principes généraux de conception – Partie 1: terminologie de base, méthodologie*

[27] ISO 13849-1, *Sécurité des machines – Parties des systèmes de commande relatives à la sécurité – Partie 1: principes généraux de conception*

[28] ISO 13849-2, *Sécurité des machines – Parties des systèmes de commande relatifs à la sécurité – Partie 2: Validation*

[29] ISO 14121, *Sécurité des machines – Principes de l'appréciation du risque*

[30] EN 954-1:1996[20], *Safety of machinery – Safety related parts of control systems – General principles for design*

[31] ANSI/ISA-84.00.01-2004 (toutes parties), *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[21], *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

---

20 Sera remplacée par l'ISO 13849-1 et/ou la CEI 62061.

21 Ce document a constitué l'un des points de départ pour l'élaboration de la présente norme. Il fait actuellement l'objet d'une révision importante.

[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

———————

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

3, rue de Varembé
PO Box 131
CH-1211 Geneva 20
Switzerland

Tel: + 41 22 919 02 11
Fax: + 41 22 919 03 00
info@iec.ch
www.iec.ch